2022

Generador de Token

DOCUMENTACIÓN

Índice

Introducción

El presente programa está orientado a brindar seguridad y fiabilidad a todo tipo de documentos en los cuales se busque constatar autenticidad.

Se logra este objetivo mediante el uso de tecnologías descentralizadas, la cadena de bloques de Ethereum y contratos inteligentes (explicado en las siguientes páginas) se logra crear un token que valida y verifica que algo existe y es inalterable, esto es así debido a la naturaleza misma de las tecnologías mencionadas.

La razón principal de la existencia de este programa, se debe a la facilidad con la que se pueden modificar documentos digitales hoy en día, ya sea con herramientas especializadas o algo tan simple como Paint en el sistema operativo Windows. Es por esto que se decide crear algo que brinde seguridad, algo simple, pero a la vez no muy fácil de corromper, prácticamente imposible de hecho como verá más adelante en este documento.

Con esta pequeña introducción se espera dilucidar de manera general el propósito del programa y el porqué de su creación. Se procederá ahora a explicar las tecnologías usadas para su construcción, las bases teóricas, usabilidad y funcionalidad final de la herramienta.

Marco teórico

Cadena de bloques

Comencemos por definir qué es una cadena de bloques, para esto nos basamos en un artículo recuperado de: Andrés Sevilla Arias, 20 de octubre, 2016, Cadena de bloques (blockchain) https://economipedia.com

"*La cadena de bloques o blockchain, es una base de datos digital donde se comparten todas las transacciones realizadas sobre algo en concreto. La información se agrupa en grupos de información llamados bloques. La información nueva que entra en esta base de datos no se puede eliminar, impidiendo que se produzcan falsificaciones.*

Las operaciones se registran en los ordenadores de todos los que participan en la cadena, incluyendo datos como cantidad, fecha, operación y participantes. Una vez compartidas y registradas en línea, no se pueden borrar. Conteniendo así, un registro fiel y verificable de todas las transacciones que se han hecho en la historia, sobre esa materia, que además están registradas con claves criptográficas.

La información de una transacción se contiene dentro de un bloque, que está entrelazado con los demás bloques (por eso se llama cadena de bloques). Cada bloque tiene una codificación criptográfica y no se puede eliminar ningún tipo de información, por lo que la seguridad es total.

El blockchain permite realizar transacciones sin la intervención de terceros, de una forma completamente segura, transparente y privada, aunque las partes estén a miles de kilómetros de distancia.

Una de las ventajas es que los movimientos de la transacción son públicos por lo que ambas partes pueden consultarlos, pero los datos concretos del intercambio son privados.

Para poder falsificar una transacción, no sería suficiente cambiar uno o varios ordenadores. Al ser un registro público pueden existir millones de copias y tendrían que cambiarse los registros de todos los ordenadores que guardan una copia, cosa completamente inviable, al ser una base abierta y pública."

Transacciones dentro de la cadena de bloques

Un término que se verá repetidamente, y que es mejor saber de qué se trata, es el de transacción, tal y como se explica en el siguiente enlace https://bfa.ar:

"*Las operaciones que se realizan para agregar información a una blockchain son denominadas transacciones.*

Su contenido puede ser muy variado y generalmente depende del tipo de red que se esté operando: hay blockchains que por medio de transacciones permiten subir archivos digitales al registro. Otras, orientadas estrictamente al intercambio de criptomonedas, toman forma de operaciones de compra-venta de activos. Una transacción puede ser simplemente una línea de texto, o incluso un hash de un documento almacenado fuera de la cadena de bloques.

Cada transacción es enviada a la red a través de un nodo, y se combina con otras transacciones para conformar un bloque. Cuando ese bloque se agrega a la cadena, la transacción queda incorporada definitivamente y se considera como completada."

Hash

Por otra parte, tenemos el concepto de hash.

Un hash es el resultado de implementar una función hash, ésta es una operación criptográfica con la cual se consigue generar identificadores únicos e irrepetibles a partir de cierta información.

"*Los hashes toman relevancia en las blockchain debido a que brindar una gran seguridad sobre las transacciones realizadas, permiten que cada transacción posea un hash, mediante el cual se puede hacer un seguimiento de esa transacción.*

Para contextualizar, aplicando hashing a los bloques como tales, cuando hay un cambio en un archivo con hash, su hash también cambiará automáticamente, y cada hash posterior está vinculado al anterior, lo cual garantiza la coherencia de todos los bloques.

