UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE CUENCA

CARRERA DE INGENIERÍA DE SISTEMAS

Trabajo de titulación previo
a la obtención del título de
Ingeniero de Sistemas

PROYECTO TÉCNICO:

"ANÁLISIS Y DESARROLLO DE UNA APLICACIÓN DE REGISTRO DE PERMISOS Y AUSENTISMOS SOBRE UNA BLOCKCHAIN MEDIANTE UN SMART CONTRACT DESPLEGADO EN UNA TESTNET DE ETHEREUM"

AUTORES:

JUAN CARLOS CAÑAR UYAGUARI

RICARDO VINICIO JARA JARA

TUTOR:

ING. MAURICIO SERGIO ORTIZ OCHOA, MSc.

CUENCA - CUADOR

2022

CESIÓN DE DERECHOS DE AUTOR

Nosotros, Juan Carlos Cañar Uyaguari con documento de identificación Nº 0106805419 y Ricardo Vinicio Jara Jara con documento de identificación Nº 0105452171, manifestamos nuestra voluntad y cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del trabajo de titulación: **"ANÁLISIS Y DESARROLLO DE UNA APLICACIÓN DE REGISTRO DE PERMISOS Y AUSENTISMOS SOBRE UNA BLOCKCHAIN MEDIANTE UN SMART CONTRACT DESPLEGADO EN UNA TESTNET DE ETHEREUM"**, mismo que ha sido desarrollado para optar por el título de: *Ingeniero de Sistemas*, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autores, nos reservamos los derechos morales de la obra antes citada. En concordancia, suscribimos este documento en el momento que hacemos entrega del trabajo final en formato digital a la biblioteca de la Universidad Politécnica Salesiana.

Cuenca, febrero de 2022.

Juan Carlos Cañar Uyaguari
C.I. 0106805419

Ricardo Vinicio Jara Jara
C.I. 0105452171

CERTIFICACIÓN

Yo, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: **"ANÁLISIS Y DESARROLLO DE UNA APLICACIÓN DE REGISTRO DE PERMISOS Y AUSENTISMOS SOBRE UNA BLOCKCHAIN MEDIANTE UN SMART CONTRACT DESPLEGADO EN UNA TESTNET DE ETHEREUM",** realizado por Juan Carlos Cañar Uyaguari y Ricardo Vinicio Jara Jara, obteniendo el *Proyecto Técnico* que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana.

Cuenca, febrero de 2022.

Ing. Mauricio Sergio Ortiz Ochoa, MSc
C.I. 0103667754

DECLARATORIA DE RESPONSABILIDAD

Nosotros, Juan Carlos Cañar Uyaguari con documento de identificación Nº 0106805419 y Ricardo Vinicio Jara Jara con documento de identificación Nº 0105452171, autores del trabajo de titulación: **"ANÁLISIS Y DESARROLLO DE UNA APLICACIÓN DE REGISTRO DE PERMISOS Y AUSENTISMOS SOBRE UNA BLOCKCHAIN MEDIANTE UN SMART CONTRACT DESPLEGADO EN UNA TESTNET DE ETHEREUM",** certificamos que el total contenido del *Proyecto Técnico*, es de nuestra exclusiva responsabilidad y autoría.

Cuenca, febrero de 2022.

Juan Carlos Cañar Uyaguari
C.I. 0106805419

Ricardo Vinicio Jara Jara
C.I. 0105452171

DEDICATORIA

En primer lugar, dedico este trabajo a Dios y a la Virgen, por permitirme cumplir mis metas y ser mi guía y fortaleza a lo largo de mi vida y también de mi formación profesional.

A mi mamá, Mercedes, le dedico este trabajo y quiero darle las gracias infinitas de todo corazón por haberme brindado su apoyo en cada momento siendo mi guía y ejemplo, ensenándome que soy capaz de alcanzar lo que me proponga y que todos los objetivos y sueños se cumplen sin importar cuán difícil parezcan.

A mi hermano y mi hermana gracias por estar siempre pendientes de mí, por sus consejos y motivación en todo momento y por ofrecerme todo el apoyo del mundo para alcanzar mis metas planteadas.

A mi abuelita y mi tía, quienes siempre han estado pendientes de mí, compartiendo momentos significativos, brindándome su cariño en todo momento, gracias por motivarme a ser mejor cada día y guiar mis pasos para alcanzar mis metas.

A Ricardo, gracias por tu tiempo y dedicación al proyecto y por la excelente amistad que tenemos dentro y fuera de la vida universitaria.

A toda mi familia, amigos y profesores quienes de una u otra forma me brindaron su apoyo y consejos, aportando con su granito de arena, guiándome y permitiendo que día a día alcance mis objetivos, me faltarían líneas para escribir todo, pero ellos saben quiénes son, viviré eternamente agradecido, millón de gracias de todo corazón.

Juan Carlos Cañar Uyaguari.

DEDICATORIA

A Dios y la Virgen María, que siempre serán mi guía espiritual y siempre tendré una fuerza de seguir adelante basada en la fe inculcada por mi familia.

A mi madre, Amparito. que, sin duda, sin su apoyo, no hubiera llegado a cumplir esta meta, gracias a su apoyo moral y económico me han llevado a terminar mis estudios y ser una mejor persona. Gracias, madre querida sin Ud. no le hubiera logrado.

A mi esposa Gabriela y mi hijo Santiago, que en el transcurso de mis estudios aparecieron, para mejorar y ser el centro de la vida. Sin su cariño, amor y su acompañamiento diario no tendría metas definidas por cumplir.

A mis hermanos, en espacial a mi hermana Patricia y mi hermano Rigoberto, les dedico este proyecto de titulación que fueron las personas que incondicionalmente me apoyaron a lo largo de mi formación personal, siempre con una preocupación atenta sobre mis estudios. Eternamente agradecido por tanto apoyo y motivación que sin duda fue necesario para continuar en los momentos difíciles.

A mi compañero de tesis, Juan, que, con su apoyo en el desarrollo de este proyecto, se logró cumplir con los tiempos establecidos, Sin su colaboración tendría mucho trabajo por realizar. Gracias Juan por la dedicación a este proyecto.

Ricardo Vinicio Jara Jara.

AGRADECIMIENTOS

Familiares, amigos y maestros con quienes hemos compartido durante las distintas etapas de nuestras vidas, si alguna vez llegan a leer esto, gracias por compartir grandes momentos, sus anécdotas y enseñanzas, por brindarnos todo el apoyo para llegar a culminar las metas propuestas.

Al grupo de trabajo que formamos durante los años de estudio, gracias por cada momento de aprendizaje y experiencias que hemos vivido a lo largo de nuestra vida universitaria.

A nuestro director de tesis el Ingeniero Mauricio Ortiz, gracias por demostrarnos de lo que podemos ser capaces y también por guiarnos en el trascurso de este proyecto de titulación y enseñarnos a estar a la vanguardia de las nuevas tecnologías.

RESUMEN

La innovación en el desarrollo tecnológico y en el ámbito informático, hoy en día permite que ciertas tecnologías tomen relevancia frente a otras debido a las grandes prestaciones que estas ofrecen, Blockchain es una de ellas, una tecnología innovadora que presenta numerosos beneficios en distintos ámbitos, tales como seguridad, descentralización de la información y transaccionalidad entre distintos entes sin necesidad de intermediarios, dichas características permiten que la información permanezca segura e inmutable.

En base a ello se realiza el planteamiento y posterior desarrollo de este proyecto de titulación con el interés de demostrar las ventajas que presenta dicha tecnología mediante el uso de Smart Contracts desplegados en la Blockchain de Ethereum, además de profundizar sobre le seguridad de la información, al ser almacenada en una Blockchain.

También, analizar y detallar las ventajas de seguridad y protección que ofrecen la tecnología Blockchain frente a la seguridad de los sistemas tradicionales los cuales generalmente son vulnerables.

Lo que se busca es brindar una solución, mediante la seguridad, a la información de datos a través del uso de las propiedades de Ethereum Blockchain, demostrando así mediante el uso de Smart Contracts, de que los datos una vez registrados en la Blockchain se vuelven inmutables.

El objetivo es desarrollar una aplicación que permita llevar un registro de permisos y ausentismos y almacene dicha información mediante el uso de la red Ethereum y su tecnología Blockchain, ya que dichas herramientas permiten la descentralización de la información, de forma que los datos que han sido generados no sean modificados y se mantengan seguros.

Palabras Clave: Blockchain, Ethereum, Smart Contract, Tesnet, Aplicaciones Descentralizadas.

ABSTRACT

Nowadays, in the computer field, the technological innovation allows certain technologies take to relevance over others, due to the great benefits they offer, Blockchain is one of them, an innovative technology that presents numerous benefits in different areas, thus providing security, decentralization of information and transactionality between different entities without the need for intermediaries, these features allow information to remain secure and immutable.

Based on this, the approach and subsequent development of this degree project is carried out with the interest of demonstrating the advantages of this technology using Smart Contracts and the Ethereum Blockchain, in this case focusing on the security of information, in turn taking into account the non-existent protection presented by the files in the operating and storage systems, since these files may contain sensitive information.

What is sought is to provide a solution, through the security of file information, making use of the properties of Ethereum Blockchain, thus demonstrating before the verification using Smart Contracts, that the files once registered in the Blockchain become immutable.

Our objetive is the development of an application that allows tracking of permits and absenteeism, making use of the Ethereum network storage and its Blockchain technology, since from these tools it is possible to find an option that allows betting on the decentralization of information, so that the files or data that have been generated are not modified and remain safe and immutable.

Keywords: Blockchain, Ethereum, Smart Contract, Tesnet, Decentralized Applications

INDICE

Tabla de contenido

DEDICATORIA 5

DEDICATORIA 6

AGRADECIMIENTOS 7

RESUMEN 8

ABSTRACT 10

INTRODUCCIÓN 18

CAPÍTULO I 20

INTRODUCCIÓN Y OBJETIVOS 20

1.1 Antecedentes 21

1.2 Importancia y Alcance 21

1.3 Beneficios 21

1.4 Definición de Objetivos 22

1.4.1 Objetivo General 22

1.4.2 Objetivos Específicos 22

CAPÍTULO II 23

ESTADO DEL ARTE 23

2.1 JavaScript 24

2.2 Babel JavaScript Compiler 24

2.2.1 Características de babel: 25

2.3 ReactJS 25

2.4 Node.js 26

2.5 Blockchain 27

2.5.1 Arquitectura de una Blockchain Ethereum 27

2.5.2 Características de Blockchain: 29

2.5.3 Estructura de un bloque: 29

2.5.4 Estructura de una cuenta 31

2.5.5 Estructura de una transacción 31

2.5.6 Criptografía en Ethereum 32

2.5.6.1 Cifrado y descifrado simétrico 32

2.5.6.2 Cifrado y descifrado asimétrico 32

2.5.7 Hashing ... 33
2.6 Tipos de Blockchain ... 33
2.6.1 Blockchains Publicas: ... 33
2.6.2 Blockchains Privadas ... 34
2.6.3 Blockchains Hibridas ... 34
2.6.4 Blockchains como Servicio (BAAS) ... 34
2.7 Web3Js ... 36
2.8 Truffle ... 36
2.8.1 Características de truffle en Ethereum Blockchain: ... 37
2.8.2 Estructura generada por Truffle Framework. ... 37
2.9 Ganache ... 38
2.10 Ethereum ... 39
2.10.1 EVM Ethereum (Máquina virtual de Ethereum) ... 40
2.10.2 Mecanismos de consenso ... 41
2.10.2.1 PoW(Proof of Work) ... 42
2.10.2.2 PoS(Proof of Stake) ... 43
2.10.3 ERC (Solicitud de comentarios de Ethereum) ... 44
2.11 Versiones de Ethereum ... 44
2.11.1 Ethereum 1.0 ... 44
2.11.2 Ethereum 2.0 ... 45
2.12 Crypto Wallet ... 46
2.13 Tesnet ... 47
2.13.1 Rinkeby Ethereum Tesnet. ... 47
2.13.2 Ethereum Faucet ... 48
2.13.3 Infura Ethereum API ... 48
2.13.4 Etherscan ... 49
2.14 Gas Ethereum ... 49
2.15 Aplicaciones Descentralizadas (DApp) ... 51
2.16 Metamask ... 52
2.17 Solidity ... 52
2.18 Smart Contracts ... 53
2.18.1 ABI (interfaz binaria de la aplicación): ... 54
2.18.2 Bytecode (código de bytes): ... 54
2.18.3 Funcionamiento de un SmartContract en una Blockchain: ... 55

2.19 Registros Distribuidos (DLT) ... 56
2.20 Oráculos en Ethereum ... 56
CAPÍTULO III ... 58
ANÁLISIS, DISEÑO E IMPLEMENTACIÓN ... 58
3.1 Manejo de herramientas ... 59
3.2 Análisis de Requerimientos de Software ... 59
3.3 Características Principales de la Aplicación ... 60
3.4 Levantamiento de Requerimientos ... 60
3.5 Definición de requerimientos de usuario ... 60
3.6 Definición de requerimientos del sistema ... 61
3.7 Definición de requerimientos funcionales ... 61
3.8 Definición de requerimientos no funcionales ... 61
3.9 Diagrama de la aplicación. ... 62
3.10 Diagrama de clases ... 64
3.11 Desarrollo de la aplicación Blockchain de registro de permisos y ausentismos. ... 65
3.12 Diagrama de secuencia del sistema. ... 65
3.13 Implementación del Sistema de registro de permiso y ausentismos ... 68
3.13.1 Instalación y configuración de las herramientas de Ethereum. ... 68
3.13.2 Instalación de Node.js ... 68
3.13.3 Instalación de herramientas de compilación de C/C++ en npm. ... 69
3.13.4 Instalación del compilador de Solidity. ... 69
3.13.5 Configuración del compilador de Solidity. ... 69
3.13.6 Instalación de Web3 Js. ... 70
3.13.7 Instalación del servidor Ganache ... 70
3.13.8 Instalación de Truffle Framework. ... 71
3.13.8.1 Instalación de Truffle Contract. ... 71
3.13.9 Instalación y configuración de Metamask Crypto Wallet. ... 71
3.13.10 Instalación de Metamask Detect Provider. ... 76
3.13.11 Desarrollo de Smart Contract ... 76
3.13.12 Diseño de la Aplicación Web ... 81
3.13.12.1 Frontend ... 81
3.13.12.2 Backend ... 85
3.13.13 Despliegue en la red Testnet Rinkeby. ... 85
3.13.14 Validación y Pruebas ... 86

CAPÍTULO IV 88
ANÁLISIS DE RESULTADOS 88
4.1 Conexión y selección de la cuenta de Metamask 89
4.2 Registro de usuarios con Metamask. 90
4.3 Creación del permiso 92
4.4 Reporte de permisos y ausentismos 94
4.5 Perfil de administrador 94
4.6 Acceso al reporte para aprobar o negar permisos y ausentismos 95
4.7 Función de registro desde perfil administrador 96
4.8 Transaccionalidad del Smart Contract. 98
CAPÍTULO V 101
CONCLUSIONES Y RECOMENDACIONES 101
Conclusiones 102
Recomendaciones 103
Bibliografía 104
Anexos 107

INDICE DE ILUSTRACIONES

Fig 1. Estructura de una Blockchain Ethereum. Fuente: (Autores) 28
Fig 2. Estructura de un bloque en una Blockchain, Fuente: (Autores) 30
Fig 3. Estructura de truffle durante el desarrollo de un SmartContract 37
Fig 4. Logotipo de Ganache. (Trufflesuite, 2021) 38
Fig 5. Logotipo de Ethereum. (ethereum.org, 2021) 39
Fig 6. Costos relativos en gas de las operaciones que una transacción puede efectuar. (Wood, 2021) 50
Fig 7. Estructura de un Smart Contract. Fuente (Autores) 53
Fig 8. Esquema del sistema. Fuente (Autores) 62
Fig 9. Diagrama de clases de la aplicación 64
Fig10. Diagrama de secuencia de usuario 66
Fig 11. Diagrama de secuencia de administrador. 67
Fig 12. Configuración de la versión del compilador. Fuente (Autores) 70
Fig 13. Interfaz de inicio de Metamask. Fuente (Autores) 72
Fig 14. Interfaz de configuración de Metamask. Fuente (Autores) 73
Fig 15. Interfaz para agregar y configurar una nueva red en Metamask. Fuente (Autores) .. 74
Fig 16. Interfaz de Ganache, con una clave privada para importar a Metamask. Fuente (Autores) 74
Fig 17. Interfaz con dos cuentas importadas desde Ganache. Fuente (Autores) 75
Fig 18. Dirección de la Billetera. Fuente (Autores) 76
Fig 19. Interfaz para obtener Ether e interactuar con una red de pruebas. Fuente (Autores) 76
Fig 20. Interfaces que se muestran al no tener instalado la billetera Metamask en el navegador. Fuente (Autores) 82
Fig 21. Interfaz con el perfil de usuario. Fuente (Autores) 83
Fig 22. Interfaz de administrador. Fuente (Autores) 84
Fig 23. Test de validación de funciones del Smart Contract. Fuente (Autores) 87
Fig 24. Selección de la cuenta en Metamask. Fuente (Autores) 89
Fig 25. Aviso al usuario cuando no se tiene instalado Metamask. Fuente (Autores) 90
Fig 26. Ventana de registro de usuarios. Fuente (Autores) 91
Fig 27. Ventana de usuario registrado. Fuente (Autores) 92
Fig 28. Ventana para generar permisos. Fuente (Autores) 92
Fig 29. Ventana de Metamask, para confirmar la transacción. Fuente (Autores) 93
Fig 30. Ventana con el reporte de permisos y ausentismos con el perfil usuario. Fuente (Autores) 94

Fig 31. Ventana con el perfil de administrador. Fuente (Autores) 95
Fig 32. Ventana con el registro de permisos y ausentismos desde el perfil administrador. Fuente (Autores) 96
Fig 33. Ventana con las opciones de administrador. Fuente (Autores) 97
Fig 34. Ventana de acceso a rinkeby. Fuente (Autores)..... 98
Fig 35. Dirección de la cuenta almacenada en Metamask. Fuente (Autores) 98
Fig 36. Direccion del Smart Contract desde Etherscan. Fuente (Autores) 99
Fig 37. Dirección del Smart Contract desde truffle. Fuente (Autores) 99
Fig 38. Transacciones asociadas al Smart Contract. Fuente (Autores)..... 100
Fig 39. Eventos registrados mediante el Smart Contract. Fuente (Autores)..... 100

INDICE DE TABLAS

Tabla 1. Blockchain más usadas en la actualidad y sus servicios. (Yousaf, 2018) 35
Tabla 2. Unidades de gas por operación en Ethereum. (Ethereum, 2021) 51

INTRODUCCIÓN

En la actualidad el avance tecnológico y su impacto se ha notado en distintas industrias, se ha visto una evolución extraordinaria dentro de ciertas tecnologías, mismas que ofrecen grandes mejoras, una de ellas es Blockchain un término que se está continuamente escuchando en el área informática, puesto a que ofrece grandes prestaciones, tales como seguridad y descentralización de la información.

La implementación de tecnologías de registro distribuido (DLT), son una gran opción, en pequeñas y grandes empresas e instituciones financieras, debido a que la información registrada es delicada y al ser almacenada en Blockchain se mantiene segura y es gestionada por varios actores de manera descentralizada, donde ningún actor posee ni tiene más poder sobre el registro, convirtiéndose así en inmutable.

La DLT, en particular Blockchain, posee una construcción muy concreta, la información se almacena en bloques, un algoritmo genera un hash de consolidación que se une de manera que cada bloque tiene su propio hash y el hash que queda ligado al anterior formando una cadena.

El potencial de las DLT y Blockchain es tal, que no es simplemente una tecnología para gestionar activos monetarios, sino que cada vez se amplía más en distintas áreas, debido a que existen empresas tecnológicas que están detrás este desarrollo, tales como IBM, que es líder en tecnología Blockchain, también Alibaba Group y detrás de estos esta MasterCard e ING Groep.

Los Smart Contracts, son scripts, cuyo objetivo principal es asegurarse, hacer que se cumpla y ejecutar acuerdos entre dos o más partes, son autoejecutables es decir asegura que los acuerdos se lleguen a cumplir para que sucedan ciertas acciones.

Por ende, este trabajo tiene como objetivo, analizar y desarrollar una aplicación de permisos y ausentismos sobre una red Blockchain, haciendo uso de Ethereum y de su red testnet (red de pruebas) para el despliegue y la implementación de los Smart Contracts mediante los cuales se validara que la información se almacene en la Blochchain y una vez registrada no se altere permaneciendo segura e inmutable para posterior presentar un historial con todos los registros que se han ido suscitando con el paso del tiempo, destacando el potencial que tienen las DLT en lo referente a seguridad de la información, sin necesidad de un ente regulador.

CAPÍTULO I

INTRODUCCIÓN Y OBJETIVOS

1.1 Antecedentes

Hoy en día no existe protección o control sobre ficheros o registros en los sistemas operativos, se puede modificar con facilidad y no se presenta un control sobre dichos cambios a excepción de ciertas aplicaciones que ofrecen un historial de los cambios pero estos son pocos, dichos cambios en los ficheros pueden ocasionar grandes pérdidas de datos, debido a que pueden contener información sensible y de suma importancia, el objetivo es implementar tecnologías de registro distribuido, donde la información una vez compartida en una Blockchain, se mantenga inmutable.

1.2 Importancia y Alcance

Para que un manejo de ficheros se mantenga seguro y la información que estos contienen sea inmutable, se debe llevar a cabo la implementación de un sistema que nos ayude a desarrollar dichas características, para esto se hace uso de tecnologías de registro distribuido.

Gracias a la innovación tecnológica que ha dado paso a la creación de tecnologías de registro descentralizado y distribuido como Blockchain y el desarrollo de DApps que hacen uso de los SmartContracts, se puede llevar a cabo el desarrollo de un sistema que registre y posterior valide que los registros de permisos y ausentismos no sean alterados, de igual forma dicha tecnología nos ayudan a descentralizar la información puesto que se almacena y se valida en una Blockchain mediante un SmartContract, permitiendo de esta forma mantenerle segura.

1.3 Beneficios

Uno de los mayores beneficios es brindar mayor seguridad al sistema (DApp), así como de igual forma al usuario y al administrador, haciendo uso de registros distribuidos mediante el uso de la Blockchain de Ethereum y el despliegue de Smart Contracts.

Fomentando así el uso de tecnologías de registro distribuidas para próximos proyectos donde quizá se requiera una verificación de extremo a extremo de estos ya que ayudan a mantener su integridad, proponiendo el uso de Blockchain para evitar su modificación.

1.4 Definición de Objetivos

1.4.1 Objetivo General

Desarrollo de una aplicación para registro de presencias y ausentismos, mediante una Blockchain, haciendo uso de Smart Contracts, desplegado en una tesnet de Ethereum.

1.4.2 Objetivos Específicos

- OE1. Análisis, estudio y aprendizaje de un registro distribuido (DLT), así como de Ethereum, sus redes y desarrollo de DApps.
- OE2. Investigación y aprendizaje del lenguaje Solidity.
- OE3. Análisis y diseño de la aplicación para registro de permiso y ausentismos.
- OE4. Desarrollo de los Smart Contracts que requiere dicha aplicación (DApp).
- OE5. Investigación y pruebas en servidores que soporten el despliegue de Smart Contracts en la red Blockchain.
- OE6. Despliegue de los Smart Contract en la red Blockchain.
- OE7. Implementación de los Smart Contracts y la interfaz web en el servidor escogido.

CAPÍTULO II

ESTADO DEL ARTE

2.1 JavaScript

El desarrollo del proyecto hace uso de este popular lenguaje de programación, que se define como un lenguaje orientado a objetos, multiparadigma creado por Netscape hoy en día Fundación Mozilla y desarrollado por Brendan Eich. Su última versión estable es ECMAScript 2016, lleva este nombre ya que JavaScript es el dialecto del estándar ECMAScript, un lenguaje muy importante y usado en el desarrollo web frontend para realizar validaciones o la construcción de webs dinámicas, convirtiéndose así en un lenguaje de suma importancia para el desarrollo de aplicaciones web y móviles mediante entornos de ejecución tales como Nodejs, además posee compatibilidad con Sistemas Operativos Windows, Linux y Mac. (Haverbeke, 2018)

Algunos Frameworks como React, Angular, Ionic, entre otros se basan en JavaScript, haciendo de este un lenguaje sumamente importante el momento de desarrollar aplicaciones multi plataforma.

A JavaScript se le designa como un lenguaje de programación interpretado ya que para ser ejecutados los scripts o programas desarrollados no es necesario compilar, por lo que dicha característica permite ejecutar los scripts directamente en un navegador sin necesidad de procesos intermedios. (Perez,2009)

2.2 Babel JavaScript Compiler

Babel es una herramienta que ayuda a convertir el código ECMAScript ES6 en una versión de JavaScript compatible con versiones anteriores en navegadores o entornos actuales y antiguos, lo que le convierte a JavaScript en una herramienta óptima para el desarrollo, dicha herramienta logra convertir la sintaxis de JavaScript moderno en un formulario que los navegadores antiguos puedan entender fácilmente, mediante Babel-Transpiler. (Babel, 2021)

2.2.1 Características de babel:

- Transformar la sintaxis
- Funciones de Polyfill que faltan en su entorno de destino (a través de un polyfill de terceros como core-js)
- Transformaciones de código fuente (codemods)
- Soporte para la última versión de JavaScript
- Funciona con la sintaxis JSX y puede conpilar en formato JSX.
- Soporta complementos como gulp, webpack, Flow, react,etc.

