



UNIVERSIDAD
NACIONAL
DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: BlockGuitars: Herramienta para las transacciones de instrumentos musicales mediante blockchain

AUTORES: Castelli Lluch, Carlos Damian (7731/4) – Tallarico, Anibal (7628/6)

DIRECTOR: Dra. Pons, Claudia

CODIRECTOR: -

ASESOR PROFESIONAL: Lic. Irazabal, Jerónimo

CARRERA: Licenciatura en Sistemas

Resumen

Debido a la escasez de software, que permita registrar la adquisición o posesión de instrumentos musicales, surge la necesidad de brindar una solución moderna, que estimule tanto a los músicos profesionales como a los recién iniciados a utilizar la herramienta y que el proceso de aprendizaje sea motivador y acorde a los tiempos de corren; y lo decidimos llamar BlockGuitars. Los problemas de seguridad involucran a todo nuestro país, y la música no escapa a esta realidad, por ende el desarrollo de este tipo de herramientas proporciona beneficios a los músico

Palabras Clave

- Criptografía
- Cifrado Simétrico
- Cifrado Asimétrico
- Firma Digital
- Blockchain
- Algoritmos de Consenso
- Ethereum
- Smart Contract
- Solidity
- Instrumentos Musicales

Trabajos Realizados

Se realizó un prototipo para realizar las operaciones alta, baja, listado y transferencia de instrumentos musicales. Para la construcción del prototipo funcional se utilizó truffle que es el framework más popular de solidity para realizar el smart contract, este se basó en el token ERC721 por su característica de ser no fungible, el framework vue.js para la parte del frontend y la librería web3.js para realizar la comunicación entre el smart contract y la parte visual.

Conclusiones

Los 'smart contracts' son una evolución del sistema legal, no una sustitución del mismo. Estos contratos inteligentes permitirán hacer negocios entre desconocidos de manera fiable y sin necesitar un intermediario de confianza. Además, el software automatizará el cumplimiento de las promesas contractuales.

Trabajos Futuros

Uno de los principales retos de esta tesis y sobre todo de los contratos inteligentes es la dificultad de unir tres mundos, el tecnológico, el musical y el legal. Los contratos inteligentes escritos por los técnicos y los contratos propiamente dichos, escritos por los especialistas en leyes. El reto no es solo trasladar todo el lenguaje legal a un mundo computacional, sino que además se tienen que dar muchos avances para lograr su validez jurídica y estandarización en la industria.

Universidad Nacional de La Plata

Facultad de Informática



BlockGuitars: Herramienta para las transacciones de instrumentos musicales mediante blockchain.

Tesina de Licenciatura en Sistemas.

Autores: Castelli Lluch, Carlos Damian y Tallarico, Anibal

Director: Dra. Pons, Claudia

Asesor Profesional: Lic. Irazábal, Jerónimo

Agradecimientos

Anibal Tallarico

Quería agradecerle a toda mi familia pero en especial a mi mamá Graciela y a mi papá Osvaldo por todo el apoyo incondicional durante todos estos años.

A Bárbara, por ser mi gran compañera y sostén desde hace cinco años.

A Damián, por todos estos años de amistad. Además, de su paciencia y su actitud positiva ante todo el proceso de la tesina.

A mis compañeros y amigos de la facultad que encontré a lo largo del camino de la carrera que elegí y sigo eligiendo.

A la Facultad de Informática por darme las herramientas y el conocimiento para mi presente y futuro. Así también, a la misma y a la UNLP por permitirme pertenecer a la Educación pública, gratuita y de calidad.

Damian Castelli Lluch

Agradezco a mi familia por el apoyo y la motivación, especialmente a mi papá, Gustavo Castelli.

A mis compañeros a lo largo de estos años, que fueron muchos y la lista sería enorme y me olvidaría seguro de alguno, uno en especial si diré, que es Nacho Althabe, quien además de introducirme en cuestiones de blockchain, fue una fuente de consulta permanente.

A Anibal por ser además de un compañero, un amigo.

A mi amor, Fatima y a mi hijo Liam por el apoyo durante la escritura de esta tesis.

A Claudia y a Jerónimo por aceptar nuestro proyecto y haber invertido su tiempo en nosotros.

Y por supuesto a la Facultad de Informática y a la UNLP por darme la posibilidad de desarrollarme profesionalmente.

Índice General

Resumen	6
----------------------	---

Capítulo 1 - Introducción

1.1 Motivación.....	7
1.2 Objetivo	8
1.3 Estructura del documento.....	8

Capítulo 2 - Conceptos generales

2.1 Seguridad de la información.....	10
2.2 Criptografía.....	10
2.3 Cifrado simétrico.....	11
2.4 Cifrado asimétrico.....	12
2.5 Algoritmos de cifrado asimétrico.....	12
2.5.1 Algoritmos de factorización.....	13
2.5.1.1 RSA.....	13
2.5.2 Algoritmos de logaritmo discreto.....	15
2.5.2.1 DSA.....	15
2.5.2.2 ECC.....	17
2.5.2.3 ECDSA.....	18
2.6 Cifrado híbrido.....	19
2.7 Hash.....	21
2.7.1 MD5.....	21
2.7.2 SHA-1.....	21
2.7.3 SHA-2.....	22
2.8 Firma digital	22
2.8.1 Certificado digital	24
2.9 Resumen del capítulo.....	24

Capítulo 3 - Conceptos generales

3.1 Blockchain	25
3.2 Blockchain - Ejemplo en la vida real.....	26
3.3 Elementos que definen una Blockchain.....	27
3.4 Bitcoin.....	35
3.5 Ethereum	36
3.5.1 API de Ethereum.....	37
3.5.2 Máquina Virtual de Ethereum.....	39
3.5.3 ¿Cómo funciona Ethereum?.....	40
3.5.4 Modelo de tokens	41
3.5.5 Estándar ERC-20 para Token.....	43
3.5.6 ERC-223.....	48
3.5.7 ERC-721.....	49
3.6 Smart Contracts	50
3.6.1 ¿Cuál es el potencial de esto?.....	51
3.6.2 Los primeros contratos inteligentes.....	52
3.6.3 Un Smart contract no es lo que se piensa.....	53
3.6.4 Los Smart Contracts mal programados.....	54
3.6.5 Smart contracts en Ethereum.....	54
3.6.6 Emisión de monedas en Ethereum.....	55
3.6.7 El GAS dentro de la plataforma.....	56
3.6.8 Usos actuales de Ethereum.....	57
3.6.9 Desplegando Smart contracts en la red de Ethereum.....	58
3.6.10 Solidity - El lenguaje de los smart contracts de Ethereum.....	59
3.7 Resumen del capítulo	59

Capítulo 4 - Instrumentos Musicales

4.1 Conceptos Generales.....	60
4.1.1 Bajo eléctrico.....	61
4.1.2 Guitarra eléctrica.....	61

4.2 Mercado de Guitarras y Bajos en Argentina.....	62
4.3 Mercado negro de Instrumentos musicales en Argentina.....	62
4.4 Seguro de instrumentos musicales.....	63
4.5 Garantía de instrumentos musicales en Argentina.....	64
4.6 Registro de instrumentos musicales.....	64
4.7 resumen del capítulo.....	65

Capítulo 5

5.1 Paso a Paso	66
5.2 Stack Tecnológico	67
5.3 Enfoque de software libre	69
5.4 Memoria	69
5.5 Escenarios	69
5.5.1 Etapa 1 - Definir el objetivo del sistema	70
5.5.2 Etapa 2 - Identificar los actores	70
5.5.3 Etapa 3 - Definir historias de usuario	70
5.5.4 Etapa 4 - Dividir el sistema en dos subsistemas	71
5.5.5 Etapa 5- Diseño del subsistema smart contract	75
5.5.6 Etapa 6 : Diseño del subsistema externo	78
5.5.7 Etapa 7 - Codificar y testear los sistemas en paralelo	83
5.5.8 Etapa 8 Integrar, testear y hacer deploy.....	89
5.6 Resumen de capítulo	96

Capítulo 6

6.1 Conclusiones	97
6.1.1 Retos y particularidades a tener en cuenta.....	97
6.2 Trabajos futuros	98
6.3 Reflexiones y conclusiones acerca de la validez jurídica de blockguitars	99
6.4 Experiencias.....	102

Referencias Bibliográficas	104
---	------------

Resumen

El robo y el hurto constituyen un problema creciente en nuestro país. Se habría hecho una práctica habitual el robo de instrumentos, ya sea en salas de ensayo, escuelas de música, estudiantes, amateurs y profesional, sin distinción. En la mayoría de los casos, los ladrones no saben a ciencia cierta que estarían robando, por ende tampoco son conscientes de la importancia sobre todo laboral que tiene para un músico y su familia quizás, y el instrumento luego terminaría siendo vendido a un tercero por un valor infinitamente menor al real. El problema se potencia al existir un comercio informal tan grande con aplicaciones como MercadoLibre, OLX, MarketPlace o los mismos grupos de Facebook de compra-venta.

Debido a la escasez de software, que permita registrar la adquisición o posesión de instrumentos musicales, surge la necesidad de brindar una solución moderna, que estimule tanto a los músicos profesionales como a los recién iniciados a utilizar la herramienta y que el proceso de aprendizaje sea motivador y acorde a los tiempos de corren; y lo decidimos llamar BlockGuitars.

Los problemas de seguridad involucran a todo nuestro país, y la música no escapa a esta realidad, por ende el desarrollo de este tipo de herramientas proporciona beneficios a los músicos.

Capítulo 1

Introducción

1.1 Motivación

Para cualquier tipo de propiedad de valor es importante mantener registros precisos que permitan a los propietarios identificarse como tal, de manera que puedan ser utilizados para proteger sus derechos, resolver disputas, asegurar que las transferencias de propiedad son realizadas y prevenir fraudes. Hasta ahora, estos sistemas han necesitado de terceras partes con las que hay que establecer confianza para el mantenimiento de estos registros, tales como fabricantes o agencias gubernamentales.

El éxito de bitcoin, y por lo tanto de blockchain, así como su puesta a prueba durante estos años (bitcoin se lanzó en 2009) en un entorno tan sensible como es el económico, donde se ha podido comprobar su seguridad y robustez, valida el concepto de registro o consenso distribuido, el cual permite registrar de forma eficaz y segura transacciones de elementos virtuales. Y este concepto, que al fin y al cabo está resolviendo un problema fundamental del entorno digital, la copia y propiedad, se puede aplicar a otros ámbitos aparte del económico o las criptomonedas. Por ejemplo el arte digital, derechos de autor, o la votación electrónica.

Blockchain se ha convertido, recientemente, en un lugar común, en una palabra de moda, en un cliché o concepto polisémico cuya ambigüedad al definirlo contrasta con los significados que se ensayan y los (innumerables) beneficios que se le atribuyen. Su impacto en el sector público es, en el mejor de los casos, incomprendido, o de plano generalmente ignorado. La complejidad técnica sesga el debate público.

El objeto de este proyecto es el de estudiar la aplicación de la tecnología blockchain para construir un sistema de registro de instrumentos musicales que ofrezca mayor seguridad y ventajas que los actuales.

1.2 Objetivos

En este contexto, uno de los objetivos de este proyecto es implementar un sistema que mantenga un registro de instrumentos musicales público y distribuido, haciendo uso de la cadena de bloques de blockchain para mantener el registro de los datos, la emisión y la transferencia de activos.

De este modo, el sistema de registro de instrumentos deberá:

- Mantener una copia propia de la cadena de bloques de blockchain, que será verificada y se utilizará para extraer los datos relativos a las propiedades.
- Detectar y procesar las transacciones para extraer la información relativa a las propiedades que representan
- Ofrecer a los propietario un sistema de acceso a la información extraída, de modo que puedan consultar y realizar operaciones sobre sus registros.
- Ofrecer a terceros la posibilidad de consultar la información extraída.
- Mantener la información actualizada, de manera que las nuevas emisiones y transferencias de propiedades que sean realizadas sean identificadas por el sistema tras la validación de las transacciones recibidas.

1.3 Estructura del documento

En el capítulo 2 se definirán conceptos generales de criptografía, algoritmos de cifrado y Hash necesarios para introducir al concepto de cadena de bloques y comprender su funcionamiento.

En el capítulo 3 se detallarán conceptos de blockchain, Ethereum, Solidity. Asimismo, se describirá cómo ha evolucionado la cadena de bloques desde que se introdujo con la llegada del Bitcoin hasta su estado actual, permitiendo el desarrollo de un conjunto de protocolos destinado a la gestión de activos.

En el capítulo 4 se definirán conceptos generales referidos a instrumentos musicales, los distintos escenarios, detallando todas las necesidades y requisitos que demanda una solución de este tipo, comparando con las soluciones actuales.

En el capítulo 5 se ofrecerá una descripción general del proyecto y se proporcionará los detalles de diseño, implementación y despliegue del sistema propuesto detallando las tecnologías y algoritmos que se utilizaran. Una vez detallada la solución, se procederá a realizar un análisis de la misma, observando las debilidades e inconvenientes del sistema, haciendo especial hincapié en la comparación con las soluciones actuales de compra-venta , también está dedicado a la descripción de las pruebas realizadas y de los resultados obtenidos en base al funcionamiento del prototipo construido en conexión con el sistema implementado.

En el capítulo 6 se presentarán las conclusiones de todo este trabajo, los condicionantes del mismo y las posibles líneas de trabajo futuro.

Capítulo 2

En este capítulo se introducirá sobre conceptos generales de Seguridad, Criptografía y algoritmos de cifrado.

Conceptos Generales

2.1 Seguridad de la información

Se puede hablar de la Seguridad de la Información como el conjunto de reglas, planes y acciones que permiten asegurar la información manteniendo las propiedades de confidencialidad, integridad y disponibilidad de la misma.

- La confidencialidad es que la información sea accesible sólo para aquéllos que están autorizados.
- La integridad es que la información sólo puede ser creada y modificada por quien esté autorizado a hacerlo.
- La disponibilidad es que la información debe ser accesible para su consulta o modificación cuando se requiera.

Para que exista seguridad ya sea de la información o informática hay que garantizar las propiedades de confidencialidad, integridad y disponibilidad. Y es aquí donde se utiliza a la criptografía. [1]

2.2 Criptografía

La criptografía se refiere a la técnica de códigos y sistemas de escritura cifrada para proteger la transmisión de información privada, de forma que para quien no posea la clave sea ilegible o prácticamente imposible de descifrar. La criptografía, además de proteger la integridad de la web, permite preservar la seguridad de los usuarios, de las comunicaciones y de las operaciones que realicen a través de internet. El objeto principal de la criptografía, pues, es garantizar la privacidad de la información que es compartida a través de la red. [2]

2.3 Cifrado simétrico

Los sistemas de cifrado simétrico son aquellos que utilizan la misma clave para cifrar y descifrar un documento. El principal problema de seguridad reside en el intercambio de claves entre el emisor y el receptor ya que ambos deben usar la misma clave. Por lo tanto se tiene que buscar también un canal de comunicación que sea seguro para el intercambio de la clave. Es importante que dicha clave sea muy difícil de adivinar ya que hoy en día los ordenadores pueden adivinar claves muy rápidamente. Debemos tener en cuenta que los algoritmos criptográficos son públicos, por lo que su fortaleza debe depender de su complejidad interna y de la longitud de la clave empleada para evitar los ataques de fuerza bruta. [3]

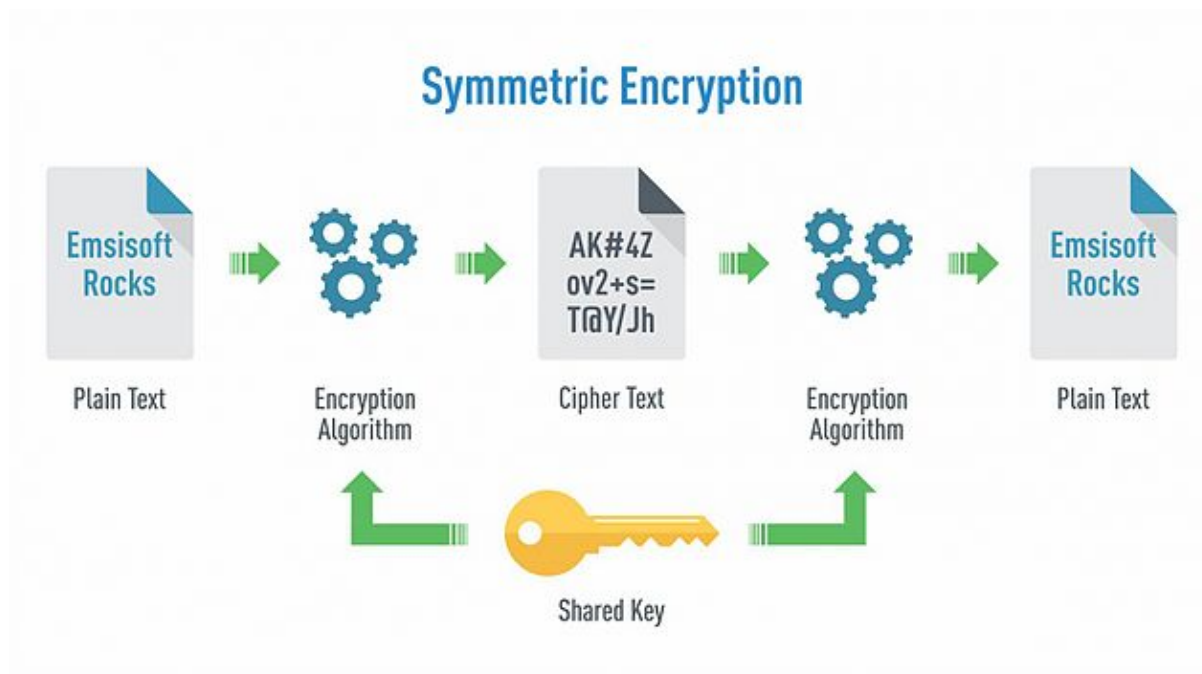


Figura 1 - Cifrado Simétrico [3]

2.4 Cifrado asimétrico

También son llamados sistemas de cifrado de clave pública. Este sistema de cifrado usa dos claves diferentes. Una es la clave pública y se puede enviar a cualquier persona y otra que se llama clave privada, que debe guardarse para que nadie tenga acceso a ella. Para enviar un mensaje, el remitente usa la clave pública del destinatario para cifrar el mensaje. Una vez que lo ha cifrado, solamente con la clave privada del destinatario se puede descifrar, ni siquiera el que ha cifrado el mensaje puede volver a descifrarlo. Por ello, se puede dar a conocer perfectamente la clave pública para que todo aquel que se quiera comunicar con el destinatario lo pueda hacer. [4]

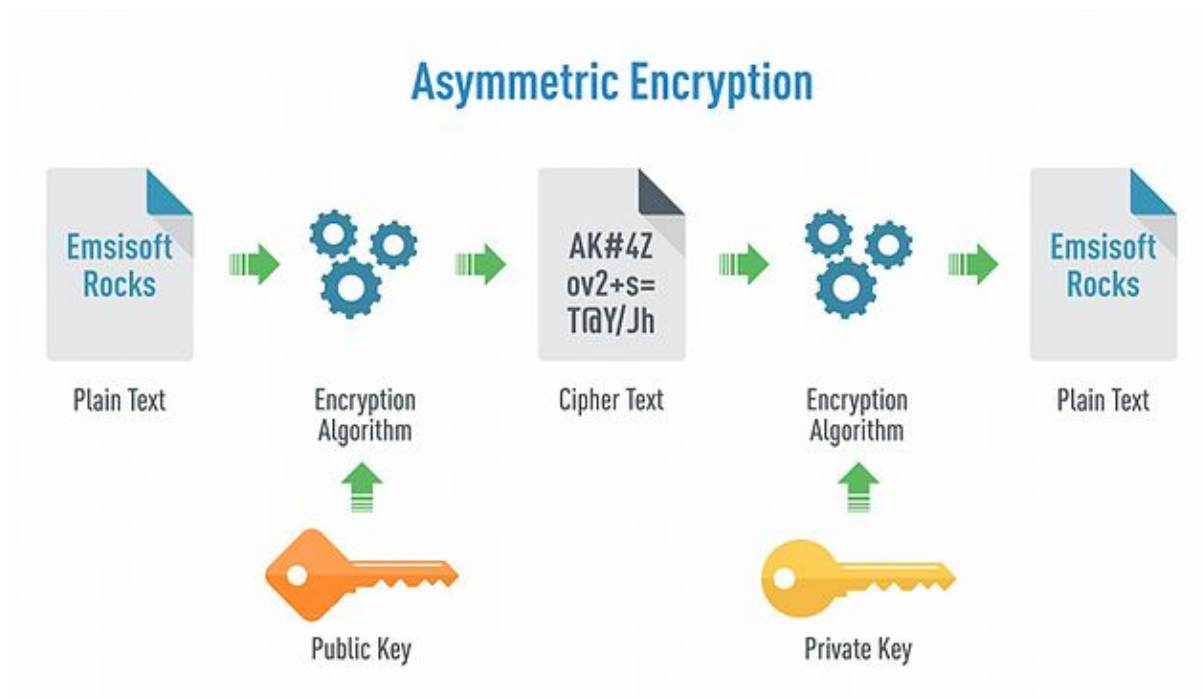


Figura 2 - Cifrado Asimétrico [3]

2.5 Algoritmos de cifrado asimétrico:

Los algoritmos asimétricos están basados en funciones matemáticas fáciles de resolver en un sentido, pero muy complicadas realizarlo en sentido inverso, a menos que se conozca la llave. Las claves públicas y privadas se generan simultáneamente y están ligadas la una a la otra. Esta relación debe ser muy compleja para que resulte muy difícil que obtengamos una a partir de la otra. [5]

Las parejas de claves tienen funciones diversas y muy importantes, entre las que destacan:

- Cifrar la información.
- Asegurar la integridad de los datos transmitidos.
- Garantizar la autenticidad del emisor.

2.5.1- Algoritmos de Factoreo

Factorear es el problema, presumiblemente difícil, en el cual se basan muchos sistemas criptográficos, incluido el RSA. Factorear un módulo RSA le permitiría a un atacante conocer la clave privada, de modo que cualquier persona que pueda factorear el módulo puede descifrar mensajes y falsificar firmas. La seguridad del sistema depende de que factorear sea difícil. Desafortunadamente, no se ha comprobado que factorear sea dificultoso y existe la posibilidad de que se descubra un método veloz de factoreo aunque los investigadores lo consideran como una posibilidad remota.

Factorear un número de muchas cifras lleva más tiempo que factorear uno sencillo. Por esta razón la medida de los módulos en el sistema RSA determina la seguridad del uso de dicho sistema: cuanto más largo sea el módulo, mayor tiempo le tomará a un atacante factorearlo y mayor será la resistencia del sistema a los atacantes.

2.5.1.1 - RSA - (Algoritmo de Factorización)

RSA es un algoritmo de cifrado asimétrico, o de clave pública, y es uno de los algoritmos más utilizados en la actualidad. De hecho, la mayor parte de los sitios hoy corren sobre SSL/TLS, y permiten la autenticación mediante cifrado asimétrico basado en RSA.

RSA sirve para cifrar y descifrar información, y por ello también provee servicios de autenticidad y de integridad, mediante lo que se conoce como Infraestructura de clave pública.

La seguridad de este algoritmo radica en el problema de la factorización de números enteros. Los mensajes enviados se representan mediante números, y el funcionamiento se basa en el producto, conocido, de dos números primos grandes elegidos al azar y mantenidos en secreto. Actualmente estos primos son del orden

de 10^{200} , y se prevé que su tamaño crezca con el aumento de la capacidad de cálculo de los ordenadores. [6]

Generando las claves:

En el criptosistema RSA son de vital importancia los números primos ya que constituyen la pieza básica en la construcción de este. Quienes deseen crear un juego de claves, pública y privada, en el criptosistema RSA primero seleccionan dos números primos p , q diferentes lo suficientemente grandes. Entonces calculan su producto $n = p \cdot q$. Después evalúan la función de Euler $\phi(n) = (p-1)(q-1)$, y seleccionan un número entero positivo e con $1 < e < \phi(n)$ tal que e sea coprimo con $\phi(n)$. Finalmente calculan el número entero d con $1 < d < \phi(n)$ tal que $d \cdot e \equiv 1 \pmod{\phi(n)}$. Tanto la verificación de que e es primo con $\phi(n)$ como la obtención de su inverso, se realizan gracias al algoritmo de Euclides extendido, algoritmo computacionalmente polinómico.

Ventajas:

Las ventajas del RSA sobre otros sistemas criptográficos con clave pública incluye el hecho de que se puede usar tanto para el encriptado como para la autenticación, y que ha estado disponible desde hace mucho tiempo y ha superado un exhaustivo escrutinio. El sistema RSA ha recibido más atención, estudio y utilización real que en ningún otro método con clave pública por lo cual el RSA posee más evidencia empírica acerca de su seguridad que otros sistemas más nuevos pero menos estudiados. En realidad, un gran número de sistemas criptográficos con clave pública que eran aparentemente seguros, con el tiempo fallaron.

Desventajas:

- La seguridad depende de la eficiencia de los ordenadores.
- Es más lento que los algoritmos de clave simétrica.
- La clave privada debe ser cifrada por algún algoritmo simétrico.

2.5.2 - Algoritmos de Logaritmo discreto

El problema del logaritmo discreto, en su fórmula más común, consiste en encontrar un exponente x en la fórmula $y = g^x \text{ mod } p$. En otras palabras, tratar de encontrar la respuesta a la siguiente pregunta: ¿A qué potencia debo elevar g para obtener y , módulo número primo p ?

Al igual que el problema del factoro, computacionalmente, el problema del logaritmo discreto se considera difícil y se lo conoce como la dirección dificultosa de una función unidireccional. Por esta razón, ha sido la base de varios sistemas criptográficos con clave pública incluidos ElGamal y DSS respectivamente. El problema del logaritmo discreto mantiene la misma relación con estos sistemas que el factoro mantiene con el RSA, la seguridad de los sistemas descansa en la suposición de que el logaritmo discreto es difícil de computar.

En los últimos años, el problema del logaritmo discreto ha recibido mucha atención, se pueden encontrar descripciones de algunos de los algoritmos más eficientes. Se espera que los mejores problemas del logaritmo discreto muestren tiempos de cómputo similares a los de los mejores algoritmos de factoro.

2.5.2.1 - DSA - (Algoritmo de Logaritmo discreto)

DSA (Algoritmo de Firma Digital), es un algoritmo de cifrado asimétrico o de clave pública. Podemos verificar la autenticidad de un mensaje, dada una clave pública y la firma del mensaje. También podemos generar pares claves pública, privada y generar firmas de datos usando la clave privada generada. [7]

Generación de claves DSA

- Elegir un número primo p de L bits, donde $512 \leq L \leq 1024$ y L es divisible por 64.
- Elegir un número primo q de 160 bits, tal que $p-1 = qz$, donde z es algún número natural.
- Elegir h , donde $1 < h < p - 1$ tal que $g = h^z \pmod{p} > 1$.
- Elegir x de forma aleatoria, donde $1 < x < q-1$.
- Calcular $y = g^x \pmod{p}$.

Los datos públicos son p , q , g e y . x es la clave privada.

Generación de una firma en DSA

Elegir un número aleatorio k , donde $1 < k < q$.

- Calcular $r = (g^k \pmod{p}) \pmod{q}$.
- Calcular $s = k^{-1}(H(m) + r*x) \pmod{q}$, donde $H(m)$ es la función hash SHA-1 aplicada al mensaje m .
- La firma es el par (r, s) .