Gracias al hashing en blockchain, a cada bloque se le asigna un identificador original, algunos detalles incluidos son:

- *Número de versión de la blockchain.*
- *Tiempo UNIX.*
- *Punteros del hash.*
- *El nonce, que es el valor que los mineros necesitan para crear un bloque.*
- *Hash de raíz Merkle.*

Recuperado de: Bybit Learn, septiembre 10, 2021, https://learn.bybitglobal.com/."

La cadena de bloques de Ethereum

Referenciando a: José A. Romero H. el 29/08/2021, en https://cadenadebloques.io/

"*Ethereum es una gran computadora virtual soportada por una blockchain en la que se puede ejecutar programas de forma descentralizada.*

Al igual que Bitcoin, Ethereum tiene una cadena de bloques asegurada por criptografía, sin embargo, no solo guarda información de transacciones de la moneda de la blockchain, que en este caso se llama Ether (ETH), sino que también almacena dentro de los bloques transacciones asociadas a smart contracts, ya sea para crearlos o para ejecutarlos, y datos asociados a estos programas. Los smart contracts, al estar en la blockchain, están en todos los nodos de la red y por tanto pueden ejecutarse de forma descentralizada.

Nos encontramos entonces con programas que una vez puestos a funcionar no pueden ser alterados, detenidos ni censurados por nadie. El código es público, auditable por todo el mundo y puede ser utilizado por quien quiera, siempre que pueda pagar por su ejecución. Y aquí llegamos a un punto clave."

¿Qué es el gas?

Siguiendo con el autor anterior:

"*En la criptoeconomía de Ethereum, gas es el combustible necesario para correr los smart contracts. Es la unidad de medida del poder computacional requerido para ejecutar transacciones. Una transacción puede comprender muchas instrucciones de máquina. Cada*

instrucción tiene un costo denominado en gas que tiene un costo preestablecido en el Yellow Paper de Ethereum (apéndice G).

Como en cualquier commodity, el precio del gas se determina mediante la oferta y demanda del mercado. Cuando hay mucha demanda de transacciones (la red está cargada) el precio del gas sube y cuando la demanda se reduce el precio de gas baja. El precio del gas se expresa en una fracción de ETH denominada gwei, que es equivalente a un mil millonésimo de esa moneda (1gwei = 10-9 ETH).

El uso de gas también sirve para evitar que los usuarios corran programas que saturen los recursos de la red o que incurran en prácticas de programación ineficientes. Un loop erróneo, por ejemplo, sólo se ejecutará hasta que se le acabe el gas.

En sitios como ETH Gas Station y EtherScan se puede monitorear los precios promedios de mercado de gas."

Smart contracts

Ahora bien, teniendo claro los conceptos de cadena de bloques y Ethereum, y algunos detalles también importantes, procedamos ahora a definir a la herramienta principal de este proyecto: los smart contracts. Siguiendo con el artículo anterior de José A. Romero H.:

"Los códigos ejecutables o programas dentro de Ethereum se llaman smart contracts (contratos inteligentes), bajo la premisa de que un programa es similar a un contrato: una vez cumplidas ciertas condiciones se desencadenan acciones. Ethereum es una plataforma descentralizada que corre smart contracts.

Este concepto viene de principios de los 90, cuando Nick Szabo el inventor de Bit Gold, propuso el término indicando que «los smart contracts combinan protocolos con interfaces de usuario para formalizar y asegurar relaciones sobre redes de computadoras. Este esquema elimina la necesidad de pagar a intermediarios como auditores, contadores, abogados y notarios públicos, al ejecutarse los acuerdos a través de un programa de computadora».

Solidity

Solidity es un lenguaje Turing-complete, es decir que se puede programar lo que uno quiera con él, lo que convierte a Ethereum en una blockchain programable de propósito general.

Sin embargo, la EVM no entiende Solidity sino Bytecode. Por ello, antes de enviar un smart contract o programa a la EVM el código creado en Solidity debe pasar por un compilador que lo convierte en Bytecode. El código en Bytecode se carga en una transacción de creación de contrato para integrarse a la EVM.

