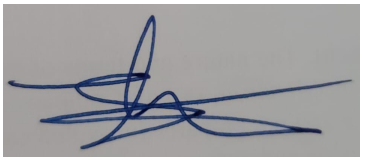
UNIVERSIDAD DE Belgrano
BUENOS AIRES - ARGENTINA

Utilización de tokens ERC-20 para adquisición y amortización de bienes o servicios referentes a instituciones Educativas

Alumno: Ignacio Arturo Fernandez Mendoza
Matricula: 502-11604
Carrera: Ingenieria en Informatica
Tutor: Leonel Jimenez



Agradecimientos:

Agradezco a la universidad de Belgrano, la cual me brindó no solo las herramientas y recursos para llegar a donde estoy ahora sino también amistades irremplazables. Agradezco también a nuestros profesores, compañeros y administrativos, a mi familia y amigos.

Resumen

En esta tesina se navegará a través de temas relacionados con Blockchain y criptomonedas, más específicamente la red ethereum y Tokens digitales/criptográficos. Incluyendo exchanges, DApps, Smart Contracts, cómo funciona la Ethereum virtual Machine y como funciona un programa en la blockchain. Por otro lado, se tratará de crear un reemplazo para las existentes monedas universitarias, el cual sea universal y tenga un respaldo tecnológico confiable, utilizando las tecnologías mencionadas anteriormente.

Abstract

In this thesis, topics related to Blockchain and cryptocurrencies will be navigated, more specifically the ethereum network and digital/cryptographic tokens. Including exchanges, DApps, Smart Contracts, how the Ethereum virtual Machine works and how a program works on the blockchain. On the other hand, it will try to create a replacement for the existing university currencies, which is universal and has reliable technological support, using the technologies mentioned above.

Indice

1 Introduccion

Con los avances tecnológicos de los últimos años, más y más gente se encuentra interesada en informarse en conceptos que previamente eran exclusivos para informáticos. Dentro de estos conceptos, están las famosas “criptomonedas” así como su base informática, las “Blockchain”. En vista del gran interés e inversiones que tuvo este campo los últimos años, logró desarrollarse al punto que cada día surgen tecnologías nuevas, o se mejoran las tecnologías ya existentes.

Junto con estas tecnologías mencionadas, existe un concepto que, si bien ya existía anteriormente, se expandió y se aplicó recientemente, junto con la red Ethereum. Este concepto son los “Tokens criptográficos”.

El objetivo de esta investigación es crear un token que pueda ser utilizado en instituciones educativas como forma de pago para adquirir productos o pagar por servicios, de forma universal y que al mismo tiempo, eduque a los estudiantes, y padre de los mismos, en el funcionamiento de esta tecnología que sigue creciendo dia a dia.

1.1 Descripción del problema

Hoy en día hay muchos alumnos que asisten a la universidad y, al mismo tiempo, aprenden y quieren invertir en criptomonedas o tokens criptográficos, o escuchan al respecto. Al mismo tiempo, alrededor del mundo hay múltiples instituciones que poseen un método de pago único para sus afiliados(alumnos, profesores, padres) el cual resulta obsoleto una vez que el alumno se gradúa o el profesor deja dicha institución. La solución que se propone en este proyecto es reemplazar todas por un token criptográfico, el cual sea universal y pueda utilizarse en caso de cambiar de universidad, realizar intercambios educativos o hasta poder intercambiarlo por dinero una vez se gradúen y dejen la universidad.

1.2. Marco metodológico

El enfoque de la investigación es cualitativo y Waterflow será el método para realizar los entregables; se considera la mejor opción ya que se trabajara el proyecto en una forma iterativa entre cada “milestone”. Finalmente se utilizará una metodología TDD, test driven design, para desarrollar el código, dado que este asegurara la funcionalidad del mismo una vez finalizado el desarrollo.

1.2.1. Objetivo

Profundizar el entendimiento sobre la red criptográfica ethereum y específicamente sobre el concepto de “Dapps” y smart contracts de forma tal que podamos hacer uso de ellos para la implementación de un token que será utilizado para realizar pagos por servicios y productos en instituciones educativas en todo el mundo, de forma universal y única.

1.2.2. Alcances

No se considerará la viabilidad del negocio financiero producto del desarrollo del token, el objetivo principal de este trabajo es explorar las tecnologías anteriormente mencionadas y determinar si las tecnologías descentralizadas existentes en la red Ethereum son provechosas para el uso masivo y merecen consideraciones a posteriori.
Se creará una página web para demostrar las utilidades del mismo, así como el código del token y el despliegue de todos los programas asociados, de forma local.
A manera de entregable:

- Documentación de uso y Documentación técnica
- Código referente a los smart contract
- Página web de interacción con el Smart Contract.
- Código perteneciente a desarrollos auxiliares para la demostración de la hipótesis, lease plataforma para la obtención de información y acceso a la ejecución de operaciones.

2 Marco conceptual y estado del Arte.

2.1 Entendiendo la red Ethereum.

2.1.1 Introduccion

Introducida en 2015, la red Ethereum es una plataforma "Open-source", descentralizada y basada en "blockchain", utilizada para su propia criptomoneda llamada "Ether". Esta plataforma permite el uso de Smart Contracts y las denominadas DApps (Decentralized applications) para ser creadas y ejecutadas sin tiempo muerto, fraude, control o interferencias por un tercero.

- Jake Frankfield [12-02-2021].

La plataforma Ethereum es una Blockchain general, con una máquina virtual (Ethereum Virtual Machine, EVM) para ejecutar Smart Contracts. Dado que el medio ambiente existe solo en blockchain en forma de máquina virtual, los contratos inteligentes están completamente aislados de la red, el sistema de archivos u otros procesos en las máquinas del nodo. Un lenguaje de alto nivel, "Turing-complete" fue creado para escribir contratos inteligentes en Ethereum. Sin embargo, ese lenguaje, Solidity, ahora se ha convertido en estándar también para otras plataformas con capacidades de utilizar Smart Contracts.

Solidity es similar a JavaScript en sintaxis, pero está escrito en un estilo completamente diferente. Una vez que se ha escrito un Smart Contract en Solidity, se compila en el código de bytes de Ethereum Virtual Machine (EVM) y luego se implementa en una dirección Ethereum específica. Para implementar e interactuar con contratos inteligentes en Ethereum, sin embargo, una biblioteca RPC especial de javaScript es utilizada junto con una API web

Ya que la programación de Smart Contracts comenzó con Ethereum y Solidity, es todavía una disciplina en desarrollo. El lenguaje Solidity tiene un número conocido de peculiaridades y una lista de cambios a venir, lo cual indica que el código siendo escrito en este momento puede volverse obsoleto en futuras versiones.

2.1.1.1 Transacciones

Según la documentación de ethereum, la denominación “Transacciones” se refiere a una instrucción firmada de forma criptográfica por una cuenta, y su función es actualizar el estado de la red ethereum. La transacción más básica utilizada en dicha red es el intercambio de Ether. Utilizando el ejemplo anterior, podríamos explicar mejor el funcionamiento y la composición de una transacción. Se analizará el caso siguiente.

Una persona (P1) quiere enviarle un ether a otra persona (P2), esto requiere que P1 sea debitado 1ETH y P2 sea acreditada 1ETH. Toda esta información y actualización de estados deberá representarse en una transacción.

Esta transacción, la cual va a alterar el estado de la red ethereum, necesita ser transmitida a la misma. Esta transmisión puede ser generada por cualquier nodo de la red y va a ser ejecutada por los denominados “Mineros”, quienes utilizan los recursos de sus equipos para actualizar el estado de la red y propagarlo al resto de los nodos.

Estas transacciones tienen una tarifa incluida, y deben ser minadas para resultar exitosas.
Esta tarifa se basa en el término “GAS”, el cual se refiere a la unidad que mide la cantidad de esfuerzo computacional requerida para ejecutar una operación en la red ethereum. Este esfuerzo tiene asignado un valor numérico y se utiliza para determinar la tarifa de cada transacción. Más adelante se explicará en detalle el funcionamiento del GAS y su estructura.

Toda transacción debe tener los siguientes atributos:

- Recipiente: La dirección de recepción
- Firma: es una forma de identificación del emisor, se genera cuando la llave privada del emisor firma la transacción y esta es aprobada por dicho emisor.
- Valor: la cantidad de ETH que el emisor está enviando al receptor.
- Data: Información adicional arbitraria para dicha transacción (por si se desea enviar un mensaje junto con dicha transacción)
- GasLimit: La cantidad máxima de GAS permitidas para realizar dicha transacción
- GasFee: La tarifa pagada por el emisor por cada unidad GAS utilizada.

Toda esta información es traducida al formato JSON y es intercambiada entre un usuario y la red, para luego ser ejecutada, y el resultado de todo este proceso resulta en una estructura similar a la siguiente:

Estructura de una transacción previo a ser firmada:

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000",
  gasPrice: "200",
  nonce: "0",
  value: "10000000000",
}
```

Estructura de una transacción luego de haber sido firmada y enviada:

