

BLOCKCHAIN DEVELOPMENT en UTEC

Ponente

Senior Smart Contract Developer en [dcSpark.io](#), el primer side chain de Cardano. Cofundador y Líder en Blockchain del juego [Pachacuy.io](#) (El Axie Infinity de South America). Desarrollador Blockchain en [CuyToken.com](#), la primera empresa de criptocréditos del Perú. 5+ años de experiencia en compañías Fintech (e.g. Yodlee, Tenpo, FIDIS y CuyToken). Estudió Computer Programming Certificate en la Universidad de Santa Clara, California, USA. Graduado del bootcamp del programa inmersivo de Ingeniería de Software Hack Reactor, Los Ángeles. Graduado de Lean UX and Service Design Program (UTEC) y del diplomado de Finanzas Corporativas (UPC).

Objetivo

Explicar y transmitir las herramientas que usan en el día a día los desarrolladores blockchain. Comunicar las buenas prácticas de la industria. Exponer casos de negocio y aplicaciones reales que se han creado dentro del Blockchain.

Modalidad de Trabajo

1. El desarrollo de las clases de programación está basado en casos de la vida real
2. Las herramientas de blockchain que se usarán en el desarrollo del código son consideradas estándar y buenas prácticas
3. Se planteará el desarrollo de una startup blockchain ficticia que será el punto de partida para el desarrollo de los Smart Contracts

Proyectos a desarrollar

1. Creación de una criptomoneda (ERC20)
2. Creación de un contrato de compra/venta de tokens
3. Desarrollo de un contrato Airdrop (lista blanca y merkle tree)
4. Desarrollo de una colección de 10,000 NFTs
5. Desarrollo de una librería NPM agnóstica (para front y back) que permita interactuar con los contratos inteligentes
6. Front end minimalista para interactuar con los smart contracts usando la librería NPM

Proyectos desarrollados

A continuación es una lista de proyectos en los que trabajo de principio a fin. Servirá de inspiración para los proyectos finales.

Cuy Token

La primera

CRİPTOMONEDA PERUANA

que respalda su valor en proyectos blockchain.

[Comprar CuyToken](#)

Enero 2022



- Sinopsis: Vender el token para poder otorgar préstamos a diferentes tipos de proyectos con potencial. El primer proyecto financiado fue Pachacuy que logró recaudar 30,000 USD por su propia cuenta.
- Criptomoneda creada usando el estándar ERC20
- Publicado en Binance
- Lanzado en Lima, Perú
- Recaudación 8,000 USD en menos de dos horas
- [Código del token](#)
- [Testing](#)

Crypto Index (FIDIS)

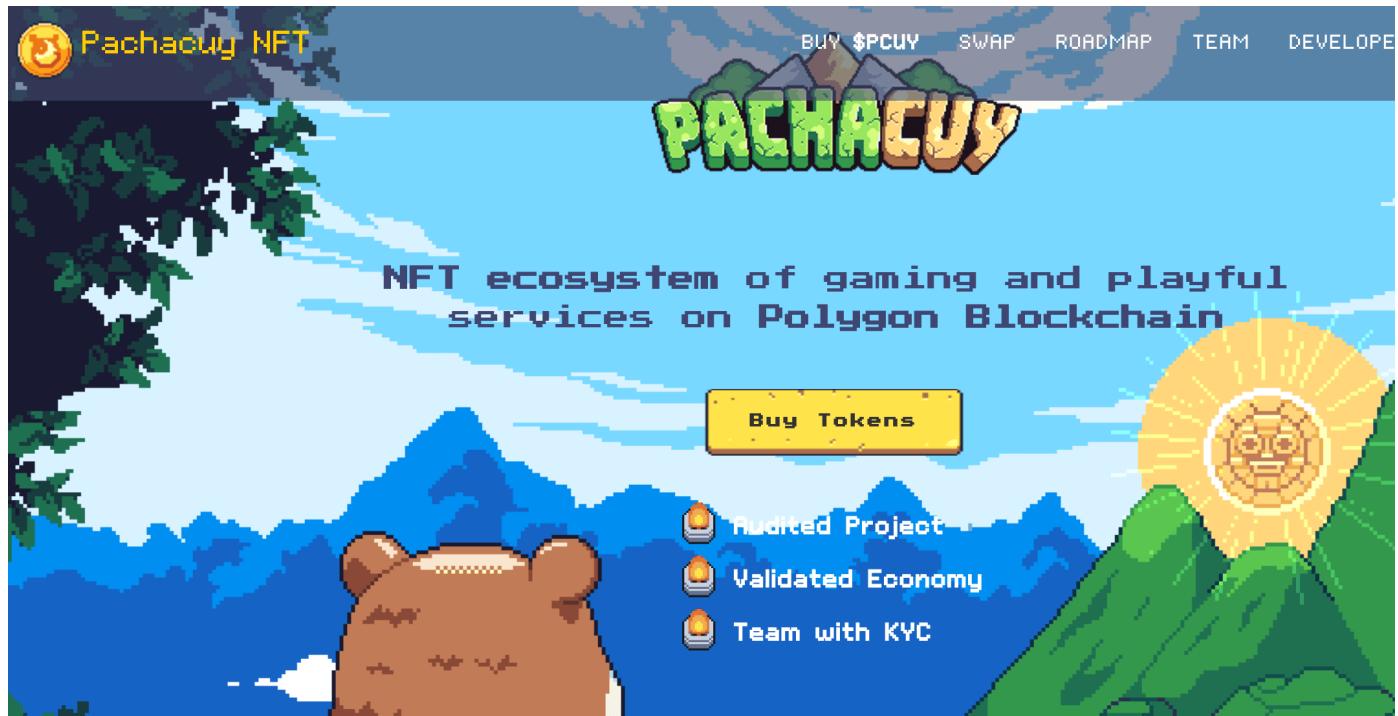
FIDIS FI25 Crypto Index

A Tokenized Cryptocurrency Index Tracking the Top Twenty-Five Cryptocurrencies via a Smart Contract Deployed on Optimistic Ethereum

December 13, 2021

- Sinopsis: Comprar una sola criptomoneda que represente a las veinticinco criptomonedas más importantes (tipo S&P 500). Esta lista variará dependiendo de la importancia de cada token.
- Índice de las 25 primera criptomonedas
- Se publicó en Optimism (Layer 2, costos de transacción son exponencialmente diminutos en comparación a Ethereum)
- Multifirma - La ejecución de métodos tiene que ser aprobado por varios administradores antes de ser aprobado
- [Código del token](#)
- [Unit Testing](#)
- [Whitepaper](#)

Pachacuy (Axie Infinity de America del Sur)



- Sinopsis: Juego que representa un mundo virtual en el cual un cuy (personaje principal - NFT) puede comprar tierras y establecer negocios dentro. Los cuyes visitantes hacen uso de los negocios. Estas transacciones tienen el potencial de generar ingresos para dueños y clientes.
- Hay creado su propia moneda llamada Pachacuy
- Toda la lógica del juego está desarrollado en Smart Contracts
- Publicado en Polygon
- Familia (17) de Smart Contracts interconectados
- Implementa diferentes estándares de tokens (ERC777, ERC1155)
- [Código del juego](#)
- [Testing](#)
- [White Paper](#)
- [Juego en producción](#)

Colección Moche



Pachacuy Moche ✅
Breakable Piece EXPLORER... 0.38 BNB ⚡



Pachacuy Moche ✅
Unfit Cousin EXPLORER... 0.11 BNB ⚡



Pachacuy Moche ✅
Abandoned Juice EXPLORER... 0.7 BNB ⚡



Pachacuy Moche ✅
Black-and-white Parkie... 0.2 BNB ⚡



Pachacuy Moche ✅
Superficial Delivery E... 0.45 BNB ⚡



Pachacuy Moche ✅
Alert Pleasure EXPLORER... 2 BNB ⚡



Pachacuy Moche ✅
Sarcastic Stranger EXPLO... 1.3 BNB ⚡



Pachacuy Moche ✅
Lighthearted Variety EX... 3 BNB ⚡



Pachacuy Moche ✅
Impartial Establish... 0.111 BNB ⚡



Pachacuy Moche ✅
Made-up Quote EXPLOR... 2.5 BNB ⚡

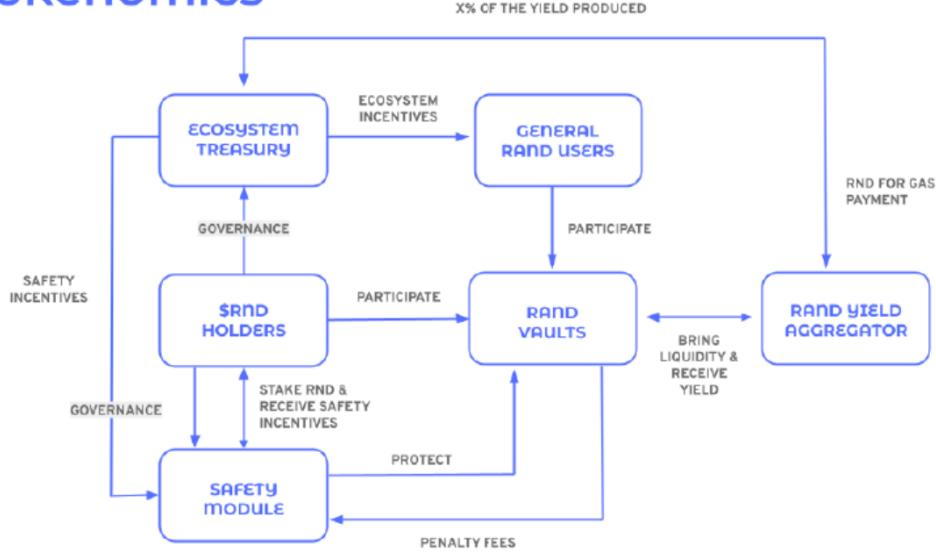
1. Sinopsis: colección de 10,000 NFTs lanzado en la red Binance. El personaje principal es un cuy y fue creado con un algoritmo de generación de imágenes por capas.
2. Se utilizó un smart contract que sigue el estándar ERC721
3. [Página de compra](#)
4. [Colección en vitrina de Tofu](#)

Rand Network

The Next Evolution Of Banking

Home About Community Get the App

Tokenomics



1. Sinopsis: Los usuarios depositan USDC (una moneda estable) en Ethereum. Al juntarse, se invierte en diferentes protocolos de Finanzas Descentralizadas (DeFi) que generan intereses y/o recompensas sobre lo depositado. Luego de un tiempo, se retiran los intereses generados para ser repartidos entre los usuarios iniciales
2. Se utilizan dos diferentes blockchain Ethereum y Moonbeam. En la red Ethereum se manejan los fondos e inversiones. En Polygon se realizan las transacciones y procesamientos pesados. La razón de la separación es el costo de transacción.
3. Utiliza contratos actualizables que le permiten arreglar bugs futuros o crear nuevas estructuras de datos internas para albergar información relevante adicional.

dcSpark



Products

Investors

Founders

Careers

Grants

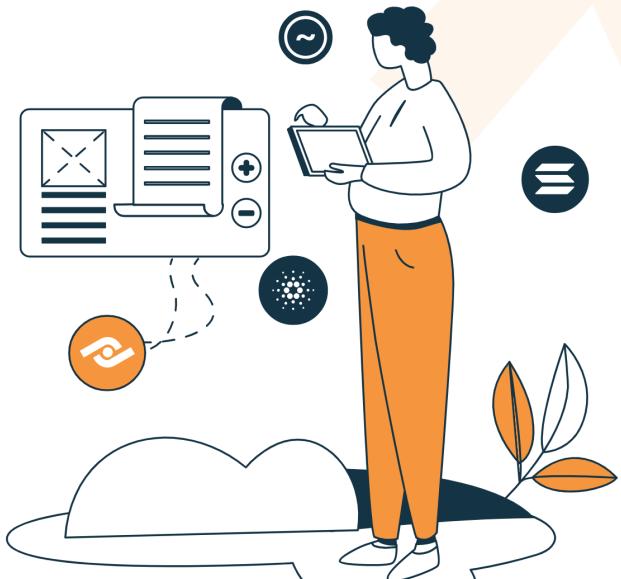
English ▾

Contact us

We build high-quality crypto products that unify the user experience within blockchain ecosystems.

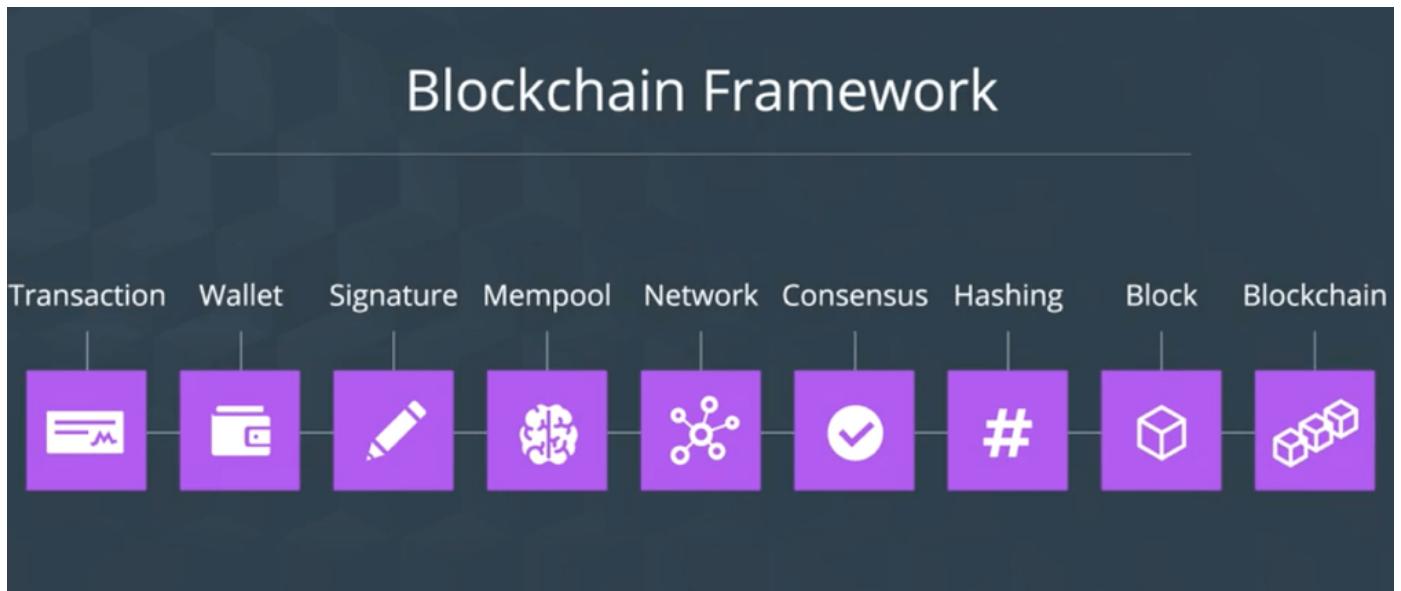
By focusing on interoperability and composability on the backend, our products provide a seamless experience for users allowing them to take their first step into the ever-growing world of Blockchain.

Learn more



1. Sinopsis: Crear compatibilidad para blockchains que no son compatible con la Maquina Virtual de Ethereum. Lo hace a través de la creación de Sidechains o la implementación de Layer 2.
2. En la actualidad, mi trabajo es crear un contrato de Staking. Es decir, un contrato que otorga beneficios por realizar depósitos, muy parecido a un depósito a plazo fijo.

Blockchain Framework:



1. Transacción: unidad fundamental dentro de un blockchain. Cualquier operación llevada a cabo se atomiza en una transacción que es enviada por un usuario para ser incluida en el siguiente bloque.
2. Wallet: Es como una cuenta de banco. Se usa para ejecutar transacciones. Así mismo, una wallet puede llevar la cuenta de los activos de una dirección (address). La wallet en sí misma no almacena los activos, solo muestra los balances.
3. Signature: Una firma digital por el usuario es necesaria firmar una transacción antes de que sea incluida en la network,
4. Mempool: Luego de que una transacción es firmada, se incluye en la Mempool. Este es el lugar donde todas las transacciones esperan por un validador para que pueda incluirlo en el bloque.
5. Network: la naturaleza de la red de nodos que mantiene en pie al Blockchain es distribuida. Cualquiera está en la posibilidad de obtener una copia desde la primera hasta la última transacción del Blockchain (no centralización de la información). Bajo este modelo, la red es capaz de determinar qué transacciones son válidas.
6. Consensus: Es una manera de crear un mecanismo de votación entre los nodos. PoS, PoW.
7. Hashing: es el proceso de generar una huella digital única. Se utilizan funciones que hacen Hash cuyo input es la data. Un cambio infinitesimal en la data y el hash obtenido es completamente diferente. Ello invalidaría al bloque.
8. Block: Es un contenedor de todas las transacciones que se añadirán al blockchain. Estos bloques están linkeados unos con otros mediante valores de hash.
9. Blockchain: Es un libro público en el cual los bloques están linkeados, lo cual nos permite ver si las transacciones son validas o no.

Ethereum Virtual Machine

Ambiente virtual

EVM significa Máquina Virtual de Ethereum. En simple, EVM es el sistema operativo de Ethereum. Dentro de esto, una máquina virtual puede proporcionar un entorno de ejecución para ejecutar contratos inteligentes.

Por lo general, una vez que se compila un contrato inteligente, genera dos salidas: Bytecode y ABI. El primero se carga en el EVM para el cálculo. El segundo es más legible por humanos. El código de bytes se distribuye a cada nodo que se ejecuta dentro de la red. El código de bytes se ejecuta y genera un "cambio de estado", que solo podría lograrse mediante el consenso de cada nodo. Eso convierte a la EVM en una "máquina de estado distribuida": rastrea el estado del Blockchain en cada transacción.

Existen diferentes lenguajes de programación que pueden ser entendidos por la EVM (Solidity, Vyper, etc.).

Computadora Mundial

La máquina virtual de Ethereum funciona como una sola entidad mantenida por miles de computadoras interconectadas llamadas nodos, que también se conoce como la computadora mundial. Estas computadoras ejecutan una implementación del cliente Ethereum y tienen una estructura de igual a igual (Peer to Peer - P2P). Su trabajo principal es procesar y validar transacciones, así como asegurar y estabilizar todo el ecosistema. Por eso, el EVM podría verse como un motor de procesamiento y una plataforma de software que utiliza computación descentralizada.

Estado de la cadena de bloques

Dentro de la EVM se definen las reglas para crear un nuevo estado válido de bloque a bloque. Una vez que se ejecutan los contratos inteligentes, el EVM calcula el nuevo estado de la red después de agregar un nuevo bloque a la cadena. En cualquier momento dado, la EVM tiene un y solo un estado 'canónico'. Es en este entorno que viven las cuentas de Ethereum y los contratos inteligentes. El protocolo Ethereum tiene como objetivo mantener esta máquina especial realizando operaciones ininterrumpidas.

En otras palabras, el objetivo de EVM es averiguar el estado general de Ethereum para cada bloque en el Blockchain. Utiliza un libro mayor (public ledger) distribuido donde se rastrean las transacciones y, al mismo tiempo, impone reglas a los usuarios sobre cómo interactuar con la red.

Capa

Se encuentra en la parte superior de la capa de red de nodos y hardware de Ethereum.

Turing completo

Puede realizar pasos lógicos para la función computacional. Es capaz de hacer cualquier cálculo o programa informático posible. Detrás de esta característica se encuentran los OPCODES que son como una lista operaciones aisladas que arman como piezas de lego.

Gas

A cada instrucción de EVM se le asigna un costo. Eso ayuda a mantener un recuento de los costos totales de cualquier transacción determinada. Las unidades de gas miden el costo de ejecutar operaciones en EVM. Para calcular el total de gas a gastar se cuentan el total de OPCODES a usar dado que cada uno de ellos tiene un costo específico. Cualquier transacción empieza en 21000 gas.

OPCODES (códigos de operación)

El EVM es capaz de ejecutar instrucciones a nivel de máquina conocidas como OPCODES (códigos de operación). Estos códigos de operación se utilizan para definir cualquier operación particular dentro del EVM. Hay códigos de operación especiales para operaciones aritméticas, así como para leer desde el almacenamiento. Cada código de operación es un byte. Se puede utilizar un máximo de 256 códigos de operación. [Ver lista completa](#).

- **Stack-manipulating opcodes** (*POP, PUSH, DUP, SWAP*)
- **Arithmetic/comparison/bitwise opcodes** (*ADD, SUB, GT, LT, AND, OR*)
- **Environmental opcodes** (*CALLER, CALLVALUE, NUMBER*)
- **Memory-manipulating opcodes** (*MLOAD, MSTORE, MSTORE8, MSIZE*)
- **Storage-manipulating opcodes** (*SLOAD, SSTORE*)
- **Program counter related opcodes** (*JUMP, JUMPI, PC, JUMPDEST*)
- **Halting opcodes** (*STOP, RETURN, REVERT, INVALID, SELFDESTRUCT*)

Contratos inteligentes

Los contratos inteligentes son líneas de código utilizadas por diferentes dos o más partes para realizar transacciones entre sí. Dado que los contratos inteligentes se cargan y ejecutan en el EVM, no se necesita un tercero fiscalizador. Un contrato inteligente es una lista de operaciones que se ejecutarán cuando se cumplan ciertas condiciones. Estas operaciones pueden ser muy diferentes (por ejemplo, creación de tokens, transferencia de fondos) y estarán guiadas por código y ejecutados por máquina.

Bytecode



Another example Solidity contract, along with its bytecode needed to deploy

El Smart Contract se compila a bytecode y ABI. El bytecode se puede traducir en OPCODES.

[Ver ejemplo de NFT del juego Pachacuy](#)

ABI (application binary interface)

```

pragma solidity 0.5.3;

contract ExampleContract {
    function HelloWorld() public returns(string memory) {
        return "Hello World!";
    }
}

```



```

> keccak256("HelloWorld()");
"7ffffb7bdf7d16635144da549e9a4eedff43ed43d64e49e18d7e365f9e5521232"
Function signature

```

```

{
    "constant": false,
    "inputs": [],
    "name": "HelloWorld",
    "outputs": [
        {
            "name": "",
            "type": "string"
        }
    ],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}

```

Example Solidity contract, along with its ABI

Usado por el front para poder instanciar el objeto 'Contrato' de las librerías como Ethers.js. Es una interface en el cual se definen qué parámetros serán pasados, qué valores ser retornarán, nombres de los métodos y otras características de los métodos y propiedades del smart contract.

Stack (tooling) de desarrollo web 3

Las herramientas de un blockchain developer son variadas e incluyen herramientas de testing, auditoría, computación en la nube y demás. Estas son las herramientas que uso en mi día a día como desarrollador blockchain:

- Hardhat
- Ethers.js
- Metamask
- Gnosis safe
- Open Zeppelin (standards)
- Open Zeppelin Defender
- Remix
- Mythril (by ConsenSys)
- Alchemy/Infura/Moralis
- Etherscan
- Solidity
- Faucets
- MythX

Hardhat

Flexible. Extensible. Fast.

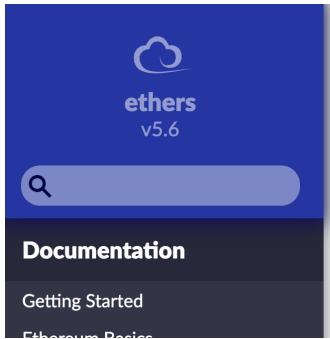
Ethereum development environment for professionals



<https://hardhat.org>

1. Es un ambiente de desarrollo profesional para Ethereum.
2. Te permite publicar contratos en diferentes blockchain (Polygon, Ethereum, Binance, Mumbai, etc.) con simples configuraciones, así como también publicar en un blockchain local para poder verificar que el script de deployment es válido.
3. Provee las bases para poder realizar tests complejos de Smart Contracts de manera automatizada.
4. Incluye un Smart Contract que te permite ver los logs en Solidity. Es decir, se puede usar 'console.log' dentro del código para analizar ciertos outputs.
5. Es posible programar la verificación de Smart Contracts en el mismo script de deployment, lo cual evita hacerlo manualmente en Etherscan.
6. Dentro de un ambiente de testing, te permite hacer un fork del blockchain Ethereum para interactuar directamente con Smart Contracts publicados en dicha red. Esto es relevante porque algunos Smart Contracts no están en Testnet y no hay otra manera de probarlos.
7. Permite configurar diferentes tipos de versión de compiladores de Solidity, así como también especificar la precisión de la optimización (runs) de los Smart Contracts.

Ethers.js



The screenshot shows the ethers.js documentation website. The header features the ethers logo (a stylized cloud icon) and the text "ethers v5.6". Below the header is a search bar with a magnifying glass icon. The main navigation menu includes "Documentation", "Getting Started", and "Ethereum Basics".

Documentation

What is Ethers?

The ethers.js library aims to be a complete and compact library for interacting with the Ethereum Blockchain and its ecosystem. It was originally designed for use with [ethers.io](#) and has since expanded into a more general-purpose library.

1. Librería compacta y completa que te permite interactuar con diferentes blockchain de manera programática. Es decir, puedes leer información del Blockchain, así como también acceder a métodos y propiedades de Smart Contracts publicados en el Blockchain.
2. Puede ser usado tanto el front-end como en el back-end para crear tareas o procesos automatizados que involucren interactuar con Smart Contracts publicados en el Blockchain.
3. Si es usado desde el front, por lo general se usa con Metamask (u otra billetera que funciona en el)

navegador). A través de Metamask, las operaciones definidas con Ether.js serán firmadas con la billetera de Metamask (donde se alberga la llave privada del usuario)

- Para ser usado desde el back, se requiere tener la llave privada alojada en un archivo .env e instanciar el objeto 'Contract' en el código del backend.

Metamask

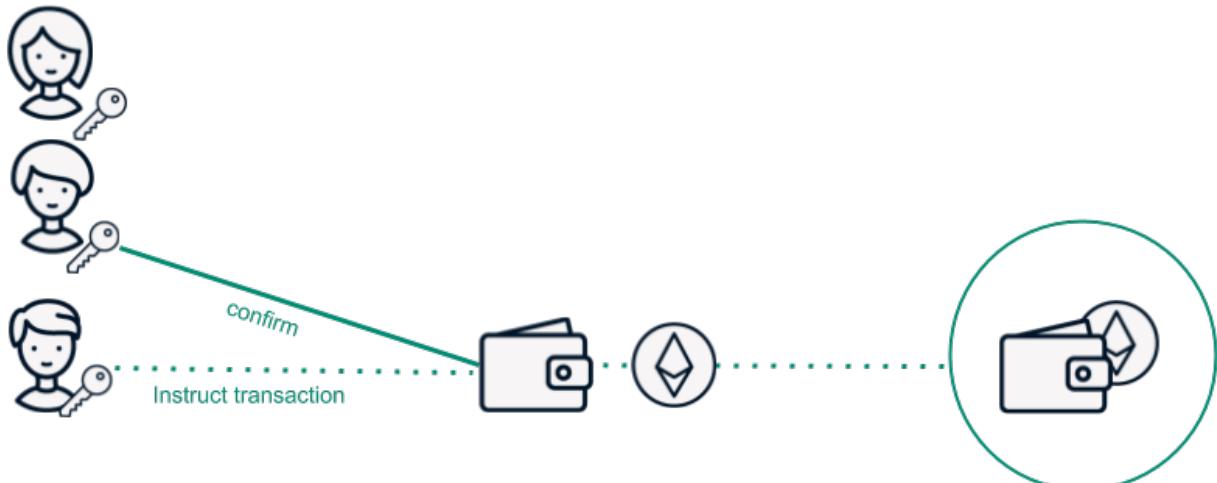
The screenshot shows the official Metamask website. At the top, there's a navigation bar with a fox logo, the word "METAMASK", and links for "Features", "Support", "About", "Build", "Download", and a toggle switch. Below the navigation is a large banner with the text "A crypto wallet & gateway to blockchain apps". It features a colorful play button graphic and a screenshot of the Metamask extension interface showing ETH and DAI balances. A subtext below the banner reads "Start exploring blockchain applications in seconds. Trusted by over 30 million users worldwide." and a "Download for Google Chrome" button.

- Es una billetera de criptomonedas que funciona como extensión del browser o aplicación de celular.
- Te permite interactuar (conectar y autenticarte) con aplicaciones descentralizadas con previa confirmación del usuario antes de firmar cada transacción.
- Con Metamask, se pueden crear llaves privadas y públicas a demanda.
- Puedes agregar diferentes criptomonedas para visualizar el balance de las mismas, así como también realizar transferencias a diferentes cuentas (addresses) con una simple interface.

Gnosis safe

The screenshot shows the Gnosis Safe website. At the top, there's a navigation bar with a safe icon, the word "Safe", and links for "Overview", "Security", "Enterprises", "Developers", "Community", "Help", "Careers", and "Open app". Below the navigation is a large callout box containing the number "107,176,958,383". To the left of the callout is the text "A new standard for smart contract security". At the bottom left, there's a section titled "How we make sure your funds are safe". At the bottom right, there's a "Statistics" button and some smaller text about USD worth of digital assets stored.

1. Plataforma para manejar activos (criptomonedas) dentro del Blockchain
2. Se pueden crear diferentes vaúles seguros (Safe) que tienen una dirección (address) propia para poder recibir fondos
3. Estos vaúles pueden realizar transferencias con la aprobación de diferentes personas (multisig). Cada persona involucrada debe firmar de manera separada para poder aprobar una transacción. Firmar esta transacción no incurre en ningún costo.



1 2 out 3 confirmations by wallet's owners are needed for a transaction.

2 Transaction is pending as the other owners need to confirm

3 Approved transaction will be executed

OpenZeppelin



We're hiring

Products ▾

Security Audits

Learn ▾

Company ▾

News & Events

The standard for secure blockchain applications

OpenZeppelin provides security products to build, automate, and operate decentralized applications. We also protect leading organizations by performing security audits on their systems and products.

1. Provee Smart Contracts base que sirven como los fundamentos para crear otras aplicaciones más complejas.
2. En esencia, provee las implementaciones de los estándares más usados en el desarrollo de Smart Contracts (ERC20, ERC721, ERC1155, ERC777, etc.)
3. Estos Smart Contracts base siguen buenas prácticas y han sido auditados y testeados innumerables veces
4. OZ es aceptado y usado en la industria del Blockchain como un estándar de desarrollo

- Sigue un desarrollo simple, modular y robusto. Como piezas de lego, se pueden unir y separar según la conveniencia

OpenZeppelin Defender

The screenshot shows the OpenZeppelin Defender web interface. At the top, there's a navigation bar with a logo, 'Defender', 'Docs', and a 'Sign up for free' button. The main heading is 'Ship faster with the security of OpenZeppelin Defender'. Below it, a subtext says 'Automate smart contract operations to deliver high-quality products with lower risk – now available with an expanded set of features.' A blue 'SIGN UP FOR FREE' button is visible. On the right, there are two panels: one for 'Pending resolution' (with 3 approvals needed) and another for 'AccountCleanup Run' (status: SUCCESS).

Create new relay

Relay name: Oracle

Ethereum network: Mainnet

Pending resolution

At least 3 approvals are needed to submit the upgrade request.

Approved 1/3: 0xb6a1...7985...

Pending 2/3: 0xa1b6...0579...

Amanda Rains

Approve **Reject**

AccountCleanup Run

Date: 10/08/2020 - 19:00 UTC Trigger: Scheduled Status: **Success**

LOG DETAILS

RESULT: **Run**

START
RequestID: 849db76a-4f03-4b0d-8e1b-7c541e0e4034
Version: SLATEST

END
RequestID: 849db76a-4f03-4b0d-8e1b-7c541e0e4034

REPORT
RequestID: 849db76a-4f03-4b0d-8e1b-7c541e0e4034

- Son cuatro los principales servicios que ofrece Defender: Admin, Relay, Autotask y Sentinel.
- Admin: Te permite ejecutar funciones de algún Smart Contract usando credenciales multifirma. Es decir, dos o más cuentas podran firmar una transacción.
- Relayer: son intermediarios que pueden tener privilegios para ejecutar smart contracts. Un intermediario consta de una llave privada y dirección (address). Se pueden crear credenciales compartidas por múltiples desarrolladores para usar el mismo intermediario. Se puede automatizar desde el front-end con una librería de NPM.
- Autotask: desarrolla scripts usando NodeJS que permite ejecutar ciertos métodos en un Smart Contract. Estos scripts se ejecutan en un servidor privado (serverless) a demanda. Se puede llamar a un autotask mediante un web hook (url).
- Sentinel: Es capaz de escuchar eventos y ejecuciones de funciones provenientes de smart contracts y actuar en consecuencia. Es posible concatenar un sentinel con un autotask. El sentinel puede disparar mensajes de correo electrónico o Slack.

Remix IDE

The screenshot shows the Remix IDE interface. On the left is the File Explorer with a workspace named "default_workspace" containing files like contracts, scripts, tests, artifacts, .deps, ipfs, README.txt, myMessage.sol, and Modifiers.sol. The main area features the title "Remix IDE" and a "Scam Alerts" section with links to learn more about URL risks and liquidity front runner bots. It also includes a cartoon character of a blue dragon-like creature holding a sword. Below this are sections for "Featured Plugins" (Solidity, Starknet, Solhint Linter, LearnETH, Sourcify, More), "File" (New File, Open Files, Connect to Localhost), and "Resources" (Documentation, Gitter channel, Featuring website). At the bottom, there are buttons for "LOAD FROM:" (Gist, GitHub, IPFS, HTTPS) and a "Code" dropdown menu.

1. Es una interfaz de desarrollo que permite la creación rápida de Smart Contracts. Es usado normalmente para crear rápidos prototipos o pruebas de concepto de Smart Contracts.
2. Gracias a su gran variedad de compiladores de Solidity, es ideal para debuguear entre diferentes versiones y comprobar lo que funciona.
3. Permite la publicación de Smart Contracts en diferentes blockchain a través de Metamask.
4. Se puede sincronizar el desarrollo local de smart contracts (e.g. en Visual Studio Code) con Remix IDE para propósitos de compilación y publicación

Mythril (by ConsenSys)

The screenshot shows the GitHub repository page for "ConsenSys/mythril". The repository has 70 stars, 564 forks, and 2.7k contributors. It contains 175 branches and 148 tags. The "Code" tab is selected. The repository is maintained by "norhh" and the latest commit is "Mythril v0.23.10 (#1688)" from 18 days ago. The "About" section describes it as a security analysis tool for EVM bytecode, supporting Ethereum, Hedera, Quorum, Vechain, Roostock, Tron and other EVM-compatible blockchains. It also links to mythril.io.

1. Herramienta de análisis de seguridad (vulnerabilidades) para Smart Contracts. Usa el bytecode que se genera al compilar los Smart Contracts.
2. Es utilizado para proceso de auditoría dado que sugiere potenciales vectores de ataque a Smart Contracts. Luego del análisis, esta herramienta señala buenas prácticas usadas para combatir las falencias encontradas.
3. Herramienta gratuita. Se ejecuta en python y la manera más sencilla de usarlo es crear un entorno virtual en PyCharm (gratuito) y ejecutar el comando de mythril que analiza el Smart Contract.
4. Permite crear graphos que muestran las conexiones entre los diferentes Smart Contracts.

Alchemy/Infura/Moralis

The web3 development platform

The most powerful set of web3 development tools
to build and scale your dApp with ease.

[Get started for free](#)
[Play demo video](#)

1. Son servicios de conexión privado a nodos de blockchain.
2. Incrementan la velocidad de respuesta ante una transacción en el blockchain
3. Disminuye drásticamente los fallos de conexión entre una aplicación descentralizada y el Blockchain

Etherscan

1. Es un explorador de bloques y analítica para blockchain. Además permite revisar el código de los smart contracts publicados así como también su verificación.
2. Permite indagar detalles (quién llamó el método, qué contrato se llamó, cuánto gas consumió la operación, qué método se ejecutó) de transacciones hash y contratos.
3. Para poder verificar contratos de manera programática, se puede obtener un API KEY y usarlo en librerías como Hardhat.

Solidity (lenguaje de programación)



v0.8.17

Search docs

BASICS

- Introduction to Smart Contracts
- Installing the Solidity Compiler
- Solidity by Example

LANGUAGE DESCRIPTION

- Layout of a Solidity Source File
- Structure of a Contract
- Types

Solidity

Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behaviour of accounts within the Ethereum state.

Solidity is a [curly-bracket language](#) designed to target the Ethereum Virtual Machine (EVM). It is influenced by C++, Python and JavaScript. You can find more details about which languages Solidity has been inspired by in the [language influences](#) section.

Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

1. Solidity es el lenguaje de programación preferido entre desarrolladores así como también la gran mayoría de blockchains usan Solidity como su primer lenguaje para desarrollar.
2. Es muy parecido a Javascript, C++ y Python. Es estéticamente tipado, soporta herencia de contratos, el uso de librerías y la definición de tipos complejos por parte del usuario.
3. El punto de partida es definir una categoría llamada 'contract' (muy parecido al concepto de clase de Java) y crear métodos y variables dentro.
4. Su compilación produce bytecode y ABI. El primero es entendido por máquinas y el segundo por seres humanos.
5. Con Solidity se pueden crear contratos de votaciones, subastas, billeteras multifirmas, entre otros.

Faucets

RINKEBY FAUCET

Fast and reliable. 0.1 Rinkeby ETH/day.

Enter Your Wallet Address (0x...) or ENS Domain

Send Me ETH

[Please signup or login](#) with Alchemy to request ETH. It's free!

Your Transactions

Time

-

1. Proveen token nativos (ether) que sirven para pagar el gas de las transacciones en el blockchain.
2. Solo las redes Testnet tienen faucets. Cada testnet (mumbai, BSC testnet, Goerli, etc) posee un faucet donde pedir tokens nativos gratuitos.
3. Si se requieren tokens en mayor cantidad se puede contactar directamente a cada Blockchain en

Telegram o Discord.

MythX (auditoría automatizada)

The screenshot shows the MythX homepage. The main heading is "Smart contract security service for Ethereum". Below it is a paragraph about the service's mission to ensure Ethereum is secure. A "GET STARTED" button is visible. To the right, there are three overlapping windows showing code snippets and analysis results:

- Top window: /files/oldBlockHash.sol Line 74. Error: High - The binary addition can overflow.
- Middle window: MythX Deep Scan /files/oldBlockHash.sol. Shows a progress bar and a warning about a weak source of randomness.
- Bottom window: /files/oldBlockHash.sol Line 74. Warning: Medium - Potential use of a weak source of randomness "blockhash".

1. Servicio de análisis de Smart Contracts a demanda. A través de una simple interface gráfica, se muestran los resultados del análisis de vulnerabilidades de Smart Contracts.
2. Desde el Visual Studio Code, se puede usar una extensión para enviar a analizar los Smart Contracts. Del mismo modo, existen comandos de terminal para realizar la misma acción

Programando en Solidity

Vamos a aprender a programar en Solidity con el objetivo de desarrollar una startup ficticia que desea llevar al mercado sus NFTs. A medida que aprendemos Solidity, iremos desarrollando diferentes Smart Contracts requeridos.