Todo lo que se programa en Ethereum es de código abierto. Esto quiere decir que cualquiera puede descargar ese código para luego modificarlo y crear una nueva aplicación.

dApps

Las dApps (decentralized Applications) son aplicaciones descentralizadas y son la forma en que normalmente interactúan los usuarios con los smart contracts de una blockchain como Ethereum. Las dApps proveen una interfaz amigable, detrás de la cual orquestan la ejecución de los diferentes smart contracts que requieren para proveer su servicio."

Token ERC721

En una vista general, un token es un código que representa un activo o valor financiero. Normalmente, se construyen en la línea del blockchain y se utilizan para desarrollar aplicaciones descentralizadas.

Ahora bien, siguiendo lo que se dice en el siguiente artículo:

"*El desarrollo de los tokens ERC721 fue una propuesta de mejora presentada por Dieter Shirley a finales de 2017 y tiene como peculiaridad, permitir que los contratos inteligentes funcionen como tokens intercambiables, pero además éstos son únicos y no fungibles.*

Esta condición, permite que cada token tenga su identidad propia y características únicas, las cuales hacen posible la fijación de un precio especial para su intercambio, según sus parámetros adicionales.

Características principales de un token ERC721

La principal característica de los tokens ERC721 es que no son fungibles (NFT – Non Fungible Tokens), esto tiene como significado el hecho de que no son consumibles, vale decir que no se pueden gastar como el dinero o una mercancía y tampoco pueden ser reemplazados por otra parte igual o en la misma cantidad.

Otra característica que resulta como consecuencia de la primera, es la singularidad del token y por lo tanto su indivisibilidad, que a diferencia de los tokens ERC20 (criptomonedas) que se pueden dividir hasta en fracciones de una millonésima, los tokens ERC721 se mantienen en su estado original y se compran, venden o intercambian de esa forma. (Carlos Herrera, marzo 27, 2018, "¿Qué son los tokens ERC721?")."

Usos y aplicaciones de los tokens ERC-721

"*Por ejemplo, podemos usarlo para adjudicar de forma criptográficamente segura la propiedad y posesión de diferentes elementos en la vida real. Dichos elementos pueden ser casas, terrenos, vehículos e incluso una identidad virtual criptográficamente segura. Como ves, el token 721 haría un papel muy similar al de título o escritura, que asigna propiedad a quien lo posee.*" (Recuperado de: ¿Qué es un token ERC 721? s.f).

Billetera virtual

"*Las wallets o monederos de criptomonedas, son el puente que nos permiten administrar nuestras criptomonedas. Una pieza de software o de hardware con los que realizar las operaciones de recepción y envío a través de la red blockchain de cada criptomoneda.*" (Recuperado de: ¿Qué es una wallet o monedero de criptomonedas? s.f).

Vale la pena mencionar este tema debido a que, para crear los tokens con el programa en cuestión, será necesario contar con una billetera virtual en MetaMask, con la cual se podrá abonar el coste de generar un token.

Introducción al programa

Como se dijo en la introducción, en un principio se buscaba dar al cliente seguridad, pero, sobre todo, brindar un medio para comprobar la veracidad y autenticidad de sus documentos.

Después de mucha investigación y recabado de información, se decidió llevar a cabo un proyecto para construir una dApp, en la que se pudiera subir documentos los cuales son respaldados por un token único dentro de la red Ethereum.

Este token, como se podrá intuir, es un token ERC721 (NFT), que por sus características logra adecuarse perfectamente a las necesidades que se planteaban.

Tecnologías utilizadas

La plataforma de generación de token se construye con las siguientes tecnologías:

- Backend:
  - Solidity (smart contract).
- Frontend:
  - React JS (UI).
  - Web3 (librería que permite comunicar con el smart contract).

Indirectamente se cuenta con una base de datos para guardar de manera complementaria, datos acerca de la creación del token, dicha base de datos está construida sobre:

- MySQL.
- phpMyAdmin.

Generalidades

Backend, Smart Contract

Partiendo de la base, tenemos un smart contract desarrollado con el lenguaje Solidity.

Este contrato se programó en el IDE Remix, un entorno de desarrollo integrado que se especializa en el lenguaje Solidity, permitiendo su compilación, despliegue y verificación posterior.

El contrato primero es desplegado en una red, ya sea ésta la principal de Ethereum o una de prueba como Rinkeby.