```
{
  "id": 2,
  "jsonrpc": "2.0",
  "method": "account_signTransaction",
  "params": [
    {
      "from": "0x1923f626bb8dc025849e00f99c25fe2b2f7fb0db",
      "gas": "0x55555",
      "gasPrice": "0x1234",
      "input": "0xabcd",
      "nonce": "0x0",
      "to": "0x07a565b7ed7d7a678680a4c162885bedbb695fe0",
      "value": "0x1234"
    }
  ]
}
```

2.1.1.2 Agents

Un agente, o cuenta, en Ethereum se refiere a un ente con un balance(o billetera) de ethers, que pueden enviar transacciones a la red ethereum. Estos agentes pueden ser controlados por usuarios(ACU) o ser ejecutados como “Smart Contracts”(ASC).

Si bien ambos tipos de agentes pueden realizar las mismas funciones, recibir, almacenar y enviar ETH, así como interactuar con Smart Contracts en la red, estos tienen sus diferencias entre sí.

La primera de ellas es en el costo. Mientras que los ACU no tienen costo de creación, los ASC requieren un costo de mantenimiento ya que consumen almacenamiento en la red.

Por otro lado, si bien las ACU pueden iniciar transacciones libremente, las ASC solamente pueden iniciar una transacción en respuesta a otra.

Por último, las transacciones entre dos ACU solamente pueden ser en ETH, mientras que las ASC pueden disparar diferentes transacciones en respuesta a una recepción, como transferir tokens o hasta crear un nuevo Smart Contract.

Las cuentas en ethereum se componen de 4 atributos:

- *Nonce*: es un contador que indica la cantidad de transacciones realizadas por esta cuenta. Se utiliza para asegurar que las transacciones se procesen solo una vez y en las Cuentas de contratos indican el número de contratos creados por esta cuenta.

- *Balance*: indica el número de WEI asociados a esa cuenta(WEI es una fracción de éter. Cada ether está compuesto por 1e+18 WEI).

- *codeHash*: este hash se refiere al código de una cuenta en la EVM. Las cuentas de contrato tienen fragmentos de código programados que pueden realizar diferentes operaciones. Este código EVM se ejecuta si la cuenta recibe una llamada de mensaje. No se puede cambiar a diferencia de los otros campos de la cuenta. Todos esos fragmentos de código están contenidos en la base de datos de estado con sus correspondientes hashes para su posterior recuperación. Este valor hash se conoce como codeHash. Para las cuentas de propiedad externa, el campo codeHash es el hash de una cadena vacía.
- storageRoot: también llamado storageHash, Un hash de 256 bits del nodo raíz de un árbol de Merkle que codifica el contenido de almacenamiento de la cuenta (un mapeo entre valores

enteros de 256 bits). Este árbol codifica el hash del contenido de almacenamiento de esta cuenta y está vacío de forma predeterminada.

Al crear una cuenta se generan un par de llaves de acceso, una privada y una pública. Esta llave privada es la herramienta que una cuenta utiliza para firmar las transacciones, y está compuesta de hasta 64 caracteres hexadecimales los cuales pueden ser encriptados.
La llave pública se genera en base a la llave privada mediante un algoritmo llamado ECDSA(Elliptic Curve Digital Signature Algorithm). Se puede obtener agarrando los últimos 20 bytes del hash keccak-256 de la llave privada y agregando un 0x al principio.

2.1.2 Tokens y sus diferentes aplicaciones

"Un token es un bloque de código, en formato de Smart Contract, que corre sobre una blockchain. Este código describe cómo se comportará dicho token , y su base de datos simplemente mantiene un registro de cuántos tokens tiene cada persona"

- *Gonzalo Arzuaga, infobae.*

Resumiendo un poco la frase dicha anteriormente, un token funciona como alimento para el funcionamiento de aplicaciones o plataformas que funcionan, en este caso, dentro de la red ethereum (haciendo referencia a las Dapps.). La diferencia principal entre una token y una criptomoneda entonces, es que el token se utiliza dentro de un smart contract para realizar tareas específicas y están basadas en criptomonedas.

Los tokens basados en Ethereum se los denomina ERC-20, los cuales almacenan la dirección de la billetera a la que pertenecen así como el balance que les queda disponible para usar. Una vez que alguien envía un token, el balance del emisor disminuye, y aumenta el del receptor. Eso es todo, bien simple. Es un libro contable que registra quién tiene cuánto, quién transfirió cuánto y a quién. De esta manera se hace fácil para cualquier startup crear y lanzar un nuevo token al mercado.

2.1.2.1 Token como activo Financiero (security Token)

Un token se considera un activo financiero si se puede comprar y vender en mercados públicos(también conocido como trading).
A mediados de 2017, la SEC (Securities and exchange commission) de Estados Unidos advirtió que si se clasifican las token de un ICO como "security" cae bajo la regulación que tiene cualquier activo financiero normal y bajo sus lineamientos y regulaciones estrictas. A partir de ese momento, ninguna ICO salió a ofrecer tokens que puedan ser catalogados como securities para la SEC para evitar escándalos o problemas legales.

La SEC establece que para que un token sea considerado "security" tiene que pasar el “Howey test". Este test consiste en verificar que:
- NO es una inversión de dinero.
- NO tiene expectativas de lucro.
- NO está basado principalmente en el esfuerzo de otros.

2.1.2.2 Token de utilidad (Utility Token)

Generalmente, los tokens caen dentro de la primer categoría pero si estos no cumplen con alguno de los puntos del test Howey no puede considerarse un “Security”

Un token de utilidad es, en pocas palabras, una promesa de uso futuro en una plataforma para quien lo posee. Para desarrollar más sobre esta explicación se tomará como ejemplo la red File Coin, la cual planea ofrecer almacenamiento descentralizado en la nube, tomando espacio disponible en los discos duros de las computadoras de los miembros de la red.
Quien posea tokens ICO de la red File Coin, podrá utilizarlos para comprar más almacenamiento en dicha red cuando lancen el servicio. Esta propuesta tuvo una gran repercusión, consiguiendo una inversión de 257 millones de dólares americanos pocos días tras su anuncio.

En casos como este, el valor del token estará dado por la demanda de los usuarios para alimentar la Dapp mencionada y, al tener una utilidad dentro de la red, esta no caería sobre la norma establecida por la SEC.

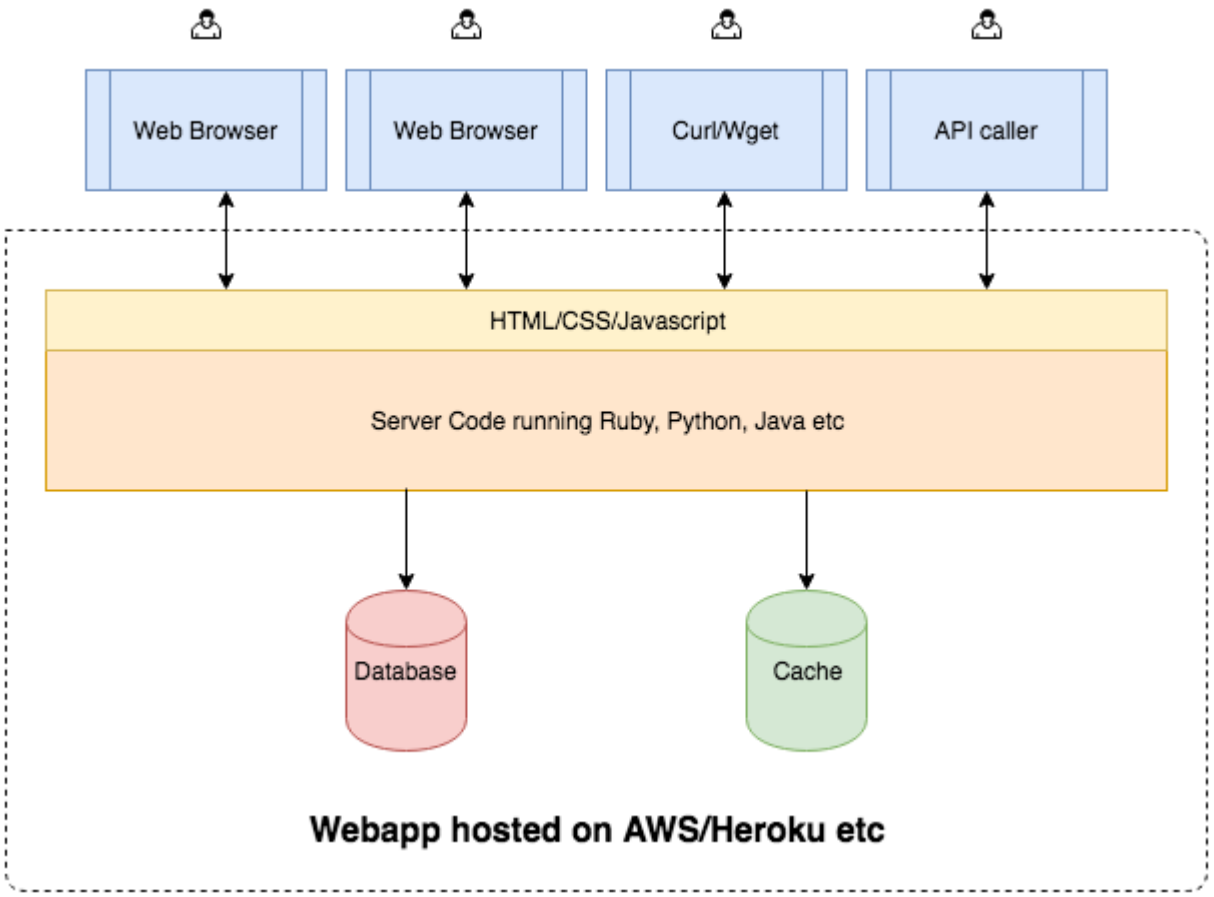
2.1.2.3 Token de acciones (Equity Token)

La última y menos común de la lista es el token como acción de una empresa. Con este tipo de token es mucho más fácil tener acciones de una empresa, principalmente una startup.
Una de las ventajas que tiene en comparación con las acciones típicas es que, a través de los tokens, los accionistas pueden tener un rol más activo en el gobierno de ese emprendimiento.
Por ejemplo, es posible hacer votaciones más rápidas, fáciles y transparentes a través de una blockchain.

Ya que el marco regulatorio para este tipo de tokens es escaso, para no decir nulo, pocos emprendimientos optaron por su utilización. Aunque se espera que en un futuro cercano estos tokens comiencen a tener más presencia.

2.1.3 Arquitectura

Una de las mejores formas de entender la arquitectura de la red ethereum es compararla con una red cliente/servidor tradicional. Para ello utilizaremos la arquitectura de una webapp convencional. A continuación se puede apreciar, a alto nivel, un diagrama de una webapp genérica.



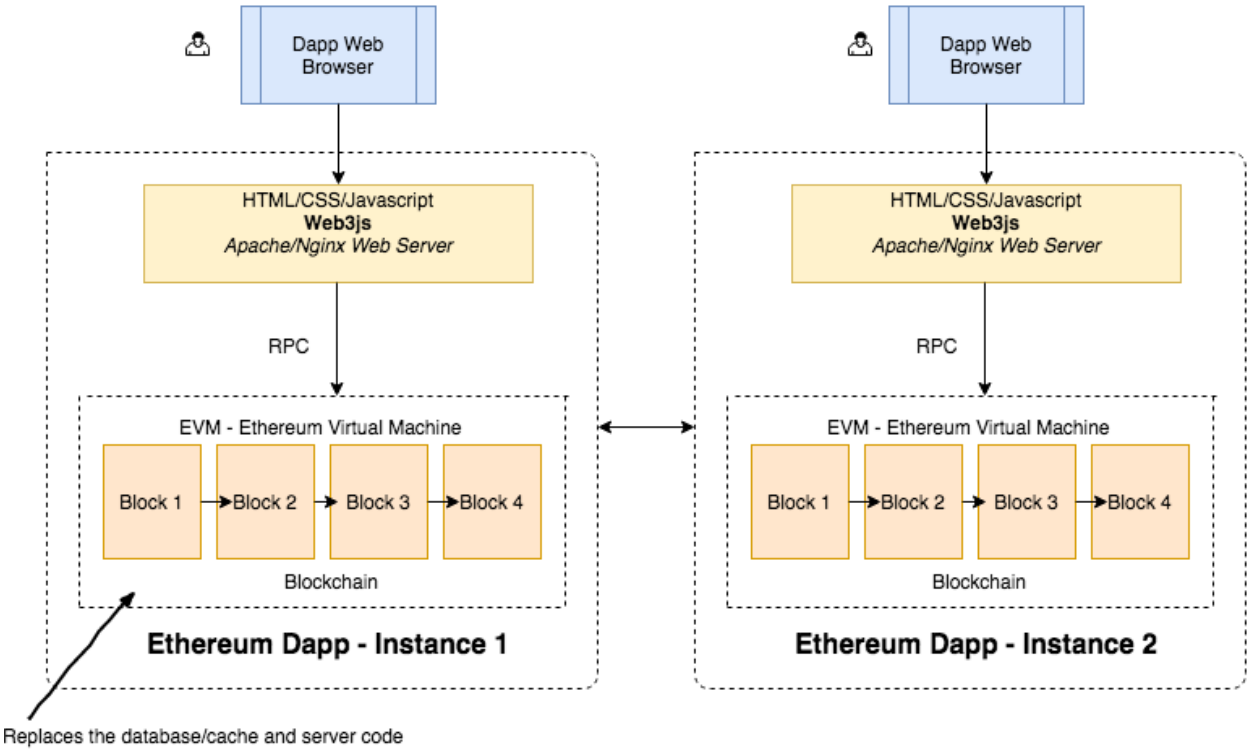
Una aplicación web típica consiste de “server side code”, el cual generalmente es escrito en un lenguaje de backend como java, C#, Python, etc… y el código frontend es implementado utilizando HTML/CSS/Javascript. Esta aplicación es luego hospedada ya sea en servidores propietarios o a través de un proveedor de host como Azure, AWS, entre otros.

Los usuarios entonces interactúan con la aplicación web utilizando un buscador (Internet explorer, Chrome, opera), línea de comando (curl/wget) o por medio de una API.

Lo importante a notar en este tipo de arquitecturas es que hay una única aplicación donde los usuarios interactúan haciendo requests, esperando que se procese el mismo e interactuando con la base de datos y/o caché.

Por otro lado, en la red ethereum cada cliente(representado como browser en el gráfico siguiente) se comunica con su propia instancia de la aplicación. No existe un servidor central al que todos los clientes se comuniquen. Esto significa, en un mundo descentralizado ideal, que cada persona que quiera interactuar con una Dapp necesitará una copia entera de la blockchain corriendo en su dispositivo.

Aunque no existe en un mundo descentralizado ideal, la idea detrás de la descentralización es no depender de un solo servidor. De esta manera, la comunidad introdujo soluciones donde no se tiene que sacrificar gran parte de la capacidad o ram de tu dispositivo al tener una copia de la blockchain corriendo, pero sin comprometer el aspecto descentralizado de la red.



La red ethereum está basada en una blockchain entonces, y esta tiene dos conceptos fundamentales:

- Base de datos: cada transacción en la red se almacena en la blockchain. Cuando se inicia la aplicación se considera una transacción. Si se tiene por ejemplo, una aplicación de votación, que permite a los usuarios votar candidatos para cumplir cierta función, cada voto por un candidato se considera una transacción. Todas estas transacciones son públicas y cualquiera puede ver esto y verificarlo. Esta información no puede ser manipulada de ninguna forma. Para asegurarse de que todos los nodos de la red tengan la misma copia de la información, y para asegurarse de que no haya información inválida en la base de datos, ethereum usa la llamada “Proof of work” para asegurar la red.

- Código: la base de datos de una blockchain solo almacena transacciones. La lógica para votar por el candidato, recopilar los votos, etc… Está almacenada en los mencionados “Smart Contracts”. Este código se crea en un lenguaje llamado solidity y es compilado en ethereum bytecode para ser lanzado en la blockchain. De esta forma, la blockchain no solo almacena las transacciones sino también los contratos.

De esta manera, se puede definir que la blockchain de ethereum almacena la información, el código y también corre dicho código en la Ethereum Virtual Machine (EVM).

2.1.4 EVM (Ethereum Virtual Machine)

La Ethereum Virtual Machine o EVM, es una máquina virtual que forma parte del ecosistema blockchain de Ethereum. Esta es capaz de ejecutar una amplia gama de instrucciones que le permiten una gran flexibilidad a la hora de realizar distintas operaciones.

Sin embargo, para hacer más sencilla la programación para esta máquina virtual se creó un lenguaje especializado de alto nivel llamado Solidity. A través de este lenguaje de programación se facilita la creación de los smart contracts. En primer lugar se transforma Solidity a los códigos de operación *(OP_CODES)* y luego a un bytecode. Este bytecode es finalmente ejecutado por la EVM para realizar las operaciones especificadas en un smart contract. Todo ello hace que la EVM pueda funcionar como un computador de verdad, ejecutando desde las más sencillas hasta las más complejas operaciones.

En pocas palabras, dada la característica descentralizada de Ethereum y su capacidad de almacenar smart contracts en los nodos de la red, y que con EVM pueden ejecutarse las órdenes programadas en dichos smart contracts, Ethereum se convierte en un gran ordenador mundial descentralizado. Uno que es capaz de realizar instrucciones que lleven a la resolución de cualquier tarea específica. De hecho, puede resolver casi cualquier problema computacional y todo esto ocurre dentro de la misma red Ethereum.

Todas estas instrucciones se encuentran escritas en los llamados Smart Contracts de Ethereum. Estos contratos son un tipo de cuenta que posee su propio código, y se habilita desde el mismo momento en el que llegan a la blockchain. Lo mejor de todo es que para poder usar el potencial de EVM tan solo debemos tener algo de ether e interactuar con alguna DApp, contrato inteligente o hacer nuestro propio contrato. No existe ninguna limitación, cualquier puede aprovechar el poder que EVM tiene a su disposición.

2.1.4.1 ¿Cómo funciona la Ethereum Virtual Machine (EVM)?

Con la finalidad de evitar que algún programador pueda atentar contra la seguridad de la red, la EVM realiza una completa abstracción del sistema. Manejando el acceso a los recursos de los computadores y limitando sus acciones en un ambiente controlado o de máquina virtual. A su vez, la EVM permite simplificar el desarrollo y actualización de aplicaciones y características disponibles para las aplicaciones descentralizadas.

La EVM permite el diseño y la ejecución de smart contract. Además gracias a Solidity crear estos smart contracts resulta muy sencillo y manejable. Aunque la EVM por sí misma no es capaz de ejecutar directamente este lenguaje, si lo hace por medio de una técnica conocida como compilación de instrucciones.

La EVM utiliza estos códigos de operación (OP_CODES) para realizar tareas específicas. Estas limitaciones llevan al EVM a crear métodos de trabajo que le permitan realizar sus operaciones, entre ellas la más importante es la memoria de contratos. Dicha memoria sirve para almacenar información a la que la EVM puede acceder rápidamente. Por otra parte, para almacenar datos de manera indefinida y hacerlos accesibles para futuras ejecuciones de contratos, se puede usar el almacenamiento por contratos. Esta actúa esencialmente como una base de datos pública, desde la cual los valores se pueden leer externamente sin tener que enviar una transacción al contrato, es decir, sin comisiones.

2.1.4.2 OP_CODES y Bytecode, las fundaciones de la Ethereum Virtual Machine

Los *OP_CODES* son una parte muy importante y esencial de la EVM. Estos códigos de operación son los que definen las operaciones válidas que la EVM puede realizar. En EVM existe la capacidad de ejecutar hasta 256 *OP_CODES* distintos, aunque en la actualidad no existen esta cantidad de códigos definidos. La razón es que el desarrollo de EVM no ha requerido de la creación de tales códigos y los desarrolladores son cuidadosos con incluir nuevos códigos debido a las limitaciones en sus números.

Un aspecto importante de los OP_CODES, es que este es un nivel intermedio de programación para la EVM. El primer nivel vendría dado por Solidity y los lenguajes de programación de alto nivel similares a este. Un segundo nivel de profundidad serían los OP_CODES. Por último, tendríamos el bytecode resultado de compilar los OP_CODES en el equivalente a lenguaje máquina de la EVM, algo prácticamente imposible de entender y escribir por un ser humano.

Sin embargo, al ser Ethereum una blockchain pública y ser un proyecto que aboga por la apertura y transparencia, el lenguaje bytecode de la EVM se puede descompilar. Es decir, podemos transformar el bytecode a OP_CODES y de allí llevarlo a un lenguaje cercano a Solidity. Esto es importante puesto que brinda a EVM la capacidad de mantener de forma abierta y clara el contenido de un smart contract. Además de permitir reconocer la interfaz binaria de aplicación(ABI) de la que dispone, la cual se encarga de codificar y decodificar información que entra o sale como ocurrencia cada transacción. Un dato importante puesto que se trata básicamente de cómo puede codificar llamadas de un contrato para el EVM y, a su vez, cómo leer los datos de las transacciones que genera dicho contrato.

2.1.4.3 Evolución futura de la EVM

La red de Ethereum ha sido nombrada como la computadora global debido a que cada uno de los nodos conectados a la red, ejecuta una instancia de la EVM. Al mismo tiempo que todos realizan las instrucciones de forma idéntica con el fin de lograr y mantener un consenso sobre el estado del sistema. Esta particularidad de Ethereum hace que su cálculo sea más lento y costoso en comparación con un ordenador común. Pero le brinda mayores ventajas, como por ejemplo una alta defensa a las fallas bizantinas, una mayor protección e integridad de los datos y una mayor resistencia a la censura.

Así, Ethereum funciona como una computadora mundial descentralizada de uso general en una red entre pares. Los smart contract y las DApps desarrolladas en la EVM podrían incluso asumir las funciones de Internet tal y como las conocemos. Además, podría permitirnos la creación de economías más estables ya que su funcionamiento puede ser garantizado y no está sujeto a interpretaciones humanas.

Con la aplicación de elementos como la distribución de archivos, la ejecución de contratos inteligentes, las aplicaciones descentralizadas y muchos otros, se elimina la confianza en terceros, ya que si no se cumplen los términos programados, simplemente el contrato no se ejecutará.

2.1.4.4 ¿La EVM puede ampliarse en funcionalidades?

Una de las principales características de la EVM es que esta puede ampliarse en funcionalidades, pero los desarrolladores son cuidadosos en incluir nuevas debido a la poca capacidad de agregar OP_CODES al sistema. Pese a ello, la EVM puede mejorarse para ser incluida en toda clase de dispositivos y es un trabajo en progreso dentro de la comunidad Ethereum con el fin de masificar aún más su uso.

2.1.4.5 Característica de las EVM

La Ethereum Virtual Machine es un software que posee muchas características o cualidades tanto positivas como negativas. Entre ellas podemos mencionar:

- La EVM está enfocada en proporcionar seguridad y ejecutar códigos no confiables en computadoras de todo el mundo.

- Las aplicaciones descentralizadas y los contratos inteligentes desarrollados en la EVM son completamente descentralizados y distribuidos. Por lo que no requiere de la participación de terceros, ni pueden ser modificadas o alteradas.

- La EVM permite el desarrollo de una mayor cantidad de aplicaciones, y que éstas puedan ejecutarse sobre una misma red blockchain, sin afectar a otras operaciones.

- Los contratos inteligentes diseñados en la EVM son invariables y pueden ejecutarse y hacerse cumplir por sí mismo, de una manera autónoma y automática. Con lo que se elimina la burocracia, los altos costos y el tiempo de espera típicos en los contratos tradicionales.

- La EVM es sustancialmente menos eficiente que muchas otras máquinas virtuales convencionales. Esto se debe a que principalmente su diseño se basó en la utilidad del momento y no en el alto rendimiento.

- Los cambios y mejoras experimentados por la EVM han sido pocos hasta ahora. Por lo que no está optimizada en cuanto a la velocidad para distintas plataformas de hardware.

- El diseño de la EVM no está dirigido a la portabilidad, lo que limita los espacios en los que dicha máquina virtual puede implementarse.

2.1.5 Ethereum testnet

Además de la red principal, existen redes de prueba públicas. Estas redes pueden ser utilizadas por los desarrolladores de protocolos o los desarrolladores de contratos inteligentes para probar tanto las actualizaciones aplicadas sobre los protocolos como en los contratos inteligentes en un entorno similar a la producción antes de la implementación en la red principal (mainnet). Este es un concepto análogo a los servidores de producción frente a los de almacenamiento intermedio.

La mayoría de las redes de prueba utilizan un *mecanismo de consenso de prueba de autoridad*. Esto significa que se elige una pequeña cantidad de nodos para validar transacciones y crear nuevos bloques, poniendo su identidad en el proceso. Es difícil incentivar la minería en una red de prueba usando un sistema de prueba de trabajo ya que puede dejarla vulnerable.

2.1.5.1 Ejemplos de testnets

- Görli: *Una red de prueba de prueba de autoridad que funciona con todos los clientes.*
- Kovan: *Una prueba de red de prueba para aquellos que ejecutan clientes de Open Ethereum.*
- Rinkeby: *Una prueba de red de prueba para aquellos que ejecutan el cliente Geth.*
- Ropsten: *Testnet de prueba de trabajo. Esto significa que es la mejor representación homogénea de Ethereum.*

2.1.5.2 Testnet Faucets

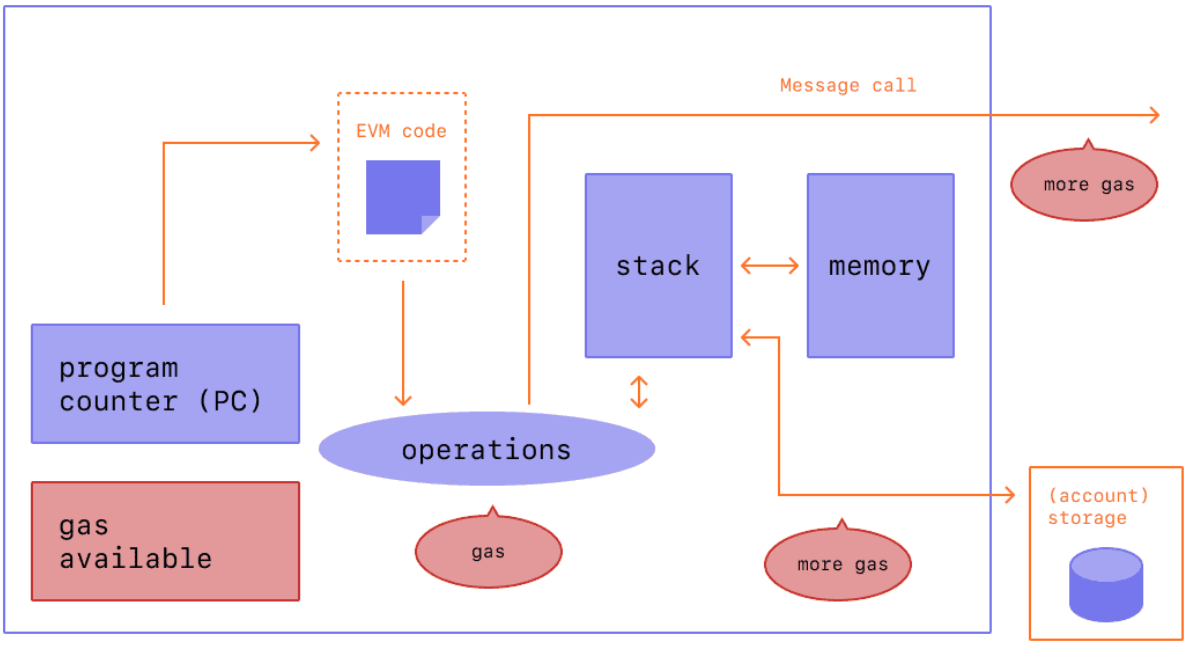
ETH en redes de prueba no tiene valor real; por lo tanto, no hay mercados para testnet ETH. Dado que necesita ETH para interactuar realmente con Ethereum, la mayoría de las personas obtienen ETH de testnet de los faucets. La mayoría de las faucets son aplicaciones web en las que puede ingresar una dirección a la que solicita que se envíe ETH. Por lo tanto existen Faucets (Grifos) en cada testnet que facilitan el fondeo de ETH para pruebas, algunos son:

- Grifo Görli
- Grifo de Kovan
- Grifo Rinkeby
- Grifo ropsten

2.1.6 Gas

Gas se refiere a la unidad que mide la cantidad de esfuerzo computacional requerido para ejecutar operaciones específicas en la red Ethereum.

Dado que cada transacción de Ethereum requiere recursos computacionales para ejecutarse, cada transacción requiere una tarifa. Gas se refiere a la tarifa requerida para realizar con éxito una transacción en Ethereum.

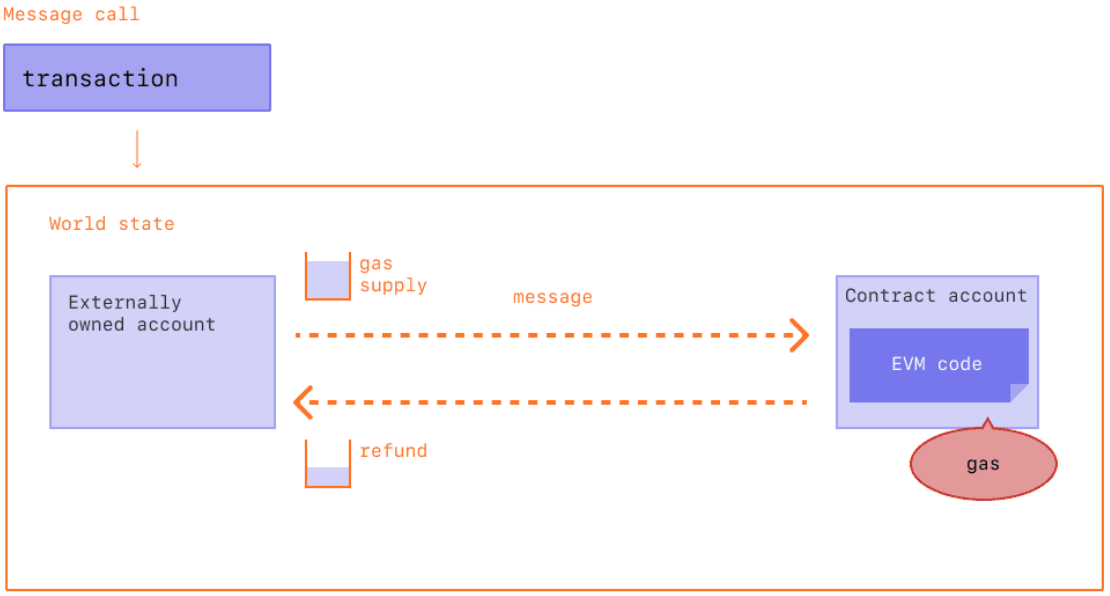


En esencia, las tarifas de gas se pagan en la moneda nativa de Ethereum, éter (ETH). Los precios del gas se indican en gwei, que en sí mismo es una denominación de ETH; cada gwei es igual a 0.000000001 ETH (10-9 ETH). Por ejemplo, en lugar de decir que su gas cuesta 0,000000001 éter, puede decir que su gas cuesta 1 gwei. Se apreciará mejor en el siguiente ejemplo:

Alice tiene que pagarle a Bob 1ETH. En la transacción, el límite de gas es de 21.000 unidades y el precio del gas es de 200 gwei.

La tarifa total será: Unidades de gas (límite) * Precio del gas por unidad, es decir, 21,000 * 200 = 4,200,000 gwei o 0.0042 ETH

Ahora, cuando Alice envíe el dinero, se deducirán 1.0042 ETH de la cuenta de Alice. A Bob se le acreditará 1.0000 ETH. El minero obtiene 0,0042 ETH.



2.1.6.1 Limites de Gas

El límite de gas se refiere a la cantidad máxima de gas que se está dispuesto a consumir en una transacción. Las transacciones más complicadas, que involucran contratos inteligentes, requieren más trabajo computacional, por lo que requieren un límite de gas más alto que un simple pago. Una transferencia estándar de ETH requiere un límite de gas de 21.000 unidades de gas.

Por ejemplo, si establece un límite de gas de 50.000 para una simple transferencia de ETH, el EVM consumiría 21.000 y recupera los 29.000 restantes. Sin embargo, si especifica muy poco gas, por ejemplo, un límite de gas de 20,000 para una simple transferencia de ETH, el EVM consumirá sus 20,000 unidades de gas intentando cumplir con el txn, pero no se completará. Luego, el EVM revierte cualquier cambio, pero dado que el minero ya ha realizado 20.000 unidades de gas, ese gas se consume.

2.1.6.2 Precio del Gas

El precio del gas se refiere a la cantidad de éter que está dispuesto a pagar por cada unidad de gas, y esto generalmente se mide en 'gwei'.

2.1.6.3 Subas del precio del Gas

Las altas tarifas de gas se deben a la popularidad de Ethereum. Realizar cualquier operación en Ethereum requiere consumir gas y el espacio de gas es limitado por bloque. Esto incluye cálculos, almacenamiento o manipulación de datos o transferencia de fichas, cada una de las cuales consumen diferentes cantidades de unidades de "gas". A medida que la funcionalidad *dapp* se vuelve más compleja, la cantidad de operaciones que realiza un contrato inteligente también aumenta, lo que significa que cada transacción ocupa más espacio de un bloque de tamaño limitado. Si hay demasiada demanda, los usuarios deben ofrecer un precio de gas más alto para intentar superar las transacciones de otros usuarios. Un precio más alto puede hacer que sea más probable que su transacción pase al siguiente bloque.

El precio del gas por sí solo no determina realmente cuánto tenemos que pagar por una transacción en particular. Para calcular la tarifa de transacción, tenemos que multiplicar el gas utilizado por el precio del gas, que se mide en gwei.

2.1.6.4 Iniciativas para reducir costos de Gas

Con las nuevas actualizaciones de red de Ethereum 2.0 (también conocido como Eth2 o Serenity). En última instancia, esto debería abordar algunos de los problemas de las tarifas del gas, lo que a su vez permitirá que la plataforma procese miles de transacciones por segundo y escale a nivel mundial.

El escalado de capa 2 es una iniciativa principal para mejorar en gran medida los costos del gas, la experiencia del usuario y la escalabilidad.

El nuevo modelo de prueba de participación debería reducir el alto consumo de energía y la dependencia de hardware especializado. El nuevo sistema PoS se introdujo en Beacon Chain. Esta cadena permitirá que la red Ethereum descentralizada llegue a un acuerdo y mantenga la red segura, pero evitará un alto uso de energía al requerir un compromiso financiero.

Cualquiera con al menos 32 ETH puede apostarlos y convertirse en un validador responsable de procesar transacciones, proponer nuevos bloques para agregar a la cadena de bloques y almacenar datos. Los usuarios que tienen menos de 32 ETH pueden unirse a grupos de participación.

2.6.1.5 Estrategias para la reduccion de Gas

Si está buscando reducir los costos de gas para su ETH, puede establecer el precio de sus propias tarifas de gas y elegir el nivel de prioridad de su transacción. Los mineros 'trabajarán' y ejecutarán transacciones que ofrezcan un precio de gas más alto, ya que pueden mantener las tarifas que usted paga y estarán menos inclinados a ejecutar transacciones con tarifas de gas más bajas establecidas. El precio del gas que establezca es cuánto está dispuesto a pagar por unidad de gas. Sin embargo, si establece la cantidad de gas demasiado baja, no podrá enviar su ETH ya que se quedará sin gasolina, entonces tendrá que volver a enviar su transacción, lo que le costará más en tarifas de gas. Puede hacer esto desde algunos proveedores de billetera al enviar ETH.

Algunas herramientas para el monitoreo del Gas son:

- Etherscan
- GasNow

2.2 Tokens ERC-20

Como se mencionó antes, las tokens pueden representar cualquier cosa en ethereum. Ya sea un monstruo de un juego, reputación para una plataforma online, una moneda fiat como el dólar. Para esto se necesita tener un estándar sólido, que permita a los desarrolladores crear aplicaciones para token que sean interoperables con otros productos y servicios. Ahí es donde el estándar ERC-20 entra en acción.

2.2.1 Qué son los tokens ERC-20 ?

ERC-20 se refiere al estándar utilizado para los tokens en la red ethereum. En otras palabras, presentan las propiedades que deben tener las tokens para ser exactamente iguales(en tipo y valor). En ese sentido, una token ERC-20 funciona similar a las criptomonedas en el sentido que, 1 token siempre va a tener el mismo valor que otra. Otro nombre para las ERC-20 es “Fungible Tokens” o tokens tangibles, debido a la propiedad mencionada anteriormente.

Este estándar, provee funcionalidades a las tokens que lo cumplen las cuales incluyen:

- Transferir tokens de una cuenta a otra.

- Obtener el balance actual de una cuenta
- Obtener la cantidad de oferta del token disponible en la red
- Aprobar la cantidad de un token que un tercero puede gastar por una cuenta

La documentación de ethereum.org nos provee los métodos necesarios para crear un token que sea considerado ERC-20. Según dicha documentación, se deberán introducir los siguientes métodos y eventos en el Smart Contract:

```
1    function name() public view returns (string)
2    function symbol() public view returns (string)
3    function decimals() public view returns (uint8)
4    function totalSupply() public view returns (uint256)
5    function balanceOf(address _owner) public view returns (uint256
     balance)
6    function transfer(address _to, uint256 _value) public returns (bool
     success)
7    function transferFrom(address _from, address _to, uint256 _value)
     public returns (bool success)
8    function approve(address _spender, uint256 _value) public returns (bool
     success)
9    function allowance(address _owner, address _spender) public view
     returns (uint256 remaining)
10
```

```
1    event Transfer(address indexed _from, address indexed _to, uint256
     _value)
2    event Approval(address indexed _owner, address indexed _spender, uint256
     _value)
3
```

2.2.2 Otros tipos de tokens utilizados en la red Ethereum.

2.2.2.1 ERC-777

Aparte del estándar ERC-20, en ethereum existen otros 2 estándares que valen la pena mencionar. El primero de ellos es ERC-777 que, según ellos, es la versión mejorada de ERC-20.

Este estándar agrega dos características muy importantes al funcionamiento del anterior ERC-20, así como compatibilidad con los mismos, siendo que se puede interactuar con estos como si fueran ERC-20.

Ganchos o **"Hooks"**, es la primera propiedad agregada a las tokens ERC-777. Estos hooks son funciones descritas en el smart contract que son llamados cuando se entregan o reciben tokens. Esto permite al Smart Contract reaccionar ante estas transacciones.

La utilización de hooks provee mejoras al estándar ERC-20 ya que:

- Permiten enviar tokens y notificar al contrato de dicho envío en una sola transacción, a diferencia de las ERC-20 que requieren una doble llamada.
- Los contratos que no tienen hooks registrados son incompatibles con las tokens ERC-777. El contrato aborta la transacción si el contrato receptor no tienen registrado un hook, para evitar transferencias accidentales a contratos no ERC-777
- Los hooks pueden rechazar transacciones.

La segunda propiedad importante que proveen los ERC-777 es la utilización de **Decimales**, lo cual soluciona la confusión de estos causada en los ERC-20.

2.2.2.2 ERC-721: NFTs

NFTs, o Non-fungible tokens, son un tipo de token utilizado para representar algo único. Lo bueno de este tipo de tokens es la versatilidad ya que pueden ser utilizados para representar una variedad de cosas como arte, propiedades, objetos coleccionables, entre otros.

La diferencia principal entre este tipo de tokens y las anteriormente mencionadas es que , lo que las define no es su precio sino sus características únicas. Ya que cada NFT es única, esa característica destaca su valor.

Las NFTs tienen tanto potencial que se dedicó un estándar específico para ellas dentro de la red ethereum: ERC-721. Este estándar permite que las NFTs sean únicas, y tengan diferente valor que las otras tokens del mismo smart contract, ya sea dado por su antigüedad, rareza, o tal vez algo más particular como su visual.

Todas las NFTs tienen una variable del tipo “uint256”(unsigned, int, 256 bits) llamada tokenId. Esto hace que para cualquier contrato del tipo ERC-721, la dirección de contacto tiene que tener un tokenId único (globalmente). Sabiendo esto, una dApp puede tener un conversor que utiliza el tokenId como input y muestra una imagen/video/sonido en pantalla como output.

Las NFTs, si bien tienen potencial para mucho más, hoy en día son utilizadas principalmente para crear objetos coleccionables, o definir algo como único, como bien se mencionó anteriormente. Los usos más comunes incluyen objetos coleccionables(más utilizado en el campo artístico) así como varios juegos virtuales que utilizan NFTs ya sea como moneda de intercambio o como un objeto utilizable dentro del mismo juego. Hay varias aplicaciones mas para las NFTs pero estas son las más utilizadas actualmente

2.2.2.4 Implementacion y despliegue de un NFT

Para implementar un NFT, un smart contract debe contar con los siguientes métodos y eventos:

Metodos

```
1        function balanceOf(address _owner) external view returns (uint256);
2        function ownerOf(uint256 _tokenId) external view returns (address);
3        function safeTransferFrom(address _from, address _to, uint256
    _tokenId, bytes data) external payable;
4        function safeTransferFrom(address _from, address _to, uint256
    _tokenId) external payable;
5        function transferFrom(address _from, address _to, uint256 _tokenId)
    external payable;
6        function approve(address _approved, uint256 _tokenId) external
    payable;
7        function setApprovalForAll(address _operator, bool _approved)
    external;
8        function getApproved(uint256 _tokenId) external view returns
    (address);
9        function isApprovedForAll(address _owner, address _operator)
    external view returns (bool);
```

Eventos

