PROYECTO DE GRADO

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito final para obtener el Título de INGENIERO DE SISTEMAS

APLICACIÓN PARA EL REGISTRO DE HISTORIAS MÉDICAS EN UNA BLOCKCHAIN

Por

Br. MARIA DANIELA VIELMA RANGEL

Tutor: Prof. Jacinto Dávila

Junio 2023

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Aplicación para el registro de historias médicas en una blockchain.

Br. Maria Daniela Vielma Rangel

Proyecto de grado – Investigación de Operaciones

Resumen: En este proyecto se desarrolla una aplicación de registros de historias médicas en una Blockchain. Con la utilización de la cadena de bloques los datos y archivos se guardan, usando la criptografía como parte de la protección de la integridad de la blockchain. La aplicación desarrollada proporciona la interfaz gráfica básica para el registro y almacenamiento de este tipo de archivos, probando la efectividad de la tecnología utilizada.

Palabras Claves: historias médicas, aplicaciones descentralizadas, blockchain.

Índice

Índice de Figuras

Capítulo 1: Introducción.

1.1 Contextualización del problema

Con el auge de la tecnología informática se ha dado origen a la globalización, haciendo necesario la actualización de muchos métodos dentro de las instituciones, en el caso de la medicina y muy en especial en el registro de las historias médicas se han establecido nuevos estándares para almacenar y procesar la información tanto dentro como fuera que mejore el funcionamiento, agilizando el diagnóstico de alguna enfermedad o patología lo que hace necesario la introducción de nuevas tecnologías que puedan ser utilizadas directamente por el involucrado, en este caso, paciente médico, permitiendo el acceso a su historial médico sin la necesidad de depender de algún organismo o dependencia central.

Siendo así, "las y los pacientes médicos, en casi cualquier condición de salud y mayormente en las capas de mayor necesidad económica de las sociedades, raramente tienen acceso a un registro de su historia médica que pueda ser usado oportunamente para diagnosticar su estado real y monitorear sus tratamientos" (Dávila,2021).

A lo largo del tiempo en el área médica se ha manejado el archivo de las historias médicas manualmente, lo que puede ocasionar que se extravié o ser manipulada por personas ajenas, además de volverse contenido difícil de entender para el público en general. En muchas ocasiones el paciente no tiene acceso a la información de su historial médico.

Con miras a una solución favorable, tanto para el equipo médico como para el paciente, se pretende desarrollar una aplicación para el registro de historias médicas en una blockchain que permitirá acceder a la información, actualizarla oportunamente y mostrarla a cualquier médico que deba tratarle, brindando la seguridad necesaria a los datos almacenados.

1.2 Antecedentes

La presente investigación tiene como base histórica el estudio de distintos proyectos, enmarcados en el mismo fin, lograr que los pacientes y personal médico lleven un registro exhaustivo de su historial. Basado en ello, Dávila (2021) en su artículo de "Historias médicas de las y los pacientes" propone estudiar y desarrollar medios informáticos modernos para

ofrecer el servicio de almacenamiento y gestión de sus datos médicos a cada paciente, completamente bajo su control.

Ahora bien, la necesidad de involucrar a los pacientes en particular, fue evaluada en un proyecto de tesis asociado con este proyecto matriz. Hidalgo (2017), realiza un "estudio de factibilidad para una red de diagnóstico molecular", donde propone crear una red de laboratorios de bioanálisis, consultorios médicos y centros de investigación que se asistan mutuamente para apoyar procesos de diagnóstico que aprovechen los recursos de la biología molecular para identificar, diagnosticar y tratar condiciones médicas y trastornos de salud de personas residentes en el área metropolitana de la ciudad de Mérida.

Haciendo énfasis en la tecnología utilizada para desarrollar este proyecto (Blockchain) se encontraron estudios previos, como es el caso del proyecto denominado "Aplicación Android con integración Blockchain en respuesta al acoso en espacios públicos" diseñado por Patricia Barrios (2018), el cual permite enviar alertas de emergencia entre dispositivos, así como el almacenamiento y consulta de las mismas y la publicación de quejas ante anuncios de acoso. Particularmente los datos generados y compartidos quedarán almacenados en Ethereum, basada en el modelo Blockchain, con el fin de mantener el pseudo anonimato de los usuarios, tener una versión consistente de los datos en todo momento y asegurarlos mediante el uso de criptografía.

Así mismo, en el año 2022, Laura Renieblas Diaz-Regañon, implementó un sistema basado en Blockchain que lleva por nombre: "Diseño y desarrollo de una aplicación para la compraventa de viviendas a través de la tecnología Blockchain", que tiene como objetivo agilizar y reducir los procesos de compra y venta de viviendas entre particulares, funcionando como una inmobiliaria. El sistema permite la compra-venta de viviendas, de forma que no haya posibilidades de cambiar los datos a posteriori sin alterar su estructura o información existente.

Por su parte, Ulises Mateo Ortega (2021) en su tesis "Aplicación de un histórico de vehículos sobre Blockchain", desarrolló un sistema de software libre para dar a conocer e incentivar el uso de tecnologías descentralizadas mediante el diseño de una aplicación que almacene información histórica de vehículos en las plataformas Ethereum e Interplanetary file System. Es un proyecto que sugiere futuras actualizaciones para optimizar su seguridad.

En un aspecto importante relacionado a la tecnología blockchain y el marco legal en que se basan este tipo de estudios se encuentra una tesis denominada "Modelado de una organización para la gestión cooperativa de historias médicas" Ovalle (2018) plantea crear una

red de servicios médicos tradicionales que se conforma para incorporar, en un esfuerzo conjunto, nuevas tecnologías de diagnóstico molecular y de la teleinformación incluyendo también la definición del marco legal que amparará la operación de la organización. Uno de los principales aportes de esta investigación es el Modelo Legal de RENDIAMO, para ello analizó un conjunto de relaciones entre leyes y normativas para representar el esquema de historias médicas compartidas sin menoscabar los derechos y obligaciones de cada uno de los participantes dentro de la red.

En este proyecto de investigación también se detalla una lista de sistemas utilizados en este campo. El desarrollo de herramientas de software para la gestión de recursos médicos, específicamente para el registro de historias médicas ha ido evolucionando con el tiempo, pero aún no existe una herramienta o herramientas, para que, las o los pacientes médicos tengan acceso a un registro de su historia médica que pueda ser usado oportunamente sin necesidad de depender de alguna institución o administración central. Las tecnologías que se citan a continuación referencian cuáles son los elementos mínimos necesarios en un registro médico y en sus servicios de acceso:

- **Care2x:** "es un sistema de información hospitalario basada en la web. Sirve para integrar diversos sistemas de información (dentro del hospital) en un solo sistema de información. Care2x es un proyecto de código abierto basado en la licencia GPL" (Care2x, 2020).
- **GNU Health:** GNU Health "es un Sistema de software libre de Gestión Hospitalaria y de información de la Salud con las siguientes funcionalidades: Historia clínica electrónica, Sistemas de información hospitalaria, Sistemas de información de Salud. Diseñado para ser multiplataforma, para que se pueda instalar en diferentes sistemas operativos (GNU/Linux, FreeBSD, MS Windows), Sistema de gestión de bases de datos (PostgreSQL), y Planificación de recursos empresariales (Tryton)" (colaboradores de Wikipedia, 2023c).
- **OpenHIS:** es el emprendimiento del Grupo BioLinux para cubrir la necesidad de un sistema informático para hospitales. OpenHIS "es un desarrollo creado totalmente desde cero, con nuevo diseño, y con el respaldo que el grupo BioLinux puede ofrecer en su larga trayectoria análisis, diseño y desarrollo de sistemas informáticos para el ámbito de la salud" (OpenHIS - Sistema Informático Hospitalario).

- **Alas HIS**: Gestión clínica y administrativa de hospitales y centros de salud a través de una sola plataforma, con apoyo de tecnología de vanguardia, desarrollada para optimizar los procesos que permiten el funcionamiento de organizaciones dedicadas a tratar pacientes en cualquier rama de la medicina. Esta plataforma ofrece:
  - Garantizar la portabilidad de la información clínica generada alrededor de la asistencia médica a los pacientes.
  - La posibilidad de llevar una Historia Clínica Electrónica (HCE) única por paciente.
  - Posibilidad de realizar consultas de segunda opinión entre médicos de diferentes instituciones de salud.
  - Almacenamiento de las imágenes médicas y los informes imagenológicos asociados.
  - Acceso de información estadística en tiempo real a todos los niveles.

Haciendo una revisión de otras tecnologías aplicadas para este campo, se encontraron las siguientes:

- **Solve.care: "**Es una empresa de TI para el cuidado de la salud que crea plataformas blockchain. La plataforma Solve.Care utiliza la tecnología blockchain como el libro mayor distribuido subyacente para coordinar la atención, los beneficios y los pagos entre todas las partes de la cadena de atención médica: pacientes, médicos, farmacias, laboratorios, empleadores, aseguradoras y otros" (*BLOCKCHAIN PLATFORM FOR DIGITAL HEALTH NETWORKS*, 2022).
- **Medicalchain: "**es una plataforma que utiliza tecnología blockchain para almacenar registros de salud. Las diferentes organizaciones, como médicos, hospitales, laboratorios, farmacéuticos y aseguradoras de salud pueden solicitar permiso para acceder al registro del paciente para cumplir su propósito y registrar las transacciones en el libro mayor distribuido" (Medicalchain., 2022).
- **doc.ai (Sharecare Company): "**doc.ai es una plataforma de inteligencia artificial empresarial que unifica todo lo relacionado a datos de salud. Crea productos para una cartera de clientes, incluidos pagadores, farmacéuticos y proveedores" (About Our Digital Health and Wellness Company - Sharecare, 2022).
- **Hashed Health**: "es una empresa de innovación de libros de contabilidad distribuidos en el sector de la salud. Se asocian con empresas de atención médica para crear software y redes que resuelvan los desafíos de confianza, transparencia y alineación. Ofrece a

los pagadores y proveedores soluciones técnicas reales de productos blockchain enfocados en disminuir el costo de la atención y reducir las insuficiencias administrativas" (Hashed Health, 2023).

- **Medical Record System (Sistema de historia clínica)**: "Sistema medico electrónico basado en roles escrito en Solidity. Un protocolo de control de registros médicos centrado en el paciente basado en blockchain. Este protocolo admite la delegación de permisos y los procesos de ruptura del cristal. Este sistema demuestra que la tecnología blockchain podría fortalecer la privacidad al brindar al paciente una forma transparente de controlar sus datos" (Medical Record System., 2017).

1.3 Objetivos de este proyecto

1.3.1 Objetivo general

Desarrollar una aplicación móvil para registros de historias médicas con datos encriptados y almacenados en una blockchain.

1.3.2 Objetivos específicos.

1.- Estudiar sistemas de historias médicas e identificar los elementos mínimos necesarios para crear un registro útil para un paciente.

2.- Desarrollar una aplicación de código abierto que pueda ser utilizada en algunos sistemas operativos para dispositivos móviles y que soporte las funcionalidades mínimas necesarias del sistema de registro de historias médicas.

3.- Conectar la aplicación con una base de datos funcional en una Blockchain.

4.- Documentación del desarrollo, implementación y beneficios de la aplicación de código abierto (software libre).

5.- Desarrollo de una posible evaluación de campo, con usuarios que son pacientes médicos.

Capítulo 2: Marco Teórico.

2.1 Historia Médica

Las historias médicas son un conjunto de datos recopilados por médicos sobre un paciente, en el que se tiene toda la información relacionada a sus antecedentes de salud, diagnósticos y tratamientos; es decir, todo su historial médico, así como también sus datos personales e información de médicos tratantes.

Las historias médicas han ido evolucionando a lo largo de la historia, como lo señala la Revista OCRONOS (2019), donde se comienza a hacer referencia en la antigua Grecia con la escuela hipocrática donde se empezó a registrar el transcurso de la enfermedad, este registro tenía un procedimiento ordenado que constaba de: numeración ordinal del enfermo dentro del grupo, nombre del enfermo, localización social, breve referencia a datos sobre enfermedades anteriores y descripción día a día del curso de la enfermedad con una ordenación cronológica de los hechos muy precisa. Luego dando paso a la Edad Media se comenzó con la formación de médicos, preparados solo en el punto de vista teórico, sin tener contacto directo con el paciente, por lo que los médicos con experiencia comenzaron a escribir los llamados Consilium, que servían de guía para los médicos que acababan de finalizar sus estudios. Los Consilium constaban de tres partes: en la primera se describían los síntomas del paciente y se emitía un diagnóstico; en la segunda se daban sugerencias para un estilo de vida que ayudase a superar la enfermedad y en la última se describía la terapia aplicada y la evolución del paciente con ese tratamiento.

En el renacimiento durante el siglo XV las Facultades de Medicina acordaron para obtener la licenciatura, que el alumno recibiera formación práctica, en las cuales tenían un libro de apoyo denominado las Observatio: una versión más actuales que los Consilium, eran historias médicas con descripciones más específicas, objetivas y carentes de interpretaciones religiosas, una característica muy particular de los Consilium en la edad media, éstas finalizaban con un diagnóstico y con indicaciones terapéuticas, aunque no había contacto directo con el paciente. Ya al siglo XVII el médico inglés Sydenham llamado también como el Hipócrates inglés, se caracterizó por establecer el contacto directo con el paciente utilizando un método basado en la experiencia y en la observación. Se caracterizaba por describir y tomar notas con precisión con los síntomas y signos de la enfermedad, clasificándolos en dos variables; constantes:

aparecen siempre pero no son propios, y accidentales: añadidos por la naturaleza del enfermo, edad, sexo, otras enfermedades.

Llegando al siglo XVIII, el holandés Herman Boerhaave, presentó el método de instrucción clínica en la cabecera del enfermo como se conoce en la actualidad, el cual estableció que el examen del enfermo constaba de tres partes: inspección, interrogatorio anamnésico y exploración objetiva. La inspección permitía al médico conocer el sexo del paciente, su estado y sus hábitos, clase social y costumbres. El interrogatorio anamnésico consistía en obtener información acerca de los antecedentes, tanto familiares como personales para conocer las enfermedades anteriores, el comienzo de la enfermedad y el desarrollo de ésta hasta el momento en el que el paciente iba al médico. La exploración objetiva tenía como finalidad averiguar el estado morfológico y funcional de las distintas partes del organismo del paciente.

Después de estas nuevas metodologías introducidas en el siglo XVIII, a principios del siglo XIX hubo varios avances médicos e invenciones, entre estos la invención del estetoscopio y una máquina para medir la presión sanguínea. En la segunda mitad del siglo, Joseph Lister desarrolló métodos quirúrgicos antisépticos, usando ácido carbólico para limpiar las heridas y los instrumentos quirúrgicos. A finales de siglo, se descubren las vacunas contra el cólera, la rabia, el tétanos y la difteria. Todos estos descubrimientos, prácticas e inventos, sumados a nuevas técnicas de análisis y exploración permitieron recoger datos más precisos de los pacientes, enriqueciendo la historia médica. En este periodo surgió la enfermería como disciplina, por lo tanto, la historia medica paso de ser un instrumento de un solo médico a convertirse en un documento elaborado y consultado por varios médicos y por otras disciplinas.

Con el siglo XX, llegaron grandes avances tecnológicos, con la invención de aparatos para el diagnóstico y detección de enfermedades según los síntomas presentados en el paciente, los cuales han ayudado en la elaboración de historias médicas más detalladas, objetivas y precisas.

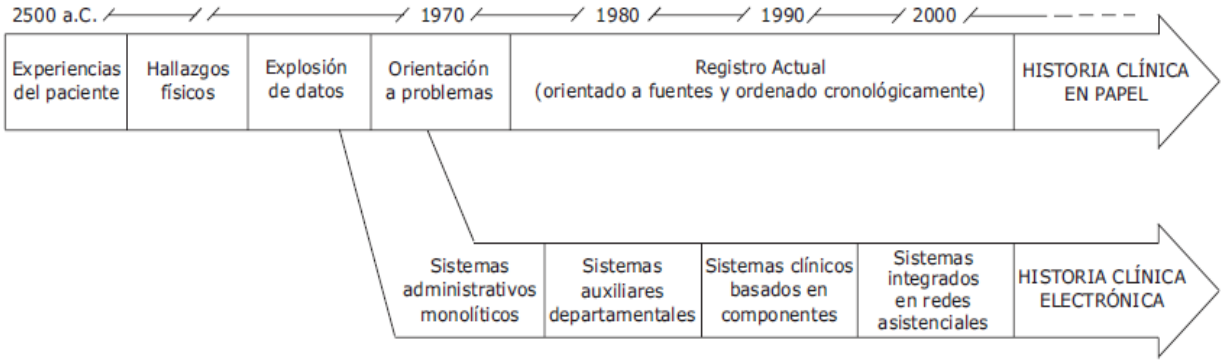

Figura 1. Evolución histórica del registro médico. Tomado de la Revista del Hospital Italiano de Buenos Aires, Articulo: Historia Clínica Electrónica.

La historia médica es una herramienta importante para los médicos y otros profesionales de la salud, debido a que permite tener información detallada sobre la salud del paciente y ser utilizada como un elemento de seguimiento para evaluar el progreso del tratamiento y la efectividad de las intervenciones médicas.

2.2 La Tecnología Blockchain

La tecnología Blockchain o también conocida como tecnología de registro distribuido se atribuye al año 2009 con la invención de la criptomoneda denominada Bitcoin a manos del autor Satoshi Nakamoto. Esta tecnología consiste en una base de datos distribuida que registra transacciones en bloques, los cuales están interconectados y asegurados mediante criptografía. Cada bloque contiene información verificable e inmutable de las transacciones realizadas, lo que garantiza la integridad y seguridad de los datos almacenados. La tecnología blockchain se utiliza en sistemas descentralizados para mantener un registro transparente y confiable de las transacciones sin necesidad de intermediarios o autoridades centrales.

