UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DESCENTRALIZADO PARA EL REGISTRO Y CONSULTA DE RUTAS DE ESCALADA EN ETHEREUM

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

CONSTANTINO IGNACIO HERNÁNDEZ GÓMEZ

PROFESOR GUÍA:
FERNANDO NOWAJEWSKI VEGA

MIEMBROS DE LA COMISIÓN:
CESAR AZURDIA MEZA
ANDRÉS CABA RUTTE

SANTIAGO DE CHILE
2022

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: CONSTANTINO IGNACIO HERNÁNDEZ GÓMEZ
FECHA: 2022
PROFESOR GUÍA: FERNANDO NOWAJEWSKI VEGA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DESCENTRALIZADO PARA EL REGISTRO Y CONSULTA DE RUTAS DE ESCALADA EN ETHEREUM

Cuando se practica el deporte de la escalada en roca en un entorno natural se debe conocer la ubicación de las rutas que se encuentran adaptadas para esta actividad, así como a qué disciplina de escalada corresponde dicha ruta y el nivel de dificultad que se le designa en su creación, expresado mediante escalas estandarizadas.

En la actualidad se distinguen dos formas de presentar los datos de estas rutas: libros guía (o "topos") que describen sectores específicos, presentados en un formato digital o físico; y bases de datos comprensivas que incluyen, al menos, funcionalidades básicas como geolocalización y filtros de búsqueda. Este trabajo se concentra en la segunda, postulando que existe un problema latente de centralización en las plataformas que se proveen indexación de rutas de escalada.

Estas bases de datos comprensivas se encuentran localizadas en servidores de propiedad privada que, si bien buscan extender el acceso a dichos datos a la comunidad, presentan un riesgo inherente en su centralización: estas bases de datos indexadas podrían verse comprometidas por situaciones que afecten la integridad de sus servidores, inviabilidad económica del proyecto y factores externos como *blackouts* de los servicios de *hosting* que las alojan o censura. Además, el acceso a los datos de su plataforma es restringido, dificultando el desarrollo libre de herramientas que hagan uso de estas bases de datos.

Se plantea como hipótesis de este proyecto que la tecnología *blockchain* puede proveer una solución para el registro indexado y consulta de rutas de escalada que prevenga los problemas latentes descritos y ofrezca acceso libre a dichos datos. Esta tecnología corresponde a un registro distribuido en los nodos de su red, y en sus implementaciones más populares, como Ethereum, cuenta con capacidades de *Smart Contracts*, código computacional alojado en el registro distribuido que define mecanismos que alteran el estado compartido de la red.

El trabajo presentado a continuación detalla el diseño y la implementación de un sistema descentralizado, operando en la *blockchain* de Ethereum, para el registro y consulta de rutas de escalada. El proyecto comprende el proceso de diseño de *Smart Contracts*, la iteración del diseño para la incorporación de optimizaciones y el despliegue de los módulos diseñados en redes de prueba para la estimación de costos de despliegue y operacionales.

Se espera que la plataforma resultante pueda ser incorporada a soluciones ya existentes de manera que efectivamente se distribuya la información de las rutas de escalada en el mundo en concordancia con la cultura del deporte, y que garantice su accesibilidad siempre que la *blockchain* de Ethereum se encuentre operacional. Se comprueba con este trabajo la viabilidad de este tipo de soluciones y se proponen trabajos futuros para el desarrollo de herramientas descentralizadas que aprovechen la transparencia y acceso público de estos datos.

DEDICADO A MI TATA

Qué ganas de poder verte orgulloso de mi en este momento,

te extraño.

AGRADECIMIENTOS

A mi familia por todo su apoyo y cariño.

A mis amigos por ser mi inspiración, son mis ídolos.

A todas las personas que con lo que aportan al mundo han motivado en mí un ímpetu por dejar una contribución positiva.

