Universidad Nacional de Misiones

Facultad de Ciencias Exactas, Químicas y Naturales

Tesis de grado Licenciatura en Sistemas de Información

Arquitectura de software basada en blockchain para la presentación digital de las rendiciones de cuentas municipales

Autor: Facundo Joel Rodriguez

Tutores: Ing. Myriam Micaela Olivera - Ing. Ricardo Daniel Urdinola

Año 2023

Dedicatoria

A mi familia, por sus sacrificios y alientos en momentos difíciles.
A mis amigos y compañeros, por motivarme a finalizar la carrera.
A quienes generosamente compartieron su tiempo para guiarme y finalizar este trabajo.

Resumen

En la actualidad, el Tribunal de Cuentas de la Provincia de Misiones ha iniciado un proceso amplio de modernización y digitalización de los procedimientos llevados a cabo por este organismo de control. Entre ellos se encuentra la digitalización de las presentaciones digitales de las rendiciones de cuentas de las municipalidades de la Provincia de Misiones. Este cambio ha generado la necesidad de asegurar la integridad y confiabilidad de dichas presentaciones. En respuesta a esta problemática, la presente investigación tiene como objetivo diseñar y validar un prototipo de arquitectura de *software* basada en tecnología *blockchain* que permita verificar la integridad y prevenir modificaciones no autorizadas de las presentaciones digitales.

Para la validación de la arquitectura, se ha desarrollado un prototipo de aplicación *web* que permite a la institución publicar el *hash* de los archivos de las presentaciones digitales en la *blockchain*. Este prototipo ha sido probado con datos ficticios sobre las rendiciones de cuentas municipales, proporcionados por el personal de la institución. Los resultados del trabajo destacan el uso de funciones criptográficas para el cifrado de información y su posterior almacenamiento en la *blockchain*, asegurando así la integridad de los datos y permitiendo su verificación futura por parte de los usuarios.

Palabras claves: cadena de bloques, contrato inteligente, integridad de datos, Ethereum, Solidity, Truffle Suite.

Abstract

Currently, the Court of Accounts of the Province of Misiones has initiated a comprehensive process of modernization and digitization of the procedures carried out by this oversight institution. Among them is the digitization of the digital presentations of the financial statements of the municipalities of the Province of Misiones. This change has generated the need to ensure the integrity and reliability of these presentations. This research aims to design and validate a software architecture prototype based on blockchain technology that allows integrity verification and prevents unauthorized modifications of digital presentations.

To validate the architecture, a web application prototype was developed, enabling the institution to publish the hash of the digital presentation files on the blockchain. This prototype was tested with fictitious data related to municipal financial statements, provided by the institution's staff. The results of the study highlight the use of cryptographic functions for data encryption and its subsequent storage on the blockchain, thus ensuring data integrity and allowing future verification by users.

Keywords: blockchain, smart contract, data integrity, Ethereum, Solidity, Truffle Suite.

Reconocimientos

A mis directores y profesores, cuya guía y dedicación han sido fundamentales para el desarrollo de este proyecto. Especialmente las valiosas sugerencias que han contribuido significativamente a la calidad de este proyecto.

Índice general

Índice de figuras

Capítulo 1

Introducción

En el presente capítulo, se aborda la temática de la investigación, se expone la razón de su relevancia y se definen los objetivos tanto generales como específicos que se buscan alcanzar.

1.1. Justificación

La Resolución N°45/2020 emitida por el Tribunal de Cuentas de la Provincia de Misiones en respuesta a la pandemia de COVID-19 del año 2020, ha puesto en marcha el proceso de digitalización de las presentaciones digitales de las rendiciones de cuentas de las municipalidades de la Provincia de Misiones. Esta medida, que ya se encuentra actualizada y normada en la Resolución IV - N° 3 CAPÍTULO V [1], la cual forma parte de un proceso más amplio de modernización y digitalización de los procedimientos llevados a cabo por este organismo de control.

En este contexto, la implementación de la tecnología *blockchain*, entendida como una tecnología de registro distribuido y transparente, puede ofrecer soluciones efectivas para mejorar el control y la transparencia en el manejo de los fondos públicos municipales. La utilización de una arquitectura de *software* basada en *blockchain* brinda una serie de beneficios significativos en este escenario. En primer lugar, proporciona una infraestructura descentralizada, lo que significa que no existe una autoridad centralizada que tenga el control absoluto sobre los datos referidos a cada rendición de cuentas. Esto evita la posibilidad de manipulación o alteración no autorizada de las presentaciones digitales de las rendiciones de cuentas. Al descentralizar la información en una red distribuida, se reduce el riesgo de fraudes y se fortalece la integridad de los datos [2].

Por otra parte, la transparencia es un elemento fundamental en la gestión, fiscalización y control de los fondos públicos. La tecnología *blockchain* permite que todas las transacciones y cambios realizados en el registro sean visibles para todos los participantes del proceso [3], [4]. Esto garantiza que tanto las municipalidades como el Tribunal de Cuentas de la Provincia de Misiones cuenten con acceso a un registro inmutable y transparente de todas las rendiciones de cuentas presentadas.

Cualquier modificación o alteración no autorizada de la información podría ser detectada de manera rápida y confiable.

Otro beneficio importante es la trazabilidad. Cada transacción registrada en la cadena de bloques queda permanentemente registrada y enlazada con las transacciones anteriores, formando un historial completo y verificable [3], [4], [5]. Esto permite un seguimiento preciso de los flujos de acción facilitando la auditoría de la información subida por los responsables municipales, ya que se puede verificar el origen y el destino de cada transacción de manera transparente.

Teniendo en cuenta la situación y beneficios mencionados, el presente trabajo plantea la definición de una arquitectura de *software* basada en *blockchain*. Esta arquitectura brinda una infraestructura descentralizada y transparente para garantizar la integridad de las presentaciones digitales y prevenir modificaciones no autorizadas. De este modo, tanto los representantes de los municipios como el Tribunal de Cuentas de la Provincia de Misiones podrán validar y verificar las presentaciones digitales correspondientes a las rendiciones de cuentas municipales.

1.2. Objetivos

Esta subsección presenta el objetivo general y los objetivos específicos del trabajo.

1.2.1. Objetivo general

El objetivo general de esta investigación es diseñar y validar un prototipo de arquitectura de *software* basado en tecnología *blockchain* que permita verificar la integridad y prevenir modificaciones no autorizadas de las presentaciones digitales de las rendiciones de cuentas municipales establecidas por el Tribunal de Cuentas de la Provincia de Misiones.

1.2.2. Objetivos específicos

- Realizar un relevamiento de las arquitecturas basadas en *blockchain* existentes que se apliquen en el ámbito de auditoría contable, con el fin de comprender las mejores prácticas y los enfoques más relevantes en este campo.
- Describir el contexto del Tribunal de Cuentas de la Provincia de Misiones y las actividades relacionadas con las presentaciones digitales de rendiciones de cuentas municipales, analizando los procesos actuales, los requisitos legales y normativos, como así también, los requisitos referidos a las rendiciones de cuentas municipales.
- Desarrollar una propuesta de arquitectura de *software* basada en *blockchain* que permita verificar la integridad de las presentaciones digitales de las rendiciones de cuentas municipales establecidas por el Tribunal de Cuentas de la Provincia de Misiones.
- Desarrollar un prototipo de aplicación basado en la arquitectura propuesta, implementando la lógica de verificación y validación de las presentaciones digitales.
- Validar la propuesta desarrollada utilizando conjuntos de datos reales o ficticios con un alto grado de similitud en lo que se refiere a los tipos de archivos y tamaño, para identificar posibles áreas de mejora y futuras líneas de investigación.

1.3. Estructura del documento

El documento se divide en 7 capítulos, fuentes bibliográficas y anexos. A continuación, se describe brevemente el contenido de cada uno:

- En el capítulo 2 se presenta el marco teórico, donde se realiza una revisión bibliográfica sobre los conceptos y teorías que sustentan la investigación. Se busca establecer un marco conceptual que permita comprender el problema de investigación.
- En el capítulo 3 se presentan los antecedentes y trabajos previos que se han realizado sobre el tema de la tesis. Se busca establecer el estado del arte y las principales líneas de investigación que se han seguido.
- En el capítulo 4 se describe el contexto en el que se desarrolla la investigación. Se presenta la institución en la que se realiza la investigación, se describen los procesos actuales y se establecen los requisitos legales y normativos que se deben cumplir. Además, se detalla la arquitectura de *software* basada en *blockchain* desarrollada como posible solución.
- En el capítulo 5 se presenta la propuesta de solución al problema planteado. Se describen las tecnologías seleccionadas para el desarrollo de la aplicación, así como los pasos para su instalación y configuración.
- En el capítulo 6 se presentan los resultados obtenidos tras el desarrollo y ejecución de la solución propuesta. Se detalla cómo se ha preparado el entorno para la ejecución de las pruebas y se describe el funcionamiento del contrato inteligente.
- En el capítulo 7 se presentan las conclusiones a las que se ha llegado a partir de la investigación realizada. Se resumen los principales hallazgos y se establecen las implicaciones y recomendaciones para futuras investigaciones.
- Anexo I. Incluye enlaces a sitios relevantes para el lector. Estos recursos proporcionan acceso directo al código fuente utilizado en el proyecto y a componentes específicos del mismo, facilitando su consulta y reutilización.
- Anexo II. Presenta el código fuente del contrato inteligente desarrollado con el lenguaje de programación Solidity.
- Anexo III. Expone las opciones de configuración seleccionadas en la aplicación de Ganache.

- Anexo IV. Presenta capturas de pantalla de la interfaz de usuario del prototipo de aplicación *web* desarrollado.

Capítulo 2

Marco Teórico

En este capítulo, se describe el marco teórico que sustenta la investigación y proporciona el contexto necesario para comprender el problema y la propuesta de solución.

2.1. Arquitectura de software

La arquitectura de *software* es la organización fundamental de un sistema, representada por sus componentes, las relaciones entre ellos y el entorno, y los principios que rigen su diseño y evolución. Incluye elementos de la estructura, el comportamiento, la interacción y los atributos de calidad, así como las decisiones de diseño y restricciones que dan forma al sistema [6].

2.2. Blockchain

Blockchain es una tecnología descentralizada que permite el almacenamiento y la verificación de transacciones basada en una cadena de bloques en la que se registran las transacciones que se van realizando en una red distribuida de computadoras [3], [5], [7], [8], [9].

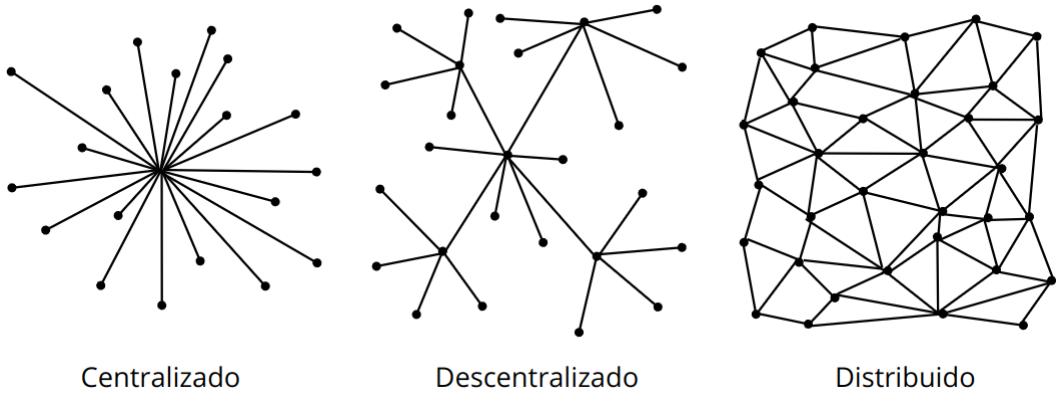

Figura 1: Esquema de organización de las redes. Fuente: Adaptado de [9].

En una *blockchain*, cada bloque contiene datos y un *hash* único que lo identifica, además, contiene una referencia al bloque que lo antecede, creando así una relación lógica entre ellos: la *cadena*. Con el uso de técnicas criptográficas, la información que contiene cada bloque sólo puede ser alterada modificando todos los bloques anteriores. El primer bloque de una *blockchain* se denomina bloque *génesis* y se

caracteriza por no tener un bloque predecesor. Por lo tanto, representa el inicio absoluto de la cadena y establece el fundamento inicial para el registro y la continuidad de las transacciones en la *blockchain* [3], [4], [7], [9].

Dentro de una *blockchain*, una *transacción* representa una acción o intercambio de valor registrado en la cadena de bloques. Estas transacciones se agrupan en bloques y se validan mediante un proceso conocido como *consenso*, en el que los *nodos* de la red colaboran para verificar y agregar nuevos bloques a la cadena. Una vez que una transacción se ha agregado a un bloque y ha sido validada por la red, se vuelve inmutable y forma parte de la historia registrada en la *blockchain* [3], [4].

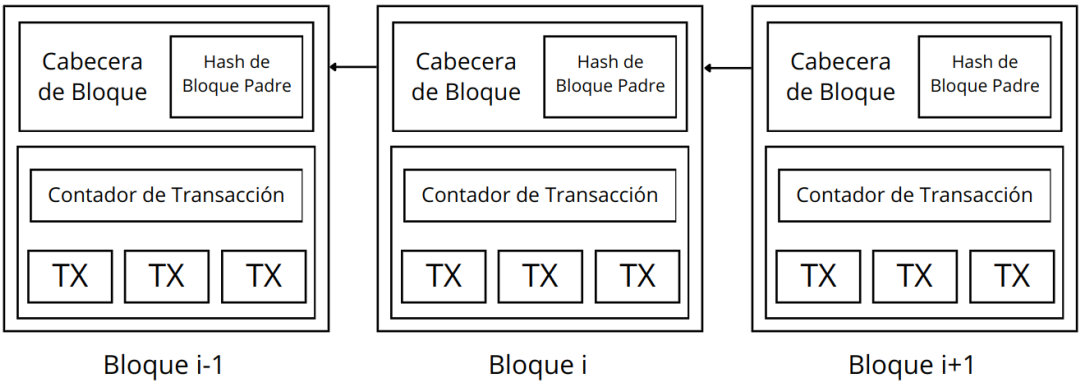

Figura 2: Un ejemplo de blockchain que consiste en una secuencia continua de bloques. Fuente: Adaptado de [4].

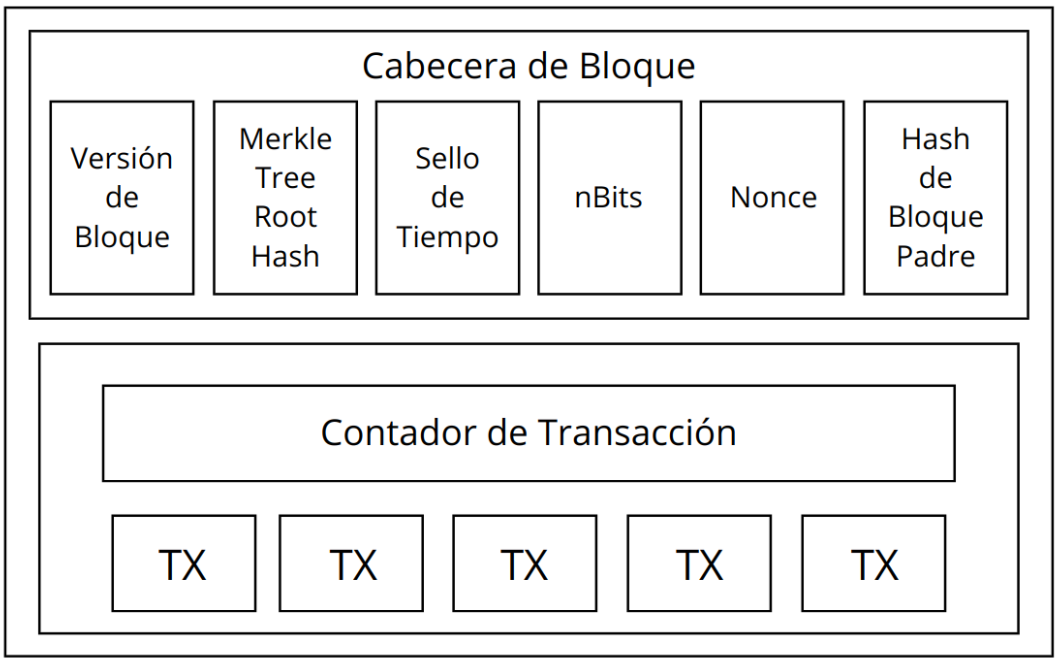

Figura 3: Estructura de bloques. Fuente: Adaptado de [4].

Una *blockchain* se compone de una red de nodos que participan en la verificación y mantenimiento de la cadena de bloques. Cada nodo tiene una copia completa de la cadena y trabaja en conjunto con otros nodos para verificar las transacciones y mantener la integridad de la red. La descentralización de la red implica que no hay una entidad central o autoridad que controle la *blockchain*. Los nodos de la red se comunican y se ponen de acuerdo mediante protocolos de consenso, como la Prueba de Trabajo (PoW, por sus siglas en inglés) o la Prueba de Autoridad (PoA, por sus siglas en inglés), los cuales son explicados más adelante [3], [5], [9], [10].

La trazabilidad es otra característica destacada. La tecnología *blockchain* permite rastrear todas las operaciones realizadas en una dirección específica y conocer el recorrido de las transacciones ejecutadas. Todas estas operaciones están vinculadas y almacenadas en los nodos de la *blockchain* [3], [5].

2.2.1. Tipos de blockchain: pública, privada e híbrida

En una *blockchain* pública, cualquier persona puede unirse a la red, participar en la validación de transacciones y acceder al historial de transacciones completo. La red es completamente descentralizada y no requiere permisos para participar. Cualquier individuo puede convertirse en un nodo de la red y contribuir a la seguridad y la validación de las transacciones. La transparencia y la inmutabilidad son características clave de las *blockchain*s públicas. Ejemplos notables de *blockchain*s públicas son Bitcoin[1] y Ethereum [3], [4].

En una *blockchain* privada, el acceso y la participación están restringidos a un grupo selecto de entidades o participantes. Estas *blockchain*s son más centralizadas que las públicas y, por lo general, están controladas por una organización central o un consorcio. Los permisos y niveles de acceso se gestionan de manera más estricta, lo que puede permitir un mayor control sobre la red y las transacciones. Las *blockchain*s privadas a menudo se utilizan en entornos corporativos donde la privacidad y la velocidad de transacción son prioridades. Ejemplos de plataformas para *blockchain*s privadas son Hyperledger Fabric y Corda [3], [4].

Una *blockchain* híbrida combina elementos de las *blockchain*s públicas y privadas. En este enfoque, se pueden tener múltiples niveles de acceso, donde algunos participantes tienen acceso completo a la red (como en una *blockchain* pública) y otros tienen acceso restringido (como en una *blockchain* privada). Esto permite un mayor grado de flexibilidad y adaptación a diferentes casos de uso. Las *blockchain*s híbridas pueden ser útiles cuando se requiere una combinación de transparencia y control. Por ejemplo, un consorcio de empresas podría utilizar una *blockchain* híbrida para compartir datos y procesos comerciales mientras mantiene una alta confidencialidad [11], [12].

1 Bitcoin: Es una moneda digital descentralizada y un sistema de pago sin banco central o administrador único [5].

2.2.2. Función hash

Una función de resumen o función *hash*, es un algoritmo matemático que toma una entrada, como un bloque de datos, y genera una salida de longitud fija, conocida como *hash* o *resumen*. Existen diversas familias de funciones *hash* disponibles, como MD5[2], SHA[3]-1, SHA-2, SHA-3, RIPEMD[4] y Whirlpool[5]. Las siguientes funciones *hash* son las más utilizadas: SHA-224, SHA-256 (utilizado en Bitcoin), SHA-384 y SHA-512, donde el número indica el número de *bits* [13]. El propósito principal de una función *hash* es mapear[6] datos de cualquier tamaño a una cadena de *bits* de longitud fija, de manera eficiente y determinista. Esta salida, o *hash*, es única para cada conjunto de datos de entrada, lo que permite la verificación de integridad y la identificación rápida de cambios en los datos originales [3], [5].

2.2.3. Protocolos de consensos

"En blockchain, el proceso de armar un bloque de transacciones y sumarlo definitivamente en la cadena se llama sellado o minado. Cuando un bloque queda sellado, la información que contiene pasa a formar parte de la cadena de forma permanente, inmutable e inalterable. El protocolo de consenso es el mecanismo que regula la forma en que los nodos que sellan bloques llegan a un acuerdo entre sí para poder hacerlo (e incorporar ese bloque a la cadena)" [10]. A continuación, se describen los tres protocolos de consensos más destacados en la actualidad.

- Prueba de trabajo (PoW, *Proof of Work*): La prueba de trabajo es un protocolo de consenso en el que los participantes de la red o nodos,

2 MD5: Es la abreviatura de *Message-Digest Algorithm* 5, que en español se conoce como algoritmo de resumen del mensaje 5.
3 SHA: Se refiere a *Secure Hash Algorithm*, que en español se traduce como algoritmo de resumen seguro.
4 RIPEMD: Acrónimo de RACE *Integrity Primitives Evaluation Message Digest,* que en español se traduce como primitivas de integridad del resumen del mensaje.
5 Whirlpool: Es una función de *hash* diseñada por Vincent Rijmen y Paulo S. L. M. Barreto.
6 Mapear: En este contexto, "mapear" se refiere al proceso de transformar o asignar datos de un tamaño variable a una cadena de *bits* de longitud fija, de acuerdo con el algoritmo utilizado.

conocidos como mineros, compiten entre sí para resolver complejos problemas matemáticos. El minero que resuelve primero el problema y encuentra la solución correcta tiene el derecho de agregar un nuevo bloque a la cadena de bloques y recibir una recompensa, en forma de criptomonedas. La prueba de trabajo garantiza la seguridad de la red, ya que para alterar un bloque existente, se requeriría un poder computacional masivo y superar la mayoría de la capacidad de la red, lo cual resulta extremadamente costoso y poco probable [5], [10].

- Prueba de participación (Pos, *Proof of Stake*): La prueba de participación es un protocolo de consenso creado para reemplazar al conocido PoW. A diferencia del PoW, que requiere que los participantes resuelvan problemas matemáticos complejos, es decir minería, para validar transacciones, el PoS se basa en la idea de que la capacidad para crear nuevos bloques y validar transacciones debe depender de la cantidad de criptomonedas que un participante *deposite* o *bloquee* como garantía en la red. Este bloqueo se utiliza como una especie de incentivo para que los validadores actúen de manera honesta, ya que si se descubre que están llevando a cabo actividades maliciosas o fraudulentas, pueden perder parte o la totalidad de su depósito [14].
- Prueba de autoridad (PoA, *Proof of Authority*): La prueba de autoridad es un protocolo de consenso basado en la reputación, que introduce una solución práctica y eficiente para las redes *blockchain*, especialmente las de carácter privado. Propuesto en 2017 por Gavin Wood, cofundador de Ethereum y ex Director de Tecnología (CTO por sus siglas en inglés), este algoritmo aprovecha el valor de las identidades, donde los validadores de bloques no apuestan monedas, sino su propia reputación. En consecuencia, las cadenas de bloques basadas en PoA están protegidas por nodos de validación seleccionados arbitrariamente como entidades confiables. La PoA opera en una red de pares, también conocida como *P2P*, *P-to-P* o *peer to peer* en

inglés, en la que se conoce la identidad de los nodos, a diferencia del protocolo de PoW, donde esto no es necesario. La PoA ofrece la ventaja de procesar un mayor número de transacciones en menos tiempo en comparación con los protocolos de PoW y PoS [10], [15].

Actualmente, en este campo de la descentralización, existen más opciones como: Prueba de Participación Delegada (DPoS, Delegated Proof of Stake), Prueba de Quema (PoB, Proof of Burn), Prueba de Capacidad (PoC, Proof of Capacity), Prueba de Espacio (PoSpace, Proof of Space), Prueba de Tiempo Transcurrido (PoET, Proof of Elapsed Time), Prueba de Historia (PoH, Proof of History), Prueba de Importancia (PoI, Proof of Importance).

2.2.4. Nodos

En *blockchain*, los nodos son los integrantes de la red de una *blockchain* específica. Cada nodo almacena los bloques y compiten por añadir nuevos bloques y validar transacciones. Los nodos son depositarios de toda la información y su presencia es esencial para la existencia misma de la *blockchain*. Cuanto más nodos haya en la red, mayor es la descentralización y la seguridad de la *blockchain* [3], [5], [16].

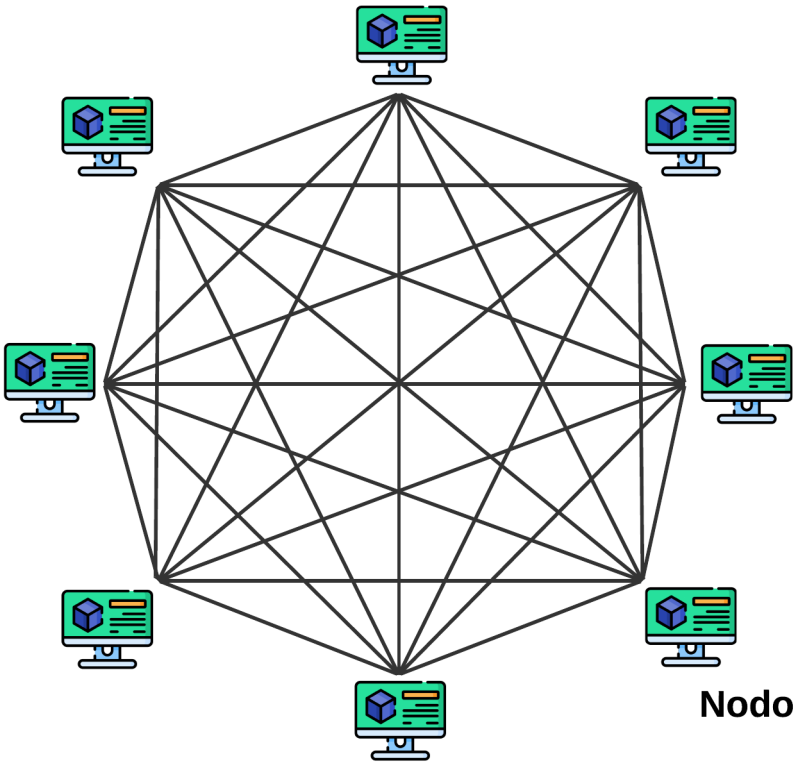

Figura 4: Estructura de nodos. Fuente: Elaboración propia.

2.2.5. Ethereum

Ethereum es una red que se forma para construir aplicaciones y organizaciones, tener activos y poder hacer transacciones comunicándose sin ser controlada por una autoridad central. No hay necesidad de entregar todos los datos personales para utilizar Ethereum, simplemente se puede mantener el control de los datos propios y lo que está compartiendo. La red de Ethereum no está controlada por una sola entidad, existe únicamente a través de la participación descentralizada y la cooperación de la comunidad. Ethereum hace uso de nodos operados por voluntarios, que reemplazan a los sistemas individuales de servidores y servicios de computación en la nube de los principales proveedores de Internet [3], [17].

2.2.5.1. Ether (ETH)

La red de Ethereum posee su propia moneda digital conocida como Ether (ETH), que se utiliza para realizar diversas operaciones en la red Ethereum. Ether incorpora un sistema de precios para calcular la capacidad de procesamiento de Ethereum. Cuando los usuarios desean hacer una transacción, deben abonar en Ether para que su transacción sea validada en la *blockchain*. Estos costos de operación se denominan *tarifas de gas*; el precio del gas varía en función de la potencia de computación necesaria para procesar la transacción y la demanda de energía de la red en el momento de la transacción [18].

2.2.5.2. Aplicación descentralizada (DApp)

Una aplicación descentralizada (DApp, por sus siglas en inglés) es una aplicación construida sobre una red descentralizada, fusionando un contrato inteligente con una interfaz de usuario. En la red Ethereum, los contratos inteligentes son transparentes y accesibles, similares a las API abiertas, permitiendo que tu DApp incorpore contratos inteligentes escritos por otros [19].

2.2.5.3. Contratos inteligentes (Smart contracts)

Un contrato inteligente o *smart contract* es un programa informático seguro e inmutable que representa un acuerdo que es automáticamente ejecutable y exigible. Estos contratos están basados en la tecnología *blockchain* y utilizan lenguajes de programación específicos para definir las reglas y condiciones del acuerdo. Un *smart contract* actúa como un conjunto de instrucciones lógicas que se ejecutan de manera automática una vez que se cumplen las condiciones preestablecidas [3], [20], [21].

En Ethereum, los contratos inteligentes ofrecen la posibilidad de almacenar datos directamente en ellos como variables, y debido a las características de la *blockchain*,

se asegura que estos datos solo pueden ser alterados mediante el uso de las funcionalidades del propio contrato inteligente. Además, cada interacción con este contrato queda registrada como una transacción en la *blockchain* [3], [20], [21].

2.2.5.4. Máquina virtual de Ethereum (EVM)

La Máquina Virtual de Ethereum (EVM, por sus siglas en inglés) es un motor de computación que actúa como un ordenador descentralizado. Proporciona un entorno de ejecución para contratos inteligentes en la red Ethereum, permitiendo la ejecución de cualquier programa codificado en cualquier lenguaje de programación. La EVM también gestiona todas las transacciones en la *blockchain* de Ethereum [22].

2.3. Red de pares

Una arquitectura de red distribuida puede ser denominada una red de pares (P-to-P o P2P, por sus siglas en inglés) si los participantes comparten parte de sus propios recursos de *hardware*, como potencia de procesamiento, capacidad de almacenamiento, capacidad de enlace de red, impresoras, etc. A su vez estos recursos compartidos son necesarios para proporcionar el servicio y el contenido ofrecido por la red. Son accesibles directamente por otros pares, sin pasar por entidades intermedias. En esta red, los participantes desempeñan el doble papel de proveedores y solicitantes de recursos. Esto significa que actúan tanto como proveedores de servicios y contenido, y también como solicitantes de servicios y contenido. Este enfoque se conoce como el concepto de *servent*, donde cada participante puede tanto ofrecer como demandar recursos en la red [5], [23].

2.4. Presentaciones digitales

Las Presentaciones Digitales son archivos o conjunto de archivos que contienen la documentación trimestral de las rendiciones de cuentas de un municipio. El

contenido de la presentación se detalla en la Resolución IV - N° 3, ARTÍCULO 125 [1], y puede incluir documentos como balances mensuales, libros contables, declaraciones juradas, entre otros.