Una vez hecho esto, se puede proceder a su verificación, la cual permite que nuestro contrato posea veracidad en la red. En caso de ser desplegado en la red principal, es obligatorio que el contrato sea verificado.

Frontend

Para el frontend se utilizó React, el cual es una biblioteca de JavaScript para construir interfaces de usuario.

Se eligió React ya que, al funcionar con JavaScript, permite la integración de una librería clave para la interacción con cualquier smart contract, esta librería es Web3.

Con las librerías necesarias se pudo hacer las instancias correctas y necesarias para comunicar el smart contract con el frontend de la aplicación, permitiendo realizar acciones como creación de un token, verificar la existencia de un token, obtener datos históricos, etc.

UI y acciones permitidas

En la interfaz gráfica nos encontramos con una vista simple, esta primera vista presenta los siguientes componentes:

- Menú de navegación.
- Logos de la empresa.
- Botón para conectar billetera virtual.
- Tabla con información del último token creado.
- Campos con datos para la generación de token.
- Botón para generar el token.
- Sección informativa acerca de tiempo de creación de un token.

De las acciones permitidas a realizar por el usuario tenemos que únicamente puede generar un token y ver el resumen de su token generado, procederemos ahora a explicar el porqué.

Cuando el usuario desee autentificar sus archivos agregando un token Ethereum, primero deberá subir sus archivos en otra vista, la cual no se aborda en esta documentación. Una vez el usuario ha subido sus archivos, éste tendrá la opción de ir al menú de generar token, el cual lo lleva a la vista que venimos explicando.

En esta vista el usuario solamente presionará el botón de “Generar”, y podrán presentarse diferentes escenarios:

1. Creación de token:
   1.1. La aplicación obtiene automáticamente el ID de su contrato, que se usará para generar el nuevo token.
   1.2. Se abrirá la extensión MetaMask (previamente conectada a la cuenta del usuario) para confirmar la transacción en cuestión (crear el token).
   1.3. Una vez el usuario confirma la transacción, aparecerá una pantalla de carga que corresponde a la generación del token.
   1.4. Transcurrido el tiempo de carga, se mostrará un detalle del token generado y un botón que dirigirá al usuario a una vista aparte en la plataforma para finalizar el proceso.
2. Error de billetera desconectada o MetaMask no instalado:
   2.1. Si la billetera está desconectada el usuario deberá presionar el botón “Conectar” en la parte superior derecha de la vista, esto le permitirá conectar su cuenta de MetaMask con la plataforma.
   2.2. Si MetaMask no se encuentra instalado, la plataforma brinda el link de descarga adecuado para que el usuario pueda hacer uso de dicha tecnología.
3. Error de Token ya creado:

3.1. Si el token que el usuario desea crear ya ha sido creado antes, no se le permitirá continuar con la operación ya que, como se dijo en el marco teórico, un token NFT es único e irrepetible.

4. Error de ID vacío:
   4.1. Aunque es una posibilidad remota, puede darse de que el ID sea inválido, es decir, no es de tipo numérico o está vacío, en este caso el token no puede ser creado.

Funcionamiento

Se explica a continuación el funcionamiento a detalle de los puntos explicados en Generalidades.

Backend

Teniendo como backend el smart contract per se, procedamos a detallar su estructura y componentes.

Se comienza por declarar el tipo de licencia que tendrá, en este caso es una licencia GPL3.0, la cual "*es una licencia de derecho de autor ampliamente usada en el mundo del software libre y código abierto,6 y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software. Su propósito es doble: declarar que el software cubierto por esta licencia es libre, y protegerlo (mediante una práctica conocida como copyleft) de intentos de apropiación que restrinjan esas libertades a nuevos usuarios cada vez que la obra es distribuida, modificada o ampliada.*" (Recuperado de https://es.wikipedia.org/wiki/GNU_General_Public_License).

Luego de especificar el tipo de licencia a usar, se procede a detallar la versión de Solidity a utilizar, y por tanto a compilar luego, así como también las librerías a importar. En este caso estos datos son:

- "pragma solidity ^0.8.9" (versión de Solidity, declarada con la palabra reservada "pragma").
- "import "@openzeppelin/contracts/token/ERC721/ERC721.sol"" (la librería que utilizamos en este proyecto es la de Openzeppelin, la cual es ampliamente usada en la comunidad y permite la creación de tokens NFT mediante su contrato ERC721).