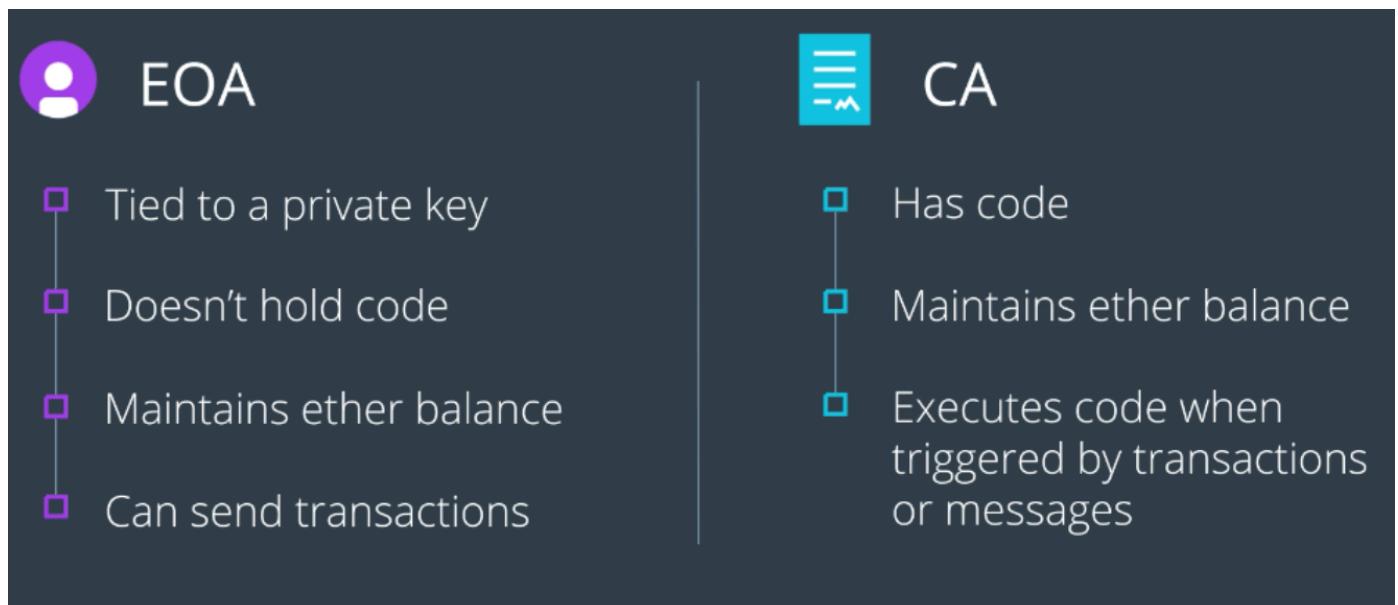
El primer Smart Contract que vamos a desarrollar será la criptomoneda (token ERC20) de la empresa que será publicado en una Testnet llamada Goerli. Antes de lograr ello, revisemos algunos conceptos sobre Solidity.

¿Qué es Solidity?

- Es un lenguaje orientado a objetos
- Lenguaje de alto nivel para la implementación de Smart Contracts
- "El código es la ley": un smart contract luego de publicado imposibilita su modificación y se ejecuta por una máquina tal cual fue redactado
- Lenguaje de llaves diseñado para desarrollar código compatible con la Máquina Virtual de Ethereum (EVM)
- Influenciado por C++, Python y Javascript

- Es estáticamente tipado, soporta la herencia (de objetos), librerías y definición de tipos de datos complejos definidos por el usuario

Dos tipo de cuentas en Ethereum



- EOA (Externally owned account): Son usuarios (personas) que posee una llave privada. No posee código. Pueden mantener un balance positivo de Ether. Firma transacciones. Puede transferir activos (assets).
- SCA (Smart Contract Account): Son cuentas controladas por código dentro del Smart Contract.

Mi primer contrato en Solidity

1_MyFirstContract

En [Remix](#), crear un nuevo archivo llamada

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract MiPrimerContrato {
    string saludo;

    function set(string memory _nuevoSaludo) public {
        saludo = _nuevoSaludo; // no se necesita 'this'
    }

    function get() public view returns (string memory) {
        return saludo;
    }
}
```

- La primera línea nos indica la licencia del código a ser publicado de una manera en que la máquina puede entender. Imprescindible cuando se desea publicar el contrato en una red blockchain.
- La siguiente línea nos indica la versión de Solidity en la cual el código fue escrito. La palabra clave `pragma` hace referencia a instrucciones para que el compilador sepa cómo tratar el código. Cada

versión de compilador podría generar un diferente output con respecto al anterior. Al especificar versiones te aseguras compile como lo específicas según las versiones.

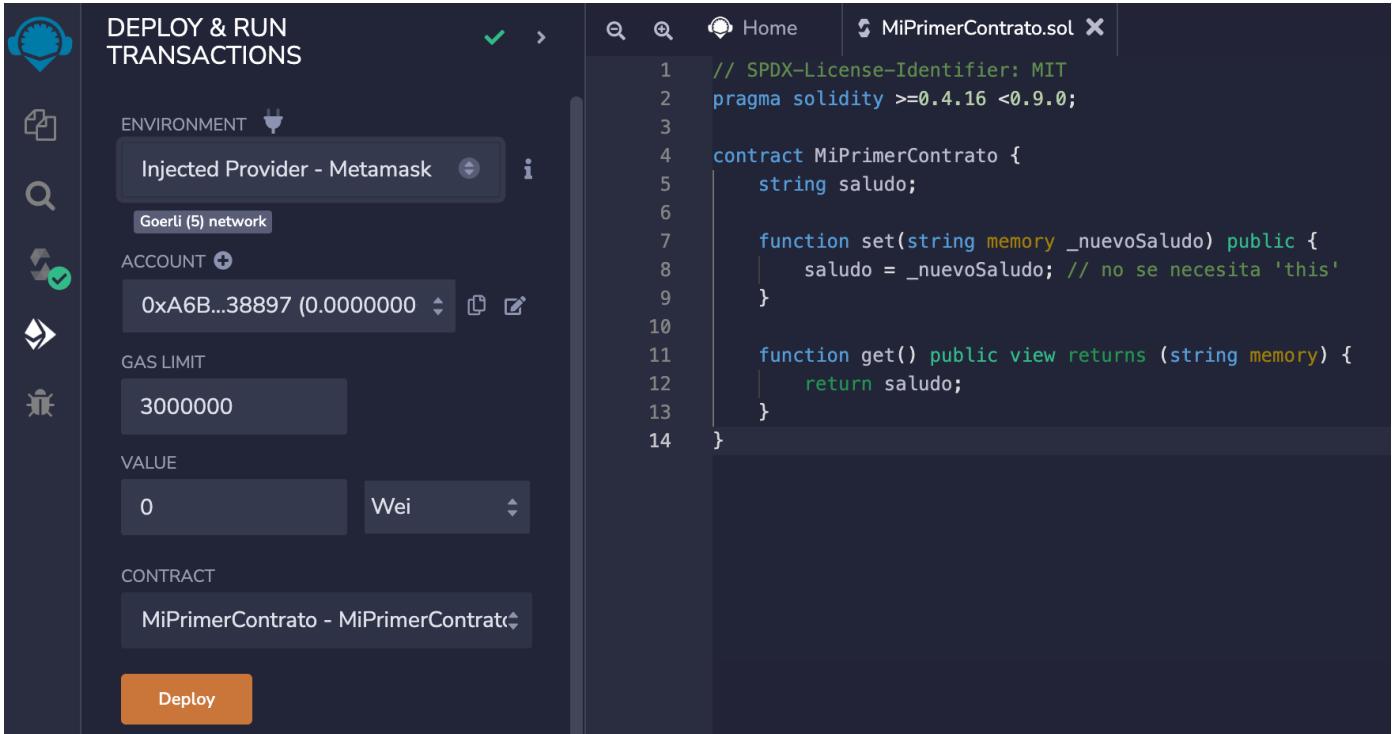
- Un contrato es una colección de código y estado (code + state) que vive dentro de una dirección específica en el Blokchain. En este contrato definimos una variable llamda `saludo` del tipo `string`. Se puede entender esta variable como si fuera una entrada en una base de datos que se puede consultar (usando `get`) y modificar (mediante `set`).

La gran diferencia de escribir este mismo código en otro lenguaje de programación, como Javascript, es que para lograr lo mismo tendríamos que levantar un servidor y una base de datos. Además de la creación de la clase (Class - ES6) que lleve los mismo métodos, se requeriría métodos para poder guardar y leer información de la base de datos. Ello sin contar la creación de la conexión con la base de datos.

Usando Smart Contracts, el "servidor" y la "base de datos" están dados por la Máquina Virtual de Ethereum (EVM). Las lecturas y escrituras a raíz de la ejecución del código, se hace desde y sobre el blockchain.

Publicar el Smart Contract

1. Para publicar el contrato usar Metamask en la red Testnet de Goerli. Previamente solicitar Ether en algún [faucet de Goerli](#).



The screenshot shows the Remix IDE interface. On the left, there's a sidebar titled 'DEPLOY & RUN TRANSACTIONS' with sections for ENVIRONMENT (set to 'Injected Provider - Metamask'), ACCOUNT (showing address 0xA6B...38897), GAS LIMIT (set to 3000000), VALUE (set to 0 Wei), and CONTRACT (set to 'MiPrimerContrato - MiPrimerContrato'). At the bottom is a large orange 'Deploy' button. On the right, the code editor displays the Solidity source code for 'MiPrimerContrato.sol':

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract MiPrimerContrato {
    string saludo;

    function set(string memory _nuevoSaludo) public {
        saludo = _nuevoSaludo; // no se necesita 'this'
    }

    function get() public view returns (string memory) {
        return saludo;
    }
}
```

2. En `ENVIRONMENT` escoger `Injected Provider - Metamask`, lo cual conectará el IDE de Remix con una billetera de Metamask.
3. En `CONTRACT` asegurar que está seleccionado el contrato que se desea publicar
4. Al hacer clic en `Deploy`, abrirá un pop-up de Metamask para poder confirmar y firmar la transacción, lo cual hará posible la creación del Smart Contract en el Blockchain.



5. Cuando la transacción haya terminado, se podrá visualizar dentro de la pestaña de `Actividad`. Hacer click en `Implementación de contrato` y se abrirá otra ventana. En dicha ventana hacer click en `Ver en el explorador de blocks`.

| Operación | Fecha | Detalles | Saldo |
|------------------------|-------|---------------------|----------------|
| Set | Oct 2 | goerli.etherscan.io | -0 GoerliETH |
| Implementación de c... | Oct 2 | remix.ethereum.org | -0 GoerliETH |
| Recibir | Oct 2 | De: 0xa6b...8897 | 0.0506835 ... |
| Recibir | | | 0.16766402 ... |

6. Serás dirigido a Goerli.etherescan.io donde podrás ver los detalles de la transacción (publicación del Smart Contract). Se puede observar que el contrato ha sido creado en la siguiente dirección: `0xc5bccf767704432a3a22318a0df3067d9a3fc217`. Esta misma dirección servirá para hacer su posterior verificación.

Transaction Details

[\(Back\)](#) [\(Forward\)](#)

[Overview](#) [State](#)

[This is a Goerli Testnet transaction only]

② Transaction Hash: [0x2deacfb4bc3d791a245fb1e8df04f3f1a0e91d704d0a40f8287997d6500c4e25](#)

② Status: Success

② Block: [7698163](#) 5 Block Confirmations

② Timestamp: 1 min ago (Oct-02-2022 11:44:12 AM +UTC)

② From: [0x5387ddeec8ddc004a217d8e172241eb5f900b302](#)

② To: Contract [0xc5bccf767704432a3a22318a0df3067d9a3fc217](#)

② Value: 0 Ether (\$0.00)

② Transaction Fee: 0.00004504500045045 Ether (\$0.00)

② Gas Price: 0.00000000100000001 Ether (1.00000001 Gwei)

Verificación de un Smart Contract

- Para verificar, hacer click en la dirección del contrato creado en el anterior paso. O en su defecto, ir al siguiente link <https://goerli.etherscan.io/address/0xc5bccf767704432a3a22318a0df3067d9a3fc217>, del cual la última parte será reemplazada por la dirección (address) de tu contrato.

The screenshot shows the Etherscan interface for the Goerli Testnet Network. The address of the smart contract is [0xc5bccf767704432a3a22318a0df3067d9a3fc217](#). The 'Contract Overview' section shows a balance of 0 Ether. The 'More Info' section shows 'My Name Tag: Not Available' and 'Contract Creator: 0x5387ddeec8ddc004a...' at txn [0x881c775390c3102dd...](#). Below this, the 'Transactions' tab is selected, showing two recent transactions:

| Txn Hash | Method | Block | Age | From | To | Value | Txn Fee |
|---|------------|---------|-------------|--|--|---------|---------------------|
| 0x2deacfb4bc3d791a24... | Set | 7698163 | 21 mins ago | 0x5387ddeec8ddc004a... | 0xc5bccf767704432a3a2... | 0 Ether | 0.00004504500045045 |
| 0x881c775390c3102dd... | 0x60806040 | 7698130 | 30 mins ago | 0x5387ddeec8ddc004a... | Create: MiPrimerContrato | 0 Ether | 0.00077465 |

At the bottom right, there is a link to [Download CSV Export].

- Hacer click en la pestaña [Contract](#) que te permitirá ver el bytecode generado del Smart Contract. Para verificar, hacer clic en [Verify and Publish](#).

3. Se abrirá una lista de opciones que tienen que ser llenadas de la siguiente manera: address del smart contract, Single File, versión del compilador (debe ser la misma usada en Remix), MIT de licencia.

Please enter the Contract Address you would like to verify

Please select Compiler Type

Solidity (Single file)

Please select Compiler Version

v0.8.7+commit.e28d00a7

Un-Check to show all nightly Commits also

3) MIT License (MIT)

I agree to the [terms of service](#)

Continue

Beset

4. En esta ventana copias y pegas el código de Remix. Verificas el CAPTCHA. Luego clic en **Verify and Publish**.

Enter the Solidity Contract Code below *

[Fetch from Gist](#)

```
contract MiPrimerContrato {
    string saludo;

    function set(string memory _nuevoSaludo) public {
        saludo = _nuevoSaludo; // no se necesita 'this'
    }

    function get() public view returns (string memory) {
        return saludo;
    }
}
```

1

Constructor Arguments **ABI-encoded** (for contracts that were created with constructor parameters)

▼

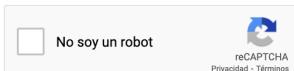
For additional information on Constructor Arguments see Our KB Entry ↗

Contract Library Address (for contracts that use libraries, supports up to 10 libraries)

↗

Misc Settings (Runs, EvmVersion & License Type settings)

↗



[Verify and Publish](#) [Reset](#) [Return to Main](#)

4. Si todos los valores fueron incluidos correctamente, se verá el siguiente resultado:

Verify & Publish Contract Source Code

Compiler Type: SINGLE FILE / CONCATENATED METHOD

Info: A simple and structured interface for verifying smart contracts that fit in a single file

[Contract Source Code](#) [Compiler Output](#)

Compiler debug log:

- ✓ Note: Contract was created during TxHash# [0xd419965f189a61df4262b1fab2a3e269e8f55a65780945a1df32c71dc3ebf1d1](#)
- ✓ Successfully generated ByteCode and ABI for Contract Address [0xf646be4fb9c956dc40fdf02378b1415b4be32e54]

Compiler Version: v0.8.7+commit.e28d00a7

Optimization Enabled: 0

Runs: 200

ContractName:

MiPrimerContrato

ContractBytecode:

```
608060405234801561001057600080fd5b506104a8806100206000396000f3fe608060405234801561001057600080fd5b50600436106100365760003560e01c80634ed388:600080fd5b61005560048036038101906100509190610234565b610075565b005b61005f61008f565b6040516006c91906102b6565b60405180910390f35b806000908051:565b60606000805461009e9061038c565b80601f01602080910402602001604051908101604052809291908181526020018280546100ca9061038c565b8015610117578060:200191610117565b820191906000526020600020905b81548152906010190602001808311610fa57829003601f168201915b505050505005090565b82805461012d9061:2090048101928261014f5760008555610196565b82601f1061016857805160ff1916838001178555610196565b82800160010185558215610196579182015b828111156101:
```

6. Al dirigirte a tu contrato en Goerli.etherscan.io con el siguiente link

<https://goerli.etherscan.io/address/0xc5bccf767704432a3a22318a0df3067d9a3fc217>, del cual la última parte será reemplazada por la dirección (address) de tu contrato, podrás (1) encontrar el código del Smart Contract, (2) interactuar con el contrato directamente ([Read Contract](#) y [Write Contract](#)) y

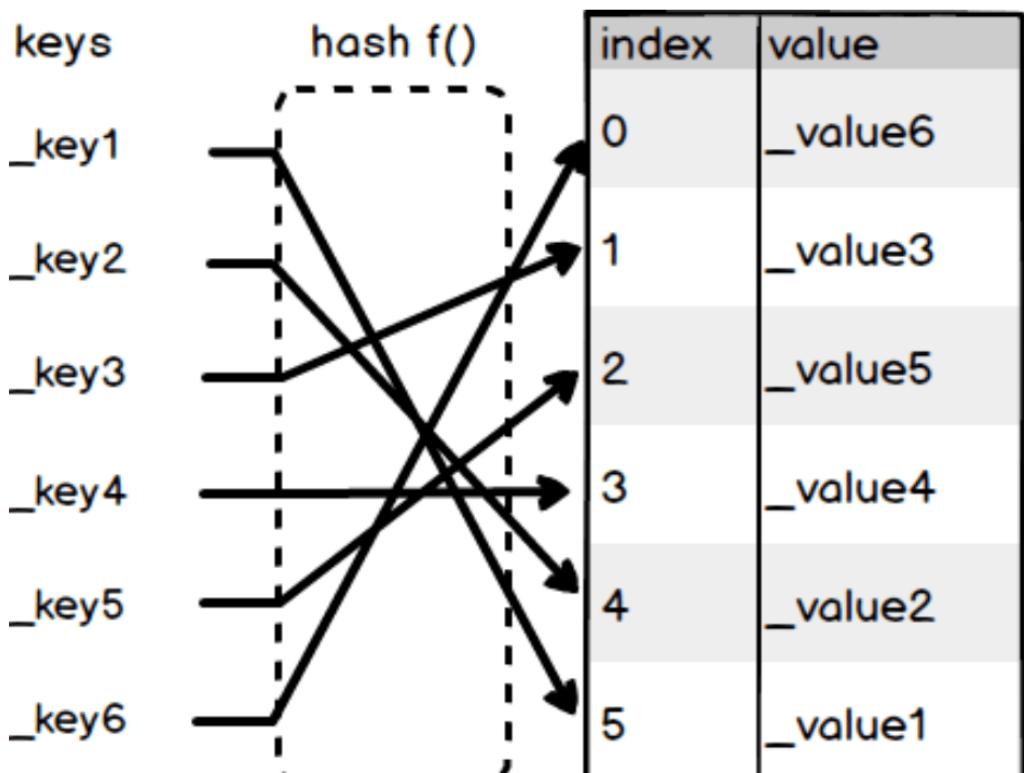
(3) observar otros detalles del mismo.

The screenshot shows the Etherscan interface for a verified Solidity contract named 'MiPrimerContrato'. The top navigation bar includes 'Transactions', 'Erc20 Token Txns', 'Contract' (which is highlighted in blue), and 'Events'. Below the navigation are three buttons: 'Code' (selected), 'Read Contract', and 'Write Contract'. A green checkmark icon indicates 'Contract Source Code Verified (Exact Match)'. The contract details section shows the name 'MiPrimerContrato', compiler version 'v0.8.7+commit.e28d00a7', optimization status 'No with 200 runs', and other settings 'default evmVersion, MIT license'. The 'Contract Source Code (Solidity)' tab is selected, displaying the following code:

```
1  /**
2  *Submitted for verification at Etherscan.io on 2022-10-02
3  */
4
5 // SPDX-License-Identifier: MIT
6 pragma solidity >=0.4.16 <0.9.0;
7
8 contract MiPrimerContrato {
9     string saludo;
10
11     function set(string memory _nuevoSaludo) public {
12         saludo = _nuevoSaludo;
13     }
14
15     function get() public view returns (string memory) {
16         return saludo;
17     }
18 }
```

Hash table en contratos inteligentes

La estructura de datos llamado mapping es uno de los más usados en Solidity. `mapping(_KeyType => _ValueType)` Es el equivalente a un Hash Table o un objeto (`var obj = {}`) en Javascript. A cada `key` le corresponde un `value` dentro del mapping.



example of a hash table

`_KeyType` no puede ser otro mapping, struct o array. `_ValueType` puede ser de cualquier tipo, incluyendo mapping, arrays y structs.

Un `mapping` empieza con una inicialización de todos los posibles valores de `_KeyType` que están mapeados a un valor por defecto que es 0. Además, con `mapping` no se lleva la cuenta de los keys cuyos valores sea 0. Ello justamente impide que no se pueda borrar un `mapping` a menos que se sepa el `key`.

Los `mapping`s solo pueden tener un tipo de ubicación de información: `storage`. No se pueden usar `mapping`s como parámetros de una función o como el valor de retorno.

Un `mapping` no tiene longitud (`length`), como lo puede tener un array. Un `mapping` tampoco es iterable porque no hay manera de conocer sus `key`s mediante ningún método. Se puede guardar las llaves del `mapping` en otro array para poder iterar luego.

En el siguiente ejemplo se incluye un `mapping` para guardar una lista de saludos en el cual el `_KeyType` se va incrementando en uno a medida que la función `set` es llamada.

2_Mapping

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract MiPrimerContrato {
    string saludo; // empieza como un string vacío ('') por definición
```

```

uint256 counter; // empieza en zero por definición
mapping(uint256 => string) listaSaludos;

function set(string memory _nuevoSaludo) public {
    saludo = _nuevoSaludo;

    // guardando en el mapping;
    listaSaludos[counter] = _nuevoSaludo;
    counter++; // counter += 1; // counter = counter + 1;
}

function get() public view returns (string memory) {
    return saludo;
}
}

```

El tipo de data `Address`

Cada EOA (externally owned account) y Smart Contract Account tiene una dirección (`address`). Se guarda como un valor de 20 bytes (160 bits o 40 caractéres hexadecimales). Siempre se le prefija el `0x` por el formato hexadecimal. Es usado para enviar y recibir Ether, así como también otras criptomonedas no nativas.

Ejemplo: `0x5387ddeec8ddC004a217d8e172241EB5F900B302`

Puede ser considerado como una indentidad pública en el Blockchain, más como un seudónimo. Para ser más preciso, se puede entender como una cuenta de banco. En el mismo modo en que necesitas una cuenta de banco para recibir y enviar dinero, se usará el `address` para enviar y/o recibir dinero, además de realizar transacciones.

En lo sucesivo se usará `address` como identificador único de un usuario involucrado en realizar alguna transacción dentro del Smart Contract.

Supongamos que deseamos asociar la edad (`uint256`) de cada usuario con su `address`. Para ello usaremos un mapping:

3_MappingEdad

```

// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract MiPrimerContrato {
    string saludo; // empieza como un string vacío ('') por definición

    uint256 counter; // empieza en zero por definición
    mapping(uint256 => string) listaSaludos;

    // mapping address a edad
    mapping(address => uint256) public edadPorAddress;
}

```

```

function set(string memory _nuevoSaludo) public {
    saludo = _nuevoSaludo;

    // guardando en el mapping;
    listaSaludos[counter] = _nuevoSaludo;
    counter++; // counter += 1; // counter = counter + 1;
}

function get() public view returns (string memory) {
    return saludo;
}

function setEdadPorAddress(address _account, uint256 _edad) public {
    edadPorAddress[_account] = _edad;
}
}

```

El método `setEdadPorAddress` nos ayuda a mapear un `address` a una `edad` en particular.

Cabe resaltar que la palabra clave `public` se ha utilizado cuando se define el mapping de `edadPorAddress`. Al usar esta palabra clave, Solidity, automáticamente, ha creado un getter. Sin la palabra clave `public`, se tendría que añadir el siguiente código:

```

function getEdadporAddress(address _account) public view returns (uint256 _edad) {
    return edadPorAddress[_account];
}

```

Tipos de llaves (key) and valores (value) permitidos:

```
mapping(keyType => ValueType) mappingName;
```

| Variable Type | Key Type | Value Type |
|------------------------------------|----------|------------|
| int / uint | ✓ | ✓ |
| string | ✓ | ✓ |
| byte / bytes | ✓ | ✓ |
| address | ✓ | ✓ |
| struct | ✗ | ✓ |
| mapping | ✗ | ✓ |
| enum | ✗ | ✓ |
| contract | ✗ | ✓ |
| * fixed-sized array T[k] | ✓ | ✓ |
| * dynamic-sized array T[] | ✗ | ✓ |
| * multi-dimentional array T[][] | ✗ | ✓ |
| variable | ✗ | ✗ |

- For arrays, T refers to Type and k to the length of the array.

El doble mapping es Smart Contracts es usando frecuentemente y cabe ahondar en su entendimiento y uso. Si hablamos de base de datos, esta relación podría considerarse *one-to-many*. Veamos el siguiente ejemplo:

3_5_DoubleMapping

```
// SPDX-License-Identifier: MIT
```

```

pragma solidity >=0.4.16 <0.9.0;

contract DoubleMapping {
    // Queremos llevar una contabilidad de las deudas de cada persona (usando su nombre)
    // Esto requiere de un simple 'mapping' que asocia string => uint256
    mapping(string => uint256) saldos;

    function fijarSaldo(string memory _name, uint256 _saldo) public {
        saldos[_name] = _saldo;
    }

    function leerSaldo(string memory _name) public view returns (uint256) {
        return saldos[_name];
    }

    /**
     * Realizar las siguientes operaciones
     * 1. Registrar que Lee tiene un saldo de 10,000
     * 2. Registrar que Jen tiene un saldo de 100,000
     */

    Desarrollo:
    fijarSaldo("Lee", 10000)
    fijarSaldo("Jen", 100000)
    */

    // Queremos llevar la contabilidad de cuanto debe una persona a otra usando su nombre
    // Esto require de un doble mapping porque una persona puede deber a varias personas
    // Lee -> Carmen: debe 100
    // Lee -> Jen: debe 200
    // Lee -> Lea: debe 300
    // Carmen -> Lee: 200
    // Carmen -> Jhon: 400
    mapping(string => mapping(string => uint256)) saldosMatrix;

    function fijarSaldoMatrix(
        string memory acreedor, // al que se le debe
        string memory deudor, // el que debe
        uint256 amount
    ) public {
        saldosMatrix[acreedor][deudor] = amount;
    }

    function leerSaldoMatrix(
        string memory acreedor, // al que se le debe
        string memory deudor // el que debe
    ) public view returns (uint256) {
        return saldosMatrix[acreedor][deudor];
    }
}

```

```

/**
Desarrollo:
    fijarSaldoMatrix("Carmen", "Lee", 100);
    fijarSaldoMatrix("Jen", "Lee", 200);
    fijarSaldoMatrix("Lea", "Lee", 300);
    fijarSaldoMatrix("Lee", "Carmen", 200);
    fijarSaldoMatrix("Jhon", "Carmen", 400);
*/
}

```

Limitaciones de la estructura de datos `mapping`:

- Existe un conjunto de tipos definidos para ser usados en la llave (`KeyType`) del mapping
- No se puede iterar sobre un `mapping` porque virtualmente todas las llaves son inicializadas. Tampoco un `mapping` tiene longitud.
- Tampoco es posible solicitar todas las llaves del `mapping`, por la razón anterior.
- Un `mapping` no se puede usar como valor de retorno de una función.

Propagación de un Error vía `require` o `revert`

`require` o `revert` en Solidity es usado para validar ciertas condiciones dentro del código y lanzar una excepción si dicha condición no es cumplida. Esto es importante para prevenir la finalización de una transacción si se detecta una condición indeseada.

Cabe mencionar que esta propagación del error será notada por el usuario en el front-end (dApp) antes de firmar una transacción mediante su billetera (de Metamask u otra).

Veamos cómo aplicamos `require` o `revert` en el código:

4_ErroRevert.sol

```

// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract MiPrimerContrato {
    //...

    function setEdadPorAddressManejaError(address _account, uint256 _edad)
        public
    {
        require(_edad >= 30, "Edad menor a 30");
        edadPorAddress[_account] = _edad;
    }

    function setEdadPorAddressManejaError2(address _account, uint256 _edad)
        public
    {
        if (_edad <= 29) {
            revert("Edad menor a 30");
        }
    }
}

```

```

edadPorAddress[_account] = _edad;
}
}

```

`revert` y `require` propagarán el error si es que no cumple las condiciones allí definidas. La única diferencia entre uno y otro es que `require` lleva el condicional y el mensaje de error como argumentos de un método. En cambio, `revert` ofrece mayor flexibilidad para validar y plantear las condiciones a cumplir. `revert` solo lleva como argumento el mensaje del error.

Ejemplo de cómo se vería un error en producción [link](#):

Transaction Details

| Overview | Logs (1) |
|--|--|
| [This is a Polygon PoS Testnet transaction only] | |
| ② Transaction Hash: | 0xc96a8ad2c078065dae2c1fb02cf2590870346e17c5055e5ddc4637eb3f85d977 🔗 |
| ② Status: | ✖ Fail with error 'Mint amount greater than max supply' |
| ② Block: | 26335176 2123318 Block Confirmations |
| ② Timestamp: | ⌚ 142 days 14 mins ago (May-16-2022 11:23:46 AM +UTC) |
| ② From: | 0xc27f232e007590eeb991f997c51f5a0bf0cd3875 🔗 |
| ② To: | Contract 0xfb3319771fefbbe4b3d3413d78b4a9976094ee64 ⚠ 🔗 ↳ Warning! Error encountered during contract execution [execution reverted] ⚠ |
| ② Value: | 0 MATIC (\$0.00) |

Usando eventos a modo de notificación

`Events` dentro de Solidity son disparados cuando algún método en particular es ejecutado. Los eventos pueden llevar información adicional para explicar lo que está sucediendo. Normalmente, el nombre del evento seguido de la información que contiene, explica muy bien un suceso dentro del blockchain.

Los eventos disparados desde un Smart Contract son propagados en el Blockchain. Dichos eventos quedan registrados por siempre. En un futuro se pueden hacer queries de eventos disparados anteriormente. Incluso se puede usar para almacenar información de manera económica. Estos eventos pueden ser captados desde el front-end en un dApp si se establece una conexión.

5_Events.sol

```

// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract MiPrimerContrato {
    string saludo; // empieza como un string vacío ('') por definición
}

```

```

uint256 counter; // empieza en zero por definición
mapping(uint256 => string) listaSaludos;

// mapping address a edad
mapping(address => uint256) public edadPorAddress;

// Eventos
// 1 - Disparar un evento cada vez que se cambia el saludo
// _account - address de la persona que está llamando el método
// _nuevoSaludo - string que representa al nuevo saludo
event CambioDeSaludo(address _account, string _nuevoSaludo);

// 2 - Disparar un evento cuando se asocia un 'address' con una edad
// _account - address para el cual se asocia la edad
// _edad - nueva edad para asociar con un address
event NuevaEdadParaAddress(address _account, uint256 _edad);

function set(string memory _nuevoSaludo) public {
    saludo = _nuevoSaludo;

    // guardando en el mapping;
    listaSaludos[counter] = _nuevoSaludo;
    counter++; // counter += 1; // counter = counter + 1;
    emit CambioDeSaludo(msg.sender, _nuevoSaludo);
}

function get() public view returns (string memory) {
    return saludo;
}

function setEdadParaAddress(address _account, uint256 _edad) public {
    edadPorAddress[_account] = _edad;
    emit NuevaEdadParaAddress(_account, _edad);
}

```

Ejemplos de eventos propagados en la red [link](#).

msg.sender

Es la cuenta (address) que llama o ha ejecutado una función (de smart contract) o ha creado una transacción.

Esta cuenta (address) puede ser una dirección de un contrato (CA) o una persona como nosotros (EOA).

`msg.sender` funciona como una variable global dentro de Solidity y puede ser usada dentro de los métodos del Smart Contract como una variable ya definida.

Otras variables globales en Solidity [link](#).

5 5 MsgSender.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract MiPrimerContrato {
    // ...
    address caller;
    function setCaller() public returns(address) {
        caller = msg.sender;
        return msg.sender;
    }
}
```

Consideraciones para la creación de una criptomoneda

Comenzaremos con la creación de una criptomoneda desde cero. Sin librerías. En la actualidad se crean criptomonedas con diez líneas de código. Como desarrolladores, es importante conocer el funcionamiento interno. Más adelante, utilizaremos librerías que acelaran el proceso mediante templates testeados y auditados. Repasemos los elementos esenciales de todo token (escrito en código en un Smart Contract):

1. Una criptomoneda debería tener un nombre que lo identifique
2. Una criptomoneda debería tener un símbolo que lo identifique
3. Definir la cantidad de decimales del token (normalmente hay 18 pero otros tokens tienen 6, como el USDC)
4. Internamente debería llevar la cuenta de los balances de cada persona que tiene criptomoneda
5. Llevar la cuenta del total de tokens repartidos
6. Método que permite la acuñación de tokens a favor de una cuenta en particular (`mint`)
7. Método que permite quemar (`burn`) tokens. La lógica detrás de esto es que genera deflación (menos dinero en la economía)
8. Método que permite transferir tus propios tokens a una segunda persona (método `transfer`)
 - o Internamente validar que el usuario tiene más tokens de los que quiere enviar
9. Llevar la cuenta de los balances de tokens a gastar que los mismos dueños (del token) han autorizado a otras cuentas para gastar en su representación
10. Método que permite transferir tokens en nombre de una segunda persona con previa aprobación de la segunda persona (método `transferFrom`)
 - o Validar que esa segunda persona tiene más tokens de lo que se planea enviar
11. Definir métodos para incrementar el permiso de gastar tokens de otra persona
12. Disparar eventos de Transferencia cada vez que se transfieren tokens de un lado a otro. Disparar eventos de Aprobación cada vez que una cuenta le da permiso a otra para gastar sus tokens
13. Método para visualizar el total de tokens de una cuenta
14. Método para visualizar la cantidad de tokens a gastar en nombre de otra persona con su previo permiso

A continuación se muestra el primer borrador en base a las consideraciones expuestas:

6 ERC20-1.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract ERC20Generic {
  /**
   1. Una criptomoneda debería tener un <u>nombre</u> que lo identifique
   2. Una criptomoneda debería tener un <u>símbolo</u> que lo identifique
   3. Definir la cantidad de <u>decimales</u> del token (normalmente hay 18 pero otros tokens tienen 6, como el USDC)
   4. Internamente debería llevar la <u>cuenta de los balances</u> de cada persona que tiene criptomoneda
   5. Llevar la <u>cuenta del total de tokens</u> repartidos
   6. Método que permite la <u>acuñación</u> de tokens a favor de una cuenta en particular (`mint`)
   7. Método que permite <u>quemar</u> (burn) tokens. La lógica detrás de esto es que genera deflación (menos dinero en la economía)
```

```

    8. Método que permite <u>transferir</u> tus propios tokens a una segunda persona (método `transfer`)
        * Internamente validar que el usuario tiene más tokens de los que quiere enviar
    9. Llevar la cuenta de los balances de tokens a gastar que los mismos dueños (del token)
    han <u>autorizado a otras cuentas para gastar</u> en su representación
    10. Método que permite <u>transferir tokens en nombre</u> de una segunda persona con
    previa aprobación de la segunda persona (método `transferFrom`)
        * Validar que esa segunda persona tiene más tokens de lo que se planea enviar
    11. Definir métodos para incrementar el permiso de gastar tokens de otra persona
    12. Disparar eventos de Transferencia cada vez que se transfieren tokens de un lado a
    otro. Dispararar eventos de Aprobación cada vez que una cuenta le da permiso a otra para gastar
    sus tokens
    13. Método para visualizar el total de tokens de una cuenta
    14. Método para visualizar la cantidad de tokens a gastar en nombre de otra persona con
    su previo permiso
 */

// 5. Llevar la <u>cuenta del total de tokens</u> repartidos
uint256 totalSupply;

// 4. Internamente debería llevar la <u>cuenta de los balances</u> de cada persona que
tiene criptomoneda
// Es decir, a cada billetero se le debe asociar la cantidad de tokens que tiene
mapping(address => uint256) balances;

// 9. Llevar la cuenta de los balances de tokens a gastar que los mismos dueños (del
token) han <u>autorizado a otras cuentas para gastar</u> en su representación
mapping(address => mapping(address => uint256)) _allowances; // permisos

// 12. Disparar eventos de Transferencia cada vez que se transfieren tokens de un lado a
otro. Dispararar eventos de Aprobación cada vez que una cuenta le da permiso a otra para gastar
sus tokens
event Transfer(address from, address to, uint256 value);

// 1. Una criptomoneda debería tener un <u>nOMBRE</u> que lo identifique
function name() public pure returns (string memory) {
    return "My primer token";
}

// 2. Una criptomoneda debería tener un <u>sÍMBOLO</u> que lo identifique
function symbol() public pure returns (string memory) {
    return "MPTK";
}

// 3. Definir la cantidad de <u>decimales</u> del token (normalmente hay 18 pero otros
tokens tienen 6, como el USDC)
function decimals() public pure returns (uint256) {
    return 18;
}

```

```

// 6. Método que permite la <u>acuñación</u> de tokens a favor de una cuenta en particular (`mint`)
// Los parámetros son la billetera (address) que recibirá los tokens y la cantidad de tokens
function mint(address _account, uint256 _amount) public {
    totalSupply += _amount;
    balances[_account] += _amount;

    emit Transfer(address(0), _account, _amount);
}

// 7. Método que permite <u>quemar</u> (burn) tokens. La lógica detrás de esto es que genera deflación (menos dinero en la economía)
function burn(address _account, uint256 _amount) public {
    totalSupply -= _amount;
    balances[_account] -= _amount;
    emit Transfer(_account, address(0), _amount);
}

// 8. Método que permite <u>transferir</u> tus propios tokens a una segunda persona (método `transfer`)
function transfer(address _account, uint256 _amount) public {
    balances[msg.sender] -= _amount;
    balances[_account] += _amount;

    emit Transfer(msg.sender, _account, _amount);
}

// 10. Método que permite <u>transferir tokens en nombre</u> de una segunda persona con previa aprobación de la segunda persona (método `transferFrom`)
// Lleva parámetros
// - la cuenta de la persona que autorizó la transferencia
// - La cuenta de la empresa que recibirá los tokens
// - la cantidad de tokens ser transferidos
function transferFrom(
    address _sender,
    address _recipient,
    uint256 _amount
) public {
    // verificar permiso
    uint256 allowance_ = allowances[_sender][msg.sender];
    require(allowance_ >= _amount, "No tiene permiso");