2.4.1. Cuentadante

El responsable de generar y enviar las presentaciones digitales lleva el nombre de cuentadante. Un cuentadante es una persona física o jurídica que administra los fondos públicos, y que, en consecuencia, tiene la obligación de rendir cuentas ante el Tribunal de Cuentas. En el caso particular de las rendiciones de cuentas municipales, el cuentadante o responsable de llevar a cabo estas presentaciones es el intendente de cada uno de los municipios [24].

2.5. Conceptos de interés

Para un mayor entendimiento sobre la temática del presente trabajo, se definen a continuación determinados términos y conceptos que facilitan la comprensión sobre las tecnologías utilizadas.

2.5.1. Lenguaje de marcas de hipertexto (HTML)

El Lenguaje de Marcas de Hipertexto (HTML, por sus siglas en inglés) es el componente más básico de la *web*. Define el significado y la estructura del contenido *web*. Además de HTML, generalmente se utilizan otras tecnologías para describir la apariencia y presentación de una página *web,* como CSS, o la funcionalidad y comportamiento, como JavaScript [25].

2.5.2. Hoja de estilo en cascada (CSS)

Las hojas de estilo en cascada (CSS, por sus siglas en inglés) es el lenguaje de estilos utilizado para describir la presentación de documentos HTML o XML. CSS describe cómo debe ser renderizado el elemento estructurado en la pantalla, en papel, en el habla o en otros medios [26].

2.5.3. Interfaz de programación de aplicaciones (API)

Las interfaces de programación de aplicaciones (API, por sus siglas en inglés) son mecanismos que permiten a dos componentes de *software* comunicarse entre sí mediante un conjunto de definiciones y protocolos [27].

2.5.4. Modelo de objetos del documento (DOM)

El modelo de objetos del documento (DOM, por sus siglas en inglés) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador *web* y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento, puede tratarse de un elemento, una cadena de texto o un comentario [28].

2.5.5. Lenguaje de consulta estructurada (SQL)

El lenguaje de consulta estructurada (SQL, por sus siglas en inglés) es un lenguaje de programación para almacenar y procesar información en una base de datos relacional [29].

2.5.6. Computación en la nube (Cloud computing)

La computación en la nube, conocida también como servicios en la nube o simplemente la *nube*, es la disponibilidad bajo demanda de recursos de computación como servicios a través de Internet. Esta tecnología evita que las organizaciones tengan que encargarse de aprovisionar, configurar o gestionar los recursos y permite que paguen únicamente por los que usen [30].

Capítulo 3

Antecedentes

En este capítulo se aborda una revisión de los trabajos relacionados más recientes en el ámbito de las arquitecturas de *blockchain* aplicadas en el sector público.

3.1. Configuración de la búsqueda

La configuración de la búsqueda de referencias bibliográficas de este trabajo incluye la definición de las fuentes, definición de los términos y cadenas de búsqueda, y definición de criterios para la evaluación de calidad de las referencias.

3.1.1. Fuentes de búsqueda

Buscadores de propósito general: Google
Metabuscadores académicos: Google Scholar
Buscadores Académicos: Scopus, IEEE Xplore, ACM Digital Library

3.1.2. Términos

Los términos utilizados para la elaboración de las cadenas de búsqueda incluyen:

Términos Principales (Inglés)	**Términos Principales (Español)**
Blockchain	Cadena de Bloques
Software Architecture	Arquitectura de Software
Audit/Auditing	Auditoría
Accounting	Contabilidad
File	Archivo
Hash/Hashing	Hash/Hashing
Data Integrity	Integridad de Datos
Distributed Ledger	Libro Mayor Distribuido
Smart Contracts	Contratos Inteligentes
State Sector	Sector Público o Estatal

3.1.3. Cadenas de búsqueda

En base a los términos definidos en la sección anterior se generaron las siguientes cadenas de búsquedas:

"*blockchain*" AND ("*data integrity*" OR "*integrity*")
"*blockchain*" AND ("*hash*" OR "*hashing*") AND ("*data integrity*" OR "*integrity*")

3.2. Reseñas de literatura académica

A continuación, las reseñas de los artículos encontrados más relevantes para esta investigación en base a la búsqueda definida en la sección anterior.

En la investigación *Using blockchain to validate audit trail data in private business applications* [31], la problemática que se menciona trata sobre la manipulación de datos, destacando que es una amenaza significativa identificada por expertos en ciberseguridad y mencionada como una de las principales amenazas de seguridad en el año 2016. Los autores destacan la preocupación sobre cómo garantizar la integridad de los datos, especialmente en el ámbito empresarial, donde la manipulación podría tener consecuencias perjudiciales. Mencionan que, aunque el uso de registros de auditoría es común, estos son vulnerables a manipulaciones si se almacenan de la misma manera que los datos de la aplicación. El objetivo de los autores es desarrollar un método para validar la integridad de cualquier dato arbitrario, y es aquí donde proponen explorar la tecnología *blockchain* como una solución prometedora para abordar esta problemática. La metodología que proponen es utilizar contratos inteligentes en la *blockchain* de Ethereum, centrada en almacenar el *hash* de los datos directamente dentro de estos contratos inteligentes como variables. El artículo carece de descripciones detalladas de las tecnologías que se utilizan para el desarrollo de la aplicación, únicamente se menciona a Ethereum, Solidity, Truffle y Ganache, y Apache Isis. De todas formas, los autores brindan

acceso al código fuente de la aplicación, que se encuentra alojado en un repositorio de Github. La evidencia que se presenta consta de tres escenarios de pruebas. En el primer escenario, un administrador inexperto sobrescribe accidentalmente la base de datos y la validación del registro de auditoría detecta las entradas faltantes. En el segundo escenario, un empleado intenta encubrir cambios financieros maliciosos y la validación revela las entradas de auditoría eliminadas, exponiendo la actividad fraudulenta. El tercer escenario, se intenta desviar la culpa mediante la manipulación de entradas de auditoría y se destaca la capacidad de la validación para detectar alteraciones en tiempo real. Los autores concluyen que la validación de *hash* basada en *blockchain* puede detectar manipulación o pérdida de datos. Sin embargo, también demuestra que no puede prevenir esta manipulación o pérdida de datos, ya que depende de que los datos originales se almacenen externamente. Sin medidas de restauración de datos perdidos o corruptos, como un protocolo regular de copias de seguridad o datos distribuidos, el método sólo funciona como validación.

En el artículo *System and method for verifying data integrity using a blockchain network [32]*, la problemática que se identifica es acerca del almacenamiento de grandes volúmenes de datos críticos en la nube, esto plantea desafíos en términos de integridad y seguridad de los datos. Los métodos tradicionales de verificación de la integridad de los datos pueden ser ineficientes y vulnerables a la manipulación. El objetivo principal de este trabajo es proporcionar un sistema y un método para verificar la integridad de los datos utilizando la tecnología *blockchain*. Los autores buscan mejorar la transparencia y la seguridad de la verificación de la integridad de los datos al reemplazar la auditoría centralizada tradicional con una red *blockchain* descentralizada. La implementación que los autores proponen es un método que incluye almacenar archivos de datos en un almacenamiento electrónico, crear valores *hash* para cada uno de los archivos y transmitir estos valores *hash* a una red *blockchain*. Los nodos en la red *blockchain* añaden estos valores *hash* como bloques a la *blockchain*. Además, se proporciona una API para monitorear las operaciones de

datos realizadas en los archivos de datos y transmitir los metadatos de cualquier operación realizada a un registro de transacciones. En cuanto a la tecnología, no se especifica ninguna en particular para la validación de método propuesto. Los autores concluyen que este método no previene la manipulación, solamente es capaz de detectarla cuando los datos han sido manipulados y es posible recuperar los datos correctos después de que se haya detectado la manipulación si se cuenta con un protocolo de respaldo.

El artículo *Monitoring File Integrity Using Blockchain and Smart Contracts* [33], aborda la problemática acerca de la seguridad de los datos almacenados en la nube. Los autores señalan que la confidencialidad, disponibilidad e integridad de los datos dependen de la calidad del servicio proporcionado por el proveedor de la nube, y que los datos pueden ser corrompidos, filtrados o eliminados deliberadamente por fallas estructurales o humanas. También afirman que las soluciones existentes para verificar la integridad de los archivos en la nube son centralizadas, dependientes de terceros, costosas o ineficientes, y que no ofrecen una forma de compartir los resultados del monitoreo de forma segura y confiable. Para abordar este problema, proponen una solución basada en la tecnología *blockchain* y contratos inteligentes para garantizar la integridad de los archivos en un Servicio de Almacenamiento en la Nube (CSS, por sus siglas en inglés). El protocolo propuesto se desarrolla en una arquitectura que consta de cuatro roles: Cliente, CSS, Servicio de Verificación de Integridad (ICS, por sus siglas en inglés) y Red de *Blockchain* (BN, por sus siglas en inglés). Este protocolo se implementa en tres fases: fase de preparación, fase de almacenamiento y fase de verificación de integridad. Para la implementación de este protocolo, los autores eligieron la plataforma Ethereum para la implementación de la BN y el lenguaje de programación Solidity para el desarrollo de los contratos inteligentes. Para el desarrollo de las aplicaciones que implementan los procesos asignados al cliente, CSS e ICS, eligieron el lenguaje JAVA EE y sus componentes, como JPA, EJB, CDI y JAX-WS. Las aplicaciones son de tipo local para el cliente y

de tipo *web service* para el CSS y el ICS. El servidor de aplicaciones elegido para las aplicaciones del CSS y el ICS es Glassfish, un servidor de código abierto que soporta las especificaciones de JAVA EE. El *software* elegido para el manejo de las bases de datos de todas las aplicaciones es PostgreSQL, un sistema de código abierto que ofrece alta fiabilidad y escalabilidad. Para implementar la comunicación entre la aplicación y la BN eligieron la librería Web3.js. Los autores no presentan el código fuente de la aplicación en su totalidad, únicamente los contratos inteligentes, y por último realizan pruebas con una implementación de referencia y una BN privada en un laboratorio controlado, demostrando la factibilidad y la efectividad de la solución.

3.3. Plataformas blockchain aplicadas en el sector público

3.3.1. Blockchain Federal Argentina (BFA)

La Blockchain Federal Argentina (BFA) es una plataforma multiservicios abierta y participativa está basada en la tecnología Ethereum. Esta iniciativa, completamente auditable, busca optimizar procesos y servir como herramienta de empoderamiento para toda la comunidad. La BFA fue concebida dentro de un espacio de trabajo colaborativo y busca reproducir ese patrón como columna vertebral de la plataforma [34].

Dentro del modelo operativo de la BFA, la plataforma se rige por el protocolo de consenso basado en prueba de autoridad y se organiza mediante un conjunto fiable de nodos selladores respaldados por instituciones, empresas u organismos, y, de manera singular. La BFA está diseñada sin depender de una criptomoneda asociada. El *gas* o *combustible* necesario para llevar a cabo las transacciones se suministra sin cargo alguno a través del proceso, que lo denominaron Destilería de Gas [35]. La red evita el almacenamiento directo de documentos o archivos en la cadena de bloques, optando por guardar únicamente los *hashes* correspondientes. Los usuarios y

servicios son responsables de resguardar estos documentos de la manera que consideren más adecuada, y al tener los digestos criptográficos sellados en la *blockchain*, se asegura la capacidad de demostrar la integridad de los documentos, evidenciando que no han sido modificados desde la obtención del *hash* [36].

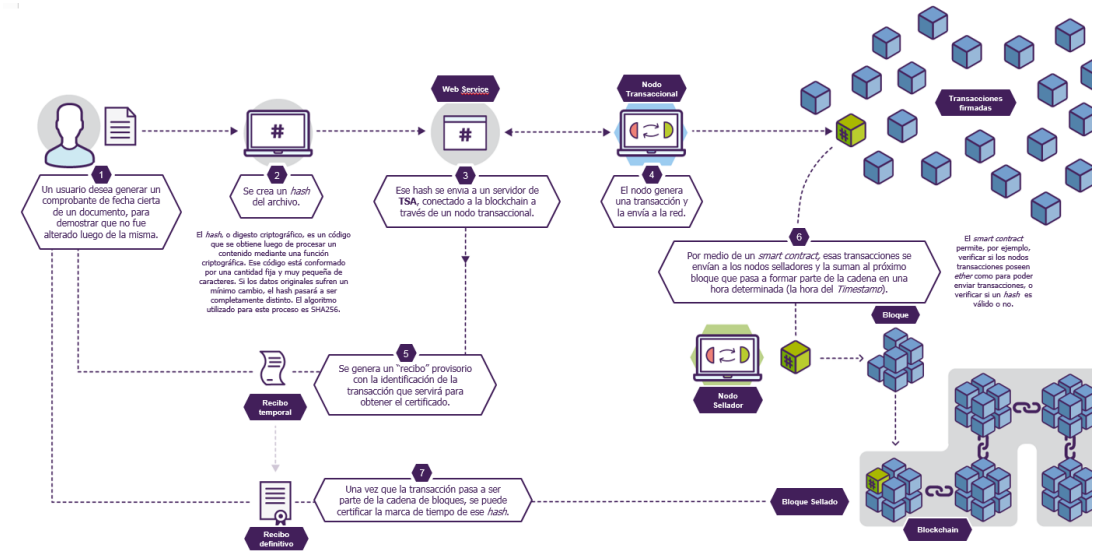

Figura 5: El servicio de Sello de Tiempo de BFA. Fuente: Figura extraída de [35].

La Blockchain Federal Argentina (BFA), en el ámbito del sector público, tiene una serie de aplicaciones significativas, entre las cuales se destaca su capacidad para mejorar la transparencia en los procesos de licitación pública. Según BFA, *"al incorporar blockchain a un proceso de licitación, encontramos nuevas formas de facilitar el proceso de auditoría, tanto a los oferentes como a la sociedad en general"* [37].

3.3.2. Portal de denuncias con Polygon

Polygon, antes conocida como Matic Network, es una plataforma de *blockchain* que crea un sistema multi cadena compatible con Ethereum. Polygon se usa en aplicaciones descentralizadas y usa un mecanismo de consenso de prueba de

participación modificado que logra un consenso por cada bloque. La prueba de participación requiere que los participantes de la red hagan *staking*[7], es decir, que bloqueen sus tokens MATIC para validar las transacciones de la red Polygon [39], [40].

Una de las problemáticas que se quiere resolver con Polygon es la corrupción y la delincuencia de la policía local en la India, especialmente en el distrito de Firozabad, en Uttar Pradesh, que cuenta con una población de 2.8 millones. Según el cofundador de Polygon, Sandeep Nailwal, muchas víctimas de delitos, especialmente de violaciones, no pueden obtener justicia debido a la manipulación o el rechazo de sus denuncias por parte de los agentes corruptos [41].

Para enfrentar esta situación, la policía de Firozabad lanzó un portal de denuncias que utiliza la tecnología *blockchain* de Polygon para evitar la alteración. El portal, denominado Primer Informe de Información (FIR, por sus siglas en inglés), permite a las víctimas de delitos registrar denuncias contra los agentes de la policía local sin que estas sean desestimadas o manipuladas por agentes potencialmente corruptos. Con el FIR en la *blockchain*, se asegura que los informes no pueden ser modificados o negados por los oficiales de nivel inferior, y se podría garantizar el derecho a la justicia [41].

El portal de denuncias con Polygon [42], se implementó en el distrito de Firozabad, en el estado de Uttar Pradesh, en la India, el 12 de octubre de 2022. Según Nailwal, el proyecto contó con el apoyo del superintendente senior de la policía de Firozabad, Ashish Tiwari, quien innovó con la tecnología para asegurar una justicia equitativa. El portal de denuncias con Polygon es el primero de su tipo en la India y podría servir de ejemplo para otras regiones que enfrentan problemas similares.

7 Hacer *staking* de Polygon significa participar en el mecanismo de verificación usado por el protocolo de consenso Prueba de Participación, un método alternativo para conseguir criptomonedas [38].

Capítulo 4

Descripción del Contexto

En este capítulo se describe el escenario de la investigación, proporcionando una comprensión detallada del entorno y los procesos relacionados con las presentaciones digitales de rendiciones de cuentas municipales en el ámbito del Tribunal de Cuentas de la Provincia de Misiones.

4.1. Escenario

El Tribunal de Cuentas de la Provincia de Misiones, es un organismo creado por la Constitución Provincial de 1958, cuyo funcionamiento se institucionalizó en 1960 a través del Decreto Ley N° 1214/60, dictado el 23 de abril de ese año. En la actualidad, su estructura está compuesta por una división administrativa y legal, a cuyo cargo está la presidencia y dos divisiones, a cargo de dos vocales jurisdiccionales: la división municipalidades y la división administración central y entes autárquicos. Por otro lado, se encuentra la fiscalía superior, quien toma vista previa de toda decisión que adopta el Tribunal de Cuentas entre otras atribuciones [24].

Los tribunales de cuentas son organismos fiscalizadores colegiados, independientes de los poderes estatales, que desempeñan una función crucial al ejercer un control externo integral sobre la gestión tanto a nivel provincial como municipal. Este papel se extiende más allá de supervisar la administración y se enfoca en la rigurosa fiscalización y vigilancia de las operaciones financieras y patrimoniales, incluso en el ámbito de las empresas estatales y las entidades de derecho privado en las que el estado tenga participación [24].

Los tribunales de cuentas, en su rol de jueces de la conducta de quienes gestionan fondos o bienes estatales, asumen la responsabilidad de investigar y evaluar cualquier posible irregularidad o inobservancia. Su propósito principal radica en la búsqueda de medidas correctivas que permitan restablecer el patrimonio público o regularizar cualquier práctica irregular. Estos organismos, cuyos líderes son miembros de un cuerpo colegiado, típicamente compuesto por tres individuos, reúnen un equipo multidisciplinario con perfiles que abarcan desde el ámbito legal hasta el económico. En muchos casos, estos profesionales poseen atribuciones

equivalentes a las de los jueces en el poder judicial, subrayando así la seriedad de su tarea [24].

Dada la extensa relación que los tribunales de cuentas mantienen con un gran espectro de actores estatales, y considerando que su labor es esencial para el control de los recursos públicos, se torna de suma relevancia que estos organismos operen con eficiencia y efectividad en el desempeño de sus funciones [24].

En el contexto actual, y en consonancia con el proceso continuo de transformación digital que se ha venido implementando desde el año 2020, se ha efectuado la digitalización de las presentaciones correspondientes a las rendiciones de cuentas municipales. Esto ha conllevado la creación de un sistema *web* diseñado para los cuentadantes de todos los municipios de la provincia de Misiones. A través de esta plataforma en línea, los cuentadantes tienen la capacidad de cargar de manera electrónica los archivos inherentes a las rendiciones de cuentas municipales.

La problemática que impulsó al desarrollo de esta investigación radica en la necesidad de garantizar la integridad y confiabilidad de las presentaciones digitales de las rendiciones de cuentas municipales, evitando así la manipulación o alteración no autorizada, en el marco de la transformación digital en curso.

4.2. Análisis para la implementación de blockchain

La implementación de *blockchain* es un proceso complejo que involucra diversos aspectos técnicos y existen múltiples enfoques para su aplicación. El primer paso consiste en seleccionar el tipo de *blockchain* más adecuado para el caso de uso específico. En este contexto, el objetivo de este trabajo es garantizar la integridad de los datos presentados en cada rendición municipal, sin compartir datos de la aplicación con partes no autorizadas. Una *blockchain* completamente privada no

ofrece las mismas garantías de seguridad e inmutabilidad que una pública, pero tiene la ventaja de ser más rápida, escalable y eficiente en el uso de recursos. De todos modos el desarrollo de una *blockchain* privada o híbrida no está previsto en esta investigación.

Para este proyecto, se considera que una *blockchain* pública sería la más adecuada. La implementación de una *blockchain* pública implica que todas las transacciones son visibles para todos los participantes de la red, lo que aumenta la transparencia y permite a cualquier participante verificar la integridad de los datos. Además, debido a su naturaleza descentralizada, una *blockchain* pública es más resistente a los ataques y a la manipulación de datos [3], [4].

El segundo aspecto a definir es cómo se almacena la información en la *blockchain*. Se presenta la opción de optar por incluir los datos dentro de una transacción, siguiendo el modelo utilizado por Blockcerts o Bitcoin. Alternativamente, se puede explorar la posibilidad de aplicar una lógica más avanzada y almacenar la información en una *blockchain* que admita contratos inteligentes. Uno de los aspectos destacados de los contratos inteligentes es su capacidad para incluir y almacenar datos específicos, tales como el año, período y municipio de una presentación, así como los documentos vinculados a esta presentación representados en forma de *hash*, entre otros datos relevantes. Estos datos se incorporan en el contrato inteligente como parte de su lógica programable y pueden utilizarse para verificar y validar la información desde una aplicación.

En esta investigación, se elige utilizar contratos inteligentes como solución para el almacenamiento de datos en la *blockchain*. Teniendo en cuenta esto, se debe seleccionar una plataforma de *blockchain* que sea compatible con la implementación de contratos inteligentes. Existen varias opciones disponibles, locales y en línea, cada una con sus propias características y ventajas:

- Ethereum: Ethereum es una de las plataformas de *blockchain* más populares y ampliamente utilizadas para contratos inteligentes. Ofrece flexibilidad y una amplia comunidad de desarrolladores. Su lenguaje de programación principal es Solidity, y las herramientas de desarrollo como Truffle y Ganache son ampliamente utilizadas en el ecosistema Ethereum. Además, cuenta con redes de pruebas en línea como Ropsten, Rinkeby. Estas redes se utilizan para probar contratos inteligentes y realizar pruebas de desarrollo en un entorno que simula la red principal de Ethereum (Ethereum Mainnet[8]) pero sin utilizar *ether* real.
- BNB Smart Chain (BSC): En 2022, Binance Chain y Binance Smart Chain se fusionaron para formar BNB Chain. Como parte de esta plataforma, BNB Smart Chain admite contratos inteligentes y protocolos que son compatibles con la Máquina Virtual de Ethereum (Ethereum Virtual Machine, EVM). Para lograr el consenso, BSC combina Prueba de Participación Delegada (DpoS, por sus siglas en inglés) y Prueba de Autoridad (PoA, por sus siglas en inglés). Desarrollada por Binance, BSC se ha ganado la popularidad debido a su velocidad superior y tarifas de transacción más bajas en comparación con Ethereum [43].
- Hyperledger Fabric: Hyperledger Fabric es una plataforma de *blockchain* empresarial desarrollada por la Fundación Linux. Está diseñada para aplicaciones empresariales y permite la creación de redes de *blockchain* privadas y permisionadas. Fabric admite contratos inteligentes creados en lenguajes de programación de propósito general como Java, Go y JavaScript, también soporta la EVM y Solidity [44].

Cada una de estas plataformas tiene sus propias ventajas y consideraciones, y la elección de Ethereum para el desarrollo de este trabajo se basa en la disponibilidad de documentación, su facilidad para crear una red para realizar pruebas, la cantidad

8 Ethereum Mainnet se refiere a la red principal y pública de Ethereum. Es la red en la que se realizan transacciones reales con ether (ETH), la criptomoneda de Ethereum, y donde los contratos inteligentes se ejecutan en un entorno de producción real.

de herramientas disponibles para apoyar el uso y la ejecución de pruebas, y su posición líder en el ámbito de contratos inteligentes [45].

4.3. Aplicación de hashing para la validación de datos

El *hashing* es una técnica que consiste en aplicar una función matemática a un conjunto de datos para obtener un valor único que los identifica. Este valor se llama *hash* o huella digital y sirve para verificar la integridad y autenticidad de los datos [13].

En el contexto de la verificación de archivos, el *hashing* permite comprobar si un archivo ha sido modificado o alterado de manera malintencionada. Al calcular el *hash* de un archivo y compararlo con el *hash* original, se puede detectar cualquier cambio en el contenido o en los metadatos del archivo. Esto es especialmente útil para garantizar la seguridad y la calidad de los datos que se transmiten o almacenan.

Como se describe en el capítulo Antecedentes, las plataformas y artículos mencionados emplean esta técnica para demostrar que una secuencia de *bits* o cualquier tipo de archivo, ya sea un documento o un certificado, ha permanecido inalterado desde el momento en que se registró su *hash* en la *blockchain* junto con su identificador único, conocido también como ID. Aunque algunas de estas plataformas presentan procesos más elaborados, todas comparten el objetivo de asegurar a los interesados si ha habido algún cambio en los datos del documento o archivo en cuestión. Después de examinar el funcionamiento de estas soluciones, se propone una arquitectura de *software* basada en redes *blockchain* que permita verificar la integridad de las presentaciones digitales de las rendiciones de cuentas municipales, conforme a las directrices establecidas por el Tribunal de Cuentas de la Provincia de Misiones.

4.4. Diseño de la arquitectura

Esta sección tiene como objetivo la creación de una arquitectura de *software* basada en *blockchain* para verificar la integridad de las presentaciones digitales. La idea principal es utilizar la inmutabilidad y la transparencia de la *blockchain* para prevenir modificaciones no autorizadas en las presentaciones.

Teniendo en cuenta que el Tribunal de Cuentas de la Provincia de Misiones se encuentra en el proceso de desarrollo de una plataforma *web* dedicada a la presentación de rendiciones de cuentas, la arquitectura propuesta se fundamenta en la combinación de elementos clave del desarrollo de aplicaciones *web*, abarcando tanto el *backend* como el *frontend*, a los cuales se incorpora una capa de *blockchain* basada en contratos inteligentes para la validación de las presentaciones.

La elección de una aplicación *web* se basa en la tendencia a ofrecer interfaces de usuario intuitivas y de fácil uso, lo que facilita la interpretación de los datos incluso para aquellos usuarios que no poseen conocimientos técnicos. Además, las aplicaciones *web* facilitan la implementación de actualizaciones y mejoras, asegurando la escalabilidad del sistema sin la necesidad de realizar cambios significativos en la infraestructura existente [46].

4.4.1. Lado del servidor (Back-End)

El *backend*, también conocido como el lado del servidor, constituye el núcleo funcional de una aplicación *web*, típicamente responsable de almacenar y manipular datos. Es responsable de gestionar y procesar la lógica empresarial, interactuar con la base de datos, y proporcionar los servicios necesarios para que el *frontend* funcione [47].

En el contexto del Tribunal de Cuentas, el *backend* de la plataforma se encarga de manejar la recepción, validación y almacenamiento de las rendiciones de cuentas presentadas. El *backend* es el que se integra con la tecnología *blockchain*, utilizando las herramientas y métodos detallados en el siguiente capítulo, y de esta forma garantizar la integridad y autenticidad de los datos a lo largo del proceso de presentación de rendiciones.

4.4.2. Lado del cliente (Front-End)

El *frontend*, también conocido como el lado del cliente, es la cara visible de la aplicación, lo que los usuarios ven y con lo que interactúan directamente. Se refiere a la interfaz de usuario, la disposición visual de elementos y la experiencia general del usuario [48].

En el caso del Tribunal de Cuentas, el *frontend* de la plataforma es la interfaz a través de la cual los Cuentadantes cargan sus rendiciones de cuentas. Aunque en este proyecto la interfaz de usuario no es el enfoque principal, es crucial para proporcionar una experiencia de usuario intuitiva, facilitando la interacción de los usuarios con el sistema.

4.4.3. Capa de blockchain

La capa de *blockchain* es un componente esencial en la arquitectura propuesta, desempeñando un papel crítico en la verificación de la integridad de las presentaciones digitales mediante la aplicación de la tecnología *blockchain*. Al aprovechar la inmutabilidad y transparencia inherentes a la *blockchain*, se establece un mecanismo para prevenir modificaciones no autorizadas en las presentaciones. Es importante destacar que esta capa no almacena directamente los archivos de una presentación, sino que gestiona el *hash* único asociado a cada archivo vinculado a

una presentación específica. El backend es responsable de generar este *hash* antes de llevar a cabo la transacción en la *blockchain* [49].

En este contexto, cada presentación queda representada en la *blockchain* mediante un identificador único generado en el momento de la transacción por el contrato inteligente. A diferencia de almacenar los *hashes* de los archivos individualmente, esta elección se realiza para abordar la posibilidad de que una presentación puede contener uno o más documentos, con la posibilidad de que algunos de estos documentos se repitan. La búsqueda de presentaciones mediante el *hash* de un archivo individual se vuelve impráctica y compleja en tales escenarios.

Al concluir la transacción, el *backend* recibe los datos de la transacción, incluyendo el identificador único de la presentación generado y el *hash* único de la transacción en la *blockchain*. Utilizando esta información, se genera un comprobante que se proporciona al usuario a través del *frontend*. Estos datos permiten al usuario validar su presentación en momentos posteriores. Este enfoque proporciona una capa adicional de seguridad al respaldar la integridad y autenticidad de las presentaciones, estableciendo así un proceso confiable y transparente en el Tribunal de Cuentas de la Provincia de Misiones.

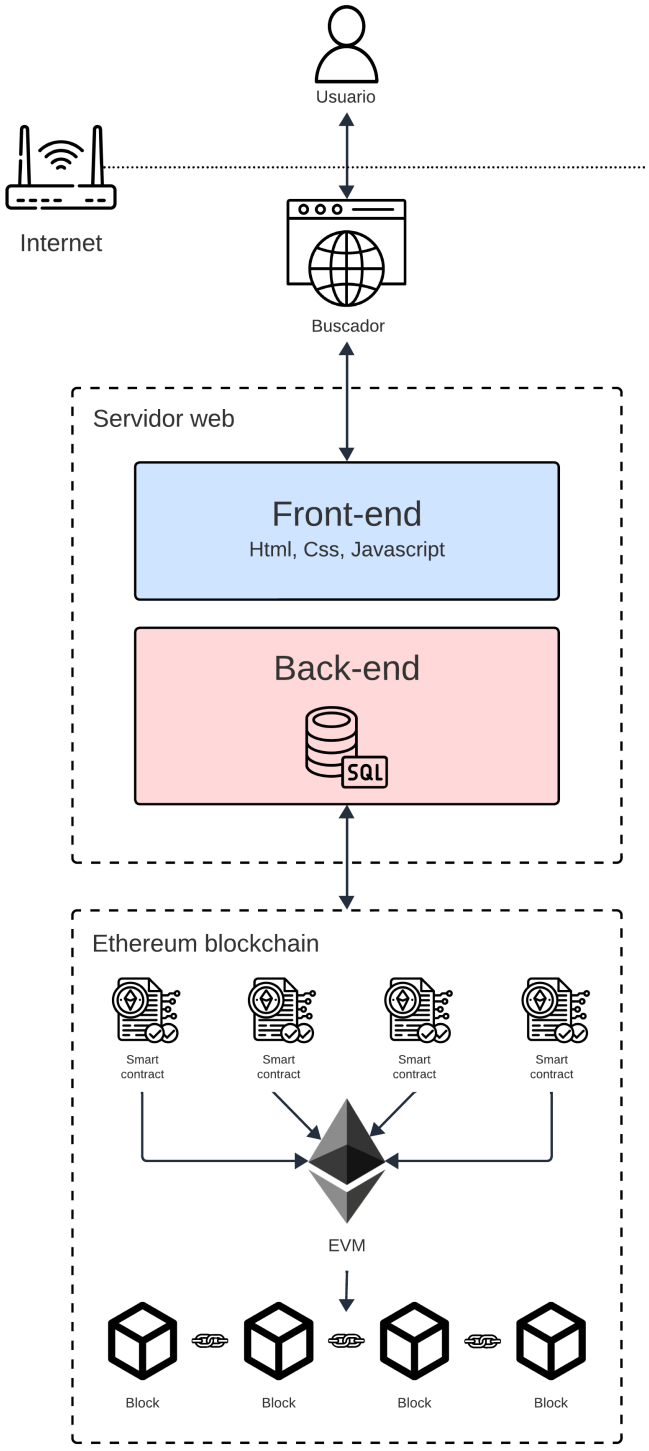

Figura 6: Diagrama de la arquitectura propuesta.
Fuente: Elaboración propia, tomando como referencia la arquitectura de una aplicación Web 3.0 [49].