Contiene un conjunto de plugins que ayudan a transformar funciones de ECMAScript desde la versión ES6 y react-JSX, en funciones o código de JavaScript, esta herramienta puede ser instalado haciendo uso del gestor de paquetes npm para aplicaciones basados en Nodejs tanto en entornos Windows y Linux.

2.3 ReactJS

React JS es una biblioteca de JavaScript utilizada en el desarrollo web para crear interfaces de usuario o UI, combina herramientas de desarrollo como HTML, JavaScript y XML, es desarrollado como una biblioteca de código abierto, basada en componentes, los cuales poseen lógica y controladores, permitiendo una creación y un desarrollo rápido, es mantenido por Facebook y la comunidad de software libre.

Facebook ha lanzado una herramienta para depuración, se trata de una extensión de Chrome que se puede utilizar para depurar aplicaciones que usen React, permitiendo que la depuración de las aplicaciones web de React sea más rápido y sencillo.

ReactDOM facilita la creación de aplicaciones dinámicas ya que requiere menos líneas de código y ofrece mayor funcionalidad, haciendo uso de Virtual DOM, lo que permite la creación de aplicaciones más ligeras y eficientes.

2.4 Node.js

Es un entorno de ejecución en tiempo real, fue creado por Ryan Dahl para hacer compatible JavaScript fuera del navegador, todo esto es posible mediante el motor V8 de código abierto creado por la empresa Google, que fue incorporado al núcleo de Node.js, permitiendo así la creación de aplicaciones escalables y de alto rendimiento, trabaja con operaciones asíncronas y orientado a eventos, lo que permite que su ejecución sea bastante rápida, es uno de los entornos más usados para programas que usan como base JavaScript, para poder desarrollar se necesitan ciertas dependencias, estas pueden ser gestionadas mediante el uso del gestor de paquetes npm. (Node.js, 2020)

El procesamiento de Nodejs, hace uso de menos recursos que otras plataformas ya que se ejecuta en un solo proceso y el código de la aplicación se ejecuta en un solo hilo, las solicitudes de los usuarios a su aplicación web serán manejadas por un hilo y no es necesario que complete la otra solicitud para manejar la siguiente, el trabajo de larga ejecución se realiza de forma asíncrona, los eventos generados ingresan a un bucle de eventos que es gestionado por libev que a su vez hace uso del grupo de subprocesos internos de C++ para proporcionar E/S asíncronas, permitiendo que el servidor responda sin bloqueos y sea altamente escalable en comparación con los servidores tradicionales que crean subprocesos limitados para manejar solicitudes. (Node.js, 2020)

Las Api de la biblioteca Nodejs sin asíncronas, permitiendo que Nodejs use un mecanismo de notificación de Eventos de Nodejs que ayuda al servidor a obtener una respuesta de la llamada API anterior.

2.5 Blockchain

Blockchain es una cadena de bloques o estructura de datos descentralizada, es decir la información se replica en varios nodos de forma P2P, dicha tecnología también hace uso de criptografía para proteger la información guardada en la cadena, permitiendo que dicha información no pueda ser modificada o borrada y cualquier intento de alterar la cadena crearía una inconsistencia fácilmente detectable de todas las réplicas, logrando así de esta forma brindar mayor seguridad y privacidad, además de evitar que el servidor tome control absoluto de los datos y la ejecución del servicio o la aplicación implementada sobre la red, consiguiendo así que la información nunca deje de estar disponible si algún servidor deja de funcionar.

Para ser miembro de una red Blockchain y poder ser partícipe de información o validar transacciones, es necesario tener un hash(identidad) en conjunto con una billetera digital(wallet) ya que por medio de esta se puede realizar transacciones. (K. Christidis y M. Devetsikiotis, 2016)

2.5.1 Arquitectura de una Blockchain Ethereum

La infraestructura de Blockchain está diseñada de tal forma que facilite un flujo de trabajo sólido, brindando mayor seguridad, dicho diseño consta de varias capas, tales como la capa de **infraestructura** en la cual se engloba todo lo referente al hardware que hace uso de **redes** para la conexión, intercambio, propagación y verificación de la información donde también se hace uso de algoritmos de **consenso** tales como prueba de trabajo y prueba de participación que son mecanismos usados por esta red Blockchain

para seleccionar el estado correcto de un registro después de realizar una transacción y también la capa de **datos** que es muy importante ya que es la información almacenada en bloques que se originan mediante transacciones que de cierta forma dan origen a la capa de **aplicación**, la cual llega a conformarse por aplicaciones descentralizadas y contratos inteligentes, logrando así definir e identificar los componentes centrales de la arquitectura Blockchain, descritos a continuación: (Chen, Pendleton, Njilla, & Xu, 2019)

Fig 1. Estructura de una Blockchain Ethereum. Fuente: (Autores)

- **Nodo:** Usuario que forma parte de la arquitectura de la cadena de bloques, el cual posee una copia independiente de todo el libro mayor de la cadena de bloques.
- **Transacción:** Es la acción llevada a cabo en el bloque más pequeño de un sistema Blockchain, donde se ejecutan registros e información.
- **Bloque:** Estructura de datos utilizada donde se almacena un conjunto de transacciones que se distribuye a todos los nodos de la red.

- **Cadena:** Secuencia de los bloques que llevan un orden especifico.
- **Mineros:** Nodos específicos que llevan a cabo un proceso de verificación de bloques mediante un algoritmo de consenso, antes de agregar algo a la estructura de la cadena de bloques.
- **Consenso:** Hace referencia al conjunto de reglas y acuerdos que se lleva a cabo para realizar operaciones de Blockchain, mediante algoritmos, tales como PoW o PoS.

2.5.2 Características de Blockchain:

- Distribuido.
- Valida la información mediante un sistema de consenso.
- Global.
- A base de criptografía.
- Transparente.
- Inmutable y sin intermediarios.
- **Casos de uso de una Blockchain:**
  - Transacciones bancarias.
  - Seguridad en los ficheros o datos de salud, seguros, educación, elecciones.
  - Validación y acreditación de propiedad intelectual.
  - En la industria musical para control, seguridad y compensación justa de regalías mediante Smart Contracts a través el proyecto (UjoMusic).

2.5.3 Estructura de un bloque:

Tiene una estructura de datos conformada por bloques donde cada bloque contiene un conjunto de transacciones y tiene un enlace a un bloque anterior conformado por un hash, el cual permite que se vaya sumando a una cadena, ya que el bloque contiene

el hash del bloque antecesor y el hash del propio bloque y también otras propiedades adicionales, las cuales se describen a continuación:

- Hash bloque anterior: Mediante el cual se crea la cadena, este hash se crea de acuerdo con la información guardada en el bloque.
- Timestamp: Campo de suma importancia, debido a que lleva un registro del instante exacto de la creación del bloque.
- Nonce: Valor que se encuentra durante el minado aplicando fuerza bruta.
- Hash: Identificador hash único de cada bloque permite referenciar el bloque y su información.

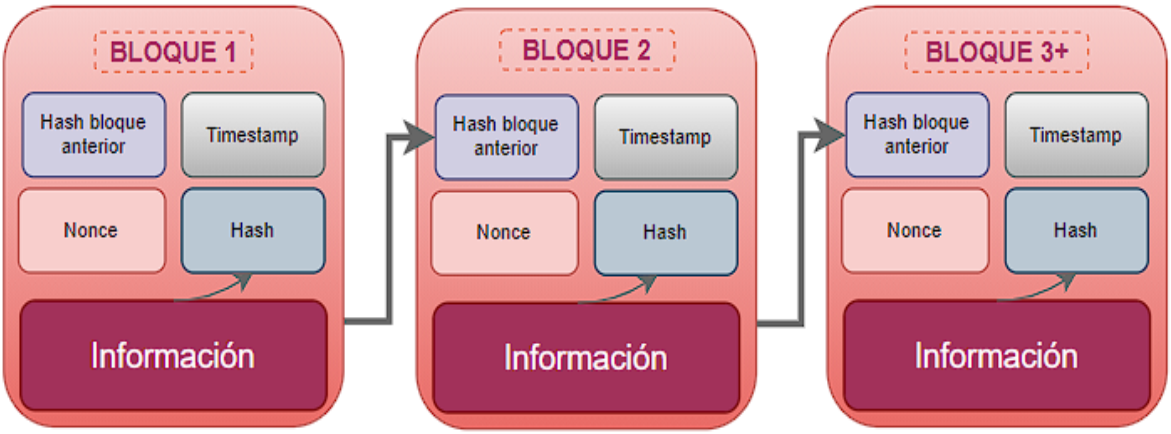


Fig 2. Estructura de un bloque en una Blockchain, Fuente: (Autores)

Una de las grandes ventajas que tiene Blockchain es que posee un alto nivel de seguridad frente a hackeos, para poder llegar a falsificar información se tendría que lograr estar de acuerdo en todos los nodos de la red al mismo tiempo, es decir en más de la mitad de las maquinas que forman parte de la red.

Además, si la red llegara a caerse, basta con que un nodo no lo haga para que la red siga brindando servicio, Blockchain maneja cuentas para poder realizar transacciones, a continuación, se describe su estructura. (Wood, 2021)

2.5.4 Estructura de una cuenta

La cuenta de la Blockchain de Ethereum se conforma por los siguientes campos:

- **nonce:** Es igual al número de transacciones enviadas desde esta dirección o también al número de creación de contratos realizada por esta cuenta.
- **balance:** Valor numérico equivalente a la cantidad de Wei que posee la dirección.
- **storageRoot:** Hash de 256 bits del nodo raíz que codifica el contenido de almacenamiento de la cuenta, consiste en un mapeo del hash Keccak de 256 bits de las claves enteras de 256 bits a los valores enteros de 256 bits.
- **codeHash:** Código inmutable que se ejecuta si la dirección recibe una llamada de mensaje, a diferencia de todos los demás campos, no se puede cambiar después de la construcción.

2.5.5 Estructura de una transacción

Existen dos tipos de transacciones aquellos que resultan en llamadas y aquellas que resultan en la creación de nuevas cuentas con código asociado denominadas como creación de contrato, los dos están conformados por la siguiente estructura. (Wood, 2021)

- **nonce:** Valor equivalente al número de transacciones enviadas por el remitente.
- **gasPrice:** Cantidad igual al número de Wei a pagar por unidad de gas para los costos de cálculos computacionales como resultado de ejecución de una transacción.
- **gasLimit:** Cantidad de gas que se paga por adelantado para ejecutar la transacción dentro el bloque, antes de que se realice cualquier cálculo.
- **to:** Campo con la dirección del destinatario, formada por 160 bits.

- **value:** Valor igual al número de Wei que se transferirá al destinatario de la llamada del mensaje.
- **v,r,s:** Valores correspondientes a la firma de la transacción y utilizados para determinar el remitente de la transacción, estos componentes son conocidos también como Tr,Ts y Tw.

2.5.6 Criptografía en Ethereum

La criptografía es una técnica que ayuda a cifrar datos o archivos con el fin de que permanezcan seguros al momento de transmitir o almacenar y que puedan ser leídos por aquellas entidades que estén autorizadas, para realizar se hace uso de dos tipos de criptografía **simétrico** y **asimétrico**.

2.5.6.1 Cifrado y descifrado simétrico

Este proceso utiliza una clave para el cifrado y descifrado, dicha clave debe ser conocida previamente por las partes involucradas, es decir la misma clave debe estar disponible para varias personas si desean intercambiar información utilizando esta forma de criptografía. (Modi R. , 2018)

2.5.6.2 Cifrado y descifrado asimétrico

Dicho proceso hace uso de dos clases para el cifrado y descifrado, se utiliza cualquier clave para el cifrado y el descifrado, en resumen, los archivos o datos cifrados con una clave pública se puede descifrar utilizando una clave privada y los mensajes cifrados con una clave privada se pueden descifrar utilizando una clave pública. (Modi R. , 2018)

2.5.7 Hashing

Procedimiento para generar datos de entrada en una salida de caracteres de extensión fija aleatorios, es casi imposible descifrar datos de entrada en función de su valor hash, garantizando así que si llega a cambiarse los datos de entrada se cambiara por completo los datos de salida, sin importar el tamaño de los datos de la cadena de entrada, la longitud de su salida siempre es fija.

Ethereum hace uso de la técnica de hash de manera bastante extensa, primero realiza el hash de cada transacción, el hash de dos transacciones a la vez y en última instancia genera un único hash para la transición raíz para cada transacción dentro de un bloque. (National Institute of Standards and Technology, 2015)

Ethereum hace uso de la función hash **Keccak-256,** como su algoritmo de hash, la cual es un motor de consenso denominado **Ethash,** ya que es un superconjunto del estándar SHA-3, Ethereum le llamo Keccak-256 en lugar de SHA-3, ya que posee parámetros ligeramente diferentes a los SHA-3 (la próxima iteración de Secure Hash Algorithms).

2.6 Tipos de Blockchain

2.6.1 Blockchains Publicas:

Son de tipo abierto en la que cualquiera puede participar, los participantes pueden libremente acceder a los datos y realizar transacciones, pero debido a que participan usuarios no verificados es necesario encriptación y verificación avanzada, por lo que la expansión de la red se torna difícil y muy lenta.

Los participantes de la red son pseudoanónimos por lo que le vuelve a la Blockchain publica inapropiada para los servicios financieros que necesitan ser controlados por la información centralizada de los sistemas de gestión. (Oh & Shong, 2017)

2.6.2 Blockchains Privadas

En este tipo de Blockchains, el dueño es quien crea y administra, además de que un grupo selecto de nodos son quienes validan y distribuyen los libros contables, para ser parte del sistema los nodos requieren de iniciación y validación, solo los nodos acreditados son responsables de mantener el consenso. Estas Blockchains, son implementadas en sistemas cerrados, puesto que se considera que los nodos miembros de la red son confiables y el que permite o no el acceso a nodos autorizados es el propietario, considerado la máxima autoridad. (Viriyasitavat & Hoonsopon, 2018)

2.6.3 Blockchains Hibridas

Estas se encuentran formadas por una combinación de Blockchains públicas y privadas, los nodos participantes son invitados, pero todas las transacciones son de carácter público, además mantienen una arquitectura distribuida y segura al tener participación limitada, resolviendo así el problema de velocidad lenta en las transacciones.

Son adecuadas para sistemas semicerrados compuestos por pocas empresas, organizados en forma de consorcio, contienen controles de acceso, para controlar el acceso en ambos participantes y la información dentro de Blockchains. (Viriyasitavat & Hoonsopon, 2018)

2.6.4 Blockchains como Servicio (BAAS)

Ciertas compañías ofrecen hoy en día servicios de Blockchain hospedados en la nube, donde se puede construir y desplegar sus funciones y contratos inteligentes, funciona de forma que se debe pagar una tarifa a la Blockchain como proveedor de servicios para implementar y mantener tus nodos. (Viriyasitavat & Hoonsopon, 2018)

Además, los proveedores de Blockchain como servicio, administran todo lo referente a alojamiento, protocolos de seguridad, distribución adecuada de recursos, gestión de ancho de banda y mucho más.

Tabla con diferentes tipos de Blockchain y sus servicios.

	Acceso	Participantes	Seguridad	Velocidad transacción	Empresas
Blockchain Publica	Todos	Sin permisos Anónimo	Mecanismos de consenso descentralizados Prueba de trabajo Prueba de participación	Lenta	Bitcoin, Ethereum
Blockchain Privada	Una sola organización	Con permisos Identidades conocidas	Algoritmos de consenso propios Transacciones verificadas dentro de una organización.	Mas ágil y rápida	Hyperledger, Corda, Quorum
Blockchain Hibrida	Varias organizaciones seleccionadas	Con permisos Identidades conocidas	Algoritmos de consenso propios Varios nodos preseleccionados dentro de las organizaciones (parcialmente descentralizado)	Mas ágil y rápida	Hyperledger, Corda, Quorum
Blockchain como servicio	Organizaciones grandes	Con permisos Identidades conocidas	Algoritmos de consenso Seguridad robusta.	Ágil y rápida	IBM

Tabla 1. Blockchain más usadas en la actualidad y sus servicios. (Yousaf, 2018)

2.7 Web3Js

Web3js, es una API diseñada y compatible con JavaScript, para hacer uso en un proyecto se puede gestionar mediante npm, contiene una gran colección de bibliotecas, las cuales nos permite conectarnos a un nodo Ethereum de forma local o a través de protocolos WebSocket, HTTP o IPC de forma remota. (Ethereum, web3js, 2021)

Dispone de algunos módulos que facilita usar el ecosistema de Ethereum, tales como:

- Web3-eth se usa para la cadena de bloques Ethereum y los contratos inteligentes.
- Web3-shh usado para comunicación P2P y broadcast.
- Web3-bzz sirve para el almacenamiento de archivos descentralizados.
- Web3-utils dispone de varias funciones para los desarrolladores de Dapp.

2.8 Truffle

Se presenta como un entorno con un conjunto de herramientas para ser implementadas en el desarrollo de Smart Contracts haciendo uso de la máquina virtual de Ethereum, escrito en JavaScript, siendo un framework muy útil y compatible con la implementación, compilación y prueba de SmartContracts.

Truffle es compatible con entornos Windows y Linux, permitiendo que sea uno de los frameworks más usados para el desarrollo de aplicaciones que hacen uso de Blockchains como Ethereum.

Está diseñado para ayudar a construir aplicaciones distribuidas (DApps), de manera fluida y sin problemas ya que permite la inserción de Smart Contracts en aplicaciones web y testear conforme se vaya desarrollando, además de ayudar en el desarrollo de la parte de front-end para DApps, en conjunto con otras herramientas. (Hiremath, 2019)

2.8.1 Características de truffle en Ethereum Blockchain:

- Soporte para la vinculación de contratos inteligentes.
- Pruebas de contrato automatizadas mediante Mocha.
- Soporta JavaScript, ES6, CoffeeScript y JSX.
- Posee consola integrada para contratos compilados.
- Cuenta con generadores que ayudan en la creación de contratos.
- Permite la ejecución de archivos JavaScript
- Compilación de contratos mediante el cliente (Llamada a procedimiento remoto) RPC.
- Gestión mediante red y gestión de paquetes.

2.8.2 Estructura generada por Truffle Framework.

Fig 3. Estructura de truffle durante el desarrollo de un SmartContract

- **Consta de tres partes:**

  - **Contracts:** Dicho directorio contiene el código del contrato desarrollado en Solidity, se crea un archivo con el nombre seguido de la extensión **".sol"**, aquí se guardan otros contratos que se desarrollan, según se requiera. (CSDN, 2018)
  - **Migrations:** Este directorio contiene scripts de implementación de contratos de los cuales **1_initial_migration.js** se usa para implementar "**Migrations.sol"** y otros scripts se ejecutarán conforme sean requeridos. (CSDN, 2018)
  - **Test:** En este directorio es donde se programan los diferentes scripts para poder realizar pruebas haciendo uso de web3, antes de ser desplegados en la red principal de Ethereum Blockchain. (CSDN, 2018)

2.9 Ganache

Fig 4. Logotipo de Ganache. (Trufflesuite, 2021)

Ganache Ethereum es parte de un conjunto de herramientas de desarrollo de Truffle Suite, permite recrear entornos de Blockchain localmente y probar contratos inteligentes y DApps antes de ser desplegados en la red principal de Ethereum, además esta herramienta presenta la información como el estado actual de todas las cuentas, incluidas sus direcciones, claves privadas, transacciones y saldos, también lleva un registro de Blockchain, es decir un registro

de la cadena de bloques interna de Ganache, donde tiene integrado las respuestas e información vital de depuración y transacciones, además de tener integrado un explorador de bloques que permite obtener información de todos los bloques y transacciones. (Trufflesuite, 2021)

Dicho software posee dos versiones una con interfaz de usuario a través de la cual se puede gestionar la cantidad de cuentas que se demanda durante las pruebas en el desarrollo además de poder revisar los logs y el balance o Ether que se tiene para cada transacción y una también una versión CLI (línea de comandos) que trae consigo ciertas funciones para poder ser usadas en la consola, esta herramienta se encuentra disponible en entornos Windows, Linux y Mac.

2.10 Ethereum

Fig 5. Logotipo de Ethereum. (ethereum.org, 2021)

Ethereum es una plataforma Blockchain descentralizada, lanzada oficialmente en 2015, con una estructura de desarrollo basado en la colaboración abierta, posee tecnología que permite él envió de criptomonedas y también la creación de contratos inteligentes y DApps, dicha tecnología fue desarrollada por el escritor y programador ruso Vitalik Buterin y dos compañeros, dicha plataforma fue la primera en introducir contratos inteligentes a una comunidad de criptografía, tecnología que se encuentra desarrollada y escrita en lenguajes de

programación tales como C, C++, Go, JavaScript, Java y Python, contando hoy en día con una de las redes de desarrolladores más grandes, logrando estar al día con el cambio en la industria Blockchain.

Como propósito principal tiene desarrollar un proyecto para descentralizar la web, mediante el uso de mensajes dinámicos, publicación de contenido estático, transacciones confiables y una interfaz de usuario integrada y funcional, además de la creación de una plataforma que permita y facilite a otros programadores el desarrollo y despliegue de aplicaciones descentralizadas haciendo uso de Smart Contracts.

También dispone de su propia criptodivisa llamada Ether (ETH), a través de la cual se realiza pagos o funciona también como incentivo para asegurar que las aplicaciones sean de calidad o recompensar a los mineros por operaciones realizadas en la red, utiliza un sistema de consenso denominado "Prueba de trabajo" o "Proof of Work" (PoW), permitiendo ponerse de acuerdo sobre el estado de toda la información registrada en la cadena de bloques de Ethereum, en ETH 2.0 el sistema de consenso cambia a un sistema llamado "Prueba de participación" o "Proof of Stake" que se basa en un sorteo, en dicho mecanismo no existe recompensa por minar el bloque, sino que se obtiene una comisión por transacción. (ethereum.org, 2021)

El costo por transacción en la versión más actual de Ethereum la generación 2.0, llega a un promedio de $15 a la cotización actual, soportando entre 15 y 40 transacciones por segundo, en dicha versión se actualizará la cadena de fragmentos y se distribuirá la carga de la red en 64 nuevas cadenas, permitiendo que Ethereum reduzca la congestión y mejore las velocidades más allá del límite actual.

2.10.1 EVM Ethereum (Máquina virtual de Ethereum)

La máquina virtual Ethereum se encuentra integrada en el software que se ejecuta en el protocolo Ethereum, manteniendo una memoria transitoria, dicha máquina está formada por nodos de la red, los cuales ejecutan el consenso en toda la cadena de bloques, lo cual obtiene niveles extremos de tolerancia a fallas, en las aplicaciones desplegadas denominados DApps, escritos en el lenguaje Solidity.

Estas se ejecutan al mismo tiempo por todos los nodos que conforman la red, llegando a ser la información descentralizada y convirtiéndose en inmutable mediante el consenso descentralizado, sin embargo, dicha ejecución no es gratuita ya que conlleva gastos a los mineros, por lo que es necesario de gas para recompensar su esfuerzo. (Ethereum, Ethereum Virtual Machine, 2016)

2.10.2 Mecanismos de consenso

La tecnología Blockchain, es básicamente una base de datos descentralizada que opera con pares de nodos distribuidos en todo el mundo, hace uso de mecanismos de consenso, dicho mecanismo es la pieza que permite funcionar la tecnología Blockchain a través de ellos y al no existir un sistema central, la red toma decisiones, (Martínez, 2020), distribuye el trabajo y permite llegar a un acuerdo entre todos los miembros de la red a diferencia de una base de datos tradicional en la que es controlada por un administrador.

Los mecanismos de consenso son procesos mediante los cuales la red permite coordinarse en un entorno distribuido, asegura que todos los miembros de la red estén de acuerdo con el contenido además de garantizar que la red puede ponerse de acuerdo con una fuente única de verdad a pesar de que puede existir intermediarios que quieran generar problemas en la toma de decisiones. (Yousaf, 2018)

Los mecanismos de consenso poseen ciertos objetivos en común que les hacen especiales además de compartir características que respaldan la igualdad y equidad, los cuales se describen a continuación:

Acción: Todos los miembros de la red poseen la misma responsabilidad.

Colaboración: Cada integrante de la red, registra un mejor acuerdo que resulte de interés para el grupo.

Cooperación: El trabajo se realiza en equipo.

Igualdad: Los miembros de la red tienen el mismo puntaje de votación.

Conseguir un acuerdo: Recolectar todos los acuerdos del grupo, tanto como puede.

Participación: Promueve que todos los miembros de la red participen en la votación, nadie es excluido de dicha actividad

2.10.2.1 PoW(Proof of Work)

El algoritmo denominada **prueba de trabajo**, fue propuesto antes de la aparición de las criptomonedas para ser empleado en escenarios antispam, hace uso de hash y se basa en problemas matemáticos que son fáciles de verificar, pero complejos de resolver. (Wu, Song, & Wang, 2020)

- **Como funciona:**

Dicho mecanismo hace que los mineros ejecuten software hash en sus computadores y aprovechen el poder de su hardware para resolver ecuaciones matemáticas complejas y verificar nuevas transacciones, cuando han llegado a encontrar el resultado, son recompensados con la criptomoneda de la red.

- **Ventajas**
    - No es necesario ETH para comenzar, conforme se mina se va obteniendo recompensas.
    - Relativamente fácil de implementar frente a (PoS)
- **Desventajas**
    - Este algoritmo de consenso trae consigo cierta desventaja debido a que el modelo requiere de una gran capacidad de cómputo, además de consumir una cantidad alta de energía para la minería, lo que le convierte en un algoritmo poco aplicable debido a las grandes emisiones de carbono.