Y finalmente, pero no menos importante,

gracias a ti por leer mi trabajo.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

Capítulo 1. INTRODUCCIÓN

1.1 CONTEXTO Y JUSTIFICACIÓN

1.1.1 Escalada

El 3 de agosto de 2021 debutó la Escalada como deporte Olímpico en los Juegos Olímpicos de Tokio 2020 con tres disciplinas: *Speed Climbing*, Escalada Deportiva (*Sport Climbing*) y *Bouldering*. La escalada es un deporte en el que los atletas escalan formaciones rocosas, o volúmenes artificiales anclados a una pared inclinada cuando se practica en interiores, en rutas preestablecidas. Si se practica competitivamente el objetivo es completarlas en la menor cantidad de intentos o menor tiempo posibles, dependiendo de la disciplina, sin embargo, la escena competitiva de la escalada es pequeña comparada con su práctica con fines recreacionales.

La *International Federation of Sport Climbing* (IFSC), organización a la que adhieren 96 federaciones nacionales de escalada en el mundo [1], en su reporte anual de 2019 detalla los resultados de uno de los pocos estudios realizados para la estimación del número de escaladores en el mundo, declarando que existen alrededor de 44.5 millones de personas que practican el deporte [2], y más de 140 países con muros de escalada artificiales.

Mientras tanto, en el contexto nacional, las cifras son mayoritariamente desconocidas, pero los últimos estimados presentados por el vicepresidente de la Federación Nacional de Escalada en 2018, Len Ríos, indican que se han duplicado el número de practicantes del deporte entre los años 2013 y 2018, llegando a más de 5000 escaladores a nivel nacional. Además, el mismo año Ríos declaraba el auge de los muros de escalada artificiales en el país, pasando de tan sólo dos en la capital en 2006, a cerca de 300 a nivel nacional en 2018 [3].

Cuando la escalada es practicada en formaciones rocosas naturales, dependiendo de la disciplina que se quiera practicar y de las características específicas de la ruta, el equipamiento requerido para la actividad será distinto. Por ejemplo, en el caso de una ruta de *free climbing* será necesario el uso de cuerdas, cintas y mosquetones; mientras que una ruta de *bouldering*, modalidad en la que se escalan volúmenes de menor altura, no necesita ese equipamiento, pero en su lugar se necesitan *crash pads*: colchonetas para contrarrestar una posible caída debido a la ausencia de cuerdas que aseguren al escalador.

Las rutas de escalada además cuentan con un nivel de dificultad expresado por medio de escalas estandarizadas, variando la selección de la escala según el lugar geográfico en que se encuentra la ruta o de dónde proviene quien la propone.

Antiguamente esta información era propagada por medio de boca-a-boca, y eventualmente la organización comunitaria llevaba al desarrollo de guías de escalada conocidas como “Topo”.

Un Topo es una guía compilando información de una o más rutas de escalada en un sector determinado. Tradicionalmente eran editorializadas en libros, sin embargo, la era de la información las ha modernizado y se han levantado plataformas digitales para el registro indexado y consulta de rutas de escalada.

1.1.2 Plataformas Indexadas

Son de particular interés de este trabajo aquellas plataformas que indexan o categorizan rutas de escalada de acuerdo a su ubicación geográfica en bases de datos comprensivas. Estas plataformas ofrecen en la actualidad servicios gratuitos de registro e indexación de rutas de escalada, siendo *theCrag.com* la con la mayor cantidad de rutas de escalada en el mundo con más de 1 millón de rutas registradas e indexadas geográficamente en un mapa interactivo [4].

Destacan otros sitios como *ukclimbing.com* que dice proveer información de más de 582 mil rutas en el mundo en una colección digital de libros guías del mundo, así como datos indexados y visualizados en un mapa interactivo [5]; y *mountainproject.com* con datos de cerca de 270 mil rutas con mayor foco en rutas ubicadas en Estados Unidos.