```
1        event Transfer(address indexed _from, address indexed _to, uint256
    indexed _tokenId);
2        event Approval(address indexed _owner, address indexed _approved,
    uint256 indexed _tokenId);
3        event ApprovalForAll(address indexed _owner, address indexed
    _operator, bool _approved);
```

2.2.2.5 Ejemplos de NFT's

Como se mencionó, las NFTs pueden tomar varias formas. A continuación se listarán las más populares en el mercado, así como un par de usos creativos para NFTs.

- CryptoKitties es un juego basado en la procreación, intercambio y recolección de NFTs llamadas Cryptokitties.

- Sorare es un fútbol de fantasía en el cual se pueden conseguir coleccionables y premios por competencias, así como manejar tus equipos entre otros.

- Unstoppable Domains, es una empresa en san francisco que permite obtener dominios en blockchain. Estos dominios reemplazan el dominio de la criptomoneda por texto comprensible por un humano(así como un dominio de internet pero para criptomonedas).

- Gods Unchained Cards, es un juego de cartas intercambiables dentro de una blockchain que utiliza NFTs para traer esa sensación de "ownership" de tus recursos. Este juego consiguió mucha popularidad en los últimos meses y planea seguir creciendo.

2.2.2.6 Adquisicion de NFT's

Para obtener un NFT lo único que hay que hacer es justamente, comprarlo. Hay varias formas ya sea compra directa o por subasta(la mayoría de los NFTs caen en esta categoría). Si bien solo hay que comprarlos, dentro de la red ethereum, se obtienen mayormente a cambio de Ether. Por eso primero se deberá tener una billetera virtual con ethers y luego comprarlos.

2.2.4 Implementación para la creación y despliegue de un token

2.2.4.1 MetaMask

El primer paso para la creación de un token es crear una metamask account. Este paso es sencillo y solo requiere la creación de la cuenta a través de la plataforma de metamask.io.
Metamask es una billetera virtual de las más conocidas en el mercado. Aparte de funcionar como una billetera, también sirve como intermediario o gateway para utilizar las extensas dApps de ethereum, una de ellas es la que nos permitirá la creación de nuestra token. Aparte de esto, se utilizará Metamask para mantener las tokens que se usarán para pagar el deployment fee.

2.2.4.2 Obtener un nodo de blockchain

El siguiente paso para la creación de una token será conseguir un nodo funcional de la blockchain asociada a la red donde se lanzará este token. El nodo es esencial ya que es el método de comunicación e interacción que está token tendrá con la blockchain. Si bien hay varios proveedores, solo se mencionara moralis ya que es actualmente el más sencillo y rápido de ellos. Desde moralis se puede obtener el nodo de la red ethereum(así como BSC, POLYGON y ARBITRUM que son otras grandes redes).

Los nodos están representados por una URL la cual se puede obtener fácilmente utilizando este proveedor.

2.2.4.3 Obtener Tokens

El último paso previo a lanzar nuestra token es obtener un capital de tokens nativos a la blockchain que planeamos usar, en este caso Ether. Obtener ether es sencillo y hay diferentes formas de hacerlo. Una de ellas es utilizar una plataforma de cambios fiat como Coinbase o robinhood. Otro ejemplo es la utilización de exchanges descentralizados(se hablará más en detalle más adelante) tales como Uniswap o pancakeswap para intercambiar una token por otra.

2.2.4.4 Desarrollo y Deployment

Para el desarrollo de la token, a parte de los métodos y eventos necesarios para que cumpla con los protocolos correspondientes(ERC-20/ERC-777 etc…) hay algunos pasos extra a considerar. Si bien no se entrará en detalle sobre el proceso de creación de la token en sí, se mencionara en aspectos generales que pasos se deben tomar para un mejor entendimiento de estas.

Primero se debe configurar el nodo. Esto se puede realizar utilizando brownie network manager, el cual es un servicio integrado en la red ethereum. Una vez seteado el nodo, utilizando la URL obtenida anteriormente, se debe configurar las licencias, la denominada Pragma Line y se debe importar el paquete “OpenZeppelin”. Una vez finalizados estos pasos se procede a la creación del token en sí.

A la hora de crear el token, se deben seguir los protocolos apropiados para dicha token, como se mencionó anteriormente. OpenZeppelin tiene varios ejemplos de tokens para que el developer se guíe en cómo crearlos, así como qué métodos son necesarios para cada protocolo.

Una vez que se desarrolla el Smart Contract de la token lo único que queda es testearlo y realizar el deployment. Este paso es sencillo ya que se realiza la compilación desde la consola brownie y si se ejecuta correctamente, el testing está finalizado.

El despliegue de la moneda se realiza de forma similar, introduciendo una serie de comandos en la consola brownie los cuales resultaran en el despliegue de la token así como la asignación de direcciones para la token implementada.

Para más información sobre cómo crear e implementar una token dentro de la red ethereum, se puede obtener desde la página ***moralis.io***, donde hay un tutorial detallado sobre cómo hacerlo.

2.3 Smart Contracts

2.3.1 Introducción a los SC’s

"Smart Contracts" o Contratos inteligentes es un término que se utiliza para describir el código informático que ejecuta automáticamente todo o parte de un acuerdo y se almacena en la blockchain. Como se analizará más adelante, el código puede ser la única manifestación del acuerdo entre las partes o puede complementar un contrato tradicional basado en texto y ejecutar ciertas disposiciones, como la transferencia de fondos de la Parte A a la Parte B. El código en sí se replica en múltiples nodos de una cadena de bloques y, por lo tanto, se beneficia de la seguridad, permanencia e inmutabilidad que ofrece una cadena de bloques. Si las partes han indicado, al iniciar una transacción, que se han cumplido ciertos parámetros, el código ejecutará el paso desencadenado por esos parámetros. Si no se ha iniciado dicha transacción, el código no tomará ningún paso. La mayoría de los contratos inteligentes están escritos en uno de los lenguajes de programación directamente adecuados para dichos programas informáticos, como Solidity, pero existen otros tales como Vyper.

En la actualidad, los parámetros de entrada y los pasos de ejecución de un contrato inteligente deben ser específicos y objetivos. En otras palabras, si se produce "x", ejecute el paso "y". Por lo tanto, las tareas reales que realizan los contratos inteligentes son bastante rudimentarias, como mover automáticamente una cantidad de criptomonedas de la billetera de una parte a otra cuando se cumplen ciertos criterios. A medida que se extiende la adopción de blockchain, y a medida que se tokenizan más activos o se “encadenan”, los contratos inteligentes se volverán cada vez más complejos y capaces de manejar transacciones sofisticadas. De hecho, los desarrolladores ya están encadenando múltiples pasos de transacción para formar contratos inteligentes más complejos.

Antes de que un contrato inteligente compilado realmente pueda ejecutarse en ciertas cadenas de bloques, se requiere un paso adicional, a saber, el pago de una tarifa de transacción para que el contrato se agregue a la cadena y se ejecute. En el caso de la cadena de bloques Ethereum, los contratos inteligentes se ejecutan en la Máquina Virtual Ethereum (EVM), y este pago, realizado a través de la criptomoneda ether, se conoce como "gas". Cuanto más complejo sea el contrato inteligente (según los pasos de la transacción a realizar), más gas se debe pagar para ejecutar el contrato inteligente. Por lo tanto, el gas actúa actualmente como una puerta importante para evitar que los contratos inteligentes excesivamente complejos o numerosos abrumen al EVM.

Actualmente, los contratos inteligentes son los más adecuados para ejecutar automáticamente dos tipos de "transacciones" que se encuentran en muchos contratos: garantizar el pago de fondos en ciertos eventos desencadenantes e imponer sanciones financieras si no se cumplen ciertas condiciones objetivas. En cada caso, la intervención humana, incluso a través de un titular de depósito de confianza o incluso del sistema judicial, no es necesaria una vez que el contrato inteligente se ha implementado y está operativo, lo que reduce los costos de ejecución y cumplimiento del proceso de contratación.

2.3.2 Lenguajes disponibles para la creación de SCs

Dentro de la red Ethereum, los dos lenguajes mas activos para la creacion de smart contracts son solidity y viper, los cuales tienen una sintaxis simple y directa.

2.3.2.1 Solidity

Solidity es un lenguaje de alto nivel, orientado a objetos utilizado para la creación de smart contracts. Este lenguaje fue influenciado profundamente por C++, tipado de forma estática(el tipo de variable se conoce al momento de compilar) y tiene soporte para herencia, librerías y tipos de datos complejos(definidos por el usuario).

Este lenguaje fue creado para la red ethereum, más específicamente, para la creación de smart contracts en dicha red.

Para poder compilar archivos en este lenguaje se pueden utilizar diferentes técnicas.

Remix, es una página web la cual puede compilar solidity de forma rápida y eficiente, muy recomendado para alguien que esté aprendiendo el lenguaje así como para realizar tests sin la necesidad de instalar varias versiones diferentes de solidity en la computadora.

npm/Node Js. es una forma conveniente y portable de instalar solcjs, el cual es un compilador de solidity accesible por línea de comando.

En la organización ethereum se puede conseguir también una imagen que utiliza solc para Docker. Esta imagen corre el ejecutable del compilador y se le pueden pasar todos los argumentos de compilación al mismo.

2.3.2.2 Vyper

Vyper por otro lado, es un lenguaje basado en python, de tipado fuerte y código compacto, el cual busca que los Smart Contracts sean más seguros y fáciles de auditar, reduciendo los “features” a diferencia de su competencia, solidity.

Vyper no soporta modificadores, herencia, overloading de funciones u operadores, llamadas recursivas o loops infinitos entre otras características que hacen solidity más versátil. Aun así, vyper está muy presente en la red ethereum como un lenguaje simplificado de smart contracts.

2.3.3 Anatomía y sintaxis de un Smart Contract

2.3.3.1 Informacion

La información de un smart contract puede ser almacenada en memoria o en disco. Almacenar información en disco en la red ethereum tiende a ser costoso así que solo se deberá usar en caso de ser necesario.

Para almacenar información en la red ethereum se representa con variables de estado. A estas se les debe definir el tipo y nombre para que el SC tenga nocion de cuanto espacio se necesitará para almacenar dicha variable.

Los tipos de variables que se pueden almacenar en la red incluyen: Booleanos, int, Fixed-Point numbers, Arrays de tamaño fijo, arrays de tamaño dinámico, racionales y enteros literales, Strings literales, hexadecimal, enums y direcciones.

Cada uno de estos tipos de variables ocupará un espacio diferente en la red, por ejemplo, una dirección ocupa 20 bytes o 160 bits, y se expresa en hexadecimal comenzando con 0x.

La siguiente imagen expresa cómo se realiza en el contrato el almacenamiento en la red.

```solidity
// Solidity example
contract SimpleStorage {
    uint storedData; // State variable
    // ...
}
```

```vyper
# Vyper example
storedData: int128
```

Las variables que sólo tienen relevancia durante la ejecución del contrato se pueden almacenar en memoria y son mucho menos costosas, ya que no requieren un espacio permanente en la red ethereum.

Adicionalmente, existen las variables de entorno las cuales son utilizadas principalmente para proveer información acerca de la blockchain o la transacción actual. Un ejemplo claro sería la variable msg.sender, del tipo address, la cual hace referencia a la dirección del emisor del mensaje que ejecutó la llamada actual.

2.3.3.2 Funciones

Las funciones funcionan como en cualquier otro lenguaje de programación y su utilización termina siendo la misma. En el ámbito de los smart contracts se definen varias categorías para ellas.

- Internas: No crean una llamada a la EVM, y solo se pueden acceder desde el contrato donde existen o sus contratos asociados.
- Externas: generan una llamada a la EVM, lo que permite que sean invocadas desde otros contratos por medio de transacciones y se invoca como una función externa en los lenguajes o.o.(f(x) no funciona pero this.f(x) si)

Fuera de internas o externas, existen diferentes categorizaciones para funciones especiales en los SC.

Las funciones en los SC deben contener la siguiente estructura e información:

- Variables de parámetro y tipo(si acepta parámetros de entrada)
- Declaración de internal/external
- Declaracion de pure/vire/payable
- Tipo de retorno(si aplica)

En la siguiente imagen se puede apreciar un SC con funciones básicas para ejemplificar

```
pragma solidity >=0.4.0 <=0.6.0;

contract ExampleDapp {
    string dapp_name; // state variable

    // Called when the contract is deployed and initializes the value
    constructor() public {
        dapp_name = "My Example dapp";
    }

    // Get Function
    function read_name() public view returns(string) {
        return dapp_name;
    }

    // Set Function
    function update_name(string value) public {
        dapp_name = value;
    }
}

```

2.3.3.2.1 Funciones de vista

Este tipo de funciones, también llamadas funciones “Getter” son utilizadas para obtener información y no realizan cambios dentro de la red o las variables involucradas en el SC.

```
// Solidity example
function balanceOf(address _owner) public view returns (uint256
_balance) {
    return ownerPizzaCount[_owner];
}
```

```
dappName: public(string)

@view
@public
def readName() -> string:
  return dappName
```

2.3.3.2.2 Funciones constructoras

Estas funciones se ejecutan sólo cuando el SC es llamado por primera vez y funcionan como un constructor en cualquier lenguaje basado en clases. Estas funciones normalmente inicializan variables de estado a los valores especificados.

```
# Vyper example

@external
def __init__(_beneficiary: address, _bidding_time: uint256):
    self.beneficiary = _beneficiary
    self.auctionStart = block.timestamp
    self.auctionEnd = self.auctionStart + _bidding_time
```