Para posibilitar la validación de las presentaciones digitales a través del modelo propuesto, se emplean los métodos descritos en la sección anterior. En términos sencillos, se genera un *hash* de los archivos cargados en las presentaciones digitales, obteniendo una salida de longitud fija mediante el uso de una función *hash*, en este caso, se opta por SHA256. Este algoritmo criptográfico se elige por ser el utilizado por defecto en Solidity, así como en las redes Ethereum y Bitcoin [5], [50]. Se distingue por ser uno de los más ampliamente utilizados, destacándose por su equilibrio entre seguridad y eficiencia computacional. Al aplicar este procedimiento, se aumenta la complejidad para conocer el contenido del documento, y se genera un identificador único, el *hash*. Este identificador asegura que el archivo no ha sido modificado y puede ser verificado con el mismo archivo original [31], [32], [36].

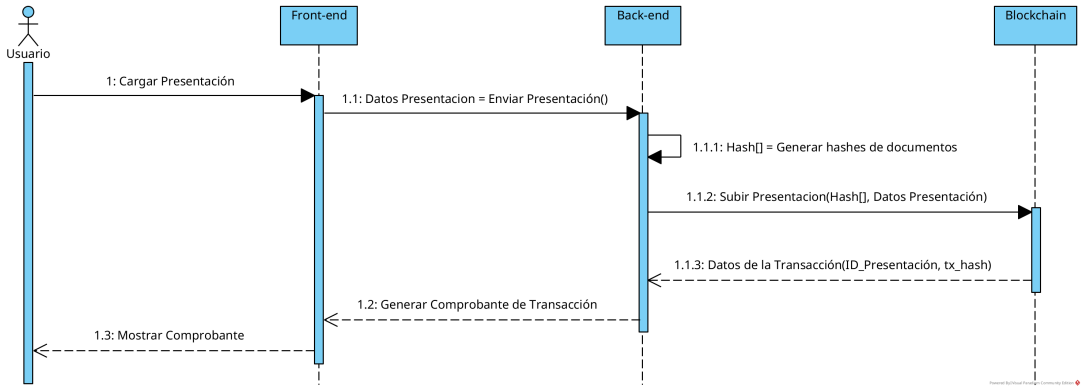

Figura 7: Diagrama de secuencia de la carga de una presentación. Fuente: Elaboración propia.

Basado en este enfoque, se procede al desarrollo de un contrato inteligente que almacene los datos asociados con las presentaciones digitales en la *blockchain*. Este contrato recibe los *hashes* de los archivos directamente desde el *backend*. El desarrollo e integración del contrato inteligente se describe con más detalle en el próximo capítulo.

Capítulo 5

Desarrollo e Implementación

En este capítulo se presenta el desarrollo y la implementación del prototipo de aplicación basado en la arquitectura propuesta.

5.1. Tecnologías seleccionadas

A continuación, se describen las tecnologías y herramientas seleccionadas para el desarrollo del prototipo de aplicación.

5.1.1. Lenguajes de programación

Los lenguajes de programación son conjuntos de instrucciones y reglas que permiten a los programadores comunicarse con las computadoras y darles órdenes para realizar tareas específicas. A continuación, se describen los lenguajes de programación seleccionados.

5.1.1.1. Python

Python [51] es un lenguaje de programación de alto nivel, interpretado y de propósito general. Es conocido por su sintaxis clara y legible que favorece la legibilidad del código. Python es ampliamente utilizado para el desarrollo *web*, análisis de datos, inteligencia artificial, aprendizaje automático y muchas otras áreas. Su amplia gama de bibliotecas y marcos de trabajo lo hacen muy versátil y popular entre los desarrolladores.

En el proyecto, se utiliza el lenguaje de programación Python para la construcción de la aplicación *web*, fundamentalmente el lado del servidor, ya que ofrece una gran facilidad para crear servicios *web* y aplicaciones dinámicas.

Las características principales de Python son:

- Orientado a objetos: Python se basa en el concepto de encapsulación de clases y objetos, lo que permite crear programas modulares y reutilizables.

Python también soporta otros paradigmas de programación, como el funcional, el imperativo y el declarativo.

- Código abierto: Python es un lenguaje de programación de código abierto, lo que significa que su código fuente es accesible y modificable por cualquier persona. Esto favorece el desarrollo colaborativo y la creación de una gran comunidad de usuarios y desarrolladores que contribuyen a mejorar y ampliar el lenguaje.
- Sintaxis: Python tiene una sintaxis simple, que se asemeja al lenguaje natural, y tipado dinámico. El tipado dinámico significa que las variables no necesitan declarar su tipo de datos explícitamente, ya que el tipo se determina en tiempo de ejecución. Esto hace que sea un lenguaje de programación que requiere menos líneas de código que otros lenguajes para realizar la misma tarea. Python también tiene una documentación extensa y numerosos recursos de aprendizaje disponibles en la *web*.
- Integración y adaptación: Python es un lenguaje de programación muy flexible y adaptable, que se puede integrar con otros lenguajes y sistemas. Python tiene soporte para múltiples plataformas, lo que significa que se puede ejecutar en diferentes sistemas operativos, como Windows, Linux o Mac OS. Python también tiene la capacidad de interactuar con otros lenguajes, como C, C++, Java o JSON, mediante el uso de librerías y extensiones.

5.1.1.2. Solidity

Solidity [50] es un lenguaje de programación de alto nivel diseñado para la implementación de contratos inteligentes en la EVM. Su sintaxis está influenciada por la de C++, Python y JavaScript, ofreciendo características como tipado estático, herencia, librerías y la posibilidad de definir tipos de datos complejos por parte del usuario, lo que lo convierte en una herramienta poderosa para el desarrollo de aplicaciones basadas en contratos inteligentes en la plataforma Ethereum.

Los contratos en Solidity son similares a las clases en lenguajes de programación orientados a objetos. Cada contrato puede contener declaraciones de:

- Variables de estado: Son variables que se almacenan de forma permanente en el contrato.
- Funciones: Son bloques de código que se pueden ejecutar y que pueden modificar el estado del contrato.
- Modificadores de funciones: Son piezas de código que se pueden usar para modificar el comportamiento de las funciones. Los modificadores se utilizan cuando se necesita verificar automáticamente una condición antes de ejecutar una función determinada.
- Eventos: Son mecanismos que permiten que las aplicaciones de interfaz de usuario reaccionen a los cambios en los contratos.
- Errores: Son declaraciones que se utilizan para lanzar excepciones si se producen condiciones de error.
- Tipos de estructuras: Son tipos personalizados que se pueden definir para agrupar variables.
- Tipos de enumeraciones: Son tipos que permiten definir una variable que tiene un número limitado de valores predefinidos.

Además, los contratos pueden heredar de otros contratos, lo que significa que pueden adquirir atributos y métodos de los contratos de los que heredan [52].

5.1.1.3. JavaScript

JavaScript [53] es un lenguaje de programación de alto nivel, interpretado y de propósito general. Es uno de los tres lenguajes principales para el desarrollo *web* en el lado del cliente (los otros dos son HTML y CSS). JavaScript permite interactividad en las páginas *web* y se ha convertido en el lenguaje de facto para la *web* con su uso en navegadores *web* modernos.

En este proyecto, se utiliza el lenguaje de programación JavaScript para la construcción de la aplicación *web*, principalmente el lado del cliente, ya que ofrece una gran variedad de funcionalidades para crear páginas *web* dinámicas y atractivas. Además, JavaScript cuenta con librerías y marcos de trabajo (*frameworks*) específicos para el desarrollo *web*, como jQuery, AngularJS, entre otros, que facilitan la creación de interfaces de usuario, la manipulación del DOM, la comunicación con el servidor y la implementación de lógica de negocio.

Las características principales de JavaScript son:

- Imperativo y estructurado: JavaScript es un lenguaje imperativo en el que se van ejecutando las sentencias de manera secuencial. Además, JavaScript permite una programación estructurada construida a base de expresiones y sentencias o bloques de sentencias, que incluyen controles de flujo, declaración de funciones, sentencias de salto, etc.
- Tipado débil y dinámico: En JavaScript no se define el tipo de una variable a la hora de instanciarla. El tipo de la variable se asigna atendiendo al valor que se le asigne a la variable. Además, si se modifica el valor asignado a la variable, esta puede cambiar de tipo de datos. Es por esto que al tipado de JavaScript, además de débil se le considera como tipado dinámico.
- Interpretado: Con el lenguaje JavaScript no se realiza un proceso de compilación a código máquina, sino que se requiere de un intérprete para poder obtener el lenguaje máquina. Cada navegador *web* tiene su propio intérprete de JavaScript, que puede variar en rendimiento y compatibilidad.
- Compatibilidad: JavaScript es un lenguaje que se puede ejecutar en diferentes plataformas, como navegadores *web*, servidores, dispositivos móviles, etc. Esto se debe a que JavaScript se basa en un estándar llamado ECMAScript, que define las especificaciones del lenguaje y que es seguido por la mayoría de los intérpretes.

5.1.2. Gestores de paquetes

Para el desarrollo de una aplicación *web*, es necesario contar con uno o más gestores de paquetes. Los gestores de paquetes son herramientas fundamentales que simplifican la administración de dependencias, entendida como una pieza de *software* que la aplicación requiere para funcionar, comúnmente llamado también como librerías o bibliotecas, y facilitan el proceso de instalación de recursos necesarios para el proyecto. Un paquete es un conjunto organizado de archivos y directorios que agrupa código fuente, recursos y metadatos relacionados con una funcionalidad específica. El uso de un gestor de paquetes ofrece múltiples beneficios, uno de los más destacados es que facilita un control de versiones más sencillo. Esto se debe a que cada paquete instalado posee una versión específica, lo que ayuda a prevenir conflictos y problemas de compatibilidad [54], [55].

En este trabajo se utilizan dos gestores de paquetes populares: *npm* [56] y *pip* [57].

- npm: Es el gestor de paquetes para Node.js [58], una plataforma de JavaScript también conocida como *node*. Npm facilita la instalación, actualización y desinstalación de paquetes de *software* publicados en el repositorio de npm.
- pip: Es el instalador de paquetes para Python. Se utiliza para instalar y administrar paquetes de *software* encontrados en el Python Package Index (PyPI) [59] y otros índices de paquetes compatibles.

Esta elección se basa en su amplia adopción en la comunidad de desarrolladores, su robustez y la gran cantidad de paquetes disponibles. Ambos gestores de paquetes son de código abierto y tienen una comunidad activa que contribuye constantemente con nuevas características y mejoras.

5.1.3. Frameworks y librerías

Para el desarrollo de la aplicación *web*, se elige utilizar Django como marco de trabajo del lado del servidor y jQuery con Bootstrap como bibliotecas del lado del cliente. Estas herramientas permiten crear una interfaz *web* dinámica, atractiva y adaptable a diferentes dispositivos. Además, se incorpora la biblioteca Web3.py en el *backend* para facilitar la comunicación con la capa de *blockchain*. Para la capa de *blockchain*, se utiliza la *suite* de herramientas Truffle, que ofrece un conjunto completo de herramientas para el desarrollo, las pruebas y la implementación de contratos inteligentes en Ethereum.

5.1.3.1. Django

Django [60] es un marco de trabajo *web* gratuito y de código abierto escrito en el lenguaje de programación Python, que facilita el desarrollo de aplicaciones *web* robustas y escalables. Django sigue la arquitectura Modelo-Vista-Controlador (MVC, por sus siglas en inglés), denominada en Django como Modelo-Plantilla-Vista (MTV, por sus siglas en inglés). Esta estructura promueve la separación de preocupaciones, donde el *modelo* gestiona la lógica de la base de datos, la *vista* maneja la presentación y la *plantilla* define el aspecto visual de la interfaz de usuario. La elección se basa en la gran cantidad de herramientas que ofrece para crear aplicaciones *web* de forma rápida y eficiente, con las siguientes ventajas:

- Mapeo Objeto-Relacional (ORM, por sus siglas en inglés): Permite a los desarrolladores interactuar con una base de datos utilizando código Python en lugar de consultas SQL, lo que simplifica el acceso y la manipulación de los datos.
- Motor de plantillas: Permite crear HTML dinámicamente, y de esta forma construir el *frontend* de la aplicación con una sintaxis sencilla y expresiva.

- Seguridad: Ofrece mecanismos integrados para prevenir ataques comunes como inyección SQL, cross-site scripting (XSS[9]), cross-site request forgery (CSRF[10]), entre otros.
- Panel de Administración: Django simplifica la gestión de aplicaciones mediante un potente panel de administración. Este componente proporciona una interfaz *web* lista para usar, permitiendo a los administradores gestionar modelos y usuarios sin necesidad de escribir código adicional. Esta funcionalidad reduce considerablemente el tiempo y esfuerzo dedicado a la creación de interfaces administrativas, liberando a los desarrolladores para enfocarse en la funcionalidad principal de la aplicación.

5.1.3.2. Truffle Suite

Truffle Suite [63] es una colección de herramientas diseñadas específicamente para el desarrollo de aplicaciones descentralizadas en *blockchain* que utilicen la EVM. Dentro de los componentes principales que incluye, se utiliza los siguientes:

- Truffle [64]: Es un entorno de desarrollo y un marco de pruebas para Ethereum. Truffle facilita la compilación, vinculación, implementación y gestión de contratos inteligentes. Con Truffle, los desarrolladores pueden construir, probar, depurar y desplegar sus proyectos de manera rápida. Ampliamente adoptado en la comunidad de Ethereum, Truffle destaca por su accesibilidad y versatilidad, convirtiéndose en una elección popular entre los desarrolladores de DApps [65].
- Ganache [66]: Es una herramienta que permite a los desarrolladores crear una *blockchain* personal de Ethereum en la red local de desarrollo. Con esto,

9 XSS es una vulnerabilidad que permite a un atacante inyectar código malicioso en una página *web* que es vista por otros usuarios. El código malicioso puede acceder a la información sensible del usuario, como las cookies, los tokens de sesión o los datos personales [61].

10 CSRF es una vulnerabilidad que permite a un atacante inducir al usuario a realizar acciones no deseadas en una aplicación *web* en la que está autenticado. El atacante engaña al usuario para que envíe una solicitud maliciosa que realiza una acción en su nombre, como cambiar su contraseña, transferir fondos o comprar algo [62].

se pueden llevar a cabo pruebas y ejecutar comandos, al mismo tiempo que inspeccionar el estado y supervisar el funcionamiento de la cadena. Ganache ofrece tanto una aplicación de escritorio (Ganache UI) como herramientas de línea de comandos (ganache-cli), brindando flexibilidad y control total sobre el entorno de desarrollo [67].

Si bien es posible optar por redes de pruebas en línea como Ropsten o Rinkeby, se considera que Ganache ofrece lo necesario para validar el desarrollo. Esta decisión estratégica asegura la libertad necesaria para desarrollar y probar los contratos inteligentes.

5.1.3.3. Web3.py

Web3.py [68] es una biblioteca de Python esencial para establecer la comunicación entre el *backend* de una aplicación y una red *blockchain*. Esta herramienta facilita la transmisión de transacciones en la *blockchain* y la invocación de métodos del contrato inteligente desplegado.

Esta elección se basa en la compatibilidad con el *backend*, escrito en el lenguaje de programación Python, al igual que esta librería, lo que facilita la lectura y comprensión del código. Además, es relevante señalar que Web3.py se deriva de la librería original de Ethereum, Web3.js [69], y cuenta con el soporte y el mantenimiento de la Fundación Ethereum, lo que refuerza su solidez y confiabilidad en el contexto del desarrollo. Esta biblioteca es ampliamente utilizada en DApps para ayudar con el envío de transacciones, la interacción con contratos inteligentes, la lectura de datos de bloques y transacciones, y una variedad de otros casos de uso [68].

5.1.3.4. jQuery

jQuery [70] es una biblioteca de JavaScript que simplifica la manipulación del DOM, el manejo de eventos, las animaciones y las llamadas AJAX en el *frontend* de una aplicación *web*. jQuery es una de las bibliotecas más populares y utilizadas en el desarrollo *web*, debido a su facilidad de uso, su compatibilidad con múltiples navegadores y su extensa documentación. La elección de esta biblioteca se basa en algunas de las ventajas que ofrece jQuery:

- Sintaxis: Permite escribir código JavaScript más limpio y conciso, utilizando selectores CSS para acceder y modificar los elementos del DOM.
- Efectos visuales: Ofrece una gran variedad de efectos y animaciones para mejorar la experiencia de usuario, tales como deslizar, desvanecer, ocultar, mostrar, entre otros.
- Complementos: Existe una gran cantidad de complementos o *plugins* disponibles para jQuery, que amplían sus funcionalidades y permiten crear componentes interactivos como carruseles, acordeones, calendarios, etc.
- AJAX: Acrónimo de *Asynchronous JavaScript And XML*, es un término que describe un nuevo modo de utilizar conjuntamente varias tecnologías existentes. Esto incluye, HTML o XHTML, CSS, JavaScript, DOM, XML, XSLT, y lo más importante, el objeto XMLHttpRequest. Cuando estas tecnologías se combinan en un modelo AJAX, es posible lograr aplicaciones *web* capaces de actualizarse continuamente sin tener que volver a cargar la página completa [71].

5.1.4. Sistema operativo

Un sistema operativo es un conjunto de programas que sirve como intermediario entre el *hardware* y las aplicaciones de un dispositivo. Su función principal es administrar los recursos, facilitar la comunicación entre el *hardware* y el *software*, y

proporcionar una interfaz para que los usuarios interactúen con el dispositivo. Una de las tareas del núcleo del sistema operativo es gestionar los recursos de localización y protección de acceso del *hardware*, lo que libera a los programadores de aplicaciones de tener que lidiar con estos detalles [72].

En cuanto a los sistemas operativos de escritorio, Microsoft Windows domina el mercado con una cuota de alrededor del 68.87%. Le sigue macOS de Apple Inc. con un 21.16%, y las diferentes distribuciones de GNU/Linux suman un 3.21%. Sin embargo, es importante destacar que las distribuciones Linux son predominantes en los sectores de servidores [73].

5.1.4.1. Linux

Linux es un sistema operativo de código abierto que se destaca por su estabilidad, seguridad y flexibilidad. Su capacidad para adaptarse a una variedad de necesidades y *hardware* lo hace ideal para proyectos de desarrollo y sistemas embebidos. Además, Linux es gratuito, lo que lo convierte en una opción sólida para proyectos con presupuestos limitados. Una de las fortalezas de Linux es su comunidad de desarrolladores y usuarios, que proporcionan soporte y recursos en línea. Esta comunidad contribuye a una amplia gama de herramientas de desarrollo disponibles para Linux [74].

DistroWatch [75] es un sitio *web* que realiza una clasificación de las distribuciones de Linux más populares a nivel mundial. Según las estadísticas de uso y la cuota de mercado de DistroWatch, las distribuciones de Linux más populares en 2023 son:

- MX Linux: MX Linux encabeza la lista gracias a su alta estabilidad, un escritorio elegante y eficiente, y también una fácil adaptabilidad.
- Manjaro: Basado en Arch Linux, Manjaro tiene como objetivo aprovechar la potencia y las características que hacen de Arch una gran opción.

- Linux Mint: Linux Mint es otra distribución popular, conocida por su facilidad de uso y su entorno de escritorio Cinnamon.

Además de estas, hay otras distribuciones de Linux que son populares por diferentes razones. Por ejemplo, Ubuntu es probablemente la distribución de Linux más conocida y es muy popular entre los nuevos usuarios. Debian es conocido por su integración de *software*, y Fedora es popular por su desarrollo impulsado por la comunidad.

Dentro de todas las opciones se opta por utilizar Manjaro como sistema operativo para el desarrollo de la aplicación.

5.1.4.2. Manjaro

Manjaro [76] es un sistema operativo Linux de gran versatilidad, conocido por ser gratuito y de código abierto. Está diseñado con un enfoque sólido en la preservación de la privacidad del usuario, proporcionando un control amplio sobre el *hardware*. Su interfaz fácil de usar lo convierte en una elección excepcional para aquellos que buscan una experiencia informática renovadora, ya que se instala sin inconvenientes en una amplia variedad de dispositivos, incluyendo aquellos con arquitecturas x86-64 y ARM.

Además de su compatibilidad con una amplia gama de *hardware*, Manjaro ofrece una variedad de opciones de personalización, permitiendo a los usuarios adaptar el sistema operativo a sus necesidades específicas. Esto, combinado con una rica selección de aplicaciones disponibles, hace de Manjaro una opción robusta tanto para usuarios principiantes como avanzados.

Entre sus principales ventajas se destacan:

- Facilidad de instalación y uso: Manjaro ofrece un instalador gráfico sencillo y una interfaz intuitiva que facilitan el proceso de configuración y uso del sistema.
- Acceso a los últimos paquetes y actualizaciones: Manjaro utiliza un modelo de actualización continua, lo que significa que los usuarios reciben las últimas versiones de los programas y el *kernel* de Linux sin tener que esperar a las nuevas versiones de la distribución.
- Compatibilidad con el *hardware*: Manjaro detecta y configura automáticamente el *hardware* del sistema, incluyendo los controladores de gráficos, sonido, red y otros dispositivos. También ofrece soporte para el arranque seguro y el *firmware* UEFI.
- Variedad de entornos de escritorio y aplicaciones: Manjaro permite elegir entre varios entornos de escritorio como XFCE, KDE, GNOME, Cinnamon y otros. También cuenta con un gestor de paquetes gráfico que facilita la instalación y el manejo de las aplicaciones.
- Pacman [77]: Pacman, derivado de Package Manager, es el gestor de paquetes predeterminado en Manjaro, una característica distintiva de Arch Linux. Proporciona algunas de las funcionalidades fundamentales que otros gestores de paquetes comunes brindan; incluida la instalación, la resolución automática de dependencias, la actualización, la desinstalación y también la degradación del *software*. Sin embargo, de manera más efectiva, está diseñado para ser simple para que los usuarios de Arch Linux puedan administrar fácilmente los paquetes.

5.1.5. Visual Studio Code

Visual Studio Code [78], conocido como VS Code, representa un entorno de desarrollo integrado (IDE, por sus siglas en inglés) de código abierto y altamente extensible, concebido por Microsoft pero disponible de manera multiplataforma para usuarios de Windows, macOS y Linux. Este potente editor de código ofrece una

experiencia eficiente con resaltado de sintaxis, autocompletado inteligente y sugerencias de código, permitiendo a los programadores adaptar el entorno a sus necesidades.

Su versatilidad se manifiesta en la amplia gama de lenguajes de programación que soporta, desde JavaScript hasta Python, Solidity y varios más. La característica distintiva de VS Code es su extensibilidad, ya que los usuarios pueden enriquecer su funcionalidad mediante la instalación de extensiones, lo que lo convierte en una elección ideal para una variedad de proyectos.

La integración fluida con sistemas de control de versiones como Git [79], facilita el seguimiento y la gestión de cambios en proyectos colaborativos. Ofreciendo herramientas de depuración potentes, puntos de interrupción y una terminal integrada, VS Code proporciona un entorno completo para el desarrollo de *software*. Además, respaldado por una comunidad activa de desarrolladores, su gratuidad y código abierto fomentan su adopción generalizada y su constante mejora como una herramienta esencial en el desarrollo de *software*.

5.2. Desarrollo de la lógica

Definir la lógica de la aplicación permite comprender los datos necesarios que se deben gestionar para facilitar la navegación y garantizar el funcionamiento de la arquitectura propuesta. En base a los requerimientos y el proceso de rendiciones de cuentas municipales se diseña un diagrama de modelo de dominio con el fin de representar los conceptos clave (entidades) del dominio del problema e identificar las relaciones entre todas las entidades comprendidas.

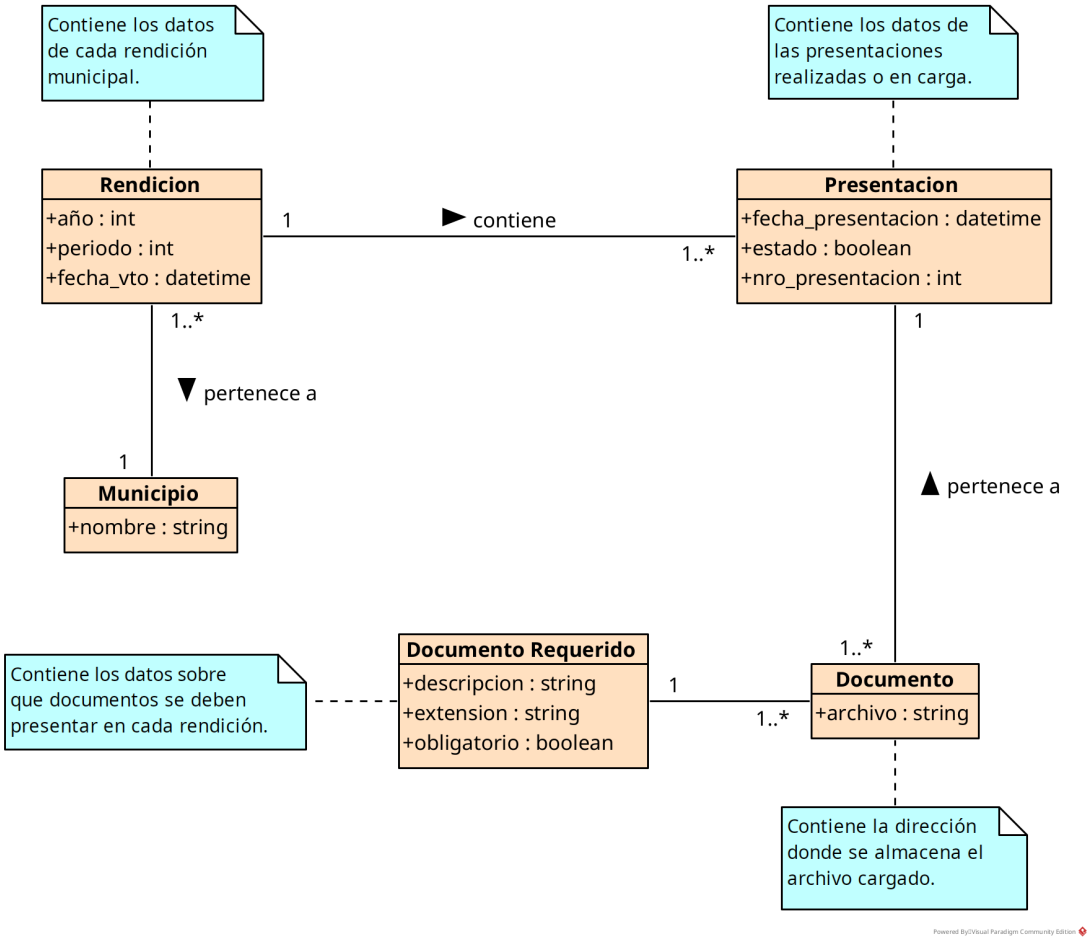

Figura 8: Modelo de dominio. Fuente: Elaboración propia.

El modelo presentado como propuesta en la Figura 8 contiene cinco entidades que son la base para el *backend*. Siguiendo el modelo de Almacenamiento *Off-Chain* de la BFA [36], las presentaciones e información relacionada a ellas, como los archivos, se almacenan dentro del servidor utilizando una base de datos centralizada. Dentro de la *blockchain*, solo se guardan los *hashes* de los archivos e información relacionada a la presentación.

5.3. Desarrollo del smart contract

En la sección anterior, se establecen las entidades fundamentales de la aplicación, lo que permite estructurar el contrato inteligente. Una vez que los contratos se publican o despliegan en la *blockchain*, no pueden ser modificados. Sin embargo, esto no implica que no puedan existir múltiples versiones de un contrato. En una *blockchain* de Ethereum, no hay un límite específico para la cantidad de contratos inteligentes que se pueden publicar. Por lo tanto, se opta por utilizar una *blockchain* local de prueba hasta asegurar de que el contrato inteligente funcione de manera óptima.

La estructura del contrato inteligente se refleja en la Figura 9, y se describe cada declaración a continuación.

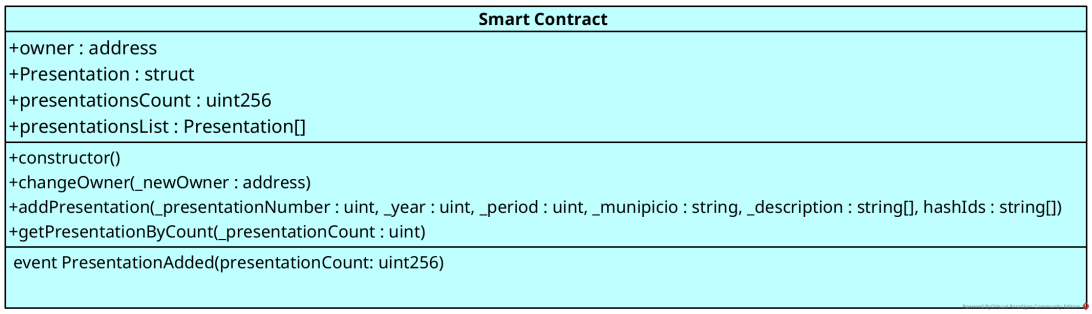

Figura 9: Atributos y métodos del contrato inteligente. Fuente: Elaboración propia.

Las variables que almacenan datos dentro del contrato son:

- *owner : address*, almacena la dirección (*address*) del propietario del contrato inteligente. El propietario, en este caso, es la entidad que despliega el contrato en la red *blockchain*.
- *Presentation : struct*, es una estructura (*struct*) diseñada para almacenar información específica de cada presentación cargada en el contrato inteligente. Dentro de esta estructura, se incluyen dos listas que describen los

documentos asociados a una presentación y los *hashes* de los archivos correspondientes a dichos documentos.