A continuación, se presentan datos que describen el número de visitantes y la demografía de los 6 sitios web de escalada más populares en el mundo, entre los que se encuentran las plataformas mencionadas anteriormente, de acuerdo a *Similarweb LTD* en su sitio *similarweb.com* [6] donde se analizan más de 100 millones de sitios web en más de 210 industrias a nivel global [7]. Estos datos corresponden al mes de Julio de 2022.

Sitio Web	**Ranking Mundial**	**Número de Visitas Mensuales**	**Demografía**	
			Nacionalidad	**Grupo Etario**
Mountainproject.com	19,346	2.1 millones	• 83.51% E.E.U.U. • 6.83% Canadá • 1.12% Reino Unido • 0.62% Italia • 0.58% Alemania • 7.33% Otras	• 18-24 : 17.53% • 25-34 : 30.57% • 35-44 : 20.41% • 45-54 : 15.76% • 55-64 : 10.06% • 65+ : 5.67%
Ukclimbing.com	30,373	1.8 millones	• 75.15% Reino Unido • 5.67% E.E.U.U. • 1.78% Canadá	• 18-24 : 15.83% • 25-34 : 28.92% • 35-44 : 21.23%

			• 1.73% Irlanda • 1.71% Francia • 13.96% Otras	• 45-54 : 16.89% • 55-64 : 10.39% • 65+ : 6.75%
Mountain-forecast.com	43,897	1.8 millones	• 34.50% E.E.U.U. • 12.12% Polonia • 10.49% Reino Unido • 9.21% Australia • 4.89% Canadá • 28.81% Otras	• 18-24 : 16.32% • 25-34 : 27.74% • 35-44 : 20.64% • 45-54 : 15.68% • 55-64 : 11.97% • 65+ : 7.65%
Climbing.com	52,865	1.3 millones	• 58.43% E.E.U.U. • 6.16% Canadá • 5.46% Reino Unido • 2.94% Alemania • 1.55% Australia • 25.46% Otras	• 18-24 : 17.19% • 25-34 : 30.06% • 35-44 : 20.16% • 45-54 : 15.41% • 55-64 : 10.52% • 65+ : 6.65%
Summitpost.org	48,384	1.3 millones	• 68.08% E.E.U.U. • 4.17% Canadá • 4.12% Reino Unido • 2.16% Italia • 1.87% Suiza • 19.59% Otras	• 18-24 : 15.48% • 25-34 : 26.22% • 35-44 : 20.00% • 45-54 : 16.50% • 55-64 : 13.41% • 65+ : 8.38%
Thecrag.com	39,247	1 millón	• 27.33% Australia • 11.90% Alemania • 7.68% E.E.U.U. • 6.71% Canadá • 6.57% Suiza • 39.81% Otras	• 18-24 : 15.48% • 25-34 : 26.22% • 35-44 : 20.00% • 45-54 : 16.50% • 55-64 : 13.41% • 65+ : 8.38%

Tabla 1: Datos de tráfico y demografía de los 6 sitios web de escalada más visitados

Se distingue que dentro de los 6 sitios web más populares de escalada en el mundo, *theCrag.com* recibe el tráfico más heterogéneo en cuanto a las nacionalidades de sus visitantes. Se le atribuye este hecho al gran número de rutas registradas en su plataforma, con orígenes en todo el mundo. Además, se reconoce que en todos los casos el grupo etario más grande de estos sitios es el que comprende entre los 25 y 34 años de edad.

1.1.3 Descripción del Problema

El desarrollo de rutas de escalada naturales es un proceso informal y con pocas restricciones, donde cualquiera con las herramientas y el conocimiento adecuado puede adaptar formaciones rocosas para habilitarlas para la escalada [8]. En otras palabras, se puede describir el proceso de creación y propagación de la información de nuevas rutas como un sistema descentralizado.

Sin embargo, las plataformas digitales que ofrecen un servicio de registro de guías de escalada centralizan el proceso: los desarrolladores de rutas deben tener acceso y permisos dentro de la plataforma, y el resultado final son datos centralizados en servidores privados que, si bien ofrecen acceso de forma gratuita, podrían verse comprometidos.