Propone una solución al problema del costo de transacción en el comercio en internet que había llegado a depender casi exclusivamente de instituciones financieras como terceros de confianza para procesar pagos electrónicos. Satoshi Nakamoto muestra una solución al utilizar una red peer-to-peer, sin intermediarios.

Sistema de seguridad en una Blockchain

La Blockchain mediante la criptografía protege la información con la utilización de algoritmos codificados, hashes y firmas. Por lo tanto, mediante el uso de la función hash en la codificación de datos se crea una cadena de caracteres única sin importar la cantidad de datos

introducidos al principio de la función. Siendo así se crea una tecnología distribuida en el que se almacena la información de un nodo de la cadena en diferentes ubicaciones, lo que hace casi imposible el ataque al sistema.

Ahora bien, al ser un registro distribuido y consensuado, en donde todos los nodos de la cadena contienen la misma información para modificarla se tendría que alterar la cadena completa en al menos un 51% de los nodos. Siendo así y dado que cada bloque está matemáticamente vinculado al siguiente, al añadir uno nuevo a la cadena, el mismo se vuelve inalterable. Si un bloque modifica su relación con la cadena, se rompe, es decir que toda la información registrada en los bloques es inmutable y perpetua.

De esta forma la tecnología blockchain permite almacenar información que jamás se podrá perder, modificar o eliminar. Además, cada nodo de la red utiliza certificados y firmas digitales para verificar la información y validar las transacciones y los datos almacenados.

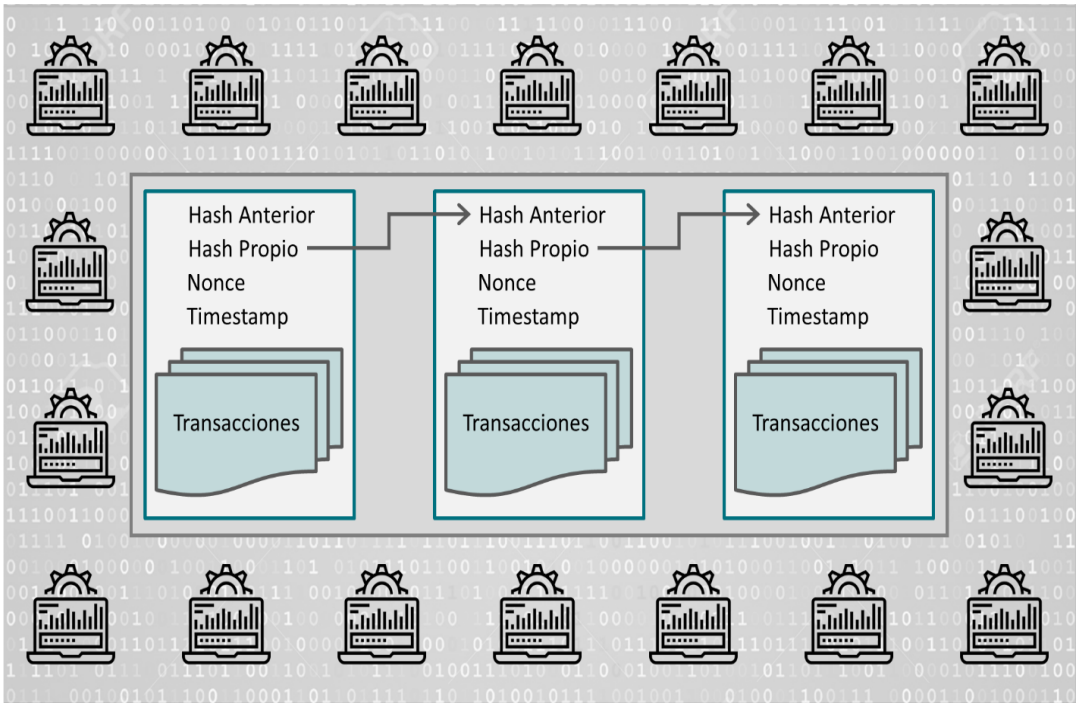

Figura 2. Esquema de la cadena de bloques. Tomado de welivesecurity.com

En el prefacio del libro Blockchain, Blueprint for a New Economy (O'Reilly Media, 1 edition, February 2015), Melanie Swan dice: "Debemos pensar en la cadena de bloques (blockchain) como algo similar a la Internet- tecnología de la información extendida con niveles organizados por capas y múltiples clases de aplicaciones para todo tipo de registro de bienes, inventario, intercambio, incluyendo cada área de las finanzas, la economía y el dinero;

bienes duros (propiedad física, casas, carros); y bienes intangibles (votos, ideas, reputación, intención, datos de salud, información, etc)" citado en (Revista Venezolana de Computación – ReVeCom., 2018) .

2.2.1 Ethereum.

Es una plataforma digital con la que es posible crear diversas aplicaciones descentralizadas. Fue lanzada en el año 2015, cuenta con su propia criptomoneda denominada ether y funciona como una plataforma de código abierto basada en la tecnología blockchain. Cada ordenador cuenta con una copia de la blockchain, por lo tanto, debe haber un acuerdo generalizado antes de aplicar cualquier cambio en la red. El blockchain de ethereum también funciona como registro del historial de transacciones. Con esta tecnología los desarrolladores pueden construir y desplegar aplicaciones descentralizadas o «dapps» con la red de Ethereum (CMC Markets.,2023). Para construir estas aplicaciones descentralizadas se debe crear un contrato inteligente y así desplegar y guardar toda la información requerida para el funcionamiento de la dapps.

2.2.2 Metamask

Es una wallet de criptomonedas que permite a los usuarios acceder a un ecosistema descentralizado de aplicaciones (dapps) (Phillips, 2022b). Se encuentra disponible como una aplicación móvil y como una extensión de navegador. La principal ventaja de usar Metamask es la facilidad que brinda para almacenar las claves privadas, permitiendo utilizar las funciones disponibles en la realización de firmas digitales de mensajes. Por otro lado, proporciona la forma más segura de conectarse a aplicaciones basadas en blockchain. Es una wallet de Ethereum utilizada para hacer transacciones funcionando de forma similar a una cuenta bancaria (Academy, 2023d).

La interacción de los usuarios con blockchains, como Ethereum, requiere un puente, y esto es lo que hace MetaMask, permite que los usuarios interactúen fácilmente con las aplicaciones descentralizadas de Ethereum. Por lo tanto, para los usuarios se convierte en una herramienta de fácil acceso al encontrarlas integradas en algunos navegadores como: Firefox, Chrome, Opera y Brave (Academy, 2023d).

2.2.3 Web3

Web3 o web3.js son un conjunto de bibliotecas de JavaScript que permiten interactuar a través de HTTP, IPC O WebSocket con nodos locales o remotos de Ethereum. Se pueden desarrollar aplicaciones cliente que interactúen con la cadena de bloques de Ethereum, leer y escribir datos de contratos inteligentes. A través del protocolo JSON RPC la biblioteca web3 expone varios métodos que se pueden usar para interactuar con los clientes de Ethereum. Este nodo, una Máquina Virtual Ethereum, pertenece a la red Ethereum (Simões, 2021c).

2.2.4 Ganache

Ganache es un simulador de blockchain para Ethereum en el que se puede desplegar contratos, ejecutar test, ejecutar comandos e inspeccionar el estado de la red. Con esta red es posible ejecutar distintas cuentas de usuario donde se obtienen hasta 100 ethers que pueden ser usados para realizar las pruebas necesarias de la aplicación a ser desarrollada. En la siguiente imagen se muestra la interfaz local de Ganache:

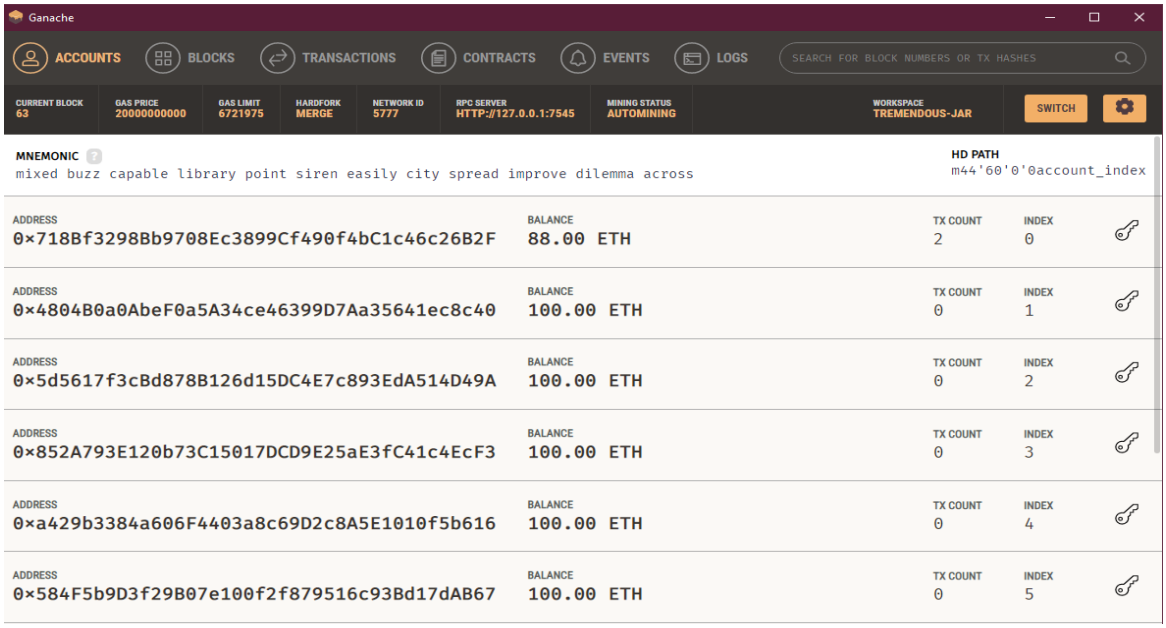

Figura 3. Interfaz Red de Pruebas Local Ganache.

2.2.5 Goerli

Goerli es una red de prueba de Ethereum que permite a los desarrolladores y usuarios experimentar con aplicaciones descentralizadas basadas en Ethereum. Utiliza un grupo

diferente de ether que la red principal de Ethereum, por lo que los desarrolladores pueden experimentar con dApps sin incurrir en costos significativos (Matthews, 2023).

2.3 Aplicaciones descentralizadas (DAPP)

Según Academy (2023c); DApp es la abreviatura de "Decentralized Applications" o "Aplicaciones descentralizadas". Se trata de una aplicación cuyo funcionamiento no depende de un punto de control o servidor central, sino que opera sobre la base de una red descentralizada. Sus usuarios tienen control total de la red sobre la que operan. Las DApps permiten a las personas acceder a diferentes servicios de forma segura. Estas aplicaciones se pueden usar en PC, teléfonos inteligentes o incluso acceder a través de la web.

El concepto de DApps no es nada nuevo. Las primeras DApps conocidas aparecieron en protocolos de intercambio de archivos como BitTorrent o DC++. Ambas aplicaciones son sistemas peert-to-peer altamente resistentes a la censura. Sin embargo, la primera DApp que utilizó la cadena de bloques fue el propio Bitcoin. Esto se debe a que su estructura y funcionamiento describen con éxito la primera DApp blockchain de la historia.

Sin embargo, no fue hasta la introducción de Ethereum, su lenguaje Solidity y la capacidad de ejecutar contratos inteligentes en 2014 que las DApps se dieron a conocer.

El canal de comunicación que utilizan las DApps es la cadena de bloques. En él, cada acción realizada a través de los contratos inteligentes que controlan la DApp deja un registro. La aceptación de las acciones realizadas por los usuarios de DApp depende de la programación de dicho contrato inteligente. De esta manera, busca asegurar que todos los participantes estén actuando dentro de su marco designado.

Las aplicaciones descentralizadas poseen tres estructuras básicas que intervienen en su interacción con el usuario y la forma en la que se manejan sus elementos:

Frontend:

La primera capa, es la interfaz que utilizan los usuarios para interactuar con la aplicación. El propósito de esta capa es simplemente permitir que los usuarios interactúen con la aplicación que están usando, recibiendo y enviando información.

Backend:

Se refiere a la lógica principal de la aplicación. En DApps, el backend está asociado con contratos inteligentes que se ejecutan en una cadena de bloques, como Ethereum. De esta forma, el contrato inteligente cuenta con un programa para asegurar el funcionamiento de la DApp. Esto garantiza un alto grado de transparencia y seguridad ya que los contratos inteligentes son visibles y públicos.

Almacenamiento de datos:

En las DApps, el almacenamiento de los datos es descentralizado y distribuido, cada usuario almacena un historial completo de las acciones realizadas en la red. Además, las interacciones se almacenan en la cadena de bloques. Todo ello de forma cifrada y segura, impidiendo el acceso a terceros no autorizados.

2.5 Servicios web

Según Natalia (2022b), los servicios web son un entorno estandarizado para distribuir la comunicación entre aplicaciones cliente y servidor en la Word Wide Web.

También puede definirse como un módulo de software diseñado para realizar un conjunto especifico de tareas. Los pilares de las aplicaciones se pueden encontrar en la web y también se pueden denominar en consecuencia, al ser llamados, el servicio web será capaz de proporcionar funcionalidad al cliente que solicita este portal, todo esto ocurre en un par de segundos.

Las condiciones de funcionamiento que permitirá entender el sistema serian:

- El cliente realizara una serie de llamadas a un servicio web haciendo peticiones al servidor que alojara el servicio web real.
- Estas solicitudes se ejecutan a través de las llamadas a procedimientos remotos. Estas llamadas a procedimientos remotos (RPC) son aquellas que se hacen en la solicitud correspondiente.

La capa de presentación o Frontend puede estar en cualquier lenguaje de programación que permita interactuar con el servicio web.

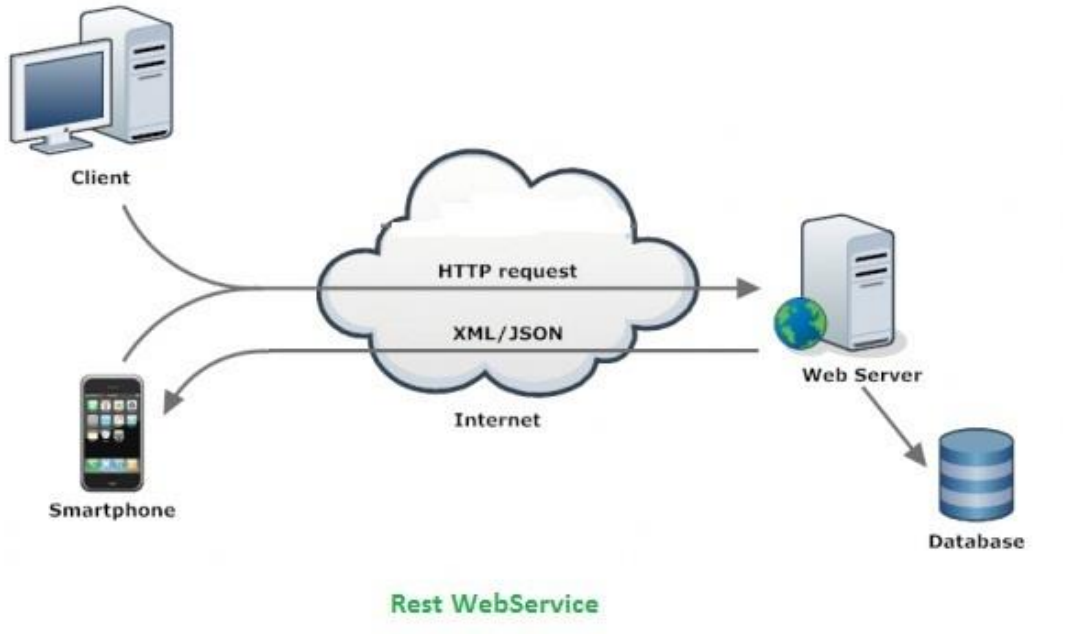

Figura 4. Esquema de funcionamiento de un servicio web. Tomado de servicios_web

2.5.1 ¿Qué son las API?

El término API es la abreviatura de Application Programming Interfaces, que significa interfaz de programación de aplicaciones en español. Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas (Fernández, 2019b).

Por lo tanto, podemos pensar en una API en una especificación formal que determina cómo un módulo de un software se comunica o interactúa con otro módulo para lograr una o más funciones. Todo depende de las aplicaciones que las vayan a usar, y de los permisos que les dé el propietario de la API a los desarrolladores de terceros.

2.6 Redes peer-to-peer (P2P).

Una red P2P, o peer-to-peer, es una red en la que participan un grupo de personas o máquinas de forma totalmente descentralizada. En otras palabras, es una red donde no hay un contacto o control central y donde las partes operan de manera independiente y se ajustan a un protocolo común de comunicación y consenso. De esta forma, los miembros de la red pueden intercambiar información directamente y sin intermediarios.

Para lograr esto, las redes P2P se basan en protocolos que funcionan sobre el Protocolo de Internet (también conocido como TCP/IP). Por lo tanto, los protocolos P2P se denominan protocolos de aplicación o Layer 7 según el modelo de Interconexión de Sistemas Abiertos u OSI. Esto significa que los protocolos P2P requieren el uso de otros protocolos más abstractos para funcionar, pero al mismo tiempo los hace más fáciles de crear y usar.

El funcionamiento de una red peer-to-peer es respectivamente sencillo. Fundamentalmente es construir un lenguaje de comunicaciones que permita a los clientes comunicarse de forma directa y sin intermediarios con otros computadores (Academy, 2023a).

