UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UN PROTOTIPO DE UNA CRIPTOMONEDA CON HERRAMIENTAS DE CODIGO ABIERTO BASADO EN ETHEREUM

Trabajo de titulación previo a la obtención del

Título de Ingeniero de Sistemas

AUTOR: BRYAN FRANCISCO OCAÑA VALDEZ

TUTOR: JULIO RICARDO PROAÑO ORELLANA

Quito – Ecuador

2024

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, Bryan Francisco Ocaña Valdez con documento de identificación N° 1718291329; manifiesto que soy el autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 14 de febrero de 2024

Atentamente,

Bryan Francisco Ocaña Valdez

1718291329

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL PRESENTE PROYECTO TÉCNICO DE TITULACIÓN A LA UNIVERSIDAD POLITECNICA SALESIANA

Yo, Bryan Francisco Ocaña Valdez con documento de identificación No. 1718291329, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy el autor del Proyecto Técnico: "Desarrollo de un prototipo de una criptomoneda con herramientas de código abierto basado en Ethereum de la Universidad Politécnica Salesiana.", el cual 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 concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 14 de febrero de 2024

Atentamente,

Bryan Francisco Ocaña Valdez

1718291329

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Julio Ricardo Proaño Orellana con documento de identificación N° 0103909412, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollada el trabajo de titulación: DESARROLLO DE UN PROTOTIPO DE UNA CRIPTOMONEDA CON HERRAMIENTAS DE CÓDIGO ABIERTO BASADO EN ETHEREUM DE LA UNIVERSIDAD POLITECNICA SALESIANA CAMPUS SUR, realizado por Bryan Francisco Ocaña Valdez con documento de identificación N° 1718291329, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todo los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 14 de febrero de 2024

Atentamente,

Ing. Julio Ricardo Proaño Orellana PhD.

0103909412

Dedicatoria

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

A mis padres, Héctor y Susana les dedico este trabajo y agradezco todo corazón por apoyarme mi guía y ejemplo, ensenándome que soy capaz de alcanzar lo que me proponga y que todos los sueños se cumplen sin importar lo difícil parezcan.

A mi hermano Héctor Jr. gracias por estar siempre pendiente, por sus consejos y motivación en todo momento y por ofrecerme todo el apoyo para alcanzar esta gran meta.

A mi esposa Tania y mi hija Emily, quienes han sido un pilar fundamental en mi trayectoria universitaria para culminar esta etapa, día a día con su compañía, fuerza y apoyo hasta conseguir el fin de culminación de mi carrera profesional.

A toda mi familia, amigos y profesores quienes de una u otra forma me brindaron su apoyo yconsejos, aportando con sus enseñanzas a lo largo de la carrera y permitiendo que día a día se alcance los objetivos esperados.

Bryan Francisco Ocaña Valdez

Índice General

Índice de Tablas

Índice de Figuras

RESUMEN

En el presente proyecto se desarrolló una investigación acerca de la implementación de un prototipo de criptomoneda utilizando herramientas de código abierto proporcionadas por Solidity de Ethereum tuvo como prioridad el explorar la viabilidad y funcionalidad de una nueva criptomoneda en la plataforma Blockchain. La necesidad de lanzar una moneda digital segura, confiable y funcional, fue comprender el proceso de implementación de contratos inteligentes y el lanzamiento en la red Ethereum.

La metodología aplicada implica el uso del lenguaje de programación Solidity para redactar los contratos inteligentes. Se emplean herramientas como Remix, un IDE de desarrollo oficial, para la creación y comprobación de funcionamiento de los Smart Contracts, así mismo, MetaMask, una billetera virtual para interactuar con el contrato y realizar transacciones en la red.

Además, se utilizaron librerías, compiladores y depuradores para garantizar la correcta funcionalidad y seguridad de los contratos inteligentes.

Los resultados obtenidos fueron demostrados en la viabilidad técnica y operativa de la criptomoneda dentro de la creación exitosa del Smart Contracts y el comportamiento en el entorno al momento de interactuar con la red Ethereum de manera segura y eficiente.

En definitiva, el desarrollo de este prototipo de criptomoneda utilizando Solidity de Ethereum representó un paso importante en la comprensión y aplicación práctica de la tecnología Blockchain.

Palabras claves: Criptomoneda, Blockchain, Ethereum, Solidity, Smart Contracts, Metamask.

ABSTRACT

In this project, an investigation was developed about the implementation of a cryptocurrency prototype using open-source tools provided by Ethereum's Solidity. The priority was to explore the viability and functionality of a new cryptocurrency on the Blockchain platform. The need to launch a secure, reliable and functional digital currency was to understand the process of implementing smart contracts and launching on the Ethereum network.

The applied methodology involves the use of the Solidity programming language to write smart contracts. Tools such as Remix, an official development IDE, are used to create and verify the operation of Smart Contracts, as well as MetaMask, a virtual wallet to interact with the contract and carry out transactions on the network.

In addition, libraries, compilers and debuggers were used to guarantee the correct functionality and security of the smart contracts.

The results obtained were demonstrated in the technical and operational viability of the cryptocurrency within the successful creation of Smart Contracts and the behavior in the environment when interacting with the Ethereum network in a safe and efficient manner.

In short, the development of this cryptocurrency prototype using Ethereum's Solidity represented an important step in the understanding and practical application of Blockchain technology.

Keywords: Cryptocurrency, Blockchain, Ethereum, Solidity, Smart Contracts, MetaMask.

CAPÍTULO I

1. INTRODUCCIÓN

1.1. ANTECEDENTES

El proyecto de Maestría descrito por (Cesar Rodríguez C, 2019) sobre el "Diseño e implementación de un modelo de tokenización de acceso a red basado en Ethereum Blockchain", se desarrolló un análisis exhaustivo del ecosistema de Blockchain proyectándose a brindar un recurso que condescienda a evaluar el valor real de esta tecnología. Específicamente, se planteó realizar una prueba de concepto en la cual se integre la capacidad de tokenización en la gestión del acceso y monitoreo de las redes de ordenadores, utilizando la plataforma de Blockchain de Ethereum como respaldo.

(Gabela R., 2019) Como parte de su labor de titulación se propuso inspeccionar las características y composición de las criptomonedas, así como las formas y métodos utilizados en monedas digitales por criminales para llevar a cabo actividades delictivas. Para ello, mediante una investigación en fiscalía ecuatoriana se aprovechó una metodología cualitativa-descriptiva.

Activos intangibles relativamente nuevos que han generado debates con relación a sus implicaciones legales son las criptomonedas con su representación descentralizada, independiente y anónima de intermediarios tradicionales ha facilitado su uso en la comisión de actividades delictivas, lo que ha preocupado por su potencial uso indebido. Las derivaciones alarmantes de este estudio se pudieron evidenciar que los delincuentes han logrado realizar transacciones sin dejar vestigio. (Álvarez Díaz, 2019)

En el trabajo de titulación de (José Romero S, 2019) se enfocó en "la aplicación de contratos inteligentes en Ethereum con el propósito principal de este contrato es administrar el

alquiler de viviendas, permitiendo la incorporación y arrendamiento de propiedades". Para facilitar la utilización del contrato, se ha implementado una aplicación web. Además, se ha integrado un dispositivo de Internet de las cosas (IoT) para simular la apertura de la puerta.

1.2. DESCRIPCIÓN DEL PROBLEMA

En general la implementación de una criptomoneda puede parecer una tarea compleja, hay varios problemas técnicos importantes que se debe afrontar para hacerlo de manera efectiva. Aquí hay algunos de los problemas técnicos más relevantes que se deben considerar:

1. Seguridad: La seguridad es uno de los mayores problemas técnicos que se debe abordar al crear una criptomoneda. La criptomoneda debe ser segura para prevenir la manipulación, el robo y el fraude. Debe contar con mecanismos de seguridad como la encriptación y la autenticación, para asegurar la privacidad y la integridad de la moneda virtual.

2. Consenso: El consenso es otro problema técnico importante al crear una criptomoneda. Se necesita establecer un sistema de consenso entre los nodos que participan en la red para validar las transacciones y evitar posibles problemas como el doble gasto. Esto se puede lograr mediante algoritmos de consenso como Proof of Work (POW) o Proof of Stake (POS). (Binance Academy, 2022)

3. Escalabilidad: La escalabilidad es otro problema técnico importante a la hora de crear una criptomoneda. A medida que la red crece, la cantidad de transacciones y nodos también aumenta, lo que puede generar problemas de rendimiento. Sc necesita diseñar una arquitectura que permita un escalado horizontal y vertical para que la red pueda manejar grandes cantidades de transacciones sin comprometer su rendimiento.

4. Interoperabilidad: La interoperabilidad es otro problema técnico importante que se debe considerar al crear una criptomoneda. La red de la criptomoneda debe poder interactuar con otras redes de criptomoneda y sistemas financieros tradicionales

Una de las herramientas de código abierto que puede facilitar este trabajo es el uso de Ethereum. Sin embargo, crear una criptomoneda con Ethereum implica enfrentar varios desafíos como:

Creación del contrato inteligente: El primer paso para crear una criptomoneda en Ethereum es desarrollar un contrato inteligente. Esto implica aplicar una tarea técnica compleja que requiere conocimientos previos y consolidados de programación en Solidity de Ethereum.

Implementación de la criptomoneda: Una vez que se ha creado el contrato inteligente, se debe implementar la criptomoneda en la red de Ethereum. Esto implica compilar el código del contrato y desplegarlo en la red. Se debe asegurar de que el contrato esté bien probado y verificado para evitar errores costosos.

Configuración de la criptomoneda: Una vez que se ha desplegado el contrato, es necesario configurar la criptomoneda. Esto implica establecer los parámetros de la criptomoneda, como el suministro máximo, la tasa de emisión, la distribución inicial de tokens y los detalles de la ICO.

Seguridad del contrato inteligente: Los contratos inteligentes son propensos a errores y vulnerabilidades de seguridad. Es importante asegurarse de que el contrato esté bien auditado y probado para garantizar su seguridad. Debes considerar también la posibilidad de ataques como el de la seguridad de las claves privadas.

Interoperabilidad: Si la idea es que la criptomoneda se integre con otras aplicaciones o sistemas, se debe asegurar de que sea compatible con los estándares de Ethereum y las interfaces de programación API's existentes.

En este proyecto se plantea la implementación y el diseño de un prototipo de una criptomoneda usando la tecnología de Código Abierto de Ethereum. Este prototipo tiene como mira en el futuro establecer un fundamento consolidado de conocimiento que condescienda el perfeccionamiento de proyectos más completos y complejos por alumnos en otros trabajos de titulación. Para ello se resume un plan de acción.

Plan de acción:

1. Crear el contrato inteligente: Lo primero que se debe hacer es crear el contrato inteligente en Solidity, El cual, debe incluir los parámetros de la criptomoneda, como la tasa de emisión, el suministro máximo y la distribución inicial de tokens.

2. Implementación de la criptomoneda: Una vez que se ha creado el contrato inteligente, se debe implementar la criptomoneda en la red de Ethereum. Es importante hacer pruebas exhaustivas y verificar la seguridad del contrato antes de desplegarlo en la red.

1.3. JUSTIFICACIÓN

El presente proyecto provee el conocimiento e investigación para el desarrollo de una criptomoneda mediante código abierto de Ethereum, ante el alcance mencionado son desafíos técnicos críticos que deben abordarse para certificar la seguridad, la eficacia y la viabilidad a largo de la ejecución de la criptomoneda; es así como en base a la complejidad se plantea un concepto menos subjetivo y más objetivo.

Tratar de familiarizarse con la seguridad es fundamental para prevenir el fraude y así garantizar la integridad de las transacciones y la privacidad de este. Es necesario un consenso para validar las mismas y prevenir el doble gasto en la red de la criptomoneda. La escalabilidad es crucial para avalar que la moneda virtual pueda manejar un gran volumen de transacciones a medida que la red crece. La interoperabilidad es importante para permitir que la criptomoneda interactúe con otras redes y sistemas financieros tradicionales.