La centralización de esta información en estas plataformas responde a la necesidad que tienen los usuarios de tener confianza en una entidad que garantice la veracidad e integridad de los datos registrados, sacrificando la propiedad y, en caso de eventualidades como desastres naturales, ciberataques o falta de financiamiento, su accesibilidad. Además, las plataformas digitales pueden operar de manera arbitraria en la aprobación, administración y mantenimiento del sistema sin la necesidad de transparentar su lógica interna.

El desarrollo de rutas de escalada abierto, público, descentralizado y aprobado por el consenso de la comunidad está en discrepancia con las plataformas de carácter privado y centralizado para su registro, y este es el problema que se pretende abordar en este trabajo. Más específicamente, se busca proponer una solución alternativa al registro digital de rutas de escalada que haga uso de tecnologías descentralizadas que garantice la copropiedad, accesibilidad y administración descentralizada de estos datos.

1.2 ALCANCES

1.2.1 Hipótesis General

Una plataforma descentralizada desplegada en la *blockchain* de *Ethereum* [9] es capaz de resolver los problemas latentes descritos de accesibilidad y propiedad de los datos distribuyendo su almacenamiento, así como las funciones que modifican este registro, en los nodos de su red.

El trabajo descrito en el presente informe hace uso de la tecnología *Blockchain* de la red de *Ethereum*, aprovechando su capacidad de despliegue de *Smart Contracts* para el montaje de un sistema descentralizado que aborde esta problemática. Se pretende de esta manera explorar la viabilidad del uso de dicha plataforma en este contexto, así como situaciones ad-hoc tras un análisis del sistema diseñado. En otras palabras, *el presente trabajo corresponde a un caso de estudio de la blockchain de Ethereum*.

Se comprobará el cumplimiento de esta hipótesis general demostrando que existe libre acceso a los datos registrados una vez se despliegan los Smart Contracts diseñados, así como una correcta indexación de las rutas de escalada que se incorporan.

1.2.2 Alcances Específicos

En este trabajo se explora el proceso de diseño de una plataforma descentralizada agnóstica de un *frontend* o interfaz de usuario específica. El motivo de esta decisión es que el ejercicio de diseño de un sistema como el presentado motiva la transparencia en los procesos fundamentales para el registro de una ruta de escalada en la *blockchain*.

El trabajo realizado es de carácter *open source* y propone un estándar para el registro de datos de rutas de escalada en Ethereum transparente y público. Existen parámetros de despliegue que permiten abrir el sistema diseñado y así contar con un registro público con capacidades de control de escritura, para la moderación de una plataforma a nivel de *Smart Contract*.

Esto introduce un grado de centralización en torno a los múltiples proyectos que quisieran abordar el montaje de una plataforma con las características descritas en este trabajo, donde se tiene una jerarquía para la gestión de permisos de escritura.

Este nivel de centralización es beneficioso para la organización de comunidades locales en torno a sus propias instancias desplegadas del sistema propuesto, manteniendo el registro transparente y el almacenamiento de sus datos distribuido en los nodos de la red de Ethereum, resolviendo uno de los principales problemas descritos en este trabajo pertinente a la accesibilidad de los datos: esta solo se vería mermada si la red de Ethereum dejara de estar disponible.

Debe entenderse este trabajo, entonces, como una guía detallada del diseño y de los parámetros y procesos del despliegue de la plataforma descrita en la *blockchain* de Ethereum, pero debido a los costos calculados del proyecto y a la ausencia de financiamiento previo, esta implementación es validada solamente en redes de prueba locales.

Además, se abarcará superficialmente la integración del sistema diseñado para la *blockchain* con interfaces de usuario web a través de librerías y *frameworks* estándar en la industria, sin embargo, esta discusión se escapa de los alcances propuestos para este trabajo, por lo que se define sólo a grandes rasgos cómo se habilita esta interacción.