Si r ó s es cero, se vuelve a repetir el procedimiento.

Verificación de una firma en DSA

- Calcular $w = (s)^{-1} \pmod{q}$.
- Calcular $u_1 = H(m)*w \pmod{q}$.
- Calcular $u_2 = r*w \pmod{q}$.
- Calcular $v = [g^{u_1} * y^{u_2} \pmod{p}] \pmod{q}$.
- La firma es válida si $v = r$.

Ventajas

- Es muy rápido y fácil de implementar.

Desventajas:

- Emplea una clave demasiado corta, lo cual hace que con el avance actual de los ordenadores, los ataques por la fuerza bruta se puedan llevar a cabo.
- Una desventaja del DSA es que requiere más tiempo de cómputo que el RSA.

2.5.2.2 - Criptografía de Curva Elíptica (ECC - Algoritmo de Logaritmo discreto)

Su seguridad descansa en el mismo problema que los métodos de Diffie-Hellman y DSA, pero en vez de usar números enteros como los símbolos del alfabeto del mensaje a encriptar (o firmar), usa puntos en un objeto matemático llamado Curva Elíptica. ECC puede ser usado tanto para encriptar como para firmar digitalmente. Hasta el momento, no se conoce ataque alguno cuyo tiempo de ejecución esperado sea sub exponencial para poder romper los ECC, esto hace que para obtener el mismo nivel de seguridad que brindan los otros sistemas, el espacio de claves de ECC sea mucho más pequeño, lo que lo hace una tecnología adecuada para utilizar en ambientes restringidos en recursos (memoria, costó, velocidad, ancho de banda, etc.). [8]

Aplicaciones de ECC

Una implementación de ECC es beneficiosa particularmente en aplicaciones donde el ancho de banda, capacidad de procesamiento, disponibilidad de energía o almacenamiento están restringidos. Tales aplicaciones incluyen transacciones sobre dispositivos inalámbricos, computación en dispositivos handheld como PDAs o teléfonos celulares, broadcasting y smart cards.

Ventajas ECC

La principal ventaja de la criptografía de curva elíptica es la posibilidad de crear claves más pequeñas, reduciendo así requisitos de almacenamiento y transmisión. Una clave basada en la criptografía de curva elíptica puede dar el mismo nivel de seguridad con un clave de 256 bits como un algoritmo RSA con una clave de 2048 bits.

2.5.2.3 - Elliptic Curve DSA (ECDSA)

ECDSA es el análogo sobre curvas elípticas al DSA. Esto es, en vez de trabajar sobre el subgrupo de orden q en Z_p^* , se trabaja en un grupo de curva elíptica $E(Z_p)$. El ECDSA fue estandarizado por el NIST, pero también están en este proceso los comités de estándares ANSI X9F1 e IEEE P1363. En concreto, el ECDSA es un esquema de firma con apéndice basado en ECC. Está diseñado para ser existencialmente infalsificable, aun ante la presencia de un adversario capaz de lanzar ataques de mensajes elegidos por él (es decir, un chosen message attack).
[9]

Generación de claves ECDSA

Cada entidad A hace lo siguiente:

1. Seleccionar una curva elíptica E definida sobre $Z * p$. El número de puntos en $E(Z_p)$ debería ser divisible por un número primo grande n .
2. Seleccionar un punto $P \in E(Z_p)$ de orden n .
3. Seleccionar un entero d estadísticamente único e impredecible en el intervalo $[1, n - 1]$.
4. Calcular $Q = dP$.
5. La clave pública de A es (E, P, n, Q) ; la clave privada de A es d .

Generación de una firma ECDSA

- Para firmar un mensaje m , A hace lo siguiente:
1. Seleccionar un entero k estadísticamente único e impredecible en el intervalo $[1, n - 1]$.
 2. Calcular $kP = (x_1, y_1)$ y $r = x_1 \bmod n$. (Aquí x_1 es tratado como a un número entero, por ejemplo por conversión de su representación binaria.) Si $r = 0$, entonces volver al paso 1. (Esta es una condición de seguridad: si $r = 0$, entonces la ecuación de firma $s = k^{-1}\{h(m) + dr\} \bmod n$ no involucra a la clave privada d !)
 3. Calcular $k^{-1} \bmod n$.
 4. Calcular $s = k^{-1}\{h(m) + dr\} \bmod n$, donde h es el Secure Hash Algorithm (SHA-1).
 5. Si $s = 0$, entonces volver al paso 1. (Si $s = 0$ entonces $s^{-1} \bmod n$ no existe; s^{-1} es requerido en el paso 2 de la verificación de firma.)
 6. La firma para el mensaje m es el par de enteros (r, s) .

Verificación de firma ECDSA

Para verificar la firma (r, s) de A sobre el mensaje m , B debería hacer:

1. Obtener una copia auténtica de la clave pública (E, P, n, Q) de A.
2. Verificar que r y s sean enteros en el intervalo $[1, n - 1]$. Si alguno no está en el intervalo $[1, n - 1]$ retornar "invalida" y parar.
3. Calcular $w = s^{-1} \bmod n$ y $h(m)$.
4. Calcular $u_1 = h(m)w \bmod n$ y $u_2 = rw \bmod n$.
5. Calcular $R = u_1P + u_2Q = (x_0, y_0)$ y $v = x_0 \bmod n$. Si $R = O$, retornar "invalida" y parar.
6. Aceptar la firma si y sólo si $v = r$. Si $v \neq r$ retornar "invalida".

2.6 Cifrado híbrido

Este sistema es la unión de las ventajas de los dos anteriores, debemos de partir que el problema de ambos sistemas criptográficos es que el simétrico es inseguro y el asimétrico es lento.

Es el sistema de cifrado que usa tanto los sistemas de clave simétrica como el de clave asimétrica. Funciona mediante el cifrado de clave pública para compartir una clave para el cifrado simétrico. En cada mensaje, la clave simétrica utilizada es diferente por lo que si un atacante pudiera descubrir la clave simétrica, solo le valdría para ese mensaje y no para los restantes. Tanto PGP como GPG (Ver apartado) usan sistemas de cifrado híbridos. La clave simétrica es cifrada con la clave pública, y el mensaje saliente es cifrado con la clave simétrica, todo combinado automáticamente en un sólo paquete. El destinatario usa su clave privada para descifrar la clave simétrica y acto seguido usa la clave simétrica para descifrar el mensaje.

Usa la criptografía asimétrica sólo para el inicio de la sesión, cuando hay que generar un canal seguro donde acordar la clave simétrica. La criptografía simétrica se usa durante la transmisión, usando la clave simétrica acordada antes, en el inicio de sesión. [10]

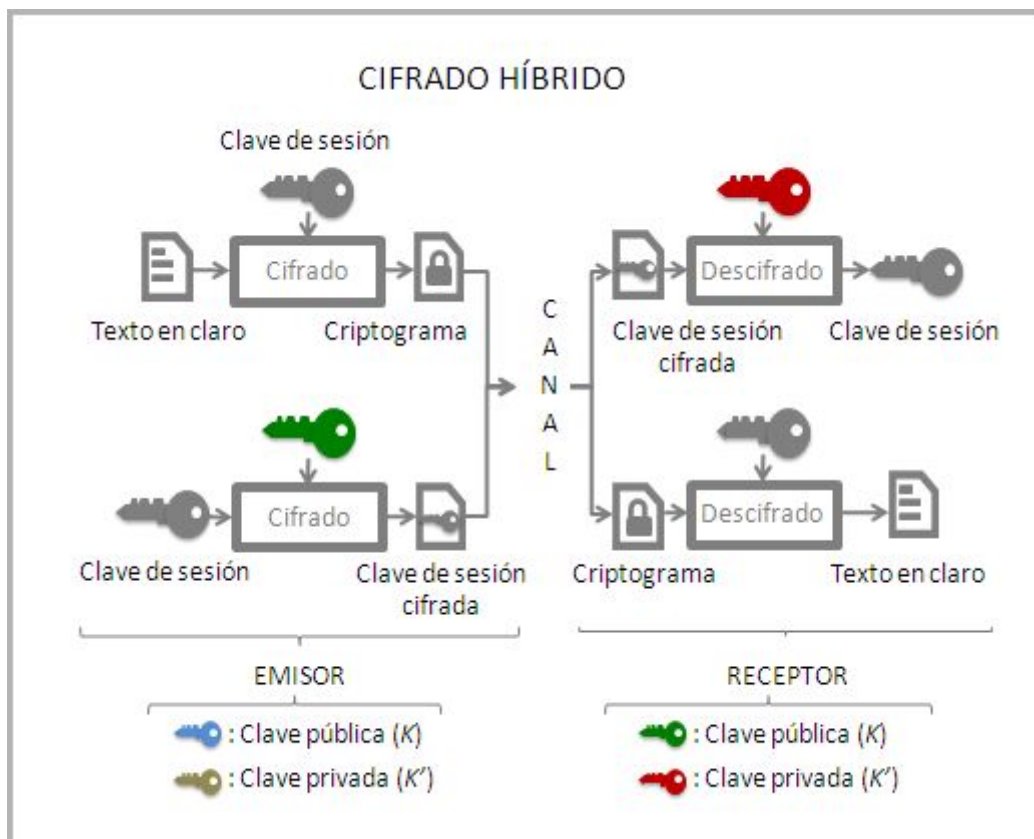


Figura 3 - Cifrado híbrido [11]

PGP es un criptosistema híbrido que combina técnicas de criptografía simétrica y criptografía asimétrica. Esta combinación permite aprovechar lo mejor de cada uno: El cifrado simétrico es más rápido que el asimétrico o de clave pública, mientras que este, a su vez, proporciona una solución al problema de la distribución de claves en forma segura y garantiza el no repudio de los datos y la no suplantación.

GPG es un software de cifrado híbrido que usa una combinación convencional de criptografía de claves simétricas para la rapidez y criptografía de claves públicas para el fácil compartimiento de claves seguras, típicamente usando recipientes de claves públicas para cifrar una clave de sesión que es usada una vez. Este modo de operación es parte del estándar OpenPGP y ha sido parte del PGP desde su primera versión.

2.7 Hash

Una función criptográfica hash- usualmente conocida como "hash"- es un algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija. Independientemente de la longitud de los datos de entrada, el valor hash de salida tendrá siempre la misma longitud.

Estas funciones no tienen el mismo propósito que la criptografía simétrica y asimétrica, tiene varios cometidos, entre ellos está asegurar que no se ha modificado un archivo en una transmisión, hacer ilegible una contraseña o firmar digitalmente un documento. [12]

2.7.1 MD5

Es una función hash de 128 bits. Como todas las funciones hash, toma unos determinados tamaños a la entrada, y salen con una longitud fija (128bits). El algoritmo MD5 no sirve para cifrar un mensaje. La información original no se puede recuperar, ya que está específicamente diseñado para que a partir de una huella hash no se pueda recuperar la información. Actualmente esta función hash no es segura utilizarla.

Este sistema de criptografía usa algoritmos que aseguran que con la respuesta (o **hash**) nunca se podrá saber cuáles han sido los datos insertados, lo que indica que es una **función unidireccional**. [13]

2.7.2 SHA-1

Es parecido a MD5, pero tiene un bloque de 160 bits en lugar de los 128 bits del MD5. La función de compresión es más compleja que la función de MD5, por tanto, SHA-1 es más lento que MD5 porque el número de pasos son de 80 (64 en MD5) y porque tiene mayor longitud que MD5 (160 bits contra 128 bits).

SHA-1 es más robusto y seguro que MD5, pero ya se han encontrado colisiones, por tanto, actualmente esta función hash no es segura utilizarla. [14]

2.7.3 SHA-2

Las principales diferencias con SHA-1 radica en en su diseño y que los rangos de salida han sido incrementados. Dentro de SHA-2 encontramos varios tipos, el SHA-224, SHA-256, SHA-384 y SHA-512. El más seguro, es el que mayor salida de bits tiene, el SHA-512, que tiene 80 rondas (pasos), como el SHA-1 pero se diferencia de éste en:

- Tamaño de salida 512 por los 160 de SHA-1.
- Tamaño del bloque, tamaño de la palabra y tamaño interno que es el doble que SHA-1.

Como ocurre con todos los cifrados y hash, cuanto más seguro, más lento su procesamiento y uso, debemos encontrar un equilibrio entre seguridad y velocidad. [15]

2.8 Firma digital

Una firma digital es un documento que permite garantizar la integridad de un documento y se puede relacionar de manera única al firmante con su firma, ya que

realiza ésta con la llave privada y únicamente el firmante posee esa llave, ésto se traduce en que se verifica la autenticidad del firmante. Antes de entrar más en detalle de cómo se realizan las firmas digitales, es importante hablar de una función denominada "Hash" o resumen del documento. Esta función lo que hace es que a partir de un documento de tamaño N bits entrega una cadena de M bits. No hay límite para el tamaño de N, pero M siempre es de tamaño constante de acuerdo con el algoritmo usado, normalmente es de 128 o 256 bits. Una de las características de este tipo de funciones es que son unidireccionales, es decir, que debe de ser imposible a partir del resumen del documento encontrar el mensaje original. También deben cumplir la propiedad de dispersión, lo que significa que si se cambia al menos un bit del documento, su resumen debe de cambiar la mitad de sus bits aproximadamente. La firma de un documento d se realiza tomando un documento digital, se extrae el resumen del documento $H(d)$ y este resumen se cifra asimétricamente con la llave privada del firmante $Ck_1(H(d))$, esto es lo que vendría siendo la firma digital, ahora hay que ponérsela al documento, para eso se concatenan el documento y su resumen cifrado.

Ahora hay que verificar la firma, para eso se separan el documento d del resumen cifrado. Se descifra asimétricamente con la llave pública k_2 del firmante el resumen cifrado $Dk_2(Ck_1(H(d)))$ obteniéndose el resumen del documento original $H(d)$. Se obtiene el resumen del documento enviado $H(d)'$ se comparan las dos digestiones $H(d) = H(d)'$ y si estos son iguales, se dice que la firma es válida, de lo contrario es inválida. Si la firma es inválida puede deberse a dos causas: una es que se está usando una llave pública que no corresponde con la privada del firmante (problema de autenticación) o la otra es que el documento que se envió fue alterado (problema de integridad). La siguiente figura ilustra el proceso descrito de firmar y validar la firma digital. [16]

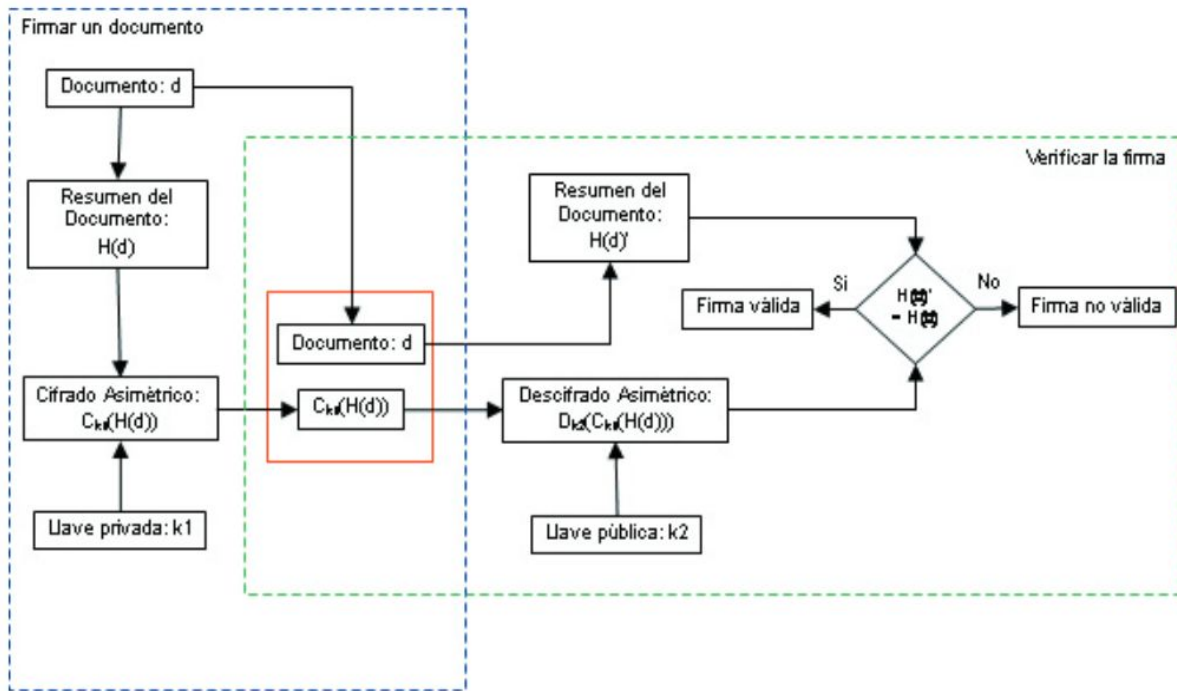


Figura 4 - Firma digital [17]

2.8.1 Certificado digital:

Se entiende por certificado digital al documento digital firmado digitalmente por un certificador, que vincula los datos de verificación de firma a su titular. (Ley 25506 - Capítulo II - Artículo 13). [17]

2.9 Resumen

En este capítulo se realiza una introducción a la criptografía y sus ventajas con respecto a la confidencialidad y seguridad, siendo esto el punto de partida para los distintos tipos de cifrado cada uno con sus ventajas y desventajas, también se hace un repaso por los distintos tipos de hash y la utilización de estos en la firma digital.

Capítulo 3

Blockchain

3.1 Definición

Según “The Blockchain (R)evolution: The Swiss Perspective”, realizada por Deloitte, *el blockchain es una gran base de datos en la cual se registran operaciones de intercambio de información entre dos o más partes. Sin embargo, esta base de datos no se encuentra almacenada en un computador específico, sino que se encuentra alojada en múltiples equipos que tienen, a su vez, acceso a toda la información, pero de forma encriptada. Es un libro registro de contabilidad*, donde se registran las distintas operaciones que, en lugar de estar guardado por una misma persona, organización u organismo, es custodiado por múltiples personas que le dan validez y garantía.

Blockchain - Definición alternativa

Una cadena de bloques o cadena articulada, conocidas en inglés como *blockchain*, es una estructura de datos en la que la información contenida se agrupa en conjuntos (bloques) a los que se les añade metainformación relativa a otro bloque de la cadena anterior en una línea temporal, de manera que gracias a técnicas criptográficas la información contenida en un bloque sólo puede ser repudiada o editada modificando todos los bloques posteriores. Esta propiedad permite su aplicación en entorno distribuido de manera que la estructura de datos blockchain puede ejercer de base de datos pública no relacional que contenga un histórico irrefutable de información. [18]

Cualquier modificación malintencionada debería hacerse de forma coordinada en todas las copias que existan, lo que lo hace virtualmente imposible.

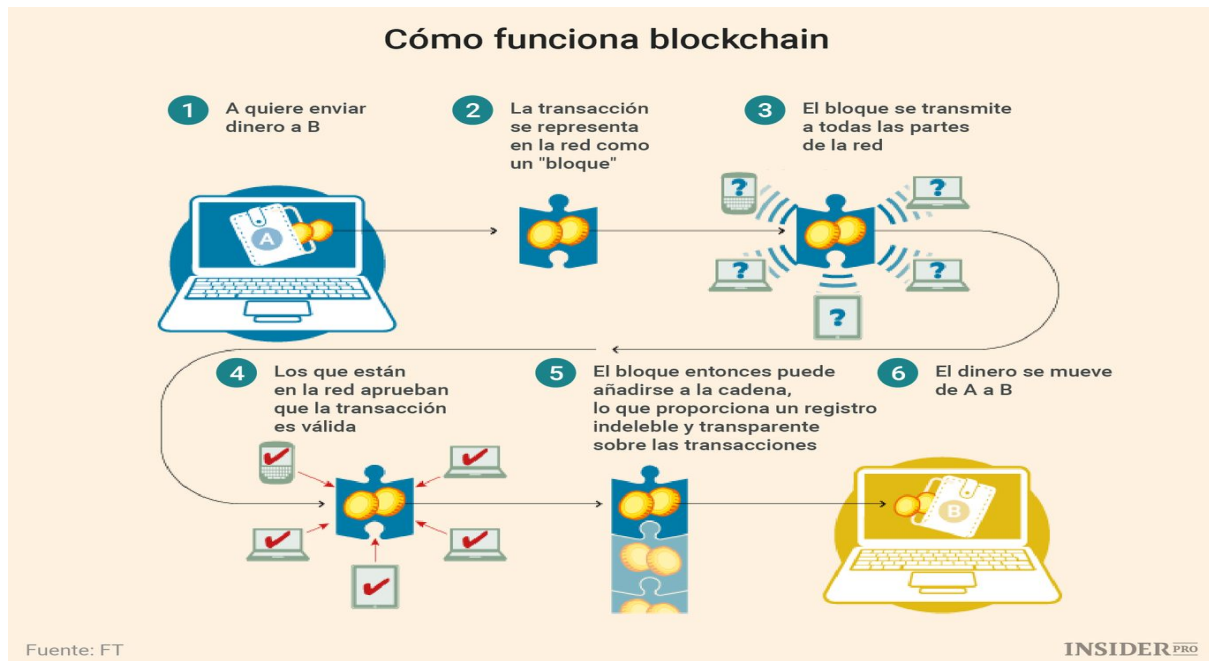


Figura 5 - Funcionamiento de blockchain [19]

3.2 Blockchain - Ejemplo en la vida real

Para empezar a entender el Blockchain -cadena de bloques- contaremos una historia. Cuatro alumnos, que no habían estudiado para un examen, deciden buscar una artimaña para dar pena al profesor y hacer que les atrase el examen. Cuentan al docente que no habían tenido tiempo para estudiar porque el fin de semana, volviendo de una boda, tuvieron un accidente con el coche. Gracias a Dios ellos salieron ilesos, pero estuvieron en el hospital acompañando a unos amigos que sí resultaron heridos. El profesor se apiada de ellos y acepta hacerles el examen días más tarde. Llega la fecha de la prueba y el profesor les coloca en cuatro clases separadas, les quita los móviles y les pone el mismo examen con cuatro preguntas:

1. ¿Quién se casaba?; 2. ¿A qué hora fue el accidente?; 3. ¿Marca y modelo en el que iban?; 4. ¿Nombre de los amigos heridos? El docente les puso una nota: Si contestan todos igual a las preguntas, tendrán un sobresaliente.

Cómo se verifica ?

El profesor no necesitó recurrir a un tercero de confianza para verificar la historia que le contaban (policías, servicios de emergencia, hospitales, etc.). Sólo validó la historia de forma distribuida. Si todos contaban lo mismo, daban datos concretos y proporcionaban el mismo nivel de detalle, estarían diciendo la verdad. Ésta es una de las características de Blockchain, eliminar al intermediario que dé fe sobre un dato concreto. La idea es basarse en una validación distribuida por un conjunto de personas (nodos), de tal manera que todos confirmen que ese dato es correcto.

[20]

3.3 - Elementos que definen una Blockchain

Los elementos que definen una Blockchain son los siguientes:

1- Criptografía de clave pública: también conocida como criptografía asimétrica, utiliza la implementación de curva elíptica para mejorar el rendimiento respecto a implementaciones tradicionales como RSA. De esta forma se valida la autenticidad del emisor de la transacción por parte de todos los nodos de la red. Para las implementaciones más populares (Bitcoin y Ethereum) esta validación se realiza comparando la clave pública del remitente con el elemento firmado con su clave privada (el elemento a firmar es una versión "hasheada" de la transacción).

2- Base de datos distribuida: cada uno de los nodos réplica completamente la base de datos al unirse a la red de la Blockchain correspondiente (actualmente se está trabajando en un "approach" relacionado con la gestión de históricos). Este proceso de réplica sincroniza todos los bloques de la cadena. Una vez sincronizada,

el nodo podrá empezar a operar con normalidad sobre la red (balance de criptomonedas, envío y recepción de transacciones). Es importante destacar que la base de datos se nutre de bloques. Es decir, hasta que una transacción no es confirmada mediante su inclusión en un bloque aceptado, la transacción en sí no se considera válida en la Blockchain.

- **Verificación de integridad con Árboles de Merkle:** La estructura está formada por un **árbol invertido de hashes, con un hash raíz en la parte superior, del cual parten nodos-rama hacia abajo**, que contienen hashes de otros nodos-rama de niveles inferiores, hasta llegar a las "hojas" del árbol, que en el caso de una cadena de bloques representan los eventos o transacciones.
- El árbol de Merkle es el producto un algoritmo **que toma como datos de entrada el conjunto de transacciones almacenadas en un bloque con el fin de verificar que las mismas no hayan sido alteradas**. El bloque no se procesa como un todo, sino que cada transacción es evaluada por separado, mientras el algoritmo las agrupa en segmentos vinculados, produciendo al final un hash del bloque completo
- Los árboles de Merkle permiten la representación arbitraria **de conjuntos de datos de tamaño considerable**, e identificar de manera eficiente si los mismos han sido comprometidos y qué parte de su estructura ha sido afectada, **sin tener que procesarlos todos. Esto es posible utilizando las llamadas pruebas de Merkle.**
- Para aplicar una prueba de Merkle **sólo hace falta conocer el hash raíz del grupo de datos que se quiere analizar**. Siempre que esta raíz sea confiable, tanto la posición como la integridad de un dato perteneciente a un determinado conjunto pueden ser verificadas, estudiando solamente la rama del árbol asociada al dato particular. Inclusive, **la prueba de Merkle no requiere que el árbol esté completo, sino que se pueda acceder al fragmento de interés.**

- La principal ventaja que se obtiene de aplicar árboles de Merkle a las cadenas de bloques es **la capacidad de verificar la integridad de cualquier parte de la cadena, un bloque específico, una transacción en particular, sin que sea necesario descargar la cadena entera.**
- Cada bloque de una blockchain es una estructura de datos. En el encabezado del bloque, junto a otros datos como la marca de tiempo, nonce, hash del bloque anterior, versión del bloque y dificultad actual, **se encuentra alojado también el hash raíz del árbol de Merkle.** Gracias a este diseño de bloque, fueron proyectados los **nodos de Verificación de Pago Simple (SPV)** que aparecen en la sección 8 del **whitepaper de Bitcoin**. Los nodos SPV, o “clientes ligeros”, sólo descargan los encabezados de bloques de la cadena más larga (actualizada), y no la cadena completa de Bitcoin. Estos nodos verifican con otros nodos de la red que, efectivamente, **los encabezados sobre los que están trabajando provienen de bloques de la cadena más larga.** Hecho esto, el nodo SPV puede proceder con el análisis de una transacción de interés mediante la aplicación de la prueba de Merkle.
- **Teóricamente, los árboles de Merkle podrían ayudar a ahorrar espacio de almacenamiento en disco.** Como el hash raíz se guarda en el encabezado del bloque, algunos bloques de la cadena, los que tienen más tiempo en ella, pueden llegar a “podarse” al eliminar las ramas del árbol que no participan en la prueba de Merkle.
- Aparte de Bitcoin, otras blockchains utilizan árboles de Merkle, esquemas similares o de mayor complejidad. Ethereum es una de esas cadenas, y **emplea una versión compuesta por varios árboles conocida como Merkle Patricia Trie (MPT).** El vocablo “trie” es prestado del idioma francés, y significa “clasificación/organización”. En computación es equivalente a usar la palabra árbol (tree). [23]

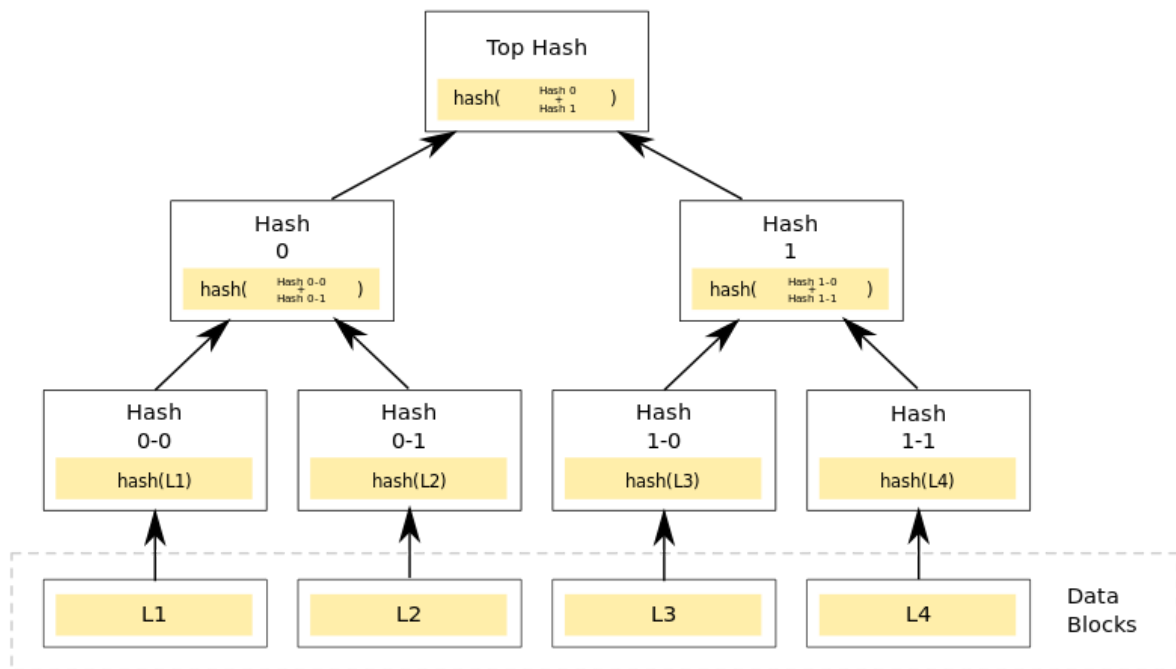


Figura 6 - Árbol de Merkle [23]

3- Algoritmo de consenso: la característica que marca la diferencia entre otros sistemas distribuidos y Blockchain es el algoritmo de consenso. Dicho algoritmo incentiva a participar mediante el envío de recompensas a los mineros encargados de crear los bloques.