- *presentationsCount : uint256*, es una variable de tipo entero sin signo (*uint256*) que lleva la cuenta total de presentaciones almacenadas en el contrato inteligente. Se utiliza para asignar un ID único a cada presentación y realizar un seguimiento del número total de presentaciones realizadas.
- *presentationsList : Presentation[]*, representa una lista que contiene las estructuras de presentación almacenadas en el contrato inteligente. Cada elemento de la lista corresponde a una presentación y encapsula la información relacionada con los documentos y archivos asociados.

Los métodos de escritura, es decir, que realizan cambios dentro del contrato son las siguientes funciones:

- *constructor()*, el constructor es una función especial del lenguaje que se ejecuta una única vez al desplegar el contrato.
- *changeOwner(_newOwner : address)*, como su nombre indica, permite cambiar al propietario del contrato, recibe como parámetro la dirección que pasa a ser propietario.
- *addPresentation(_presentationNumber : uint, _year : uint, _period : uint, _munipicio : string, _description : string[], hashIds : string[])*, esta función permite agregar una nueva presentación al contrato, recibiendo como parámetros los datos asociados a la rendición, como el número de presentación, el año, el período, el municipio, una lista con los *hashes* únicos de cada archivo y una lista con la descripción de cada documento subido.

Cabe mencionar que se utiliza la convención de Solidity de agregar un guion bajo (_) antes de cada nombre de parámetro.

Los métodos para lectura de datos y eventos son:

- *getPresentationByCount(_presentationCount : uint)*, esta función permite obtener los detalles de una presentación específica por su ID. Devuelve los detalles de la presentación, incluyendo la fecha y hora de la presentación, el número de presentación, el año, el período, el municipio y los documentos asociados.
- *PresentationAdded(presentacionCount : uint256)*, este evento cumple la función de realizar un seguimiento de las adiciones de presentaciones en el contrato inteligente. Cada vez que se invoca la función *addPresentation* a través de una transacción, se emite este evento. Posteriormente, al examinar los eventos emitidos, es posible identificar y rastrear el ID único asignado a cada presentación recién agregada.

La estructura de permisos del contrato es bastante restrictiva, donde solo el propietario puede realizar cambios, mientras que cualquier usuario puede leer los datos. De este modo, se evita almacenar datos sensibles relacionados a los propietarios de las presentaciones. A continuación, se expone la estructura de permisos:

- Propietario: El propietario (*owner*) es la única entidad que tiene permisos para realizar cambios en el contrato. Esto incluye la capacidad de agregar nuevas presentaciones y cambiar el propietario del contrato. El propietario se establece inicialmente en el método constructor del contrato como la dirección de la cuenta que desplegó el contrato.
- Cualquier usuario: Cualquier usuario, independientemente de si es el propietario o no, puede leer los datos del contrato. Esto incluye la capacidad de obtener detalles de una presentación específica y ver el número total de presentaciones.

El código fuente del contrato inteligente, que se adhiere a la estructura previamente descrita, se encuentra en el Anexo II. Se establecieron los atributos y métodos especificados, incorporando además restricciones para que únicamente el propietario del contrato inteligente pueda ejecutar los métodos de escritura.

5.4. Integración con la blockchain

Luego de definir el contrato inteligente, es necesario compilar y desplegar el mismo en una *blockchain* de prueba. Para realizar esta tarea se utilizan las herramientas de Truffle y Ganache, a continuación, se presenta su instalación y configuración.

5.4.1. Instalación de Truffle Suite

La primera herramienta que se incorpora al sistema es Truffle Suite. Antes de comenzar con la instalación de Truffle, es esencial contar con Node.js y npm debidamente instalados en el sistema. Para instalar Node.js y npm, utilizando el gestor de paquetes Pacman se debe instalar el Administrador de Versiones de Node (NVM, por sus siglas en inglés) ejecutando el siguiente comando en una terminal:

```
sudo pacman -S nvm
```

La salida debe ser similar a la que se muestra en la Figura 10.

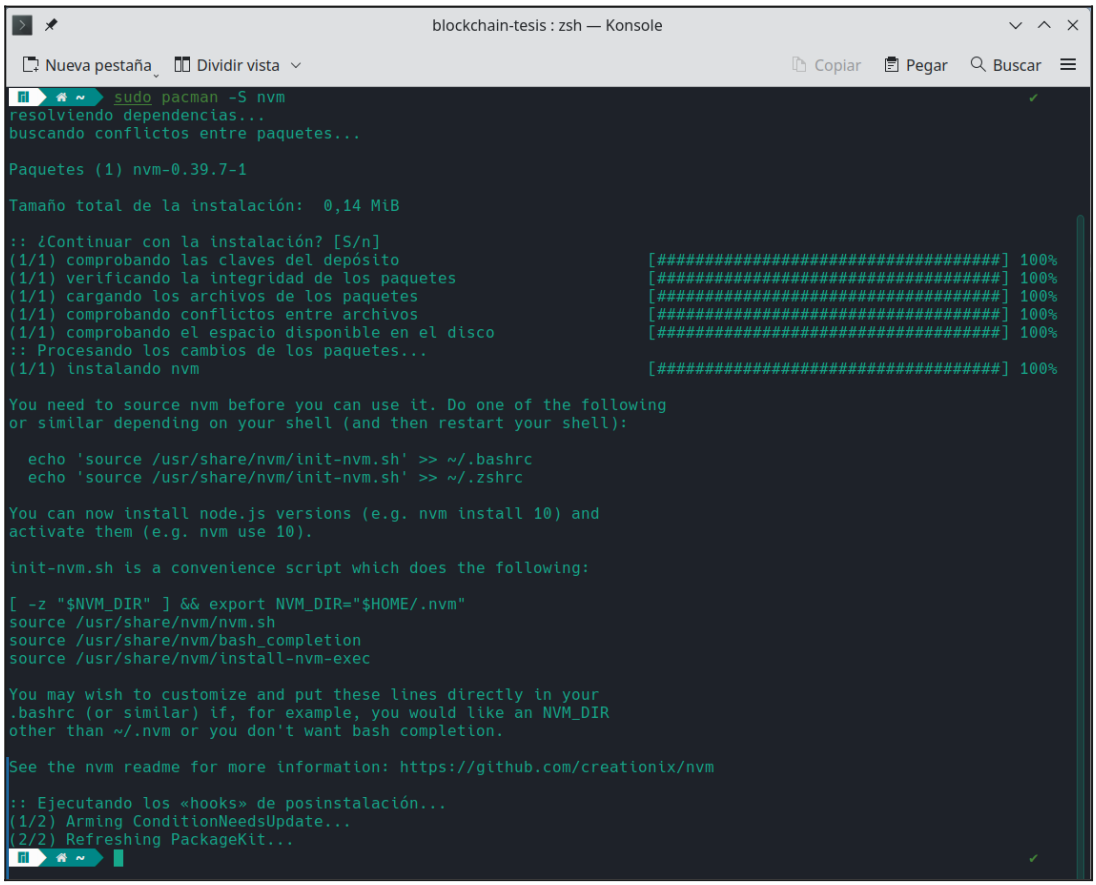

Figura 10: Instalación de NVM con Pacman. Fuente: Elaboración propia.

Para que la terminal reconozca el comando *nvm*, se debe ejecutar los siguientes comandos y luego reiniciar la terminal:

```
echo 'source /usr/share/nvm/init-nvm.sh' >> ~/.bashrc
echo 'source /usr/share/nvm/init-nvm.sh' >> ~/.zshrc
```

Una vez reiniciada la terminal, se puede verificar si los paquetes se han instalado correctamente ejecutando el siguiente comando:

```
nvm -v
```

Si la salida de este comando es similar a la Figura 11, se confirma la correcta instalación.

Figura 11: Versión de NVM instalada. Fuente: Elaboración propia.

Seguidamente, se puede usar NVM para instalar la versión de Node.js que se desee. Para este trabajo se decidió instalar la versión LTS (Long Term Support, en español soporte a largo plazo) más reciente de Node.js, utilizando el siguiente comando:

```
nvm install –lts
```

Este comando descarga e instala la versión LTS de Node.js, así como el gestor de paquetes npm asociado. Se puede comprobar las versiones instaladas de Node.js y npm con los siguientes comandos:

```
node -v
npm -v
```

La salida de ambas ejecuciones debe ser similar a la que se muestra en la Figura 12.

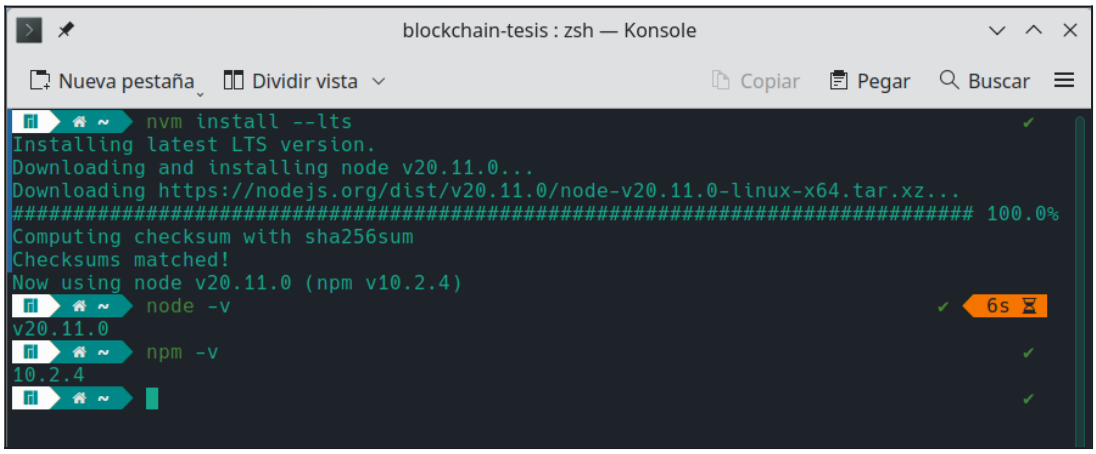

Figura 12: Versión de Node.js y npm instaladas. Fuente: Elaboración propia.

Finalmente, para instalar Truffle Suite, se puede utilizar el siguiente comando:

```
npm install truffle -g
```

Este comando instala Truffle Suite de forma global, lo que permite usarlo desde cualquier directorio. A continuación, se puede verificar la versión instalada de Truffle Suite por medio del uso del siguiente comando:

```
truffle version
```

La salida debe ser similar a la que se muestra en la Figura 13.

Figura 13: Versión de Truffle Suite. Fuente: Elaboración propia.

La siguiente herramienta que se integra en el sistema es Ganache. La instalación de Ganache es un proceso directo; simplemente se debe visitar su sitio oficial [66] y seleccionar la opción de descarga para Linux. Aquí, la descarga resulta en un archivo denominado:

```
ganache-2.7.1-linux-x86_64.AppImage
```

En entornos como Manjaro, los archivos con extensión *.appimage* se ejecutan con tan solo realizar un doble clic derecho en el archivo descargado. Después de ejecutarlo, se debería visualizar la pantalla inicial de la aplicación, la cual se expone en la Figura 14.

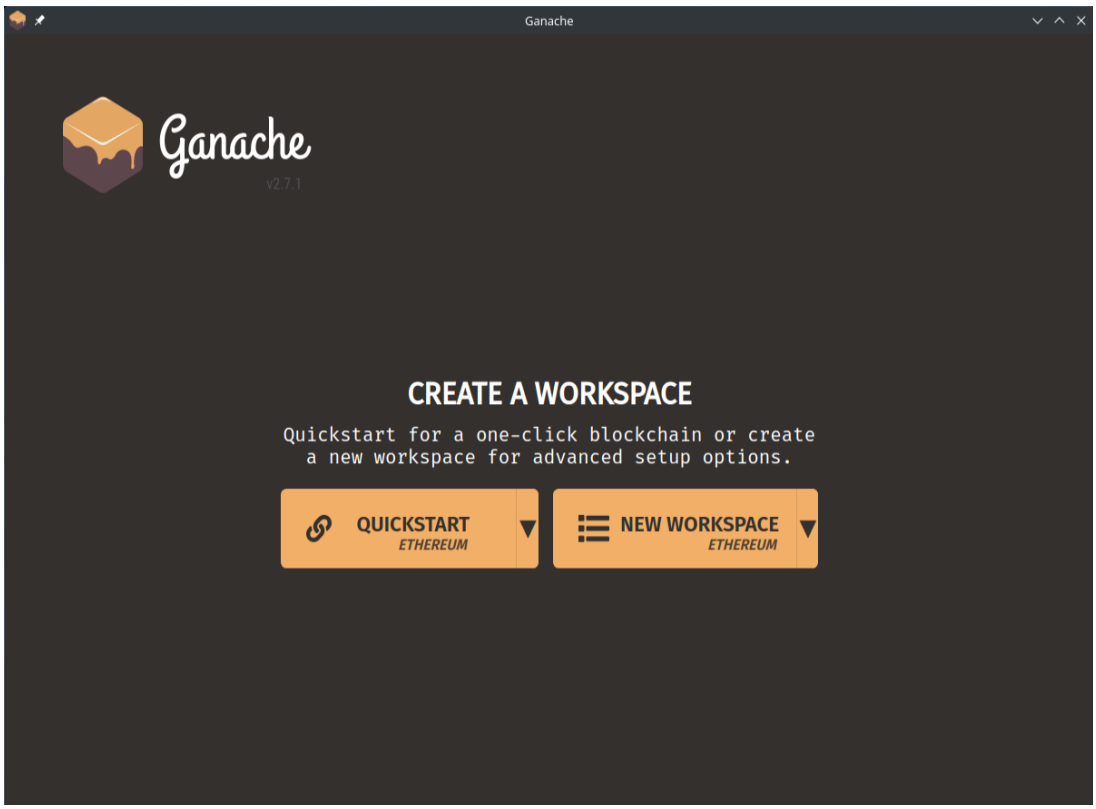

Figura 14: Pantalla inicial de Ganache. Fuente: Elaboración propia.

Ganache, en su versión 2.7.1, ofrece una interfaz que brinda información detallada sobre bloques, transacciones y cuentas de Ethereum en el entorno de desarrollo local. Esta herramienta, al emular una red *blockchain*, facilita la interacción y prueba de contratos inteligentes antes de su despliegue en entornos de producción. En la siguiente sección, se explica la configuración y el uso de Ganache para el flujo de trabajo definido para este trabajo.

5.4.2. Configuración de Truffle y Ganache

Tras haber instalado Truffle Suite y Ganache, resulta crucial avanzar con la configuración de estas herramientas. A continuación, se presentan los pasos necesarios para configurar correctamente Truffle y Ganache.

5.4.2.1. Configuración de la red en Ganache

En primer lugar, se debe iniciar una red Ethereum en Ganache. Una vez que Ganache esté en ejecución, se abre su interfaz principal. En este punto, se debe seleccionar la opción Nuevo Espacio de Trabajo (en inglés, *New Workspace*). En el Anexo III. se incluyen capturas de pantalla que proporcionan detalles específicos sobre la configuración de la red. Inmediatamente, se muestran una serie de pasos para la creación, compilación y despliegue de un contrato inteligente en la red iniciada.

5.4.2.2. Inicialización de Truffle

En segundo lugar, se debe inicializar Truffle en un nuevo directorio utilizando Visual Studio Code y su terminal integrada. Para ello, se debe seleccionar *truffle_setup* como el nombre para este directorio. Una vez dentro de este directorio a través de la terminal, se debe ejecutar el siguiente comando:

`truffle init`

Al ejecutar este comando, se inicializa un nuevo proyecto Truffle, generando la estructura de directorios básica y los archivos necesarios, como se ilustra en la Figura 15.

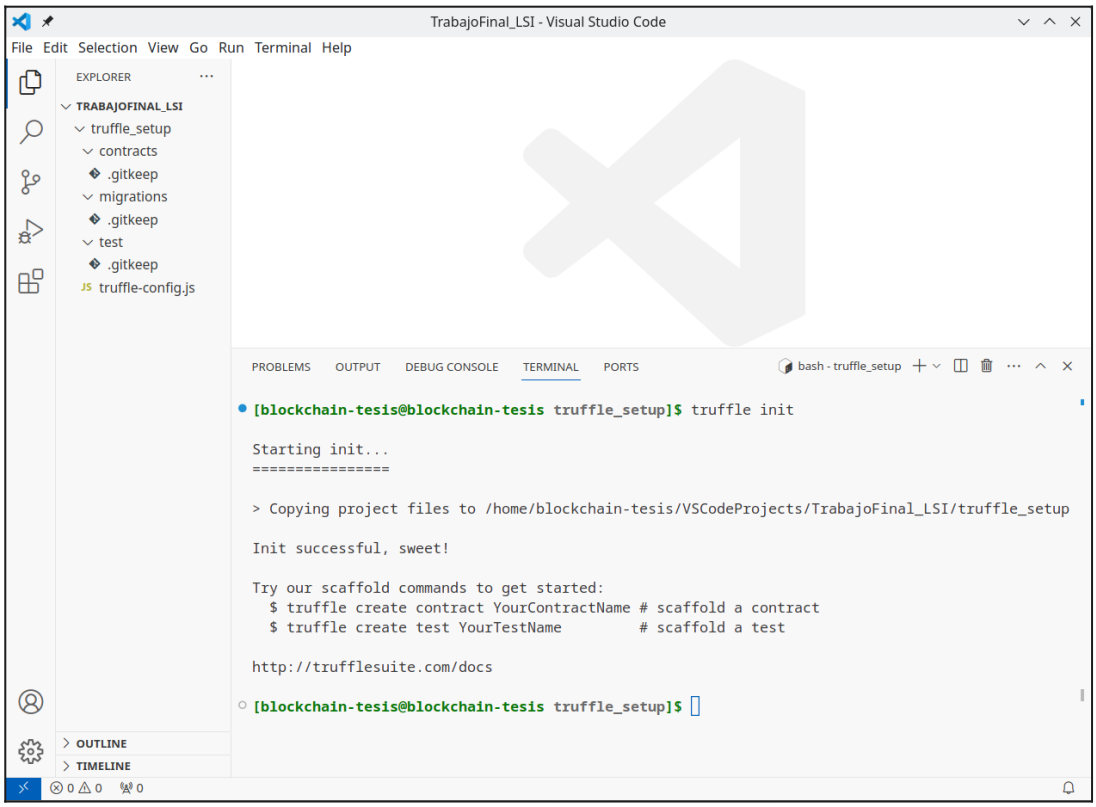

Figura 15: Inicializando Truffle. Fuente: Elaboración propia.

A continuación, se procede a configurar el archivo *truffle-config.js* para definir la red de desarrollo local. Es necesario descomentar y ajustar la sección *networks* para que coincida con el puerto de la red de Ganache, así como la sección *compilers* para especificar la versión del compilador Solidity. Seguidamente, se expone el código resultante:

```
networks: {
 ganache: {
  host: "127.0.0.1",
  port: 7545,
  network_id: "*",
 },
},

compilers: {
 solc: {
  version: "0.8.19",
 }
},
```

5.4.2.3. Creación del smart contract

Dentro del directorio *contracts*, se procede a crear un contrato utilizando el comando de Truffle:

```
truffle create contract PresentationStorage
```

A continuación, se procede a redactar el contrato inteligente, que fue definido en secciones anteriores, en el archivo *PresentationStorage.sol*. A modo de referencia, se presenta en la Figura 16 un ejemplo de cómo queda definido el contrato.

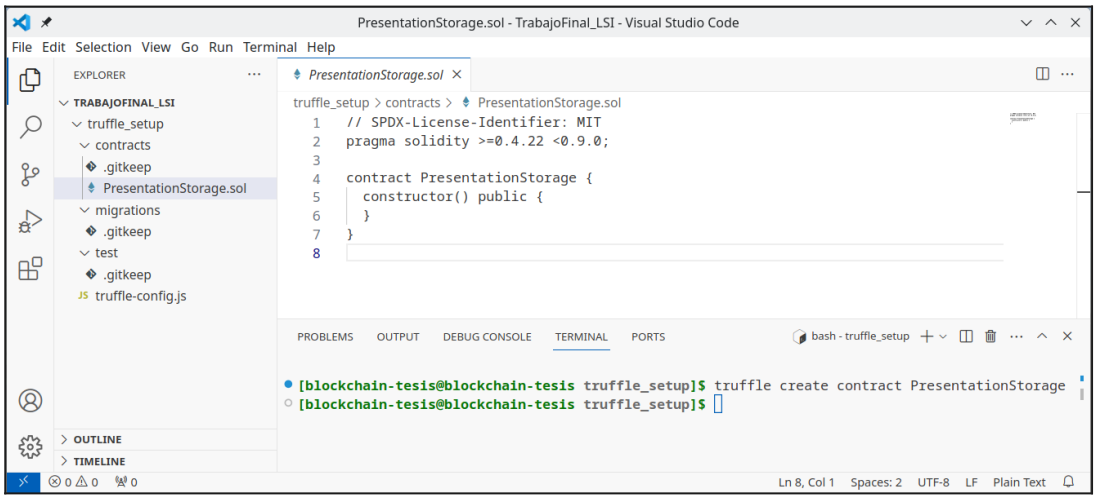

Figura 16: Creación de contrato. Fuente: Elaboración propia.

5.4.2.4. Compilación y despliegue del contrato

Después de redactar el contrato inteligente en el archivo *PresentationStorage.sol*, se debe proceder a compilarlo utilizando el siguiente comando Truffle:

```
truffle compile
```

Este comando inicia el proceso de compilación del contrato, verificando la sintaxis y generando los artefactos necesarios para su despliegue en la red Ethereum. Después de compilar el contrato con Truffle, se genera un directorio llamado *build* en la estructura de carpetas. Este directorio contiene un archivo *PresentationStorage.json* con información y artefactos relacionados con la compilación del contrato. La Figura 17 muestra cómo queda esta estructura después de la compilación.

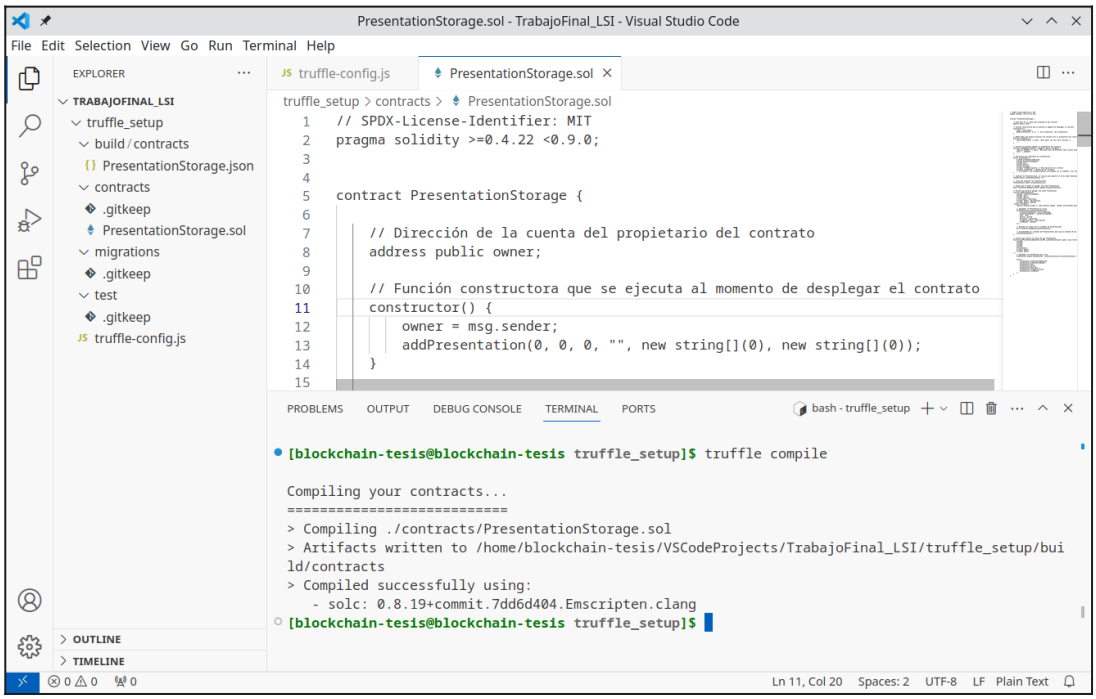

Figura 17: Compilación del contrato. Fuente: Elaboración propia.

Finalmente, se debe desplegar el contrato en la red Ethereum de Ganache previamente iniciada. Para llevar a cabo este despliegue, se deben seguir los pasos definidos a continuación.

En primer lugar, se debe crear un archivo dentro del directorio *migrations* denominado *1_initial_migration.js*. Este archivo debe contener el siguiente código:

```
const PresentationStorage =
artifacts.require("PresentationStorage");

module.exports = function (deployer) {
  deployer.deploy(PresentationStorage);
};
```

Este código utiliza el objeto *deployer* para gestionar el despliegue del contrato inteligente *PresentationStorage* en la red Ethereum de Ganache.

Posteriormente, se debe tener abierta y seleccionada la red creada en Ganache, para ingresar el siguiente comando en la terminal:

```
truffle migrate
```

Este comando inicia el proceso de migración, que ejecuta el código del archivo de despliegue y lleva a cabo la publicación del contrato en la red de Ganache. En la Figura 18 se ilustra la salida del comando ejecutado.

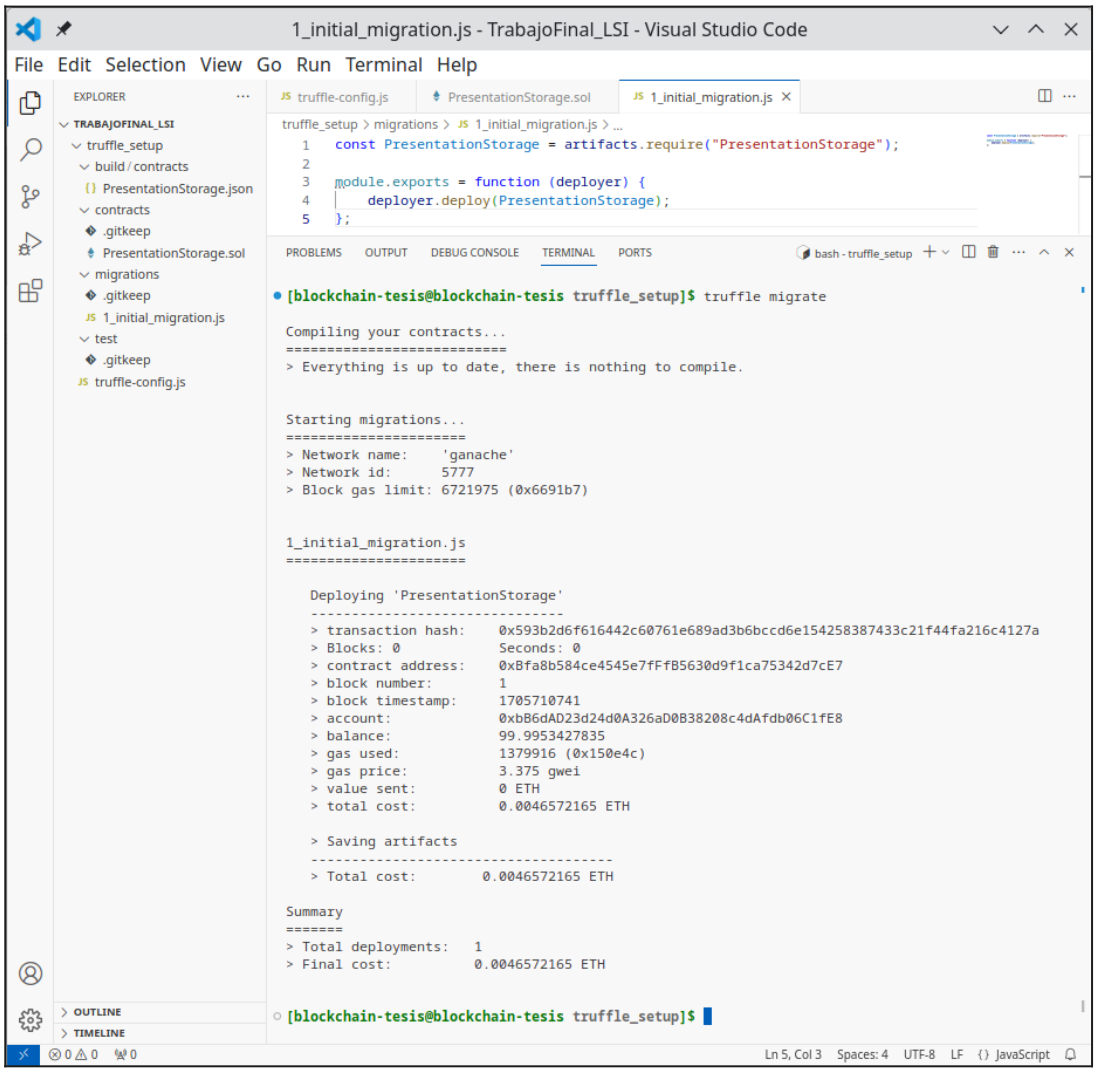

Figura 18: Despliegue del contrato. Fuente: Elaboración propia.

Una vez completada la migración, la información relevante sobre el contrato desplegado, incluyendo su dirección en la red se encuentra cargado en el archivo *PresentationStorage.json* ubicado dentro del directorio *build/contracts/*. Esta dirección es esencial para interactuar con el contrato desde el *backend* de la aplicación *web*.

Con estos pasos, se culmina el proceso de creación, compilación y despliegue de un contrato inteligente en la red Ethereum de Ganache. Si se verifica la aplicación de

Ganache, en las secciones *Blocks* y *Transactions* se puede visualizar una transacción con la etiqueta de creación de contrato (*Contract Creation*).

Como paso adicional, dentro de la aplicación de Ganache se puede vincular la configuración de Truffle en la sección de *Contracts*. Para ello, se debe seleccionar el archivo *truffle-config.js* del proyecto. La Figura 19 ilustra cómo queda configurado Ganache.