Por otro lado, hay que abordar que en el proyecto se trabajará con Blockchain y se adaptará los contratos inteligentes para dar cumplimiento a un registro entendiendo que no intervienen seres humanos sino al contrario se configura mediante un script que compila códigos de operación a bajo nivel para proceder a almacenar a través de la base de datos del Blockchain con una dirección en específico, para después enviar por cada nodo dentro de la red de una máquina virtual, que es quién compila los códigos del script utilizando todos los datos almacenados en cada transacción.

Con la creación un SmartContracts en el Blockchain de Ethereum conocido como Turing-complete, conceptualizado en una computadora distribuida globalmente que puede leer y ejecutar ordenes codificados correctamente mediante la Máquina Virtual Ethereum (EVM).

La dificultad que se reconoce en este proyecto al ampliar la investigación sobre un contrato inteligente y llevar a cabo juntamente con esta herramienta.

1.4. OBJETIVOS

1.4.1. Objetivos General

Desarrollar un prototipo de una criptomoneda usando herramientas disponibles de código abierto proporcionado por Solidity de Ethereum.

1.4.2. Objetivos Específicos:

- Recopilar información acerca de los elementos clave de una criptomoneda usando herramientas de código abierto de Ethereum.
- Desarrollar el prototipo de una criptomoneda y ejecutar un contrato inteligente escalable en la red Blockchain.
- Implementar mecanismos de seguridad y pruebas para garantizar la confidencialidad y validar el prototipo de criptomoneda.
- Crear documentación detallada sobre los resultados obtenidos en la implementación de la criptomoneda.

1.5. ALCANCE

Para el objetivo específico A: “Recopilar información sobre los elementos clave de una criptomoneda usando herramientas de código abierto de Ethereum.” Se obtendrá información relevante sobre los componentes fundamentales de una criptomoneda utilizando herramientas de código abierto disponibles en la plataforma Ethereum, las mismas que pueden incluir exploradores de bloques, bibliotecas y frameworks de mejora de Smart Contracts, como Open Zeppelín, que proporciona funcionalidades predefinidas y facilitan el proceso de creación y despliegue.

Para el objetivo específico B: “Desarrollar el prototipo de una criptomoneda y ejecutar un contrato inteligente escalable en la red Blockchain.” Una vez determinados todos los requerimientos necesarios para el prototipo de la criptomoneda, se procederá a desarrollar la programación en Solidity, especializado en Smart Contracts. Además, el manejo del entorno de Metamask para gestionar la comunicación de los protocolos de transferencias del prototipo en la Blockchain.

Para el objetivo específico C: “Implementar mecanismos de seguridad y pruebas para garantizar la confidencialidad y validar el prototipo de criptomoneda.” En esta parte se buscará garantizar que la criptomoneda sea segura, confiable y funcional. Esto es importante por la descentralización y la importancia de la confianza en las criptomonedas.

Para el objetivo específico D: “Crear documentación detallada sobre los resultados obtenidos en la implementación de la criptomoneda.” Al concluir el proyecto se documentará de manera concisa todos los efectos causados en el aprendizaje, investigación y desarrollo de la implementación de la criptomoneda.

CAPÍTULO II

2.1. MARCO TEÓRICO

2.1.1 Definición, características y beneficios de las criptomonedas.

Las criptomonedas son una manera de conocer el mundo virtual que utiliza técnicas de criptografía para asegurar y proteger las transacciones. Estas monedas digitales se basan en tecnologías como la cadena de bloques, (Blockchain) para certificar la seguridad e integridad de las transacciones. A comparación de las monedas habituales emitidas por los bancos centrales a nivel mundial, las criptomonedas son descentralizadas y no están respaldadas por ninguna entidad central ni gobierno autónomo. Las transacciones con criptomonedas pueden hacerse de forma directa y segura entre los participantes sin necesitar intermediarios. Además, las criptomonedas ofrecen ventajas como la rapidez en las transacciones, la privacidad y la posibilidad de realizar transacciones globales sin restricciones geográficas.

Tabla 1 *Principales criptomonedas y sus características*

Cripto-moneda	Fecha Lanzamiento	Market cap [M$]	Algoritmo de Hash	Consenso	Precio [$]
Bitcoin [BTC]	1/1/2009	16050	SHA-256	PoW (Proof of Work)	$ 9926
Ethereum [ETH]	1/8/2015	999	SHA3-256	PoW (PoS (Proof of Stake))	$ 1125
Ripple [XRP]	1/9/2013	233	ECDSA	Byzantine Consensus	$ 6295
LiteCoin [LTC]	1/10/2011	185	Scrypt	PoW	$ 373
Monero [XMR]	1/5/2014	170	Cryptonight	PoW	$ 1222
Dash [DASH]	1/1/2014	120	X11	PoW/PoS	$1693

Entre las principales criptomonedas a nivel mundial se encuentran las siguientes:

Bitcoin (BTC): creada por Satoshi Nakamoto, la primera criptomoneda desarrollada en 2009. Es la moneda virtual más reconocida. Utiliza el algoritmo hash criptográfico SHA-256.

Ethereum (ETH): creada en 2015, es la segunda criptomoneda más grande por capitalización de mercado. Está diseñado para permitir y utilizar una cadena de bloques basada en Prueba de trabajo (POW) en conjunto con contratos inteligentes y aplicaciones descentralizadas (DApps).

Ripple (XRP): una moneda digital basada en Blockchain creada en 2013. Permite pagos instantáneos y de bajo costo en diferentes monedas y países. Ripple utiliza una plataforma informática distribuida llamada XRP Ledger.

LiteCoin (LTC): creada en 2011, es una criptomoneda que tiene como proporcionar una solución más eficiente y escalable para Bitcoin, el mismo que utiliza el algoritmo Scrypt para minar.

Monero (XMR): una criptomoneda anónima que utiliza el algoritmo Cryptonight para la minería. El objetivo de Monero es brindar privacidad y anonimato a sus usuarios.

Dash (DASH): creada en 2014, es una moneda digital descentralizada que proporciona tiempos de procesamiento de transacciones más rápidos en comparación con Bitcoin. Dash utiliza un mecanismo de consenso llamado Prueba de participación (POS).

NEO (NEO): una plataforma china de activos digitales que utiliza el algoritmo de consenso de Prueba de participación (POS) para implementar y ejecutar varios tipos de contratos inteligentes en su Blockchain.

Todas estas criptomonedas usan varios algoritmos de consenso y hash criptográficos.

El control sobre la creación de nuevas unidades de criptomoneda se realiza a través de un proceso conocido como minería. Los mineros utilizan su poder de cómputo para resolver problemas matemáticos complejos que verifican las transacciones y añaden nuevos bloques al Blockchain. Como recompensa por su trabajo, los mineros reciben nuevas unidades de criptomoneda.

La importancia de las criptomonedas en la actualidad radica en varios aspectos:

Seguridad: La criptografía proporciona un valioso nivel de seguridad en las transacciones. Esto se debe a que se manejan algoritmos criptográficos para proteger la integridad de los datos y autenticar las transacciones.

Privacidad: Al utilizar criptomonedas, los usuarios pueden mantener un mayor nivel de privacidad en comparación con las transacciones habituales. Aunque estos son registradas en una cadena de bloques pública, la información privada de los usuarios no están directamente asociados con esas transacciones.

Acceso global: Las criptomonedas permiten transacciones globales rápidas y seguras. No hay restricciones geográficas ni barreras internacionales, lo que facilita el intercambio de valor entre personas de diferentes partes del mundo.

Eliminación de intermediarios: Al realizar transacciones con criptomonedas, no es necesario contar con intermediarios tradicionales, como bancos o servicios de pago. Esto reduce los costos y los tiempos de espera asociados con las transacciones financieras tradicionales.

Inclusión financiera: Es relevante en áreas donde hay una falta de infraestructura financiera. Las criptomonedas pueden proponer acceso a servicios financieros a usuarios que no tienen conocimiento profundo a los sistemas bancarios convencionales.

Innovación tecnológica: Estas tecnologías tienen el potencial de revolucionar diversas industrias, como las finanzas, la logística y la gestión de la cadena de información. Las criptomonedas están protegidas por tecnologías en procesos innovadores, como la cadena de bloques (Blockchain), que permiten la creación de aplicaciones descentralizadas y contratos inteligentes.

La relevancia de la criptomoneda en la actualidad se basa en su habilidad para transformar la manera en que se llevan a cabo las transacciones y se administra el valor en el ámbito digital.

Existen diversas monedas digitales con su propia tecnología y protocolo subyacente, lo que las hace únicas en términos de seguridad, escalabilidad y funcionalidad.

A continuación, se redacta cada implemento que puede ser utilizado para aplicar los términos mencionados.

Tabla 2 *Requisitos de una criptomoneda*

Requisitos	Descripción
Contrato Inteligente	Es necesario desarrollar un contrato inteligente utilizando el lenguaje Solidity. El contrato será responsable de la creación, transferencia y gestión de la criptomoneda.
Funciones de Transferencia	El contrato debe incluir funciones para permitir la transferencia de la criptomoneda entre diferentes cuentas de usuarios. Estas funciones deben avalar la integridad y la seguridad de las transacciones.
Emisión y Quema	Se deben implementar mecanismos para la emisión inicial de la criptomoneda y la posibilidad de quemar o retirar monedas de circulación. Esto ayudará a controlar la oferta total de la criptomoneda.
Seguridad y Protección	Es de suma importancia aplicar medidas de seguridad y protección con el fin de evitar ataques y vulnerabilidades en el contrato inteligente. Para lograrlo, es necesario seguir las mejores prácticas de seguridad de Solidity y llevar a cabo auditorías de seguridad.
Interoperabilidad	Si se desea que la criptomoneda sea compatible con otras plataformas o contratos inteligentes, es importante considerar la interoperabilidad y asegurarse de que se pueda interactuar con otros sistemas de forma eficiente.

Auditoría y Pruebas	Antes de implementar la criptomoneda, se deben realizar pruebas exhaustivas y auditorías para garantizar su correcto funcionamiento y detectar posibles problemas o errores en el código.
Documentación y Transparencia	Se debe proporcionar una documentación detallada y clara sobre el funcionamiento y las características de la criptomoneda. Además, se debe fomentar la transparencia en relación con los aspectos técnicos y la gestión de la criptomoneda.

Es relevante resaltar que las criptomonedas son descentralizadas, lo cual implica que no están sujetas al control ni respaldo de ningún gobierno o entidad central. Esta característica proporciona a los usuarios un mayor grado de control sobre sus activos y los libera de la dependencia de los sistemas financieros convencionales.

Por otro lado, un consenso es un elemento fundamental en una red descentralizada que se encarga de validar, almacenar y transmitir información en la cadena de bloques. Este proceso unifica los nodos en una cadena de bloques, que pueden ser completos o ligeros, y su función principal es garantizar la integridad y seguridad de la cadena de bloques.