Los más destacados por el momento son los siguientes:

- **Proof-of-Work:** es el algoritmo más extendido, utilizado por Bitcoin y la versión estable de Ethereum y está orientado a redes públicas.
 - Se basa en la generación de un hash teniendo en cuenta el Merkle Root (raíz hash del árbol que toma a pares los hashes de las transacciones a incluir en un bloque), el hash del bloque anterior, la hora, la dificultad y el nonce (“valor único”).
 - Los valores se van combinando por los mineros hasta generar un hash que tenga N 0s (en la elaboración de esta presentación, el valor de N para Bitcoin es 18).

- El valor N se va modificando por el protocolo para encajar con los tiempos medios de subida de bloques (10' para Bitcoin, 15" para Ethereum).
 - Una vez generado el hash de bloque correcto, se envía al resto de la red. La aceptación de dicho bloque se consigue cuando los mineros en la red empiezan a utilizar dicho hash como valor del bloque previo.
 - A cambio del bloque subido, se obtiene una recompensa por bloque y transacciones. En el caso de que se llegue a una solución del hash pero no consiga llegar a ser el nuevo bloque de la cadena (porque alguien lo haya conseguido antes). No se obtendrá recompensa en Bitcoin pero sí en Ethereum (bloques huérfanos y tíos).
- **Proof-of-Authority:** algoritmo utilizado por Ethereum para la gestión de redes privadas.
 - El coste de PoW no tiene sentido en redes privadas en las que todas las partes se conocen.
 - Para acelerar la subida de bloques y sus transacciones se definen autoridades, una por cada parte implicada.
 - Cada autoridad dispone de una clave privada en la red que permite firmar las transacciones. La parte pública de la misma se distribuye al resto de autoridades.
 - Cuando se emite una transacción, se valida la firma del remitente y se incluye en un bloque que sube "inmediatamente".
 - De esta forma los tiempos se reducen exponencialmente, basado en una confianza predefinida.

- **Proof-of-Stake (en desarrollo):** algoritmo en desarrollo por Ethereum (y otros protocolos de criptomonedas) que se basa en que las direcciones que tienen una mayor participación en la red (mayor número de criptomonedas, “stake”) tendrán una mayor posibilidad de subir nuevos bloques. [21]

- **Proof of Elapsed Time (PoET):** Este algoritmo fue desarrollado inicialmente por Intel. El algoritmo de consenso de Prueba de tiempo transcurrido o *proof of elapsed time* emula la Prueba de trabajo de estilo Bitcoin. La implementación de Sawtooth de Hyperledger es un ejemplo de PoET.

En lugar de competir para resolver el desafío criptográfico y extraer el próximo bloque, como en el blockchain de Bitcoin, el algoritmo de consenso PoET es un híbrido de una lotería aleatoria y de un orden de llegada. En PoET, cada validador recibe un tiempo de espera aleatorio. El validador con el menor tiempo de espera para un bloque de transacción en particular es elegido líder. Este “líder” crea el siguiente bloque de la cadena.

Para garantizar que efectivamente no se está “mintiendo” para ser el líder del bloque de la cadena, PoET confía en unas instrucciones “seguras” de Intel, ejecuta una instrucción SGX del procesador para generar este tiempo aleatorio de espera. [22]

- **Simplified Byzantine Fault Tolerance (SBFT):** El algoritmo de consenso SBFT implementa una versión adaptada del algoritmo *practical byzantine fault tolerance* (PBFT), y busca proporcionar mejoras significativas sobre el protocolo de consenso *proof of work* de Bitcoin. La idea básica involucra a un único validador que agrupa las transacciones propuestas y forma un nuevo bloque. Tenga en cuenta que, a diferencia de la cadena de bloques de Bitcoin, el validador es una

parte conocida, dada la naturaleza permitida del *ledger* libro mayor. El consenso se logra como resultado de un número mínimo de otros nodos en la red que ratifican el nuevo bloque. Para ser tolerante a errores bizantinos, el número de nodos que debe alcanzar el consenso es $2f + 1$ en un sistema que contiene $3f + 1$ nodos, donde f es el número de fallas en el sistema. Por ejemplo, si tenemos 7 nodos en el sistema, entonces 5 de esos nodos deben estar de acuerdo si 2 de los nodos están actuando de manera incorrecta.

El ejemplo práctico sería el de ByzCoin, que busca realizar mejoras importantes sobre el protocolo original de Bitcoin. Al abordar el desafío de la escalabilidad debido a la alta latencia, las transacciones de ByzCoin se comprometen irreversiblemente con la cadena de bloques en cuestión de segundos. La ventaja adicional es que los árboles de comunicación optimizan los compromisos de transacción y la verificación en operaciones normales. [23]

- **Raft** es un protocolo de consenso que trabaja eligiendo un líder central sobre el que se hacen las peticiones y coordina al resto de nodos (seguidores) para implementarlas. El ejemplo típico de uso de este algoritmo es para la escritura de mensajes en un log replicado.

Roles

Los nodos participantes en el algoritmo pueden estar en tres estados:

Líder.- Todos los cambios que se realicen en el clúster de nodos pasan por él primero. Como máximo puede haber uno en cada momento. Si se cae el líder actual se abre un proceso para elegir al siguiente.

Candidato.- Nodo que no ha encontrado líder y solicita su elección como tal. El líder será seleccionado entre los distintos candidatos.

Una vez que un nuevo líder ha sido elegido se dice que comienza un **término**

Seguidor.- Nodo cuya responsabilidad es actuar frente a las peticiones del nodo líder.

Funcionamiento

Supongamos que estamos en un instante en el que todos los nodos son seguidores y por tanto están esperando recibir comunicación de un nodo que actúe como líder. Cada nodo tiene un tiempo de espera **aleatorio** después del cual, si un líder no se comunica con él, pasa a estado candidato y pide ser elegido como líder.

Para ello el algoritmo divide el tiempo en **términos** que son plazos de tiempo definidos por el algoritmo. Para que un nodo sea elegido como líder es necesario que cuando pasa a estado candidato reciba la mayoría de votos de los seguidores en un **término** (se dice que hay quórum). Si tenemos N nodos la mayoría es al menos $(N/2)+1$ nodos. Solo los nodos que se encuentren en estado seguidor pueden votar una única vez a la primera solicitud que reciban dentro de un plazo. Si un equipo candidato no recibe los votos suficientes en el plazo y agota su tiempo de espera, espera al siguiente término y repite el proceso de votación. Para evitar problemas de varios nodos sincronizados en las peticiones, el tiempo de espera es **aleatorio**.

Una vez elegido el líder esté, cada cierto tiempo, debe enviar mensaje, ya sea con información de actualización o no, a fin de que los nodos seguidores no agoten su tiempo de espera.

Supongamos que se pide un cambio en la información: El líder registra el cambio, lo pasa al resto de nodos para que lo registren y,

una vez que un quórum de estos responde con que el **cambio ha sido registrado**, el líder hace efectivo el cambio en su sistema y finalmente notifica a los nodos seguidor indicando que el **cambio ha sido efectivo**. [24]

3.4 Bitcoin: es el “padre” del Blockchain. Es concebido como “el oro” de las criptomonedas (comparativa relacionada con la proporción de oro respecto a fiat que existió en el pasado con monedas “físicas”).

Las principales cualidades son las siguientes:

- Se basa en el envío de transacciones entre hashes de claves públicas (“direcciones”). **Estas transacciones no están encriptadas**. Las transacciones son emitidas adicionalmente a todos los nodos participantes en la Blockchain.
- Se utilizan “comandos” denominados “**opcodes**” para ejecutar las posibles operaciones. Es un lenguaje basado en pila, de izquierda a derecha. Esto implica que **Bitcoin no es Turing completo** (impidiendo bucles y condiciones). Está hecho a propósito para evitar posibles ciberataques.
- En las transacciones, es necesario especificar una **dirección para el cambio** por cómo está implementado el protocolo. Es decir, si tengo 1 BTC y quiero transferir 0.4, si no indico qué hago con el 0.6 restante será enviado al minero que incluya dicha transacción en un bloque como incentivo. Es decir, **hay que incluir una segunda dirección para el cambio (la nuestra)** en la transacción.
- Los nodos mineros “recogerán” las transacciones (incentivadas con BTC) y empezarán a ejecutar la PoW de acuerdo a lo descrito en anteriores diapositivas **para generar el hash de un bloque**.

- La recompensa por haber subido un bloque a la cadena se ve reflejada en la primera transacción de unos de los bloques posteriores (**es conocida como transacción “coinbase”, generando nuevos BTC**).
- El **límite máximo de BTC** que podrá distribuirse por la red es de **21 millones** (por características del protocolo).
- El balance de BTC de cada dirección se realiza a través de Unspent Transaction Output (UTXO). Se accede a los hashes de las transacciones que no han gastado BTC para una cuenta, de tal manera que para comprobar el balance de una dirección hay que ir “hacia atrás” en estas transacciones para ir sumando o restando los valores transferidos. Este comportamiento se define como “transaction-based”. [25]

3.5 Ethereum: es popularmente conocido por la concepción de los **Smart Contracts** (programas que se ejecutan en la máquina virtual de Ethereum (EVM) que permiten añadir funcionalidad más allá de la transferencia de criptomonedas a la Blockchain.

Las principales cualidades son las siguientes:

- Utiliza **actualmente el algoritmo de consenso PoW**, pero actualmente se encuentra en un proceso de adaptación a PoS. También da soporte para redes privadas a PoA.
- Utiliza LLL, Serpent y **Solidity** como lenguajes de programación (principalmente este último) para la definición de Smart Contracts.
- Estos lenguajes **son Turing completos**, potenciando con ello el ecosistema de desarrollo sobre la plataforma. No obstante, esto implica que el sistema es potencialmente hackeable (de hecho, ya lo ha sido).
- El hacking realizado fue durante un proceso conocido como “crowdsales” para la Decentralized Autonomous Organization (**DAO**).

- En resumen, se creó una empresa autónoma a través de Smart Contracts, y uno de los métodos facilitaba la compra de tokens (“acciones”) de la empresa pagando con ETH.
- Debido al comportamiento de las **funciones de callback en Solidity**, el atacante pudo hacerse con 150M de dólares en ETH. Este evento provocó lo que se conoce como un “hard fork” en el que se creó una nueva moneda (Ethereum Classic, ETC) con la versión previa al ataque del software de esta Blockchain. ETH es la versión modificada del software tras el ataque.
- Un concepto asociado a la ejecución de los Smart Contracts es **el gas** (unidad de energía). Esta unidad mide la ejecución de un Smart Contract en la red, debiendo pagar en ETH dicho gasto. Existe un límite de gas (4,7 millones por bloque) para que la ejecución de un Smart Contract no repercuta sobremanera en los recursos generales de la red.
- Una diferencia con Bitcoin es la **recompensa a los mineros por aquellos bloques que no forman parte de la cadena principal** (“stales” hasta que se incluyen en la cadena principal, a partir de dicho momento pasan a denominarse “tíos”).
- Ethereum es un **protocolo “account-based”**, es decir, la dirección (cuenta) no debe ir hacia atrás en sus transacciones no gastadas para obtener el balance total, ya que el valor está asociado de forma inmediata a la misma.
- El **Patricia Trie es la estructura de datos que utiliza Ethereum** para almacenar su información (también conocidos como Merkle Patricia Trie). Dentro de estos Patricia Tries, los bytecodes de los Smart Contracts se encuentran en los llamados “Storage Trie” (al desplegarse es asociado a una cuenta nueva). [26]

3.5.1 API de Ethereum: La más extendida es la implementada en **JavaScript** (web3.js). Proporciona una interfaz JSON RPC para interactuar con la Blockchain.

Los **clientes** más populares para esta API son los siguientes:

- **TestRPC:** es el más básico de todos y está destinado para un entorno local. Utiliza un único nodo para la red Blockchain y proporciona varias cuentas por defecto.
- **Geth:** cliente implementado utilizando Golang. Sería el adecuado para un entorno de Desarrollo, al permitir la utilización de diversos nodos. Permite implementar algoritmos de consenso diferentes, como PoW y PoA.
- **Parity:** el cliente más cómodo para el desarrollador y eficiente en el uso de los datos de la Blockchain. Incluye una UI para crear cuentas, enviar transacciones, desplegar Smart Contracts y toda la funcionalidad necesaria para interactuar con la misma. Permite implementar algoritmos de consenso diferentes, como PoW y PoA. Está basada en Geth.
- **Truffle** es el framework de **JavaScript** que facilita la gestión de los **Smart Contracts**, compilando y desplegando automáticamente los mismos (o no haciéndolo si detecta que no hay cambios) además de proporcionar servicios como test unitarios.
- **Hyperledger:** es una plataforma de “Blockchain de negocio” promovida por la **Linux Foundation**. Esta plataforma es aprovechada por diversos desarrolladores que han creado diferentes versiones (Burrow, Sawtooth...). La más popular es la implementada por IBM, **Fabric**, cuyas cualidades son las siguientes:
 - **No utiliza criptomonedas.** Al estar concebido para redes internas, no tiene sentido incentivar a las partes para que participen en la red.
 - **Usa redes autorizadas.** Los nodos de la red utilizan diferentes CA (Certification Authorities) para firmar con los certificados asociados a su grupo.
 - **Algoritmo de consenso.** A partir de la versión 1.4 se utiliza el algoritmo de consenso raft.

¿Por qué hablar de ella?

- A las empresas importantes que tienen un entorno altamente “IBMizado” les resulta interesante utilizar un nuevo componente de la “suite” que les ha funcionado hasta el momento.
- En líneas generales, las compañías siguen sin estar convencidas de tener Blockchain públicas à Hyperledger Fabric aporta un entorno privado securizado por la criptografía.
- Los Smart Contracts se denominan **Chaincodes**. Estos Chaincodes pueden ser desarrollados en Golang (plenamente funcional) y Java (en progreso). Se despliegan en contenedores Docker, que emulan el comportamiento de la EVM de Ethereum.
- La **primera versión** de Hyperledger Fabric fue anunciada a **mediados de julio de 2017**. [27]

3.5.2 La máquina virtual de Ethereum (EVM):

Ethereum es una blockchain programable. La cual en lugar de ofrecer a los usuarios un ciertas operaciones predefinidas (Por ejemplo las transacciones de bitcoin) Ethereum permite a los usuarios crear operaciones de cualquier complejidad que ellos deseen. De esta manera, esta funciona como una plataforma para varios diferentes tipos de aplicaciones descentralizadas basadas en la blockchain, incluyendo pero no limitado a criptomonedas.

Ethereum en su más estricto sentido se refiere a una suite de protocolos que definen una plataforma ideal para aplicaciones descentralizadas. En el corazón de esta se encuentra la Máquina Virtual de Ethereum o mejor conocida como Ethereum Virtual Machine (“EVM”), la cual puede ejecutar código de complejidad algorítmica de manera arbitraria. En términos de ciencias de la computación, Ethereum es “Turing complete” o Turing completo, lo que significa que teóricamente podría resolver cualquier problema computacional razonable. Los desarrolladores pueden crear aplicaciones que se ejecutan en la Máquina Virtual de

Ethereum usando lenguajes de programación amigables modelados en lenguajes existentes como JavaScript y Python.

Como cualquier otra blockchain, Ethereum también incluye un protocolo de red P2P o de Igual a Igual. La base de datos de la blockchain de Ethereum es mantenida y actualizada por una gran cantidad de nodos conectados a la red. Todos y cada uno de los nodos en la red de ethereum ejecutan la máquina virtual de Ethereum y ejecutan las mismas instrucciones. Por esta razón, algunas veces Ethereum es llamada la Computadora mundial o en inglés "World computer".

Esta paralelización masiva de poder computacional a lo largo de toda la red de Ethereum no se realiza para volver la computación más eficiente. De hecho, este proceso hace la computación en Ethereum mucho más lenta y costosa que en una "computadora" tradicional. En lugar de esto, cada nodo de Ethereum ejecuta la máquina virtual de Ethereum para mantener un consenso a través de toda la blockchain. El consenso descentralizado otorga a Ethereum muy altos niveles de tolerancia ante errores, asegura que la red nunca se encuentre inactiva y vuelve la información almacenada en la blockchain imposible de modificar y resistente a la censura.

Ethereum está hecha para **aplicaciones que automatizan interacciones directas entre iguales o facilitan la organización de grupos dentro de una red**. Por ejemplo, aplicaciones para coordinar tiendas donde se venden productos entre iguales, o la automatización de contratos financieros complejos. Además de las aplicaciones financieras, cualquier situación en la que la confianza, seguridad y permanencia son importantes – por ejemplo, registros de activos, votaciones, gobierno, y el internet de las cosas "IoT" – podrían ser impactados de manera positiva por la plataforma de Ethereum. [25]

3.5.3 ¿Cómo funciona Ethereum?

Al tiempo que la blockchain de Bitcoin era únicamente una lista de transacciones, la unidad básica de Ethereum es la cuenta o "Account". La blockchain de Ethereum

rastrea el estado de todas las cuentas, y todas las transiciones de estado en la blockchain son transferencias de valores e información entre las cuentas.

Actualmente existen dos tipos de cuentas:

- Cuentas propiedad de externos o (EOAs) Externally Owned Accounts, las cuales son administradas con claves privadas.
- Cuentas basadas en Contratos, las cuales son controladas y administradas por el código de sus contratos y únicamente pueden ser activadas por una cuenta propiedad de externos o (“EOAs”)

Para la mayoría de los usuarios, la diferencia entre estas es que los humanos manejan las EOAs porque pueden controlar las claves privadas lo cual otorga control sobre las mismas. Las cuentas contrato, en la otra mano, son gobernadas por su código interno. Si son controladas por un usuario humano, es porque están programadas para ser controladas por un EOA con una cierta dirección preestablecida, la cual es controlada por quien sea que tenga la llave privada que controla esa multicitada EOA. El término popular “Contratos Inteligentes” o “smart contracts” hace referencia a un código en la cuenta contrato – programas que se ejecutan cuando una transacción es enviada a esa cuenta. Los usuarios pueden crear contratos nuevos desplegando código hacia la blockchain.

Al igual que con el Bitcoin, los usuarios deben pagar pequeñas comisiones por transacción a la red, esto protege la red de ethereum de tareas computacionales maliciosas como son los ataques de denegación de Servicios o “DDoS” y los bucles infinitos. El remitente de una transacción debe pagar por cada paso del programa que activaron incluyendo el poder computacional y memoria de almacenamiento. Estas comisiones son pagaderas en la moneda moneda de valor nativa de Ethereum denominada, ether.

Estas transacciones son recolectadas por los nodos que validan las transacciones. Estos “mineros” son nodos en la red de ethereum que reciben, propagan, verifican y ejecutan transacciones. Los mineros agrupan las transacciones, las cuales incluyen actualizaciones a el estado de las cuentas dentro de la blockchain de

ethereum – lo que se llama “bloques”, y los mineros compiten entre unos y otros para que su bloque sea el siguiente en ser añadido a la cadena. Los mineros son recompensados con ether por cada bloque efectivamente minado. Esto provee un incentivo económico para la gente que dedica hardware y energía eléctrica a la red de ethereum.

3.5.4 Modelo de tokens

El modelo de financiamiento seguido por la Fundación Ethereum se conoce como *initial coin offering* (ICO) u oferta inicial de monedas. Es un tipo de crowdfunding (financiamiento colectivo) para proyectos de código abierto. Permite a la comunidad contribuir con proyectos que les resultan valiosos. Los miembros de la comunidad transfieren criptomonedas, y los desarrolladores del proyecto les dan a cambio un token que cumple cierta función dentro de la plataforma.

La visibilidad que obtuvieron algunos proyectos que realizaron ventas de tokens exitosas hizo que muchos emprendedores comenzaran a plantearse la utilización de ICOs como forma de financiamiento. Sin embargo, es necesario realizar unas aclaraciones.

Las ICOs no sirven para todo tipo de proyectos. Los modelos de tokens generalmente son aptos para plataformas basadas en efectos de red. Es decir, plataformas que conectan a compradores con vendedores, y donde la presencia de ambas partes es necesaria para su funcionamiento.

Una posible aplicación sería una organización descentralizada similar a Uber. Esta empresa conecta a choferes con pasajeros. Cuantos más pasajeros demanden servicio de transporte, más incentivos tendrán los choferes de sumarse a la red. Cuanto más choferes haya en la red, más fácil será conseguir un auto y más incentivo tendrán los pasajeros para utilizar Uber.

Si quisiéramos construir una versión descentralizada de Uber (llamémosla, dUber), podríamos crear un token específico para realizar pagos en la red. Los choferes podrían recibir su pago en dUber coins. Los primeros usuarios podrían recibir una

cantidad importante de monedas por cada viaje (al igual que los mineros de bitcoin recibían una cantidad importante de monedas en los primeros tiempos). Esto genera un incentivo para que los usuarios se unan a la red desde los primeros tiempos. Si la plataforma es exitosa, los primeros usuarios ganarán una cantidad considerable de dinero a través de la apreciación de la moneda.

Este tipo de recompensa a los primeros usuarios permite romper el problema del “huevo y la gallina” que típicamente afecta a las plataformas. Como los usuarios se vuelven propietarios de una parte de la red, tienen un incentivo a empezar a usarla desde el comienzo y conseguir que la utilice la mayor cantidad de gente posible. Los fundadores del proyecto podrían realizar una venta de tokens que permita recaudar fondos para el desarrollo y también poner los tokens en manos de los usuarios.

En general, los fundadores conservan entre un 10 y un 15% de los tokens, como su incentivo por impulsar el desarrollo del producto. Otro porcentaje queda en manos de una entidad (que suele ser una fundación), que tendrá la responsabilidad de avanzar en el desarrollo y cumplir con las especificaciones del producto prometido. Y entre un 60 y 70% de los tokens salen a la venta pública.

Si la plataforma funciona, el token debería apreciarse y volverse un excelente negocio para todos los que participaron desde las primeras etapas. De esta forma, el blockchain permite conciliar el desarrollo de proyectos open source con los incentivos económicos del mundo de las startups tradicionales. [28]

3.5.5 Estándar ERC-20 para Token

A diferencia del protocolo de Bitcoin, el protocolo de Ethereum fue diseñado explícitamente para hacer más que simplemente crear y registrar transferencias de sus propias criptomonedas.

Es un protocolo más generalizado y permite crear otros tokens ‘encima’ de su blockchain.

Los tokens que se ajustan a estas especificaciones se conocen como tokens ERC-20

(ERC significa *Solicitud de Comentarios de Ethereum*) y hoy en día **son el estándar más utilizado en la industria de las criptomonedas** para crear nuevos tokens.

ERC20

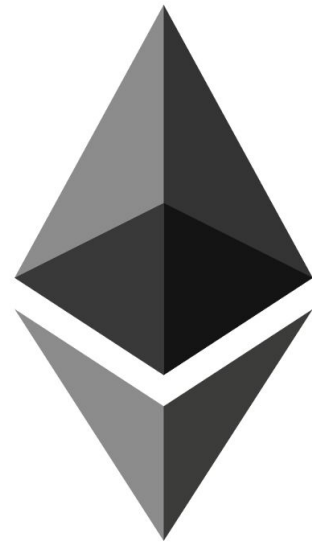


Figura 7 - ERC 20 [29]

En esencia, los tokens ERC-20 son contratos inteligentes que se ejecutan en la blockchain de Ethereum. Mientras que estos tokens funcionan dentro del marco establecido por el equipo de Ethereum, el marco es lo suficientemente amplio como para permitir simultáneamente a los desarrolladores una flexibilidad considerable en el diseño y la función de los tokens.

La mayoría de los tokens creados para lanzar ICOs en Ethereum cumplen con el estándar ERC-20.

¿Y por qué alguien querría crear un token 'encima' de Ethereum?

1. Principalmente porque estos tokens creados mediante el estándar ERC-20 **se benefician de la infraestructura existente** de Ethereum en lugar de tener que construir una blockchain completamente nueva para ellos, con el ahorro de tiempo y recursos que eso conlleva.

2. Por otra parte la creación de nuevos tokens **fortalece el ecosistema** de Ethereum impulsando la demanda del Ether, lo que hace que toda la red se vuelva aún más segura.
3. Y por último por su **interoperabilidad**. Si todos los tokens creados en la red Ethereum utilizan el mismo estándar, esos tokens serán fácilmente intercambiables y podrán trabajar fácilmente con otras Dapps del mismo ecosistema.

Lo que hace un token “estandarizado” es que utiliza un cierto conjunto de funciones. Si los desarrolladores saben de antemano cómo funcionará un token, pueden integrarlo fácilmente en sus proyectos con menos temor a cometer errores.

Si varios tokens se comportan de manera similar, llamando a las mismas funciones de la misma manera, entonces una Dapp puede interactuar más fácilmente con diferentes sub-monedas.

En resumen, **el ERC-20 define una lista común de reglas para todos los tokens creados encima de Ethereum**, lo cual facilita muchísimo la tarea a los desarrolladores al permitirles predecir con precisión cómo funcionarán esos nuevos tokens dentro del sistema.