2.7 JavaScript, CSS y HTML.

HTML: El Lenguaje de Marcado de Hipertexto (HTML) es el código que se utiliza para estructurar y desplegar una página web y sus contenidos. Provee etiquetas para describir los diferentes tipos de contenidos (elementos) de la web. Gracias a ello, el navegador podrá comprender el contenido enviado por el servidor y representarlo en pantalla.

CSS: (Cascading Style Sheets) es un lenguaje de hojas de estilo, es decir, permite aplicar estilos de manera selectiva a elementos en documentos HTML como fuentes y color a textos o cajas, se modifican tamaños, se añaden imágenes de fondo, márgenes o incluso se puede cambiar completamente la apariencia de un elemento HTML como una lista para convertirla en una barra o menú de navegación.

Javascript: es un lenguaje de programación del lado del cliente que permite implementar dinamismo y funcionalidad a la página web. Además, se puede mostrar actualizaciones de contenido, vincular eventos dinámicos a elementos HTML y almacenar datos en variables.

2.8 Node JS

Node.js se concibió como un entorno de ejecución de JavaScript basado en una arquitectura de procesos asincrónicos destinado a crear aplicaciones de red escalables. Es un entorno de tiempo de ejecución multiplataforma de código abierto que permite a los desarrolladores crear una variedad de herramientas y aplicaciones del lado del servidor (Acerca | Node.js, s. f.-b). La ejecución en tiempo real está destinada a ser utilizada fuera del contexto de un navegador web. El entorno omite las APIs de JavaScript específicas del navegador web y agrega soporte para las API de sistemas operativos más tradicionales, incluidas las bibliotecas

de sistemas de archivos y HTTP (Introducción a Express/Node - Aprende desarrollo web | MDN, 2023b).

2.8.1 Node Package Manager o npm

Es el administrador de paquetes de Node incluido que ayuda con cada desarrollo relacionado con Node.js. Es usado por los desarrolladores de JavaScript para compartir herramientas, instalar varios módulos y administrar sus dependencias (A & A, 2023b).

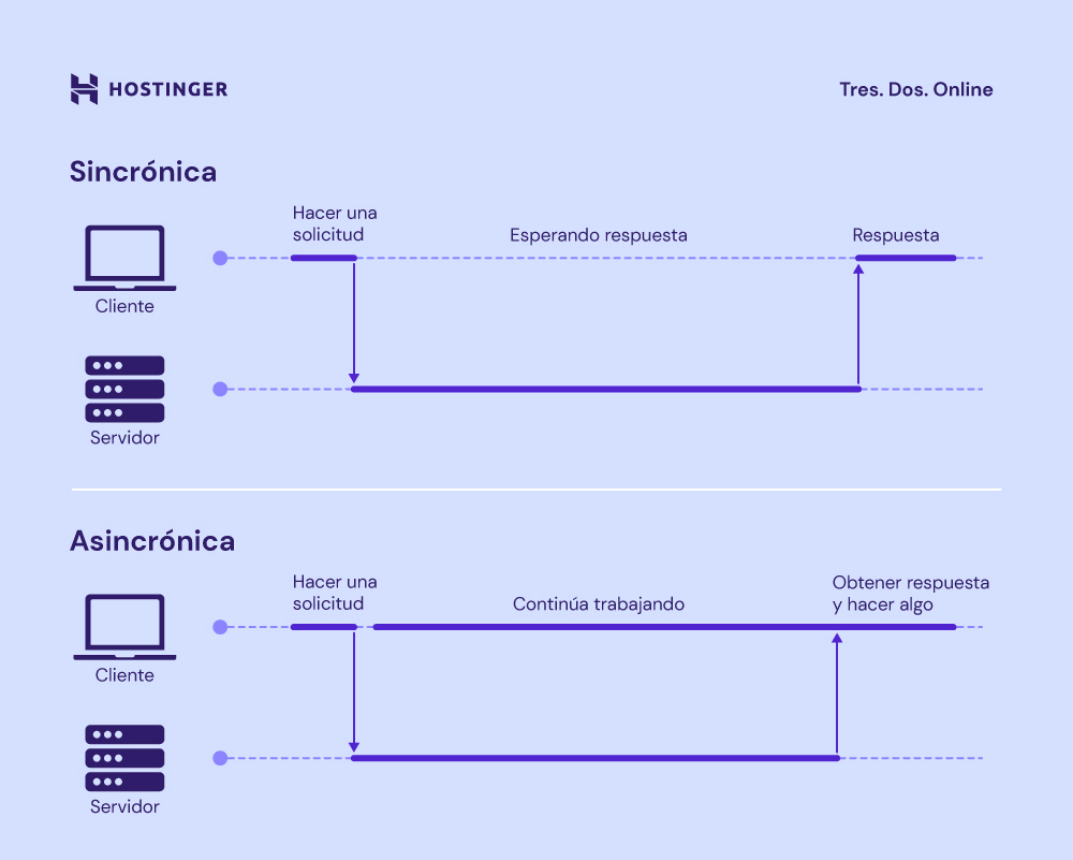

Figura 5. Arquitectura de procesos sincrónicos y asincrónicos. Tomado de (Infante & Infante, 2023b).

2.9 Contratos Inteligentes

Un contrato inteligente es un programa almacenado en la cadena de bloques que tiene la capacidad de ejecutar automáticamente expresiones lógicas o comandos contenidos en él cuando se cumplen ciertas condiciones. Se considera un acuerdo entre dos o más partes creado como fragmento de código informático diseñado específicamente para cumplir con cada indicación (Camerfirma, 2023).

Funcionamiento de un Contrato Inteligente

Los contratos en lenguaje de programación se crean, almacenan y ejecutan en una cadena de bloques, un espacio virtual que consiste en una red informática, donde las transacciones se realizan de forma automática e independiente.

Debido a esta red de ordenadores o nodos, la cadena de bloques gestiona la base de datos de forma descentralizada e inmutable. En otras palabras, sus datos almacenados no se pueden encontrar en una sola computadora o sistema. Por el contrario, todos los contratos se registran con códigos criptográficos en cada computadora que conforma la cadena de bloques. De esta forma, los contratos criptográficos se ejecutan de forma inmutable, siguiendo siempre los mensajes o sentencias lógicas de su código de programación almacenado en la cadena de bloques.

Si la red informática asegura el cumplimiento de una de las condiciones especificadas en el contrato, las medidas se activan. Una vez que se completa una actividad o transacción, la base de datos de blockchain se actualiza y no puede ser modificada o revocada por un tercero o terceros. Sólo las partes verán los resultados.

Capítulo 3: Implementación de la aplicación descentralizada.

Para el desarrollo de la aplicación se tomaron las referencias, el código fuente y los pasos realizados en la Tesis de Ulises Mateo Ortega que lleva por nombre: “Aplicación de un Histórico de vehículos sobre Blockchain” (febrero 2021); debido a que, al realizar las consultas referidas al desarrollo de aplicaciones con esa orientación, es la que mejor se adapta a los objetivos del proyecto, teniendo en consideración los cambios y mejoras propuestas.

3.1 Instalación de Metamask

Para el desarrollo de la aplicación es necesaria una *wallet* o billetera de Ethereum para desplegar el contrato inteligente, también para hacer las pruebas necesarias y gestionar la aplicación. En este caso, se utilizó Metamask para obtener una cuenta de Ethereum. El primer paso es instalar la extensión de la página principal (https://metamask.io/download/) al entrar al enlace, detectará automáticamente el navegador en el que se va a instalar la extensión (En este caso, Chrome).

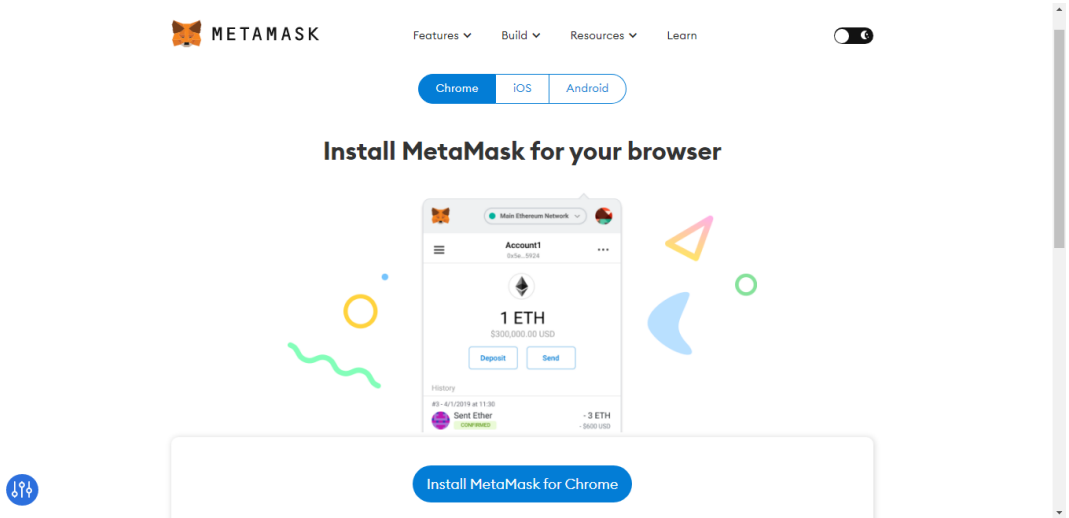

Figura 6. Entorno de instalación de Metamask.

Al hacer clic en el botón: Install Metamask for Chrome, abre otra ventana del navegador para instalar la extensión como se muestra en la figura 7.

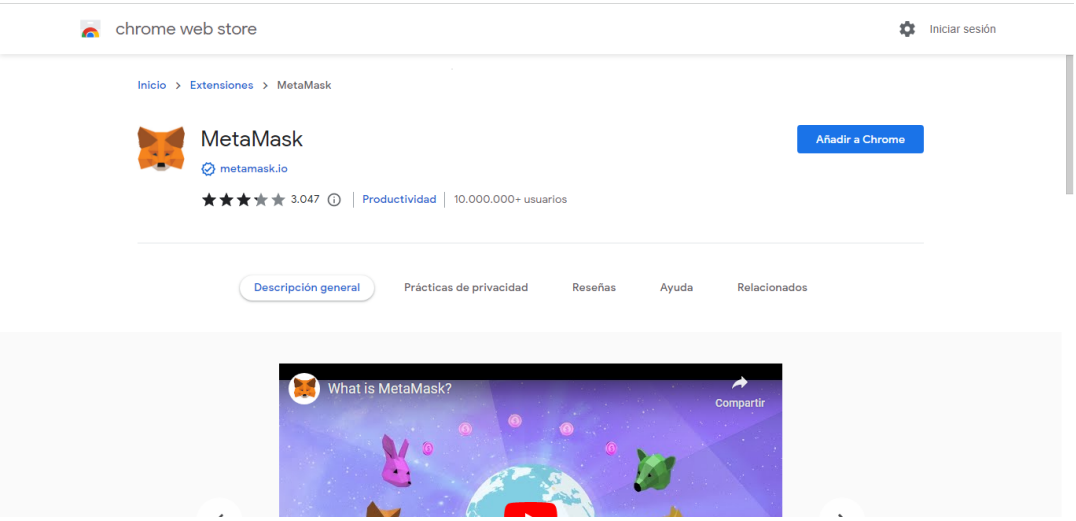

Figura 7. Instalación a la extensión de Metamask en Chrome.

Se pulsa en el botón de añadir extensión y saldrá una ventana emergente para confirmar la selección (Figura 8), se espera unos segundos y redireccionará a la pantalla de inicio de Metamask donde se puede seleccionar el idioma de preferencia, se aceptan los términos y condiciones, si aún no se tiene una cuenta, se pulsará en Crear una cartera nueva (Figura 9).

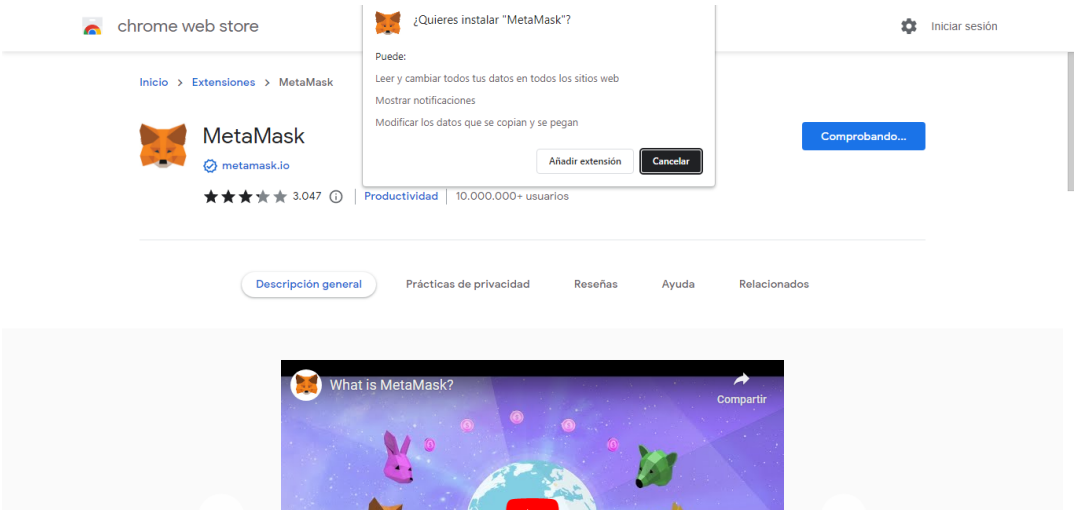

Figura 8. Ventana de confirmación de Metamask.

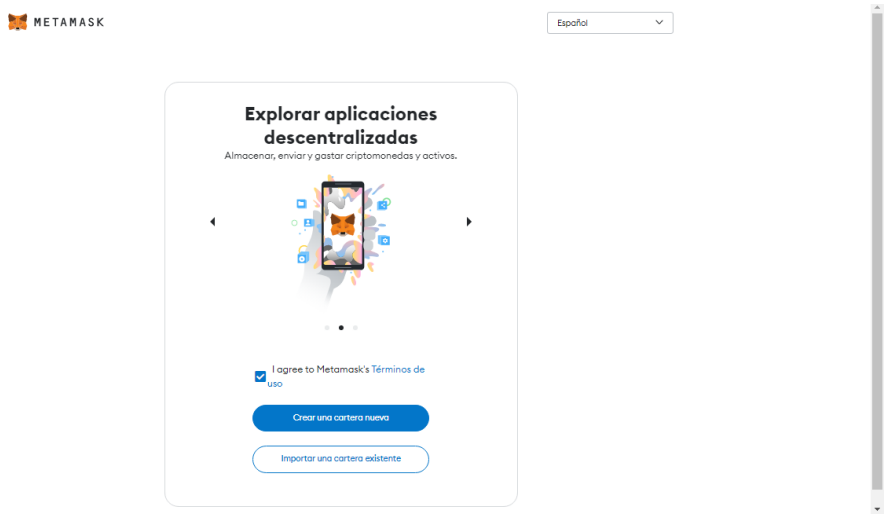

Figura 9. Pantalla de inicio de Metamask.

Al pulsar el botón de “crear nueva cartera” se aceptarán los términos (Figura 10) y pedirá la creación de una contraseña y respectivamente su confirmación (Figura 11), que será la que el usuario utilizará al ingresar a la cartera. En el segundo paso muestra una ventana de información con las indicaciones a seguir para proteger la cartera creada (Figura 12) se puede seleccionar cualquiera de las dos opciones para seguir con la configuración, aunque lo más recomendable es asegurar la cartera creada.

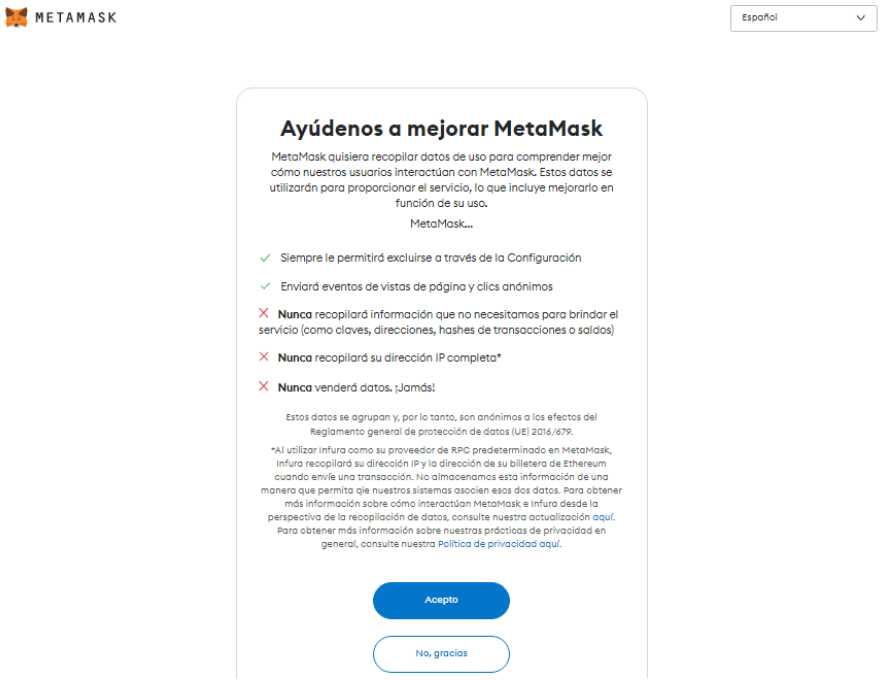

Figura 10. Pantalla de aceptar los términos de Metamask.