Algunos ejemplos comunes de algoritmos de consenso utilizados en criptomonedas incluyen Proof of Work (PoW) y Proof of Stake (PoS) (Binance Academy, 2018)

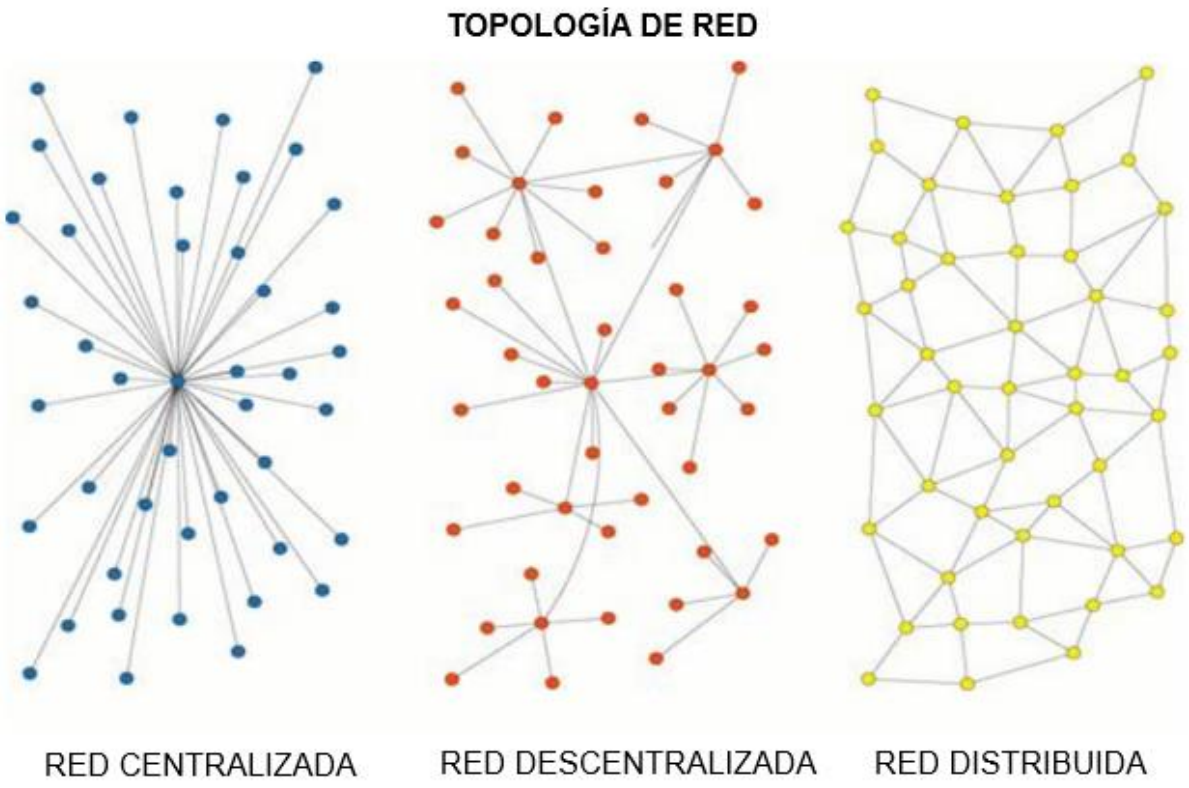
2.1.2 Blockchain y su funcionamiento.

La tecnología Blockchain es esencial para el funcionamiento de una criptomoneda y puede ser concebida como un registro público descentralizado que almacena todas las transacciones efectuadas en una red. Cada transacción es agrupada en bloques y se vincula con el

bloque precedente, conformando una cadena de bloques. Esta cadena de bloques garantiza la transparencia y la seguridad de las transacciones.

La transparencia en la tecnología Blockchain se deriva de su carácter público, en el cual todas las transacciones son visibles y accesibles para todos los participantes de la red. Esto implica que cualquier individuo puede verificar la autenticidad de una transacción y asegurarse de que no haya manipulación o fraude..(*Los ocho componentes de una blockchain pública y los ocho componentes de Ethereum | Ignacio G.R. Gavilán*, s/f)

Figura 1 *Tipos de topología de red*



Nota: 1 (Fuente: Ignacio G.R. Gavilán, s/f)

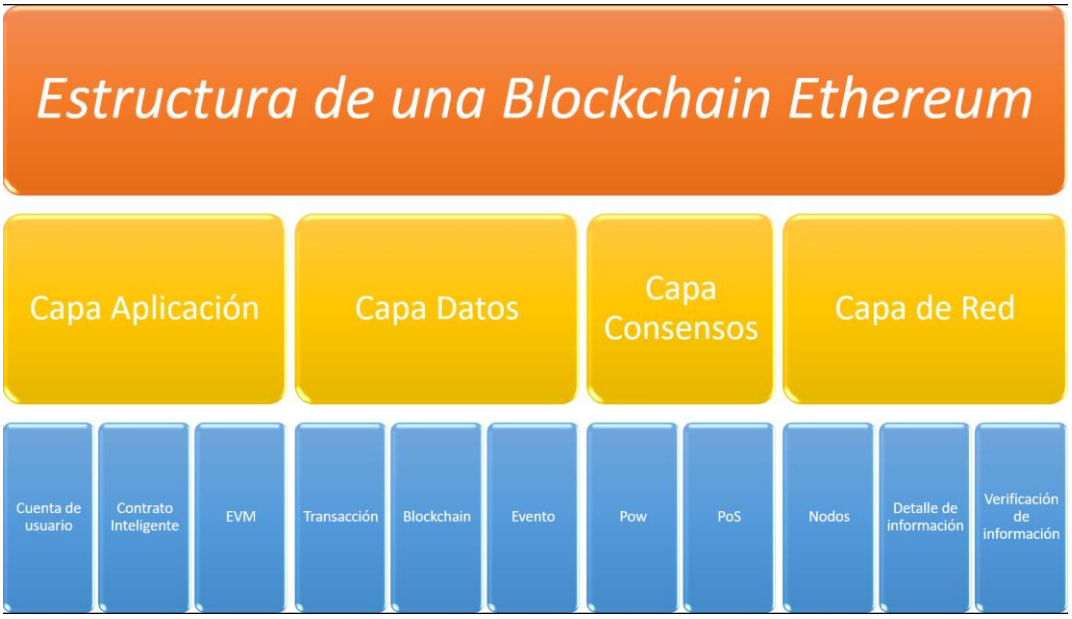
Además de la transparencia, la seguridad es otro aspecto clave de la tecnología Blockchain. Cada bloque contiene un registro de transacciones, un hash que lo identifica de forma única y el mismo del bloque anterior. Esta estructura hace que sea muy difícil actualizar o manipular datos previos en la cadena sin ser detectado. También permite realizar transacciones sin la necesidad de intermediarios de forma rápida y segura, utilizando algoritmos criptográficos complejos para acreditar la integridad y la autenticidad de la información almacenada. (*Blockchain - Qué es, definición y concepto | 2023 | Economipedia*, s/f)

2.1.3 Estructura de Blockchain.

La estructura de una Blockchain Ethereum se basa en una cadena continua de bloques que administra información acerca de las transacciones realizadas y el estado actual del sistema. Cada vínculo con el bloque de cadena contiene un conjunto de transacciones y un hash que es el resultado de la combinación del contenido del bloque anterior y la información del actual bloque. El Hash sirve como sello temporal y asegura que la integridad de la cadena se mantenga, ya que cualquier tentativa de alterar un bloque modificaría su contenido y, por lo tanto, toda la cadena sería invalidada. (Algorand, 2023)

Además de las transacciones y bloques, Ethereum también utiliza una base de datos de estado que almacena el estado actual de todas las cuentas y contratos en la red. Cada nodo en el árbol contiene el resultado de aplicar una función de hash criptográfica al contenido de los nodos inferiores, lo que garantiza que cualquier cambio en cualquier parte del árbol sea detectado y verificado. (Gavilán, I. G.R., 2019)

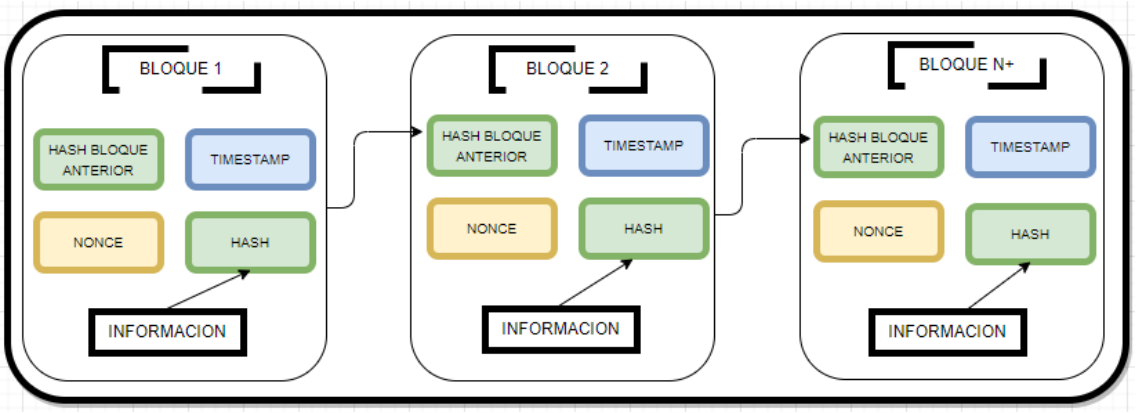
Figura 2 *Estructura de una Blockchain Ethereum.*



Nota: 2 (Elaborado por: Autor)

Una de las virtudes de Blockchain es que es muy segura y puede prevenir ataques de piratas informáticos. Para fraguar un mensaje, todos los nodos de la red deben alcanzar un consenso a la vez; más de la mitad de las máquinas comparten. Además, si la red se cae, siempre que el nodo no se caiga, la red puede continuar brindando servicios. Blockchain realiza transacciones a través de cuentas de administración y su estructura es la siguiente.(Alharby & van Moorsel, 2017)

Figura 3 *Estructura de un bloque Blockchain*



Nota: 3 (Elaborado por: Autor)

En resumen, se presenta una comparativa de los diferentes tipos de Blockchain existentes considerando sus principales particularidades.

Figura 4 *Comparativo de tipos de Blockchain (Allende y Colina, 2018, p. 26)*

	Públicos Bitcoin, Ethereum, Litecoin	**Privados** Hyperledger, Corda, Quorum	**Federados** Hyperledger, Corda, Quorum	***Blockchain as a Service*** IBM, Microsoft, Amazon
Cualquiera puede participar	✓	✗	✗	NA
Los participantes actúan, en general, como nodos	✓	✗	✗	NA
Transparencia	✓	~	~	NA
Hay un único administrador	✗	✓	✗	NA
Hay más de un administrador	✗	✗	✓	NA
No hay administradores	✓	✗	✗	NA
Ningún participante tiene más derechos que los demás	✓	✗	✗	NA
Se pueden implementar Smart *Contracts*	✓	✓	✓	NA
Existe recompensa por minado de bloques	~	✗	✗	NA
Soluciona problema de falta de confianza	✓	✗	~	NA
Seguridad basada en protocolos de consenso	✓	✗	~	NA
Seguridad basada en funciones *hash*	✓	~	~	NA
Provee servicios en la nube	NA	NA	NA	✓

2.1.4 Ethereum como plataforma Blockchain

Ethereum admite a los desarrolladores construir y ejecutar DApps y contratos inteligentes. Fue creado y conceptualizado por Vitalik Buterin en 2013, con el objetivo de superar las limitaciones de Bitcoin y suministrar una plataforma más versátil para la ejecución de contratos inteligentes, Ethereum tiene su propia criptomoneda llamada Ether (ETH), que se utiliza como medio de intercambio en la red y como incentivo para los mineros que validan las transacciones. Además de ser una criptomoneda, el Ether también se utiliza para pagar las tarifas de gas necesarias para ejecutar las transacciones y los contratos inteligentes en la red Ethereum.

A diferencia de Bitcoin, Ethereum es una plataforma Turing complete, lo cual implica que tiene la capacidad de ejecutar cualquier tipo de programa informático. Esto brinda a los desarrolladores la posibilidad de crear una diversidad de aplicaciones descentralizadas, que van desde juegos hasta sistemas de votación o servicios financieros más sofisticados.

La plataforma de Ethereum ha promovido el desarrollo de estándares y protocolos que permiten la interoperabilidad entre diferentes aplicaciones y tokens en la red. Esto ha llevado al aparecimiento de un ecosistema vibrante de proyectos y tokens implementados sobre la plataforma Ethereum.

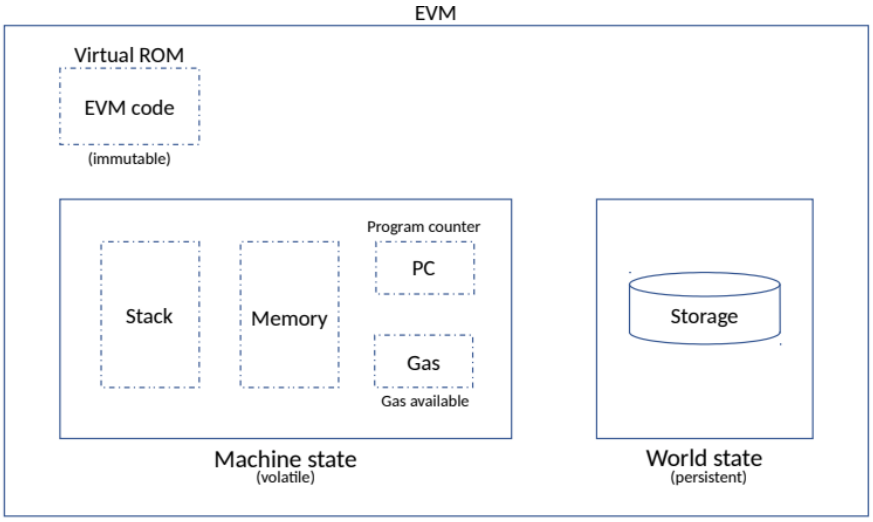
2.1.4 Arquitectura de Ethereum Virtual Machine y Smart Contracts

La arquitectura de una Blockchain basada en Ethereum incluye varios componentes. En primer lugar, está la propia cadena de bloques, que almacena todas las transacciones que se han realizado en la red. Además, Ethereum utiliza una máquina virtual basada en bytecode, conocida

como EVM (Ethereum Virtual Machine), es esencial para la ejecución de los contratos inteligentes en la plataforma.

