



Universidad Nacional de San Luis
Facultad de Ciencias Fisico Matematicas y Naturales

Sistema de Certificación Digital de Títulos Universitarios en Blockchain (UNSL Cert)

Proyecto Final Integrador
para optar por el grado de ingeniero en informática

Autor

Vergés Federico Agustín

Director

Ana Gabriela Garis

Codirector:

Pablo Cristian Tissera

San Luis - Argentina

2024

A mis padres Graciela Arce y Juan Luis Vergés.

Gracias por enseñarme lo que es la constancia y el esfuerzo.

Gracias por hacer que mi única preocupación fuese estudiar.

Agradecimientos

Dedico este trabajo a mi familia Graciela, Juan Luis, Juan Ignacio y Francisco, motor estructural de mi vida y razón por la cual puedo concluir con mis estudios, gracias por bancar cada día de examen, cada hora de cursada y por ayudarme a levantarme temprano todos los días.

Agradezco a mis compañeros donde transitamos estos años de cursada, trabajos grupales, exámenes, mates y largas jornadas de charla y estudio, donde hice grandes amistades y encontré una camada de colegas fascinante con una calidad humana excepcional.

Quiero agradecer a mi terapeuta Cristian Quiroga, quien me ayudó en momentos complicados y logró encontrar las palabras justas para que pueda encontrar el lado positivo de las cosas y lograr aprender en las situaciones difíciles de mi vida.

Agradezco a Agustin Lopez, amigo que en una tarde de pandemia por videollamada me enseñó la tecnología blockchain y que luego esa idea finalmente se transformó en las bases de este proyecto.

La Universidad Nacional de San Luis ha sido un pilar fundamental en mi desarrollo personal y profesional, agradezco a todo el personal de la universidad que dedican su tiempo en formar futuros profesionales.

Por último, agradecer a mis tutores del proyecto final Ana Garis y Cristian Tissera por la paciencia en cada reunión y por acompañarme en este arduo camino aconsejándome de la mejor manera.

Índice General

Índice General	3
Índice de Figuras	4
Capítulo 1: Introducción	7
1.1 Propuesta	8
1.2 Antecedentes	8
1.3 Objetivos	10
1.4 Estructura del documento	10
Capítulo 2: Marco teórico	12
2.1 Sistemas Distribuidos y la Fundación de la Blockchain	12
2.2 Criptografía y la Seguridad de la Blockchain: Árboles de Merkle	13
2.3 Blockchain	14
2.3.1 Tipos de blockchain	15
2.3.2 Protocolos de consenso	16
2.3.3 Bitcoin	18
2.4 Ethereum	19
2.4.1 Ether	20
2.4.2 Cuentas	20
2.4.3 Transacciones	23
2.4.4 Ethereum Virtual Machine (EVM)	26
2.4.5 Gas	27
2.4.6 Contratos inteligentes	28
2.4.7 Protocolo JSON-RPC	31
2.4.8 Cadenas Laterales	32
2.5 Web 3	33
2.6 Aplicaciones descentralizadas Dapps	34
2.7 Modelo de certificación de títulos según BFA	34
Capítulo 3: Soporte de Tecnologías	37
3.1 Desarrollo Web	37
3.2 Blockchain	38
3.3 Patrones de Diseño	39
3.4 Despliegue de Backend y Frontend con Docker y AWS	45
Capítulo 4: Planificación y Costos	46
4.1 Planificación de actividades	46

4.2 Costos	47
4.2.1 Costo de desarrollo	47
4.2.2 Costo de despliegue	48
4.2.3 Costos operativos	50
4.3 Análisis de inversión	51
Capítulo 5: Sistema de Certificación Digital de Títulos Universitarios en Blockchain	52
5.1 Análisis	52
5.1.1 Procesos de certificación actuales	52
5.1.2 Problemas a resolver	53
5.1.3 Modelos de análisis	54
Modelo de casos de uso	54
Modelo conceptual	61
5.1.4 Requerimientos No Funcionales	61
5.2 Diseño	62
5.3 Implementación	64
5.3.1 Tecnologías aplicadas	64
5.3.2 Comunicación Frontend-Backend	65
5.3.3 Estructura del contrato inteligente	65
5.3.4 Compilación y despliegue	72
5.3.5 Comunicación Backend-Blockchain	72
5.3.6 Funcionalidades Básicas	74
5.3.7 Problemáticas asociadas	78
Manejo de transacciones y eventos	78
Costo de gas	79
Protección de claves	79
Idempotencia y verificabilidad de los datos	80
5.4 Prueba	81
Capítulo 6: Caso de estudio	84
6.1 Perfil administrativo	84
6.2 Perfil graduado	88
6.3 Perfil tercero	91
6.4 Funcionalidades adicionales	93
6.5 Casos Excepcionales	94
Capítulo 7: Conclusiones y Trabajos Futuros	96
7.1 Conclusiones	96
7.2 Trabajos Futuros	97
Referencias	98

Índice de Figuras

Figura 2.1: Árbol de Merkle [10]	14
Figura 2.2: Ejemplo de intento de corromper los datos de un bloque en la blockchain [12]	15
Figura 2.3: Esquema de la estructura de bloques de información de la red de Ethereum[20]	20
Figura 2.4: Tipos de cuentas en Ethereum [21]	21
Figura 2.5: Tipos de cuentas, composición y claves [21]	22
Figura 2.6: Transacciones y estados [21]	24
Figura 2.7: Diagrama de la ejecución de una transacción [22]	24
Figura 2.8: Objeto en formato JSON enviado en una transacción [24]	25
Figura 2.9: Diagrama de la EVM [25]	26
Figura 2.10: Extracto de la tabla de referencia de los códigos de operación de la EVM [26]	30
Figura 2.11: Código en solidity de una función [31]	31
Figura 2.12: Programa en bytecode de la función antes mencionada [32]	31
Figura 2.13: Modelo de proceso de certificación de BFA [40]	35
Figura 3.2: Código de parte del servicio de autenticación	40
Figura 3.3: Método de suscripción al observable	40
Figura 3.4: Patrón singleton en servicio de notificaciones.	41
Figura 3.5: Diagrama de clases DTOs	42
Figura 3.6: Código de PersonDto	43
Figura 3.7: Código de StudentDto	43
Figura 3.8: Función toDto	43
Figura 3.9: Estructura del patrón Restricción de Acceso en Solidity.	44
Figura 4.1: Diagrama de Gantt con planificación de tareas	46
Figura 4.2 Costo mensual en USD de EC2 según calculadora de AWS	49
Figura 4.3: Costo mensual en USD de S3 según calculadora de AWS	49
Figura 5.1: Diagrama de Casos de Uso	55
Figura 5.2: Diagrama de clases	60
Figura 5.3: Diagrama de Componentes	63
Figura 5.4: Código de las entidades del contrato.	66
Figura 5.5: Código de los mappings.	67
Figura 5.6: Signatura de la función.	67
Figura 5.7: Control de número de oblea repetido.	67
Figura 5.8: Control de certificado repetido para estudiante	68
Figura 5.9: Creación de título	69
Figura 5.10: Caso estudiante nuevo	69
Figura 5.11: Caso estudiante ya cargado	70
Figura 5.12: Emitir evento	70
Figura 5.13: Código de función de búsqueda de títulos por id de estudiante	71
Figura 5.14: Código de función de baja del título.	71
Figura 5.16: Parte del código de Web3Service	73
Figura 5.17: Código de getCertificateContract	73
Figura 5.18: Diagrama de secuencia crear título	74
Figura 5.19: Creación de una transacción	75
Figura 5.20: Firma de transacción	75
Figura 5.21: Propiedad gasPrice	75
Figura 5.22: Firma de transacción	76

Figura 5.23: Envío de una transacción	76
Figura 5.24: Diagrama de secuencia “buscar título”	77
Figura 5.25: Diagrama de secuencia baja de título	78
Figura 5.26: Manejo de eventos en la creación de títulos.	79
Figura 5.27: Estructura de título de prueba	82
Figura 5.28: Código del test	82
Figura 6.1: Inicio de sesión para ingresar al sistema	84
Figura 6.2: Formulario de “crear graduado” y barra de navegación.	85
Figura 6.4: Graduado creado con éxito.	86
Figura 6.6: Interfaz para completar la carga de un título de un nuevo graduado.	87
Figura 6.8: Listado de estados de transacciones	88
Figura 6.9: Mis títulos.	88
Figura 6.10: Detalle del título	89
Figura 6.11: Información del contrato inteligente	89
Figura 6.12: Sitio web de Etherscan.io con información detallada de la transacción y el contrato.	90
Figura 6.13: Documento .pdf con datos del título	91
Figura 6.17: Búsqueda de estudiante y su título	93
Figura 6.18: Eliminar título	93
Figura 6.20: Listado de transacciones erróneas	95
Figura 6.21: Transaccion pendiente en Etherscan.io	95

Capítulo 1: Introducción

Cuando un estudiante finaliza su carrera universitaria, todo su conocimiento se encuentra representado en un documento denominado “título universitario”. El título es un documento que certifica las capacidades de una persona para desempeñar una profesión y la habilita a realizar diferentes actividades laborales y sociales.

El título tradicional, en su versión física, está propenso a su falsificación, dado que carece de mecanismos para asegurar su integridad y origen. La falsificación de títulos universitarios es una problemática recurrente en el ámbito académico y profesional. Ante la escasa digitalización, los certificados aún siguen siendo emitidos en papel con sellos institucionales y firmas. Sin embargo, con el avance de la tecnología y herramientas de edición se logran recrear copias falsificadas que pasan desapercibidas por empleadores que necesitan a un profesional certificado.

Los procesos actuales de verificación enfrentan diversos desafíos entre los que se puede mencionar: la dependencia de terceros, la pérdida de tiempo, riesgos de manipulación del papel original y otros desafíos asociados con la expedición de estos certificados. Esto puede provocar retrasos debido a incompatibilidades administrativas o a la falta de comunicación, así como dificultades en las transferencias de credenciales entre facultades/universidades.

Actualmente el avance de la tecnología pone a disposición nuevas herramientas que permiten mejorar los procesos de certificación y verificación, la tecnología Blockchain o Cadena de Bloques ofrece una variedad de características importantes como la descentralización y eliminación de intermediarios. Ésto frecuentemente lleva a mejorar los niveles de transparencia, asegurando además la integridad de transacciones y la autenticidad, así como la agilización de los procesos.

Los beneficios que provee la Blockchain pueden ser aprovechados para desarrollar sistemas más robustos. En particular, en el ámbito educativo puede ser utilizada como soporte para almacenar y verificar certificados. El presente trabajo propone el desarrollo de una aplicación que utiliza tecnología de Cadena de Bloques o “Blockchain” para el almacenamiento y verificación de información de títulos universitarios. La misma contribuirá a que las universidades puedan obtener información de los títulos de manera más segura, garantizando integridad y autenticidad. Finalmente, la aplicación permitirá que una tercera parte (empresas, empleadores, etc) pueda verificar su autenticidad posterior a su inserción a la plataforma.

1.1 Propuesta

El proceso actual de certificación posee algunas desventajas que este proyecto plantea resolver:

- **Riesgo de manipular el título original:** Para aplicar en ciertos puestos laborales, realizar cursos de postgrado, entre otras actividades, es necesario presentar el título original, este tipo de manipulaciones tienen un alto riesgo de perder o dañar el documento. Para evitar esto se realiza una copia fiel del mismo otorgada por la universidad y firmada por escribano público. Lo que también conlleva un costo económico y de tiempo por parte del estudiante, además esta práctica no es siempre aceptada.
- **Dificultad para verificar la autenticidad de la impresión:** Si bien existen diversos mecanismos para identificar la originalidad del papel del título, verificar la información en él no es un proceso sencillo por parte de un tercero.
- **Falsificación:** Con el avance de la tecnología, las técnicas de falsificación cada vez son más exactas, como impresión de sellos y firmas apócrifas de autoridades, muchas logran imitar con bastante exactitud al papel original logrando así su cometido.

En este proyecto integrador se desarrollará un sistema basado en la tecnología Blockchain, el cual permitirá almacenar información de los certificados en forma de archivador digital y verificar la autenticidad de los mismos. El sistema pretende dar solución a los problemas antes planteados, otorgando un certificado digital fácilmente verificable, cuyos datos estarán almacenados en una red descentralizada robusta en lo que se refiere a seguridad.

Para el desarrollo se utilizarán contratos inteligentes de la red de Ethereum que permitirán gestionar los títulos e información de los graduados de manera segura, descentralizada y distribuida. Así mismo, se dispondrá de una interfaz cliente para que el graduado pueda visualizar sus certificados y generar un documento digital para que un tercero pueda verificarlo. De esa manera, se reducirán los riesgos de manipulación de certificados originales y los asociados a la autenticidad.

1.2 Antecedentes

Existen varios trabajos que han estudiado el uso de Blockchain para la gestión de títulos universitarios.

Signatura es una plataforma para la certificación y firma electrónica de documentos a través de la blockchain de Bitcoin [1]. Presenta una fortaleza importante que es la estabilidad y la consolidación de la red para los usuarios dado que existen millones de nodos en todo el mundo, sin embargo presenta algunas limitaciones para la manipulación de contratos inteligentes.

Blockcerts es un proyecto de código abierto para la certificación de documentos en la blockchain de Bitcoin, creado por el MIT [2]. Presenta la misma fortaleza y limitación que Signatura.

BCERT es un sistema descentralizado para la certificación de documentos académicos en la blockchain de Ethereum. La información se almacena en la blockchain y el certificado digital en un sistema de archivos descentralizado [3]. Como fortaleza de la propuesta se destaca la versatilidad de sus contratos inteligentes para la obtención y certificación de sus documentos, también que utiliza criptografía para almacenar la información sensible lo que aporta seguridad. Sin embargo, posee algunas limitaciones referidas al riesgo que implica tener un sistema descentralizado de archivos. Esto es, los archivos quedan expuestos a ser corrompidos, ya que son guardados fuera de la cadena de bloques.

Camilo Ruiz desarrolló un sistema para la certificación de títulos universitarios soportado por una blockchain privada de la red Ethereum [4]. La red está formada por cinco nodos validadores ubicados en entidades estratégicas para la certificación de títulos como el Ministerio de Educación. De esa forma aumenta la seguridad considerablemente, una fortaleza a considerar teniendo en cuenta la información sensible, pero comprometiendo la descentralización y la disponibilidad ya que no utiliza la totalidad de la red sino solo cinco nodos privados.

Blockchain Federal Argentina (BFA) [5] es una plataforma multiservicios abierta y participativa pensada para integrar servicios y aplicaciones sobre blockchain. Es un servicio de infraestructura en donde individuos, organismos, instituciones o empresas de cualquier sector pueden desplegar aplicaciones y servicios. Posee una blockchain basada en la red Ethereum. Utilizando esta plataforma, se ha implementado desde 2018 el Registro Público de Graduados [6] que permite chequear la veracidad de los títulos universitarios. Sin embargo, a diferencia de esta propuesta el registro sólo considera los títulos emitidos desde 2012 y expone únicamente el número de oblea del título vinculado.

Tal como será explicado en el próximo capítulo, el presente proyecto integrador tomará de base también el modelo de BFA pero incorporará nuevas funcionalidades acorde a las necesidades y particularidades de la Universidad Nacional de San Luis.

La propuesta de este proyecto integrador fue presentada como trabajo estudiantil en CoNaIISI 2021 [7]. En el trabajo se describió el modelo de una aplicación descentralizada que almacena los títulos universitarios de los estudiantes y se planteó una primera versión del trabajo actual.

1.3 Objetivos

El objetivo general de este proyecto es crear un sistema web que permita la certificación de títulos universitarios de los graduados de la Universidad Nacional de San Luis (UNSL) utilizando la tecnología blockchain.

Objetivos particulares:

- Promover el uso de nuevas tecnologías, en este caso blockchain y los contratos inteligentes, para la certificación de títulos en el ámbito universitario.
- Complementar la certificación actual brindando al graduado un respaldo a su título en papel.
- Aumentar la disponibilidad y portabilidad de los certificados de los graduados, logrando que los mismos puedan estar disponibles para su verificación desde cualquier lugar del mundo y por cualquier entidad, incluyendo empresas empleadoras y otras instituciones educativas.
- Evitar la falsificación de un título usando como soporte una plataforma segura para su almacenamiento y transparente para su verificación.

1.4 Estructura del documento

El proyecto se encuentra estructurado de la siguiente manera: durante este primer capítulo se dará a conocer los objetivos de este proyecto y qué problema intenta resolver. En adición se detallará el estudio del estado del arte respecto a diferentes trabajos realizados con la tecnología blockchain, su integración con la educación y diversos procesos de certificación. En el siguiente capítulo se presentarán los conceptos vinculados a Sistemas Distribuidos, Redes, Blockchain, Contratos inteligentes, Ethereum Virtual Machine y Aplicaciones descentralizadas. En el capítulo tres se hará una descripción del soporte tecnológico necesario para llevar a cabo este proyecto como el framework Angular y el entorno Nodejs para desarrollar aplicaciones web, el lenguaje Solidity y el framework Truffle para desarrollar contratos inteligentes en Ethereum, entre otros. Continuamos con la planificación del proyecto donde se expondrán los costos de desarrollo y operacionales, además de la duración de las tareas que se deben realizar. En el capítulo siguiente se explica la propuesta de desarrollo de la aplicación, se realizará un análisis más profundo de la problemática y se detallarán los requerimientos funcionales y no funcionales de la aplicación para luego exponer el diseño y la implementación de la misma. En adición se detalla un caso de estudio completo mostrando de manera gráfica las funcionalidades principales del sistema con sus respectivos actores. Finalmente se exponen las conclusiones y trabajos futuros.

Capítulo 2: Marco teórico

2.1 Sistemas Distribuidos y la Fundación de la Blockchain

En el corazón de la tecnología blockchain yace el concepto de sistemas distribuidos, donde múltiples nodos de una red colaboran para crear un entorno descentralizado y resistente a fallos. Como lo describe Tanenbaum,

“Un sistema distribuido es una colección de computadoras independiente que aparecen ante los usuarios del sistema como una única computadora”[8]

En una red blockchain, como la de Bitcoin o Ethereum, este concepto se materializa a través de nodos distribuidos globalmente que trabajan en conjunto para mantener y validar un registro compartido de transacciones. Cada nodo opera de manera autónoma, ejecutando el software de la red y contribuyendo a la verificación y confirmación de transacciones.

Clasificación de Flynn y su Relación con la Blockchain

La clasificación de Flynn[9] nos proporciona un marco para entender cómo los sistemas con múltiples CPUs procesan datos e instrucciones. En el contexto de la blockchain, destacamos dos clasificaciones relevantes:

- **SISD (Single Instruction Single Data):** hace referencia a los computadores con un solo CPU tradicionales que poseen una sola instrucción a un solo conjunto de datos, es decir, que las instrucciones se aplican de una a la vez a los datos de manera secuencial. Esta es la arquitectura de las primeras computadoras como la Máquina de Von Neumann.
- **SIMD (Single Instruction Multiple Data):** es un tipo de arquitectura de un sistema multiprocesador, se ejecutan instrucciones que permiten aplicar una misma operación sobre diferentes conjuntos de datos al mismo tiempo en forma paralela. Un ejemplo de este tipo son las unidades de procesamiento gráfico GPU que son muy utilizadas en el ámbito profesional para procesamiento de gráficos en tiempo real como videojuegos.
- **MISD (Multiple Instruction Single Data):** para este tipo de clasificación no existen implementaciones reales, se trata de aplicar múltiples instrucciones al mismo conjunto de datos.
- **MIMD (Multiple Instruction Multiple Data):** aquí se aplican múltiples instrucciones a múltiples conjuntos de datos, se considera una evolución a la clasificación SIMD.

Esencialmente son un grupo independiente de computadoras que poseen sus propios datos. Sobre esta clasificación podemos destacar dos tipos:

- Multiprocesadores: conocidas como computadoras con memoria compartida, poseen un mismo espacio de memoria que es utilizado para comunicarse entre ellos.
- Multicomputadores conocidos como computadoras con memoria distribuida, se comunican por medio de mensajes.

2.2 Criptografía y la Seguridad de la Blockchain: Árboles de Merkle

La seguridad en la blockchain se logra mediante técnicas criptográficas avanzadas, como los árboles de Merkle y Patricia. Estos árboles permiten una verificación eficiente de la integridad de los datos almacenados en la blockchain.

- **Árboles de Merkle:** Estos árboles, que consisten en nodos con un gran número de nodos hoja y un único nodo raíz, garantizan la integridad de los datos almacenados. Cada nodo contiene el hash de sus nodos hijos, lo que permite detectar cualquier modificación en los datos con eficiencia.[10]

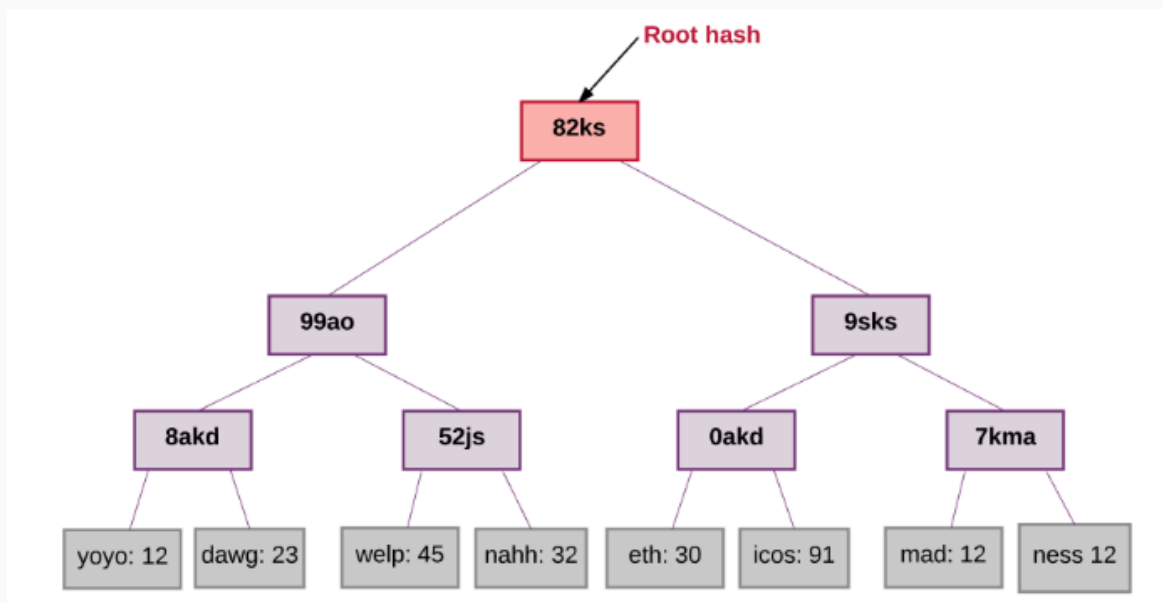


Figura 2.1: Árbol de Merkle [10]

- **Aplicación en la Blockchain:** En la blockchain, los bloques de transacciones se organizan en una cadena, y cada bloque contiene un hash que apunta al bloque anterior. Los árboles de Merkle se utilizan para comprimir múltiples transacciones en un solo hash, facilitando la verificación de la integridad de las transacciones en un bloque.

2.3 Blockchain

Según el sitio web de la documentación del proyecto Ethereum define a la blockchain como “una especie de base de datos pública que se actualiza y se comparte en una serie de ordenadores conectados en red” [11].

"Block" se refiere al hecho de que los datos y el estado se almacenan en lotes secuenciales o "bloques" de transacciones, "Chain" se refiere al hecho de que cada bloque hace referencia criptográficamente a su antecesor, en otras palabras: los bloques están encadenados, tal como se visualiza en la figura 2.2 Los datos de un bloque no pueden modificarse sin cambiar todos los bloques posteriores, lo cual requeriría en consenso de toda la red.

Cada ordenador de la red debe aceptar los nuevos bloques y la cadena en su conjunto. Estos ordenadores se conocen como «nodos». Los nodos garantizan que todas las personas que interactúan con la cadena de bloques tengan los mismos datos. Para lograr este acuerdo distribuido, las cadenas de bloques necesitan un mecanismo de consenso.



Figura 2.2: Ejemplo de intento de corromper los datos de un bloque en la blockchain [12]

2.3.1 Tipos de blockchain

Una blockchain puede ser clasificada teniendo en cuenta dos criterios, participación y capas.

- Participación: Éste concepto va de la mano con la descentralización y la gobernabilidad de la red. Hay tres tipos [12]:
 - Pública: Es controlada y mantenida por la comunidad, es decir, por todos los ordenadores nodos que validan y mantienen segura la red. Este tipo de blockchain mantiene abierto al público sus datos, el software y su desarrollo, de forma que cualquier persona puede revisar, auditar, desarrollar o mejorar los mismos. Bitcoin, Ethereum son los ejemplos más populares.
 - Privada o permissionada: Este tipo de blockchain generalmente cuenta con los mismos elementos que una blockchain pública, pero a diferencia de éstas, las blockchain permissionadas dependen de una unidad central que controla todas las acciones dentro de la misma, por lo que la descentralización se ve comprometida. Esta unidad central es la que permite dar acceso a los usuarios, además de controlar sus funciones y permisos dentro de la blockchain. Existen desarrollos de software libre, uno de los desarrollos de blockchain privadas más importantes del mundo criptográfico es “Hyper Ledger”[13].

- Híbrida o federada: Este tipo de blockchain es una fusión entre las blockchain públicas y las privadas. En estas blockchain, la participación en la red es privada. Es decir, el acceso a los recursos de la red es controlado por una o varias entidades. Sin embargo, los datos de la cadena de bloques son accesibles de forma pública. Esto significa que cualquier persona puede explorar bloque a bloque todo lo que sucede en dicha blockchain. Un ejemplo de esto es Blockchain Federal Argentina, un proyecto del gobierno Argentino.
- Capas: Existen diferentes capas para el desarrollo de una blockchain, esto depende de 3 factores, escalabilidad, seguridad y descentralización.
 - Capa 0: Los componentes de la capa 0 incluye Internet, el hardware y las conexiones que permitirán que la capa uno funcione sin problemas.
 - Capa 1: Esta capa se encarga de los procesos de consenso, los lenguajes de programación, el tiempo de bloque, la resolución de disputas y las reglas y parámetros que mantienen la funcionalidad básica de una red blockchain. También se conoce como capa de implementación. Bitcoin es un ejemplo de blockchain de capa uno.
 - Capa 2: Se refieren a una red o tecnología que opera sobre una blockchain para mejorar su escalabilidad y eficiencia. Este tipo de soluciones pasa parte de la carga de transacciones a un sistema adyacente que maneja la mayor parte del procesamiento de la red y, al finalizar los resultados, informa a la blockchain principal. De esta forma se “descongestiona” la blockchain principal.

2.3.2 Protocolos de consenso

Los mecanismos de consenso, también llamados protocolos o algoritmos de consenso, permiten que los sistemas distribuidos creen un entorno para colaborar y mantenerse seguros. Este mecanismo implica la aceptación de todos los nodos miembros de la blockchain sobre la información que hay en la misma. Es decir, que todos los nodos aceptan que el último bloque ha sido agregado a la cadena de manera correcta, que es el mismo para todos, que no presenta manipulaciones ni datos erróneos [14].