```
// Solidity example
// Initializes the contract's data, setting the `owner`
// to the address of the contract creator.
constructor() public {
    // All smart contracts rely on external transactions to trigger its functions.
    // `msg` is a global variable that includes relevant data on the given transaction,
    // such as the address of the sender and the ETH value included in the transaction.
    // Learn more: https://solidity.readthedocs.io/en/v0.5.10/units-and-global-variables.html#block-and-transaction-properties
    owner = msg.sender;
}

```

2.3.3.2.3 Funciones internas(Built-in)

Aparte de las funciones que un programador puede definir en un smart contract, también existen funciones internas o preprogramadas. La mas comun seria:

- address.send() - Solidity
- send(address) - Vyper

Estas funciones permiten enviar ETH a otras cuentas.

2.3.3.3 Eventos y logs

Los eventos y logs permiten la comunicación entre el smart contract y el frontend u otras aplicaciones suscritas. Cuando una transacción es minada, un smart contract puede emitir eventos y escribir logs a la blockchain para que el frontend lo procese.

2.3.4 Usos de SCs

Los contratos inteligentes se pueden utilizar en una variedad de campos, desde la atención médica hasta la cadena de suministro y los servicios financieros. Algunos ejemplos son los siguientes definidos.

2.3.4.1 Sistema de votación del gobierno

Los contratos inteligentes proporcionan un entorno seguro que hace que el sistema de votación sea menos susceptible a la manipulación. Los votos que utilizan contratos inteligentes estarían protegidos por un libro mayor, lo cual es extremadamente difícil de decodificar.

Además, los contratos inteligentes podrían aumentar la rotación de votantes, que históricamente es baja debido al sistema ineficiente que requiere que los votantes se alineen, muestren identidad y completen formularios. La votación, cuando se transfiere en línea mediante contratos inteligentes, puede aumentar el número de participantes en un sistema de votación.

2.3.4.2 Cuidado de la salud

Blockchain puede almacenar los registros de salud codificados de los pacientes con una clave privada. Solo se concedería acceso a los registros a determinadas personas por motivos de privacidad. Del mismo modo, la investigación se puede realizar de forma confidencial y segura mediante contratos inteligentes.

Todos los recibos hospitalarios de los pacientes pueden almacenarse en la cadena de bloques y compartir automáticamente con las compañías de seguros como prueba de servicio. Además, el libro mayor se puede utilizar para diferentes actividades, como la gestión de suministros, la supervisión de medicamentos y el cumplimiento de la normativa.

2.3.4.3 Cadena de suministro

Tradicionalmente, las cadenas de suministro sufren debido a los sistemas basados en papel donde los formularios pasan por múltiples canales para obtener aprobaciones. El laborioso proceso aumenta el riesgo de fraude y pérdida.

Blockchain puede anular tales riesgos al entregar una versión digital accesible y segura a las partes involucradas en la cadena. Los contratos inteligentes se pueden utilizar para la gestión de inventario y la automatización de pagos y tareas.

2.3.4.4 Servicios financieros

Los contratos inteligentes ayudan a transformar los servicios financieros tradicionales de múltiples formas. En el caso de las reclamaciones de seguros, realizan la verificación de errores, el enrutamiento y la transferencia de pagos al usuario si todo se considera apropiado.

Los contratos inteligentes incorporan herramientas críticas para la contabilidad y eliminan la posibilidad de infiltración de registros contables. También permiten a los accionistas participar en la toma de decisiones de forma transparente. Además, ayudan en la compensación comercial, donde los fondos se transfieren una vez que se calculan los montos de las liquidaciones comerciales.

2.3.5 Únicos a la red Ethereum ?

Los Smart Contracts si bien tuvieron su origen en la red Ethereum esta tecnología se extendió a otros proyectos similares tales como:

- Cardano (ADA)
- Polkadot (DOT)
- VeChain (VET)
- Stellar (XLM)
- Tron (TRX)
- Neo (NEO)
- Klatyn (KLAY)
- EOS (EOS)
- Cosmos (ATOM)

2.3.6 Ejemplos de SCs

Los contratos inteligentes podrían eliminar las llamadas brechas entre la compra y el pago. Cuando llega un producto y se escanea en un almacén, un contrato inteligente podría desencadenar inmediatamente solicitudes para las aprobaciones requeridas y, una vez obtenido, transferir fondos inmediatamente del comprador al vendedor. A los vendedores se les pagaría más rápido y ya no necesitan participar en reclamaciones, y los compradores reducirían los costos de sus cuentas por pagar. Esto podría afectar los requisitos de capital de trabajo y simplificar las operaciones financieras para ambas partes. En el lado de la aplicación, se podría programar un contrato inteligente para

cortar el acceso a un activo conectado a Internet si no se recibe un pago. Por ejemplo, se puede denegar automáticamente el acceso a cierto contenido si no se recibe el pago.

2.3.7 Implementación y despliegue de un SCs en la red Ethereum

Para desplegar un SC lo único que se debe realizar es enviar una transacción conteniendo el código compilado del SC sin especificar recipientes.

Para realizar el despliegue se necesita el código compilado, ETH para pagar el gas, el script o plugin de despliegue y acceso a un nodo de ethereum, ya sea uno personal o unirse a uno público o también usando el servicio node a través de una API.

Una vez que se tiene todo lo mencionado anteriormente, solo hace falta utilizar una herramienta especializada para realizar la transacción y el SC ya estará montado en la red ethereum. Las herramientas más utilizadas son Hardhat y Truffle, y la documentación necesaria para realizar la transacción, incluyendo los scripts y los pasos necesarios para hacerla, se puede encontrar en la página principal de ambas.

2.4 Decentralized Applications (DAPPS)

Una aplicación descentralizada, o dapp, es una aplicación que corre dentro de una red descentralizada, en este caso ethereum, cuya información es almacenada en la blockchain y la lógica del backend en Smart Contracts.

Una Dapp puede tener una interfaz gráfica como cualquier otra aplicación y realizar llamadas al backend, la diferencia es que estas llamadas se realizan a la red en vez de a un código centralizado.

Una vez que se monta una dapp en la red, no se puede sacar y queda a libertad de uso de los usuarios de la red. Estas aplicaciones corren en la EVM para, en caso de que haya un bug en la misma, este no afecte a la red entera.

2.4.1 Ventajas y desventajas de una DAPP

2.4.1.1 Ventajas

- Zero Downtime - Una vez que el SC es implementado y desplegado a la blockchain, la red entera va a ser capaz de proveer este servicio a los clientes que quieran hacer uso de ella,

haciendo imposible que algún actor malicioso pueda lanzar ataques hacia una dapp específica.

- Privacidad - no es necesario proveer identificación real para interactuar con una dapp.

- Resistencia al censurado - ninguna entidad individual puede impedir que usuarios desplieguen dapps, realicen transacciones o lean información de la blockchain.

- Integridad completa de datos - La información almacenada en la blockchain es inmutable e indiscutible, gracias a las primitivas criptográficas. Eso impide la falsificación de transacciones y otros datos que ya se han hecho públicos.

- Trustless Computation / comportamiento verificable - los SC pueden ser analizados y su ejecución está garantizada de forma predecible, sin la necesidad de confiar en una autoridad central. Esto es diferente en las apps convencionales, por ejemplo, una aplicación bancaria, donde un usuario debe confiar en que la institución financiera no haga mal uso de la información, altere nuestros registros o que sea hackeada.

2.4.1.1 Desventajas

- Mantenimiento - es difícil dar mantenimiento a una dapp ya que alterar el código y la información publicada en una blockchain es difícil.

- Performance overhead - Hay una gran cantidad de gastos generales y escalar una plataforma es difícil de realizar. Para conseguir el nivel de seguridad, integridad y transparencia que ethereum aspira a alcanzar, cada nodo ejecuta y almacena cada transacción. Aparte de esto, el proof of work toma tiempo también.

- Congestión de red - Cuando una dapp usa demasiados recursos computacionales, la red entera necesita realizar un backup. Actualmente la red puede procesar sólo 10-15 transacciones por segundo, lo cual puede generar tiempos de espera largos para la ejecución de una.

- Experiencia de usuario - tiende a ser más difícil crear interfaces y experiencias User-friendly ya que los usuarios pueden encontrar difícil la configuración de las herramientas necesarias para interactuar con la blockchain de una forma segura.

- Centralización - Las soluciones User-friendly y developer friendly que actualmente existen tienden a comportarse como una aplicación centralizada, lo cual pierde el propósito de funcionar sobre una red descentralizada y el uso de block chains.

2.4.1 Ejemplos de DAPPS

Las dapps pueden tomar diferentes formas y tener múltiples usos, hoy en día existen muchas dapps siendo utilizadas a nivel mundial. Un ejemplo claro de esto podría ser el juego EOS Dynasty, el cual es el primer juego RPG Online PVP(Rol Playing Game, Player vs Player) basado en blockchain. Otro ejemplo claro es el navegador web Brave, el cual también está basado en token y recompensa a los usuarios con una criptomoneda propietaria

2.5 Monedas universitarias

2.5.1 Introduccion

Como se mencionó en la introducción, las universidades hoy en día manejan un sistema de moneda interno el cual pueden intercambiar por productos o servicios de manera sencilla y rápida. Para tomar por ejemplo, se va a utilizar la universidad central de Florida(UCF) con su sistema monetario interno llamado Knight Cash,nombrado así debido a que el equipo deportivo de la universidad son los caballeros, o Knights.

2.5.2 Como funciona

Knight Cash es una moneda interna de la universidad UCF, la cual se puede adquirir si uno es alumno, profesor o empleado de la institución. Segun su pagina web, https://ucfcard.ucf.edu/knight-cash/, se puede optar por pagar con esta moneda en casi todos los locales dentro del campus y es práctico debido a que, gracias a ella, el alumno puede no tener efectivo encima para poder acceder a productos y necesidades básicas.

Para cargar dinero, tienen cuatro opciones diferentes, dos en persona la cual puede ser en una especie de ATM distribuidos a través del campus o yendo al edificio de servicios de tarjeta. Y dos de forma virtual, ya sea desde la APP o desde la página web.

2.5.3 Ventajas y desventajas

Los beneficios que provee esta forma de pago principalmente incluyen, el hecho de que se puede cargar de forma virtual, está implementado en todo el campus y, recientemente, lograron crear un partnership con Fairwinds Credit Union para poder agregarlo al sistema bancario del mismo. De esta forma un alumno puede acceder a su dinero de forma sencilla y sin necesidad de cargar efectivo encima.
El problema principal de este es, como se mencionó anteriormente, que esta moneda no funciona fuera de la institución mencionada. Así como se utilizó de ejemplo Knight Cash, múltiples universidades a través del mundo sufren del mismo problema, entorpeciendo las finanzas de sus alumnos, en mayor o menor medida, y creando un problema principal: No poder acceder al dinero en otra universidad.

2.6 Exchanges y conceptos centrales

2.6.1 Qué es un exchange?

Una exchange es un mercado donde se negocian valores, materias primas, derivados y otros instrumentos financieros. La función principal de una exchange es garantizar un comercio justo y ordenado y la difusión eficiente de la información de precios para cualquier negociación de valores en esa bolsa. Los intercambios brindan a las empresas, gobiernos y otros grupos una plataforma desde la cual vender valores al público inversionista.

Un exchange puede ser una ubicación física donde los comerciantes se reúnen para realizar negocios o una plataforma electrónica. También pueden denominarse exchange de acciones o "bolsa", según la ubicación geográfica. Los exchange se encuentran en la mayoría de los países del mundo. Las bolsas más destacadas incluyen la Bolsa de Valores de Nueva York (NYSE), el Nasdaq, la Bolsa de Valores de Londres (LSE) y la Bolsa de Valores de Tokio (TSE).

2.6.1.1 Exchanges electronicos

En la última década, el comercio ha pasado a intercambios totalmente electrónicos. La igualdad de precios algorítmica sofisticada puede garantizar un comercio justo sin requerir que todos los miembros estén físicamente presentes en un piso de negociación centralizado.

2.6.1.2 Requisitos de listado

Cada bolsa tiene requisitos de cotización específicos para cualquier empresa o grupo que desee ofrecer valores para negociar. Algunas bolsas son más rígidas que otras, pero los requisitos básicos para las bolsas de valores incluyen informes financieros regulares, informes de ganancias auditados y requisitos de capital mínimo. Por ejemplo, la NYSE tiene un requisito clave de cotización que estipula que una empresa debe tener un mínimo de 4 millones en capital social (SE).

2.6.1.3 Los exchanges brindan acceso al capital

Una bolsa de valores se utiliza para recaudar capital para empresas que buscan crecer y expandir sus operaciones. La primera venta de acciones por parte de una empresa privada al público se denomina oferta pública inicial (OPI u ICO). Las empresas que cotizan en la bolsa de valores suelen tener un perfil mejorado. Tener más visibilidad puede atraer nuevos clientes, empleados talentosos y proveedores ansiosos por hacer negocios con un líder prominente de la industria.

Las empresas privadas a menudo dependen de los capitalistas de riesgo para la inversión, y esto generalmente resulta en la pérdida del control operativo. Por ejemplo, una empresa de financiación inicial puede requerir que un representante de la empresa de financiación ocupe un puesto destacado en la junta. Alternativamente, las empresas que cotizan en una bolsa de valores tienen más control y autonomía porque los inversores que compran acciones tienen derechos limitados.

2.6.2 SWAPS

"Swap" generalmente significa "Intercambiar un artículo o servicio de valor por otro (o muchos) artículo o servicio de valor (de valor equivalente)". De manera similar, en criptomonedas, un "Swap" se refiere al intercambio de una criptomoneda que tienes por el valor equivalente de otra criptomoneda.