Los contratos inteligentes son aplicaciones de computadora que de manera automática ejecuta y satisface determinadas condiciones predefinidas. También hay nodos en la red que operan como puntos de entrada y salida, y que verifican y validan las transacciones antes de agregarlas a la cadena de bloques. Por último, se utilizan diferentes tipos de criptografía para garantizar la seguridad de la red y la privacidad y confidencialidad de las transacciones que se realizan. En definitiva, la arquitectura de una Blockchain de Ethereum se compone de una cadena de bloques con una máquina virtual, contratos inteligentes, nodos y criptografía para certificar la integridad y seguridad de la red.

Figura 5 *Arquitectura EVM*



Nota: 4 (Fuente: Algorand.2023)

En cuanto a la EVM, se observan los tipos de criptografía en Ethereum, conjuntos de técnicas y protocolos criptográficos utilizados para proteger la seguridad de la red Ethereum y sus operaciones. Esto incluye la encriptación de los datos de las transacciones, la autenticación de las identidades de los usuarios y la protección de los Smart contracts. Uno de los aspectos más

importantes de la criptografía en Ethereum es el uso de claves públicas y privadas para verificar y autorizar las transacciones en la red. Además, Ethereum utiliza una variedad de algoritmos criptográficos avanzados para proteger la integridad y la privacidad de los datos almacenados en la Blockchain.

2.1.5 Remix de Ethereum e implementación de la criptomoneda

Para comprender en detalle la Máquina Virtual de Ethereum (EVM), se hace mención a la plataforma de Remix de Ethereum como una herramienta de desarrollo de contratos inteligentes basada en el navegador. Esta plataforma permite a los desarrolladores escribir, implementar y probar contratos inteligentes en su entorno. Remix proporciona una interfaz de usuario intuitiva y fácil de usar, además de ofrecer funcionalidades como la compilación y despliegue de contratos, la depuración de contratos en tiempo real y la interacción con contratos existentes en la cadena de bloques de Ethereum.

Algunas de las características clave de Remix Ethereum incluyen:

Editor de código: Remix Ethereum proporciona un editor de código integrado que permite a los desarrolladores escribir y editar contratos inteligentes en lenguaje Solidity. Este editor ofrece resaltado de sintaxis y sugerencias automáticas para facilitar la escritura de código.

Compilador de Solidity: Remix Ethereum incluye un compilador de Solidity que permite a los desarrolladores compilar sus contratos inteligentes en bytecode ejecutable. Esto es necesario para desplegar los contratos en la cadena de bloques Ethereum.

Despliegue de contratos: Remix Ethereum facilita el despliegue de contratos inteligentes en diferentes redes de Ethereum. Los desarrolladores pueden especificar la red de destino y proporcionar los parámetros necesarios para desplegar sus contratos.

Depurador de contratos: Remix Ethereum ofrece un depurador integrado que permite a los desarrolladores rastrear y solucionar problemas en sus contratos inteligentes. Los desarrolladores pueden establecer puntos de interrupción, examinar el estado de las variables y realizar seguimiento de la ejecución del contrato paso a paso.

2.1.6 Principales características de la criptografía simétrica y asimétrica

En el ámbito de cifrado hay un método criptográfico con dos claves para cifrar y descifrar los datos. Estas claves se conocen como clave privada y clave pública, y están matemáticamente relacionadas entre sí de manera única.

En el cifrado asimétrico, el remitente utiliza la clave pública del receptor para cifrar los datos antes de enviarlos. Una vez que los datos cifrados se reciben, solo el receptor, que posee la clave privada correspondiente, puede descifrarlos y acceder a la información original.

Este tipo de cifrado ofrece beneficios añadidos en seguridad y autenticación. Permite una comunicación segura en entornos abiertos, ya que no es necesario compartir la clave privada. Además, se utiliza para firmar digitalmente los datos, proporcionando un nivel adicional de verificación y autenticación.

Es importante tener en cuenta que el cifrado y descifrado asimétrico es computacionalmente más intensivo que el cifrado simétrico, lo que lo hace menos eficiente en términos de velocidad y rendimiento. Por eso, se suele usar junto al cifrado simétrico, donde la clave simétrica se cifra con el cifrado asimétrico y se comparte seguramente entre las partes.

En cambio, el cifrado y descifrado simétrico es una técnica ampliamente empleada en la seguridad de la información con el propósito de preservar la confidencialidad de los datos. En

este enfoque, se utiliza una única clave compartida entre el emisor y el receptor para llevar a cabo el proceso de cifrado y descifrado de la información.

En el cifrado simétrico, el remitente toma los datos originales y la clave compartida, y mediante un algoritmo criptográfico, aplica una serie de transformaciones para convertir los datos en una forma ilegible llamada texto cifrado. El receptor, que posee la misma clave compartida, utiliza el mismo algoritmo, pero en sentido inverso para aplicar transformaciones y obtener nuevamente los datos originales, proceso conocido como descifrado.

Este cifrado ofrece ventajas en velocidad y eficiencia, ya que el proceso de cifrado y descifrado se realiza con la misma clave. Sin embargo, también presenta el desafío de asegurar que la clave compartida se mantenga en secreto, ya que, si alguien no autorizado la obtiene, podría descifrar la información cifrada.(*Conceptos*, s/f)

Para asegurar la confidencialidad en la comunicación, es común que el cifrado simétrico se combine con técnicas como el intercambio seguro de claves o el cifrado de clave pública. Estas técnicas ayudan a establecer una clave compartida de forma segura entre las partes involucradas antes de utilizar el cifrado simétrico para proteger los datos. (Kinsta Inc, 2023)

2.1.7 Lenguaje de programación Solidity y el desarrollo de un contrato inteligente.

Solidity es un lenguaje de alto nivel que proporciona una sintaxis similar a otras populares tipologías de programación, lo que facilita su aprendizaje y uso para programadores familiarizados. Además, ofrece una amplia gama de funciones y características que permiten la implementación de lógica compleja en los contratos inteligentes.

Una de las más destacadas características de Solidity es su capacidad para permitir la programación de contratos inteligentes autónomos y autoejecutables. Esto significa que, cuando

un contrato inteligente se ha implementado en la red de Ethereum, puede funcionar independientemente sin necesidad de intervención externa.

Solidity también ofrece soporte para la gestión de tokens, lo que es especialmente relevante en el contexto de la implementación de una criptomoneda. Mediante Solidity, es posible crear y gestionar de manera personalizada que pueden ser utilizados como unidades de valor dentro de la criptomoneda. (Lucern et al., 2017)

Figura 6 *Ejemplo de Solidity*

```solidity
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("MyToken", "MTK") {
        _mint(msg.sender, initialSupply);
    }
}
```

2.1.8. Mecanismo de consenso

Los mecanismos de consenso son métodos utilizados por una red distribuida para coordinarse y garantizar que todos los usuarios que interactúen con el contrato inteligente estén de acuerdo en el contenido y puedan llegar a un consenso sobre una fuente de información confiable. Estos mecanismos también aseguran que la red pueda resistir cualquier intento de interferencia por parte de intermediarios que puedan tratar de influir en las decisiones tomadas.

2.1.9 MetaMask y su funcionamiento como billetera virtual

Las billeteras virtuales o CriptoWallet son utilizadas para almacenar, emitir o receptar criptomonedas. Estas billeteras poseen características que involucran el generar y almacenar

claves privadas y públicas, la capacidad de ser compatibles con distintas criptomonedas, la habilidad de realizar transacciones seguras y la integración con otras aplicaciones o servicios de Blockchain.

Existen diferentes tipos de CriptoWallet, tales como hardware, software, móviles o en línea, y cada uno de ellos presenta sus propias ventajas y desventajas en términos de seguridad y facilidad de uso. (Introducción | Documentación de MetaMask, s.f.)

2.1.10 Metamask y la interacción con el contrato inteligente.

Después de desarrollar un contrato inteligente en Ethereum, puedes utilizar Metamask para interactuar con ese contrato y realizar transacciones en la red. Mediante una extensión de navegador que admite a los usuarios interactuar con aplicaciones descentralizadas (dApps) en la red Ethereum. (*Introduction | MetaMask developer documentation*, s/f)

Figura 7 *Billetera MetaMask*

Nota: 5 (Fuente: Documentación de Metamask, s.f.)

Algunos puntos importantes para la debida configuración e interacción de Metamask y contratos inteligentes.

1. Configuración de Metamask:

Después de instalar la extensión de Metamask en el navegador, debes crear una nueva billetera o importar una existente.

MetaMask proporciona una serie de recuperación de 12 palabras que se debe guardar de forma segura. Esta frase se utiliza para restaurar tu billetera en caso de pérdida o robo del aplicativo.

Una vez que hayas configurado tu billetera, puedes conectarte a la red Ethereum principal o a una red de prueba como Ropsten o Rinkeby.

2. Interacción con contratos inteligentes:

Metamask permite interactuar con contratos inteligentes en la red Ethereum utilizando su interfaz de usuario.

Puedes agregar contratos inteligentes a tu billetera Metamask proporcionando la dirección del contrato y su ABI (Application Binary Interface).

Después de agregar un contrato inteligente, puedes llamar a sus funciones, enviar transacciones y realizar otras operaciones relacionadas con el contrato directamente desde Metamask.

3. Realización de transacciones:

Con Metamask, se puede enviar y recibir criptomonedas (como Ether) y tokens ERC-20.

Para enviar una transacción, debes especificar la dirección de destino, la cantidad de criptomonedas a enviar y la tarifa de gas que estás dispuesto a pagar.

Metamask te mostrará un resumen de la transacción antes de confirmarla. Una vez confirmada, la transacción se enviará a la red Ethereum para su procesamiento.

Es importante tener en cuenta que la implementación y la interacción con Metamask pueden variar dependiendo de la aplicación o el contrato inteligente específico que se esté utilizando.

CAPÍTULO III

3. METODOLOGÍA

Para el desarrollo de un prototipo de una criptomoneda se requiere una metodología XP (Xtreme Programing) con un enfoque ágil de manera concisa y eficiente donde incluye un proceso de fases.

Fase de planeación: Se define los objetivos y requisitos del prototipo de la criptomoneda. Es importante establecer una visión clara del producto final y determinar los elementos clave que se deben incluir.

Identificar los requisitos: Se analiza y define los requisitos funcionales y no funcionales del prototipo de la criptomoneda. Esto implica determinar las características principales, como la emisión de tokens, la transferencia de fondos y la seguridad.

Establecer el alcance: Considerando las funcionalidades y características que se implementarán inicialmente, se define los límites y alcance del prototipo,

Planificar las iteraciones: Se divide el trabajo en iteraciones cortas y se planifican las tareas que se realizarán en cada una de ellas. Esto permitirá un enfoque ágil y una entrega incremental del prototipo.

Fase de codificación: En esta fase, se lleva a cabo la implementación del prototipo de la criptomoneda utilizando el lenguaje de programación Solidity, y una billetera digital MetaMask.

Diseñar la arquitectura: Se define la estructura y organización del código fuente, asegurando una buena sintaxis y modularidad.

Redactar los contratos inteligentes: Utilizando Solidity para escribir los contratos inteligentes que definirán las reglas y funcionalidades de la criptomoneda.