2.10.2.2 PoS(Proof of Stake)

Este mecanismo conocido como **prueba de participación**, requiere que los usuarios participen con sus criptomonedas de Ethereum, para poder convertirse en validadores de la red, de igual forma que los mineros los validadores son responsables de ordenar las transacciones y crear nuevos bloques para que todos los nodos coincidan, es necesario apostar 32 ETH para poder ser validador, además de que este mecanismo trae consigo ciertas ventajas. (Ethereum & wackrow, Prueba de Participación, 2021)

- **Como funciona:**

A diferencia de (PoW), este mecanismo no necesita potencia computacional sino que hace que se apueste el ether para ser miembro de la red, por ejemplo, Ethereum usa para elegir de manera aleatoria a los validadores y estos a su vez ya no necesitan minar bloques, sino que son recompensados por la creación y

validación de bloques, (Ribera, 2018), después de realizar la validación conocida como certificación(attesting), obtienen recompensas al igual que cuando crean, estos pueden llegar a ser sancionados si validan y aceptan operaciones fraudulentas.

- **Ventajas:**

- Mejor eficiencia energética.
- Barreras de entrada más bajas, debido a que no se requiere hardware avanzado para crear nuevos bloques.
- Mayor inmunidad de centralización en la red, ya que conduce a la existencia de más nodos en la red.
- Mayor compatibilidad con cadenas fragmentadas, mediante el proceso de segmentación horizontal divide la base de datos en varias tablas de acuerdo con una determinada regla.

- **Desventajas:**

- Se necesita tener una cantidad alta de ETH, para poder ser un validador.

2.10.3 ERC (Solicitud de comentarios de Ethereum)

Etiqueta asignada para la comunidad Ethereum como un mecanismo para definir y especificar estándares de forma que los tokens definidos con dichos estándares tengan propiedades comunes y sea interoperables.

2.11 Versiones de Ethereum

2.11.1 Ethereum 1.0

La generación 1.0 denominada Ethereum, se hizo pública en 2014, años después en 2017 se creó la fundación que lleva el mismo nombre mediante la cual se logró impulsar

un proyecto para que no sea simplemente una moneda, sino que también permita crear aplicaciones distribuidas lo que le diferencia de bitcoin.

La red de Ethereum 1.0 solo puede admitir alrededor de 30 transacciones por segundo, por lo que Ethereum en su siguiente generación ofrece llegar a 100.000 transacciones por segundo y cambiar su algoritmo de consenso.

Esta generación hace uso de un mecanismo de consenso denominado, **prueba de trabajo (PoW),** mediante el cual valida las transacciones en los bloques y evita transacciones fraudulentas.

El mecanismo de consenso **(PoW),** permite que el primer minero en resolver la operación matemática agregue una nueva transacción al registro de todas las transacciones que componen la Blockchain, para luego obtener su recompensa con la criptodivisa de la red, pero este mecanismo tiene un consumo alto de energía, por lo que Ethereum está migrando su proyecto al nuevo mecanismo.

2.11.2 Ethereum 2.0

La generación 2.0 conocida también como Eth2 o "Serenity", tiene como objetivo ser más eficiente en cuanto a velocidad, escalabilidad y también resolver problemas de seguridad, para cumplir con todos estos objetivos según la documentación de (Ethereum.org, 2021) tiene en marcha un plan de actualización con las siguientes fases.

- **Primera fase llamada "Cadena Beacon".**

  Se presenta una prueba de participación en Ethereum, dicha prueba se despliega en una cadena de bloques separada de la red principal.

- **Segunda fase llamada “The Merge”.**

  En esta etapa se fusionará la red principal de “Eth1” con la Cadena de Baliza de “Eth2” y el sistema de fragmentación, donde se marca la transición completa al mecanismo de **PoS** “**prueba de participación**” y la etapa final para el mecanismo de **PoW** “**prueba de trabajo**”.

- **Tercera fase llamada “Shard Chains o Cadenas de Fragmentos”.**

  Según lo especificado en la hoja de ruta (Vitalik, 2020), en dicha etapa distribuyen la carga de la red en 64 nuevas cadenas, facilitando mejorar la escalabilidad y la capacidad de Ethereum, donde también consta la finalización de la transición del algoritmo de consenso de **PoW (prueba de trabajo)** a **PoS (prueba de participación)**, obteniendo así una red Ethereum con mayor capacidad de transacciones.

Cabe recalcar que la generación 2.0 hace uso del algoritmo de consenso denominado, prueba de participación (PoS), el cual trae consigo grandes ventajas debido que no requiere mayor computo ni alto consumo de energía, el usuario apuesta la criptomoneda nativa de una red y de acuerdo con la cantidad que aposto en la red, es seleccionado para ser validador y obtener recompensas por la propuesta exitosa de un bloque.

La actualización a Ethereum 2.0 promete hasta 100.000 transacciones por segundo, ya que tienen planificado la implementación de cadenas de fragmentos y PoS. (Kelly, 2021)

2.12 Crypto Wallet

La billetera criptográfica es un software que permite interactuar al cliente y la red de la criptomoneda que en nuestro caso es Ethereum, además de gestionar y almacenar las claves

públicas y privadas, algunas billeteras como MetaMask, permite guardar de manera segura el Ether o criptoactivo que se ha comprado u obtenido en distintas transacciones, a la par que ofrecen una puerta de entrada a las aplicaciones Blockchain, presenta distintas funcionalidades similares a una aplicación de un banco tales como leer el saldo, enviar transacciones, conectarse a aplicaciones y desplegar contratos inteligentes.

2.13 Tesnet

Dicho software tiene el mismo comportamiento y es una copia idéntica de la Blockchain que tiene que usar una criptomoneda, en el desarrollo de esta DApp se hace uso de la cadena de bloques de Ethereum con fines de prueba sin coste alguno.

Esta red de prueba está diseñada para realizar experimentos conforme se vaya desarrollando las DApps, sirve también como simulador para futuros desarrolladores, esto no causa ninguna interrupción a la red principal de Ethereum, las monedas o divisas y los montos de las cuentas asignadas son entidades que están separadas de la red principal de Ethereum es por esta razón por lo que carecen de valor real, para conseguir divisas se hace uso de Ethereum Faucets.

2.13.1 Rinkeby Ethereum Tesnet.

Es una red de prueba de autoridad de Ethreum, por lo que utiliza mecanismos de consensos distintos a la red principal ya que esta hace uso de la red de prueba de trabajo donde cada transacción que se realice se va sumando como un bloque a una cadena, además cifra la información para proteger la seguridad y privacidad de las transacciones, lo que le convierte las transacciones en anónimas, dicha plataforma junto con infura (Infraestructura como servicio), permite mediante una api realizar transacciones dentro la Blockchain de Ethereum.

2.13.2 Ethereum Faucet

En las plataformas denominadas Ethereum Faucet, se obtiene Ether de prueba para poder desplegar aplicaciones en la red de prueba de Ethereum, dichas plataformas piden al usuario que ingrese el enlace de una publicación echa en alguna red social para comprobar de esta forma que no sea un robot, una vez se pasa la validación se obtiene la cantidad de Ether solicitada para poder desplegar en una testnet.

Para que esta red de prueba funcione y poder obtener Ether es necesario seguir los siguientes pasos, en una plataforma.

1. Copiar la dirección de Rinkeby de la billetera de Metamask a twitter o alguna red social.
2. Una vez publicado en la red social, se procede a copiar el enlace de la publicación.
3. Pegue la url copiada en el campo de búsqueda https://faucet.rinkeby.io/
4. A continuación, seleccionar la mayor cantidad de Ethers.

2.13.3 Infura Ethereum API

Plataforma en la nube, específicamente es una plataforma con infraestructura como servicio (IaaS), que contiene un conjunto de herramientas de alta disponibilidad para desarrolladores, proporciona acceso rápido y confiable a las redes Ethereum para la creación de DApps, además de proporcionar una API, la cual nos permite desplegar Smart Contracts en una red de pruebas y conectarnos a la cadena de bloques. (ConsenSys, 2018)

Se le considera un servicio público el cual nos ayuda a reducir la barrera de entrada y simplificar el acceso a los datos de Ethereum.

2.13.4 Etherscan

Es una plataforma líder en análisis y exploración de bloques de Ethereum, se creó como una plataforma descentralizada de contratos inteligentes que permite explorar, realizar búsquedas y proporcionar datos analíticos e información de las distintas direcciones, oficialmente fue lanzada en 2015 con el objetivo de proporcionar acceso equitativo a los datos de Blockchain. (Etherscan, 2021)

Mediante esta plataforma podemos ver las transacciones que se realizan en la Blockchain de Ethereum y de igual forma ver como el Smart Contract, interactúa con la Blockchain y la aplicación descentralizada, en caso de ser los desarrolladores y dueños del contrato se puede verificar el contrato, permitiendo el acceso público.

2.14 Gas Ethereum

Se denomina gas a la unidad que mide la cantidad de esfuerzo computacional necesaria para ser ejecutada dichas transacciones, por lo tanto, cualquier fragmento de cálculo programable, ya sea creación de contratos, llamadas de mensajes, transacciones de una cuenta a otra o por la ejecución de un Smart Contract tiene un costo acordado universalmente en términos de gas. (WOOD, 2021)

Para mayor seguridad la red Ehtereum implementó las tarifas de gas para cada calculo en la red, evitando así spam, bucles infinitos o desperdicio computacional, además sirven para dar mantenimiento a la red y recompensar a los mineros, dichas tarifas se cobran de acuerdo con el uso computacional, el gas que no se haga uso en las transacciones se le retorna al usuario quien realizo la transacción, llegar a almacenar alrededor de 1 kb en la cadena de bloques, tiene un coste de USD $2.88, cuanto cada Ether valía $450 en 2017. (Bruno, 2017)

Name	Value	Description
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
$G_{jumpdest}$	1	Amount of gas to pay for a JUMPDEST operation.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{warmaccess}$	100	Cost of a warm account or storage access.
$G_{coldaccountaccess}$	2600	Cost of a cold account access.
$G_{coldsload}$	2100	Cost of a cold storage access.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	2900	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{selfdestruct}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{selfdestruct}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	50	Partial payment when multiplied by the number of bytes in the exponent for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the *Homestead* transition.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	16	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

Fig 6. Costos relativos en gas de las operaciones que una transacción puede efectuar. (Wood, 2021)

El gas tiene un precio que se paga usando unidades decimales de Ether, denominados Gwei que equivale a 0.000000001 ETH (10-9 ETH) y también posee un límite que es un valor donde indica la cantidad máxima de unidades de gas que la red Ethereum puede manejar en un momento dado. El límite varía de acuerdo con la operación que se vaya a realizar. (Bruno, 2017)

Operación	**Unidades de Gas**
Transacción	21.000 unidades de gas.
Smart Contract	Varía entre 130 y 145 mil unidades de gas.

Bloque	Establecido en no superar los 8 millones de unidades de gas.

Tabla 2. Unidades de gas por operación en Ethereum. (Ethereum, 2021)

2.15 Aplicaciones Descentralizadas (DApp)

Una DApp consiste en aplicaciones descentralizadas de código abierto que hace uso de criptografía en su núcleo para almacenar sus registros e información en una Blockchain mediante el uso de un Smart Contract, se les describe como aplicaciones peer to peer esto debido a que no existe un servidor o entidad que lo controle como es el caso en un modelo cliente-servidor. (Yano, Dai, Masuda, & Kishimoto, 2020 ed.)

La arquitectura que maneja las aplicaciones descentralizadas se basa en lenguajes de etiquetado como es HTML en conjunto con JavaScript y CSS o frameworks tales como Angular, ReactJS para el front-end y un servidor basado en java o NodeJs para el back-end.

Para crear las aplicaciones descentralizadas con Ethereum, se incorpora una librería llamada web3.js, esta librería se encarga de procesar y enviar los datos a la red de Ethereum, mediante el uso de un protocolo RPC.

Debido a que la Blockchain de Ethreum pesa más de 100GB, se hace uso de una extensión llamada Metamask compatible en los principales navegadores, conectando nuestro navegador con Ethereum, esta extensión permite firmar transacciones en la red de Blockchain. (Yano, Dai, Masuda, & Kishimoto, 2020 ed.)

Al usar Blockhain en las DApps estas descentralizan la información y también logran adquirir ciertas propiedades tales como: (Yano, Dai, Masuda, & Kishimoto, 2020 ed.)

1. Para un usuario le es visible lo que va a suceder antes de ejecutar una función o enviar datos.

2. Si el usuario realizó una transacción, esta no se puede manipular o eliminar.
3. La gobernanza se puede descentralizar para que los usuarios de la aplicación participen directamente en su gestión.

2.16 Metamask

Es una billetera criptográfica y una puerta de entrada a las aplicaciones blockchain, ayuda a la administración de cuentas y conecta al usuario con la cadena de bloques, permite el manejo de claves y cuentas, incluidas carteras de hardware, brindando así mayor seguridad con respecto a almacenamiento de las claves de usuario en solo servidor central. (MetaMask, 2021)

Metamask cuenta con grandes características para un ambiente de desarrollo, ya que dispone a nivel mundial de una API de Ethereum que identifica a los usuarios de navegadores compatibles con Web3 y permite firmar transacciones, indicando al usuario que se va a realizar dicha acción logrando así evitar ataques o suplantación de identidad.

Contiene varias conexiones que facilitan el acceso rápido a la cadena de bloques de Ethereum y varias redes de prueba tales como Rinkeby.

Metamask es compatible con cualquier Blockchain, que disponga de una API JSON RPC, compatible con Ethereum.

2.17 Solidity

Solidity, un lenguaje de programación de alto nivel y también define estándares para el futuro de la tecnología Blockchain, es un lenguaje orientado a objetos, acepta herencias y librerías definidas por el usuario. Creado para el desarrollo de Smart Contracts, posee una sintaxis muy similar a JavaScript, dicho lenguaje utiliza como extensión .sol, mediante el uso

de este se puede desarrollar funciones que posterior residen en una dirección especifica de la Blockchain de Ethereum. (solidity, 2021).

La máquina virtual de Ethereum admite la ejecución de código de Solidity, este facilita a los desarrolladores la creación y despliegue de Smart Contracts a la vez que permite tener acceso al estado y propiedad de Smart Contracts ya desplegados.

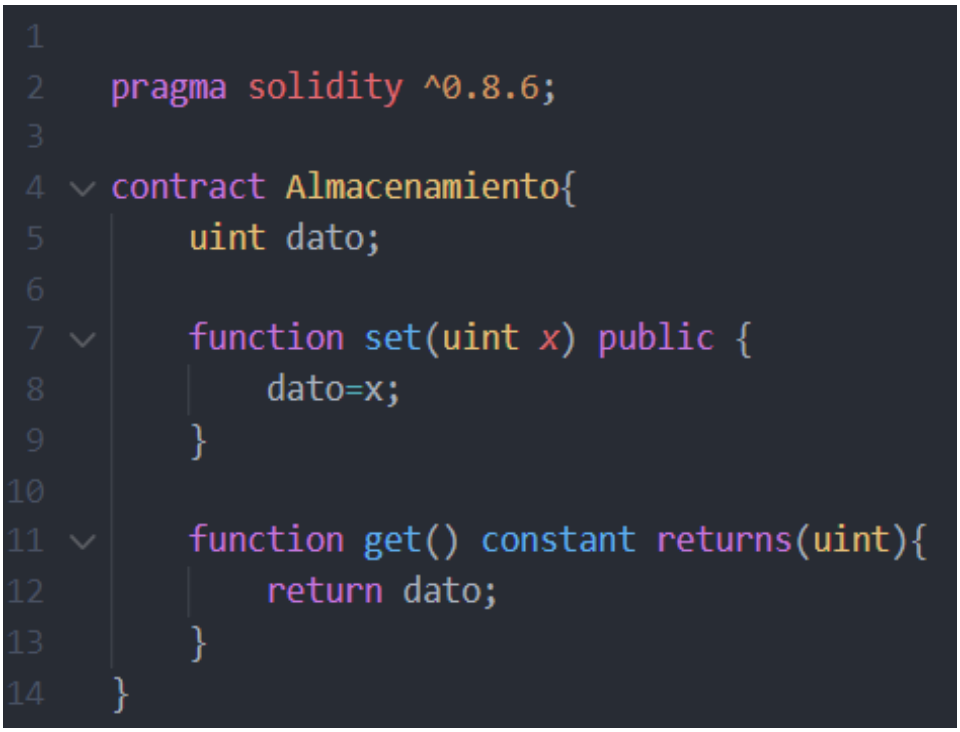


Fig 7. Estructura de un Smart Contract. Fuente (Autores)

2.18 Smart Contracts

Los contratos Inteligentes de acuerdo a (Levi & Lipton, 2018), se definen como acuerdos establecidos entre dos o más partes donde su ejecución es automatizada, es decir hace que ciertas acciones lleguen a suceder o no como resultado del cumplimiento o incumplimiento de una serie de condiciones específicas, son contratos como un documento legal en el mundo real pero completamente digitales los cuales consisten en un pequeño programa o script informático que es codificado dentro de una Blockchain, que trae consigo un gran aporte a los servicios

jurídicos que a su vez respaldan y hacen cumplir, en un Smart Contract no es necesario de un tercer ente cada vez que se quiera validar o dar fe de un contrato.

Con el contenido de un contrato inteligente se establece una limitación que recae sobre las distintas interpretaciones, que pudiese tener el contenido de lo convenido, ya que este se configura como un protocolo informático dispuesto hacer realidad lo pactado y nada más que lo pactado, al contrario de lo que puede surgir con los contratos convencionales que surgen disputas sobre el contenido. (Levi & Lipton, 2018)

Se pueden usar para realizar funciones en una gran variedad de industrias, son relevantes en áreas como el comercio de activos financieros digitales con transferencia legal de propiedad, en tramites o servicios bancarios, procesos logísticos, seguimiento del origen y ruta de los bienes, almacenamiento descentralizado y uso de energía renovable.

La elaboración de los Smart Contracts, hace uso del compilador y lenguaje Solidity, el cual genera lo siguiente: (Modi R. , 2018)

2.18.1 ABI (interfaz binaria de la aplicación):

Interfaz que contiene todas las funciones públicas y externas junto con sus parámetros y tipos de retorno en un formato json, almacenados en la cadena de bloques, de cierta forma es la interfaz definida en el contrato y que se puede invocar a esta para la interacción en las aplicaciones mediante librerías.

2.18.2 Bytecode (código de bytes):

Representa el contrato y se implementa en el ecosistema Ethereum mediante la ejecución de la máquina virtual de Ethreum.

Según el texto relatado por (Palacios, 2018), menciona que los Smart Contracts poseen ciertas propiedades que heredan de la tecnología Blockchain, las cuales se describen a continuación.

- **Distribuido:** Los contratos se distribuyen por la red, y son validados por un algoritmo de consenso.
- **Inmutable:** Cuando la información es creada no se puede después modificar, nadie puede alterar el código del contrato.
- **Determinista:** Es decir dada una entrada todos los nodos deben producir el mismo resultado, eso implica que el código no debiera tener ninguna aleatoriedad.
- **Verificable:** Una vez implementado un Smart Contract, obtendrá una dirección única, antes de usarse las partes que demuestran interés, pueden ver y verificar el código para tener mayor seguridad.

2.18.3 Funcionamiento de un SmartContract en una Blockchain:

- La lógica que se desee implementar con Smart Contracts se implementa haciendo uso del lenguaje Solidity.
- Se compila el código de Solidity y se genera Bytecode para poder distribuir en la red Blockchain.
- La transacción incorpora el código de bytes y el minero extrae el bloque con la transacción. Simultáneamente, la transacción se registra en la red Blockchain.
- El usuario genera un código de bytes para llamar a la función definida por el código del contrato inteligente distribuido utilizando ABI (interfaz binaria de aplicación) que permite la construcción de una interfaz amigable y posterior entregarlo a la red Blockchain como una transacción.

- El minero ejecuta el código de bytes de acuerdo con código de contrato inteligente distribuido en la EVM (máquina virtual de Ethereum), se calculan las tarifas de gas y se agregan al bloque, si los resultados son válidos se refleja en el estado de la cadena de bloques.

2.19 Registros Distribuidos (DLT)

Consiste en una base de datos, donde existen copias de sus datos distribuidas en distintos equipos, la base de datos se actualiza por medio de consenso de los participantes, esto permite que la información siempre se mantenga sincronizada.

Es debido a esto que Blockchain es una innovación tecnológica que está siendo muy usada dentro de los DLT debido a que permite registrar, sincronizar transacciones y distribuir todos esos datos entre múltiples usuarios en múltiples ubicaciones.

Hoy en día las tecnologías DLT que más sobresalen debido a las características que ofrecen son Cardano, Bitcoin, Ethereum esta última mantiene actualizaciones y soportes constates además que posee un propio lenguaje llamado Solidity para el desarrollo de Smart Contracts y también permite hacer pruebas de los desarrollos mediante su red Tesnet, lo que le convierte en una DLT muy eficiente para el desarrollo de nuestra DApp.

2.20 Oráculos en Ethereum

Es una herramienta que se comporta como un puente y nos ayuda a obtener información confiable del mundo real y llevarlo hacia la cadena de bloques o viceversa, son muy usados debido a que los contratos están limitados a interactuar solo con información de la propia red.

Para una mejor explicación, daremos un ejemplo del uso de los oráculos en los Smart Contract, supongamos que nuestro Smart Contract necesita saber el valor de Ethers en dólares

para calcular cuánto se debe cobrar por cada permiso que se registre y como se sabe que el valor del Ether es fluctuante, por lo que no podemos llegar a definir un valor estándar de Ethers en nuestro contrato y es aquí donde haríamos uso de los oráculos para que estos a su vez consulten a una fuente externa segura el valor actual y defina cuanto deberíamos de cobrar por las transacciones.

CAPÍTULO III

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN

3.1 Manejo de herramientas

Para el desarrollo del proyecto, el primer paso es adquirir conocimiento para un manejo adecuado de las herramientas que permiten el desarrollo de aplicaciones en Ethereum en conjunto con el uso de tecnologías Blockchain, para esto es necesario aprender sobre el manejo de su lenguaje de programación denominado Solidity, así como de igual manera aprender sobre el despliegue de Smart Contracts en una cadena de bloques en Ethereum, donde también se incluye su correcta instalación y configuración de herramientas como Ganache, el Framework Truffle, Visual Studio Code para el desarrollo, ademad de tomarse en cuenta la instalación, configuración e integración de librerías como Web3js y babel y el manejo de la billetera Metamask para desplegar en una red de pruebas y por ultimo adquirir conocimientos de ReactJs para el desarrollo Frontend.

Otro punto muy importante que tomar en cuenta para interactuar y llevar a cabo el desarrollo con las distintas herramientas es obtener los conocimientos necesarios, mediante la investigación y la capacitación en plataformas que ofertan cursos online sobre el desarrollo en Blockchain, así como también la revisión de la documentación que ofrecen las distintas herramientas en sus plataformas web.

3.2 Análisis de Requerimientos de Software

Se obtuvo las versiones más recientes y estables de las distintas librerías y herramientas utilizadas dentro el desarrollo, esto debido a que se encuentran en constantes actualizaciones con el objetivo de ser compatibles en distintas plataformas y con lenguajes de desarrollo modernos, además de brindar mayor seguridad al desarrollador y también controlar las brechas de inseguridad existentes en las librerías usadas en los distintos desarrollos de DApps con tecnología Blockchain.

3.3 Características Principales de la Aplicación

Lo que se requiere de la aplicación es poder llevar un registro y a su vez gestionar y tener un historial de las presencias y ausentismos del personal que forma parte de un departamento o empresa, para llegar a cumplir con dichas funcionalidades se hará uso como herramienta de almacenamiento la Blockchain de Ethereum, debido a que admite la implementación de Smart Contracts, lo cual permite que la información registrada se mantenga inmutable, además de permitir validar mediante roles (usuario, administrador), se apruebe el acceso a ciertas funciones que tenga el sistema.

El usuario final obtendrá una aplicación web que permite al personal de Recursos Humanos, revisar los permisos de ausentismos y estar seguros de que estos no fueron alterados ya que se encuentran desplegados en una Blockchain.

3.4 Levantamiento de Requerimientos

En esta fase se define que busca el usuario, cual es el propósito y la finalidad del software que se va a desarrollar, se lleva a cabo el levantamiento de los requerimientos para el desarrollo de la DApp, manteniendo controles continuos con la parte interesada para la validación de estos y en caso de ser necesario realizar cambios.

Con la ayuda del tutor se establecerá que parámetros son necesarios para que la aplicación cuente con los estandartes que debe cumplir a nivel de desarrollo como a nivel de implementación dentro del departamento.

3.5 Definición de requerimientos de usuario

- Acceso al sistema.
- Registro de usuarios.

- Registro de presencias o ausentismos.
- Visualización de reporte de presencia y ausentismos.

3.6 Definición de requerimientos del sistema

- Smart Contract que permita registro de personal.
- Smart Contract para generar permisos donde se permita el cambio de estado de aprobación de los permisos.
- Desarrollo donde el Smart Contract pueda cambiar de owner (dueño).
- Validación para evitar duplicidad en el registro.
- Despliegue en la red de pruebas de Ethereum.

3.7 Definición de requerimientos funcionales

- Administración de usuarios.
- Interfaz que permita registro de personal e interacción con el Smart Contract.
- Interfaz para registrar presencias o ausentismos.
- Manejo de roles administrador & personal.
- Acceso a reportes y creación de nuevos usuarios mediante rol administrador.
- Visualizar reporte de registros del personal.

3.8 Definición de requerimientos no funcionales

- **Requerimiento de Rendimiento**

  El rendimiento del sistema en este caso depende tanto de la velocidad del internet como de igual forma de la velocidad de transacciones que permite la Blockchain de Ethereum, que en este caso la generación Ethereum 2.0 soporta entre 15 y 40 transacciones por segundo, mientras dura el cambio de algoritmo de consenso.