Para completar un intercambio, lo más probable es que utilice un servicio (normalmente centralizado). Esto es similar a un intercambio, con la principal diferencia de que se trata de una moneda FIAT. En cambio, el comercio es de criptomonedas. Durante un Swap, paga con una criptomoneda alternativa diferente (distinta) a la criptografía que comprará.

Cuando compra una criptomoneda a través de una operación (ya sea en un intercambio o mediante un corredor), puede pagar su compra de criptomonedas con una moneda emitida por el gobierno "local" para una criptomoneda en particular (ejemplo: USD, EUR, JPY, etc.).

Hay muchos servicios de intercambio en el mercado que permiten a los usuarios comprar y vender criptomonedas por monedas tradicionales o por otras criptomonedas. Sin embargo, debido a la liquidez limitada y la cantidad de pares comerciales en cada intercambio, los usuarios que desean comerciar directamente entre dos tokens criptográficos a veces no pueden hacerlo.

Esto es especialmente cierto para los tokens menos populares, porque a menudo solo están disponibles en una pequeña cantidad de intercambios. En lugar de un comercio directo, los usuarios se ven obligados a incluir el paso intermedio de convertir dentro y fuera de dinero fiduciario o una de las criptomonedas más populares, como BTC o ETH.

Sin embargo, algunos servicios de intercambio se enfocan en este problema específicamente agregando otros múltiples intercambios y obteniendo liquidez de ellos. El resultado final es que los usuarios pueden intercambiar entre dos criptomonedas directamente sin los inconvenientes y las tarifas dobles asociadas con la realización de una operación en dos pasos. Algunos de los servicios que permiten el intercambio de tokens son Metamask, ShapeShift y AirSwap.

Además de cumplir su propósito directo, como dinero digital descentralizado, algunas criptomonedas como ETH, NEO y QTUM permiten a los usuarios lanzar otros tokens criptográficos sobre sus cadenas de bloques.
Estos tokens de segunda capa pueden "aprovecharse" de las plataformas subyacentes y disfrutar de algo de su seguridad y popularidad sin tener que gastar tiempo y recursos en hacer crecer su propio ecosistema desde cero.

Sin embargo, en algunos casos, la plataforma sobre la que se basa un token puede resultar inadecuada para sus necesidades actuales. Por ejemplo, los desarrolladores pueden construir su token en la cadena de bloques de Ethereum para aprovechar su gran base de usuarios durante la oferta inicial de monedas, pero luego deciden que necesitan diferentes parámetros subyacentes para el lanzamiento real del producto.

En tales casos, es posible un intercambio de token, en el que los desarrolladores migran su token de una base de blockchain a otra, mientras mantienen todos los saldos de direcciones.

2.6.3 Liquidity Pools

Una "liquidity pool" o reserva de liquidez es un conjunto de fondos bloqueados en depósito en un "smart contract". Las reservas de liquidez se emplean para facilitar el trading y lending descentralizado.

Las "liquidity pools" son la columna vertebral de muchos exchanges descentralizados (DEX), como por ejemplo Uniswap. Un tipo de usuarios denominados proveedores de liquidez (LPs) aportan un valor equivalente de dos tokens en una reserva (pool) para así crear un mercado. A cambio de aportar sus fondos, ganarán comisiones de trading a partir de los trades que tengan lugar en su "pool" o reserva, de manera proporcional a su participación en la liquidez total.
Dado que cualquiera puede ser un proveedor de liquidez, los AMMs han hecho que el "market making" resulte más accesible.

Uno de los primeros protocolos que utilizará "liquidity pools" sería Bancor, aunque el concepto atraería más atención con la popularización de Uniswap. Otros exchanges populares en Ethereum que utilizan "liquidity pools" son SushiSwap, Curve y Balancer. Las reservas de liquidez de estas plataformas contienen tokens ERC-20. Otros equivalentes similares en Binance Smart Chain (BSC) son PancakeSwap, BakerySwap y BurgerSwap -cuyas "pools" contienen tokens BEP-20.

2.6.3.1 Liquidity pools vs. libros de órdenes

Para entender en qué se diferencian las "liquidity pools" o reservas de liquidez, vamos a analizar el pilar fundamental del trading electrónico –el libro de órdenes. Dicho de manera simple, el libro de órdenes es una colección de las órdenes actualmente abiertas de un mercado concreto.

El sistema que empareja entre sí las órdenes se denomina matching engine (motor de emparejamiento). Junto con el motor de emparejamiento, el libro de órdenes es el núcleo de todo exchange centralizado (CEX). Este modelo es genial para facilitar intercambios eficientes, y permitió la creación de mercados financieros complejos.

El trading en DeFi, sin embargo, conlleva ejecutar trades on-chain, sin una tercera parte centralizada que esté en posesión de los fondos. Esto supone un problema en lo relativo a libros de órdenes. Cada interacción con el libro de órdenes requiere comisiones de gas, lo que hace que resulte mucho más caro ejecutar los trades.

También hace que la labor de los market makers -traders que aportan liquidez a pares tradeables- sea muy costosa. Pero por encima de todo, la mayoría de blockchains no pueden ofrecer el rendimiento necesario para tradear billones de dólares cada día.
Esto significa que en una blockchain como Ethereum, un exchange con libro de órdenes on-chain es prácticamente imposible.

A pesar de ello, dado que muchos de los activos del sector de las cripto se encuentran en Ethereum, no será posible intercambiarlos en otras redes, salvo que utilices algún tipo de cross-chain bridge (puente entre cadenas).

2.6.3.2 ¿Cómo funcionan las liquidity pools?

Los "automated market makers" (AMM) han cambiado las reglas del juego. Son una innovación significativa que posibilita el trading "on-chain" sin necesidad de un libro de órdenes. Dado que no se necesita una contraparte directa para ejecutar los trades, los traders pueden entrar y salir de posiciones en pares de tokens que, probablemente, serían muy ilíquidos en exchanges basados en libros de órdenes.

Es posible imaginar un "order book exchange" como una plataforma peer-to-peer, en la que compradores y vendedores están conectados por el libro de órdenes.

Tradear utilizando un AMM es equivalente a un peer-to-contract (es decir, entre par y contrato).

Como hemos comentado, una reserva de liquidez (liquidity pool) es un conjunto de fondos depositados en un "smart contract" por proveedores de liquidez. Cuando ejecutas un "trade" en un AMM, no tendrás una contraparte en el sentido tradicional. En su lugar, ejecutaras el "trade" contra la liquidez de la "liquidity pool" (reserva de liquidez). Para que el comprador pueda comprar, no es necesario que en el momento haya un vendedor, sino tan sólo suficiente liquidez en la reserva.

Cuando compras la última moneda de comida en Uniswap, no hay un vendedor del otro lado en el sentido tradicional. En cambio, tu actividad es administrada por el algoritmo que gobierna lo que sucede en la reserva. Además, el precio también está determinado por este algoritmo en función de las operaciones que ocurren en la reserva.

2.6.3.3 ¿Para qué se utilizan las reservas de liquidez?

Uno de ellos es la agricultura de rendimiento o la minería de liquidez. Las reservas de liquidez son la base de plataformas de generación de rendimiento automatizadas como yearn, donde los usuarios agregan sus fondos a las reservas que luego se utilizan para generar rendimiento.

Distribuir nuevos tokens en manos de las personas adecuadas es un problema muy difícil para los proyectos cripto. La minería de liquidez ha sido uno de los enfoques más exitosos. Básicamente, los tokens se distribuyen algorítmicamente a los usuarios que ponen sus tokens en una reserva de liquidez. Luego, los tokens recién acuñados se distribuyen proporcionalmente a la participación de cada usuario en la reserva.

“ Tener en cuenta; Estos pueden incluso ser tokens de otras reservas de liquidez llamados pools tokens. Por ejemplo, si estás proporcionando liquidez a Uniswap o prestando fondos a Compound, obtendrás tokens que representan tu participación en la reserva. Es posible que puedas depositar esos tokens en otra reserva y obtener una devolución. Estas cadenas pueden volverse bastante complicadas, ya que los protocolos integran tokens de reservas de otros protocolos en sus productos, y así sucesivamente. ”

También podríamos pensar en la gobernanza como un caso de uso. En algunos casos, se necesita un umbral muy alto de votos simbólicos para poder presentar una propuesta de gobernanza formal. Si, en cambio, los fondos se juntan, los participantes pueden unirse a una causa común que consideren importante para el protocolo.

Otro uso aún más innovador de las reservas comunes de liquidez es el tramo. Es un concepto tomado de las finanzas tradicionales que implica dividir los productos financieros en función de sus riesgos y beneficios. Como era de esperar, estos productos permiten a las LP seleccionar perfiles personalizados de riesgo y retorno.

La acuñación de activos sintéticos en la blockchain también depende de las reservas de liquidez. Agrega algo de garantía a una reserva de liquidez, se conecta a un oráculo confiable y se obtiene un token sintético que está vinculado a cualquier activo que desees. Muy bien, en realidad, es un problema más complicado que eso, pero la idea básica es así de simple.

2.6.3.4 Los riesgos de las reservas de liquidez

Si proporcionas liquidez a un AMM, debes conocer un concepto llamado pérdida impermanente. En resumen, es una pérdida de valor en dólares en comparación con HODLing cuando proporcionan liquidez a un AMM.

Si se está proporcionando liquidez a un AMM, probablemente esté expuesto a una pérdida impermanente. A veces puede ser diminuta; a veces puede ser enorme. Asegúrate de leer nuestro artículo al respecto si estás considerando colocar fondos en una reserva de liquidez de dos lados.

Otra cosa a tener en cuenta son los riesgos de los contratos inteligentes. Cuando depositan fondos en una reserva de liquidez, están en la reserva. Entonces, aunque técnicamente no hay intermediarios que retengan tus fondos, el contrato en sí puede considerarse como el custodio de esos fondos. Si hay un error o algún tipo de explotación a través de un préstamo flash, por ejemplo, tus fondos podrían perderse para siempre.

2.7 Estrategia de Arbitraje

2.7.1 Introduccion

“El arbitraje es una estrategia financiera que consiste en aprovechar la diferencia de precio entre diferentes mercados sobre un mismo activo financiero para obtener un beneficio económico, normalmente sin riesgo.”

Para realizar arbitraje se realizan operaciones complementarias (comprar y vender) al mismo tiempo y esperar a que los precios se ajusten. El arbitraje aprovecha esa divergencia y obtiene una ganancia libre de riesgo. Dicho de otra manera, el arbitrajista se posiciona en corto (vende) en el mercado con mayor precio y largo (compra) en el mercado con menor precio. El beneficio vendría dado por la diferencia entre ambos mercados.

El arbitraje es posible debido a ineficiencias en los mercados. Cuando no hay posibilidad de hacerlo se dice que se cumple la condición de no arbitraje. Cuanto más eficiente sea un mercado, más difícil será realizarlo.

El arbitraje se considera libre de riesgo. Sin embargo, si la diferencia viene dada por una ineficiencia como por ejemplo la liquidez, es posible que esa liquidez no nos permita aprovechar la diferencia de precio o que para aprovecharla estemos asumiendo riesgo de variación de precio y por tanto no estemos realizando arbitraje realmente.

2.7.2 Mercados en el arbitraje

Los diferentes mercados en los que se puede llevar a cabo esta estrategia son:

Mercados situados en diferentes lugares: Por ejemplo arbitraje entre Frankfurt y Madrid o Chicago.

Diferentes tipos de mercado: Por ejemplo, mercado de derivados y mercados al contado.
Las operaciones de arbitraje tienden a regular los mercados, ya que al vender en el mercado con mayor precio generan un aumento de la oferta que hace que el precio baje y al comprar en el mercado de menor precio generan un aumento de la demanda que hace el precio aumente (ley de la oferta y demanda).

Esta subida y bajada de precio continúa hasta el momento en el que ya no sale rentable hacer este tipo de operaciones. Es decir, cuando el precio de ambos mercados está igualado y por tanto en equilibrio.

2.7.3 Diferentes estrategias de arbitraje

Podemos distinguir varios tipos de arbitraje:

Arbitraje a dos puntos: Se trata de la diferencia entre los precios de dos mercados directamente.

Arbitraje a tres puntos o arbitraje triangular: En este caso es necesario utilizar tres mercados. La diferencia entre dos mercados es imperceptible, pero al llevarlo a un tercer mercado se hace más elevado. Por ejemplo, la diferencia entre tres pares de divisas, cambiando EUR/USD, luego USD/GBP y después EUR/GBP. Es un arbitraje muy difícil de observar y suele ser necesario herramientas informáticas para arbitrar.

Cabe destacar que se puede realizar arbitraje de tres o más puntos. Ahora bien, dada su complejidad lo pasaremos por alto en esta explicación. El objetivo es explicar de forma sencilla el concepto, no ahondar en sus dificultades.

La estrategia utilizada en este trabajo de investigación será la de dos puntos ya que haremos uso del intercambio de el par *USDT/DAI* en las diferentes plataformas de intercambio.

3 Marco Practico

3.1 Introduccion

El objetivo de este trabajo, como se mencionó anteriormente, es la creación de una token ERC-20 para usos académicos. Para poder realizar esto se necesitarán diferentes herramientas.

3.1.2 Herramientas y sus usos

3.1.2.1 NPM y NodeJS

La primera herramienta a utilizar va a ser NODE JS, más importante, el módulo NPM (Node package manager) de node js. Este permite instalar el resto de las herramientas necesarias para la creación de la token, así como mantener un registro del versionado de cada una.

3.1.2.2 Truffle Framework y Ganache

Truffle facilita la creación de Dapps en la red ethereum. Este framework brinda una variedad de herramientas para poder crear, testear e implementar smart Contracts en la red ethereum.
Para instalar este framework se puede ir a la terminal (en linux y macOs) y tipear
“Npm install -g truffle” o “sudo npm install -g truffle”.

Aparte del framework truffle, se necesita instalar la extensión “Ganache”. Esta es una blockchain local que permite desarrollar y testear smart contracts.
Se puede instalar directamente desde la pagina web “https://trufflesuite.com/ganache/”

3.1.2.3 Metamask

Metamask es una extensión del explorador Google Chrome, que funciona como una billetera virtual, así como un vínculo de conexión entre la blockchain de ethereum y una PC. Esta extensión funciona para Google Chrome, Firefox y Opera de forma estable, aunque debería funcionar bien en cualquier explorador basado en Chromium.