    // transferir entre dos cuentas
    balances[_sender] -= _amount;
    balances[_recipient] += _amount;

    allowances[_sender][msg.sender] = allowance_ - _amount;
}

```

```

}

// 11. Definir métodos para incrementar el permiso de gastar tokens de otra persona
// Este método solo puede ser llamado por la persona que desea otorgar el permiso a
// otra
function approve(address spender, uint256 amount) public returns (bool) {
    _allowances[msg.sender][spender] = amount;
    return true;
}

// 13. Método para visualizar el total de tokens de una cuenta
function balanceOf(address _account) public view returns (uint256) {
    return balances[_account];
}

// 14. Método para visualizar la cantidad de tokens a gastar en nombre de otra persona con
// su previo permiso
function allowance(address _owner, address _spender)
    public
    view
    returns (uint256)
{
    return _allowances[_owner][_spender];
}
}

```

Publicar y verificar el Smart Contract en la red Goerli Testnet del mismo modo que se publicó [aquí](#).

Sin embargo, este Smart Contract no tiene las verificaciones necesarias y requeridas para evitar errores. Por ejemplo, no se verifica si un usuario antes de transferir tokens, tenga el balance suficiente. Tampoco se verifica que en una transferencia, el destinatario sea una cuenta sin llave privada. A continuación se muestra un Smart Contract que hace exactamente lo mismo que el anterior y además incluye las validaciones necesarias:

6 ERC20-2.sol

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract ERC20Generic {
    uint256 totalSupply;

    mapping(address => uint256) balances;
    mapping(address => mapping(address => uint256)) _allowances; // permisos

    event Transfer(address from, address to, uint256 value);
    event Approval(address owner, address spender, uint256 value);

    function name() public pure returns (string memory) {

```

```

        return "My primer token";
    }

function symbol() public pure returns (string memory) {
    return "MPTK";
}

function decimals() public pure returns (uint256) {
    return 18;
}

// 1. Esta función está expuesta para ser llamada por cualquier persona
// Solo cuentas (addresses) con privilegios deberían poder llamar el método de acuñación
de tokens
// Vamos a convertirla a una función 'interna', es decir, que solo puede ser usada
dentro del Smart Contract y no llamada públicamente
// Más adelante aprenderemos sobre 'modifiers' para poder otorgar privilegios de
ejecución a ciertas cuentas (addresses)
// 2. Es una buena práctica verificar que el destinatario no sea un address sin llave
privada (e.g. address(0) === 0x000000...000)
function _mint(address _account, uint256 _amount) internal {
    // 2. Es una buena práctica verificar que el destinatario no sea un address sin llave
privada (e.g. address(0) === 0x000000...000)
    require(_account != address(0), "Mint to the zero address");

    totalSupply += _amount;
    balances[_account] += _amount;

    emit Transfer(address(0), _account, _amount);
}

// 1. Con este método se pueden quemar los tokens de cualquier cuenta
// Vamos a convertir este método en una función interna
// 2. Validar que no se estén quemando los tokens del address(0)
// 3. Vamos a validar que la cuenta tenga la cantidad suficiente de tokens para quemar
function _burn(address _account, uint256 _amount) internal {
    // 2. Validar que no se estén quemando los tokens del address(0)
    require(_account != address(0), "Burn from the zero address");

    uint256 balance_ = balances[_account];
    require(balance_ >= _amount, "Not enough tokens to burn");

    totalSupply -= _amount;
    balances[_account] -= _amount;
    emit Transfer(_account, address(0), _amount);
}

// Solución al burn sin protección
// _burn es una función más generica pero interna

```

```

// burn obliga a que el que llama el método (msg.sender), se vea afectado. Método público
function burn(uint256 _amount) public {
    _burn(msg.sender, _amount);
}

// 1. Es buena práctica verificar que el que envía y el que recibe no sean un address sin
// clave privada (e.g. address(0))
// 2. El que envía los tokens debería tener la suficiente cantidad de tokens
function transfer(address _to, uint256 _amount) public returns (bool) {
    // 1. Es buena práctica verificar que el que envía y el que recibe no sean un address
    // sin clave privada (e.g. address(0))
    require(_to != address(0), "Sending to zero address");

    // 2. El que envía los tokens debería tener la suficiente cantidad de tokens
    uint256 senderBalance = balances[msg.sender];
    require(senderBalance >= _amount, "Sender does not have enough tokens");

    balances[msg.sender] -= _amount;
    balances[_to] += _amount;

    emit Transfer(msg.sender, _to, _amount);
    // _transfer(msg.sender, _to, _amount);

    return true;
}

// 1. Es buena práctica verificar que el que envía y el que recibe no sean un address sin
// clave privada (e.g. address(0))
// 2. De la persona que se envía los tokens, debería tener la suficiente cantidad de tokens
function transferFrom(
    address _sender,
    address _recipient,
    uint256 _amount
) public {
    // verificar permiso
    uint256 allowance_ = _allowances[_sender][msg.sender];
    require(
        allowance_ >= _amount,
        "No hay permiso para manejar esta cantidad de tokens"
    );
}

// 1. Es buena práctica verificar que el que envía y el que recibe no sean un address
// sin clave privada (e.g. address(0))
require(_sender != address(0), "Sending from zero address");
require(_recipient != address(0), "Sending to zero address");

// 2. De la persona que se envía los tokens, debería tener la suficiente cantidad de
tokens
uint256 senderBalance = balances[_sender];

```

```

require(senderBalance >= _amount, "Sender does not enough tokens");

// transferir entre dos cuentas
balances[_sender] -= _amount;
balances[_recipient] += _amount;

emit Transfer(_sender, _recipient, _amount);
// _transfer(_sender, _recipient, _amount);

// disminuir la cantidad del permiso de tokens otorgados
_allowances[_sender][msg.sender] = allowance_ - _amount;
}

// Los métodos 'transfer' y 'transferFrom' repiten código que puede ser abstraído en
'_transfer', un método interno
function _transfer(
    address _sender,
    address _recipient,
    uint256 _amount
) internal {
    require(_sender != address(0), "Sending from zero address");
    require(_recipient != address(0), "Sending to zero address");

    uint256 senderBalance = balances[_sender];
    require(senderBalance >= _amount, "Sender does not enough tokens");

    balances[_sender] -= _amount;
    balances[_recipient] += _amount;

    emit Transfer(_sender, _recipient, _amount);
}

// Este método es llamado por la persona que desee dar el permiso a otra cuenta
// Es buena práctica no darle permiso a una cuenta sin llave privada (address(0))
function approve(address spender, uint256 amount) public returns (bool) {
    require(spender != address(0), "Spender is address zero");
    _allowances[msg.sender][spender] = amount;
    emit Approval(msg.sender, spender, amount);
    return true;
}

function balanceOf(address _account) public view returns (uint256) {
    return balances[_account];
}

function allowance(address _owner, address _spender)
    public
    view
    returns (uint256)
}

```

```

    {
        return _allowances[_owner][_spender];
    }
}

```

Modifiers

Supongamos que queremos proteger el acceso de ciertos métodos del Smart Contract. Una manera simple de lograr ello es validando la cuenta (address) que llama al método. Si es una cuenta (address) no conocida se interrumpe la ejecución. Veamos la manera ingenua de implementarlo seguido del uso de modifiers:

7_Modifier-1.sol

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Modifier {
    uint256 totalSupply;
    address owner = 0x5387ddeec8ddC004a217d8e172241EB5F900B302;
    mapping(address => uint256) balances;
    event Transfer(address from, address to, uint256 value);

    function _mint(address _account, uint256 _amount) internal {
        require(_account != address(0), "Mint to the zero address");

        totalSupply += _amount;
        balances[_account] += _amount;

        emit Transfer(address(0), _account, _amount);
    }

    function mintProtegido(address _to, uint256 _amount) public {
        if (msg.sender != owner) revert("No autorizado");
        _mint(_to, _amount);
    }

    function funcParaProteger1() public view {
        if (msg.sender != owner) revert("No autorizado");
    }

    function funcParaProteger2() public view {
        if (msg.sender != owner) revert("No autorizado");
    }

    function funcParaProteger3() public view {
        if (msg.sender != owner) revert("No autorizado");
    }

    modifier verificarAdmin() {
        if (msg.sender != owner) revert("No autorizado");
    }
}

```

```

    // require(msg.sender == owner, "No autorizado");
}

function mintProtegidoConModifier(address _to, uint256 _amount)
public
verificarAdmin
{
    // if (msg.sender != owner) revert("No autorizado");
    _mint(_to, _amount);
}

function funcProtegerModifier1() public view verificarAdmin {}

function funcProtegerModifier2() public view verificarAdmin {}

function funcProtegerModifier3() public view verificarAdmin {}
}

```

¿Qué es un `modifier`?

Un modificador intenta cambiar el comportamiento de la función en la cual está incluida. Por ejemplo, un `modifier` podría revisar o validar una condición antes de que se ejecute la función.

Son muy útiles porque ayudan a reducir código redundante. Se puede reutilizar el mismo `modifier` en múltiples funciones revisando las misma condiciones en todos los métodos donde se incluye.

Si al revisar/validar la condición (dentro del `modifier`) no se cumple, un error es propagado y la ejecución del método se interrumpe.

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Modificador2 {
    modifier MiModificador() {
        _;
        // cuerpo del modifier
        _;
    }

    // Se puede crear modifiers con y sin paréntesis
    modifier MiModificador2() {
        _;
    }
    modifier MiModificador3() {
        _;
    }
    // Se puede crear modifiers con y sin argumentos
    modifier MiModificadorWithArgs(uint256 a) {

```

```
    require(a >= 10, "a es menor a 10");
}
}
```

¿Qué es `_;`?

"Returns the flow of execution to the original function code" (docs)

A este símbolo se le conoce como el **comodín fusión** (merge wildcard). Fusiona el código del método con el `modifier` donde el comodín es ubicado.

En otras palabras, el cuerpo de la función donde el modificador está incluido, será insertado donde aparezca el símbolo de `_;`.

¿Dónde ubicar el `_;` dentro del `modifier`?

Puede ir al inicio, a la mitad y al final. Incluso puede repetirse varias veces.

La manera más segura de usar el patrón del modifier es poner el `_;` al final. De esta manera, el `modifier` sirve como un consistente punto de validación, revisa ciertas condiciones antes de proseguir.

7_Modifier-2

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Modifier2 {
// Ejemplo de cómo es la manera más segura dado que valida antes de proseguir
    function todoEnOrden() public pure returns (bool) {
        // ejecuta las validaciones necesarias
        return true;
    }

    function estaAutorizado() public pure returns (bool) {
        // ejecuta las validaciones necesarias
        return true;
    }

    modifier SoloSiTodoEnOrdenYAutorizado() {
        require(todoEnOrden());
        require(estaAutorizado());
        _;
    }
}
```

¿Cómo usar modifiers para pausar un contrato?

Es tan simple como utilizar un flag dentro de un modifier. Este flag tomará valores `true` o `false`. Dentro del modifier se tendrá un `require` que validará el flag. Si el flag es `false`, quiere decir que no está pausado y debería permitir que el flujo continúe. De lo contrario, si el flag está en `true`, el `modifier`, a través del `require`, debería impedir la continuación del flujo, por ende, impedir la ejecución del método donde se encuentra ese modifier.

7_Modifier-3.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Modifiers3 {
    uint256 totalSupply;
    address owner = 0x5387ddeec8ddC004a217d8e172241EB5F900B302;

    mapping(address => uint256) balances;

    event Transfer(address from, address to, uint256 value);

    function _mint(address _account, uint256 _amount) internal {
        require(_account != address(0), "Mint to the zero address");

        totalSupply += _amount;
        balances[_account] += _amount;

        emit Transfer(address(0), _account, _amount);
    }

    function mintProtegido(address _to, uint256 _amount) public {
        if (msg.sender != owner) revert("No autorizado");
        _mint(_to, _amount);
    }

    modifier onlyOwner() {
        require(owner == msg.sender, "No es el owner quien llama al contrato");
        _;
    }

    // modifier que verifica que no se encuentre en pausa los métodos
    modifier whenNotPaused() {
        require(!paused, "Los métodos estan pausados");
        _;
    }

    bool paused;

    function mintProtegidoConModifier(address _to, uint256 _amount)
        public
        onlyOwner
```

```

whenNotPaused

{
    _mint(_to, _amount);
}

function pauseMethods() public onlyOwner {
    paused = true;
}

function unpauseMethods() public onlyOwner {
    paused = false;
}

```

Constructors

Los constructores son un concepto muy conocido en Programación Orientada a Objetos. En muchos lenguajes, cuando se definen clases, también se puede definir un mágico método que solamente se ejecutará una sola vez en el momento en que una nueva instancia del objeto es creada.

En el caso de Solidity, el código definido dentro del constructor, solo se ejecutará una sola vez cuando el contrato es creado y publicado en la red.

Es una función opcional declarada. Cuando no hay constructor, el contrato asumirá un constructor por defecto que es `constructor() {}`.

Es importante mencionar que el `bytecode` publicado en la red, no contiene el código del `constructor`, dado que el constructor corre solo una vez al ser publicado.

¿Cómo se define un constructor en Solidity?

Se utiliza la palabra clave `constructor()`. No hay necesidad de añadir la palabra clave `function` dado que el constructor es una función especial.

```

contract Constructor {
    constructor() {
        // código que será ejecutado una sola vez
        // cuando se cree el contrato
    }
}

```

Inicializar el Smart Contract con valores externos

El constructor es muy útil para pasar valores de inicialización al Smart Contract. Nada impide que estos valores sean escritos en el futuro. Sin embargo, ello implicaría definir métodos adicionales de `set`.

```

contract Constructor {
    string saludo;
    constructor(string memory _saludo) {
        saludo = _saludo;
    }
}

```

¿Cómo saber quién publicó el smart contract?

Ya sabemos que `msg.sender` es una variable global en solidity que nos permite saber quién ha llamado un método en particular. Cuando usamos `msg.sender` dentro de un `constructor` en solidity, podemos saber la cuenta (address) que hizo la publicación del contrato. Veámoslo:

8_Constructor.sol

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Constructor {
    address public publicador;
    mapping(address => bool) whitelist;
    mapping(address => bool) blacklist;

    event Published();

    string saludo;

    constructor(string memory _saludo) {
        publicador = msg.sender;
        whitelist[msg.sender] = true;
        blacklist[msg.sender] = true;

        emit Published();

        // valores iniciales en un SC
        saludo = _saludo;
    }
}

```

Al publicar este contrato, la variable `publicador` obtendrá como valor el address de la persona que lo publicó. Además de ello, todas las inicializaciones dentro del `constructor`, serán ejecutadas una sola vez.

Práctica de reforzamiento

Vamos a crear un smart contract que consolide todos los conceptos hasta ahora expuestos.

1. Crear un sistema de contabilidad para una empresa que permita llevar una cuenta de todo lo gastado por cada uno de sus clientes. Para cada cuenta (address), vincular lo gastado por dicho cliente en una lista.

2. Cada cliente puede consultar la cantidad gastada hasta el momento.
3. Se creará un lista blanca de cuentas de administrador que tendrán el privilegio de actualizar la lista de cuentas. Antes de actualizar la lista de saldos, se corroborará que dicha cuenta sea parte de la lista blanca.
4. Nuevas cuentas de pueden incluir en la lista blanca de administradores. Este método también debe estar protegido para ser llamado solo por admins.
5. Cada vez que se guarda o actualiza información de la lista, se disparará un evento con información relevante a la transacción.
6. Cada usuario puede decidir incluirse en una lista negra para no monitorear sus gastos en el sistema

9_Practica.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract Practica {
    /**
     * 1. Crear un sistema de contabilidad para una empresa que permita llevar una cuenta de todo lo gastado por cada uno de sus clientes. Para cada cuenta (address), vincular lo gastado por dicho cliente en una lista.
     * 2. Cada cliente puede consultar la cantidad gastada hasta el momento.
     * 3. Se creará un lista blanca de cuentas de administrador que tendrán el privilegio de actualizar la lista de cuentas. Antes de actualizar la lista de saldos, se corroborará que dicha cuenta sea parte de la lista blanca.
     * 4. Nuevas cuentas de pueden incluir en la lista blanca de administradores. Este método también debe estar protegido para ser llamado solo por admins.
     * 5. Cada vez que se guarda o actualiza información de la lista, se disparará un evento con información relevante a la transacción.
     * 6. Cada usuario puede decidir incluirse en una lista negra para no monitorear sus gastos en el sistema
    */
    modifier onlyAdmin() {
        require(whiteList[msg.sender], "Not a registered admin");
    }

    event UpdatedBalance(address _account, uint256 _amount, uint256 _timestamp);

    // lista de saldos
    mapping(address => uint256) balances;

    // lista de admins
    mapping(address => bool) whiteList;

    // black list
    mapping(address => bool) blackList;

    constructor() {
        // Dado que el contructor se ejecuta cuando se publica el SC
    }
}
```

```

// msg.sender sería el address que también está publicando el SC
// aprovechamos para guardar el primer admin usando el constructor
whiteList[msg.sender] = true;
}

function viewSpent(address _account) external view returns (uint256) {
    return balances[_account];
}

// List blanca
function updateBalance(address _account, uint256 _amount)
    external
    onlyAdmin
{
    require(
        !blackList[msg.sender],
        "El cliente decidió no ser monitoreado"
    );

    balances[_account] += _amount;
    emit UpdatedBalance(_account, balances[_account], block.timestamp);
}

function includeInWhiteList(address _account) external onlyAdmin {
    whiteList[_account] = true;
}

// black list
function includeInBlackList() external {
    blackList[msg.sender] = true;

    // eliminar su información
    balances[msg.sender] = 0;
    // delete balances[msg.sender];
}
}

```

Herencia en Contratos

En Solidity, se puede decir que los contratos se comportan como las clases en cualquier otro lenguaje de Programación Orientada a los Objetos. Es decir, contratos pueden heredar y también pueden ser heredados. De este modo podemos construir patrones de diseño complejos.

¿Cómo se hereda en Solidity?

Se utiliza una palabra clave llamada `is` seguido del contrato. Veámoslo:

```

contract A {} // base contract
contract B is A {} // derived contract

```

Un contrato también puede heredar de múltiples contratos:

```
contract A {} // base contract
contract B {} // base contract
contract C is A, B {} // derived contract
```

Cuando un contrato hereda otros contratos, en realidad solo un único contrato es creado en el blockchain. El código de los contratos base (A en el primer ejemplo; A y B en el segundo ejemplo) es en realidad compilado en el único contrato publicado (B en el primero ejemplo; C en el segundo ejemplo).

Para poder exponer un método de un contrato base a otro contrato derivado, de manera tal que no sea usada externamente, dicho método debe definirse como interno (`internal`).

Orden para heredar

Solidity guía la herencia de contratos usando el algoritmo de linearización C3 ([c3 linearization](#)). Con éste, se posibilita la herencia múltiples propiedades y métodos de modo tal que no hacen conflicto.

En Solidity, se tienen que listar los contratos desde el contrato más base a la izquierda hasta el más derivado hacia la derecha.

```
contract Humano{}
contract Hombre is Humano{}

// contract Marcos is Hombre, Humano{} // INCORRECT
contract Marcos is Humano, Hombre {}
```

La palabra clave `super`

`super` es usado en herencia de contratos cuando se desea llamar un método definido en un nivel arriba en la jerarquía de herencia. `super.[method]` encontrará el método más cerca del smart contract que lo esté llamando.

10_InheritanceEasy.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
contract Humano {
    function saludoHumano() public pure returns (string memory) {
        return "Hola, como vamos. Soy Humano";
    }
}

contract Hombre is Humano {
    function saludoHombre() public pure returns (string memory) {
        return "Hola, como vamos. Soy Hombre";
    }
}

// Llamando al método 'saludar' del papá
```

```

function bienvenidaDeHumano() public pure returns (string memory) {
    return super.saludoHumano();
}

function bienvenidaDeHumano2() public pure returns (string memory) {
    return Humano.saludoHumano();
}

contract Marcos is Humano, Hombre {
    function saludoMarcos() public pure returns (string memory) {
        return "Hola, como vamos. Soy Marcos";
    }

    // llamando al contrato papá
    function bienvenidaDeHombre() public pure returns (string memory) {
        return super.saludoHombre();
    }

    // llamando al contrato abuelo
    function saludoDeHumano() public pure returns (string memory) {
        return super.saludoHumano();
    }

    // notar que cuando se publica el contrato Marcos
    // también se han heredado los métodos de los
    // contratos base (Humano y Hombre).
}

```

Sobrescritura de métodos (function overriding)

En herencia de contratos, los métodos también se heredan a los contratos derivados. Un contrato derivado puede sobrescribir un método ya definido en un contrato base. Para poder volver a escribir un método de un contrato base, se deben usar las palabras clave `virtual` y `override` en los métodos de la base y derivado respectivamente. Veamos:

- palabra clave `virtual` : posibilita que ese método se pueda sobrescribir en contratos derivados
- palabra clave `override` : especifica que dicha función está siendo sobrescrita en ese mismo lugar

Herencia y métodos en contratos

10_Inheritance.sol

```

contract Humano {
    function saludar() public pure virtual returns (string memory) {
        return "Hola, como vamos. Soy Humano";
    }
}

contract Hombre is Humano {

```

```

function saludar()
    public
    pure
    virtual
    override(Humano)
    returns (string memory)
{
    return "Hola, como vamos. Soy Hombre";
}

// Llamando al método 'saludar' del papá
function bienvenidaDeHumano() public pure returns (string memory) {
    return super.saludar();
}

function bienvenidaDeHumano2() public pure returns (string memory) {
    return Humano.saludar();
}

contract Marcos is Humano, Hombre {
    function saludar()
        public
        pure
        override(Humano, Hombre)
        returns (string memory)
    {
        return "Hola, como vamos. Soy Marcos";
    }
}

```

Herencia y constructores en contratos

Cuando se heredan contratos, los contratos base pueden tener argumentos que se requieren en su constructor para poder inicializarlos. En dichos casos, el contrato derivado (hijo), tiene que ejecutar el constructor del contrato base.

¿Cómo inicializar el constructor del contrato Base?

1. Directamente en la lista de herencia de contratos
2. A través de un "modifier" para el `constructor` del contrato derivado

10_InheritanceContructors.sol

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Humano {
    string description;
    string origin;
}

```

```

        uint256 year; // no existe year.toString();

constructor(
    string memory _description,
    string memory _origin,
    uint256 _year
) {
    description = _description;
    origin = _origin;
    year = _year;
}

function saludo() public view virtual returns (string memory) {
    return
        string(
            abi.encodePacked(
                "I'm a ",
                description,
                " from ",
                origin,
                ". How are you?"
            )
        );
}

/**
 * 1
 * Directamente en la lista de herencia
 * - cuando sabes los argumentos a priori
 */
contract Hombre is Humano("Homo Sapiens", "Earth", 2022) {
    function getDescription() public view returns (string memory) {
        return description;
    }

    function getOrigin() public view returns (string memory) {
        return origin;
    }

    function getYear() public view returns (uint256) {
        return year;
    }

    function saludo() public pure override returns (string memory) {
        return "Hola, soy Hombre";
    }

    function saludoPapa() public view returns (string memory) {

```

```

        return super.saludo();
    }
}

/***
 * 2
 * Como si fuera un modifier del constructor
 * - si deseas tener mayor control sobre los argumentos del constructor
 */
contract HombreV2 is Humano {
    constructor(
        string memory _description,
        string memory _origin,
        uint256 _year
    ) Humano(_description, _origin, _year) {
        // esto puede estar vacío
        // lo único que importa es pasar los argumentos al contrato Base
    }
}

/***
 * Herencia múltiple
 * Hombre V3 no hereda Humano
 */
contract HombreV3 {
    uint256 height;

    constructor(uint256 _height) {
        height = _height;
    }
}

contract Programmador is HombreV3, Humano {
    constructor(
        string memory _description,
        string memory _origin,
        uint256 _year,
        uint256 _height
    ) HombreV3(_height) Humano(_description, _origin, _year) {}
}

/***
 * Herencia múltiple
 * Hombre V4 hereda Humano
 */
contract HombreV4 is Humano {
    uint256 height;
}

```

```

constructor(
    string memory _description,
    string memory _origin,
    uint256 _year,
    uint256 _height
) Humano(_description, _origin, _year) {
    height = _height;
}

contract Programmado2 is Humano, HombreV4 {
    constructor(
        string memory _description,
        string memory _origin,
        uint256 _year,
        uint256 _height
    ) HombreV4(_description, _origin, _year, _height) {}
}

```

Herencia y modifiers

Los modifiers pueden ser aislados en otros contratos y reutilizados de la misma manera en que los métodos se reciclan de los contratos base.

10_InheritanceModfrs.sol

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Owner {
    address owner;

    modifier onlyOwner() {
        require(owner == msg.sender, "No es el owner quien llama al contrato");
        _;
    }

    constructor() {
        owner = msg.sender;
    }
}

contract Paused is Owner {
    bool paused;

    modifier whenNotPaused() {
        require(!paused, "Los metodos estan pausados");
        _;
    }
}

```

```

function pauseMethods() public onlyOwner {
    paused = true;
}

function unpauseMethods() public onlyOwner {
    paused = false;
}

contract InheritanceModfrs is Owner, Paused {
    uint256 totalSupply;
    mapping(address => uint256) balances;
    event Transfer(address from, address to, uint256 value);

    function _mint(address _account, uint256 _amount) internal {
        require(_account != address(0), "Mint to the zero address");
        totalSupply += _amount;
        balances[_account] += _amount;
        emit Transfer(address(0), _account, _amount);
    }

    function mintProtegidoConModifier(address _to, uint256 _amount)
        public
        onlyOwner
        whenNotPaused
    {
        _mint(_to, _amount);
    }
}

```

Especificadores de visibilidad en métodos

Los métodos en Solidity tienen diferentes niveles de exposición y de herencia. La capacidad de tener diferentes niveles de visibilidad, facilita el uso de patrones de diseño en los smart contract.

En solidity existen cuatro tipos de visibilidad y son `public`, `private`, `internal` y `external`. Cada uno de ellas tiene efectos diferentes que se listan a continuación:

`public`

- puede ser consumido internamente dentro del mismo contrato por otros métodos
- otros Smart Contract y externally owned accounts podrán llamar al método
- puede ser heredado en contratos derivados y ser usado desde contratos derivados
- este método será incluido en el ABI generado al compilar el smart contract
- puede ser sobreescrito en contratos derivados

`internal`

- puede ser consumido internamente dentro del mismo contrato por otros métodos
- puede ser heredado en contratos derivados y ser usado desde contratos derivados
- otros Smart Contract y externally owned accounts no podrán llamar al método

- este método será no incluido en el ABI generado al compilar el smart contract.
- puede ser sobreescrito en contratos derivados

`private`

- puede ser consumido internamente dentro del mismo contrato por otros métodos
- no heredado en contratos derivados ni ser usado desde contratos derivados
- otros Smart Contract y externally owned accounts no podrán llamar al método
- este método será no incluido en el ABI generado al compilar el smart contract
- no sobreescrito en contratos derivados

`external`

- no consumido internamente dentro del mismo contrato por otros métodos
- no heredado en contratos derivados ni ser usado desde contratos derivados
- otros Smart Contract y externally owned accounts podrán llamar al método
- este método será incluido en el ABI generado al compilar el smart contract
- puede ser sobreescrito en contratos derivados

Esta tabla resume lo descrito anteriormente:

| Visibility | Contrato | Derivado | SC/EOA | ABI | Override |
|-----------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| <code>public</code> | <code>true</code> | <code>true</code> | <code>true</code> | <code>true</code> | <code>true</code> |
| <code>internal</code> | <code>true</code> | <code>true</code> | <code>false</code> | <code>false</code> | <code>true</code> |
| <code>private</code> | <code>true</code> | <code>false</code> | <code>false</code> | <code>false</code> | <code>false</code> |
| <code>external</code> | <code>false</code> | <code>true</code> | <code>true</code> | <code>true</code> | <code>true</code> |

11_visibilityMethods.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract A {
    //1
    // 'public'
    // llamado por EOA y SC
    // heredable
    // de uso interno en el contrato
    // ABI
    // sobreescribible
    function funcHeredablePublicaEInterna() public virtual {}

    // 2
    // 'internal'
    // heredable
    // de uso interno en el contrato y en derivados
    // sobreescribible
    function funcHeredableEInterna() internal virtual {}
}
```

```

// 3
// 'private'
// de uso interno en el contrato únicamente
// no sobreescribible: no soporta 'virtual'
function funcInterna() private {}

// 4
// 'external'
// llamado por EOA y SC
// heredable
// sobreescribible
function funcExternaYHeredable() external {}

contract B is A {
    // 1
    // function funcHeredablePublicaEIInterna heredada en B
    // 2
    // function funcHeredableEIInterna heredada en B
    // 3
    // function funcInterna NO heredada en B
    // 4
    // function funcExternaYHeredable heredada en B
}

```

Modificadores de métodos

Los métodos en Solidity poseen varios modificadores que les añaden o limitan ciertas propiedades o capacidades. Lista de modificadores en métodos: `pure`, `view`, `payable`, `virtual` y `override`.

`virtual` y `override` fueron vistos en sobreescritura de métodos. Ahora veremos los modificadores `view` y `pure`. Luego, `payable` será estudiado para realizar transferencias de monedas nativas (`Ether`).

`view`

- Puede leer del `storage`. Es decir, puede leer cualquier información de variables que a su vez se guardan en el smart contract.
- Por lo general los métodos `view` son usados para crear `getters` de información.
- No consume gas siempre y cuando (1) sea llamado externamente o (2) es llamado internamente por otro método `view`.
- Consume gas cuando un método no `view` de un contrato llama la función `view`. Dado que la función que llama genera una transacción (porque no es `view`), el cálculo/proceso interno que se hace dentro del método `view` será incluido con su costo de gas correspondiente.

`pure`

- no pueden leer del `storage`
- no pueden modificar variables del `storage`
- No consume gas siempre y cuando (1) sea llamado externamente o (2) es llamado internamente por

otro método `view` o `pure`.

- Consume gas cuando un método no `view` de un contrato llama la función `pure`.

12_viewPure.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Pure {
    event Number(uint256 number);

    function functionPure() public pure returns (uint256) {
        return 3;
    }

    function functionPure2(uint256 _a) internal pure returns (uint256) {
        // emit Number(_a); // no se puede disparar un evento
        return _a * 10**3;
    }

    function functionPure3() external pure returns (string memory) {
        return "Function pura";
    }
}

contract View {
    uint256 a = 12314324235435346;
    event Number(uint256 number);

    function functionView() public view returns (uint256) {
        // emit Number(a); // no se puede disparar un evento
        return a;
    }

    function functionView2() internal view returns (uint256) {}

    function functionView3() external view returns (uint256) {}
}
```

Mecanismo de roles para acceso

Usando modifiers hemos logrado proteger métodos para que sean ejecutados solamente por cuentas que son `onlyAdmin`. Sin embargo, usar `onlyAdmin` es muy restrictivo cuando se quiere crear diferentes privilegios para diferentes métodos. Es decir, que una cuenta puede ejecutar el método A, pero no el método B.

Hay otra manera de lograr un esquema de roles con privilegios más complejos. El resultado final debería lucir así:

13_accessRoles.sol

```

contract AccessRoles {
    function funcParaSoloRoleA() public onlyRole(ONLY_ROLE_A) {}

    function funcParaSoloRoleB() public onlyRole(ONLY_ROLE_B) {}

    function funcParaSoloRoleC() public onlyRole(ONLY_ROLE_C) {}
}

```

De aquí se infiere que una `address` en particular a quien se le ha asignado el `ONLY_ROLE_A`, es la única cuenta que tiene el privilegio de llamar al método `funcParaSoloRoleA()`.

Con una algoritmo de otorgar roles más complejos, podemos crear la siguiente table de roles y addresses.

| | MINTER | BURNER | PAUSER |
|-----------|---------------|---------------|---------------|
| Account 1 | True | True | True |
| Account 2 | True | False | True |
| Account 3 | False | False | True |

14_accessRoles.sol

```

contract AccessRoles {
    /**
     *          | MINTER | BURNER | PAUSER |
     * ----- | ----- | ----- | ----- |
     * Account 1 | True  | True  | True  |
     * Account 2 | True  | False | True  |
     * Account 3 | False | False | True  |

    1. definir un mapping doble para guardar una matriz de información
    mapping 1 -> address => role
    mapping 2 -> role => boolean
    mapping(address => mapping(bytes32 => bool)) roles;

    2. definir metodo de lectura de datos de la matriz
        hasRole

    3. definir método para escribir datos en la matriz
        grantRole
        mapping[accout 1][MINTER] = true
        mapping[accout 1][BURNER] = true
        mapping[accout 1][PAUSER] = true

        mapping[accout 2][MINTER] = true
        mapping[accout 2][PAUSER] = true

```

```

mapping[accout 3][PAUSER] = true

4. crear modifier que verifica el acceso de los roles

5. utilizar el constructor para inicializar valores
*/

```

```

bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

mapping(address => mapping(bytes32 => bool)) roles;

constructor() {
    roles[msg.sender][DEFAULT_ADMIN_ROLE] = true;
}

modifier onlyRole(bytes32 _role) {
    require(roles[msg.sender][_role], "Account has no role/privilege");
    _;
}

function grantRole(address _account, bytes32 _role)
    public
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    roles[_account][_role] = true;
}

function revokeRole(address _account, bytes32 _role)
    public
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    roles[_account][_role] = false;
}

function hasRole(address _account, bytes32 _role)
    public
    view
    returns (bool)
{
    return roles[_account][_role];
}
}

```

Optimización ERC20

Con todo lo visto hasta el momento, ha llegado el momento de optimizar una vez más el código del contrato ERC20 que se trabajo anteriormente en `6_ERC20-2.sol`.

`15_ERC20-3.sol`

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

// 2. convertir en Template este contrato
contract ERC20Template {
    string nameCrypto;
    string symbolCrypto;

    uint256 totalSupply;

    mapping(address => uint256) balances;
    mapping(address => mapping(address => uint256)) _allowances;

    event Transfer(address from, address to, uint256 amount);
    event Approval(address owner, address spender, uint256 amount);

    // 1. Definir constructor
    constructor(string memory _nameCrypto, string memory _symbolCrypto) {
        nameCrypto = _nameCrypto;
        symbolCrypto = _symbolCrypto;
    }

    function name() public view returns (string memory) {
        return nameCrypto;
    }

    function symbol() public view returns (string memory) {
        return symbolCrypto;
    }

    function decimals() public pure returns (uint256) {
        return 18;
    }

    function _mint(address _account, uint256 _amount) internal {
        require(_account != address(0), "Acuniando a address 0");

        balances[_account] += _amount;
        totalSupply += _amount;

        emit Transfer(address(0), _account, _amount);
    }

    function burn(uint256 _amount) public {
        _burn(msg.sender, _amount);
    }

    function _burn(address _account, uint256 _amount) internal {
        require(_account != address(0), "Quemando de address 0");
    }
}
```

```

// uint256 miBalance = balances[_account];
uint256 miBalance = balanceOf(_account);
require(miBalance >= _amount, "Insuficientes tokens para quemar");

balances[_account] -= _amount;
totalSupply -= _amount;

emit Transfer(_account, address(0), _amount);
}

// Se transfieren MIS TOKENS
function transfer(address _to, uint256 _amount) public returns (bool) {
    return _transfer(msg.sender, _to, _amount);
}

function transferFrom(
    address _from,
    address _to,
    uint256 _amount
) public {
    uint256 permisoParaGastar = allowances[_from][msg.sender];
    require(permisoParaGastar >= _amount, "No tengo suficiente permiso");

    _transfer(_from, _to, _amount);

    allowances[_from][msg.sender] = permisoParaGastar - _amount;
}

function _transfer(
    address _from,
    address _to,
    uint256 _amount
) internal returns (bool) {
    require(_from != address(0), "Enviado desde address zero");
    require(_to != address(0), "Enviado a address zero");

    uint256 balanceFrom = balanceOf(_from);
    require(balanceFrom >= _amount, "Insuficientes tokens");

    balances[_from] -= _amount;
    balances[_to] += _amount;

    emit Transfer(_from, _to, _amount);
    return true;
}

function approve(address _spender, uint256 _amount) public {
    require(_spender != address(0), "Spender es address zero");
}

```

```

        _allowances[msg.sender][_spender] = _amount;
        emit Approval(msg.sender, _spender, _amount);
    }

function balanceOf(address _account) public view returns (uint256) {
    return balances[_account];
}

function allowance(address _owner, address _spender)
public
view
returns (uint256)
{
    return _allowances[_owner][_spender];
}
}

// 7. Pasar el modifier a otro contrato
contract Protegido {
    address _owner;

    constructor() {
        _owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == _owner, "No es el owner");
        _;
    }
}

// 10. Pausable
contract Pausable is Protegido {
    bool paused;

    modifier whenNotPaused() {
        require(!paused, "Contrato pausado");
        _;
    }

    modifier whenPaused() {
        require(paused, "Contrato no pausado");
        _;
    }

    function pause() public onlyOwner {
        paused = true;
    }

    function unpause() public onlyOwner {

```

```

        paused = false;
    }
}

// 3. Heredar el contrato
// 8. Heredar 'Protegido'
contract MiPrimerToken is ERC20Template, Protegido, Pausable {
    // address owner;

    // 6. Create modifier
    // modifier onlyOwner() {
    //     require(msg.sender == owner, "No es el owner");
    //     _;
    // }

    // 4. Definiendo el constructor
    // constructor() ERC20Template("Mi Primer Token", "MPRTK") {
    constructor(string memory _nameCrypto, string memory _symbolCrypto)
        ERC20Template(_nameCrypto, _symbolCrypto)
    {
        // 9. eliminar owner aqui
        // owner = msg.sender;
    }

    // 5. Exponiendo y protegiendo mint
    function mint(address _account, uint256 _amount)
        public
        onlyOwner
        whenNotPaused
    {
        _mint(_account, _amount);
    }
}

// en caso quisiéramos utilizar el template en otro contrato
contract MiPrimerToken2 is ERC20Template, Protegido {
    constructor() ERC20Template("Mi Primer Token2", "MPRTK2") {}

    function funParaProteger() public view onlyOwner {}

    function funParaProteger2() public view onlyOwner {}

    function funParaProteger3() public view onlyOwner {}
}