Continuamos ahora con la declaración del contrato en donde definimos su nombre y si hereda de algún otro contrato. Para este caso:

- "contract PDFToken is ERC721" (con la palabra reservada "contract" especificamos que se va a declarar el nombre de un nuevo contrato, y con "is" aclaramos que hereda de otro contrato).

El siguiente bloque de código corresponde a la declaración de variables que serán utilizadas en la ejecución del smart contract:

- "address public ownerAddress": se declara una variable de tipo "address", un tipo de dato en Solidity, con el nombre de "ownerAddress". En esta variable se guardará la cuenta que genere un token, cada vez que uno sea generado.

- “uint256 public idToken”: se declara una variable de tipo “uint256”, un tipo de dato numérico en Solidity, con el nombre de “idToken”. En esta variable se guardará el id del cada vez que uno sea generado.
- “bytes32 public code”: se declara una variable de tipo “bytes32”, un tipo de dato hexadecimal en Solidity, con el nombre de “code”. En esta variable se guardará un código generado en base a la cuenta y el id del token, cada vez que uno sea generado.
- “uint256 public timestamp”: se declara una variable de tipo “uint256”, con el nombre de “timestamp”. En esta variable se guardará la fecha en que se genere un token, cada vez que uno sea generado, en formato del tiempo UNIX.
- “mapping(uint256 => address) private _owners”: se declara una variable de tipo mapping, un tipo de dato de Solidity, con el nombre de “_owners”. Este dato consta de dos campos internos, una clave y un valor, en este caso una clave uint256 y un valor address. En esta variable se guardará un idToken y un ownerAddress cada vez que un token sea generado, de manera que se lleva un registro de todos los tokens generados, a su vez esto permite que se pueda verificar la existencia o no de un token “x”.
- “event nftCreated(address owner, uint256 idToken, bytes32 shaCode, uint256 timestamp)”: se declara un evento con la palabra reservada “event”, con el nombre de “nftCreated”. Con este evento se registra dentro del contrato cada token generado. El evento en cuestión recibe como parámetros cuatro datos: “owner, idToken, shaCode, timestamp”.

Una vez declaradas las variables, se procede a declarar un constructor, que servirá para inicializar el contrato, de la siguiente manera:

- “constructor() ERC721("PDFTK", "PTK") {}”: primero se especifica el constructor de nuestro contrato, y luego el constructor del contrato instanciado, en este caso ERC721, el cual recibe dos parámetros, el nombre del token a crear y su abreviatura.

En los últimos bloques de código se declararán las funciones a utilizar:

- “function safeMint(address ownr, uint256 _idToken) external”: esta función de nombre “safeMint”, recibe como parámetros dos datos, uno de tipo “address” y otro de tipo “uint256”, los cuales se usarán para crear el token. El modificador de acceso “external” permite que podamos invocar esta función desde fuera del contrato, pero no internamente.
  Dentro de la función declaramos las siguientes líneas de código:
  - “_safeMint(ownr, _idToken)”: invocamos la función _safeMint, propia de ERC721, y le pasamos los parámetros que recibimos. Con esto el token se crea.
  - “code = keccak256(abi.encodePacked(ownr, _idToken))”: asignamos a nuestra variable “code” el valor obtenido de la función “keccak256”, la misma encripta los datos que se pasen por parámetro, en este caso los

parámetros que recibimos en nuestra función “safeMint”, y devuelve un código hexadecimal.
    - “ownerAddress = ownr”: asignamos a la variable “ownerAddress” el valor de “ownr” que es recibido por parámetro.
    - “timestamp = block.timestamp”: asignamos a la variable “timestamp” el valor obtenido de la instancia de la clase “block.timestamp”, la cual devuelve un valor de tiempo UNIX.
    - “idToken = _idToken”: asignamos a la variable “idToken” el valor de “_idToken” que es recibido por parámetro.
    - “emit nftCreated(ownr, idToken, code, timestamp)”: con la palabra reservada “emit”, emitimos el evento “nftCreated”, y como parámetros le pasamos las variables que asignamos anteriormente.
    - “_owners[_idToken] = ownr”: invocamos al mapping que creamos y le pasamos como clave el “_idToken” y como valor “ownr”.
- “function getInfo() view external returns (address, uint256, bytes32, uint256)”: con esta función devolvemos los datos del último token creado. “view” establece que esta función no modifica el estado del contrato, sino que solamente lee datos. “external” modificador de acceso ya visto, y “returns” declara los tipos de datos que va a devolver la función. Dentro de la función encontramos la siguiente declaración:
    - “return (ownerAddress, idToken, code, timestamp)”: con esta línea devolvemos los valores almacenados en las variables que habíamos asignado anteriormente.
- “function exists(uint256 tokenId) external view returns (bool)”: con esta función retornamos un valor booleano, el cual será verdadero si el ID pasado por parámetro ya está registrado en un token creado, de lo contrario devolverá falso.
    - “return _owners[tokenId] != address(0)”: mediante la comprobación en nuestro mapping podemos saber si el token ya existe o no.