Para instalarlo se puede ir a la página principal de metamask “https://metamask.io/” y seguir las instrucciones en la misma. Una vez instalada la extensión, hay que asegurarse de que esté activa en el buscador y crear una cuenta en meta mask.

3.1.2.4 Resaltador de sintaxis y editor de texto.

Ya que la mayoría de los IDEs no admiten el resaltador de sintaxis para código escrito en el lenguaje solidity, hace falta instalar un paquete extra en el editor de texto a utilizar.
Como editor de texto se utilizará Sublime Text, el cual es un editor open source con gran flexibilidad de lenguajes y para el resaltador de sintaxis se va a utilizar un paquete llamado “Ethereum”.

Para instalar este paquete en sublime text primero se debe agregar el control de paquetes en “herramientas/instalar control de paquetes.”. Una vez hecho esto, yendo a “Sublime text/Preferencias/Control de paquetes” se puede instalar buscando por el nombre “Ethereum” en la barra de búsqueda

3.2 Casos de uso y requerimientos funcionales

3.2.1 Requerimientos funcionales

R1 - El token deberá tener un nombre, símbolo y decimales que representen el precio del token.

- *Fundamento*: Los usuarios deben tener acceso a la información básica del token, esto incluye el nombre, el símbolo, usualmente marcado como siglas, y el precio base del mismo. El sistema deberá no solo tener esos atributos sino poder mostrarlos a los usuarios. Este requerimiento no solo es obligatorio para cumplir con el estándar ERC-20 sino que también es necesario para una utilización apropiada del token.
- *Prioridad*: Alta
- *Volatilidad*: Baja
- *Criticidad*: Alta
- *Factibilidad*: Alta
- *Costo de implementación*: Bajo
- *Riesgo*: Bajo

Casos de uso

Consultar información del token.

- Actores: Usuario (consumidor)

- Inputs:
- Outputs:
    - Información del token
- Precondiciones
    - El usuario debe tener acceso a la página de compra del token
- Post Condiciones
    - Información mostrada
- Trigger

Casos de uso alternativo

Consultar información del token(consola/Backend)

- Actores: Usuario (consumidor)
- Inputs:
    - Función invocada por javascript que solicite la información
- Outputs:
    - Información del token
- Precondiciones
    - El usuario debe tener acceso a la red ethereum y conocimientos en informática para realizar la conexión al Smart Contract, así como una computadora que le permita invocar las funciones.
- Post Condiciones
    - Información mostrada
- Trigger
    - Ejecución del script solicitando la información

R2 - Se debe poder consultar cuántos tokens totales hay circulando, así como cuantos fueron vendidos

- *Fundamento*: Ayuda a mantener un registro de ventas, y mostrar a los usuarios la disponibilidad del token, así como cumplir con uno de los estándares ERC-20.
- *Prioridad*: Alta
- *Volatilidad*: Baja
- *Criticidad*: Alta
- *Factibilidad*: Alta
- *Costo de implementación*: Medio
- *Riesgo*: Bajo

Casos de uso

Consultar tokens en circulamiento.

- Actores: Usuario (consumidor)
- Inputs:
- Outputs:
    - Información del token
- Precondiciones
    - El usuario debe tener acceso a la página de compra del token
- Post Condiciones
    - Información mostrada
- Trigger

Casos de uso alternativo

Consultar tokens en circulamiento(consola/Backend)

- Actores: Usuario (consumidor)
- Inputs:
    - Función invocada por javascript que solicite la información
- Outputs:
    - Información del token
- Precondiciones
    - El usuario debe tener acceso a la red ethereum y conocimientos en informática para realizar la conexión al Smart Contract, así como una computadora que le permita invocar las funciones.
- Post Condiciones
    - Información mostrada
- Trigger
    - Ejecución del script solicitando la información

R3 - El sistema permitirá integrarse con la billetera digital de criptomonedas MetaMask.

- *Fundamento*: Los usuarios deben de tener control absoluto de sus fondos y cómo utilizarlos, por ello se debe de facilitar la integración con MetaMask para que puedan hacer uso de sus billeteras para los aportes y retiros de fondos.
- *Prioridad*: Alta
- *Volatilidad*: Baja
- *Criticidad*: Alta

- *Factibilidad*: Baja
- *Costo de implementación*: Medio
- *Riesgo*: Medio

Casos de uso

Utilizar billetera de metamask

- Actores: Usuarios (todos), Sistema
- Inputs: -
- Outputs: -
- Precondiciones:
  - Meta Mask instalado en el navegador del usuario.
- Post Condiciones:
  - Plataforma captura datos de la billetera del usuario
- Trigger:
  - Ir a ruta indice del proyecto

R4 - Se debe poder intercambiar ether por tokens y viceversa

- *Fundamento*: La forma principal de adquisición del token es el intercambio de la criptomoneda Ether (ETH) por el mismo. Partiendo de esta premisa, los usuarios deben poder intercambiar dicha criptomoneda por el token en cuestión y viceversa. El objetivo del proyecto es que, no solo lo puedan hacer a través de la plataforma inicial, sino que cada institución provea un servicio de intercambio para “comprar” tokens que ellos acepten como forma de pago. De esta forma el token seguirá en circulación y las instituciones pueden utilizar el ether obtenido como forma de pago para cambiar por moneda física y generar ganancia con los intercambios o venta de servicios usando el token.
- *Prioridad*: Alta
- *Volatilidad:* Media
- *Criticidad*: Alta
- *Factibilidad*: Alta
- *Costo de implementación:* Alto
- *Riesgo*: Alto

Casos de uso

Adquirir tokens.

- Actores: Usuario (consumidor)
- Inputs:
  - Cantidad a adquirir
- Outputs:
  - Mensaje de confirmación de adquisición
- Precondiciones
  - El usuario debe tener acceso a la página de compra del token
  - El usuario debe tener una cuenta registrada de metamask
  - El usuario debe tener ETH en su billetera de metamask
  - El usuario debe tener la extension metamask en su navegador
- Post Condiciones
  - Transacción completada
- Trigger
  - Ejecución de la transacción

R5 - Se debe poder transferir tokens de una cuenta a otra.

- *Fundamento*: El objetivo de este proyecto es, que el token creado, pueda ser utilizado como método de pago para varios productos y servicios diferentes. Para ello el requerimiento mínimo para que esto suceda es que se pueda realizar un intercambio del mismo entre diferentes usuarios y/o instituciones.
- *Prioridad*: Alta
- *Volatilidad*: Baja
- *Criticidad*: Alta
- *Factibilidad*: Alta
- *Costo de implementación:* Medio
- *Riesgo*: Alto

Casos de uso

Transferir tokens.

- Actores: Usuario (consumidor), Institución.
- Inputs:
  - Cantidad a transferir
  - Cuenta a la cual transferir

    - (de ser necesario, y dependiendo del motivo de la transferencia) cuenta de la cual se está transfiriendo.
- Outputs:
    - Mensaje de confirmación de transferencia.
- Precondiciones
    - El emisor debe tener metamark instalado en su navegador.
    - El emisor debe tener Tokens en su billetera y esta debe estar conectada a metamask.
    - El receptor debe tener una cuenta creada en Metamask o, un address de una billetera virtual a la cual poder transferir.
- Post Condiciones
    - Transferencia realizada, mensaje de transferencia realizada mostrado.
- Trigger
    - Realizar transferencia.

Casos de uso alternativo

Transferir tokens(consola/Backend)

- Actores: Usuario (consumidor)
- Inputs:
    - Función invocada por javascript que solicite la transferencia de tokens
    - Address de la cuenta del emisor
    - Address de la cuenta del receptor
    - Cantidad a transferir
- Outputs:
    - Mensaje de confirmación de transferencia.
- Precondiciones
    - El usuario debe tener acceso a la red ethereum y conocimientos en informática para realizar la conexión al Smart Contract, así como una computadora que le permita invocar las funciones.
    - El emisor debe tener tokens en su cuenta.
    - Ambos actores deben tener una billetera virtual, y acceso al address de la misma.
    -
- Post Condiciones
    - Transferencia realizada.
- Trigger
    - Ejecución del script solicitando la transferencia.

R6 - Se debe poder otorgar una mesada a otro usuario.

- Fundamento: Un usuario (A) debe poder otorgarle una mesada a otro usuario (B). Estas tokens a utilizar por el usuario B, serán descontadas del balance del usuario A, habiendo un límite mensual definido por el valor de mesada otorgado. El objetivo a lograr con esto es, darle a un usuario la capacidad de “enviar tokens” a otro de forma controlada y que no requiera invertir tiempo mes a mes para hacerlo. Esta mesada se puede aumentar o disminuir o cancelar, ya que el valor de la mesada se sobreescribe cada vez que se ejecuta la función.
- Prioridad: Alta
- Volatilidad: Baja
- Criticidad: Alta
- Factibilidad: Alta
- Costo de implementación: Medio
- Riesgo: Medio

Casos de uso

Otorgar Mesada

- Actores: Usuario (consumidor).
- Inputs:
  - Cantidad a permitir
  - Cuenta a la cual permitir uso de tokens
- Outputs:
  - Mensaje de confirmación de allowance(Mesada).
- Precondiciones
  - El emisor debe tener metamark instalado en su navegador.
  - El emisor debe estar conectado a metamask.
  - El receptor debe tener una cuenta creada en Metamask o, un address de una billetera virtual a la cual poder otorgar la mesada.
  - Acceder a la pestaña de allowance de la página principal del token.
- Post Condiciones
  - Mesada otorgada, mensaje de confirmación mostrado.
- Trigger

Casos de uso alternativo

Otorgar Mesada(consola/Backend)

- Actores: Usuario (consumidor)
- Inputs:
    - Función invocada por javascript que solicite concesión de mesada
    - Address de la cuenta del emisor
    - Address de la cuenta del receptor
    - Cantidad a permitir.
- Outputs:
    - Mensaje de confirmación de mesada otorgada.
- Precondiciones
    - El usuario debe tener acceso a la red ethereum y conocimientos en informática para realizar la conexión al Smart Contract, así como una computadora que le permita invocar las funciones. .
    - Ambos actores deben tener una billetera virtual, y acceso al address de la misma.

- Post Condiciones
    - Mesada otorgada
- Trigger
    - Ejecución del script solicitando la concesión de mesada.

R7 - Se debe poder consultar cuantas tokens tiene una cuenta.

- *Fundamento*: Los usuarios deberán poder ver el balance de una cuenta, y un administrador debe poder consultar el balance de otros usuarios siempre y cuando disponga del address de la cuenta que quiera consultar.
- *Prioridad:* Alta
- *Volatilidad:* Baja
- *Criticidad:* Alta
- *Factibilidad:* Alta
- *Costo de implementación:* Medio
- *Riesgo:* Medio

Casos de uso

Consultar balance

- Actores: Usuario (consumidor).

- Inputs:
- Outputs:
    - Balance del usuario
- Precondiciones
    - El Usuario debe tener metamark instalado en su navegador.
    - El Usuario debe estar conectado a metamask.
    - El receptor debe tener una cuenta creada en Metamask.
    - El usuario debe agregar el address del contrato del token a metamask, para que la información del token pueda ser mostrado en la billetera.
- Post Condiciones
    - Cantidad de tokens mostrada.
- Trigger

Casos de uso alternativo

Consultar balance(consola/Backend)

- Actores: Usuario (consumidor)
- Inputs:
    - Función invocada por javascript que solicite la cantidad de tokens en una cuenta
    - Address de la cuenta a consultar
- Outputs:
    - Mensaje de confirmación de mesada otorgada.
- Precondiciones
    - El usuario debe tener acceso a la red ethereum y conocimientos en informática para realizar la conexión al Smart Contract, así como una computadora que le permita invocar las funciones.
    - El usuario debe tener una billetera virtual, y acceso al address de la misma.

R8 - Se debe poder utilizar la token como medio de pago/cobro.

- Fundamento: Es el objetivo principal de este trabajo, y utilización práctica del requerimiento funcional 5 “Transferir tokens”. El motivo por el cual se debe poder transferir tokens es para poder utilizarlos como medio de pago, principalmente. Este requerimiento depende altamente del uso que den las instituciones al mismo. Ya que para poder utilizarlo como medio de pago tiene que haber un producto o servicio a adquirir a cambio del token. Debido a esto, el requerimiento funcional 8 es, en esencia, la utilización práctica del RF5.
- Prioridad: Alta
- Volatilidad: Baja
- Criticidad: Alta

- Factibilidad: Alta
- Costo de implementación: Bajo
- Riesgo: Alto

Consultar balance

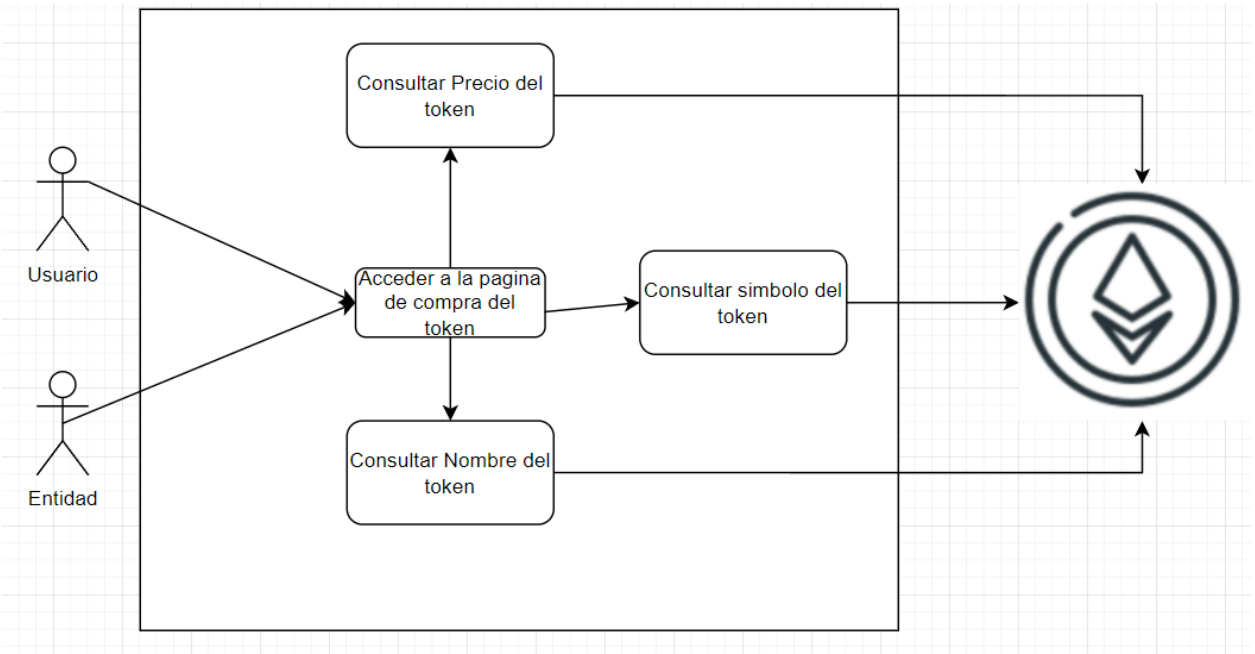
- Actores: Usuario (consumidor), institución.
- Inputs:
- Outputs:
- Precondiciones
    - La institución debe tener una página donde poder adquirir productos a cambio de tokens.
    - El Usuario debe estar conectado a metamask.
    - El receptor debe tener una cuenta creada en Metamask.
    - El usuario debe tener tokens en su billetera.

- Post Condiciones
    - Cantidad de tokens mostrada.
- Trigger
    - Adquisición de un producto o servicio a través de la página de la institución.

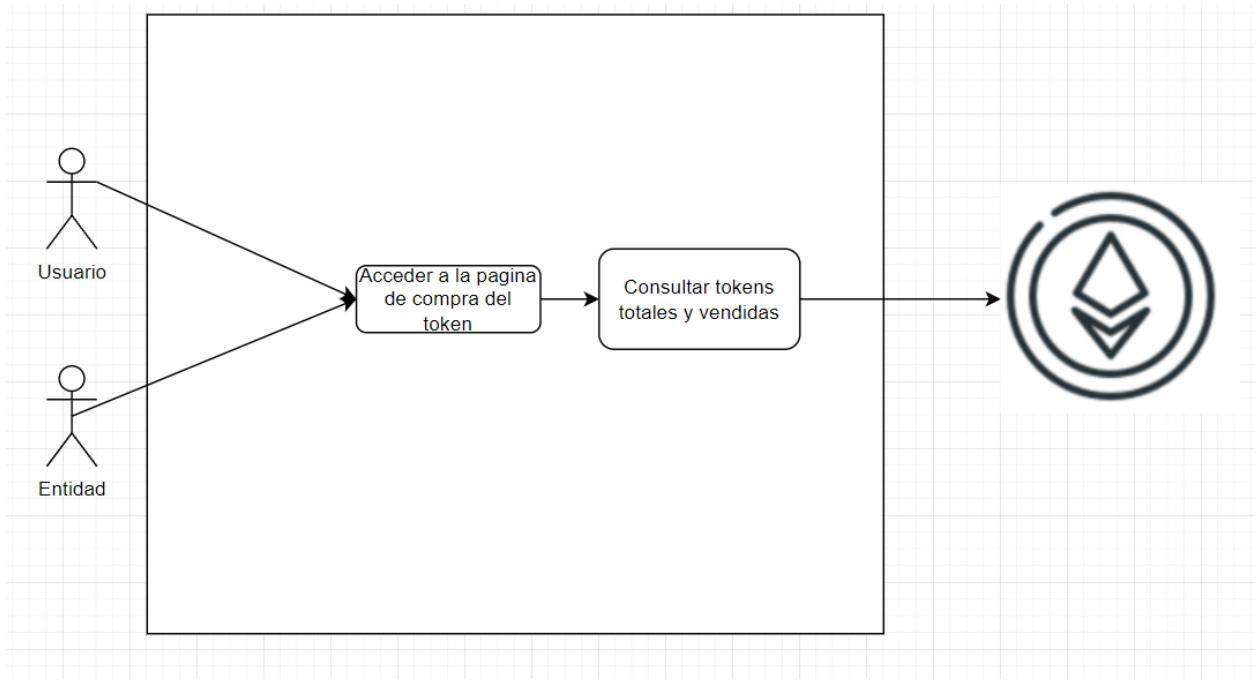
3.2 Diagramas

3.2.1 Diagramas casos de uso

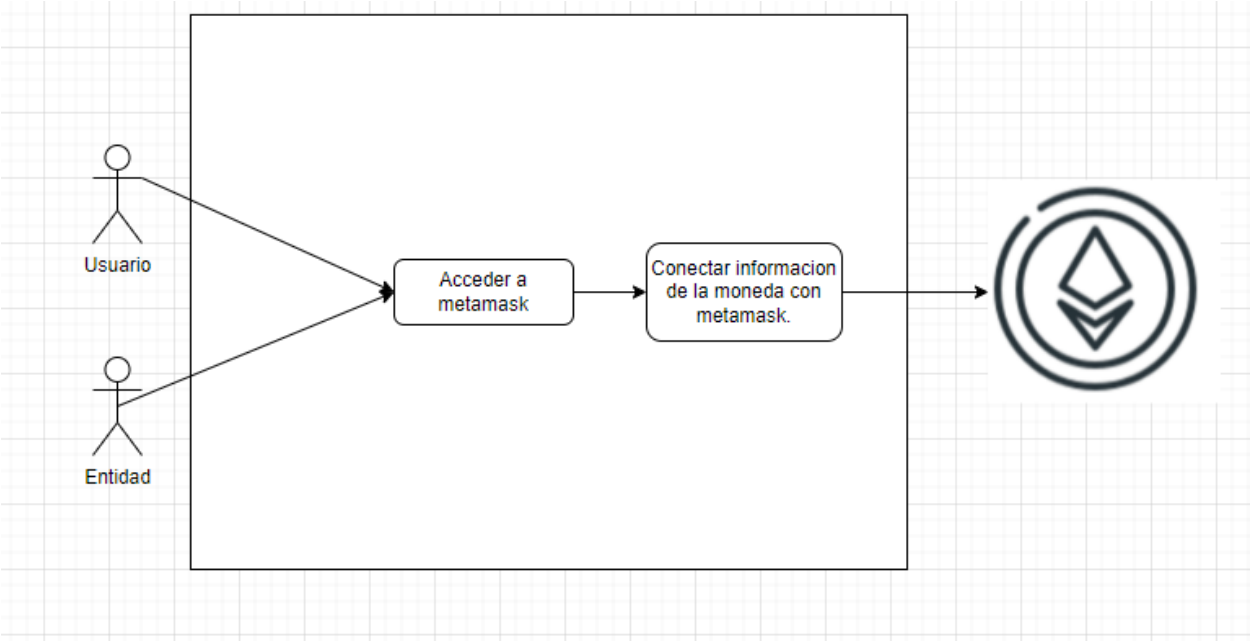
3.2.1.1 Consultar información del Token.



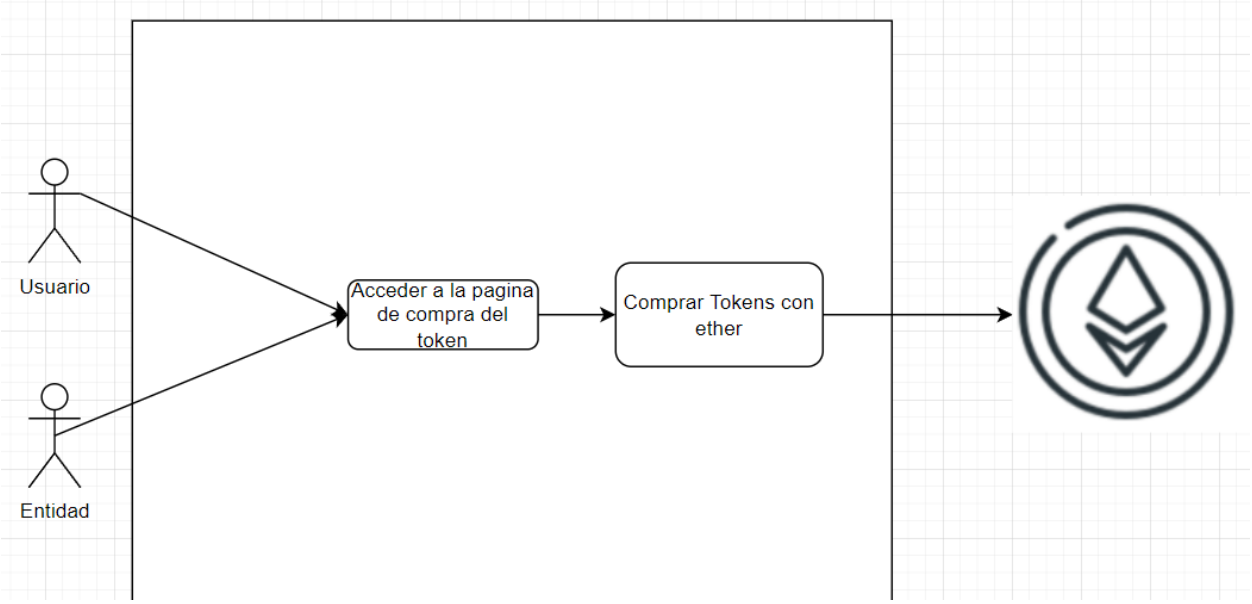
3.2.1.2 Consultar tokens totales y vendidas



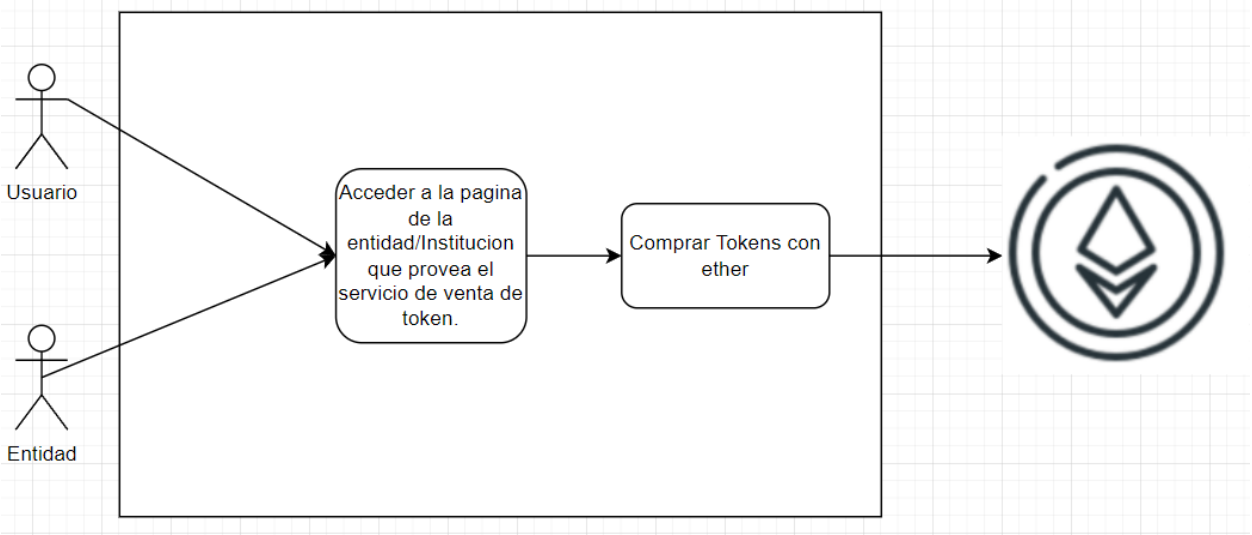
3.2.1.3 Usar billetera de metamask



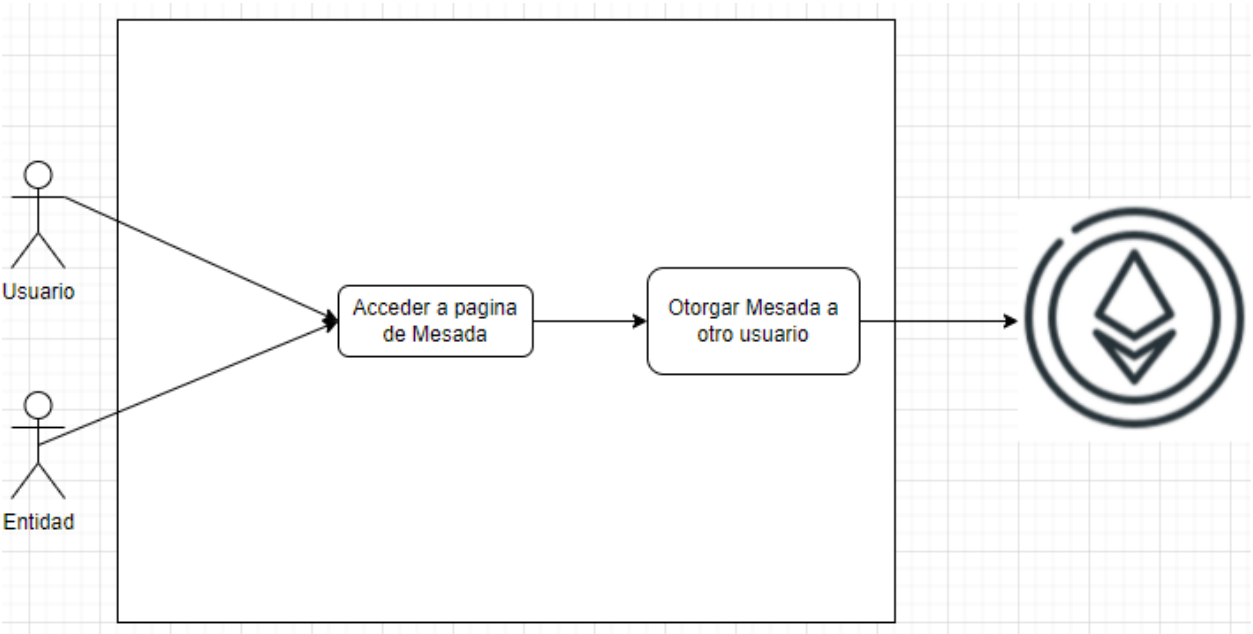
3.2.1.4 Comprar tokens con ether



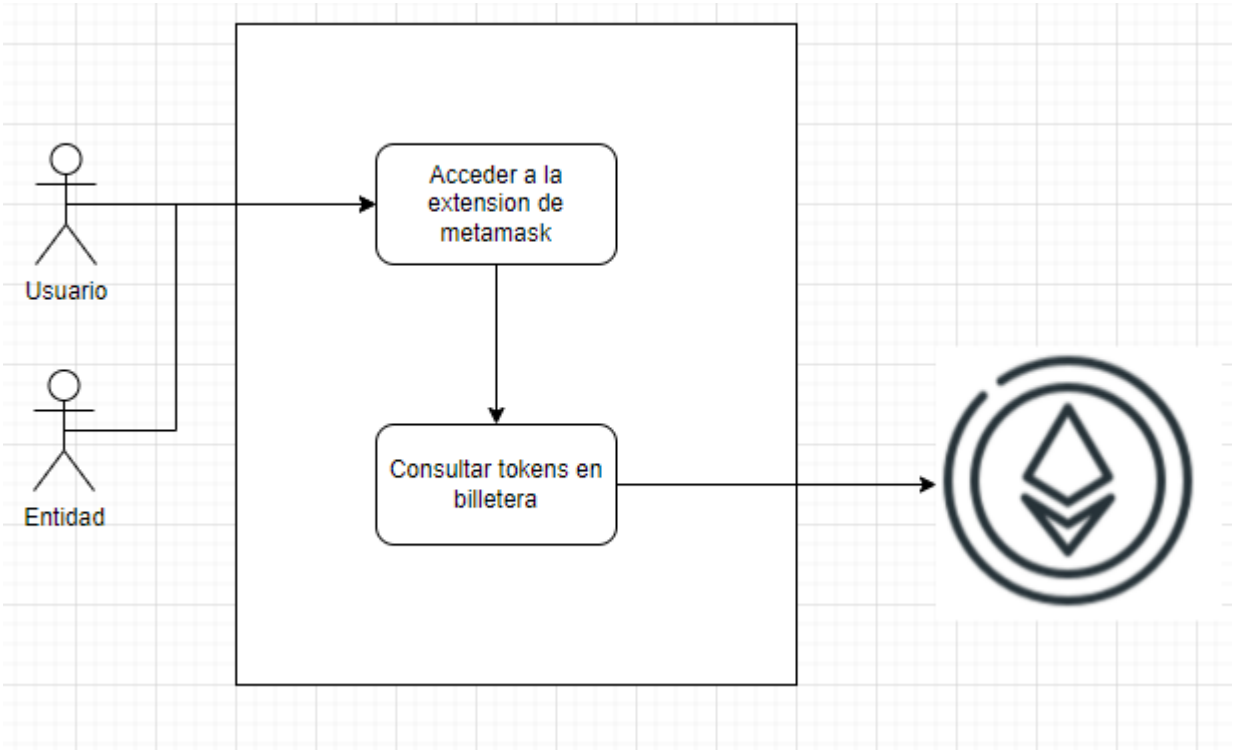
3.2.1.4 (bis) Comprar tokens con ether.



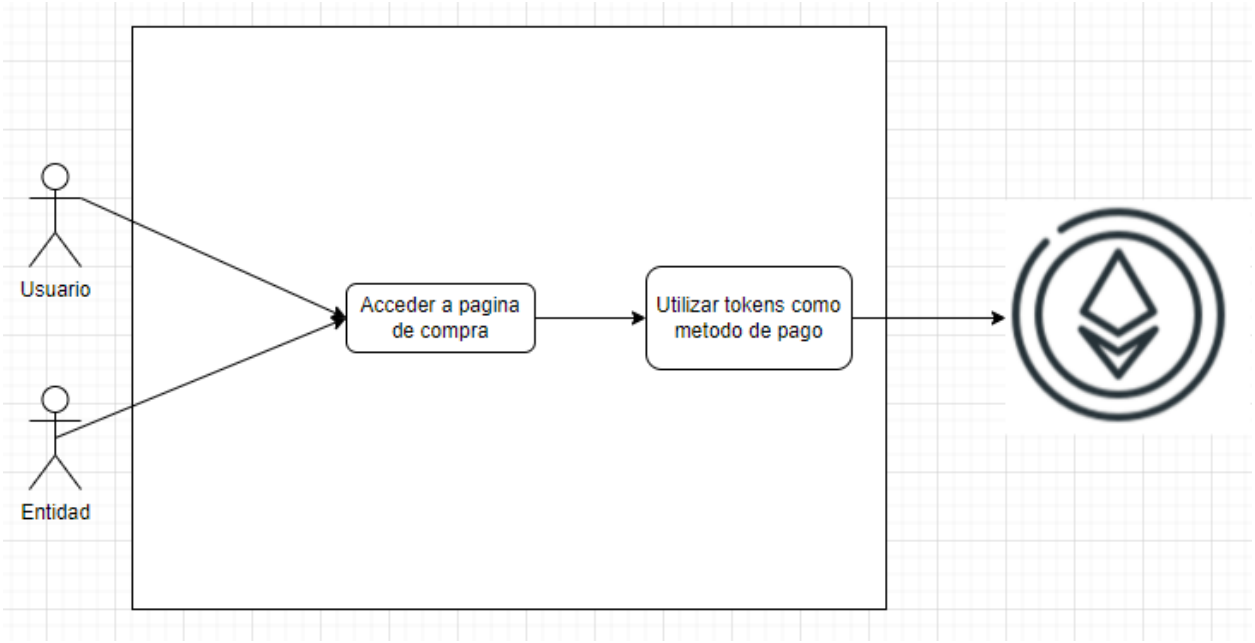
3.2.1.5 Otorgar Mesada



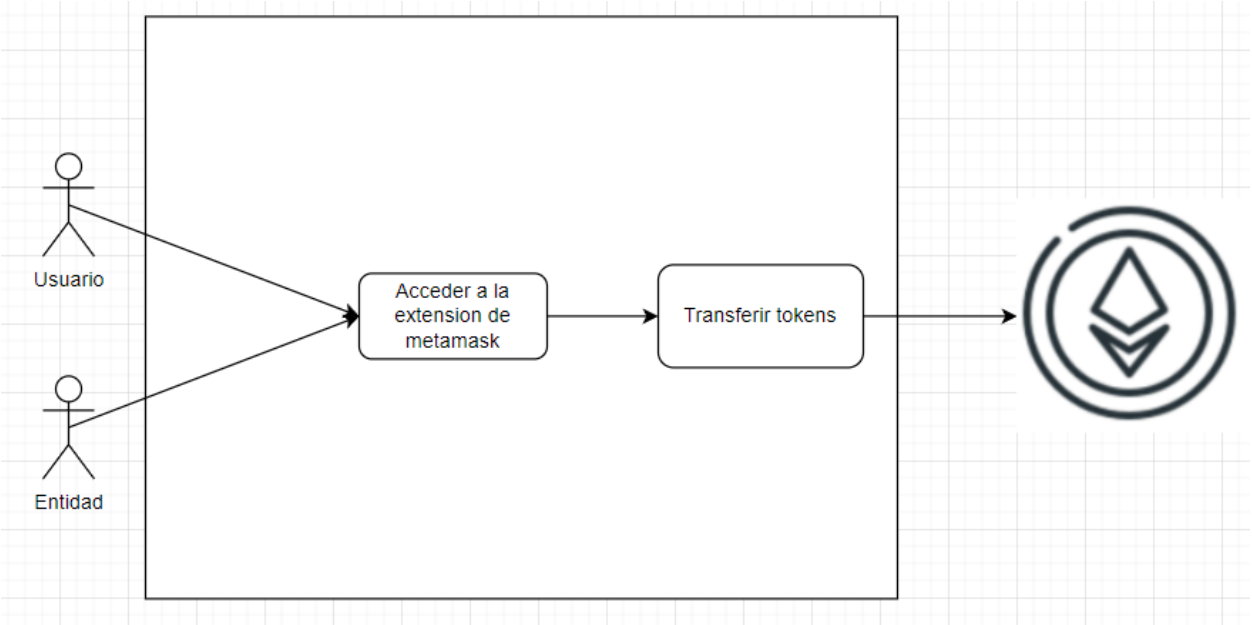
3.2.1.6 Consultar cantidad de tokens en una billetera.



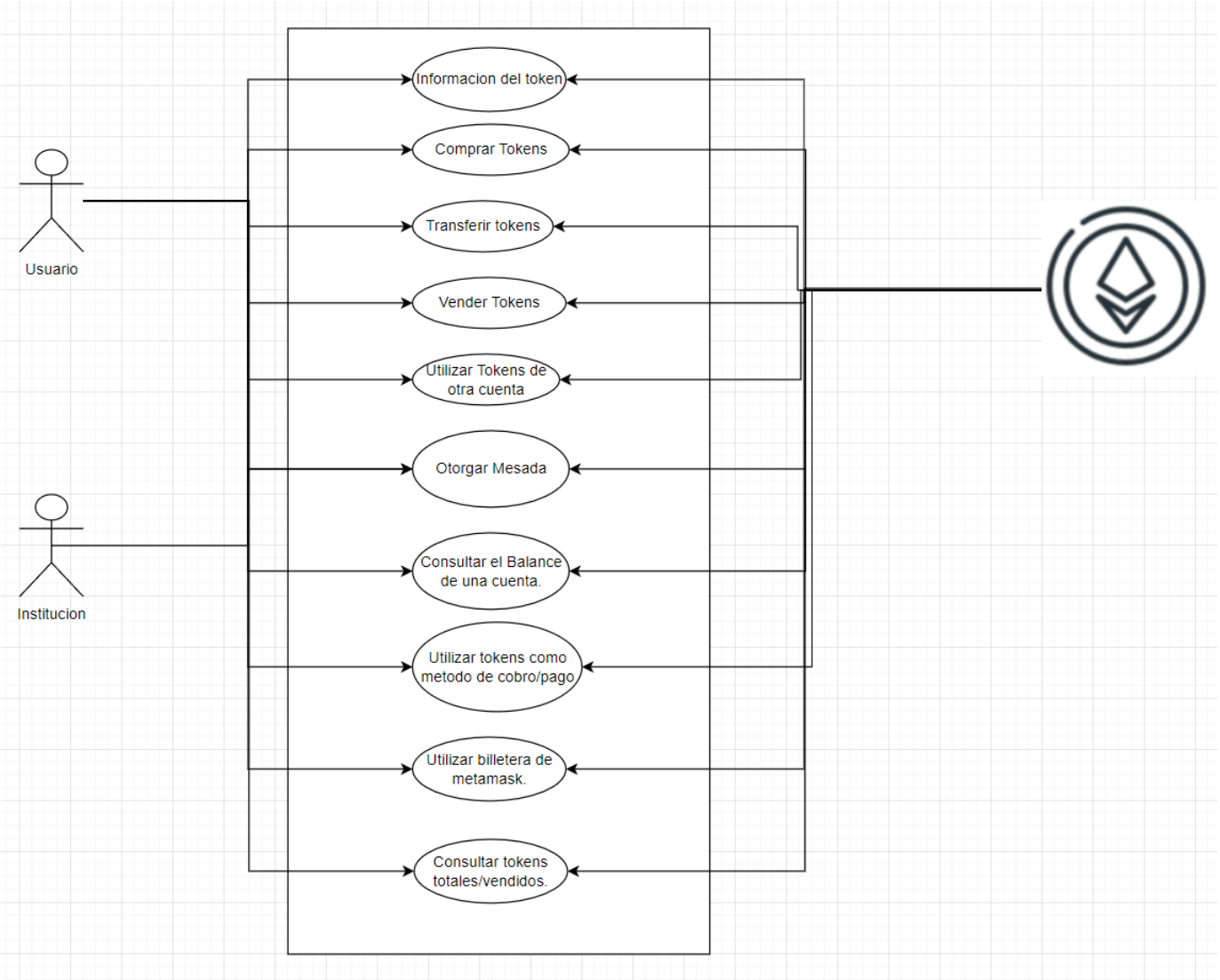
3.2.1.Utilizar tokens como método de pago/cobro



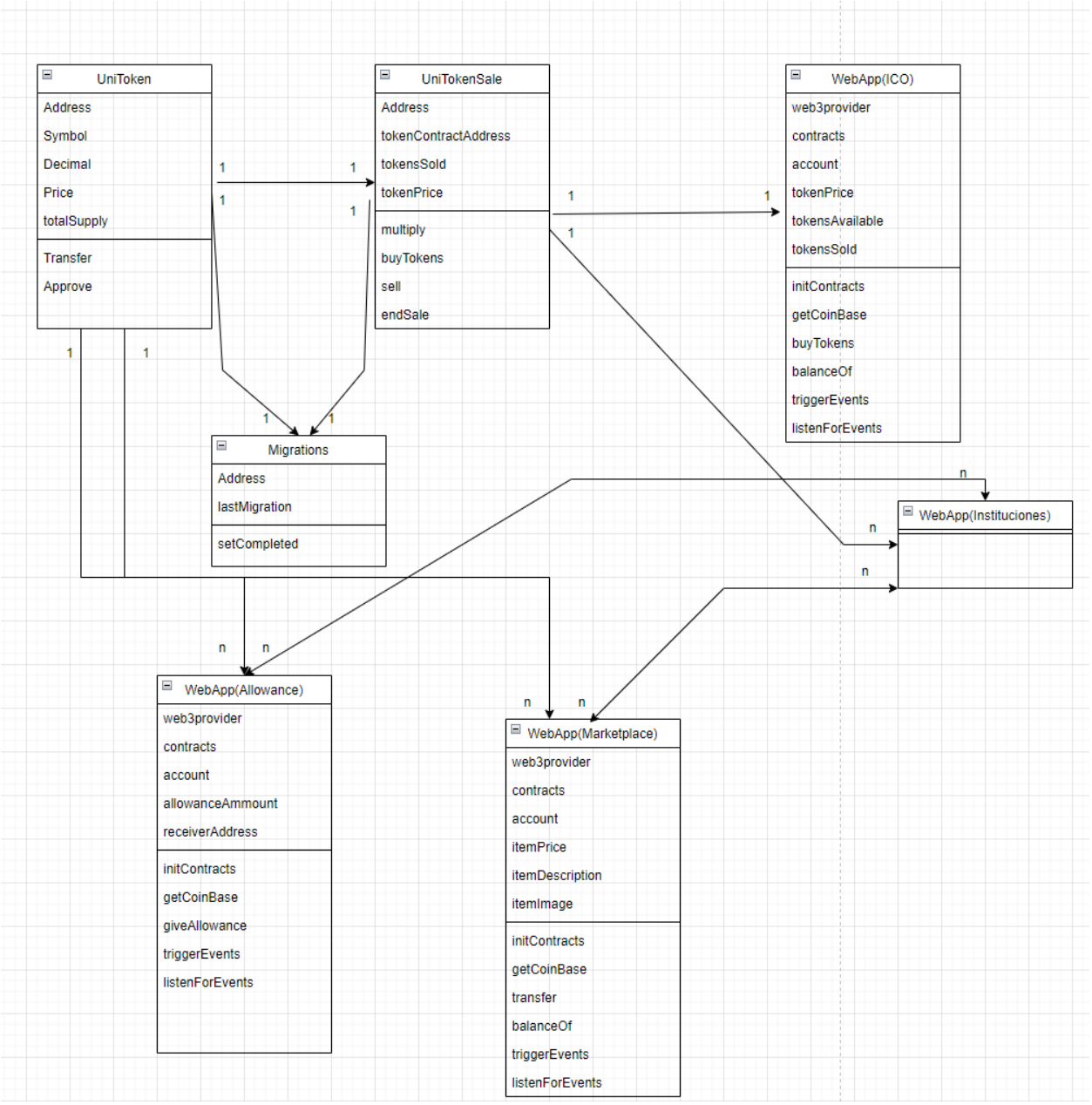
3.2.1.8 Transferir tokens



3.2.2 Diagrama de casos de uso - Contexto

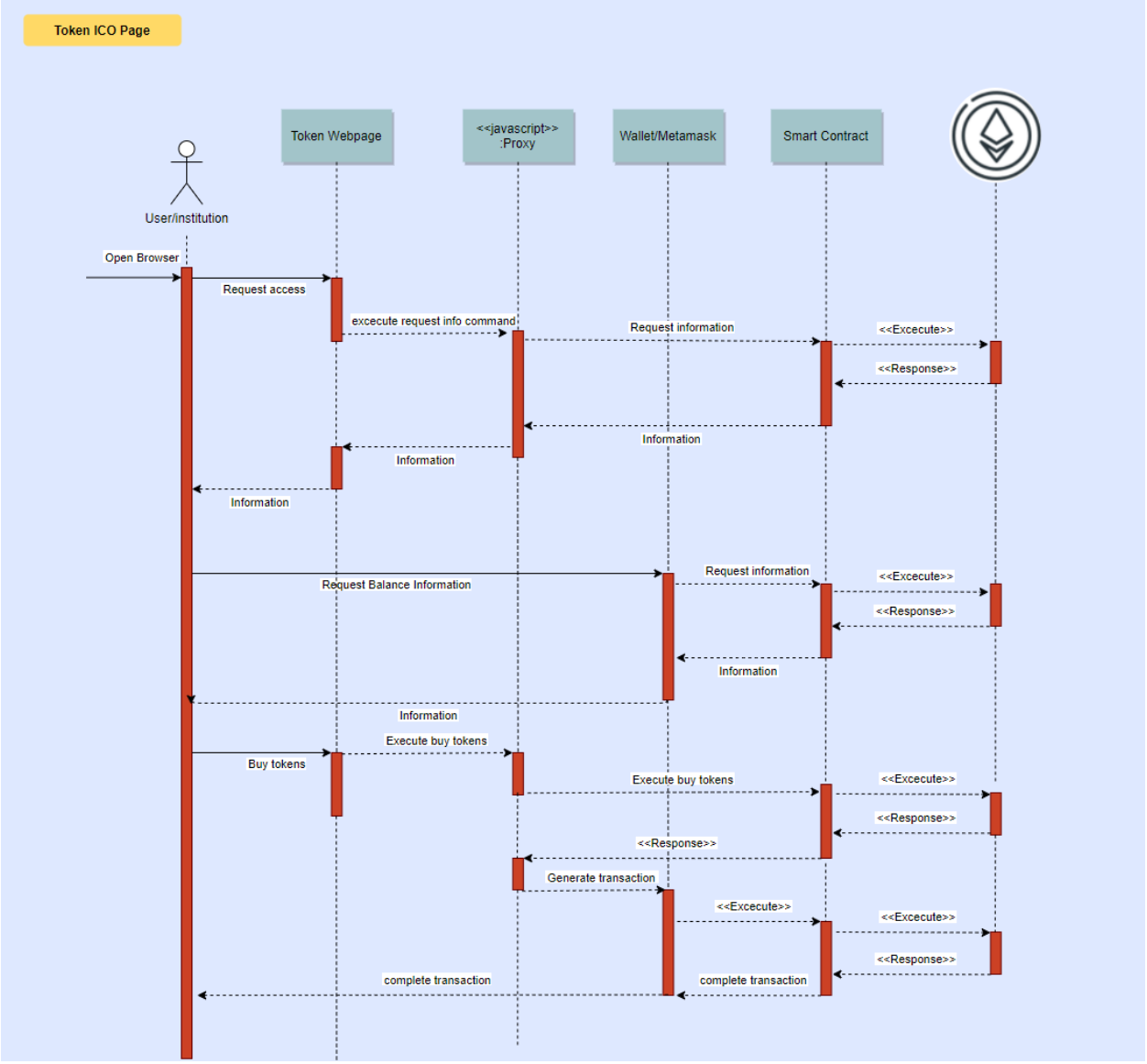


3.2.3 Diagrama de Clases

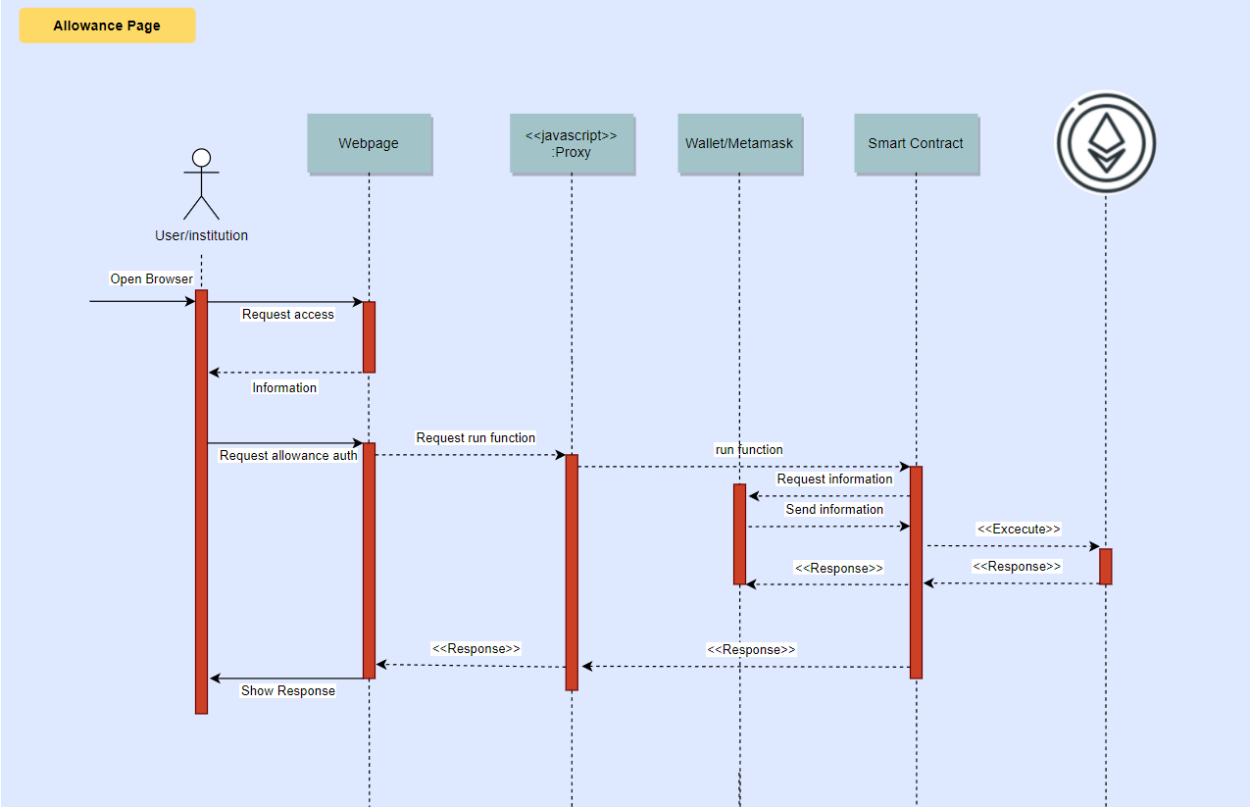


3.2.5 Diagrama de secuencias

3.2.5.1 Pagina de compra del token.

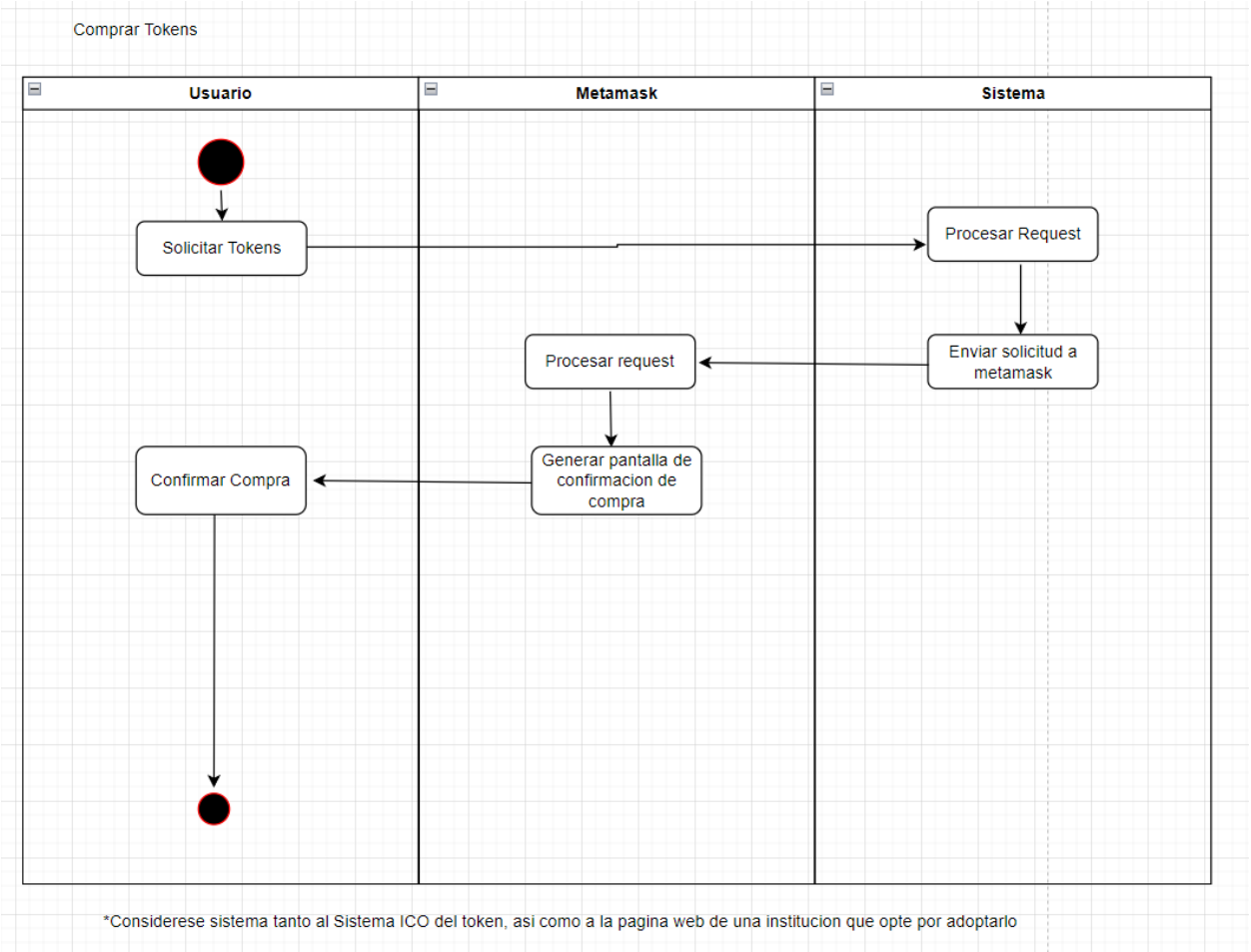


3.2.5.2 Pagina de Allowance

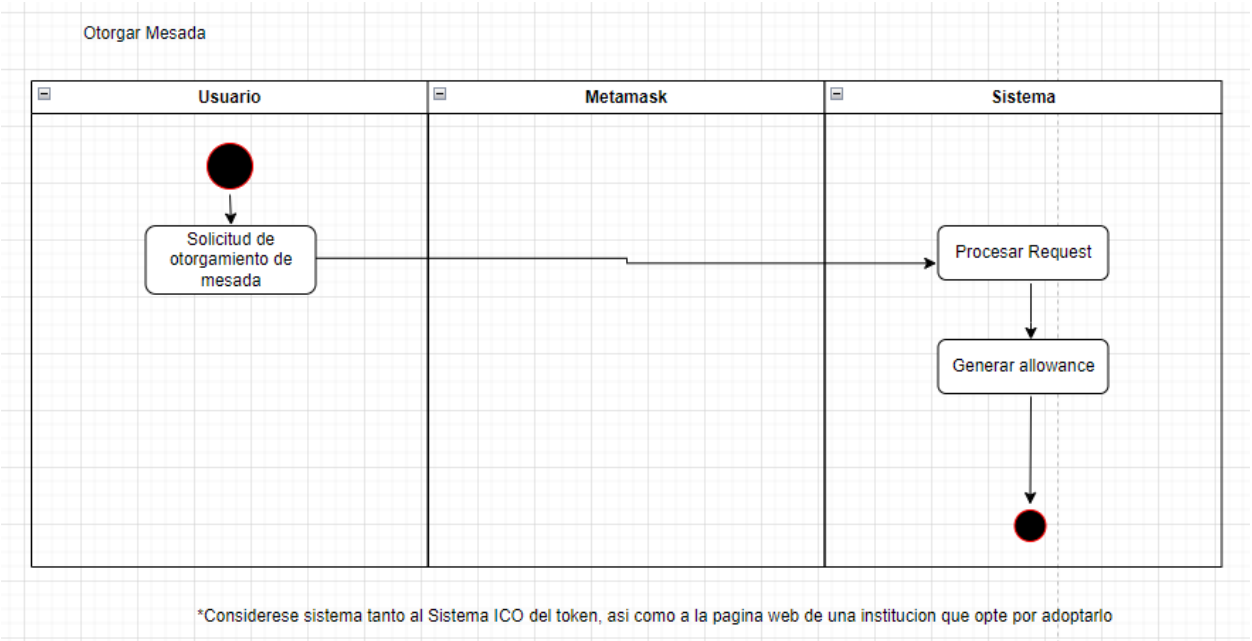


3.2.6 Diagrama de Actividades(Token)

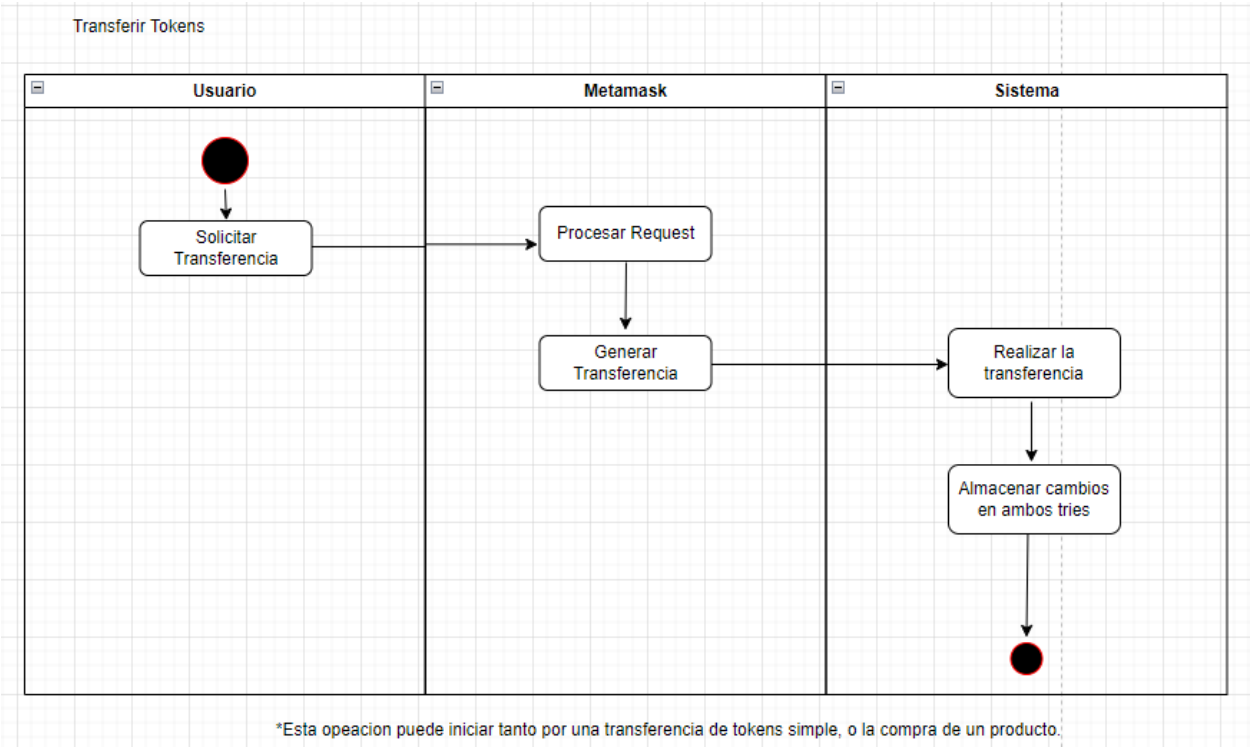
3.2.6.1 Comprar tokens



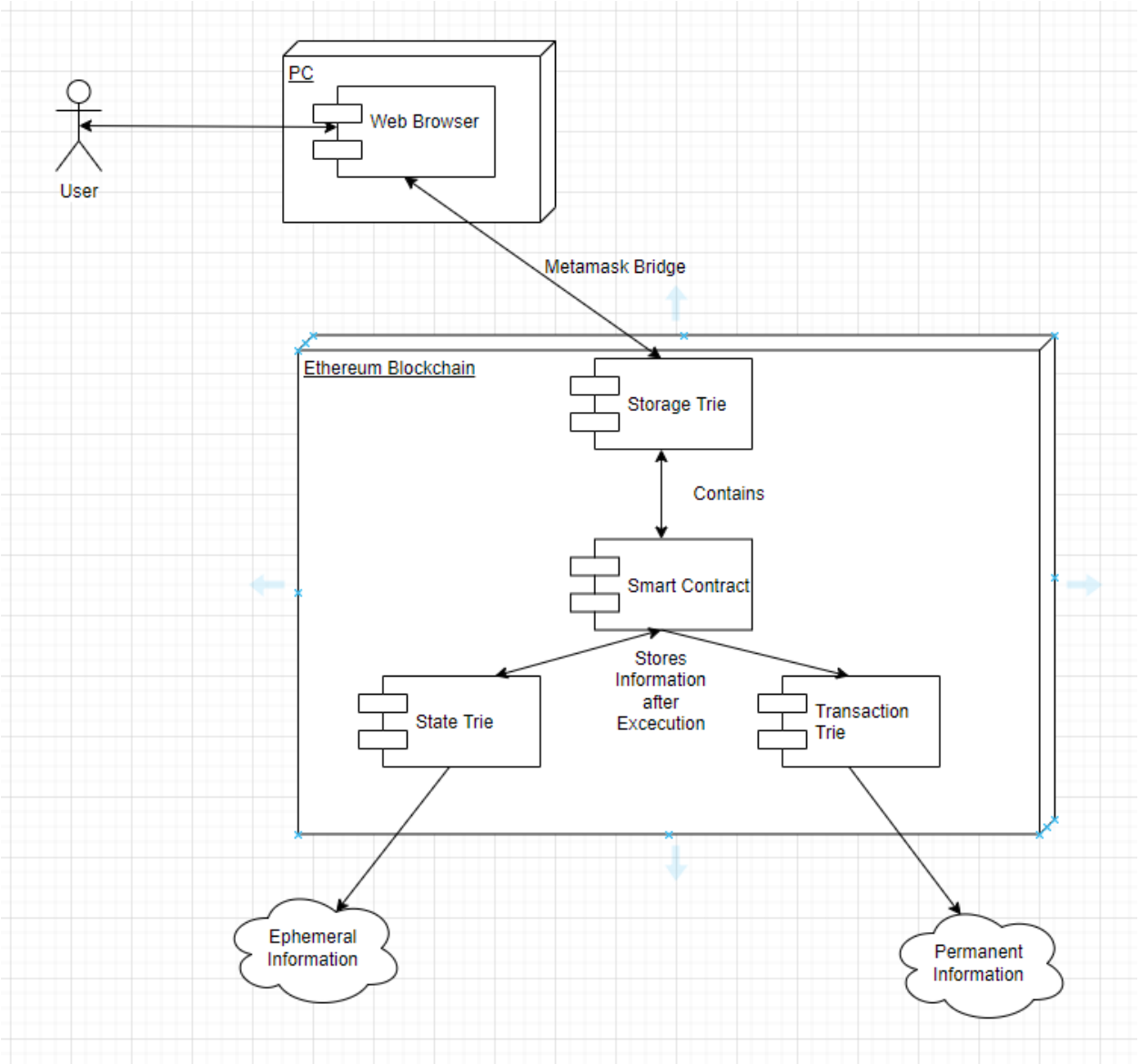
3.2.6.2 Otorgar Mesada



3.2.6.2 Transferir Tokens



3.2.7 Diagrama de despliegue



3.3 Manual de Usuario

Manual de usuario UniToken v1.0

3.3.1 Documentación de despliegue

3.3.1.1 Requerimientos

- Git
- Node.Js
- Solidity
- bash / zsh

3.3.1.2 Plataformas auxiliares

- Metamask
- Github
- Ganache
- Truffle

3.3.1.4 Instalación para testeo local

Clonar repositorio

El primer paso consta de una vez instalado y seteado git en la terminal, en clonar el repositorio del proyecto de la siguiente manera