Prueba de trabajo

Fue el primer algoritmo de consenso que surgió y, hasta la fecha, sigue siendo el dominante. Fue introducido por Satoshi Nakamoto en el libro blanco de Bitcoin de 2008, pero la tecnología en sí fue concebida mucho antes. Anteriormente mencionamos que una blockchain mantiene su información en bloques que están encadenados criptográficamente, esto aporta mucha seguridad a la red ya que los datos son prácticamente incorruptibles, sin embargo, añadir un bloque no es barato. La prueba de trabajo requiere que un minero (el usuario que crea el bloque) utilice algunos de sus propios recursos

para tener ese privilegio. Ese recurso es la potencia de cálculo, que se utiliza para hacer un hash de los datos del bloque hasta que se encuentra una solución a un rompecabezas.

Hacer un hash de los datos del bloque significa pasarlos por una función de hash para generar un hash del bloque. El hash de bloque funciona como una "huella digital": es una identidad para los datos de entrada y es única para cada bloque. Es prácticamente imposible invertir un hash de bloque para obtener los datos de entrada. Sin embargo, conociendo una entrada, es trivial confirmar que el hash es correcto. Sólo tienes que enviar la entrada a través de la función y comprobar si la salida es la misma.

En la prueba de trabajo, debes proporcionar datos cuyo hash coincida con ciertas condiciones. Pero no sabes cómo llegar a ello. Tu única opción es pasar tus datos a través de una función hash y comprobar si coincide con las condiciones. Si no lo hacen, tendrás que cambiar ligeramente tus datos para obtener un hash diferente. Si cambias incluso un carácter de tus datos, el resultado será totalmente diferente, por lo que no hay forma de predecir cuál puede ser la salida.

Como resultado, si quieres crear un bloque, es necesario realizar prueba y error. Lo normal es que se tome la información de todas las transacciones que quieres añadir y algunos otros datos importantes, y luego se juntan todos. Pero como el conjunto de datos no va a cambiar, se debe añadir un dato que sea variable. De lo contrario, siempre se obtendría el mismo hash como salida. Este dato variable es lo que se denomina un nonce. Es un número que cambiará en cada intento, por lo que se obtiene un hash diferente cada vez. Y esto es lo que llamamos minería.

Prueba de participación

El algoritmo de consenso Proof of Stake se introdujo en 2011 en el foro de Bitcointalk. Se propuso como solución a los problemas de performance de Proof of Work ya que éste consume mucha cantidad de procesamiento por cada nodo que, finalmente, la mayoría es desperdiciado. Este algoritmo utiliza un proceso de elección pseudoaleatorio para seleccionar validadores de un grupo de nodos. El sistema utiliza una combinación de factores, como la edad de la apuesta, un elemento de aleatoriedad y la riqueza del nodo. En los sistemas Proof of Stake, los bloques se "falsifican" en lugar de ser minados. La mayoría de las criptomonedas Proof of Stake se lanzan con un suministro de monedas "pre-forjadas" para que los nodos puedan empezar inmediatamente.

Los usuarios que participan en el proceso de forja deben bloquear una cierta cantidad de monedas en la red como su apuesta. El tamaño de la apuesta determina las posibilidades de que un nodo sea seleccionado como próximo validador: cuanto mayor sea la apuesta, mayores serán las posibilidades.

En el proceso de selección se añaden métodos únicos para favorecer no sólo a los nodos más ricos de la red. Los dos métodos más utilizados son la selección aleatoria de bloques y la selección de la edad

de las monedas.

- Selección aleatoria de bloques: En el método de selección de bloques aleatorios, los validadores se seleccionan buscando los nodos con una combinación del valor hash más bajo y la apuesta más alta. Dado que los tamaños de las apuestas son públicos, los demás nodos pueden predecir el próximo falsificador.
- Selección de la edad de las monedas: El método de selección de la edad de las monedas elige los nodos en función del tiempo que llevan apostados sus tokens. La edad de las monedas se calcula multiplicando el número de días que llevan apostadas por el número de monedas apostadas.

Una vez que un nodo ha forjado un bloque, su edad de la moneda se pone a cero, y debe esperar un cierto período para poder forjar otro bloque - esto evita que los nodos con grandes apuestas dominen el blockchain.

Validación de las transacciones

Cada criptomoneda que utiliza un algoritmo Proof of Stake tiene su propio conjunto de reglas y métodos combinados para lo que cree que es la mejor combinación posible para la red y sus usuarios. Cuando un nodo es elegido para falsificar el siguiente bloque, comprobará si las transacciones del bloque son válidas. A continuación, firma el bloque y lo añade a la cadena de bloques. Como recompensa, el nodo recibe las tasas de transacción del bloque y, en algunas blockchains, una recompensa en forma de moneda.

Si un nodo quiere dejar de ser un falsificador, su participación y las recompensas ganadas se liberarán después de un cierto período, dando tiempo a la red para verificar que no hay bloques fraudulentos añadidos a la blockchain por el nodo [16].

2.3.3 Bitcoin

Bitcoin fue la primera criptomoneda, anunciada en 2008 (y lanzada en 2009) por una persona o grupo de personas bajo el seudónimo de Satoshi Nakamoto, cuya identidad concreta se desconoce. Ofrece a los usuarios la posibilidad de enviar y recibir dinero digital (bitcoins, ó BTC).

Es un sistema de efectivo electrónico persona a persona (peer-to-peer), resuelve el problema del doble gasto (un bitcoin no puede gastarse dos veces) mediante el uso de la tecnología de blockchain como una herramienta de consenso distribuido, junto con el mecanismo de consenso de Prueba de Trabajo. El protocolo de consenso de Bitcoin asegura que cada transacción sea verificada y agregada de manera consensuada al registro público descentralizado, el blockchain, sin necesidad de un intermediario central. Pero, ¿qué es un intermediario? Un intermediario es una autoridad central como

un banco o gobierno que interviene en una transacción entre el remitente y el receptor. Un intermediario tiene el poder de afirmar, censurar o revertir transacciones y puede compartir los datos confidenciales que recopilan con terceros [17].

Criptomoneda

Las criptomonedas presentan un enfoque diferente en comparación con las transacciones convencionales. Estas se realizan directamente entre el remitente y el destinatario, sin necesidad de una autoridad central. Nadie más tiene acceso a los fondos ni puede limitar los servicios que se pueden utilizar. Esto es posible gracias a la tecnología de cadena de bloques sobre la que operan las criptomonedas.

Las cadenas de bloques utilizan técnicas criptográficas para garantizar la seguridad de los fondos. Estas técnicas han sido utilizadas en la industria bancaria durante años para asegurar las transacciones monetarias. Todo comenzó con Bitcoin, que puede emplearse para enviar dinero a cualquier persona en cualquier parte del mundo. Lo que diferencia a las criptomonedas de las transacciones bancarias u otros servicios financieros es que, por primera vez, no existen intermediarios [18].

2.4 Ethereum

Ethereum es un proyecto basado en la tecnología blockchain de código abierto creado en 2013 por Vitalik Buterin y posteriormente lanzado en 2015 con el propósito de crear un protocolo para desarrollar aplicaciones descentralizadas, lo logra construyendo lo que es esencialmente la capa fundacional abstracta definitiva: una blockchain con un lenguaje de programación Turing completo, que permite a cualquiera escribir contratos inteligentes y aplicaciones descentralizadas donde pueden crear sus propias reglas [19].

La red Ethereum es un ordenador único y canónico (llamado máquina virtual de Ethereum o EVM), cuyo estado es consensuado y compartido por todos los participantes de la red. Cualquier persona que participe en la red de Ethereum (cada nodo de Ethereum) guarda una copia del estado de este ordenador. Asimismo, cualquier participante puede emitir una petición para que éste ordenador realice una tarea en específico como un cálculo arbitrario. Cuando se emite una solicitud de este tipo, otros participantes de la red verifican, validan y ejecutan el cálculo. La ejecución causa un cambio en el estado de la EVM, el cual se realiza y propaga a toda la red.

Las solicitudes de cálculos se denominan solicitudes de transacción; el registro de todas las transacciones y el estado de la EVM se almacena en la cadena de bloques y se confirma en todos los nodos.

El estado global de Ethereum tiene millones de transacciones. Estas transacciones se agrupan en "bloques". Un bloque contiene una serie de transacciones, y cada bloque se encadena con su bloque anterior.

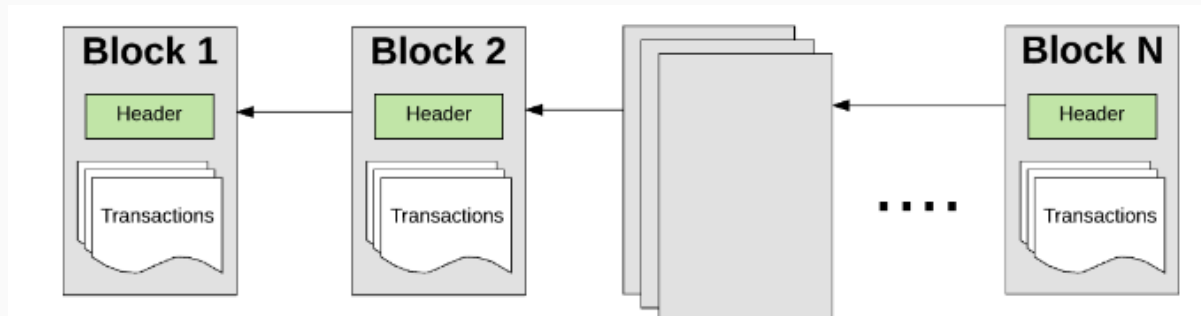


Figura 2.3: Esquema de la estructura de bloques de información de la red de Ethereum[20]

Dada la esencia de la tecnología blockchain, los mecanismos criptográficos garantizan que, una vez que las transacciones se verifican y se añaden a la cadena de bloques, no pueden manipularse más tarde, tal como se ve en la figura 2.3

2.4.1 Ether

Ether (ETH) es la criptomoneda nativa de Ethereum, incluye un mecanismo de fijación de precios para determinar el poder de computación de la red. Cuando los usuarios quieren realizar una transacción, deben pagar Ether para que su transacción sea reconocida en la cadena de bloques. Estos costes de uso se conocen como comisiones de Gas; la comisión depende de la cantidad de potencia de computación requerida para ejecutar la transacción y la demanda de energía de la red en el momento de la transacción.

El propósito del ether es posibilitar la existencia de un mercado de computación, un mercado de este tipo proporciona un incentivo económico a los participantes que verifican y ejecutan las solicitudes de transacciones y proporcionan recursos a la red.

2.4.2 Cuentas

Como mencionamos anteriormente, la red Ethereum es un ordenador canónico que posee un estado global que es mantenido compartido por todos los participantes de la red, el mismo se compone de objetos denominados "cuentas" que pueden interactuar entre sí a través pasaje de mensajes. Cada cuenta tiene una dirección de 20 bytes y transiciones de estado, que son transferencias directas de valor e información entre cuentas [23].

Existen dos tipos de cuentas:

- De propiedad externa: cualquier persona que disponga de las claves privadas puede controlarla. Una cuenta de propiedad externa no tiene código, y se puede enviar mensajes desde una cuenta de propiedad externa creando y firmando una transacción. La creación de estas cuentas no tienen ningún coste.
- De contrato: se trata de un contrato inteligente implementado en la red, que se controla mediante código, por lo tanto no se necesitan claves privadas. Cada vez que la cuenta de contrato recibe un mensaje, su código se activa, permitiéndole leer y escribir en el almacenamiento interno y enviar otros mensajes o crear contratos uno por uno.
La creación de un contrato tiene un coste porque está usando almacenamiento en la red, solo se pueden enviar transacciones como respuesta a una transacción recibida.

Las cuentas tienen la capacidad de interactuar con otros contratos inteligentes además de recibir, almacenar y enviar ETH u otros tokens.

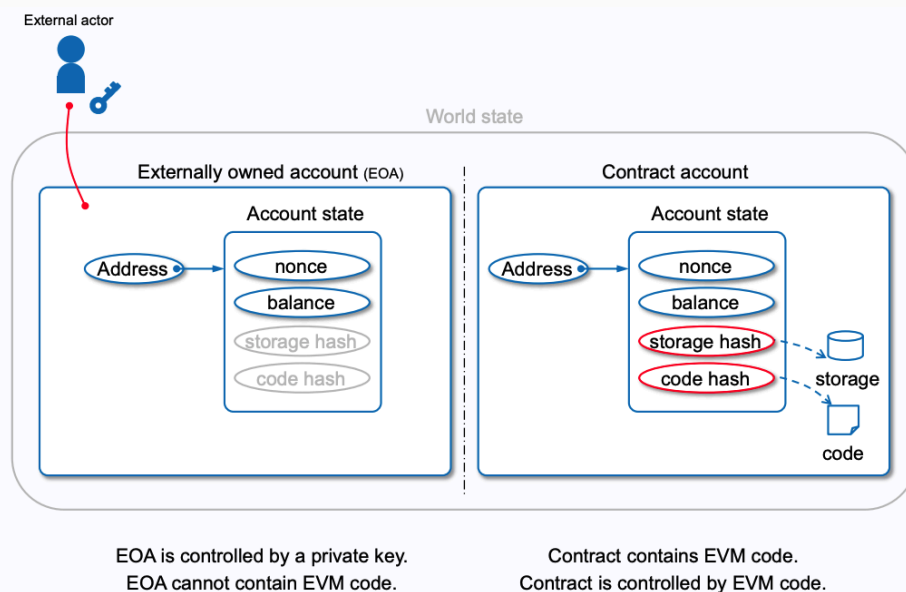


Figura 2.4: Tipos de cuentas en Ethereum [21]

Cuentas de propiedad externa

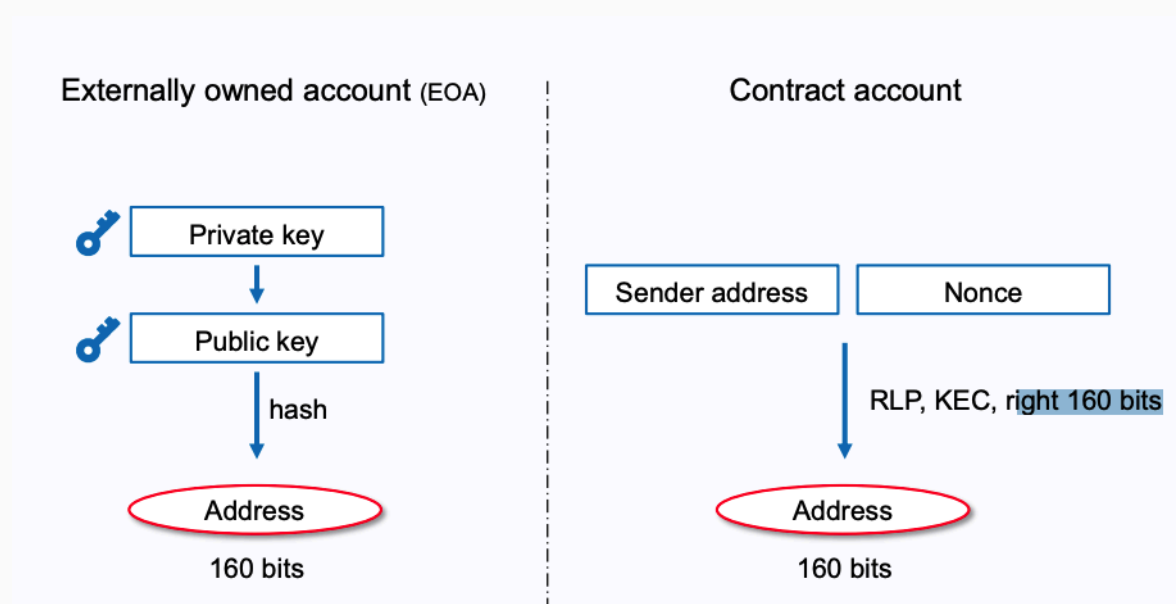
Este tipo de cuenta está formada por un par criptográfico de claves: pública y privada. Estas claves ayudan a probar que una transacción fue realmente firmada por el remitente y prevenir falsificaciones. La clave privada es utilizada para firmar transacciones, por lo que otorga al individuo la custodia de los fondos relacionados a su cuenta. Realmente nunca se almacenan criptomonedas, sino que se dispone de claves privadas: **los fondos están siempre en la cadena de bloques de Ethereum**. Esto evita que actores maliciosos difundan transacciones falsas a la red, ya que siempre se puede verificar el remitente de una transacción.

Al momento de crear una nueva cuenta en Ethereum, se genera una clave privada aleatoria con la cual se puede generar una clave pública mediante el uso del algoritmo de firma digital de curva elíptica. Éste proceso es de una sola vía, es decir, que es posible obtener nuevas claves públicas a partir de nuestra clave privada, pero no podemos obtener la clave privada a partir de las claves públicas. Esto significa que es vital mantener una clave privada segura y, como su nombre sugiere, cerciorarse de que sea privada.

Cuentas de contrato

Un mensaje de una cuenta de propiedad externa a una cuenta de contrato activa el código de la cuenta de contrato, permitiéndole realizar varias acciones (por ejemplo, transferir tokens, escribir en el almacenamiento interno, acuñar nuevos tokens, realizar algún cálculo, crear nuevos contratos, etc.); las cuentas de contrato no pueden iniciar nuevas transacciones por sí mismas.

La dirección del contrato se asigna cuando un contrato se implementa en la cadena de bloques de Ethereum. La dirección se obtiene de la dirección del creador y del número de transacciones enviadas desde esa dirección (el «nonce»).



A 160-bit code used for identifying accounts.

Figura 2.5: Tipos de cuentas, composición y claves [21]

Composición

Las cuentas Ethereum tienen cuatro campos:

- **Nonce:** Un contador que indica el número de transacciones enviadas desde la cuenta. Esto asegura que las transacciones solo se procesan una vez. En una cuenta de contrato, este número representa el número de contratos creados por la cuenta.
- **Saldo de la cuenta:** El saldo en Ether que posee la cuenta.
- **CodeHash:** Las cuentas de contrato tienen fragmentos de código programados que pueden realizar diferentes operaciones. Este código se ejecuta si la cuenta recibe una llamada de mensaje. Este campo no se puede modificar, a diferencia de otros campos de la cuenta. Para las cuentas de propiedad externa, el campo codeHash es el hash de una cadena vacía.
- **StorageRoot:** a veces conocido como hash de almacenamiento. Un hash de 256 bits del nodo raíz del árbol de Merkle Patricia, la estructura de datos que codifica el contenido de almacenamiento de la cuenta y está vacío por defecto, tal como se describe en la Figura 2.5.

Una cuenta no es una billetera. Una cuenta consiste en un par de claves para una cuenta de Ethereum que pertenece a un usuario. Una cartera es una interfaz o aplicación que le permite interactuar con su cuenta de Ethereum.

Billeteras digitales

Las billeteras digitales de Ethereum son aplicaciones que le permiten interactuar con su cuenta de Ethereum. Similar a una aplicación bancaria de Internet (comúnmente Homebanking).

La billetera le permite leer el saldo en ella, enviar transacciones y conectarse a aplicaciones. Es únicamente una herramienta para gestionar una cuenta de Ethereum. Esto significa que puede intercambiar proveedores de billeteras en cualquier momento. Algunas permiten administrar varias cuentas de Ethereum desde una sola aplicación. Esto es debido a que las billeteras no tienen la custodia de sus fondos; la tiene el usuario.

2.4.3 Transacciones

Una transacción de Ethereum se refiere a una acción iniciada por una cuenta propiedad externa, es decir, una cuenta gestionada por un humano, no por un contrato. Por ejemplo, si Juan envía a Pedro 1 ETH, el dinero se descuenta de la cuenta de Juan y se abona a la cuenta de Pedro. Esta acción de cambio de estado tiene lugar dentro de una transacción, que luego es ejecutada por un minero que finalmente propagará el cambio de estado a toda la red.

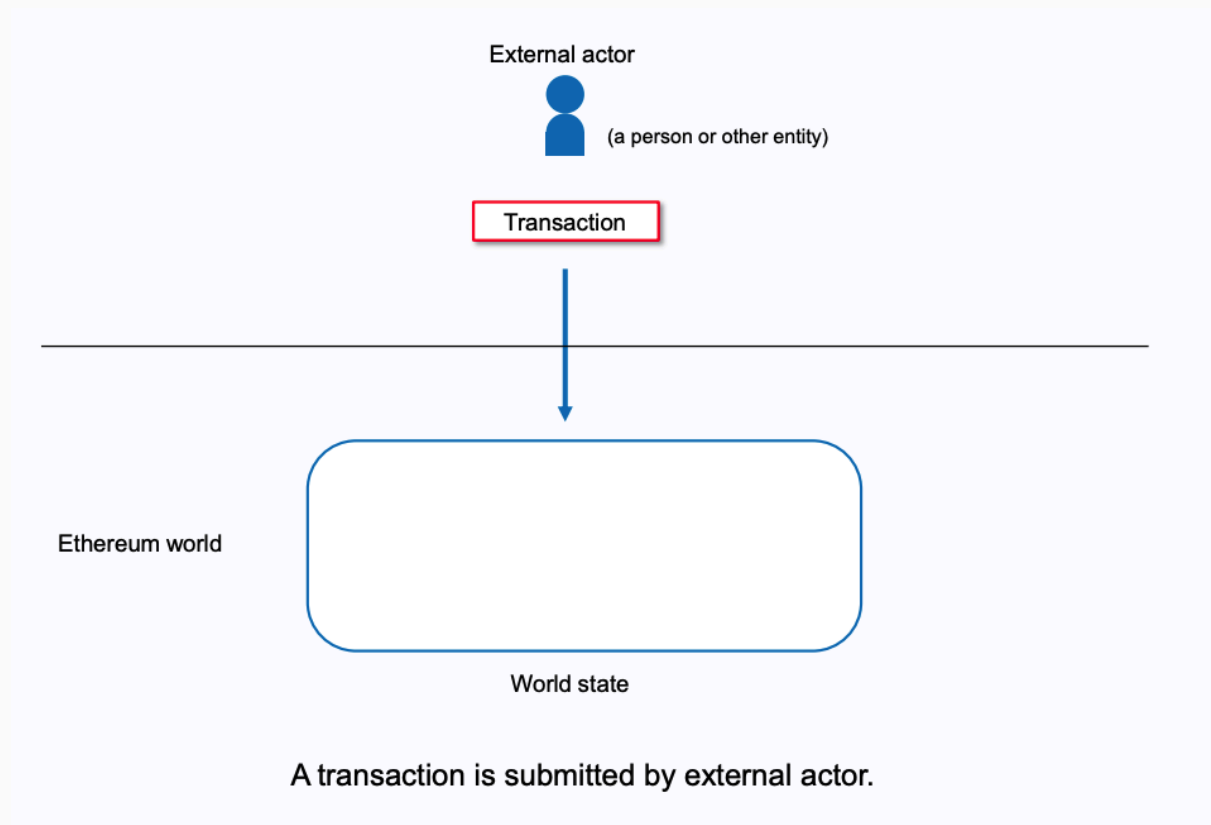


Figura 2.6: Transacciones y estados [21]

Las transacciones deben ser ejecutadas por un minero y cargadas a la cadena a través de un bloque validado. El procesamiento de las mismas tienen una tarifa, tal como se visualiza en la figura 2.7, en siguientes secciones detallaremos las tarifas de Gas.

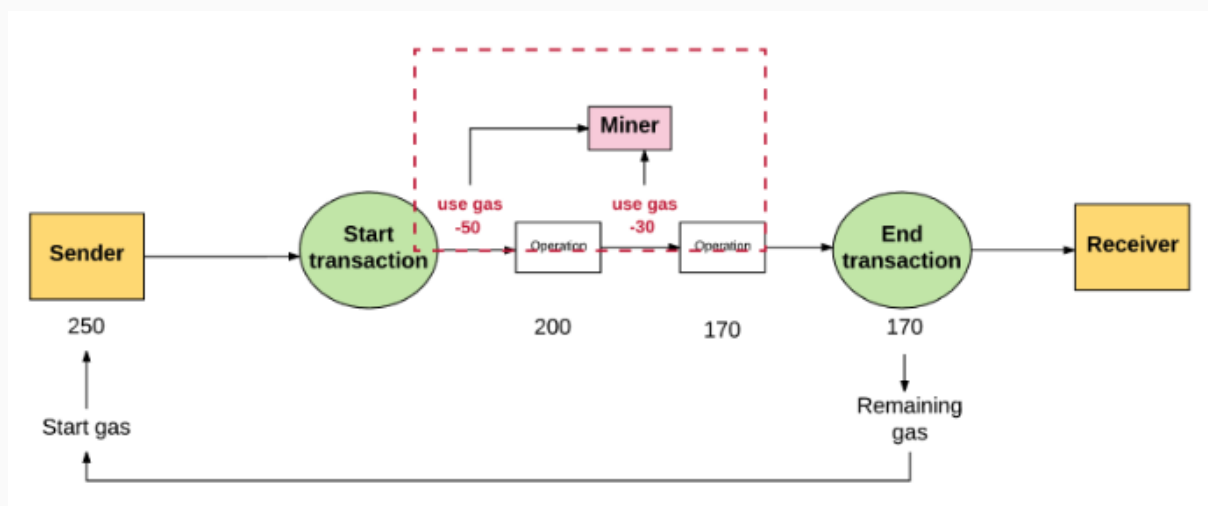


Figura 2.7: Diagrama de la ejecución de una transacción [22]

Componentes de una transacción

El término "transacción" se utiliza en Ethereum para referirse al paquete de datos firmados que almacena un mensaje para ser enviado desde una cuenta de propiedad externa. Las transacciones contienen:

- **From:** la dirección del remitente, que firmará la transacción. Esta será una cuenta de titularidad externa, ya que las cuentas contractuales no pueden enviar transacciones.
- **To:** la dirección de recepción (destinatario), si se trata de una cuenta de propiedad externa, la transacción transferirá el valor, por el contrario, si se trata de una cuenta de contrato, la transacción ejecutará el código del contrato.
- **Firma:** el identificador del remitente. Se genera cuando la clave privada del remitente firma la transacción y confirma que el remitente ha autorizado esta transacción.
- **Nonce:** un contador que se incrementa secuencialmente y que indica el número de transacción de la cuenta.
- **Value:** cantidad de ETH a transferir del remitente al destinatario.
- **Data:** campo opcional para incluir datos arbitrarios
- **GasLimit:** la cantidad máxima de unidades de Gas que puede consumir la transacción, estas unidades representan pasos computacionales.
- **maxPriorityFeePerGas:** la cantidad máxima de Gas que se incluirá como propina al minero.
- **maxFeePerGas:** la cantidad máxima de Gas que se quiere pagar por la transacción.

El Gas es una referencia al cálculo requerido para procesar la transacción por un minero, los usuarios tienen que pagar una tarifa por este cálculo.