- **Requerimiento de Seguridad**

  La información se mantiene protegida por medio de la tecnología Blockchain y el uso de criptografía simétrica y asimétrica, la interfaz de usuario tiene una arquitectura propia de seguridad, mantenida por la comunidad de software libre.

- **Rendimiento de Usabilidad**

  El uso del sistema es bastante intuitivo y fácil de usar, lo que cambia es los permisos que son asignados dependiendo los roles que se tenga y el acceso a ciertas opciones, debido a que el rol administrador tiene acceso a todas las opciones es decir goza de todos los permisos, lo que no sucede con el personal que tiene permitido el acceso a ciertas opciones.

3.9 Diagrama de la aplicación.

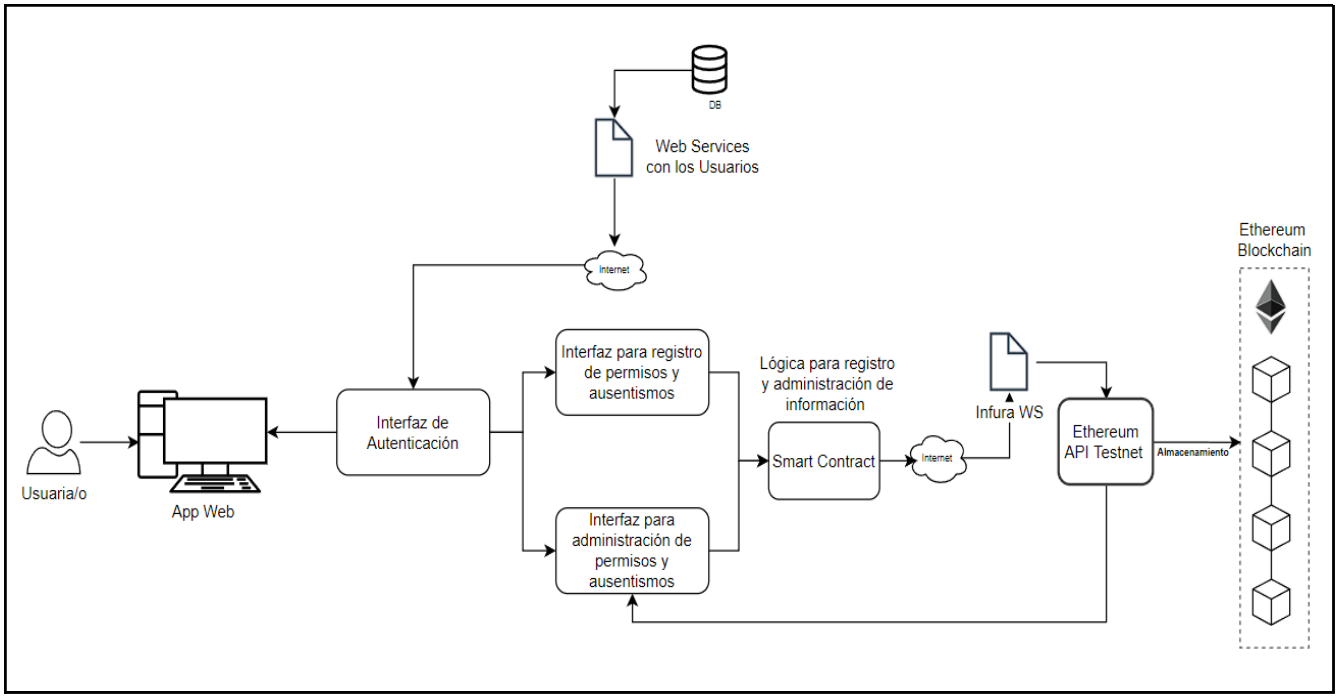


Fig 8. Esquema del sistema. Fuente (Autores)

En el esquema anterior se define al usuario en primer lugar, ya que es la entidad principal que se beneficiara de la aplicación web, a dicha aplicación web está atado la interfaz de autenticación, mediante la cual se tiene acceso al sistema completo.

El sistema está dividido en dos partes, la primera parte es el registro del personal mediante un Smart Contract, el cual puede registrar y solicitar aprobación de la petición que realice, y la segunda parte es donde existe un administrador que es la persona quien tenga acceso a la interfaz de administración y es la encargada de aprobar o negar la solicitud de permiso o ausentismo enviada por el personal.

Posterior a esto, se hace uso de la Blockchain de Ethereum para almacenar los datos de manera que se queden almacenados y se conviertan en inmutables, el administrador es el único que tiene acceso a ver toda la información que se ha generado durante el transcurso del tiempo y que se ha ido almacenando en la Blockchain.

Todo esto es gestionado por una aplicación web, que permite en primer lugar realizar a los empleados la consulta de todos los eventos que ha realizado y se han ido registrando, además por otro lado permitir al administrador o Recursos Humanos, validar la inmutabilidad de la información y revisar por cada empleado los registros.

En esta etapa se lleva a cabo el desarrollo de la aplicación en conjunto con el desarrollo del contrato inteligente haciendo uso de Vs Code para el desarrollo y de Remix para realizar ciertas pruebas rápidas, ambos son IDEs que en conjunto con Solidity como lenguaje base nos permiten el desarrollo e integración con una interfaz gráfica basada en HTML y JavaScript. Dichos contratos estarán plenamente vinculados con la red de prueba de Ethereum, mediante el plugin

de la billetera Metamask, la cual nos permite desplegar la aplicación en la red de pruebas haciendo uso de la API que nos provee Infura1.

3.10 Diagrama de clases

La aplicación consta de dos clases dentro el Smart Contract, la clase Employee es usada para la gestión de empleados y la clase WorkPermission se usa para la gestión de permisos y ausentismos.

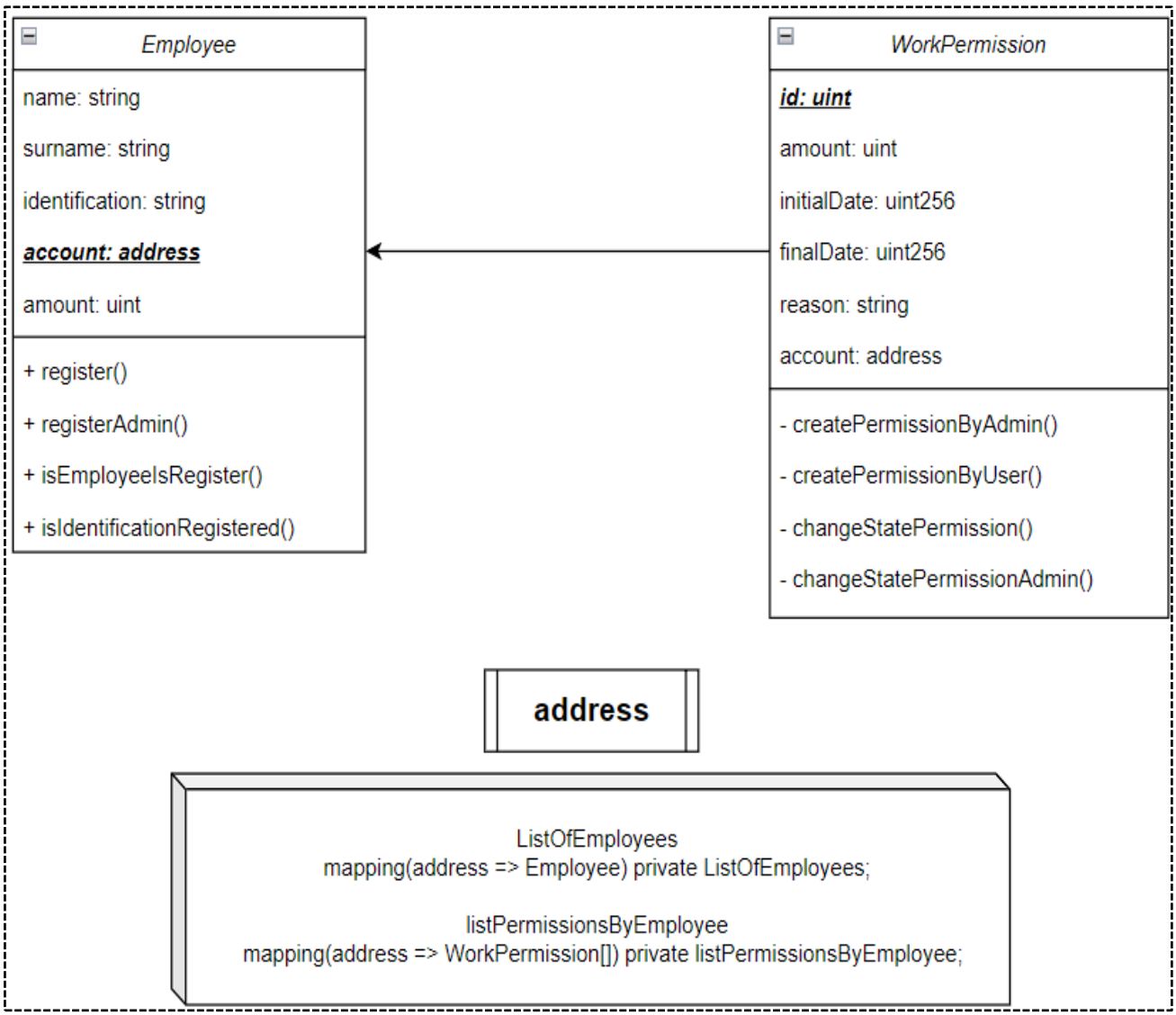


Fig 9. Diagrama de clases de la aplicación

[1] Revisar la sección 2.13.3 Infura Ethereum API

3.11 Desarrollo de la aplicación Blockchain de registro de permisos y ausentismos.

Se pretende que la aplicación sea capaz de llevar un registro de permisos y ausentismos por cada persona registrada y posterior demostrar que la información se almacenó en la Blockchain, convirtiéndose en información capaz de no ser modificada, debido a que esta lleva un histórico de todos los datos proporcionados previamente y de los cambios realizados.

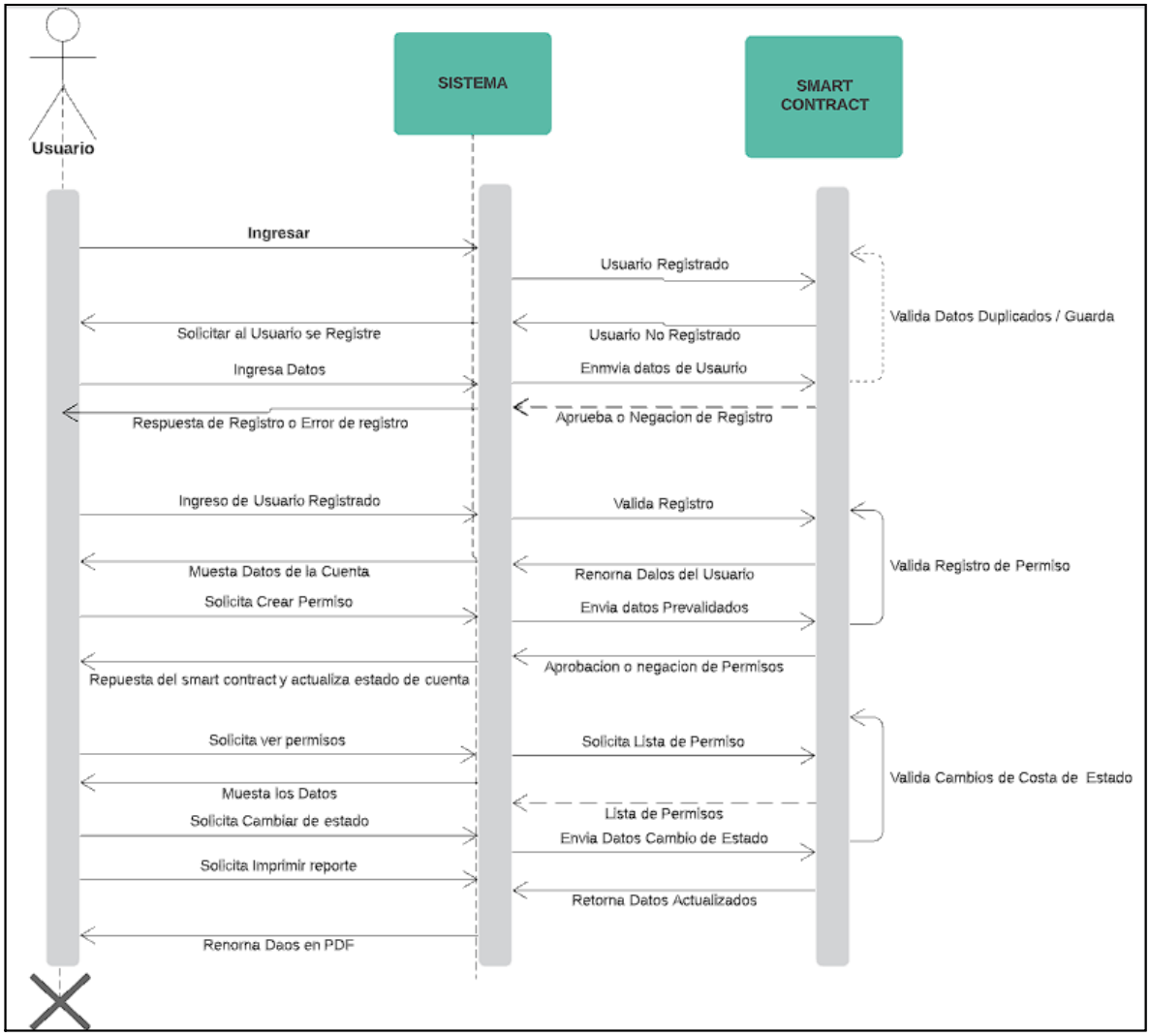
Para lo cual la solución que se plantea es primero registrar a los usuarios mediante una interfaz de registro y luego mediante otra interfaz el usuario podrá ir registrando información cada que se presente algún evento ya sea de permiso o ausentismo y esta a su vez se ira almacenando en la Blockchain de Ethereum para posterior mediante una interfaz mostrar el historial de todos los registros a cada usuario y de forma completa al administrador.

El sistema contara con dos roles, administrador y usuario, siendo el primero el que tenga acceso a los perfiles de todos los usuarios y sea quien apruebe o rechace los permisos, además de contar con la opción de registrar nuevos usuarios, también tiene la opción de cambiar de administrador, a su vez con el rol de usuario tendrá acceso a su propio perfil donde podrá registrar nuevos permisos y contara con la opción de ver el reporte de los registros anteriores o puede cancelar si así fuese el caso.

3.12 Diagrama de secuencia del sistema.

Los siguientes diagramas nos ayudan a ver cómo y en qué orden interactúan las diferentes componentes en el sistema, haciendo énfasis en la secuencia de los mensajes intercambiados por los objetos.

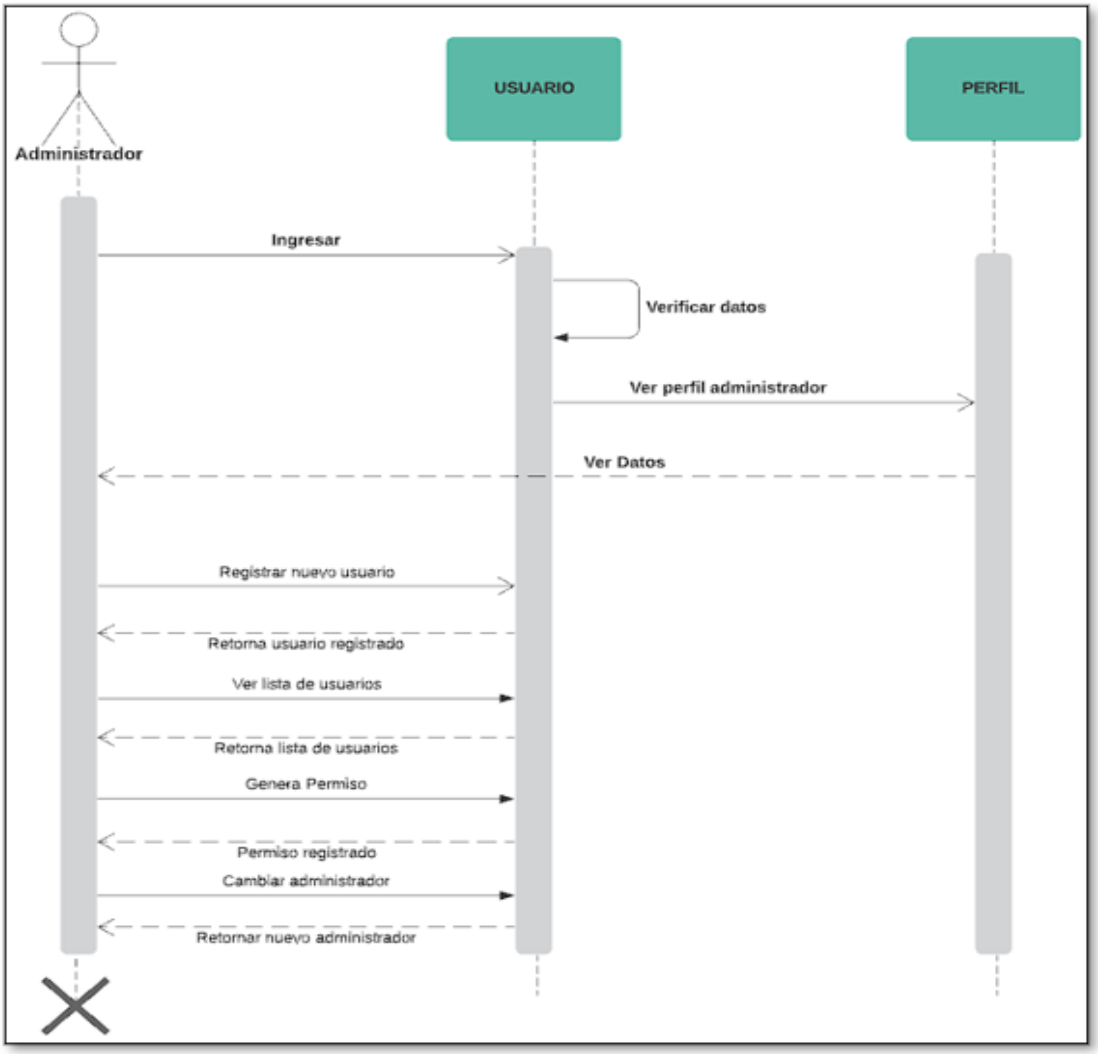
Diagrama de secuencia para acceso como usuario:



ACTOR/OBJETOS	ACCIONES
Usuario	Inicia la secuencia y es quien puede registrarse en el sistema y a su vez es quien puede anotar las presencias o ausentismos en el sistema, así como ver el reporte.
Sistema	Valida que tipo de rol tiene.
Smart Contract	Interactúa con la Blockchain de Ethereum y es donde se encuentra la lógica de la aplicación.

Fig10. Diagrama de secuencia de usuario.

Diagrama de secuencia para administración de usuarios:



ACTOR/OBJETOS	ACCIONES
Administrador	Inicia la secuencia y es quien puede registrar un nuevo usuario, ver listas de usuarios y generar permisos.
Usuario	Valida que tipo de rol tiene el usuario.
Perfil	Acceso a ciertas funcionalidades dependiendo del rol.

Fig 11. Diagrama de secuencia de administrador.

3.13 Implementación del Sistema de registro de permiso y ausentismos

Primero se lleva a cabo la instalación y configuración de herramientas para poder interactuar con Ethereum y a su vez desarrollar y testear los Smart Contracts, además de librerías necesarias para el desarrollo de la interfaz que interactúe con el personal y el contrato.

3.13.1 Instalación y configuración de las herramientas de Ethereum.

Las siguientes herramientas que van a ser instaladas son compatibles en sistemas operativos tales como Windows o Linux, que para el desarrollo de este trabajo se llevó a cabo en ambas plataformas tanto en Windows 10 como la distribución de Linux Ubuntu 20.04, además de que las herramientas instaladas nos permitirán procesar la información en la Blockchain de Ethereum, así como facilitarnos la gestión de librerías y paquetes mediante npm2, necesarias durante el desarrollo.

3.13.2 Instalación de Node.js

Para el desarrollo del proyecto se hizo uso de la versión 14.18.1 LTS de Node.js en conjunto con la versión 6.14.15 de npm, como se puede apreciar en el anexo 1 & 2, dichas versiones son compatibles con las herramientas de desarrollo de Ethereum y nos permite gestionar librerías necesarias durante el desarrollo, además de que Node.js nos ayuda en cuanto a la implantación de Web3 y Solidity para hacer uso de métodos y llamadas mediante JavaScript.

2 Gestor de paquetes para el entorno de ejecución de Node.js.

3.13.3 Instalación de herramientas de compilación de C/C++ en npm.

La instalación de la herramienta en Linux se instaló como se observa en el anexo 3, mediante los comandos **sudo apt-get update** y **sudo apt-get install build-essential**, este último instala el compilador de C, C++, Make, etc, herramientas que son necesarias para que las librerías de Ethereum lleven a cabo la compilación de paquetes C/C++, descargados mediante el gestor de paquetes npm.

En la instalación de Windows se debe ejecutar el siguiente comando **npm install --global windows-build-tools** desde PowerShell, con permisos de administrador haciendo uso del gestor de paquetes npm, cabe recalcar que en las últimas versiones Node.js ya incluye dichas herramientas de compilación, solo se debe seleccionar durante la instalación. Ver anexo 4.

3.13.4 Instalación del compilador de Solidity.

Para llevar a cabo la instalación del compilador de Solidity, ejecutamos la siguiente línea de comandos **npm install solc@0.8.6 --save-dev**, esta librería realiza el trabajo de compilar el código escrito dentro el Smart Contract, si se desea revisar cual es la última versión se puede ejecutar **npm show solc versions**.

3.13.5 Configuración del compilador de Solidity.

En el caso de Visual Studio IDE, si se desea trabajar con una versión en específico del compilador, se debe ingresar a file: preferencias:settings y en la pestaña de User dirigirse hasta extensions, en el apartado de Solidity: Compile Using Remote Version, escribir la versión con la que se va a trabajar del compilador que en este caso es la v0.8.6+commit.11564f7e o también se puede editar el archivo

JSON de settings y agregar la siguiente línea, {“solidity.compileUsingRemoteVersion": "v0.8.6+commit.11564f7e"}

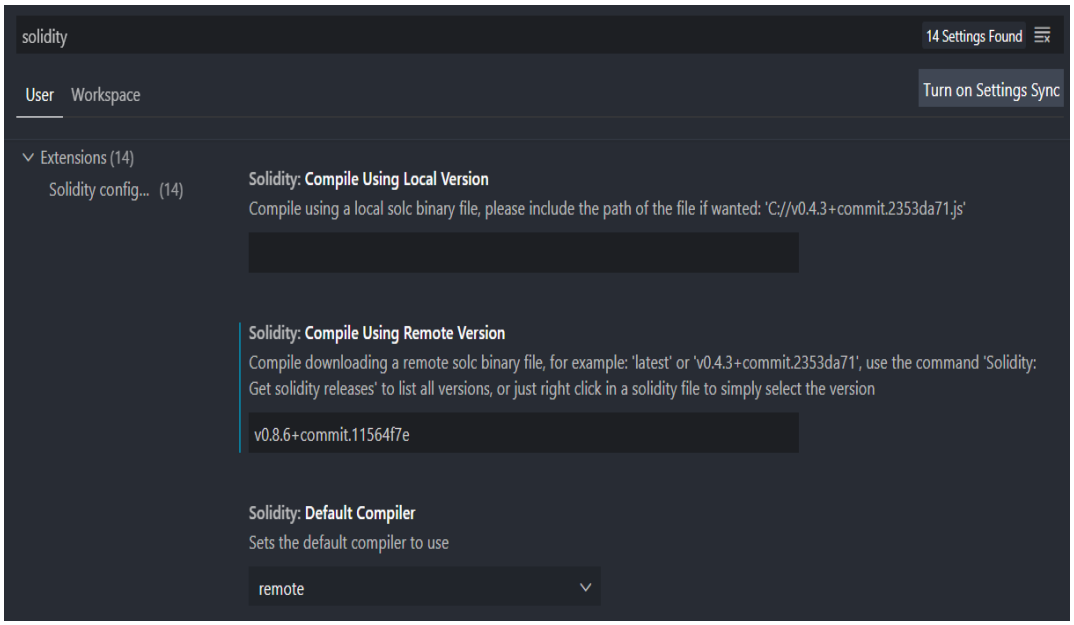


Fig 12. Configuración de la versión del compilador. Fuente (Autores)

3.13.6 Instalación de Web3 Js.

Para la instalación de la librería Web3, ejecutamos la siguiente línea de comandos **npm install web3@1.6.0 --save-dev**, dicha librería nos permite interactuar con un nodo Ethereum mediante el Smart Contract, si se desea ver cuál es la última versión se puede ejecutar **npm show web3 versions.**

3.13.7 Instalación del servidor Ganache

Para poder realizar las pruebas se hace uso de Ganache, software que nos provee de una red de pruebas de Ethereum para el despliegue de contratos, ejecución de pruebas y comandos para su correcto funcionamiento además de permitirnos inspeccionar las transacciones de la red y proveernos de cuentas con Ether que mediante la billetera y el puerto HTTP://127.0.0.1:7545 el cual nos permite interactuar con la aplicación que se está desarrollando, dicho software es compatible

en Linux y Windows, la instalación se llevó a cabo en ambas plataformas como se puede observar en el anexo 5.

3.13.8 Instalación de Truffle Framework.

La instalación del Framework Truffle, nos ayuda a desarrollar, testear y desplegar los Smart Contracts, para poder instalar una versión en específico necesitamos ejecutar la siguiente línea de comandos **npm install truffle@6.14.15 -g** desde la consola ejecutada como administrador como se realiza en el anexo 7, cabe recalcar que se puede instalar la última versión con la siguiente línea de comandos, **npm install -g truffle**.