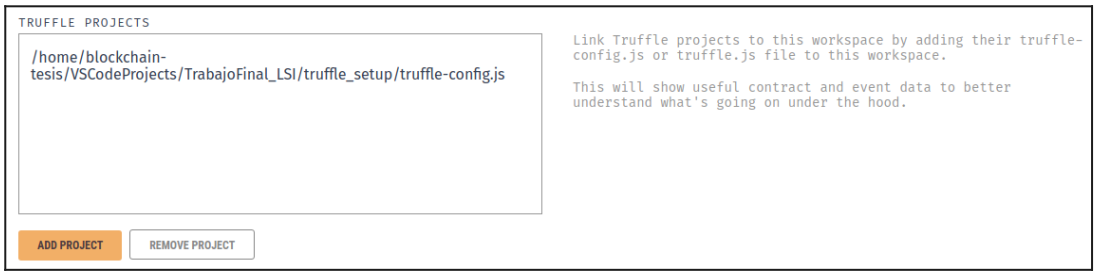

Figura 19: Vinculación de la configuración de Truffle en Ganache. Fuente: Elaboración propia.

Este último paso permite visualizar el contrato desplegado en la red y con ello cada operación que se vaya realizando.

5.5. Desarrollo de la aplicación web

En esta sección, se describe el desarrollo de la aplicación *web* que interactúa con el contrato inteligente desplegado en la red Ethereum de Ganache. Para ello, se utilizan las librerías Django y Web3.py mencionadas en las secciones anteriores.

5.5.1. Instalación de Django

Para iniciar el desarrollo, se debe proceder con la instalación de Django. Para ello, ya se debe contar con Python en el sistema, siendo Manjaro la plataforma de referencia. Caso contrario, deben ser instalados antes de continuar.

Con el objetivo de gestionar las dependencias de manera aislada y mantener la coherencia en el entorno de desarrollo, se debe proceder a crear un entorno virtual. Este entorno se establece a la misma altura del directorio donde se debe configurar Truffle, mediante la ejecución del siguiente comando:

```
python -m venv .venv
```

La Figura 20 muestra cómo debe quedar la estructura de directorio luego de ejecutar el comando en la terminal.

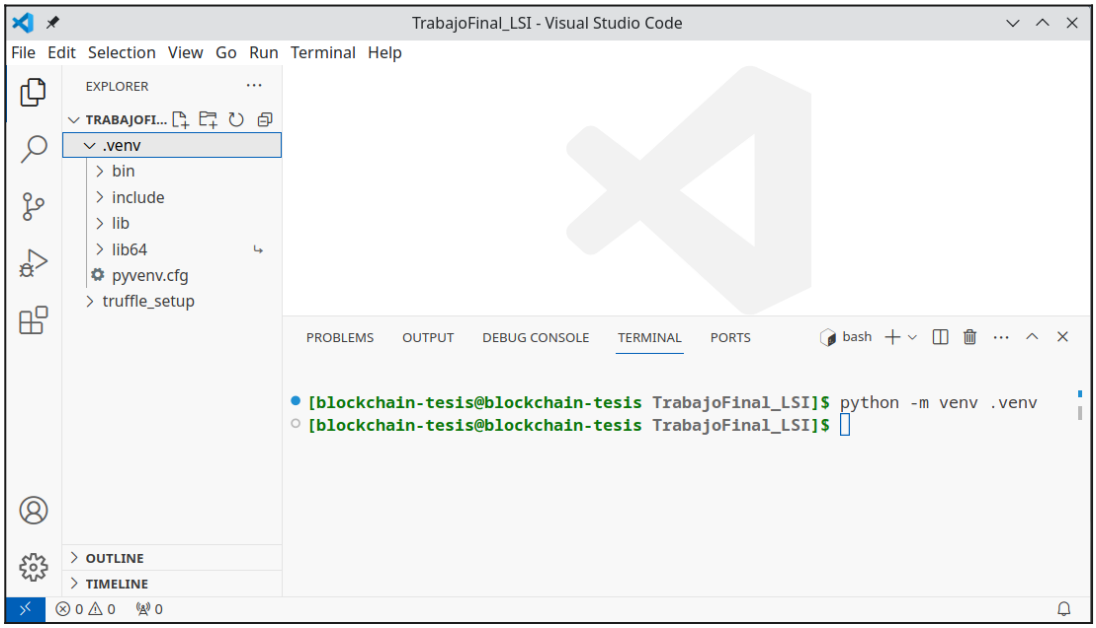

Figura 20: Creación de un entorno virtual. Fuente: Elaboración propia.

Antes de proceder a la descarga e instalación de Django se debe activar el entorno virtual. Para ello, asumiendo que el prompt de la terminal está ubicado en la raíz del proyecto se debe ejecutar el siguiente comando:

```
source .venv/bin/activate
```

El cambio en el prompt de la terminal, mostrando el nombre asignado al entorno virtual (*venv*), confirma que se está utilizando el entorno virtual. Esto se ilustra en la Figura 21.

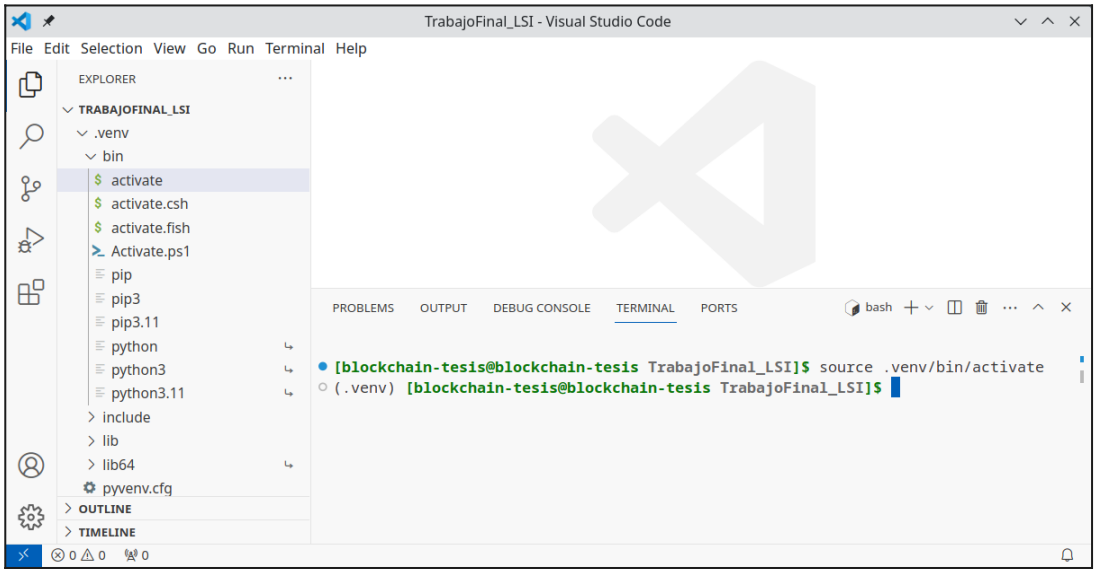

Figura 21: Activación del entorno virtual. Fuente: Elaboración propia.

Con el entorno virtual activado, se procede a instalar Django mediante el siguiente comando:

```
pip install django
```

La Figura 22 muestra la ejecución de este comando y resalta cómo se modifica el directorio */venv/bin* mediante la adición del archivo *django-admin*. Dicho archivo se utiliza en pasos posteriores para crear un nuevo proyecto Django.

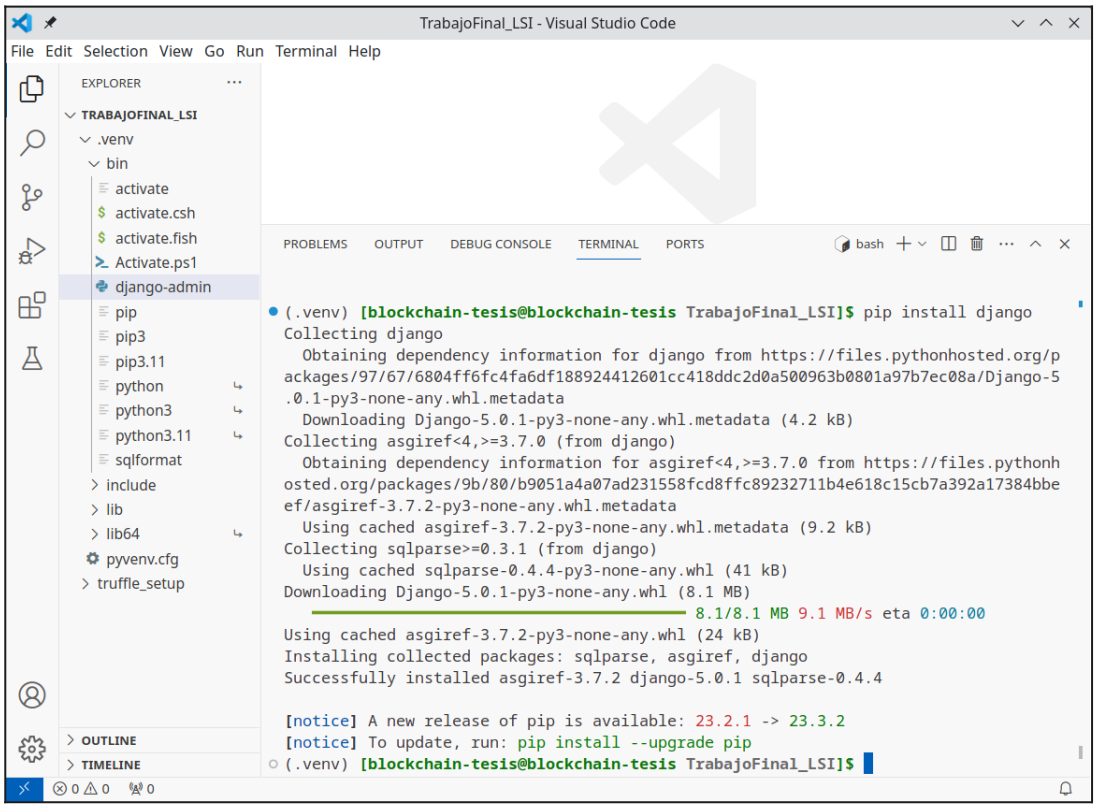

Figura 22: Instalación de Django. Fuente: Elaboración propia.

Además, es posible que Visual Studio Code sugiera la instalación de la extensión de Python. Se recomienda seguir esta sugerencia para garantizar un entorno de desarrollo integrado y facilitar el proceso de codificación.

El siguiente paso es iniciar un proyecto con Django con el siguiente comando:

```
django-admin startproject <nombre_del_proyecto>
```

El segundo argumento del comando indica cuál es el nombre del proyecto, en este caso se llama *django_setup* para distinguirlo de la configuración de Truffle. Luego de la ejecución del comando, es menester mover el *prompt* de la terminal dentro del directorio creado con el comando:

```
cd django_setup/
```

Asumiendo que el indicador de la terminal está ahora en el directorio *django_setup*, se inicia el servidor de desarrollo para confirmar que Django se ha instalado correctamente con el siguiente comando:

```
python manage.py runserver
```

En la Figura 23 se presenta la salida de los comandos anteriores y se verifica que el servidor de desarrollo se ha iniciado correctamente.

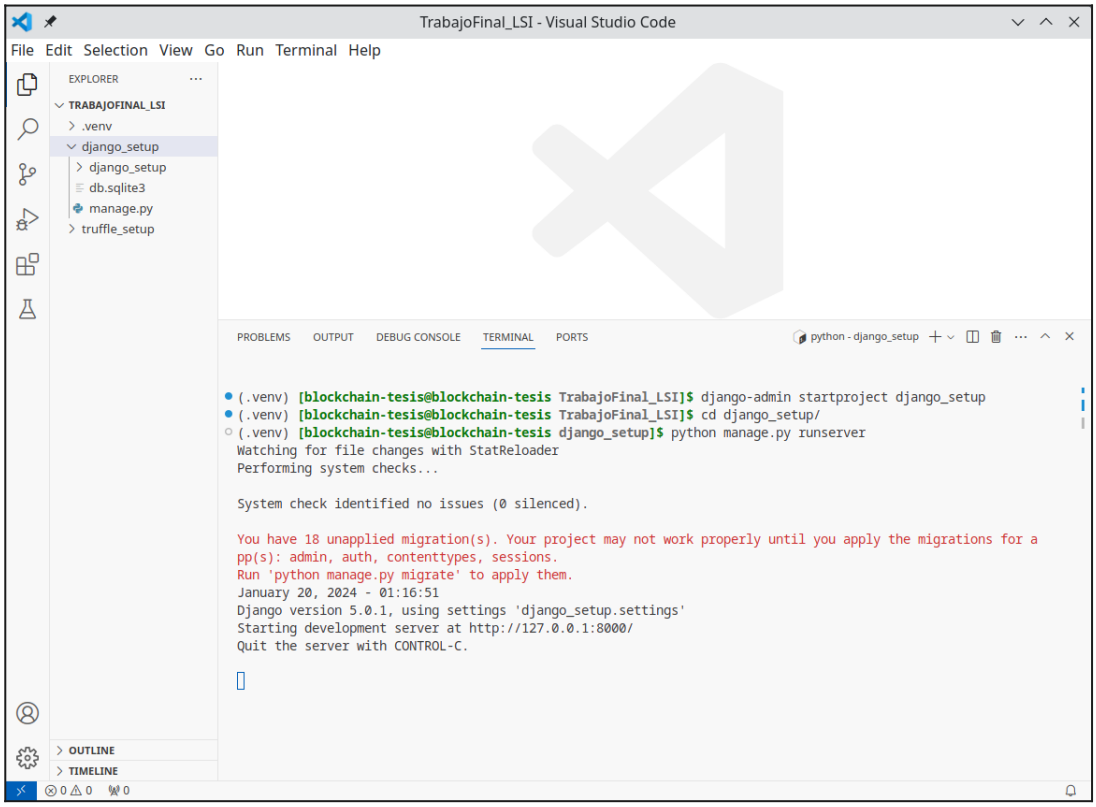

Figura 23: Inicio del servidor de desarrollo. Fuente: Elaboración propia.

Se verifica la correcta instalación accediendo a *http://127.0.0.1:8000/* en un navegador *web*.

Con estos pasos, se establece el entorno de desarrollo necesario para avanzar con el desarrollo de la aplicación *web*. La siguiente sección, presenta la integración de Web3.py para facilitar la interacción con el contrato inteligente desde la aplicación Django.

5.5.2. Instalación de Web3.py

Una vez configurado el entorno de desarrollo Django, se procede con la integración de Web3.py para facilitar la interacción con el contrato inteligente desde la aplicación Django. Con el entorno virtual creado anteriormente activado, se procede a instalar Web3.py mediante el siguiente comando:

```
pip install web3
```

La ejecución de este comando se muestra en la Figura 24, donde se observa parte de la instalación y además las dependencias de la librería Web3.py.

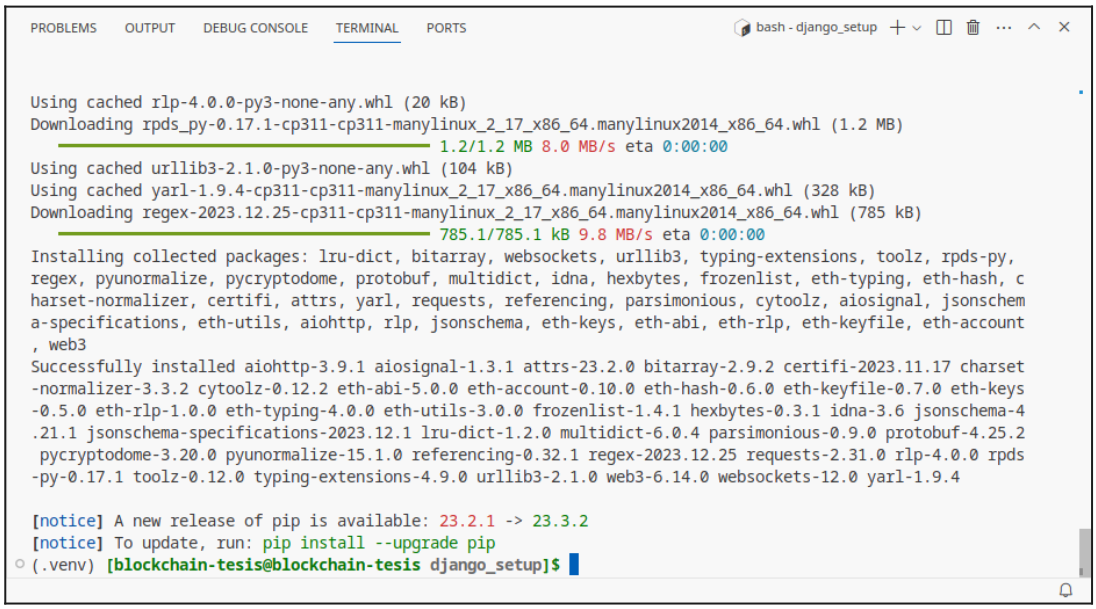

Figura 24: Instalación de Web3.py. Fuente: Elaboración propia.

Con Web3.py instalado correctamente, el siguiente paso es establecer una conexión a la red Ethereum desde la aplicación Django utilizando Web3.py.

5.5.3. Conexión a la red de Ganache

Para establecer una conexión con la red Ethereum de Ganache desde la aplicación Django, se debe realizar ajustes en el archivo de configuración principal del proyecto, *settings.py*. A continuación, se detallan las líneas añadidas al final del archivo mencionado, junto con una descripción.

```
import os
import json
from web3 import Web3

WEB3_PROVIDER_URI = "http://127.0.0.1:7545"
```

```
WEB3_CONTRACT_JSON_PATH = os.path.join(
  BASE_DIR,
"../truffle_setup/build/contracts/PresentationStorage.js
on"
)

with open(WEB3_CONTRACT_JSON_PATH, "r") as f:
  datastore_json = json.load(f)
  contract_abi = datastore_json["abi"]
  contract_address = datastore_json["networks"]["5777"]
["address"]

WEB3_CONNECTION =
Web3(Web3.HTTPProvider(WEB3_PROVIDER_URI))

WEB3_CONTRACT = WEB3_CONNECTION.eth.contract(
address=contract_address,
abi=contract_abi
)
```

El código comienza por definir la dirección del proveedor de servicios de la red Ethereum local, utilizando la variable WEB3_PROVIDER_URI, que apunta *http://127.0.0.1:7545*. Este URI indica la ubicación del nodo Ethereum al que se conecta.

A continuación, se especifica la ruta al archivo JSON generado durante la compilación del contrato con Truffle. La variable WEB3_CONTRACT_JSON_PATH se construye utilizando la función *os.path.join*,

concatenando la ruta base (*BASE_DIR*) con la ruta relativa al archivo del contrato (*../truffle_setup/build/contracts/PresentationStorage.json*).

El código utiliza la declaración *with open(WEB3_CONTRACT_JSON_PATH, "r") as f:* para abrir el archivo JSON del contrato en modo lectura (*r*). Dentro de este bloque, carga el contenido del archivo JSON en la variable *datastore_json* utilizando *json.load(f)*.

A continuación, se extrae la información necesaria del archivo JSON. Se obtiene la Interfaz binaria de aplicación (Application binary interface, ABI[11]) del contrato y la dirección del contrato en la red Ethereum local.

Luego, se establece una conexión a la red Ethereum local utilizando la dirección del proveedor de servicios (WEB3_PROVIDER_URI). Por medio de lo cual se crea una instancia de la clase Web3, con esta conexión que se almacena en la variable WEB3_CONNECTION.

Finalmente, se inicializa una instancia del contrato inteligente utilizando la dirección del contrato y la ABI obtenidos del archivo JSON. Esta instancia se almacena en la variable WEB3_CONTRACT y está lista para ser utilizada en la aplicación Python para interactuar con el contrato en la red Ethereum local.

Se presenta la Figura 25 de referencia de cómo queda el archivo *settings.py* luego del ajuste.

11 ABI (Interfaz Binaria de Aplicación) es el estándar para interactuar con contratos inteligentes en Solidity, tanto desde fuera como entre ellos. Especifica las reglas de codificación y decodificación de los datos de entrada y salida de las funciones del contrato [80].

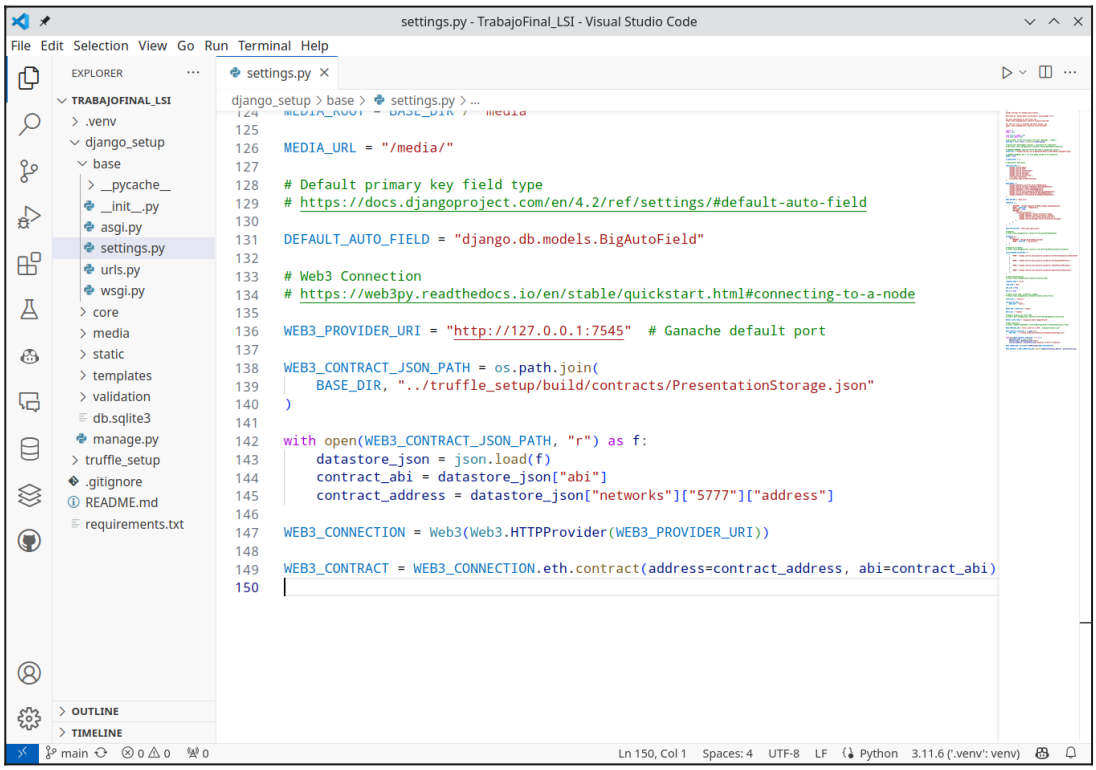

Figura 25: Ajuste del archivo settings.py. Fuente: Elaboración propia.

A partir de este momento, si se inicia el servidor de desarrollo con la aplicación de Ganache en segundo plano, se logra iniciar con éxito el proyecto en Django y se establece la conexión con la red de Ganache que se creó anteriormente.

5.5.4. Creación de aplicaciones en Django

En Django, se manejan dos conceptos fundamentales: *proyecto* y *aplicación*. Según la documentación de Django, *"una aplicación es una unidad de código web que realiza una función específica, como un sistema de blog, una base de datos de registros públicos o una aplicación de encuestas. Por otro lado, un proyecto es una colección de configuraciones y aplicaciones destinadas a un sitio web en particular, y puede contener múltiples aplicaciones, cada una de las cuales puede ser compartida entre varios proyectos"* [81]. Siguiendo esta estructura, se continúa con

la creación de aplicaciones que interactúen con el usuario y con el contrato inteligente desplegado.

Antes de proceder a la creación de las aplicaciones, es importante mencionar que se debió realizar un cambio en la estructura del proyecto renombrando el directorio que almacena las configuraciones del proyecto de *django_setup* a *base*. Este cambio incluye la actualización de los archivos *settings.py*, *asgi.py*, *wsgi.py* y *manage.py* [82].

La estructura del proyecto después de este cambio se presenta de la siguiente manera:

```
django_setup/
     base/
          __init__.py
          settings.py
          asgi.py
          wsgi.py
          urls.py
     manage.py
```

El cambio de nombre del directorio de *django_setup* a *base* se realizó para mejorar la legibilidad y la organización del proyecto. El nombre *base* resulta más intuitivo y da una mejor indicación de que este directorio contiene los archivos de configuración base del proyecto.

En este paso, se procede a crear dos aplicaciones clave para el proyecto Django: *core* y *validation*. Cada una de estas aplicaciones cumple un papel específico en la arquitectura.

5.5.4.1. Aplicación core

La aplicación *core* es el núcleo del proyecto, encargándose de gestionar los modelos de datos, las vistas y las plantillas. A continuación, se detallan los pasos para crear esta aplicación.

Se comienza por crear la aplicación *core* mediante el siguiente comando en la terminal:

```
python manage.py startapp core
```

Este comando genera automáticamente la siguiente estructura inicial de la aplicación *core*, que incluye varios directorios y archivos esenciales:

```
core/
     __init__.py
     admin.py
     apps.py
     migrations/
     __init__.py
     models.py
     tests.py
     views.py
```

Cada uno de estos archivos tiene un propósito específico en la aplicación:

- *__init__.py:* Este archivo vacío es crucial ya que le indica a Python que el directorio debe considerarse un paquete Python, lo que significa que es posible importar otros módulos y paquetes desde este directorio. Aunque está

vacío, su presencia permite la importación de módulos entre diferentes archivos de manera más fácil y ordenada.

- *admin.py:* En este archivo se registran los modelos para que aparezcan en la interfaz de administración de Django. Al registrar un modelo aquí, es posible agregar, editar y eliminar instancias de ese modelo a través de la interfaz de administración.
- *apps.py:* Este archivo se utiliza para la configuración de la aplicación. Aquí es donde Django guarda la configuración de la aplicación y donde es posible colocar cualquier configuración específica de la aplicación. Esto puede incluir la configuración de la base de datos, la configuración de la plantilla, la configuración de la ruta URL, etc.
- *migrations/:* Este directorio almacena los archivos de migración, que Django utiliza para realizar cambios en la base de datos. Las migraciones son la forma en que Django propaga los cambios hechos en los modelos (como agregar un campo, eliminar un modelo, etc.) en la base de datos. Este directorio está vacío al principio, pero a medida que se realicen cambios en los modelos y se creen migraciones, se llena con archivos correspondientes.
- *models.py:* Aquí es donde se definen los modelos de datos para la aplicación. Un modelo en Django es una representación de una tabla de base de datos construida con Python. Cada modelo se mapea a una sola tabla de base de datos y cada atributo del modelo se mapea a un campo de la tabla. Los modelos son el corazón de cualquier aplicación Django, ya que definen la estructura de los datos que la aplicación maneja.
- *tests.py:* Este archivo se utiliza para escribir pruebas para la aplicación en caso de ser requerido.
- *views.py:* Aquí es donde se manejan las solicitudes y respuestas de la aplicación. Una vista en Django es una función o clase que toma una solicitud *web* y devuelve una respuesta *web*. Esta respuesta puede ser el HTML de una página *web*, un redireccionamiento, un error 404, un archivo

XML, una imagen o cualquier otra cosa. Las vistas acceden a los datos necesarios para satisfacer las solicitudes a través de los modelos y delegan la presentación de los datos a las plantillas.

Una vez creada la aplicación, en el archivo *models.py* dentro del directorio *core* se definen los modelos de datos necesarios para el proyecto. Los modelos son esencialmente la estructura de la base de datos. Aquí, se especifica qué datos se almacena y cómo se relacionan entre sí.

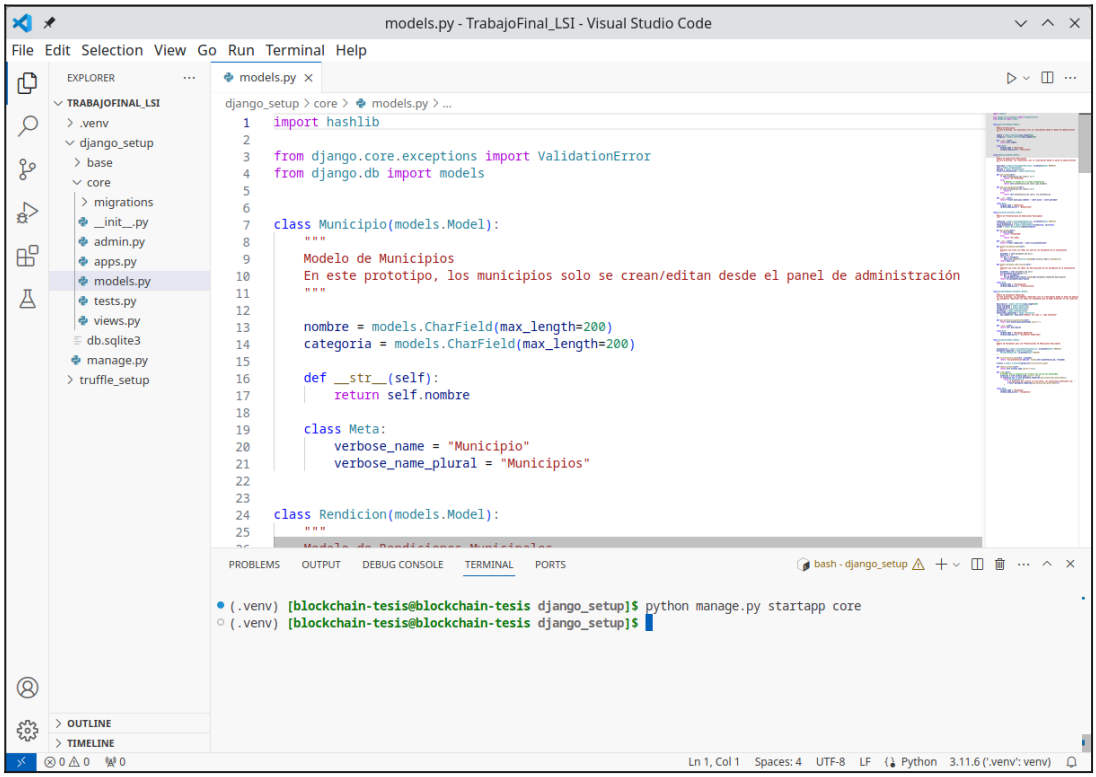

Figura 26: Referencia al archivo models.py. Fuente: Elaboración propia.

Tras definir los modelos, se procede a desarrollar las vistas en el archivo *views.py* de la aplicación *core*. Estas vistas gestionan las solicitudes de los usuarios interactuando con los modelos para recuperar o almacenar datos y determinar la respuesta adecuada para el usuario.

Para optimizar el código y hacerlo más legible, se introduce un nuevo archivo en el directorio *core* llamado *http_verbs.py*. Este archivo contiene funciones relacionadas con los métodos HTTP POST y GET. Estas funciones se utilizan en las vistas definidas en el archivo *views.py*, permitiendo así una reducción significativa del código dentro de las clases de vistas.