¿Afecta entonces el valor del Ether al valor del token?

En principio no. El valor particular de cualquier token dependerá de una variedad de factores descritos en el código de la aplicación (por ejemplo el suministro total de tokens), así como la demanda del token en el mercado o por la especulación sobre su valor futuro.

Lo importante es que el hecho de que sea la Máquina Virtual Ethereum (EVM) la que ejecuta los cálculos compartidos que describen la distribución y los movimientos del token no significa que el valor del token sea el mismo que el del

Ether.

El Ether se usará como 'gas' para ejecutar cualquier aplicación relacionada con los tokens (y en cierto sentido si el valor del Ether cayera en picado podría poner en peligro la estabilidad y utilidad de los tokens que se ejecutan sobre Ethereum), pero **el valor del token generalmente está desvinculado del valor del Ether.**

El desarrollo de un token ERC-20

Adentrándonos un poco más en el aspecto puramente técnico de este estándar (en el que aún hoy en día la fundación y la comunidad de Ethereum sigue trabajando para mejorar), puedes encontrar sus especificaciones en [esta página de Github](#). En ella se explica que el código del token debe de estar formado por nueve métodos y dos eventos:

- Métodos:

- Name (opcional) – Nombre del token.
- Symbol (opcional) – Símbolo del token.
- Decimals (opcional) – El número de decimales que utiliza el token.
- TotalSupply – Suministro total de tokens que existirán.
- BalanceOf – Saldo de la cuenta del propietario.
- Transfer – Transferencia a...
- TransferFrom – Transferencia desde...
- Approve – Permite la retirada de fondos.
- Allowance – Devuelve la cantidad que se puede retirar.

- Eventos:

- Transfer – Activado cuando se transfieren los tokens.
- Approval – Activado siempre que se aprueba la transferencia.

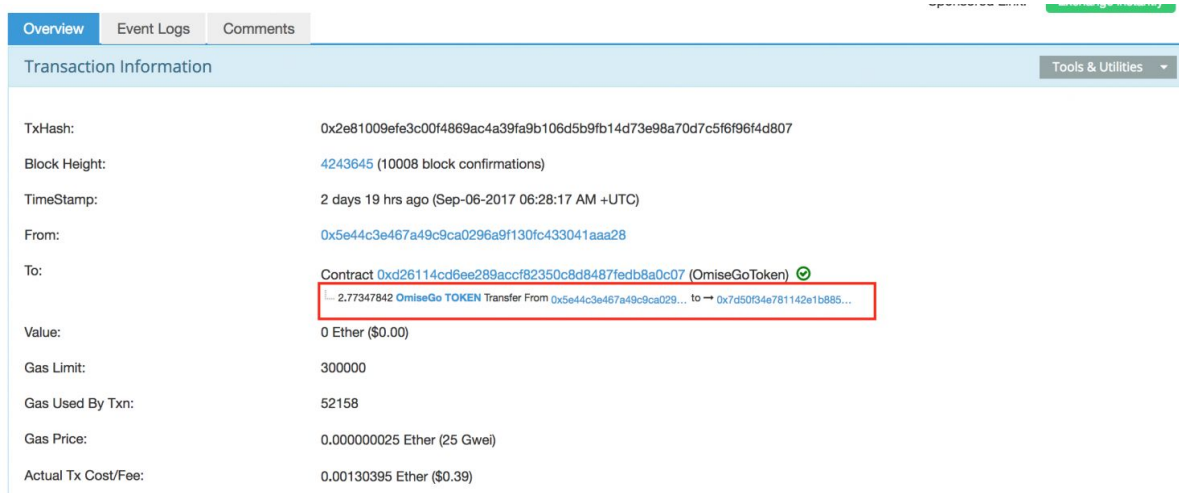
En general, estos son tipos básicos de funcionalidad, que incluyen cómo se transfieren los tokens y cómo los usuarios pueden acceder a los datos sobre un

token. En la [wiki de Ethereum](#) podrás encontrar más información técnica al respecto.

Juntos, este conjunto de funciones y eventos aseguran que los tokens de Ethereum de diferentes tipos funcionarán normalmente de la misma forma en cualquier lugar dentro del sistema Ethereum. Esto significa que casi todos los monederos (*wallets*) que admiten la moneda Ether también admiten tokens compatibles con ERC-20.

Igual que Bitcoin y Ether, los tokens ERC-20 también puede rastrearse en la blockchain

Esto se debe a que los tokens son sólo un tipo específico de contrato inteligente que “vive” en la cadena de bloques Ethereum, tal y como podemos ver en [ésta transacción](#) de ejemplo a continuación:



The screenshot displays the 'Transaction Information' section of an Ethereum transaction. The 'To:' field is highlighted with a red box and contains the text: 'Contract 0xd26114cd6ee289accf82350c8d8487fedb8a0c07 (OmiseGoToken) ✓' followed by a redacted line: '2.77347842 OmiseGo TOKEN Transfer From 0x5e44c3e467a49c9ca0296a9f130fc433041aaa28... to 0x7d50f34e781142e1b885...'. Other transaction details include TxHash, Block Height (4243645), TimeStamp (2 days 19 hrs ago), From address, Gas Limit (300000), Gas Used (52158), Gas Price (0.00000025 Ether), and Actual Tx Cost/Fee (0.00130395 Ether).

Field	Value
TxHash:	0x2e81009efe3c00f4869ac4a39fa9b106d5b9fb14d73e98a70d7c5f6f96f4d807
Block Height:	4243645 (10008 block confirmations)
TimeStamp:	2 days 19 hrs ago (Sep-06-2017 06:28:17 AM +UTC)
From:	0x5e44c3e467a49c9ca0296a9f130fc433041aaa28
To:	Contract 0xd26114cd6ee289accf82350c8d8487fedb8a0c07 (OmiseGoToken) ✓ 2.77347842 OmiseGo TOKEN Transfer From 0x5e44c3e467a49c9ca0296a9f130fc433041aaa28... to 0x7d50f34e781142e1b885...
Value:	0 Ether (\$0.00)
Gas Limit:	300000
Gas Used By Txn:	52158
Gas Price:	0.00000025 Ether (25 Gwei)
Actual Tx Cost/Fee:	0.00130395 Ether (\$0.39)

Figura 8 - Transacción de ethereum [30]

A primera vista, esta transacción puede parecerse a una transacción de Ether vacía, ya que el “valor” del Ether es cero, pero si observamos el texto en el recuadro rojo vemos que se trata del envío de tokens OmiseGo (2.77 OMG) y las direcciones de red de Ethereum que estuvieron involucradas en él.

La dirección de encima del recuadro rojo “*Contract 0xd26114...*” es el Contrato inteligente de **OmiseGo** – una aplicación que maneja la distribución y transferencias de tokens OMG en la red Ethereum.

Por tanto podemos deducir que, efectivamente, OmiseGo utiliza el estándar ERC-20 para sus tokens OMG.

Personalización de los tokens

Además de los métodos estándar que acabamos de ver, los tokens ERC-20 se pueden personalizar para habilitar las siguientes funciones:

1- **Compra y venta automática:** Puedes vincular el valor del token con el de otro token o divisa creando un fondo que compra o vende automáticamente tokens para mantener el saldo.

2- **Recarga automática:** Las transacciones en la blockchain de Ethereum requieren pagos a los mineros en ‘gas’. Puedes programar tu token para recargar el gas automáticamente de cara a transacciones futuras si cae por debajo de cierto nivel.

3- **Agregar una ‘casa de moneda’ central que pueda cambiar el número de tokens en circulación:** podría ser útil si tu token refleja o simula monedas del gobierno.

4- **Tokens congelados:** Si un organismo regulador te lo indica, puedes congelar los tokens propiedad de un usuario y descongelarlos cuando sea necesario.

5- **Proof-of-Work:** Puedes vincular su suministro de tokens al suministro de Ether programando un contrato para ejecutar una “minería fusionada” con Ethereum. Un minero que encuentre un bloque en Ethereum también obtendrá un número predeterminado de tus tokens como recompensa de bloque. [\[31\]](#)

Las Propuestas ERC-223 y ERC-721

En algunas situaciones, los tokens ERC-20 pueden presentar dificultades de cara a los usuarios. Por ejemplo, si alguien utiliza un token ERC-20 para enviar 3 ETH a un

contrato que no es compatible con ERC-20, la transacción no será rechazada porque el contrato no reconocerá la transacción entrante. El ETH podría quedarse atascado en el limbo y acabar perdiéndose.

3.5.6 Modelo de token ERC-223

ERC-223 es muy parecido a ERC-20, solo que más nuevo y mejor que el ERC-20. En pocas palabras, ERC-223 es una versión mejorada y modificada del ERC-20.

Al igual que un token en Ethereum que se considera como ERC-20, tiene que incorporar ciertos estándares y funciones en su código en la red de Ethereum, ERC-223 también tiene ciertas funciones obligatorias que debe incluir mientras se implementa para cumplir con el estándar ERC-223.

La motivación es, por supuesto, hacer mejoras continuas. A continuación se presentan los objetivos enumerados para el mismo:

- 1- Para detener la pérdida de fichas.
- 2- Para hacer un mecanismo de transferencia de fichas similar a la transferencia de Ether.
- 3- Para corregir la incapacidad de manejar las transacciones de token entrantes para tokens no compatibles
- 4- Para eliminar un paso del proceso de dos pasos de transacciones que ocurren en una transferencia de token.

Ventajas de la norma ERC 223 Token

Las siguientes son las 3 ventajas clave de la actualización al estándar ERC 223:

1- Elimina el problema de la pérdida de tokens que ocurre durante la transferencia de tokens ERC-20 a un contrato (cuando las personas usan erróneamente las instrucciones para enviar tokens a una billetera). ERC-223

permite a los usuarios enviar sus tokens a la billetera o al contrato con la misma función de transferencia, eliminando así la posibilidad de confusión y la pérdida de tokens.

2- Permite a los desarrolladores manejar las transacciones de token entrantes y rechazar tokens no compatibles. En este caso, no perderá los tokens, ya que le será devuelto el reembolso menos el Gas, algo que no es posible con ERC-20.

3- Ahorro de energía: la transferencia de tokens ERC-223 a un contrato es un proceso de un solo paso en lugar de un proceso de 2 pasos (para ERC-20), y esto significa dos veces menos gas y no más hinchazón de blockchain. Esto, como resultado, también reduce las tarifas de transacción que se pagan por la transferencia de tokens. [\[32\]](#)

3.5.7 Modelo de token ERC-721

El desarrollo de los token ERC-721 fue una propuesta de mejora presentada por Dieter Shirley a finales de 2017 y tiene como peculiaridad, permitir que los contratos inteligentes funcionan como tokens intercambiables de forma similar a los ERC-20, con la única diferencia de que estos 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. De esta forma la red puede contener ciertos tokens más deseables que otros y de mayor valor para los miembros de la comunidad.

Es así como se ha desarrollado un nuevo mercado en la plataforma de Ethereum y un grupo de usuarios coleccionistas de tokens, que funcionan bajo el estándar ERC-721.

¿Cuáles son las Características principales de un Token ERC-721?

La fungibilidad es una característica básica de un activo físico o digital como lo son

en este caso los tokens ERC-20, determinando la posibilidad de intercambio por otros de igual valor o su capacidad de ser gastados en fracciones. Sin embargo, La principal características de los tokens ERC-721 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 ERC-20 que se pueden dividir hasta en fracciones de una millonésima, los tokens ERC-721 se mantienen en su estado original y se compran, venden o intercambian de esa forma.

Como resultado, las características únicas de los ERC-721 es lo que los hace no fungibles y que se valoren de forma distinta según su singularidad y rareza, en el intercambio que se utilice. Tomando en cuenta además, que no siempre se pueden usar de forma indistinta.

Es así como de esta forma, se añade el elemento coleccionable de los tokens. Entonces si algún usuario valora más sus cualidades distintivas que otro, terminará ofreciendo un mayor precio por su adquisición.

Los tokens ERC-721 se pueden usar en cualquier intercambio, pero su valor es el resultado de la singularidad y la visibilidad asociada con cada token. El estándar define las funciones name , symbol , totalSupply , balanceOf , ownerOf , approve , takeOwnership , transfer , tokenOfOwnerByIndex , y tokenMetadata . También define dos eventos: Transfer y Approval.

Similar al ERC-20, el estándar ERC-721 recientemente propuesto ha abierto una puerta de entrada para que los nuevos contratos inteligentes actúen como elementos no fungibles. Como se puede ver en aplicaciones como CryptoKitties , Decentraland , CryptoPunks y muchas otras, los tokens no fungibles han demostrado ser productos de muy alta demanda. [33]

3.6 Smart Contracts

Para entender un *smart contract* primero hemos de recordar que significa un contrato. Un contrato no es más que un acuerdo entre dos o más partes, un entorno donde se define lo que se puede hacer, cómo se puede hacer, qué pasa si algo no se hace... Es decir, unas reglas de juego que permite, a todas las partes que lo aceptan, entender en qué va a consistir la interacción que van a realizar.

Hasta ahora los contratos han sido documentos verbales o caros documentos escritos, sujetos a las leyes y jurisdicciones territoriales, y en ocasiones requiriendo de notarios, es decir, más costes y tiempo. Algo no accesible para cualquier persona. Y esto no es lo peor: los contenidos de los contratos pueden estar sujetos a la interpretación.

En cambio un contrato inteligente es capaz de ejecutarse y hacerse cumplir por sí mismo, de manera autónoma y automática, sin intermediarios ni mediadores. Evitan el lastre de la interpretación al no ser verbal o escrito en los lenguajes que hablamos. Los *smart contracts* se tratan de "scripts" (códigos informáticos) escritos con lenguajes de programación, siendo los términos del contrato puras sentencias y comandos en el código que lo forma.

Por otro lado, un *smart contract* puede ser creado y llamado por personas físicas y/o jurídicas, pero también por máquinas u otros programas que funcionan de manera autónoma. Un *smart contract* tiene validez, sin depender de autoridades, debido a su naturaleza: es un código visible por todos y que no se puede cambiar al existir sobre la tecnología *blockchain*, la cual le da ese carácter descentralizado, inmutable y transparente. [34]

3.6.1 ¿Cuál es el potencial de esto?

Es importante destacar que, al estar distribuido por miles de ordenadores, se evita

así que una gran compañía los custodie, lo que elimina burocracia, censuras y los grandes costes / tiempos implícitos de este proceso que, dicho sea de paso, hasta ahora es el cotidiano.

Si juntamos los principios de un *smart contract* con la creatividad de muchos desarrolladores del planeta, el resultado son posibilidades jamás vistas, accesibles para todos y a costes que rozan la gratuidad. Ecosistemas sin figuras autoritarias que someten a su voluntad a sus integrantes. Hablamos de un mundo más justo.

Imagina un coche Tesla auto-conducido, comprado en grupo, capaz de autogestionarse y alquilarse por sí solo pero sin una compañía tipo Uber detrás llevándose el 10%. [35]

3.6.2 Los primeros contratos inteligentes:

La primera vez que se tiene constancia de forma pública sobre los Smart Contracts es a través de Nick Szabo, jurista y criptógrafo Nick Szabo que mencionó públicamente el término en un documento en 1995. Dos años después, en 1997, desarrolló un documento mucho más detallado explicando los Smart Contracts.

Lamentablemente, pese a definir la teoría, era imposible hacerla realidad con la infraestructura tecnológica existente. Para que los contratos inteligentes se puedan ejecutar es necesario que existan las transacciones programables y un sistema financiero que las reconozca, digitalmente nativo.

Precisamente, lo que Nick definía como inexistente en 1995, en 2009 (casi 15 años después) se haría realidad con la aparición de Bitcoin y su tecnología, la cadena de bloques (*blockchain*).

Bitcoin tiene algunos *smart contracts* ya creados que se ejecutan por defecto y de

manera transparente al usuario. Cuando hablamos de contratos de distribución nos referimos a uno de los casos de uso de Bitcoin para formar acuerdos entre personas a través de la *blockchain*. Y es que Bitcoin, entre todas sus ventajas, permite añadir lógica al dinero, algo único de este tipo de dinero: es **dinero programable**. Esta lógica aplicada al dinero nos permite resolver problemas comunes que podemos encontrarnos en la actualidad, pero aumentando el nivel de confianza a lo largo de todo el proceso automatizado en el que se desarrolla la interacción.

Ejemplificando, podrían desarrollarse nuevos productos o aplicaciones como por ejemplo:

- Mercados distribuidos que permitieran implementar contratos P2P y *trading* en los mercados con Bitcoin postulándose como un competidor completo al sistema financiero actual.
- Propiedades como automóviles, teléfonos, casas o elementos no físicos controlados a través de la cadena de bloques conforman las nombradas *smart property*. Mediante el uso de los contratos y con propiedades inteligentes se permite que el nivel de confianza sea muy superior reduciendo el fraude, los honorarios de mediación para terceras partes y permite que las operaciones se lleven a un nuevo nivel.
- Automatización de herencias estableciendo la asignación de los activos tras el fallecimiento. En cuanto llegase el fallecimiento, el contrato entraría en vigor y se ejecutaría repartiendo en este caso los fondos a la dirección establecida en el contrato.
- Seguros: Partes de accidente, pagos de la compañía para reparaciones, reducción del fraude en accidentes, etc. [36]

3.6.3 Un Smart Contract no es lo que se piensa:

Hoy en día todo está controlado por sistemas informáticos. Todo interactúa con ellos.

En el desarrollo de aplicaciones es normal que los programadores creen una serie de “puertas” a su aplicación (llamadas APIs) con las que otros programadores pueden entrar a tu aplicación para crear o obtener información. Casi todas las web o programas tienen las suyas. Es decir, se define un protocolo, un contrato, una forma conocida en la que se llama a la aplicación con una estructura de datos, por la cual vamos a obtener una respuesta, pero con la estructura de datos predecible. En este caso para que no falle la comunicación, y consigo los programas.

Pero este contrato no está garantizado. El servidor de la aplicación está controlado por alguien que tiene la capacidad de hacer que mañana el programa funcione diferente. Está centralizado y puede mutar a la decisión de ese tercero. No es “Smart”.

La gente necesita entornos predecibles, transparentes e incorruptibles.

Los smart contracts son pedazos de códigos similares, es decir tienen formas de llamarlos y obtener unas respuestas, tienen un contrato, pero además son inmutables pues están distribuidos en miles de nodos que no pueden alterar su contenido. De esa forma obtienes un programa que siempre va a actuar de la misma forma sin requerir de la buena voluntad de ese tercero. Algo que para casi cualquier caso de uso es necesario.

Los Smart Contracts son programas en la nube que siempre actúan igual, y permiten almacenar información que no puede ser modificada a traición.

Son los programas más seguros jamás creados en la humanidad y solo fallan cuando están mal programados. [\[37\]](#)

3.6.4 Los Smart Contracts mal programados:

Los Smart Contracts son capaces de gestionar activos digitales, sujetos a un determinado valor económico, por lo que en realidad los Smart Contracts pueden gestionar dinero. Esto requiere que se haga especial énfasis en la correcta

programación del mismo, pues el Smart Contract podría tener fallos de seguridad o fallos que generasen errores de ejecución o comportamientos inesperados.

Cuando ocurre esto el dinero puede perderse por completo. No ha ocurrido pocas veces, y sin el Know How suficiente les seguirá ocurriendo a muchas iniciativas.

Lamentablemente esto ocurre con frecuencia, por lo que es realmente importante prestar especial atención en el desarrollo y testeo de estas piezas de software. [38]

3.6.5 Smart contracts en Ethereum

Ethereum lleva los *smart contracts* de Bitcoin a otro nivel y es quien, principalmente, ha inspirado a otras soluciones, como Counter Party o Rootstock, que quieren conseguir lo mismo pero sobre la red de Bitcoin.

Ethereum, que es uno de los proyectos más famosos en el sector de los *smart contracts*, es una plataforma de computación distribuida basada en una *blockchain* pública como Bitcoin y que además permite ejecutar contratos inteligentes P2P (entre los nodos, sin servidores centrales) en una máquina virtual descentralizada llamada Ethereum Virtual Machine (EVM).

Se basa en toda la teoría de Bitcoin en cuanto a estar distribuido, tener su propia criptomoneda, mineros e incluso su propio *blockchain* entre otras cosas pero, a diferencia de Bitcoin, Ethereum ha creado un intérprete de lenguaje de programación mucho más extenso (Turing completo), permitiendo añadir lógica mucho más compleja dentro del *blockchain*. Es decir, se podría asemejar a un ordenador distribuido, el cual utiliza su criptomoneda (el ether) como la "gasolina" que necesitan el contrato para que los mineros puedan ejecutarlo. Es decir, ahora con Ethereum los contratos son programas con muchas más funcionalidades y posibilidades.

Todas las aplicaciones funcionan en una *blockchain* con una potencia de cómputo muy alta que permite a los desarrolladores crear aplicaciones descentralizadas (DAPPS): Organizaciones Autónomas Descentralizadas (DAO), Mercados de intercambio,...

La implementación de Ethereum en el mundo real ha dado lugar a conceptos como las denominadas DApps, los tokens para Ofertas Iniciales de Criptomonedas (ICO) y la creación de Organizaciones Autónomas Descentralizadas (DAO).

El principal cliente es **Go-Ethereum**, el cual está mantenido y desarrollado por la Fundación Ethereum, una organización sin ánimo de lucro que apoya la plataforma y el desarrollo de la misma acorde a unos estándares preestablecidos.

Un cliente es el medio para que los usuarios interactúen con la red Ethereum en diferentes entornos operativos. [39]

3.6.6 Emisión de monedas en Ethereum

Su funcionamiento es algo diferente a otras criptomonedas:

- Se presentó en una conferencia de Bitcoin en Miami en enero de 2014.
- Se lanzó una preventa en julio de ese mismo año para conseguir capitalización para la Fundación Ethereum, recaudando 18 millones de dólares.
- Se estableció un suministro de ETH de 72 millones, que se repartieron en 60 millones para los suscriptores y 12 millones para la Fundación Ethereum.

Difiere en otras monedas en que el total de Ethers (ETH) posible tiene un mecanismo diferente, para controlar la inflación o lo que es lo mismo, el aumento de valor de su token. Ethereum limita el número de monedas que se pueden crear cada año a 18 millones, que es el 25% del suministro inicial.

Una de las diferencias más importantes con Bitcoin es que Ethereum no tiene un límite de Ether (ETH) la criptomoneda de la plataforma, aunque se limita a un máximo de 18 millones de monedas creadas en un año. Otra de las grandes diferencias es que las comisiones de las transacciones se pagan con Gas, un token que tiene la misión de hacer más resistente a la red. El Gas también se usa como pago para subir un Smart Contract a la Ethereum Virtual Machine (EVM) y para la creación de las DAO. [40]

	ETHEREUM	BITCOIN
Nº Total de Monedas	No tiene una cantidad fija	21 millones
Algoritmo	Ethash (Proof-of-work)	SHA-256 (Proof-of-work)
Tiempo de Emisión de Bloques	12-20 segundos	10 minutos
Tamaño de bloques	menos de 1 MB	2 MB

3.6.7 El GAS dentro de la plataforma

Debemos tener claro que ETH es la moneda nativa de la plataforma, la cual se utiliza para dar 'combustible' a los Smart Contracts y DApps de la red.

Adicionalmente a las transacciones de ETH entre dos usuarios, cada transacción tiene dos propiedades, que juntas se definen como la tarifa de transacción en ETH para que se pueda realizar la transacción.

- **Startgas:** Se le denomina también como 'límite de gas' y define la cantidad máxima de pasos de computación para que la transacción se pueda realizar.
- **Gasprice:** define el precio en ETH para el cálculo de la transacción.
- **Comisión total de la transacción:** inferior al (número de operaciones de computación * Gasprice) o bien (Startgas * Gasprice)

- **Explicación:** Las transacciones, incluido el código del contrato dentro de ellas, pueden continuar ejecutándose hasta que lleguen al destino, completando todos los pasos de ejecución del código o se agoten, que es cuando se llegue al límite de computaciones establecido.

Explicado de manera sencilla, lo que se busca es que la comisión de la transacción sea proporcional a la carga que añade la transacción a la red. Esto incentiva que los usuarios legítimos creen transacciones bien formadas y permite disuadir a los usuarios maliciosos. Esto lo hace más resistente a ataques de denegación de servicio o DoS. [41]

3.6.8 Usos actuales de Ethereum

- **Sistema de pago:** Ether se puede usar como cualquier otra criptomoneda para realizar pagos. Estos son validados por nodos que participan en la red y se registran las transacciones en la blockchain.
- **Smart Contracts:** Los contratos inteligentes son sencillamente programas informáticos complejos que agilizan el intercambio de dinero, contenido, propiedad o cualquier cosa de valor. El programa, se introduce en la blockchain y se ejecuta automáticamente cuando se dan las condiciones establecidas en este.
- **DApps:** Sobre una smart contract se puede construir una interfaz amigable para el usuario final, esto sería una DApp, un smart contract con una interfaz que permite realizar acciones de forma simple para alguien que no es un desarrollador. Con estos dos elementos tendríamos una aplicación descentralizada. Son como aplicaciones comunes, pero no están controlados por una sola persona u organismo, sino por los propios usuarios. Entre sus características más interesantes, está el eliminar intermediarios, el incremento de la seguridad y la transparencia.
- **DAO:** Ethereum permite crear las Organizaciones Autónomas Descentralizadas. Serían una especie de empresa sin líder, autónoma y descentralizada. Las reglas se fijan al inicio y se ejecutan mediante código,

mediante una correlación de Smart Contracts dentro de la blockchain de Ethereum. [\[42\]](#)

3.6.9 Desplegando Smart contracts en la red de Ethereum

Los contratos son scripts que definen procesos desde simples intercambios de criptomoneda como los que hace Bitcoin hasta políticas complejas que se desencadenan como consecuencia de eventos. Los contratos trabajan sobre tokens, que determinan quiénes participan en un contrato y con cuánto valor. Tanto los participantes del contrato como el propio contrato están identificados mediante una cuenta con una dirección única. Las cuentas de participantes se llaman Externally Owned Accounts (EOA).

Además de identificar a cada uno de los elementos de la red Ethereum, las cuentas sirven de almacenamiento persistente de información.

Un contrato normal especifica un acuerdo entre partes con unas acciones y condiciones de cumplimiento. Un contrato se hace “inteligente” cuando incluye código y lógica que ejecuta las acciones y fuerza la aplicación de las condiciones, es decir, código que hace que se cumpla el contrato sin necesidad de que ninguna de las partes intervenga manualmente. Lo más interesante más allá de esa supuesta “inteligencia” es que el contrato se ejecuta dentro de la red, en la Ethereum Virtual Machine (EVM).

Esta ejecución no sale gratis sino que se paga en forma de “gas”, una pequeña cantidad de moneda ether aportada por uno o más de los participantes en el contrato. Un detalle importante es que la ejecución es descentralizada pero no distribuida, lo cual significa que se produce en todos los nodos de la red. Esta es una de las razones por las que la EVM es tan lenta y solamente puede procesar unas 15 transacciones por segundo. El contrato se desarrolla en un lenguaje concreto (Solidity, Serpent o Mutan), se compila a EVM y, salvo excepciones, cumple una interfaz estándar llamada ERC20. [\[43\]](#)

3.6.10 Solidity - El lenguaje de los smart contracts de Ethereum

En Ethereum, los smart contract son programados en Solidity, un lenguaje específicamente creado para tal fin. Digamos que es un batiburrillo de algunas convenciones ya usadas por programadores web y de aplicaciones de red. La sintaxis del código es similar a JavaScript y C.