Integrar MetaMask: Se debe configurar la integración con MetaMask para permitir a los usuarios interactuar con el prototipo de la criptomoneda a través de la billetera digital.

Fase de pruebas: En esta fase, se realizan pruebas exhaustivas para garantizar la funcionalidad y la seguridad del prototipo de la criptomoneda.

Pruebas unitarias: Se realizan pruebas para verificar el correcto funcionamiento de los contratos inteligentes y las funcionalidades implementadas.

Pruebas de seguridad: Evaluar la seguridad del prototipo, identificando posibles vulnerabilidades y aplicando medidas de protección.

Es importante tener en cuenta que la metodología XP promueve un enfoque iterativo e incremental, lo que significa que estas fases se repitieron a lo largo del desarrollo del prototipo, permitiendo la mejora continua y la adaptación a los cambios.

3.1. MODELADO DEL SISTEMA

3.1.1. Análisis de manejo de herramientas

Las herramientas necesarias para la implementación del prototipo son las siguientes:

- IDE Visual Studio Code con su extensión compilador Solidity.
- IDE Remix Ethereum.
- MetaMask.
- Etherscan.

3.1.2. Levantamiento de requerimientos

En esta fase se especifica el propósito de la criptomoneda y las funcionalidades que se desean implementar, como contratos inteligentes, emisión de tokens, entre otros.

Se recalca que la criptomoneda se desarrollará en Solidity, para después la necesidad de integrar Metamask, e interactuar con aplicaciones descentralizadas en la red de Ethereum, y contar con la importancia para desplegar.

Se deben definir los escenarios de prueba que se deben cubrir para la ejecución de contratos inteligentes y verificación de saldos.

De esa misma manera, hay que indicar que se utilizará la billetera MetaMask para realizar las transacciones locales y verificar los resultados.

3.1.3. Análisis de Requerimientos de Software

Se adquirieron las últimas y más estables versiones de las diversas librerías y herramientas utilizadas en el desarrollo tanto como Visual Studio Code, Remix Ethereum y MetaMask. Esto se debe a que estas librerías se actualizan constantemente para ser compatibles con plataformas y lenguajes de programación modernos. Además, estas actualizaciones proporcionan mayor seguridad al desarrollador y ayudan a controlar las vulnerabilidades presentes en las librerías utilizadas en el desarrollo de DApps con tecnología Blockchain.

3.1.4. Definición de requerimientos funcionales

Creación de tokens: El prototipo debe permitir la creación de tokens basados en el estándar ERC-20 de Ethereum. Estos tokens representarán la criptomoneda en el sistema.

Billetera y gestión de claves: El prototipo debe integrarse con MetaMask, para permitir a los usuarios gestionar sus claves privadas y realizar transacciones de tokens.

Registro de transacciones: El prototipo debe conservar un registro de las transacciones de tokens realizadas en el sistema. Esto puede incluir información como el remitente, el destinatario, la cantidad de tokens transferidos junto con la fecha y hora de la transacción.

Seguridad y validación: El contrato inteligente debe implementar mecanismos de seguridad para prevenir posibles ataques y garantizar la integridad de las transacciones.

3.1.5. Definición de requerimientos no funcionales

Requerimiento no funcional de rendimiento

El sistema debe ser capaz de procesar la información de la criptomoneda en un tiempo razonable, evitando demoras en la confirmación de datos como nombre o dirección de token.

Requerimiento no funcional de seguridad

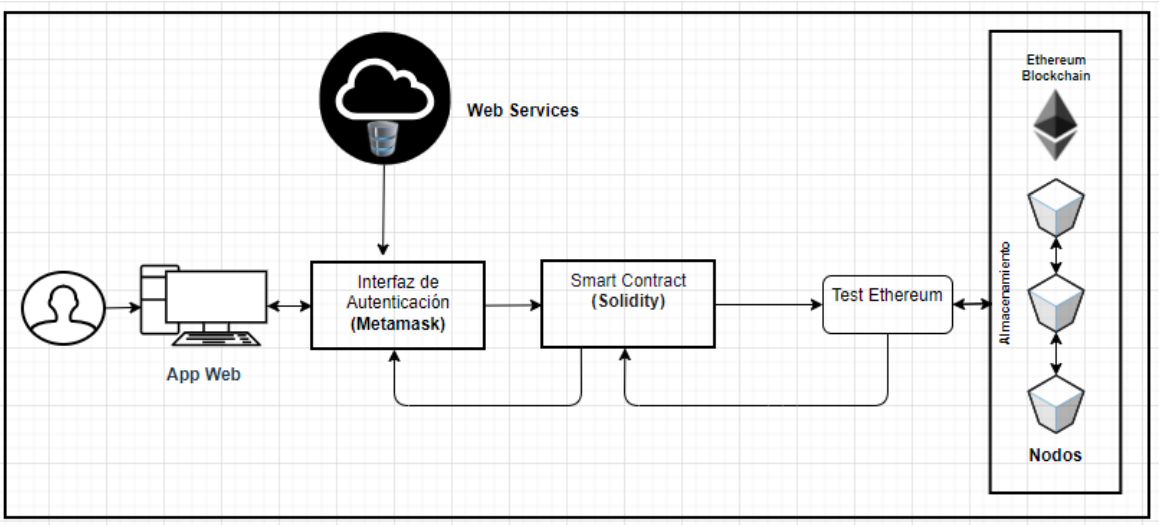
Seguridad de los contratos inteligentes: Es importante garantizar la seguridad de los contratos para evitar vulnerabilidades.

Seguridad de la red Ethereum: La red de Ethereum es descentralizada y está compuesta por nodos que validan las transacciones y ejecutan los contratos inteligentes. se deben implementar medidas de protección contra ataques como el doble gasto, el ataque del 51% y el ataque de denegación de servicio.

Seguridad de MetaMask: Para garantizar a los usuarios, se deben seguir las mejores prácticas de seguridad, como mantener actualizada la extensión, utilizar contraseñas seguras y proteger las claves privadas.

3.2. DIAGRAMA DE APLICACIÓN

Figura 8 *Esquema general del sistema.*



Nota: 6 Elaborado por: (Autor)

En el diagrama presentado se define en primer lugar al usuario como la entidad principal que se beneficia de la aplicación web de Ethereum, donde el Blockchain almacena la información ingresada.

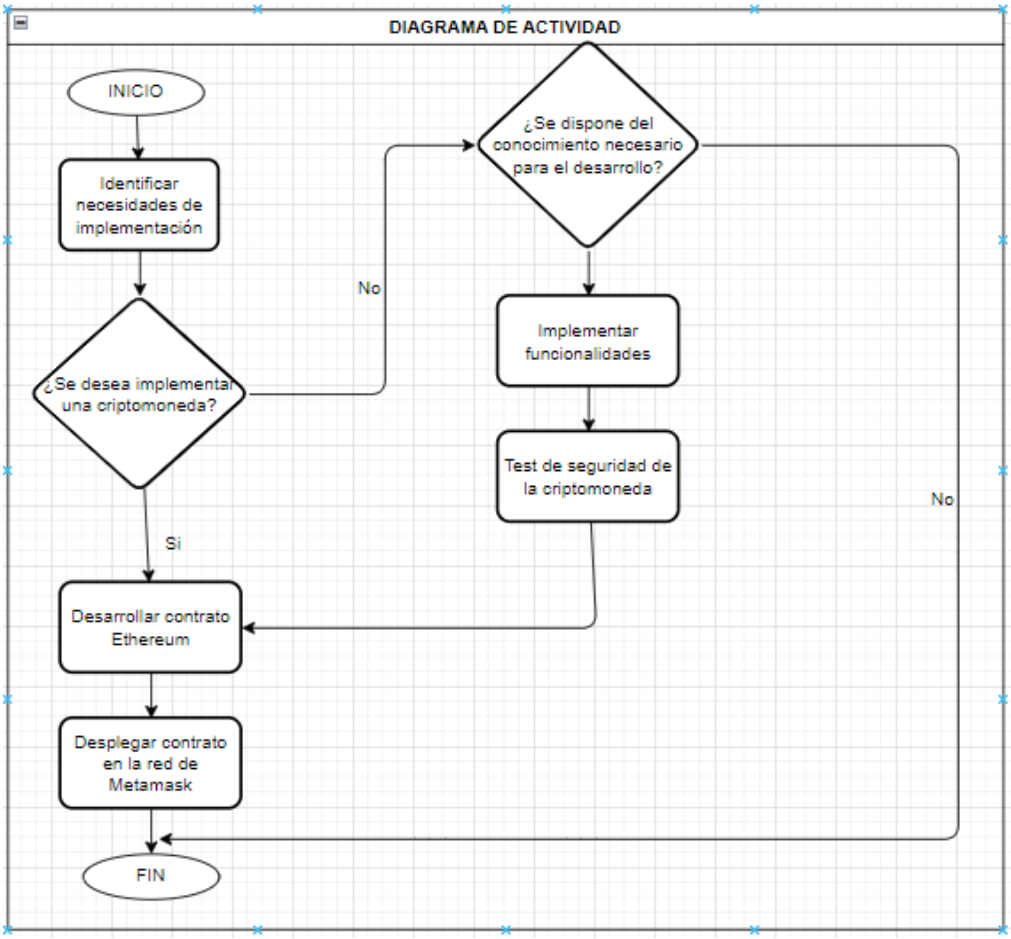
Una vez interactuado con el sistema se define el contrato inteligente con las reglas y restricciones implementadas para garantizar la funcionalidad de la criptomoneda.

Los contratos inteligentes evalúan los datos y desencadenan eventos específicos de aprobación, actualizando el estado del usuario y garantizando la integridad y autenticidad de la información.

Con la verificación de los datos, la criptomoneda aparece con el nombre de usuario, detalle de la criptomoneda y la dirección encriptada.

3.3. DIAGRAMA DE ACTIVIDAD

Figura 9 *Diagrama de Actividad*



Nota: 7 (Elaborado por Autor)

En el Diagrama de Actividad se dispone del conocimiento necesario para el desarrollo de la criptomoneda antes de la implementación es necesario buscar recursos y cursos en línea para aprender el desarrollo de contratos inteligentes, estos que sean simultáneos como es Solidity, que es compatible con Ethereum. A continuación, implementar las funcionalidades específicas de la criptomoneda, como la emisión de monedas y las pruebas de intercambio locales.

Antes de desplegar el contrato, es fundamental realizar pruebas de seguridad, integración y despliegue para asegurar que el contrato es seguro y no puede ser vulnerado.

Figura 10 *Test de Despliegue*

```
//DESPLIEGUE
describe("Deployment", function () {
  it("Deberia establecer el propietario correcto", async function () {
    expect(await UpsToken.owner()).to.equal(owner.address);
  });

  it("Debe asignar el suministro total de tokens al propietario", async function () {
    const ownerBalance = await UpsToken.balanceOf(owner.address);
    expect(await UpsToken.totalSupply()).to.equal(ownerBalance);
  });

  it("Debe establecer el suministro máximo limitado en el argumento proporcionado durante la implementación", async function () {
    const cap = await UpsToken.cap();
    expect(Number(hre.ethers.utils.formatEther(cap))).to.equal(tokenCap);
  });

  it("Debería establecer la recompensa en el argumento proporcionado durante la implementación", async function () {
    const blockReward = await UpsToken.blockReward();
    expect(Number(hre.ethers.utils.formatEther(blockReward))).to.equal(tokenBlockReward);
  });
});
```

Nota: 8 (Elaborado por Autor)

Una de las 15 pruebas que se han determinado para el proyecto es Despliegue, que tiene la funcionalidad de controlar si el propietario es correcto, así mismo, se encarga de asignar el suministro total de tokens, establecer el suministro máximo limitado y de establecer la recompensa proporcionada durante la implementación.

Después de pasar las pruebas de seguridad, se puede desplegar el contrato en la red de Metamask. Aquí, se creará una dirección encriptada para la criptomoneda y se activará la emisión de monedas.