```

Introducción a Hardhat

Hardhat y configuración de pipeline

Requisito: Tener una versión de Node.js superior al 14

Comenzaremos con la creación de un proyecto Hardhat desde cero. Crear una carpeta nueva y continuar con la instalación descrita a continuación:

1. En el terminal, ejecutar `npm init -y` para instalar el `package.json` del repositorio
2. En el terminal, ejecutar `npx hardhat` para comenzar la instalación. Al hacerlo, preguntará lo siguiente:

```
Need to install the following packages:  
  hardhat  
Ok to proceed? (y)
```

Tipear `y` y luego Enter. Al hacerlo, aparecerá el siguiente mensaje:

```
888 888          888 888          888  
888 888          888 888          888  
888 888          888 888          888  
888888888888 8888b. 888d888 .d888888 888888b. 8888b. 8888888  
888 888 "88b 888P" d88" 888 888 "88b "88b 888  
888 888 .d8888888 888 888 888 888 .d8888888 888  
888 888 888 888 Y88b 888 888 888 888 888 Y88b.  
888 888 "Y8888888 888 "Y888888 888 "Y8888888 "Y888  
  
👷 Welcome to Hardhat v2.12.0 🎊 •  
  
? What do you want to do? ...  
❯ Create a JavaScript project  
  Create a TypeScript project  
  Create an empty hardhat.config.js  
  Quit
```

Escogemos la opción `Create a JavaScript project` con el teclado y luego presionamos Enter. Para continuar con el set up, darle a todo Enter.

```
✓ What do you want to do? • Create a JavaScript project  
✓ Hardhat project root: • /Users/steveleec/Documents/UTEC/solidity-utec-coding  
✓ Do you want to add a .gitignore? (Y/n) • y  
✓ Do you want to install this sample project's dependencies with npm (hardhat  
@nomicfoundation/hardhat-toolbox)? (Y/n) • y
```

Al finalizar la instalación, obtendremos el siguiente mensaje:

```
✨ Project created ✨
```

Con todo esto, tendremos el cascarón de un entorno de desarrollo usando hardhat. Continuemos con la configuración

3. Instalar librería npm dotenv mediante `npm install --save dotenv`. Seguido a ello creamos un archivo llamado `.env` con el comando `touch .env` ejecutado en el terminal. Replicar el siguiente modelo dentro del archivo `.env`:

```
ADMIN_ACCOUNT_PRIVATE_KEY= es la llave privada obtenida de metamask
GOERLI_TESNET_URL= URL de servicios de Alchemy
MUMBAI_TESNET_URL= URL de servicios de Alchemy
ETHERSCAN_API_KEY= API KEY obtenida de Etherscan
POLYGONSCAN_API_KEY= API KEY obtenida de Polygonscan
```

4. Instalar librería de contratos de Open Zeppelin mediante el comando `npm install --save @openzeppelin/contracts`. De esta librería reutilizaremos código ya testeado y validado.
5. Instalamos la librería Chai para poder extender las funcionalidad de testing de Hardhat `npm install -save-dev chai`. Esto sería una dependencia en `devDependencies`.
6. Instalamos `@nomiclabs/hardhat-etherscan` para poder verificar los contratos mediante scripts desde Hardhat: `npm install --save-dev @nomiclabs/hardhat-etherscan`.
7. Actualizamos el archivo `hardhat.config.js` y debería contener lo siguiente:

```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config();

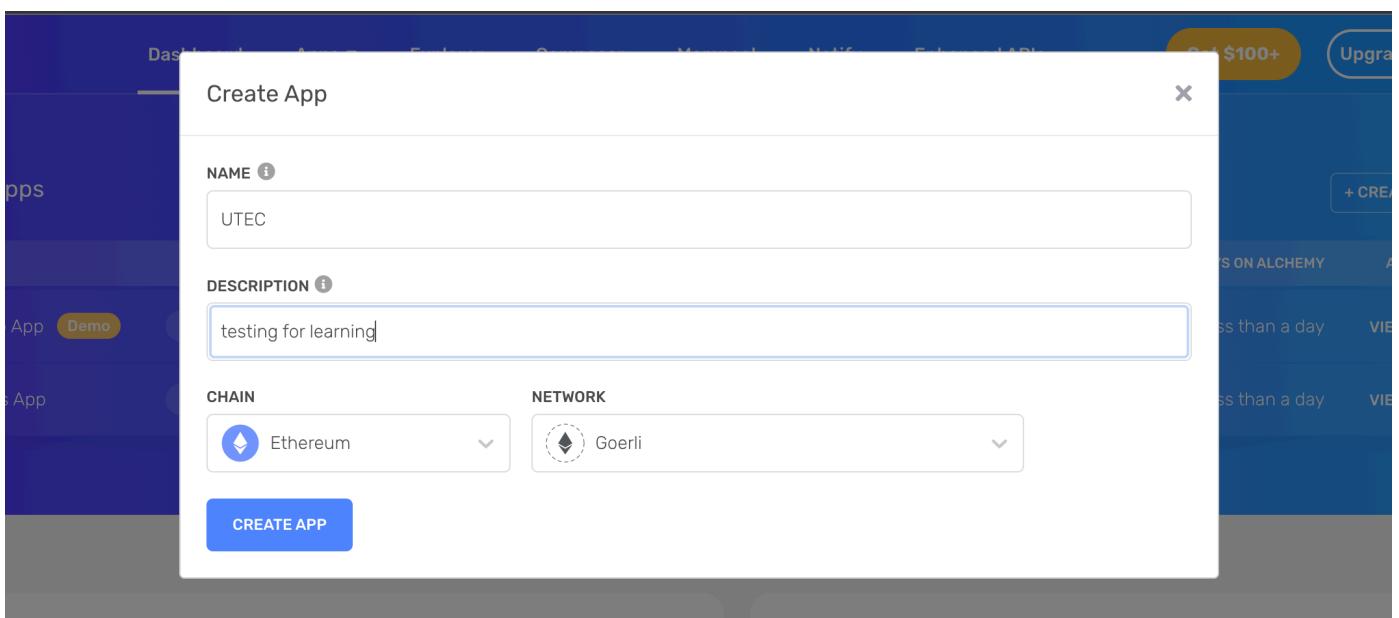
/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.9",
  networks: {
    localhost: {
      url: "HTTP://127.0.0.1:8545",
      timeout: 800000,
      gas: "auto",
      gasPrice: "auto",
    },
    goerli: {
      url: process.env.GOERLI_TESNET_URL,
      accounts: [process.env.ADMIN_ACCOUNT_PRIVATE_KEY],
      timeout: 0,
      gas: "auto",
      gasPrice: "auto",
    },
    matic: {
      url: process.env.MUMBAI_TESNET_URL,
      // url: "https://matic-mumbai.chainstacklabs.com",
      accounts: [process.env.ADMIN_ACCOUNT_PRIVATE_KEY],
      timeout: 0,
      gas: "auto",
      gasPrice: "auto",
    },
  },
},
```

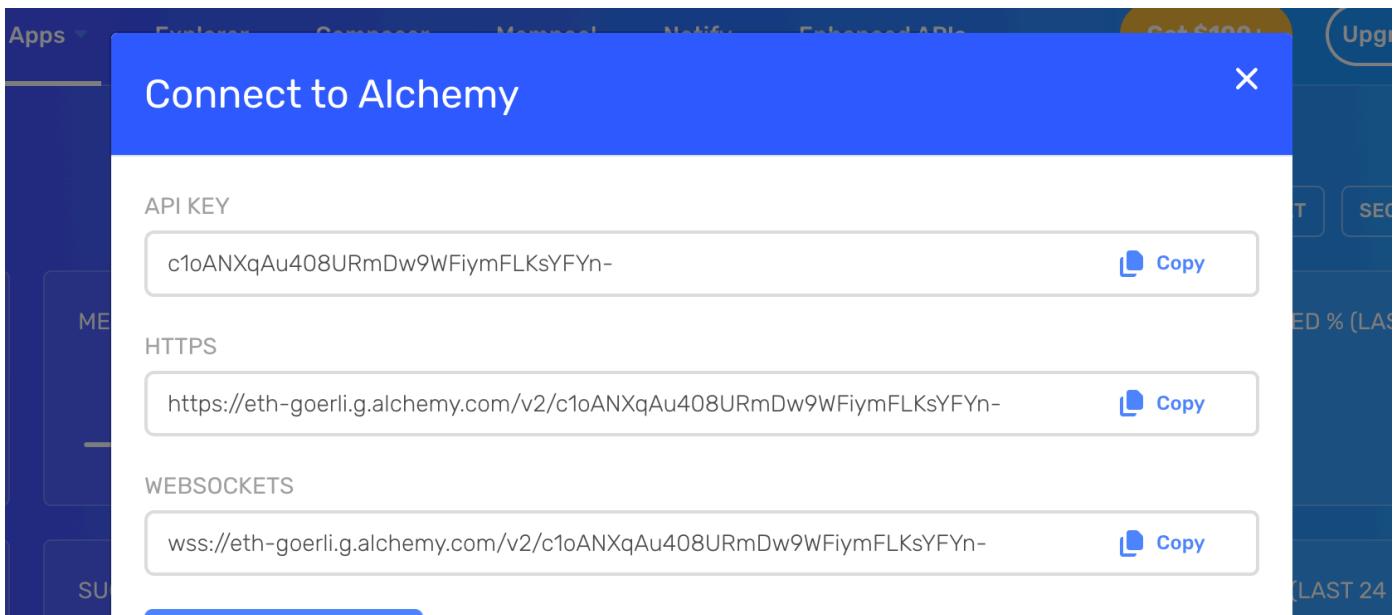
```
// etherscan: { apiKey: process.env.ETHERSCAN_API_KEY },
etherscan: { apiKey: process.env.POLYGONSCAN_API_KEY },
};
```

- `localhost`, `goerli` y `matic` son las redes que hardhat utilizará para poder publicar los contratos inteligentes.
- `url` es uno de los urls usados para poder conectarse a algún nodo privado. En la actualidad existen muchos servicios de conexión. En este caso en particular usaremos `Alchemy`.
- `accounts` es un array que contiene todas las llaves privadas de los address que serán usados para publicar los contratos
- `etherscan` hace referencia a una key obtenida en el explorador de bloques de cada blockchain (usualmente en mainnet) que permite hacer la verificación de smart contracts de manera automática

8. Rellenar las claves del archivo `.env`:

- `ETHERSCAN_API_KEY`: Dirigirte a [Etherscan](#). Click en `Sign in`. Click en `Click to sign up` y terminar de crear la cuenta en Etherscan. Luego de crear la cuenta ingresar con tus credenciales. Dirigirte a la columna de la derecha. Buscar `OTHER > API Keys`. Crear un nuevo api key haciendo click en `+ Add` ubicado en la esquina superior derecha. Darle nombre al proyecto y click en `Create New API Key`. Copiar el `API Key Token` dentro del archivo `.env`.
- `POLYGONSCAN_API_KEY`: Repetir el anterio paso para [Polygonscan](#)
- `ADMIN_ACCOUNT_PRIVATE_KEY`: Obtener el `private key` de la wallet que se creó en el punto [2](#) siguiendo [estos pasos](#) y copiarlo en esta variable en el archivo `.env`.
- `GOERLI_TESNET_URL`: Crear una cuenta en [Alchemy](#). Ingresar al dashboard y crear una app `+ CREATE APP`. Escoger `NAME` y `DESCRIPTION` cualquiera. Escoger `ENVIRONMENT = Development`, `CHAIN = Ethereum` y `NETWORK = Goerli`. Hacer click en `VIEW KEY` y copiar el link `HTTPS` en el documento `.env` para esta variable de entorno. Saltar el paso que te pide incluir tarjeta de débito.
- `POLYGONSCAN_API_KEY`: Repetir el paso anterior en Alchemy para `CHAIN = Polygon` y `NETWORK = Mumbai`.





Project Setup

Para comenzar un proyecto con la configuración inicial, partir de la branch `setup` mediante el siguiente comando: `npm checkout setup`. Allí hacer `npm install` desde el terminal. Desde aquí empezaremos a desarrollar smart contracts en Hardhat.

Hardhat: Publicando Smart Contracts

1. Crear el archivo `MiPrimerToken.sol` dentro de la carpeta `contracts`. Aquí pegamos el código de nuestro primer token que tomamos del [wizard](#):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import "@openzeppelin/contracts/security/Pausable.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";

contract MiPrimerToken is ERC20, ERC20Burnable, Pausable, AccessControl {
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");

    constructor(string memory _name, string memory _symbol)
        ERC20(_name, _symbol)
    {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(PAUSER_ROLE, msg.sender);
        _grantRole(MINTER_ROLE, msg.sender);
    }

    function pause() public onlyRole(PAUSER_ROLE) {
        _pause();
    }
}
```

```

    }

    function unpause() public onlyRole(PAUSER_ROLE) {
        _unpause();
    }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }

    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal override whenNotPaused {
        super._beforeTokenTransfer(from, to, amount);
    }
}

```

2. Crear el archivo `deploy.js` dentro de la carpeta `scripts`.

```

const hre = require("hardhat");

async function main() {
    const MiPrimerToken = await hre.ethers.getContractFactory("MiPrimerToken");
    const miPrimerToken = await MiPrimerToken.deploy(
        "Mi Primer Token",
        "MPRMTKN"
    );

    var tx = await miPrimerToken.deployed();
    await tx.deployTransaction.wait(5);

    console.log(`Deploy at ${miPrimerToken.address}`);

    await hre.run("verify:verify", {
        address: miPrimerToken.address,
        constructorArguments: ["Mi Primer Token", "MPRMTKN"],
    });
}

// We recommend this pattern to be able to use async/await everywhere
// and properly handle errors.
main().catch((error) => {
    console.error(error);
    process.exitCode = 1;
});

```

3. Crear el archivo `testToken.js` dentro de la carpeta `test`

```
const { expect } = require("chai");

describe("MI PRIMER TOKEN TESTING", function () {
  var MiPrimerToken, miPrimerToken;

  describe("Set up", () => {
    it("Deploys correctly", async () => {
      MiPrimerToken = await hre.ethers.getContractFactory("MiPrimerToken");
      miPrimerToken = await MiPrimerToken.deploy("Mi Primer Token", "MPRMTKN");

      await miPrimerToken.deployed();
    });
  });

  describe("Name y Symbol", () => {
    it("Retrieves correct token name", async () => {
      var tokenName = "Mi Primer Token";
      expect(await miPrimerToken.name()).to.be.equal(tokenName);
    });

    it("Retrieves correct token symbol", async () => {
      var tokenSymbol = "MPRMTKN";
      expect(await miPrimerToken.symbol()).to.be.equal(tokenSymbol);
    });
  });
});
```

Hasta aquí tenemos el script de deployment de los smart contracts, así como también el archivo de testing donde veremos una pequeña introducción al testing.

Comandos en Hardhat para publicar y testear

- `npx hardhat compile`: compila los smart contracts y verifica si hay algún error
- `npx hardhat clear`: limpia caché (artifacts y cache). Ayuda a solucionar errores desconocidos en el deployment
- `npx hardhat --network nombreDeLaNetwork verify seguidoDeAddres SeguidoArgsSOptional`: verifica un contrato con argumentos en el constructor
- `npx hardhat test test/testToken.js`: corre los tests definidos en el archivo `testToken.js`
- `npx hardhat run scripts/deploy.js`: Publicará el contrato para el blockchain local que Hardhat ejecuta
- `npx hardhat --network matic run scripts/deploy.js`: A diferencia del anterior comando, en este caso la red es testnet (o Mainnet) y nos permite publicar a testnets o mainnets dependiendo del argumento `--network matic`.

Publicando Smart Contracts

1. Correr el comando `npx hardhat --network matic run scripts/deploy.js` para publicar en Testnet El resultado que obtendríamos sería el siguiente:

```
Deploy at 0x959D7dCad2B90fC42c54d838f3d43cf06cbBBd60
```

2. Automáticamente el script empezará con la verificación del mismo y nos avisará cuando esté listo

Para llegar a este lugar, podemos usar el branch `scPublicado` con el comando `git checkout scPublicado` y tendremos el código para publicar y verificar.

Interactuando con Smart Contracts

Dentro de Hardhat, se puede crear una conexión entre un smart contract publicado y su address para poder ejecutar o leer métodos de manera programática. Para lograr que este sea posible, se requiere de una conexión a un node y para ello Alchemy

Abrir una nueva ventana en el terminal. Tipear `npx hardhat console` o, si deseamos incluir una red en particular, podemos especificarlo así: `npx hardhat --network mumbai console`. De este modo, Hardhat reconocerá la configuración que se tiene para esta red en el archivo `hardhat.config.js`. Es decir, tomará el `url` y `accounts` en consideración.

```
var addressSC = "0x959D7dCad2B90fC42c54d838f3d43cf06cbBBd60";
var MiPrimerToken = await ethers.getContractFactory("MiPrimerToken")
var miPrimerToken = await MiPrimerToken.attach(addressSC);

// evaluan
await miPrimerToken.name() // Mi Primer Token
await miPrimerToken.symbol() // MPTK
await miPrimerToken.totalSupply() // 0
```

Troubleshooting in deployment

- El archivo `.env` no tiene las claves correctas
- La llave privada de la billetera de Metamask no cuenta con los fondos suficientes
- NodedeJS es una versión antigua

Interfaces

¿Qué es una Interface?

Es común en varios lenguajes de programación. Su objetivo es separar la definición de una funcionalidad del actual comportamiento de la funcionalidad.

Una interface no se enfoca en el proceso o en el comportamiento sino en el resultado/objetivo; se enfoca en lo que puede hacer.

¿Qué son las interfaces de los Smart Contracts?

Las interfaces en los contratos inteligentes son como su esqueleto. Ayuda a definir las funcionalidades del contrato y cómo interactuar con ellas.

Al tener la interfaz de un contrato inteligente definida, dApps u otros smart contracts podrán saber cómo comunicarse con cualquier smart contract.

En otras palabras, el definir interfaces contribuye a tener un .estándar.

Dos maneras de usar una interface

1. Enforcer - Al heredar una interface en un Smart Contract, la interface forzará a que dicho smart contract implemente los métodos definidos en la interface. De faltar algún método definido en la interfaz pero no desarrollado en el contrato, no se podrá compilar

```
interface IEnforcer {
    function balance(address _account) external returns(uint256);
}

contract A is IEnforcer {
    function balance(address _account) public view returns(uint256) {
        // ... function with body
    }
}
```

2. Door - Permite comunicarte con otro Smart contract siguiendo las definiciones de sus métodos en la interfaz. En este caso no es necesario la herencia de la interface. Esta interface podrá ser tratada como un método más para poder acceder a sus definiciones.

```
interface IDoor {
    function balance(address _account) external returns(uint256);
}

contract B {
    address _doorAddress;
    function getBalanceOfAnotherContract(address _account) public view returns(uint256) {
        return IDoor(_doorAddress).balance(_account);
    }
}
```

¿Cómo se definen las interfaces?

```
interface IEnforcer {
    function balance(address _account) external returns(uint256);
}
```

- Una interface se define con la palabra clave `interface`
- Cada definición de función incluye el nombre de la función, tipos de parámetros, y tipos de valores de retorno
- En interfaces, las funciones terminan en `;`, mientras que en los smart contracts, terminan en `{}`.
- Métodos en interfaces pueden ser tanto de lectura y escritura
- Los métodos dentro de una interface deben ser declarados como `external`
- La convención es comenzar cada interface con la letra I, seguido del nombre del contrato (e.g. `IERC20`,

IERC1155)

Interfaces y herencia

Interfaces pueden heredar otras interfaces pero no pueden heredar otros contratos. Las mismas reglas de herencia para contratos se aplican a interfaces.

16_interfacesInhe.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

interface IA {
    function balanceOf(address _account) external returns (uint256);
}

interface IB is IA {
    function totalSupply() external returns (uint256);
}

// contract Token is IA, IB {
contract Token is IB {
    function totalSupply() external virtual returns (uint256) {}

    function balanceOf(address _account) external virtual returns (uint256) {}
}
```

Función overloading y Eventos en interfaces

Las interfaces soportan la definición de métodos con el mismo nombre pero que tengan diferentes argumentos (y posiblemente diferentes valores de retorno). A esta característica se llama **función overloading**. En estas situaciones, el contrato que hereda una interface con función overloading tendrá que implementar ambos métodos con sus diferentes argumentos.

También es posible definir los eventos dentro de la misma interface para luego heredárselos. Veamos cómo funcionan:

17_interfaceOverloadinEven.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

interface IERC20 {
    event Transfer(address _account);
    event Transfer2(address _accountOne, address _accountTwo);

    function balanceOf(address _account) external returns (uint256);

    function balanceOf(address _accountOne, address _accountTwo)
        external
        returns (uint256, uint256);
```

```

}

contract Token is IERC20 {
    function balanceOf(address _account) external returns (uint256) {
        emit Transfer(_account);
        return 3;
    }

    function balanceOf(address _accountOne, address _accountTwo)
        external
        returns (uint256, uint256)
    {
        emit Transfer2(_accountOne, _accountTwo);
        return (3, 5);
    }
}

```

Llamadas intercontratos

Veamos un ejemplo de cómo un contrato A interactúa con un contrato B para leer y escribir sobre él. Por lo general, se siguen estos pasos:

1. Definamos un contrato con el cual se quiere interactuar
2. Creamos la interface necesaria para interactuar
3. Capturamos la dirección del Smart Contract con el cual se desea interactuar
4. Creamos una "puerta" con la `interface` y `address` del contrato

Desarrollo:

```

// 1
contract ContratoParaSerLeidoYEjecutado is AccessRoles {
    bytes32 public constant SMART_C_ROLE = keccak256("SMART_C_ROLE");

    uint256 _totalBalance = 10**6 * 10**18;

    function totalBalance() public view returns (uint256) {
        return _totalBalance;
    }

    function funcWithPrivilege() public onlyRole(SMART_C_ROLE) {
        _totalBalance += _totalBalance;
    }
}

```

```
// 2
interface ICParaSerLeidoYEjecutado {
    function totalBalance() external returns (uint256);

    function funcWithPrivilege() external;
}
```

```
// 3 y 4
contract ContratoQueLeeYEjecuta {
    address cParaLeerAddress = address(0);

    // Manera 1: definir la interface como variable: "instanciar"
    IContParaLeerYEjec contratoParaLeerYEj =
        IContParaLeerYEjec(cParaLeerAddress);

    function leerBalanceDeContratoExterno() public returns (uint256) {
        return contratoParaLeerYEj.totalBalance();
    }

    function ejecutarFuncConPrivilegioExterno() public {
        contratoParaLeerYEj.funcWithPrivilege();
    }

    // Manera 2: utilizar la misma interface como function con propiedades
    function leerBalanceDeContratoExterno2() public returns (uint256) {
        return IContParaLeerYEjec(cParaLeerAddress).totalBalance();
    }

    function ejecutarFuncConPrivilegioExterno2() public {
        IContParaLeerYEjec(cParaLeerAddress).funcWithPrivilege();
    }
}
```

Ejercicio:

Leer la información del siguiente contrato:

```
contract Leer {
    string public name = "Contrato Leer";
    uint256 public edad = 100;
}
```

Structs

¿Qué es un struct?

Con `structs` se pueden definir tipos de datos personalizados en Solidity. Un struct es una colección de variables de diferentes tipos agrupados bajo un tipo definido por el usuario.

Definir un struct

Veamos como se construye un struct:

```
contract Structs {
    // 1. palabra clave 'struct'
    // 2. nombre del struct: 'Balance'
    struct Balance {
        // 3. definición de las variables internas del struct
        //     son como los atributos del struct
        uint256 monto;
        uint256 limiteMensualGasto;
        uint256 interes;
        string name;
    }
}
```

Declaración de una nueva variable tipo struct

Hay al menos dos maneras: la manera leible (recomendada) y la corta.

```
// Declarando un struct
// Legible - Recomendada
Balance miBalance =
    Balance({
        monto: 123,
        limiteMensualGasto: 467,
        interes: 120,
        name: "Cuenta Ahorro"
    });

// Corta
Balance tuBalance = Balance(999, 10000, 200, "Cuenta corriente");
```

Tipo de las variables miembro

Pueden ser de cualquier tipo (uint, bool, address, string, bytes, etc.)

Veamos diferentes combinaciones de Structs: Structs en Arrays, en Mappings, etc.

18_struct.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract Structs {
    // 1. palabra clave 'struct'
    // 2. nombre del struct: 'Balance'
    struct Balance {
        // 3. definición de las variables internas del struct
```

```

//    son como los atributos del struct
uint256 monto;
uint256 limiteMensualGasto;
uint256 interes;
string name;
}

// Declarando un struct
// Legible - Recomendada
Balance miBalance =
Balance({
    monto: 123,
    limiteMensualGasto: 467,
    interes: 120,
    name: "Cuenta Ahorro"
});

// Corta
Balance tuBalance = Balance(999, 10000, 200, "Cuenta corriente");

// Array y Structs
struct DNI {
    uint256 numero;
    uint256 fechaNac;
    uint256 fechaExp;
    string nombre;
    string segundoNombre;
    string apellido;
}
// DNI[] se lee como array cuyos elementos son del tipo DNI
// Hemos creado un array de lista de votantes cuyos miembros son del tipo DNI
DNI[] public listaDeVotantes;

function guardarVotante() public {
    DNI memory nuevoVotante = DNI({
        numero: 44444444,
        fechaNac: 30971989,
        fechaExp: 30072022,
        nombre: "STEVE",
        segundoNombre: "LEE",
        apellido: "MARREROS"
    });
    // el único lugar donde se puede hacer push para un array
    listaDeVotantes.push(nuevoVotante);
}

// Mappings and Structs
struct Profesor {
    string name;

```

```

        string course;
        uint256 edad;
        string locacion;
        string[] materiasADictar;
    }

mapping(address => Profesor) public profesoresPorAddress;

function guardarProfesor1(address _account) public {
    Profesor memory profesor = Profesor({
        name: "Luis",
        course: "Solidity",
        edad: 33,
        locacion: "Lima",
        materiasADictar: new string[](3)
    });
    profesor.materiasADictar[0] = "Algebra";
    profesor.materiasADictar[1] = "Matematica";
    profesor.materiasADictar[2] = "Trigonometria";

    profesoresPorAddress[_account] = profesor;
}

function guardarProfesor2(address _account) public {
    string[] memory _materiasADictar = new string[](3);
    // _materiasADictar.push("A"); => ERROR for arrays that are not storage
    _materiasADictar[0] = "Algebra";
    _materiasADictar[1] = "Matematica";
    _materiasADictar[2] = "Trigonometria";

    Profesor memory profesor = Profesor({
        name: "Luis",
        course: "Solidity",
        edad: 33,
        locacion: "Lima",
        materiasADictar: _materiasADictar
    });

    profesoresPorAddress[_account] = profesor;
}

function leerMaterias(address _account)
public
view
returns (string[] memory)
{
    return profesoresPorAddress[_account].materiasADictar;
}

// Nested structs and mapping

```

```

struct Persona {
    DNI dni;
    uint256 altura;
    uint256 peso;
    // "Casa" => 100,000 soles
    mapping(string => uint256) listaDeActivos;
}

// "Marco" => Persona
mapping(string => Persona) public listaDePersonas;

function guardandoPersona(string memory _namePersona) public {
    DNI memory _dni = DNI({
        numero: 44444444,
        fechaNac: 30971989,
        fechaExp: 30072022,
        nombre: "STEVE",
        segundoNombre: "LEE",
        apellido: "MARREROS"
    });

    Persona storage persona = listaDePersonas[_namePersona];
    persona.dni = _dni;
    persona.altura = 166;
    persona.peso = 66;
    persona.listaDeActivos["Casa"] = 100000;
}

function leerActivos(string memory _name, string memory _activo)
external
view
returns (uint256)
{
    return listaDePersonas[_name].listaDeActivos[_activo];
}
}

```

Construyendo un número "random" en Solidity:

1. Usaremos el método de hasheo nativo que existe en Solidity que es `keccak256`. Una de sus propiedades es que al más ligero cambio de input, el output varía de manera exponencial. Dado que genera identificadores únicos en base a inputs, nos ayudará mucho a construir un número random. `keccak256` recibe variables que son del tipo `bytes`.

e.g.: `keccak256("NUMERO_RAMDON")` nos da como resultado:

```
0xbe3e896f6dae3faacf3f027fd22a9c18167ef88158f428e44ef0ec60bcf1eb68
```

2. En Solidity, podemos hacer `casting`, es decir, convertir un tipo de dato a otro de la siguiente manera `uint256([valor a castear])`. Probemos aplicando lo mismo a lo hallado anteriormente:

e.g. `uint256(keccak256("NUMERO_RANDOM"))` nos da como resultado:

```
86049934292191185529727600163442687223747677739616906022928880938328423263080
```

3. Aquí podemos encontrar una manera para poder incluir más variables dentro del `keccak256`. En solidity, el método `abi.encodePacked` nos ayuda a empaquetar varios valores y retorna un tipo `bytes`, que es al mismo tiempo lo que recibe `keccak256`.

Ejemplo de cómo se compacta:

```
0xfffff42000348656c6c6f2c20776f726c6421
    ^^^^          int16(-1)
      ^^          bytes1(0x42)
    ^^^^          uint16(0x03)
    ~~~~~~ string("Hello, world!") without a length field
```

e.g. `uint256(keccak256(abi.encodePacked(block.timestamp, msg.sender, ...)))`

Con todo lo mencionado podemos crear un método `view` en Solidity que nos retorne un valor pseudo random.

```
function _getRandomNumberOne2() internal view returns (uint256) {
    return
        uint256(keccak256(abi.encodePacked(block.timestamp, msg.sender)));
}
```

Smart Contract Airdrop

¿Qué es un Airdrop?

- Es una manera de distribuir tokens en los usuarios
- Tienen la intención de crear una comunidad alrededor del proyecto
- Premia o recompensa ciertos comportamientos del usuarios
- Cuando envías tokens, la idea detrás es que motiva a las personas a usar tu producto

Vamos a desarrollar dos smart contracts con variaciones de Airdrops:

1. LISTA BLANCA Y NÚMERO ALEATORIO

- Se necesita ser parte de la lista blanca para poder participar del Airdrop
- Los participantes podrán solicitar un número random de tokens de 1-1000 tokens. Crear método `participateInAirdrop`.
- Total de tokens a repartir es 10 millones
- Solo se podrá participar una sola vez
- Si el usuario permite que el contrato airdrop queme 10 tokens, el usuario puede volver a participar una vez más
- El contrato Airdrop tiene el privilegio de poder llamar `mint` del token

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
```

```

import "@openzeppelin/contracts/security/Pausable.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";

/** 
<u>LISTA BLANCA Y NÚMERO ALEATORIO</u>

* Se necesita ser parte de la lista blanca para poder participar del Airdrop - done
* Los participantes podrán solicitar un número rándom de tokens de 1-1000 tokens
* Total de tokens a repartir es 10 millones
* Solo se podrá participar una sola vez
* Si el usuario permite que el contrato airdrop queme 10 tokens, el usuario puede volver a
participar una vez más
* El contrato Airdrop tiene el privilegio de poder llamar `mint` del token
* El contrato Airdrop tiene el privilegio de poder llamar `burn` del token
*/



interface IMiPrimerTKN {
    function mint(address to, uint256 amount) external;

    function burn(address from, uint256 amount) external;

    function balanceOf(address account) external view returns (uint256);
}

contract AirdropONE is Pausable, AccessControl {
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");

    uint256 public constant totalAirdropMax = 10**6 * 10**18;
    uint256 public constant quemaTokensParticipar = 10;

    uint256 airdropGivenSoFar;

    address miPrimerTokenAdd = 0x5FbDB2315678afecb367f032d93F642f64180aa3; // cambiar por la
dirección correcta

    mapping(address => bool) public whiteList;
    mapping(address => bool) public haSolicitado;

    constructor() {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(PAUSER_ROLE, msg.sender);
    }

    function participateInAirdrop() public whenNotPaused {
        // lista blanca
        require(whiteList[msg.sender], "No está en lista blanca");

        // ya solidito tokens
    }
}

```

```

require(!haSolicitado[msg.sender], "Ya ha participado");

// pedir numero random de tokens
uint256 tokensToReceive = _getRandomNumberBelow1000();

// verificar que no se exceda el total de tokens a repartir
require(
    airdropGivenSoFar + tokensToReceive <= totalAirdropMax,
    "No hay tokens disponibles"
);

// actualizar el conteo de tokens repartidos
airdropGivenSoFar += tokensToReceive;
// marcar que ya ha participado
haSolicitado[msg.sender] = true;

// transferir los tokens
IMiPrimerTKN(miPrimerTokenAdd).mint(msg.sender, tokensToReceive);
}

function quemarMisTokensParaParticipar() public whenNotPaused {
    // verificar que el usuario aun no ha participado
    require(haSolicitado[msg.sender], "Usted aun no ha participado");

    // Verificar si el que llama tiene suficientes tokens
    uint256 balanceToken = IMiPrimerTKN(miPrimerTokenAdd).balanceOf(
        msg.sender
    );
    require(
        balanceToken >= quemaTokensParticipar,
        "No tiene suficientes tokens para quemar"
    );

    // quemar los tokens
    IMiPrimerTKN(miPrimerTokenAdd).burn(msg.sender, quemaTokensParticipar);

    // dar otro chance
    haSolicitado[msg.sender] = false;
}

/////////////////////////////// HELPER FUNCTIONS ///////////////////
/////////////////////////////// HELPER FUNCTIONS ///////////////////
/////////////////////////////// HELPER FUNCTIONS ///////////////////

function addToWhiteList(address _account)
public
onlyRole(DEFAULT_ADMIN_ROLE)
{
    whiteList[_account] = true;
}

```

```

}

function removeFromWhitelist(address _account)
    public
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    whiteList[_account] = false;
}

function pause() public onlyRole(PAUSER_ROLE) {
    _pause();
}

function unpause() public onlyRole(PAUSER_ROLE) {
    _unpause();
}

function _getRandomNumberBelow1000() internal view returns (uint256) {
    return
        (uint256(keccak256(abi.encodePacked(block.timestamp, msg.sender))) %
        1000) + 1;
}
}

```

2. REPETIBLE CON LÍMITE, PREMIO POR REFERIDO

- El usuario puede participar en el airdrop una vez por día hasta un límite de 10 veces
- Si un usuario participa del airdrop a raíz de haber sido referido, el que refirió gana 3 días adicionales para poder participar
- El contrato Airdrop mantiene los tokens para repartir (no llama al `mint`)
- El contrato Airdrop tiene que verificar que el `totalSupply` del token no sobrepase el millón
- El método `participateInAirdrop` le permite participar por un número random de tokens de 1000 - 5000 tokens

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/security/Pausable.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";

/**
2. REPETIBLE CON LÍMITE, PREMIO POR REFERIDO

* El usuario puede participar en el airdrop una vez por día hasta un límite de 10 veces
* Si un usuario participa del airdrop a raíz de haber sido referido, el que refirió gana 3 días adicionales para poder participar
* El contrato Airdrop mantiene los tokens para repartir (no llama al `mint` )
* El contrato Airdrop tiene que verificar que el `totalSupply` del token no sobrepase el millón

```

```

* El método `participateInAirdrop` le permite participar por un número random de tokens de 1000
- 5000 tokens
*/



interface IMiPrimerTKN {
    function transfer(address to, uint256 amount) external returns (bool);

    function balanceOf(address account) external view returns (uint256);
}

contract AirdropTwo is Pausable, AccessControl {
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");

    // instanciamos el token en el contrato
    address miPrimerTokenAdd = 0x5FbDB2315678afecb367f032d93F642f64180aa3; // cambiar por la
    dirección correcta
    IMiPrimerTKN miPrimerToken = IMiPrimerTKN(miPrimerTokenAdd);

    struct Participant {
        address cuentaParticipante; // eso me ayudará a saber si ya está registrado
        uint256 participaciones;
        uint256 limiteParticipaciones;
        uint256 ultimaVezParticipado;
    }
    mapping(address => Participant) public participantes;

    constructor() {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(PAUSER_ROLE, msg.sender);
    }

    function participateInAirdrop() public {
        _participateInAirdrop(address(0));
    }

    function participateInAirdrop(address _elQueRefirio) public {
        _participateInAirdrop(_elQueRefirio);
    }

    function _participateInAirdrop(address _account) internal {
        // si esto se cumple, es porque el usuario es nuevo
        // solo funciona si es la primera vez que participa
        // de lo contrario ya se tiene guardado el registro del usuario
        if (participantes[msg.sender].cuentaParticipante == address(0)) {
            // crear el registro del participante
            Participant memory participant = Participant({
                cuentaParticipante: msg.sender,
                participaciones: 1,
                limiteParticipaciones: 10,
            });
            participantes[_account] = participant;
        } else {
            uint256 participacionesActual = participantes[_account].participaciones;
            if (participacionesActual < participantes[_account].limiteParticipaciones) {
                participantes[_account].participaciones++;
            }
        }
    }
}

```

```

        ultimaVezParticipado: block.timestamp
    });

    // guardar el registro del participante
    participantes[msg.sender] = participant;
} else {
    // está reclamando por segunda vez
    Participant memory participant = participantes[msg.sender];

    // verificar que no se haya excedido el límite de participaciones
    require(
        participant.participaciones < participant.limiteParticipaciones,
        "Llegaste limite de participaciones"
    );

    // verificar que no se haya participado en el último día
    // 1 days = 86400 seconds
    require(
        participant.ultimaVezParticipado + 1 days < block.timestamp,
        "Ya participaste en el ultimo dia"
    );

    // actualizar el registro del participante
    participantes[msg.sender].participaciones++;
    participantes[msg.sender].ultimaVezParticipado = block.timestamp;
}

// calcular el número random de tokens a recibir
uint256 tokensToReceive = _getRadomNumber10005000();

// verificar que el Contrato Airdrop tenga los tokens para repartir
uint256 balTokensAirdrop = miPrimerToken.balanceOf(address(this));
require(
    balTokensAirdrop >= tokensToReceive,
    "El contrato Airdrop no tiene tokens suficientes"
);

// transferir los tokens al usuario haciendo uso de la interfaz
// transfer descuenta los tokens de quien llama el método
// en este caso el que llama es el contrato Airdrop
miPrimerToken.transfer(msg.sender, tokensToReceive);

// manejar el caso de que el usuario haya sido referido
if (_account != address(0)) _manejarReferido(_account);
}

function _manejarReferido(address _elQueRefirio) internal {
    if (participantes[_elQueRefirio].cuentaParticipante != address(0)) {
        participantes[_elQueRefirio].limiteParticipaciones += 3;
    }
}

```

```

    }

    ///////////////////////////////////////////////////////////////////
    /////////////////////////////////////////////////////////////////// HELPER FUNCTIONS ///////////////////////////////////////////////////////////////////
    ///////////////////////////////////////////////////////////////////


    function pause() public onlyRole(PAUSER_ROLE) {
        _pause();
    }

    function unpause() public onlyRole(PAUSER_ROLE) {
        _unpause();
    }

    function _getRandomNumber10005000() internal view returns (uint256) {
        return
            (uint256(keccak256(abi.encodePacked(block.timestamp, msg.sender))) %
            4000) +
            1000 +
            1;
    }
}

```

Parte 2

Desarrollando una colección de NFTs

En esta sección vamos a desarrollar una colección de NFTs. Usaremos el siguiente stack:

1. Interplanetary File System (IPFS)
2. ERC721 standard
3. Librería de generación imágenes (npm library)

IPFS

La misión de IPFS es crear una red resiliente, mejorable y abierta para preservar e incrementar el conocimiento de la humanidad.