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000",
  maxFeePerGas: "300",
  maxPriorityFeePerGas: "10",
  nonce: "0",
  value: "10000000000"
}
```

Figura 2.8: Objeto en formato JSON enviado en una transacción [24]

Un objeto de transacción debe ser firmado con la clave privada del remitente. Esto muestra que la transacción sólo puede proceder del remitente y que no ha sido enviada de forma fraudulenta.

2.4.4 Ethereum Virtual Machine (EVM)

El protocolo de Ethereum mantiene el funcionamiento de una máquina de estado especial llamada Ethereum Virtual Machine (EVM). Esta máquina almacena todas las cuentas y contratos inteligentes de Ethereum. En cada bloque de la cadena, Ethereum tiene un estado único y la EVM define las reglas para calcular un nuevo estado válido de bloque a bloque.

Para explicar esta característica más compleja se requiere una analogía más sofisticada. En lugar de un libro mayor distribuido, Ethereum es una máquina de estado distribuida. El estado de Ethereum es una gran estructura de datos, que no solo sostiene todas las cuentas y saldos, sino que también alberga el estado de la máquina. Este puede cambiar de bloque a bloque según un conjunto de reglas predefinidas, así como ejecutar un código de máquina arbitrario. Las reglas específicas de cambiar el estado de bloque a bloque las define la EVM. En la figura 2.9 se pueden observar las diferentes partes de la máquina virtual.

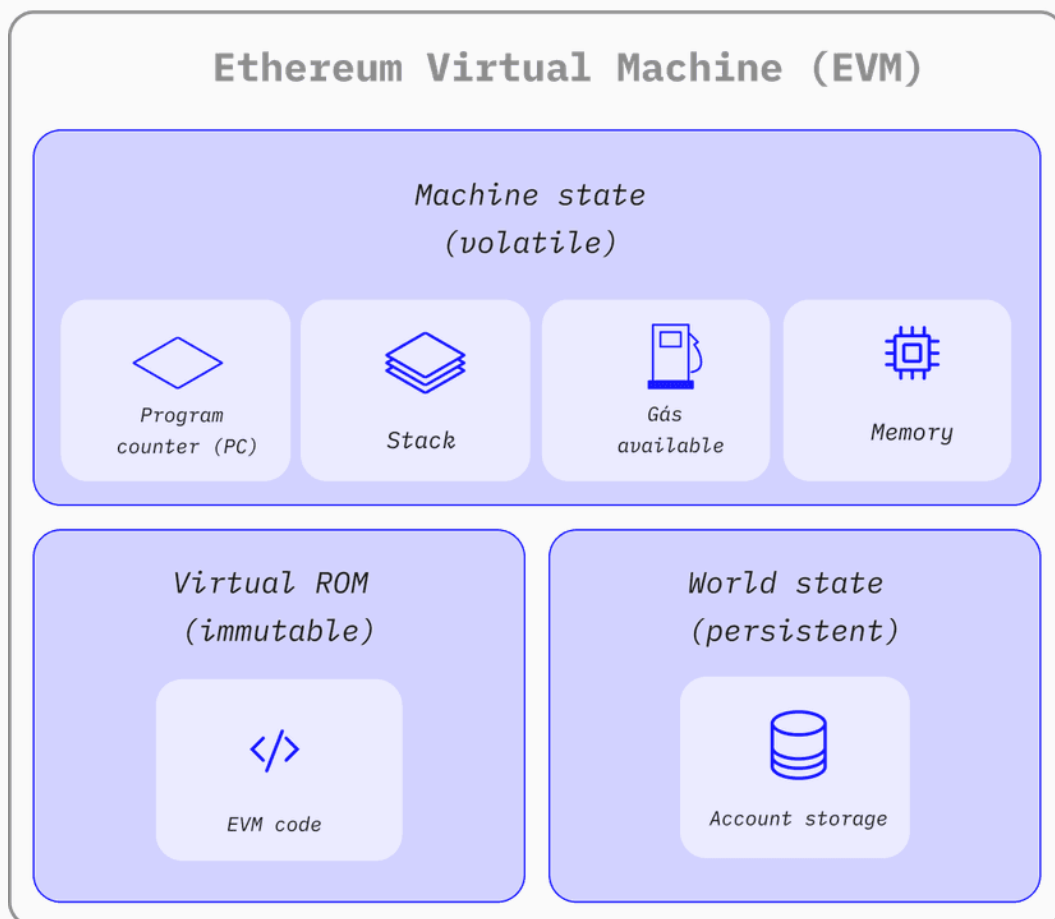


Figura 2.9: Diagrama de la EVM [25]

Funcion de transicion de estado

La EVM se comporta como una función matemática: dada una entrada, esta produce una salida determinista. Por tanto, es bastante útil para describir formalmente a Ethereum como una función de transición de estado: $Y(S, T) = S'$.

Dado un estado válido anterior (S) y un nuevo conjunto de transacciones válidas (T), la función de transición de estado de Ethereum $Y(S, T)$ produce un nuevo estado de salida válido S' .

Transacciones y EVM. Las transacciones son instrucciones firmadas criptográficamente desde las cuentas. Hay dos tipos de transacciones: aquellas que resultan de llamadas de mensajes y aquellas que resultan de la creación de contratos.

La creación de contratos da lugar a la creación de una nueva cuenta de contrato, que contiene el bytecode compilado del contrato inteligente. Además contienen un Merkle Patricia trie de almacenamiento (como una matriz de palabras con direccionamiento por palabra), asociado a la cuenta en cuestión y que forma parte del estado global. Cada vez que otra cuenta realiza una llamada de mensaje al contrato, este ejecuta su bytecode [25].

Instrucciones de la EVM

La EVM se ejecuta como una máquina de pila con una profundidad de 1024 ítems. Cada ítem es una palabra de 256 bits, que se selecciona para utilizar fácilmente con la criptografía de 256 bits (como los hashes Keccak 256) [26].

2.4.5 Gas

El gas hace referencia a la unidad que mide la cantidad de esfuerzo computacional requerida para ejecutar operaciones específicas en la red de Ethereum.

Dado que la ejecución de cada transacción en Ethereum requiere recursos informáticos, dichos recursos tienen que ser de pago para así poder asegurar que Ethereum no sea vulnerable al spam y no se pueda quedar atascado en un bucles infinitos de computación. El pago por este recurso computacional se realiza en forma de tarifa de gas.

La tarifa de gas es **la cantidad de gas usado para hacer alguna operación, multiplicado por el coste unitario del gas**. Al margen de que la transacción se procese de forma exitosa o fallida, se paga igualmente una tarifa.

Las comisiones de Gas se pagan en la moneda nativa de Ethereum, es decir, el ether (ETH). Los precios del gas están indicados en Gwei, que es una denominación de ETH, cada Gwei equivale a 0,000000001 ETH [27].

2.4.6 Contratos inteligentes

Según el sitio web oficial de Ethereum

“Los contratos inteligentes son los bloques de construcción fundamentales de la capa de aplicación de Ethereum. Son programas informáticos almacenados en la cadena de bloques que siguen la lógica «si ocurre esto, entonces se produce aquello» y garantizan ejecutarse siguiendo las reglas definidas por su código, que no se puede cambiar una vez creado.” [28]

Los contratos inteligentes en Ethereum funcionan como cuentas con saldo capaces de realizar transacciones en la red, pero a diferencia de las cuentas de usuario, no están bajo el control directo de un individuo. Estos contratos son implementados en la red y ejecutados según su programación. Los usuarios pueden interactuar con ellos enviando transacciones que activen funciones predefinidas. Además, los contratos inteligentes pueden establecer reglas y se ejecutan automáticamente a través de su código. Estos contratos son interoperables y públicos en Ethereum, lo que los convierte en APIs abiertas que pueden ser utilizadas y combinadas entre sí.

Además, los contratos inteligentes son útiles para auditorías y seguimiento, ya que toda la información relacionada con ellos está disponible en la cadena de bloques pública de Ethereum. Esto permite realizar un seguimiento transparente de las transferencias de activos y otros datos relevantes, lo que contribuye a una mayor transparencia y confianza en las transacciones.

Desarrollo de contratos inteligentes

Para los desarrolladores, es necesario aprender a programar en un lenguaje específico para contratos inteligentes, como Solidity [29] o Vyper [30], y contar con ETH suficiente para su implementación. Aunque implementar un contrato inteligente implica pagar tarifas de gas, los costos suelen ser más elevados que en las transacciones regulares.

Sin embargo, los contratos inteligentes tienen limitaciones, como la incapacidad de obtener información externa sin la ayuda de oráculos, y requieren una cuidadosa gestión de los datos, ya que su almacenamiento en la cadena de bloques puede resultar costoso.

Anatomía de un contrato inteligente

- **Datos:** Cualquier dato del contrato debe ser asignado a una ubicación: ya sea al almacenamiento o a la memoria. Es costoso modificar el almacenamiento en un contrato inteligente, por lo que es importante considerar dónde deben vivir tus datos.
- **Almacenamiento:** Los datos persistentes se denominan almacenamiento y están representados por variables de estado. Estos valores se almacenan permanentemente en el blockchain, cualquier acción que modifique éstos datos tiene un costo de gas asociado.

- Memoria: Los valores que sólo se almacenan durante la ejecución de una función de contrato se denominan variables de memoria. Como no se almacenan permanentemente en la cadena de bloques, su uso es mucho más barato.
- Variables de entorno: Además de las variables que defines en tu contrato, hay algunas variables globales especiales. Se utilizan principalmente para proporcionar información sobre el blockchain o la transacción actual.
- Funciones de visualización: Estas funciones prometen no modificar el estado de los datos del contrato. Ejemplos comunes son las funciones "getter" - se podría utilizar para recibir el saldo de un usuario, por ejemplo. [31]

Compilación de contratos

Como se ha mencionado con anterioridad, los contratos inteligentes son programas que son ejecutados por la máquina virtual de Ethereum, similar a los programas desarrollados en el lenguaje de programación Java pero de manera distribuida. La EVM posee un conjunto finito de operaciones con diferentes costos en unidades de Gas, tal como se observa en la Fig. 2.10.

Stack Name	Gas	Initial Stack
00 STOP	0	
01 ADD	3	a, b
02 MUL	5	a, b
03 SUB	3	a, b
04 DIV	5	a, b
05 SDIV	5	a, b
06 MOD	5	a, b
07 SMOD	5	a, b
08 ADDMOD	8	a, b, N
09 MULMOD	8	a, b, N
0A EXP	A1	a, b
0B SIGNEXTEND	5	b, x
0C- 0F		<i>invalid</i>
10 LT	3	a, b

Figura 2.10: Extracto de la tabla de referencia de los códigos de operación de la EVM [26]

Para que la EVM pueda ejecutar su contrato, debe estar en bytecode. La compilación convierte este código en Solidity:

```
pragma solidity 0.4.24;

contract Greeter {

    function greet() public constant returns (string) {
        return "Hola";
    }

}
```

A este bytecode, como se observa en la Figura 2.12

Figura 2.12: Programa en bytecode de la función antes mencionada [32]

El compilador además genera un archivo en formato JSON que contiene la especificación formal del contrato denominada ABI (Application Binary Interface). Sirve para que cualquier aplicación pueda interactuar con el contrato y ejecutar llamadas a funciones, ya sea entre contratos (contrato a contrato) o externas (aplicación web - contrato). Es muy importante ya que permite realizar un puente entre las aplicaciones que funcionan en web2 a la web3.

2.4.7 Protocollo JSON-RPC

Para este propósito, cada cliente de Ethereum implementa una especificación JSON-RPC para que haya un conjunto uniforme de métodos que las aplicaciones puedan usar, sin importar la implementación de nodo o cliente específica.

31

sentido de que los conceptos se pueden usar dentro del mismo proceso, por sockets, HTTP o distintos entornos de intercambio de mensajes. Utiliza JSON (RFC 4627) como formato de datos. [34]

2.4.8 Cadenas Laterales

Una cadena lateral (sidechain) es una cadena de bloques diferente que funciona de manera independiente de Ethereum, está conectada a la red principal de Ethereum por un puente bidireccional. Las mismas pueden tener distintos mecanismos de consenso y variaciones en los protocolos, por ejemplo: para mejorar el procesamiento de transacciones. Son útiles para darle soporte a la red principal y así lograr una mejora en el procesamiento de transacciones por segundo y disminuyendo considerablemente los costos de GAS, sin embargo, esto implica sacrificios en términos de descentralización o seguridad. [35]

Compatibilidad con la EVM

Existen cadenas laterales que son compatibles con la EVM, por lo tanto pueden ejecutar contratos de la red ethereum escritos en Solidity o en otros lenguajes, también son consideradas como una solución de escalabilidad útil para los contratos de Ethereum. Ésto contribuye a tarifas de gas menores, así como transacciones más rápidas, especialmente si la red principal se encontrara congestionada.

Polygon

Es una solución de escalabilidad de Ethereum que permite crear una red de blockchain dedicada que combine todas las mejores características de los blockchains independientes (flexibilidad, soberanía y escalabilidad) con lo mejor de Ethereum (seguridad e interoperabilidad). [36]

El token de Polygon se denomina MATIC y contribuye al funcionamiento de prueba de participación de la red Polygon que premia a quienes utilizan sus propios token MATIC para ayudar a validar las operaciones de la red.

Blockchain Federal Argentina

Blockchain Federal Argentina es una plataforma multiservicios abierta y participativa pensada para integrar servicios y aplicaciones sobre blockchain. Una iniciativa confiable y completamente auditable que permita optimizar procesos y funcione como herramienta de empoderamiento para toda la comunidad. Fue concebida dentro de un espacio de trabajo colaborativo, y apunta a reproducir ese patrón como columna vertebral de la plataforma.

Siguiendo el modelo de Múltiples Partes Interesadas, Blockchain Federal Argentina mantiene un modelo de gobernanza que asegura la representación de todos los sectores en la toma de decisiones. Pero al ser una plataforma pública, su uso no estará restringido a las organizaciones que participen del consorcio. Toda la comunidad tiene las puertas abiertas para participar en BFA.

BFA está basada en la tecnología Ethereum y funciona bajo Prueba de Trabajo, permite a cualquier desarrollador crear y ejecutar smart contracts garantizados por la cadena de bloques [37].

2.5 Web 3

Durante los primeros inicios de lo que hoy conocemos como internet, existía la web 1 donde la navegación consistía en páginas estáticas que contenían texto, algunas imágenes y links a otras páginas, la interacción con el usuario era mínima, sólo se podía leer la información que provenía del emisor, quien era el dueño de la información.

Con el tiempo esto empezó a cambiar logrando evolucionar a la web 2, los usuarios podían enviar y recibir información, la forma de interactuar con la web comenzó a ser más dinámica y bidireccional, dicha interacción creó grandes cantidades de información y contenido que fueron migrando hacia ambientes más centralizados en la que una gran parte de la comunicación y el comercio tiene lugar en plataformas cerradas.

Web 3 promete romper con esa barrera de la centralización devolviendo la propiedad de la información a los usuarios, a través de protocolos descentralizados y el intercambio de tokens las unidades digitales de intercambio de tecnologías blockchain.[38]

Limitaciones de Web 3

- Escalabilidad: Las transacciones son más lentas en web3 porque son descentralizadas. Los cambios a estado, como un pago, deben ser procesados por un minero y propagados a través de la red.
- UX: Interactuar con aplicaciones web3 puede requerir pasos adicionales, software y formación. Esto puede ser un obstáculo para su implantación.
- Accesibilidad: la falta de integración en los navegadores web modernos hace que web3 sea menos accesible para la mayoría de los usuarios.
- Coste: las dapps más exitosas introducen partes muy pequeñas de su código en la cadena de bloques, ya que resulta costoso.

2.6 Aplicaciones descentralizadas Dapps

Una aplicación descentralizada (DApp) es una aplicación diseñada en una red descentralizada, que combina un contrato inteligente y una interfaz de usuario frontal, el contrato inteligente, como se ha mencionado previamente, es un código autoejecutable que reside en una blockchain, la interfaz de usuario frontal de una DApp es la capa visible con la que interactúan los usuarios finales. Esta interfaz puede adoptar diversas formas, como una aplicación web, una aplicación móvil o incluso una interfaz de línea de comandos [39].

Arquitectura de aplicaciones descentralizadas

- Completamente descentralizadas: estos tipos de aplicaciones poseen toda la lógica de la aplicación en la red blockchain a través de contratos inteligentes, por lo que todas las funcionalidades de la aplicación son proveídas por los contratos, además existe una interfaz de usuario (frontend) donde el usuario interactúa directamente con los contratos.
- Semi-descentralizadas: en la actualidad, la capacidad computacional y el almacenamiento de las redes blockchain son algo limitadas, por lo que se utilizan sistemas tradicionales (backend) para manejar aquellas funcionalidades de la aplicación donde se requiera mayor performance y escalabilidad. El grado de descentralización depende de qué tantas funcionalidades serán gestionadas desde blockchain.

En el capítulo 5 se realizará una explicación en detalle de la arquitectura utilizada en el proyecto junto con las ventajas y desventajas.

2.7 Modelo de certificación de títulos según BFA

Según la información encontrada en la web de Blockchain Federal Argentina:

Se estableció un caso de uso con la certificación de títulos universitarios explicando el proceso actual de certificación y cómo esta blockchain puede contribuir a esta problemática de una manera más transparente y auditable.

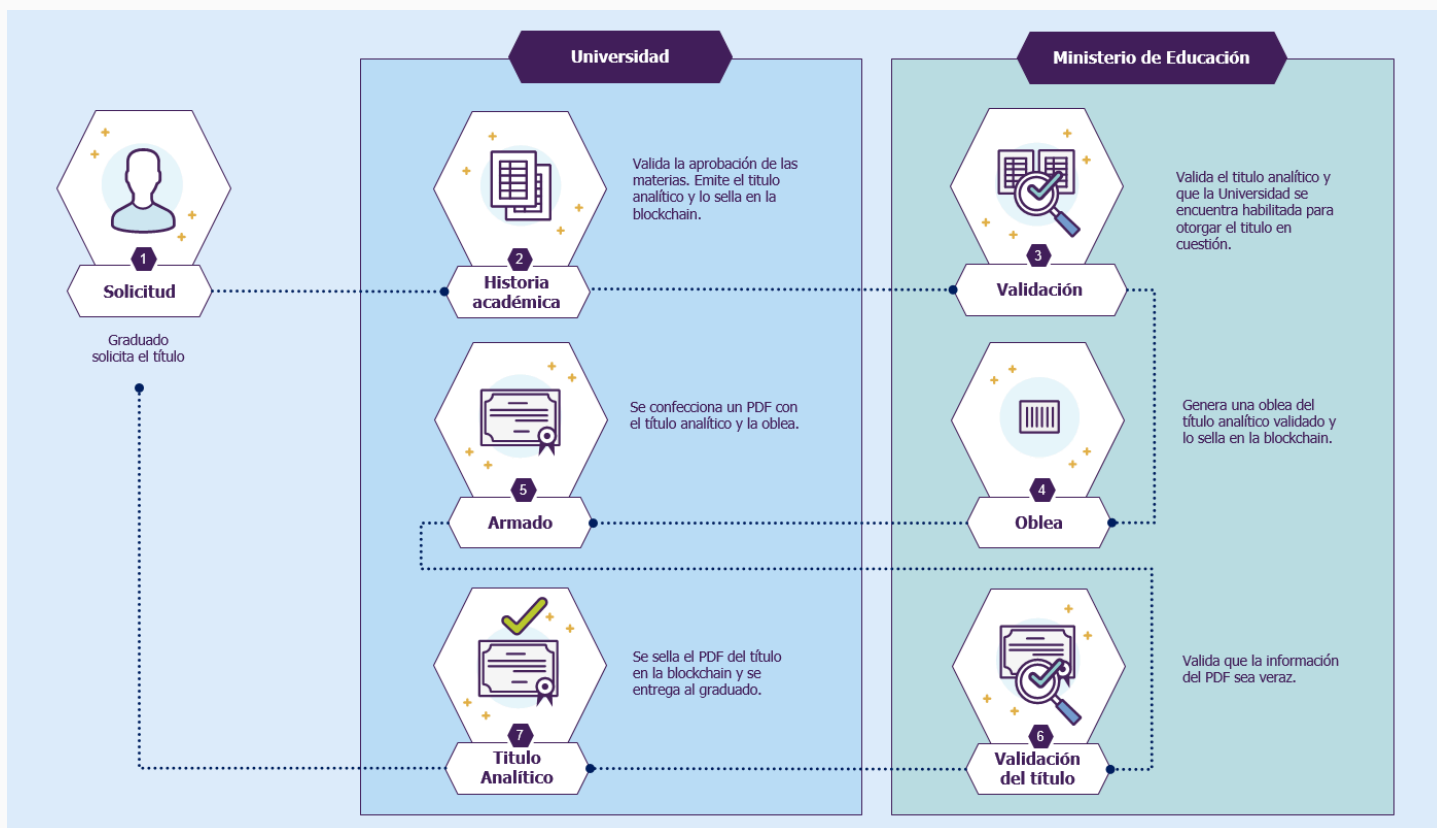


Figura 2.13: Modelo de proceso de certificación de BFA [40]

“En la actualidad, los modelos de digitalización para procesos de certificación de títulos o actas no proveen un mecanismo para garantizar la integridad de los datos, sobre todo en un marco donde los títulos analíticos circulan entre diversas entidades durante todo el ciclo de tramitación. Al mismo tiempo este modelo acarrea la necesidad de legalizar copias de cada título cada vez que es necesario, y genera que los diplomas sean fácilmente falsificables.” elaboraron el siguiente diagrama mostrando el proceso de certificación donde se puede ver la aplicación de blockchain en el paso 7.

También agregan una expansión del modelo integrándolo con un portadocumentos digital:

“Al complejizar la solución, se puede apuntar a que las copias digitales de los títulos académicos sean fácilmente verificables, o incluso integrarlas en un Portadocumentos digital para que los titulares de esos documentos puedan otorgar y quitar permisos de acceso” [40]

Desafortunadamente es una propuesta de posible caso de uso y carece de especificidad y detalles de implementación.

BFA vs UNSL-Cert

- En este proyecto se hará uso ilustrativo del modelo planteado en BFA puesto que refleja el procedimiento de certificación actual. UNSL-Cert implementaría el paso 7 almacenando la

información de títulos en blockchain e interacción con el usuario.

- El modelo no especifica cómo almacenaría los datos en blockchain y qué se le dará al graduado.
- El modelo no especifica cómo el graduado puede validar y verificar la autenticidad del pdf entregado por BFA.
- No hay garantías de que la información almacenada del usuario en blockchain pueda ser eliminada o “dada de baja” por el usuario si éste quisiera.
- El modelo planteado por BFA es un posible caso de uso y carece de detalles de implementación, en cambio, UNSL-Cert es una aplicación implementa los aspectos relevantes y permite verificar los títulos universitarios almacenados.

Capítulo 3: Soporte de Tecnologías

En este capítulo se describirán las tecnologías utilizadas para la implementación, considerando aquellas necesarias para el desarrollo web cómo las vinculadas a Blockchain. Los frameworks utilizados cuentan con amplia popularidad y madurez en la industria del software, razón por la cual fueron seleccionados para el proyecto. Cabe destacar, que la innovación se ve reflejada en la tecnología blockchain de Ethereum y la implementación de un contrato inteligente.

3.1 Desarrollo Web

Typescript (Lenguaje de programación)

Es una alternativa a Javascript [41], cuenta con ventajas como la compatibilidad directa con éste y la inclusión de tipos que le brinda más robustez y seguridad, también garantiza aplicaciones menos propensas a errores [42].

Angular (Front-end)

Es un framework de desarrollo frontend que permite crear interfaces de usuario de una manera estructurada, escalable y de calidad. Posee una estructura de desarrollo muy marcada y restrictiva, esto ayuda a realizar aplicaciones con interfaces de usuario amigables para el usuario, performantes [43].

NodeJs (Back-end)

Es un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node.js está diseñado para crear aplicaciones network escalables. Frecuentemente se utiliza para desarrollar aplicaciones del lado del servidor. Por otro lado, posee soporte para Typescript [44].

ExpressJs (Back-end)

Es un framework de Javascript para el desarrollo de APIs en NodeJs, se caracteriza principalmente por poder desarrollar una API de manera rápida, sencilla y flexible. Además, tiene un vasto soporte por la comunidad [45].

Sequelize (Back-end)

Es una herramienta para NodeJs que sirve como ORM (Object Relational Mapping) que es una técnica de programación para convertir datos entre el sistema de tipos en Typescript y la utilización de una base de datos relacional. Esto permite interactuar con la base de datos a través de las clases definidas en Typescript sin necesidad de utilizar el lenguaje para bases de datos relacionales SQL, además facilita el desarrollo de consultas y posibles errores de “inyección SQL” [46].

PostgreSQL (Base de datos)

Es un sistema de base de datos relacional de objetos de código abierto que utiliza y amplía el lenguaje SQL combinado con características que almacenan y escalan de forma segura las cargas de trabajo de datos más complicadas. Los orígenes de PostgreSQL se remontan a 1986 como parte del proyecto POSTGRES de la Universidad de California en Berkeley y cuenta con más de 35 años de desarrollo activo en la plataforma central [47].

WebSockets

Es una librería para el manejo de conexiones punto a punto entre sistemas web, en este caso backend y frontend. Permite un canal de comunicación entre ambos sistemas en tiempo real a través de sockets, de ésta manera ante algún evento producido por el backend como una transacción exitosa puede ser comunicado en tiempo real al frontend y generar una alerta informando dicho evento.[48]

3.2 Blockchain

Solidity (Lenguaje de programación)

Es uno de los lenguajes utilizados en el desarrollo de contratos inteligentes. En particular, es el lenguaje utilizado dentro de la red de Ethereum. Solidity es un lenguaje diseñado para apuntar a la máquina virtual Ethereum (EVM). Está influenciado por C, Python y JavaScript.

Truffle

Es un framework de desarrollo blockchain desarrollado para NodeJs, provee un conjunto de funcionalidades que permite compilar contratos inteligentes escritos en Solidity, realizar testing y desplegar los mismos.

A través de un script de implementación provisto por Truffle, se realiza una conexión con una API del servicio Informa. Éste posee un nodo de Ethereum en diversas redes, como la principal (Main Net) y otras de Test (Testnet) [49].

Ganache

Se utiliza para la creación del ambiente de pruebas. Permite la creación de una cadena de bloques de prueba montada sobre un host local y permite interactuar con ella rápidamente y monitorear los movimientos en la cadena de una manera amigable y sencilla [50].

Web3js

Es una librería de javascript desarrollada para NodeJs que posee cientos de funcionalidades para interactuar con las redes blockchain de Ethereum a través del protocolo JSON-RPC, permite ejecutar

contratos inteligentes, generar transacciones, interactuar con wallets, etc. Es una pieza fundamental para la interoperabilidad entre el desarrollo web y la blockchain [51].

Infura

Es un proveedor de infraestructura como servicio (IaaS) que permite tener un nodo de la red Ethereum de forma privada para poder desplegar contratos, en su versión gratuita ofrece la posibilidad de montar un nodo con 100,000 peticiones por día lo que es más que suficiente para realizar las pruebas necesarias y futuros despliegues para este proyecto.

Para el proceso de despliegue es necesario conseguir un lugar donde alojarlo. Existen diversas maneras, una es ejecutar un nodo en local (ejecutando el programa GETH), otra es conseguir nodos de Ethereum públicos o un proveedor de servicios como Infura [52].

3.3 Patrones de Diseño

Patrones de Diseño Generales

Los patrones de diseño permiten aplicar diseños probados o catálogos de elementos reusables para el desarrollo de sistemas de software de alta calidad. A continuación se detallan los patrones aplicados en el proyecto a nivel general.

Observer

Define una dependencia entre un objeto denominado Sujeto y múltiples otros objetos denominados observadores. Entonces cuando el sujeto cambia de estado, debe notificar a todos sus observadores dicho cambio para que se actualicen sus estados [53].

En este proyecto se ha implementado de manera indirecta a lo largo de todo el desarrollo del frontend a través del framework Angular. Un ejemplo de ello es el servicio de autenticación del frontend que permite compartir los datos de la sesión a los diferentes componentes de la aplicación, se utiliza este patrón para que cuando haya un cambio en los datos de sesión, sean comunicados a los componentes suscriptos.

En la figura 3.1 se visualiza el *authenticationService* que tiene un objeto *_user* de tipo *BehaviorSubject* que es la implementación del observable, el método *getCurrentUser* retorna un Observable para que cualquier componente pueda subscribirse.