Permite compilar dichos contratos en el *bytecode* utilizado por Ethereum que será desplegado en las máquinas virtuales que representa la combinación de nodos. Es decir, Ethereum Virtual Machine (EVM), las responsables de ejecutar ese código, pequeñas computadoras distribuidas de 256-bits.

Se pueden programar esos smart contract localmente y después desplegarlo para que sean propagados alrededor de toda la red siendo alojados de forma descentralizada. En ese sentido, Ethereum combina las facetas de networking, app hosting y base de datos para almacenar estados de dichos contratos.

Cuando se tienen esos smart contracts desplegados se puede crear una serie de aplicaciones entorno a ellos. Ese es el papel de las *distributed application*, conocidas como DApp. [44]

3.7 Resumen del capítulo

En este capítulo se habló del primero de los dos vértices de esta tesina, blockchain, introduciendo al lector con un ejemplo de la vida real, descripción de las principales implementaciones de esta tecnología, contratos inteligentes y la anatomía de Ethereum junto con su lenguaje de programación Solidity.

Capítulo 4

Instrumentos musicales

4.1 Conceptos generales

Tanto la guitarra como el bajo son instrumentos de cuerda que por su naturaleza y apariencia son muy similares, pero no son lo mismo. (Por simplicidad sólo diferenciaremos los instrumentos en sus formatos Eléctricos). El número de serie que identifica al instrumento y con el cual se trabajara en el desarrollo del software se encuentra en el clavijero. Algunas diferencias:

- **Mástil:** El mástil, la pieza longitudinal que soporta el diapasón y las cuerdas, es más largo en el bajo que en la guitarra.
- **Cuerdas:** Normalmente el bajo tiene 4 cuerdas, mientras que la guitarra tiene normalmente 6. No siempre es así, pero la mayoría de las veces sí. Las cuerdas del bajo son además más gruesas que las de la guitarra.
- **Sonido:** El bajo reproduce las frecuencias bajas y la guitarra las medias / agudas.
- **Afinación:** es en ambos casos la afinación por lo general es estándar - La 440 Hercios , es decir 440 ciclos por segundo, (Mi - La - Re - Sol, con bajos de 4 cuerdas, Mi - La - Re - Sol - Si - Mi, guitarra), ambos tienen técnicas de ejecución tanto con púa como con dedos. [\[45\]](#)

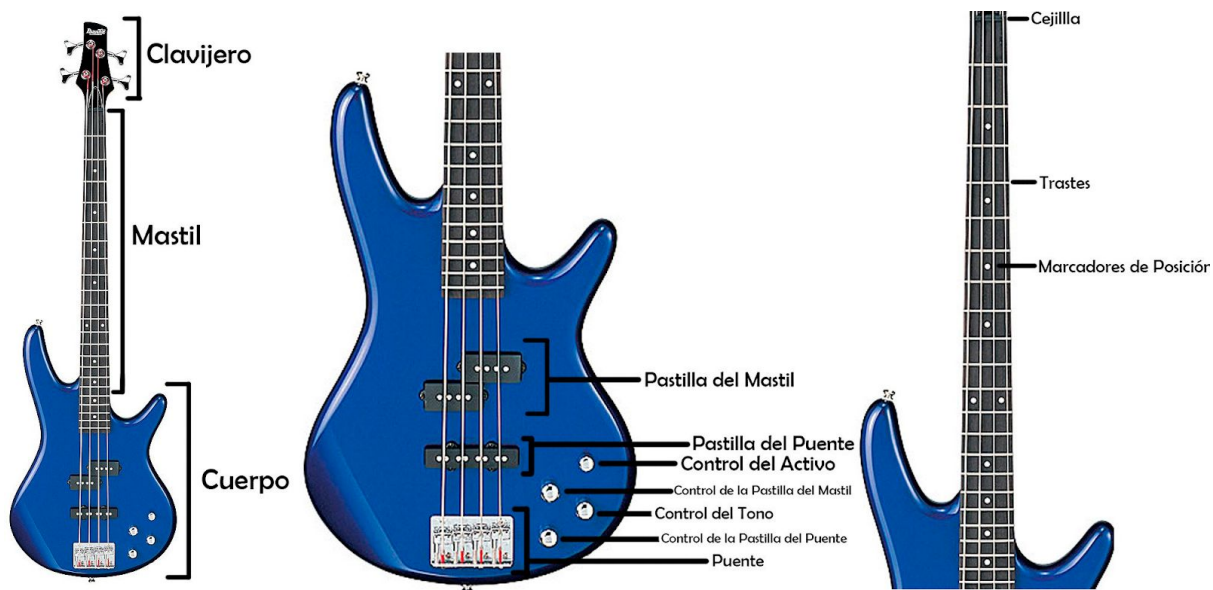


Figura 9 - Bajo eléctrico- partes [46]

GUITARRA ELÉCTRICA

- 1 - Clavijero (o pala)
- 2 - Diapasón
- 3 - Clavija de afinación
- 4 - Cejilla (o cejuela)
- 5 - Traste
- 6 - Marcador de posición
- 7 - Alma (o tensor de mástil)
- 8 - Cuello (o tacón)
- 9 - Cutaway
- 10 - Selector de Pastillas
- 11 - Pastilla
- 12 - Cintura
- 13 - Golpeador (o protector)
- 14 - Puente
- 15 - Cordal
- 16 - Potenciómetro
- 17 - Salida de jack
- 18 - Pin sujetacorrea

TOOLSONThEROAD.COM



Figura 10 - Guitarra eléctrica - partes [47]

4.2 Mercado de Guitarras y Bajos en Argentina

En la Argentina existe la Cámara de Instrumentos Musicales, Audio, Video e Iluminación (CAIMAVI) que agrupa a cuarenta representantes o distribuidores y La Cámara Argentina de Fabricantes de Instrumentos Musicales (CAFIM) reúne a 200 socios, desde luthiers a pequeñas empresas, ambos, concentran el 85 % del mercado. Actualmente existen altos aranceles para la importación por lo que el acceso a instrumentos de alta gama se limita en la mayoría de los casos por contrabando o bien por personas que tienen la posibilidad de viajar y comprar en el exterior, limitando el acceso a la cultura. Las **guitarras eléctricas y bajos eléctricos** pagan un **35%**.

Las fronteras del país son permeables, especialmente en la zona de **Paraguay**. Es conocido que **marcas internacionales** están teniendo **problemas** con sus distribuidores argentinos y paraguayos, porque el **distribuidor paraguayo** termina colocando productos en el **mercado argentino**, importando oficialmente en Paraguay con un **arancel** mucho **más bajo** y filtrándolo por las fronteras argentinas para abastecer nuestro mercado. [48]

4.3 Mercado negro de Instrumentos musicales en Argentina

Hoy en nuestro país los músicos que han sufrido robos de instrumentos utilizan las redes sociales para buscar sus objetos sustraídos, ya sea compartiendo una publicación, o esperando a que el instrumento sea ofrecido en las redes. En los casos de instrumentos seriados, se tiene la ventaja de que el instrumento puede identificarse, en los casos de los instrumentos no seriados (Luthiers), el reencuentro se limita a poder identificarlo directamente por parte del dueño real. La informalidad del sector facilita la circulación de instrumentos robados. En los pocos casos que se recuperan y las víctimas negocian por su cuenta sin intervención del estado.

En los otros casos los instrumentos son cambiados por estupefacientes o una cantidad ínfima de dinero. Por lo que luego depende de la voluntad de quien haya hecho la transacción, el devolverle el instrumento al propietario real.

Las denuncias a la policía en muchos casos no son realizadas, ya sea porque no se posee factura de compra (en los casos de instrumentos de segunda mano), como por la poca credibilidad tanto de la policía como del sistema jurídico.

Sindicatos, agrupaciones y músicos lanzan campañas para desestimar las compras de instrumentos sustraídos, pero aún así no alcanza. Se necesita algo más. [49]

4.4 Seguro de instrumentos musicales

En Argentina y en el mundo existen algunas empresas y bancos que aseguran instrumentos ante hurtos. La diferencia radica en el coste del mismo. En Argentina cuesta el doble o incluso más que en Europa o USA respecto a una cotización anual. Realizamos una comparativa con una conocida empresa de seguro de instrumentos del país, Musicshield: <http://www.musicshield.com.ar/> y Casablancas, <http://www.mcasablancas.com/>, seguro español.



The screenshot shows the Music Shield website interface. At the top left is the logo 'MS Music Shield Seguros Para Músicos'. At the top right, it says 'Propuesta de Seguro Hola Damian' with links for 'Enviar Mensaje', 'Volver a cotizar', and 'Salir'. The main content area is divided into two columns. The left column is titled 'Coberturas Cotizadas:' and lists '✓ Todo Riesgo Instrumentos Musicales' with a link 'Agregar Coberturas - Volver a cotizar'. Below this is 'Sumas Aseguradas:' with 'Todo Riesgo Instrumentos: usd 1500' and a link 'Modificar Sumas Aseguradas'. The right column is titled 'Costo Anual TR Instrumentos Musicales' and shows 'USD 96,33 o 4 pagos de USD 24,48' with a note 'Adicional Mundo Entero: USD 19,20 anuales'. At the bottom center is a button labeled 'Contratar Seguro'.

Figura 11 - Cotización Musicshield [50]

www.mcasablancas.com/tarificador-instrumentos

SEGUROS PARA INSTRUMENTOS MUSICALES, ARTE Y CULTURA CASTELLANO CATALÀ (+34) 93 727 81 09

cb CASABLANCAS Correduría de seguros Quiénes somos Seguros Algunos clientes **Tarificador** Luthieres Blog

SOLICITUD Y TARIFICADOR DE SEGUROS PARA INSTRUMENTOS MUSICALES (SOLO CLÁSICOS Y ACÚSTICOS, PARA ELÉCTRICOS ESCRIBIR A INFO@MCASABLANCAS.COM) 32.12€

1 Instrumentos

Valor: 1500 Instrumento: Pequeño

Uso: Individual

Área geográfica: Territorio nacional

Datos tomador para iniciar tramitación del seguro

damian

LLuch

castelli

12/10/1984

30937607

Zurara

Figura 12 - Cotización Casablanca [51]

4.5 Garantía de instrumentos musicales en Argentina

Las primeras marcas sólo ofrecen un sitio web donde registrar el instrumento a través de su número de serie para dar lugar al inicio de la garantía o solicitar una extensión de la misma.

Estas empresas a su vez no operan en la Argentina, por lo que el uso de estas garantías se reduce a la posibilidad del músico de poder trasladarse a sitios donde se operen o utilizar la garantía del importador.

4.6 Registro de instrumentos musicales

En la actualidad existen sitios en línea para denunciar / consultar instrumentos robados como **StolenGuitarRegistry** <http://www.stolenguitarregistry.com/> o **PlayChecked** <https://www.playchecked.com/>, servicio para el registro de instrumentos. Ambos servicios son gratuitos y ofrecen registro de instrumentos a través de su número de serie. Estos proyectos carecen de un marco legal y sólo constituyen un sistema cerrado para el músico ofreciéndole un servicio básico. Por otro lado, existe un servicio como ScreamingStone

<http://www.screamingstone.com/>, quienes utilizando técnicas de SEO ofrecen el servicio de recuperación de instrumentos robados.

4.7 Resumen del capítulo

En este capítulo se habló del segundo de los dos vértices de esta tesina, instrumentos musicales de cuerda, introduciendo al lector en conceptos generales de estos objetos, junto a su anatomía, diferencias entre bajos y guitarras, situación actual en la Argentina y un pantallazo sobre el código asociado en el nuevo código civil y comercial.

Capítulo 5

Metodología

5.1 Paso a Paso

Debido a que ninguno de nosotros ha trabajado profesionalmente con blockchain, la primer etapa consistió en investigar los distintos algoritmos de cifrado hasta que llegar a los algoritmos de curvas elípticas, que son los utilizados por la tecnología. Paralelamente practicamos el lenguaje Solidity a través de cryptozombies.io para ir ganando confianza.

A su vez estudiamos la situación de los instrumentos musicales en Argentina comparándola con otros países.

También realizamos reuniones preliminares con la directora del trabajo para luego continuar los informes de avance por email y la herramienta Google Docs compartiendo acceso al mismo.

Luego de estudiar los distintos algoritmos, nos centramos en estudiar las distintas alternativas que ofrece blockchain para elegir una con la que centrarnos, eligiendo Ethereum.

Paralelamente estudiamos las cuestiones del nuevo código civil y comercial que deben tenerse en cuenta para llevar que el desarrollo propuesto de aplicación tenga un marco legal.

Luego empezamos a realizar pequeños programas de ejemplos utilizando Web3.js conectado a un contrato desplegado en Remix, Metamask para desplegar contratos en la red Kovan. También ejecutamos los primeros contratos con el framework Truffle y el compilador de Solidity local, ejecutando los contratos de manera local con Ganache (blockchain local).

5.2 Stack Tecnológico

Nuestra aplicación final está dividida en un contrato y una interfaz web. La interfaz web es independiente y se ha desarrollado mediante tecnologías web del lado del cliente. El contrato está escrito en Solidity. La conexión entre ambas partes se realiza mediante el uso de una librería proporcionada por la propia plataforma de Ethereum.

- **Ethereum:** Utilizaremos la red de Ethereum para publicar contratos.
- **Plugin Metamask** para navegadores Google Chrome/Mozilla Firefox: Nos permitirá conectar nuestro navegador a la red Ethereum en sencillos pasos.
- **Remix IDE:** Se trata de un entorno de programación para *Solidity* totalmente online. Nos permitirá programar y ejecutar nuestro *Contrato Inteligente*, realizar llamados a la funciones internas del contrato, añadir y modificar información del mismo.
- **Red privada Kovan** para solicitar Kovan Ether (KETH). Publicar contratos en la red Ethereum conlleva un gasto en ETHER, por lo que antes de publicar algo, tenemos que fijarnos que todo esté muy bien programado, porque volver a subir un contrato nos podría salir caro... existen redes de prueba dentro de la red Ethereum como Kovan que nos permitirán ejecutar contratos inteligentes sin gastar ETHER real.
- **Saldo Kovan Ethereum de prueba:** A través del siguiente sitio, obtuvimos saldo para realizar nuestras pruebas <https://faucet.kovan.network/>
- **Tecnologías Web:** HTML5, Javascript nativo, Javascript Frameworks (Node JS a través de NVM, JQuery, Web3, NPM), CSS Frameworks (Bootstrap) y Solidity.
<https://github.com/nvm-sh/nvm>
<https://nodejs.org/>
<https://www.npmjs.com/>
- **Web3.js:** Es una API en Javascript compatible con Ethereum que implementa las especificaciones genéricas RPC en formato JSON. Para que la aplicación funcione en Ethereum se puede usar el objeto web3 proporcionado por la

biblioteca web3.js. La librería por dentro se comunica con un nodo local a través de llamadas RPC. Web3.js funciona con cualquier nodo Ethereum que expone una capa RPC.

Web3 contiene el objeto eth (web3.eth) para interacciones específicas de cadena de bloques de Ethereum y el objeto shh (web3.shh) para la interacción de Whisper. <https://github.com/ethereum/web3.js/>

- **Truffle:** es el framework más popular para el desarrollo en Ethereum actualmente, posee las siguientes características:
 - Compilación, enlace y despliegue de Smart contracts desde el propio framework
 - Depuración y testing automatizado de contratos. Truffle utiliza el framework de prueba Mocha.js y Chai.js para las afirmaciones proporcionando un marco sólido desde el cual poder escribir pruebas.
 - Framework con scripts de despliegue y migraciones en redes públicas y privadas
 - Acceso a cientos de paquetes externos y gestión con EthPM & NPM
 - Consola interactiva para comunicación directa con los contratos
 - Interacción con contratos mediante scripts externos.

<https://www.trufflesuite.com/>
 - **Mocha:** Mocha es un framework de pruebas de JavaScript que se ejecuta en Node.js. Nos da la posibilidad de crear tanto tests síncronos como asíncronos de una forma muy sencilla. Nos proporciona muchas utilidades para la ejecución y el reporte de los tests. <https://mochajs.org/>
 - **Chai:** Es un librería de aserciones, la cual se puede emparejar con cualquier framework de pruebas de Javascript. Chai tiene varias interfaces: assert, expect y should, que permiten al desarrollador elegir el estilo que le resulte más legible y cómodo a la hora de desarrollar sus tests.
- <https://www.chaijs.com/>

- **Openzeppelin:** Es una biblioteca para el desarrollo seguro de contratos inteligentes. Proporciona implementaciones de estándares como ERC20 y ERC721. <https://openzeppelin.org/>
- **EthPM:** Gestor de paquetes inmutable para consumir, distribuir y administrar de manera fácil y segura cualquier sistema de contrato inteligente. <https://www.ethpm.com/>
- **ERC721ExampleDeed:** Biblioteca basada en OpenZeppelin para la escritura de objetos como propiedad. <https://github.com/nastassiasachs/ERC721ExampleDeed>

5.3 Enfoque de software libre

Debido a que Ethereum es una plataforma para promover las aplicaciones distribuidas en Internet excluyendo a los servidores centralizados, pensamos que lo más lógico es que todo el software desarrollado y este documento sean libres.

El código fuente de la aplicación realizada en esta tesina es accesible a través de la plataforma Gitlab en el siguiente link: <https://gitlab.com/damianlluch/blockguitars>

.

5.4 Memoria

Ha sido realizada de forma colaborativa con Google docs, aportando cada uno su parte en la redacción y corrección del documento.

5.5 Escenarios

El desarrollo de aplicaciones basados en blockchain es una disciplina relativamente nueva por lo que, todavía no existen procesos estandarizados dentro de la ingeniería de software, por lo tanto para el desarrollo dentro de esta tesina se utilizó la propuesta realizada en el documento llamado "An Agile Software

Engineering Method to Design Blockchain Applications”[53], este consta de 8 etapas las cuales se irán describiendo a medida que vaya avanzando el desarrollo del prototipo.

5.5.1 Etapa 1 - Definir el objetivo del sistema.

Como se dijo anteriormente el objetivo del sistema es realizar operaciones de instrumentos musicales tales como alta, baja y transferencias de estos entre personas

5.5.2 Etapa 2 - Identificar los actores (humanos y sistemas externos)

- **Administrador:** Es el usuario que crea el sistema.
- **Persona:** Este usuario puede dar de alta el instrumento, darlo de baja , listar todos los instrumentos y transferirlos a otras personas.

5.5.3 Etapa 3 - Definir historias de usuario

User stories: Escribir los requerimientos del sistema en base a historias de usuario

- **Creación de sistema:** El administrador crea el sistema que registra con su dirección de ethereum.
- **Listado de instrumentos:** El usuario puede ver los instrumentos que se encuentran registrados en el sistemas .
- **Alta de instrumento:** El usuario dará de alta un instrumento ingresando marca y código de serie.
- **Baja de instrumento :** El usuario podrá seleccionar un instrumento de su propiedad y darlo de baja.
- **Transferir instrumento:** El usuario podrá seleccionar un instrumento de su propiedad y transferirlo a la dirección que desee.

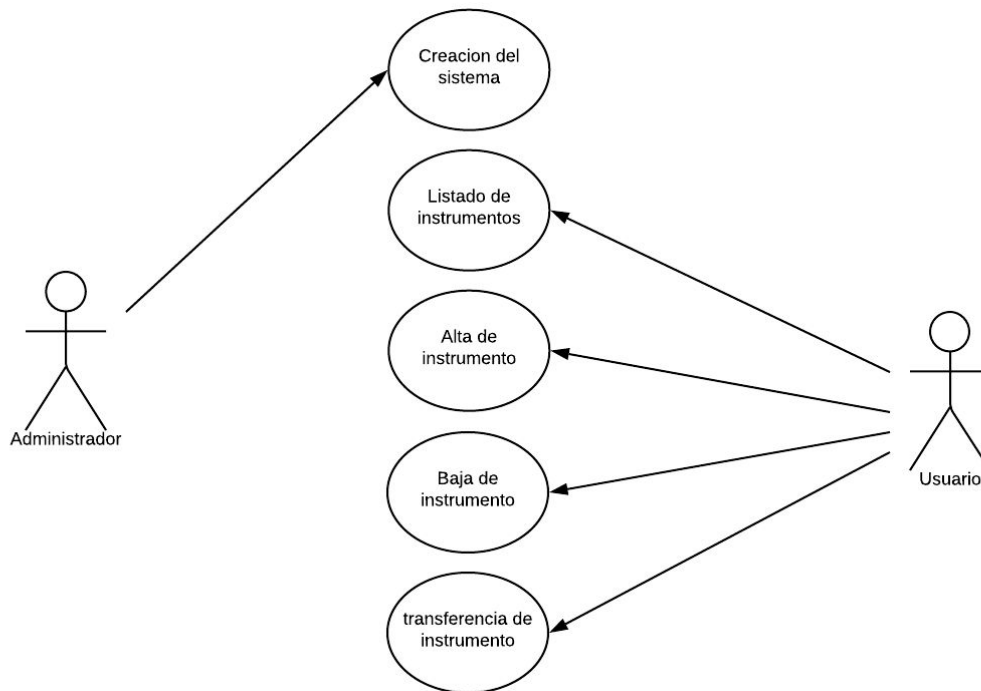


Figura 13- Diagrama de casos de uso

5.5.4 Etapa 4 - Dividir el sistema en dos subsistemas

Dividir el sistema en 2 subsistemas separados:

1. El sistema blockchain compuesto por smart contracts, este interactúa con el exterior exclusivamente a través de transacciones blockchain, sus actores son reconocidos a través de la dirección ethereum, puede utilizar librerías y contratos externos y puede generar transacciones a otros contratos. En este caso la subdivisión es trivial ya que todas las user stories hacen uso de los smart contracts.
2. El sistema externo que interactúa con el anterior enviando transacciones blockchain y recibiendo resultados a través de interfaces. Lo mismo ocurre para las user stories del subsistema externo, además agregan al blockchain como un actor adicional.

Los diagramas de clase UML se utilizan para representar Smart Contracts (SC) y estructuras. En Solidity, no existe el concepto de clase, pero los SC son muy similares a las clases. Al igual que una clase, un SC puede tener una estructura de datos, funciones públicas y privadas, y pueden heredar de una o más SCs. Sin embargo, los SC tienen una naturaleza específica; son creados por transacciones, pero una transacción puede crear como máximo un solo SC. Un SC sin embargo, puede enviar mensajes a otros SC, que residen en el mismo blockchain en Solidity, también es posible definir estructuras, es decir estructuras de datos complejas, que no cuentan con funciones.

En consecuencia, el modelo de un SC creado por una transacción puede incluir otros SC de los que hereda, las estructuras utilizadas y el SC externos a los que se envían mensajes.

Otros conceptos específicos de Ethereum SC son eventos, indicadores que son planteado cuando sucede algo relevante, y eso lo señala a la mundo externo (que tiene que observar de forma autónoma el SC, y actuar correspondientemente), y modificadores, funciones especiales que son llamado antes de una función, verificando sus restricciones y posiblemente deteniendo la ejecución. La siguiente tabla muestra los estereotipos para hacer posible representar conceptos SC en diagramas de clase UML. Los eventos podrían ser representados en otro compartimento, además de los que contienen el nombre, el atributo y las funciones (operaciones).

Estereotipo	Posición	Descripción
<<contract>>	Símbolo de clase- sección superior	Denota un smart contract
<<library contract>>	Símbolo de clase- sección superior	Contrato obtenido de una librería estándar
<<struct>>	Símbolo de clase- sección superior.	Un struct almacena datos definido y utilizado en la estructura de datos en el contrato
<<enum>>	Símbolo de clase- sección superior	Contiene una lista de posibles valores
<<interface>>	Símbolo de clase- sección superior	Un contrato que solo tiene declaraciones de funciones
<<modifier>>	Símbolo de clase - sección inferior	Tipo particular de función definida en solidity
<<array>>	Rol de una asociación	La relación 1 a n es implementada usando un arreglo
<<map>>	Rol de una asociación	La relación 1 a n es implementada usando un mapeo
<<map[uint]>>	Rol de una asociación	La relación 1: n se implementa utilizando un mapeo de entero al valor

Figura 14 . Estereotipos para diagrama de clases

Diseño de diagrama de clases modificado.



Figura 15 - Diagrama de clases

5.5.5 Etapa 5- Diseño del subsistema smart contract

- **Definir las descomposición en smart contracts (uno o más).**

Se realizará un smart contract que heredara de las interfaces ERC721 y ERC721Metadata y del contrato ERC 721 Deed , también utilizara contratos de la librería OpenZepellin como pausable y SafeMath

- **Para cada smart contract definir la estructura, diagrama de estado en caso de ser necesario.**

El contrato poseerá una estructura para almacenar los instrumentos con los siguientes campos : name, serialNumber, manufacturer, created, deleted.

Se definirán **2 eventos**, uno para crear o otro para destruir un objeto:

event Creation(uint256 indexed id, bytes32 indexed serialNumber, bytes32 indexed manufacturer, address owner);

event Destruction(uint256 indexed id);

- Se definen **3 estructuras de datos**, una enlazar el propietario al instrumento (mapping), una para controlar que no haya elementos repetidos (mapping), y otra para tener todas las escrituras (array).

mapping (uint256 => Guitar) public **deeds**;

mapping (bytes32 => bool) private **deedNameExists**;

uint256[] private **deedIds**;

- Se definirán **test unitarios** para los contratos con Mocha.js y Chai.js

Para empezar, ¿Qué son las test unitarias?

Las pruebas unitarias son las encargadas de probar individualmente métodos, funciones y clases, ingresando datos con el fin de obtener un resultado esperado.

¿Qué es Mocha?

- Mocha es una herramienta / framework de pruebas para Node.js, que permite ejecutar pruebas de manera síncrona, asíncrona y ordenada tanto en el lado del servidor como del navegador.
- Este framework contiene dos elementos importantes que permiten que las pruebas sean sencillas de realizar y entender:
- El primer elemento es la **función “describe”** que es la encargada de definir la estructura y agrupar las pruebas unitarias; dentro de esta función estará el código asíncrono.
- El segundo elemento es la **función “it”**, la cual define las pruebas unitarias que se van a realizar y el código que contiene los llamados a clases o métodos que queremos probar. La función it() recibe dos parámetros: el nombre de la prueba y una función que espera un callback done() como respuesta; éste se enviará vacío si la respuesta es exitosa o con valor del error si ha ocurrido algo no esperado en el proceso. [54]
- Se definirán dos scripts extra, uno para cargar contenido de prueba en el smart contract y otro para visualizar dicho contenido.