IPFS desea hacer de la web Peer to Peer (P2P) en vez de tener el tradicional modelo de cliente y servidor. IPFS interconetará nodos de manera resiliente.

Está basado en al direccionamiento basado en el contenido.

IPFS plantea solucionar varios problemas actuales: censura, links rotos (no se basa en un servidor en el contenido que puede estar distribuido en varios nodos), plantea un modelo de seguridad (asegura que el recurso que estás solicitando sea realmente el que estás pidiendo a través del hash del mismo recurso).

Casos de uso: guardar recurso estáticos. páginas web. archivar data. construir Dapps. bases de datos científicas. publicaciones científicas.

El internet como lo conocemos tiene un problema que reside en la centralización. La información está guardada en granjas de servidores que son controlados por una empresa individual. La centralización trae otro problema que es la censura. El gobierno puede bloquear el acceso a ciertos recursos si es que lo desea, por ejemplo aquella vez que impidió el acceso a Wikipedia por llamarlo una amenaza nacional.

La razón por la cual aún se sigue usando este modelo es porque la centralización de servidores permite a las empresas tener control sobre la rapidez en que el contenido puede ser entregado.

El objetivo de IPFS es hacer de la web completamente distribuida en una similar manera en que BitTorrent funciona.

¿Cómo se accede a la información actualmente en el internet?

Cuando deseas descargar un archivo, tu le dices al navegador exactamente dónde de dónde descargarlo. Por ejemplo se utiliza el siguiente link <https://webiste.com/archivo.jpg>. Es decir, la ubicación del archivo será el IP address o el nombre del dominio. A esto se le llama Direccionamiento basado en la ubicación (location-based addressing). En el caso en que el servidor de ese archivo esté caído, no habría la posibilidad de obtener dicho archivo, incluso aunque otra persona lo hubiera obtenido.

Para solucionar ese problema, IPFS propone el Direccionamiento basado en el contenido (content-based addressing) y ya no en la ubicación. En vez de decirle al navegador dónde conseguir el recurso, ahora se le dirá qué es lo que se quiere conseguir. Para lograr ello, es necesario que cada recurso posea un hash único, que es como su huella digital. Entonces, cuando deseas descargar cierto archivo, se preguntará a la red quién tiene el archivo con dicho hash.

¿Y cómo podrías saber que la persona que te envía el recurso no lo ha alterado? Al recibir el recurso, puedes aplicar un método hash en el recurso y comparar el resultado con el hash inicialmente usado para solicitar dicho recurso.

Cuando múltiples personas publican el mismo recurso en la red, este recurso se crea una sola vez y se evita la duplicación, lo cual hace la red más eficiente.

¿Cómo IPFS almancena recursos y lo hace accesible para otras personas?

IPFS utiliza "objetos de IPFS" y puede almacenar hasta 256 kb de información. Dentro de este objeto se puede incluir links a otros objetos de IPFS. En el caso en que se almacene un recurso que es mucho mayor al límite de 256 kb, el recurso se dividirá en múltiples objetos de IPFS de 256 kb cada uno. Seguido a ello, el sistema creará un objeto IPFS vacío que se encargará de juntar los links a todos los objetos de IPFS creados.

Dado que IPFS utiliza el direccionamiento basado en el contenido (content-based addressing), una vez que un recurso es añadido a IPFS, ya no puede ser cambiado. IPFS es una base de datos inmutable, parecida a un blockchain.

El más grande problema que tiene IPFS es el de mantener los archivos disponibles. Todos los nodos de la red guarda un caché de los archivos que ha descargado. También ayudan a compartir el recurso si es solicitado desde otro lugar. El problema surge cuando los nodos que tienen ciertos recursos en memoria se desconectan y nadie más puede obtener dichos recursos. Es como tener BitTorrent sin clientes que surtan un recurso que se está descargando.

¿Cómo podemos solucionarlo? Podemos incentivar a las personas a mantener activo sus nodos para guardar recursos, de modo tal que estén disponibles. O también podemos preventivamente distribuir los recursos en varios nodos de modo tal que siempre hay copias disponibles. Eso es exactamente lo que Filecoin intenta hacer.

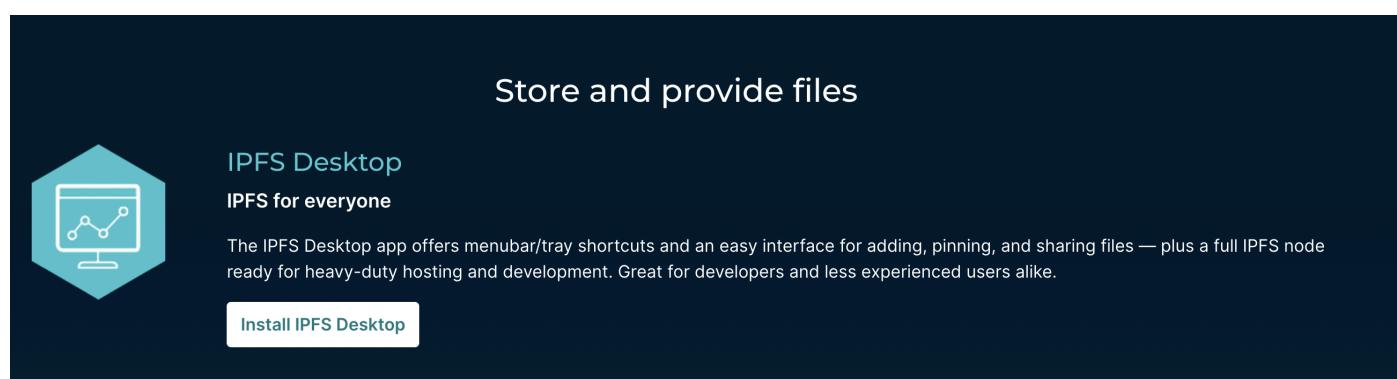
Filecoin ha sido creado por el mismo grupo de gente que creó IPFS. Filecoin es un blockchain crede encima de IPFS que busca ser un mercado descentralizado para guardar información. Es decir, si dispones de un espacio de memori disponible, lo pudes rentar y hacer dinero. De ese modo, Filecoin incentiva a los nodos a mantenerse conectados tan largo como sea posible para poder obtener las recompensas. Del mismo modo, el sistema se preocupa que los recursos se dupliquen en varios nodos.

¿Cómo se puede usar IPFS? En el año 2017, el gobierno de Turquía decidió prohibir el acceso a Wikipedia. La respuesta frente a ello, es que se puso una copia de Wikipedia en IPFS. Dado que IPFS es distribuido y no hay servidores centrales, el gobierno no puede bloquearlo.

¿Por qué es llamado interplanetario? Pues en cada planeta, una vez que algn recurso ya se ha solicitado por primera vez, estará cacheado en memoria para ser obtenido dentro del mismo planeta y no habría la necesidad de viajar hasta otro planeta para obtener dicho recurso.

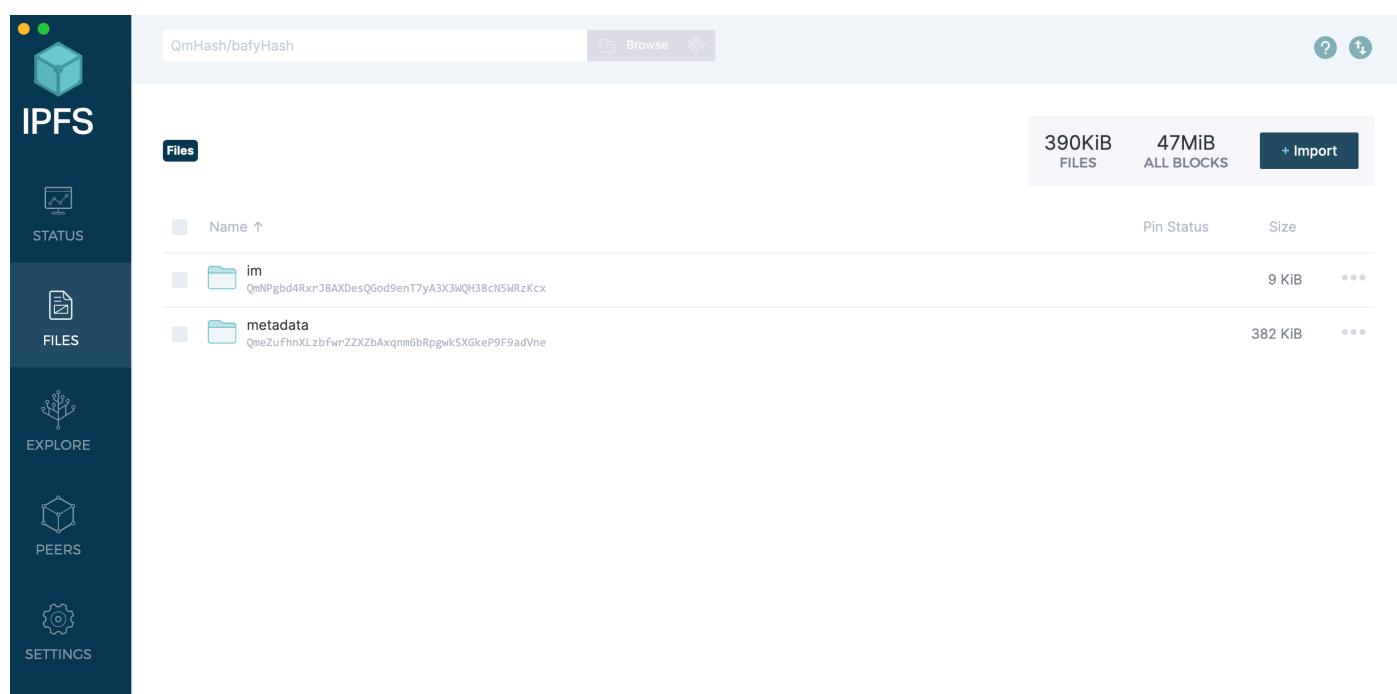
Instalación de IPFS

Dirígete a <https://ipfs.tech/#install> e instala la versión desktop de IPFS.



The screenshot shows the IPFS Desktop app download page. At the top, it says "Store and provide files". Below that is a large teal hexagonal icon containing a white line graph. To the right of the icon, the text "IPFS Desktop" and "IPFS for everyone" is displayed. A descriptive paragraph states: "The IPFS Desktop app offers menubar/tray shortcuts and an easy interface for adding, pinning, and sharing files — plus a full IPFS node ready for heavy-duty hosting and development. Great for developers and less experienced users alike." A prominent "Install IPFS Desktop" button is located at the bottom of this section.

En la sección de **FILES** es donde guardaremos la información para la colección de NFTs.



The screenshot shows the IPFS desktop application interface. On the left is a sidebar with icons for IPFS (blue cube), STATUS (bar chart), FILES (document), EXPLORE (tree), PEERS (cube with network lines), and SETTINGS (gear). The main area has a header with "QmHash/bafyHash" and "Browse" buttons. Below is a "Files" section with a table:

| Name | Pin Status | Size |
|----------|------------|------|
| im | 9 KiB | ... |
| metadata | 382 KiB | ... |

At the top right of the main area, there are stats: "390KiB FILES" and "47MiB ALL BLOCKS", along with a "+ Import" button. The "FILES" tab is highlighted in the sidebar.

ERC721 Standard

El ERC721 es un tipo de estándar o formato que los desarrolladores acuerdan seguir. No es obligatorio pero ayuda a crear compatibilidad con una serie de aplicaciones descentralizadas. En Ethereum, el estándar ERC721 se usa para crear NFTs.

¿Qué es un NFT? NFT significa un Token No Fungible. Fungible significa intercambiable o reemplazable. Por ejemplo, un bitcoin es fungible ya que al ser intercambiado por exactamente otro bitcoin, su valor no ha cambiado. En cambio, los NFT son completamente únicos y no existe equivalencia de uno a uno con otro NFT.

Con el estándar ERC721, cada token del smart contract puede tener un valor diferente a raíz de su antigüedad, rareza o incluso por como luce visualmente. Cada NFT tiene un token id y un método especial que al introducir dicho token id, devuelve un elemento visual que representa al NFT.

Este tipo de token es perfecto para ser usado en plataformas que ofrecen colecciónables, accesos privados, tickets de lotería, sitios enumerados para conciertos, etc.

Uno de los proyectos más tempranos y conocidos hasta el momento es [CryptoKitties](#) que usa internamente el estándar ERC721.



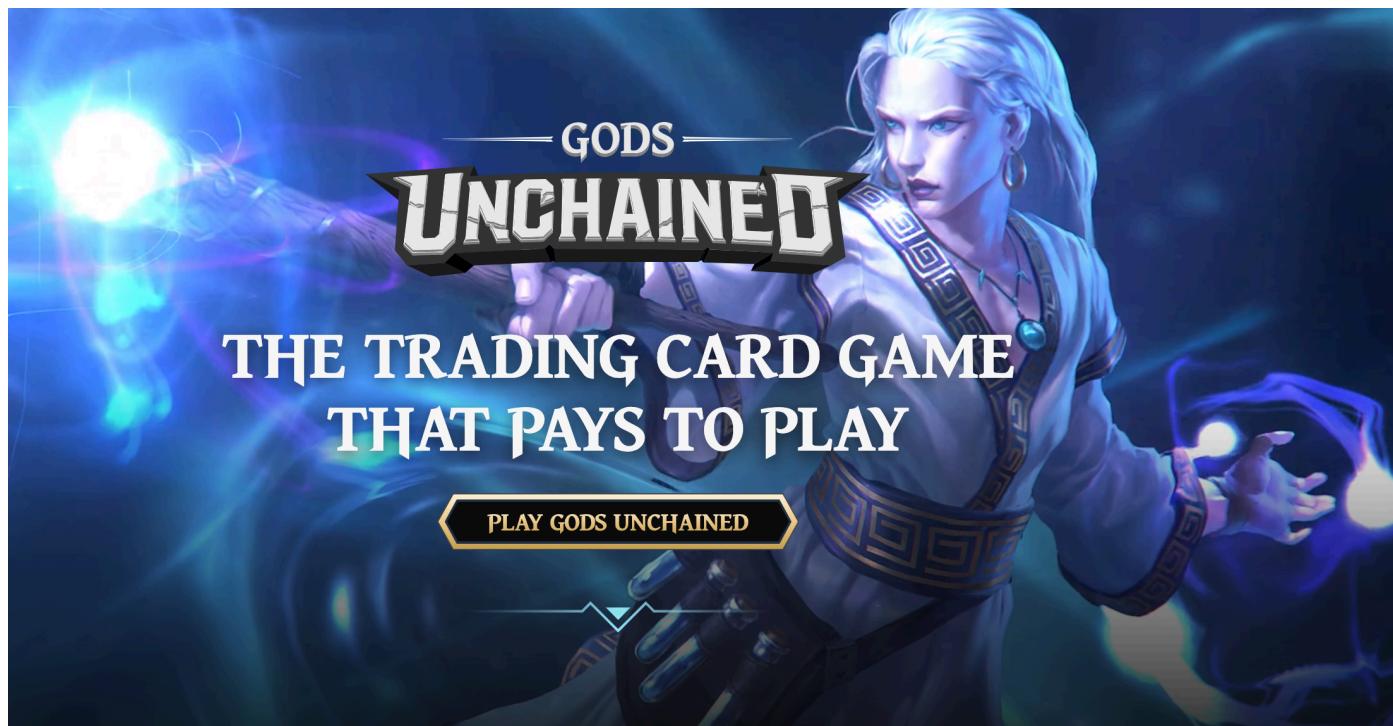
Se puede observar cómo es que uno de los smart contracts de Crypto Kitties hereda el contrato ERC721 en la siguiente ilustración:

```
466
467
468 /// @title The facet of the CryptoKitties core contract that manage
469 /// @author Axiom Zen (https://www.axiomzen.co)
470 /// @dev Ref: https://github.com/ethereum/EIPs/issues/721
471 /// See the KittyCore contract documentation to understand how the
472 contract KittyOwnership is KittyBase, ERC721 {
473
474     /// @notice Name and symbol of the non fungible token, as defined
475     string public constant name = "CryptoKitties";
```

Bajo el estándar ERC721, se pueden crear tokens que son únicos. Con este modelo se ha difundido la idea de tener activos únicos en Ethereum.

El día de hoy, el uso más común para el ERC721 es arte digital. Las personas compran estos NFTs por una variedad de razones. Algunos quieren apoyar al artista, otros buscan una inversión de largo plazo con la esperanza de que el precio subirá. O quizás simplemente les gusta el arte que representa dicho NFT.

Sin embargo, los casos de uso de los NFT se extiende más allá del arte. Los NFTs también puede ser usados en juegos que son basados en el blockchain. Estos NFTs representan activos únicos dentro del juego. Ejemplo de ello es [Gods Unchained](#). En ese juego puedes coleccionar cartas que se pueden tranzar en un mercado de segunda mano.



Los NFTs en la música también se han adaptado. Aplicaciones como Audius permite a los artistas acuñar su trabajo como si fueran tokens ERC721.

Construyendo un ERC721

Antes de revisar el estándar ERC721, revisemos algunas claves diferencias con el estándar ERC20.

Balance

Dado que en el estándar ERC721 los items son diferentes, cuando se pregunta por el balance, el valor a retornar representa la cantidad de items diferentes que tiene una cuenta.

Propiedad

Cuando se acuña un nuevo NFT, se asocia el token id del NFT con su dueño. Se puede identificar quién es el dueño de un particular token de NFT bajo el estándar ERC721.

Visualización

Los NFTs tienen propiedades visuales en base a su rareza. Hay un método en particular que te permite obtener un recurso visual del NFT, así como también propiedades extras.

Permiso para operar

Mientras que en el estándar ERC20, se podía otorgar permiso a otra cuenta para operar una cantidad de tokens, en el caso del ERC721, el permiso se da o bien por cada token id (`approve`) o bien el permiso es general (`setApprovalForAll`) para todos los NFTs de una cuenta.

Ello implica que dentro del smart contract, se tenga que manejar dos tipos de permisos con su propio mecanismo de registro.

En el primer tipo de permiso, el dueño del token otorgará el privilegio para que otra cuenta pueda disponer uno de sus NFT en particular. Este dueño puede repetir la operación varias veces. Sin embargo, por cada operación, un solo token (usando el `tokenId`) será entregado.

Usando el segundo tipo de permiso, el dueño da el acceso a que otra cuenta pueda manejar la totalidad de sus NFTs a su discreción. Aquí no es necesario usar un id del token porque el permiso es general.

Ambos permisos descritos se pueden otorgar y quitar cuando el dueño de los NFTs lo desee.

No existe el método `transfer`

El método `transfer` es del estándar ERC20 pero no aparece en el estándar ERC721. En cambio, sí tiene el `transferFrom` que cumple el mismo propósito y tiene tres parámetros: `from`, `to` y `tokenId`. Es decir, se especifica el address origen, el address destino y el identificador del token que se desea enviar.

Método `safeTransferFrom`

The principal diferencia entre `safeTransferFrom` y `transferFrom` es que el primero se encarga de verificar que la cuenta a la cual se hace la transferencia sea una cuenta compatible para recibir NFTs. Especialmente es usada para verificar que ciertos smart contracts que reciben NFTs sean capaces de recibir y transferir NFTs. De otro modo, dichos NFTs quedarían bloqueados por siempre en el smart contract.

Elementos complementarios del ERC721

Para terminar de implementar el estándar ERC721 desde cero, hay otros elementos necesarios que tener en cuenta. Será necesario poder convertir un tipo de dato entero a un tipo de dato string para poder crear una ruta de almacenamiento (donde se guarda la metadata). Adicionalmente, en el caso en que se deseé enviar NFTs a otro smart contract, este debe implementar cierta interfaz, la cual será válida dentro del estándar ERC721. Veamos estos dos puntos a continuación.

Librería `Strings`

Haremos uso de esta librería para poder convertir números enteros al tipo de dato `string`. Ello será necesario para poder concatenar de manera dinámica el `tokenId` de un token con la ruta en donde se encuentra almacenada su metadata.

La manera de incorporar una librería en un tipo de dato, es de la siguiente manera:

```
using Strings for uint256;
```

De esta manera se especifica que el tipo de dato `uint256` ahora tendrá funcionalidades extras descritas en la librería `Strings`. Para este caso en particular, dicha librería tiene un método llamado `toString` que convierte un tipo de dato `uint256` en un tipo `string`. Se usa de la siguiente manera:

```
uint256 number = 2;
uint256 numberString = number.toString(); // "2"
```

Interface `IERC721Receiver`

Cuando se intente enviar NFTs a otro smart contract, se solicita que el otro smart contract implemente el siguiente método para evaluar su compatibilidad:

```
function onERC721Received(
    address operator,
    address from,
    uint256 tokenId,
    bytes calldata data
) external returns (bytes4);
```

Cuanto el contrato ERC721 intente enviar tokens a otro contrato, se verificará que dicho método (`onERC721Received`) exista en el otro contrato y retorne el `selector` del mismo método. Caso contrario, la operación se revierte. En el otro contrato, el método a definir sería el siguiente:

```
interface IERC721Receiver {
    function onERC721Received(
        address operator,
        address from,
        uint256 tokenId,
        bytes calldata data
    ) external returns (bytes4);
}

contract ReceiveNFT is IERC721Receiver {
    // ...
    function onERC721Received(address, address, uint256, bytes calldata) external pure returns (bytes4) {
        return IERC721Receiver.onERC721Received.selector;
    }
}
```

Cabe mencionar que cada vez que se intenta realizar una transferencia, se llevará a cabo esta verificación de manera necesaria.

Construyendo desde cero el ERC721

1. `balanceOf` - lleva la cuenta de la cantidad de NFTs de una cuenta. un argumento: `owner` (`address`)
2. `name` - devuelve el nombre de la colección de NFTs
3. `symbol` - devuelve el símbol de la colección de NFTs
4. `approve` - aprueba a otra cuenta el manejar un id de NFT. Dos argumentos: `to` (`address`) y un `tokenId(uint256)`.
 1. `getApproved`: devuelve quién tiene el `approve` de un `tokenId`. un argumento: `tokenId` (`uint256`)

5. burn - quema un NFT usando tokenId. un argumento: tokenId(uint256).
6. safeMint - acuña un nuevo NFT. un argumento: to(address)
 1. ownerOf: devuelve el dueño (address) de un tokenId. un argumento: tokenId (uint256)
7. safeTransferFrom - transfiere un NFT con su id de una cuenta a otra. tres argumentos: from (address), to (address) y tokenId (uint256).
8. safeTransferFrom - transfiere un NFT con su id de una cuenta a otra. cuatro argumentos: from (address), to (address), tokenId (uint256), data (bytes).
9. setApprovalForAll - aprueba a una cuenta la operación de todos los NFTs. dos argumentos: operator(address) y approved (bool)
 1. isApprovedForAll: comprueba si un address puede operar todos los NFTs de otra cuenta. dos argumentos: owner (address) y operator(address)
10. transferFrom - transfiere un NFT con su id de una cuenta a otra. tres argumentos: from (address), to (address) y tokenId (uint256).
11. tokenURI: devuelve la ruta del IPFS donde se encuentra la metadata de un NFT de manera dinámica. un argumento: tokenId (uint256)

Tomando en cuenta las dos consideraciones de `Strings` y `IERC721Receiver`, podemos pasar a implementar un contrato basado en el estándar ERC721.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

interface IERC721 {
    event Transfer(
        address indexed from,
        address indexed to,
        uint256 indexed tokenId
    );
    event Approval(
        address indexed owner,
        address indexed approved,
        uint256 indexed tokenId
    );
    event ApprovalForAll(
        address indexed owner,
        address indexed operator,
        bool approved
    );

    function name() external view returns (string memory);

    function symbol() external view returns (string memory);

    function tokenURI(uint256 tokenId) external view returns (string memory);
}
```

```
function balanceOf(address owner) external view returns (uint256 balance);

function ownerOf(uint256 tokenId) external view returns (address owner);

function safeTransferFrom(
    address from,
    address to,
    uint256 tokenId,
    bytes calldata data
) external;

function safeTransferFrom(
    address from,
    address to,
    uint256 tokenId
) external;

function transferFrom(
    address from,
    address to,
    uint256 tokenId
) external;

function approve(address to, uint256 tokenId) external;

function setApprovalForAll(address operator, bool _approved) external;

function getApproved(uint256 tokenId)
    external
    view
    returns (address operator);

function isApprovedForAll(address owner, address operator)
    external
    view
    returns (bool);
}

interface IERC721Receiver {
    /**
     * @dev Whenever an {IERC721} `tokenId` token is transferred to this contract via {IERC721-safeTransferFrom}
     * by `operator` from `from`, this function is called.
     *
     * It must return its Solidity selector to confirm the token transfer.
     * If any other value is returned or the interface is not implemented by the recipient, the transfer will be reverted.
     */
}
```

```

    * The selector can be obtained in Solidity with
`IERC721Receiver.onERC721Received.selector`.

*/
function onERC721Received(
    address operator,
    address from,
    uint256 tokenId,
    bytes calldata data
) external returns (bytes4);
}

library Strings {
    /**
     * @dev Converts a `uint256` to its ASCII `string` decimal representation.
     */
    function toString(uint256 value) internal pure returns (string memory) {
        // Inspired by OraclizeAPI's implementation - MIT licence
        // https://github.com/oraclize/ethereum-
api/blob/b42146b063c7d6ee1358846c198246239e9360e8/oraclizeAPI_0.4.25.sol

        if (value == 0) {
            return "0";
        }
        uint256 temp = value;
        uint256 digits;
        while (temp != 0) {
            digits++;
            temp /= 10;
        }
        bytes memory buffer = new bytes(digits);
        while (value != 0) {
            digits--;
            buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
            value /= 10;
        }
        return string(buffer);
    }
}

contract MiPrimerNFT is IERC721 {
    // 1. balanceOf: lleva la cuenta de los NFTs de un address. un argumento: owner(address)
    // 2. name: devuelve el nombre de la colección
    // 3. symbol: devuelve el símbolo de la colección
    // 4. burn: quema el NFT. lo pasa al address 0. un argumento: tokenId(uint256)
        // * si otra persona tiene permiso, lo puede quemar
    // 5. safeMint: acuña un NFT a un address. secuencial. un argumento: to (address)
        // * es safe porque verifica que el recipiente, si es un SC, puede manejar NFTs
    // 5.1 ownerOf: me devuelve quien es el dueño de un tokenId
        // un argumento: tokenId(uint256)
}

```

```

// 6. approve: da permiso a una cuenta de manejar UN NFT
    // dos argumentos: to (address), tokenId(uint256)
    // 6.1 getApproved: me indica quien es el operador de UN nft con el tokenId
        // un argumento: tokenId(uint256)
// 7. transferFrom: transfiere un NFT usando el tokenId, de una cuenta from a una cuenta
to.
    // 3 argumentos: from(address), to(address), tokenId(uint256)
// 8. safeTransferFrom: transfiere un NFT usando el tokenId, de una cuenta from a una cuenta to.
    // 3 argumentos: from(address), to(address), tokenId(uint256)
    // * ademas de transferir, revisa que el recipiente puede manejar NFTs
// 9. safeTransferFrom: transfiere un NFT usando el tokenId, de una cuenta from a una cuenta to.
    // 4 argumentos: from(address), to(address), tokenId(uint256), data(bytes)
    // * ademas de transferir, revisa que el recipiente puede manejar NFTs

// 10. setApprovalForAll: otorga el permiso a todos los NFTs de una cuenta
    // 2 argumentos: operator (address), approved(bool)
    // * approved: true => da permiso, false => quita permiso
    // 10.1 isApprovedForAll: me devuelve el address que puede operar todos los NFTs
        // 2 argumentos: owner (address), operator(address)

// 11. tokenURI: nos devuelve (de manera dinamica) la ruta de la metadata para un tokenId
    // un argumento: tokenId(uint256)

using Strings for uint256;

// 1. balanceOf: lleva la cuenta de los NFTs de un address. un argumento: owner(address)
// address => uint256
mapping(address => uint256) _balances;
function balanceOf(address owner) public view returns(uint256) {
    return _balances[owner];
}

string _nameCollection;
string _symbolCollection;
constructor(string memory _name, string memory _symbol) {
    _nameCollection = _name;
    _symbolCollection = _symbol;
}

// 2. name: devuelve el nombre de la colección
function name() public view returns(string memory) {
    return _nameCollection;
}

// 3. symbol: devuelve el simbolo de la colección
function symbol() public view returns(string memory) {
    return _symbolCollection;
}

```

```

}

// 5. safeMint: acuña un NFT a un address. secuencial. un argumento: to (address)
    // * es safe porque verifica que el recipiente, si es un SC, puede manejar NFTs
    // 5.1 ownerOf: me devuelve quien es el dueño de un tokenId
        // un argumento: tokenId(uint256)

/**
    tokenId      |      addresses
    0           |      Lee
    1           |      Lee
    2           |      Pedro
    3           |      Jhon
    mapping(uint256 => address) _owners;
*/
mapping(uint256 => address) _owners;
uint256 counter;
function safeMint(address to) public {
    uint256 tokenId = getCurrentCounter();

    _owners[tokenId] = to;
    _balances[to]++;
    incCounter();

    // hacer el checking
    require(_checkOnERC721Received(address(0), to, tokenId, ""), "El recipiente no soporta
NFTs");

    emit Transfer(address(0), to, tokenId);
}

function ownerOf(uint256 tokenId) public view returns(address) {
    return _owners[tokenId];
}

// 4. burn: quema el NFT. lo pasa al address 0. un argumento: tokenId(uint256)
    // * si otra persona tiene permiso, lo puede quemar
function burn(uint256 tokenId) public {
    // si le dio permiso
    address segundoDuenio = _aprobadoParaUno[tokenId];

    // si es el dueño
    address owner = _owners[tokenId];

    // si tiene el permisos de TODOS los NFTs
    bool permisoTotal = _aprobadoTodos[owner][msg.sender];

    require(
        msg.sender == owner || segundoDuenio == msg.sender || permisoTotal ,

```

```

        "No eres el dueño para quemar el NFT o no tienes permiso"
    );

    _owners[tokenId] = address(0);
    _balances[msg.sender]--;
}

emit Transfer(msg.sender, address(0), tokenId);
}

// 6. approve: da permiso a una cuenta de manejar UN NFT
// dos argumentos: to (address), tokenId(uint256)
// 6.1 getApproved: me indica quien es el operador de UN nft con el tokenId
// un argumento: tokenId(uint256)

/** El segundo "dueño"
tokenId | addresses
0       | Lee
1       | Lee
2       | Pedro
3       | Jhon
mapping(uint256 => address) _aprobadoParaUno;
*/
mapping(uint256 => address) _aprobadoParaUno;
function approve(address operator, uint256 tokenId) public {
    require(msg.sender != operator, "No puedes darte permiso (a ti mismo)");

    address owner = _owners[tokenId];
    require(msg.sender == owner, "No eres el dueño del NFT");

    _aprobadoParaUno[tokenId] = operator;

    emit Approval(msg.sender, operator, tokenId);
}

function getApproved(uint256 tokenId) public view returns(address) {
    return _aprobadoParaUno[tokenId];
}

// 7. transferFrom: transfiere un NFT usando el tokenId, de una cuenta from a una cuenta
to.
// 3 argumentos: from(address), to(address), tokenId(uint256)
function transferFrom(address from, address to, uint256 tokenId) public {
    // si le dio permiso de un NFT
    address segundoDuenio = _aprobadoParaUno[tokenId];

    // si es el dueño
    address owner = _owners[tokenId];

    // si le dio permiso de TODOS los NFTs
    bool permisoTotal = _aprobadoTodos[owner][msg.sender];
}

```

```

require(
    msg.sender == owner || segundoDuenio == msg.sender || permisoTotal,
    "No eres el dueño del NFT o no tienes permiso"
);

_owners[tokenId] = to;
_balances[from]--;
_balances[to]++;
emit Transfer(from, to, tokenId);
}

function safeTransferFrom(address from, address to, uint256 tokenId) public {
    safeTransferFrom(from, to, tokenId, "");
}

function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data)
public {
    transferFrom(from, to, tokenId);
    // adicional hace un checking
    require(_checkOnERC721Received(from, to, tokenId, data), "El recipiente no soporta
NFTs");
}

// 10. setApprovalForAll: otorga el permiso a todos los NFTs de una cuenta
// 2 argumentos: operator (address), approved(bool)
// * approved: true => da permiso, false => quita permiso
// 10.1 isApprovedForAll: me devuelve el address que puede operar todos los NFTs
// 2 argumentos: owner (address), operator(address)
/***
(duenios de NFTs)          Pedro          Juan          Carlos
Lee                      Sí            No            Sí
Marcos                   NO           No            No
Jhon                     Sí            Sí           Sí
owner => operator => approved
mapping(address => mapping(address => bool)) _aprobadoTodos;
*/
mapping(address => mapping(address => bool)) _aprobadoTodos;
function setApprovalForAll(address operator, bool approved) public {
    require(msg.sender != operator, "No puedes darte permiso (a ti mismo)");

    _aprobadoTodos[msg.sender][operator] = approved;

    emit ApprovalForAll(msg.sender, operator, approved);
}
function isApprovedForAll(address owner, address operator) public view returns(bool) {
    return _aprobadoTodos[owner][operator];
}

```

```

// 11. tokenURI: nos devuelve (de manera dinámica) la ruta de la metadata para un tokenId
    // un argumento: tokenId(uint256)
string baseURI = "ipfs://QmVZkuCVeMStEYnYj1vFYDEdggwoQ2evHFn7wmj97RMUmf/";
function tokenURI(uint256 tokenId) public view returns(string memory) {
    return string(
        abi.encodePacked(baseURI, tokenId.toString(), ".json")
    );
}

/////////////////////////////// HELPERS /////////////////////////////////
//



function _checkOnERC721Received(
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) private returns (bool) {
    if (to.code.length > 0) { // esto valida que es un SC
        try
            IERC721Receiver(to).onERC721Received(
                msg.sender,
                from,
                tokenId,
                data
            )
        returns (bytes4 retval) {
            return retval == IERC721Receiver.onERC721Received.selector;
        } catch (bytes memory reason) {
            if (reason.length == 0) {
                revert(
                    "ERC721: transfer to non ERC721Receiver implementer"
                );
            } else {
                /// @solidity memory-safe-assembly
                assembly {
                    revert(add(32, reason), mload(reason))
                }
            }
        } else {
            return true; // sino se trata de un EOA
        }
    }
}

```