```

11 export class AuthService {
12   private _user: BehaviorSubject<UserDto | null> = new BehaviorSubject<UserDto | null>(null);
13
14   getCurrentUser(): Observable<UserDto | null> {
15     return this._user.asObservable();
16   }
17
18   setCurrentUser(value: UserDto | null) {
19     this._user.next(value);
20   }
21 }

```

Figura 3.2: Código de parte del servicio de autenticación

Una vez suscrito, cada vez que haya un cambio en los datos del usuario, se ejecutará la lógica dentro del método *subscribe* como es el caso de la asignación de la figura 15.

```

47 initSubscriptionSelectStudent() {
48   this.subscription = this.authService.getCurrentUser().subscribe((user) => {
49     this.user = user;
50     | You, 1 second ago • Uncommitted changes
51   });
52 }

```

Figura 3.3:Método de suscripción al observable

Singleton

Es un patrón de diseño que restringe la creación de instancias de una clase a una sola y asegura de que exista un único punto de acceso a esa instancia única. En este proyecto éste patrón fue muy útil para la creación del servicio de notificaciones del backend que posee un servidor websocket para enviar notificaciones al frontend.

En la figura 3.4 se puede visualizar la estructura del servicio. La propiedad estática *instance* indica la instancia del servicio, el método *Instance()* retorna *Instance* en el caso de que ya haya sido creada, de lo contrario, se crea una nueva instancia de *NotificationService* y se asigna a *NotificationService.instance*. Esto garantiza que solo se cree una instancia de la clase y que esa instancia se almacene en la propiedad estática *instance*. Finalmente en la línea 56 se encuentra la invocación del servicio.


```

6  class NotificationService {
7      private _webSocketInstance: WebSocketServer;
8      private _port = 9090;
9
10     get webSocketInstance(): WebSocket.WebSocketServer {
11         return this._webSocketInstance;
12     }
13
14     get port(): number {
15         return this._port;
16     }
17
18     // Instancia única del servicio.
19     private static instance: NotificationService;
20
21     // Método que retorna la instancia.
22     public static get Instance(): NotificationService {
23         return (
24             NotificationService.instance ||
25             (NotificationService.instance = new NotificationService())
26         );
27     }
28
29     constructor() {
30         // Inicializa el servidor WebSocket.
31         this._webSocketInstance = new WebSocket.Server({ port: this._port });
32     }
33
34     > connect() { ...
35     }
36
37     > sendNotification(clientId: number, message: NotificationDto) { ...
38     }
39 }
40
41 You, 16 months ago • se agrega servicio de notificaciones
42
43 // Retorna la instancia.
44 export const notificationService = NotificationService.Instance;

```

Figura 3.4: Patrón singleton en servicio de notificaciones.

Data Transfer Object (DTO)

Éste patrón se utiliza frecuentemente en la comunicación entre sistemas web para reducir el costo de las llamadas entre el cliente y el servidor.

El patrón DTO tiene como finalidad reducir la cantidad de llamadas al servidor mediante la creación de objetos planos con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarse en una única clase simple [54].

En las figuras 3.5, 3.6 3.7 se muestra la implementación para obtener los datos de los estudiantes, creando las clases *PersonDto* que posee la información de la entidad *Person* y *StudentDto* que unifica información dispersa en *Person*, *Student* y *Degree* cómo se puede visualizar en las figuras siguientes.

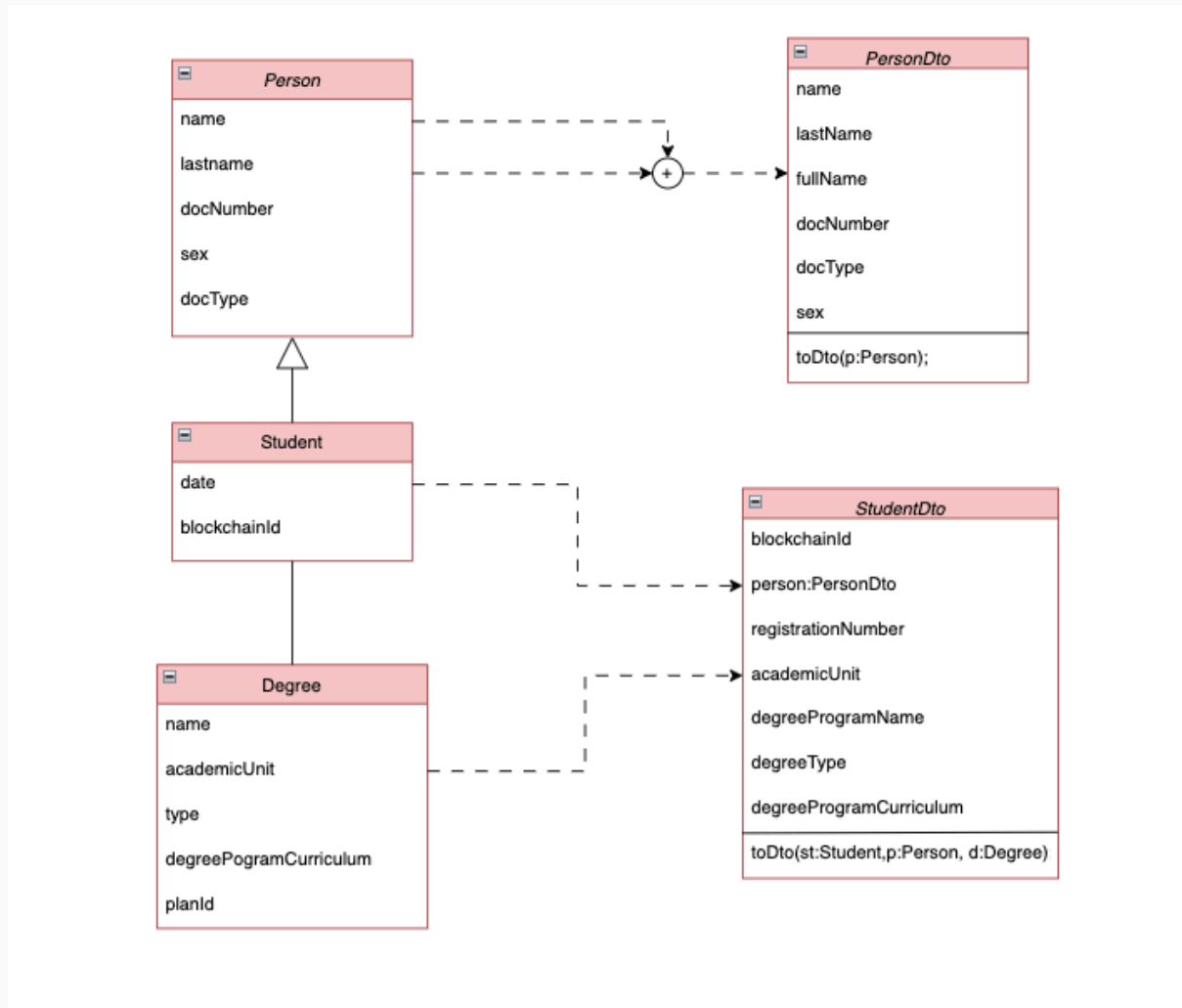


Figura 3.5: Diagrama de clases DTOs

```
export interface PersonDto {
  id: number;
  name: string;
  lastname: string;
  fullname?: string;
  docNumber: string;
  docType: string;
  sex: string;
}
```

Figura 3.6: Código de PersonDto

```
export interface StudentDto {
  id: number;
  blockchainId: number;
  person: PersonDto;
  registrationNumber: number;
  academicUnit: string; // Facultad
  degreeProgramName: string; // Nombre de la carrera
  degreeType: string; // Tipo de carrera.
  degreeProgramCurriculum: string; // Plan de estudios.
}
```

Figura 3.7: Código de StudentDto

Una característica fundamental de éste patrón es que la clase DTO no debe tener comportamiento de lógica de negocio, solo debe poseer información que pueda ser expuesta a otro servicio y métodos de conversión, como es el caso del método *toDo* de la figura 3.8.

```
toDto(st:Student, p:Person, d:Degree): StudentDto {
  return {
    id: st.id,
    universityName: d.university,
    academicUnit: d.academicUnit,
    degreeProgramCurriculum: d.planId,
    degreeProgramName: d.name,
    blockchainId: st.blockchainId ? Number(st.blockchainId) : null,
    degreeType: d.type,
    registrationNumber: st.registrationNumber,
    person: p ? Person.toDto(p) : null
  } as StudentDto;
}
```

Figura 3.8: Función toDto

Patrones para Contratos Inteligentes

En la programación de los contratos inteligentes, también es necesario evitar malas prácticas y el mal diseño ya que pueden provocar fallos en la seguridad y en costos excesivos de Gas para quién ejecuta el contrato. Por esta razón, existen diversos patrones de diseño de contratos inteligentes que intentan resolver problemáticas de dicha índole. En el presente trabajo se aplica el patrón detallado a continuación.

Ownership o Restricción de Acceso

Dado que los contratos inteligentes de Ethereum son de código abierto no es posible restringir a personas u ordenadores la lectura del contenido de las transacciones o el estado del contrato.

Se puede restringir el acceso de lectura al estado del contrato por parte de otros contratos. Esto se hace por defecto, a menos que se declaren variables de estado públicas. Además, se puede restringir quién puede hacer modificaciones al estado del contrato o llamar a las funciones del contrato. El uso de modificadores de función hace que estas restricciones sean altamente legibles [55].

Este patrón ayuda a implementar modificadores de acceso para los datos y funciones del contrato, similar a los mecanismos de roles y permisos de los sistemas de autenticación tradicionales. Inicialmente se designa owner la dirección de la cuenta creadora del contrato, luego se permiten mecanismos para cambiar de dueño y control de acceso como se puede ver en la figura 3.9.