3.13.8.1 Instalación de Truffle Contract.

Dicha librería nos permite conectar e interactuar desde el front-end de la aplicación con los datos que genera el Smart Contract, para poder instalar es necesario ejecutar la siguiente línea de comandos **npm i @truffle/contract,** es una herramienta útil para poder extraer la información generada del Smart Contract en formato Json y renderizar en la vista.

3.13.9 Instalación y configuración de Metamask Crypto Wallet.

El software está disponible en la página oficial de https://metamask.io/, es un plugin compatible con navegadores como Chrome, Opera y Firefox, dicha herramienta permite tener una puerta de acceso a aplicaciones Blockhain, la instalación como se puede apreciar en el anexo 6, se llevó a cabo en el navegador Chrome, una vez instalado el plugin dentro del navegador se procede a realizar los siguientes pasos.

- **Configuración de la red para uso local con Ganache**

1. Cuando se ingresa a la aplicación, se pide que digite la contraseña con la que se creó la cuenta, una vez ingresado se dirige al círculo ubicado al extremo superior derecho, el cual nos lleva a la ventana con un menú de opciones.

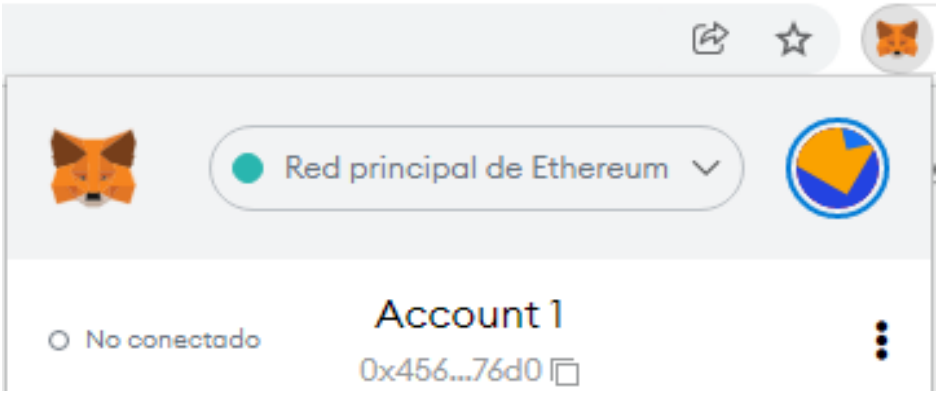


Fig 13. Interfaz de inicio de Metamask. Fuente (Autores)

2. Se despliega un menú el cual contiene una opción llamada **Configuración** la cual nos conduce a una ventana donde debemos seleccionar el apartado **redes** para proceder con la configuración de nuestra red local.

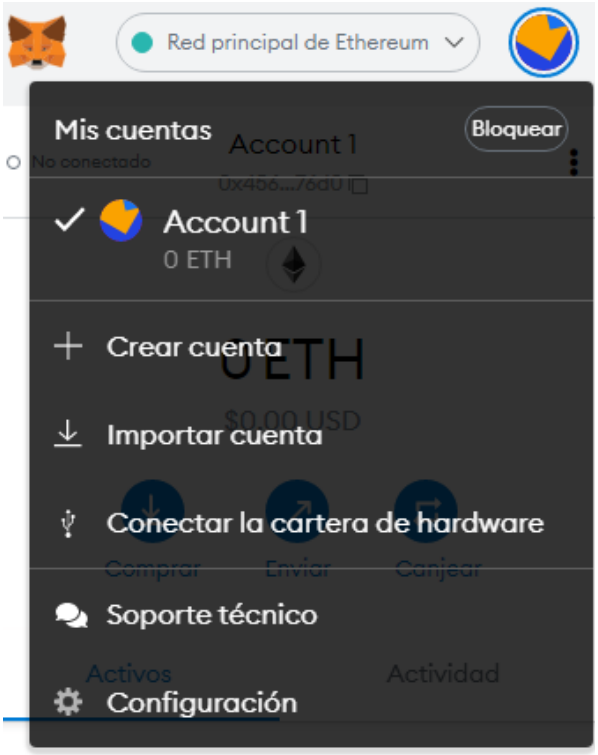


Fig 14. Interfaz de configuración de Metamask. Fuente (Autores)

3. En la ventana **Redes** mediante el botón **Agregar red** creamos una nueva red, en este caso para pruebas locales creamos una red que nos permita la conexión con Ganache, llenando los siguientes campos.
    a. **Nombre de la red** que nos ayude a identificar.
    b. **Identificador de la cadena** el cual nos ayuda a firmar la red, en este caso es **1337**.
    c. La **nueva dirección URL de RPC**, es la dirección ip y el puerto en la que se ejecuta ganache.
    d. El campo símbolo de la moneda y dirección url son opcionales, una vez llenado los campos anteriores procedemos a guardar la configuración y nos aparecerá que la red ha sido agregada exitosamente.

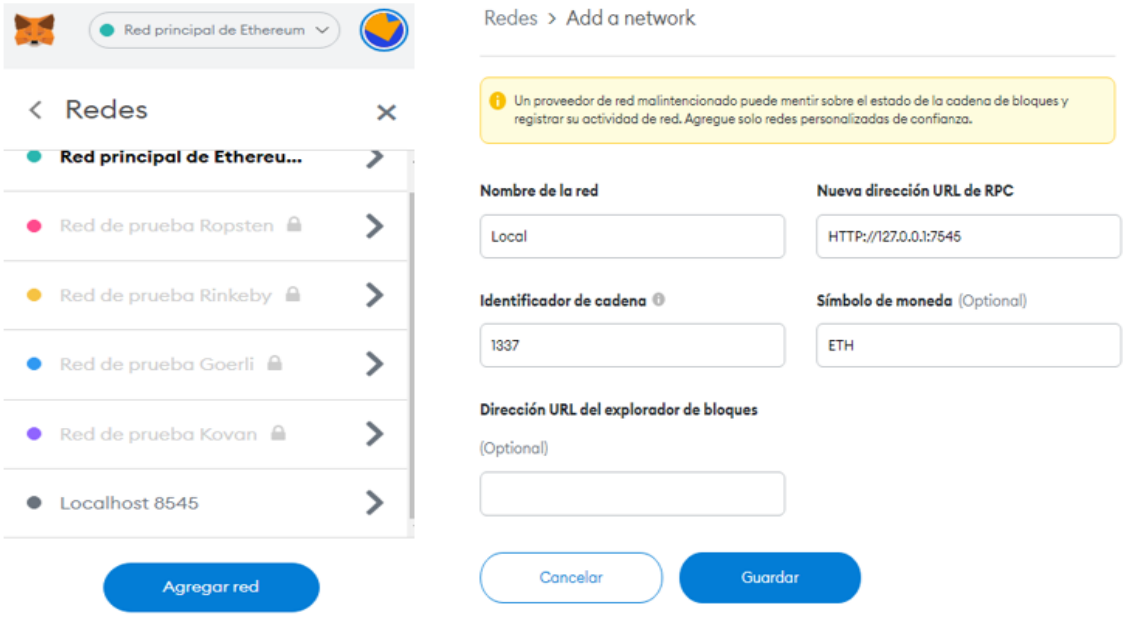


Fig 15. Interfaz para agregar y configurar una nueva red en Metamask. Fuente (Autores)

4. El siguiente paso es importar cuentas desde Ganache a Metamask, para lo cual se lleva a cabo el siguiente procedimiento.

    a. Primero se dirige al círculo del punto 1, el cual nos lleva a la ventana donde seleccionamos **importar cuenta**, se puede copiar la dirección privada de las cuentas que nos proporciona Ganache o seleccionar un archivo JSON, con todas las cuentas.

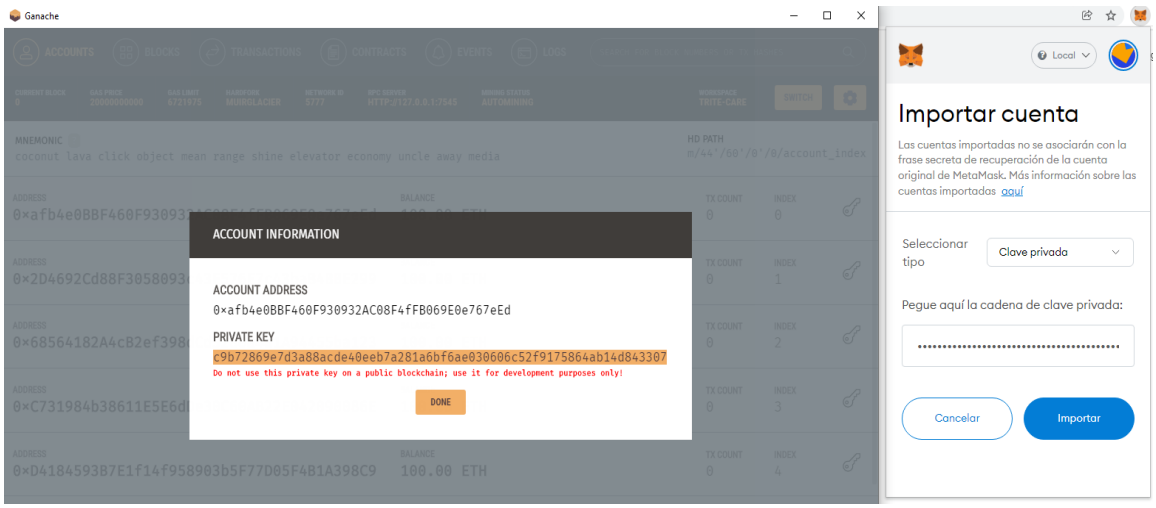


Fig 16. Interfaz de Ganache, con una clave privada para importar a Metamask. Fuente (Autores)

b. Metamask permite importar la clave privada de varias cuentas, para poder interactuar con los Smart Contract, cada cuenta posee 100 ETH.

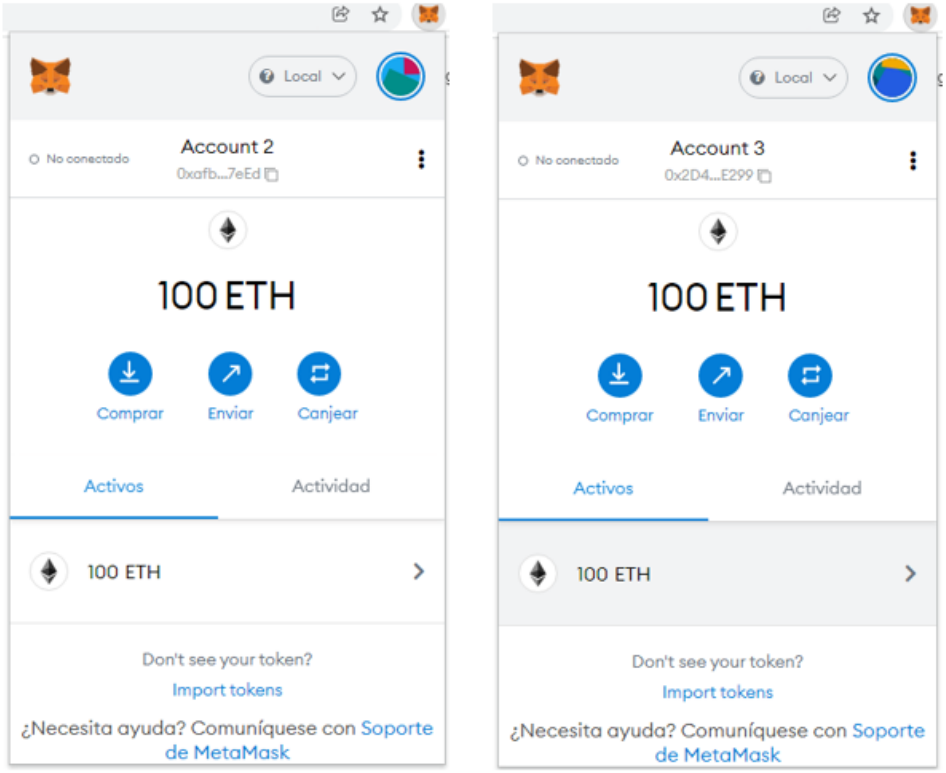


Fig 17. Interfaz con dos cuentas importadas desde Ganache. Fuente (Autores)

- **Configuración de la red con Rinkeby.**

1. Primero se necesita obtener Ether para poder desplegar el Smart Contract en la red de Rinkeby, para lo cual se debe llevar a cabo los siguientes pasos.
    a. Realizar una publicación en Twitter o Facebook, seleccionando la dirección de la primera cuenta, dicha dirección es la que nos asignada Metamask como dirección de nuestra billetera.

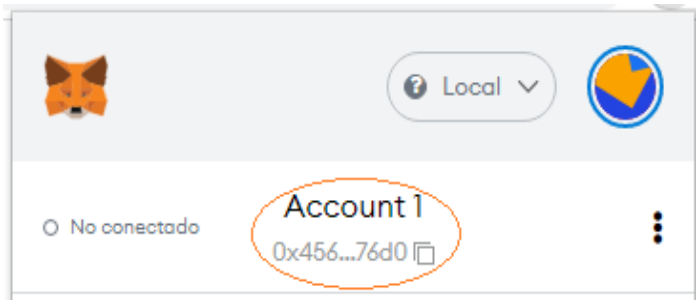


Fig 18. Dirección de la Billetera. Fuente (Autores)

b. Una vez realizada la publicación con la dirección de la cuenta en la red social, se debe copiar el enlace generado por dicha publicación y pegar en la barra de Rinkeby y elegir la cantidad que se desee para realizar las transacciones.

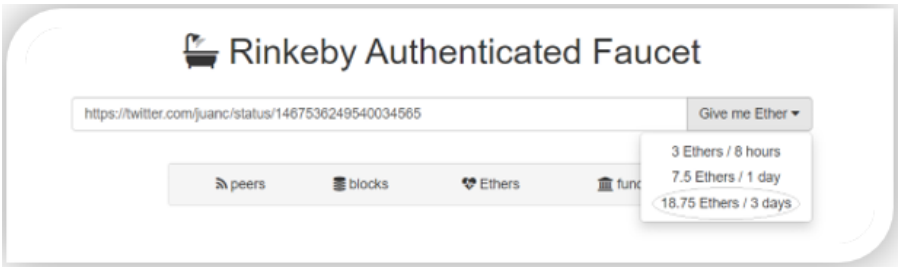


Fig 19. Interfaz para obtener Ether e interactuar con una red de pruebas. Fuente (Autores)

3.13.10 Instalación de Metamask Detect Provider.

Para la instalación de la librería detect provider, ejecutamos la siguiente línea de comandos **npm i @metamask/detect-provider**, dicha librería nos permite detectar si existe la billetera Metamask en el navegador.

3.13.11 Desarrollo de Smart Contract

A continuación, se describe la estructura del Smart Contract que lleva por nombre "ThesisContract", el cual está desarrollado con la versión ^0.8.6.

Dentro el contrato está definida la dirección **address public owner**, la cual permite identificar al dueño del **Smart Contract** y definir que es la persona quien despliegue.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.6;
/*
 @ERR_001: Cuenta ya Registrada
 @ERR_002: Identificacion ya registrada
 @ERR_003: No existe cuente registrada
 @ERR_004: No existe identification registrada
 @ERR_005: No es el creador
 @ERR_006: No es el existe permiso
 @ERR_007: Plata diferente
 @Test: "A","A","1"
 **/

contract ThesisContract {
    address public owner;

    struct Employee {
        string name;
        string surname;
        string identification;
        address account;
        uint amount;
    }

    enum StateWorkPermission {Requested, Approved, Cancel, Denied, Finished}
    struct WorkPermission {
        uint id;
        uint256 initialDate;
        uint256 finalDate;
        string reason;
        StateWorkPermission state;
        address account;
    }

    mapping(address => Employee) private ListOfEmployees;
    mapping(address => bool) private listOfEmployeesRegistration;
    address [] private listAccounts;

    mapping(address => WorkPermission[]) private listPermissionsByEmployee;
    uint idOfPermissions;

    event employeeRegister(address adr, string data);

    constructor() {
        owner = msg.sender;
        idOfPermissions = 0;
    }
```

Como se puede apreciar en el código del contrato inteligente consta la definición de struct llamado **Employee,** con los siguientes campos:

- **name:** Almacena el nombre con el que se registra al usuario.
- **surname:** Sirve para almacenar los apellidos de la persona que se registra.
- **identification:** Variable que sirve para la cédula de la persona.
- **account:** Ayuda a identificar la cuenta que pertenece al empleado.
- **amount:** Maneja el monto por cada transacción.

También se tiene una variable restringida llamada **StateWorkPermission**, la cual nos permite restringir una variable con los siguientes estados de **Solicitado**, **Aprobado**, **Cancelado**, **Denegado** y **Finalizado,** debido a que estos son los únicos estados que manejamos dentro la etapa de aprobación de permiso o ausentismo.

Además de contar con la definición de un segundo struct llamado **WorkPermission,** el cual contiene los siguientes campos:

- **id:** Otorga un valor entero, el cual nos ayuda a ir identificando al usuario.
- **initialDate:** Fecha inicial la cual registra el inicio de la solicitud enviada por el usuario.
- **finalDate:** Fecha final que registra hasta cuando se solicita el permiso.
- **reason:** Se registra el motivo por el que el usuario solicita el permiso.
- **StateWorkPermission state:** Guarda el estado en el que se encuentra la solicitud.
- **account:** Identifica la cuenta a la que pertenece la solicitud.

Y también se define a continuación listas con los siguientes nombres: **ListOfEmployees**, **listOfEmployeesRegistration**, **listAccounts**,

listPermissionsByEmployee que nos ayudan a realizar validaciones y construir funciones conforme se vaya desarrollando el Smart Contract.

```
function register(string memory name, string memory surname, string memory identification) public {
    require(!isEmployeeIsRegister(msg.sender), "ERR_001");
    require(!isIdentificationRegistered(identification), "ERR_002");
    Employee storage employee = ListOfEmployees[msg.sender];
    employee.name = name;
    employee.surname = surname;
    employee.identification = identification;
    employee.account = msg.sender;
    employee.amount = 1;
    listOfEmployeesRegistration[msg.sender] = true;
    listAccounts.push(msg.sender);
}

function registerAdmin(address adr, string memory name, string memory surname, string memory identification) public {
    require(!isEmployeeIsRegister(adr), "ERR_001");
    require(!isIdentificationRegistered(identification), "ERR_002");
    Employee storage employee = ListOfEmployees[adr];
    employee.name = name;
    employee.surname = surname;
    employee.identification = identification;
    employee.account = adr;
    employee.amount = 1;
    listOfEmployeesRegistration[adr] = true;
    listAccounts.push(adr);
}

function isEmployeeIsRegister(address adr) private view returns (bool){
    return listOfEmployeesRegistration[adr];
}

function isIdentificationRegistered(string memory identification) private view returns (bool){
    for (uint i = 0; i < listAccounts.length; i++) {
        Employee memory employee = ListOfEmployees[listAccounts[i]];
        if (keccak256(bytes(employee.identification)) == keccak256(bytes(identification)))
            return true;
    }
    return false;
}

function getEmployeeByAccount(address adr) public view returns (string memory, string memory, string memory, address, uint){
    require(isEmployeeIsRegister(msg.sender), "ERR_003");
    Employee memory employee = ListOfEmployees[adr];
    return (employee.name, employee.surname, employee.identification, employee.account, employee.amount);
}

function getEmployeeByIdentification(string memory identification) public view returns (string memory, string memory, string memory, address, uint){
    require(isIdentificationRegistered(identification), "ERR_004");
    for (uint i = 0; i < listAccounts.length; i++) {
        Employee memory employee = ListOfEmployees[listAccounts[i]];
        if (keccak256(bytes(employee.identification)) == keccak256(bytes(identification)))
            return (employee.name, employee.surname, employee.identification, employee.account, employee.amount);

    }
    return ("", "", "", msg.sender, 0);
}

function createPermissionByAdmin(uint256 initialDate, uint256 finalDate, string memory reason, address adr) public payable {
    require(isEmployeeIsRegister(adr), "ERR_003");
    Employee storage employee = ListOfEmployees[adr];
    require((employee.amount * (1 ether)) <= msg.value, "ERR_007");
    idOfPermissions ++;
    uint id = idOfPermissions;
    listPermissionsByEmployee[msg.sender].push(WorkPermission(id, initialDate, finalDate, reason, StateWorkPermission.Requested, adr));
    employee.amount ++;
}
```

```
    function createPermissionByUser(uint256 initialDate, uint256 finalDate, string memory reason) public payable {
        createPermissionByAdmin(initialDate, finalDate, reason, msg.sender);
    }

    function getAmount() public view returns (uint) {
        require(isEmployeeIsRegister(msg.sender), "ERR_003");
        return ListOfEmployees[msg.sender].amount;
    }

    function getAmountByAdmin(address adr) public view returns (uint) {
        require(isEmployeeIsRegister(adr), "ERR_003");
        return ListOfEmployees[adr].amount;
    }

    function getPermissionByAccount(address adr) public view returns (WorkPermission[] memory){
        require(isEmployeeIsRegister(adr), "ERR_003");
        return listPermissionsByEmployee[adr];
    }

    function getPermission() public view returns (WorkPermission[] memory){
        require(isEmployeeIsRegister(msg.sender), "ERR_003");
        return listPermissionsByEmployee[msg.sender];
    }

    function getAllPermission() public onlyOwner view returns (WorkPermission[] memory) {
        return listPermissionsByEmployee[msg.sender];
    }

    function changeStatePermissionAdmin(uint id, address adr, StateWorkPermission state) public view returns (bool) {
        require(isEmployeeIsRegister(msg.sender), "ERR_003");
        WorkPermission[] memory list = listPermissionsByEmployee[adr];
        for (uint i = 0; i < list.length; i++) {
            WorkPermission memory permission = list[i];
            if (permission.id == id) {
                permission.state = state;
                return true;
            }
        }
        return false;
    }

    function changeStatePermission(uint id, StateWorkPermission state) public view returns (bool) {
        require(isEmployeeIsRegister(msg.sender), "ERR_003");
        WorkPermission[] memory list = listPermissionsByEmployee[msg.sender];
        for (uint i = 0; i < list.length; i++) {
            WorkPermission memory permission = list[i];
            if (permission.id == id) {
                permission.state = state;
                return true;
            }
        }
        return false;
    }



    modifier onlyOwner() {
        require((msg.sender == owner), "ERR_005");
        _;
    }

    function changeOwner(address newOwner) public onlyOwner {
        owner = newOwner;
    }