Finalizada la explicación del backend mediante el smart contract, expliquemos ahora el frontend.

Frontend

Por la parte de frontend, se había dicho anteriormente que se utilizó React y JavaScript para su construcción.

React es una librería de JavaScript, con la cual se pueden modelar todo tipo de aplicaciones web, contiene además todo un ecosistema de módulos, herramientas y componentes para poder cubrir objetivos avanzados con relativamente poco esfuerzo.

Una característica interesante de React es que permite que las vistas se asocien con los datos, de tal manera que, si cambian los datos, también cambia la vista. Esto es algo útil en este proyecto dado que requerimos que ciertas vistas se muestren o no en determinados momentos, como, por ejemplo: cuando el usuario termina de crear un

token es necesario que se muestre una vista con el detalle del token que ha generado, para esto tenemos una variable que podemos asignar verdadero cuando el proceso de creación del token finaliza, y una vez pasa eso entonces la vista se muestra porque la variable se ha asignado con un valor verdadero.

Por otra parte, tanto la conexión con MetaMask y el smart contract, se realizó con JavaScript, la primera se logra conectar simplemente manipulando la ventana con la clase “window” de JavaScript, permitiéndonos acceder así a MetaMask con “window.ethereum”, ya que MetaMask inyecta parte de su API en “window”.

La segunda conexión, con el smart contract, se logra con la librería web3, la cual posee multitud de funciones y métodos para interactuar con un smart contract que esté desplegado en la red.

Como se dijo en la sección de UI y acciones permitidas, el usuario solamente tiene la posibilidad de conectarse a su billetera y crear un token posteriormente, por lo que la interfaz de usuario es muy intuitiva y no requiere de una gran guía de uso.

FAQ

- ¿Puedo visualizar mi token una vez creado?

¡Sí! Cuando se crea un token lo que en realidad sucede detrás es que se crea un nuevo bloque en la cadena de bloques, por lo que tomará unos momentos, pero luego podrá ver su token creado en la plataforma de Etherscan usando su hash de transacción que se le brindará una vez finalizado el proceso de crear el token.

- ¿Cuánto cuesta crear un token?

Es algo difícil de definir, ya que los precios en la red Ethereum fluctúan con mucha facilidad y todo el tiempo, aun así, podemos hacer una estimación de menos de dos dólares por transacción, inclusive a veces menos de un dólar y cincuenta centavos.

- ¿Cómo se relaciona el token con mis documentos?

Los datos generados con el token son guardados y asociados en nuestras bases de datos, por lo que se crea una relación inmediata de cada token con el usuario y los documentos que se usaron para crear el token, de esta manera luego podremos verificar qué documentos pertenecen a qué token y qué usuario.

- ¿Alguien más puede ver mis tokens?

Dentro de la plataforma de Etherscan se pueden ver todos los tokens generados con el smart contract que implementamos, sin embargo, los datos visibles no representan nada por sí solos, es recién en nuestra plataforma que cobran sentido, y, una vez en nuestra plataforma, solamente tú puedes ver tus documentos y tokens generados.

- ¿Qué pasa si no finalizo correctamente la creación de un token?

Cuando aceptas pagar por la transacción en MetaMask, el proceso de creación de token en la red Ethereum comienza, y no puede ser revertido, sin embargo, si esos datos del detalle del token generado no se guardan en nuestra plataforma, entonces no podrá realizarse la correcta relación entre el token generado y los documentos que se han cargado anteriormente. Es por esto que siempre debe tratar de finalizar correctamente el proceso de creación de un token.