3.4 DIAGRAMA DE SECUENCIA DEL SISTEMA

Figura 11 *Diagrama de secuencia Actor*

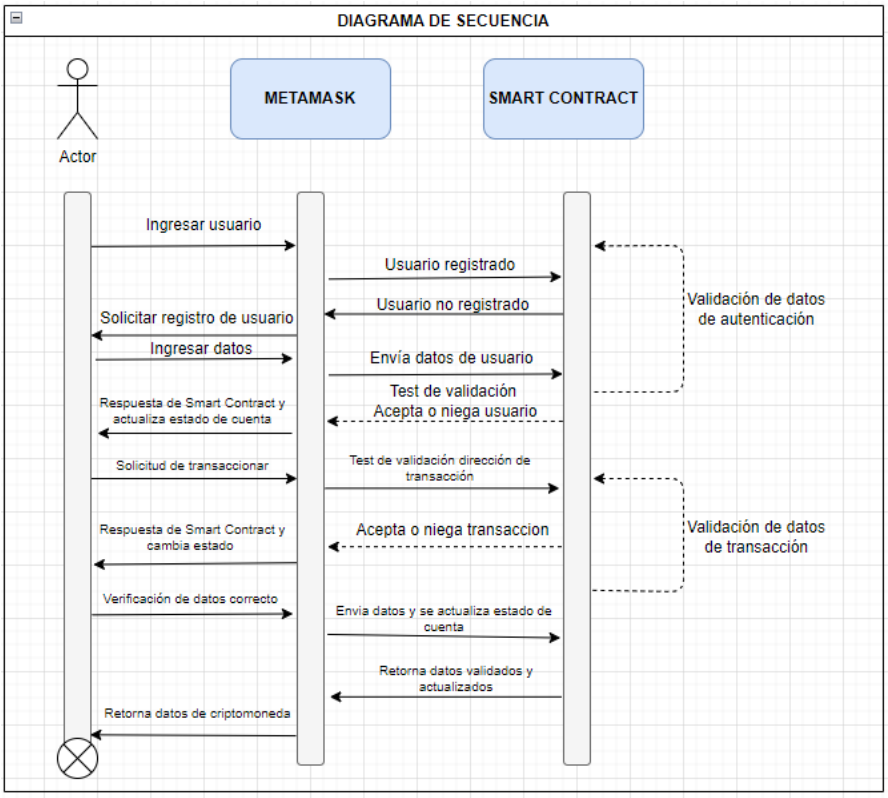


Tabla 3 *Explicación de Diagrama de secuencia Actor (Elaborado por: Autor)*

ACTOR/OBJETO	ACCIONES
Actor	Es quien inicia la secuencia, ingresa nombre de usuario, y envía la información al contrato inteligente para validar.
Metamask	Verifica credenciales autorizados y actúa como aplicación intermediaria entre el usuario y el Blockchain de Ethereum (Smart Contract).

Smart contract	Es la parte lógica de la aplicación, ejecuta las pruebas de seguridad necesarias y retorna datos validados, verificados y actualizados.

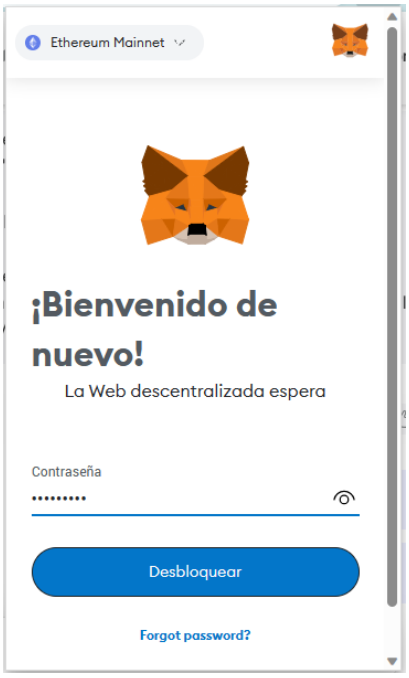
Nota: 9 (Elaborado por: Autor)

3.5. ESTRUCTURA DEL PROYECTO

3.5.1. Instalación y configuración de Metamask Crypto Wallet.

La aplicación de Metamask está disponible en su página oficial https://metamask.io/, es un plugin que va anexado a cualquier navegador web como Chrome, Firefox, Opera siguiendo el proceso de instalación, dentro de la herramienta se crea una nueva billetera aceptando todos los términos y condiciones para uso de datos y recuperación segura de la información.

Figura 12 *Interfaz de logeo de la billetera de Metamask*

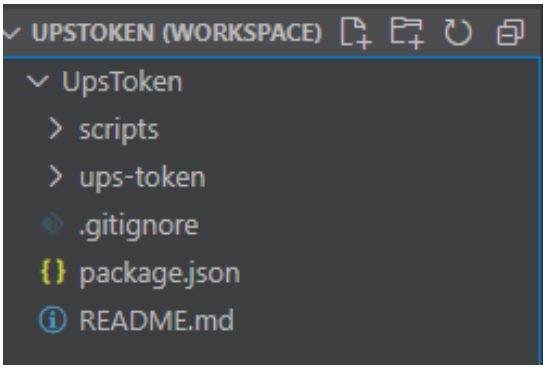


Nota: 10 (Fuente: Documentación de Metamask, s.f.)

3.5.2 Desarrollo de Smart Contract.

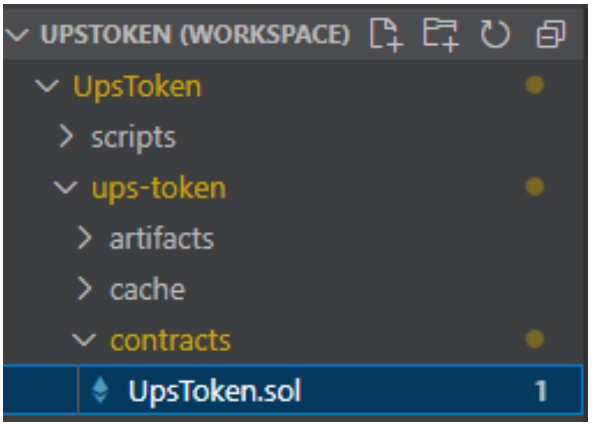
El árbol del directorio del proyecto se encuentra dividido en carpetas, como cabecera el nombre de la criptomoneda implementada “UpsToken”

Figura 13 Árbol de directorio inicial



A continuación, se desglosa a UPSTOKEN encontramos con su versión 0.8.18, donde describe la librería @openzeppelin que deriva las extensiones de ERC20, ERC20Capped, ERC20Burnable y lo que se hereda como dirección principal de ejecución del contrato inteligente.

Figura 14 Descripción de Smart Contract



Dentro del contrato, se encuentra establecida la dirección pública "owner", la cual sirve para identificar al propietario del Smart Contract y determinar que es la persona que lo ha desplegado.

Al principio disponemos de la versión de Solidity, de esta manera se puede compilar el contrato inteligente con versiones superiores.

Figura 15 *Versión de Solidity*

```
pragma solidity ^0.8.18;
```

En el siguiente fragmento se observa la importación de las bibliotecas de open Zeppelin, proporcionando una implementación estándar del Token ERC20 en la plataforma Ethereum, es así, que se define una interfaz común para los tokens fungibles en Ethereum, lo que significa que cualquier otro token de diferente tipo, puede ser compatible y no tiene derechos o comportamientos distintos en el momento de transaccionar.

Figura 16 *Librerías Open Zeppelin*

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
```

El contrato inteligente “UpsToken” es un token ERC20 que tiene la capacidad de quemar, comprimir tokens y agregar recompensas para los mineros.

El token sigue las reglas de ERC20Capped, lo que significa que tiene un suministro fijo, y las reglas de ERC20Burnable, lo que permite que los poseedores de tokens puedan quemarlos.

Figura 17 *Explicación ERC20 heredado del Contrato Inteligente*

```
//ERC20 HEREDAD0
contract UpsToken is ERC20Capped , ERC20Burnable{
    address payable public owner;
    uint256 public blockReward;

    constructor(uint256 cap, uint256 reward) ERC20("UpsToken", "UPS")ERC20Capped(cap * (10 ** decimals())) {
        owner = payable(msg.sender);
        _mint(owner, 70000000 * (10 ** decimals()));
        blockReward = reward * (10 ** decimals());
    }
    function _mint(address account, uint256 amount) internal virtual override(ERC20Capped, ERC20) {
        require(ERC20.totalSupply() + amount <= cap(), "ERC20Capped: cap exceeded");
        super._mint(account, amount);
    }
    function _mintMinerReward() internal {
        _mint(block.coinbase, blockReward);
    }
    function _beforeTokenTransfer(address from, address to, uint256 value) internal virtual override {
        if(from != address(0) && to != block.coinbase && block.coinbase != address(0)) {
            _mintMinerReward();
        }
        super._beforeTokenTransfer(from, to, value);
    }
    function setBlockReward(uint256 reward) public onlyOwner {
        blockReward = reward * (10 ** decimals());
    }
    function destroy() public onlyOwner {
        selfdestruct(owner);
    }
    modifier onlyOwner {
        require(msg.sender == owner, "Solo el propietario puede llamar a esta funcion");
        _;
    }
}
```

Las variables agregadas e inicializadas en el constructor son las siguientes:

- owner: Se refiere a la cuenta de usuario de Ethereum que va a desplegar el contrato.
- name: El nombre de usuario del token.
- symbol: El símbolo utilizado para representar el token.
- decimals: El número de decimales que va a tener el token.
- totalSupply: La cantidad total de tokens que se va a lanzar durante el despliegue.
- balances: Una estructura de tipo mapping (address, uint256) que permite identificar los saldos correspondientes a cada cuenta que posea tokens.

El token también cuenta con una función "beforeTokenTransfer" que añade una recompensa al minero que extrae con éxito un nuevo bloque. La recompensa del minero es una cantidad fija que está definida por la variable blockReward.

En definitiva, el Smart Contract “UpsToken” tiene las siguientes características y funcionalidades:

- Hereda los contratos ERC20Capped y ERC20Burnable, lo que permite que el token tenga un suministro fijo y admita la quema de tokens.
- Permite que los mineros sean recompensados por extraer nuevos bloques utilizando la función _mintMinerReward.
- Proporciona una función "beforeTokenTransfer" que se ejecuta antes de cada transferencia de tokens.
- Ofrece la opción de cambiar la recompensa del minero utilizando la función setBlockReward.
- Proporciona una función de destrucción que permite al propietario del contrato destruirlo y devolver el saldo restante a la dirección del propietario.

En la parte de las pruebas realizados se visualiza 15 test de comprobación del funcionamiento de la criptomoneda.

Figura 18 *Clase (UpsToken.js) de Test de validaciones de la criptomoneda.*

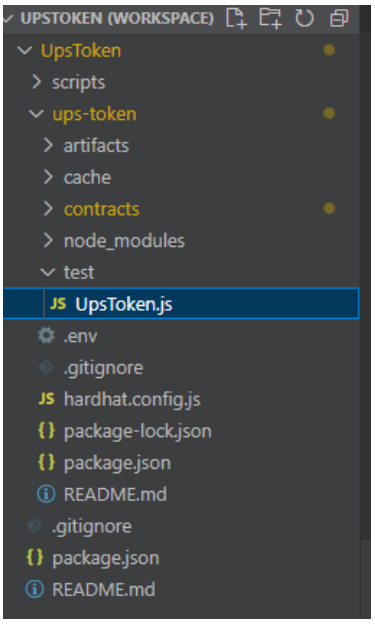
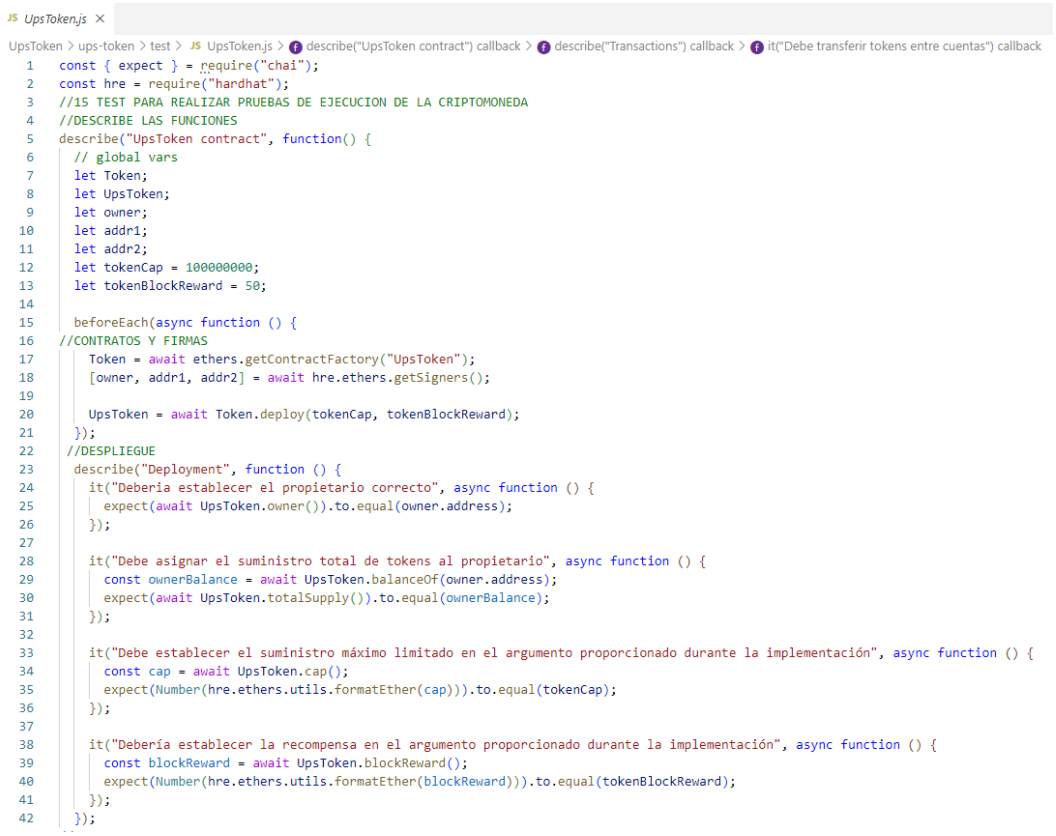


Figura 19 *Test de validaciones parte 1*