}
```

Una vez finalizado el desarrollado y desplegado el Smart Contract en la testnet de Ethereum mediante la API de infura, es posible acceder a la información mediante Etherscan3, la cual mediante su interfaz nos permite buscar con la dirección del contrato toda la información perteneciente al Smart Contract.

3.13.12 Diseño de la Aplicación Web

3.13.12.1 Frontend

Para el desarrollo frontend de la aplicación propuesta se decidió utilizar la biblioteca de JavaScript, React la cual nos permite realizar interfaces de usuario y combinar con frameworks tales como Bootstrap, además se realizó un mockup de cómo se va a diseñar la aplicación el cual se detalla en las siguientes páginas.

En primer lugar, el sistema al iniciar muestra la página principal y automáticamente detecta si se tiene instalado la billerera Metamask, en caso de ya tener instalado se pide que inicie sesión en Metamask y se cuente con ether para poder llevar a cabo las transacciones, en caso de detectar que no se tiene instalado en el navegador la billetera antes mencionada se habilita un botón donde le avisa de donde debe descargarse, en la página se muestra información de herramientas que se implementó en el desarrollo del proyecto así como de la Blockchain de Ethereum, en la parte superior derecha donde dice Smart Contract, se despliega el menú el cual contiene dos opciones la primera nos permite listar los usuarios en el caso de tener el perfil de administrador y la segunda opción es el poder registrarnos

3 Etherscan se define en la sección 2.13.4

con Metamask y en el caso de ya estar registrados se puede acceder al perfil para poder solicitar los permisos, además de revisar los permisos que se tienen.

Fig 20. Interfaces que se muestran al no tener instalado la billetera Metamask en el navegador. Fuente (Autores)

Como se puede apreciar en los bocetos de interfaces, si se desea acceder a cualquier función de la aplicación tanto en el perfil de usuario como de administrador, se pide que se instale una billetera en caso de no contar, ya que la aplicación necesita para poder interactuar con la Blockchain, se le notifica mediante un mensaje que instale y que la actividad solicitada no ha podido realizarse.

Fig 21. Interfaz con el perfil de usuario. Fuente (Autores)

En los bocetos anteriores se puede apreciar como seria las interfaces con el perfil de usuario, donde al acceder desde el menú con la opción “Login con Metamask”, esta nos redirige primero a la ventana con sus respectivos campos para registrarse como usuario, una vez ya registrado se visualiza una ventana donde se puede llevar a cabo el registro de permiso o ausentismo, en dicha ventana se cargan automáticamente los datos del perfil y se habilita los campos de selección de fecha y motivo, donde una vez confirmado el registro se redirige a la siguiente interfaz donde se visualiza el reporte con el total de registros que se han llevan a cabo.

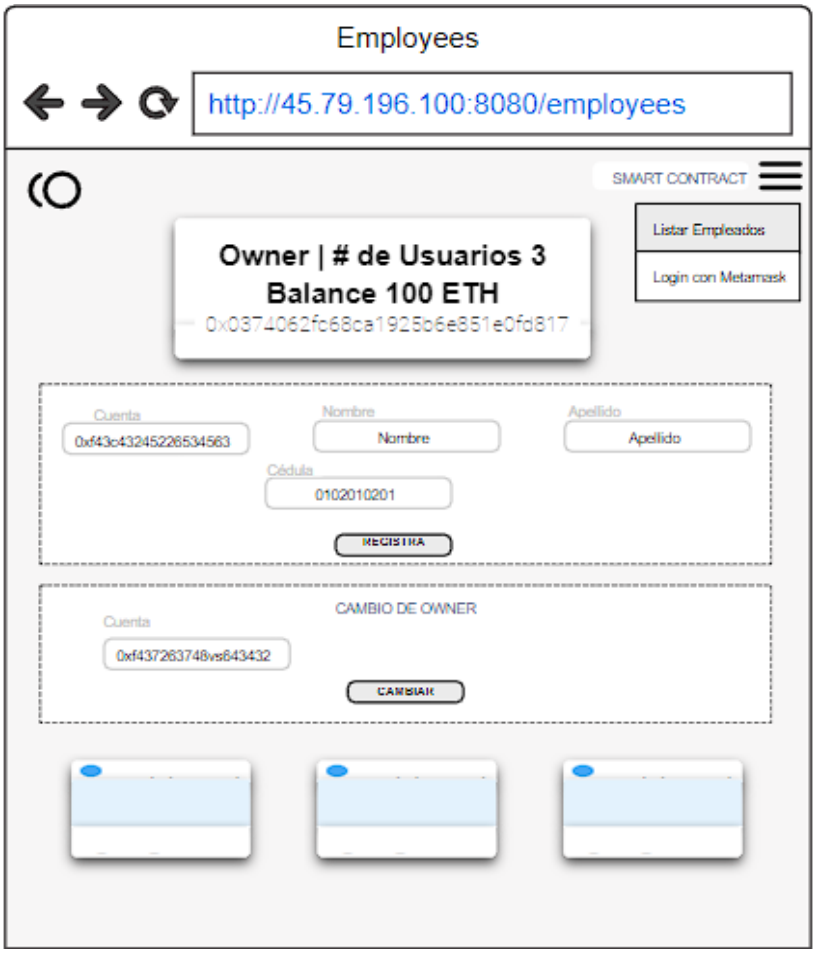


Fig 22. Interfaz de administrador. Fuente (Autores)

En el boceto anterior se aprecia, la ventana de administrador que se puede acceder mediante la opción de "**Listar Empleados**", dicha ventana contiene el número de usuarios, el balance y la dirección pública del administrador, mediante un botón se puede habilitar la opción de registrar nuevos usuarios, además de tener una opción que permita cambiar el owner(administrador) mediante la dirección pública haciendo que automáticamente la cuenta anterior quede deshabilitada y por último se tiene listado a todos los usuarios a los cuales el administrador puede acceder al reporte de permisos de cada usuario y también llegar a aprobar o negar los permisos o ausentismos como también puede registrar permisos y ausentismos por usuario.

3.13.12.2 Backend

Para el desarrollo en la parte del backend, se hizo uso de JavaScript, Web3 y Solidity, en la que JavaScript realiza peticiones al Smart Contract y a su vez Web3, nos ayuda a enviar la información al Frontend.

En cuanto al desarrollo del Smart Contract se usó el framework Truffle, debido a que esta herramienta nos ayuda con el desarrollo, compilación y despliegue de Smart Contracts en las redes de Ethereum.

Además, Truffle posee una opción mediante la configuración de un archivo el cual nos permite hacer pruebas localmente con Ganache y posterior poder configurar una red Testnet de Ethereum como lo es Rinkeby y desplegar el Smart Contract en dicha red a través del api que nos proporciona dicha red.

3.13.13 Despliegue en la red Testnet Rinkeby.

Como etapa final de nuestra aplicación se realiza el despliegue del Smart Contract en una red Testnet, en nuestro caso se implementó la red denominada Rinkeby, la cual permite revisar las transacciones, mediante etherscan, para poder desplegar es necesario hacer uso de un api que nos proporciona infura, la cual nos permite interactuar con la cadena de bloques de Ethereum.

Es necesario que se definan ciertos parámetros tales como el wallet proveedor el cual recibe como parámetro el **mnemonic** que es la frase secreta otorgada por la billetera y el parámetro **providerOrurl,** donde se coloca el api de infura, también el **numberOfAddresses**, que es **1**, el shareNonce que se hablita con true y por último el **network_id**, que de rinkeby es el **4**.

```
    rinkeby: {
      provider: () =>
        new HDWalletProvider({
          mnemonic: {
            phrase: mnemonic
          },
          providerOrUrl: 'https://rinkeby.infura.io/v3/' +
'70314c95bbeb4aaea6ac2bf43c254e08',
          numberOfAddresses: 1,
          shareNonce: true,
        }),
      network_id: '4',
    }
```

3.13.14 Validación y Pruebas

Mediante el framework Truffle, se puede llegar a realizar test que nos ayuden a validar el correcto funcionamiento del Smart Contract antes de ser desplegado, para este proyecto se realizó 5 tipos de validaciones.

- **Is deployed:**

  Dicha validación nos permite ver si el Smart Contract esta desplegado correctamente.

- **Register and Get by accounts:**

  En esta validación, se revisa si el registro del usuario se está realizando de manera correcta mediante el Smart Contract, a la vez que se obtiene el usuario por la cuenta a la que está asociada.

- **Not Register by account**

  Nos ayuda a asegurarnos que el momento de registrar a un usuario este no se registra con una cuenta que ya está asociada a otro usuario.

- **Not Register by identifiaction**

  En esta prueba se llega a validar que, al momento de registrar a los usuarios, no se registre con la identificación errónea o una identificación que ya ha sido registrada anteriormente.

- **Register by Admin**

  Se valida que el que tenga el rol de administrador, es quien tenga todos los permisos para realizar validaciones

```
PS C:\Users\Juan\Documents\GitHub\SmartContract> truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Compiling .\contracts\ThesisContract.sol
> Compiling .\contracts\UsersContracts.sol
> Artifacts written to C:\Users\Juan\AppData\Local\Temp\test--24048-FARgbVReA8HQ
> Compiled successfully using:
   - solc: 0.8.6+commit.11564f7e.Emscripten.clang



  Contract: Thesis Contract
    √ Is deployed
    √ Register and Get by accounts (1418ms)
    √ Not Register by account (2798ms)
    √ Not Register by identification (967ms)
    √ Register by Admin (1441ms)