```
4  abstract contract Ownable {
5
6      address private _owner;
7
8      constructor() {
9          _owner = msg.sender;
10     }
11
12     function owner() public view virtual returns (address) {
13         return _owner;
14     }
15
16     modifier onlyOwner() {
17         require(owner() == msg.sender, "El llamador a esta funcion no es el owner del contrato. No se puede ejecutar.");
18         _;
19     }
20
21     function transferOwnership(address newOwner) public virtual onlyOwner {
22         _owner = newOwner;
23     }
24 }
```

Figura 3.9: Estructura del patrón Restricción de Acceso en Solidity.

3.4 Despliegue de Backend y Frontend con Docker y AWS

En este proyecto, realizamos el despliegue de un backend y un frontend utilizando contenedores Docker [56] y servicios de Amazon Web Services (AWS) EC2 [57] para el backend y S3 [58] para el frontend.

Despliegue del Backend con Docker y AWS EC2

1. **Dockerización del Backend:** El primer paso es “dockerizar” el backend de la aplicación. Docker permite empaquetar la aplicación y sus dependencias en un contenedor ligero y portátil, lo que garantiza que se ejecute de manera consistente en cualquier entorno.
2. **Creación de la Imagen Docker:** Se crea un archivo Dockerfile que especifica cómo se construye la imagen del contenedor. Este archivo incluirá las instrucciones para instalar las dependencias, bases de datos, compilar el código y configurar el entorno de ejecución.
3. **Despliegue en AWS EC2:** Se lanza una instancia de EC2 en AWS, que servirá como el servidor para el backend de la aplicación. Durante la configuración de la instancia, se puede especificar la imagen del contenedor Docker que se ejecutará en ella.

Despliegue del Frontend con Docker y AWS S3

1. **Compilación de Front-end:** Angular compila el código y convierte en archivos HTML, CSS y Javascript optimizados.
2. **Despliegue en AWS S3:** AWS S3 se utiliza para el almacenamiento y la entrega de contenido estático, se cargan los archivos compilados del frontend y se publican en un dominio público de AWS.

Capítulo 4: Planificación y Costos

En este capítulo se hará una descripción de cuáles serán las tareas necesarias para la realización del sistema planteado en este proyecto junto con un diagrama de Gantt que expone el tiempo estimado para resolver cada tarea. De manera seguida se expondrán los costos de desarrollo, despliegue y operativos del sistema y finalmente se realizará un análisis de inversión dando a conocer un posible modelo de negocio con una futura rentabilidad.

4.1 Planificación de actividades

TAREAS	MES 1				MES 2				MES 3				MES 4				MES 5				MES 6			
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24
Análisis de tecnologías																								
Análisis y diseño																								
Desarrollo del contrato inteligente																								
Desarrollo del sistema web																								
Creación e integración de base de datos																								
Integración de blockchain y plataforma web																								
Despliegue de los sistemas																								
Prueba																								
Correcciones y despliegue																								

Figura 4.1: Diagrama de Gantt con planificación de tareas

Descripción de tareas:

- **Tarea 1 - Análisis de tecnologías.** Analizar las tecnologías utilizadas en la industria para el desarrollo de aplicaciones que permitan la interoperabilidad con la tecnología blockchain y que cuenten el soporte necesario para afrontar los objetivos propuestos.
- **Tarea 2 - Análisis y diseño.** Contempla la elaboración de modelo de casos de uso, modelo conceptual, diseño de arquitectura y modelo de componentes. Debido al reciente surgimiento de aplicaciones con tecnología blockchain, esta tarea contempla la investigación de las diferentes arquitecturas para una correcta integración y buen aprovechamiento de las ventajas de ésta tecnología.
- **Tarea 3 - Desarrollo de contrato inteligente.** Contempla la implementación de un contrato inteligente, el cual será codificado en Solidity, uno de los lenguajes más frecuentemente utilizados para la implementación de contratos en Blockchain. En el contrato se almacenará la información de los títulos universitarios de los graduados.
- **Tarea 4 - Desarrollo del sistema web.** Contempla el desarrollo de una plataforma web donde interactuarán Graduados, Administrativos y cualquier Tercero o usuario final. Para ésto, se prevé usar Angular para el front-end, NodeJs y Express para el desarrollo back-end y Sequelize junto con Postgre para la integración de la base de datos.

- **Tarea 5 - Creación e integración de base de datos.** Contempla la creación de una base de datos relacional y la integración con la plataforma, se realiza a través de la técnica ORM (Object Relational Mapping) que es utilizada en el desarrollo de la plataforma web.
- **Tarea 6 - Integración de Blockchain a sistema web.** Contempla la integración del contrato inteligente con la plataforma web utilizando el framework Truffle.
- **Tarea 7 - Despliegue de los sistemas.** Contempla la utilización de la plataforma proveedora de servicios Blockchain Infura para desplegar el contrato inteligente.
- **Tareas 8 - Prueba.** Contempla los test unitarios realizados sobre la funcionalidad del contrato inteligente y la validación de los requerimientos funcionales a través de casos que permitan probar los flujos de ejecución más importantes de la aplicación.
- **Tarea 9 - Correcciones y despliegue.** Contempla la resolución de posibles errores en el funcionamiento y ajustes de parámetros de configuración para el despliegue del contrato inteligente como “costo promedio de gas”.

4.2 Costos

Para realizar este proyecto integrador es necesario contratar personal capacitado para poder desarrollar el mismo. Vamos a suponer que será realizado por un profesional trabajando aproximadamente 20 horas semanales, a esto lo denominamos *costo de desarrollo*. Una vez finalizado el desarrollo, es necesario publicar la aplicación en internet para que el público pueda utilizarla, esto llevo un costo denominado *costo de despliegue*, además se debe desplegar el contrato inteligente en una blockchain compatible con Ethereum que también posee un costo adicional. Finalmente, la utilización de los mismos tienen un costo uso que denominamos *costos operativos*.

4.2.1 Costo de desarrollo

Recursos humanos

El costo de construcción del sistema web se calcula multiplicando el pago por hora del programador por el número total de horas estimadas para el desarrollo del proyecto. Cabe destacar, que en el análisis se omite distinguir entre los diferentes roles específicos que puede estar implicado un profesional informático, tales como analista, arquitecto o tester. Para facilitar los cálculos se asume el costo por hora del pago correspondiente a un programador de aplicaciones junior de acuerdo al último registro del Colegio Profesional de Ciencias Informáticas de la Provincia de Buenos Aires [59].

En este caso, el programador trabajará aproximadamente 20 horas a la semana a lo largo del tiempo estimado, con un costo por hora de 15 USD. El cálculo del costo de construcción del sistema web se haría de la siguiente manera:

$$\text{Costo de construcción} = \text{precio por hora} * \text{horas semanales trabajadas} * \text{cantidad de semanas}$$

Si, de acuerdo a la planificación, el tiempo en semanas es 24, el costo sería:

$$\text{Costo de construcción} = \$15 \text{ USD/hs} * 20 \text{ hs} * 24 = \$7.200 \text{ USD}$$

Este costo de construcción incluye los honorarios del programador por su trabajo durante el desarrollo del sistema web. Es importante tener en cuenta que este cálculo solo considera el costo directo del trabajo del programador y no incluye otros posibles costos asociados, como el despliegue de la aplicación, el mantenimiento posterior al lanzamiento, o cualquier otra inversión adicional que pueda ser necesaria para el proyecto.

Licencias

En este desarrollo se utilizaron herramientas y frameworks de código abierto por lo que no se incurrieron en costos de licencias ni herramientas de terceros.

4.2.2 Costo de despliegue

Frontend y Backend en AWS:

AWS tiene servicios de instancias de EC2 para el backend y S3 para el frontend para desplegar la aplicación web del proyecto, el costo de estos servicios dependerá del uso, como la cantidad de instancias, el almacenamiento utilizado, el tráfico de red, entre otros.

Utilizando la calculadora de costos provisto por AWS [60] se obtiene una estimación de un costo 7 USD por mes para el uso de EC2 y 1 USD mensual para S3, cómo se visualizan en las figuras 4.2 y 4.3.

☒ **Compute Savings Plans**
 Un plan que se aplica automáticamente a todo el uso de EC2, Fargate y Lambda. Hasta un 66 % de descuento. [Obtenga más información](#)

Plazo de reserva
☐ 1 year
☒ 3 year

Opciones de pago
☒ Sin pagos iniciales
☐ Pago inicial parcial
☐ Pago inicial total

Inicial: 0.00
Mensual: 6.79/Mes

Figura 4.2 Costo mensual en USD de EC2 según calculadora de AWS

Coste de S3 Estándar (Mensual): 0.88
Costo inicial total: 0.00 USD
Costo total mensual: 0.88 USD

Figura 4.3: Costo mensual en USD de S3 según calculadora de AWS

Despliegue del contrato en la red Ethereum

Desplegar un contrato inteligente en la red Ethereum tiene un costo asociado que es pagado por el usuario autor del mismo con dinero en su cuenta de Ethereum, esto debe hacerse en cada despliegue.

Se puede calcular los costos de la siguiente manera:

$$\text{Costo} = \text{Gas utilizado} * \text{Precio del gas}$$

El *gas utilizado* es la cantidad de unidades de gas usadas por la red para ejecutar la transacción que crea el contrato inteligente, suele ser estable ya que depende de la complejidad del mismo. El contrato utilizado en este proyecto posee un costo total de despliegue de 2.600.000 unidades de gas.

El *precio del gas* depende de las condiciones actuales de la red (como la congestión), está expresado en GWEI (equivalente a 1×10^{-9} ETH) (cap. 2.4.5).

La red Mainnet de Ethereum es una de las principales opciones al momento de desplegar un contrato inteligente ya que es una de las redes más grandes y populares del mundo, desafortunadamente los costos de gas son excesivamente elevados por lo que se decide utilizar una cadena lateral que permite una mayor escalabilidad con menores costos (ver capítulo 2.4.8) una iniciativa pública y gratuita es la red BFA sin embargo para fines ilustrativos utilizaremos la red Polygon, en ésta red el gas se paga en MATIC, el token de la red, por lo tanto el costo se calcula de la siguiente manera:

$$\text{Costo} = \text{Gas utilizado} * \text{Precio del gas en MATIC}$$

Teniendo en cuenta que el precio del gas promedio en Polygon es de 150 GWEI [61] (15×10^{-7} ETH) y realizando una conversión a USD teniendo en cuenta el valor de mercado de cada MATIC en febrero de 2024 (aprox. 0,845 USD) obtenemos el siguiente resultado.

$$\text{Costo de despliegue} = 2.600.000 * (150 / 100000000) * 0,9 \approx 3,5 \text{ USD}$$

4.2.3 Costos operativos

Costo de utilización de contrato inteligente

Almacenar datos de un título en el contrato inteligente requiere recursos computacionales de manera distribuida que tienen un costo de gas, éste es menor al de despliegue mencionado anteriormente ya que realizan menos operaciones que desplegar un contrato. La fórmula para el cálculo del costo es la misma que en el despliegue pero se necesitan muchas menos unidades de gas para ejecutar una función, sin embargo, esto depende de la complejidad que le haya dado el programador.

Las operaciones de almacenamiento del contrato de este proyecto poseen en el peor de los casos 600.000 unidades de gas de costo por operación, por lo tanto si utilizamos la red de Polygon obtenemos el siguiente resultado.

$$\text{Costo operacional} = 600.000 * (150 / 100000000) * 0,9 \approx 0,8 \text{ USD}$$

Cabe destacar que el costo de las operaciones pueden ir variando dependiendo de la lógica con la que fueron programadas, aquellas funciones más simples costarán menos gas que otras, en el capítulo 5 se detallarán la implementación de las mismas.

4.3 Análisis de inversión

Desarrollo del Producto:

- Honorarios de programador junior: \$7.200 USD

Despliegue del Backend y Frontend:

- Costo estimado del despliegue en AWS: \$8 USD por mes
- Costo de despliegue del contrato inteligente en la red Polygon: 3,5 USD aprox.

Costos Totales:

- Inversión inicial: \$7.248,5 USD

Ingresos:

- Un solo pago de \$9000 USD a partir del mes 7.

Análisis de ROI:

ROI (Retorno de inversión) es una métrica financiera utilizada para evaluar la rentabilidad de una inversión. Se calcula dividiendo la ganancia neta de la inversión entre el costo inicial de la inversión y luego multiplicando el resultado por 100 para expresarlo como un porcentaje. Si la ROI es positiva, indica que los beneficios superan los costos, lo que significa que el proyecto es rentable.

$$ROI = ((Ingresos - Costos) / Costos) * 100$$

Con los datos mencionados anteriormente la métrica queda de la siguiente manera:

$$ROI = ((\$9000 USD - \$7.248,5 USD) / \$7.248,5 USD) * 100 = 24,16\%$$

Valor Actual Neto (VAN):

La VAN (Valor Actual Neto) es una medida financiera utilizada para determinar la rentabilidad de una inversión al calcular el valor actual de los ingresos futuros generados por el proyecto, descontando la inversión inicial. Si la VAN es positiva, indica que el proyecto tiene un valor presente neto positivo y puede considerarse rentable. Una VAN positiva significa que el proyecto generará más valor del que cuesta implementarlo.

$$VAN = Ingresos futuros - Inversión inicial$$

$$VAN = \$9000 USD - \$7.248,5 USD = \$1.751,5 USD$$

Capítulo 5: Sistema de Certificación Digital de Títulos Universitarios en Blockchain

En este capítulo se detallarán las distintas etapas del desarrollo del sistema de certificación digital de títulos universitarios en blockchain UNSL-CERT. En la etapa de análisis se describe el problema a resolver junto con las propuestas de solución, adjuntando los modelos de casos de uso con las funcionalidades y el modelo conceptual. En la etapa de diseño se mostrará la arquitectura del sistema, considerando las particularidades del uso de blockchain como tecnología soporte. En la etapa de implementación se detallarán las tecnologías aplicadas en el proyecto y cómo se comunican entre sí, además se detalla la estructura, funcionamiento y despliegue del contrato inteligente junto con diagramas de secuencia y ejemplos en código. Finalmente se exponen algunas problemáticas asociadas a la implementación propuesta junto con soluciones.

5.1 Análisis

Una vez concluidos los estudios, la universidad se encarga de emitir un certificado que sirva como comprobación y dé fe de que la persona cuyos datos aparecen en él ha cumplido con el plan de estudios. Por lo tanto, posee los conocimientos y aptitudes necesarios para desempeñar el puesto de trabajo descrito en dicho certificado. Contar con un título académico certifica los conocimientos adquiridos durante una carrera universitaria. Poseer un título universitario garantiza la validez del documento en el entorno académico y proporciona la oportunidad de seguir avanzando en la formación profesional.

5.1.1 Procesos de certificación actuales

Los títulos universitarios son certificados con alto grado de importancia a nivel institucional para una entidad educativa como una universidad. Por lo tanto, emitir un certificado de este calibre no es una tarea sencilla ni fácil de auditar, está sujeto a rigurosos procedimientos de validación que recorren todas las jerarquías de autoridades en la institución hasta culminar en las autoridades ministeriales nacionales que realizan una completa revisión de todo el trayecto estudiantil de la persona.

El presente proyecto se enfoca en analizar y dar solución a las necesidades de la Universidad Nacional de San Luis (UNSL). Para establecer dichas necesidades se llevaron a cabo entrevistas con administrativos del área de Diplomas con el propósito de determinar cuáles son los procesos de negocio intervinientes.

En la UNSL la emisión de un título universitario inicia una vez finalizado el plan de estudios de la carrera y luego de dar cumplimiento a una serie de requisitos para solicitar el título, tales como presentar libre deuda de biblioteca y otros servicios brindados por la universidad.

El proceso de certificación en la UNSL se resume como se sigue.

1. El estudiante graduado que requiere su título, debe presentar la documentación solicitada por la universidad.
2. El personal administrativo de la universidad debe validar toda la documentación presentada y se procede a la creación de un expediente.
3. El personal administrativo y las autoridades superiores corroboran y validan toda la historia académica del estudiante. Este es un proceso riguroso ya que se deben corroborar todas las notas del estudiante con las actas presentadas por los docentes de las respectivas materias.
4. Se informa el egreso del estudiante a la entidad ministerial quien valida la información curricular del egresado.
5. La entidad ministerial valida la información provista por la universidad y suministra las obleas para la impresión del título.
6. El departamento de diplomas de la universidad imprime el título e informa al estudiante graduado. Este último es el único que puede retirar el documento. El título se entrega en la ceremonia de colación frente a las autoridades más importantes de la institución.
7. En cuanto a la verificación del título, el proceso depende de la entidad que quiera verificar la validez del mismo, por lo general se suele utilizar una copia fiel firmada y certificada por un escribano público o el título original en papel.

El proyecto pretende complementar las dos últimas etapas del proceso de certificación actualmente vigente, especialmente una vez que la entidad ministerial nacional da su completa aprobación, y también al momento de requerir la verificación de validez por parte del graduado.

5.1.2 Problemas a resolver

Riesgo de pérdida o extravío

Uno de los problemas más complejos al momento de manipular un título universitario es la pérdida o extravío. La recuperación del título no es un proceso sencillo: se debe presentar documentación para comprobar la pérdida del documento. En la UNSL se suele pedir una denuncia policial de extravío y toda la documentación necesaria para realizar una revalidación de los datos. Una vez presentada la documentación se debe repetir el proceso de emisión del título mencionado anteriormente, esto

significa que se debe validar nuevamente toda la información del estudiante en la universidad y en la entidad ministerial nacional. Esto conlleva a nuevos costos de administración y de tiempo por parte de ambas instituciones.

Exigencia de asistencia presencial

El título es el único documento que certifica los estudios universitarios del estudiante graduado. Por esta razón, se exige que éste se presente personalmente en el establecimiento para recibir el documento, significando un problema para aquellos graduados que, por ejemplo, se encuentran viviendo en el exterior del país. Tener una copia de respaldo en formato digital permitiría obviar para algunos interesados recibir el título en formato papel.

Carencia de mecanismos de verificación

Actualmente existen pocos mecanismos para verificar la validez de documentos de esta índole debido a su extensa rigurosidad. El método frecuentemente utilizado es la comprobación visual del papel original. La falsificación de estos documentos es compleja, pero el avance de la tecnología pone a disposición herramientas que facilitan su uso. Por otro lado, tener un mecanismo de verificación que pueda ser chequeado por un tercero en cualquier parte del mundo, tal como un posible empleador, brinda un beneficio adicional a los graduados universitarios.

5.1.3 Modelos de análisis

Para resolver las problemáticas planteadas, se desarrollará un sistema web para almacenar información relevante de los títulos universitarios de los estudiantes graduados. El sistema interactuará con estudiantes graduados de la UNSL, administrativos de la misma y cualquier persona o tercero que desee verificar los títulos de algún graduado.

Modelo de casos de uso

Como se mencionó anteriormente, con el desarrollo de este proyecto se pretende complementar las últimas etapas del proceso de certificación actualmente vigente, especialmente una vez que la entidad ministerial nacional da su completa aprobación.

A pedido del estudiante graduado (por fuera del sistema), el título será almacenado o dado de alta en el sistema por un administrativo de la universidad. Este último será el encargado también de gestionar la baja del título, en caso que sea necesario.

La figura 5.1 muestra un diagrama de casos de uso donde se especifican las funcionalidades del sistema y los actores o usuarios que interactúan con él.

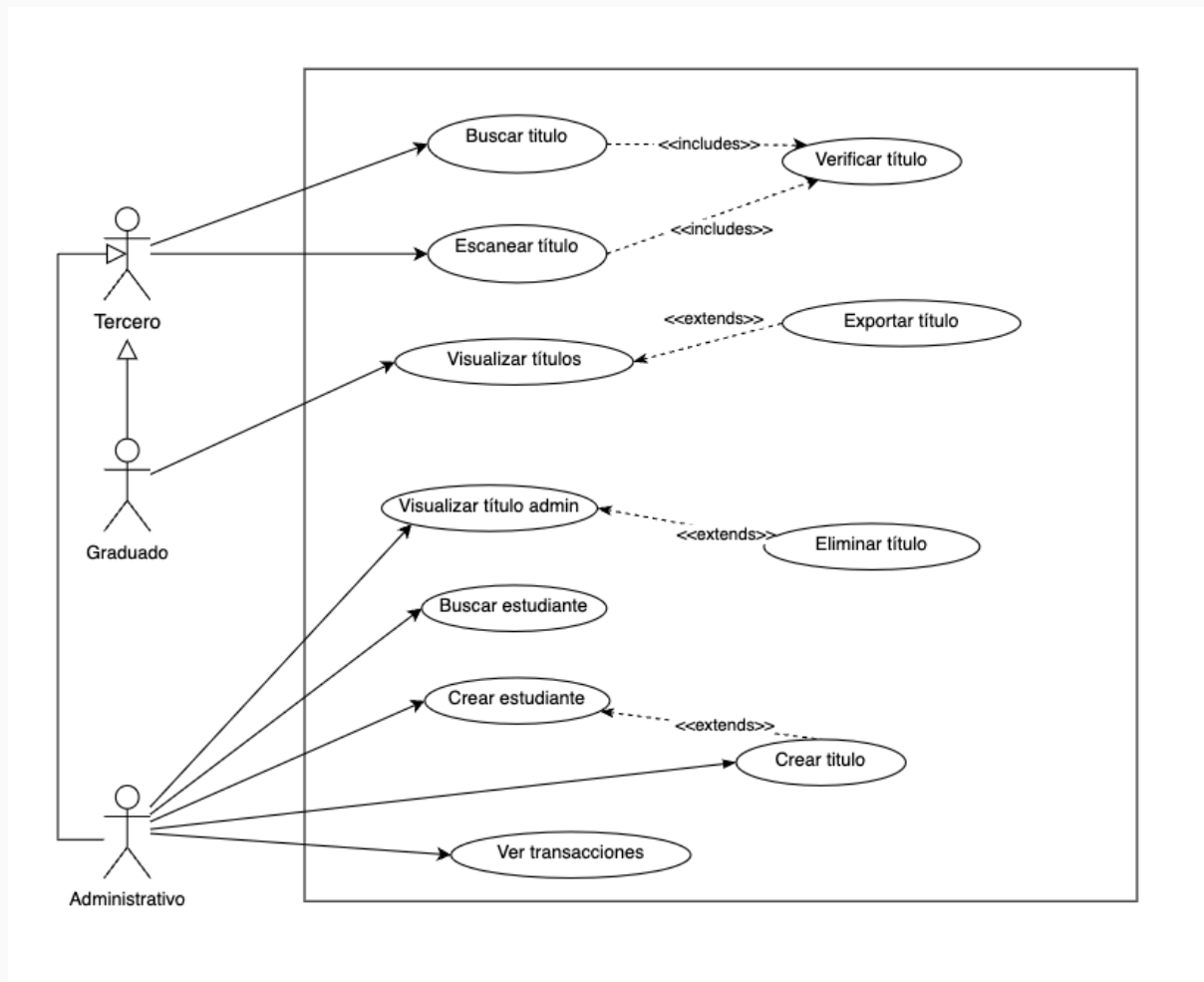


Figura 5.1: Diagrama de Casos de Uso

Actores

- Tercero: Cualquier persona o entidad que desee verificar el título universitario.
- Graduado: Estudiante graduado de la universidad.
- Administrativo: Personal administrativo de la universidad.

Casos de uso

El sistema estará constituido por 3 módulos principales:

- Público general o Tercero

Contendrá dos funcionalidades básicas:

- Buscar títulos de graduados: permitirá visualizar los títulos de los estudiantes graduados a través de su DNI para su verificación.

- Escanear un título: permitirá verificar, escaneando un código QR de los comprobantes de los títulos. Estos últimos deben ser emitidos previamente por los graduados, también a través del sistema.
- Graduado

Contendrá todas las funcionalidades del módulo público y además será posible:

 - Visualizar título: permitirá mostrar la información referida al título tal como número de oblea, datos de la institución, etc. y los datos que indican su almacenamiento en Blockchain.
 - Exportar título: permitirá exportar un título en formato pdf con un código QR que servirá para que cualquier usuario pueda luego verificarlo utilizando el módulo público.
- Administrativo

Contendrá todas las funcionalidades de los módulos anteriores y además será posible:

 - Crear graduado: permitirá dar de alta a estudiantes graduados.
 - Crear título: permitirá vincular títulos universitarios de los estudiantes graduados.
 - Eliminar título: permitirá realizar una baja lógica del título en el sistema, por lo que se eliminarán de las búsquedas y no podrá ser accedido por el público, solo por graduados y administrativos.
 - Buscar estudiante: permite buscar a los estudiantes del sistema por su número de dni.
 - Ver transacciones: permite visualizar un listado con todas las transacciones generadas por la creación y eliminación de títulos.
 - Visualizar título administrativo: Es una vista adicional para el usuario administrativo donde puede realizar acciones sobre el mismo como por ejemplo eliminar el título.

De esta forma, se le proporcionará al estudiante graduado un mecanismo para emitir un comprobante de cualquier título registrado y enviarlo a un tercero para que éste, a través del sistema, pueda contrastar y verificar la información del comprobante.

En el caso de que algún estudiante graduado desee registrar los títulos en el sistema, deberá contactarse con los administrativos de la universidad para que éstos tramiten la creación de una cuenta.

Cabe destacar que, la gestión de cuentas estaría fuera del alcance del sistema propuesto dado que el foco está en el desarrollo de los 3 módulos descritos anteriormente, la gestión de cuentas debería estar a cargo de personal técnico de la universidad perteneciente al centro de cómputos.

Los administrativos contarán con su propio perfil donde podrán dar de alta títulos de estudiantes graduados o dar de baja a aquellos títulos que hayan sido invalidados por alguna autoridad ministerial o por el mismo graduado si lo desea.

Buscar título

1. El tercero ingresa al sistema para buscar el título de un graduado.
2. El sistema solicita el número de documento del graduado.
3. El sistema realiza una búsqueda del título en la blockchain ,
4. El sistema retorna la información al tercero,

Flujos alternativos

- El sistema no encuentra al graduado:
 - 4.2. El sistema informa un error por pantalla.

Escanear título

Precondición: El tercero debe poseer el código QR provisto por el graduado.

1. El tercero ingresa al sistema a través de el link del código QR en el certificado del graduado.
2. El sistema valida el link.
3. El sistema busca la información del título en blockchain.
4. El sistema retorna la información al tercero.

Flujos alternativos

- El sistema no encuentra a graduado:
 - 4.2. El sistema informa un error por pantalla.

Visualizar título

Precondición: El graduado debe estar autenticado en el sistema.

1. El sistema solicita el número de documento del graduado.
2. El sistema realiza una búsqueda del título en la blockchain
3. El sistema muestra un listado con los títulos almacenados en blockchain, además de información de trazabilidad.

Flujos alternativos

- El sistema no encuentra a graduado:
 - 3.2. El sistema informa un error por pantalla.

Flujo excepcional: Se ejecuta en el caso de uso “Exportar título”.

Exportar título

El graduado ingresa al sistema a través de los casos de uso “Visualizar título”.

1. El graduado selecciona la opción “exportar título”.
2. El sistema genera un documento que contiene información del título del graduado junto con un código QR para ser escaneado.

Buscar estudiante

Usuario: Administrativo

Precondición: El administrador debe estar autenticado en el sistema.

1. El sistema le solicita al usuario que ingrese el número de documento del graduado.
3. El sistema realiza una búsqueda de los estudiantes asociados a ese número de documento en el sistema.
4. El sistema retorna la información.

Flujos alternativos

- El sistema no encuentra a graduado:
 - 4.2. El sistema informa un error por pantalla.

Crear graduado

Usuario: Administrativo

Precondición: El administrativo debe estar autenticado en el sistema.

El administrativo ingresa al sistema para crear un nuevo graduado.

1. El sistema le pide al administrativo que ingrese los datos del estudiante.
2. El sistema valida la misma y crea el estudiante en el sistema.
3. El sistema informa el éxito de la operación. En caso de error, informa el mismo.

Flujos alternativos

- El administrativo ingresa al sistema para asociar una nueva carrera a un estudiante ya existente en el sistema.

Precondición: Debe ejecutar el caso de uso “Buscar Estudiante”.

- 3.3. El sistema devuelve el/los estudiantes buscados.
- 3.4. El administrativo selecciona el estudiante al que le quiere asociar la nueva carrera.
- 3.5. El sistema autocompleta la información personal del estudiante
- 3.6. El administrativo ingresa la información del estudiante
- 3.7. El sistema valida la misma.
- 3.8. Crea el nuevo estudiante
- 3.9. El sistema informa el éxito.

Flujo excepcional: En caso de error, informa el mismo.

Crear título

Usuario: Administrativo

Precondición: El administrativo debe estar autenticado en el sistema y debe ejecutar el caso de uso “Buscar estudiante”.

El administrativo ingresa al sistema para cargar un nuevo título universitario.

1. El sistema le pide al administrativo que ingrese la información relevante del título.
2. El administrativo de la universidad ingresa la información del certificado.
3. El sistema valida la misma y carga esa información en la blockchain ejecutando un contrato inteligente en la red Ethereum.
4. El sistema informa el estado de la transacción por pantalla.

Flujo excepcional: El administrativo quiere cargar un título a un estudiante que ya posee uno, el sistema informa el error por pantalla.

Visualizar título admin

Usuario: Administrativo

Precondición: El administrativo debe estar autenticado en el sistema y debe ejecutarse en el caso de uso “Buscar título”.

El administrativo ingresa al sistema para consultar un título universitario.

1. El administrador ejecuta el caso de uso “Buscar Título”.
2. El sistema le ofrece al usuario la información del título junto con datos de trazabilidad.

Flujo excepcional:

1. Se ejecuta el caso de uso “Eliminar título”.

Eliminar título

Usuario: Administrativo

Precondición: El administrativo debe estar autenticado en el sistema y ejecutar el caso de uso “Visualizar título admin” previamente.

El administrativo ingresa al sistema para dar de baja un título universitario.

2. El sistema le solicita que confirme la acción que va a realizar mediante la solicitud de la contraseña.
3. El administrativo confirma su accionar ingresando su contraseña
4. El sistema ejecuta un contrato inteligente que da de baja el título en la red.

Ver transacciones

Usuario: Administrativo

Precondición: El administrativo debe estar autenticado en el sistema.

El administrativo ingresa al sistema para ver el listado de todas las transacciones que se hayan realizado en el sistema.

1. El sistema le ofrece un listado con todas las transacciones realizadas hasta la fecha.

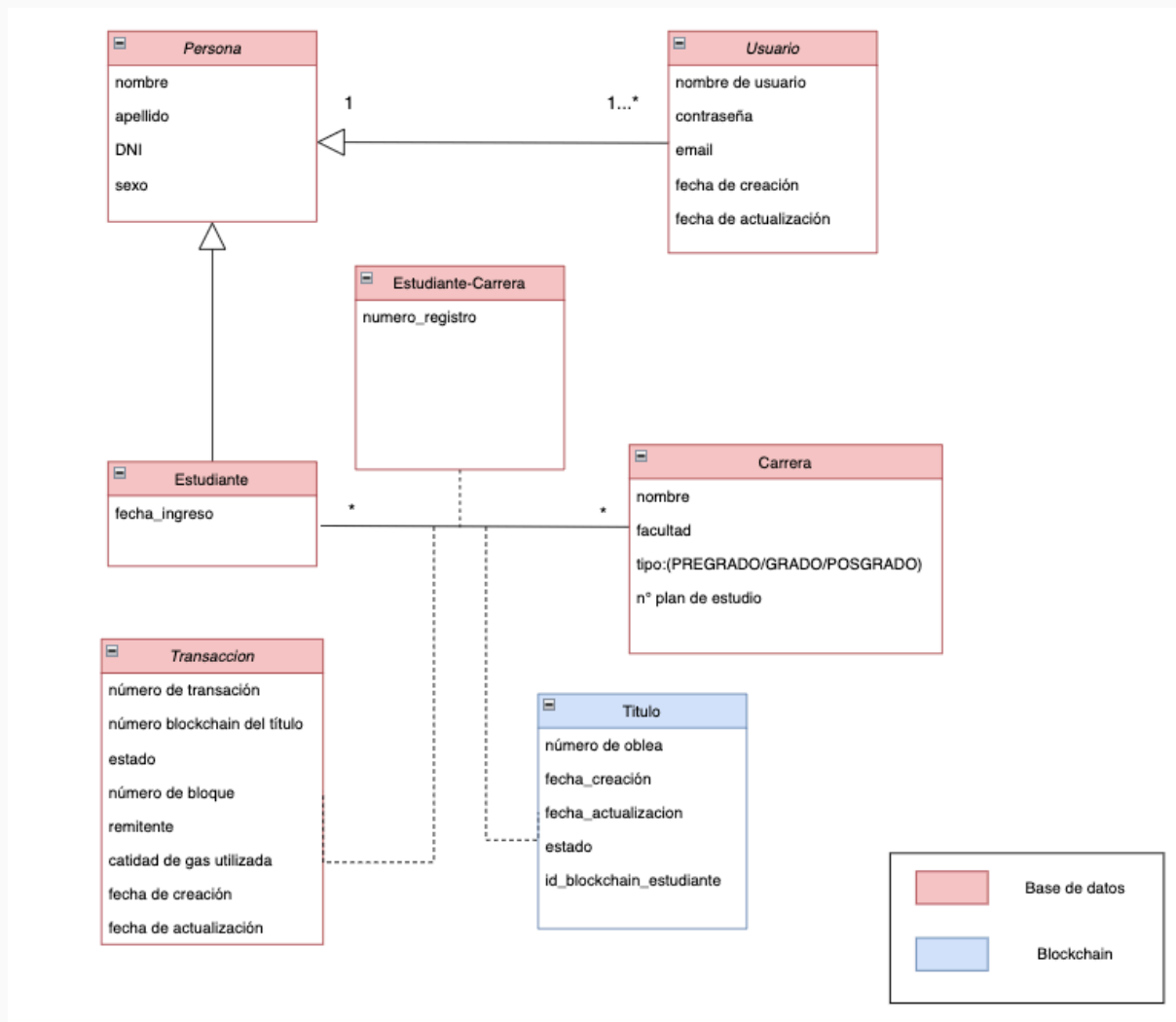


Figura 5.2: Diagrama de clases

Modelo conceptual

En este proyecto integrador interactúan diferentes entidades las cuales son mostradas en el diagrama de clases detallado en la figura 5.2.

A continuación se describen sintéticamente las clases que aparecen en el diagrama.

- Persona: Usuario o Estudiante graduado.
- Usuario: Persona registrada en el sistema.
- Estudiante: Estudiante graduado cuyo título será registrado.
- Carrera: Carrera que puede haber terminado un estudiante graduado.
- Estudiante-carrera: Carrera que ha finalizado un estudiante graduado.

- Transacción: Movimiento realizado en blockchain cuando se digitaliza el título.
- Título: Título de estudiante graduado registrado en blockchain.

5.1.4 Requerimientos No Funcionales

- Seguridad
 - Integridad: No se debe permitir que se modifiquen datos de los títulos cargados ni la pérdida de los mismos. Esto causaría daños catastróficos ya que la información escrita en ellos tiene mucha validez institucional.
 - Disponibilidad: Los servicios de la plataforma deben estar disponibles para el usuario cuando los necesite, de lo contrario, la misma puede perder oportunidades laborales o acceso a actividades académicas. Por lo que se debe evitar la denegación del servicio.
 - Confidencialidad: permitir el acceso solo a las partes autorizadas.
- Trazabilidad: Debido al manejo de información sensible, cada movimiento de los datos debe ser trazable y conocer quién fue el autor de los mismos y cuándo se realizaron.

El almacenamiento de información de títulos universitarios no es una tarea sencilla, se debe tener sumo cuidado ya que esos datos son críticos para una institución educativa. Se debe tener certeza de que los datos almacenados son los correctos y que esa información no pueda ser modificada por ningún tercero. Para asegurarnos la inmutabilidad de los datos utilizaremos la tecnología blockchain y los contratos inteligentes de la red Ethereum que nos brindarán herramientas adicionales para gestionar los datos almacenados.

En la tecnología blockchain se destaca principalmente el alto grado de trazabilidad ya que con las transacciones se obtiene un registro de todas las operaciones que se han realizado, ordenadas en bloques lo que nos aporta un orden temporal de las operaciones. Un valor agregado muy importante es la seguridad, los bloques se encuentran cifrados digitalmente unos con los otros lo que hace que sea casi imposible alterar el orden de los mismos. Además un aspecto fundamental de las implementaciones de esta tecnología es su descentralización, las redes blockchain poseen un conjunto de nodos que, gracias a los mecanismos de consenso mantienen un libro mayor de manera distribuida, con nodos distribuidos por todo el mundo.

5.2 Diseño

Ethereum provee una funcionalidad crucial para el desarrollo de este proyecto, sus contratos inteligentes, éstos son programas que, en la práctica, no se pueden modificar una vez creados,

tampoco se pueden manipular durante su ejecución y siempre se ejecutan según lo diseñado, y ninguna de las partes puede influir en su comportamiento.

Para ejecutar un contrato inteligente en la red Ethereum es necesario realizar una transacción, la misma posee un costo denominado Gas que está asociado a la velocidad, la congestión de la red y al esfuerzo computacional requerido para ejecutar las funcionalidades programadas en el mismo.

El Gas hace referencia al costo para llevar a cabo una transacción en Ethereum con éxito. Las mismas se deben pagar en la moneda nativa de Ethereum, el ether (ETH) que es abonado por quién realiza la transacción.

En esta aplicación los administrativos harán uso de la red blockchain para almacenar los títulos de los graduados de la universidad, siendo los únicos capaces de generar transacciones en la red. El resto de los actores en el sistema realizará operaciones de consulta. Por ejemplo, el estudiante graduado puede consultar sobre sus títulos, esta operación lee información almacenada en la red, por lo tanto no es necesaria la realización de una transacción.

Arquitectura del sistema

Para el diseño de la arquitectura de este proyecto se tiene en cuenta el desarrollo de una aplicación descentralizada (DAPP) que gestione la información de los títulos universitarios de los graduados, sus datos serán almacenados en un contrato inteligente de Ethereum y de esa manera permanecen en el blockchain por siempre, además el mismo contará con operaciones de consulta, creación y eliminación.

Es importante tener en cuenta que en la actualidad las redes blockchain presentan complejidades de escalabilidad y adopción por parte de la población, por lo tanto se decidió optar por una arquitectura semi-descentralizada, donde se utiliza un API REST que permite la conexión con la blockchain y los contratos inteligentes.

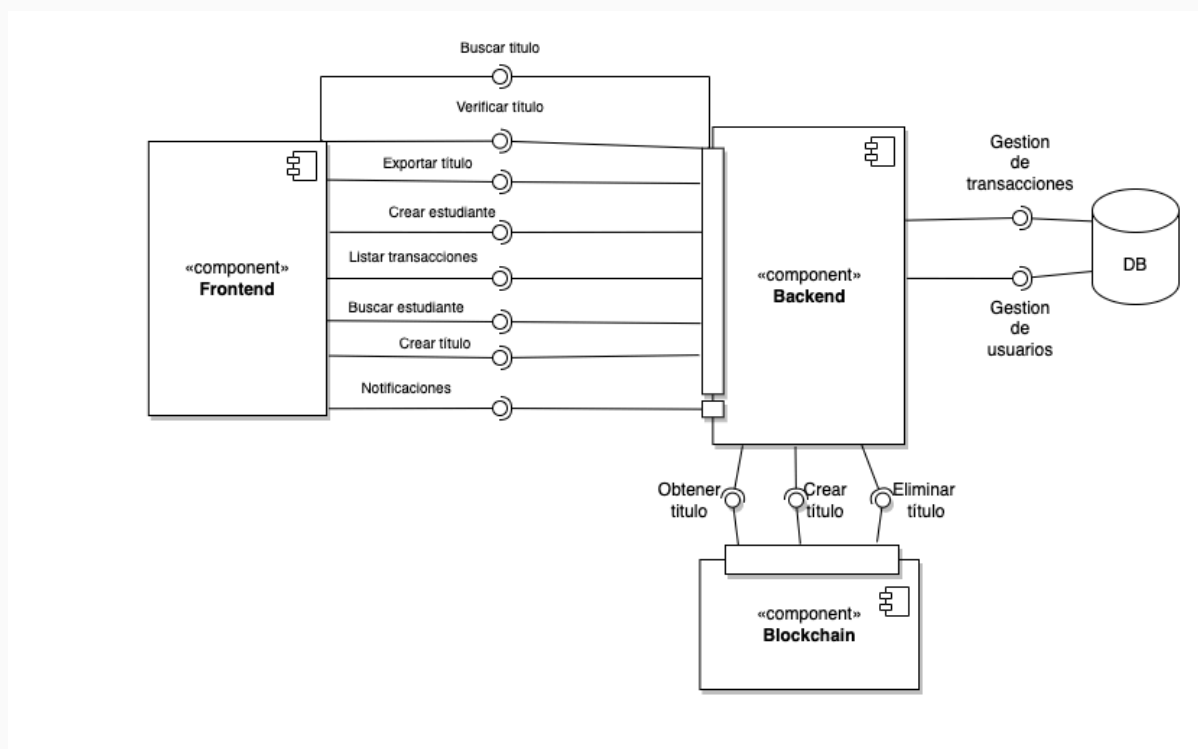


Figura 5.3: Diagrama de Componentes

Tal como se observa en la figura 5.3, el sistema cuenta con los siguientes componentes:

- **Frontend:** Provee de una interfaz gráfica donde el usuario puede interactuar con las diferentes funcionalidades del el sistema, para esto consume los diferentes servicios que el sistema backend provee a través de una API REST.
- **Backend:** Es un intermediario entre el frontend y la blockchain, se encarga de consumir los servicios que blockchain ofrece a través de la ejecución de un contrato inteligente permitiéndole consultar información de los títulos, crear o dar de baja los mismos y expone dichas funcionalidades al frontend para que el usuario pueda utilizarlas.

A través de una base de datos relacional se almacena información de los usuarios, permisos y roles para la gestión de los mismos proporcionando más seguridad al sistema, además posee un registro de las transacciones generadas para realizar un seguimiento de las mismas. Así mismo, se notifica al usuario cada vez que existe un cambio de estado en una transacción enviada.

- **Blockchain:** Se encarga de almacenar la información de los títulos universitarios de los estudiantes en la blockchain de Ethereum, a través de un contrato inteligente que provee dos

funcionalidades principales: el registro de los títulos universitarios de los estudiantes graduados y la posibilidad de deshabilitar la visibilidad de dicha información.

5.3 Implementación

En esta sección se describirán brevemente las tecnologías aplicadas para la implementación del Frontend, Backend y Blockchain. Posteriormente, se explicará cómo fue el desarrollo de cada una de las componentes y la comunicación entre las mismas: la comunicación Frontend-Backend y la comunicación Backend-Blockchain. Se mostrará parte de la codificación, así como el diseño, implementación y despliegue del contrato inteligente. De éste último se detallarán sus componentes principales y la implementación de las funcionalidades planteadas en la etapa de análisis, a través de un diagrama de secuencia. Finalmente se expondrán las problemáticas asociadas al dominio del problema y su solución pertinente.

5.3.1 Tecnologías aplicadas

Frontend y Backend

Se utilizará Angular para el desarrollo frontend y la librería Web3JS para interactuar con los contratos inteligentes en la Blockchain. Para el backend se hará uso del lenguaje de programación Typescript.

Base de Datos

El servidor contará con una base de datos para la persistencia de la información de los títulos, sesiones de los usuarios, perfiles, roles e información necesaria para la interacción con blockchain. Se utilizará el gestor de bases de datos PostgreSQL junto con el ORM Sequelize para mapear entidades del servidor a tablas en la base de datos.

Blockchain

Se utilizará el lenguaje de programación Solidity para implementar contratos inteligentes en la red de Ethereum, y se integrará Truffle para la compilación, testing y despliegue de los contratos. Para la creación del ambiente de pruebas se utilizará Ganache.

En este proyecto los casos de uso que requieren la realización de una transacción son aquellas que modifican los datos almacenados en blockchain por lo tanto, las funcionalidades de alta y baja de los títulos universitarios deberán ejecutar transacciones y se necesitará una cuenta de usuario en Ethereum (cuenta de propiedad externa) con saldo en ETH para poder solventar los costos de las mismas y firmarlas. Los únicos autorizados a gestionar la cuenta serán los administrativos de la universidad.

Para desplegar un contrato en la red y que pueda ser utilizado por el público es necesario un proceso de compilación del contrato para generar un código de bytes que luego son ejecutados por la máquina virtual distribuida de Ethereum, realizar estas operaciones tienen un costo en GAS pagado con Ether disponible en el saldo de la cuenta que vaya a implementar dicho contrato. Se realizará la implementación en el nodo de Infura en donde la EVM ejecutará el contrato, una vez desplegado, el contrato tendrá una dirección de Ethereum.

5.3.2 Comunicación Frontend-Backend

El frontend interactúa con el backend a través del protocolo HTTP ya que el servidor es una API REST que tiene todas las funcionalidades para la gestión de títulos, usuarios y notificaciones.

Para el envío de notificaciones se utiliza la tecnología WebSockets que hace posible abrir una sesión de comunicación punto a punto entre el navegador del usuario y el servidor, de esta manera el cliente se suscribe al canal de notificaciones y el servidor envía mensajes sobre el estado de la conexión con la blockchain y notifica sobre el estado de las transacciones enviadas.

5.3.3 Estructura del contrato inteligente

Durante el ciclo de vida de este contrato interactúan las siguientes entidades:

- Certificado (“Certificate”): Contiene todos los datos relevantes del título universitario.
- Carrera universitaria (“UniversityDegree”): Posee información sobre la universidad y la carrera del estudiante graduado.
- Estudiante (“Student”): Posee los datos personales del estudiante graduado.