```

}

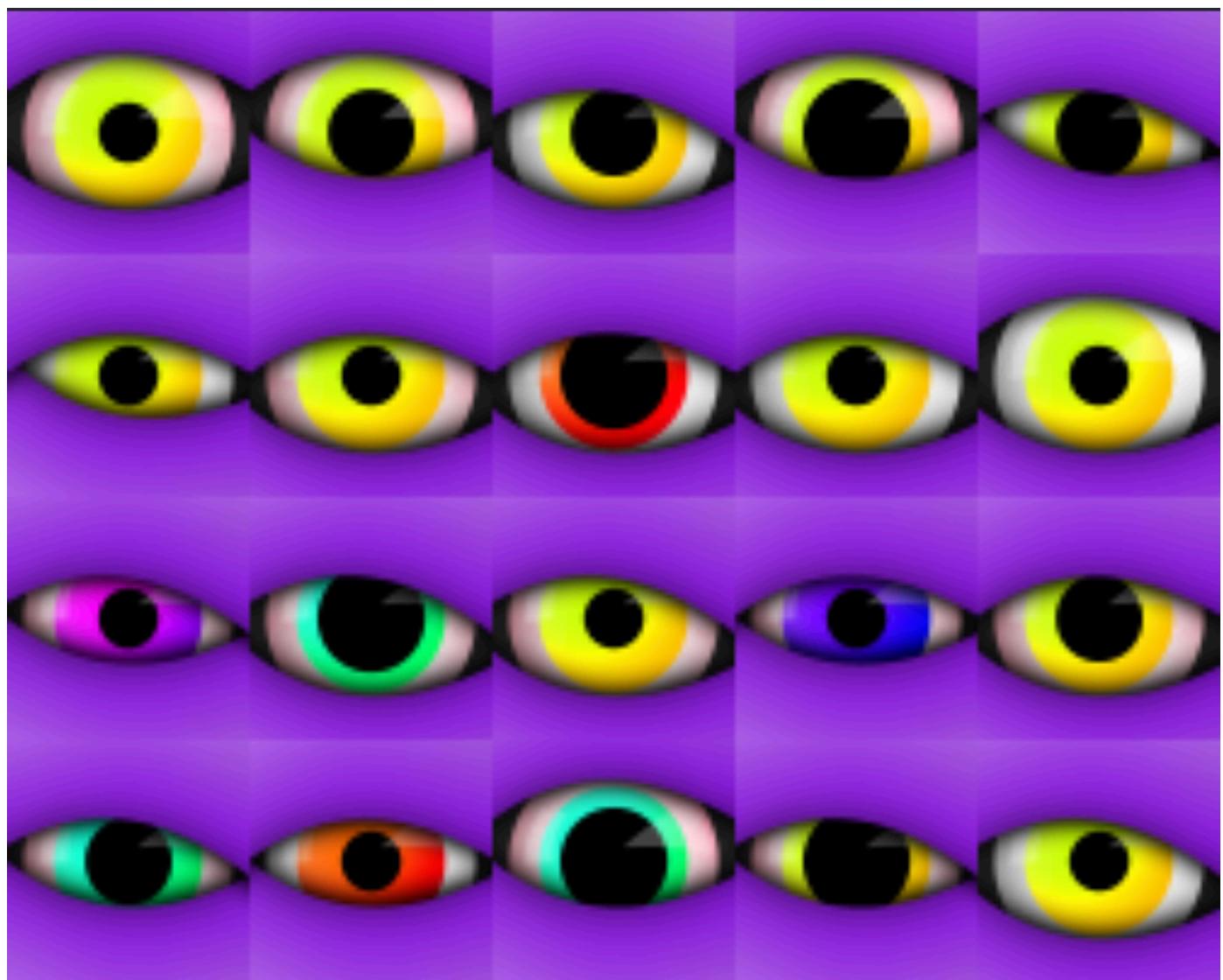
function getCurrentCounter() internal view returns(uint256) {
    return counter;
}

function incCounter() internal {
    counter++;
}
}

```

Generación de Imágenes usando HashLips

Imagen creada con software de generación de imágenes usando capas superpuestas. Descargar la librería [HashLips art engine](#).



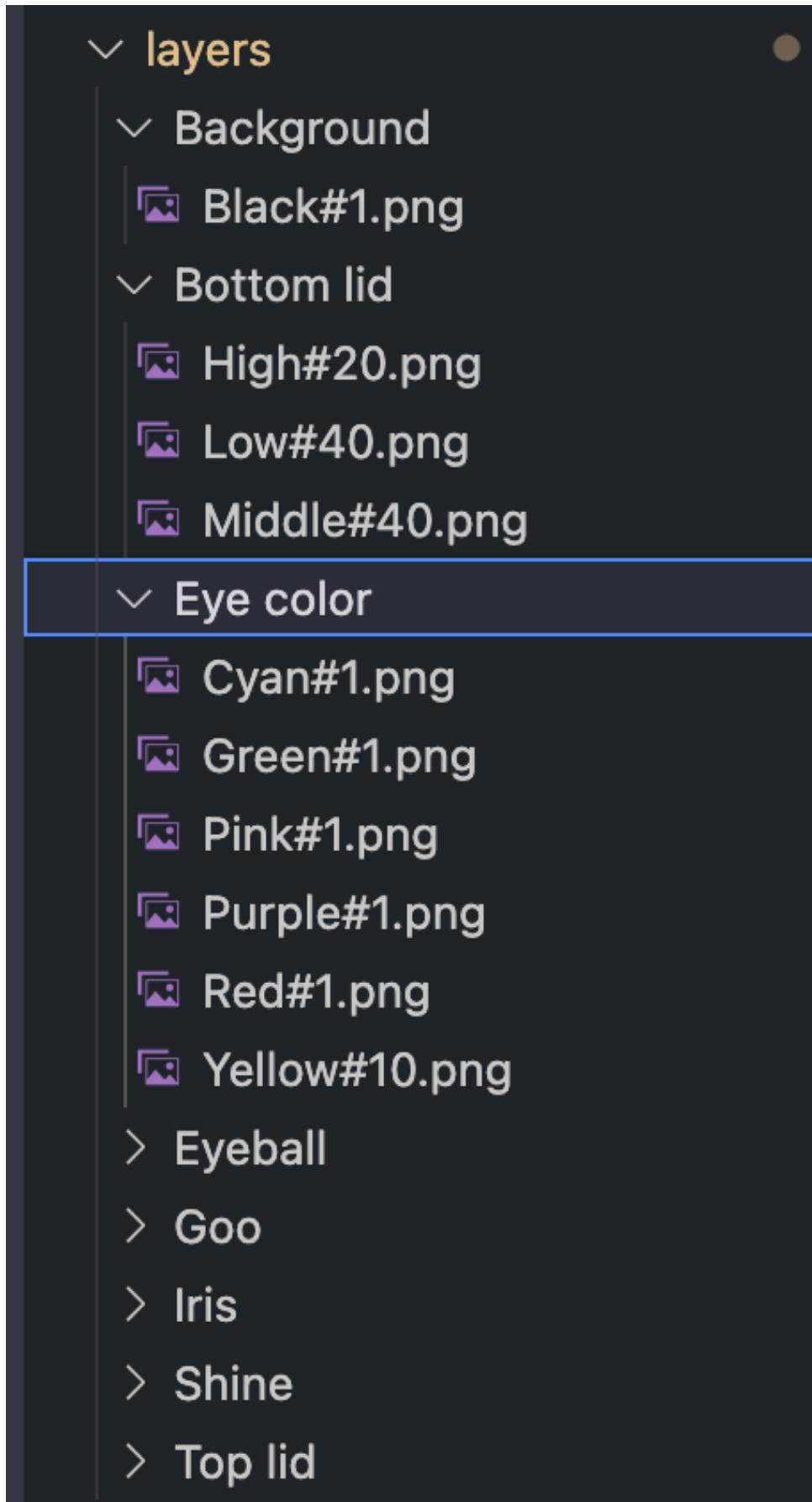
Carpeta Layers

El punto de partida para la generación de NFTs, es la creación de capas a combinar. Cada NFT se puede descomponer en diferentes capas superpuestas. Estas capas se pueden combinar de manera aleatoria o incluir ciertos parámetros para asegurar que ciertas capas se repitan menos que otras.

Por lo general, dichas capas son creadas en programas de edición de imágenes. Es importante que dicha capa mida en ancho y largo que cualquier otra capa y que, además, mantengan un fondo transparente permitir la visibilidad de una capa inferior.

En dicha carpeta `layers`, se crea una carpeta por cada capa a usar. En dicha carpeta pueden existir varias imágenes. El orden de las capas a combinar se configura en el archivo `config.js`. Los nombres de dichas imágenes tienen un formato especial que contribuye con la cantidad de veces que se repetirá.

El formato es como sigue: `Black#1.png` `High#20.png`. Se guardan los archivos seguidos de un `#` y un número que representa su rareza. Si el valor de la rareza es mayor, su repetición en las imágenes será mayor.



Archivo `src/config.js`

Este es el único archivo que se tiene que cambiar para poder crear una colección. La configuración más importante se da en la siguiente variable del archivo `config.js`:

```

const layerConfigurations = [
  {
    growEditionSizeTo: 5,
    layersOrder: [
      { name: "Background" },
      { name: "Eyeball" },
      { name: "Eye color" },
      { name: "Iris" },
      { name: "Shine" },
      { name: "Bottom lid" },
      { name: "Top lid" },
    ],
  },
  /**
   //incluir otras combinaciones de capas
  {...},
  {...},
 */
];

```

La variable `layerConfigurations` es un array de objetos en el que cada objeto representa una combinación diferente de capas. Se pueden considerar tantas capas como se desee. Las capas a usar se encuentran ya creadas en la carpeta `layers`, cuya creación se debe hacer con anterioridad a crear las imágenes.

Cada objeto (combinación de capas) tiene dos propiedades: `growEditionSizeTo` y `layersOrder`. La primera variable (`growEditionSizeTo`) indica la cantidad de NFTs que se desea crear (puede ser un número grande como 10,000). La segunda variable (`layersOrder`) es un array que especifica las capas a usar y además el orden que se ubicarán las capas para crear los NFTs.

Nota: en el caso en que existan varias combinaciones de capas, la variable `growEditionSizeTo` de cada objeto debe incluir el conteo del anterior.

En el ejemplo mostrado, se ha considera una sola combinación de capas. En dicha combinación, se especifica que se generarán cinco NFTs. Las capas a usar y el orden están descritas en `layersOrder`. El nombre de las capas descritas en `layersOrder`, debe corresponder con el nombre de las carpetas que se encuentran en la carpeta `layers`.

Generando los NFTs via terminal

Para generar las imágenes que serán parte de los NFTs, ejecutar el siguiente comando desde el terminal en la carpeta raíz: `node index.js`. Al hacerlo, se producirá el siguiente resultado:

```

Created edition: 1, with DNA: 092b37710d6bfc92aa049d4b05accc841e7c22fd
Created edition: 2, with DNA: 57980f389a30ca9db9bf31c11ee8a2794dcbbb72
Created edition: 3, with DNA: a07a9dad7fa467f9841644fe96e551bec34fca68
Created edition: 4, with DNA: 594219bfa83c650c4811a0db3ad0c2ec03f5bdf2
Created edition: 5, with DNA: 749f101fac6b14a3ad2024d979ad264167b8f104

```

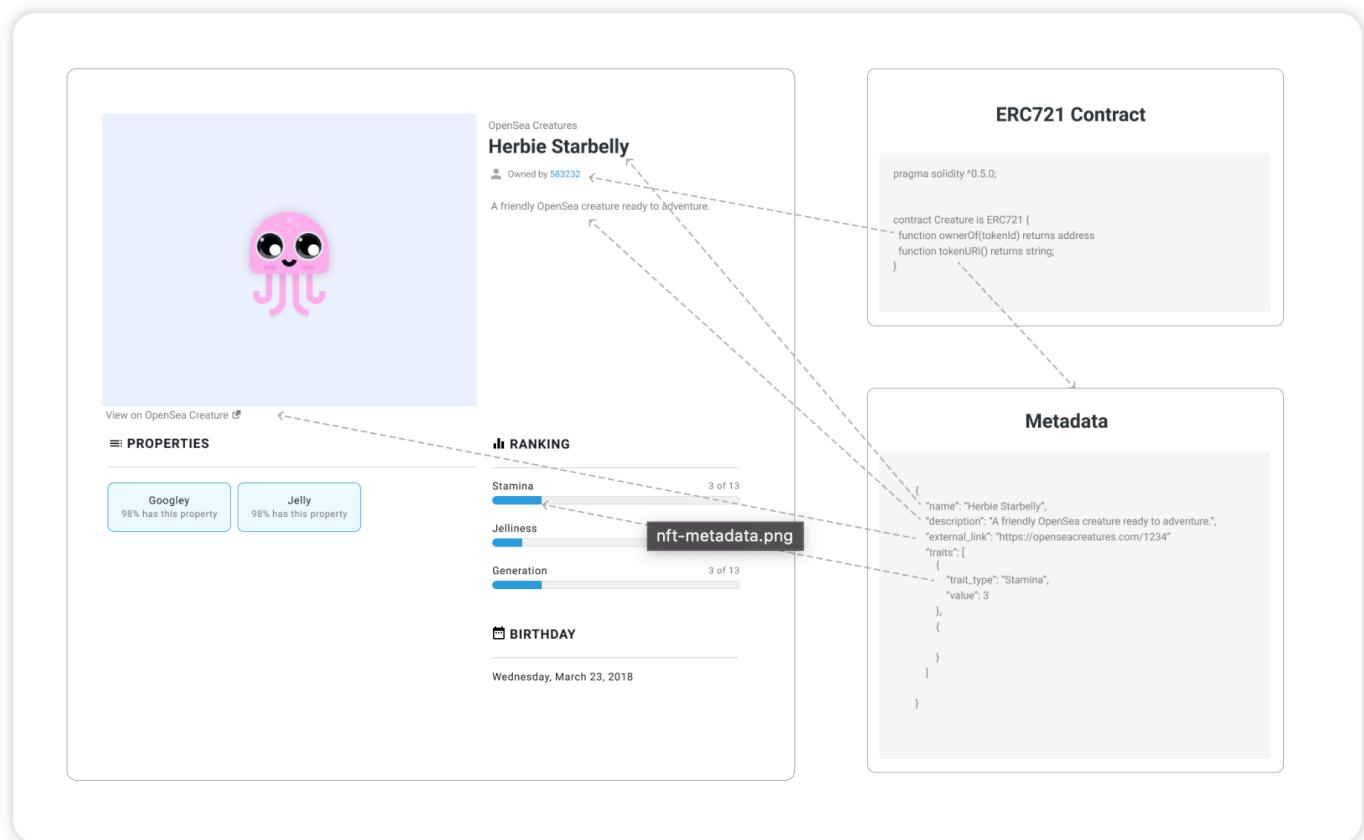
Al mismo tiempo, notar que se ha creado una carpeta que se llama `build` que contiene dos carpetas: `images` y `json`. En la primera carpeta se guardan las imágenes generadas del resultado de combinar capas. La segunda carpeta guarda la metadata de cada imagen generada.

Cada imagen posee su correspondiente archivo de metadata que ayuda a describirlo.

Al ejecutar el comando otra vez, se generará otro set de imágenes diferentes al anterior producto de la aleatoriedad.

¿Qué es la `metadata`?

Según la [documentación de Open Sea](#), la metadata de cada imagen guardada en los archivos json, ayuda a describir los atributos y características de cada NFT para poder entender mejor su naturaleza.



Dentro de cada archivo json hay un atributo llamado `attributes` que es un array de todos los atributos que describen dicha imagen en particular. Dichos atributos se verán reflejados en los diferentes marketplaces que pueden leer la matadato desde el front-end.

Al generar los archivos json, notar que en el array `attributes`, se han creado dos atributos más: `trait_type` y `value`. `trait_type` ha adquirido el nombre de una carpeta de capas y `value` adquirió el nombre de uno de los archivos de esa carpeta capas. Veamos el siguiente ejemplo:

```
"attributes": [
  {
    "trait_type": "Background",
    "value": "Black"
  },
  ...
]
```

Aquí, "Background" es el nombre de una carpeta capa y "Black" el nombre de un archivo de esa carpeta.

El archivo .json

Además de los atributos arriba mencionados, los archivos de formato `.json` generados en `build`, tienen otras propiedades que se muestran a continuación:

```
{  
  "name": "Your Collection #2",  
  "description": "Remember to replace this description",  
  "image": "ipfs://NewUriToReplace/2.png",  
  "dna": "57980f389a30ca9db9bf31c11ee8a2794dcbbb72",  
  "edition": 2,  
  "date": 1666894593527,  
  "attributes": [...]  
}
```

Siendo las más importantes:

- `name`: nombre de la colección
- `description`: descripción de la colección
- `image`: ruta en IPFS en donde la imagen está guardada (se actualizará después)

Modificar la rareza de cada capa

Supongamos que dentro de la carpeta `layers`, tenemos la siguiente capa `Eyeball` y dentro hay cuatro opciones de imágenes con su respectiva rareza denotada después del `#`:

```
Eyeball  
- Gold#10.png  
- Purple#20.png  
- Read#30.png  
- White#40.png
```

Se realiza una suma de todas las rarezas ($10 + 20 + 30 + 40$) y cada imagen aparecerá en una proporción de su rareza con respecto al total ($10/100, 20/100, 30/100, 40/100$). Es decir, `Purple#20` aparecerá dos veces más que `Gold#10`. `White#40` aparecerá dos veces más que `Purple#20` y así sucesivamente.

Crear las imágenes en orden aleatorio

Cuando se tienen varias configuraciones de capas en el objeto `layerConfigurations`, para lograr que las diferentes imágenes se produzcan en un orden aleatorio (y no en el orden en que fueron puestos en `layerConfigurations`), dentro del archivo `src/config.js`, alterar la siguiente variable.

```
const shuffleLayerConfigurations = true; // false => no cambiar el orden
```

Incrementar tamaño de imágenes

En el archivo `config.js` la variable `format` puede ser modificada para cambiar el tamaño de las imágenes. Incrementar o decrementar `width` o `height` para incrementar o decrementar la resolución de las imágenes:

```
const format = {
  width: 512,
  height: 512,
  smoothing: false,
};
```

Incluir background en las imágenes

En el caso en que no se tenga una capa de background para las imágenes, la librería lo puede generar. La siguiente variable dentro del archivo `config.js` nos ayudará con ello:

```
const background = {
  generate: true,
  brightness: "80%",
  static: false,
  default: "#000000",
};
```

Tolerancia para la repetición

Si no existe una cantidad suficiente para generar imágenes, el sistema puede arrojar un error si la sensibilidad es alta. Para incrementar o disminuir la tolerancia, usar la siguiente variable:

```
const uniqueDnaTorrance = 10000;
```

Indica la cantidad de imágenes únicas que deben crearse antes de arrojar un error.

Guardar imágenes en IPFS

Al correr el comando `node index.js`, se genera la carpeta `images` dentro de `build`. Abrimos la aplicación desktop de IPFS y nos dirigimos a `FILES` para poder arrastrar nuestra carpeta `images`.

The screenshot shows the IPFS desktop application interface. On the left sidebar, there are five tabs: STATUS, FILES, EXPLORE, PEERS, and SETTINGS. The FILES tab is currently selected. In the main area, there is a search bar at the top with the text "imagesEyes". Below it, a summary bar shows "4MiB FILES" and "51MiB ALL BLOCKS" with a "+ Import" button. The main content area displays a table with two rows. The first row has a checked checkbox, a folder icon, and the name "imagesEyes" followed by its CID: "QmfYqFm3Nygt...Eb2f". The second row has an unchecked checkbox, a folder icon, and the name "metadata" followed by its CID: "QmeZufhnX...adVne". At the bottom right, a modal window titled "Imported 20 items" shows a progress bar with "20 of 20" and a green checkmark.

Notemos que el CID para esta carpeta es la siguiente: `QmfYqFm3Nygt...Eb2f`. Este hash es importante porque apunta a la ubicación de nuestra carpeta dentro de IPFS y es requerido en los archivos json que contienen la metadata de las imágenes.

Con el CID hallado, nos podemos dirigir a la siguiente ruta y encontrar las imágenes en IPFS: <https://ipfs.io/ipfs/QmfYqFm3Nygt...Eb2f>

Actualizando metadata con CID

Tenemos que actualizar la metadata con el nuevo CID generado en IPFS. Pasamos de:

```
{  
  "name": "Your Collection #1",  
  "description": "Remember to replace this description",  
  "image": "ipfs://UPDATEDURI/1.png",  
  //...  
}
```

a lo siguiente:

```
{
  "name": "Your Collection #1",
  "description": "Remember to replace this description",
  "image": "ipfs://QmfYqFm3Nygt0X7kb7y9ukwV2Q9vF5UUdEKFUrCzn4Eb2f/1.png",
  //...
}
```

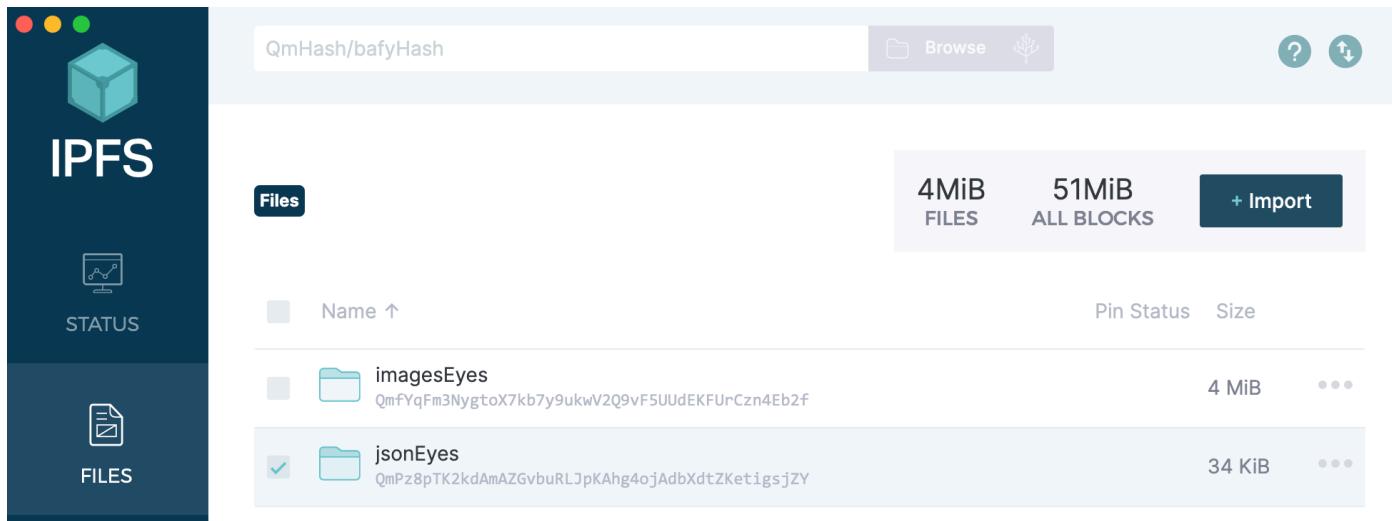
Para lograr actualizar el CID en todos los archivos json, nos dirigimos al archivo `config.js` y actualizamos la variable `baseUri`:

```
const baseUri = "ipfs://QmfYqFm3Nygt0X7kb7y9ukwV2Q9vF5UUdEKFUrCzn4Eb2f";
```

Luego, corremos el siguiente comando para que se vea reflejado en los archivos json de metadata `node utils/update_info.js`.

Subiendo metadata a IPFS

Al correr el comando, se actualiza los archivos json. Arrastrar la carpeta a IPFS y copiar el CID



Para el archivo de metadata, el CID que nos arroja IPFS es

`Qmpz8pTK2kdAmAZGvbuRLJpKAhg4ojAdbXdtZKetigsjZY`. Usaremos este valor dentro del smart contract.

Con el CID hallado, nos podemos dirigir a la siguiente ruta y encontrar los archivos metadata en IPFS: <https://ipfs.io/ipfs/Qmpz8pTK2kdAmAZGvbuRLJpKAhg4ojAdbXdtZKetigsjZY>

Pinata

En muchas circunstancias, es necesario usar un tercer servicio que nos ayuda a mantener la información disponible. Incluso cuando hay nodos que están fuera de línea, Pinata nos ayudará a mantener dichos recursos online.

Subir recursos

Para subir recursos a Pinata, es posible hacerlo también usando el CID obtenido en IPFS. Seguir la siguiente ruta en [Pinata Manager](#):

```
Upload + > * CID > IPFS CID to Pin > Copiar y Pegar el CID > Search and Pin
```

Ver recursos

Al finalizar la subida, se podrá visualizar los recursos en Pinata

The screenshot shows the Pinata dashboard. At the top, there's a purple header bar with the Pinata logo, a search bar, and user account information. Below the header, a banner says "Hey, Lee👋". The main area is titled "My Files" and contains a table of uploaded files. The columns are "Name", "CID", "Submarined", and "Actions". There are two files listed:

| Name | CID | Submarined | Actions |
|----------------|--|------------|---------|
| imagesEyes 🐈 | QmfYqFm3Nygt0X7kb7y9ukwV2Q9vF5UUdEKUrCzn4Eb2f | False | More |
| eyesMetadata 🐈 | QmPz8pTK2kdAmAZGvbuRLJpKAhg4ojAdbXdtZKetigsjZY | False | More |

At the bottom left of the table, there are navigation arrows.

Cabe notar que Pinata nos arroja diferentes links para acceder a los recursos:

<https://gateway.pinata.cloud/ipfs/QmfYqFm3Nygt0X7kb7y9ukwV2Q9vF5UUdEKUrCzn4Eb2f>

y

<https://gateway.pinata.cloud/ipfs/QmPz8pTK2kdAmAZGvbuRLJpKAhg4ojAdbXdtZKetigsjZY>

Ya sea através del link provisto por IPFS o Pinata, podemos acceder a los recursos guardados en IPFS.

Actualizando el Smart Contract

El contrato ERC721 posee un método que le permite leer los archivos de metadata guardados en IPFS. Dicho método es consultado para poder mostrar los atributos e imágenes guardados en IPFS.

Debemos modificar el siguiente método:

```
function _baseURI() internal pure returns (string memory) {
    return "ipfs://QmPz8pTK2kdAmAZGvbuRLJpKAhg4ojAdbXdtZKetigsjZY/";
}
```

Una vez puesto el CID que apunta a los archivos metadata, el contrato puede ser publicado.

Desarrollo de un script para deployment e interacción con el Smart Contract

Hardhat y Ethers trabajan en conjunto para ayudarnos en la publicación y verificación de smart contracts. En esta sección explicaremos el script usa las librerías Ethers y Hardhat parar lograr dichos objetivos.

Hay al menos tres partes para la publicación de un smart contract:

1. El archivo `hardhat.config.js`
2. Script de deployment (definido en la carpeta `scripts`)

3. Comando de deployment (usa una de las configuraciones de `hardhat.config.js`)

1- El archivo `hardhat.config.js`

Empecemos por entender el archivo `hardhat.config.js` en el cual se encuentra definido cierta configuración que nos ayudará con el deployment.

```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config();

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.9",
  networks: {
    matic: {
      url: process.env.MUMBAI_TESNET_URL,
      accounts: [process.env.ADMIN_ACCOUNT_PRIVATE_KEY],
      timeout: 0, // tiempo de espera para terminar el proceso
      gas: "auto", // limite de gas a gastar (gwei)
      gasPrice: "auto", // precio del gas a pagar (gwei)
    },
    etherscan: { apiKey: process.env.POLYGONSCAN_API_KEY },
  };
};
```

`solidity`: define la versión del compilador con la cual se compilarán los smart contracts a publicarse dentro del script de deployment.

`url`: aquí se añaden los URL de los RPC nodes. Hay dos tipos: públicos y privados. Los RPC privados por lo general son provistos en la documentación de cada Blockchain. Por ejemplo en [Wiki Polygon](#) se menciona que el RPC es el siguiente: <https://polygon-rpc.com/>. Este RPC se usa tanto para leer y escribir información del blockchain. Sin embargo, en mi experiencia, muchas veces los RPC públicos tienen problemas de latencia y errores inesperados (aunque con el tiempo se perfeccionan). Por ello, existen servicios privados de RPC de empresas que han construido sus propios nodos para conectarse al Blockchain. Por ejemplo [Alchemy](#), [Moralis](#), [Infura](#), entre otros. Cada uno tiene otras funcionalidades que complementan el acceso a nodos de Blockchain mediante `remote procedure calls` (RPC).

`accounts`: Hardhat requiere del uso de llaves privadas (una o más) para poder realizar firmas de los métodos que se llaman desde el script de deployment. Por ello, en `accounts` se agrega en un array la lista de llaves privadas. Desde el script de deployment, se puede acceder a cada una de las llaves públicas (`addresses`) de estas llaves privadas. Para lograr ello se usa el método: `var [cuenta1, cuenta2, ...] = await hre.ethers.getSigners();`. De este modo, se puede personalizar las firmas de los métodos a ser ejecutados desde el script de deployment.

`timeout`: Define la cantidad de tiempo a esperar antes de que el script de deployment falle. Si se especifica `0`, el tiempo a esperar es indefinido. Aunque se fije `0` en este campo, el escaner, que es con quien se interactúa para la publicación de smart contracts, puede devolver un `timeout`. En dichas situaciones la verificación si el smart contract se llegó a publicar o no se realizará manualmente.

`gas` : Define un cantidad límite de gas a gastar en cada transacción. Se puede usar particularmente cuando se tiene presupuestos de gas estrictos. Una de dichas transacciones a rastrear es el costo de publicar un smart contract en el blockchain. Si el campo `gas` tiene un número, se usa como límite. En cambio, si se define como `auto`, el gas se estima automáticamente para cada transacción.

`gasPrice` : Define un precio del gas a pagar para el total del gas a gastar en cada transacción. El valor de `gasPrice` está unidades de `wei`. Un `wei` es la más pequeña denominación de Ether. $1 \text{ Eth} = 10^{18} \text{ wei}$. Si `gasPrice` se define como `auto`, el precio se calcula automáticamente. De otro modo, el número definido en `gasPrice` será el que prima.

`etherscan` : Aquí se incluye la el `API KEY` que obtiene en el escáner de cada Blockchain (e.g. [PolygonScan](#)). Crear una cuenta en el Escáner y dirigirte a `API-KEYs` para poder crear una. Sin este `API KEY`, no se puede automatizar la verificación de smart contracts. El no tenerlo, no impide su publicación. Dicho `API KEY` puede ser obtenido en

2- El script de deployment e interacción

Desarrollar scripts implica el poder crear una automatización para publicar los smart contracts desarrollados y, lo más importante, poder ejecutar métodos del smart contract de configuración post publicación.

Este script de deployment, cuando es utilizado en testnets, facilita la publicación iterativa de smart contracts. Es ideal para el desarrollo incremental de funcionalidades de smart contracts, así como también agiliza la publicación de smart contracts luego del arreglo de bugs.

Script de deployment

Para comenzar, se crea un archivo dentro de la carpeta `scripts`. Vamos publicar dos smart contracts que tienen las siguientes interfaces:

```
// nombre del smart contract: MiPrimerToken
interface IMiPrimerToken {
    function mint(address to, uint256 amount) external;
    function burn(address to, uint256 amount) external;
}

// nombre del smart contract: AirdropONE
interface IAirdropONE {
    function participateInAirdrop() external;
    function quemarMisTokensParaParticipar() external;
    function addToWhitelist(address _account) external;
    function removeFromWhitelist(address _account) external;
    function setTokenAddress(address _tokenAddress) external;
}
```

Usaremos los métodos definidos en las interfaces como guía para poder ejecutarlos de ser necesarios. Vamos a publicar el contrato `AirdropONE.sol` que tiene una dependencia con `MiPrimerToken.sol` dado que el contrato de airdrop reparte tokens.

Dentro de la carpeta `scripts` creamos el archivo `deployAirdropOne.js` (`$ touch deployAirdropOne.js`). En la cabecera importamos `hardhat` en la variable `hre`. Si se usan variables secretas en `.env`, ejecutar `config()`. Dentro de la función `main` se desarrollarán

```
// deployAirdropOne.js
const hre = require("hardhat");
require("dotenv").config();

async function main() {
    // desarrollar el script
}

main().catch((error) => {
    console.error(error);
    process.exitCode = 1;
});
```

En primer lugar publicaremos el contrato `MiPrimerToken.sol`. Este smart contract tiene el siguiente constructor: `constructor (string memory _name, string memory _symbol)`, por lo tanto, ambos argumentos del constructor deben ser incluidos en el script de deployment.

```
// deployAirdropOne.js
async function main() {
    // Definiendo los argumentos del constructor: name y symbol
    var name = "Mi Primer Token"; // 1er argumento
    var symbol = "MPRTKN"; // 2do argumento

    // 'getContractFactory(arg: nombre del smart contract)'
    // - Obtiene el código del smart contract
    // - Se usa el nombre exacto del smart contract como arg
    var MiPrimerToken = await hre.ethers.getContractFactory("MiPrimerToken");

    // Se crea una instancia del contrato
    // Para su inicialización, se pasa los argumentos del constructor
    var miPrimerToken = await MiPrimerToken.deploy(name, symbol);

    // Obtenemos el address del smart contract
    console.log("MiPrimerToken Address", miPrimerToken.address);

    // El smart contract se publica en testnet/mainnet
    // Capturamos el resultado en 'tx'
    var tx = await miPrimerToken.deployed();

    // El smart contract es confirmado por cinco bloques
    // Necesario para poder interactuar con el smart contract
    // La publicación implica guardar el bytecode del smart contract en el blockchain
    // Esperar ayuda a propagar la información en todo el blockchain
    await tx.deployTransaction.wait(5);
```

```
}
```

```
// ...
```

Testing de Contratos Inteligentes

El testing de contratos inteligentes implica la validación y verificación de su lógica. Un contrato inteligente, una vez publicado, no tiene manera de ser corregido en caso se encontraran fallas técnicas. Ello implica poner mucho énfasis en el testeo de casos típicos y, especialmente, en los casos especiales o extremos.

Como guía, un contrato inteligente debería considerar las siguientes validaciones:

1. Lógica de funcionamiento de un método
2. Los `require` son disparados en las condiciones correctas
3. Los eventos se emiten en los métodos correctos con la información esperada
4. Los modifiers cumplen el papel correcto de protección o validación de lógica adicional no incluida en el método

Al usar ethers para el desarrollo de casos de prueba, los siguientes métodos son los más usados:

```
// 'something' debe ser true
expect(something).to.be.true;

// 'something' es igual a 'something2'
expect(something).to.be.equal(something2)

// la transacción 'tx' que pertenece al contrato 'smartContract'
// emite un evento del nombre 'nombreDelEvento'
// con los siguientes argumentos: arg1, arg2 y arg3
await expect(tx).to.emit(smartContract, nombreDelEvento).withArgs(arg1, arg2, arg3);

// el método 'métodoDeSmartContract' al ser llamado debe fallar
// con el mensaje de 'mensajeDeRequire'
await expect(métodoDeSmartContract).to.revertedWith(mensajeDeRequire)

// el valor 'value1' debe ser cercano a el valor 'value 2'
// con una diferencia por encima o por debajo de un valor 'delta'
expect(value1).to.be.closeTo(value2, delta);
```

Front-end y Smart Contracts

Vamos a construir el front-end para el contrato inteligente de AirdropOne. Este contrato de AirdropOne implica también el interactuar con el contrato del token ERC20.

Pasos:

1. Convertir los contratos en actualizables, tanto el AirdropOne como el token ERC20 (opcional)
2. Crear la librería agnóstica para interactuar desde el front-end con los smart contracts
3. Crear un front minimalista para interactuar con la librería de los smart contracts

Desarrollo:

1

Contrato token Acualizable:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";

contract TokenUpgradeableAirdrop is
    Initializable,
    ERC20Upgradeable,
    AccessControlUpgradeable,
    UUPSUpgradeable
{
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
    bytes32 public constant BURNER_ROLE = keccak256("BURNER_ROLE");

    /// @custom:oz-upgrades-unsafe-allow constructor
    constructor() {
        _disableInitializers();
    }

    function initialize(
        string memory _name,
        string memory _symbol
    ) public initializer {
        __ERC20_init(_name, _symbol);
        __AccessControl_init();
        __UUPSUpgradeable_init();

        _mint(msg.sender, 100000 * 10 ** decimals());

        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(PAUSER_ROLE, msg.sender);
        _grantRole(MINTER_ROLE, msg.sender);
    }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }

    function burn(address from, uint256 amount) public onlyRole(BURNER_ROLE) {
```

```

        _burn(from, amount);
    }

    function _authorizeUpgrade(
        address newImplementation
    ) internal override onlyRole(MINTER_ROLE) {}
}

```

Contrato Airdrop actualizable:

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";

interface IMiPrimerTKN {
    function mint(address to, uint256 amount) external;

    function burn(address from, uint256 amount) external;

    function balanceOf(address account) external view returns (uint256);
}

contract AirdropONEUpgradeable is
    Initializable,
    PausableUpgradeable,
    AccessControlUpgradeable,
    UUPSUpgradeable
{
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");

    uint256 public constant totalAirdropMax = 10 ** 6 * 10 ** 18;
    uint256 public constant quemaTokensParticipar = 10 * 10 ** 18;

    uint256 airdropGivenSoFar;

    address public miPrimerTokenAdd;

    mapping(address => bool) public whiteList;
    mapping(address => bool) public haSolicitado;

    /// @custom:oz-upgrades-unsafe-allow constructor
    constructor() {
        _disableInitializers();
    }
}

```

```

function initialize(address _tokenAddress) public initializer {
    miPrimerTokenAdd = _tokenAddress;
    __AccessControl_init();
    __UUPSUpgradeable_init();

    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(PAUSER_ROLE, msg.sender);
}

function participateInAirdrop() public whenNotPaused {
    // lista blanca
    require(whiteList[msg.sender], "No esta en lista blanca");

    // ya solidito tokens
    require(!haSolicitado[msg.sender], "Ya ha participado");

    // pedir numero random de tokens
    uint256 tokensToReceive = _getRandomNumberBelow1000();

    // verificar que no se exceda el total de tokens a repartir
    require(
        airdropGivenSoFar + tokensToReceive <= totalAirdropMax,
        "No hay tokens disponibles"
    );

    // actualizar el conteo de tokens repartidos
    airdropGivenSoFar += tokensToReceive;
    // marcar que ya ha participado
    haSolicitado[msg.sender] = true;

    // transferir los tokens
    IMiPrimerTKN(miPrimerTokenAdd).mint(msg.sender, tokensToReceive);
}

function quemarMisTokensParaParticipar() public whenNotPaused {
    // verificar que el usuario aun no ha participado
    require(haSolicitado[msg.sender], "Usted aun no ha participado");

    // Verificar si el que llama tiene suficientes tokens
    uint256 balanceToken = IMiPrimerTKN(miPrimerTokenAdd).balanceOf(
        msg.sender
    );
    require(
        balanceToken >= quemaTokensParticipar,
        "No tiene suficientes tokens para quemar"
    );

    // quemar los tokens
    IMiPrimerTKN(miPrimerTokenAdd).burn(msg.sender, quemaTokensParticipar);
}

```

```
// dar otro chance
haSolicitado[msg.sender] = false;
}

////////////////////////////// HELPER FUNCTIONS //////////////////

function addToWhiteList(
    address _account
) public onlyRole(DEFAULT_ADMIN_ROLE) {
    whiteList[_account] = true;
}

function removeFromWhitelist(
    address _account
) public onlyRole(DEFAULT_ADMIN_ROLE) {
    whiteList[_account] = false;
}

function pause() public onlyRole(PAUSER_ROLE) {
    _pause();
}

function unpause() public onlyRole(PAUSER_ROLE) {
    _unpause();
}

function _getRandomNumberBelow1000() internal view returns (uint256) {
    uint256 random = (uint256(
        keccak256(abi.encodePacked(block.timestamp, msg.sender))
    ) % 1000) + 1;
    return random * 10 ** 18;
}

function setTokenAddress(address _tokenAddress) external {
    miPrimerTokenAdd = _tokenAddress;
}

function _authorizeUpgrade(
    address newImplementation
) internal override onlyRole(DEFAULT_ADMIN_ROLE) {}

}
```

Contratos Actualizables

El código compilado (bytecode) de un contrato inteligente no puede ser cambiado una vez que es publicado. Para ciertos casos de uso, esto es ideal dado que se asegura la inmutabilidad de las reglas planteadas en el contrato inteligente. Lo cual crea un carácter de predictibilidad a las transacciones. Sin embargo, otros protocolos más complejos y evolutivos podrían requerir mayor flexibilidad.

La estrategia de Proxy permite a los desarrolladores reemplazar la lógica de un contrato incluso después de que se haya publicado. Se utiliza la estrategia Proxy para los siguientes escenarios:

- Desarrollo orgánico (evolutivo): si se logra reemplazar la lógica de un contrato ya publicado para agregar mayores funcionalidades, se crea valor para el proyecto. Incluso más si ello no implica tener que migrar los usuarios, balances y demás información guardada en el smart contract.
- Publicaciones más económicas: en ciertos escenarios se publican múltiples veces el mismo smart contract. Usando la estrategia Proxy, se puede, significativamente, reducir el costo de publicación. Ello porque la lógica del contrato solo tiene que ser publicada una sola vez.
- Límites en el tamaño de código: Solo se pueden publicar smart contracts que estén por debajo de 24KB en peso, lo cual puede representar una gran limitante para proyectos grandes. Cuando la lógica se espalda en múltiples contratos, se puede circunvalar esa limitación.

¿Cómo funciona?

La estrategia de proxys se basa en la utilización de dos funcionalidades de Solidity: `delegatecall` y `fallback functions`.

`delegatecall()`

Una llamada normal de función a otro contrato (objetivo) en Solidity usará internamente `address.call()` o `address.staticcall()`. Cualquier de estas dos funciones serán ejecutadas en el contexto del contrato que es llamado. Esto significa que estará accediendo a su propio estado (`address`, `storage`, `balances`, etc.)

Sin embargo, si se llegara a usar `address.delegatecall()`, se llegaría a ejecutar el código o lógica definido en `address` pero en el contexto del contrato que está llamando. Esto significa que se utilizarán el `storage` del contrato que está realizando la llamada. Se realizarán lecturas y escrituras en el contrato que hace la llamada. Es como si se reemplazara el código del contrato que está llamando por el código del contrato siendo llamado dentro del contexto del primero.

Indagaremos `address.delegatecall()` con un ejemplo. Notar que para que esta estrategia funcione el `storage layout` (orden en que se definen las variables y los tipos de variables) debe ser el mismo en todos los smart contracts a usarse.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

// REQUISITO: Ambos contratos tienen el mismo storage layout
// Publicar en primer lugar el contrato 'contratoASerLlamado'

// En el contrato 'contratoQueVaA_llamar',
// se ejecuta el método 'addToWhitelist'
// Al hacerlo, se escribe sobre 'contratoQueVaA_llamar'
```

```

// usando el código de 'contratoASerLlamado'

contract contratoASerLlamado {
    mapping(address => bool) public whitelist;
    function addToWhitelist(address _account) external {
        whitelist[_account] = true;
    }
}

contract contratoQueVaALLamar {
    mapping(address => bool) public whitelist;
    function addToWhitelist(address _scAddress, address _account) external {
        (bool success, bytes memory data) = _scAddress.delegatecall(
            abi.encodeWithSignature("addToWhitelist(address)", _account)
        );

        if (!success) {
            revert(string(data));
        }
    }
}

```

En este ejemplo:

1. El address del deployer para este ejemplo es: `0x5B38Da6a701c568545dCfcB03FcB875f56beddC4`
2. Publicamos el contrato `contratoASerLlamado`. Obtenemos su address. Digamos que es `0xd9145CCE52D386f254917e481eB44e9943F39138`.
3. Publicamos el otro contrato `contratoQueVALlamar` y obtenemos el address `0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8`
4. Desde el contrato `contratoQueVaALLamar` ejecutamos el método `addToWhitelist(address _scAddress, address _account)` con los siguientes argumentos:
`addToWhitelist(0xd9145CCE52D386f254917e481eB44e9943F39138, 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4)`. Es decir, como primer argumento pasamos el address del smart contract que será llamado. Como segundo argumento pasamos el address del deployer.
5. Dado que el mapping `whitelist` es público ambos contratos, podemos usar su getter para poder leer su información. Haré la consulta haciendo uso del address del deployer para ver los resultados en ambos smart contracts.
6. Resultado en `contratoASerLlamado : mapping(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4)`.
 Resultado false.
7. Resultado en `contratoQueVaALLamar : mapping(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4)`.
 Resultado true.