  5 passing (8s)
```

Fig 23. Test de validación de funciones del Smart Contract. Fuente (Autores)

CAPÍTULO IV

ANÁLISIS DE RESULTADOS

Para la presente tesis, se desarrolló una aplicación con dos roles, uno como administrador y adicionalmente uno de usuario, para acceder a los distintos perfiles, solo es necesario inicializar nuestra cuenta con la wallet de Metamask, cabe recalcar las cuentas deben estar vinculas y contar con ethers para realizar las distintas operaciones.

4.1 Conexión y selección de la cuenta de Metamask

En primer lugar, se debe tener instalado la extensión de Metamask, en el navegador y autorizar a dicha wallet que pueda interactuar con la página o aplicación en la que se encuentra desplegada el Smart Contract.

Cave recalcar que para poder realizar operaciones dentro la red de pruebas de la blockchain Ethereum se necesita gestionar Ether de prueba, el cual después nos permite ejecutar las funciones dentro de la aplicación.

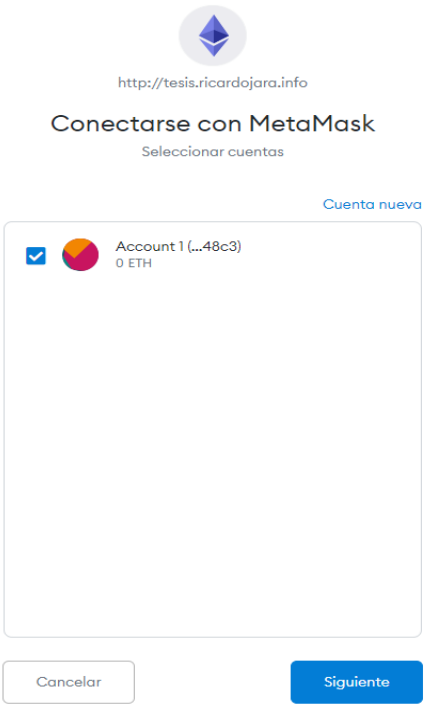


Fig 24. Selección de la cuenta en Metamask. Fuente (Autores)

En caso de no tener instalada la extensión en el navegador no podemos utilizar la aplicación, y si se accede al menú para tratar de registrase nos mostrara un mensaje en ambos perfiles obteniendo los siguientes resultados.

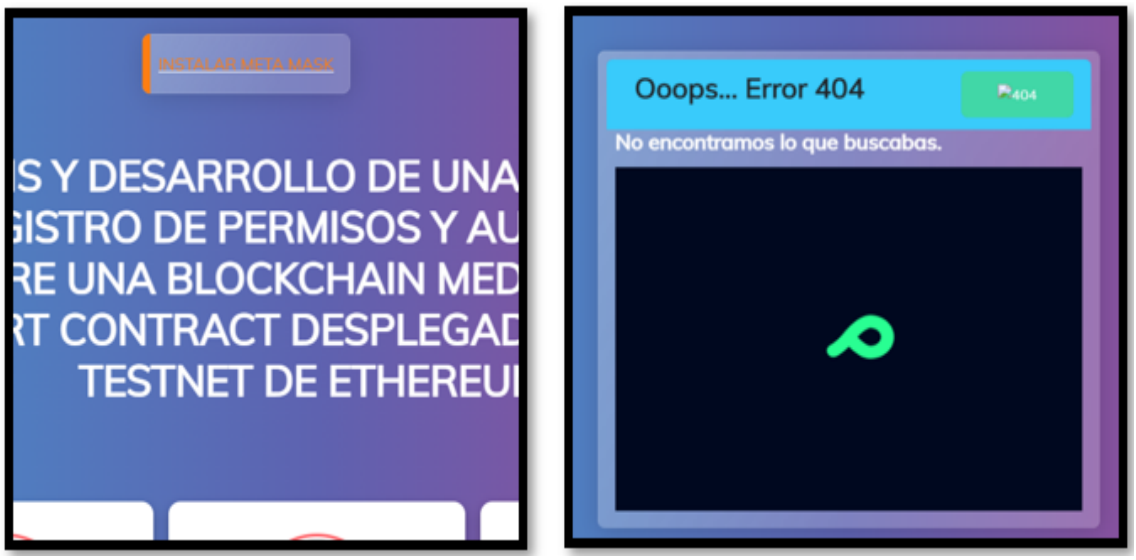


Fig 25. Aviso al usuario cuando no se tiene instalado Metamask. Fuente (Autores)

4.2 Registro de usuarios con Metamask.

Con la extensión instalada y con las cuentas vinculadas, podemos acceder al perfil de usuario, primero debemos seleccionar en el menú, “Login con metamask” y si la cuenta no está registrada obtendremos la siguiente ventana, donde se podrá ingresar los datos y al aceptar se registrará, dicha transacción nos costará un porcentaje de gas, posterior a esto se podrá obtener los siguientes resultados:

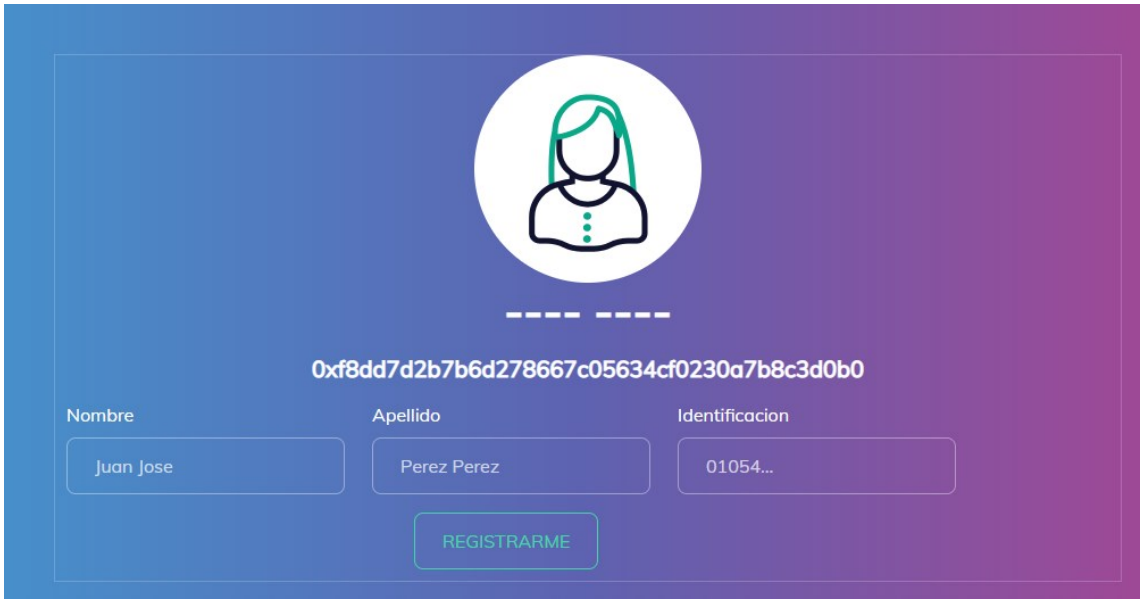


Fig 26. Ventana de registro de usuarios. Fuente (Autores)

1. Si la transacción se realizó con éxito, a continuación, se activará un menú para poder generar y ver los permisos.
2. La cuenta ya ha sido vinculada. Error que cancela la transacción ya que la cuenta ha sido registrada, cabe destacar que si ya ha sido registrada no se mostrara el menú para registrar, se redirige automáticamente al interfaz de un usuario registrado.
3. La identificación ya ha sido registrada, este error salta cuando la identificación ingresada ya existe en otra cuenta. Para continuar con el registro se necesita ingresar otra identificación.

Cuando el proceso de registro se ha completado, se refresca automáticamente la ventana y se muestra los datos con los que se ha registrado la cuenta, también la cantidad de Ether que tiene disponible la cuenta para realizar las transacciones y adicional a esto contara con dos botones para poder ver el reporte de los permisos o ausentismos registrados y crear un nuevo permiso.

A continuación, el perfil de usuario registrado y las opciones que posee cada usuario:

Fig 27. Ventana de usuario registrado. Fuente (Autores)

4.3 Creación del permiso

Cuando se dé clic en el botón de **GENERAR PERMISO**, se abrirá una ventana donde se tiene que ingresar la fecha de inicio y la fecha final del permiso, asimismo se debe ingresar el motivo.

Fig 28. Ventana para generar permisos. Fuente (Autores)

Cada permiso tiene un costo en Ethers, esto se define por el número de permisos que tiene el usuario. El primer permiso costará 1 dólar[4], el segundo permiso 2 dólares y así sucesivamente, para confirmar la transacción debemos hacer uso de la extensión de Metamask donde podemos aprobar o cancelar la misma, dicha aplicación además nos muestra la cantidad y precio de gas que cuesta las transacciones en la aplicación. Una vez la transacción haya sido minada, la ventana de Metamask desaparece y se actualiza la lista de permisos.

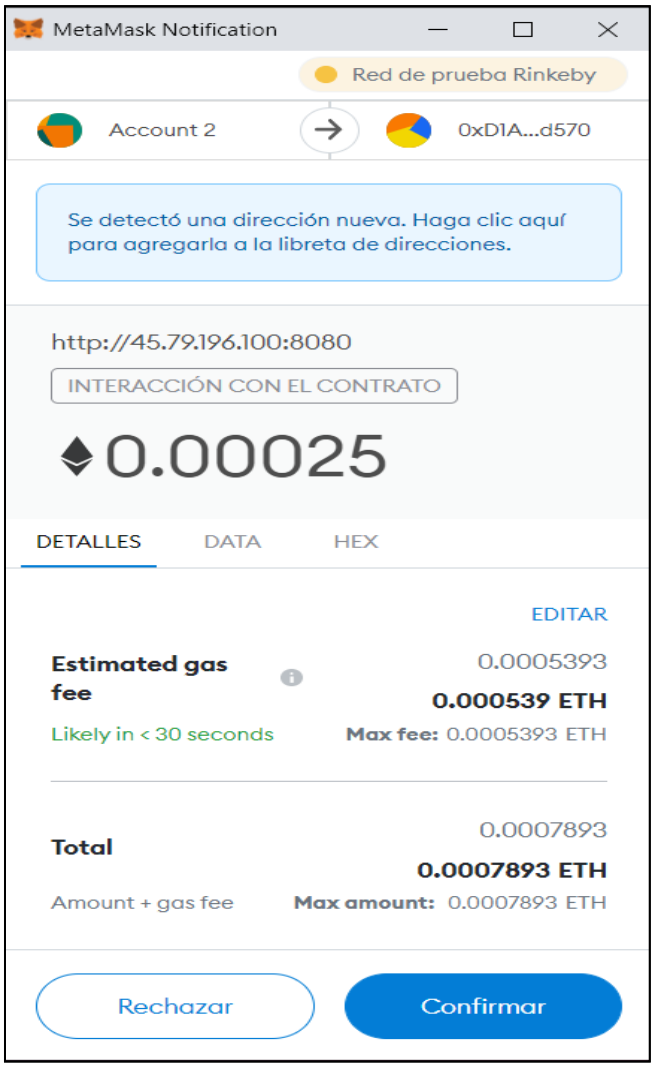


Fig 29. Ventana de Metamask, para confirmar la transacción. Fuente (Autores)

[4] A la fecha 16 de diciembre del 2021, un dólar equivale a 0.00025 ethers.

4.4 Reporte de permisos y ausentismos

Dando Clic en el botón de **MIS PERMISOS**, veremos lista detallada con la información necesaria de cada permiso. Así mismo se mostrar las acciones que puede realizar por cada permiso. En el perfil de usuario puede cancelar la aplicación.

Por cada permiso cancelado se le devuelve la mitad del precio que se pagó por cada permiso. Es decir, si se canceló un permiso que llego a costar 4 dólares, el Smart Contract le devolverá 2 dólares.

Fig 30. Ventana con el reporte de permisos y ausentismos con el perfil usuario. Fuente (Autores)

4.5 Perfil de administrador

Para acceder al perfil de administrador, en primera instancia tiene que ser el que desplego el contrato. Y para acceder al perfil de administrador tiene que seleccionar **"Listar Empleados"**, a continuación, se muestra una imagen.

Fig 31. Ventana con el perfil de administrador. Fuente (Autores)

Donde en la cabecera de la página podemos ver el address del contrato y la cantidad de Ethers con la que dispone la cuenta que desplego el Smart Contract, dicha cuenta es la que figura como administrador dentro la aplicación. En la parte inferior encontraremos una lista con los usuarios registrados, donde podemos acceder con el botón de **generar permiso** mediante la cual nos abrirá una la venta para generar el permiso del usuario, con la diferencia que ahora los costos de generar un permiso los asumirá la cuenta del administrador.

4.6 Acceso al reporte para aprobar o negar permisos y ausentismos

Para listar los permisos del usuario desde el perfil de administrador se tiene que dar clic en el botón de listar permisos de usuario, donde se visualizará una ventana similar a la del usuario, donde ahora el administrador podrá, aceptar o denegar el permiso de cada usuario.

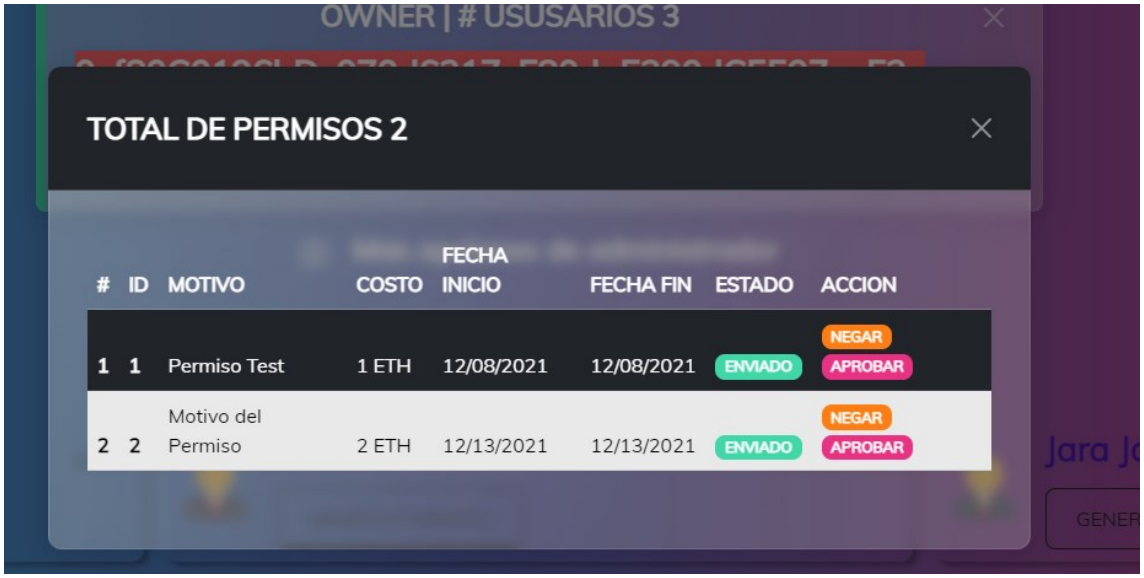


Fig 32. Ventana con el registro de permisos y ausentismos desde el perfil administrador. Fuente (Autores)

En caso de llegar a aprobar el permiso:

El permiso solo cambiará de estado y se mantendrá el mismo coste del permiso, pero en caso de que el administrador desee negar la solicitud, se le retorna la mitad del coste a la cuenta vinculada del permiso.

4.7 Función de registro desde perfil administrador

Para acceder a más características que ofrece el Smart Contract, se deberá activar el check de **Mas Opciones**, la cual nos mostrará dos funciones adicionales.

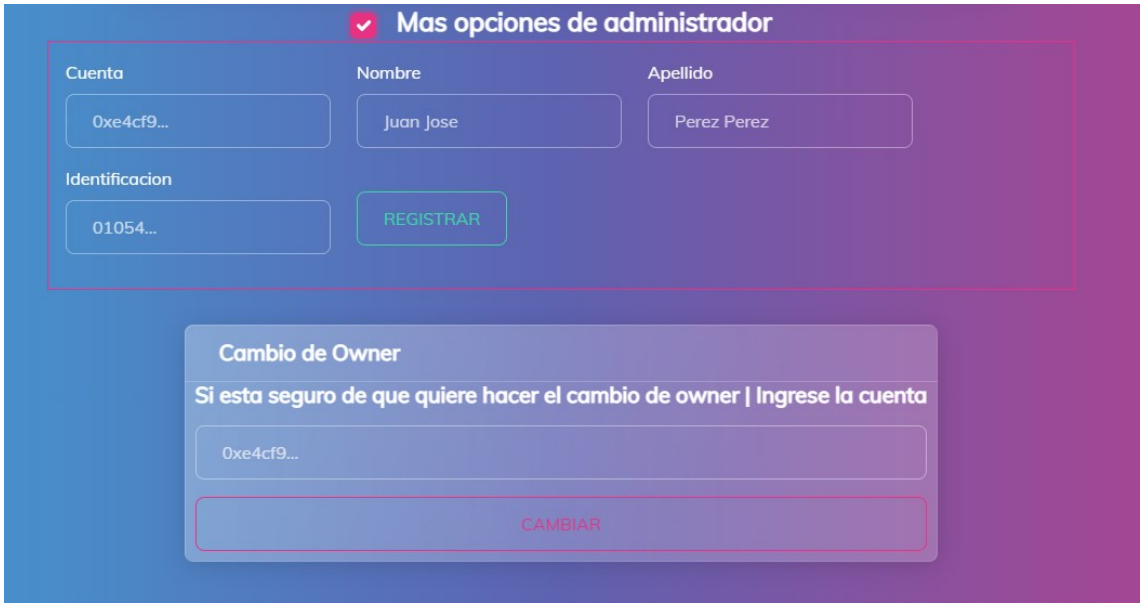


Fig 33. Ventana con las opciones de administrador. Fuente (Autores)

Como primer caso tenemos, la opción de registrar una nueva cuenta, donde se le tiene que pasar la cuenta a la que se le vinculara los datos. Se puede obtener los mismos resultados creando una cuenta independiente del administrador, en caso de estar ya registrada o mal digitada se adiciona un error por address incorrecta.

Como segundo caso tenemos, la opción de cambiar de propietario, es decir cambiar la cuenta que tendrá los permisos de administrador. Para realizar el proceso necesitamos ingresar el address de la cuenta que tendrá el permiso de administrador. Y podemos obtener los siguientes resultados.

1. La transacción se ha realizado con éxito, donde después de efectuar dicha acción se perderá los permisos de administrador y nos mostrará una página de error.
2. La cuenta es invalida. Esto retorna cuando el address de la cuenta no es correcto y es necesario ingresar una address valida.

Cade descartar, que la mayoría de gestión de permisos y perfiles se los realizar por le cuanta vinculada a Meta Mask y las cuentas que se encuentran registradas, no se hace

unos de ninguna forma de almacenamiento tradicional, como una base de datos o un fichero.

4.8 Transaccionalidad del Smart Contract.

1. **Ingresamos a la plataforma https://rinkeby.etherscan.io/.**

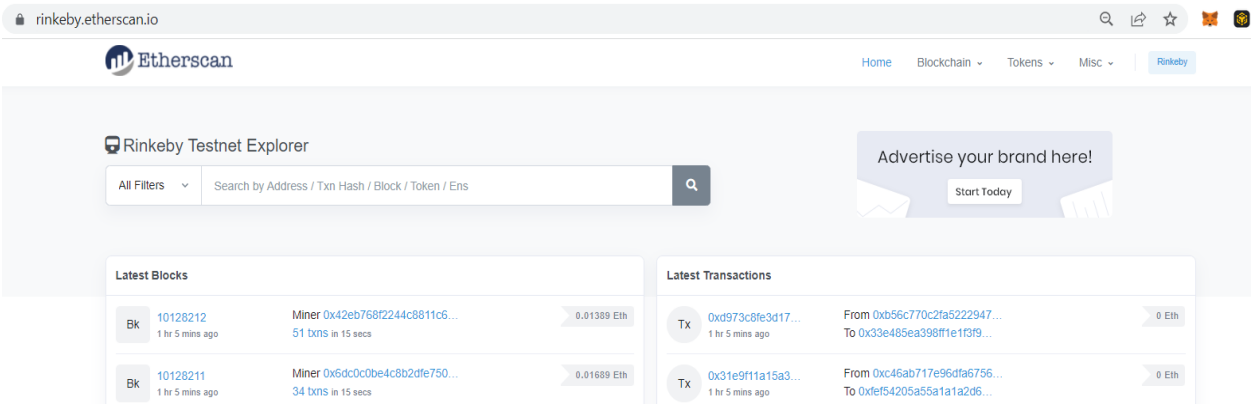


Fig 34. Ventana de acceso a rinkeby. Fuente (Autores)

2. **Copiamos la dirección de la billetera.**
    a. Cabe recalcar que se debe elegir la dirección con la que se realizó el despliegue del Smart Contract.

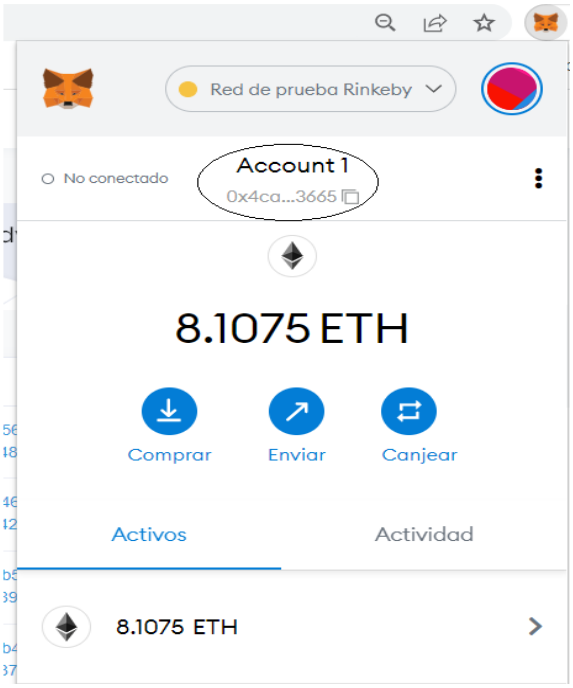


Fig 35. Dirección de la cuenta almacenada en Metamask. Fuente (Autores)

3. **Pegamos en la barra de búsqueda de Etherscan.**

   a. Aquí se puede seleccionar la dirección del contrato o también se puede ejecutar desde la línea de comandos dentro del proyecto como se observa a continuación.

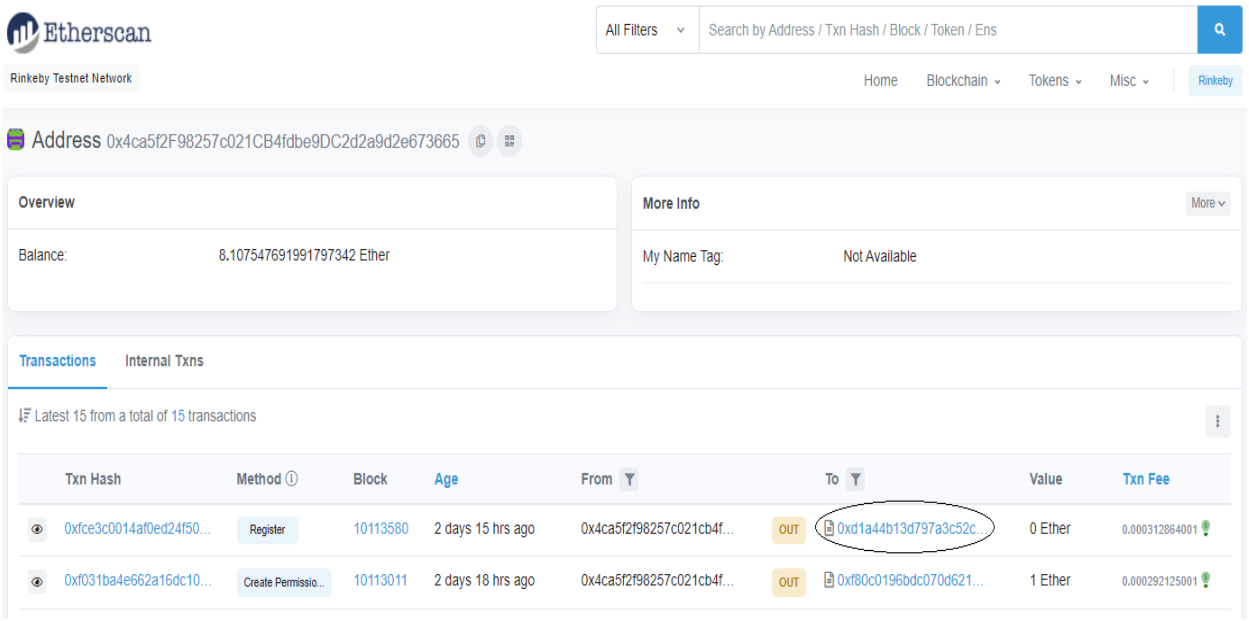


Fig 36. Dirección del Smart Contract desde Etherscan. Fuente (Autores)

   b. En este caso el siguiente comando nos retorna la dirección del contrato que ya se encuentra desplegado en la Red de pruebas llamada Rinkeby, mediante el cual podemos ver las transacciones que se han llevado a cabo desde dicha dirección del Smart Contract.

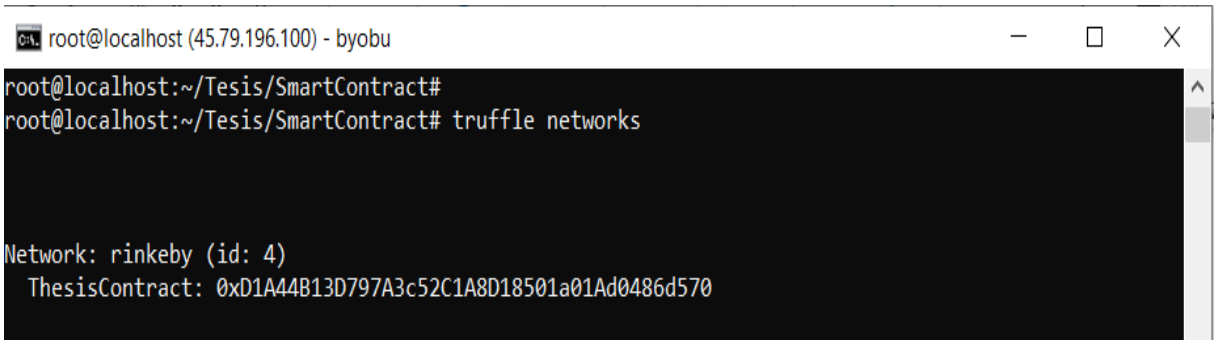


Fig 37. Dirección del Smart Contract desde truffle. Fuente (Autores)

4. **A continuación, podemos ver las transacciones realizadas en la Blockchain mediante el Smart Contract.**

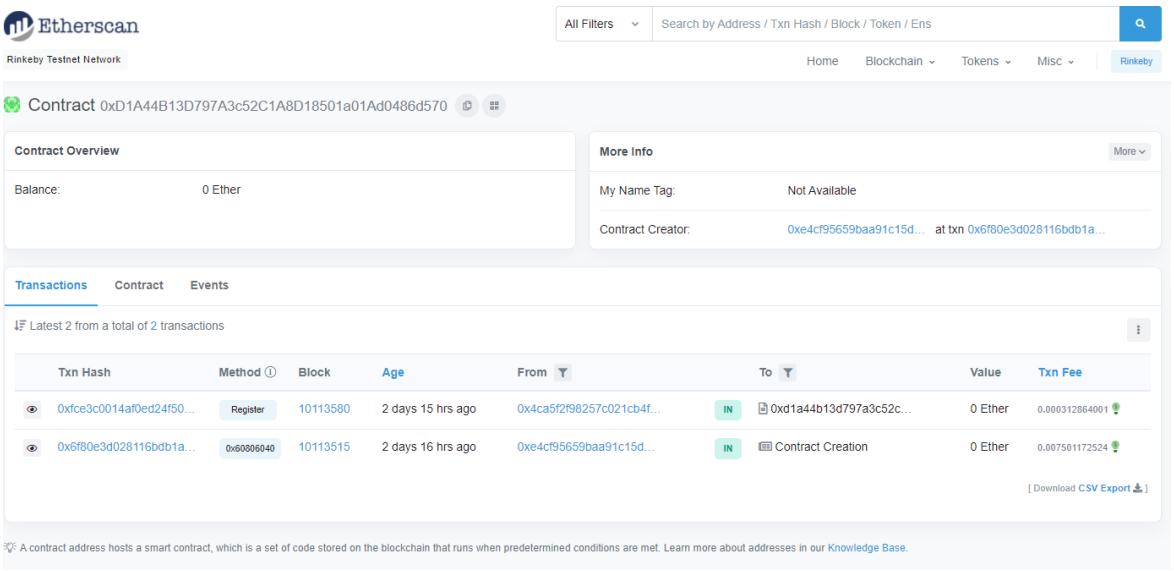


Fig 38. Transacciones asociadas al Smart Contract. Fuente (Autores)

5. **En la pestaña eventos se puede ver los registros que se ha realizado desde la aplicación mediante el Smart Contract.**

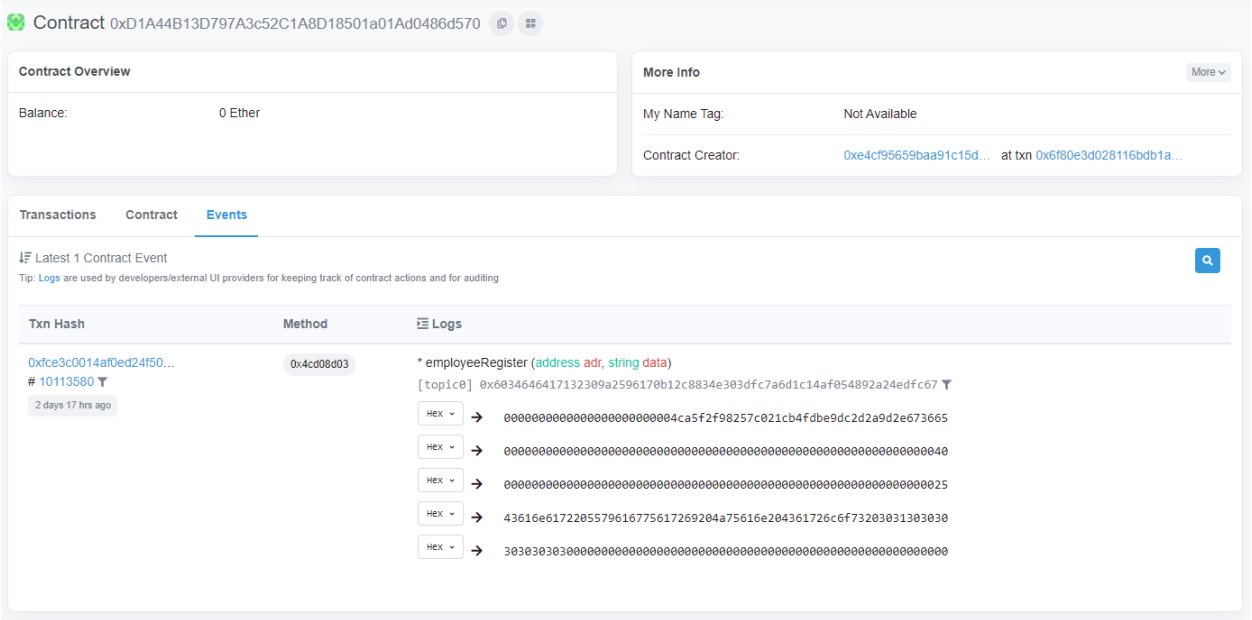


Fig 39. Eventos registrados mediante el Smart Contract. Fuente (Autores)

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1. Ethereum es una plataforma que posee una gran infraestructura, permitiendo realizar pruebas y test en su red de pruebas, lo que le convierte en una de las mejores opciones en cuanto a desarrollo de Dapps y Smart Contracts.
2. Posee herramientas compatibles con entornos Linux y Windows que nos permiten desarrollar y testear antes de ser desplegada en su red principal.
3. Mediante la implementación de Smart Contracts se logra que la información permanezca inmutable, al ser almacenada en una cadena de bloques de Ethereum.
4. Cuando desarrollamos un Smart Contract, no debemos pensar un sistema complejo o que involucre muchas variables o métodos de validación ya que esto generará un incremento en el consumo de gas y su validación será más demorada.
5. Nuestra tesis ha sido desarrollada con la finalidad de demostrar la capacidad que puede tener la tecnología Blockchain junto con los Smart Contracts para brindar seguridad a los datos que llegan a almacenarse y a su vez tener la certeza que lo que se llegó a fijar en dicho contrato entre los participantes no va a modificarse y va a llegar a cumplirse sin la necesidad de un ente intermedio.

Recomendaciones

1. Se recomienda realizar el desarrollo con las versiones estables de web3 y Solidity, debido a que poseen mejoras en cuanto a seguridad y mayores opciones de compatibilidad con nuevas tecnologías.
2. Se debe tomar en cuenta que durante el desarrollo de Smart Contracts es necesario realizar las suficientes pruebas en las redes testnet, para evitar futuros errores y problemas en el despliegue en la red principal, ya que una vez desplegado el Smart Contract no puede ser alterado.
3. La rapidez de las transacciones no es lo fuerte de los Smart Contracts en Ethreum, ya que ha parte de tener un precio de gas, también tiene que ser minadas es decir validadas, lo que lleva a tener un tiempo de demora dependiendo la cantidad de mineros que tenga la red y el algoritmo de consenso que haga uso.
4. Conocer bien el estado actual de un Ether ya que cada vez que interactuamos con la red principal se estará haciendo uso de dinero fiat.
5. Cuando necesitamos interactuar con datos exteriores y nuestro Smart Contract necesita estos datos para desplegar una función, podemos hacer uso de los oráculos, siendo este un proceso que obtiene datos de fuentes seguras de apis exteriores mediante las cuales se pueden

Bibliografía

Babel. (12 de 01 de 2021). Obtenido de Babeljs: https://babeljs.io/docs/en/

Bruno. (12 de 05 de 2017). *Ethereum Gas and Transaction Fees Explained!* Obtenido de bitfalls: https://bitfalls.com/2017/12/05/ethereum-gas-and-transaction-fees-explained/

Chen, H., Pendleton, M., Njilla, L., & Xu, S. (13 de 08 de 2019). *A Survey on Ethereum Systems Security*. Obtenido de https://scholarphi.semanticscholar.org/: https://scholarphi.semanticscholar.org/?file=https://arxiv.org/pdf/1908.04507v1.pdf

ConsenSys. (2018). *Infura* . Obtenido de Infura : https://consensys.net/blog/news/why-infura-is-the-secret-weapon-of-ethereum-infrastructure/

CSDN. (25 de 01 de 2018). *Desarrollo de Ethereum Truffle y Ganache.* Obtenido de csdn.net: https://blog.csdn.net/TurkeyCock/article/details/79165602

Ethereum. (2016). *Ethereum Virtual Machine*. Obtenido de Ethereum: https://ethdocs.org/en/latest/introduction/what-is-ethereum.html#ethereum-virtual-machine

Ethereum. (11 de 2021). *GAS AND FEES.* Obtenido de GAS AND FEES: https://ethereum.org/en/developers/docs/gas/

Ethereum. (10 de 07 de 2021). *web3js*. Obtenido de web3js: https://web3js.readthedocs.io/en/v1.3.4/getting-started.html

Ethereum, & wackrow. (2021). *Prueba de Participación.* Obtenido de PRUEBA DE PARTICIPACIÓN(POS): https://ethereum.org/es/developers/docs/consensus-mechanisms/pos/

ethereum.org. (13 de 07 de 2021). *Ethereum 2.0*. Obtenido de ethereum: https://ethereum.org/es/learn/

Ethereum.org. (19 de Agosto de 2021). *The merge.* Obtenido de The merge: https://ethereum.org/en/eth2/merge/

Etherscan. (1 de 09 de 2021). *Etherscan*. Obtenido de Etherscan: https://etherscan.io/aboutus

Haverbeke, M. (2018). *Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming*. No Starch Press.

Hiremath, O. S. (22 de 03 de 2019). *Truffle Ethereum Tutorial – Developing Ethereum DApps with Truffle.* Obtenido de DApps with Truffle: https://www.edureka.co/blog/developing-ethereum-dapps-with-truffle

K. Christidis y M. Devetsikiotis. (2016). Blockchain and Smart Contracts for Internet of Things,. *IEEE Access*, 2-3. Obtenido de Blockchain and Smart Contracts for Internet of Things,.

Kelly, R. M. (10 de Sep de 2021). *What is Ethereum 2.0 and Why Does It Matter?* Obtenido de What is Ethereum 2.0 and Why Does It Matter?: https://decrypt.co/resources/what-is-ethereum-2-0

Levi, S. D., & Lipton, A. B. (26 de 05 de 2018). *An Introduction to Smart Contracts and Their Potential and Inherent Limitations.* Obtenido de Harvard: https://corpgov.law.harvard.edu/2018/05/26/an-introduction-to-smart-contracts-and-their-potential-and-inherent-limitations/#:~:text=How%20Smart%20Contracts%20Function,on%20a%20blockchain%2Dbased%20platform.&text=That%20replication%20also%20means%20that,is

Martínez, A. A. (2020). *Blockchain, algoritmos de consenso.* Obtenido de Blockchain, algoritmos de consenso: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/127926/6/aamoresmTFM1220memoria.pdf

MetaMask. (27 de 01 de 2021). *MetaMask's Developer Documentation*. Obtenido de MetaMask's: https://docs.metamask.io/guide/

Modi, R. (2018). *Solidity Programming Essentials.* Obtenido de Packt: https://subscription.packtpub.com/book/application_development/9781788831383/1/ch01lvl1sec11/why-blockchains

Modi, R. (2018). *Solidity Programming Essentials: A beginner's guide to build smart contracts for Ethereum and blockchain (English Edition).* Packt Publishing.

National Institute of Standards and Technology. (2015). *csrc.nist.gov.* Obtenido de SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions: https://csrc.nist.gov/csrc/media/publications/fips/202/final/documents/fips_202_draft.pdf

Node.js. (01 de 7 de 2020). *Node.js*. Obtenido de Node.js: https://nodejs.org/es/about/:

Oh, J., & Shong, I. (12 de 04 de 2017). *A case study on business model innovations using Blockchain: focusing on financial institutions.* Obtenido de Researchgate : https://www.researchgate.net/publication/321809248_A_case_study_on_business_model_innovations_using_Blockchain_focusing_on_financial_institutions

Palacios, V. M. (06 de 02 de 2018). *Explorando la Blockchain de Ethereum y el desarrollo de.* Obtenido de upc: https://upcommons.upc.edu/bitstream/handle/2117/127784/memoria.pdf?sequence=1&isAllowed=y

Pérez, J. E. (25 de marzo de 2009 y). *Introducción a JavaScript.* Espana: librosweb.

Ribera, E. G. (2018). *Design and implementation of a proof-of-stake consensus algorithm for blockchain.* Obtenido de Design and implementation of a proof-of-stake consensus algorithm for blockchain: https://bedan.ir/modules/Translator/PDFs/AR55.pdf

solidity. (13 de 07 de 2021). *solidity-es*. Obtenido de solidity-es: https://solidity-es.readthedocs.io/es/latest/index.html

Trufflesuite. (2021). *ganache*. Obtenido de ganache: https://www.trufflesuite.com/ganache

Viriyasitavat, W., & Hoonsopon, D. (2018). *Blockchain characteristics and consensus in modern business processes.* Obtenido de Journal of Industrial Information: https://www.sciencedirect.com/science/article/pii/S2452414X18300815

Vitalik, B. (Dic de 2020). *This is my approximate view of what the next 5-10 years of eth2 and beyond might look like.* Obtenido de This is my approximate view of what the next 5-10 years of eth2 and beyond might look like: https://twitter.com/VitalikButerin/status/1333922620857745408/photo/1

Wood, D. G. (2021). ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER*, pag 27.

WOOD, D. G. (2021). *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER.* Obtenido de ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER: https://ethereum.github.io/yellowpaper/paper.pdf

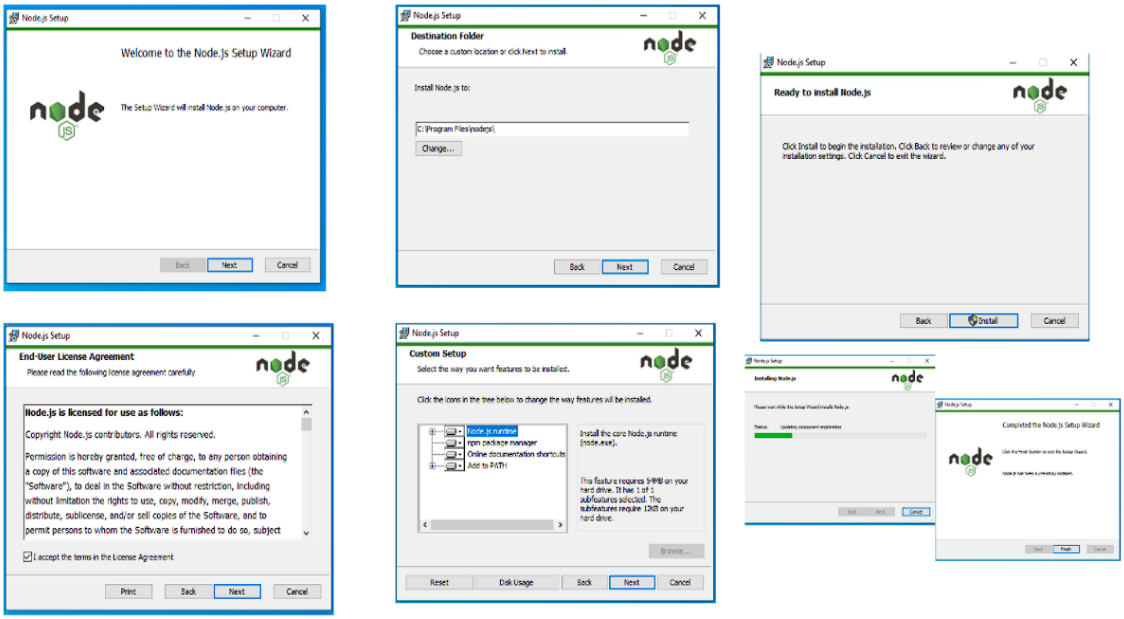
Wu, Y., Song, P., & Wang, F. (13 de 04 de 2020). *Hybrid Consensus Algorithm Optimization: A Mathematical Method Based on POS and PBFT and Its Application in Blockchain.* Obtenido de Hybrid Consensus Algorithm Optimization: A Mathematical Method Based on POS and PBFT and Its Application in Blockchain: https://doi.org/10.1155/2020/7270624

Yano, M., Dai, C., Masuda, K., & Kishimoto, Y. (2020 ed.). *Blockchain and Crypto Currency: Building a High Quality Marketplace for Crypto Data.* Springer.

Yousaf, N. C. (2018). *Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities.* Obtenido de Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8632190&isnumber=8632002

Anexos

Anexo 1. Instalación de Node.js Windows



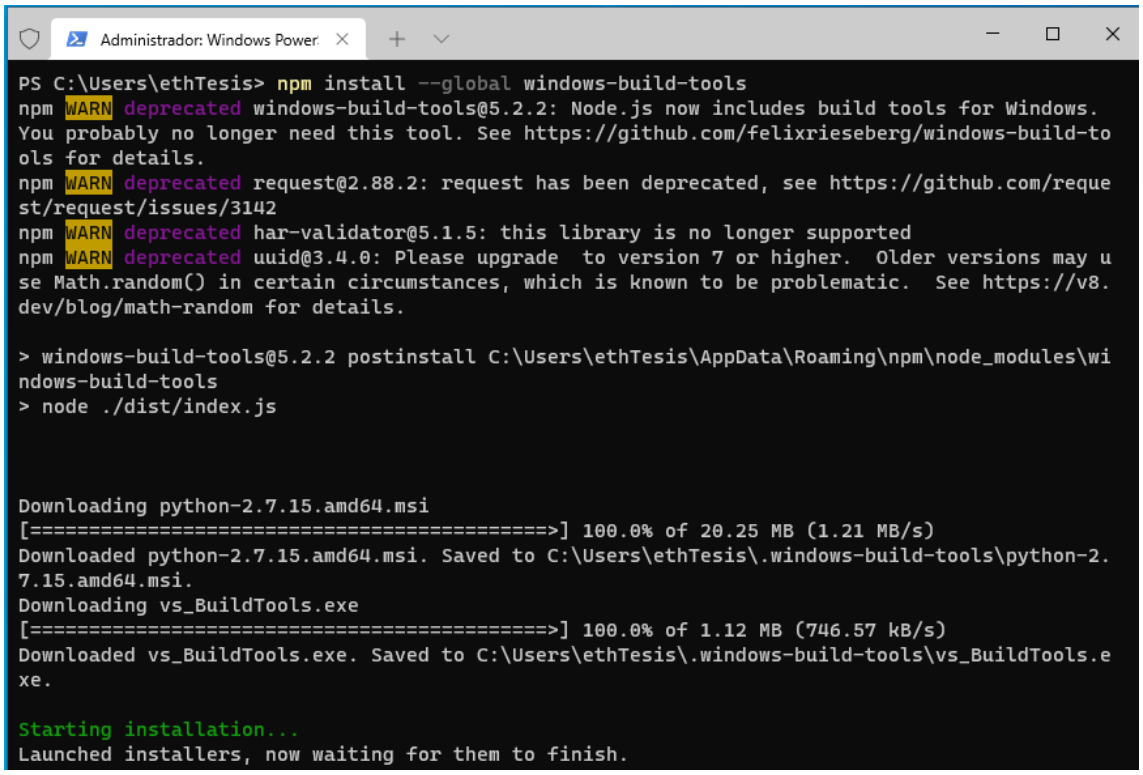
Anexo 2. Instalación de Node.js en Linux