<https://gitlab.com/damianlluch/blockquitar/tree/master/scripts>


```

module.exports = function(callback) {
  var Web3 = require('web3');
  var provider = new Web3.providers.HttpProvider("http://localhost:8545");
  var contract = require('truffle-contract');
  var GuitarBlock = contract(require('./build/contracts/GuitarBlock.json'));
  var web3 = new Web3(provider);
  /** ----- Setemos los propietarios ----- */
  // Setemos las cuentas de Ganache
  const _creator = web3.eth.accounts[0];
  const _owner1 = web3.eth.accounts[1];
  const _owner2 = web3.eth.accounts[2];
  // Setemos las marcas de los instrumentos
  const _manufacturer1 = "Fender";
  const _manufacturer2 = "Yamaha";
  const _manufacturer3 = "MusicMan";
  // Setemos los respectivos números de serie
  const _serialNumber1 = "SBC973528365";
  const _serialNumber2 = "M1024";
  const _serialNumber3 = "LTS7300927";
  const _price1 = 1.0;
  const _price2 = .001;
  const _price3 = .02;
  GuitarBlock.setProvider(provider); // Setemos al provider como localhost
  GuitarBlock.defaults({from: _creator, gas: 900000 });
  const populateDeeds = async () => {
    let deed = await GuitarBlock.deployed();
    let name = await deed.name();
    try {
      // These will appear when you click the "All Guitars" link or when the index.html first loads.
      await deed.create(_serialNumber1, _manufacturer1, _owner1);
      await deed.create(_serialNumber2, _manufacturer2, _owner2);
      await deed.create(_serialNumber3, _manufacturer3, _owner3);
      // Optionally add your PERSONAL Guitar(s) here, so the "My Guitars" link will work.
      await deed.create("My$3r1AlnuM53R", "MARCA INSTRUMENTO", "0xbf00309c721accdf1f44f60c601b99bf2863b338");
      await deed.create("My$3r1AlnuM53R", "MARCA INSTRUMENTO", "0xbf00309c721accdf1f44f60c601b99bf2863b338");
    } catch (error) {
      console.log(error.message);
    }
  }
  populateDeeds();
}

```

Figura 16 - Populate de instrumentos.

```

module.exports = function(callback) {

  var Web3 = require('web3');
  var provider = new Web3.providers.HttpProvider("http://localhost:8545");
  var contract = require('truffle-contract');
  var GuitarBlock = contract(require('../build/contracts/GuitarBlock.json'));
  var web3 = new Web3(provider);

  const _creator = web3.eth.accounts[0];

  GuitarBlock.setProvider(provider);
  GuitarBlock.defaults({from: _creator, gas: 900000 });

  const displayGuitars = async () => {
    let deed = await GuitarBlock.deployed();
    let deedIds = await deed.ids();
    const FIELD_NAME = 0
    const FIELD_SERIAL_NUMBER = 1
    const FIELD_MANUFACTURER = 2
    const FIELD_DATE_CREATED = 4
    const FIELD_DATE_DELETED = 5
    let guitarStructs = []
    for (let i = 0; i < deedIds.length; i++) {
      var deedId = deedIds[i];
      var guitarBlock = await deed.deeds(deedId);
      const guitar = {
        name: web3.toAscii(guitarBlock[FIELD_NAME]),
        serialNumber: web3.toAscii(guitarBlock[FIELD_SERIAL_NUMBER]),
        manufacturer: web3.toAscii(guitarBlock[FIELD_MANUFACTURER]),
        dateCreated: guitarBlock[FIELD_DATE_CREATED],
        dateDeleted: guitarBlock[FIELD_DATE_DELETED]
      }
      guitarStructs.push(guitar)
    }
    console.log('guitarStructs =', guitarStructs)
  }

  displayGuitars();
}

```

Figura 17 - Display de instrumentos

5.5.6 Etapa 6 : Diseño del subsistema externo

- **Redefinir los actores y las user stories**, agregando el nuevo actor (pasivo) representado por los smart contract. Por lo tanto los actores del subsistema serían: Administrador, usuario y el subsistema smart contract.

- **Decidir la arquitectura del sistema:** La arquitectura utilizada para el prototipo va a ser una aplicación web responsive a la cual accederán los usuarios para poder realizar las transacciones de los instrumentos.
- **Definir la interfaz de usuario de los módulos relevantes.**

A continuación se definirán los mockups de las diferentes pantallas del sistema

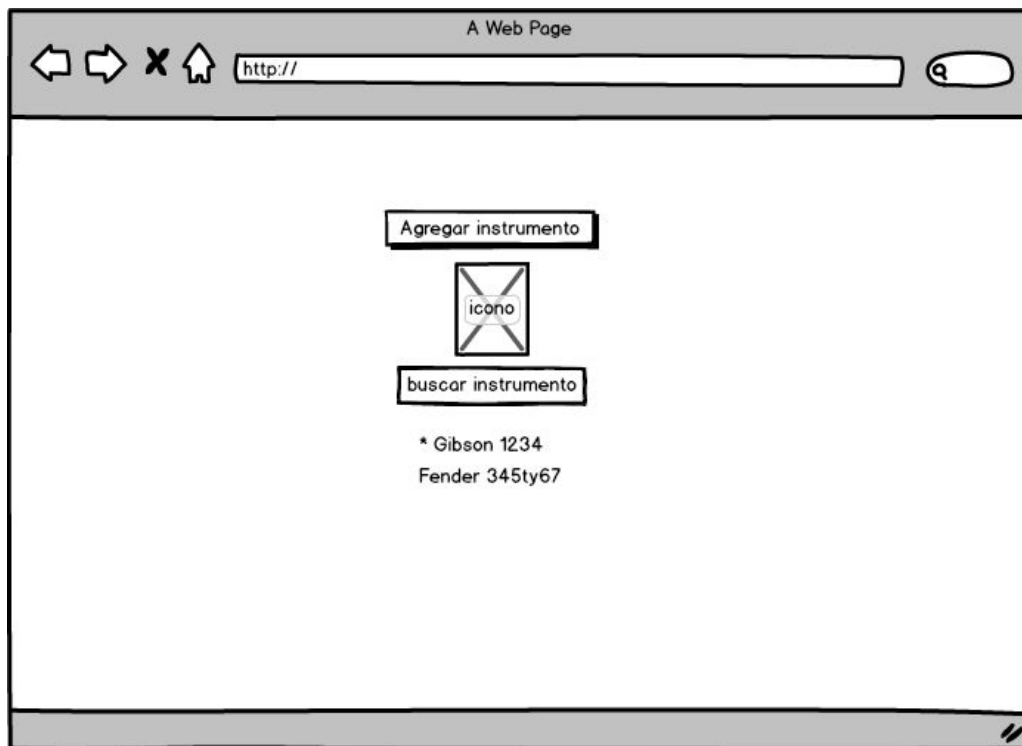


Figura 18- Listado de instrumento

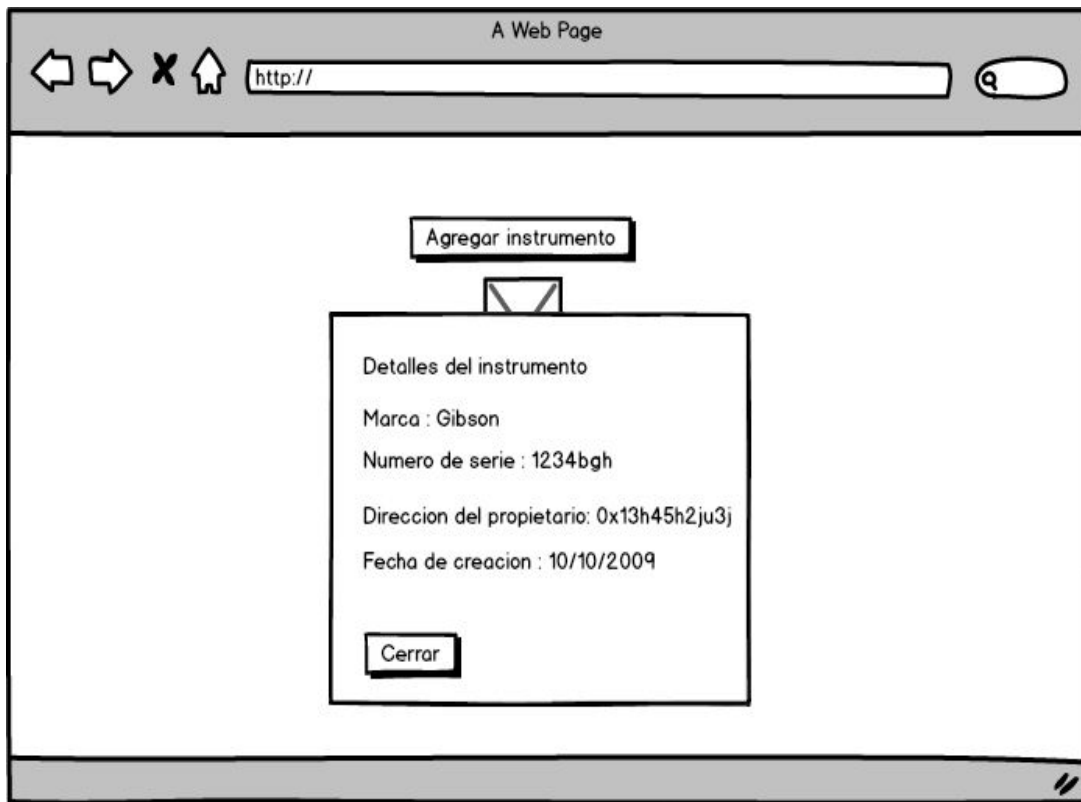


Figura 19 - Detalle de instrumento no propio

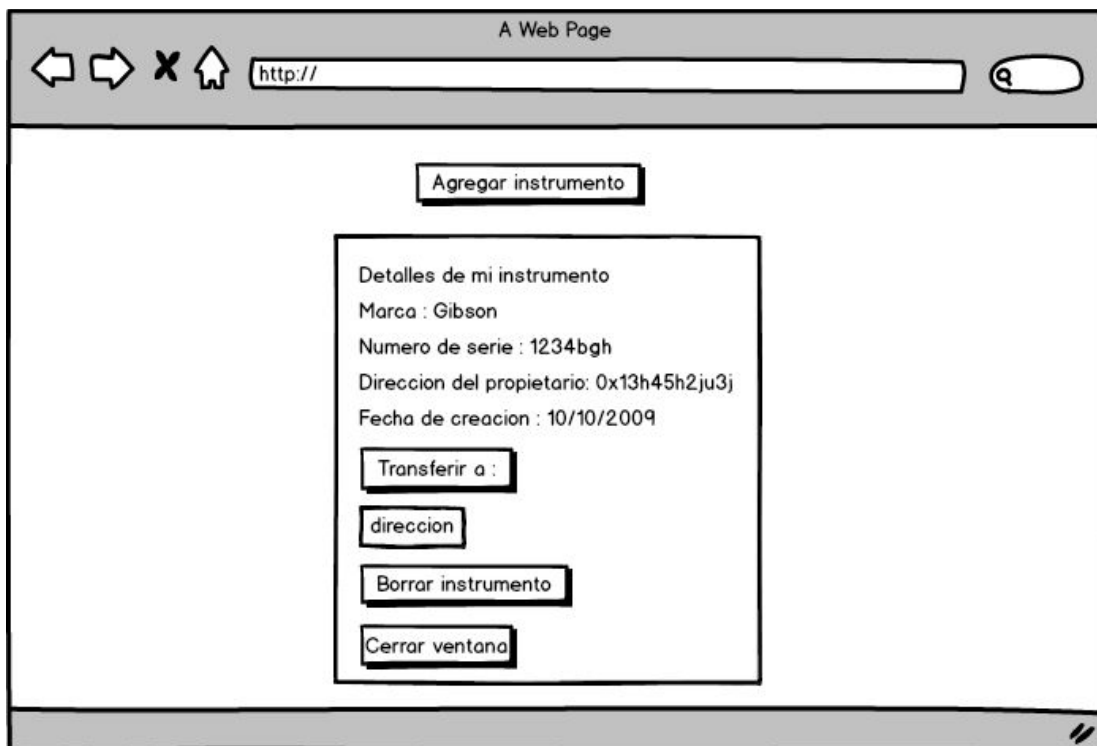


Figura 20- Detalle del instrumento propio y las operaciones permitidas

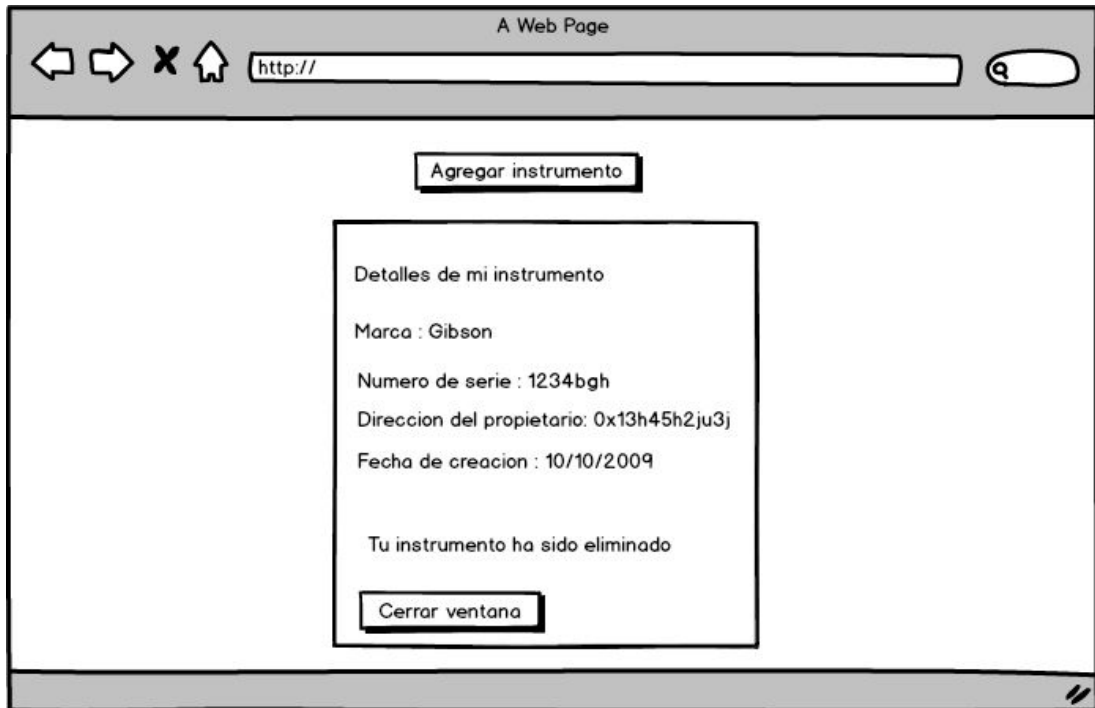


Figura 21 - Borrado de instrumento

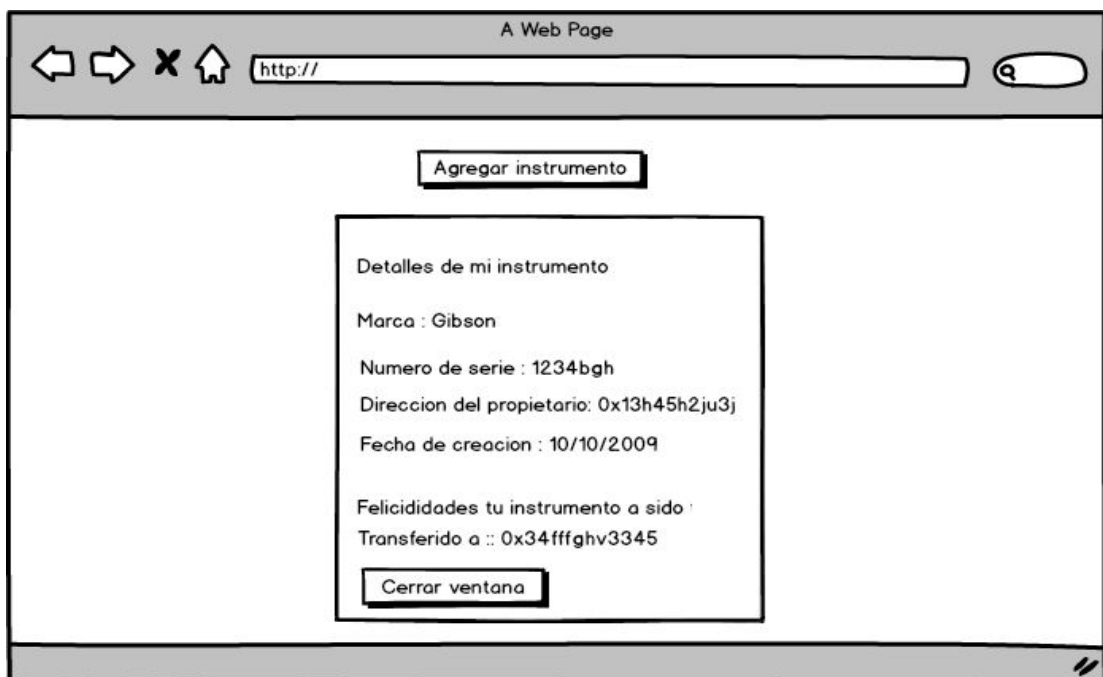


Figura 22 - Transferencia de instrumento

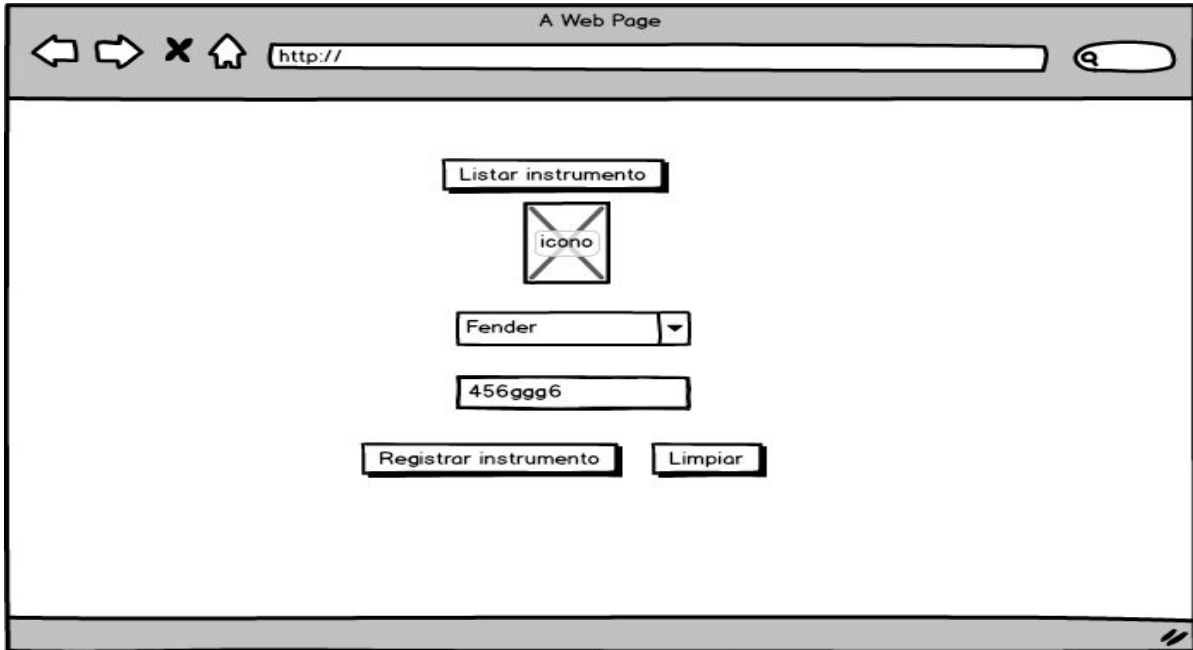


Figura 23 - Formulario de alta de instrumento.

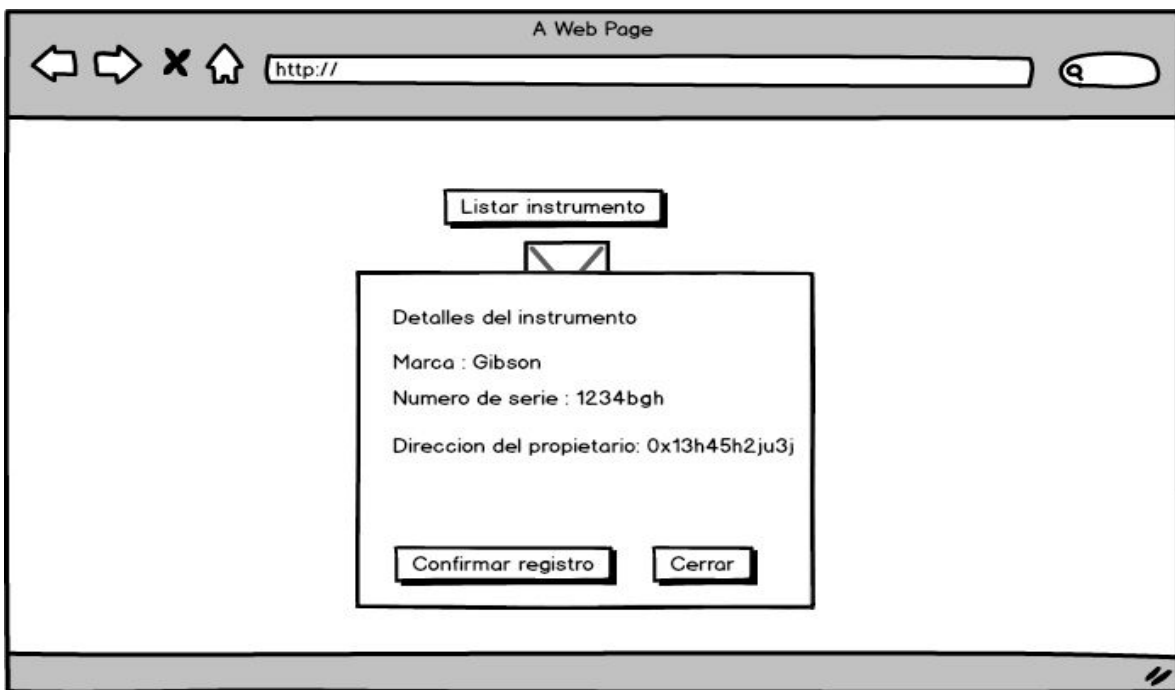


Figura 24 - Confirmación de alta de instrumento

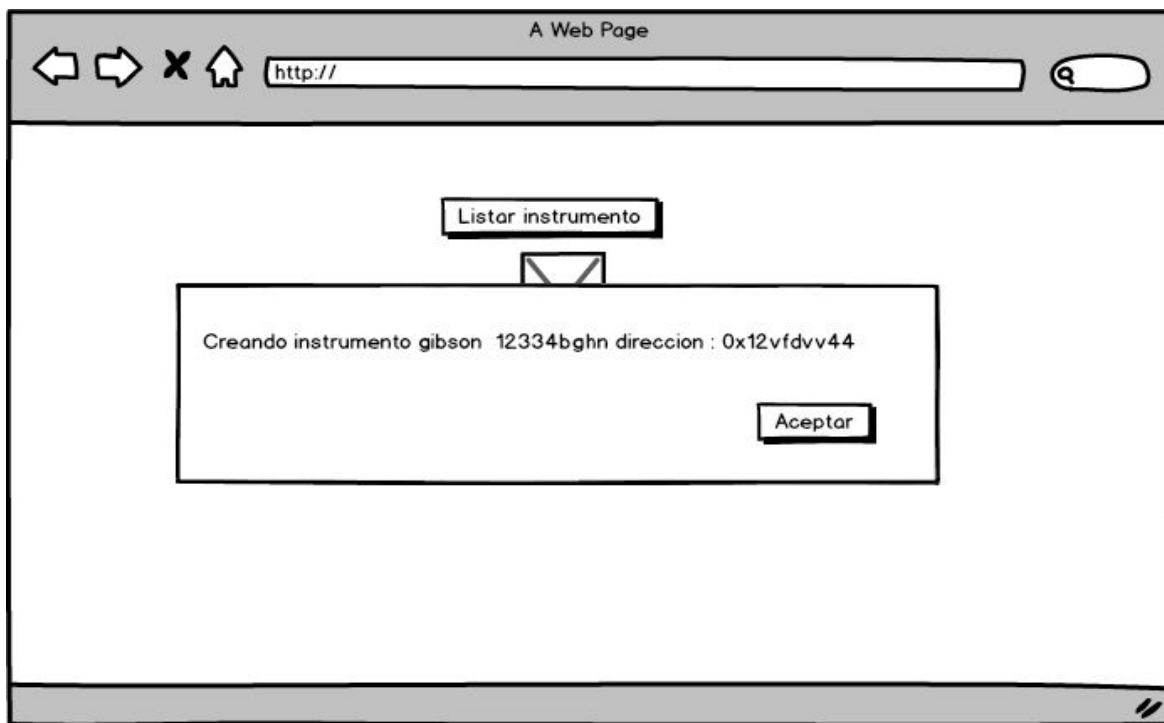


Figura 25 - Creación del instrumento

5.5.7 Etapa 7 - Codificar y testear los sistemas en paralelo

- **Escribir y testear los smart contract empezando por sus estructuras de datos y funciones**

A continuación se muestra el smart contract desarrollado mediante el framework truffle.

```

pragma solidity ^0.4.18;

import "zeppelin/contracts/math/SafeMath.sol";
import "zeppelin/contracts/lifecycle/Pausable.sol";
import "./ERC721Deed.sol";
import "./ERC721Metadata.sol";

/**
 * Notes on this ERC721 implementation:
 * Notas sobre esta implementación de ERC721:
 */
contract GuitarBlock is ERC721Deed, ERC721Metadata, Pausable {

    using SafeMath for uint256;

    /** Events */
    // When a deed is created by the contract owner.
    // Cuando el propietario del contrato crea una escritura.

    event Creation(uint256 indexed id, bytes32 indexed serialNumber, bytes32 indexed manufacturer, address owner);

    // When a deed needs to be removed. The contract owner needs to own the deed in order to be able to destroy it.
    // Cuando una escritura necesita ser eliminada. El propietario del contrato debe ser el propietario de la escritura para poder destruirla.

    event Destruction(uint256 indexed id);

    // The data structure of the guitar
    // La estructura de datos de la guitarra.

    struct Guitar {
        bytes32 name;
        bytes32 serialNumber;
        bytes32 manufacturer;
        uint256 created;
        uint256 deleted;
    }
}

```

Figura 26 -Smart contract - parte 1

```

uint256 => Guitar; public deeds;

g from deed name to boolean indicating if the name is already taken
ción de nombre a booleano que indica si el nombre ya existe

bytes32 => bool) private deedNameExists;

to make all deeds discoverable. The length of this array also serves as our deed ID.
rio para que todas las escrituras sean reconocibles. La longitud de esta matriz también sirve como nuestra identificación de escritura.

private deedIds;

ntract owner can change the base URL, in case it becomes necessary. It is needed for Metadata.
pietario del contrato puede cambiar la URL base, en caso de que sea necesario. Es necesario para los metadatos.

blic url = "http://ipfs.io/ipfs/";

5 Metadata
ternal constant INTERFACE_SIGNATURE_ERC165 = // 0x01ffc9a7
4(keccak256('supportsInterface(bytes4)'));

ternal constant INTERFACE_SIGNATURE_ERC721 = // 0xda671b9b
4(keccak256('ownerOf(uint256)')) ^
4(keccak256('countOfDeeds()')) ^
4(keccak256('countOfDeedsByOwner(address)')) ^
4(keccak256('deedOfOwnerByIndex(address,uint256)')) ^
4(keccak256('approve(address,uint256)')) ^
4(keccak256('takeOwnership(uint256)'));

ternal constant INTERFACE_SIGNATURE_ERC721Metadata = // 0x2a786f11
4(keccak256('name()')) ^
4(keccak256('symbol()')) ^
4(keccak256('deedUri(uint256)'));

GuitarBlock() public {}