De esta manera, hemos logrado ejecutar el código de un contrato dentro del contexto de otro contrato a través de `delegatecall()`.

The fallback function

En Solidity, un `fallback function` es ejecutado si es que ninguna de las otras funciones coincide con el identificador de función siendo llamado. Hay dos maneras de definir el `fallback function`:

1. `fallback() external [payable]`
2. `fallback(bytes calldata _input) external [payable] returns(bytes memory _output)`

En ambos casos se debe marcar como `external` y no se debe agregar el keyword `function` antes de definir el `fallback function`. Además, éste, puede ser `virtual`, ser sobreescrito y también llevar uno o varios `modifier`.

En primer lugar entendamos cómo funciona el método `fallback` en un smart contract. Desde el contrato `Llamante`, llamaremos a un método inexistente en `ContratoConFallback`. Ello para demostrar que cuando ningún método de un smart contract coincide con el que está siendo llamado, su `fallback function` atrapa esa llamada.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

interface IContratoConFallback {
    function metodoNoExiste() external;
}

// Nosotros
contract Llamante {
    // Ambos métodos producen los mismos efectos
    function llamarContratoFallback(address _scAddress) external {
        IContratoConFallback(_scAddress).metodoNoExiste();
    }
    function llamarContratoFallback2(address _scAddress) external {
        (bool success, ) = _scAddress.call(
            abi.encodeWithSignature("metodoNoExiste()")
        );
        require(success, "Error en la llamada al Proxy");
    }
}

// Proxy
contract ContratoConFallback {
    uint256 public counter;
    fallback() external {
        counter++;
    }
}
```

En este ejemplo:

1. Publicamos el contrato `ContratoConFallback` y obtenemos el siguiente address:
`0xD4Fc541236927E2EAf8F27606bD7309C1Fc2cbee`.
2. Publicamos el contrato `Llamante` y obtenemos el siguiente address:

```
0x5FD6eB55D12E759a21C09eF703fe0CBa1DC9d88D.
```

3. En el contrato `Llamante` ejecutamos el método `llamarContratoFallback` usando como argumento el address de `ContratoConFallback`:

```
llamarContratoFallback(0xD4Fc541236927E2EAf8F27606bD7309C1Fc2cbee)
```

4. Al ejecutar ese método, estamos llamando un método inexistente en `ContratoConFallback`, lo cual será atrapado por su `fallback function`. Al ser llamado el `fallback`, el counter se incrementará en cada vez.

Vamos un paso más allá. Para poder utilizar la estrategia de Proxies, necesitamos hacer uso del argumento y valor de retorno de la función `fallback`. Veamos cuáles son:

```
fallback(bytes calldata _input) external [payable] returns(bytes memory _output)
```

- `_input`: captura toda la data enviada al contrato. Es decir, si por ejemplo, se intentase ejecutar el método inexistente `.transfer(320)` de un smart contract, dentro del `fallback`, `_input` representaría el ABI-encoded de esa llamada. Ello incluye tener el selector del método más los argumentos pasados a dicho método.
- `_output`: tener `returns` en el `fallback` permite que el consumidor de éste método pueda obtener un resultado a raíz de cualquier proceso que suceda dentro de la función `fallback`, incluso si ello implica hacer una llamada a otro contrato que a su vez retorna valores. Dentro del `fallback`, el valor del `_output` puede ser asignado para representar el valor de retorno del `fallback`.

Nota: incluir el keyword `payable` es opcional en caso se desee recibir Ether a través del `fallback function`. Si este es el caso, solo se limitará a gastar 2300 de gas para como presupuesto para ejecutar el método `fallback`. Sin embargo, es recomendable usar el método `receive` para habilitar que el smart contract reciba Ether. Usar `receive` y `fallback` en el mismo smart contract, ayuda a diferenciar el uso particular de cada uno. Con `receive` se podrá recibir Ether en el smart contract y claramente quitarle ese rol al `fallback function`.

En el siguiente ejemplo veremos cómo usar la función `fallback` usando su argumento y valor de retorno:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

// Nosotros
contract Llamante {
    bytes public argCodified = abi.encodeWithSignature(
        "metodoNoExiste(uint256)", 123456
    );
    uint256 public resultFromCallback;

    function llamarContratoFallback2(address _scAddress) external {
        (bool success, bytes memory output) = _scAddress.call(argCodified);
        require(success, "Error en la llamada al Proxy");

        // decodificando el resultado que viene de fallback
        resultFromCallback = abi.decode(output, (uint256));
    }
}
```

```

}

// Proxy
contract ContratoConFallback {
    uint256 public counter;
    bytes public input;

    fallback(bytes calldata _input) external returns(bytes memory _output){
        input = _input;
        counter++;
        _output = abi.encode(counter);
    }
}

```

- Al hacer llamadas intercontratos, internamente dicha llamada se realizar usando `.call`. Para explicar este ejemplo, hacemos una llamada intercontrato en `_scAddress.call(argCodified)`. Aquí `argCodified` es un argumento codificado que incluye tanto la función (selector) como también sus argumentos.
- En solidity se puede codificar la función junto con sus argumentos usando `abi.encodeWithSignature`. Usando este método, juntamos en un solo valor (del tipo `bytes`) tanto el método a ser llamada como los argumentos que recibe ese método. Por ello hacemos lo siguiente:

```

bytes public argCodified = abi.encodeWithSignature(
    "metodoNoExiste(uint256)", 123456
);

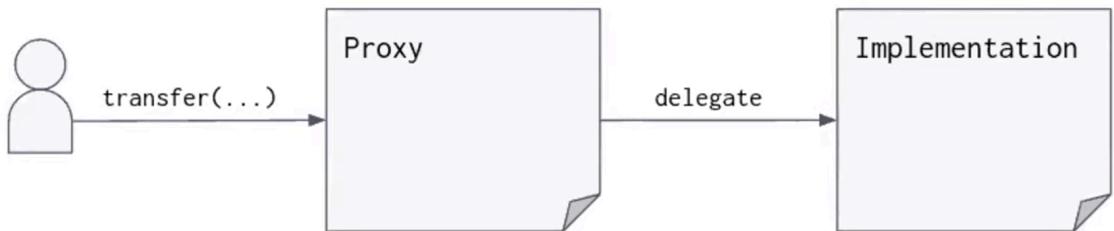
```

- Al ejecutar el método `llamarContratoFallback2` en el contrato `Llamante`, se realiza la llamada intercontrato. Al usar el método `.call` en `_scAddress.call`, obtenemos dos resultados: `bool success` y `bytes memory output`. Estos dos resultados tienen dos escenarios posibles:
 - Llamada intercontrato exitosa: en este caso, el primer valor de retorno (`bool success`), será `true`. Adicionalmente, el segundo valor de retorno (`bytes memory output`), podría contener un valor de retorno del método que fue llamada en el otro contrato. Éste valor de retorno estará codificado y para decodificarlo, se puede usar `abi.decode`.
 - Llamada intercontrato fallida: aquí, el primer valor de retorno (`bool success`), será `false`. El segundo valor de retorno, (`bytes memory output`), podría contener el error si éste se dio en un `require` o `revert` en el método del contrato que fue llamado. Para poder propagar el error dentro del contrato `Llamante`, se requiere del uso de `assembly`.

El patrón Proxy

El usuario en vez de interactuar con el contrato de lógica, se comunica con el contrato Proxy. El contrato proxy delega el llamado al contrato de lógica.

Proxy Pattern



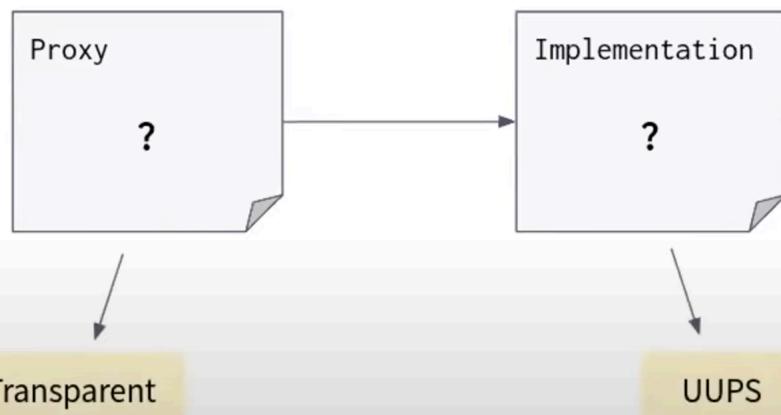
El contrato de implementación puede ser reemplazado por futuras versiones cuando se desee. Eso se viene a llamar la actualización de un contrato.

El contrato proxy contiene dentro el address del contrato a donde está delegando las llamadas. Con el objetivo de actualizar la lógica a través del tiempo, debe existir un método que permite actualizar el address del contrato lógica.

Dicho método es `upgradeTo(newImplementation) onlyAdmin` y se define dentro de uno de los smart contracts (ya sea en el Proxy o en la implementación). Como se puede notar, este método debe estar protegido de alguna manera.

De hecho, dependiendo en dónde pongámos este método, utilizaremos un tipo diferente de actualización.

Upgrade - Where?

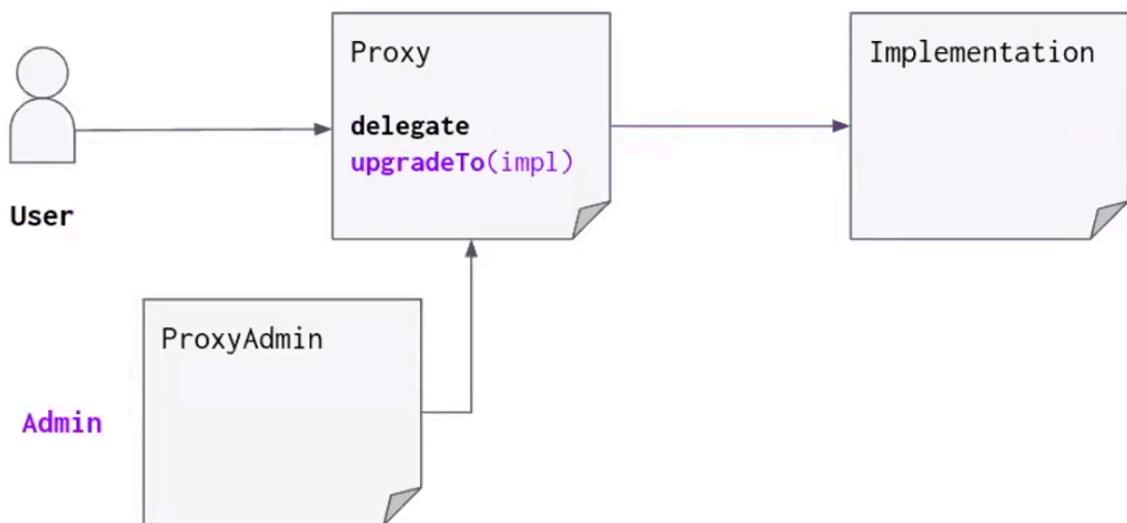


Si dicho método es puesto en el Proxy, usamos el tipo `Transparent`. Si lo colocamos en la `Implementation`, se trataría del tipo `UUPS`.

Transparent Proxy

En este tipo, el contrato Proxy, además de delegar las llamadas al contrato de implementación, también contiene la lógica de actualización del smart contract. El método `upgradeTo`, se define aquí. Si el admin llama el proxy, se le muestra la interface de actualización. Si otro usuario que no sea el admin llama al proxy, se muestra la interface de delegación. Adicional al Proxy y la Implementación, en este tipo, se añade un tercer smart contract que es el Admin smart contract.

Transparent Proxy



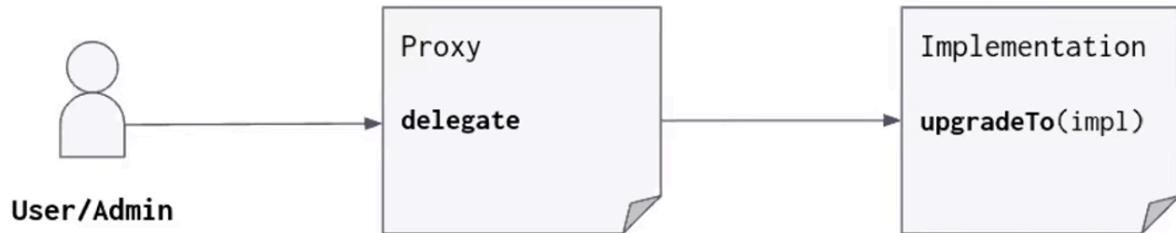
Es este smart contract que decide qué interfaz mostrar al que se está comunicando con el Proxy smart contract. Una deficiencia de este este método, es que el Admin Smart Contract tiene que comunicarse con el Proxy para poder determinar quién lo está llamando. El Proxy guarda las interfaces y dependiendo de quién lo llama, mostrará una interfaz diferente. Esta lógica tiene que suceder cada vez que alguien llama al Poxy. Esto incrementa los costos de acceso.

Después de las actualizaciones de Istanbul and Berlin, los costos para guardar en storage se incrementaron.

UUPS Proxy

En la propuesta EIP-1822 se documenta el tipo de Proxy en el que la lógica de actualización se guarda en la Implementación.

UUPS



La razón por la cual se puede definir el método de actualización `upgradeTo` en la Implementación, es porque a través de `delegatecall` se puede ejecutar un método de smart contract dentro de otro.

Ello implica que cada vez que se crea una nueva versión de implementación, el método `upgradeTo` se tiene que volver a incluir. No incluirlo eliminaría la posibilidad de seguir actualizando los contratos en el futuro.

Transparent vs UUPS

Cost Comparison

| | Transparent | UUPS |
|---------------------------|---------------------------|--------|
| Proxy Deployment | 740k + 480k ProxyAdmin | 390k |
| Implementation Deployment | + 0 | + 320k |
| Runtime Overhead | 7.3k | 4.9k |

(unit = gas)

1. Proxy Deployment: Usando el método `Transparent` es más costoso dados los tres contratos requeridos para su funcionamiento.

2. Implementation Deployment: En el método `UUPS`, la lógica de actualización se incluye en la implementación y las futuras actualizaciones. Ello lo hace más costoso.
3. Runtime overhead: En el método `Transparent`, hay una comunicación entre tres contratos, lo cual lo hace más costoso. Mientras que en `UUPS`, solo hay dos contratos en comunicación. Ello genera el ahorro en gas.

Contratos Actualizables

Empezaremos desde cero en la creación de un contrato actualizable. A cada paso explicaremos las modificaciones necesarias para convertir un contrato no actualizable a uno que sí lo es.

¿Qué herramientas requerimos?

- OpenZeppelin Upgradeable Contracts
 - UUPSUpgradeable
 - Ownable
- OpenZeppelin Upgrades Plugins
 - Chequeos de seguridad
 - Publicación

Utilizaremos una librería adicional llamada `@openzeppelin/hardhat-upgrades`. Corremos los siguientes comandos en el terminal:

```
$ npm install --save-dev @openzeppelin/hardhat-upgrades
```

```
$ npm install --save @openzeppelin/contracts-upgradeable
```

En el archivo `hardhat.config.js`, incluimos en la cabecera la siguiente línea que importará los métodos necesarios para hacer publicación de contratos inteligentes actualizables:

```
require("@openzeppelin/hardhat-upgrades");
```

¿Cómo convertir a un smart contract en actualizable?

1

Dirigirte al wizard y crear el contrato de un token ERC20.

```
// UpgradeableToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

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

contract UpgradeableToken is ERC20 {
    constructor() ERC20("UpgradeableToken", "UPGRDTKN") {
        _mint(msg.sender, 100000 * 10 ** decimals());
    }
}
```

2

Desarrollamos un conjunto de pruebas simples que verifican el nombre y símbolo del smart contract (que aún no es actualizable)

```
// testUpgradeableToken.js
const { expect } = require("chai");

describe("UPGRADEABLE TOKEN", function () {
  var UpgradeableToken;
  var upgradeableToken;
  var name = "UpgradeableToken";
  var symbol = "UPGRDTKN";

  describe("Set Up", () => {
    it("Publicar los contratos", async () => {
      UpgradeableToken = await hre.ethers.getContractFactory(
        "UpgradeableToken"
      );
      upgradeableToken = await UpgradeableToken.deploy();
    });
  });

  describe("Nombre y símbolo", () => {
    it("Verifica nombre del token", async () => {
      var nameToken = await upgradeableToken.name();
      expect(nameToken).to.be.equal(name);
    });

    it("Verifica symbolo del token del token", async () => {
      var symbolToken = await upgradeableToken.symbol();
      expect(symbolToken).to.be.equal(symbol);
    });
  });
});
```

Ejecutamos este test y evaluamos que las pruebas pasen: `npx hardhat test ./testUpgradeableToken.js`.

Hasta aquí, notaremos que estamos ejecutando las pruebas de un contrato no actualizable.

Ahora, incorporemos el script de los contatos actualizables. Realicemos el siguiente reemplazo:

```
// upgradeableToken = await UpgradeableToken.deploy();
upgradeableToken = await hre.upgrades.deployProxy(UpgradeableToken);
```

Al hacerlo, estamos indicando que el contrato `UpgradeableToken` se convierta en actualizable. Además, estamos usando otro método de `hardhat` que es `upgrades.deployProxy`.

Sin embargo, notaremos que nos arroja el siguiente error:

```
Error: Contract `contracts/UpgradeableToken.sol:UpgradeableToken` is not upgrade safe

contracts/UpgradeableToken.sol:7: Contract `UpgradeableToken` has a constructor
  Define an initializer instead
  https://zpl.in/upgrades/error-001

@openzeppelin/contracts/token/ERC20/ERC20.sol:54: Contract `ERC20` has a constructor
  Define an initializer instead
  https://zpl.in/upgrades/error-001
```

Este error se basa en el hecho que los contratos actualizables no pueden tener constructores. Ello por el simple hecho que los constructores no se ejecutarán en el contexto del Proxy. Es por ello que se necesita un método especial para las inicializaciones.

Retoquemos el contrato del token `UpgradeableToken.sol` para ajustarlo a un contrato actualizable.

```
// UpgradeableToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";

contract UpgradeableToken is Initializable, ERC20Upgradeable {
    function initialize() public initializer {
        __ERC20_init("UpgradeableToken", "UPGRDTKN");
        _mint(msg.sender, 100000 * 10 ** decimals());
    }
}
```

Veamos los cambios realizados hasta el momento:

1. Cambiamos el constructor por otro método llamada `initialize`. Se define como `public` y además tiene un modificador llamado `initializer`. A través de este modificador garantizamos que este método sea llamado una sola vez (como si fuera un constructor), solo que en este caso este método sí se ejecutará en el contexto del Proxy.
2. El contrato `UpgradeableToken` ahora hereda dos contratos que son apropiados para convertir el token en actualizable. `Initializable` provee el modificador `initializer`. `ERC20UPgradeable` es un contrato de token ERC20 adaptado para no tener constructor. Es por ello que se usa un método especial `__ERC20_init` para configurar el nombre y símbolo del token.

Luego de realizar estos cambios, volvemos a ejecutar los tests con el comando `npx hardhat test ./testUpgradeableToken.js` y deberían estar pasando.

Cabe notar que el tipo de Proxy que hemos usado hasta el momento es del tipo `Transparent`. Sin embargo, ahora se considera una buena práctica usar `UUPS`. Ello implica un ligero cambio en nuestro script de deployment.

```
// upgradeableToken = await hre.upgrades.deployProxy(UpgradeableToken);
upgradeableToken = await hre.upgrades.deployProxy(UpgradeableToken, {
    kind: "uups",
});
```

De este modo, hacemos explícito el usar el modelo `UUPS` para la publicación. Volvemos a ejecutar el comando `npx hardhat test ./testUpgradeableToken.js` y obtendremos el siguiente error:

```
Error: Contract `contracts/UpgradeableToken.sol:UpgradeableToken` is not upgrade safe

contracts/UpgradeableToken.sol:7: Implementation is missing a public `upgradeTo(address)` function

Inherit UUPSUpgradeable to include this function in your contract
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol
https://zpl.in/upgrades/error-008
```

Dado que estamos usando el modelo `UUPS`, el error nos sugiere que nos falta implementar el método llamado `upgradeTo(address)`, el cual nos garantizará que nuestro smart contract sea actualizable en el futuro. El error también nos sugiere que incluyamos `UUPSgradeable` en el contrato.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract UpgradeableToken is
    Initializable,
    ERC20Upgradeable,
    UUPSUpgradeable,
    OwnableUpgradeable
{
    function initialize() public initializer {
        __ERC20_init("UpgradeableToken", "UPGRDTKN");
        _mint(msg.sender, 10000 * 10 ** decimals());
    }

    function _authorizeUpgrade(
        address newImplementation
    ) internal override onlyOwner {}
}
```

Al incluir el contrato `UUPSUpgradeable` como sugerido, nos pedirá a su vez que definamos el método `_authorizeUpgrade`. Sin embargo, para poder proteger este método de ser llamado por cualquier address, utilizamos el modifier `onlyOwner` para poder protegerlo. Esto último nos conlleva a importar también el contrato `OwnableUpgradeable.sol`, que es la versión actualizable de `Ownable.sol`. Ahora sí podemos correr los tests y deberían pasar sin problemas.

Ahora definamos una nueva implementación para nuestro token. En esta nueva lógica, incluyamos el método `mint` que antes no estaba incluido. Esta nueva lógica se vería así:

```
contract UpgradeableToken2 is
    Initializable,
    ERC20Upgradeable,
    UUPSUpgradeable,
    OwnableUpgradeable
{
    function initialize() public initializer {
        __ERC20_init("UpgradeableToken", "UPGRDTKN");
        _mint(msg.sender, 100000 * 10 ** decimals());
    }

    // nuevo método
    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }

    function _authorizeUpgrade(
        address newImplementation
    ) internal override onlyOwner {}
}
```

Ahora, definamos la manera en cómo actualizar el contrato. La nueva lógica de deployment se vería así:

```
const { expect } = require("chai");

describe("UPGRADEABLE TOKEN", function () {
    var UpgradeableToken;
    var upgradeableToken;
    var UpgradeableToken2;
    var upgradeableToken2;
    var name = "UpgradeableToken";
    var symbol = "UPGRDTKN";

    var owner, alices;

    before(async () => {
        [owner, alices] = await hre.ethers.getSigners();
    });
})
```

```

describe("Set Up", () => {
  it("Publicar los contratos", async () => {
    UpgradeableToken = await hre.ethers.getContractFactory(
      "UpgradeableToken"
    );
    // upgradeableToken = await UpgradeableToken.deploy();
    // upgradeableToken = await hre.upgrades.deployProxy(UpgradeableToken);
    upgradeableToken = await hre.upgrades.deployProxy(UpgradeableToken, {
      kind: "uups",
    });
  });

  var implmntAddress = await upgrades.erc1967.getImplementationAddress(
    upgradeableToken.address
  );
  console.log("El Proxy address es (V2):", upgradeableToken.address);
  console.log("El Implementation address es (V2):", implmntAddress);
});
});

describe("Nombre y simbolo", () => {
  it("Verifica nombre del token", async () => {
    var nameToken = await upgradeableToken.name();
    expect(nameToken).to.be.equal(name);
  });

  it("Verifica simbolo del token del token", async () => {
    var symbolToken = await upgradeableToken.symbol();
    expect(symbolToken).to.be.equal(symbol);
  });
});

describe("Actualiza Smart Contract", () => {
  it("Publica Smart Contract", async () => {
    UpgradeableToken2 = await hre.ethers.getContractFactory(
      "UpgradeableToken2"
    );
    upgradeableToken2 = await hre.upgrades.upgradeProxy(
      upgradeableToken,
      UpgradeableToken2
    );
    var implmntAddress = await upgrades.erc1967.getImplementationAddress(
      upgradeableToken.address
    );
    console.log("El Proxy address es (V2):", upgradeableToken2.address);
    console.log("El Implementation address es (V2):", implmntAddress);
  });
});

```

```

it("Ejecuta el método mint de V2", async () => {
  var MIL_TOKENS = hre.ethers.utils.parseEther("1000");
  await upgradeableToken2.mint(alice.address, MIL_TOKENS);

  expect(await upgradeableToken2.balanceOf(alice.address)).to.be.equal(
    MIL_TOKENS
  );
  console.log(
    (await upgradeableToken2.balanceOf(owner.address)).toString()
  );
});
});
});
});
});

```

Lo más resaltante a notar es cómo a través de `hre.upgrades.upgradeProxy`, podemos volver a publicar una nueva lógica para nuestro contrato `UpgradeableToken`.

DEFI

Si hoy en día podemos enviar un correo electrónico a cualquier persona, ¿por qué no podemos enviar dinero así de fácil? ¿O por qué no podemos ofrecerles un préstamo del mismo modo? Estos son los fundamentos de las finanzas descentralizadas.

¿Qué es DeFi?

Es un ecosistema de aplicaciones financieras que se construyen encima del blockchain. Su principal propuesta de valor es la de operar de una manera descentralizada. Ello implica que no existan intermediarios como bancos, proveedores de servicios pago o fondos de inversión.

Hasta el momento van acumulados más de dos billones de dólares en aplicaciones DeFi que ofrecen los siguientes servicios: préstamos, servicios de cambio de criptomonedas, creación de monedas estables, tokenización de servicios y activos, derivados y otros instrumentos financieros. En general se tratan de servicios financieros ofrecidos a través del blockchain que remueven al intermedio.

¿Cómo se construyen los servicios financieros?

El ladrillo más pequeño en el mundo de DeFi son los contratos inteligentes. Estos contratos inteligentes posee código que ejecuta las funciones financieras principales. Dichas funciones son las siguientes: pagos a través de criptomonedas, créditos, apalancamientos, intercambios de criptomonedas, otorgamiento de seguro, trading, entre otras. Todas las funciones se ofrecen de manera descentralizada y sin intermediarios.

Los smart contracts que ofrecen estas funciones financieras principales se les conoce como primitivos financieros. Es más, cualquier producto establecido en el actual mundo financiero, puede tener su paralelo en DeFi. En teoría, se puede programar una gran variedad de casos de uso.

El valor creado por DeFi es que estos primitivos financieros permiten crear ecosistemas de servicios. Se juntan como piezas de lego según se necesite.

Stack de DeFi

Decentralized Finance (DeFi) Stack: Product & Application View



Source: StakingRewards.com

DeFi vs Wall Street

DeFi se presenta como una alternativa al sistema financiero tradicional. Está construido desde sus raíces de una manera descentralizada, de bajo costo, automatizada y anti censura.

En DeFi no se necesita tener permiso y cualquier persona puede contribuir con código de nuevas funcionalidades o usar los protocolos de código abierto existentes. No importa el estatus social o el lugar de origen, que sí son importantes y determinan el grado de acceso a diferentes servicios bancarios en el sistema financiero tradicional. Por ejemplo, ciertos derivados solo pueden ser manejados por grandes instituciones financieras.

En DeFi, todas las transacciones son transparentes y pueden ser rastreadas, mientras que en Wall Street, las instituciones funcionan como cajas negras y los usuarios confían en ellas, aunque éstas mismas son altamente eficientes.

Capas en DeFi

DeFi se puede dividir en cinco diferentes capas (según Schär):

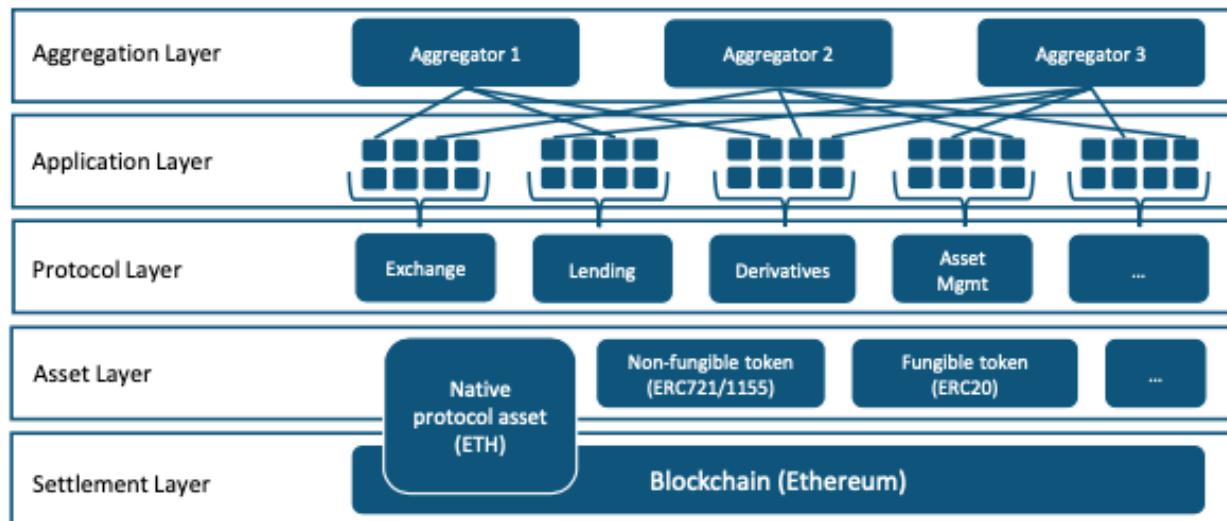
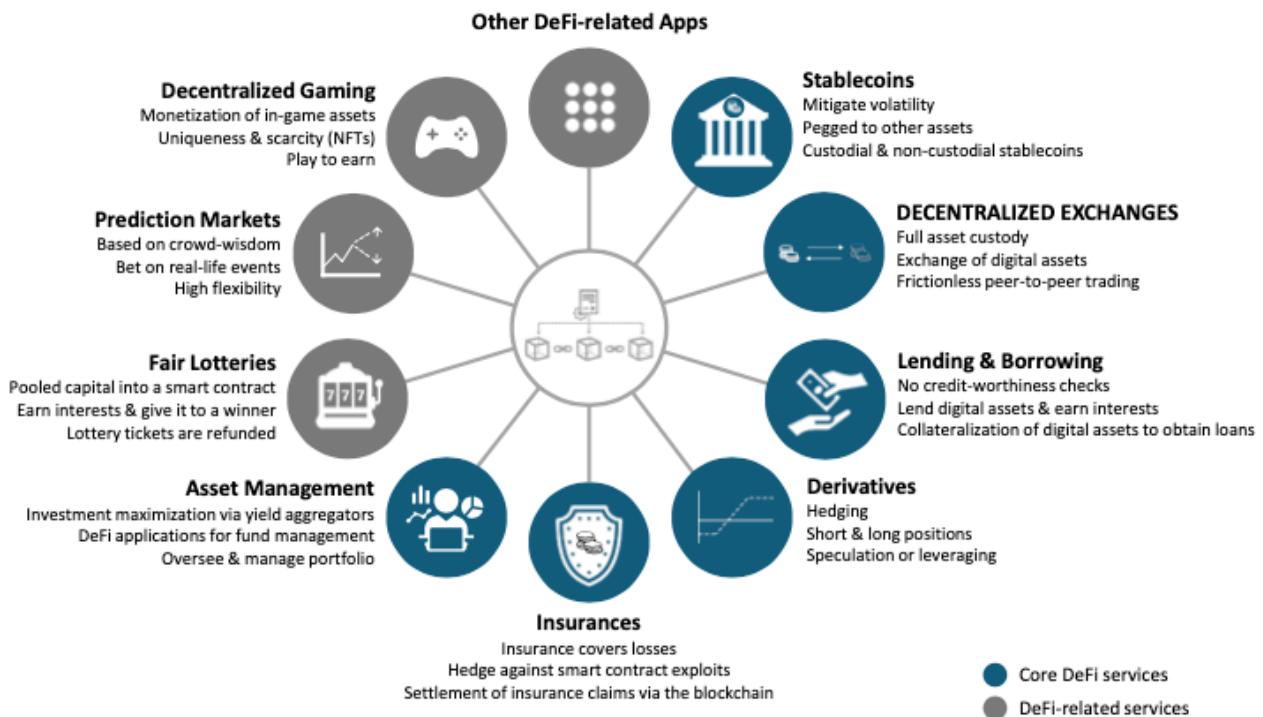


Figure 2: The DeFi-Stack on Ethereum (based on Schär (2021)).

1. **Settlement Layer:** representado por el blockchain que maneja operaciones de contabilidad, mantiene el acceso a los fondos y ejecuta las transacciones. Aquí es donde se aplican modelos de consenso para asegurar la integridad y seguridad de una manera descentralizada
2. **Asset Layer:** se pueden crear y tranzar diferentes activos digitales encima del Blockchain. Estos activos digitales tienen la función de ser unidades de valor y medios de intercambio. Adicional a ello, DeFi permite que los usuarios se conviertan en proveedores de servicios dado que usuario puede proveer activos a una aplicación DeFi para su funcionamiento.
3. **Protocol Layer:** aquí se incluyen contratos inteligentes que proveen una solución estandarizada o servicio. Por ejemplo, el de proveer un servicio completo de intercambio de criptomonedas. Los protocolos tienen una aplicación variada y determinística basada en código. Algunos ejemplos son: casa de cambio, derivados y préstamos.
4. **Application Layer:** Los contratos inteligentes o protocolos son usados como backend para una solución web con una interfaz amigable. Estas aplicaciones permiten hacer los servicios de DeFi más accesible a una audiencia mayor.
5. **Aggregation Layer:** agrega servicios que pueden ser técnicamente independientes uno de otro. También son aplicaciones descentralizadas que proveen más utilidad si combinan varias aplicaciones. Por ejemplo el de combinar varios servicios de seguro en una sola plataforma.

Aplicaciones y Casos de Uso



Stablecoins

Muchos tokens están sujetos a la especulación, manipulación de mercado. Otros poseen baja liquidez. Ello contribuye a que muchos tokens padecan de severas fluctuaciones en el precio, lo cual obstaculiza usar el token como medio de intercambio. Un medio de intercambio requiere cierta estabilidad en el precio que involucra la gestión de la oferta y demanda de un activo, normalmente llevado a cabo por un banco.

Para solucionar este problema, las monedas estables han surgido con el propósito de proveer un intercambio estable vinculadas a un valor en específico (e.g. como el dolar americano). Su principal propuesta es minimizar la volatilidad para su uso en servicios y productos financieros. Su existencia, incrementará la adopción de aplicaciones de DeFi.

Casas de cambio decentralizadas

Una casa de cambios tradicional se basa en libros de órdenes. Esta entidad centralizada se encarga de almacenar órdenes de compra y venta para luego juntar una orden de compra y otra de orden de venta que tienen la misma cantidad y precio.

Los DEXes más populares operan bajo algoritmos de AMM. Existe un pool de liquidez que contiene los dos pares de tokens a tranzar. Este pool actúa como la parte que hace el intercambio de tokens entre dueños. El AMM determina el precio a pagar por el intercambio basado en la cantidad de tokens a cambiar y el ratio de activos que existen en el pool de liquidez.

Cada vez que se hace un intercambio de tokens en el pool de liquidez, implica que un token incremente en cantidad y otro disminuya, lo cual cambia el ratio de activos dentro del pool de liquidez.

La liquidez de cada pool es creado por los proveedores de liquidez, quienes se encargan de proveer dos activos al pool usando al el ratio de tokens actual.

Si se da la posibilidad de que existan discrepancias en los precios de los tokens en diferentes DEXes, dicha brecha es eliminada a través del arbitraje que se encarga de mantener los precios en equilibrio.

Préstamos

Las personas pueden depositar sus cripto activos en contratos inteligentes y ganar intereses variables por sus depósitos. A su vez, otras personas tomarán estos activos como préstamos que incluyen intereses por pagar. Todos los activos a ser solicitados como préstamos, incluyen un colateral que asegure la liquidez del primero. Este colateral existe dado que no hay una evaluación crediticia inicial.

Aunque la sobre colateralización socava la idea de financiación, en realidad, estos protocolos son usados para proveer una utilidad financiera adicional. Se puede utilizar estos tipos de préstamos para hacer short sells y leveraged longs.

En la actualidad, Aave es la más grande aplicación de préstamos que tiene fondos bloqueados que ascienden a los cinco billones de dólares (2022).

La tasa de interés se calcula de manera diferente. En Aave, está en función de la tasa de utilización. Es decir, depende de la cantidad de préstamos totales divididos por los depósitos totales. La función de interés puede ser linear, no linear. Es esta tasa de interés la principal motivación para que las personas presten sus activos digitales al protocolo

Seguros

Los contratos inteligentes pueden estar sujetos a hackeos y errores de código, incluso con una alta probabilidad. El seguro puede ayudar a mitigar ese riesgo.

Los contratos inteligentes tienen el código publicado y cualquier persona puede obtenerlo para encontrar fallas y estrategias explotarlo. Entonces, los usuarios pueden comprar seguros para prevenir las pérdidas millonarias.

[Arbol](#) usa oráculos para obtener información relevante del clima para asegurar a los agricultores ante la eventual pérdida de sus cosechas si ciertos parámetros son cumplidos.

Otro ejemplo es [Etheris](#) que automáticamente compensa a sus clientes cuando hay vuelos demorados.

Implementar un seguro que sea púramente basado en contratos inteligentes es muy complicado. Por ejemplo, ante un eventual hackeo, es muy difícil comprobar con solo código. Por lo general, se necesita un criterio externo que valide que dicho hackeo ha ocurrido.

Uniswap

Uniswap es una casa de cambios descentralizada en la cual las personas pueden intercambiar Ether y otros tokens que fueron creados en la blockchain de Ethereum.

Provee dos servicios en particular. El primero es que usuarios pueden proveer liquidez que permite crear pozos de pares de tokens para que otras personas realicen los intercambios. El segundo uso permite que los usuarios usen los pozos de tokens para poder intercambiar dos tokens diferentes.

El incentivo para proveer de liquidez es que los inversionistas reciben una comisión de 0.3% por cada transacción de intercambio que se realiza en Uniswap.

Publicando tokens

Se publicarán dos tokens en la red Goerli para poder crear un pool de liquidez usándolo como un par en UNISWAP.

El token A y token B tienen el siguiente contenido:

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";

contract TokenA is ERC20, AccessControl {
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");

    constructor() ERC20("TokenA", "TKNA") {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(MINTER_ROLE, msg.sender);
    }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }
}

contract TokenB is ERC20, AccessControl {
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");

    constructor() ERC20("TokenB", "TKNB") {
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(MINTER_ROLE, msg.sender);
    }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }
}
```

El objetivo es crear un pool que posea estos dos tokens para que otras personas puedan cambiar tokens A por tokens B y viceversa.