```
ubuntu@ip-172-31-29-208:~$ node --version
v14.18.1
ubuntu@ip-172-31-29-208:~$ npm --version
6.14.15
ubuntu@ip-172-31-29-208:~$
```

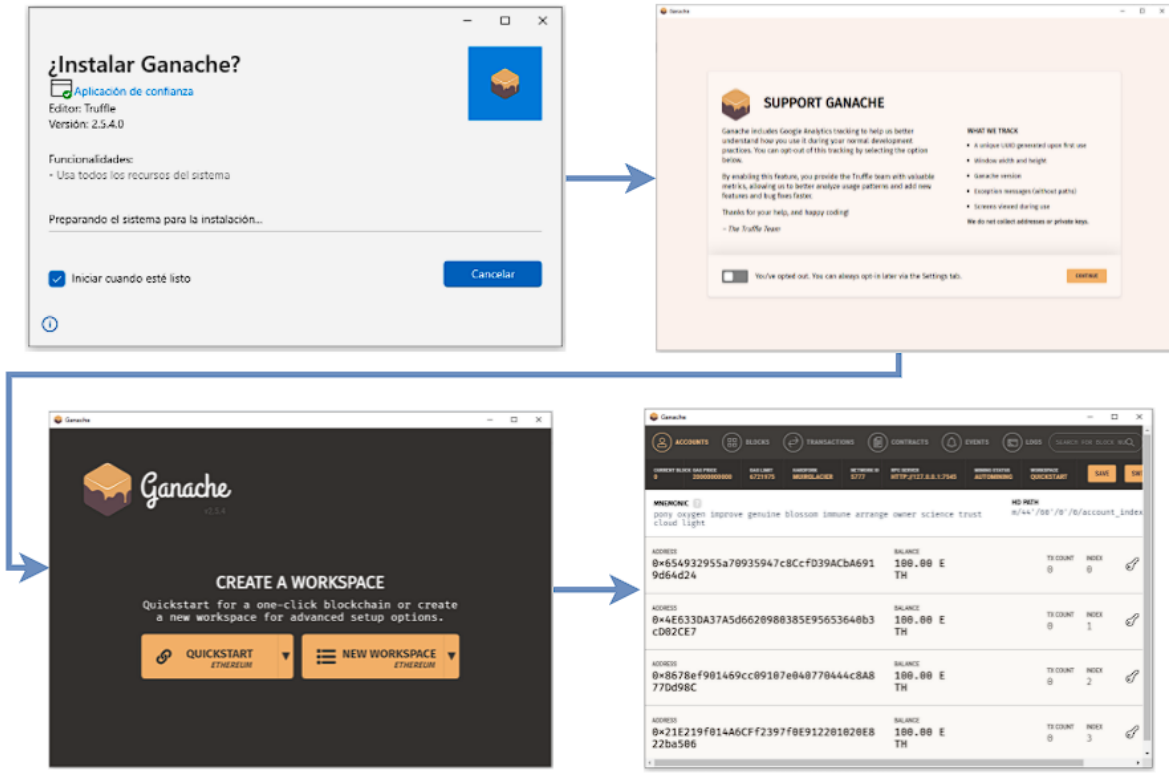
Anexo 3. Instalación de herramientas de compilación.

```
ubuntu@ip-172-31-29-208:~$ sudo apt-get update
Hit:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Hit:4 https://deb.nodesource.com/node_14.x focal InRelease
Get:5 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [1346 kB]
Get:6 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [877 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Fetched 2551 kB in 1s (2902 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-29-208:~$
ubuntu@ip-172-31-29-208:~$ sudo apt-get install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Anexo 4. Instalación en Windows.



Anexo 5. Instalación del servidor Ganache Windows & Linux.



1. **Instalación de Ganache en Linux, mediante el gestor de paquetes de npm.**

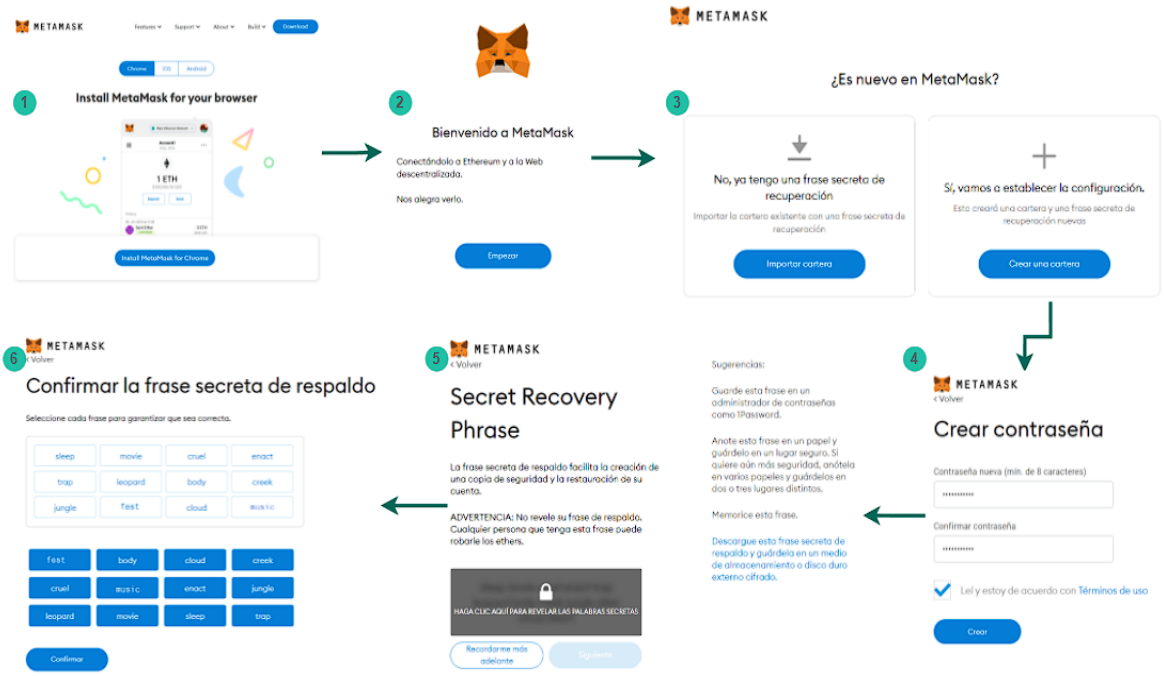
```
juan@juanc:~$ ganache-cli
ganache v7.0.0-beta.2 (@ganache/cli: 0.1.1-beta.2, @ganache/core: 0.1.1-beta.2)
Starting RPC server

Available Accounts
==================
(0) 0xB90BDe24ab4CcAb75244f32662998fCb2b8c590f (1000 ETH)
(1) 0x365bc3721D3A208Dcc1a7903ac7A64a4143AF15E (1000 ETH)
(2) 0x58f0a76bcf15fbc7609F67Ca468E319be62d4ee7 (1000 ETH)
(3) 0x4D95c6FfC814a032d870e83D99802937971Ce276 (1000 ETH)
(4) 0x1055f7E64575200de54DeF416c1a3B3Ea160599f (1000 ETH)
(5) 0xAA4C32a79D8d29Bae5A8aEdbaa479E76C27121Ef (1000 ETH)
(6) 0x884E3e0814064716cFBcDb91DCA1a1A19125C7bA (1000 ETH)
(7) 0x99a756f88f27b4F61e1e474cf7a35Da02952595D (1000 ETH)
(8) 0x8b44fC4Cde0D37eDab1DAC3C2953832E3081145B (1000 ETH)
(9) 0x0eD0836a536c419CdCd235686316880Fb14dEAA9 (1000 ETH)

Private Keys
==================
(0) 0x8c1e407d95ad1cceb513054eb89a3a984bb0856d6b5bcbf3cdd84b3e02709e62
(1) 0x50d449e3c02ce65d1e2ea5899458c2a612c6c80ae1d60709d148c78f4709c359
(2) 0x40a2103615685d291530a7792523d58b41adf95490bdfdbac37d42de4f49992e
(3) 0xc4c587ffdfbb7cbeaf1e9503c52d45fd87b4b6d3705cb30acc9a24d18dc4a300
(4) 0xac5f6eed5791771677e3d027cc680a5d035611f0f8bd55a5589d39bb326682ae
(5) 0x34fff2f889e07ac73b972756675ba25b58b028064b980df86c7385f29089801f
(6) 0x37c049f16466cfc5161ed4753f4d40feccb4c3ea5e323a48a617e1131a15a06e
(7) 0xf3826117496939053749a68405c60420afacbed1d063a2a56c9609a2d4c385f0
```

2. **Comando para arrancar en cliente de ganache en Linux.**

```
juan@juanc:~$ npm install ganache@beta --global

changed 28 packages, and audited 29 packages in 41s

found 0 vulnerabilities
juan@juanc:~$
juan@juanc:~$
juan@juanc:~$
```

Anexo 6. Instalación de Metamask.

```
HD Wallet
==================
Mnemonic:      hawk lizard slot pyramid vibrant hockey rather horn like
ugust
Base HD Path:  m/44'/60'/0'/0/{account_index}

Default Gas Price
==================
2000000000

BlockGas Limit
==================
12000000

Call Gas Limit
==================
50000000

Chain Id
==================
1337

RPC Listening on 127.0.0.1:8545
```



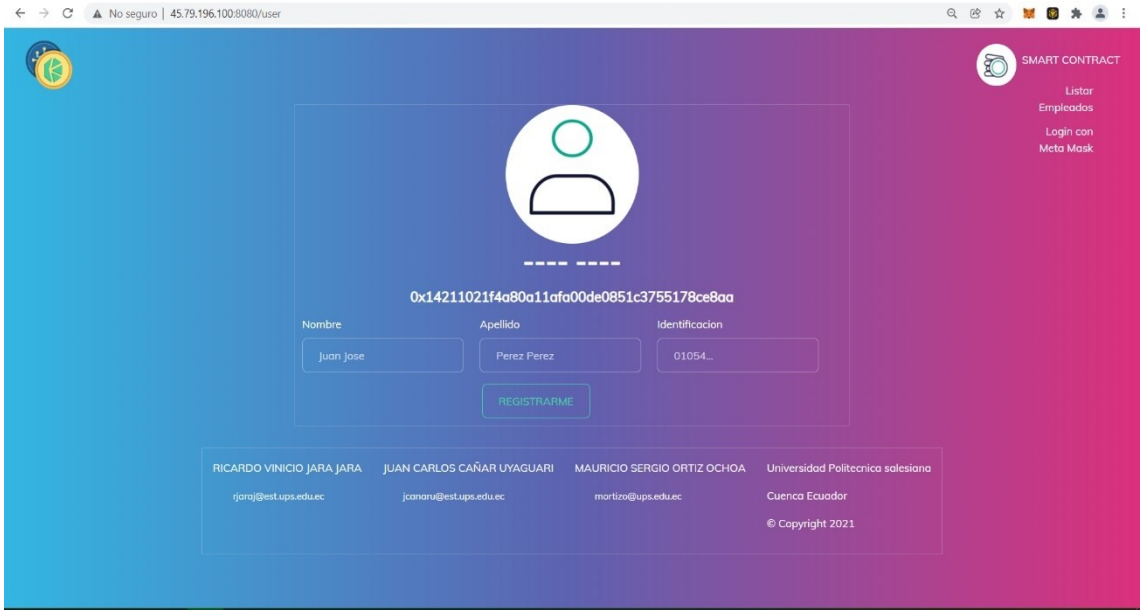
Anexo 7. Instalación de Framework Truffle.

```
Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.19042.1348]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>npm install -g truffle
npm WARN deprecated uuid@2.0.1: Please upgrade  to version 7 or higher.  Older versions may use Math.random() in certain
 circumstances, which is known to be problematic.  See https://v8.dev/blog/math-random for details.
npm WARN deprecated mkdirp-promise@5.0.1: This package is broken and no longer maintained. 'mkdirp' itself supports prom
ises now, please switch to that.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated uuid@3.4.0: Please upgrade  to version 7 or higher.  Older versions may use Math.random() in certain
 circumstances, which is known to be problematic.  See https://v8.dev/blog/math-random for details.
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated uuid@3.3.2: Please upgrade  to version 7 or higher.  Older versions may use Math.random() in certain
 circumstances, which is known to be problematic.  See https://v8.dev/blog/math-random for details.
npm WARN deprecated multicodec@0.5.7: This module has been superseded by the multiformats module
npm WARN deprecated cids@0.7.5: This module has been superseded by the multiformats module
npm WARN deprecated multicodec@1.0.4: This module has been superseded by the multiformats module
npm WARN deprecated multibase@0.6.1: This module has been superseded by the multiformats module
npm WARN deprecated multibase@0.7.0: This module has been superseded by the multiformats module
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated graphql-tools@4.0.8: This package has been deprecated and now it only exports makeExecutableSchema.\
nAnd it will no longer receive updates.\nWe recommend you to migrate to scoped packages such as @graphql-tools/schema, @
graphql-tools/utils and etc.\nCheck out https://www.graphql-tools.com to learn what package you should use instead
npm WARN deprecated apollo-cache-control@0.14.0: The functionality provided by the `apollo-cache-control` package is bui
lt in to `apollo-server-core` starting with Apollo Server 3. See https://www.apollographql.com/docs/apollo-server/migrat
ion/#cachecontrol for details.
npm WARN deprecated apollo-tracing@0.15.0: The `apollo-tracing` package is no longer part of Apollo Server 3. See https:
//www.apollographql.com/docs/apollo-server/migration/#tracing for details
npm WARN deprecated graphql-extensions@0.15.0: The `graphql-extensions` API has been removed from Apollo Server 3. Use t
he plugin API instead: https://www.apollographql.com/docs/apollo-server/integrations/plugins/
npm WARN deprecated mkdirp@0.5.1: Legacy versions of mkdirp are no longer supported. Please update to mkdirp 1.x. (Note
that the API surface has changed to use Promises in 1.x.)
npm WARN deprecated uuid@3.2.1: Please upgrade  to version 7 or higher.  Older versions may use Math.random() in certain
 circumstances, which is known to be problematic.  See https://v8.dev/blog/math-random for details.
npm WARN deprecated node-pre-gyp@0.11.0: Please upgrade to @mapbox/node-pre-gyp: the non-scoped node-pre-gyp package is
deprecated and only the @mapbox scoped package will recieve updates in the future
npm WARN deprecated redux-devtools-core@0.2.1: Package moved to @redux-devtools/app.
npm WARN deprecated redux-devtools-instrument@1.10.0: Package moved to @redux-devtools/instrument.
npm WARN deprecated querystring@0.2.1: The querystring API is considered Legacy. new code should use the URLSearchParams
 API instead.
npm WARN deprecated remotedev-serialize@0.1.9: Package moved to @redux-devtools/serialize.
npm WARN deprecated querystring@0.2.0: The querystring API is considered Legacy. new code should use the URLSearchParams
 API instead.
npm WARN deprecated core-js@2.6.12: core-js@<3.3 is no longer maintained and not recommended for usage due to the number
 of issues. Because of the V8 engine whims, feature detection in old core-js versions could cause a slowdown up to 100x
even if nothing is polyfilled. Please, upgrade your dependencies to the actual version of core-js.
npm WARN deprecated cids@1.1.9: This module has been superseded by the multiformats module
npm WARN deprecated multibase@3.1.2: This module has been superseded by the multiformats module
```

```
> core-js@2.6.12 postinstall C:\Users\ethTesis\AppData\Roaming\npm\node_modules\truffle\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)


> core-js-pure@3.20.1 postinstall C:\Users\ethTesis\AppData\Roaming\npm\node_modules\truffle\node_modules\core-js-pure
> node -e "try{require('./postinstall')}catch(e){}"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js:
> https://opencollective.com/core-js
> https://patreon.com/zloirock
> https://paypal.me/zloirock
> bitcoin: bc1qlea7544qtsmj2rayg0lthvza9fau63ux0fstcz

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)


> truffle@5.4.26 postinstall C:\Users\ethTesis\AppData\Roaming\npm\node_modules\truffle
> node ./scripts/postinstall.js

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: @zondax/filecoin-signing-tools@github:Digital-MOB-Filecoin/filecoin-sign
ing-tools-js (node_modules\truffle\node_modules\filecoin.js\node_modules\@zondax\filecoin-signing-tools):
npm WARN enoent SKIPPING OPTIONAL DEPENDENCY: Error while executing:
npm WARN enoent SKIPPING OPTIONAL DEPENDENCY: undefined ls-remote -h -t ssh://git@github.com/Digital-MOB-Filecoin/fileco
in-signing-tools-js.git
npm WARN enoent SKIPPING OPTIONAL DEPENDENCY:
npm WARN enoent SKIPPING OPTIONAL DEPENDENCY:
npm WARN enoent SKIPPING OPTIONAL DEPENDENCY: spawn git ENOENT
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.1.2 (node_modules\truffle\node_modules\chokidar\node_modules
\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: sqlite3@4.2.0 (node_modules\truffle\node_modules\sqlite3):
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: sqlite3@4.2.0 install: `node-pre-gyp install --fallback-to-build`
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: Exit status 1

+ truffle@5.4.26
added 22 packages from 16 contributors, removed 809 packages, updated 70 packages and moved 2 packages in 611.615s
```

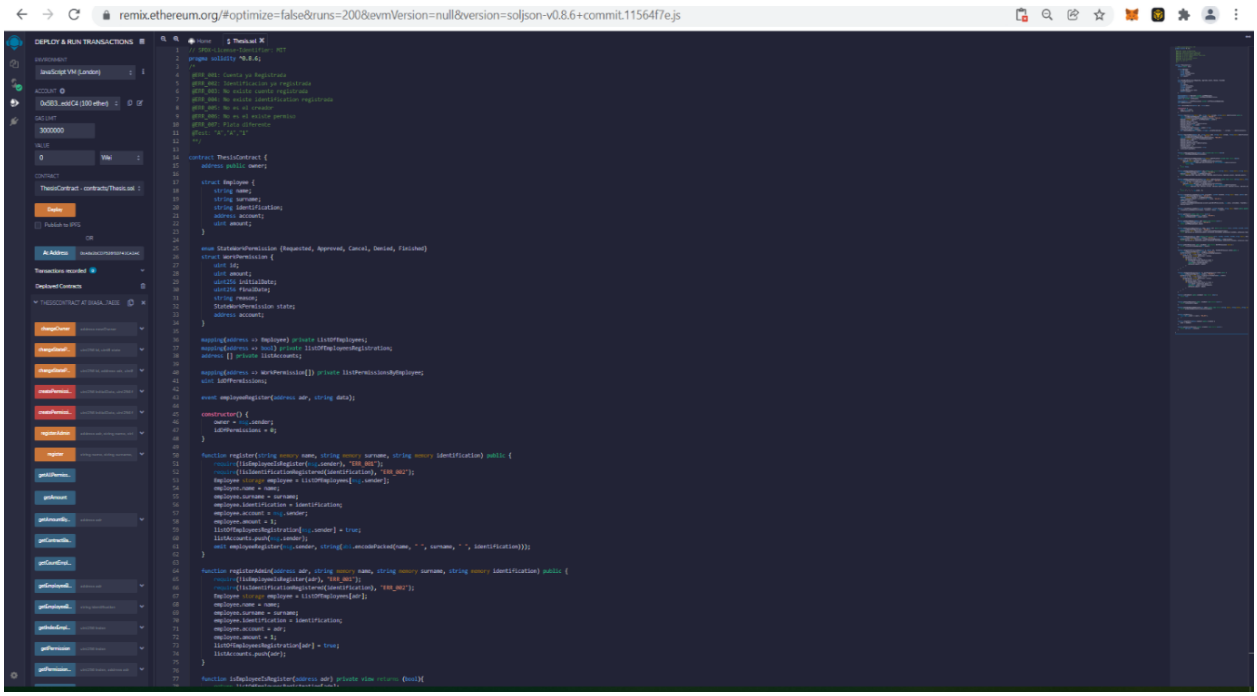
Anexo 8. Aplicación desplegada en un servidor en la nube.

1. Registro de usuario.



2. **Interfaz de administrador.**

Anexo 9. Testing en Ethereum Remix.



Anexo 10. Reporte

Cuenca - Ecuador

DETALLE DE REGISTRO

Nombres: Juan Perez

Identificación: 0100000000

Cuenta: 0x6b325dd6095549b7ad079d02f8edf5cc03012cab

#	Motivo	Fecha Ini	Fecha Fin	Estado
1	Permiso por calamidad domØstica de 3 meses.	12/25/2021	03/26/2022	ENVIADO
2	Ausentismo por cita medica y reposo por cirugía.	08/25/2021	09/25/2021	ENVIADO

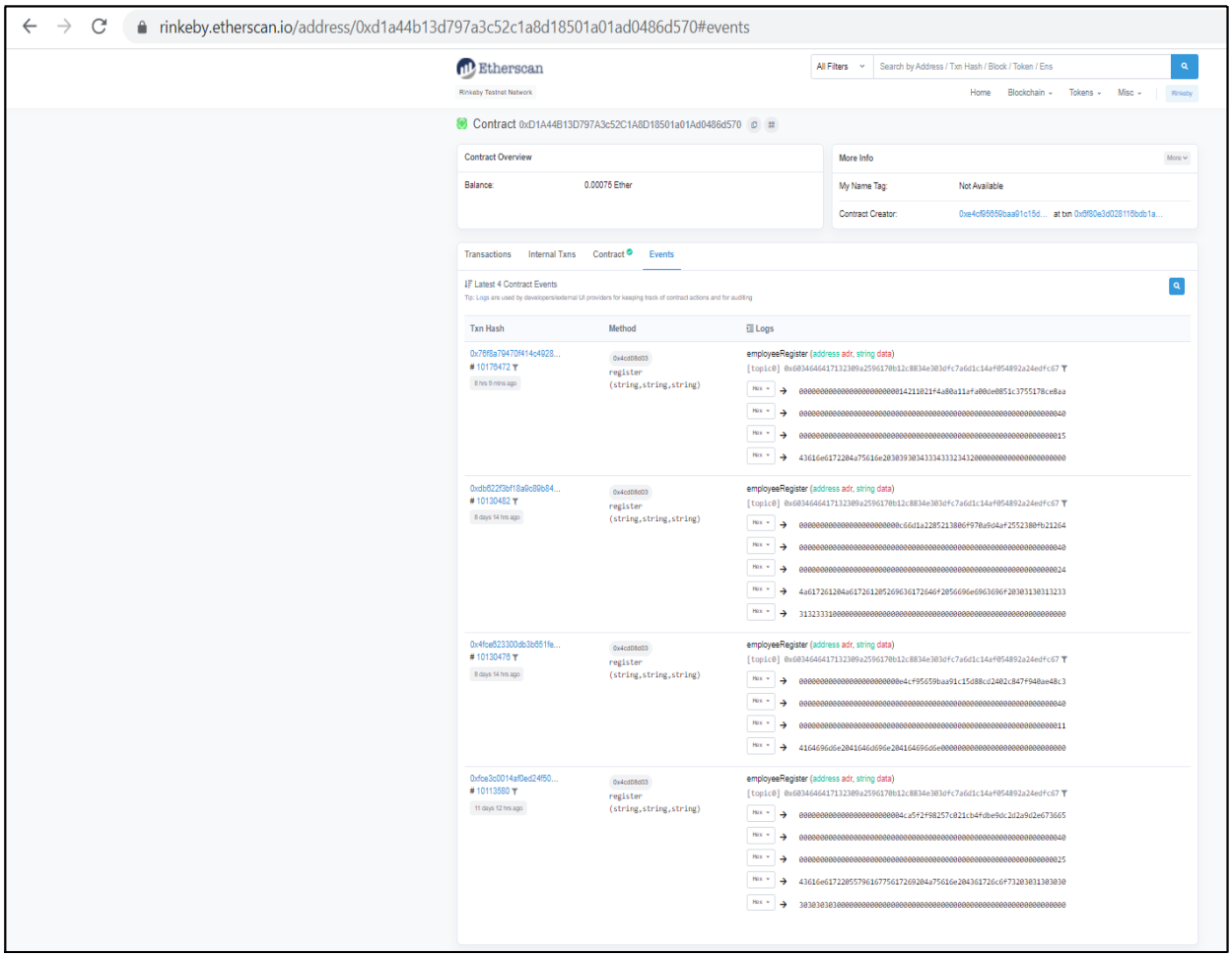
Cuenca - Ecuador

DETALLE DE REGISTRO

Nombres: Juan Perez

Identificación: 0904343242

Cuenta: 0xdf84ffbb0b413b824430aa29baaf760e4d577029

#	Motivo	Fecha Ini	Fecha Fin	Estado
1	Ausentismo por cita medica y reposo por cirugía.	12/29/2021	12/29/2021	Aprobado
2	Permiso por calamidad domØstica de 3 meses.	12/29/2021	12/29/2021	Negado
3	Comprasss IT	12/29/2021	12/29/2021	Enviado

Anexo 11. Transaccionalidad del Smart Contract.