Adicionalmente, se crea la carpeta *templates* dentro del directorio *core/*, que almacena los archivos HTML que se utilizan para renderizar las páginas *web*. Estos *templates* definen cómo se presentan los datos al usuario. Cada vista puede estar asociada a uno o más *templates*, dependiendo de la complejidad de la página *web*.

Además de los *templates* específicos de cada vista en la aplicación *core*, es útil utilizar la herencia de plantillas para reutilizar partes comunes para las páginas *web*. Para ello, se crean plantillas base en la raíz del proyecto, dentro de una carpeta llamada también *templates*, siguiendo la convención de Django. Estas plantillas base contienen elementos comunes como importaciones de estilos, la cabecera, el pie de página, la barra de navegación, entre otros. Luego, cada *template* específico de una vista puede extender una plantilla base y rellenar las partes que varían.

En un proyecto Django, los archivos estáticos son aquellos que no están destinados a cambiar durante la ejecución del programa, como las hojas de estilo CSS, las imágenes y los archivos JavaScript. Estos archivos se almacenan en un directorio llamado *static* dentro de cada aplicación o en la raíz del proyecto. Para que Django pueda utilizar estos archivos estáticos, se debe configurar la variable STATIC_URL en el archivo *settings.py*. Esta variable indica la URL que se utiliza para referenciar los archivos estáticos en las plantillas [83].

En la raíz del proyecto, se crean dos directorios importantes: *static* y *media*.

El directorio *static* se utiliza para almacenar los archivos estáticos, que incluyen bibliotecas y *frameworks* de CSS y JavaScript. Dentro de este directorio, se utiliza npm para gestionar y mantener las dependencias del *frontend*, como bibliotecas y *frameworks* de JavaScript. Para descargar las dependencias especificadas en el archivo *package.json* ubicado dentro del directorio, se debe ubicar el *prompt* de la terminal en el directorio *static* y ejecutar el siguiente comando:

```
npm install
```

Este comando instala todas las dependencias declaradas en un nuevo directorio llamado *node_modules*.

Por otro lado, el directorio *media* tiene un propósito diferente. Este directorio se utiliza para almacenar los archivos que son cargados por los usuarios durante el desarrollo. Estos pueden incluir imágenes, documentos y cualquier otro tipo de archivo multimedia que un usuario pueda subir a través de la aplicación. Para que Django pueda servir estos archivos estáticos, se debe configurar la variable MEDIA_ROOT en el archivo *settings.py*.

Finalmente, las URLs de la aplicación *core* se definen en un archivo llamado *urls.py* que se encuentra en el directorio *base/*. Este archivo define las rutas URL para las diferentes vistas de la aplicación. Cada ruta URL se asocia con una vista específica, y cuando un usuario solicita esa URL, Django invoca la vista asociada para manejar la solicitud. Por lo tanto, el archivo *urls.py* actúa como un mapa que dirige las solicitudes entrantes a las vistas correctas.

A continuación, se expone la estructura de archivos del proyecto Django que se obtiene luego de las configuraciones y modificaciones realizadas.

```
django_setup/
     base/
          __init__.py
          settings.py
          asgi.py
          wsgi.py
          urls.py
     core/
          migrations/
               __init__.py
          templates/
          __init__.py
          admin.py
          apps.py
          forms.py
          http_verbs.py
          models.py
          tests.py
          views.py
     media/
     static/
          node_modules/
          package-lock.json
          package.json
     templates/
          root.html
     manage.py
```

5.5.4.2. Aplicación validation

La aplicación *validation* se dedica a validar las presentaciones realizadas por usuarios por medio de la conexión con la red *blockchain* de Ganache. Esta facilita la extracción de información sobre las presentaciones desde la red Ganache, incluyendo detalles como el *hash* único de cada archivo, datos de la rendición asociada, la fecha y hora de presentación, entre otros datos. A continuación, se detallan los pasos para crear esta aplicación y su estructura básica.

Se inicia creando la aplicación *validation* mediante el ingreso del siguiente comando en la terminal:

```
python manage.py startapp validation
```

Este comando genera automáticamente la estructura inicial de la aplicación *validation*, similar a la creada para la aplicación *core*. En este caso solamente se mantiene los archivos *__init__.py*, *apps.py* y *views.py*. Además, se agrega el directorio llamado *templates* para almacenar los archivos HTML destinados a renderizar las vistas definidas. La estructura de directorios resultante es la siguiente:

```
validation/
     templates/
     __init__.py
     apps.py
     views.py
```

Bajo esta configuración simplificada, se desarrollan las vistas en el archivo *views.py* de la aplicación *validation*. Estas vistas se encargan de gestionar la lógica de validación y se comunican con la red *blockchain* de Ganache para garantizar la integridad de las presentaciones realizadas por los usuarios.

Además, el directorio *templates* es utilizado para almacenar los archivos HTML necesarios para la representación visual de las vistas en la interfaz de usuario. A medida que se avance en el desarrollo, este directorio se llena con plantillas específicas que facilitan la presentación de la información obtenida de la red de Ganache.

Con el desarrollo de ambas aplicaciones, se procede a validar la propuesta desarrollada utilizando conjuntos de datos reales o ficticios con un alto grado de similitud en lo que se refiere a los tipos de archivos y tamaño, para identificar posibles áreas de mejora. Las vistas y detalles de ambas aplicaciones se presentan en el Anexo IV.

Capítulo 6

Resultados Obtenidos

En el presente capítulo, se exhiben los resultados obtenidos tras el desarrollo y ejecución de la solución propuesta. Se describe la preparación del entorno y el funcionamiento del *smart contract* para la ejecución de las pruebas.

6.1. Preparación de datos

Antes de iniciar los ensayos, es necesario poblar la base de datos de los modelos que configuran el proyecto, es decir, los Municipios, las Rendiciones y los Documentos Requeridos. El primer paso para lograr esto es crear la base de datos, utilizando los modelos definidos en la aplicación *core*. Para ello, hay varios comandos que Django ofrece para interactuar con las migraciones y el manejo del esquema de la base de datos, entre ellos *makemigrations* y *migrate*.

El comando *makemigrations* se utiliza para crear migraciones de los modelos, es decir crea un conjunto de instrucciones para Django, detallando cómo cambiar la base de datos para que se refleje los cambios realizados en los modelos. Por otro lado, el comando *migrate* se utiliza para aplicar y deshacer las migraciones, es decir es el comando que efectivamente aplica las migraciones a la base de datos para que coincida con los modelos.

Es importante notar que antes de usar *migrate*, se deben crear las migraciones para los modelos utilizando el comando *makemigrations*. A continuación, se muestra cómo ejecutar el comando para crear y aplicar las migraciones:

```
python manage.py makemigrations
```

Una vez que se han creado las migraciones dentro del directorio *core/migrations/*, se aplican con el comando *migrate*. Para aplicar las migraciones pendientes y crear la base de datos, se debe ejecutar el siguiente comando en la raíz del proyecto:

```
python manage.py migrate
```

Para llevar a cabo este proceso, el comando *migrate* analiza los modelos definidos en la aplicación *core* y genera las tablas correspondientes en la base de datos. En este contexto, la base de datos predeterminada en Django es SQLite, y crea un archivo llamado *db.sqlite3* en la raíz del proyecto para almacenar estas tablas. Además, se emplea DbVisualizer [84], una herramienta especializada en la administración de bases de datos y generación de modelos relacionales, para visualizar y comprender la estructura resultante.

La Figura 27 exhibe el modelo relacional específico de las tablas de la aplicación *core*. Es importante destacar que, en la representación gráfica, se han omitido las tablas generadas por Django que aunque juegan un papel fundamental en el funcionamiento integral del proyecto no son pertinentes a los objetivos de este trabajo, como las relacionadas a usuarios, grupos, permisos, migraciones, entre otras.

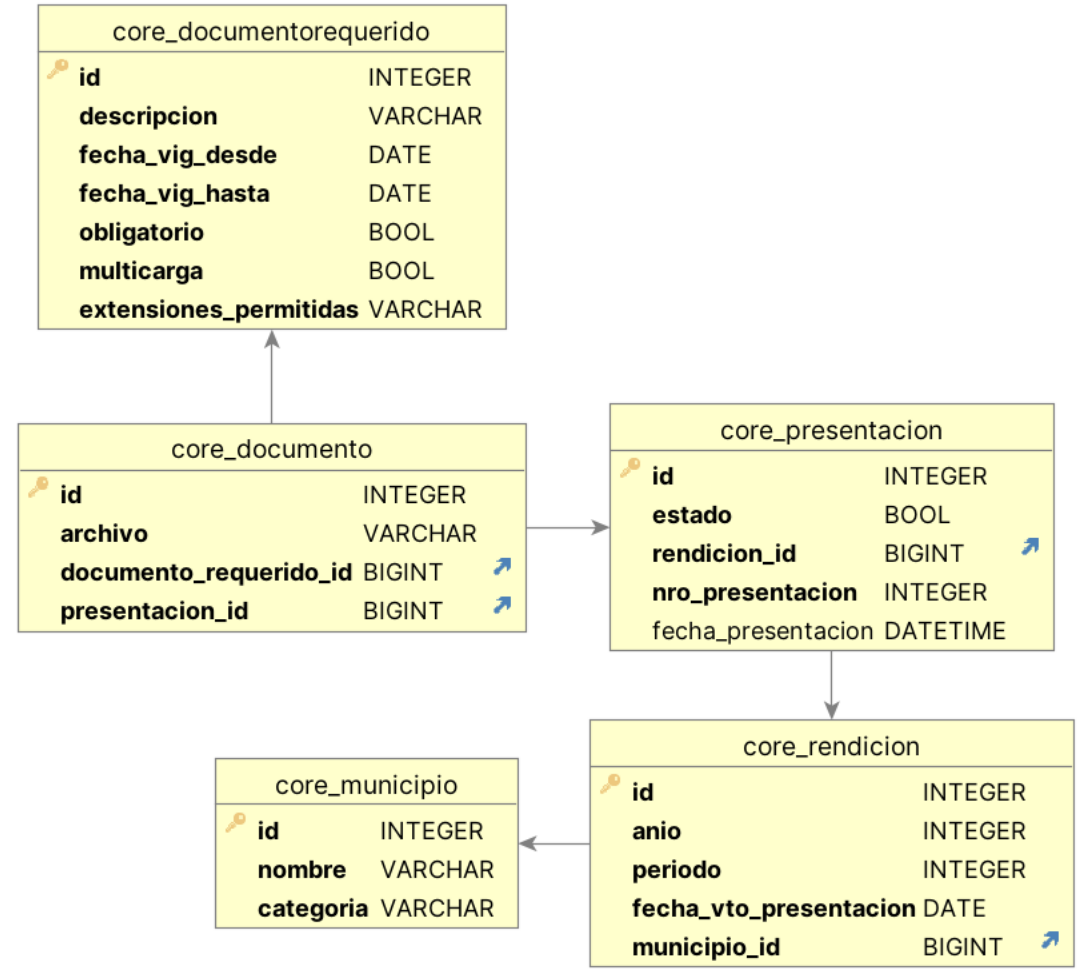

Figura 27: Modelo relacional de la aplicación core. Fuente: Elaboración propia.

Django facilita la carga inicial de datos a través de archivos llamados *fixtures*. Estos archivos contienen datos en un formato específico (JSON, XML, YAML, etc.) que Django puede procesar e insertar en la base de datos. Para este proyecto se crea un archivo JSON dentro del directorio *core/fixtures/* con datos de ejemplo para los Municipios, las Rendiciones y los Documentos Requeridos. Se crea un archivo para cada tabla obteniendo de ese modo los archivos *municipios.json*, *rendiciones.json* y *documentorequeridos.json*.

Para cargar los datos en la base de datos, se deben ejecutar el siguiente comando:

```
python manage.py loaddata core/fixtures/*.json
```

La ejecución de los comandos mencionados, presenta en la terminal una salida similar a como se ilustra en la Figura 28. En el caso que las migraciones ya estén almacenadas en el directorio *core/migrations/*, si no hay cambios en los modelos, la salida del comando *makemigrations* es un mensaje describiendo que no se detectan cambios.

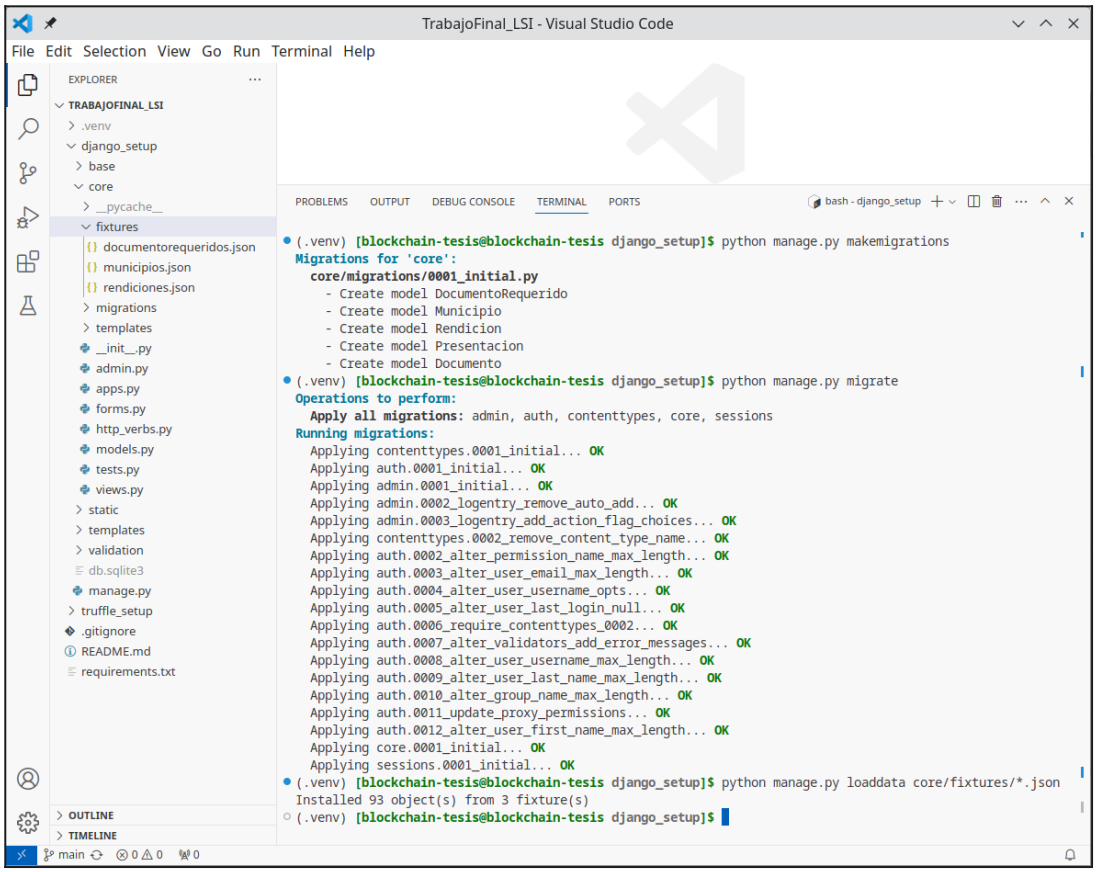

Figura 28: Creación de la base de datos y carga de datos. Fuente: Elaboración propia.

A partir de este momento es posible utilizar la interfaz de usuario de la aplicación *web* que se ha desarrollado con Django para interactuar con el contrato inteligente y

así agregar nuevas presentaciones a la base de datos de la aplicación y a la red *blockchain* de Ganache.

6.2. Funcionalidad del smart contract

Esta sección detalla las pruebas de funcionamiento del *smart contract* ejecutadas siguiendo el flujo estándar de la aplicación *web*. Las pruebas se realizan utilizando la cuenta del propietario del *smart contract* con documentos que poseen extensiones tales como *.pdf*, *.zip*, *.rar* y *.xlsx*. Es importante destacar que la aplicación Ganache debe encontrarse en ejecución para el correcto funcionamiento del contrato.

Al iniciar el servidor de Django, como se muestra en la Figura 29, la primera interfaz presenta una lista de períodos por municipio para que se pueda proceder con la presentación de la rendición municipal.

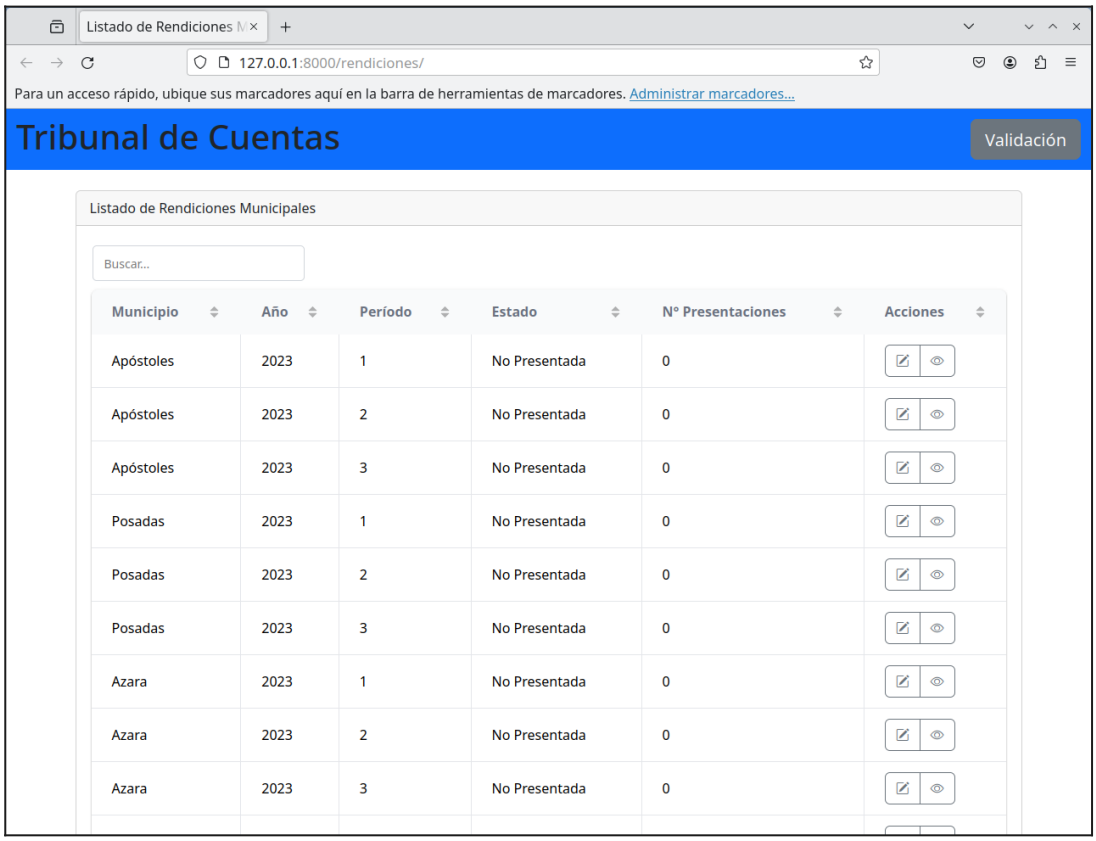

Figura 29: Listado de rendiciones municipales ordenadas por municipios. Fuente: Elaboración propia.

El flujo de uso normal es seleccionar un período de la lista para llevar adelante la presentación digital de los documentos requeridos. A continuación, como se ilustra en la Figura 30 el usuario puede avanzar con la carga de los archivos correspondientes.

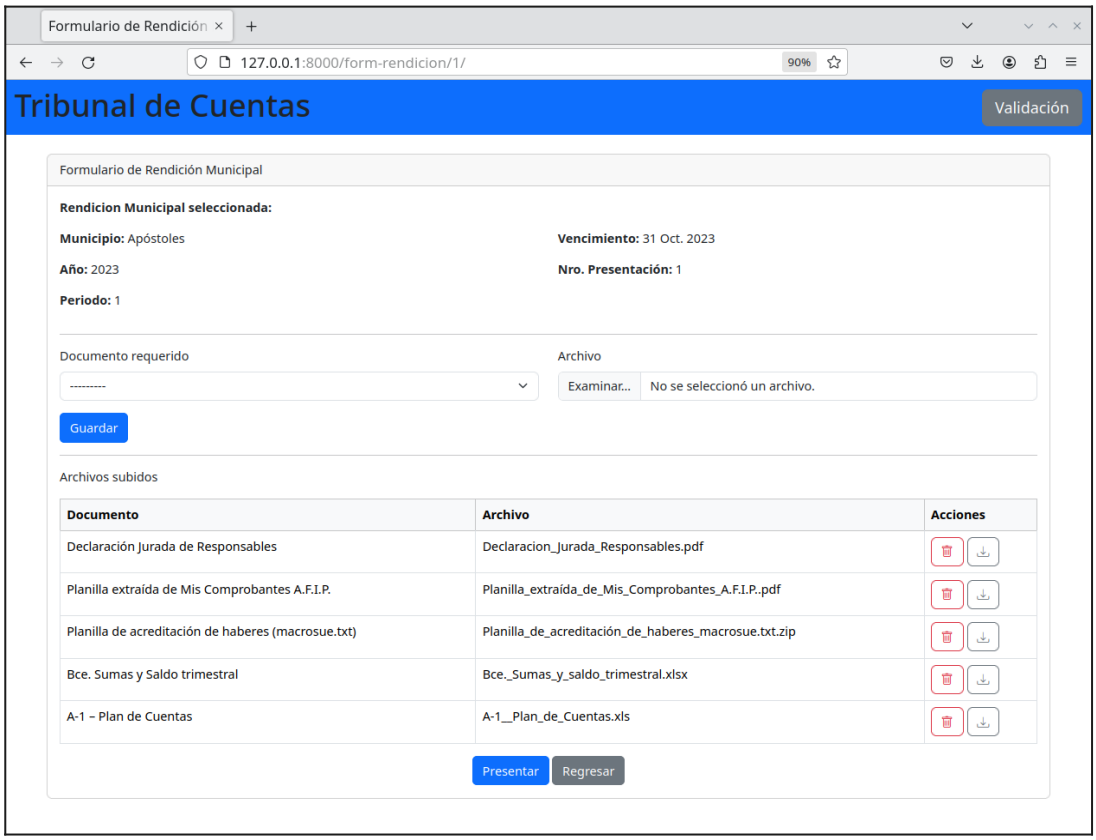

Figura 30: Interfaz que muestra la selección y carga de documentos a la rendición municipal seleccionada. Fuente: Elaboración propia.

Una vez que el usuario haya cargado todos los documentos definidos ya puede avanzar con la presentación haciendo clic en el botón *Presentar.*

Una vez presionado el botón *Presentar* el usuario visualiza una pantalla que expone el comprobante de la operación con los datos de la transacción en la *blockchain*. Estos datos son esenciales para validar la presentación en el futuro con la aplicación *validation*. A continuación, se expone por medio de la Figura 31 un ejemplo del comprobante que puede visualizar el usuario.

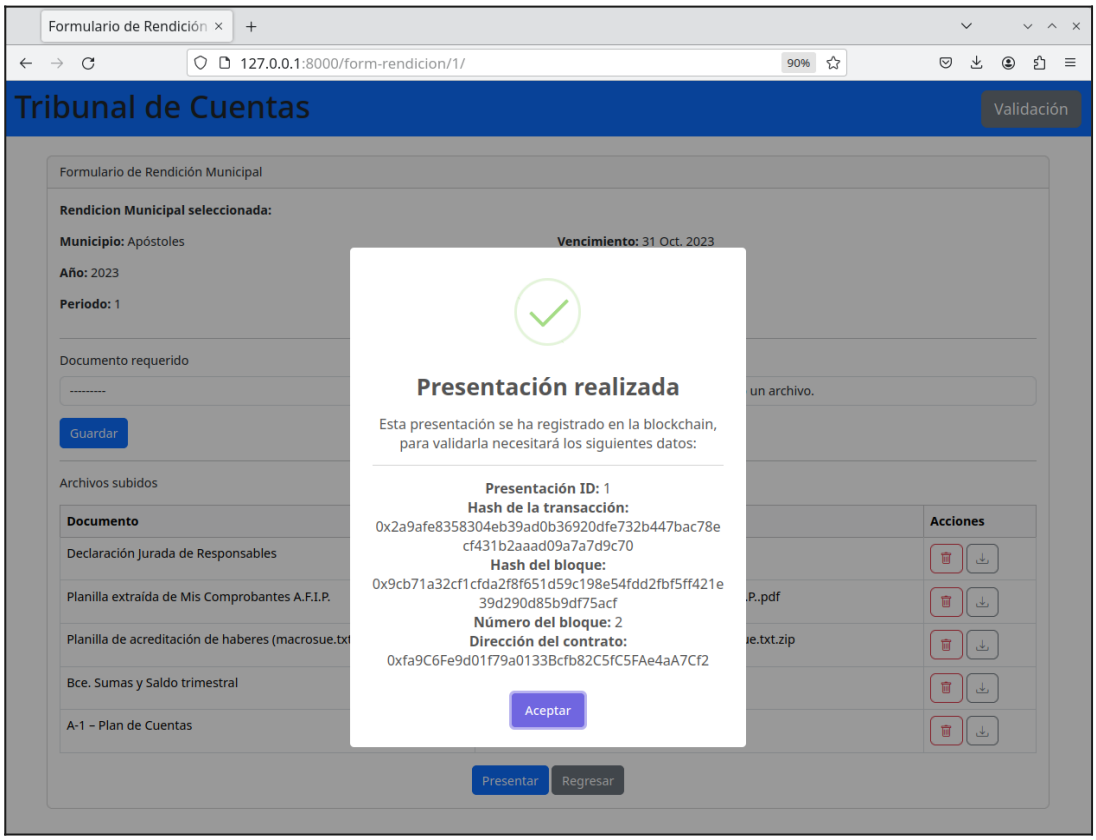

Figura 31: Comprobante de operación que muestra los datos de la transacción en la blockchain. Fuente: Elaboración propia.

En la aplicación Ganache, dentro de la sección *Contratos*, existe la opción de visualizar el contrato desplegado para acceder al detalle de *Almacenamiento*. Tal como se ilustra en la Figura 32, es posible visualizar y verificar el contenido de las variables del contrato, incluyendo la lista de presentaciones. El último elemento de esta lista corresponde a los datos de la presentación más reciente. Si estos datos están presentes, indica que la carga en la *blockchain* se realizó con éxito. Este proceso es una confirmación visual de que la transacción se ha completado correctamente.

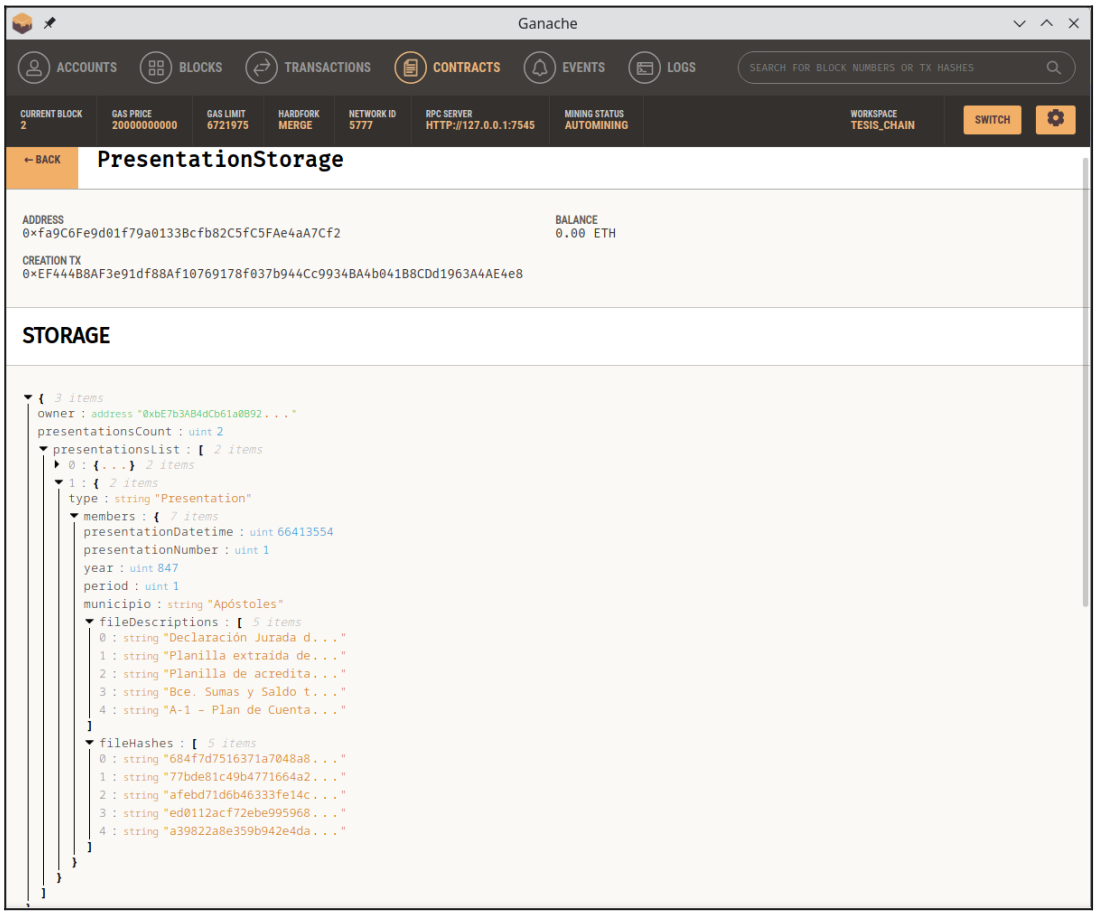

Figura 32: Almacenamiento del contrato con la interfaz de usuario de Ganache. Fuente: Elaboración propia.

Con la presentación cargada en la *blockchain*, la aplicación *validation* permite obtener los datos utilizando el ID de la presentación o el *hash* de la transacción, proporcionados en el comprobante al momento de la presentación. La interfaz que recupera estos datos se diseñó utilizando los métodos de la librería Web3.py para llamar a la función del contrato denominada *getPresentationByCount(presentationCount)*, pasando el ID de la presentación como argumento y obteniendo los datos correspondientes. También es posible utilizar los métodos de la librería Web3.py para buscar eventos dentro de una transacción, utilizando el *hash* único de la transacción. Este evento devuelve los datos de la transacción y el ID de la presentación creada en esa transacción.

En la Figura 33 se ilustra el formulario de la aplicación *validation*. Dicha interfaz presenta una sección de búsqueda que incluye dos campos y cada uno de ellos se encuentra acompañado de un botón que permite lanzar la búsqueda. En el primer campo, se puede ingresar el ID de la presentación, activando posteriormente la función que busca la presentación dentro del contrato inteligente. El segundo campo, está destinado a la búsqueda por el *hash* de la transacción.