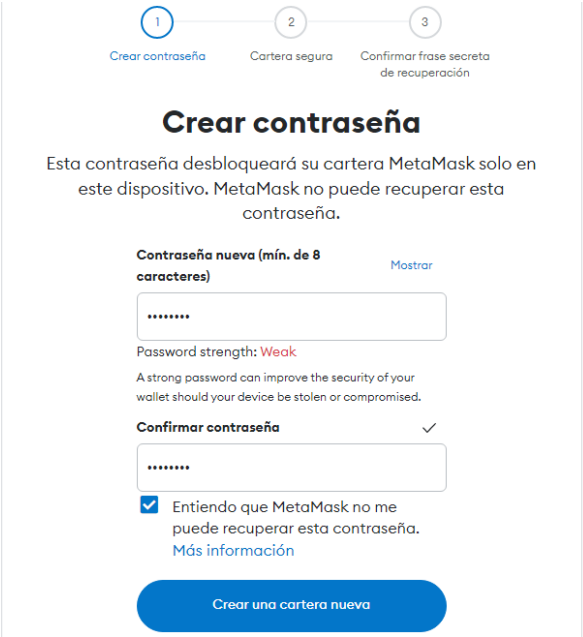

Figura 11. Pantalla de creación de contraseña de Metamask.

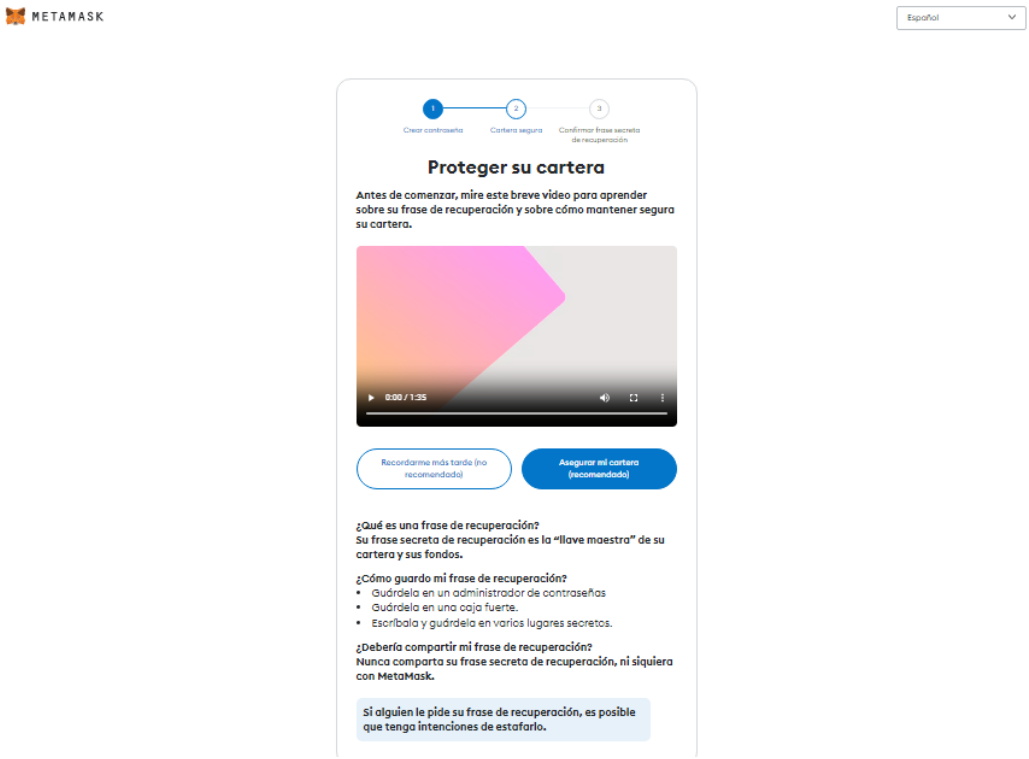

Figura 12. Pantalla de protección de cuentas de Metamask.

Al darle al botón de asegurar cuenta generará una clave secreta de recuperación de 12 palabras (Figura 13), la cual es imprescindible para recuperar la contraseña, por lo que es

necesario su almacenamiento seguro, en el siguiente paso (figura 14) pide confirmar la frase secreta y se pulsa confirmar, así se obtiene la cartera segura. Al finalizar este paso sale en la pantalla “Creación Exitosa de Cartera” (Figura 15). Luego al pulsar el botón de entendido saldrá la pantalla de finalización de instalación (Figura 16) se siguen los pasos para agregar el icono al navegador y saldrá la interfaz de la cartera con la cuenta de Ethereum que se acaba de crear (Figura 17).

Figura 13. Pantalla de generación de frase secreta de Metamask.

Figura 14. Pantalla muestra la frase secreta de Metamask.

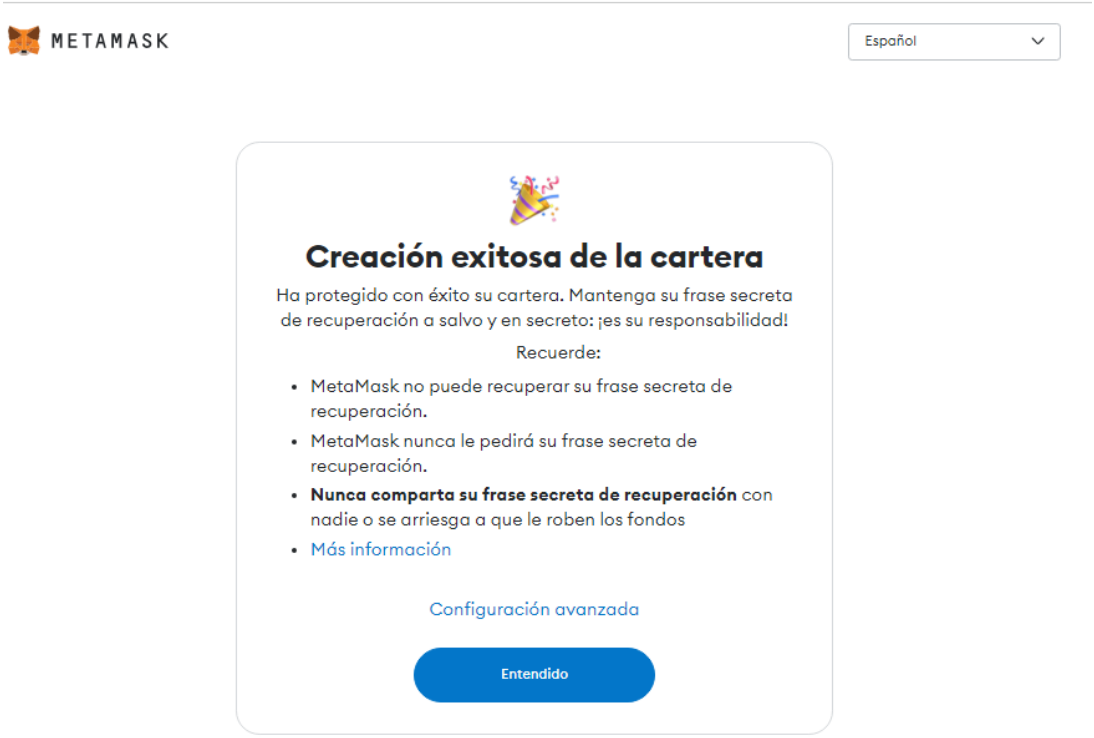

Figura 15. Pantalla de creación de la cuenta de Metamask.

Figura 16. Pantalla de finalización de la instalación de Metamask.

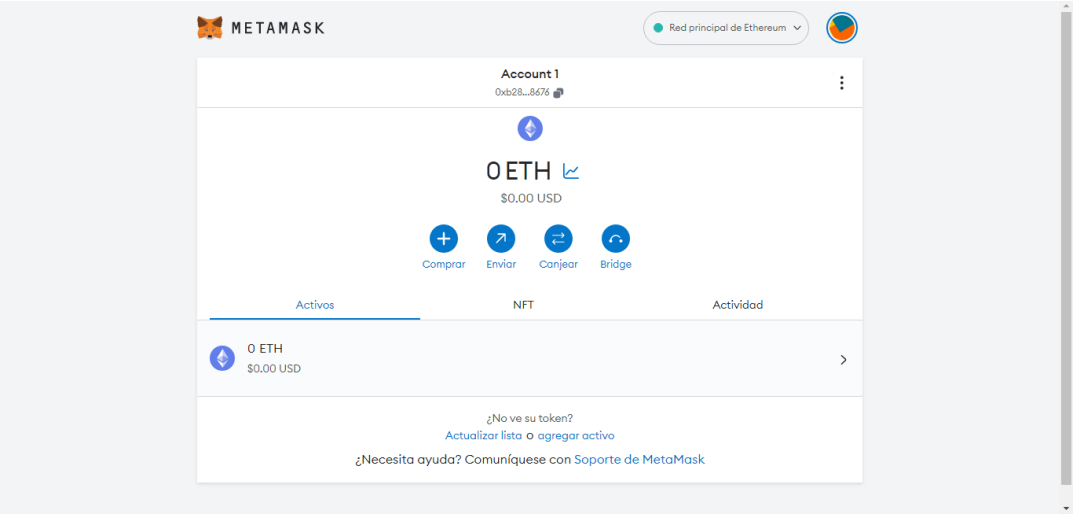

Figura 17. Pantalla de Inicio de Metamask.

3.1.2 Conexión con Ganache

Para hacer uso de la billetera de metamask y así ejecutar las operaciones necesarias en la aplicación descentralizada es necesario cierta cantidad de gas, en este caso ether. Se utilizó Ganache para simular el gas necesario y así lograr realizar las pruebas.

El primer paso es descargar el software desde la página principal, la suite de truffle (https://trufflesuite.com/ganache/).

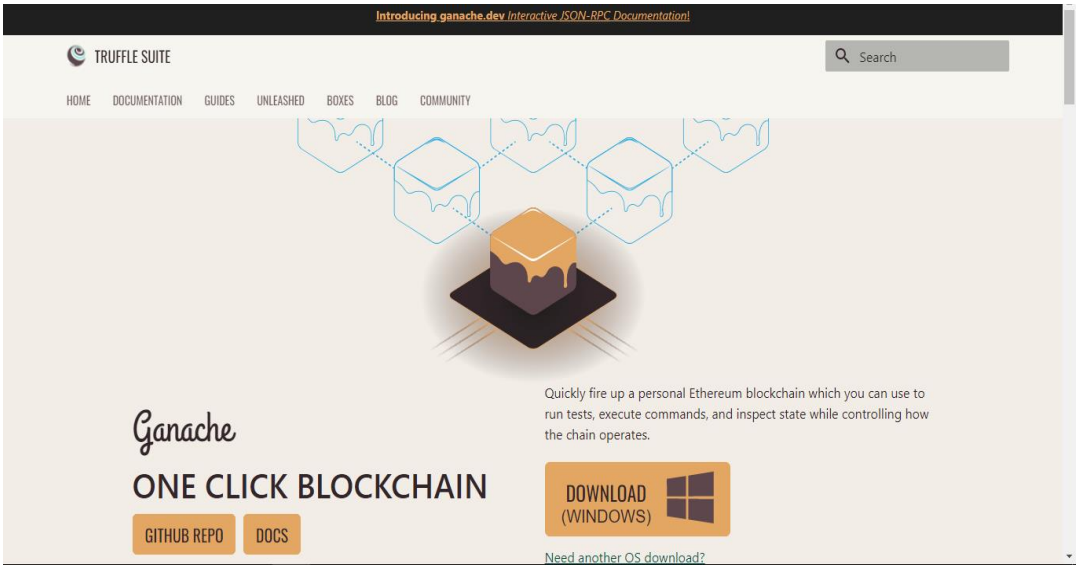

Figura 18. Capture de la página para descargar Ganache.

Luego al descargar el software, se ejecuta y se siguen los pasos para la instalación, al finalizar mostrará una ventana como la de la figura 19, donde se observan todas las cuentas disponibles, incluidas las direcciones, claves privadas, transacciones y saldos.

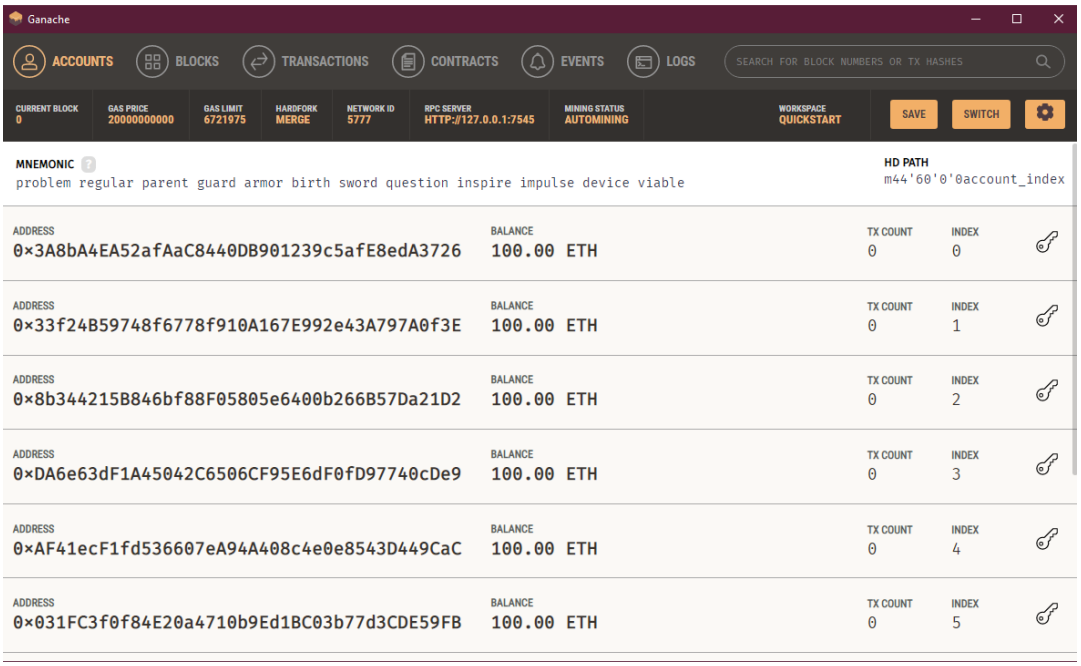

Figura 19. Información de las cuentas disponibles en Ganache.

Al tener instalada la red local (Ganache) se procede a configurar Metamask con la red de pruebas. El primer paso es agregar la red local, por defecto siempre va a aparecer la Red

principal de Ethereum (en caso de usar gas real) así que, hay que agregar una nueva red (Figura 20), luego sale una nueva ventana donde se pulsa en agregar red manual (figura 21) para configurar luego la red local con la dirección que nos proporciona Ganache HTTP://127.0.0.1:7545 (Figura 22) y se guarda.

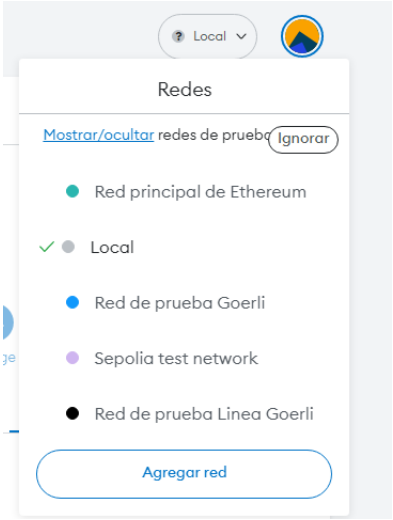

Figura 20. Agregar red en Metamask.

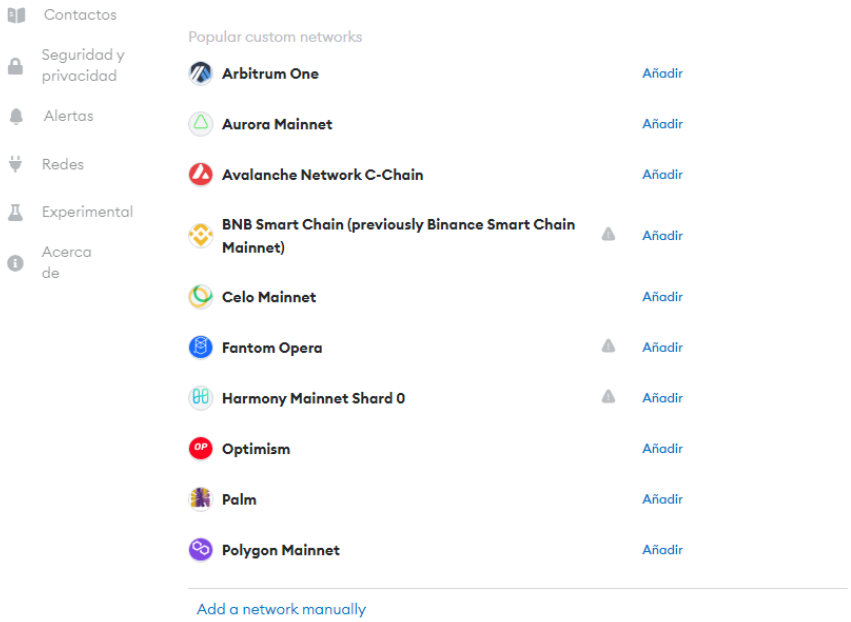

Figura 21. Agregar red manualmente en Metamask.