```
JS UpsToken.js ×
UpsToken > ups-token > test > JS UpsToken.js > describe("UpsToken contract") callback > describe("Transactions") callback > it("Debe transferir tokens entre cuentas") callback
1   const { expect } = require("chai");
2   const hre = require("hardhat");
3   //15 TEST PARA REALIZAR PRUEBAS DE EJECUCION DE LA CRIPTOMONEDA
4   //DESCRIBE LAS FUNCIONES
5   describe("UpsToken contract", function() {
6     // global vars
7     let Token;
8     let UpsToken;
9     let owner;
10    let addr1;
11    let addr2;
12    let tokenCap = 100000000;
13    let tokenBlockReward = 50;
14
15    beforeEach(async function () {
16  //CONTRATOS Y FIRMAS
17      Token = await ethers.getContractFactory("UpsToken");
18      [owner, addr1, addr2] = await hre.ethers.getSigners();
19
20      UpsToken = await Token.deploy(tokenCap, tokenBlockReward);
21    });
22   //DESPLIEGUE
23    describe("Deployment", function () {
24      it("Deberia establecer el propietario correcto", async function () {
25        expect(await UpsToken.owner()).to.equal(owner.address);
26      });
27
28      it("Debe asignar el suministro total de tokens al propietario", async function () {
29        const ownerBalance = await UpsToken.balanceOf(owner.address);
30        expect(await UpsToken.totalSupply()).to.equal(ownerBalance);
31      });
32
33      it("Debe establecer el suministro máximo limitado en el argumento proporcionado durante la implementación", async function () {
34        const cap = await UpsToken.cap();
35        expect(Number(hre.ethers.utils.formatEther(cap))).to.equal(tokenCap);
36      });
37
38      it("Debería establecer la recompensa en el argumento proporcionado durante la implementación", async function () {
39        const blockReward = await UpsToken.blockReward();
40        expect(Number(hre.ethers.utils.formatEther(blockReward))).to.equal(tokenBlockReward);
41      });
42    });
```

Nota: 11 En la figura 19 se describe la primera parte de los test de validaciones. (Elaborado por: Autor)

En esta primera parte de los test del contrato inteligente es una implementación JavaScript comprobable de un token ERC20.

En este conjunto de pruebas, hay varios casos diseñados para validar el comportamiento del contrato inteligente.

- "Debe transferir tokens entre cuentas": Esta prueba verifica que los tokens se puedan transferir exitosamente entre cuentas.
- "El remitente no tiene suficientes tokens para la transacción": Esta prueba asegura que la transferencia falla si el remitente no tiene suficientes tokens.

- "Debería actualizar su saldo": Esta prueba verifica que el proceso de transferencia actualice correctamente el saldo del remitente y del destinatario.

Cada prueba se ejecuta de forma asincrónica, lo que significa que se ejecutan en el orden en que se escriben, pero pueden finalizar en cualquiera. También tienen un tiempo de espera de 5 segundos para evitar colgar las pruebas.

Para poder ejecutar correctamente existe diferentes plugin como Rinkeby, EtherScan, cada uno con su plataforma web.

Para la segunda parte de las pruebas del contrato inteligente se despliega de la siguiente manera.

Figura 20 *Test de validaciones parte final*

UpsToken.js

UpsToken > ups-token > test > UpsToken.js > ...