```
git clone git@github.com:turuletee/Thesis.git
```

Instalar dependencias

Una vez clonado el repositorio y node.js instalado en la terminal, el siguiente paso es instalar las dependencias pertenecientes a este, para ello nos debemos dirigir a la carpeta donde el repositorio fue clonado y posicionarnos *dentro* del proyecto, por ejemplo.

```
nachofernandez@Nachos-MBP ~ % cd documents
nachofernandez@Nachos-MBP documents % cd thesis
nachofernandez@Nachos-MBP thesis % cd dappToken
```

El siguiente comando a ingresar es

`npm install`

el cual instalará las dependencias definidas en el archivo package-lock.json

---

3.3.1.5 Testeo

Tras tener las dependencias instaladas, es importante verificar que la última versión de los contratos sea funcional y retrocompatible con viejas funcionalidades para ello debemos de correr los tests del proyecto, el siguiente comando nos permitirá correrlos.

`truffle test`

Para que el comando se ejecute, primero habrá que hacer un deployment local, el cual está indicado en el siguiente paso, pero es importante previo a hacer un deployment en EVM probar que funcione localmente todo. De los tests, se deberá obtener el siguiente output:

```
nachofernandez@Nachos-MBP dappToken % truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: UniToken
    ✓ initializes the contract with the correct values (120ms)
    ✓ sets the total supply upon deployment (68ms)
    ✓ transfers token ownership (556ms)
    ✓ approves tokens for delegated transfers (279ms)
    ✓ handles delegated token transfers (792ms)

  Contract: UniTokenSale
    ✓ initializes the contract with correct values (61ms)
    ✓ facilitates token Buying (526ms)
    ✓ ends token Sale (305ms)


  8 passing (3s)
```

3.3.2 Despliegue Contratos

3.3.2.1 Requerimientos

Los contratos van a estar desplegados en la red ethereum. No se requiere re-deployment de los mismos ya que para el uso del token, solo se requiere tener acceso a la página web ICO del mismo.

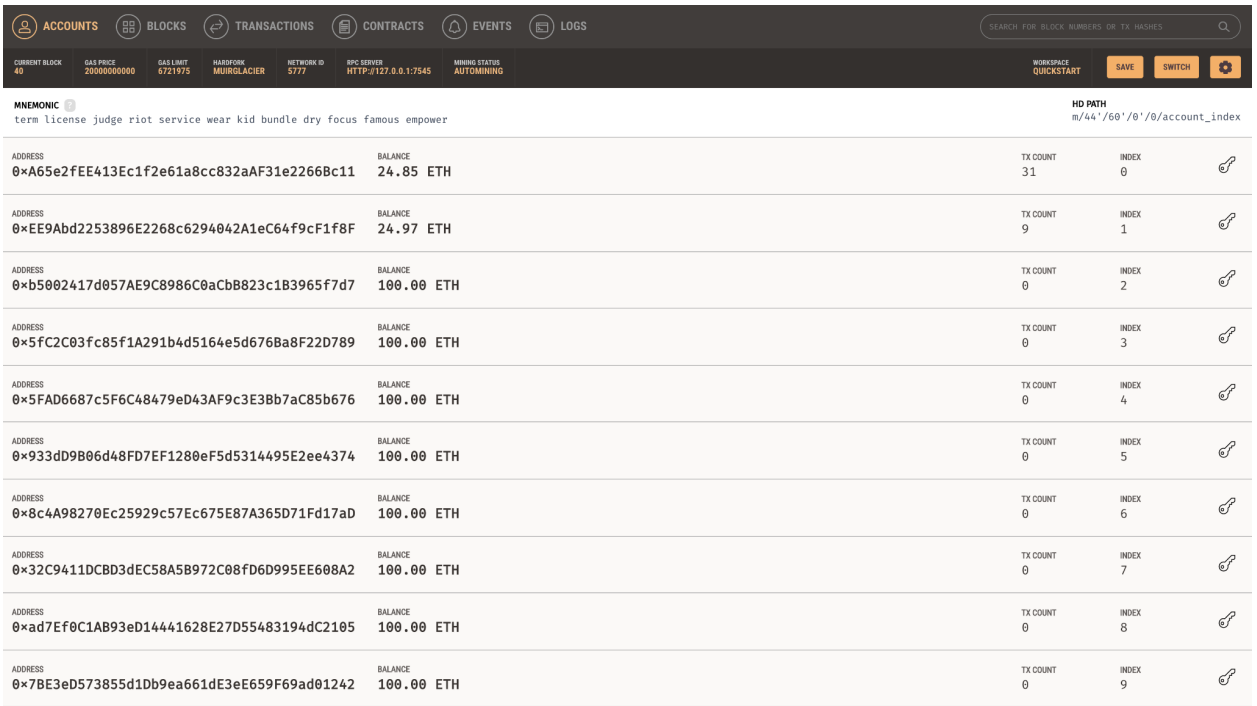
3.3.3 Despliegue local webapp

3.3.3.1 Requerimientos

Para realizar un despliegue local, principalmente para motivos de testeo, se debe tener instalado Truffle, ganache y Node en el computador. Una vez que se tienen esos 3 softwares instalados se procede con el despliegue

3.3.3.2 Ejecución

Tras clonar e instalar las dependencias y los software necesarios para la ejecución, el siguiente paso es abrir ganache. Ganache genera cuentas ficticias con eth en ellas para poder realizar testeos de forma local y centralizada.



Una vez ejecutado Ganache y asegurando que el repositorio este clonado, desde la terminal se correrán los siguientes comandos:

```
truffle migrate
```

Este comando despliega los contratos en la red local creada con truffle. Se debe ver lo siguiente en consola

```
1_initial_migration.js
======================

   Replacing 'Migrations'
   ----------------------
   ⠏ Blocks: 0            Seconds: 0   > transaction hash:    0xcf67e56b6dafa110594605d9491b0b4caa78e6fc50d7bdfc50bd1f99b64f82b4
   > Blocks: 0            Seconds: 0
   > contract address:    0xAc701154D4f1247ADF0d6198F92f3AcF7A8F62Be
   > block number:        1
   > block timestamp:     1645480431
   > account:             0xA65e2fEE413Ec1f2e61a8cc832aAF31e2266Bc11
   > balance:             99.99502316
   > gas used:            248842 (0x3cc0a)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00497684 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00497684 ETH
```

```
2_deploy_contracts.js
=====================

   Replacing 'UniToken'
   --------------------
   ⠏ Blocks: 0            Seconds: 0   > transaction hash:    0x28aecc795823e01baed83db6464b4ff14180cb71dd5d93eea043ec8c7b3e42a8
   > Blocks: 0            Seconds: 0
   > contract address:    0x82c163627EC815F2119648E00960bc7398E15D6c
   > block number:        3
   > block timestamp:     1645480431
   > account:             0xA65e2fEE413Ec1f2e61a8cc832aAF31e2266Bc11
   > balance:             99.97666244
   > gas used:            875523 (0xd5c03)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01751046 ETH


   Deploying 'UniTokenSale'
   ------------------------
   ⠏ Blocks: 0            Seconds: 0   > transaction hash:    0x9909a66b1575184390425bb5521e59d92870aff6facd49c8f9818499240103dd
   > Blocks: 0            Seconds: 0
   > contract address:    0xFe66743925A951CD6A184a33E1C3407a1Bf0D99E
   > block number:        4
   > block timestamp:     1645480432
   > account:             0xA65e2fEE413Ec1f2e61a8cc832aAF31e2266Bc11
   > balance:             99.96419282
   > gas used:            623481 (0x98379)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01246962 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.02998008 ETH
```

Una vez desplegados los contratos se deben correr los siguientes dos comandos:

```
truffle test
```

Este ejecutará los tests y lo siguiente deberá mostrarse en consola para asegurarse de que estos hayan corrido apropiadamente.

```
Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: UniToken
    ✓ initializes the contract with the correct values (120ms)
    ✓ sets the total supply upon deployment (68ms)
    ✓ transfers token ownership (556ms)
    ✓ approves tokens for delegated transfers (279ms)
    ✓ handles delegated token transfers (792ms)

  Contract: UniTokenSale
    ✓ initializes the contract with correct values (61ms)
    ✓ facilitates token Buying (526ms)
    ✓ ends token Sale (305ms)


  8 passing (3s)
```

Por último, una vez asegurado que todo este corriendo apropiadamente, se correrá el comando:

`npm run dev`

Obteniendo el siguiente output:

```
> university-token-sale@1.0.0 dev
> lite-server

** browser-sync config **
{
  injectChanges: false,
  files: [ './**/*.{html,htm,css,js}' ],
  watchOptions: { ignored: 'node_modules' },
  server: {
    baseDir: [ './src', './build/contracts' ],
    middleware: [ [Function (anonymous)], [Function (anonymous)] ]
  }
}
```

Con esto, se termina el despliegue local y se debe tener acceso a la pagina web desde la direccion 127.0.0.1:3000 o, para simplificar, localhost:3000

3.3.5 Documentación de uso

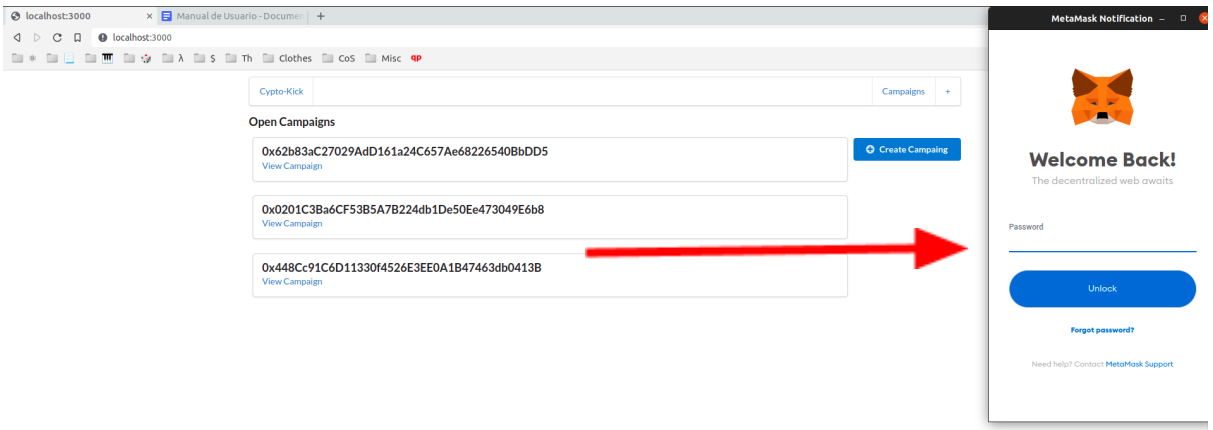
3.3.5.1 Requerimientos

- Navegador web
- Metamask

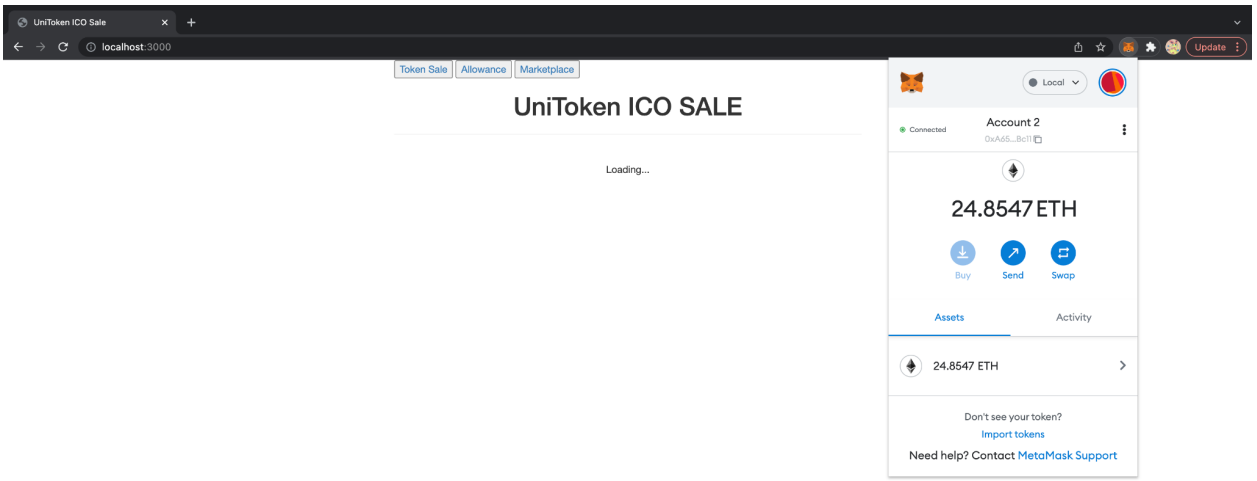
Es necesario y fundamental utilizar un navegador web basado en chromium, tales como Google Chrome, Brave u Opera y tener instalada la extensión de Metamask con una billetera ya integrada

3.3.5.2 Procesos

Una vez que se ingrese a la plataforma, en caso de que metamask no se encuentre inicializado, tendrá que ingresar sus credenciales como en la siguiente foto.

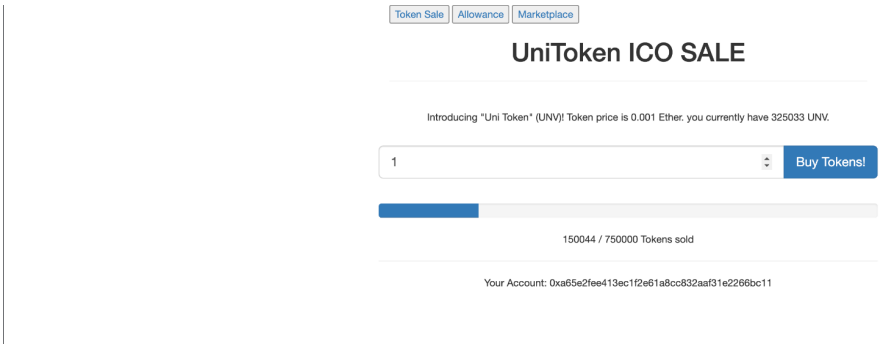


Una vez ingresadas las credenciales, verá lo siguiente.

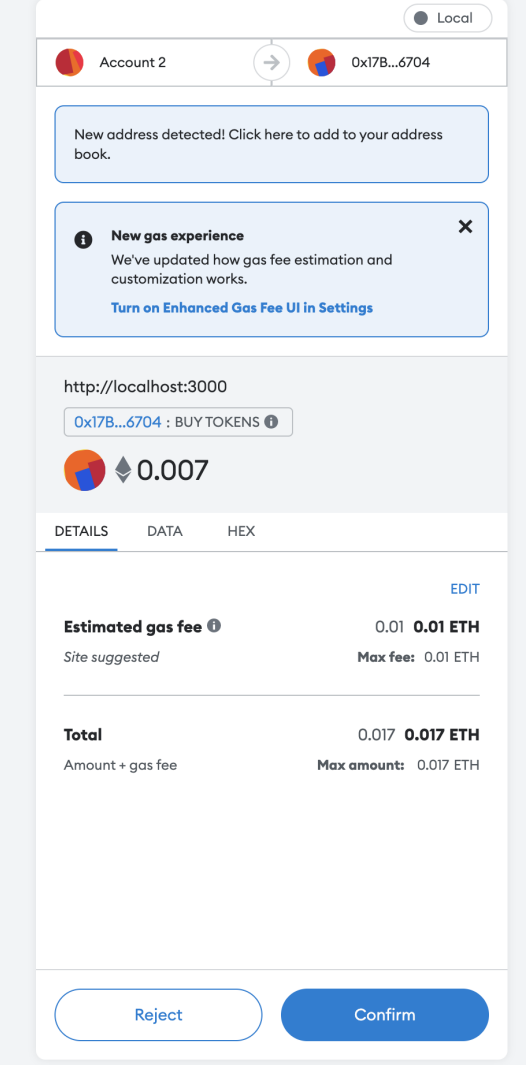


Comprar Tokens

Simplemente se debe seleccionar la cantidad de tokens que se quieren adquirir y clickear en el botón “Buy Tokens”



Metamask le pedirá confirmar la transacción y lo redirigirá a la página de confirmación de compra.



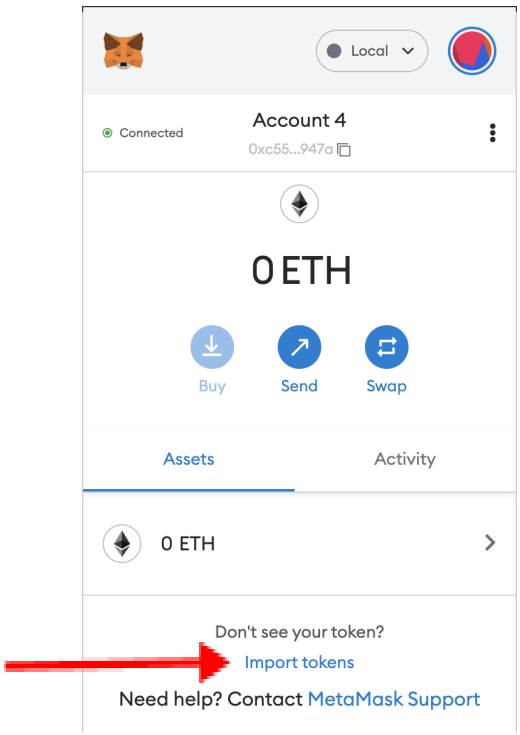
Una vez que se presiona confirmar, se podrá ver el cambio en la wallet, así como en la página ICO, donde figura la cantidad de tokens que han sido vendidas.

Como agregar la token para que aparezca en metamask

Para agregar una token nueva a metamask se debe conseguir 3 atributos de la misma:

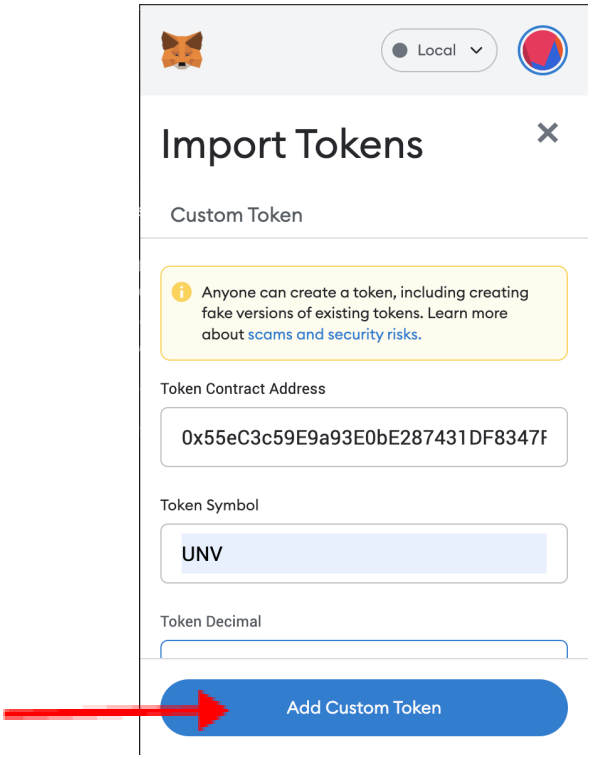
- Address: Suele ser un string de 20 dígitos+0x al principio, se puede consultar en la página principal ICO o contactando a un administrador del sistema. En el ambiente de testeo, figura cuando se ejecuta el comando truffle migrate o truffle migrate –reset
- Símbolo: en el caso de esta token será UNV
- Decimales: Depende del tipo de token, pero en el caso de este, y la mayoría de los token ERC-20, suele ser 18.

Una vez obtenidos estos tres atributos, se procede a abrir la extensión de metamask, donde se podrá ver el siguiente link:

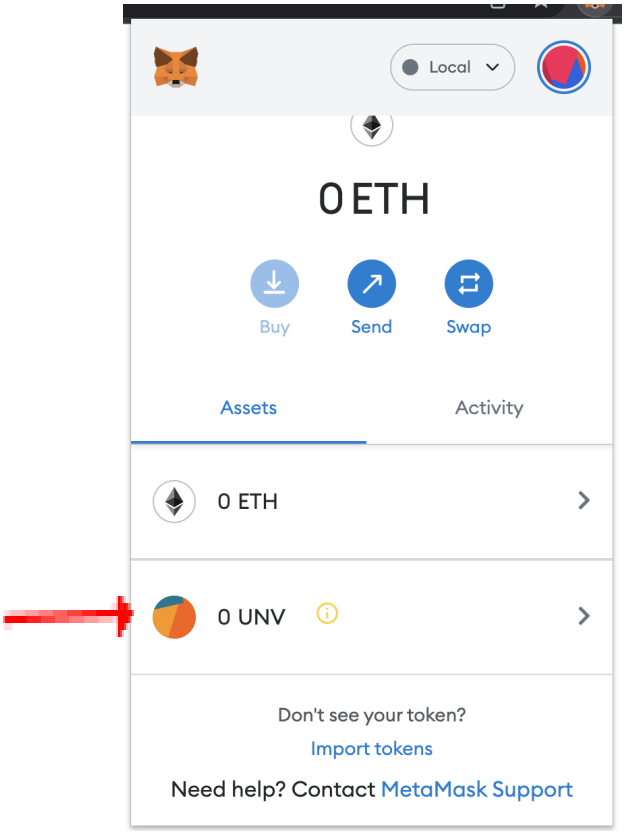


.

Una vez que se clickea ahi, la pantalla cambiará a la siguiente, donde se debe ingresar los atributos mencionados anteriormente y apretar el botón “Add custom Token”

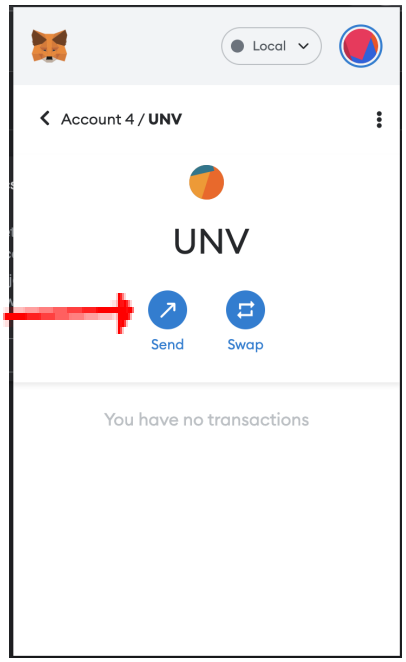


Una vez seguido estos pasos, el token deberá figurar correctamente en Metamask:



Transferir tokens

Se ingresa al token desde metamask una vez importado, luego se debe apretar el boton send:



Luego solo se deben seguir las instrucciones e introducir la dirección de la cuenta a la cual se desea enviar tokens.

Otorgar Allowance:

Se deberá dirigir, dentro de la página ICO del token, a la tab de “Allowance”.

Desde allí, simplemente se debe ingresar la cuenta de la cual se utilizaran los tokens, y la cuenta a la cual se desea otorgar el allowance. Una vez ingresados esos datos, presionar el botón “Otorgar allowance” y la transacción se ejecutará automáticamente.

Como comprar algo con mi nuevo token

Este paso dependerá 100% de cómo implemente el marketplace cada institución, a modo de referencia, una compra funciona similar a una transferencia, por ende, los pasos van a ser muy similares a los mencionados en la parte de “Comprar tokens”, sumado a recibir algo a cambio de esa transferencia de tokens.

3.4 Código fuente

El código fuente desarrollado se encuentra alojado en la plataforma github, la ruta para acceder a este es la siguiente:

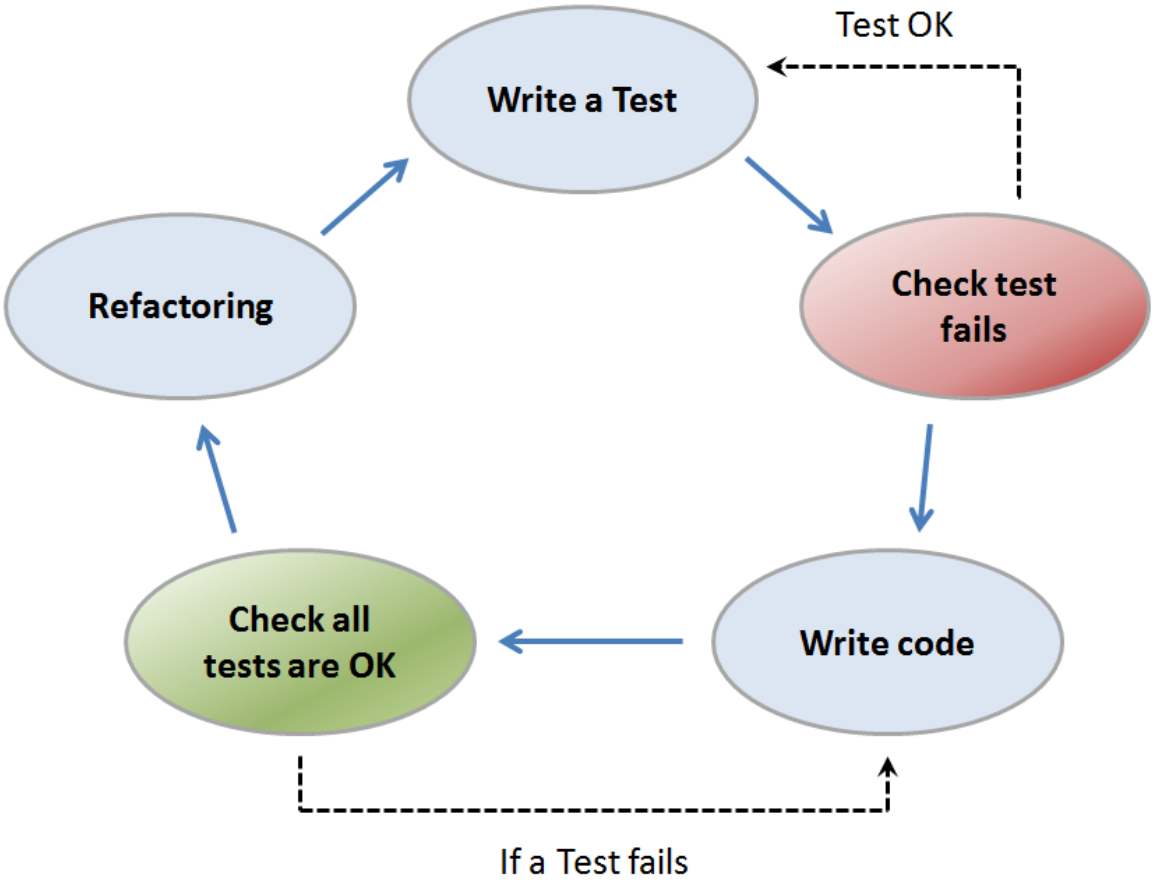
Código fuente

Sino también del siguiente archivo: Código fuente(drive)

.

3.5 Tests

Ya que se desarrolló el Proyecto con una metodología TDD(Test Driven Design), los casos de testeo fueron creados previo a la creación de las funciones del Proyecto. A continuación se puede apreciar el plan de testeo de TDD, así como el código de los tests realizados:



Codigo de Testeo:

Ya que el código es extenso, se adjuntan los links a los archivos en github:
Smart Contract del Token
Contrato de venta del token

3.6 Información Adicional sobre el proyecto

Para analizar la popularidad del producto, y si la inversión es redituable, se realizó una encuesta, mostrada a continuación, la cual fue contestada por 600 personas. Si bien la muestra es baja, se utilizará para analizar la popularidad y viabilidad de inversión de la misma, sin tener en cuenta múltiples factores como viabilidad económica o viabilidad técnica.

Encuesta

- Edad:
- Ocupacion:
- Region/Pais:
- Nivel de estudios (cursando o completo):
    - Universitario
    - Terciario
    - Secundario
    - No finalizó el secundario
- ¿Asiste usted o algún familiar directo a la universidad?:
- Si eligió si, Usted o algún familiar o amigo realizó alguna vez un intercambio educativo?
- ¿Sabe lo que es una criptomoneda?
- ¿Conoce la diferencia entre una criptomoneda y un token digital?
- Si se implementara un token digital, cuyo uso sería principalmente para obtener productos y pagar por servicios en instituciones universitarias de forma universal, lo utilizaría?

- Explique porque.

- Invertiría usted en el token mencionado anteriormente, si este se introdujera al mercado?

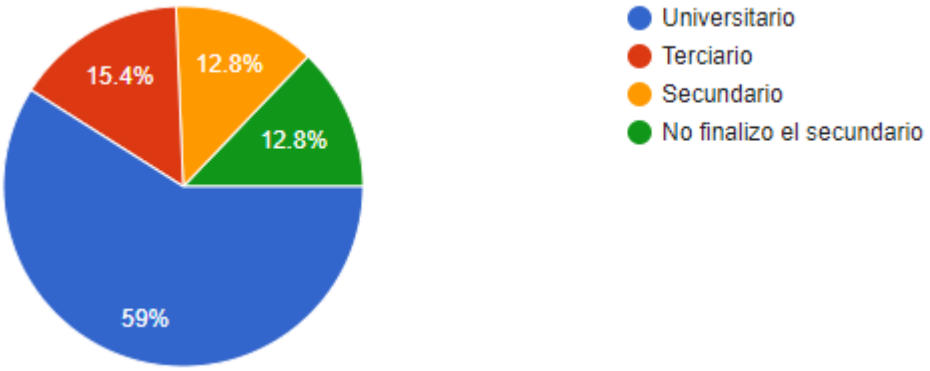
Luego de analizar las respuestas, se obtiene lo siguiente de los campos importantes:

- Edad:
  Una gran parte de la muestra,aproximadamente 70%, consiste en gente entre 20 y 30 años, el público al cual este producto está orientado.

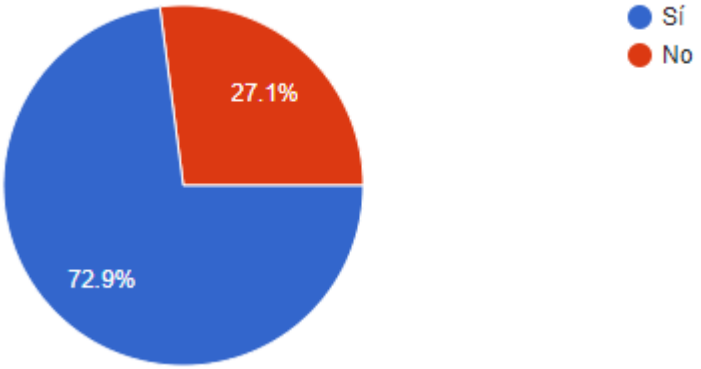
- Ocupacion:
  En ocupación, un 50% de las respuesta fue estudiante, siendo el otro 50% dividido entre múltiples carreras diferentes y un 2,63% de desempleados

- Region/Pais:
  el 63% eligió Argentina, mientras que el otro 37% eligió Estados Unidos como país.

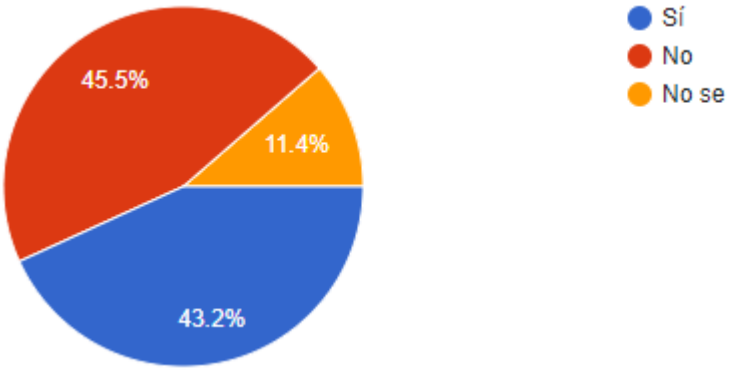
- Nivel de estudios (cursando o completo):



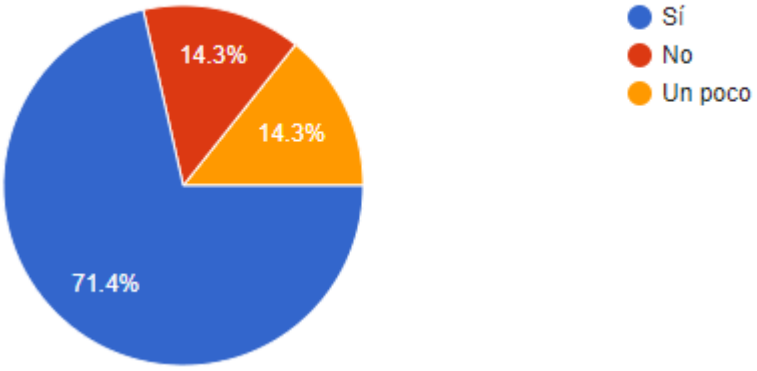
- ¿Asiste usted o algún familiar directo a la universidad?:



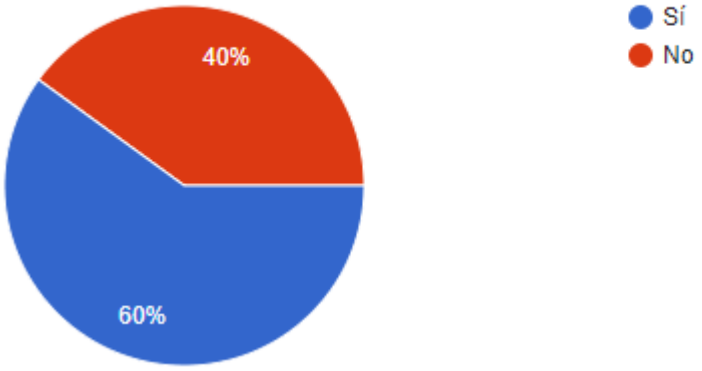
- Si eligió si, Usted o algún familiar o amigo realizó alguna vez un intercambio educativo?



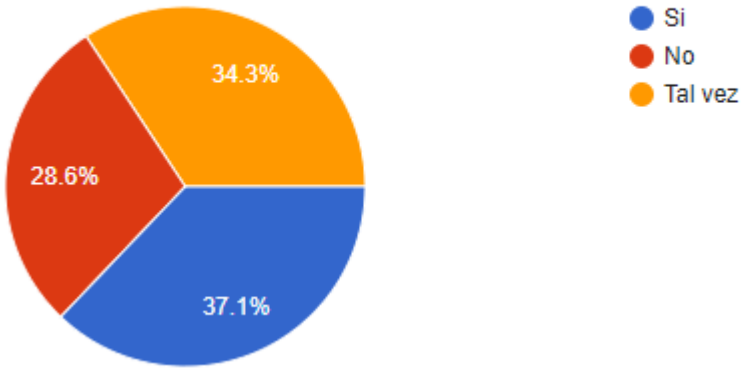
- ¿Sabe lo que es una criptomoneda?



- ¿Conoce la diferencia entre una criptomoneda y un token digital(por ejemplo, un NFT)?



- Si se implementara un token digital, cuyo uso sería principalmente para obtener productos y pagar por servicios en instituciones universitarias de forma universal, lo utilizaría?



- Explique porque

Dado el índole de la pregunta, se generalizaron las respuestas según el tipo de respuesta dada anteriormente.

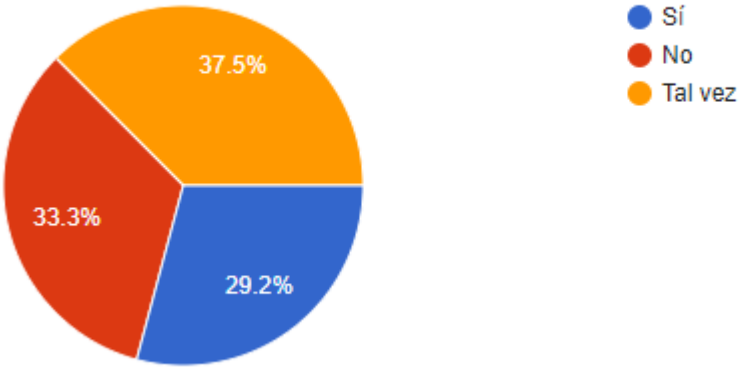
NO: La gente que eligió esta respuesta, en su mayoría no asisten a la universidad o no comprenden el concepto el cual se les presenta. Un pequeño porcentaje también, no creen que crypto sea una opción estable o prefieren la seguridad de la moneda común para realizar pagos.

Tal Vez: La mitad de la gente que optó por esta respuesta,desconfían de las criptomonedas y tokens digitales, habiendo un 30%(aproximadamente) que definieron que necesitan más información sobre el funcionamiento del mismo así como la utilidad que brindaría el mismo para ellos personalmente. El otro 20%(aproximadamente) plantearon que si se ofreciera un incentivo por utilizar esta forma de pago en vez de moneda corriente estarían dispuestos a

utilizarla. Estos incentivos, para citar algunos ejemplos, varían entre un precio descontado por utilizar este medio de pago, beneficios en locales adheridos y otorgamiento de créditos universitarios, los cuales se utilizan en algunas instituciones y no en todas.

SI: el 80%(aproximadamente) de la gente que contestó que sí proviene de estados unidos y asisten a la universidad allí, según sus respuestas, cada institución tiene un tipo de moneda universitaria personal ya implementado y en funcionamiento que cumple un propósito muy similar al que nuestro producto ofrece, con la diferencia que esta moneda mencionada solo funciona en la institución que lo emite, perdiendo valor si el alumno decide cambiar de institución o realizar un intercambio educativo. En general, este gran porcentaje de personas pensaron que la idea de utilizar un token universal en vez de esa moneda era una gran idea, ya que reemplaza el sistema actual, el cual solo funciona en una universidad.
El otro 20%(aproximadamente) plantearon que era una idea innovadora y creativa y demostraron entusiasmo en las respuestas, algunos también indagando sobre si ya esta en el mercado y solicitando más información para poder utilizarla una vez lanzada

- Invertiría usted en el token mencionado anteriormente, si este se introdujera al mercado?



Links de la documentación:

Para finalizar la documentación, todos los archivos de documentación y planificación se pueden encontrar en el siguiente enlace de google drive.

Carpeta de Documentación - Google Drive

4. Conclusiones

Si bien se ha demostrado que es posible la creación e implementación del token, así como se han demostrado la utilidad de sus funcionalidades en un ámbito práctico, lo que aún resta considerar es la viabilidad económica de este proyecto. Habiendo realizado un análisis de mercado no profundo y una encuesta a variedad de personas, el potencial del token es real y se debería poder implementar de forma Rápida y con una propagación media.

Por otro lado, si bien la popularidad aparenta ser buena, no se tienen suficientes muestras para demostrar que esto sea efectivamente cierto y ,aun así, se estima que se deberá invertir una gran cantidad de dinero en hacer funcionar este sistema en un ambiente real.

Basado en la información mencionada anteriormente, mi opinión personal es que el token tiene potencial de uso, pero tendrá que ser respaldado ya sea por una institución educativa que quiera estar al frente de la revolución tecnológica o por un agente independiente,que posea tanto el capital como la iniciativa de empujar este proyecto hacia adelante. Otra buena alternativa sería proponerlo a varias instituciones educativas de Estados Unidos, las cuales podrían realizar un proyecto en conjunto para lanzar dicho token al mercado, debido al buen impacto que tuvo la encuesta en ese mercado.

5 Referencias

[1]"Ethereum Developer Resources | ethereum.org", *ethereum.org*, 2021. [Online]. Available: https://ethereum.org/en/developers/. [Accessed: 28- Feb- 2021]

[2]"Ethereum Developer Documentation| ethereum.org", *ethereum.org*, 2021. [Online]. Available: https://ethereum.org/en/developers/docs. [Accessed: 28- Feb- 2021]

[3]"Ethereum Developer Tutorials| ethereum.org", *ethereum.org*, 2021. [Online]. Available: https://ethereum.org/en/developers/tutorials. [Accessed: 28- Feb- 2021]

[4]"Ethereum Developer Learning Tools| ethereum.org", *ethereum.org*, 2021. [Online]. Available: https://ethereum.org/en/developers/learning-tools. [Accessed: 28- Feb- 2021]

[5]"Ethereum Developer Local Environment| ethereum.org", *ethereum.org*, 2021. [Online]. Available: https://ethereum.org/en/developers/local-environment. [Accessed: 28- Feb- 2021]

[6]"Decentralized applications (dapps) | ethereum.org", *ethereum.org*, 2021. [Online]. Available: https://ethereum.org/en/dapps/. [Accessed: 28- Feb- 2021]

[7] *Solidity Programming Language*. [Online]. Available: soliditylang.org/. [Accessed: 28-Feb.-2021].

[8] L. Willems, “How to set up a private Ethereum testnet blockchain using Geth and Homebrew”, *Coinmonks*. 5-Aug.-2017. [Online]. Available: medium.com/coinmonks/how-to-set-up-a-private-ethereum-testnet-blockchain-using-geth-and-homebrew-1106a27e8e1e. [Accessed: 28-Feb.-2021].

[9] Antonopoulos, A. M., & Wood, G. (2018). *Mastering Ethereum: Building smart contracts and DApps*. O'Reilly Media.

[10] “List of Decentralized Exchanges - Best DEX Decentralized exchanges”, *Best DEX Decentralized exchanges* . [Online]. Available: defiprime.com/exchanges. [Accessed: 10-Apr.-2021].

[11]"How to Create Your Own ERC-20 Token in 10 Minutes Moralis The Ultimate Web3 Development Platform", Moralis The Ultimate Web3 Development Platform. 19-Aug.-2021.[Online].Available:moralis.io/how-to-create-your-own-erc-20-token-in-10-minutes/. [Accessed: 23-May-2022].

[12]"Smart contract languages", Ethereum.Org. 20-May-2022. [Online]. Available: ethereum.org/en/developers/docs/smart-contracts/languages/. [Accessed: 23-May-2022].

[13]"Anatomy of smart contracts", Ethereum.Org. 20-May-2022. [Online]. Available: ethereum.org/en/developers/docs/smart-contracts/anatomy/. [Accessed: 23-May-2022].

[14]"Knight Cash", UCF Card Services. [Online]. Available: ucfcard.ucf.edu/knight-cash/. [Accessed: 23-May-2022].

[15] "Smart Contracts Cryptocurrencies |", CryptoSlate. [Online]. Available: cryptoslate.com/cryptos/smart-contracts/. [Accessed: 27-Apr.-2021].

[16] "What is a smart contract? Cardano Documentation 1.0.0 documentation", [Online]. Available: docs.cardano.org/en/latest/explainers/cardano-explainers/smart-contract-exp.html. [Accessed: 27-Apr.-2021].

[17] "Smart Contracts Polkadot Wiki", 27-Apr.-2021. [Online]. Available: wiki.polkadot.network/docs/en/build-smart-contracts. [Accessed: 27-Apr.-2021].

[18] "vechain/ThorNode-contracts", Non-Fungible Token Standard.. 27-Mar.-2020. [Online]. Available: github.com/vechain/ThorNode-contracts. [Accessed: 27-Apr.-2021].

[19] "Stellar Smart Contracts", Stellar Developers. [Online]. Available: www.stellar.org/developers/guides/walkthroughs/stellar-smart-contracts.html. [Accessed: 27-Apr.-2021].

[20] J. Iadeluca, "An Introduction to TRON Smart Contract Development", BTCMANAGER. 20-Jan.-2019. [Online]. Available: btcmanager.com/introduction-tron-smart-contract-development/. [Accessed: 27-Apr.-2021].

[21] "Ethereum Virtual Machine (EVM)", Ethereum.Org. 27-Apr.-2021. [Online]. Available: ethereum.org/en/developers/docs/evm/. [Accessed: 27-Apr.-2021].

[22] G. Scott, “Exchanges Offer a Platform for Trading”, 31-July-2020. [Online]. Available: www.investopedia.com/terms/e/exchange.asp. [Accessed: 27-June-2021].

[23]“Token Swap”, CoinMarketCap. [Online]. Available: coinmarketcap.com/alexandria/glossary/token-swap. [Accessed: 27-June-2021].

[24]“Qué son las Liquidity Pools (Reservas de Liquidez) en DeFi y Cómo Funcionan?”, Binance Academy. 14-Dec.-2020. [Online]. Available: academy.binance.com/es/articles/what-are-liquidity-pools-in-defi. [Accessed: 27-June-2021].

[25]S. Anderson, “What Is Ethereum?”, 27-June-2020. [Online]. Available: www.investopedia.com/terms/e/ethereum.asp. [Accessed: 20-July-2021].

[26]A. Hertig, “What Is DeFi?”, CoinDesk. 18-Sept.-2020. [Online]. Available: www.coindesk.com/what-is-defi. [Accessed: 21-July-2021].

[27]“Decentralized finance (DeFi) | ethereum.org”, Ethereum.Org. 15-July-2021. [Online]. Available: ethereum.org/en/defi/. [Accessed: 21-July-2021].

[28]“Transactions” | Ethereum.Org. 15-July-2021. [Online]. Available: ethereum.org/en/developers/docs/transactions/. [Accessed: 25-July-2021].

[29]“Cules son los tres tipos de tokens y cmo se diferencian de las criptomonedas”, Infobae. 11-Aug.-2018.[Online].Available:www.infobae.com/cripto247/educacion-cripto247/2018/08/10/cuales-son-los-tres-tipos-de-tokens-y-como-se-diferencian-de-las-criptomonedas/. [Accessed: 23-May-2022].