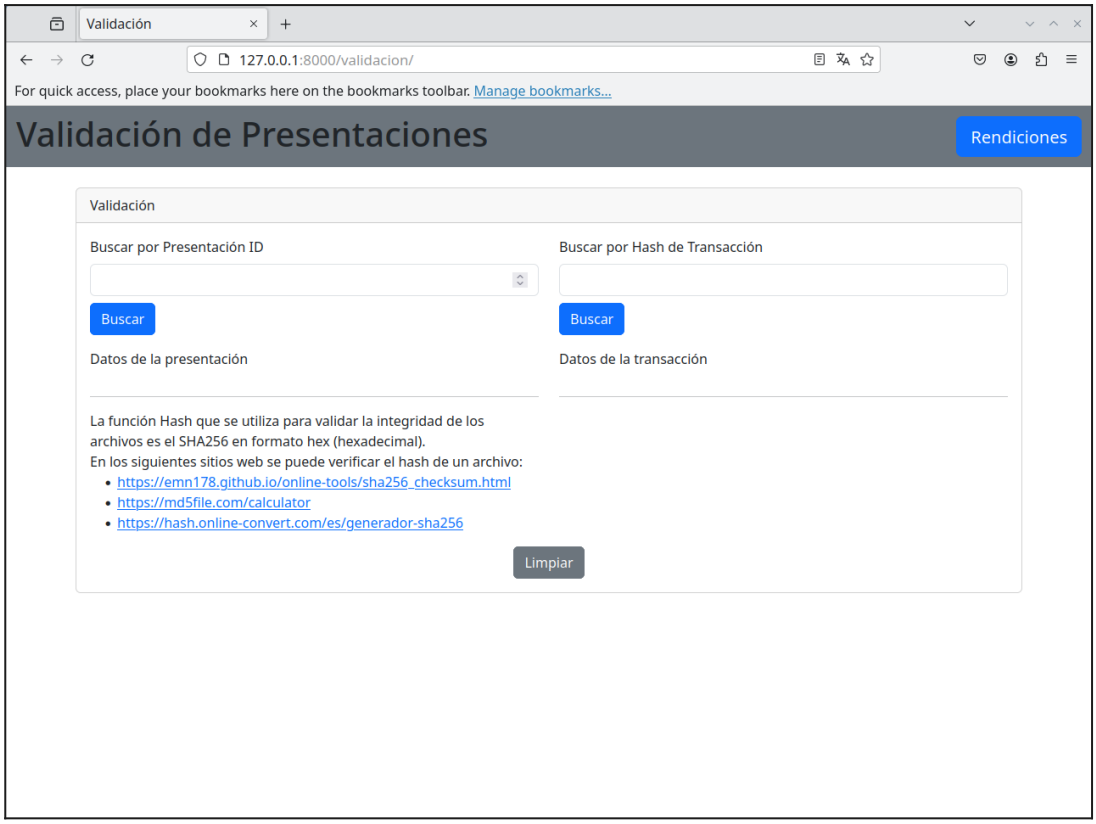

Figura 33: Interfaz de usuario de la aplicación validation. Fuente: Elaboración propia.

Al llevar adelante la búsqueda por medio del primer campo, tal y como se ilustra en la Figura 34, se puede recuperar los detalles de la presentación asociada, tales como la fecha de presentación, el número de la presentación, el período y año de la

rendición, el municipio y los *hashes* de los documentos en formato hexadecimal identificados por su descripción.

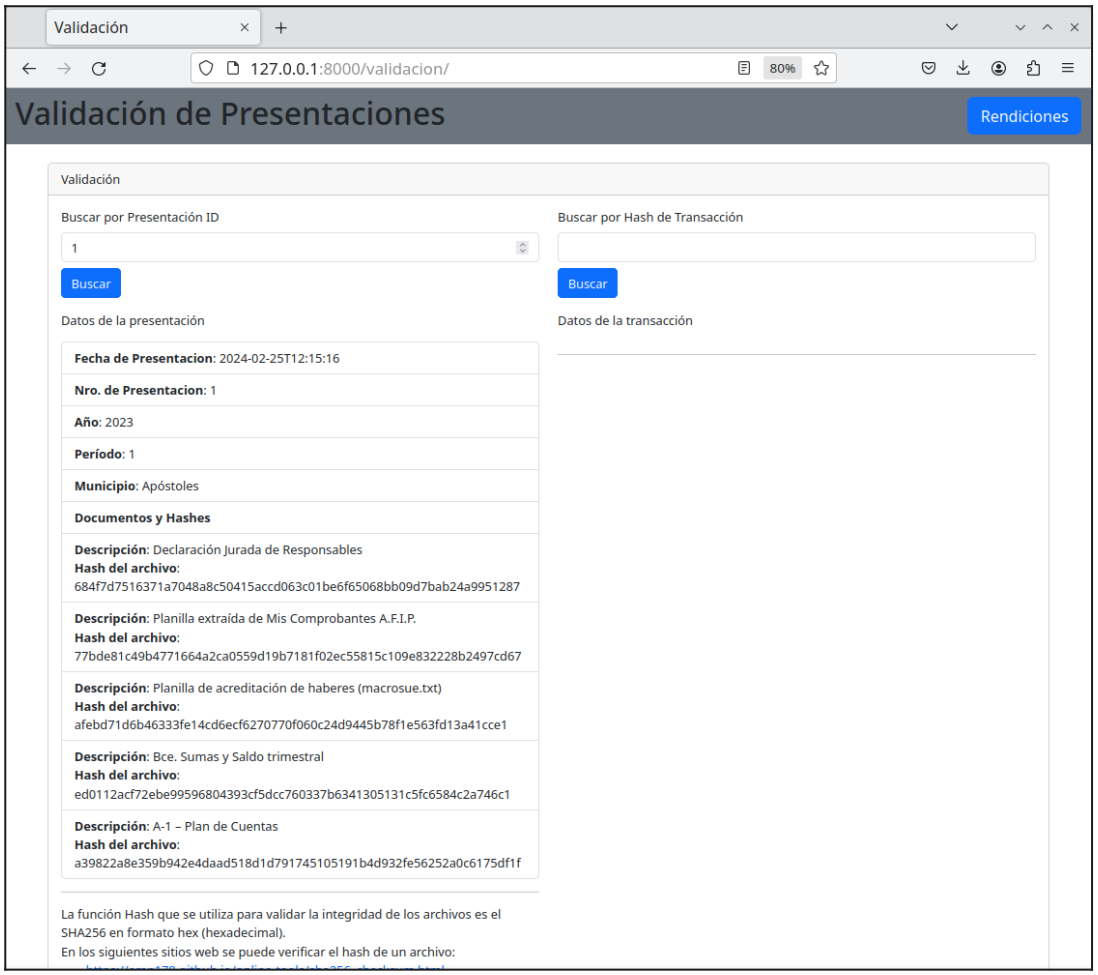

Figura 34: Búsqueda de presentación por medio de su ID. Fuente: Elaboración propia.

En caso de que la búsqueda se realice por medio del ingreso del *hash* de la transacción, como se muestra en la Figura 35, se puede recuperar información detallada sobre la transacción correspondiente, tales como el ID de la presentación agregada, el *hash* y número del bloque donde se aloja la transacción y la dirección del contrato inteligente.

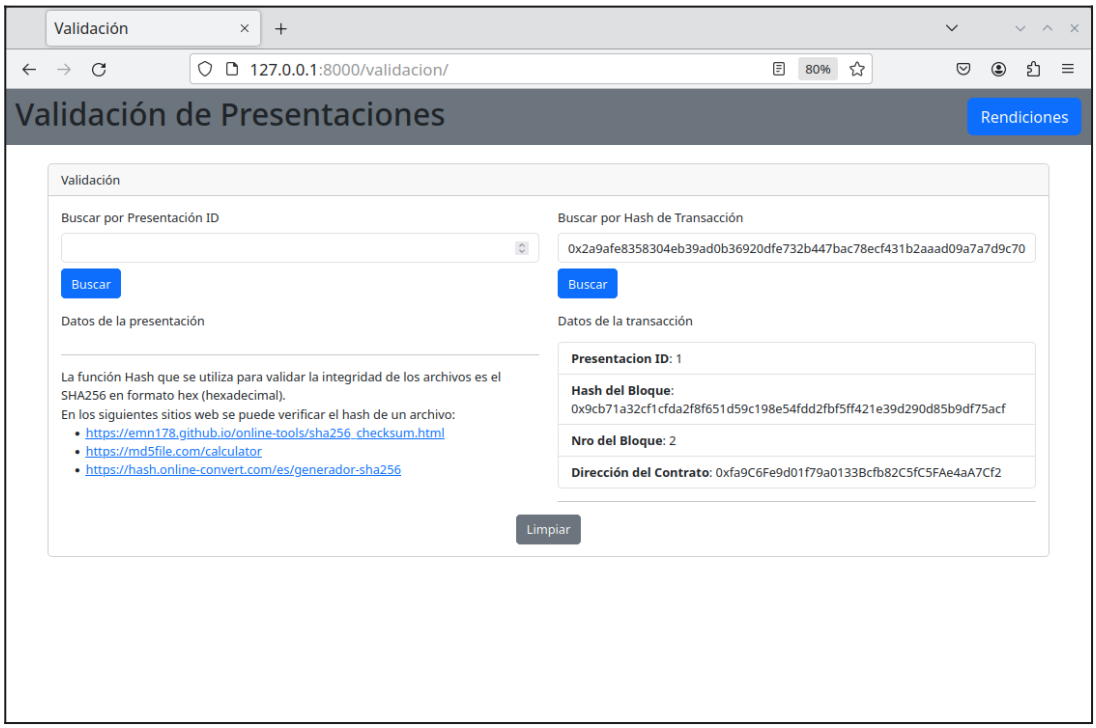

Figura 35: Búsqueda y resultados por hash de transacción. Fuente: Elaboración propia.

Es importante destacar que la verificación de esta interfaz se lleva a cabo mediante los datos recibidos del comprobante, que se muestra en la Figura 31, y los resultados obtenidos de ambas búsquedas se presentan en la Figura 36. En el caso de no encontrarse una correspondencia en las búsquedas, se muestra al usuario una alerta que contiene la descripción del error.

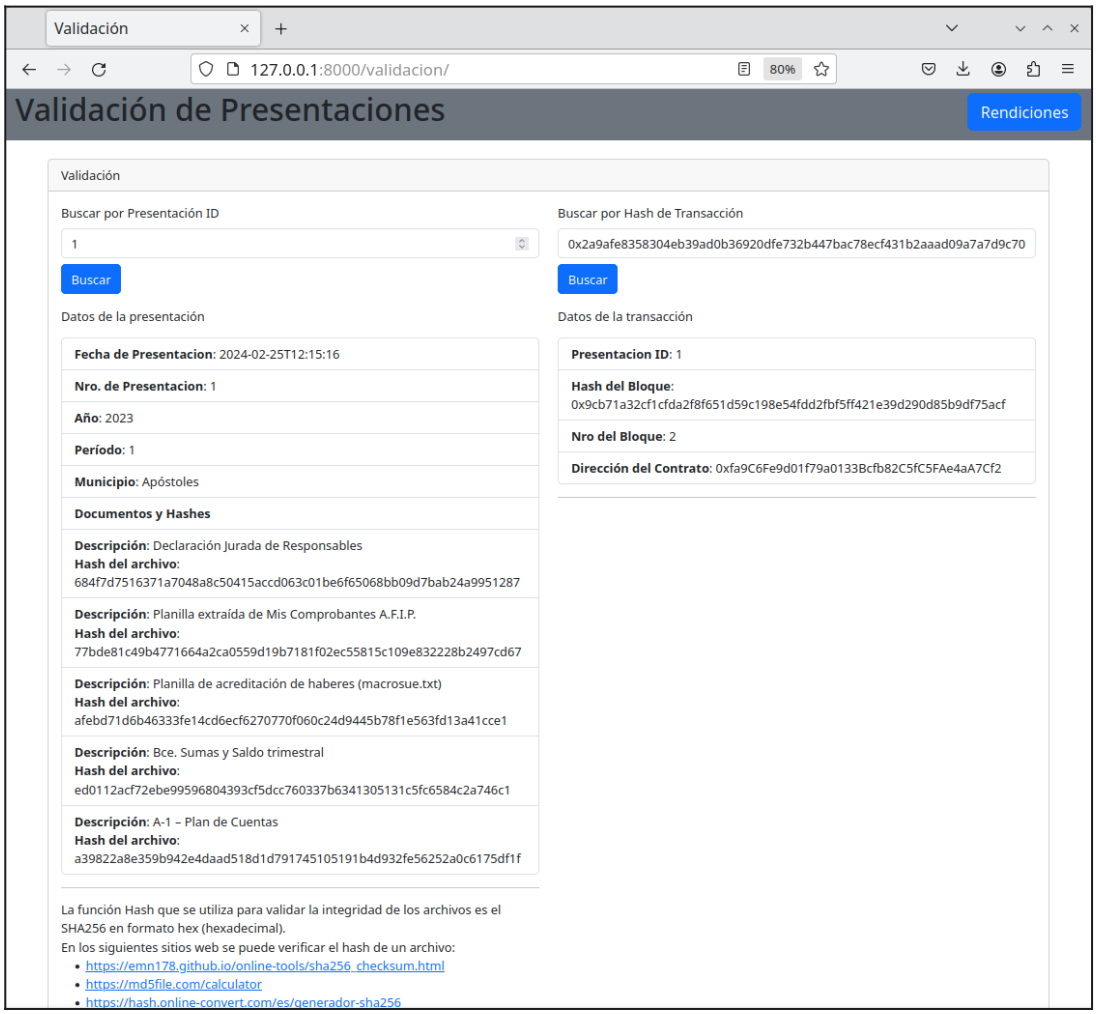

Figura 36: Resultados de ambas búsquedas en la aplicación validation. Fuente: Elaboración propia.

La demostración realizada confirma el adecuado funcionamiento del *smart contract* y su integración al prototipo de aplicación *web*. Es importante destacar que la utilización de un único contrato inteligente permite definir tanto su dirección como el ABI del contrato en el archivo *settings.py* del proyecto, garantizando así la coherencia en el uso de la misma red y contrato entre las aplicaciones. Esta configuración es clave, especialmente al operar en una red *blockchain* de pruebas local, ya que es esencial que el contrato inteligente empleado se sincronice con la base de datos de la aplicación *web*. Cabe mencionar que cualquier nuevo contrato

desplegado en Ganache a partir de este punto inicia sin registros previos de presentaciones, lo cual es crucial considerar durante el desarrollo y pruebas subsiguientes.

El almacenamiento de los *hashes* de los archivos y los datos de las presentaciones en la red *blockchain* proporciona a la aplicación *web* un soporte confiable contra alteraciones intencionadas o accidentales en la base de datos. Este soporte asegura la integridad y seguridad de la información, añadiendo una capa de protección adicional frente a posibles manipulaciones no autorizadas. Dicha configuración no solo mejora la integridad de los datos en la base de datos y en la *blockchain*, sino que también ofrece un respaldo confiable a los cuentadantes y personal del Tribunal de Cuentas de la Provincia de Misiones.

6.3. Observaciones reconocidas

Como se menciona anteriormente, la aplicación *validation* permite recuperar la descripción y el *hash* de los documentos obtenidos de la presentación. Sin embargo, no realiza una comparación directa de estos *hashes* con los documentos almacenados en la base de datos correspondiente a la misma presentación. Para llevar a cabo una verificación de *hashes*, archivo por archivo, especialmente en caso de detectarse diferencias, es necesario recurrir a una aplicación externa especializada en esta tarea.

El desarrollo del prototipo demuestra que la generación de los *hashes* de documentos digitales no es el único método de validación. Y aunque es una práctica común según los estándares y arquitecturas investigadas, es importante destacar que las pruebas del prototipo se llevaron a cabo en una *blockchain* de ensayo, lo que implica que su eficacia en otras *blockchains* no está garantizada. Además, para que el contrato inteligente opere correctamente, es esencial que la *blockchain* de destino sea compatible con Solidity, el lenguaje de programación utilizado. Esto subraya la

necesidad de considerar las especificaciones técnicas de la *blockchain* al implementar soluciones basadas en contratos inteligentes.

Los archivos utilizados para la demostración han sido facilitados por personal del Tribunal de Cuentas, lo que añade un nivel significativo de realismo y relevancia a las pruebas realizadas. Al emplear datos y documentos simulados en el entorno de prueba, se asegura de que las simulaciones sean lo más cercanas posible a escenarios reales, permitiendo así una evaluación más precisa de la funcionalidad de la propuesta presentada en este trabajo. Esta aproximación garantiza que los resultados obtenidos no solo reflejan el desempeño en condiciones ideales, sino que también proporcionan una visión clara de cómo el sistema se comportaría en el ámbito práctico del organismo.

Capítulo 7

Conclusiones

En este capítulo, se presentan de manera concisa las conclusiones derivadas del trabajo, así como las perspectivas futuras que se abren a partir de los resultados obtenidos.

Durante el desarrollo de esta investigación, se aborda el ámbito de la aplicación de la tecnología *blockchain* en el sector público. En primer lugar, se lleva a cabo una revisión teórica y un relevamiento de trabajos relacionados en el ámbito de *blockchain* y la integridad de datos. Este proceso revela diversas prácticas para respaldar el contenido de documentos digitales, proporcionando así una sólida base teórica que orienta y enriquece la investigación.

Posteriormente, se presentan antecedentes tanto a nivel local como internacional, demostrando una clara tendencia hacia la adopción de la tecnología *blockchain* para el resguardo y la validación de documentos digitales y otros activos.

Con el objetivo de abordar el problema planteado, se diseña una arquitectura de *software* basada en *blockchain*. A partir de esta arquitectura, se desarrolla un prototipo de aplicación *web* que permite a la institución publicar el *hash* de los archivos de las presentaciones digitales en la *blockchain*. Este enfoque permite a los usuarios interesados verificar la validez de dichos archivos, demostrando la inmutabilidad de los datos y detectando cualquier modificación en los documentos. El trabajo detalla con precisión las herramientas y tecnologías utilizadas en el desarrollo de este proyecto.

Este proyecto alcanza con éxito sus objetivos propuestos, ofreciendo no solo una solución al problema planteado, sino también abriendo nuevas perspectivas para futuras investigaciones en el ámbito de la tecnología *blockchain*. Aun así, se considera que existen futuras líneas de investigaciones, tales como:

1. Implementación en entornos de producción: Investigar la efectividad y aplicabilidad de la arquitectura propuesta en entornos reales de producción mediante la recopilación, organización y análisis de información adicional.

2. Seguridad y privacidad en *blockchain*: Explorar en mayor profundidad los aspectos relacionados con la seguridad y privacidad de la información almacenada en la cadena de bloques, así como posibles mejoras o enfoques innovadores para abordar estos temas.
3. Automatización de alertas para detección de modificaciones no autorizadas: Investigar y desarrollar un sistema automatizado de alertas que utilice contratos inteligentes y tecnologías *blockchain* para detectar modificaciones no autorizadas en documentos digitales almacenados. Explorar la integración de mecanismos de monitoreo que emitan alertas en tiempo real ante cualquier cambio en la integridad de los archivos, mejorando la seguridad y proporcionando una respuesta proactiva frente a posibles amenazas.
4. Integración de *blockchain* con diferentes *stacks* tecnológicos: Investigar métodos efectivos de integración de *blockchain*s, tanto públicas como privadas, con sistemas y aplicaciones existentes en instituciones públicas y privadas, evaluando casos de estudio y mejores prácticas para una implementación exitosa.

BIBLIOGRAFÍA

[1] Tribunal de Cuentas Provincia de Misiones, "Resolución IV - N° 3 (Antes Resolución T.C.298/86)". 2020. Consultado: el 17 de junio de 2023. [En línea]. Disponible en: https://digesto.htcmisiones.gob.ar/digesto20/src/web/viewer.html?file=https://digesto.htcmisiones.gob.ar/digesto20/src/static/Procedimiento%20Jurisdiccional/Archivos%20PDF/IV%20-%20N%203.pdf&buscar=&search=

[2] "Cómo la transparencia a través de blockchain ayuda a la comunidad de ciberseguridad - IBM Blog". Consultado: el 17 de junio de 2023. [En línea]. Disponible en: https://www.ibm.com/blog/how-transparency-through-blockchain-helps-the-cybersecurity-community/

[3] I. Bashir, *Mastering blockchain: distributed ledger technology, decentralization, and smart contracts explained*, Second edition, Fully revised and Updated. en Expert insight. Birmingham Mumbai: Packt, 2018.

[4] Z. Zheng, S. Xie, H. Dai, X. Chen, y H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends", en *2017 IEEE International Congress on Big Data (BigData Congress)*, Honolulu, HI, USA: IEEE, jun. 2017, pp. 557–564. doi: 10.1109/BigDataCongress.2017.85.

[5] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", 2008. Consultado: el 7 de junio de 2023. [En línea]. Disponible en: https://bitcoin.org/en/bitcoin-paper

[6] L. Bass, P. Clements, y R. Kazman, *Software architecture in practice*, 3rd ed. en SEI series in software engineering. Upper Saddle River, NJ: Addison-Wesley, 2013.

[7] B. Singhal, G. Dhameja, y P. S. Panda, *Beginning Blockchain*. Berkeley, CA: Apress, 2018. doi: 10.1007/978-1-4842-3444-0.

[8] "Blockchain - Blockchain Federal Argentina". Consultado: el 29 de junio de 2023. [En línea]. Disponible en: https://bfa.ar/blockchain/blockchain

[9] C. Mg. Brys, *La Cadena de Bloques, Blockchain. Un abordaje comprensible a su definición y posibles usos*, 1°. 2019. Consultado: el 7 de junio de 2023. [En línea]. Disponible en: https://www.researchgate.net/publication/347471812_La_Cadena_de_Bloques_Blockchain_Un_abordaje_comprensible_a_su_definicion_y_posibles_usos

[10] "Protocolos de consenso - Blockchain Federal Argentina". Consultado: el 9 de junio de 2023. [En línea]. Disponible en: https://bfa.ar/blockchain/protocolos-de-consenso

[11] "Hybrid Blockchain - GeeksforGeeks". Consultado: el 29 de agosto de 2023. [En línea]. Disponible en: https://www.geeksforgeeks.org/hybrid-blockchain/

[12] H. Desai, M. Kantarcioglu, y L. Kagal, "A Hybrid Blockchain Architecture for Privacy-Enabled and Accountable Auctions", en *2019 IEEE International Conference on Blockchain (Blockchain)*, Atlanta, GA, USA: IEEE, jul. 2019, pp. 34–43. doi: 10.1109/Blockchain.2019.00014.

[13] "Introducción a las funciones Hash | KeepCoding Bootcamps". Consultado: el 9 de junio de 2023. [En línea]. Disponible en: https://keepcoding.io/blog-frr/introduccion-a-las-funciones-hash/
[14] "¿Qué es Prueba de participación / Proof of Stake (PoS)?" Consultado: el 9 de junio de 2023. [En línea]. Disponible en: https://academy.bit2me.com/que-es-proof-of-stake-pos/
[15] "Proof of Authority Explained | Binance Academy". Consultado: el 9 de junio de 2023. [En línea]. Disponible en: https://academy.binance.com/en/articles/proof-of-authority-explained
[16] D. Drescher, *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. Berkeley, CA: Apress, 2017. doi: 10.1007/978-1-4842-2604-9.
[17] "¿Qué es Ethereum? | ethereum.org". Consultado: el 11 de junio de 2023. [En línea]. Disponible en: https://ethereum.org/es/what-is-ethereum/
[18] "Introducción a ether | ethereum.org". Consultado: el 21 de noviembre de 2023. [En línea]. Disponible en: https://ethereum.org/es/developers/docs/intro-to-ether/
[19] "Introducción a las DApps | ethereum.org". Consultado: el 21 de noviembre de 2023. [En línea]. Disponible en: https://ethereum.org/es/developers/docs/dapps/
[20] "Smart Contracts - Blockchain Federal Argentina". Consultado: el 28 de junio de 2023. [En línea]. Disponible en: https://bfa.ar/blockchain/smart-contracts
[21] "Contratos inteligentes | ethereum.org". Consultado: el 28 de junio de 2023. [En línea]. Disponible en: https://ethereum.org/es/smart-contracts/
[22] "Máquina virtual de Ethereum (EVM) | ethereum.org". Consultado: el 21 de noviembre de 2023. [En línea]. Disponible en: https://ethereum.org/es/developers/docs/evm/
[23] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications", en *Proceedings First International Conference on Peer-to-Peer Computing*, Linkoping, Sweden: IEEE Comput. Soc, 2002, pp. 101–102. doi: 10.1109/P2P.2001.990434.
[24] Tribunal de Cuentas Provincia de Misiones, "Inducción general para ingresantes al Tribunal de Cuentas de Misiones". 2022.
[25] "HTML: Lenguaje de etiquetas de hipertexto | MDN". Consultado: el 30 de noviembre de 2023. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTML
[26] "CSS | MDN". Consultado: el 16 de diciembre de 2023. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/CSS
[27] "¿Qué es una API? - Explicación de interfaz de programación de aplicaciones - AWS". Consultado: el 23 de noviembre de 2023. [En línea]. Disponible en: https://aws.amazon.com/es/what-is/api/
[28] "DOM - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web | MDN". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Glossary/DOM

[29] "¿Qué es SQL? - Explicación de lenguaje de consulta estructurado (SQL) - AWS". Consultado: el 30 de noviembre de 2023. [En línea]. Disponible en: https://aws.amazon.com/es/what-is/sql/
[30] "¿Qué es cloud computing? | Google Cloud". Consultado: el 23 de noviembre de 2023. [En línea]. Disponible en: https://cloud.google.com/learn/what-is-cloud-computing?hl=es
[31] R. Kalis, "Using blockchain to validate audit trail data in private business applications", jun. 2018, [En línea]. Disponible en: https://kalis.me/uploads/bsc-thesis.pdf
[32] Igor Barinov, Victor Lysenko, Serguei Beloussov, Mark Shmulevich, y Stanislav Protasov, "System and method for verifying data integrity using a blockchain network". Enero de 2020. [En línea]. Disponible en: https://patentimages.storage.googleapis.com/88/b4/cc/631631b28e99d7/US20180025181A1.pdf
[33] A. Pinheiro, E. D. Canedo, R. T. De Sousa, y R. De Oliveira Albuquerque, "Monitoring File Integrity Using Blockchain and Smart Contracts", *IEEE Access*, vol. 8, pp. 198548–198579, 2020, doi: 10.1109/ACCESS.2020.3035271.
[34] "Qué es BFA - Blockchain Federal Argentina". Consultado: el 8 de noviembre de 2023. [En línea]. Disponible en: https://bfa.ar/bfa/que-es-bfa
[35] "Tecnología - Blockchain Federal Argentina". Consultado: el 17 de noviembre de 2023. [En línea]. Disponible en: https://bfa.ar/bfa/tecnologia
[36] "Cómo funciona - Blockchain Federal Argentina". Consultado: el 17 de noviembre de 2023. [En línea]. Disponible en: https://bfa.ar/bfa/como-funciona
[37] "Licitaciones - Blockchain Federal Argentina". Consultado: el 8 de noviembre de 2023. [En línea]. Disponible en: https://bfa.ar/blockchain/casos-de-uso/licitaciones
[38] "Bit2Me Staking". Consultado: el 18 de diciembre de 2023. [En línea]. Disponible en: https://bit2me.com/es/suite/earn/polygon
[39] "What is Polygon (MATIC)? | Coinbase". Consultado: el 18 de diciembre de 2023. [En línea]. Disponible en: https://www.coinbase.com/es-LA/learn/crypto-basics/what-is-polygon
[40] "Polygon Knowledge Layer". Consultado: el 18 de diciembre de 2023. [En línea]. Disponible en: https://docs.polygon.technology/
[41] "Polygon impulsa el portal de denuncias de la policía india y lucha contra la corrupción". Consultado: el 18 de diciembre de 2023. [En línea]. Disponible en: https://es.cointelegraph.com/news/polygon-powers-india-police-complaint-portal-battling-corruption
[42] "Queja de Firozabad sitio web". Consultado: el 18 de diciembre de 2023. [En línea]. Disponible en: https://www.policecomplaintonblockchain.in/
[43] "Introduction | BNB Chain Documentation". Consultado: el 26 de septiembre de 2023. [En línea]. Disponible en: https://docs.bnbchain.org/docs/learn/intro
[44] Hyperledger, "Hyperledger Fabric whitepaper". Consultado: el 26 de septiembre de 2023. [En línea]. Disponible en:

https://8112310.fs1.hubspotusercontent-na1.net/hubfs/8112310/hyperledger_fabric_whitepaper.pdf
[45] "3 razones por las que el dominio de mercado de Ethereum va en aumento". Consultado: el 22 de noviembre de 2023. [En línea]. Disponible en: https://es.cointelegraph.com/news/3-reasons-why-ethereum-market-cap-dominance-is-rising
[46] "¿Qué es una aplicación web? - Explicación de las aplicaciones web - AWS". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://aws.amazon.com/es/what-is/web-application/
[47] "Introducción al lado servidor - Aprende desarrollo web | MDN". Consultado: el 30 de noviembre de 2023. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Learn/Server-side/First_steps/Introduction
[48] "Front End frente a back-end: diferencia entre el desarrollo de aplicaciones - AWS". Consultado: el 30 de noviembre de 2023. [En línea]. Disponible en: https://aws.amazon.com/es/compare/the-difference-between-frontend-and-backend/
[49] "The Architecture of a Web 3.0 application". Consultado: el 24 de noviembre de 2023. [En línea]. Disponible en: https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application
[50] "Solidity — Solidity 0.8.20 documentation". Consultado: el 11 de junio de 2023. [En línea]. Disponible en: https://docs.soliditylang.org/en/v0.8.20/
[51] "Welcome to Python.org". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://www.python.org/
[52] "Structure of a Contract — Solidity 0.8.23 documentation". Consultado: el 3 de diciembre de 2023. [En línea]. Disponible en: https://docs.soliditylang.org/en/v0.8.23/structure-of-a-contract.html
[53] "JavaScript | MDN". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript
[54] "Python Packaging User Guide". Consultado: el 29 de noviembre de 2023. [En línea]. Disponible en: https://packaging.python.org/en/latest/
[55] "npm Docs". Consultado: el 29 de noviembre de 2023. [En línea]. Disponible en: https://docs.npmjs.com/
[56] "npm | Home". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://www.npmjs.com/
[57] "pip · PyPI". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://pypi.org/project/pip/
[58] "Node.js". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://nodejs.org/en
[59] "PyPI · El Índice de paquetes de Python". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://pypi.org/
[60] "The web framework for perfectionists with deadlines | Django". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://www.djangoproject.com/

[61] "Cross Site Scripting (XSS) | OWASP Foundation". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://owasp.org/www-community/attacks/xss/
[62] "Cross Site Request Forgery (CSRF) | OWASP Foundation". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://owasp.org/www-community/attacks/csrf
[63] "Home - Truffle Suite". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://trufflesuite.com/
[64] "Truffle | Overview - Truffle Suite". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://trufflesuite.com/docs/truffle/
[65] "¿Qué es Truffle? | KeepCoding Bootcamps". Consultado: el 30 de noviembre de 2023. [En línea]. Disponible en: https://keepcoding.io/blog/que-es-truffle/
[66] "Ganache - Truffle Suite". Consultado: el 9 de diciembre de 2023. [En línea]. Disponible en: https://trufflesuite.com/ganache/
[67] "Ganache | Overview - Truffle Suite". Consultado: el 30 de noviembre de 2023. [En línea]. Disponible en: https://trufflesuite.com/docs/ganache/
[68] "gm — web3.py 6.11.4 documentation". Consultado: el 30 de noviembre de 2023. [En línea]. Disponible en: https://web3py.readthedocs.io/en/stable/
[69] "Web3.js — Javascript Ethereum API". Consultado: el 20 de octubre de 2023. [En línea]. Disponible en: https://web3js.org/
[70] "jQuery". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://jquery.com/
[71] "AJAX - Aprende desarrollo web | MDN". Consultado: el 6 de diciembre de 2023. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data
[72] Abraham Silberschatz, Peter Baer Galvin, y Greg Gagne, *Fundamentos de Sistemas Operativos*, 7a. ed. Madrid: McGraw-Hill, 2005.
[73] "Desktop Operating System Market Share Worldwide | Statcounter Global Stats". Consultado: el 7 de diciembre de 2023. [En línea]. Disponible en: https://gs.statcounter.com/os-market-share/desktop/worldwide/
[74] "¿Qué es Linux?" Consultado: el 7 de diciembre de 2023. [En línea]. Disponible en: https://www.redhat.com/es/topics/linux
[75] "DistroWatch.com: Put the fun back into computing. Use Linux, BSD." Consultado: el 7 de diciembre de 2023. [En línea]. Disponible en: https://distrowatch.com/
[76] "Manjaro". Consultado: el 7 de diciembre de 2023. [En línea]. Disponible en: https://manjaro.org/
[77] "pacman - ArchWiki". Consultado: el 7 de diciembre de 2023. [En línea]. Disponible en: https://wiki.archlinux.org/title/pacman
[78] "Visual Studio Code - Code Editing. Redefined". Consultado: el 7 de diciembre de 2023. [En línea]. Disponible en: https://code.visualstudio.com/
[79] "Git". Consultado: el 7 de diciembre de 2023. [En línea]. Disponible en: https://git-scm.com/

[80] "Contract ABI Specification — Solidity 0.8.25 documentation". Consultado: el 29 de enero de 2024. [En línea]. Disponible en: https://docs.soliditylang.org/en/latest/abi-spec.html
[81] "Writing your first Django app, part 1 | Django documentation | Django". Consultado: el 14 de diciembre de 2023. [En línea]. Disponible en: https://docs.djangoproject.com/en/5.0/intro/tutorial01/
[82] "Easily rename Django project - Stack Overflow". Consultado: el 15 de diciembre de 2023. [En línea]. Disponible en: https://stackoverflow.com/questions/18293875/easily-rename-django-project
[83] "How to manage static files (e.g. images, JavaScript, CSS) | Django documentation | Django". Consultado: el 16 de diciembre de 2023. [En línea]. Disponible en: https://docs.djangoproject.com/en/4.2/howto/static-files/
[84] "SQL Client and Database Management Software - DbVisualizer". Consultado: el 8 de enero de 2024. [En línea]. Disponible en: https://www.dbvis.com/

ANEXOS

Anexo I.