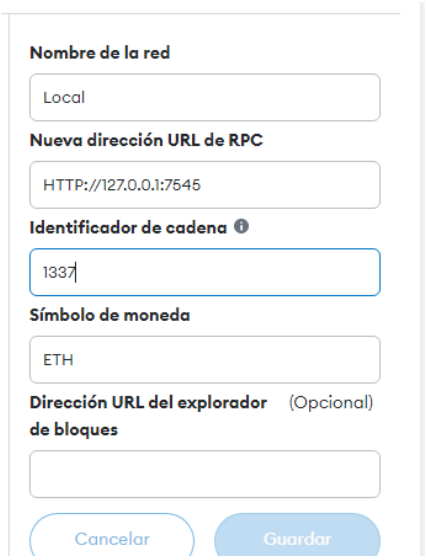

Figura 22. Configuración de la red local.

Cuando se tiene ya configurada la red local, se importa la cuenta ganache a la billetera de metamask (Figura 23) y se pulsa en importar por clave privada (figura 24). Esta clave se puede visualizar en ganache en el menú de cuentas haciendo clic en la llave de cualquiera de las cuentas (figura 25). Al final se puede visualizar la cantidad de saldo disponible en la cuenta (figura 26).

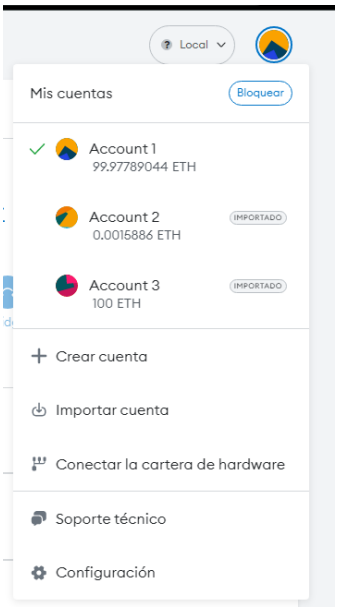

Figura 23. Menú de importación de cuenta.

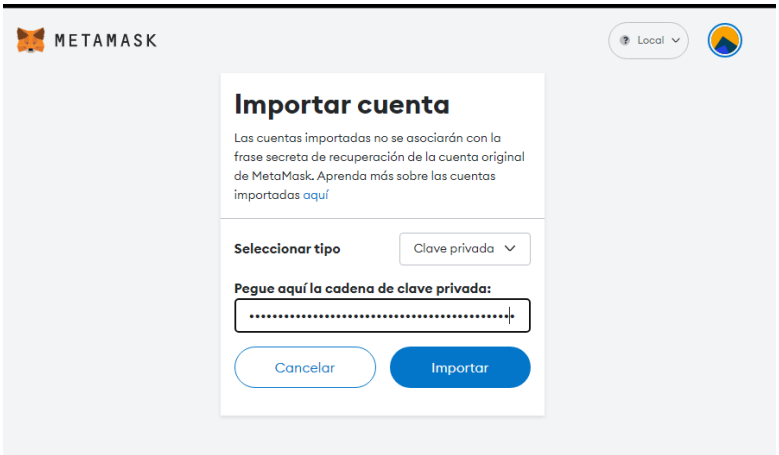

Figura 24. Importación de cuentas mediante la clave privada.

Figura 25. Muestra de la clave privada.

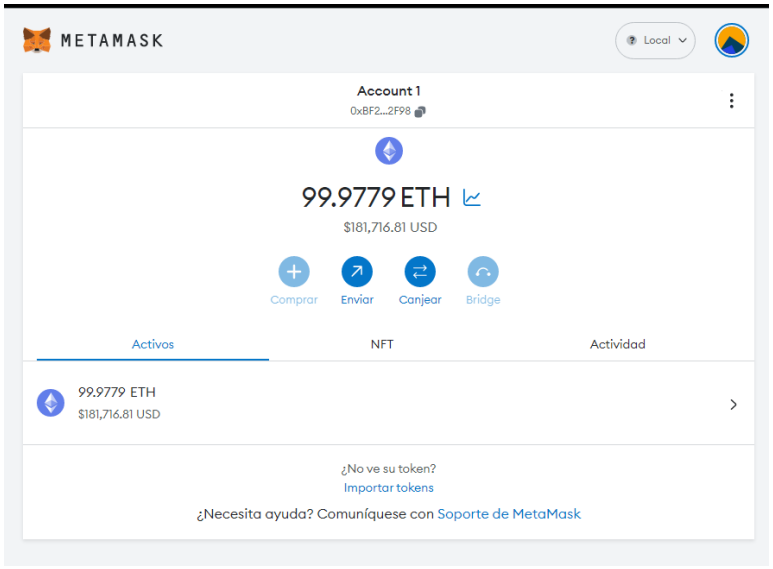

Figura 26. Saldo de la cuenta importada.

Cada vez que el usuario desee importar tantas cuentas como sean necesarias, deberá aplicar el mismo procedimiento.

3.1.3 Conexión con la red de prueba Goerli.

Esta red de prueba se puede añadir directamente en metamask, seleccionándola como se muestra en la figura 27. Cuando ya está seleccionada la red de pruebas, se obtiene el gas mediante la página: https://testnet.help/es/ethfaucet/goerli#log, será necesario llenar el formulario y aceptar los términos para que el dinero ficticio se pueda visualizar en la wallet.

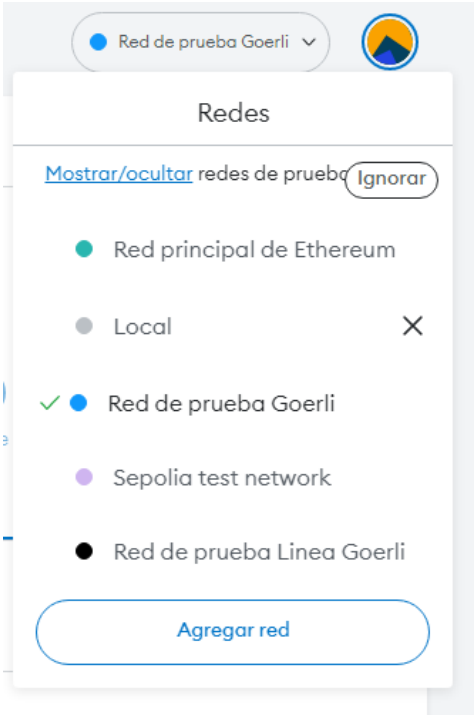

Figura 27. Conexión con la red de pruebas Goerli.

Las pruebas para la aplicación se pueden realizar con cualquiera de las dos redes, tomando en cuenta que la red utilizada para el despliegue del contrato inteligente debe de ser la misma que se implemente para la aplicación.

3.2 Implementación del contrato inteligente.

Para crear una Dapp es necesaria la implementación de un contrato inteligente, ya que como se describió en el capítulo 2, este es el encargado de controlar las operaciones realizadas dentro de la aplicación. El contrato inteligente implementado en este proyecto está basado en el código de la tesis de Ulises Mateo Ortega: “Aplicación de un Histórico de vehículos sobre Blockchain”, adaptando su funcionalidad a los objetivos propuestos en este proyecto. Fue

desplegado en el entorno de desarrollo integrado Remix (https://remix.ethereum.org/) (Figura 28) y en el lenguaje de programación: Solidity.

Básicamente el contrato inteligente, en este caso, va a permitir gestionar las funciones o eventos realizados dentro de la aplicación, el código utilizado para la implementación del contrato es el que se muestra en la figura 29.

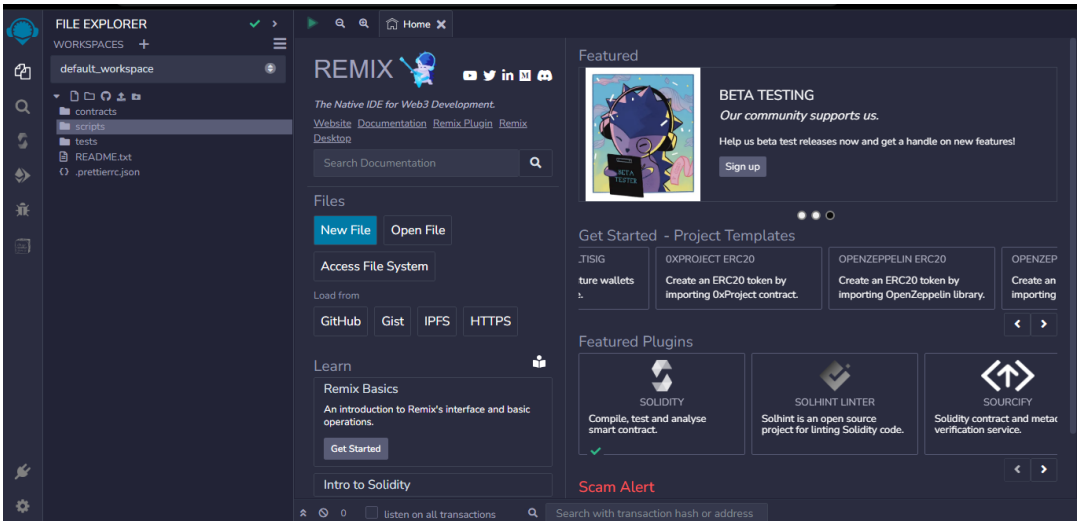

Figura 28. Entorno de desarrollo integrado Remix.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
pragma experimental ABIEncoderV2;

contract Health_registry {
    event thereisanewhistory(string _userid);

    struct Health {
        string userid;
        string[] history;
    }

    mapping(string => Health) internal hh;

    function newHH(string memory _userid) public {
        string[] memory s;
        Health memory user_aux = Health(_userid, s);
        hh[_userid] = user_aux;
        emit thereisanewhistory(_userid);
    }

    function addnewevent(
        string memory _userid,
```

```
        string memory _evento,
        string memory _urloffile
    ) public returns (bool OK) {
        bytes memory tempEmptyStringTest = bytes(hh[_userid].userid);
        if (tempEmptyStringTest.length != 0) {
            hh[_userid].history.push(
                string(abi.encodePacked(_evento, _urloffile))
            );
            //hh[_userid].history.push(_urloffile);
            return true;
        } else return false;
    }

    function getEvents(string memory _userid, uint256 numofchunk)
        public
        view
        returns (string[10] memory _history)
    {
        for (uint256 i = 0; i < 10; i++) {
            _history[i] = "";
        }

        if (hh[_userid].history.length > 0) {
            for (uint256 i = 0; i < 10; i++) {
                if (((numofchunk - 1) * 10) + i < hh[_userid].history.length)
                    _history[i] = hh[_userid].history[
                        ((numofchunk - 1) * 10) + i
                    ];
                else _history[i] = "";
            }
            return _history;
        } else {
            return _history;
        }
    }
}
```

Figura 29. Vista del Contrato Inteligente.

Al desplegar el contrato inteligente en Remix, se debe guardar la Dirección del contrato (Figura 30) y el ABI (Figura 31), ya que serán utilizados más adelante.

```
to = '0xe53e55eF11ED3673C7633AcbC91DD298cfAB0135'; // dirección del contrato goerli metamask
to = '0xeA553aC9fc4a76E7F78db72e6f7Fa68C0Af826C2'; // dirección del contrato ganache metamask
```

Figura 30. Dirección del Contrato Inteligente.

```
ABI =
[
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": false,
                "internalType": "string",
                "name": "_userid",
                "type": "string"
            }
        ],
        "name": "thereisanewhistory",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "string",
                "name": "_userid",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "_evento",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "_urloffile",
                "type": "string"
            }
        ],
        "name": "addnewevent",
        "outputs": [
            {
                "internalType": "bool",
                "name": "OK",
                "type": "bool"
            }
        ],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
```

```
            {
                "internalType": "string",
                "name": "_userid",
                "type": "string"
            }
        ],
        "name": "newHH",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "string",
                "name": "_userid",
                "type": "string"
            },
            {
                "internalType": "uint256",
                "name": "numofchunk",
                "type": "uint256"
            }
        ],
        "name": "getEvents",
        "outputs": [
            {
                "internalType": "string[10]",
                "name": "_history",
                "type": "string[10]"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    }
```


Figura 31. ABI del contrato inteligente.

3.3 Desarrollo de la aplicación

Para el desarrollo de la interfaz gráfica necesaria en la interacción entre el usuario y la Blockchain se utilizó la aplicación base proporcionada por el mismo metamask, que se puede encontrar y descargar del repositorio de github https://github.com/BboyAkers/simple-dapp-

tutorial, al adaptar los cambios necesarios en este proyecto se clonó dicho repositorio, referenciado en el proyecto de tesis de Ulises Mateo Ortega.

3.3.1 Instalación del entorno de desarrollo.

En la implementación se trabajó con diferentes lenguajes, como lo son: HTML, CSS y JavaScript, por lo tanto, es de gran utilidad contar con un editor de texto que soporte todas las funcionalidades del software a ser modificado. El más utilizado y también recomendado en la propia documentación de metamask, para el desarrollo de este tipo de aplicaciones, es el Visual Studio Code (https://code.visualstudio.com/) así como también, es indispensable la descarga del paquete Node.JS (https://nodejs.org/en) que permite la interacción en tiempo real con el servidor, como se explicó en el capítulo 2.

Al tener instalado el software requerido se muestran las carpetas: finished y start, los archivos a utilizar están en la carpeta "start" por lo que se puede eliminar la otra para evitar confusiones, si así lo requiere.

Al abrir la carpeta start se pueden visualizar los archivos que se muestran en la figura 32, de los cuales se modificarán más adelante el contract.js (controlador de la aplicación) y el index.html (interfaz gráfica de la aplicación).

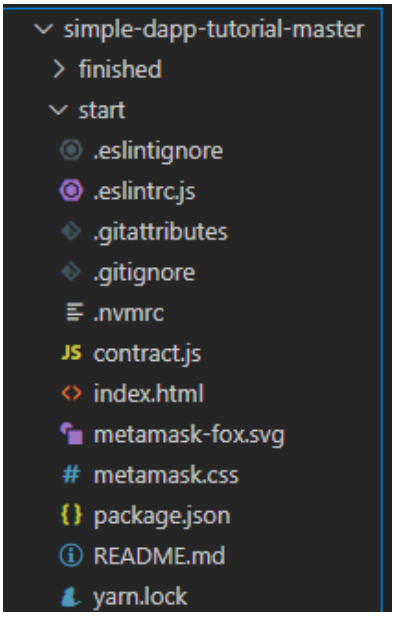

Figura 32. Repositorio de Metamask.

Para realizar las modificaciones pertinentes y probar la aplicación, se necesita instalar todas las dependencias necesarias para el proyecto. Al instalar se abre el terminal y se ubica en la carpeta "start", una vez dentro se ejecuta el siguiente comando:

```
$ npm install
```

Al finalizar la descarga se visualiza una carpeta con el nombre “node_modules” donde se guardan los archivos necesarios para la ejecución de la aplicación. Para ejecutar se utiliza siguiente comando:

```
$ npm run serve
```

Con esta línea de comando se dará servicio al puerto: http://localhost:9011/ que al hacer clic nos redireccionará a la página de prueba de metamaks con Ethereum.

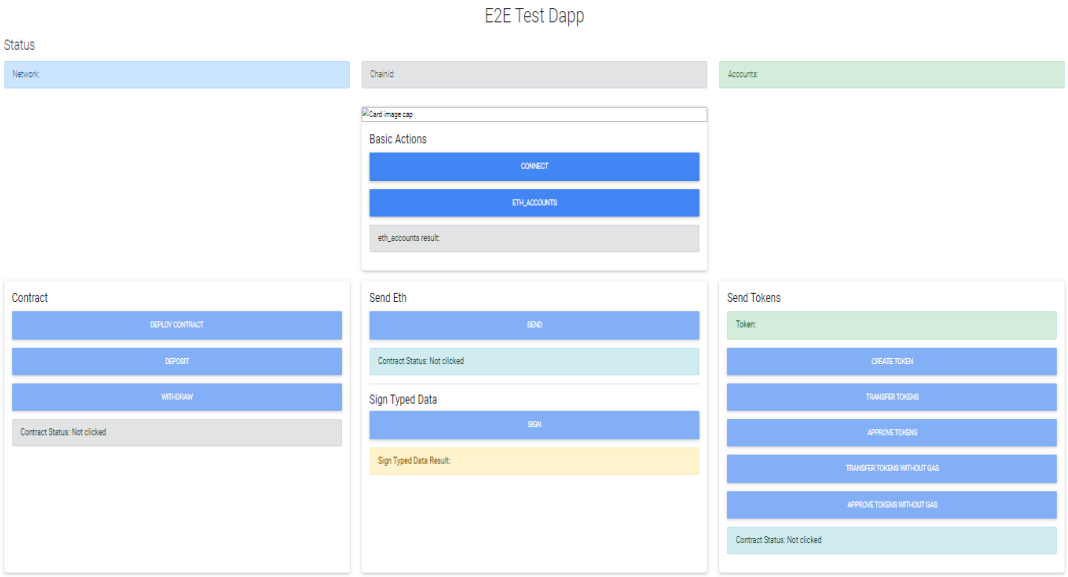

Figura 33. Página de prueba de Metamask.

3.3.2 Desarrollo de la Interfaz gráfica.

En el desarrollo de la interfaz para este proyecto se modificará el archivo “index.html” que se encuentra en el repositorio de metamask. Con esta interfaz se pretende ejecutar las funciones del contrato inteligente, por lo que la página web será dividida en tres partes más la función de conectar la wallet de Metamask donde se podrá desplegar el contrato con Ethereum.

El primer fragmento de código HTML se nombra como: “DATA MEDICA” (figura 34) en el cual se dejaron las importaciones preestablecidas de algunas dependencias.