En UNISWAP, una manera de crear dicho pool es ejecutando el método de añadir liquidez. Al hacerlo, automáticamente, se crea un pool (si es que este pool no existe) y además se hace el depósito de tokens en las cantidades que representarán el ratio entre tokens.

Añadiendo liquidez

Los usuarios pueden convertirse en proveedores de fondos para crear un pool de un par de tokens de modo que otras personas puedan realizar intercambios entre tokens de un mismo pool. Las personas que proveen liquidez reciben como beneficio las comisiones de intercambio de tokens.

Para añadir liquidez de manera programática, vamos a interactuar con el contrato de Uniswap llamado `UniswapV2Router02`. El address de este contrato es `0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D` en Ethereum. Existe un método en particular que permite proveer de liquidez llamado `addLiquidity` y tiene la siguiente forma:

```
function addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB, uint liquidity);
```

En la siguiente tabla, se definen todos los argumentos de este método:

| Name | Type | Description |
|---------------------------|---------|--|
| tokenA | address | A pool token. |
| tokenB | address | A pool token. |
| amountADesired | uint | The amount of tokenA to add as liquidity if the B/A price is <= amountBDesired/amountADesired (A depreciates). |
| amountBDesired | uint | The amount of tokenB to add as liquidity if the A/B price is <= amountADesired/amountBDesired (B depreciates). |
| amountAMin | uint | Bounds the extent to which the B/A price can go up before the transaction reverts. Must be <= amountADesired. |
| amountBMin | uint | Bounds the extent to which the A/B price can go up before the transaction reverts. Must be <= amountBDesired. |
| to | address | Recipient of the liquidity tokens. |
| deadline | uint | Unix timestamp after which the transaction will revert. |
| Valores de retorno | | |
| amountA | uint | The amount of tokenA sent to the pool. |
| amountB | uint | The amount of tokenB sent to the pool. |
| liquidity | uint | The amount of liquidity tokens minted. |

En este método, `tokenA` y `tokenB` representan las addresses de los tokens a depositar. `amountADesired` y `amountBDesired` son los dos montos que se planean enviar al pool de liquidez. `amountAMin` y `amountBMin`, son las dos mínimas cantidades que tienen que ir en el pool, o de otra manera, la función fallará.

La razón por la cual se tienen monto deseado (`amountDesired`) y monto mínimo (`amountMin`) es para ayudar a mantener el pool con el mismo ratio de tokens en todo momento. Por ejemplo, se podría tener que el token A se está cambiando por el token B a un ratio de 1 a 1000. Es decir, 1 token A representa 1000 token B. Al añadir más liquidez, se debe guardar la misma proporción. Si solo se tiene para depositar 0.5 del token A y 2000 del token B, entonces, los montos mínimos a depositar de cada token sería 0.5 del token A y 1,000 del token B. Si se pasan como montos mínimos 0.5 del token A y 1,500 del token B, la función va a fallar.

`to` es el address que recibirá los tokens de liquidity pool.

`deadline` especifica el timestamp después del cual la transacción fallará.

Los valores de retorno son tres: `amountA` y `amountB` indican las cantidades de tokens que finalmente se depositaron en el pool de liquidez. `liquidity` indica la cantidad de tokens de liquidez recibes como resultado de haber creado el pool de liquidez. Estos tokens son enviados a `to`.

Consideraciones adicionales:

- El address llamante (`msg.sender`) debe dar allowance al router de Uniswap al menos las cantidades de `amountADesired` y `amountBDesired`.
- Es recomendable añadir los tokens en el ratio ideal
- Si el pool del par de tokens no existe, uno se crea automáticamente. Adicional a ello, la cantidad exacta de `amountADesired` y `amountBDesired` de tokens son añadidos a dicho pool.

Interactuando con el Router de UNISWAP

En UNISWAP, existe un contrato que se encarga de crear pool y añadir liquidez llamado Router. Usaremos ese contrato y definiremos una interface para poder llamarlo.

```
// Ethereum (mainnet y testnet)
// address: 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
interface IUniswapV2Router02 {
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
}
```

Adicionalmente, para poder obtener el contrato de pool donde se guardarán los tokens A y B, usaremos el contrato Factory de UNISWAP llamado `IUniswapV2Factory` que tiene el address de `0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f` en Ethereum. El método que nos permite saber el contrato de pool de liquidez se llama `getPair`.

```
// Ethereum (mainnet y testnet)
// address: 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f
interface IUniswapV2Factory {
    function getPair(
        address tokenA,
        address tokenB
    ) external view returns (address pair);
}
```

El contrato para añadir liquidez luce de la siguiente manera:

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

interface IUniswapV2Router02 {
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
}

interface IUniswapV2Factory {
    function getPair(
        address tokenA,
        address tokenB
    ) external view returns (address pair);
}

// Address Token A: 0x52A525D4c44b0E0491c14CA7FF5A45a3884c15B3
// Address Token B: 0x89EC644A1224eC1595952D6f0b90c041A46a0765
contract LiquidityPool {
    // Router Goerli
    address routerAddress = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;
    IUniswapV2Router02 router = IUniswapV2Router02(routerAddress);
```

```

address factoryAddress = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;
IUniswapV2Factory factory = IUniswapV2Factory(factoryAddress);

IERC20 tokenA = IERC20(0x52A525D4c44b0E0491c14CA7FF5A45a3884c15B3);
IERC20 tokenB = IERC20(0x89EC644A1224eC1595952D6f0b90c041A46a0765);

event LiquidityAdded(uint amountA, uint amountB, uint liquidity);

function addLiquidity(
    address _tokenA,
    address _tokenB,
    uint _amountADesired,
    uint _amountBDesired,
    uint _amountAMin,
    uint _amountBMin,
    address _to,
    uint _deadline
) external returns (uint amountA, uint amountB, uint liquidity) {
    // Approve the router to spend the token
    tokenA.approve(routerAddress, _amountADesired);
    tokenB.approve(routerAddress, _amountBDesired);

    // Add liquidity
    (amountA, amountB, liquidity) = router.addLiquidity(
        _tokenA,
        _tokenB,
        _amountADesired,
        _amountBDesired,
        _amountAMin,
        _amountBMin,
        _to,
        _deadline
    );

    emit LiquidityAdded(amountA, amountB, liquidity);
}

function getPair(
    address _tokenA,
    address _tokenB
) external view returns (address pair) {
    pair = factory.getPair(_tokenA, _tokenB);
}
}

```

Luego de la publicación de este contrato en la red Goerli, obtenemos la siguiente address:

0x73E9D688842E6AbFaCe854fE7Fd880BE82ED6670.

El script de crear el pool y añadir liquidez, sería el siguiente:

```

async function addLiquidity() {
  var [owner] = await hre.ethers.getSigners();

  var tokenAAdd = "0x52A525D4c44b0E0491c14CA7FF5A45a3884c15B3";
  var TokenA = await hre.ethers.getContractFactory("TokenA");
  var tokenA = TokenA.attach(tokenAAdd);

  var tokenBAdd = "0x89EC644A1224eC1595952D6f0b90c041A46a0765";
  var TokenB = await hre.ethers.getContractFactory("TokenB");
  var tokenB = TokenB.attach(tokenBAdd);

  var liquidityPoolAdd = "0x73E9D688842E6AbFaCe854fE7Fd880BE82ED6670";
  var LiquidityPool = await hre.ethers.getContractFactory("LiquidityPool");
  var liquidityPool = LiquidityPool.attach(liquidityPoolAdd);

  // Depositar tokens en el contrato que creará el pool de liquidez
  var tx = await tokenA.mint(liquidityPoolAdd, pEth("1000"));
  await tx.wait();
  var tx = await tokenB.mint(liquidityPoolAdd, pEth("1000"));
  await tx.wait();

  // Definir un ratio
  // 10 token A = 25 token B
  // El pool se creará con los montos definidos en _amountAMin y _amountBMin

  // Añadir liquidez
  var _tokenA = tokenAAdd;
  var _tokenB = tokenBAdd;
  var _amountADesired = pEth("1000");
  var _amountBDesired = pEth("2500");
  var _amountAMin = pEth("1000");
  var _amountBMin = pEth("2500");
  var _to = owner.address;
  var _deadline = new Date().getTime();
  var tx = await liquidityPool.addLiquidity(
    _tokenA,
    _tokenB,
    _amountADesired,
    _amountBDesired,
    _amountAMin,
    _amountBMin,
    _to,
    _deadline
  );
  var res = await tx.wait();
  console.log("Transaction Hash", res.transactionHash);
}

```

La transaction hash es la siguiente

[0xd883a4b3eb7778bcd7e6b098a33470907b02a617ef50908fb9b31190b70ff76.](#)

Analicemos esta transacción:

| ERC-20 Tokens Transferred: | From | To | For | Token |
|----------------------------|-----------------------|-----------------------|--------------------------|---------------------|
| 1 | 0x73e9d688842e6... | 0xf2f57cd6ef4078... | 1,000 | TokenA (TKNA) |
| | 0x73e9d688842e6... | 0xf2f57cd6ef4078... | 2,500 | TokenB (TKNB) |
| | 0x0000000000000000... | 0x0000000000000000... | 0.0000000000000001 | Uniswap V2 (UNI-V2) |
| | 0x0000000000000000... | 0xca420cc41ccf54... | 1,581.138830084189664999 | Uniswap V2 (UNI-V2) |

Observamos que del contrato han salido 1000 tokens A y 2500 tokens B. Así también, se ha depositado en el address del `owner` la cantidad de 1581 tokens que representan los tokens de liquidez. En el futuro, usando estos tokens de liquidez se puede retirar los tokens A y B que fueron depositados en el pool.

Cada vez que se crea un par en UNISWAP, se crea también un contrato del pool de liquidez donde podemos recabar mayor información.

Para encontrar el token de liquidez que representa al par, primero lo buscamos en el contrato `IUniswapV2Factory` de Uniswap. Lo encontramos en la siguiente address:

[0x5C69bFe701ef814a2B6a3EDD4B1652CB9cc5aA6f.](#)

Vamos a consultar el método `getPair` y pondremos los dos tokens (A y B) usados para crear el pair de tokens en el pool de liquidez.

5. `getPair`

<input> (address)
0x52A525D4c44b0E0491c14CA7Ff5A45a3884c15B3

<input> (address)
0x89EC644A1224eC1595952D6f0b90c041A46a0765

Query

└ address

[`getPair(address,address)` method Response]
» address : [0xf2F57cd6Ef40789f36A9B418D8b6C79377e4F441](#)

Al hacerlo, obtendremos el address del contrato del pool de liquidez recientemente creado:

[0xf2F57cd6Ef40789f36A9B418D8b6C79377e4F441](#). En este contrato, vamos a revisar el método llamado `getReserves` que nos permitirá saber la cantidad de tokens en reserva.

8. getReserves

Query

```
└ _reserve0 uint112, _reserve1 uint112, _blockTimestampLast uint32

[ getReserves method Response ]
» _reserve0 uint112: 10000000000000000000000000000000
» _reserve1 uint112: 25000000000000000000000000000000
» _blockTimestampLast uint32: 1671732048
```

Se usarán estos montos para poder realizar los swaps.

Remover liquidez del pool

El método de remover liquidez, es el siguiente:

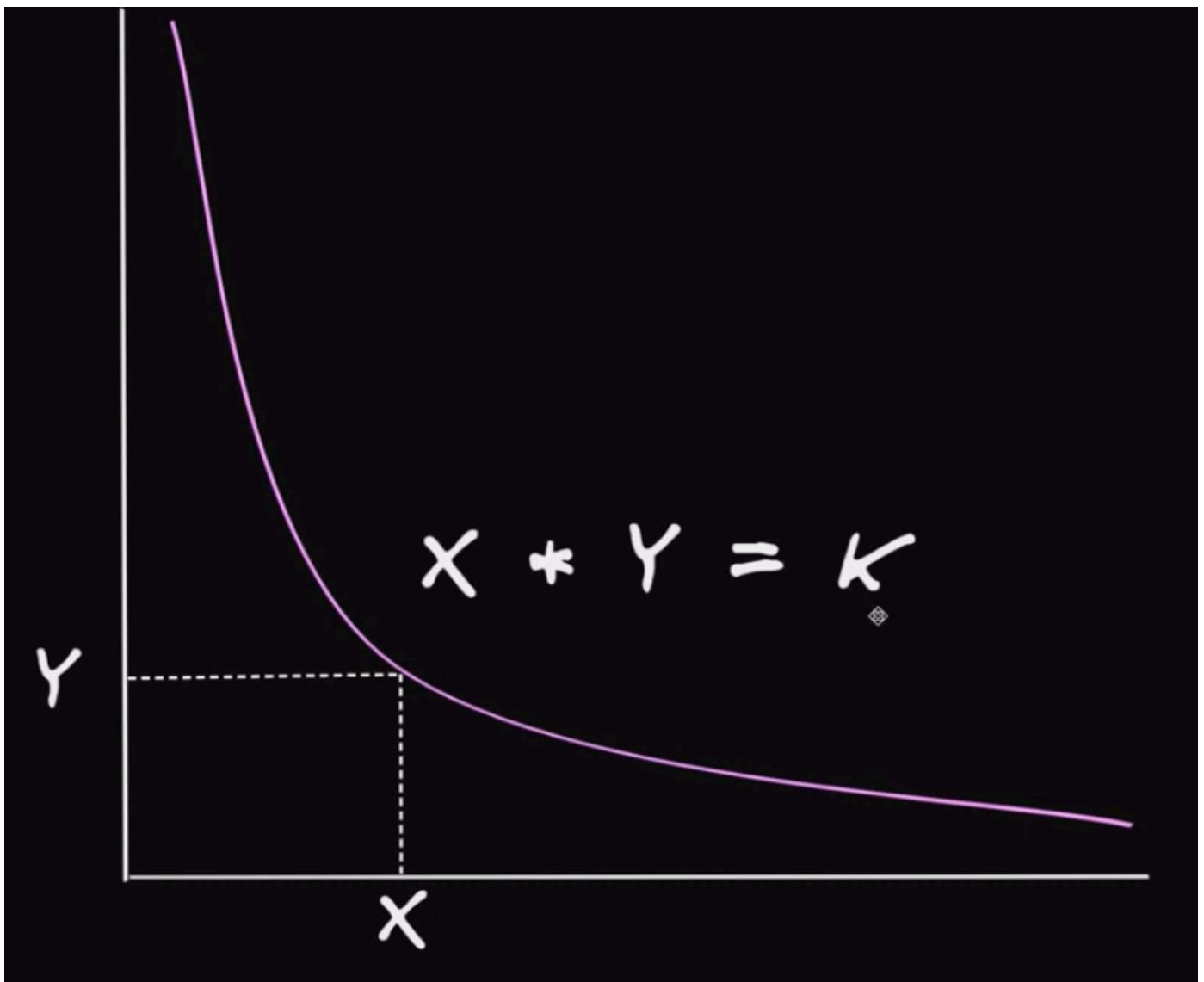
```
function removeLiquidity(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB);
```

Este método ejecuta el proceso inverso de cuando se crea el pool de liquidez.

Precio en Uniswap

Uniswap no funciona mediante un libro de órdenes de venta. En cambio, Uniswap utiliza una fórmula llamada Automated Money Making (AMM)

Empecemos definiendo la fórmula de constant product market: $X * Y = K$

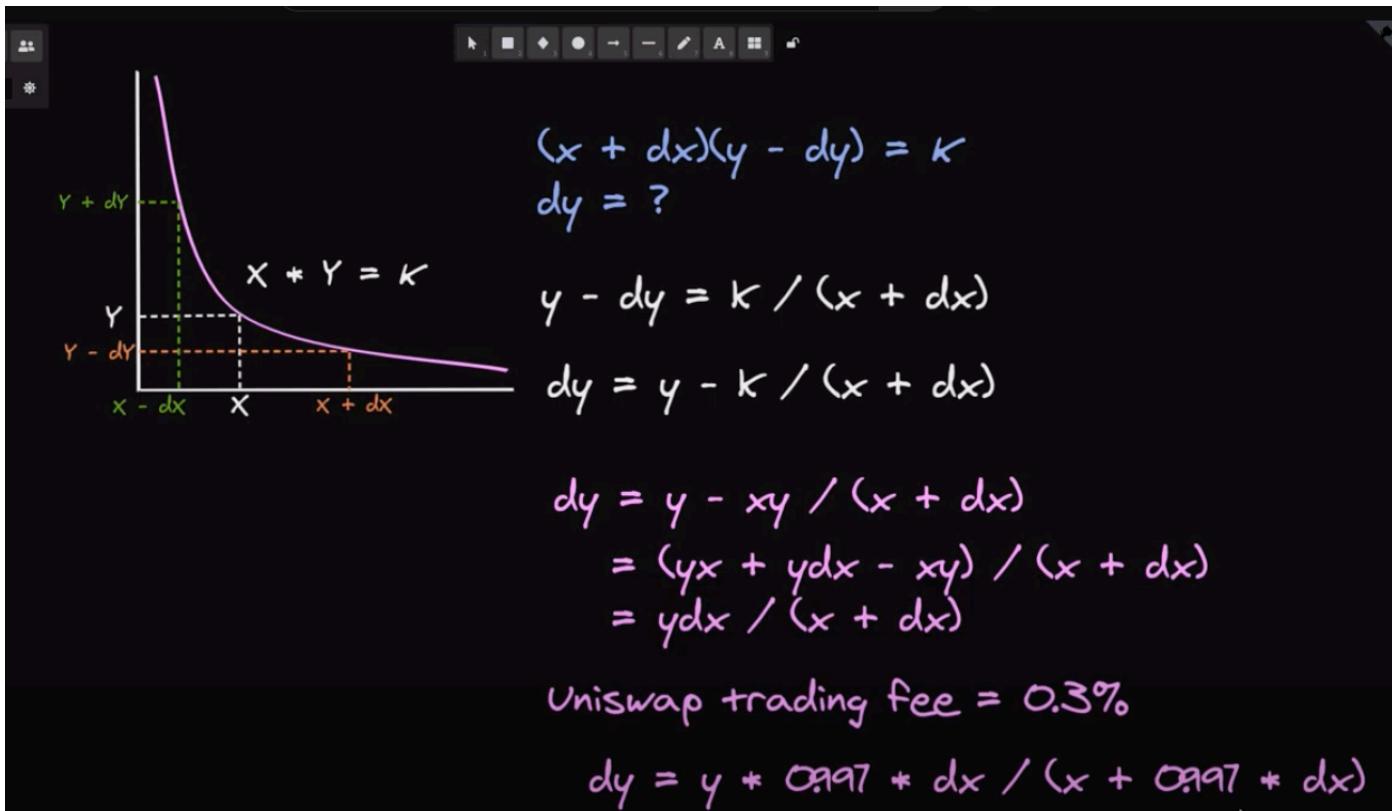


En esta fórmula, podemos definir X como la cantidad de tokens A y Y como la cantidad de tokens B en un liquidity pool. Su multiplicación crea una constante llamada K . El producto de X y Y no cambia a través de los sucesivos intercambios que se hacen en el tiempo. Es decir, K no cambia.

En la siguiente hoja de cálculo se muestra un ejemplo en concreto ([ver aquí](#)).

Calculando la cantidad de tokens a recibir

En la siguiente fórmula, vamos a encontrar la cantidad exacta de token B (dy) a recibir en caso queramos entregar una cantidad dx de tokens B.



1. X representa la cantidad de tokens A y Y representa la cantidad de tokens B.
2. Sabemos que $X * Y = K$ debe permanecer constante. Entonces, si X aumenta, Y debe disminuir.
3. Incluyamos el aumento y disminución en X y Y: $(X + dx)(Y - dy) = K$
4. Despejando la cantidad de Y a recibir tenemos que: $dy = Y * dx / (x + dx)$.
5. Si incluimos la comisión del 3% cobrada por Uniswap al realizar un intercambio, obtenemos que: $dy = Y * (1 - 0.03) * dx / (x + (1 - 0.03) * dx)$

Supongamos que queremos intercambiar 10 tokens A. Veamos cuantos tokens B puedo recibir según la fórmula:

$$X (\text{Q de A}) = 1000$$

$$Y (\text{Q de B}) = 2500$$

$$dx = 10 \text{ token A}$$

$$dy = ?$$

$$dy = 2500 * 0.997 * 10 / (1000 + 0.997 * 10) = 24.6789\dots$$

Swap Tokens

Para poder intercambiar tokens de manera programática, debemos interactuar con el contrato `UniswapV2Router02` de Uniswap. En este contrato inteligente, encontramos los siguientes métodos: `swapTokensForExactTokens` y `swapExactTokensForTokens`. Cada uno de estos métodos nos permite realizar intercambios entre tokens A y B.

Cada uno de estos tokens realiza el intercambio de tokens pero de diferente manera. Veamos:

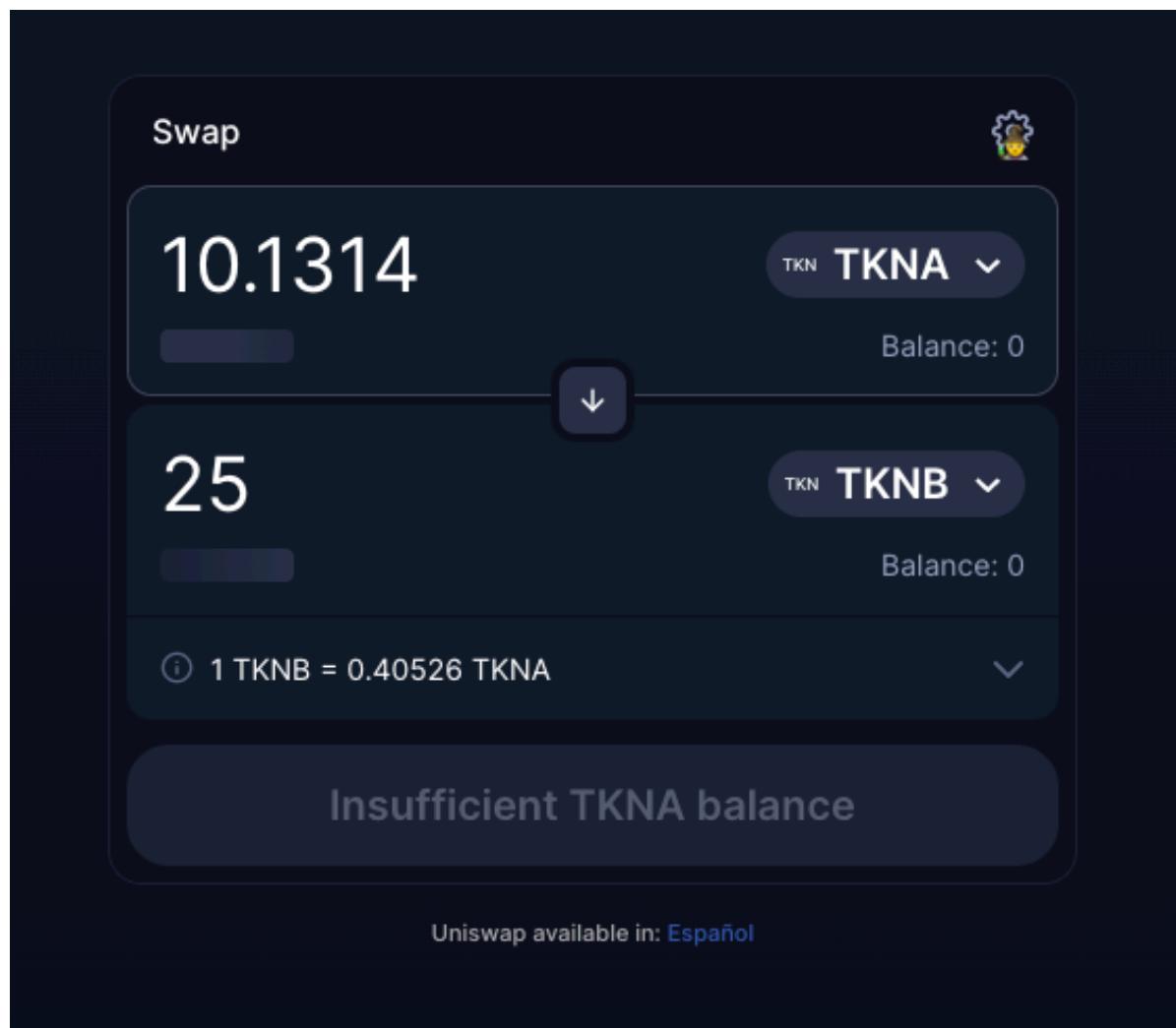
`swapTokensForExactTokens`

Cuando usas este método, defines una cantidad exacta de tokens a recibir. Supongamos que se desea entregar tokens A por tokens B. Es decir, sin importar cuantos tokens A necesites entregar, estás pidiendo un monto fijo de tokens B a recibir.

```
function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
```

En dicho caso, `amountOut` define la cantidad exacta de tokens B que deseas recibir. `amountInMax` define la cantidad máxima de tokens A que estarías dispuesto a renunciar para recibir la cantidad exacta de tokens B.

La siguiente gráfica explica este caso:



En este caso, le estoy pidiendo a UNISWAP que me entregue exactamente 25 tokens B. Uniswap se encargará de decirme cuántos tokens A se necesitan.

Descripción de los argumentos del método:

| Name | Type | Description |
|-------------|---|--|
| amountOut | <code>uint</code> | The amount of output tokens to receive. |
| amountInMax | <code>uint</code> | The maximum amount of input tokens that can be required before the transaction reverts. |
| path | <code>address[]</code> <code>calldata</code> | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| to | <code>address</code> | Recipient of the output tokens. |
| deadline | <code>uint</code> | Unix timestamp after which the transaction will revert. |
| amounts | <code>uint[]</code> <code>memory</code> | The input token amount and all subsequent output token amounts. |

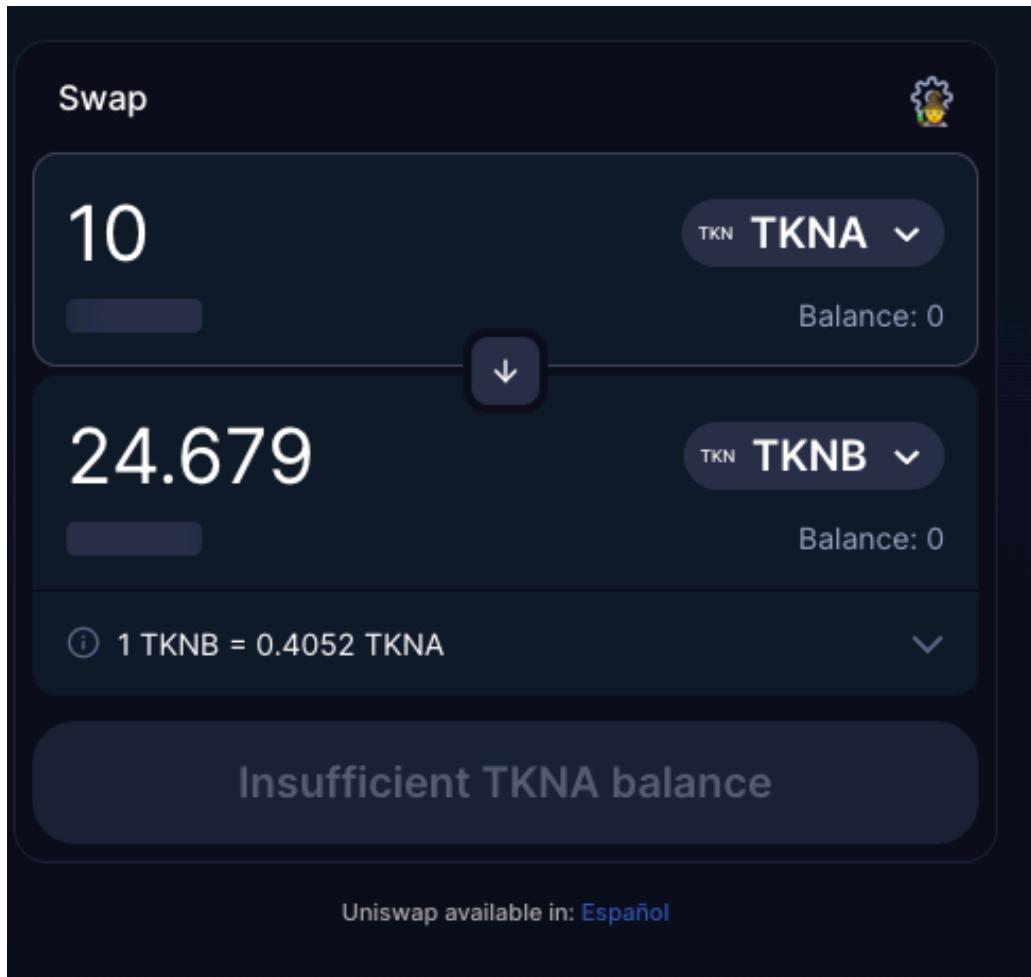
swapExactTokensForTokens

Cuando usas este método, defines una cantidad exacta de tokens a entregar. Supongamos que se desea entregar tokens A por tokens B. Es decir, sin importar cuantos tokens B necesites recibir, estás entregando una cantidad exacta de tokens A.

```
function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
```

En dicho caso, `amountIn` define la cantidad exacta de tokens A que deseas entregar. `amountOutMin` define la cantidad mínima de tokens B que estarías dispuesto a recibir como resultado de entregar una cantidad exacta de tokens A.

La siguiente gráfica explica este caso:



En este caso, le estoy pidiendo a UNISWAP que entregaré exactamente 10 tokens A. Uniswap se encargará de decirme cuántos tokens B voy a recibir como parte del intercambio.

Descripción de los argumentos del método:

| Name | Type | Description |
|--------------|---|--|
| amountIn | <code>uint</code> | The amount of input tokens to send. |
| amountOutMin | <code>uint</code> | The minimum amount of output tokens that must be received for the transaction not to revert. |
| path | <code>address[]</code> <code>calldata</code> | An array of token addresses. <code>path.length</code> must be ≥ 2 . Pools for each consecutive pair of addresses must exist and have liquidity. |
| to | <code>address</code> | Recipient of the output tokens. |
| deadline | <code>uint</code> | Unix timestamp after which the transaction will revert. |
| amounts | <code>uint[]</code> <code>memory</code> | The input token amount and all subsequent output token amounts. |

Vamos a crear un contrato Swapper que nos permitirá comunicarnos con el router `IUniswapV2Router02` de UNISWAP para poder realizar el intercambio de tokens A por tokens B. En este contrato, definiremos los dos métodos descritos arriba para realizar intercambios: `swapTokensForExactTokens` y `swapExactTokensForTokens`.

Contrato Swapper:

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

interface IUniswapV2Router02 {
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);

    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
}

// Goerli: 0xE65D464aC7D3C195e18413EbEA7f7a989449Aa83
contract Swapper {
    // Router Goerli
    address routerAddress = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;
    IUniswapV2Router02 router = IUniswapV2Router02(routerAddress);

    event SwapAmounts(uint[] amounts);

    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external {
        address tokenAAdd = path[0];
        IERC20(tokenAAdd).approve(routerAddress, amountInMax);

        uint[] memory amounts = router.swapTokensForExactTokens(
```

```

        amountOut,
        amountInMax,
        path,
        to,
        deadline
    );
}

function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external {
    address tokenAAdd = path[0];
    IERC20(tokenAAdd).approve(routerAddress, amountIn);

    uint[] memory amounts = router.swapExactTokensForTokens(
        amountIn,
        amountOutMin,
        path,
        to,
        deadline
    );

    emit SwapAmounts(amounts);
}
}

```

Del mismo modo, vamos a definir el script que nos permitirá realizar el intercambio de tokens usando el primer método `swapTokensForExactTokens`.

```

async function swapTokensForExact() {
    var tokenAAdd = "0x52A525D4c44b0E0491c14CA7FF5A45a3884c15B3";
    var TokenA = await hre.ethers.getContractFactory("TokenA");
    var tokenA = TokenA.attach(tokenAAdd);

    var tokenBAdd = "0x89EC644A1224eC1595952D6f0b90c041A46a0765";
    var TokenB = await hre.ethers.getContractFactory("TokenB");
    var tokenB = TokenB.attach(tokenBAdd);

    var swapperAdd = "0xE65D464aC7D3C195e18413EbEA7f7a989449Aa83";
    var Swapper = await hre.ethers.getContractFactory("Swapper");
    var swapper = Swapper.attach(swapperAdd);

    // Enviamos al contrato Swapper 100 tokens A

```

```

// El contrato Swapper no tiene tokens B
// El ratio de tokens A a tokens B es 10:25
// Vamos a solicitar la cantidad exacta de 100 tokens B
// No sabemos cuantos tokens A necesitamos para obtener 100 tokens B
// A través del liquidity pool, se intercambiará los tokens A por tokens B

// Enviar tokens A al contrato Swapper
var tx = await tokenA.mint(swapperAdd, pEth("100"));
await tx.wait();

var amountOut = pEth("100"); // 100 tokens B
var amountInMax = pEth("45"); // Aprox, estoy dispuesto a entregar 45 tokens A
var path = [tokenAAdd, tokenBAdd];
var to = swapperAdd;
var deadline = new Date().getTime();

var tx = await swapper.swapTokensForExactTokens(
    amountOut,
    amountInMax,
    path,
    to,
    deadline
);

var res = await tx.wait();
console.log("Transaction Hash", res.transactionHash);

console.log("Token A Bal: ", (await tokenA.balanceOf(swapperAdd)).toString());
console.log("Token B Bal: ", (await tokenB.balanceOf(swapperAdd)).toString());
}

```

Al ejecutar el script, obtenemos el siguiente resultado:

```

Transaction Hash 0xb03df3ff166cca10715b2f5d69a140e47ade6fd1c926f3b95fc9785d259f1695
Token A Bal: 58207957204948177866
Token B Bal: 10000000000000000000000000000000

```

En la [transacción](#), podemos ver que el monto de tokens B obtenidos, es exactamente 100.

| | | |
|--------------------------------|--|---------------|
| ② ERC-20 Tokens Transferred: ② | From 0xe65d464ac7d3c... To 0xf2f57cd6ef4078... For 41.792042795051822134 | TokenA (TKNA) |
| | From 0xf2f57cd6ef4078... To 0xe65d464ac7d3c... For 100 | TokenB (TKNB) |

Ahora vamos a definir el script que nos permitirá pasar una cantidad exacta de tokens A a entregar, sin saber la cantidad de tokens B a recibir. Es decir, usaremos el otro método de intercambio llamado `swapExactTokensForTokens`:

```
async function swapExactTokens() {
```

```

var tokenAAdd = "0x52A525D4c44b0E0491c14CA7FF5A45a3884c15B3";
var TokenA = await hre.ethers.getContractFactory("TokenA");
var tokenA = TokenA.attach(tokenAAdd);

var tokenBAdd = "0x89EC644A1224eC1595952D6f0b90c041A46a0765";
var TokenB = await hre.ethers.getContractFactory("TokenB");
var tokenB = TokenB.attach(tokenBAdd);

var swapperAdd = "0xE65D464aC7D3C195e18413EbEA7f7a989449Aa83";
var Swapper = await hre.ethers.getContractFactory("Swapper");
var swapper = Swapper.attach(swapperAdd);

// El contrato Swapper tiene los siguientes balances de tokens
// Token A Bal: 58207957204948177866
// Token B Bal: 1000000000000000000000000
// El ratio de tokens A a tokens B es 10:25
// Vamos a enviar a cambiar la cantidad exacta de 100 tokens B
// No sabemos cuantos tokens A vamos
// A través del liquidity pool, se intercambiará los tokens B por tokens A

var amountIn = pEth("100"); // Envío exactamente 100 tokens B
var amountOutMin = pEth("35"); // Aprox, recibiré al menos 35 tokens A
var path = [tokenBAdd, tokenAAdd];
var to = swapperAdd;
var deadline = new Date().getTime();

var tx = await swapper.swapExactTokensForTokens(
    amountIn,
    amountOutMin,
    path,
    to,
    deadline
);

var res = await tx.wait();
console.log("Transaction Hash", res.transactionHash);

console.log("Token A Bal: ", (await tokenA.balanceOf(swapperAdd)).toString());
console.log("Token B Bal: ", (await tokenB.balanceOf(swapperAdd)).toString());
}

```

Al ejecutar el script, obtenemos el siguiente resultado:

```

Transaction Hash 0x78f43f0b74826902fec65ffda5072f73cf96fe42ee9627348226cbcd6530200
Token A Bal: 99759610069958645788
Token B Bal: 0

```

En la [transacción](#), podemos ver que el monto de tokens B que fueron entregados es exactamente 100.

② ERC-20 Tokens Transferred: 2

- › From [0xe65d464ac7d3c...](#) To [0xf2f57cd6ef4078...](#) For 100 ⓘ TokenB (TKNB)
 - › From [0xf2f57cd6ef4078...](#) To [0xe65d464ac7d3c...](#) For 41.551652865010467922 ⓘ TokenA (TKNA)
-