En este anexo se presentan enlaces a sitios relevantes para el lector. Los enlaces han sido agregados en la fecha 29/05/2024.

- Repositorio completo del código fuente: https://github.com/facurodrij/TrabajoFinal_LSI
- Código fuente del contrato inteligente: https://github.com/facurodrij/TrabajoFinal_LSI/blob/main/truffle_setup/contracts/PresentationStorage.sol

Anexo II.

En este anexo se presenta el código fuente del contrato inteligente desarrollado con el lenguaje de programación Solidity.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract PresentationStorage {
  address public owner;

  constructor() {
    owner = msg.sender;
    addPresentation(0, 0, 0, "", new string[](0), new
string[](0));
  }


  modifier onlyOwner() {
    require(msg.sender == owner, "Only owner can call
this function.");
    _;
  }


  function changeOwner(address _newOwner) public
onlyOwner {
    require(_newOwner != owner, "New owner must be
different than current owner.");
```

```
    owner = _newOwner;
  }

  struct Presentation {
    uint256 presentationDatetime;
    uint256 presentationNumber;
    uint256 year;
    uint256 period;
    string municipio;
    string[] fileDescriptions;
    string[] fileHashes;
  }

  uint256 public presentationsCount = 0;
  Presentation[] public presentationsList;

  event PresentationAdded(uint256 indexed
presentationCount);

  function addPresentation(
    uint256 _presentationNumber,
    uint256 _year,
    uint256 _period,
    string memory _municipio,
    string[] memory _descriptions,
    string[] memory _hashIds
  ) public onlyOwner {
```

```
    require(_hashIds.length == _descriptions.length,
"Hashes and document descriptions must have the same
length.");


    presentationsList.push(Presentation({
      presentationDatetime: block.timestamp,
      presentationNumber: _presentationNumber,
      year: _year,
      period: _period,
      municipio: _municipio,
      fileDescriptions: _descriptions,
      fileHashes: _hashIds
    }));

    emit PresentationAdded(presentationsCount);
    presentationsCount++;
  }

  function getPresentationByCount(uint256
_presentationCount) public view returns (
    uint256,
    uint256,
    uint256,
    uint256,
    string memory,
    string[] memory,
    string[] memory
  ) {
```

```
    Presentation memory presentation =
presentationsList[_presentationCount];

    return (
      presentation.presentationDatetime,
      presentation.presentationNumber,
      presentation.year,
      presentation.period,
      presentation.municipio,
      presentation.fileDescriptions,
      presentation.fileHashes
    );
  }
}
```

Anexo III.

Las figuras que se ilustran a continuación son capturas de pantalla de la interfaz de usuario de Ganache, donde se detalla las opciones de configuración seleccionadas de la red mencionada en la subsección 5.4.2. Configuración de Truffle y Ganache.

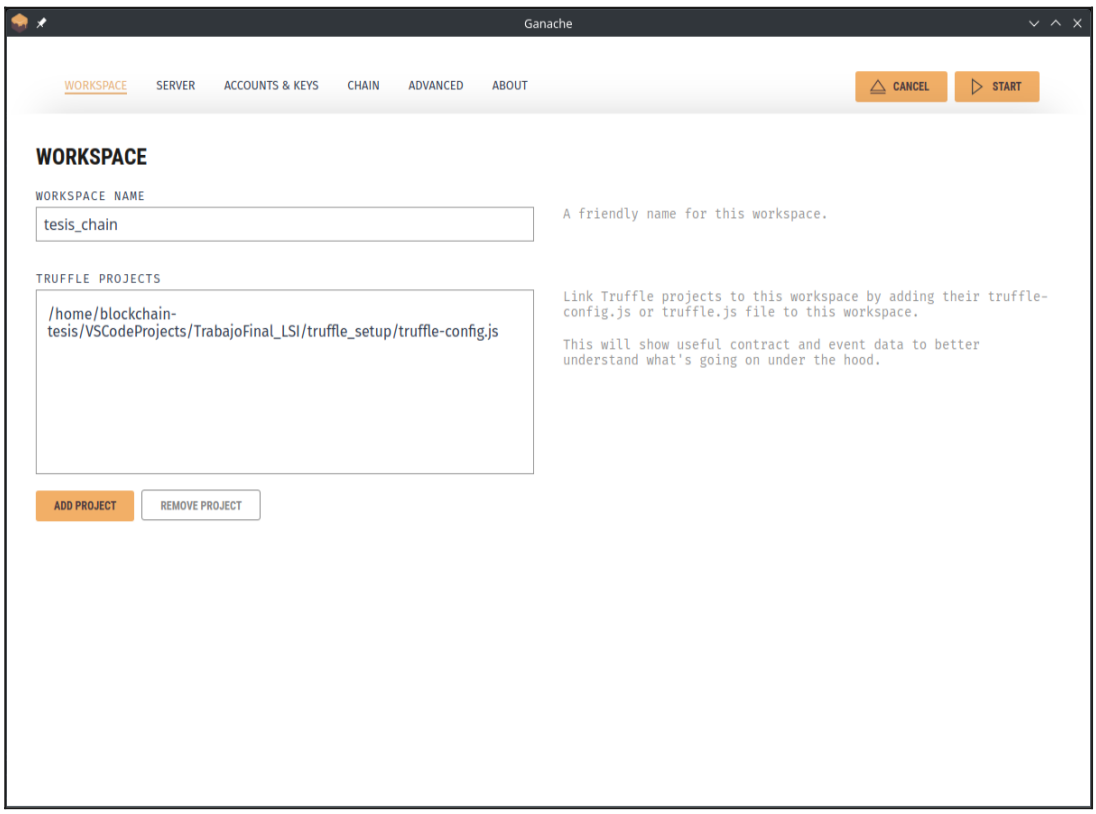

Figura 37: Configuración del espacio de trabajo (en inglés, Workspace). Fuente: Elaboración propia.

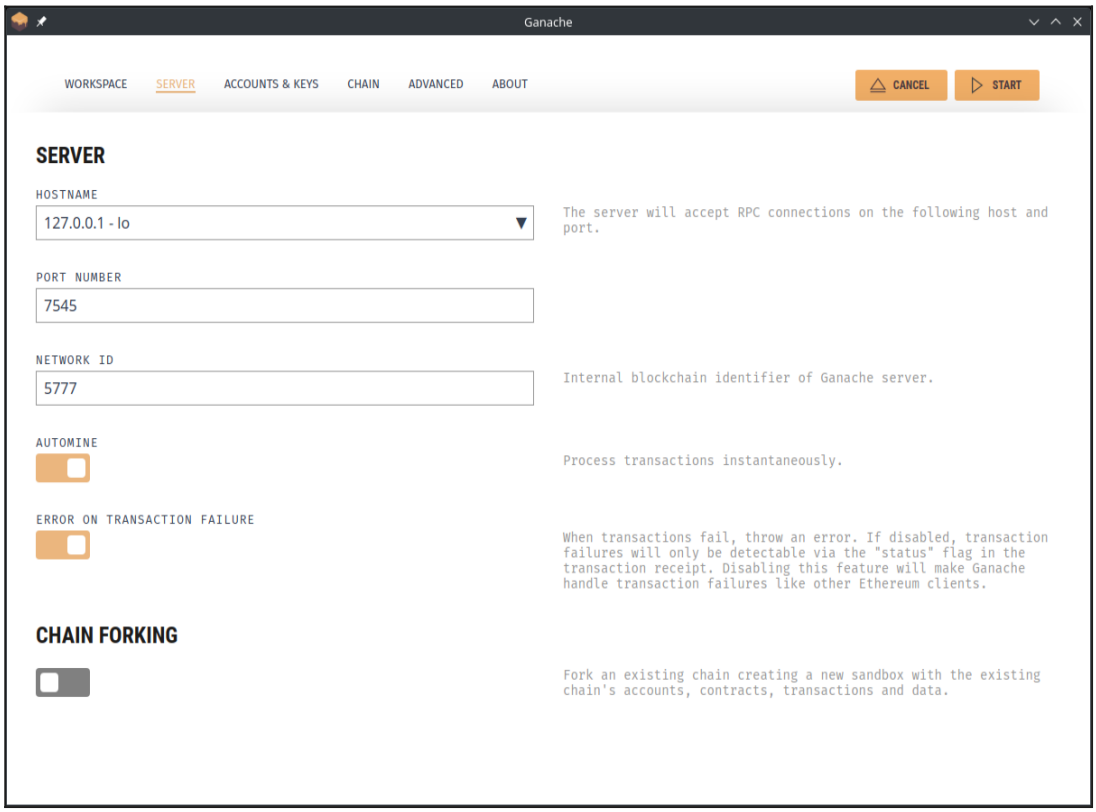

Figura 38: Configuración del servidor (en inglés, Server). Fuente: Elaboración propia.

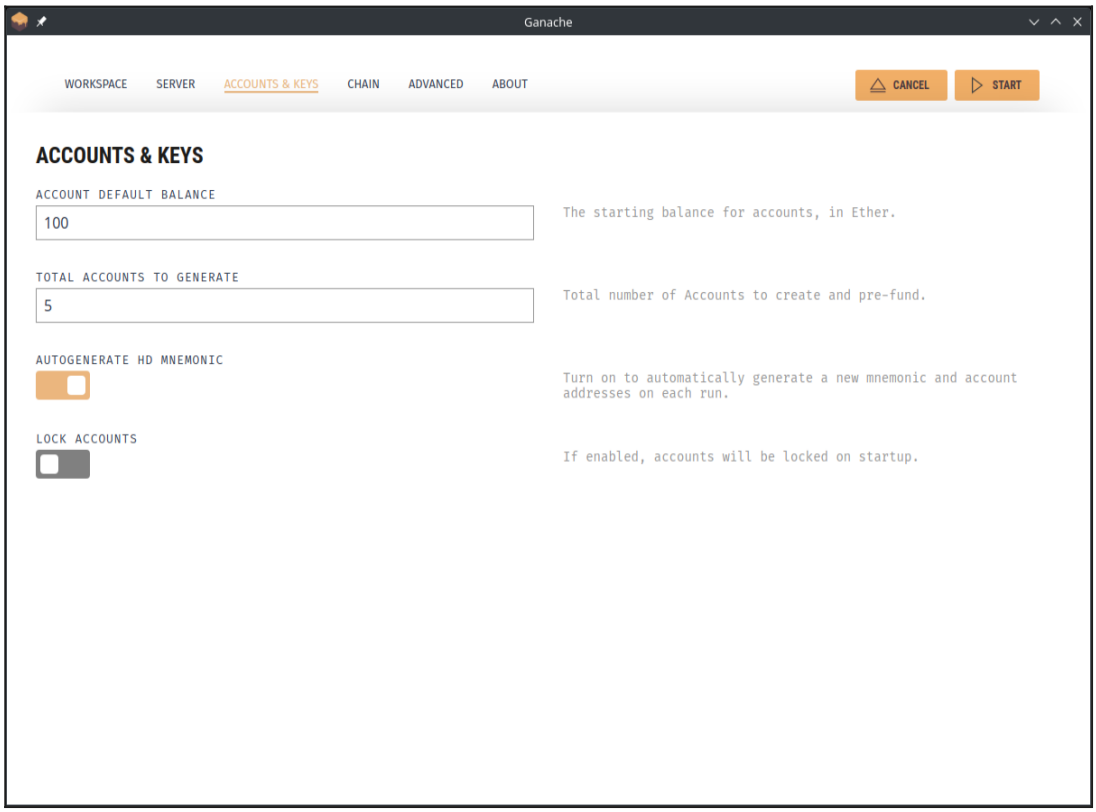

Figura 39: Configuración de cuentas y claves (en inglés, Accounts and Keys). Fuente: Elaboración propia.

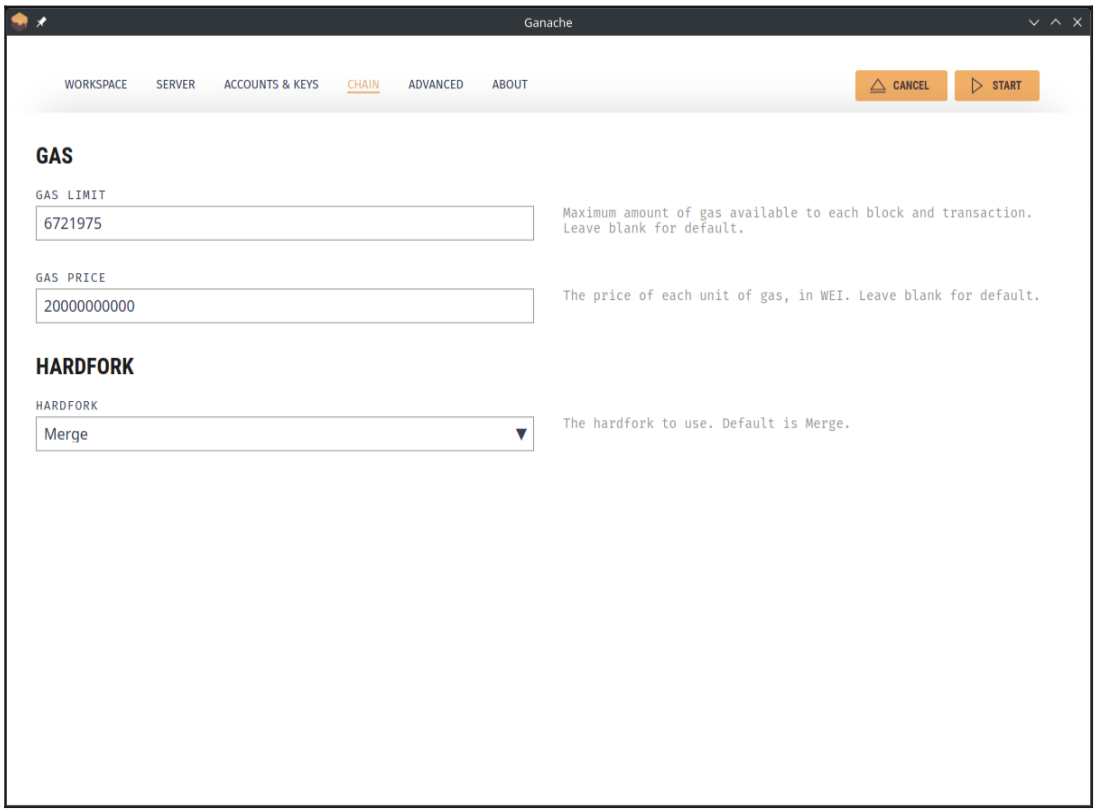

Figura 40: Configuración de la cadena (en inglés, Chain). Fuente: Elaboración propia.

Anexo IV.

Las figuras que se presentan a continuación son capturas de pantalla de la interfaz de usuario del prototipo de aplicación *web* creado con Django.

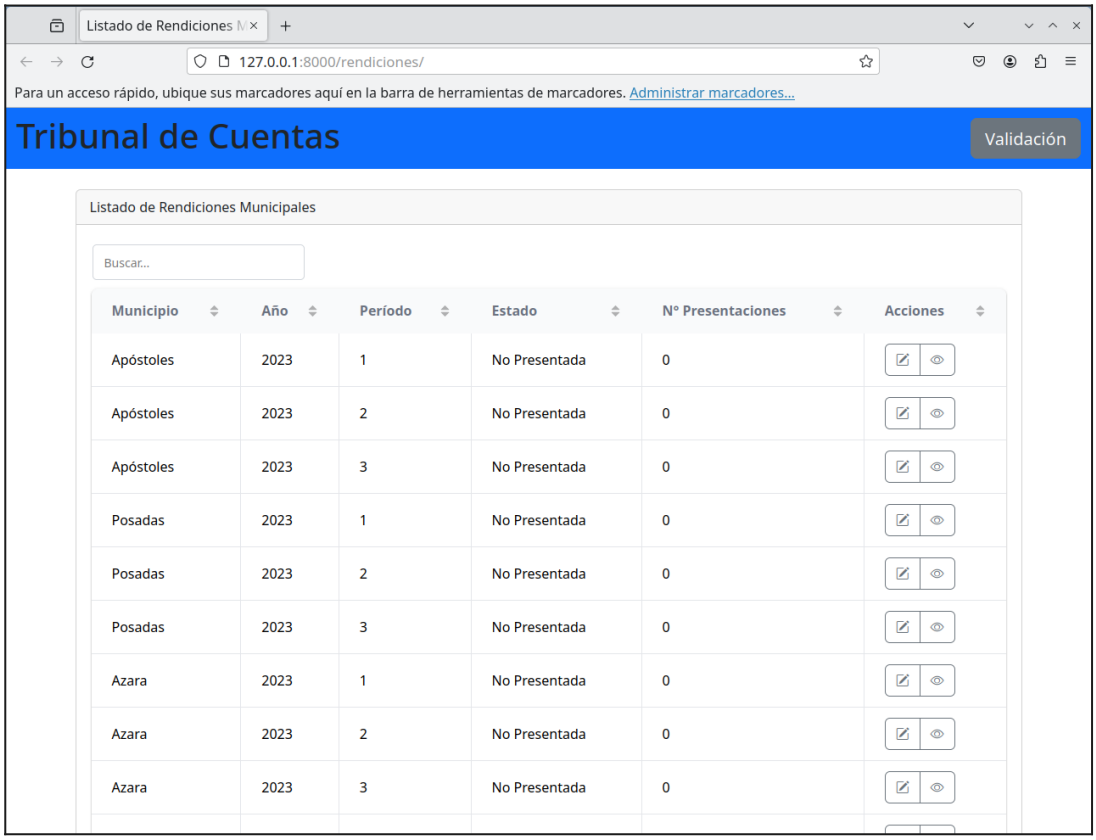

Figura 41: Listado de rendiciones municipales. Fuente: Elaboración propia.

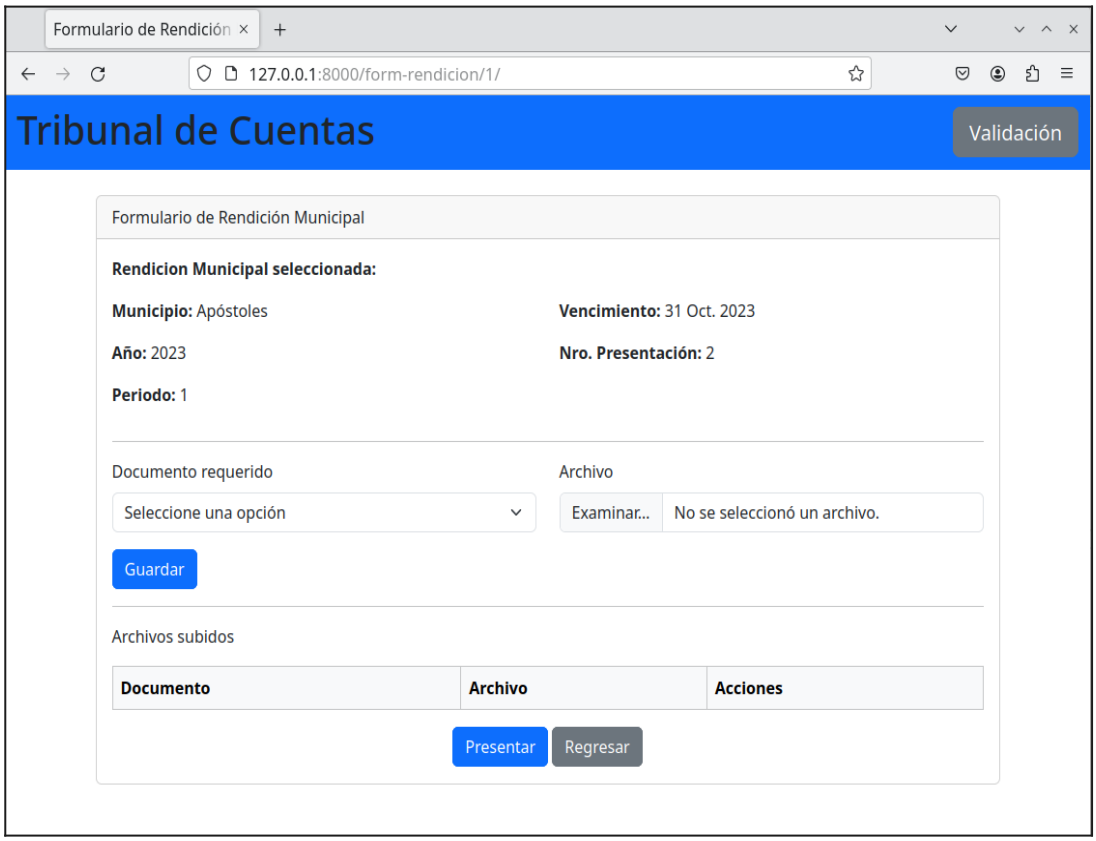

Figura 42: Formulario de rendición municipal. Fuente: Elaboración propia.

Figura 43: Detalle de una rendición municipal. Fuente: Elaboración propia.

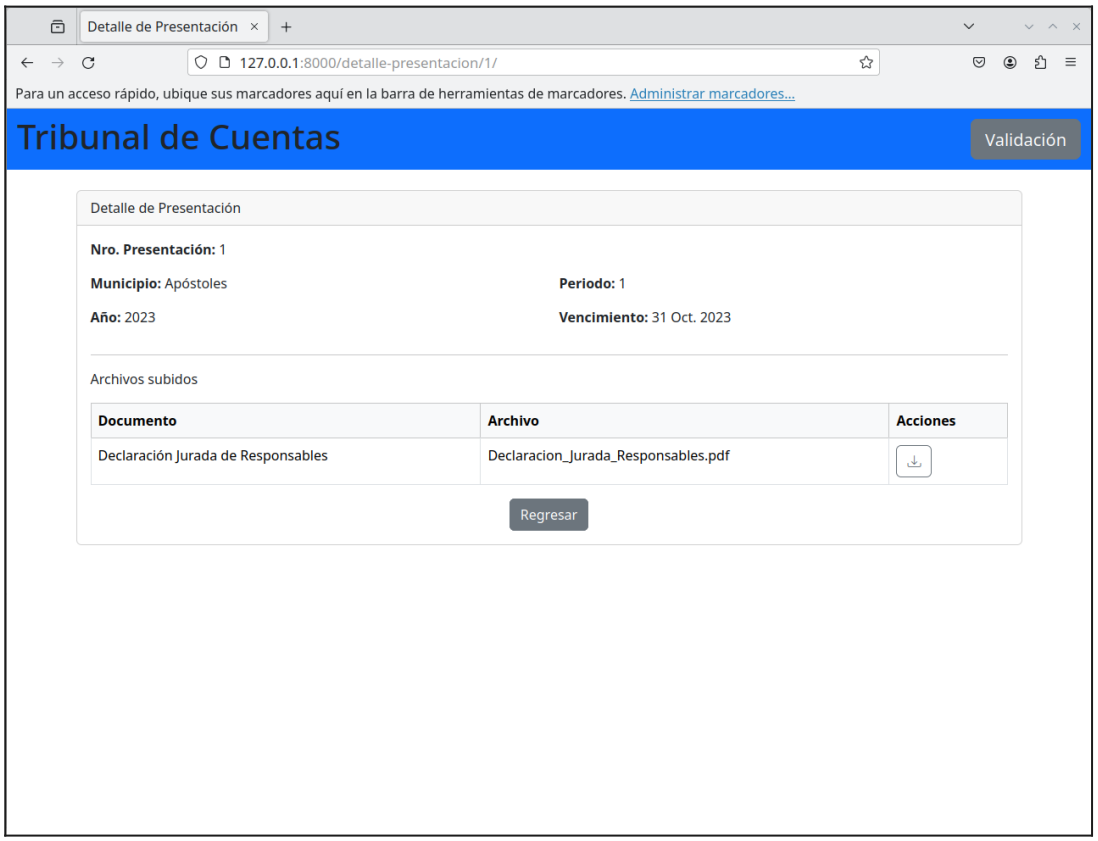

Figura 44: Detalle de una presentación realizada. Fuente: Elaboración propia.

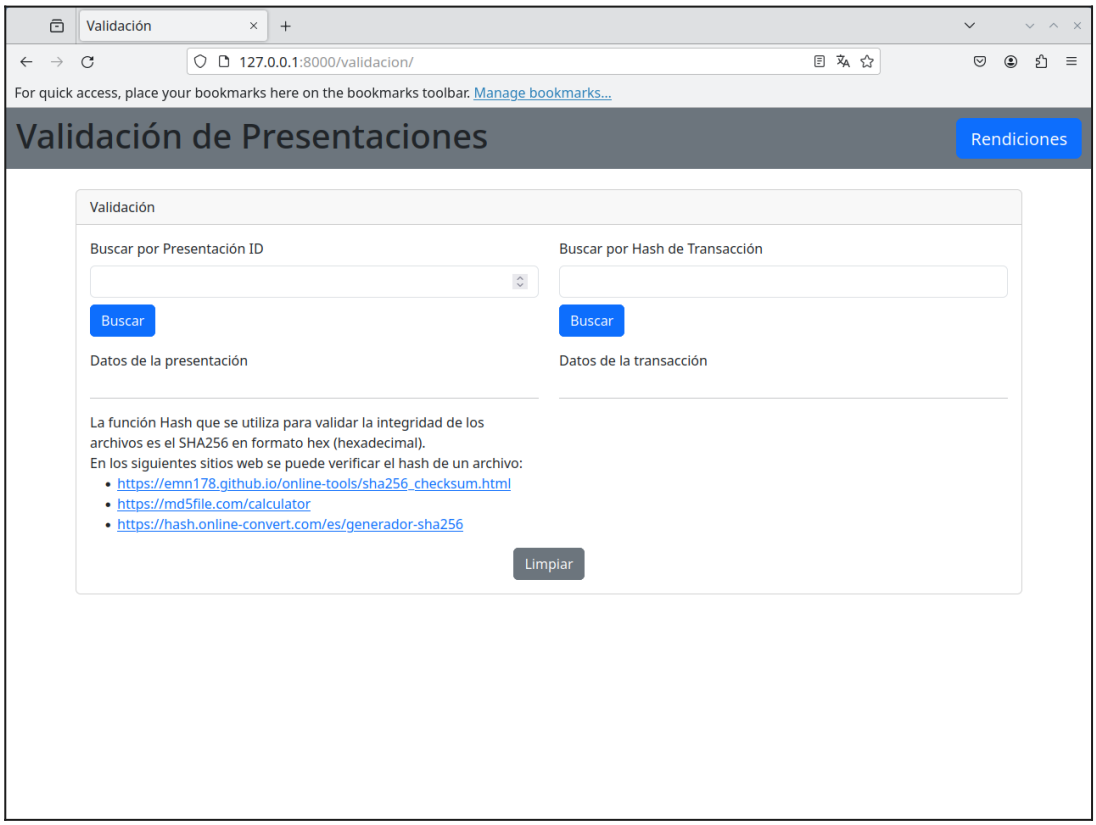

Figura 45: Formulario de validación de presentaciones. Fuente: Elaboración propia.