```

4  contract Certificates {
5      // Variables.
6      uint256 public amountCertificates;
7
8      constructor() {
9          amountCertificates = 0;
10     }
11
12     // Estudiante.
13     struct Student {
14         uint256 id;
15         string name;
16         string lastname;
17         string docNumber;
18         string docType;
19         string sex;
20         uint256 registrationNumber;
21     }
22
23     // Carrera.
24     struct UniversityDegree {
25         string universityName;
26         string academicUnit; // Facultad
27         string degreeProgramName; // Nombre de la carrera
28         string degreeProgramCurriculum; // Plan de estudios
29         string degreeType; // Tipo de carrara
30     }
31
32     // Titulo.
33     struct Certificate {
34         uint256 id;
35         Student student;
36         UniversityDegree universityDegree;
37         string waferNumber; // Numero de oblea.
38         uint256 createdAt;
39         uint256 updatedAt;
40         bool active; // Activo
41     }

```

Figura 5.4: Código de las entidades del contrato.

En la figura 5.4 se observa que el contrato utiliza un contador *amountCertificates* que tiene la función de índice y límite superior de las estructuras, es inicializado cuando se ejecuta la función constructor del contrato, este se ejecuta por única vez cuando se despliega el mismo en blockchain.

Mappings

Se utilizan tres mappings para el almacenamiento y control de los títulos universitarios de los graduados, tal como se observa en la figura 5.5:

- Certificates: Almacena los datos de los títulos de los graduados.
- Wafers: Relaciona a las obleas con los títulos. Sirve para controlar que no hayan repetidos.
- Student_has_certificates: Relaciona el id del estudiante con el id de los títulos que posee.

```

49
50 // Titulos (idTitulo ⇒ titulo).
51 mapping(uint256 ⇒ Certificate) private certificates;
52
53 // Obleas (codigo_oblea ⇒ idTitulo).
54 mapping(string ⇒ uint256) private wafers;
55
56 // Estudiantes con titulos. (idEstudiante ⇒ idTitulo).
57 mapping(uint256 ⇒ uint256[]) private student_has_certificates;
58

```

Figura 5.5: Código de los mappings.

Funcionalidades

El contrato posee las siguientes funcionalidades:

Crear un título universitario (*createCertificate*)

Crea y almacena un nuevo título universitario y lo relaciona a un un estudiante graduado, demás datos como el número de oblea se almacenan en los mappings recientemente mencionados a fines de un mayor control e idempotencia (ver figura 5.6). Debido al impacto en el almacenamiento, esta funcionalidad posee un costo de GAS asociado. .

```

function createCertificate(Certificate memory _certificate) public {
    // Aumento el indice.
    amountCertificates++;
    // Creo el nuevo Id del titulo.
    uint256 newCertificateId = amountCertificates;
}

```

Figura 5.6: Signatura de la función.

Inicialmente la función recibe como parámetro *_certificate* el certificado a crear, luego se crea el id del certificado utilizando el contador *amountCertificates* que es una variable global del contrato. Se realiza un control sobre el número de oblea para chequear que no esté repetido en el contrato, para ello se consulta en el mapping el número de oblea pasado por parámetro el mismo debe retornar 0 (ver figura 5.7).

```

// Controlo que no exista el codigo de la oblea en el sistema.
require(
    wafers[_certificate.waferNumber] == 0,
    "El numero de oblea ya existe."
);

```

Figura 5.7: Control de número de oblea repetido.

En la figura 5.8 se puede observar otro control realizado para que no hayan títulos repetidos contrastando con los previamente cargados al estudiante. Se busca los títulos del estudiante por su id, luego se recorre dicha estructura y se compara los nombre de las carreras de los certificados, no debe haber otro título bajo el mismo nombre.

```
// Obtengo los ids de los titulos del estudiante.
uint256[] memory studentCertificatesids = student_has_certificates[
    _certificate.student.id
];

// Verifica si el estudiante ya tiene un certificado asociado a la misma carrera.
if (studentCertificatesids.length > 0) {
    for (uint i = 0; i < studentCertificatesids.length; i++) {
        uint256 certificateId = studentCertificatesids[i];
        require(
            keccak256(
                bytes(
                    certificates[certificateId]
                        .universityDegree
                        .degreeProgramName
                )
            ) !=
            keccak256(
                bytes(
                    _certificate.universityDegree.degreeProgramName
                )
            ),
            "Ya existe un certificado con el mismo nombre para este estudiante"
        );
    }
}
```

Figura 5.8: Control de certificado repetido para estudiante

Luego se procede a la creación del título, primero se crea el certificado y se guarda en la variable *certificates*, luego se relaciona el número de oblea añadiendo el identificador del título a la variable *wafers*, tal como se observa en la figura 5.9.

```
// Creo el titulo con el nuevo id y lo guardo en el mapa.
certificates[newCertificateId] = Certificate(
    newCertificateId,
    _certificate.student,
    _certificate.universityDegree,
    _certificate.waferNumber,
    block.timestamp,
    block.timestamp,
    true
);

// Relaciono la oblea con el titulo.
wafers[_certificate.waferNumber] = newCertificateId;
```

Figura 5.9: Creación de título

Se pueden discriminar dos casos, el del estudiante nuevo en el sistema y el que ya tiene títulos cargados se pueden visualizar en la figura 5.10 y 5.11 respectivamente.

```
// Si el estudiante es nuevo. No tiene titulos asociados.
if (studentCertificatesids.length == 0) {
    // Reutilizo la lista.
    studentCertificatesids = new uint256[](1);
    // Guardo el id del titulo.
    studentCertificatesids[0] = newCertificateId;
    student_has_certificates[
        _certificate.student.id
    ] = studentCertificatesids;
```

Figura 5.10: Caso estudiante nuevo

Para crear un nuevo estudiante, se debe crear una nueva lista de ids vacía y se agrega el nuevo id del título a ésta, luego se asigna al mapping *student_has_certificates* para relacionar el estudiante con su lista de títulos.

```

} else {
    // Creo la nueva lista de studentCertificatesids.
    uint256[] memory newStudentCertificatesIds = new uint256[](
        studentCertificatesids.length + 1
    );
    for (uint256 i = 0; i < studentCertificatesids.length; i++) {
        newStudentCertificatesIds[i] = studentCertificatesids[i];
    }
    // Agrego el nuevo id del titulo.
    newStudentCertificatesIds[
        studentCertificatesids.length
    ] = newCertificateId;
    student_has_certificates[
        _certificate.student.id
    ] = newStudentCertificatesIds;
}

```

Figura 5.11: Caso estudiante ya cargado

En el caso de que el estudiante ya esté cargado en el sistema, es necesario agregar el nuevo id a la lista (para hacer esto se tiene que crear una nueva lista y mover todas las referencias), y asignarla a *student_has_certificates*.

Finalmente se emite un evento que sirve para comunicar que el certificado ha sido creado con éxito al exterior, tal como se ve en la figura 5.12.

```

emit CertificateCreated(
    newCertificateId,
    block.timestamp,
    _certificate.student.docNumber,
    block.chainid,
    tx.origin
);

```

Figura 5.12: Emitir evento

Obtener todos los títulos de un estudiante

A partir de un número identificador se realiza una búsqueda en los títulos almacenados y se retornan al usuario. Esta operación de lectura no tiene costo de gas asociado debido a que no modifica el almacenamiento, tal como se observa en la figura 5.13.

```

65     function getCertificatesByStudentId(
66         uint256 studentId
67     ) public view returns (Certificate[] memory) {
68         uint256[] memory studentCertificatesids = student_has_certificates[
69             studentId
70         ];
71         Certificate[] memory studentCertificates = new Certificate[](
72             studentCertificatesids.length
73         );
74         for (uint256 i = 0; i < studentCertificatesids.length; i++) {
75             // Filtro aquellas activas.
76             if (certificates[studentCertificatesids[i]].active) {
77                 studentCertificates[i] = certificates[
78                     studentCertificatesids[i]
79                 ];
80             }
81             // studentCertificates[i] = certificates[studentCertificatesids[i]];
82         }
83         return studentCertificates;
84     }

```

Figura 5.13: Código de función de búsqueda de títulos por id de estudiante

Dar de baja el título

Permite dar de baja un título de un graduado mediante el atributo “*active*” de la entidad *Certificate*. Una vez que se ha dado de baja, el título no es tenido en cuenta al momento de la búsqueda. tal como se observa en la figura 5.14.

```

177     function deleteCertificate(uint256 certificateId) public {
178         // Obtengo el titulo.
179         Certificate memory certificate = certificates[certificateId];
180         // Inhabilito el acceso del titulo.
181         certificate.active = false;
182         certificate.updatedAt = block.timestamp;
183         certificates[certificateId] = certificate;
184
185         // Elimino la oblea.
186         wafers[certificate.waferNumber] = 0;
187
188         emit CertificateCreated(
189             certificate.id,
190             block.timestamp,
191             certificate.student.docNumber,
192             block.chainid,
193             tx.origin
194         );
195     }

```

Figura 5.14: Código de función de baja del título.

5.3.4 Compilación y despliegue

Los contratos una vez codificados deben ser compilados, en este proceso se transforman las instrucciones de Solidity a una serie de códigos de operación. Dado que los mismos tienen un costo en unidades de Gas, se puede realizar una estimación del consumo de la ejecución del contrato.

Para este proyecto se utilizó el framework Truffle ya que permite realizar la compilación y el despliegue del contrato inteligente, Truffle utiliza los archivos compilados para desplegar en la red Ethereum que deseemos, esto consiste en emitir una transacción que crea el contrato inteligente en la red almacenando su código en la cadena de bloques. Como mencionamos anteriormente, los contratos inteligentes son un tipo de cuenta de Ethereum que no están controladas por usuarios, su comportamiento está definido por su código y tienen una dirección. Una vez que el contrato es desplegado exitosamente, se obtiene la dirección del mismo proveniente de la blockchain la cual es fundamental para identificar unívocamente al contrato en la cadena de bloques.

Existen diversos ambientes en el desarrollo de contratos en Ethereum:

- Mainnet (Red principal): Se encarga de ejecutar las transacciones reales dentro de la red y de almacenarlas en la cadena de bloques para su uso público. Aquí los costos de transacciones y despliegues deben ser pagados con Ether (ETH) real con un valor monetario en el mercado de criptomonedas.
- Testnet: La red de pruebas proporciona un entorno alternativo que imita la funcionalidad de la red principal para permitir a los desarrolladores construir y probar proyectos sin necesidad de facilitar las transacciones en vivo o el uso de criptomonedas. Aquí los costos se pagan en Ether de prueba que puede suministrado por numerosos sitios web que regalan ETH de prueba.
- Local: Ejecutando una blockchain de pruebas en una computadora local, como por ejemplo: Ganache.

Para lograr el despliegue del contrato en la red es necesario que un usuario lo haga; es decir, se necesita una cuenta de usuario con saldo en ETH suficiente para pagar las comisiones necesarias para realizar el despliegue del contrato.

La cuenta de usuario y los datos de la red de despliegue son configurables desde un archivo llamado “truffle-config.js” donde se puede especificar el tipo de red que se quiere desplegar y la dirección de la cuenta.

5.3.5 Comunicación Backend-Blockchain

Para interactuar con contratos inteligentes desde una aplicación web es necesario tener el archivo ABI del contrato en formato JSON para poder leer información estática del contrato y utilizar la librería

Web3js, esta librería provee una interfaz sobre el protocolo JSON RPC que permite comunicarse con la red Ethereum, permite obtener información de la blockchain, utilizar cuentas Ethereum, crear instancias de contratos para utilizar sus funciones, crear transacciones, validarlas y firmarlas digitalmente. En el backend de la aplicación se desarrolló un servicio encargado de realizar la conexión e interacción con los contratos, llamado *Web3Service*. Tal como se ve en la figura 5.16.

```
16 const URL = process.env.NETWORK_URL!;  
17 const CONTRACT_ADDRESS = process.env.CONTRACT_ADDRESS!;  
18 const contractArtifact = require('../..../contracts/Certificates.json');  
19  
20 You, last month | 1 author (You)  
21 class Web3Service {  
22     private _web3!: Web3;  
23  
24     readonly certificateContract: Contract | undefined = undefined;  
25     certificates = [];  
26     networkId: number = 0;  
27  
28     public get web3(): Web3 {  
29         return this._web3;  
30     }  
31  
32     public set web3(value: Web3) {  
33         this._web3 = value;  
34     }  
35  
36     constructor() {  
37         if (URL) {  
38             this.web3 = new Web3(URL);  
39             this.certificateContract = this.getCertificateContract();  
40         } else {  
41             throw new Error('No existe URL para conectar con blockchain. URL:${URL}');  
42         }  
43     }  
44 }
```

Figura 5.16: Parte del código de Web3Service

En la línea 18 se puede observar el *import* del archivo ABI del contrato en formato JSON a la variable *contractArtifact*, que luego es utilizado por el método “*getCertificateContract()*” para instanciar el contrato en el servicio.

En la línea 59 de la figura 5.17 se muestra cómo se accede finalmente a las propiedades del ABI y luego junto con la dirección del contrato obtenida en el proceso de despliegue se instancia el contrato en la variable *certificateContract* en tiempo de ejecución para su posterior uso.

```
57 private getCertificateContract() {  
58     if (CONTRACT_ADDRESS) {  
59         const abi = contractArtifact.abi;  
60         const certificateContract = new this.web3.eth.Contract(  
61             abi,  
62             CONTRACT_ADDRESS  
63         );  
64         return certificateContract;  
65     } else {  
66         throw new Error('No existe Direccion de contrato.');67     }  
68 }
```

Figura 5.17: Código de *getCertificateContract*

5.3.6 Funcionalidades Básicas

Creación de título en blockchain

Esta funcionalidad es realizada por el administrador del sistema, que carga manualmente los datos del graduado en el sistema web, luego los mismos son validados y enviados al servidor. Una vez allí se crea una transacción con los datos, se firma digitalmente y se envía a la red blockchain para que se guarden en el contrato inteligente (ver figura 5.18).

En la figura 5.19 se puede observar el código que permite la creación de un certificado. La propiedad *methods* de la instancia del contrato utiliza el método *createCertificate* para crear la transacción que posteriormente ejecutará, recibe como parámetro un objeto *certificate* con toda la información del título del graduado.

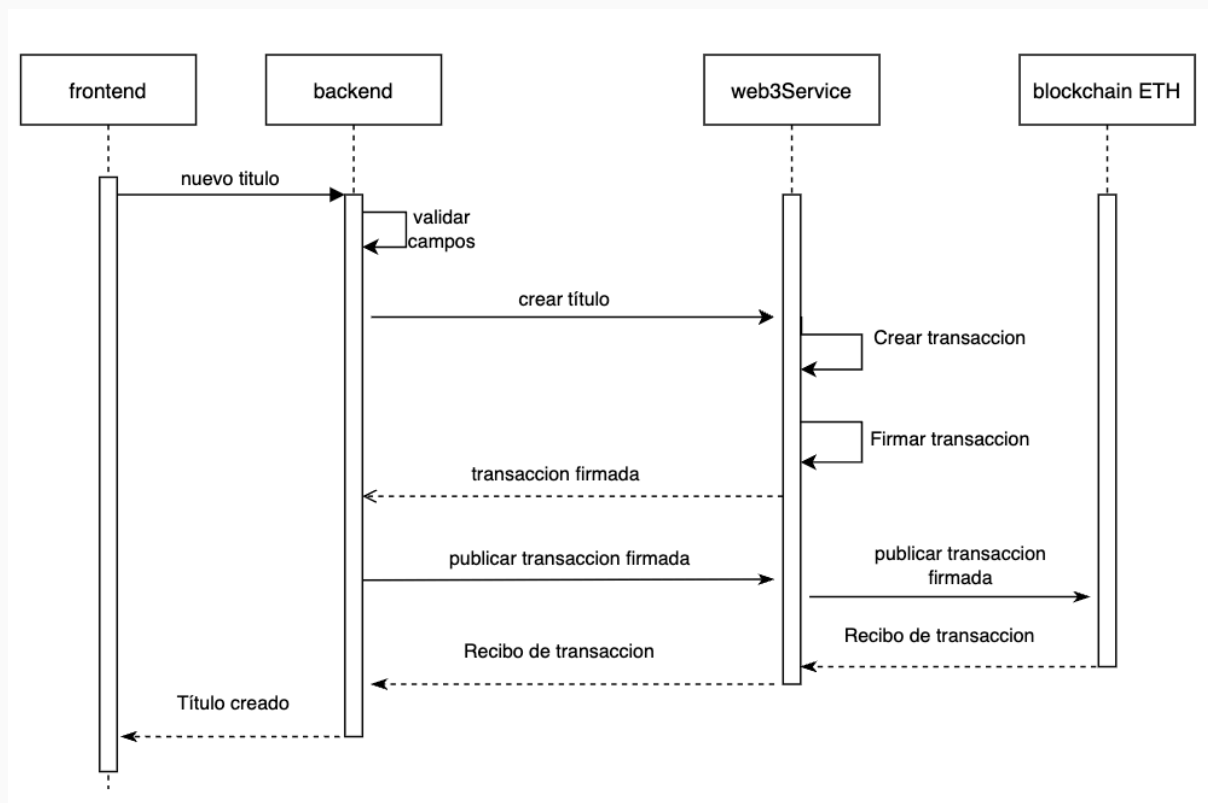


Figura 5.18: Diagrama de secuencia crear título

```

167 // Creo la transaccion con el metodo a ejecutar del smart-contract.
168 const transaction =
169   this.certificateContract!.methods.createCertificate(certificate);
170
171 // Calculo el gas estimado de la transaccion.
172 const gas = await transaction.estimateGas({ from: account?.address! });
173
174 // Codifico la transaccion para ser firmada.
175 const data = transaction.encodeABI();
176

```

Figura 5.19: Creación de una transacción

Luego se calcula de manera estática la cantidad de gas estimada que puede costar la creación del título utilizando la función *estimateGas*, ésto evita costos excesivos y posibles errores de tipo. Seguido a esto se codifica la transacción para poder ser firmada y enviada, tal como se puede observar en la figura 5.20.

Una vez codificada la transacción en la variable *data*, se crea la configuración de la transacción en la variable *options* donde se establecen los parámetros necesarios para la firma de la transacción como la dirección de la cuenta creadora, un número secuencial *nonce* y los datos codificados (ver figura 5.21).

```

// Creo la configuracion de la transaccion con los datos para ser firmada.
const options = {
  to: transaction._parent._address,
  data: data,
  nonce: nonce,
  gas: gas
} as TransactionConfig;

```

Figura 5.20: Firma de transacción

```

// Creo la configuracion de la transaccion con los datos para ser firmada.
const options = {
  to: transaction._parent._address,
  data: data,
  nonce: nonce,
  gas: gas,
  gasPrice: 500000
} as TransactionConfig;

```

Figura 5.21: Propiedad gasPrice

```
// Firmo la transacción con la clave privada.
const signed = await this.web3.eth.accounts.signTransaction(
  options,
  account.privateKey!
);
```

Figura 5.22: Firma de transacción