```

Figura 27 -Smart contract - parte 2


```

modifier onlyExistingNames(uint256 _deedId) {
    require(deedNameExists[deeds[_deedId].name]);
    _;
}

modifier noExistingNames(bytes32 _serialNumber, bytes32 _manufacturer) {
    bytes32 _name = _buildName(_serialNumber, _manufacturer);
    require(!deedNameExists[_name]);
    _;
}

modifier notDeleted(uint256 _deedId) {
    require(deeds[_deedId].deleted == 0);
    _;
}

/* ERC721Metadata */

function name()
public pure returns (string) {
    return "GuitarBlock";
}

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

function supportsInterface(bytes4 _interfaceID)
external pure returns (bool) {
    return (
        _interfaceID == INTERFACE_SIGNATURE_ERC165
        || _interfaceID == INTERFACE_SIGNATURE_ERC721
        || _interfaceID == INTERFACE_SIGNATURE_ERC721Metadata
    );
}

```

Figura 28 -Smart contract - parte 3

```

function deedName(uint256 _deedId)
public view onlyExistingNames(_deedId) returns (string _name) {
    _name = _bytes32ToString(deeds[_deedId].name);
}

/* Enable listing of all deeds (alternative to ERC721Enumerable to avoid having to work with arrays). */
/* Habilita la lista de todas las escrituras (alternativa a ERC721Enumerable para evitar tener que trabajar con matrices). */
function ids()
external view returns (uint256[]) {
    return deedIds;
}

function deed(uint256 _deedId)
external view returns (Guitar) {
    return deeds[_deedId];
}

/* Owner Functions */
/* Funciones del propietario */
// Anyone creates deeds. Newly created deeds are initialised with a derived name, serialNumber, manufacturer, owner address.
// Cualquiera crea escrituras. Las mismas se inicializan con un nombre, número de serie, fabricante, dirección del propietario.
function create(bytes32 _serialNumber, bytes32 _manufacturer, address _owner)
public noExistingNames(_serialNumber, _manufacturer) {
    bytes32 _name = _buildName(_serialNumber, _manufacturer);
    deedNameExists[_name] = true;
    uint256 deedId = deedIds.length;
    deedIds.push(deedId);
    super.mint(_owner, deedId);
    deeds[deedId] = Guitar({
        name: _name,
        serialNumber: _serialNumber,
        manufacturer: _manufacturer,
        created: now,
        deleted: 0
    });
    Creation(deedId, _serialNumber, _manufacturer, _owner);
}

```

Figura 29 -Smart contract - parte 4

```

function destroy(uint256 _deedId)
public onlyOwnerOf(_deedId) notDeleted(_deedId) {
    // We deliberately let the name stay in use, so that each name remains a unique identifier forever.
    // Dejamos deliberadamente que el nombre permanezca en uso, para que cada nombre siga siendo un identificador único para siempre.
    // Iterating over an array of IDs is too expensive, so we mark the deed as deleted instead.
    // Iterar sobre una matriz de ID es demasiado costoso, por lo que marcamos la escritura como eliminada.
    deeds[_deedId].deleted = now;

    super._burn(_deedId);
    Destruction(_deedId);
}

function setUrl(string _url)
public onlyOwner {
    url = _url;
}

/* Private helper functions */
/* Funciones de ayuda privadas */
function _buildName(bytes32 _serialNumber, bytes32 _manufacturer)
private pure returns(bytes32) {
    return _stringToBytes32(_strConcat(_bytes32ToString(_serialNumber),
    _bytes32ToString(_manufacturer)));
}

function _bytes32ToString(bytes32 _bytes32)
private pure returns (string) {
    bytes memory bytesString = new bytes(32);
    uint charCount = 0;
    for (uint j = 0; j < 32; j++) {
        byte char = byte(bytes32(uint(_bytes32) * 2 ** (8 * j)));
        if (char != 0) {
            bytesString[charCount] = char;
            charCount++;
        }
    }
    bytes memory bytesStringTrimmed = new bytes(charCount);
    for (j = 0; j < charCount; j++) {
        bytesStringTrimmed[j] = bytesString[j];
    }
}

```

Figura 30 -Smart contract - parte 5

- **Implementar las user stories del subsistema externo.**
se implementó mediante el framework vue.js y la librería web3js que interactúa con el smart contract.

```

loadAllGuitars: function() {
  let self = this;
  this.allguitars.length=0;
  const loadGuitars = async () => {
    let deed = await GuitarBlock.at(this.contractAddress);
    let deedIds = await deed.ids();

    const FIELD_NAME = 0;
    const FIELD_SERIAL_NUMBER = 1;
    const FIELD_MANUFACTURER = 2;
    const FIELD_DATE_CREATED = 3;
    const FIELD_DATE_DELETED = 5;

    for (let i = 0; i < deedIds.length; i++) {
      var deedId = deedIds[i];
      var guitarBlock = await deed.deeds(deedId);
      try {
        var guitarOwner = await deed.ownerOf(deedId);
      } catch(error) {
        // probably a deleted token and therefore has no owner.
        continue;
      }
      var url = await deed.deedUri(deedId);
      const guitar = {
        id: deedId,
        name: web3.toAscii(guitarBlock[FIELD_NAME]),
        serialNumber: web3.toAscii(guitarBlock[FIELD_SERIAL_NUMBER]),
        manufacturer: web3.toAscii(guitarBlock[FIELD_MANUFACTURER]).replace(/u0000/g, ''),
        dateCreated: moment.unix(guitarBlock[FIELD_DATE_CREATED]).format('format: "DD/MM/YYYY HH:mm:ss"'),
        dateDeleted: guitarBlock[FIELD_DATE_DELETED],
        owner: guitarOwner,
        guitarUrl: url
      };
      if (web3.isAddress(guitarOwner)) {
        this.allguitars.push(guitar);
      }
    }
  }
}

```

Figura 31- Implementación del listado de instrumentos.

```

registerGuitar: function() {
  this.showDetailsModal = false;

  const registerGuitarOnBlockchain = async () => {
    var self = this;
    GuitarBlock.defaults({from: this.userAccount, gas: 900000 });
    let deed = await GuitarBlock.at(this.contractAddress);
    console.log(deed);
    this.processingMessage = "Registrando instrumento en la blockchain . Esto puede tardar un rato...";
    this.showSpinner = true;
    try {
      alert("creando instrumento con " + this.guitarSerialNumber + " " + this.guitarManufacturer + " " + this.userAccount);
      let result = await deed.create(this.guitarSerialNumber, this.guitarManufacturer, this.userAccount);
      console.log(result);
    } catch (error) {
      console.log(error.message);
      this.showSpinner = false;
      error.message = "ha ocurrido un error al registrar el instrumento";
      this.processingMessage = error.message;
      alert(error.message);
      return false;
    }
    this.processingMessage = "Felicitaciones! tu instrumento fue registrado en la blockchain.";
    this.showSpinner = false;
    this.clearRegistrationForm();
    return true;
  }

  this.initAccounts();

  if (!registerGuitarOnBlockchain()) {
    return;
  }
}

```

Figura 32 - Implementación del registro de instrumento

```

deleteGuitarDeed: function() {
  const destroyDeed = async () => {
    var self = this;
    GuitarBlock.defaults({from: this.userAccount, gas: 900000 });
    let deed = await GuitarBlock.at(this.contractAddress);
    this.processingMessage = "Eliminando Guitarra. Esto puede tardar un rato...";
    this.showSpinner=true;
    this.displayRegistrationComponents = false;
    this.sleep( milliseconds: 1000);
    try {
      let result = await deed.destroy(this.guitarId);
    } catch (error) {
      console.log(error.message);
      this.processingMessage = error.message;
      alert(error.message);
      this.showSpinner=false;
      this.displayRegistrationComponents = true;
      return;
    }
    this.processingMessage = "Tu guitarra ha sido eliminada!";
    this.guitarOwner = this.newOwnerAddress;
    this.showSpinner=false;
    this.displayRegistrationComponents = false;
    this.initAccounts();
    this.loadAllGuitars();
  }
}

this.initAccounts();

```

Figura 33 - Implementación borrado de instrumento

```

transferOwnership: function() {
  const transfer = async () => {
    var self = this;
    GuitarBlock.defaults({from: this.userAccount, gas: 900000 });
    let deed = await GuitarBlock.at(this.contractAddress);
    this.displayRegistrationComponents=false;
    this.processingMessage = "Transfiriendo propiedad del instrumento a " + this.newOwnerAddress + ". esto puede tardar un rato...";
    this.showSpinner = true;
    try {
      let result = await deed.transfer(this.newOwnerAddress, this.guitarId);
    } catch (error) {
      console.log(error.message);
      this.processingMessage = error.message;
      alert(error.message);
      this.displayRegistrationComponents=true;
      this.showSpinner = false;
      return true;
    }
    this.processingMessage = "Felicidades! tu instrumento ha sido transferido a " + this.newOwnerAddress + "!";
    this.showSpinner = false;
    this.guitarOwner = this.newOwnerAddress;
    return true;
  }
}

// Not sure why this has to be done.
this.initAccounts();

if (!web3.isAddress(this.newOwnerAddress)) {
  var errorMsg = "No es una direccion valida!";
  console.log(errorMsg);
  this.processingMessage = errorMsg;
  return true;
}

```

Figura 34 - Implementación transferencia de instrumento

```

showGuitarDetails:function(guitar) {
|
  this.initAccounts();
  this.guitarId = guitar.id;
  this.guitarOwner = guitar.owner;
  this.guitarSerialNumber = guitar.serialNumber;
  this.guitarManufacturer = guitar.manufacturer;
  this.guitarDateCreated = guitar.dateCreated;
  this.guitarUrl = guitar.guitarUrl;
  if (this.userAccount == guitar.owner) {
    this.showMyDetailsModal=true;
    this.displayRegistrationComponents=true;
    this.processingMessage = ""
  }
  else {
    this.showDetailsModal=true;
  }
},

```

Figura 35 - Implementación detalle de instrumento

5.5.8 Etapa 8 Integrar, testear y hacer deploy

- Para iniciar el sistema se necesita realizar los siguientes comandos:
 - npm install** : sirve para instalar las librerías necesarias especificadas en el proyecto
 - truffle compile** : Sirve para compilar los archivos fuentes del contrato.
 - ganache-cli -p 8545**: mediante este comando se inicia ganache en el servidor local en el puerto 8545, esto no proveerá una billetera con 10 direcciones y sus claves privadas.
 - truffle migrate** : Se utiliza para ejecutar las migraciones para implementar los contratos.

```
Deploying 'GuitarBlock'
-----
> transaction hash: 0xd0dd75d2b85abc6455fb71ea65249b6d5b31d885fe18dc3789e044c4fd5ccf29
> Blocks: 0 Seconds: 0
> contract address: 0xA513016de9e9D483Bb9b5669bfE81b827a6A97e8
> block number: 3
> block timestamp: 1571602193
> account: 0xF9Fd66f8308AA5D91503321e5790c83d744D9D27
> balance: 99.93229814
> gas used: 3073505
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.0614701 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.0614701 ETH
```

Figura 36 - Migración de contrato

Esto nos brindara la dirección del contrato, la que luego agregaremos en la variable `contractAddress` perteneciente al archivo `guitarappvue.js` de nuestro sistema para poder trabajar con el contrato creado.

npm run build :

npm run server:

- **Listado de instrumento:** En el inicio de la aplicación se podrá visualizar el listado de los instrumentos registrados, con los datos de código de serie y marca como así también un buscador y el enlace al alta de instrumento

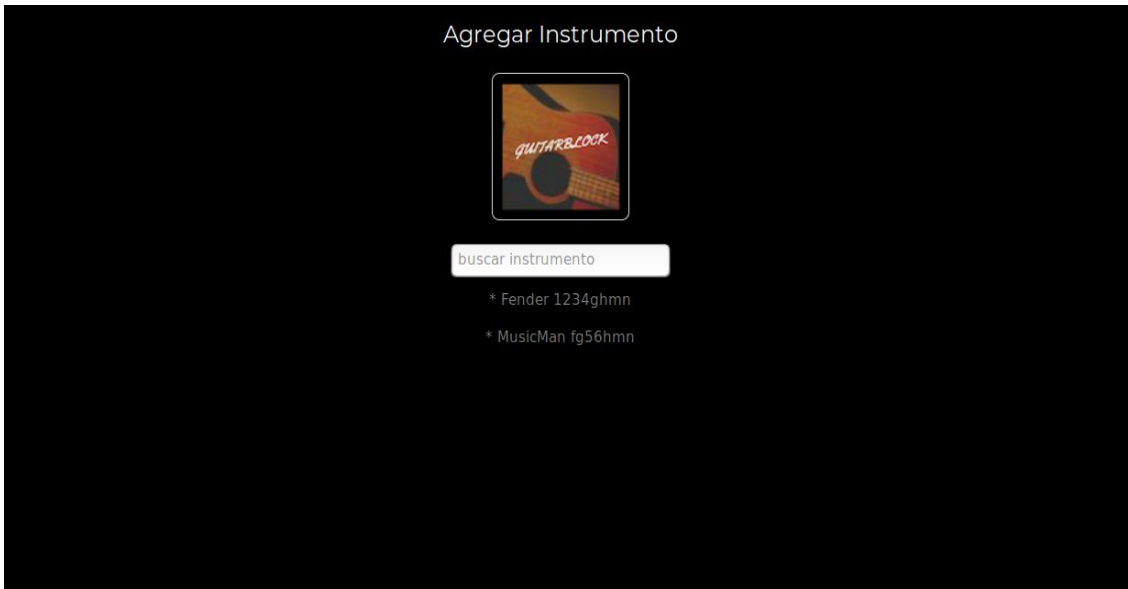


Figura 37 - Listado de instrumentos

Al hacer click en un instrumento del listado se abra un menu y dependiendo si el usuario es dueño o no de ese instrumento , se mostraran ciertas opciones de operaciones , en el caso de que no sea dueño se visualizará de la siguiente forma:

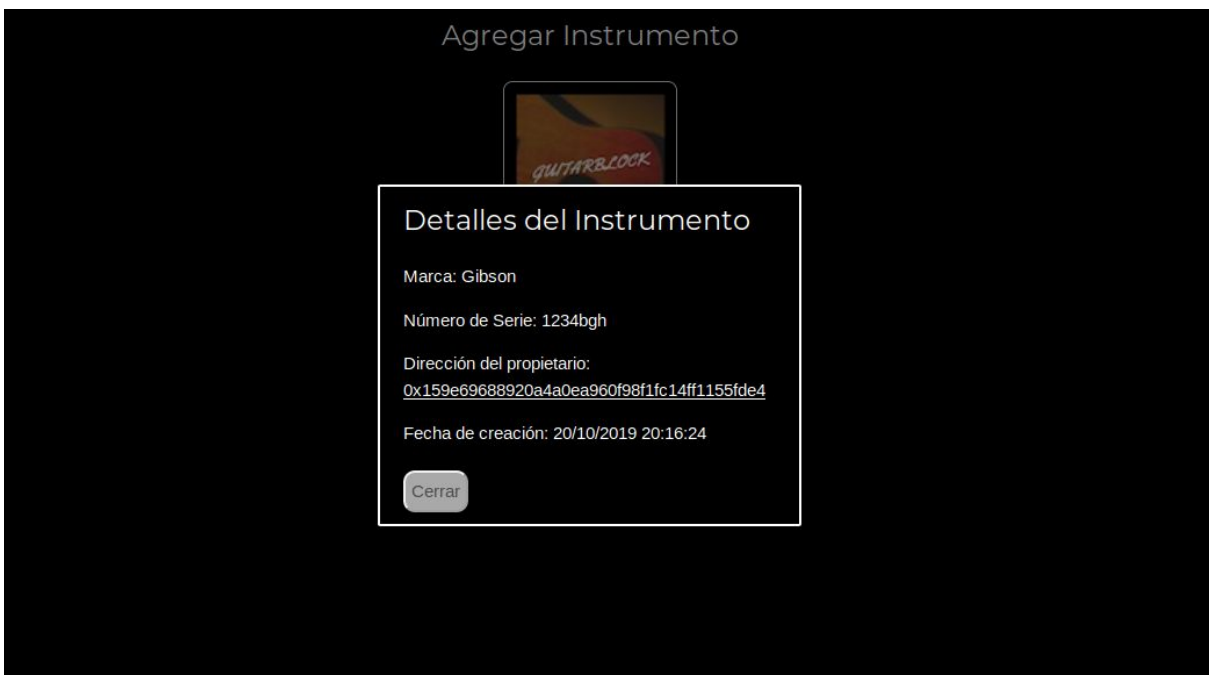


Figura 38 - Detalle de instrumento con diferente dirección

En cambio si el usuario posee ese instrumento se presentará de la siguiente manera:

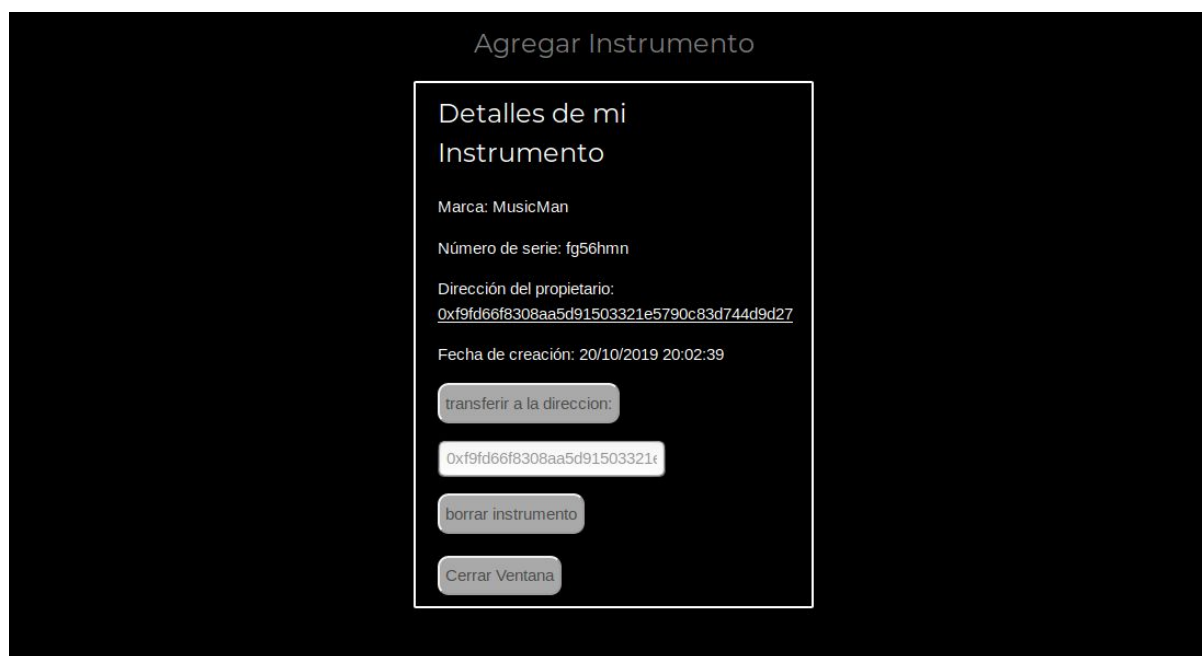


Figura 39 - Detalle de instrumento del usuario.

En este caso se permitirá realizar dos operaciones baja y transferencia las cuales se describirán a continuación:

- **Baja de instrumento:** seleccionando esta opción aparecerá pantalla confirmando la baja del instrumento

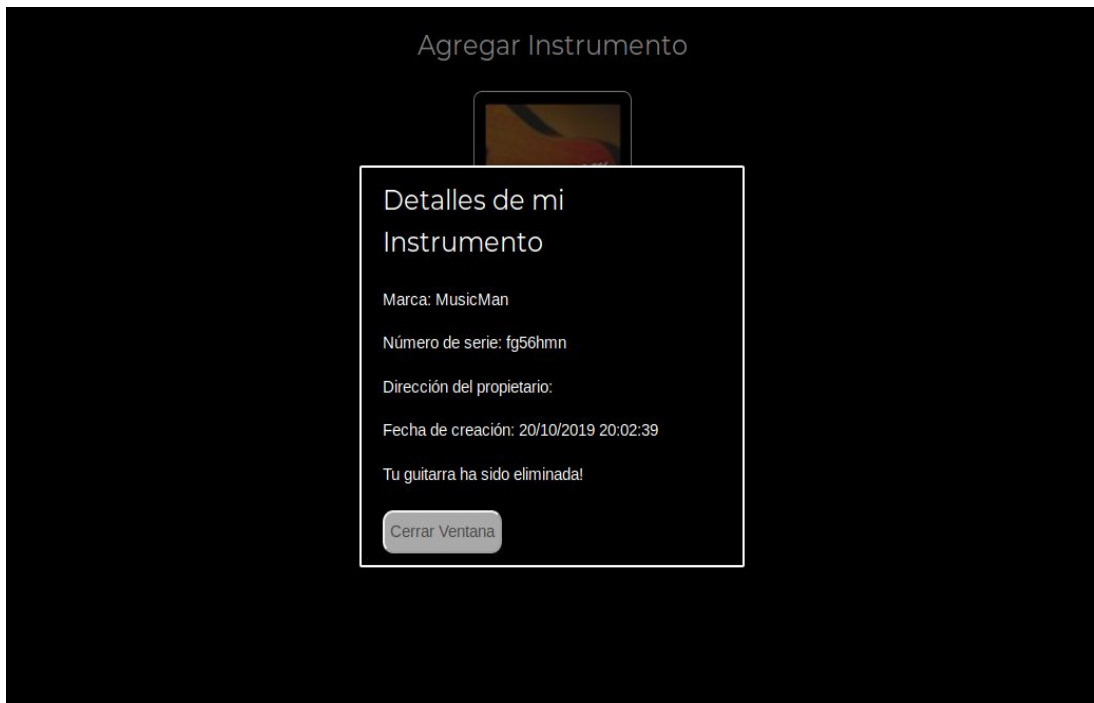


Figura 40 - Confirmación de baja

- **Transferencia de instrumento:** El usuario ingresa una dirección de destino y selecciona la opción "transferir a la dirección", el sistema mostrará la confirmación de la transferencia.

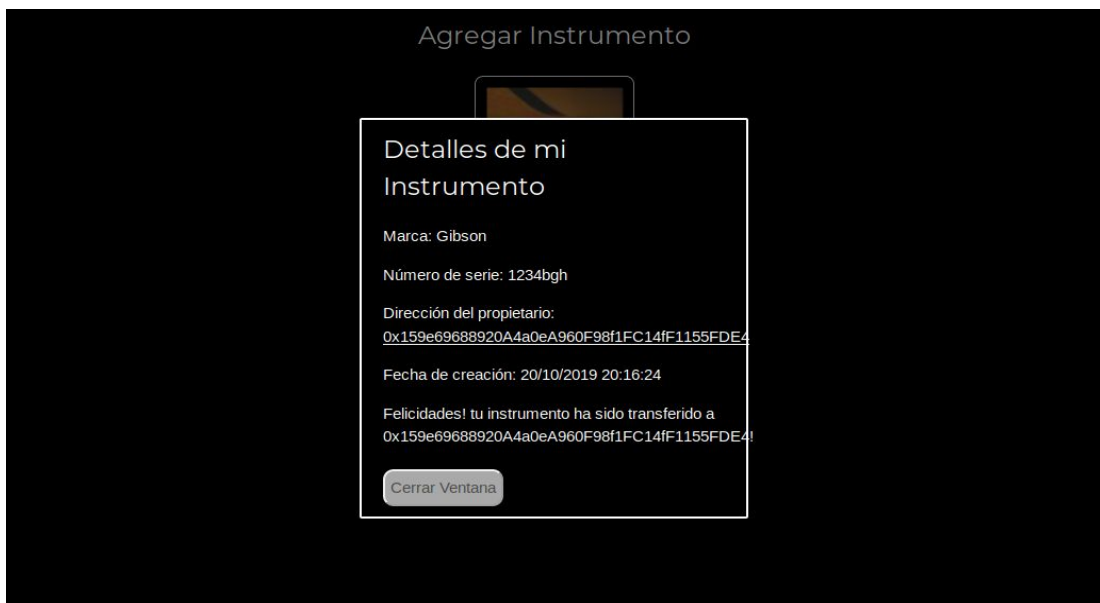
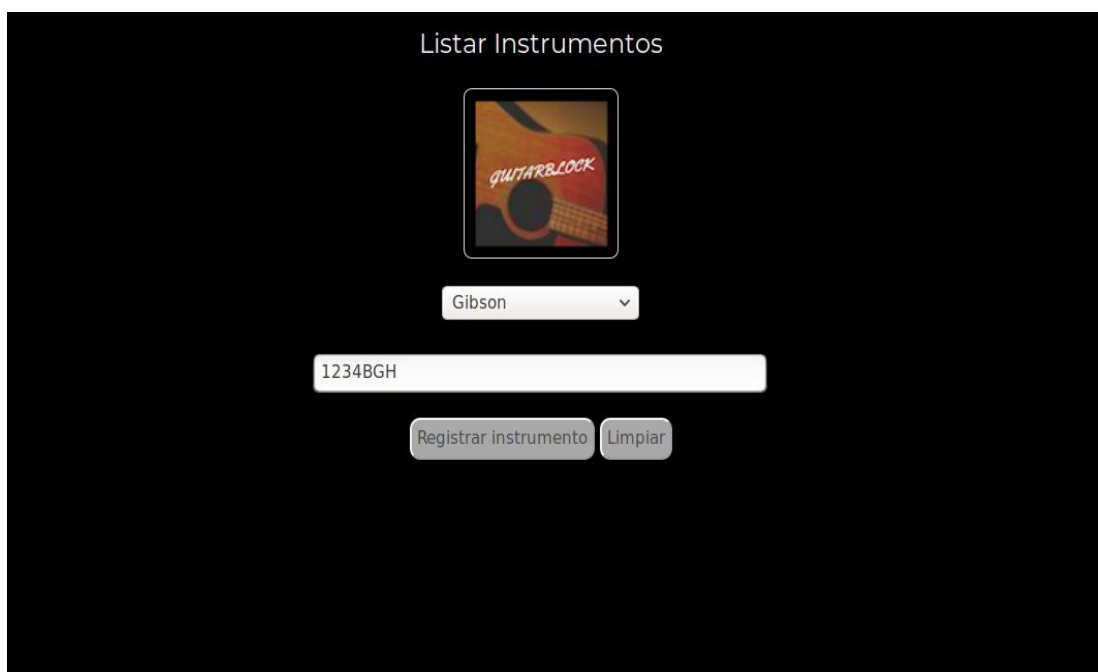


Figura 41 - Confirmación de transferencia

- **Alta de instrumento:** Al hacer click en el botón alta de instrumento se dirigirá a la pantalla en la que mediante un formulario se podrá ingresar código de serie y seleccionar la marca del instrumento , al hacer click pedirá una confirmación, en el caso de que no exista el instrumento se registrara correctamente, por el contrario, si ya está asignado a un usuario, aparecerá un mensaje de advertencia.



The screenshot shows a registration form titled "Listar Instrumentos" on a black background. At the top center is a square image of a red guitar with the text "GUITAR LOCK" written on it. Below the image is a dropdown menu with "Gibson" selected and a downward arrow. Underneath the dropdown is a white text input field containing the serial number "1234BGH". At the bottom of the form are two buttons: "Registrar instrumento" and "Limpiar".