```
<head>
  <meta charset="UTF-8">
  <title>DATA MEDICA</title>
  <link rel="icon" type="image/png" href="metamask-fox.svg">


  <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap">
  <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.4.1/css/bootstrap.min.css" rel="stylesheet">
  <link href="https://cdnjs.cloudflare.com/ajax/libs/mdbootstrap/4.14.1/css/mdb.min.css" rel="stylesheet">
  <link rel="stylesheet" href="metamask.css" type="text/css">
</head>
```

Figura 34. Data médica-Fragmento del código del index.html.

En las modificaciones realizadas al cuerpo de la página desde la etiqueta <body> se resalta el cambio del color de fondo para darle un toque más atractivo a la interfaz, en el comienzo se agregó la línea “style="background-color: #a8e5eb;" dentro de la etiqueta de inicialización del cuerpo. Así mismo se adicionaron algunos estilos, como el tamaño en el texto y el cambio de algunos colores para realizar una distinción notable entre las interfaces del repositorio proporcionado por metamask.

La Primera parte de la codificación para la visualización de la página, es el nombre y el botón de conexión con metamask, este cambia según el contexto, si detecta que la extensión ya se encuentra instalada aparecerá “Conectar con metamask” si no está instalada aparecerá “¡Hacer clic aquí para instalar metamask!”, todo esto esta codificado en el archivo contract.js el cual se explicará más adelante. En esta parte de la instalación solo se requirió “activar sistema” y las otras opciones se eliminaron ya que no son necesarias, así como también las columnas previas a esa función, quedando el código como se muestra en la figura 35.

```
<body style="background-color: #a8e5eb;">
  <main class="container-fluid">
    <header>
      <h1 class="text-center" style="color: blue;"><b>Data Médica</b></h1>
    </header>


    <!-- Part 1 Configuracion de acciones y estados -->
    <section>
      <div class="row d-flex justify-content-center">
        <div class="col-xl-4 col-lg-6 col-md-12 col-sm-12 col-12">
          <h2 class="card-title" style="color: rgb(10, 10, 10)"><b>Activar Sistema</b></h2>
          <button class="btn btn-primary btn-lg btn-block mb-3" id="connectButton" disabled></button>
        </div>
      </div>
    </div>
  </section>
```

Figura 35. Activar sistema del código index.html.

La siguiente sección (<! -- Part 2 Contract -->) es donde se codificaron las funciones que se conectan con el controlador, fueron útiles las tres columnas del repositorio eliminando solo algunas opciones de cada columna.

En el primer solo se utilizó una opción para hacer el registro de usuario, conservando el botón que se estableció para la conexión con la wallet y el despliegue de la primera condición del contrato inteligente (figura 36).

```
<!-- Part 2 Contract  -->
 <section>
 <div class="row">
      <!-- Registro de usuario-->
      <div class="col-md-4 card-body " >
        <h2 class="card-title"> <b>Registro de Usuario</b></h2>

        <label><h4>ID de usuario: </h4></label>
        <input type="text" id="user_registry"></input>
        <button class="btn btn-primary btn-lg btn-block mb-3" id="registry"><b>Registrar Persona</b></button>
      </div>
```

Figura 36. Registro de usuario del código index.html.

En el segundo se utilizó el mismo botón para conectar con la wallet y añadir el archivo. Se agregaron algunos elementos: el type=“text” para identificar que se trata de un texto y el id= “user_event”g para identificar el ID de usuario. Al definir los tipos de eventos, se agregaron: el type="file" para seleccionar el evento y el id="file_event" para identificar el tipo de evento (consulta médica y reporte de laboratorio) el resultado de la codificación se puede visualizar en la figura 37.

```
      <!-- Introducción de un evento -->
      <div class="col-md-4 card-body ">
        <h2 class="card-title"><b>Guardar Informe</b></h2>
        <label><h4>ID de usuario: </h4></label>
        <input type="text" id="user_event"></input><br>
        <label for="type_event"><h4>Tipo: </h4></label>
        <select class="form-control" style= "color: #064d53" id="type_event" >
          <option><h4>Consulta Médica</h4></option>
          <option><h4>Reporte de Laboratorio</h4></option>
        </select>
        <label><b><h4>Subir informe: </h4></b></label>
        <input type="file" id="file_event" ></input>
        <button class="btn btn-primary btn-lg btn-block mb-3" id="addEvent"><b>Añadir Informe</b></button>
      </div>
```

Figura 37. Introducción de un evento del código index.html.

Ahora en la tercera y última división se hicieron las configuraciones para leer y mostrar el archivo del apartado anterior, agregando los elementos: id="user_read" para leer la ID del usuario y el id="chunk_read" para leer el número de informe (figura 38).

```
<!-- Lectura de informes-->
<div class="col-md-4 card-body " >
  <h2 class="card-title"><b>Lectura de informes</b></h2>
  <label><h4>ID de usuario: </h4></label>
<input type="text" id="user_read"></input>
<label><h5>Número de informe: </h5></label>
<input type="number" id="chunk_read"></input>
<button class="btn btn-primary btn-lg btn-block mb-3" id="getEvents"><b>Mostrar informe</b></button>

</div>
```

Figura 38. Lectura de Informes del código index.html.

Luego de realizar todos los cambios y modificaciones para adaptar la interfaz gráfica al proyecto actual, se logra visualizar el contexto del archivo index.html con el navegador de preferencia, como se muestra en la figura 39.

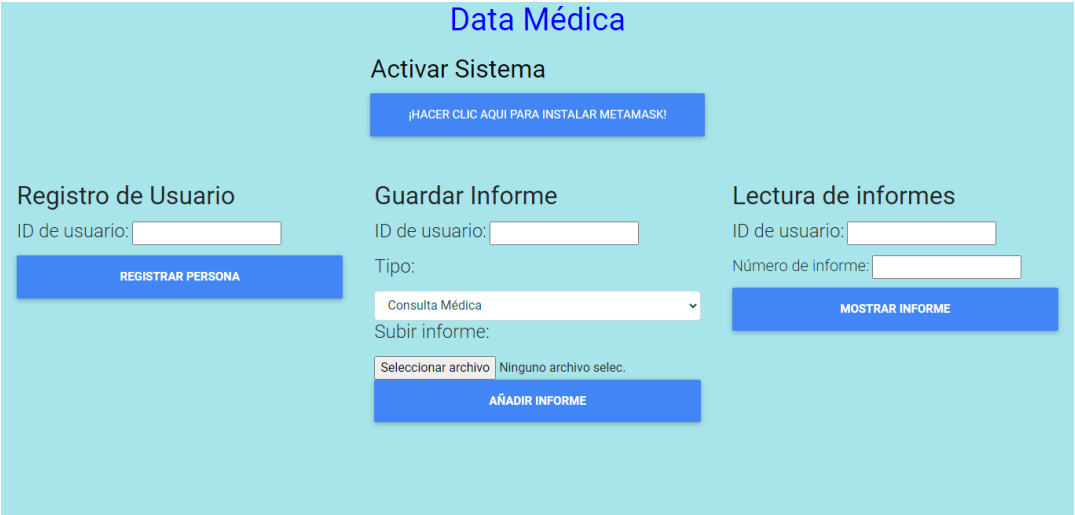

Figura 39. Interfaz gráfica de la aplicación.

3.3.3 Desarrollo del controlador.

El controlador es uno de los archivos más importantes, es allí donde se ejecuta la mayor parte del proyecto. Se modifica el archivo “contract.js” que se encuentra en el repositorio de metamask y de él se conservan las funciones de instalación y conexión con la wallet de metamask, además del botón de redireccionamiento:

- ✓ **`const isMetamaskInstalled`** => Función de verificación que comprueba si la extensión de metamask está instalada en el navegador (Figura 40).

- ✓ **`const onboarding = new MetaMaskOnboarding({ forwarderOrigin })`** => crea un nuevo objeto de incorporación de metamask para usar en la aplicación y es necesario para hacer uso del siguiente método (Figura 40).
- ✓ **`const onClickInstall`** => este método es el que redirige al usuario a la página para instalar metamask al hacer clic cuando esté en ese estado (Figura 41).
- ✓ **`const onClickConnect`** => Cuando metamask ya se encuentra instalado, salta a la ventana para seleccionar la cuenta (Figura 41).
- ✓ **`const MetamaskClientCheck`** => esta función es la que llama a los métodos de instalar y conectar, si no está instalado aparece en el botón "¡Hacer click aquí para instalar Metamask!" de lo contrario aparecerá "Conectar con metamask" (Figura 41).

Estas son las primeras acciones de la aplicación, con las que se conecta a la wallet para ejecutar el contrato inteligente en las siguientes funciones. A parte de estos métodos también se conserva la primera línea de código porque es la dirección del servidor web que proporciona Node.js:

```
const forwarderOrigin = 'http://localhost:9010'
```

```
console.log('initializing');

//Basic Actions Section
const onboardButton = document.getElementById('connectButton');

//creación de la funcion de verificación creada para ver si la extension metamask esta instalada
const isMetaMaskInstalled = () => {
  //comprueba si el navegador tiene la extensión Metamask instalada
  const { ethereum } = window;
  return Boolean(ethereum && ethereum.isMetaMask);
};

 //Se crea un nuevo objeto de incorporación de metamask para usar en la app
 const onboarding = new MetaMaskOnboarding({ forwarderOrigin }); // para hacer uso del metodo on onClickInstall
```

Figura 40. Función de verificación de Metamask.

```
const onClickInstall = () => {
  // Al clicar en el botón en este estado, redirige al usuario a una pagina para instalar metamask
  onboardButton.innerText = 'Incorporación en curso'; // Incorporacion en curso
  onboardButton.disabled = true;
  //En este objeo tenemos startOnboarding que iniciara el proceso de incorporación a la pagina
  onboarding.startOnboarding();
};

const onClickConnect = async () => {
  try {
    //salta la ventana emergente de Metamask para que se seleccione la cuenta
    await ethereum.request({ method: 'eth_requestAccounts' }); // eth_solicitud de cuentas
  } catch (error) {
    console.error(error);
  }
};


const MetamaskClientCheck = async () => {
  if (!isMetaMaskInstalled()) {
    //si no esta instalado se llama al método que redirija al usuario
    onboardButton.innerText = '¡Hacer clic aqui para instalar Metamask!'; // Haga clic aqui para instalar Metamask!
    onboardButton.onclick = onClickInstall;
    onboardButton.disabled = false;
  } else {
    //Si ya lo estuviera, se conecta con la cuenta seleccionada
    onboardButton.innerText = 'Conectar con metamask'; //Conectar
    onboardButton.onclick = onClickConnect;
    onboardButton.disabled = false;
  }
};
```

Figura 41. Funciones de instalación y redirección de Metamask.

En la modificación de los métodos para interactuar con el contrato inteligente, lo primero es declarar los atributos para cada función, luego de la primera línea de código (conexión con el servidor) como se muestra en la figura 42.

```
const forwarderOrigin = 'http://localhost:9010';

//declaracion de todos los botones y inputs de la pagina

const userRegistryButton      = document.getElementById('registry');
const addEventButton          = document.getElementById('addEvent');
const getEventsButton        = document.getElementById('getEvents');
const user_event              = document.getElementById('user_event');
const type_event              = document.getElementById('type_event');
const user_read               = document.getElementById('user_read');
const chunk_read              = document.getElementById('chunk_read');
```

Figura 42. Declaración de los atributos de las funciones.

Ya declarados todos los botones e inputs de la página se codifica cada método:

En primer lugar, se conecta con el contrato instanciando un “provider”, el cual es un objeto auxiliar que se conectará con el contrato inteligente y un “signer” que es una firma que permite hacer cambios en la blockchain con autorización del cliente. Siendo así, se crearon dos variables donde sus valores son, la dirección del contrato y el ABI, obtenidos previamente al

ser desplegados y para finalizar se crearon dos constantes donde se conectaron los elementos (Figura 43).

```
const provider = new ethers.providers.Web3Provider(window.ethereum);

const signer = provider.getSigner();


//const contractAddress = '0xe53e55eF11ED3673C7633AcbC91DD298cfAB0135'; // dirección del contrato goerli metamask
const contractAddress = '0xeA553aC9fc4a76E7F78db72e6f7Fa68C0Af826C2'; // dirección del contrato ganache metamask

console.log('connecting to contract:', contractAddress);

console.log('provider', provider, 'code', provider.getCode(contractAddress));

// ABI del contrato
const  contractABI = [···
]

const myContract = new ethers.Contract(contractAddress, contractABI, provider);

const myContractWithSignature = myContract.connect(signer);
```

Figura 43. Declaración de las acciones para conectar con contrato.

Luego se declara la variable “fileData” vacía, que sirve para guardar el archivo y luego una función donde se hace uso del elemento declarado en el index.html, el id = “file _event” que recoge la información del archivo (Figura 44).

```
var fileData = '';
const fileSelector = document.getElementById('file_event');
fileSelector.addEventListener('change', (evento) => {
    fileData = evento.target.files[0];
    console.log(fileData);
});
```

Figura 44. Declaración de la variable para guardar el archivo.

1.- Método para el registro de usuario:

Es el encargado de llamar a la función del contrato “newHH” donde se guarda el ID del usuario. Este método se ejecuta al hacer clic en el primer botón de la interfaz donde se registra al usuario.

```
//Metodo que se encarga de registrar usuario
userRegistryButton.onclick = async () => {
try {
  let user = document.getElementById('user_registry').value;
  await myContractWithSignature.newHH(user);
  console.log(user);
} catch (error) {
  console.error(error);
}
};
```

Figura 45. Método para el registro de usuario.

2.- Método para añadir informe:

Con esta función se añadirá el archivo. Primero se declaran los atributos “user_event” y “type_event” que ya fueron llamados en el index.html, lo siguiente será agregar la dirección hash que devolverá un string, para guardar el evento y luego subirlo a Ethereum (Figura 46).

```
//Metodo que se encarga de guardar un evento (consulta medica,  reporte de laboratorio)
addEventButton.onclick = async () => {
  try {
        let user = user_event.value;
        console.log(user);
        //Los tipos de eventos pueden ser (consulta medica , reporte de laboratorio)
        let typeOfEvent = type_event.value;
        console.log(typeOfEvent);
        //Se recoge el hash del fichero subido
        const _urloffile = URL.createObjectURL(fileData);
        console.log(_urloffile);
        //Se sube el evento a Ethereum
        console.log("guardar archivo");
        let file = await myContractWithSignature.addnewevent(user,typeOfEvent, _urloffile);
        console.log(file);
  } catch (error) {
        console.error(error);
  }
};
```

Figura 46. Método para añadir informe.

2.- Método para mostrar informe:

En este método se llamarán los id del index.html para la lectura de los informes, el id="user_read" para guardar el ID de usuario y el id="chunk_read” para el número de informe. Por último, se añadirá código HTML para declarar los contenedores que mostrará el archivo guardado (figura 47).

```
getEventsButton.onclick = async () => {
  try {
        let user = user_read.value;
        let numOfChunk = chunk_read.value;
        //Se recogen los eventos en funcion a su usuario
        console.log('_userid', user, 'numOfChunk', numOfChunk, 'myContract', myContract);
        let result = await myContract.getEvents(user, numOfChunk);
        console.log('events -> ', result);
        let cadena = result[0].split("Consulta Médica");
        let cadena2 = result[0].split("Reporte de Laboratorio");
        console.log(result[0]);

        // En JS se puede "inyectar" codigo HTML en cualquier elemento teniendo únicamente su id
        if (cadena  != "undefined") {
            document.getElementById("contenedor").innerHTML = '<table>'
            + ' <tr><td> Archivo:  </td><td><a target="_blank" href="'+ cadena[1] +'">'+ cadena[1] +'</a></td>'
            + '</table>';
        }
        if (cadena2 != "undefined") {
            document.getElementById("contenedor2").innerHTML = '<table>'
            + ' <tr><td> Archivo:  </td><td><a target="_blank" href="'+ cadena2[1] +'">'+ cadena2[1] +'</a></td>'
            + '</table>';
        }
  } catch (error) {
        console.error(error);
    }
};
```

Figura 47. Método para mostrar informe.

Luego de la inicialización se codifica la siguiente línea de código, que se inicia al cargar toda la página html.

```
window.addEventListener('DOMContentLoaded', initialize);
```

Por último, será necesario agregar al archivo “index.html” los id (id="contenedor”, id="contenedor2”) y los scripts para interactuar con las dependencias y el controlador (Figura 48), ya con esto se han terminado las modificantes y se ha adaptado el modelo al proyecto, por lo tanto, se puede ejecutar la aplicación.