```
105   async sendTransaction(signed: SignedTransaction) {
106     let receipt = null;
107     receipt = await this.web3.eth.sendSignedTransaction(
108       signed.rawTransaction!
109     );
110     return receipt;
111   }
112
```

Figura 5.23: Envío de una transacción

Existen más opciones de configuración de la transacción como la propiedad *gasPrice* que sirve para indicar el precio del costo del gas máximo que puede costar la transacción, sin embargo, este valor debe ser calculado con detenimiento ya que puede interferir en la aprobación de la misma. Este caso será explicado en la sección 6.5

Finalmente, tal como se muestra en la figura 5.22, se crea la transacción firmada en la variable *signed* que posteriormente será enviada a la red.

A través del método *sendSignedTransaction* de web3js permite enviar la transacción firmada a la red, la misma una vez realizada retornará un recibo de transacción en la variable *receipt* (ver figura 5.23).

Buscar título

Para obtener los títulos universitarios de un estudiante que se encuentran almacenados en blockchain es necesario realizar una llamada a función del contrato que busque la información y la devuelve al usuario. Los títulos universitarios en blockchain se encuentran relacionados a los estudiantes a través de un número identificador (Blockchain ID), está compuesto por una concatenación entre el número de documento de identidad y el número de registro de estudiante que es otorgado por la universidad, por ejemplo: el estudiante graduado con número de documento “36046454”, cuyo número de registro antes de su graduación “3028516”, tendrá como identificador “360464543028516”. El motivo de esta metodología es para evitar duplicados ya que el documento de identidad no siempre es único.

Como mencionamos anteriormente, esta aplicación utiliza una base de datos relacional para el almacenamiento de los estudiantes graduados y las cuentas, por lo tanto para realizar la búsqueda en blockchain primero se debe buscar al estudiante para obtener su Blockchain ID y con éste realizar la búsqueda al contrato inteligente.

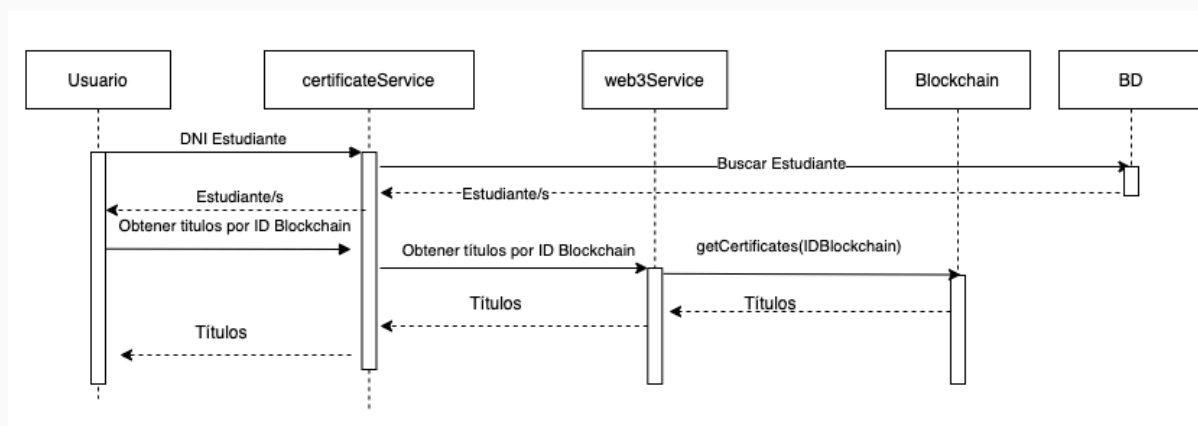


Figura 5.24: Diagrama de secuencia “buscar título”

El usuario ingresa el número de documento del estudiante, éste es utilizado para buscar en la base de datos a los estudiantes asociados y retornarlos, luego el usuario envía el Blockchain ID del estudiante será buscado en la red, en la figura 5.24 podemos observar el flujo de mensajes entre las diferentes partes de la aplicación, teniendo como intermediario principal al servicio “web3Service”. Más adelante, en el capítulo 6.3 se detalla el funcionamiento con capturas de pantalla de la interfaz de usuario.

Tratamiento de las bajas

Dado que la tecnología blockchain garantiza la inmutabilidad de los datos, para el desarrollo de este proyecto utilizaremos la técnica de baja lógica de los datos, esto quiere decir que la información no será eliminada, sino que se limita su acceso. Esto es posible dado que la entidad “certificado” del contrato posee un atributo denominado “active” que indica si este está visible o no para el usuario, si un título se da de baja no es tenido en cuenta para las búsquedas.

Debido a que ésta operación modifica los datos del contrato, es necesario la generación de una transacción que ejecuta la función realiza la baja del título, en la figura 5.25 se observa la interacción de las distintas partes de la aplicación.

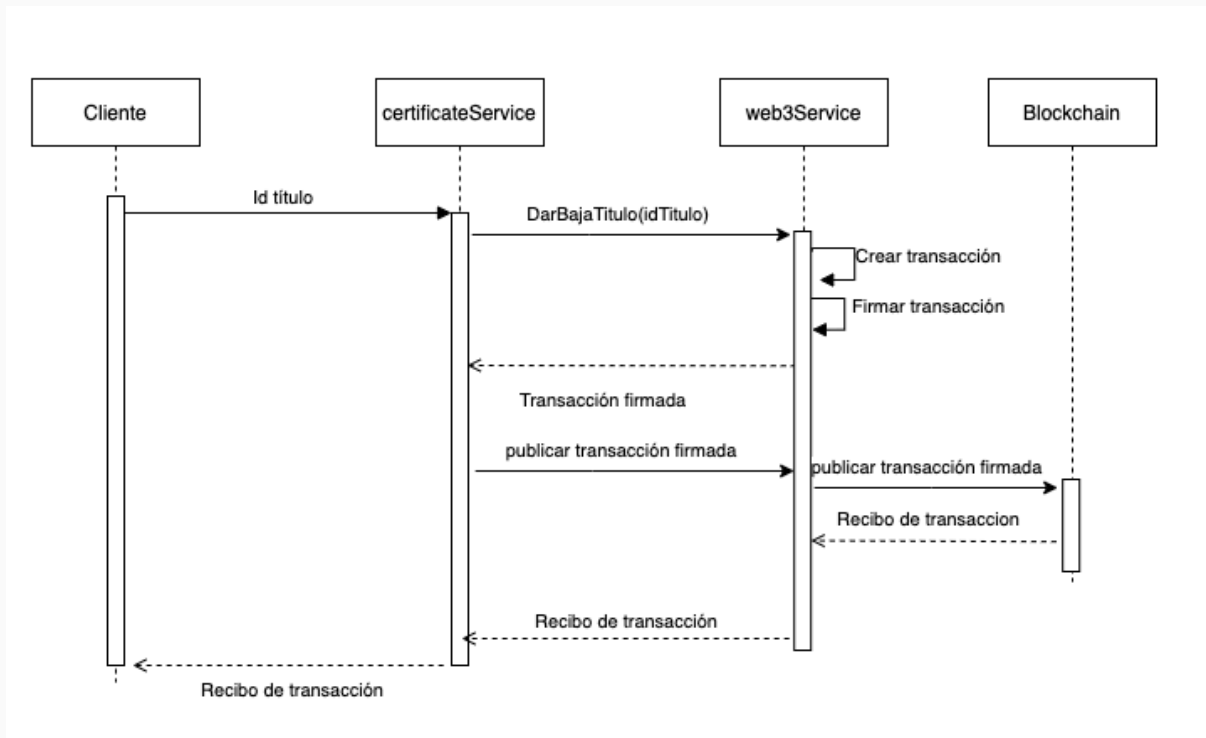


Figura 5.25: Diagrama de secuencia baja de título

5.3.7 Problemáticas asociadas

Manejo de transacciones y eventos

Debido a la naturaleza asincrónica del envío de transacciones a la blockchain, pueden ocurrir casos donde la misma puede fallar, perderse y nunca ejecutarse, además que el tiempo de respuesta de la ejecución de las mismas es considerable teniendo en cuenta que se realizan operaciones de validación y consenso de manera distribuida. No obstante, para mejorar la usabilidad para el usuario se desarrolló un historial de las transacciones enviadas utilizando la base de datos, de esa manera se puede tener mayor trazabilidad del estado de las transacciones.

Previo a enviar cualquier transacción se guarda una copia de los datos de la misma en la base de datos de forma “pendiente” hasta que la transacción sea exitosa o falle. El servidor deberá escuchar los eventos de la blockchain para obtener información del estado de las transacciones publicadas, en la figura 5.26 se observa el objeto BD que almacena el historial de transacciones y además se crean mensajes asincrónicos que están sujetos a eventos disparados por la blockchain.

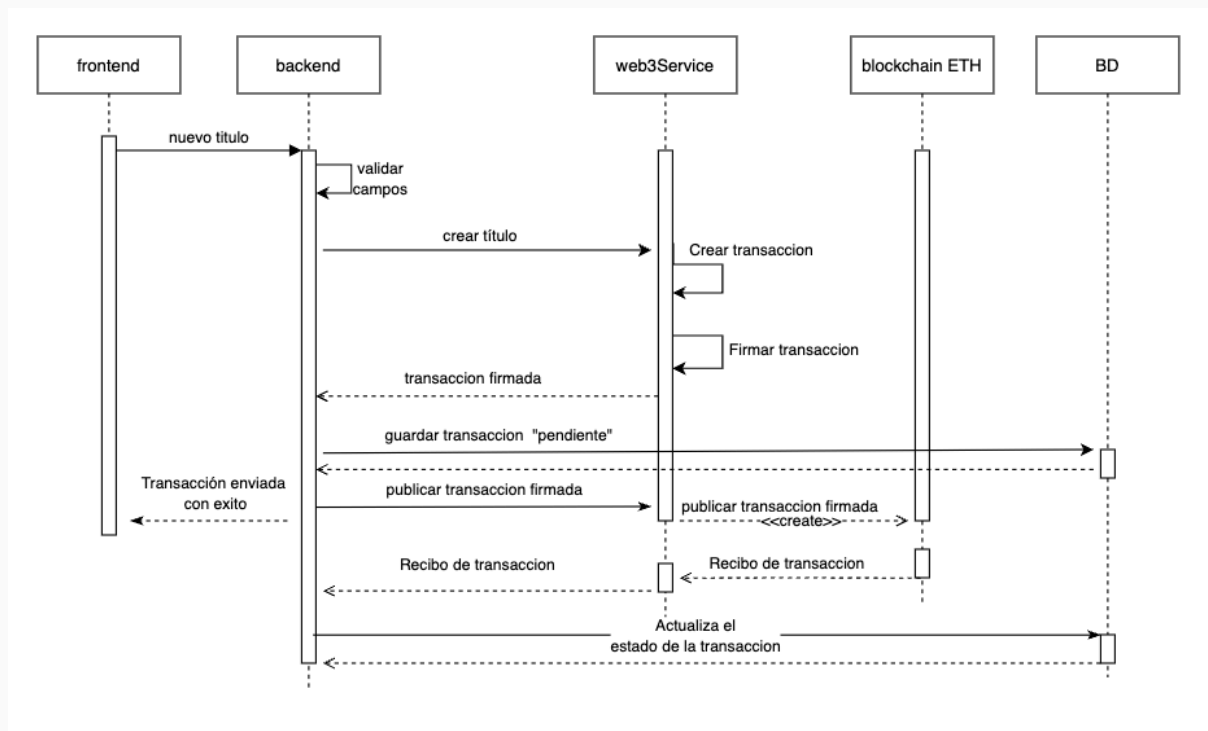


Figura 5.26: Manejo de eventos en la creación de títulos.

Costo de gas

La ejecución de una función en un contrato inteligente implica la ejecución de un conjunto de operaciones primitivas de la máquina virtual de Ethereum, cada una de éstas tiene un costo asociado, por lo tanto tener funcionalidades muy complejas o con malas prácticas de programación pueden incurrir en costos excesivos de Gas.

Los errores en tiempo de ejecución causados por una falla en la lógica del contrato como por ejemplo bucles infinitos pueden provocar costos excesivos en unidades de Gas para ejecutar la transacción, lo cual puede provocar que la misma falle por falta de fondos. Por lo tanto, se deben realizar mecanismos para la gestión y prevención de dichos errores como por ejemplo, limitar las iteraciones al momento de recorrer una estructura a un número fijo y/o evitar el uso de listas para almacenar datos donde se realizan búsquedas, ya que el costo de recorrer dicha estructura es mucho mayor al que tienen otras como las tablas de hash (“mapping” en Solidity) que fueron utilizadas en este proyecto para el almacenamiento de los títulos de los graduados.

Protección de claves

En el desarrollo de este proyecto se tuvo consideración a plantear una solución para la seguridad de las claves, ya que el sistema posee un esquema de usuario y contraseña tradicional junto con un esquema de roles y permisos, aquellos usuarios de tipo administrador contarán con una cuenta Ethereum personal que son utilizadas para firmar las transacciones. Para el usuario final este proceso

es totalmente transparente, la utilización de billeteras y la interacción con blockchain es hecha por el servidor lo que otorga usabilidad a la aplicación pero reduce la seguridad centralizando las cuentas en un solo lugar. Como mencionamos en el capítulo 2, la cuenta de Ethereum posee una clave privada que es utilizada para generar la firma digital en las transacciones, por lo que es crucial tener estas claves almacenadas en un lugar seguro. El almacenamiento de las claves de las cuentas están cifradas en la base de datos del servidor, al igual que se hace con las contraseñas de todos los usuarios del sistema. En el caso de que se olvide la contraseña o se pierda la clave privada de la cuenta Ethereum, existe una frase semilla de recuperación llamada “mnemónico”, son 12 palabras al azar que sirven para recuperar la cuenta, a través de las herramientas de provee Ethereum o de terceros se puede recuperar fácilmente. Esta frase mnemónica deberá ser almacenada y resguardada por la institución.

Por motivos de alcance del proyecto integrador, se utiliza una única cuenta de usuario para todos los administrativos que realicen transacciones para la creación o baja de los títulos universitarios, además el sistema es desarrollado para que sea utilizado con una sola universidad, sin embargo, puede ser escalado hacia múltiples universidades y facultades donde cada una de ellas posee una o varias billeteras únicas de esa institución.

Finalmente, existe un mecanismo de seguridad más robusto que consiste en crear otro contrato inteligente que cumple la funcionalidad de auditor, este contiene una “lista blanca” de las direcciones de cuentas que son válidas para crear títulos, esto sería útil en el caso de que algún tercero no deseado se apodere de alguna de las claves privadas y quiera crear títulos apócrifos.

Idempotencia y verificabilidad de los datos

En este proyecto procuramos garantizar que los datos almacenados en blockchain sean únicos y que puedan ser verificados, esto es, que, independientemente de las interfaces que propone el sistema para la verificación por parte de un tercero, sea posible verificar los títulos montando una red propia de blockchain que ejecute la EVM.

Esto podría dar lugar a que se pueda crear otro contrato inteligente logrando almacenar información de títulos apócrifos, este caso puede ser posible debido a que la tecnología Ethereum es de código abierto y los contratos desplegados allí son de carácter público. Ésto conlleva a responder a la siguiente pregunta.

¿Es posible garantizar que los datos provienen de un origen válido?

La respuesta a esta pregunta es afirmativa dado el tipo de blockchain en la cual el contrato es desplegado. Anteriormente, se hacía referencia a los distintos tipos de ambientes que la cadena de bloques de Ethereum posee, Mainnet y Testnet, sin embargo, existen numerosas implementaciones alrededor del mundo pero la cadena oficial que posee todas las transacciones reales y por lo tanto es la única que certifica es Mainnet.

Cuando se realiza la publicación de un contrato inteligente en la red se debe generar una transacción que es firmada por el remitente (dirección de una cuenta Ethereum de usuario) quien es el dueño y poseedor del contrato, además se genera una dirección del contrato, su identificador unívoco. Con este identificador se conoce qué contrato se ejecutó y quién fue el encargado de publicarlo, sin embargo, queda abierta una pregunta referida a la verificabilidad de la información:

¿De qué manera se puede verificar la información?

Si bien hay una sola cadena de bloques que contenga al certificado válido, ¿cómo es posible conocer si esa información no fue ingresada por un tercero que se apoderó de las cuentas de administradores y le da posibilidad de subir títulos apócrifos? Para este punto es fundamental tener una entidad que se encargue de verificar las direcciones de las cuentas Ethereum que firman transacciones en la red. Un ejemplo de esto sería el Ministerio de Educación que se encargue de brindar las cuentas a las instituciones para que puedan firmar transacciones.

5.4 Prueba

Se realizaron test unitarios sobre la funcionalidad del contrato inteligente que permite la creación de los títulos. Se utilizó Truffle como soporte para la creación y ejecución de los test. Para comprobar los mecanismos de control de idempotencia del contrato inteligente la prueba consiste en crear dos títulos iguales, concretamente con el mismo número de obla que es el identificador único del título, en la figura 5.27 se puede ver la estructura del mismo.

Inicialmente se realiza una llamada al contrato inteligente para crear el título, luego se realizan comprobaciones para conocer si fue efectiva la creación, se verifica que el evento de creación del contrato se haya disparado, luego se consulta por id de estudiante para verificar que el título se haya relacionado correctamente. Finalmente se realiza una nueva llamada al contrato para crear el mismo título como se puede visualizar en la línea 64 de la figura 5.28, para verificar si el test se ejecutó con éxito dicha llamada debe arrojar una excepción indicando el error de idempotencia y revirtiendo los cambios.

```

it("Prueba de idempotencia: No se pueden crear dos títulos iguales (prueba numero de oblea.)", async () => {
  // Datos título. You, 9 minutes ago • Uncommitted changes
  const certificate1 = {
    student: {
      docNumber: "41221778",
      docType: "DNI",
      name: "Federico",
      lastname: "Verges",
      sex: "Masculino",
      registrationNumber: 3028516,
      id: 412217783028516,
    },
    active: true,
    universityDegree: {
      academicUnit: "Facultad de Ciencias Fisico Matemáticas y Naturales",
      degreeProgramName: "Ingeniería en Electrónica",
      degreeProgramCurriculum: "16-10",
      degreeType: "POSGRADO",
      universityName: "Universidad Nacional de San Luis",
    },
    createdAt: 0,
    id: 0,
    updatedAt: 0,
    waferNumber: "123123213",
  };
});

```

Figura 5.27: Estructura de título de prueba