Figura 42 - Formulario de alta

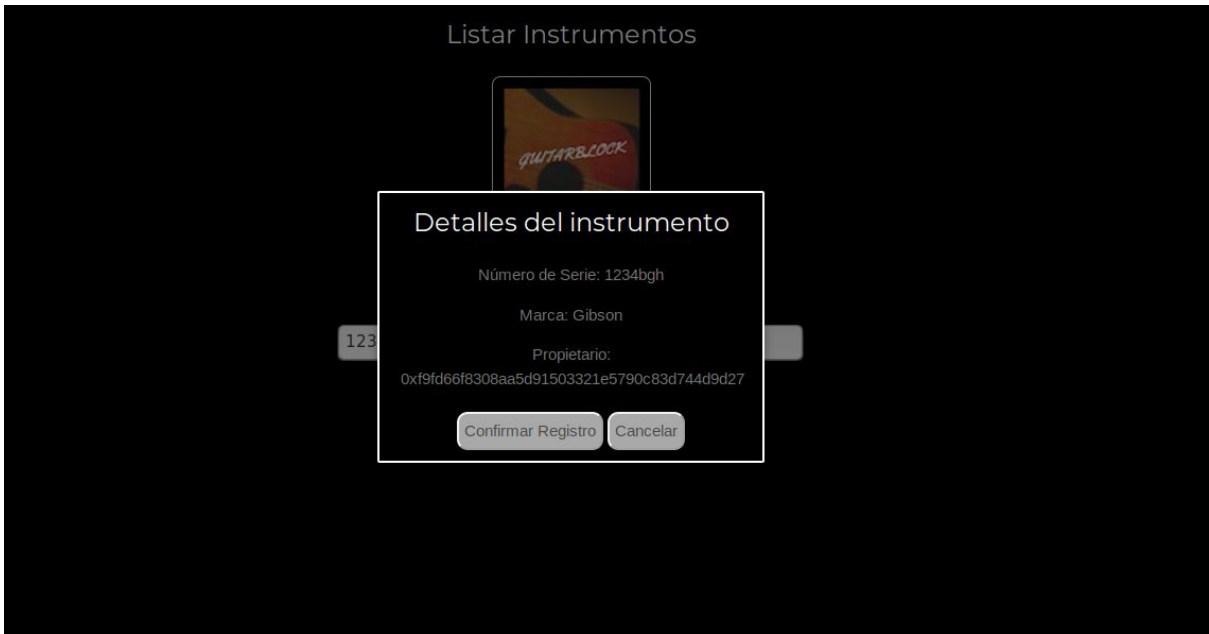


Figura 43 - Pantalla de confirmación de alta

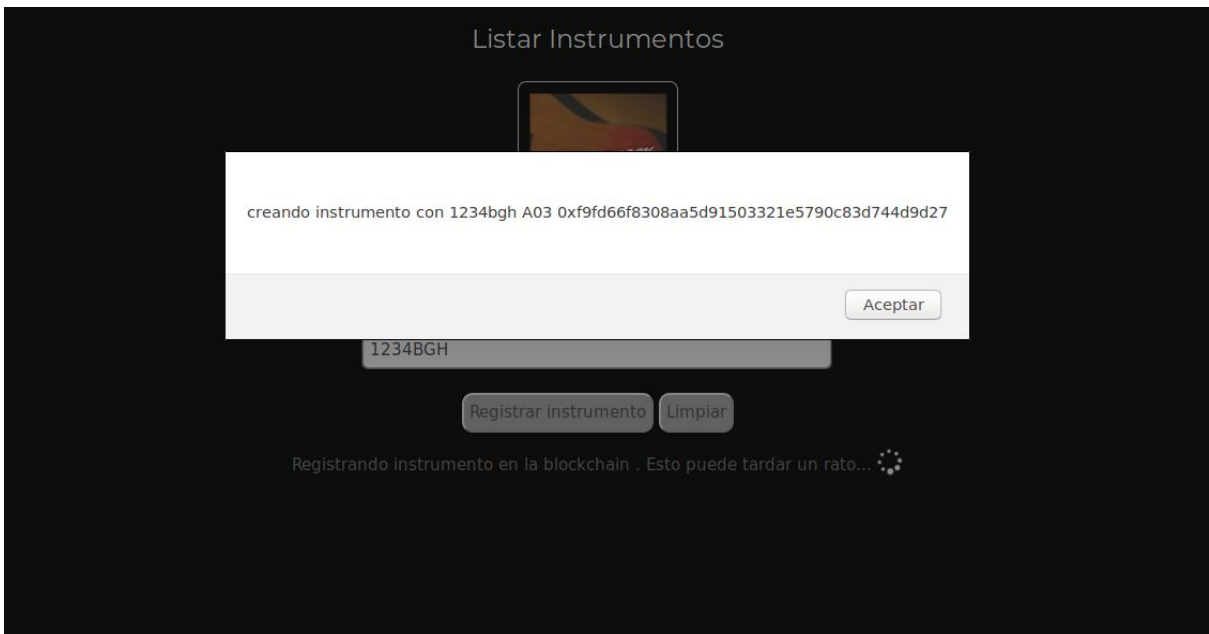


Figura 44 - Creación de instrumento

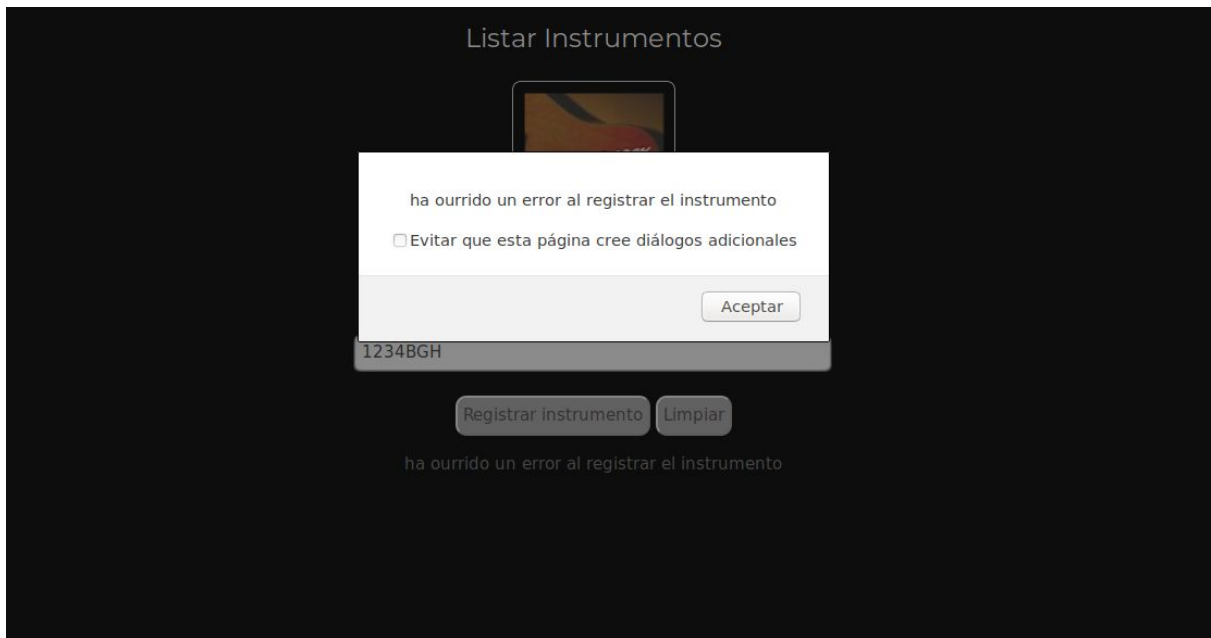


Figura 45 -Mensaje de advertencia de alta duplicada

5.6 Resumen del capítulo

En este capítulo se detallan las tecnologías y metodologías utilizadas para la creación de la herramienta, y la realización de experimentos en base a los distintos escenarios.

Capítulo 6

Conclusiones - Trabajos Futuros

6.1. Conclusiones

Durante el desarrollo de esta tesina se planteó la falta de una herramienta para el seguimiento de transacciones de instrumentos musicales que sea inmutable, segura y distribuida, esto nos llevó a investigar la tecnología blockchain y todas sus ventajas, luego se continuó con un concepto muy importante como es smart contract y su lenguaje principal solidity, posteriormente se mencionó los tokens ERC20 y ERC721.

Para la construcción del prototipo funcional se utilizó truffle que es el framework más popular de solidity para realizar el smart contract, este se basó en el token ERC721 por su característica de ser no fungible, el framework vue.js para la parte del frontend y la librería web3.js para realizar la comunicación entre el smart contract y la parte visual.

Los 'smart contracts' son una evolución del sistema legal, no una sustitución del mismo. Estos contratos inteligentes permitirán hacer negocios entre desconocidos de manera fiable y sin necesitar un intermediario de confianza. Además, el software automatizará el cumplimiento de las promesas contractuales.

Desde el punto de vista legal, una de las principales barreras es que es un fenómeno multi jurisdiccional, es decir, las partes están situadas en distintos países y el activo sobre el que se está transaccionando puede ser algo virtual, sin una presencia física. Para un abogado, esto es muy importante porque la mayoría de las veces se aplica una norma u otra en función de dónde esté físicamente el activo.

6.1.1 Retos y particularidades a tener en cuenta

1- Los programas son inalterables. Una vez desplegados en la red, no se pueden eliminar. Cuando necesitemos modificar un contrato, desplegaremos una nueva versión y haremos que nuestra aplicación invoque este nuevo contrato. Para ello, necesitamos hacer un buen diseño de nuestra aplicación siguiendo buenas prácticas de desarrollo (algunas clásicas y algunas aún en definición a medida que van surgiendo frameworks y nuevas funcionalidades en el universo blockchain).

2- Uno de los principales retos de los contratos inteligentes es la dificultad de unir dos mundos, el tecnológico y el legal. Los contratos inteligentes son escritos por los técnicos y los contratos propiamente dichos son escritos por los especialistas en leyes. Contar con alguien que domine este nuevo paradigma es esencial para poder aprovechar al máximo todo lo que nos ofrece.

3- Avanzar en legislación y aceptación. El reto no es solo traducir el mundo legal al computacional, sino que es necesario avanzar en validez jurídica de los contratos y la estandarización en los distintos sectores. [55]

6.2. Trabajos Futuros

Uno de los principales retos de esta tesis y sobre todo de los contratos inteligentes es la dificultad de unir tres mundos, el tecnológico, el musical y el legal. Los contratos inteligentes escritos por los técnicos y los contratos propiamente dichos, escritos por los especialistas en leyes.

El reto no es solo trasladar todo el lenguaje legal a un mundo computacional, sino que además se tienen que dar muchos avances para lograr su validez jurídica y estandarización en la industria.

Esta nueva tecnología disruptiva implica un desafío al trabajo de los abogados. Podría aventurarse que en dicha evolución, el profesional legal dejará de efectuar contratos individuales para focalizarse en el desarrollo de plantillas de contratos. Sin embargo, aún resulta incierto el resultado de unir el mundo tecnológico con el

legal. Parte importante de la incertidumbre en el resultado está vinculada al marco jurídico que resta establecer para regular la entidad legal de una tecnología global como blockchain.

6.3 Reflexiones y conclusiones acerca de la validez jurídica de BlockGuitars Código Civil y Comercial de la Nación - Posesión

“ARTÍCULO 1909.- Posesión. Hay posesión cuando una persona, por sí o por medio de otra, ejerce un poder de hecho sobre una cosa, comportándose como titular de un derecho real, lo sea o no.”

Teniendo en cuenta el código civil y comercial hoy, un sistema de blockchain de instrumentos musicales no podría ser viable en términos legales dado que un instrumento musical no es un bien registrable como una casa o un vehículo. En el caso que un proyecto así pueda ser llevado al congreso y aprobado, el comprador de un instrumento no podría ignorar que compró un bien robado, por lo cual no podría hacer valer el derecho por encima del dueño desapoderado.

Un sistema distribuido que permita establecer el historial de propiedad de un bien evitaría la compra-venta de instrumentos robados, ya que habría una interrupción en la cadena de tenedores que no podría ser desconocida por el poseedor actual y por lo tanto no se podría sostener que el instrumento se compró de buena fe a una persona que no es la que figura como propietario del bien.

“ARTÍCULO 1918.- Buena fe. El sujeto de la relación de poder es de buena fe si no conoce, ni puede conocer que carece de derecho, es decir, cuando por un error de hecho esencial y excusable está persuadido de su legitimidad”.

“ARTÍCULO 1919.- Presunción de buena fe. La relación de poder se presume de buena fe, a menos que exista prueba en contrario”.

La mala fe se presume en los siguientes casos:

- a) cuando el título es de nulidad manifiesta;
- b) cuando se adquiere de persona que habitualmente no hace tradición de esa clase de cosas y carece de medios para adquirirlas;
- c) cuando recae sobre ganado marcado o señalado, si el diseño fue registrado por otra persona

Código Civil y Comercial de la Nación - Herencia / Testamento

En nuestro país, el Código Civil, que regula las herencias, combina los dos sistemas.

A Partir de estos dos sistemas, se generan tres tipos de herederos:

1) Los herederos forzosos, que no pueden ser privados de la herencia mediante un testamento porque la ley no lo permite. Son los hijos, cónyuge y padres de la persona que fallece.

2) Los herederos no forzosos que heredan si no hay herederos forzosos ni testamento. Son los demás parientes hasta el cuarto grado, como los hermanos, sobrinos, tíos y primos.

3) Los herederos testamentarios, son aquellos a quienes la persona fallecida les dejó bienes mediante un testamento.

Nuestro sistema, al ser combinado permite hacer un testamento, sin embargo lo que no permite es el desheredar a un heredero forzoso.

El nuevo Código Civil y Comercial modificó el porcentaje de la legítima herencia: aumenta de un 20% a un 33,3% la proporción de la herencia que alguien puede dejar a otra persona o institución, sin importar la existencia de un vínculo familiar.

Para que se hiciera efectiva esta acción de disponer del 20% de sus bienes tras su deceso, la persona debía dejarlo expresado en un testamento. Lo que cambió con el nuevo Código Civil y Comercial es que se otorga ahora la posibilidad de que ese porcentaje sea mayor y se elevó a un tercio, y que los herederos sean depositarios de los dos tercios restantes.

En caso de no expresar su voluntad, la división del 100% de sus pertenencias se hace entre sus parientes directos herederos por legítimo derecho.

Marcos Córdoba es uno de los letrados que participó, junto con el doctor Francisco Martín Ferrer, en la redacción del Código Civil y Comercial que rige desde el 1º de agosto. Estuvo en Paraná para participar como disertante en las Segundas Jornadas Nacionales de Derecho Sucesorio, organizadas por el Colegio de Abogados de Entre Ríos y la Facultad Teresa de Ávila de la Universidad Católica Argentina, y opinó: “La muerte constituye el hecho generador de la producción más intensa de efectos jurídicos”.

En este sentido, explicó que cualquier persona mayor de 18 años y que mantenga sus aptitudes mentales puede disponer a quién dejarle un tercio de sus bienes, en caso de tenerlos. “Lo puede hacer en su casa, de puño y letra y se le llama testamento ológrafo. Tiene que contener la fecha, para saber que la persona era mayor de edad y estaba lúcida, y llevar su firma; con eso es suficiente”, indicó.

Otra forma es realizarlo ante un escribano público, y está sujeto a una tarifa emitida por el Colegio de Escribanos de cada región. En este caso queda doblemente salvaguardado. Sin embargo, Córdoba recalcó que las dos formas tienen validez, y que en el caso del testamento ológrafo se lo puede dejar a una persona encargada de difundirlo luego de su muerte, a un grupo de personas o hasta incluso a un abogado o escribano. [\[56\]](#)

Teniendo en cuenta el código civil y comercial hoy, un sistema de blockchain de instrumentos musicales no podría ser viable en términos legales ya que nuestro país no admite el testamento en su forma digital. En países como España, el tema ya ha sido tratado y llevado al congreso, por lo que Argentina

como en la mayoría de los países se deberá tratar en un futuro por decantación.

Todos sabemos lo que es un testamento o declaración de últimas voluntades. Pero seguro que pocos hemos oído hablar del **testamento digital**.

En España aparece la nueva Ley de Protección de Datos y Garantía de Derechos digitales.

Podemos definir el testamento digital como el documento con todos los servicios de correo, páginas web, redes sociales, sistemas de crédito, etc. todo aquello donde se tiene cuenta abierta en Internet. Este documento no dista mucho del testamento normal, de hecho se haría ante un notario.

El testamento digital mejora obligatoria en caso que los instrumentos se conviertan en bienes registrables o podría formar parte de futuro testamento digital siendo el instrumento un token que sólo tomará valor como tal cuando el heredero definido por el smart contract lo certifique.

6.4 Experiencias

Recientemente el Gobierno Nacional de Argentina ha certificado las ediciones electrónicas del Boletín Oficial mediante la utilización de la blockchain. De esta forma el Boletín Oficial adoptó un mecanismo adicional para que sus usuarios puedan verificar la autenticidad y obtener prueba de existencia de la edición electrónica.

Uno de los primeros países en experimentar con blockchain ha sido Estonia, desarrollando su propia blockchain privada denominada X-road, la cual permite el intercambio de datos en un entorno tecnológico y organizativo seguro. En este caso, la arquitectura de blockchain es utilizada para generar entornos interoperables y registros unificados de información. En la Argentina, la provincia de Neuquén ha tomado el modelo estonio como horizonte y ha implementado el mismo principio en su plan de integrabilidad, utilizando una blockchain privada

interestatal para el intercambio de datos. A pesar de ser una experiencia reciente, el desarrollo de la provincia en infraestructura TIC y en su modelo de integrabilidad ha permitido un rápido despliegue de esta tecnología.

El registro de tierras y de inmuebles en blockchain ha sido uno de los temas en los cuales más se ha experimentado, primero fallidamente en Honduras y actualmente en Georgia (Collindres et. al. 2016, Shin 2017). El registro de bienes inmuebles en Georgia es implementado por la Agencia Nacional de Registro Público (NAPR) dependiente del Ministerio de Justicia. Los trabajos sobre integración de registros en Blockchain comenzaron en 2016, cuando la empresa BitFury se interesó en la infraestructura de TI de la NAPR y en la posición de liderazgo de Georgia en el ranking del Banco Mundial de egov.

En resumidas cuentas, el sector público puede tomar ventaja del blockchain y la velocidad en que los ecosistemas están creciendo alrededor de esta tecnología. Sin embargo, es natural que se encuentre un cierto temor que nace de la sensación de riesgo y retos que implementar blockchain significa. [57]

Referencias Bibliográficas

[1] Seguridad de la información:

https://es.wikipedia.org/wiki/Seguridad_de_la_informaci%C3%B3n

[2]: Concepto de criptografía:

<https://www.significados.com/criptografia/>

[3],[4]: Métodos de cifrado:

<https://www.sertecomsa.com/single-post/2017/06/23/Spotlight-en-Ransomware-M%C3%A9todos-de-cifrado-de-Ransomware>

[5] Algoritmos de cifrado asimétrico

<https://www.redeszone.net/2010/11/16/criptografia-algoritmos-de-cifrado-de-clave-asimetrica/>

[6] RSA

<https://juncotic.com/rsa-como-funciona-este-algoritmo/>

http://repositori.uji.es/xmlui/bitstream/handle/10234/139037/TFG_2015_EcobarBenetM.pdf

[7] DSA

<https://es.wikipedia.org/wiki/DSA>

[8] ECC

https://es.wikipedia.org/wiki/Criptograf%C3%ADa_de_curva_el%C3%ADptica

[9] ECDSA

<https://es.wikipedia.org/wiki/ECDSA>

[10] Métodos de cifrado híbrido

https://es.wikipedia.org/wiki/Criptograf%C3%ADa_h%C3%ADbrida

[11] Criptografía (XXII): criptología para todos (III)

<http://mikelgarcialarragan.blogspot.com/2016/07/criptografia-xxi-criptologia-para-todos.html>

[12] Hash

<https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/>

[13] MD5, [14] SHA-1, [15] SHA2

<https://www.redeszone.net/2010/11/09/criptografia-algoritmos-de-autenticacion-hash/>

[16] Firma digital

<http://www.revista.unam.mx/vol.7/num7/art55/art55-3.htm>

[17] Certificado digital

<https://www.conicet.gov.ar/wp-content/uploads/Ley-N%C2%BA-25506-Firma-Digital.pdf>

[18] Blockchain

https://es.wikipedia.org/wiki/Cadena_de_bloques

[19] Funcionamiento Blockchain

<https://www.xataka.com/especiales/que-es-blockchain-la-explicacion-definitiva-para-la-tecnologia-mas-de-moda>

[20] Ejemplo en la vida real

<http://www.expansion.com/blogs/peon-de-dama/2018/01/18/blockchain-para-novatos.html>

[21] Elementos que definen una blockchain, [25] Bitcoin, [26] Ethereum, [27] API de Ethereum

<https://www2.deloitte.com/es/es/pages/technology/articles/blockchain-vision-tecnologica.html>

[22] Proof of Elapsed Time (PoET), [23] Simplified Byzantine Fault Tolerance (SBFT)

<https://viviendo20.wordpress.com/2018/06/11/blockchain-algoritmos-de-consenso/>

[23] Árboles de Merkle

<https://bitcoin.es/criptomonedas/funciones-hash-y-arboles-de-merkle-protogen-blockchain/>

[24] Raft

<https://es.wikipedia.org/wiki/Raft>

[25] La máquina virtual de Ethereum, ¿Cómo funciona Ethereum?

<https://ethereum-homestead-es.readthedocs.io/en/latest/introduction/what-is-ethereum.html>

[28] Modelo de tokens

<https://medium.com/la-disrupci%C3%B3n-del-blockchain/initial-coin-offering-un-nuevo-modelo-de-financiamiento-para-la-internet-descentralizada-86f63d0c47e8>

[29] Estándar ERC20: La base de la mayoría de las ICOs

<https://www.crypto-economy.net/estandar-erc20-la-base-de-la-mayoria-de-las-icos/>

[30] Ethereum Transaction

<https://etherscan.io/tx/0x2e81009efe3c00f4869ac4a39fa9b106d5b9fb14d73e98a70d7c5f6f96f4d807>

[31] Estándar ERC-20 para Token

<https://miethereum.com/smart-contracts/erc20/>

[32] Estándar de Token ERC 223

<https://coinsutra.com/erc223/>

[33] ¿Qué son los tokens ERC 721?

<https://www.coincrispy.com/2018/03/27/tokens-erc721/>

[34], [35], [36], [37], [38] ¿Que son los smart-contracts?

<https://academy.bit2me.com/que-son-los-smart-contracts/>

[39] Smart-contracts en Ethereum

<https://academy.bit2me.com/que-son-los-smart-contracts/> y

<https://academy.bit2me.com/ethereum-aplicaciones-descentralizadas/>

[40], [41], [42] Aplicaciones descentralizadas en Ethereum

<https://academy.bit2me.com/ethereum-aplicaciones-descentralizadas/>

[43] Smart-contracts en Ethereum

<https://www.adictosaltrabajo.com/2018/02/06/contratos-inteligentes-en-ethereum/>

[44] Solidity, el lenguaje para programar los smart-contracts

<https://www.xataka.com/empresas-y-economia/por-que-los-programadores-amamos-ethereum>

Remix: entorno de desarrollo integrado (IDE) que permite escribir contratos inteligentes basados en Solidity.

<https://remix.ethereum.org/>

Ethfiddle: Compilador en línea de Solidity más limitado que Remix.

<https://ethfiddle.com/>

Código Fuente Ethereum:

<https://github.com/ethereum/>

Curso de Solidity:

<https://tutoriales.online/curso/solidity>

Metamask: Es un plugin que hace de puente entre varias dApps y tu navegador Chrome sin comprometer tu seguridad, usando varias cuentas y sin necesidad de usar un nodo Ethereum completo.

<https://metamask.io/>

CryptoZombies: Es una escuela de programación interactiva que enseña a crear contratos inteligentes en Solidity mientras construyes tu propio juego cripto-coleccionable.

<https://cryptozombies.io/>

Manual en línea de Solidity:

<https://solidity.readthedocs.io/en/latest/>

White Paper Ethereum:

<https://github.com/ethereum/wiki/wiki/White-Paper>

Documentación oficial Web3.js:

<https://github.com/ethereum/wiki/wiki/JavaScript-API>

Sitio para solicitar Kovan Ether (KETH) para la red de pruebas de Kovan Ethereum.

<https://faucet.kovan.network/>

Conversor de criptomonedas:

<https://converter.murkin.me/>

Comunidad de desarrolladores de Ethereum:

<https://ethereum.stackexchange.com/>

Framework de Ethereum:

<http://truffleframework.com/>

Geth: que es un cliente de Ethereum programado en Go

<https://geth.ethereum.org/downloads/>

Comunidad de Slack Ethereum Chile:

<https://etherians.slack.com/>

OpenZeppelin: Framework open source de smart contracts

<https://openzeppelin.org/>

NodeJS: entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome.

<https://nodejs.org/es/>

[45] Diferencias Bajo y Guitarra:

<http://www.saberia.com/en-que-se-diferencia-el-bajo-de-la-guitarra/>

<https://es.wikipedia.org/wiki/Afinaci%C3%B3n>

[46] Bajo eléctrico

https://www.taringa.net/+hazlo_tu_mismo/parte-1-aprende-lo-basico-para-tocar-el-bajo_tuvs6

[47]Guitarra eléctrica

[48] Mercado de Guitarras y Bajos en la Argentina

<https://www.iprofesional.com/notas/262706-impuestos-importaciones-industria-monedas-mercado-competitividad-aranceles-actividad-caimavi-diego-carullo-instrumentos-musicales-Camara-de-instrumentos-musicales-piden-quita-de-aranceles-Mejorar-el-acceso-a-la-cultura-y-no-impactaria-en-la-recaudacion>

[49] Mercado negro de instrumentos en Argentina

<https://www.lavoz.com.ar/sucesos/al-rescate-de-los-instrumentos-robados>

<https://www.lacapital.com.ar/la-ciudad/roban-instrumentos-musicales-la-nueva-moda-los-delincuentes-la-ciudad-n449050.html>

<http://www.diaadia.com.ar/cordoba/suena-raro-robo-de-instrumentos-musicales>
<https://viapais.com.ar/cordoba/120682-un-nuevo-robo-de-instrumentos-se-suma-a-la-seguidilla-de-hechos-similares-y-aumenta-la-preocupacion/>

[50] Cotización musicshield

<http://www.musicshield.com.ar/>

[51] Cotización casablancas

<http://www.mcasablancas.com/>

[52] Stack tecnológico

Ethereum:

<https://www.ethereum.org>

Metamask: (Plugin que hace de puente entre varias dapps y los navegadores Firefox - Chrome sin necesidad de usar un nodo Ethereum completo).

<https://metamask.io/>

Remix: anteriormente conocido como Browser Solidity, proporciona un entorno de desarrollo integrado (IDE) que permite escribir contratos inteligentes basados en Solidity.

<https://remix.ethereum.org/>

[53] An Agile Software Engineering Method to Design Blockchain Applications

<https://arxiv.org/pdf/1809.09596.pdf>

[54] Tutorial de test unitarios con Mocha.js y Chai.js

<https://www.paradigmadigital.com/dev/testeando-javascript-mocha-chai/>

[55] Retos y particularidades a tener en cuenta

<https://www.paradigmadigital.com/dev/necesito-un-smart-contract-como-lo-hago/>

[56] Testamento / Herencia - Código Civil y Comercial Argentino - Testamento digital en España

<https://www.unoentrerios.com.ar/la-provincia/el-testamento-vuelve-cobrar-relevancia-el-nuevo-codigo-n951411.html>

<https://www.capacitarte.org/blog/nota/blog-74-el-nuevo-codigo-civil-generara-cambios-en-herencias-y-sucesiones>

<https://www.europapress.es/sociedad/noticia-congreso-regula-testamento-digital-herederos-gestionen-supriman-redes-sociales-fallecido-20181004110859.html>

[57] Blockchain y transparencia. La experiencia en la ciudad de Bahía Blanca.