```
    </section>
    <div id="contenedor"></div>
    <div id="contenedor2"></div>
  </main>

  <script src="node_modules/@metamask/onboarding/dist/metamask-onboarding.bundle.js" defer></script>
  <script src="https://cdn.ethers.io/lib/ethers-5.0.umd.min.js" type="application/javascript"></script>
  <script src="contract.js" defer></script>

</body>

</html>
```

Figura 48. Scripts del index.html.

3.3.4 Pasos para ejecutar la aplicación.

Se abre el terminal del editor de texto que se esté utilizando, en este caso se usó Visual Studio Code (https://code.visualstudio.com/) y se siguen los siguientes pasos:

Paso 1: Se instalan las dependencias necesarias para ejecutar el código, si aún no se tienen instaladas (como se describió al comienzo de este capítulo), esto con el siguiente comando:

```
$ npm install
```

Paso 2: Se instala el paquete para ejecutar el scripst de ethers:

```
$ npm install –save ethers
```

Paso 3: Para ejecutar con el comando:

```
$ npm run serve
```

Luego se da clic en el vínculo: http://localhost:9011 proporcionando servicio a la aplicación, se utiliza **Ctrl+c** para finalizar la ejecución**.**

Capítulo 4: Pruebas y resultados de la aplicación descentralizada.

1. Prueba de conexión con metamask

Para conectar con la wallet de metamask la aplicación detecta si la extensión ya se encuentra instalada en el navegador por lo tanto se tienen los siguientes casos: El primero es cuando no se encuentra instalado, por lo que se apreciará en el botón de activar sistema, la descripción ¡Hacer clic aquí para instalar Metamask!.

Al hacer clic en el botón de activar sistema, direcciona a la página de instalación de la extensión, se añade en el navegador y se siguen los pasos descritos en el apartado 3.1 del capítulo 3. Mientras se instala en el navegador se mostrará en el botón "Incorporación en curso" hasta que detecte que la extensión está habilitada.

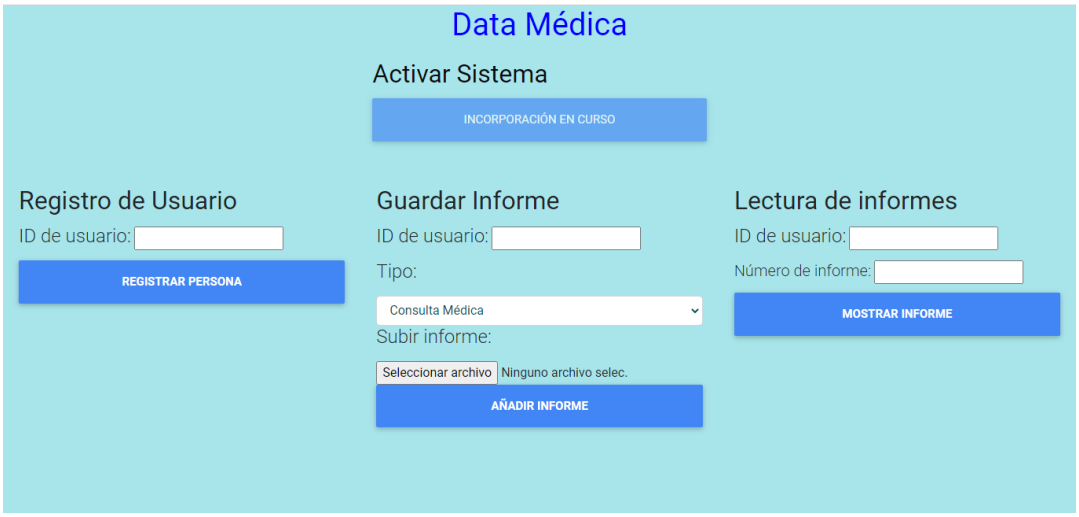

Figura 49. Incorporación en curso.

El segundo caso es cuando la aplicación detecta que ya la extensión de la wallet se encuentra instalada. En este caso solo se le da clic al botón Conectar con metamask (Figura 50) y muestra una ventana emergente para conectar a la wallet (Figura 51). Esta conexión es necesaria, ya que sin wallet no se puede desplegar el contrato ni realizar las pruebas necesarias para la aplicación.

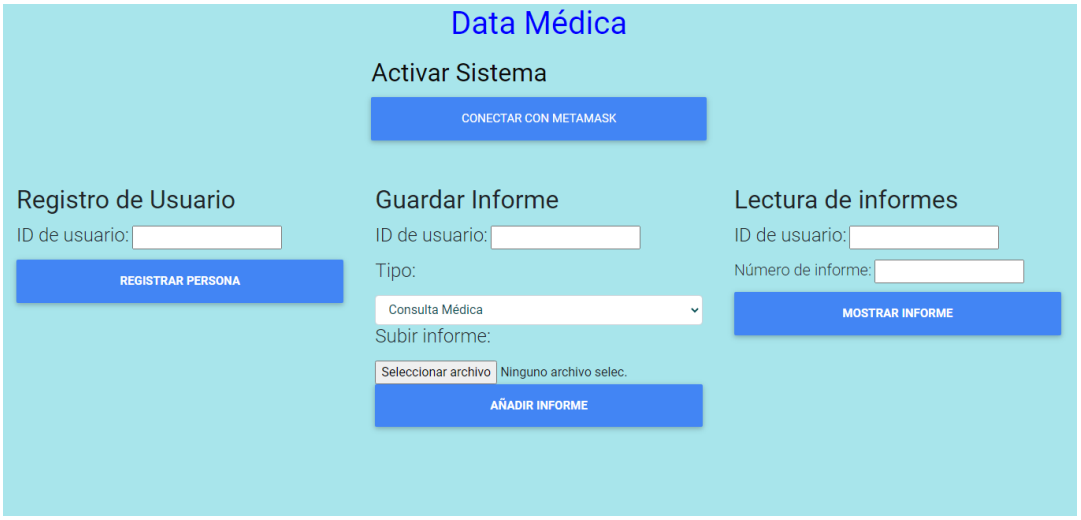

Figura 50. Conexión con metamask.

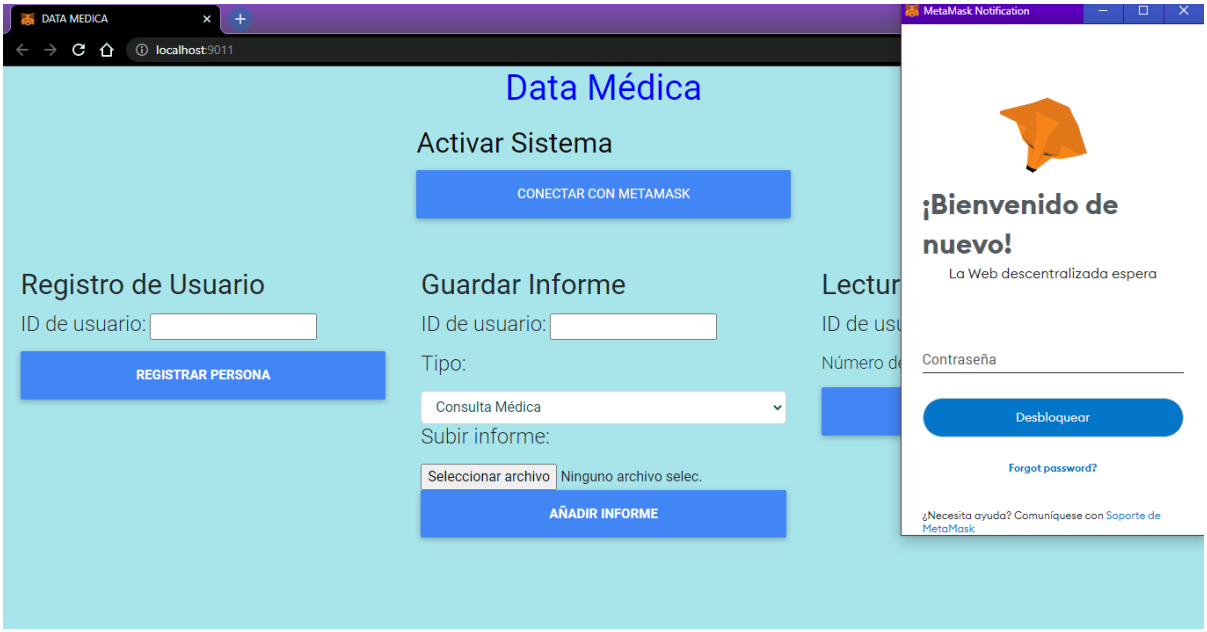

Figura 51. Ventana emergente de la conexión con metamask .

2. Prueba del registro de usuarios.

En este apartado de la aplicación se guarda el registro del ID del usuario para incorporarlo al contrato. Se escribe en el recuadro en blanco el ID de preferencia y se pulsa el botón de registrar persona, al pulsar dicho botón se muestra la ventana emergente de metamask, donde la wallet calcula la cantidad de gas requerido para ejecutar la transacción (estos cálculos no siempre son los mismos, hay un cambio cada vez que se ejecutan las transacciones). En esta opción se debe esperar unos segundos para que se muestre la

opción de rechazar o confirmar. Al confirmar se cierra la ventana emergente, y así se ha incorporado el registro al contrato.

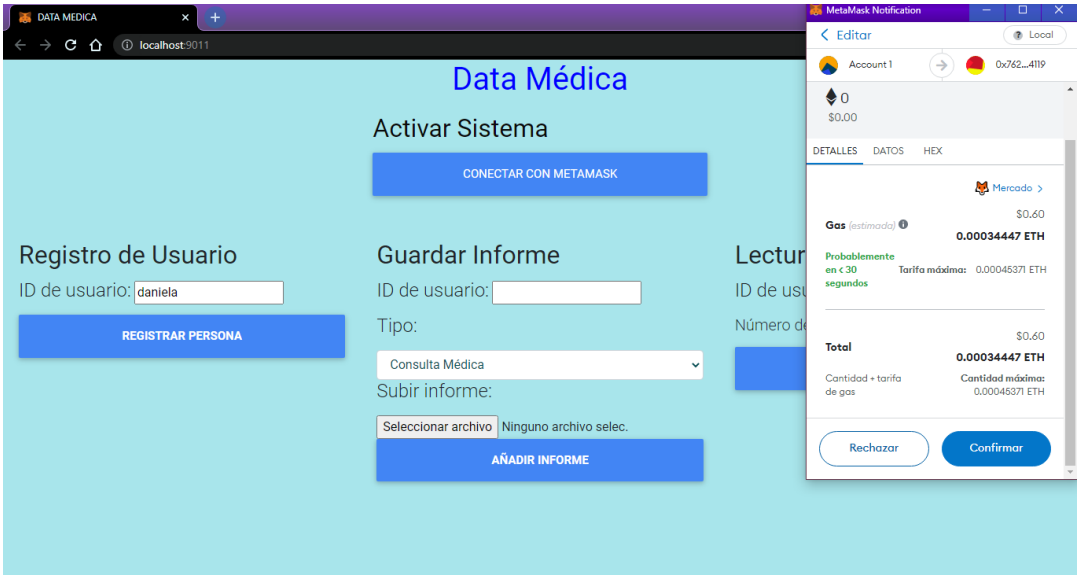

Figura 52. Prueba del registro de usuario.

3. Prueba para añadir informe.

En esta acción, se debe proporcionar el ID de usuario ya registrado y seleccionar el tipo de archivo que será subido, en este caso el de consulta médica o reporte de laboratorio, luego de seleccionar el tipo de archivo, se debe subir el informe, esto mediante el botón de “seleccionar archivo”, al hacer clic sobre ese botón se selecciona el archivo (jpg, png, pdf…) y al añadirlo se despliega nuevamente la ventana emergente para confirmar la transacción.

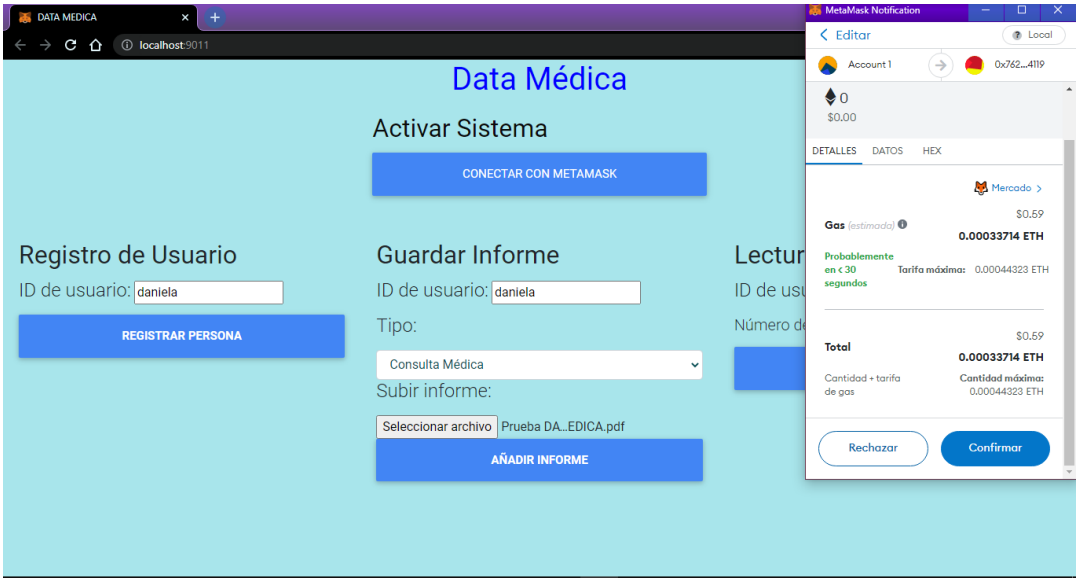

Figura 53. Prueba para añadir informe.

4. Prueba para la muestra de informe.

En esta última acción se registra solo la lectura del archivo, por lo que no requiere la conexión con la wallet, siendo un procedimiento casi instantáneo. En este paso se proporciona igualmente la ID de usuario y se selecciona el número de informe. Al introducir estos valores, solo se pulsa el botón “Mostrar Informe” donde automáticamente aparece la url del archivo en la parte de abajo al lado izquierdo de la página, el cual se le puede dar clic para mostrarse en otra dirección donde se visualizará para leer, imprimir o descargar.

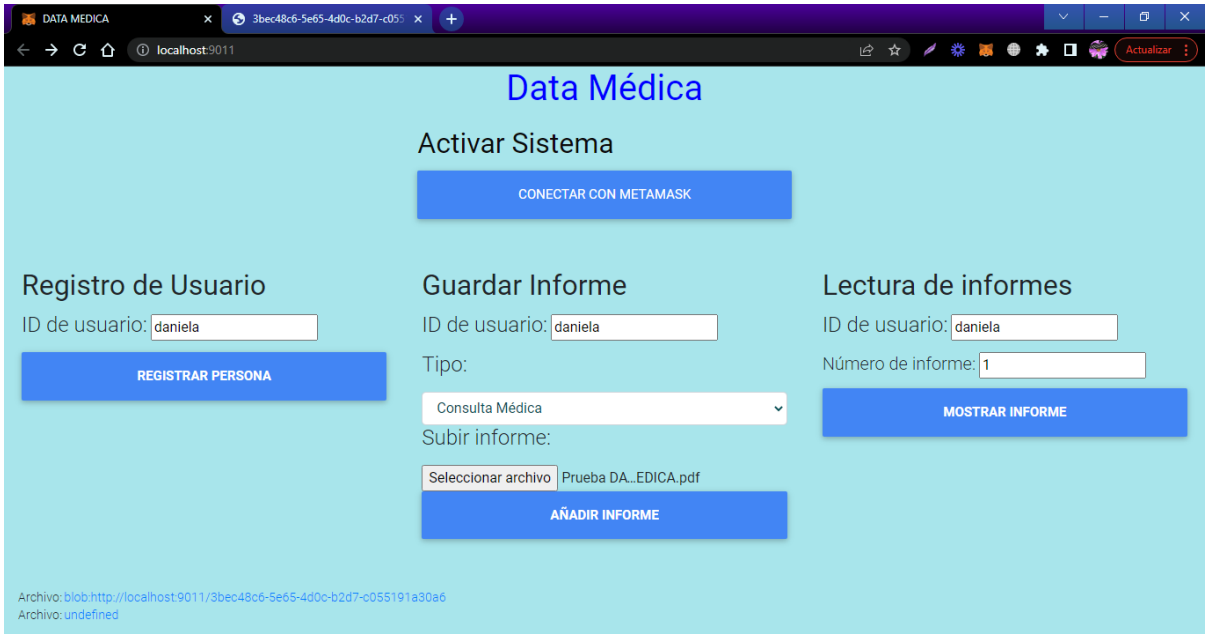

Figura 54. Prueba para la muestra del informe.

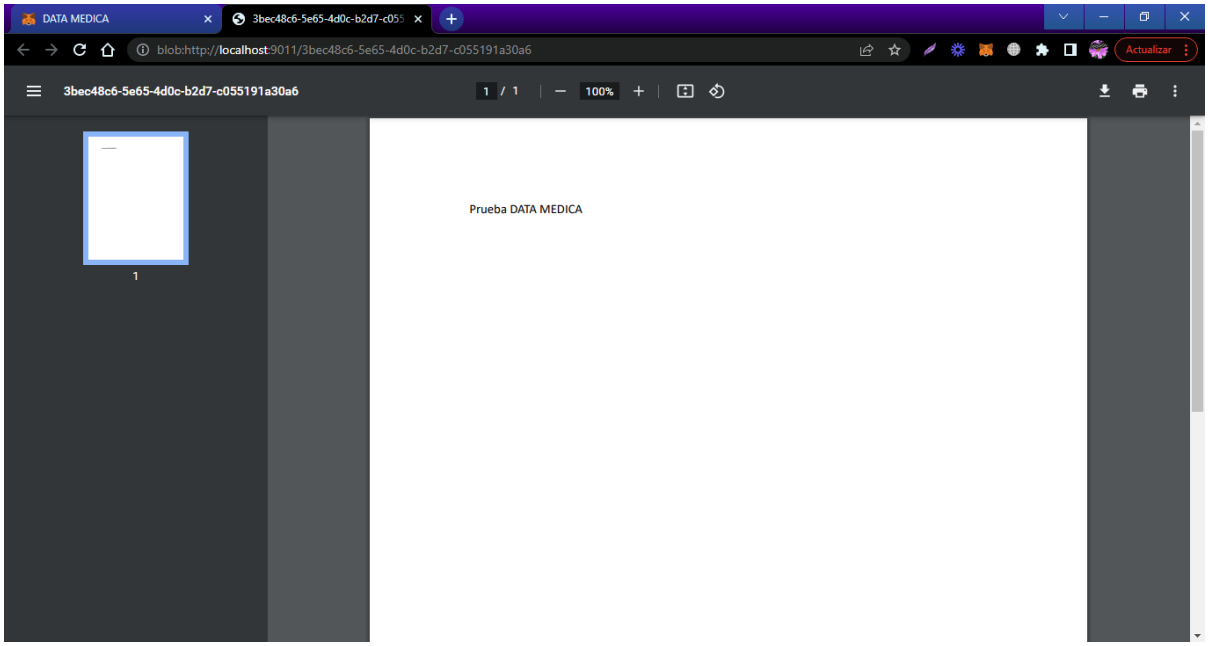

Figura 55. Visualización del archivo.

Capítulo 5: Conclusiones.

El registro de historias médicas, es uno de los elementos de contacto indispensable entre el personal de salud y el paciente; en ello radica la importancia que el paciente tenga acceso a la información de su historial de salud.

Con el uso de esta aplicación se introducen los datos a una blockchain, siendo una tecnología con gran auge en la industria, la cual trabaja con cadenas de bloques, proporcionando la seguridad necesaria para el registro de este tipo de archivos.

El proyecto fomenta la posibilidad del registro de historias médicas a través de una aplicación manipulada por parte del paciente, sin necesidad de depender de las instituciones centrales. Aunado a ello, se logró implementar una interfaz gráfica básica para la conexión con un contrato inteligente.

Por medio de la interfaz implementada, se puede conectar a una wallet de Ethereum, en este caso metamask; realizando la conexión del contrato para guardar la información necesaria. La interfaz cuenta con tres funciones básicas: registro de usuario, añadir informe y mostrar informe; siendo así, se logró conectar la aplicación con una Blockchain consiguiendo guardar los datos y el archivo, así como también recuperarlos de la red para mostrarlo, descargarlo o para imprimirlo.

Cabe destacar, que siendo una aplicación descentralizada es capaz de implementarse en diferentes sistemas operativos, lo que trae consigo un beneficio para la ejecución de la aplicación y al ser de código abierto puede quedar sujeta a futuras modificaciones, para agregarle los campos y/o cambios necesarios para el registro de otros posibles datos donde el usuario pueda guardar la información que necesite de su historial médico.

Anexos

ENLACE CÓDIGO EN GITHUB:

Pueden encontrarse los detalles del código en el repositorio:

https://github.com/mdani21/Aplicacion-DATAMEDICA

CODIGO DEL CONTROLADOR DE LA APLICACIÓN.

contract.js