```
43  //TRANSACCION
44    describe("Transactions", function () {
45      it("Debe transferir tokens entre cuentas", async function () {
46        // Transfer 50 tokens from owner to addr1
47        await UpsToken.transfer(addr1.address, 50);
48        const addr1Balance = await UpsToken.balanceOf(addr1.address);
49        expect(addr1Balance).to.equal(50);
50
51        // TRANSFERENCIA ENTRE addr1 to addr2
52        // se utiliza .connect(signer) para enviar transaccion a otra cuenta
53        await UpsToken.connect(addr1).transfer(addr2.address, 50);
54        const addr2Balance = await UpsToken.balanceOf(addr2.address);
55        expect(addr2Balance).to.equal(50);
56      });
57      it("El remitente no tiene suficientes token para la transaccion", async function () {
58        const initialOwnerBalance = await UpsToken.balanceOf(owner.address);
59        // Se puede enviar desde 1 token a addr1 (0 tokens) al propietario (1000000 tokens).
60        // `require` se evaluará la transaccion falsa y revertirá.
61        await expect(
62          UpsToken.connect(addr1).transfer(owner.address, 1)
63        ).to.be.revertedWith("ERC20: El monto de la transaccion excede el su saldo");
64
65        // Cuando se procede reverso el saldo del propietario no cambia.
66        expect(await UpsToken.balanceOf(owner.address)).to.equal(
67          initialOwnerBalance
68        );
69      });
70      it("Debería actualizar su saldo", async function () {
71        const initialOwnerBalance = await UpsToken.balanceOf(owner.address);
72        // Transferir 100 tokens del propietario a addr1.
73        await UpsToken.transfer(addr1.address, 100);
74        // Transferir otros 50 tokens del propietario a addr2.
75        await UpsToken.transfer(addr2.address, 50);
76        // Consultar el saldo.
77        const finalOwnerBalance = await UpsToken.balanceOf(owner.address);
78        expect(finalOwnerBalance).to.equal(initialOwnerBalance.sub(150));
79        const addr1Balance = await UpsToken.balanceOf(addr1.address);
80        expect(addr1Balance).to.equal(100);
81        const addr2Balance = await UpsToken.balanceOf(addr2.address);
82        expect(addr2Balance).to.equal(50);
83      });
84    });
85
86  });
```

Nota: 12 En la figura 20 se describe la parte final de los test de validaciones. (Elaborado por: Autor)

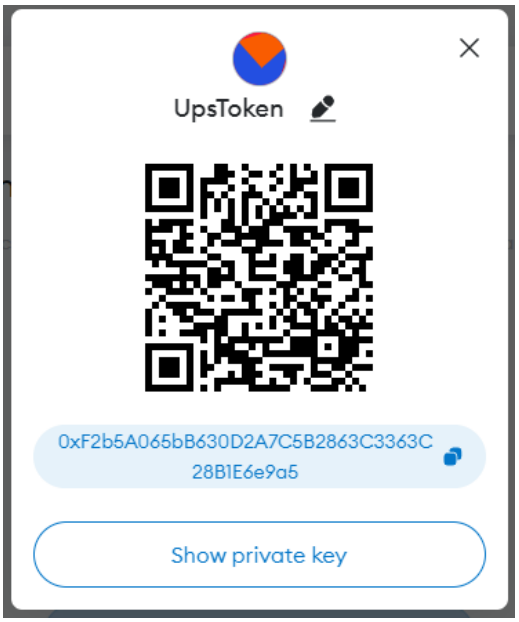
Completando este conjunto de pruebas para un contrato inteligente de UpsToken.

Se activan tres casos de prueba:

- Transferencia de tokens entre cuentas: esta prueba verifica que la función de transferencia transfiere tokens con éxito entre dos direcciones.
- Fondos insuficientes para la transacción: esta prueba verifica que se muestre un mensaje de error cuando el remitente no tiene suficientes tokens para completar la transacción.
- Actualizar saldo después de múltiples transacciones: esta prueba garantiza que el contrato inteligente actualice correctamente el saldo del propietario después de múltiples transacciones que involucran diferentes direcciones.

Una vez finalizado el desarrollo y despliegue del Smart Contract en la testnet de Ethereum utilizando la API de Infura, tendrá la posibilidad de acceder a la información a través de Etherscan, el mismo que se contextualiza como una plataforma que ofrece una interfaz que te permite buscar toda la información relacionada con el Smart Contract utilizando su dirección.

Figura 21 *Lanzamiento de la criptomoneda en la billetera Metamask*



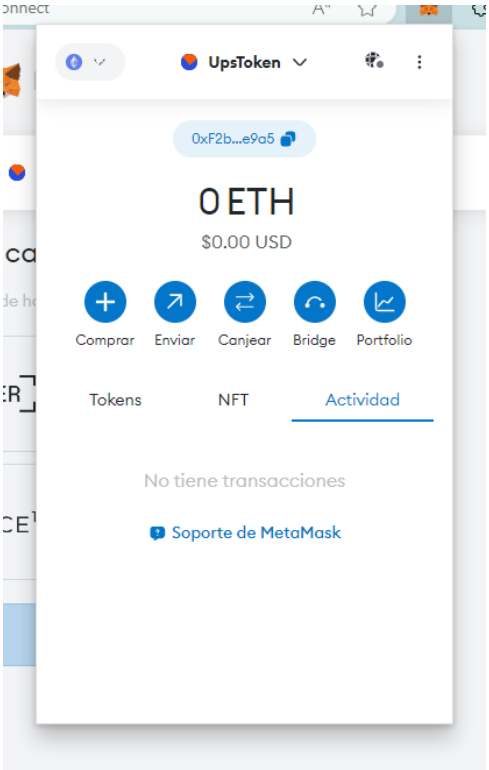
En la Figura 21 se visualiza la sección de nombre de la moneda virtual con la dirección de billetera y la cartera de un usuario.

El portafolio consta de dos secciones principales:

- Activos: esta sección muestra el saldo total de cada criptomoneda y token en la billetera del usuario. La cartera muestra el símbolo de la criptomoneda, el monto y su valor correspondiente en USD.
- Actividad: esta sección enumera el historial de transacciones del usuario, incluido el tipo de transacción, la criptomoneda involucrada, el monto de la transacción y la fecha y hora de la transacción.

Además de la cartera y el historial de transacciones, la imagen también proporciona soporte al usuario para MetaMask.

Figura 22 *Cuenta creada en la billetera Metamask*



En la Figura 22 se observa la configuración de la cuenta MetaMask conectado a UpsToken, visualizándose sus activos NFT (Non Fungible Token), donde se podrá revisar los detalles de la transacción de activación y confirmar la transacción.

Después de activar el contrato, puede comenzar a realizar diversas operaciones, como cancelar tokens, recopilar tokens o actualizar los precios de los tokens es importante mencionar que para realizar transacciones de tokens con la moneda virtual implementada UpsToken, debe estar conectado a la red Ethereum mediante MetaMask o una billetera similar.

CAPÍTULO IV

4. ANÁLISIS DE RESULTADOS

Para el presente proyecto técnico, se desarrolló una aplicación con un rol de administrador, en donde solo se requiere inicializar la cuenta con la billetera de Metamask. Es importante destacar que las cuentas deben estar vinculadas.

4.1. Conexión de la cuenta implementada en la billetera Metamask

Es esencial contar con la extensión de Metamask instalada en el navegador y conceder los permisos necesarios para interactuar con la página o aplicación que aloja el Smart Contract. De esta manera, la billetera puede llevar a cabo dicha interacción de manera adecuada.

Figura 23 *Selección de cuenta en Metamask*

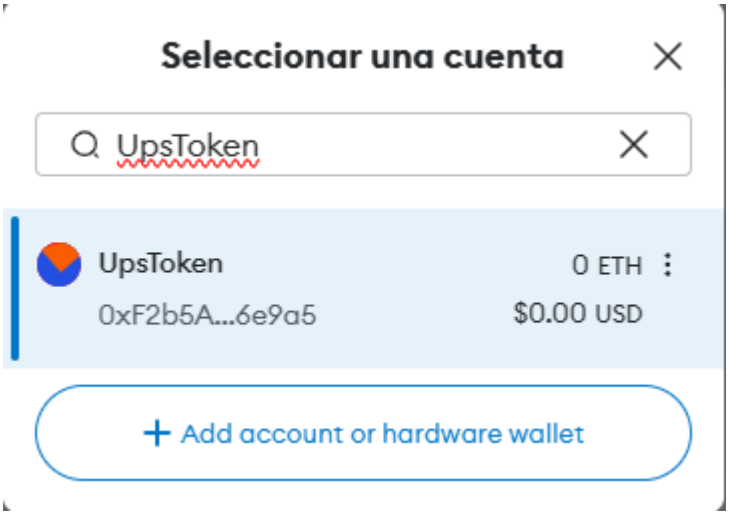
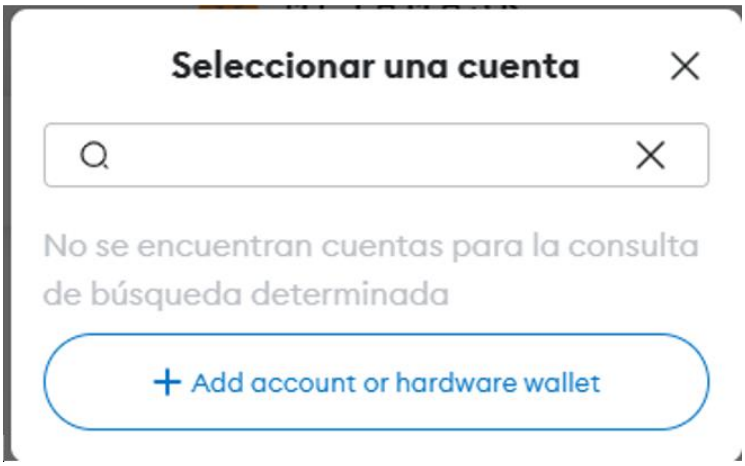


Figura 24 *Resultado fallido de búsqueda de la criptomoneda*



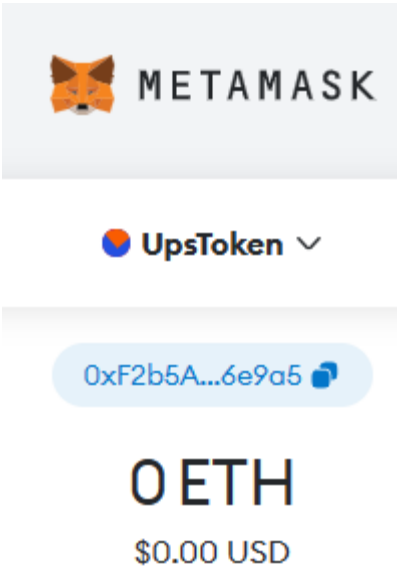
Si no cuenta con la extensión instalada en el navegador, no se podrá utilizar la aplicación. Si se intenta realizar una búsqueda, no se mostrará la moneda virtual implementada y se obtendrá los siguientes resultados.

La búsqueda de la cuenta se lo puede realizar de diferente manera, en primer lugar, con el nombre integrado desde Visual Studio Code a la billetera Metamask y en segundo término con la dirección desplegada desde el Smart Contract.

4.2. Funcionalidad de Smart Contract.

Desde la página oficial de Rinkeby se visualiza el acceso al contrato inteligente y la vinculación de la criptomoneda implementada con Solidity, cabe recalcar que es importante elegir la dirección que fue realizó en el contrato inteligente, en este caso la dirección es: 0xF2b5A065bB630D2A7C5B2863C3363C28B1E6e9a5.

Figura 25 *Dirección de cuenta de MetaMask*



Pruebas de verificacion de caja blanca:

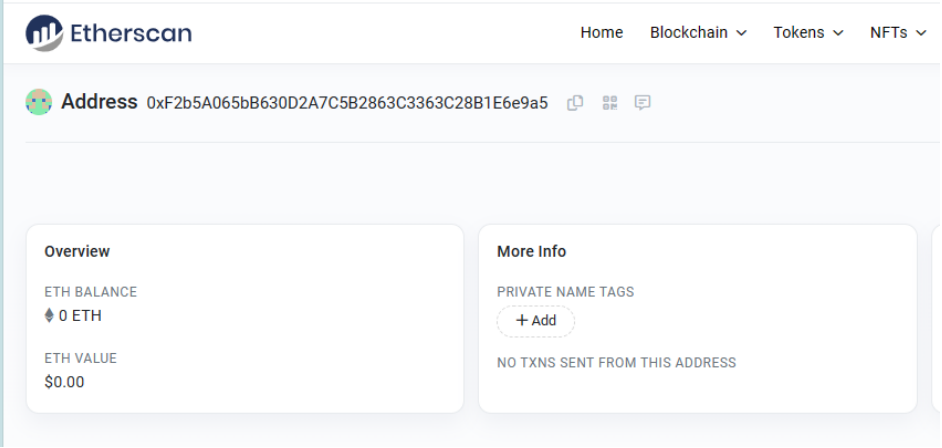
Para el proceso de finalización y comprobación que la criptomoneda se encuentre lanzada correctamente en Metamask cada caso de prueba sigue un patrón similar: se comienza inicializando el contrato inteligente de UpsToken y creando dos direcciones, addr1 y addr2. Luego, los casos de prueba realizan transacciones y verifican que los saldos de las direcciones se actualicen correctamente.

En el caso de prueba "Transferencia de tokens entre cuentas", el propietario transfiere por 50 tokens a addr2. Luego, la prueba verifica que los saldos de estas direcciones se actualicen correctamente.

En el caso de prueba "Fondos insuficientes para la transacción", la prueba intenta transferir 1 token de la dirección 1 al propietario. Dado que addr1 no tiene ningún token inicialmente, esta transacción falla y la prueba verifica y muestra el mensaje de error.

En el caso de prueba "Actualizar saldo después de múltiples transacciones", el propietario transfiere 100 tokens a addr1 y 50 tokens a addr2. Luego, la prueba verifica que el saldo del propietario esté actualizado correctamente después de estas transacciones.

Figura 26 *Visualización de criptomoneda desde Etherscan de Rinkeby.*



Nota: 13 (Fuente: Etherscan, 2023)

Una vez obtenido el resultado se observa de la siguiente manera desde Etherscan y Metamask la criptomoneda implementada mediante herramientas de código abierto de Ethereum.

CONCLUSIONES

Centrarse en un proyecto basado en el desarrollo de una criptomoneda mediante Ethereum puede variar ampliamente, en el ámbito de que se podrían incluir una mejor comprensión de la tecnología y su aplicabilidad, profundizar tanto en brindar nuevas soluciones innovadoras y limitaciones identificadas.

El proyecto proporcionó una oportunidad valiosa para adquirir habilidades técnicas en el lenguaje de programación Solidity y en la plataforma Ethereum en general.

En el proceso de investigación se determinó un resultado positivo frente a un nivel de seguridad que ofrece los algoritmos de prueba de trabajo (PoW), o la prueba de participación (PoS), mitigando las amenazas y brindando confiabilidad y seguridad desde la Blockchain al lanzar la criptomoneda en la plataforma MetaMask sin inconvenientes.

Se obtuvo una comprensión más profunda de cómo funcionan los contratos inteligentes, los tokens ERC-20 y otras funciones de Ethereum.

El estudio y la investigación también reveló desafíos y limitaciones que podrían enfrentar los proyectos de criptomonedas en el futuro, estos podrían ser la complejidad y entendimiento del funcionamiento dentro del sistema financiero, aceptación legal y regulatoria, así mismo como la volatilidad y estabilidad de las criptomonedas debido al manejo de inversiones o la velocidad a la que sus reservas pueden liquidarse.

RECOMENDACIONES

Durante la fase de investigación y planificación del proyecto, es fundamental tener una sólida comprensión de la tecnología Blockchain y la plataforma Ethereum, así como de las características y funcionalidades de las criptomonedas en general.

Prestar atención a la seguridad y protección de los activos digitales, incluyendo la implementación de medidas de seguridad robustas y protocolos de respaldo, esto incluye, la protección contra ataques cibernéticos en contra de la información de la criptomoneda, la prevención de pérdida de datos, así mismo la seguridad de contratos inteligentes certificando la integridad y la funcionalidad de estos contratos cumpliendo una normativa y confianza del usuario.

Durante la fase de desarrollo del proyecto, asegurarse de llevar a cabo pruebas exhaustivas y rigurosas para garantizar la estabilidad y la funcionalidad de la criptomoneda antes de su lanzamiento.

Es fundamental realizar el desarrollo utilizando las versiones más recientes de web3 y Solidity, ya que ofrecen mejoras en términos de seguridad y una mayor compatibilidad con nuevas tecnologías.

Finalmente, documentar todo el proceso de desarrollo en detalle, incluyendo cualquier desafío o limitación que se haya detectado durante el proceso, para que esto pueda ser utilizado como base para futuras investigaciones en el campo.

REFERENCIAS BIBLIOGRÁFICAS

Algorand. (2023). *Technologies Blockchain*. https://algorandtechnologies.com/es/resources/blog/algorand-smart-contract-architecture

Alharby, M., & van Moorsel, A. (2017). *BLOCKCHAIN-BASED SMART CONTRACTS: A SYSTEMATIC MAPPING STUDY*. 125–140. https://doi.org/10.5121/csit.2017.71011

Álvarez Díaz, L. J. (2019). *Criptomonedas: Evolución, crecimiento y perspectivas del Bitcoin Cryptocurrencies: Evolution, growth and perspectives of Bitcoin*. *25*(49), 130–142. https://doi.org/10.18004/pdfce/2076-054x/2019.025.49.130-142

Blockchain - Qué es, definición y concepto | 2023 | Economipedia. (s/f). Recuperado el 16 de noviembre de 2023, de https://economipedia.com/definiciones/blockchain.html

Conceptos. (s/f). Recuperado el 16 de noviembre de 2023, de https://www.gnupg.org/gph/es/manual/c190.html

Introduction | MetaMask developer documentation. (s/f). Recuperado el 16 de noviembre de 2023, de https://docs.metamask.io/wallet/

Kinsta Inc. (2023). *Qué es Node.js y por qué deberías usarlo*. https://kinsta.com/es/base-de-conocimiento/que-es-node-js/

Los ocho componentes de una blockchain pública y los ocho componentes de Ethereum | Ignacio G.R. Gavilán. (s/f). Recuperado el 16 de noviembre de 2023, de https://ignaciogavilan.com/los-ocho-componentes-de-una-blockchain-publica-y-los-ocho-componentes-de-ethereum/

Lucern, F. S., Student, S., Niya, S. R., & Bocek, T. (2017). *Design and Implementation of a Smart Contract Application*.

¿Qué es un Algoritmo de Consenso? | Binance Academy. (2022). https://academy.binance.com/es/articles/what-is-a-blockchain-consensus-algorithm

Welcome to Remix's documentation! — Remix - Ethereum IDE 1 documentation. (s/f). Recuperado el 16 de noviembre de 2023, de https://remix-ide.readthedocs.io/en/latest/