```

49 // Llama a la función para crear el certificado.
50 await this.certificateContract.createCertificate(certificate1);
51
52 // Verifica que el evento "CertificateCreated" haya sido emitido.
53 const events = await this.certificateContract.getPastEvents("CertificateCreated");
54 assert.equal(events.length, 1, "El evento CertificateCreated no se emitió correctamente.");
55 console.log(events);
56
57 // Verifica que el certificado fue creado correctamente.
58 const certificates = await this.certificateContract.getCertificatesByStudentId(certificate1.student.id);
59
60 assert.lengthOf(certificates, 1, "título: Unknown word. cSpell rrectamente.");
61 try {
62   // Vuelvo a cargar el mismo título.
63   await this.certificateContract.createCertificate(certificate1);
64
65   // Si continua normalmente, arrojo error.
66   throw new Error('No se pueden crear dos certificados con el mismo numero de oblea.');
```

Figura 5.28: Código del test

Test de aceptación

Posterior a las pruebas de desarrollo, se llevó a cabo el test de aceptación con el objetivo de probar las expectativas del usuario final. Debido a la imposibilidad de contar con clientes reales, la prueba fue

realizada por usuarios que simularon ser cada uno de los perfiles posibles que permite el sistema: administrativo, graduado y tercero.

A través de la prueba fue posible establecer situaciones y escenarios de uso reales del software, que permitieron validar su usabilidad y funcionalidad. En el capítulo siguiente se realizará una exposición de un caso de estudio detallando las funcionalidades de la aplicación con imágenes reales del software en funcionamiento.

Capítulo 6: Caso de estudio

En este capítulo se presentará un caso de estudio que involucra los flujos principales de ejecución del sistema UNSL Cert. En particular, se expondrán 3 los flujos de ejecución: 1) desde el perfil Administrativo, 2) desde el perfil Estudiante y 3) desde el perfil Tercero. El primero mostrará cómo el administrativo crea un nuevo título para un graduado que solicita el servicio. El segundo involucra el ingreso del graduado a la plataforma para visualizar títulos que tiene cargados, dando la opción a que exporte un comprobante en formato pdf el cuál podrá ser validado por un tercero quien desea confirmar si una persona es realmente graduado con un título determinado. Finalmente, el último flujo expondrá la interacción con el sistema de una persona externa, un tercero quien requiere validar el certificado.

6.1 Perfil administrativo

Para crear un título el administrativo debe primero loguearse con un usuario y contraseña asignada, tal como se observa en la Figura 6.1.

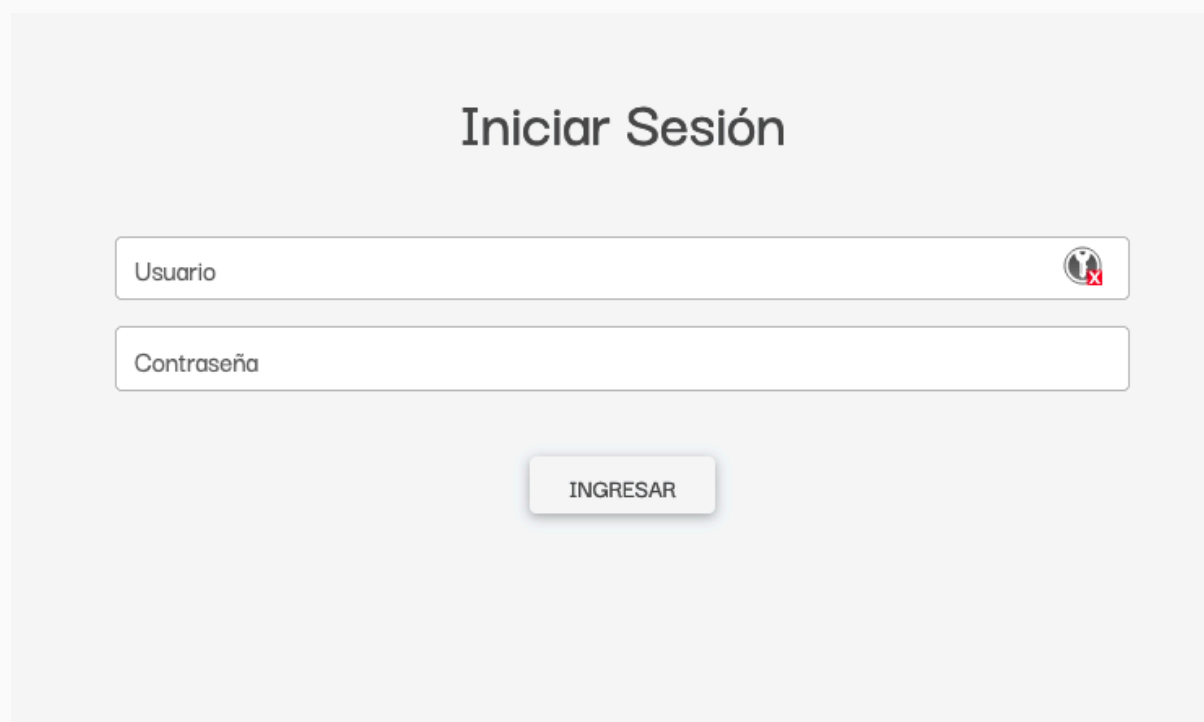
El formulario de inicio de sesión tiene un fondo gris claro. En el centro superior, el título "Iniciar Sesión" está escrito en una fuente gris oscura. Debajo del título, hay dos campos de entrada de texto blancos con bordes grises. El primer campo está etiquetado "Usuario" y tiene un ícono de usuario con una X roja a su derecha. El segundo campo está etiquetado "Contraseña". Debajo de estos campos, hay un botón rectangular con el texto "INGRESAR" en mayúsculas.

Figura 6.1: Inicio de sesión para ingresar al sistema

UNSL CERT

Buscar título **Crear Graduado** Crear título Historial

Buscar estudiante

Número de documento

Nombre

Apellido

Elija el sexo

Tipo de documento N° Documento

Elija unidad académica

Tipo de carrera

Elija una carrera

Elija el plan de estudio

N° Registro

Figura 6.2: Formulario de “crear graduado” y barra de navegación.

Una vez que ingresó, el usuario puede seleccionar diferentes opciones: “Crear Graduado”, “Crear Título”, “Buscar título” e “Historial” como se observa en la figura 6.2.

Nuevo Graduado

En primera instancia el administrativo debe ingresar los datos académicos del graduado quien solicita el servicio. Para tal fin, el administrativo, debe cargar Nombre y Apellido, Sexo, Tipo y N° Documento, Unidad Académica, Carrera y Plan de Estudios y N° Registro.

UNSL CERT

Buscar título **Crear Graduado** Crear título Historial

Buscar estudiante

Número de documento

Nombre

Apellido

Elija el sexo

DNI N° Documento

Facultad De Ciencias Fisico Matemáticas Y Naturales

Grado

Ingeniería En Informática

26/12

N° Registro

Figura 6.3: Interfaz con datos del estudiante cargados.

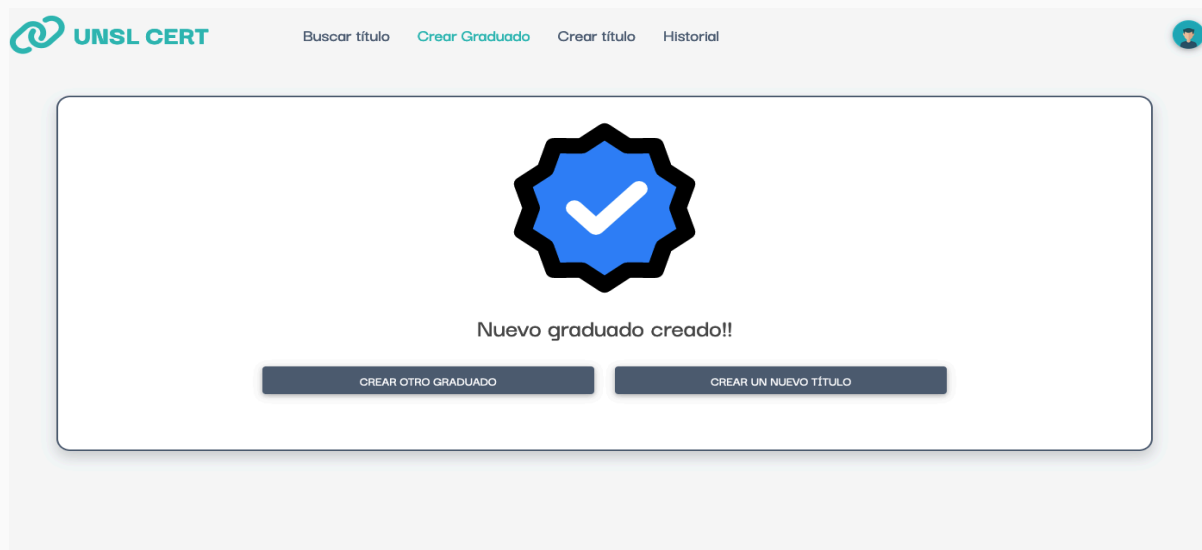


Figura 6.4: Graduado creado con éxito.

Para el caso de estudio se supondrá que el administrativo ingresa los datos de Armando Perez, graduado de la Universidad Nacional de San Luis que estudia la carrera “Ingeniería en Informática”, tal como se ve en la figura 6.3. Una vez finalizada la carga, el usuario debe presionar el botón “Guardar” para que los datos sean validados, luego el sistema automáticamente genera un usuario y contraseña por defecto para el graduado y se habilita la carga de un nuevo título vinculado a dicho graduado como se visualiza en la figura 6.4.

En el caso de que un estudiante graduado ya se encuentre cargado en el sistema y se desee cargarle otra carrera, se puede utilizar el buscador de estudiantes.

Figura 6.5: Agregar una carrera a un estudiante.

UNSL CERT

Buscar título Crear Graduado **Crear título** Historial

Crear Título

Número de documento
30123123

BUSCAR

Armando Perez
Ingeniería En Informática
DNI : 30123123 - Masculino
Registro: 45454545
Universidad Nacional De San Luis
Facultad De Ciencias Fisico Matemáticas Y Naturales

Datos del certificado

N° Oblea

CREAR TÍTULO UNIVERSITARIO

Figura 6.6: Interfaz para completar la carga de un título de un nuevo graduado.

En la figura 6.5 se visualiza el caso del estudiante Francisco Vargas que se le agrega la carrera de Ingeniería en Electrónica.

Nuevo Título

Una vez creado el estudiante en el sistema, el administrativo procede a cargar los datos del título universitario a través de la pestaña “Crear título”. Una vez allí se debe buscar al graduado por su número de documento. El sistema muestra los datos del estudiante. Además, le solicita al administrativo que coloque el número de oblea brindado por el Ministerio de Educación (ver figura 6.6).

Finalizada la carga el sistema valida los datos y procede a enviar los datos a la blockchain. Para esto es necesario crear una transacción para guardar los datos que en la brevedad serán almacenados. Una vez enviada la misma el sistema informa el envío (ver figura 6.7).

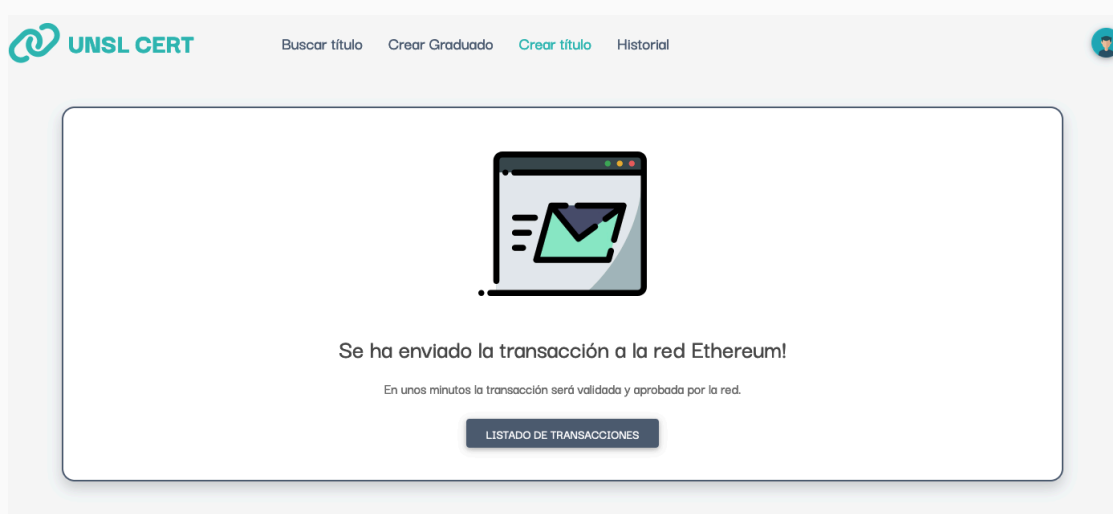
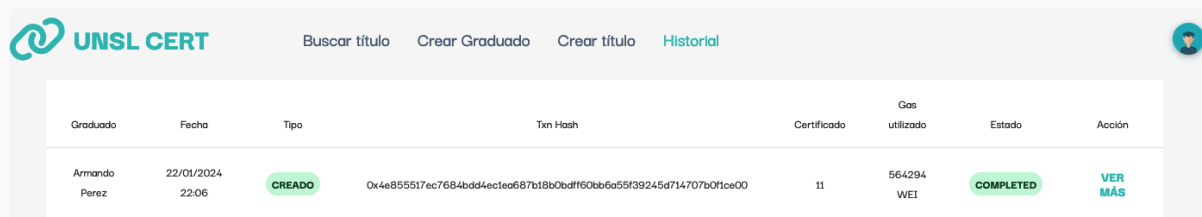


Figura 6.7: Mensaje de envío de transacción



Graduado	Fecha	Tipo	Txn Hash	Certificado	Gas utilizado	Estado	Acción
Armando Perez	22/01/2024 22:06	CREADO	0x4e855517ec7684bdd4ec1ea687b18b0bdf60bb6a55f39245d714707b0f1ce00	11	564294 WEI	COMPLETED	VER MÁS

Figura 6.8: Listado de estados de transacciones

Historial

El administrativo puede ingresar al listado de todas las transacciones a través del botón “Listado de transacciones” o bien en el menú “historial”, como se observa en la figura 6.8. El sistema informa la confirmación de la transacción y se puede visualizar el nuevo estado como “COMPLETED”.

Una vez completada la transacción, el administrativo deberá informarle al graduado que su título ya se encuentra cargado para que éste pueda visualizarlo ingresando con su usuario y contraseña a la pestaña de “mis títulos”.

6.2 Perfil graduado

Mis títulos

El estudiante puede ingresar a la pestaña de “mis títulos” donde puede visualizar los mismos, tal como se observa de la figura 6.9.



Figura 6.9: Mis títulos.

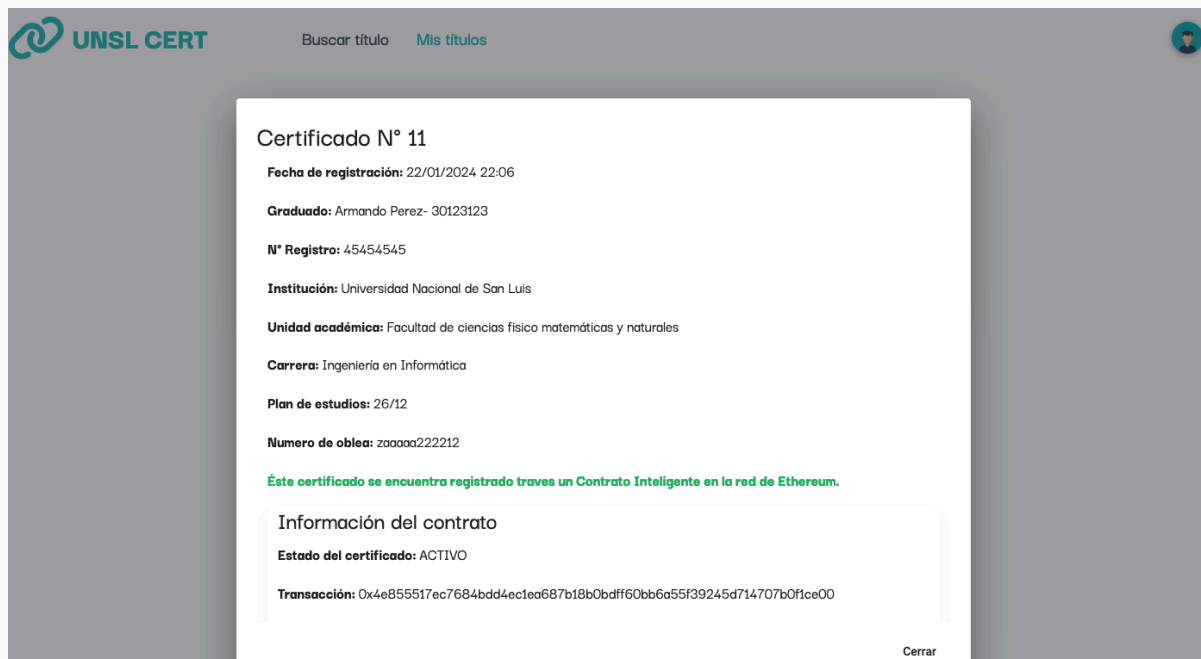


Figura 6.10: Detalle del título

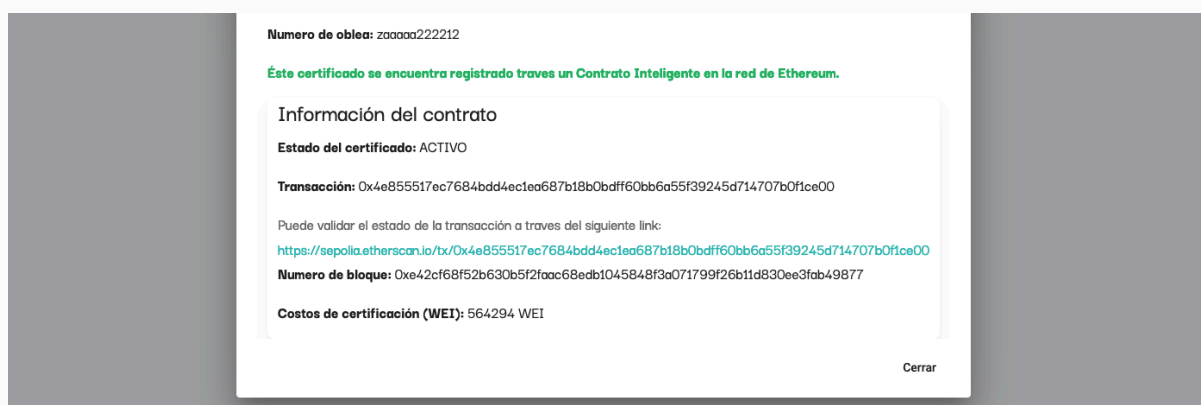


Figura 6.11: Información del contrato inteligente

Ver más

Al presionar el botón de “Ver más” el usuario puede visualizar una ventana emergente con toda la información detallada del título, como el número de oblea, datos de la institución, etc. (ver figura 6.11). El sistema informa una leyenda indicando que los datos que se visualizan están almacenados en un contrato inteligente de la red Ethereum. En el apartado de información adicional se puede visualizar datos de certificación como el estado del título, el número de transacción (que es único para cada título), los costos en Wei de la transacción, entre otros (ver figura 6.11).

También posee un link a la plataforma de Etherscan para revisar datos de la transacción, tal como se observa en la figura 6.12.

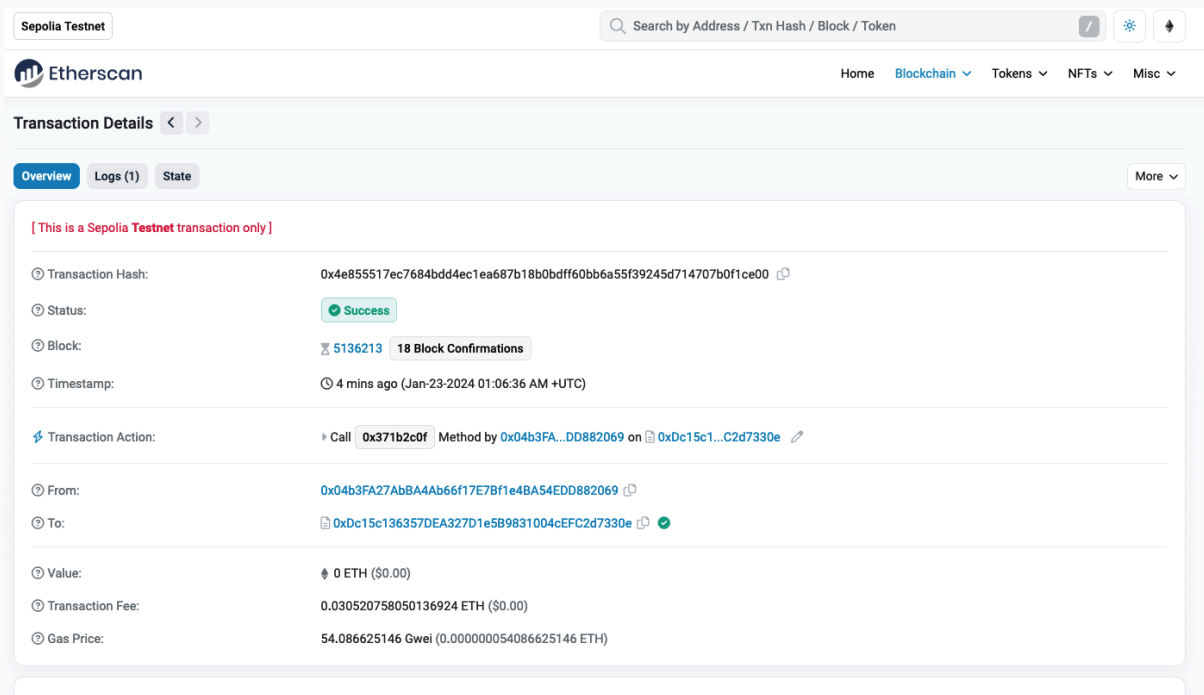


Figura 6.12: Sitio web de Etherscan.io con información detallada de la transacción y el contrato.

Exportar

Al presionar el botón de “Exportar”, el sistema descarga un archivo PDF con información relevante y un código QR donde al escanear se puede verificar la veracidad del mismo (ver figura 6.13).

Una vez descargado el archivo .pdf, puede ser enviado a cualquier persona para que pueda verificarlo escaneando el código QR, éste se dirigirá a través de un link al sistema donde le indicará si el certificado emitido es auténtico.



Figura 6.13: Documento .pdf con datos del título

6.3 Perfil tercero

Validador de títulos

En la figura 6.14 podemos visualizar el resultado de escanear el código QR, donde figuran todos los datos relevantes del título y su registro junto a un link de Etherscan.io para verificar la veracidad de los datos mostrados.

Buscar título

Cualquier usuario puede ingresar al sistema, colocar el número de documento del estudiante graduado y buscarlo para verificar la validez de su título, tal como se muestra en la figura 6.15.

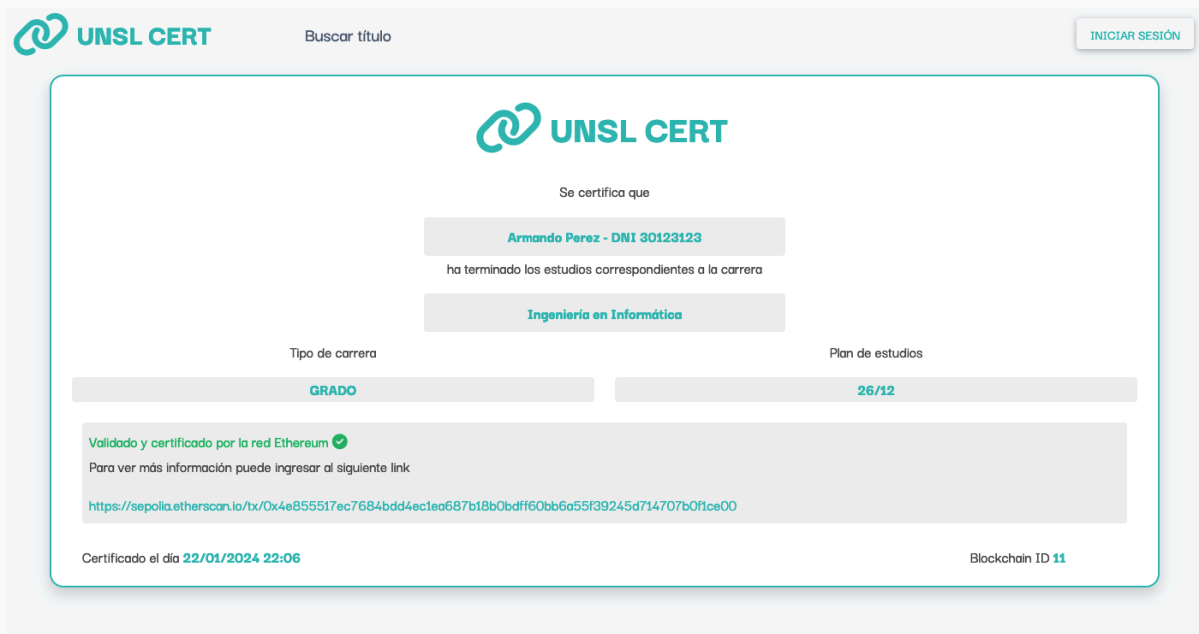


Figura 6.14: Resultado del escaneo del código QR.

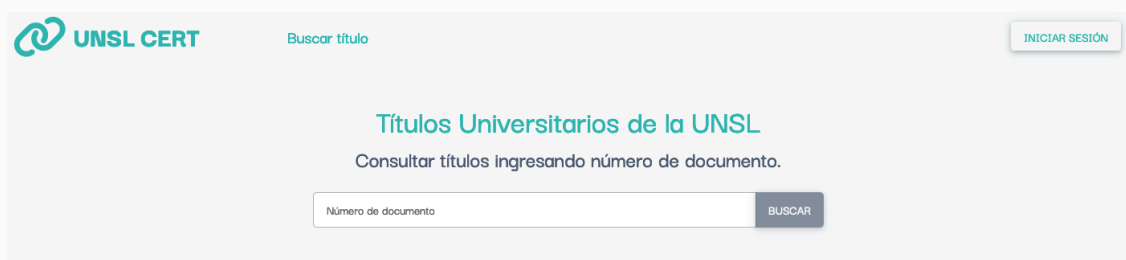


Figura 6.15: Buscar de títulos

El sistema le arrojará los resultados de la búsqueda, los datos personales y académicos del mismo junto con el listado de títulos que posea en la plataforma, en este caso cuenta con su respectivo título de ingeniero y demás datos comprobables, tal como se observa en la figura 6.16 y figura 6.17.



Figura 6.16: Resultado de búsqueda

Figura 6.17: Búsqueda de estudiante y su título

6.4 Funcionalidades adicionales

Baja de título

El administrativo tiene habilitada la funcionalidad de dar de baja los títulos, de esta manera el mismo ya no será visible y no podrá ser consultado (ver figura 6.18). Cabe destacar que esta acción no se puede deshacer.

Figura 6.18: Eliminar título

FedericoVerges	14/07/2023 18:15	BAJA	0x4f8a99c189a6554e481be42b0835a9abab3bf86927cfce5746313b74ce6b2b5c	1	110027 WEI	COMPLETED	VER MÁS
Federico Verges	14/07/2023 18:14	CREADO	0xab91e7dc48d20ed72fda957ca2fe4f4f14009e5928ef110f15d0c37db2d42763	1	564386 WEI	COMPLETED	VER MÁS

Figura 6.19: Listado de transacciones con título dado de baja

Dar de baja un título constituye la creación de una transacción que debe ser enviada a la blockchain, el proceso es idéntico a la creación del título y se puede ver en el historial, como se ve en la figura 6.19.

6.5 Casos Excepcionales

En Ethereum, una vez enviada una transacción puede aceptarse o rechazarse ya que las mismas son validadas antes de ser ejecutadas. En algunos casos excepcionales, la transacción puede perderse y quedar en un estado intermedio que suele suceder cuando los costos de GAS están demasiado altos por congestión de la red. Es por esto que el sistema realiza una trazabilidad de las transacciones enviadas, teniendo registro de cada una de ellas y conocer su estado durante el envío.

Una vez enviada la transacción, esta arroja un error al momento de enviarlo y se queda en estado de “error” (ver figura 6.20). Puede suceder que la misma ingrese en estado “pending” y luego quedarse allí en forma permanente. Presionando el botón “ver más” el sistema redirige al usuario a Etherscan.io donde se puede observar el detalle de la transacción y el estado provisto (ver figura 6.21).

Graduado	Fecha	Tipo	Txn Hash	Certificado	Gas utilizado	Estado	Acción
Armando Perez	22/01/2024 22:06	CREADO	0x4e855517ec7684bdd4ec1ea687b18b0bdf60bb6a55f39245d714707b0f1ce00	11	564294 WEI	COMPLETED	VER MÁS
Armando Perez	22/01/2024 22:03	CREADO	0x565874e0715d00744bf88caf878fed8e637ce567df4091654322247e9edc178d		WEI	ERROR	VER MÁS
Armando Perez	22/01/2024 21:56	CREADO	0x35dc0629bfeeb263d0ed6396a748c6a6c775d6081e6064c0b20871949166302d		WEI	PENDING	VER MÁS

Figura 6.20: Listado de transacciones erróneas

The screenshot shows the Etherscan website interface. At the top, the Etherscan logo is on the left, and navigation links for Home, Blockchain, Tokens, NFTs, and Misc are on the right. Below the header, the page title is "Transaction Details". A blue "Overview" tab is selected. A red warning message states: "[This is a Sepolia Testnet transaction only]". The transaction details are listed as follows:

Transaction Hash:	0x74767583d24679a6df537a8f6910e83afbfb997c136303f8923d899a9438e7c
Status:	Pending
Block:	(Pending)
Time Last Seen:	00 days 00 hr 00 min 24 secs ago (Jul-25-2023 12:43:50 AM)
From:	0x04b3FA27AbBA4Ab66f17E7Bf1e4BA54EDD882069
Interacted With (To):	0x0F395C72d539b4a629a19E3b84A9c772c082d6DD
Value:	0 ETH (\$0.00)
Max Txn Cost/Fee:	0.000000273625 ETH (\$0.00)
Gas Price:	0.0005 Gwei (0.00000000000005 ETH)

At the bottom, there is a "More Details:" section with a link "+ Click to show more". A small footnote at the very bottom explains: "A transaction is a cryptographically signed instruction that changes the blockchain state. Block explorers track the details of all transactions in the network. Learn more about transactions in our [Vocabulary Page](#)."

Figura 6.21: Transaccion pendiente en Etherscan.io

Una manera sencilla de resolver este inconveniente es eliminando dicha propiedad de la configuración, de esa manera la librería web3js se elegirá un costo por defecto obteniendola de los últimos bloques de la red. Cabe destacar que los datos son guardados de manera permanente una vez que la transacción es ejecutada exitosamente, por lo tanto si la misma entrase en estos estados conflictivos, se puede repetir el proceso de creación de título hasta que sea exitoso.

Capítulo 7: Conclusiones y Trabajos Futuros

7.1 Conclusiones

En este proyecto final integrador se ha desarrollado el sistema UNSL-CERT, basado en la tecnología blockchain y los contratos inteligentes, con el objetivo de certificar los títulos universitarios de los graduados de la UNSL.

UNSL-CERT ha logrado:

- Complementar los procesos de certificación actuales, brindando al graduado un respaldo a su título en papel.
- Ampliar la disponibilidad y portabilidad de los títulos, logrando que los mismos puedan estar disponibles para su verificación desde cualquier lugar del mundo y por cualquier entidad, incluyendo empresas empleadoras y otras instituciones educativas.
- Contribuye a evitar la falsificación de un título usando como soporte una plataforma segura para su almacenamiento y transparente para su verificación.

Después de una extensa investigación, desarrollo y pruebas, UNSL-CERT pudo registrar con éxito información sobre los títulos de los estudiantes graduados y compartirla con otros de forma segura a través de la tecnología blockchain. Además, habilitó la verificación de los títulos de manera eficiente y segura minimizando los riesgos de manipulación y traslado del papel original.

El uso de sistemas de registros distribuidos hizo posible que los datos almacenados en la plataforma permanezcan inmutables y fácilmente trazables. Por otro lado, los usuarios pudieron verificar los títulos de los demás sin tener contacto directo entre ellos debido únicamente a su confianza en la estructura de red subyacente que permitió este proceso de verificación sin problemas, siempre que tuvieran una conexión a Internet disponible en ese momento.

Este proyecto sirve como un gran ejemplo para futuros proyectos que apunten a desarrollar aplicaciones similares utilizando tecnologías modernas como los contratos inteligentes, no obstante, brinda información valiosa sobre cómo se puede lograr una implementación exitosa cuando se combinan enfoques tradicionales de ingeniería de software junto con tecnologías modernas como las técnicas de criptografía y descentralización utilizadas por las redes de cadenas de bloques.

7.2 Trabajos Futuros

Como trabajo futuro se deben evaluar los aspectos legales implicados en algunos procesos, tales como los exigidos por la entidad ministerial para la legalización de los títulos. También se pretende estudiar algunas de las limitaciones que presenta la solución propuesta, tales como los costos de las transacciones y el manejo de información sensible. En este sentido, se deben analizar las funciones hash a incluir y evaluar diferentes implementaciones en otras redes para abaratar costos de transacción.

Desarrollo con NFTs y wallets

Para mejorar la experiencia del usuario y aprovechar las últimas innovaciones tecnológicas, un aspecto a desarrollar sería la incorporación de Tokens No Fungibles (NFT) [62] para respaldar los títulos y dárselos a los graduados junto con una wallet personalizada para que puedan interactuar con ellos.

Beneficios:

1. **Mayor seguridad y autenticidad:** La utilización de NFT garantiza la autenticidad y propiedad de los títulos académicos, reduciendo el riesgo de falsificación o fraude.
2. **Experiencia del usuario mejorada:** La implementación de wallets personalizadas ofrece una experiencia más intuitiva y conveniente para los graduados, permitiéndoles gestionar fácilmente sus títulos y realizar otras actividades relacionadas.
3. **Adopción de tecnologías emergentes:** La incorporación de NFT y wallets en la plataforma demuestra el compromiso de la institución con la innovación y la adopción de tecnologías de vanguardia en el ámbito educativo.

Referencias

- [1] Drangosch, J. (2021, 7 diciembre). Una introducción a Signatura - Signatura blog. Medium. <https://blog.signatura.co/una-introducci%C3%B3n-a-signatura-8f2cd6d260f3>
- [2] Blockcerts. “Blockchain Credentials”. Blockcerts.org. Disponible en <https://www.blockcerts.org/> (última visita 11/7/2023).
- [3] Elva, L., Selimi, B. “BCERT – A Decentralized Academic Certificate System Distribution using Blockchain Technology”. International Journal on Information Technologies & Security, No 4 (vol. 12), 103. 2020.
- [4] Camilo, R. P. J. (2020). Diplomas académicos sobre una plataforma Blockchain. Repositorio Institucional Séneca. <http://hdl.handle.net/1992/48707>
- [5] Blockchain Federal Argentina, <https://bfa.ar/> (última visita 13/7/2023).
- [6] Ministerio de Educación de la Nación, Secretaría de Políticas Universitarias. Registro Público de Graduados Universitarios. <https://registrograduados.siu.edu.ar/> (última visita 13/7/2023).
- [7] Verges F., Garis A., Tissera C. “Sistema de Certificación de Títulos Universitarios a través de Blockchain”. Trabajos Estudiantiles. CoNaIISI 2021.
- [8] Tanenbaum, A. S., Guerrero, G., & Velasco, Ó. A. P. (1996). Sistemas operativos distribuidos (No. QA76. 76063. T35. 3 1996.). México: Prentice Hall.
- [9] Hoeger, H. (2011). Introducción a la computación paralela. *Centro Nacional de Cálculo Científico Universidad de Los Andes, Mérida (Venezuela)–Ce-CalCULA*, 48.
- [10] Colaboradores de Wikipedia. (2022, 3 enero). Árbol de Merkle. Wikipedia, la Enciclopedia Libre. https://es.wikipedia.org/wiki/%C3%81rbol_de_Merkle
- [11] Introducción a Ethereum | ethereum.org. (s. f.-b). ethereum.org. <https://ethereum.org/es/developers/docs/intro-to-ethereum#what-is-a-blockchain>
- [12] Brownworth, A. (s. f.). Blockchain demo. Recuperado 22 de febrero de 2024, de <https://andersbrownworth.com/blockchain>
- [13] Academy, B. (2023, 21 abril). Cuántos tipos de blockchain existen. Bit2Me Academy. <https://academy.bit2me.com/cuantos-tipos-de-blockchain-hay/>
- [14] Hyperledger - The Open Global Ecosystem for Enterprise Blockchain. (s. f.). <https://www.hyperledger.org/>
- [15] Díaz, F. J. (2022). Protocolos de consenso. <http://sedici.unlp.edu.ar/handle/10915/144937>
- [16] Academy, B. (2023a, marzo 3). ¿Qué es Prueba de trabajo / Proof of Work (PoW)? Bit2Me Academy. <https://academy.bit2me.com/que-es-proof-of-work-pow/>

- [17] Academy, B. (2023c, junio 9). ¿Qué es Proof of Stake (PoS)? Binance Academy.
<https://academy.binance.com/es/articles/proof-of-stake-explained>
- [18] Bitcoin - Dinero P2P de código abierto. (s. f.). <https://bitcoin.org/es/>
- [19] Introducción a ether | ethereum.org. Recuperado 26 de febrero de 2024, de
<https://ethereum.org/es/developers/docs/intro-to-ether#what-is-a-cryptocurrency>
- [20] Centro de aprendizaje | ethereum.org. Recuperado 26 de febrero de 2024, de
<https://ethereum.org/es/learn>
- [21] Takenobu, T. T. (2018). Ethereum EVM illustrated: exploring some mental models and implementations (0.1.1).
https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf
- [22] How does Ethereum work, anyway? Recuperado 26 de febrero de 2024 de
<https://www.preethikasireddy.com/post/how-does-ethereum-work-anyway>
- [23] Cuentas de Ethereum | Ethereum.org. Recuperado 26 de febrero de 2024, de
<https://ethereum.org/es/developers/docs/accounts>
- [24] Objeto JSON de una transacción [Imagen]. Transacciones | Ethereum.org. Recuperado 26 de febrero de 2024, de <https://ethereum.org/es/developers/docs/transactions>
- [25] Máquina virtual de Ethereum (EVM) | ethereum.org. Recuperado 26 de febrero de 2024, de. <https://ethereum.org/es/developers/docs/evm>
- [26] Opcodes para la EVM | ethereum.org. Recuperado 26 de febrero de 2024, de
<https://ethereum.org/es/developers/docs/evm/opcodes>
- [27] Gas y tarifas | ethereum.org. Recuperado el 6 de septiembre de 2021 de
<https://ethereum.org/es/developers/docs/gas>
- [28] Contratos inteligentes | ethereum.org. Recuperado 26 de febrero de 2024, de.
<https://ethereum.org/es/smart-contracts>
- [29] Solidity — Solidity 0.8.24 documentation. (s. f.). <https://docs.soliditylang.org/en/v0.8.24/>
- [30] Vyper. (s. f.). Vyper Documentation. <https://docs.vyperlang.org/en/latest/index.html>
- [31] Anatomía de los contratos inteligentes | ethereum.org. (s. f.). ethereum.org.
<https://ethereum.org/es/developers/docs/smart-contracts/anatomy>
- [32] Compilación de contratos inteligentes | ethereum.org. (s. f.). ethereum.org.
<https://ethereum.org/es/developers/docs/smart-contracts/compiling>
- [33] Especificación de Application Binary Interface — Documentación de Solidit. (s. f.).
<https://solidity-es.readthedocs.io/es/latest/abi-spec.html> (última visita 13/05/2023).
- [34] JSON-RPC 2.0 specification. (s. f.). <https://www.jsonrpc.org/specification>
- [35] Cadenas laterales | ethereum.org. (s. f.-b). ethereum.org.
<https://ethereum.org/es/developers/docs/scaling/sidechains>
- [36] Whitepaper. (s. f.). <https://polygon.technology/papers/pol-whitepaper>

- [37] Qué es BFA. (s.f).Blockchain Federal Argentina. Recuperado el 1 de enero de 2024. <https://bfa.ar/bfa/que-es-bfa>
- [38] Comparación entre web2 y web3 | ethereum.org. (s. f.). ethereum.org. <https://ethereum.org/es/developers/docs/web2-vs-web3>
- [39] Introducción a las DApps | ethereum.org. (s. f.). ethereum.org. <https://ethereum.org/es/developers/docs/dapps>
- [40] Títulos académicos. (s. f.). Blockchain Federal Argentina. Recuperado 1 de enero de 2024, de <https://bfa.ar/blockchain/casos-de-uso/titulos-academicos>
- [41] JavaScript | MDN. (2023, 24 julio). Disponible en <https://developer.mozilla.org/es/docs/Web/JavaScript> (última visita 1/8/2023).
- [42] Typescript Documentation. “TypeScript: JavaScript With Syntax For Types”. Disponible en <https://www.typescriptlang.org/docs/handbook/utility-types.html> (última visita 13/7/2023).
- [43] Angular. (s. f.). <https://angular.io/docs> (última visita 1/8/2023).
- [44] Acerca | Node.js. (s. f.). Node.js. <https://nodejs.org/es/about> (última visita 13/5/2023).
- [45] Express - Node.js Web Application Framework. (s. f.). <https://expressjs.com/> (última visita 13/05/2023).
- [46] Sequelize. (s. f.). Feature-rich ORM for modern TypeScript & JavaScript. <https://sequelize.org/> (última visita 13/05/2023).
- [47] PostgreSQL: about. (s. f.). The PostgreSQL Global Development Group. <https://www.postgresql.org/about/> (última visita 13/05/2023).
- [48] WebSockets - referencia de la API Web | MDN. (2023, 21 febrero). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/API/WebSockets_API
- [49] Truffle | Overview - Truffle Suite. (s. f.). <https://trufflesuite.com/docs/truffle/> (última visita 13/05/2023).
- [50] Ganache | Overview - Truffle Suite. (s. f.). <https://trufflesuite.com/docs/ganache/> (última visita 13/05/2023).
- [51] Web3.js - Ethereum JavaScript API — Web3.js 1.0.0 documentation. (s. f.). <https://web3js.readthedocs.io/en/v1.10.0/#> (última visita 13/05/2023).
- [52] Infura API documentation | INFURA. (2024, 28 febrero). <https://docs.infura.io/api>
- [53] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. Design Patterns Elements of Reusable Object Oriented Software. 1-358. Addison Wesley Longman Inc. USA (1998).
- [54] Fowler, Martin (2010). Data Transfer Object. Patterns of Enterprise Application Architecture.
- [55] Patrones comunes — documentación de Solidity - Restringiendo el acceso. (s. f.). Solidity - Docs. Recuperado 12 de febrero de 2024, de <https://solidity-es.readthedocs.io/es/latest/common-patterns.html#restringiendo-el-acceso>

- [56] Docker: Accelerated Container Application Development. (2024, 23 de enero). Docker.
<https://www.docker.com/>
- [57] AWS re:Invent 2023 - Compute Innovation Talk. (s. f.). Amazon Web Services, Inc.
Recuperado 5 de enero de 2024, de https://aws.amazon.com/es/ec2/?nc2=h_ql_prod_fs_ec2
- [58] AWS | Almacenamiento de datos seguro en la nube (S3). (s. f.). Amazon Web Services, Inc.
https://aws.amazon.com/es/s3/?nc2=h_ql_prod_fs_s3
- [59] Consejo Profesional de Ciencias Informáticas de la Provincia de Buenos Aires. (s. f.).
Recuperado 10 de enero de 2024, de <https://cpciba.org.ar/honorarios>
- [60] AWS Pricing Calculator. (s. f.-b). Recuperado 5 de enero de 2024, de
<https://calculator.aws/#/createCalculator/ec2-enhancement>
- [61] Gráfico del precio del gas. (s. f.). Polygonscan. Recuperado 5 de enero de 2024, de
<https://polygonscan.com/chart/gasprice>
- [62] Tokens no fungibles (NFT) | ethereum.org. (s. f.). ethereum.org. Recuperado 23 de marzo de 2024, de <https://ethereum.org/es/nft/#what-are-nfts>