```
const forwarderOrigin = 'http://localhost:9010';

//declaracion de todos los botones e inputs de la pagina

const userRegistryButton      = document.getElementById('registry');
const addEventButton          = document.getElementById('addEvent');
const getEventsButton        = document.getElementById('getEvents');
const user_event              = document.getElementById('user_event');
const type_event              = document.getElementById('type_event');
const user_read               = document.getElementById('user_read');
const chunk_read              = document.getElementById('chunk_read');


//Bloque de iniciación para conectar metamask
const initialize = () => {
  //You will start here.
  console.log('initializing');

  //Basic Actions Section
  const onboardButton = document.getElementById('connectButton');

  //creación de la función de verificación creada para ver si la extension metamask está instalada
  const isMetaMaskInstalled = () => {
    //comprueba si el navegador tiene la extensión Metamask instalada
    const { ethereum } = window;
    return Boolean(ethereum && ethereum.isMetaMask);
  };

   //We create a new MetaMask onboarding object to use in our app
   const onboarding = new MetaMaskOnboarding({ forwarderOrigin }); // para hacer uso
del metodo on onClickInstall



  const onClickInstall = () => {
```

```
    // Al clicar en el botón en este estado, redirige al usuario a una página para instalar metamask
    onboardButton.innerText = 'Incorporación en curso';
    onboardButton.disabled = true;
    //On this object we have startOnboarding which will start the onboarding process for our end user
    onboarding.startOnboarding();
  };

  const onClickConnect = async () => {
    try {
      //salta la ventana emergente de Metamask para que se seleccione la cuenta
      await ethereum.request({ method: 'eth_requestAccounts' }); // eth_solicitud de
cuentas
    } catch (error) {
      console.error(error);
    }
  };


  const MetamaskClientCheck = async () => {
    if (!isMetaMaskInstalled()) {
      //si no está instalado se llama al método que redirija al usuario
      onboardButton.innerText = '¡Hacer clic aquí para instalar Metamask!';
      onboardButton.onclick = onClickInstall;
      onboardButton.disabled = false;
    } else {
      //Si ya lo estuviera, se conecta con la cuenta seleccionada
      onboardButton.innerText = 'Conectar con metamask';
      onboardButton.onclick = onClickConnect;
      onboardButton.disabled = false;
    }
  };

  //Método que se encarga de registrar usuario
  userRegistryButton.onclick = async () => {
  try {
    let user = document.getElementById('user_registry').value;

    let user1 = await myContractWithSignature.newHH(user);
    console.log("registro de usuario");
    console.log(user1);
  } catch (error) {
    console.error(error);
  }
  };

  //Método que se encarga de guardar un evento (consulta médica, reporte de laboratorio)
```

```
  addEventButton.onclick = async () => {
    try {
            let user = user_event.value;
            console.log(user);
          //Los tipos de eventos pueden ser (consulta médica, reporte de laboratorio)
            let typeOfEvent = type_event.value;
            console.log(typeOfEvent);
            //Se recoge el hash del fichero subido
            const _urloffile = URL.createObjectURL(fileData);
            console.log(_urloffile);
           //Se sube el evento a Ethereum
            console.log("guardar archivo");
            let file = await
myContractWithSignature.addnewevent(user,typeOfEvent, _urloffile);
            console.log(file);
    } catch (error) {
            console.error(error);
    }
  };

  // Método que recoge los eventos por id de usuarios
  getEventsButton.onclick = async () => {
    try {
            let user = user_read.value;
            let numOfChunk = chunk_read.value;
            //Se recogen los eventos en función a su usuario
            console.log('_userid', user, 'numOfChunk', numOfChunk, 'myContract',
myContract);
            let result = await myContract.getEvents(user, numOfChunk);
            console.log('events -> ', result);
            let cadena = result[0].split("Consulta Médica");
            let cadena2 = result[0].split("Reporte de Laboratorio");
            console.log(result[0]);

           // En JS se puede "inyectar" código HTML en cualquier elemento teniendo únicamente su id
            if (cadena  != "undefined") {
                document.getElementById("contenedor").innerHTML = '<table>'
                + ' <tr><td> Archivo:  </td><td><a target="_blank" href="'+
cadena[1] +'">'+ cadena[1] +'</a></td>'
                + '</table>';
            }
            if (cadena2 != "undefined") {
                document.getElementById("contenedor2").innerHTML = '<table>'
                + ' <tr><td> Archivo:  </td><td><a target="_blank" href="'+
cadena2[1] +'">'+ cadena2[1] +'</a></td>'
                + '</table>';
```

```
            }
    } catch (error) {
            console.error(error);
        }
 };
  console.log('initialized');

  MetamaskClientCheck();

};

window.addEventListener('DOMContentLoaded', initialize); //Lanza la inicialización según cargue toda la página HTML

const provider = new ethers.providers.Web3Provider(window.ethereum);

const signer = provider.getSigner();


//const contractAddress = '0xe53e55eF11ED3673C7633AcbC91DD298cfAB0135'; // dirección del contrato goerli metamask
const contractAddress = '0xeA553aC9fc4a76E7F78db72e6f7Fa68C0Af826C2'; // dirección del contrato ganache metamask

console.log('connecting to contract:', contractAddress);

console.log('provider', provider, 'code', provider.getCode(contractAddress));

// ABI del contrato
const  contractABI = [
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": false,
                "internalType": "string",
                "name": "_userid",
                "type": "string"
            }
        ],
        "name": "thereisanewhistory",
        "type": "event"
    },
    {
        "inputs": [
            {
```

```
            "internalType": "string",
            "name": "_userid",
            "type": "string"
        },
        {
            "internalType": "string",
            "name": "_evento",
            "type": "string"
        },
        {
            "internalType": "string",
            "name": "_urloffile",
            "type": "string"
        }
    ],
    "name": "addnewevent",
    "outputs": [
        {
            "internalType": "bool",
            "name": "OK",
            "type": "bool"
        }
    ],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "string",
            "name": "_userid",
            "type": "string"
        }
    ],
    "name": "newHH",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "string",
            "name": "_userid",
            "type": "string"
        },
```

```
            {
                "internalType": "uint256",
                "name": "numofchunk",
                "type": "uint256"
            }
        ],
        "name": "getEvents",
        "outputs": [
            {
                "internalType": "string[10]",
                "name": "_history",
                "type": "string[10]"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    }
]

const myContract = new ethers.Contract(contractAddress, contractABI, provider);

const myContractWithSignature = myContract.connect(signer);



var fileData = '';
const fileSelector = document.getElementById('file_event');
    fileSelector.addEventListener('change', (evento) => {
    fileData = evento.target.files[0];
    console.log(fileData);
});
```

CÓDIGO DEL CONTRATO INTELIGENTE

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
pragma experimental ABIEncoderV2;

contract Health_registry {
```

```
        event thereisanewhistory(string _userid);

        struct Health {
            string userid;
            string[] history;
        }

        mapping(string => Health) internal hh;

        function newHH(string memory _userid) public {
            string[] memory s;
            Health memory user_aux = Health(_userid, s);
            hh[_userid] = user_aux;
            emit thereisanewhistory(_userid);
        }

        function addnewevent(string memory _userid, string memory _evento, string
memory _urloffile) public returns (bool OK) {
            bytes memory tempEmptyStringTest = bytes(hh[_userid].userid);
            if (tempEmptyStringTest.length != 0) {
                hh[_userid].history.push(string(abi.encodePacked(_evento,
_urloffile)));
                //hh[_userid].history.push(_urloffile);
                return true;
            } else return false;
        }

        function getEvents(string memory _userid, uint numofchunk) view public
returns (string[10] memory _history) {
            for (uint i = 0; i < 10; i++) {
                _history[i] = "";
            }

            if (hh[_userid].history.length > 0) {
                for (uint i = 0; i < 10; i++) {
                    if (((numofchunk - 1) * 10) + i < hh[_userid].history.length)
```

```
                    _history[i] = hh[_userid].history[((numofchunk - 1) * 10)
+ i];
                else
                    _history[i] = "";
            }
            return _history;
        } else {
            return _history;
        }
    }
}
```

Referencias.

[1] Ovalle, D. (2018). Modelado de una organización para la gestión cooperativa de historias médicas. Credencial presentada para optar al título de Magíster Scientiae en Modelado y Simulación de Sistemas.

[2] Dávila, J. (2021). Historias Médicas de las y los Pacientes. Bitácora de Jacinto Dávila.

[3] Hidalgo, R. (2017). Estudio de factibilidad para una red de diagnóstico molecular. Requisito parcial para obtener el título de Ingeniero de Sistemas.

[4] Ortega, U. M. (2021). *Aplicación de un histórico de vehículos sobre blockchain* [Tesis de fin de grado]. Universidad Politécnica de Madrid. https://oa.upm.es/66448/

[5] Diaz-Regañon, L. R. (2022). *Diseño y desarrollo de una aplicación para la compraventa de viviendas a través de la tecnología blockchain* [Trabajo de fin de grado]. Universidad Politécnica de Madrid.

[6] Hidalgo Guerrero, R. A. (2023). *Modelo de un sistema de almacenamiento de datos médicos basados en blockchain* [Título de Magíster]. Universidad de los Andes.

[7] Care2x Team. (2020). http://www.care2x.org/

[8] colaboradores de Wikipedia. (2023c). GNU Health. *Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/GNU_Health

[9] *OpenHIS - Sistema Informático Hospitalario.* (s. f.-b). https://openhis.sourceforge.net/

[10] Alas His. (s. f.). http://www.softwarelibre.gob.ve/index.php/gestion-hospitalaria-hiscon-software-libre

[11] *BLOCKCHAIN PLATFORM FOR DIGITAL HEALTH NETWORKS*. (2022). https://solve.care/

[12] Medicalchain. (2022). https://medicalchain.com/en

[13] *About Our Digital Health and Wellness Company - Sharecare*. (2022, 25 octubre). Sharecare. https://about.sharecare.com/

[14] Hashed Health. (2023, 22 marzo). *Home - Hashed Health*. https://hashedhealth.com/

[15] Medical Record System. (2017). https://www.stateofthedapps.com/dapps/medical-record-system.

[16] Nakamoto, Satoshi (2002). Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/en/

[17] Revista Venezolana de Computación - ReVeCom (ISSN: 2244-7040)-SVC Vol. 5, No. 2, diciembre 2018-Selección de los Mejores Artículos de CONCISa 2018.

[18] Revista OCRONOS ISSN 2603-8358-depósito legal: CA-27-2019. https://revistamedica.com/historia-clinica-origen/#:~:text=La%20evoluci%C3%B3n%20de%20la%20historia%20cl%C3%Adnica%20se%20debi%C3%B3%20a%20cambios,se%20deben%20a%20fen%C3%B3menos%20naturales

[19] welivesecuritybyeset. Blockchain: que es y cómo funciona esta tecnología (2022). https://www.welivesecurity.com/la-es/2022/05/13/blockchain-que-es-como-funciona-y-como-se-esta-usando-en-el-mercado/

[20] Luna, D. (2007). Historia clínica electrónica. Revista del hospital Italiano de Buenos Aires. Vol.27 N.º 2, diciembre 2007.

[21] Digital Guide IONOS. ¿Qué es una interfaz gráfica de usuario (GUI)? https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-una-gui/

[22] CMC Markets. (2023). *¿Qué es Ethereum?* https://www.cmcmarkets.com/es-es/aprenda-a-operar-con-criptomonedas/que-es-ethereum

[23] Phillips, M. H. D. (2022, 9 mayo). ¿Qué es MetaMask? Cómo Utilizar la Mejor —Y Más Popular— Wallet de Ethereum. *Decrypt*. https://decrypt.co/es/resources/que-es-metamask-como-utilizar-la-mejor-y-mas-popular-wallet-de-ethereum

[24] Academy, B. (2023d, mayo 11). *Bit2Me Academy - Formación en Bitcoin y Criptomonedas*. Bit2Me Academy. https://academy.bit2me.com/que-es-metamask-la-forma-mas-facil-de-usar-dapps/v

[25] Phillips, M. H. D. (2022b, mayo 9). ¿Qué es MetaMask? Cómo Utilizar la Mejor —Y Más Popular— Wallet de Ethereum. *Decrypt*. https://decrypt.co/es/resources/que-es-metamask-como-utilizar-la-mejor-y-mas-popular-wallet-de-ethereum

[26] Simões, C. (2021c). JavaScript y Ethereum: ¿Qué es Web3? *Blog ITDO - Agencia de desarrollo Web, APPs y Marketing en Barcelona*. https://www.itdo.com/blog/javascript-y-ethereum-que-es-web3/

[27] Ethereum. (s. f.-b). *Gas y tarifas | ethereum.org*. ethereum.org. https://ethereum.org/es/developers/docs/gas/

[28] *Ganache - Truffle Suite*. (s. f.). https://trufflesuite.com/ganache/

[29] Pérez, M. A. (s. f.-b). REDES CENTRALIZADAS VS. DISTRIBUIDAS. *www.linkedin.com*. https://www.linkedin.com/pulse/redes-centralizadas-vs-distribuidas-miguel-%C3%A1ngel-p%C3%A9rez-garc%C3%ADa?originalSubdomain=es

[30] Academy, B. (2023c, marzo 28). ¿Qué son las DApps? *Bit2Me Academy*. https://academy.bit2me.com/que-son-las-

dapps/#:~:text=Las%20DApps%20o%20aplicaciones%20descentralizadas,nodos%20interactuando%20unos%20con%20otros.

[30] Natalia. (2022, 27 mayo). *¿Qué son los servicios web?* Ceupe. https://www.ceupe.cl/blog/que-son-los-servicios-web-1.html

[31] Fernández, Y. (2019). API: qué es y para qué sirve. *Xataka*. https://www.xataka.com/basics/api-que-sirve

[32] *apuntes:servicios_web [Desarrollo Web en Entorno Servidor]*. (s. f.). https://servidor.codeandcoke.com/apuntes:servicios_web

[33] Academy, B. (2023a, marzo 6). ¿Qué es una red P2P? *Bit2Me Academy*. https://academy.bit2me.com/que-es-una-red-p2p/

[34] Latam, A. (2021). HTML, CSS y Javascript, ¿cuáles son las diferencias? *Alura*. https://www.aluracursos.com/blog/html-css-javascript-cuales-son-las-diferencias

[35] Infante, D. C. H., & Infante, D. C. H. (2023). Qué es Node.js: Casos de uso comunes y cómo instalarlo. *Tutoriales Hostinger*. https://www.hostinger.es/tutoriales/que-es-node-js

[36] *Acerca | Node.js*. (s. f.). Node.js. https://nodejs.org/es/about

[37] *Introducción a Express/Node - Aprende desarrollo web | MDN*. (2023, 9 marzo). https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

[38] A, D., & A, D. (2023b). ¿Qué es npm? Una introducción básica para principiantes. *Tutoriales Hostinger*. https://www.hostinger.es/tutoriales/que-es-npm

[39] Camerfirma. (2023). Smarts contracts o contratos inteligentes. ¿Qué son y cómo funcionan? https://www.camerfirma.com/smarts-contracts-contratos-inteligentes-que-son-como-funcionan/

[40] *Desarrollo con Truffle I.* (2018, 6 abril). APRENDE BLOCKCHAIN. https://aprendeblockchain.wordpress.com/desarrollo-en-ethereum/desarrollo-con-truffle-i/

[41] Matthews, D. (2023). Add Goerli to MetaMask. *www.buybitcoinbank.com*. https://www.buybitcoinbank.com/es/criptomoneda/add-goerli-to-metamask#:~:text=Goerli%20Testnet%20es%20una%20red,un%20entorno%20seguro%20y%20rentable.