1.3 OBJETIVOS

1.3.1 Objetivos Generales

- Diseñar una solución descentralizada al problema de registro y consulta virtual de rutas de escalada e Implementar un prototipo del sistema base en Ethereum.

- Mejorar la accesibilidad y copropiedad de los datos en concordancia con la cultura en que se gestan las rutas.

1.3.2 Objetivos Específicos

- Modelar un registro descentralizado para el almacenamiento de datos de rutas de escalada que minimice los costos de transacción en *Smart Contracts*.
- Incorporar un sistema de indexación geográfica de rutas para su posterior identificación en plataformas descentralizadas.
- Generar documentación que facilite el desarrollo de herramientas independientes que interactúen con los *Smart Contracts* del Sistema diseñado.

1.4 METODOLOGÍA

A continuación, se detalla la metodología utilizada a lo largo del proceso de diseño de la plataforma. Este proceso se desarrolla de manera iterativa, primero con implementaciones *naive* que resuelvan el problema planteado, para luego iterar sobre el diseño propuesto con optimizaciones de bajo nivel.

Las principales optimizaciones son a nivel estructural y en la codificación de los datos para minimizar el uso de *storage* dentro de los *Smart Contracts* de acuerdo con las consideraciones de costos detalladas en el marco teórico de este informe.

1.4.1 Definición del Estado del Arte

Previo al proceso de diseño como tal se realiza una revisión bibliográfica de las tecnologías que sustentan la solución. Se estudian los detalles de las implementaciones de la *blockchain* de Ethereum, su funcionamiento y la escritura y despliegue de *Smart Contracts*. Se estudian también patrones de diseño que permiten optimizar los contratos para la minimización de costos de transacción.

1.4.2 Diseño e Implementación de Base de Datos fundamental en Ethereum

En esta primera etapa de diseño se toman en consideración las limitaciones de la *blockchain* de Ethereum para describir una infraestructura base de *Smart Contracts* que satisfaga como mínimo la identificación geográfica de rutas de escalada.

Adicionalmente, del estudio de plataformas digitales para el registro de rutas de escalada disponibles actualmente se determina qué información adicional es de vital importancia y será registrada en la *blockchain*.

Una vez diseñada una propuesta, es desplegada en una red de prueba (*testnet*) para la verificación de su correcto funcionamiento y de la integridad de los datos registrados. Además, con el

despliegue de prueba se obtiene un estimado del costo de transacción de inicializar el contrato en la red y de ejecutar sus funciones, con el cual se puede calcular el costo aproximado en dólares de desplegar y operar instancias de dicho contrato.

1.4.3 Diseño e Implementación de la Aplicación Descentralizada

En la segunda etapa de diseño del sistema se definen las interfaces mediante las cuales se modificará el registro de la *blockchain*.

En particular, se definen tanto *Smart Contracts* que hagan de interfaz entre contratos internos y externos al sistema, así como las interfaces de usuario que expondrán la información del registro y los métodos disponibles para su modificación.

Se implementan los módulos diseñados en la *testnet* y de la misma forma que en la primera etapa de diseño se evalúan los costos de despliegue y operacionales en distintos escenarios.

1.4.4 Análisis del Sistema

Con los datos recopilados en los procesos de despliegue de prueba durante el proceso de diseño se itera sobre la solución propuesta y se determina la complejidad de la solución en base a si los costos a los que converge la nueva iteración se justifican como gastos operacionales de mantener un registro descentralizado. Para ello se evalúan tanto costos de despliegue como costos de la ejecución de funciones esenciales para el funcionamiento del sistema completo integrado.

Finalmente se analizan los resultados obtenidos de las simulaciones del diseño final propuesto, se comparan resultados de distintas iteraciones del diseño para ilustrar las consideraciones de diseño importantes en el ahorro de costos del sistema y se comenta sobre el diseño final propuesto.

Capítulo 2. MARCO TEÓRICO

2.1 CONCEPTOS FUNDAMENTALES

2.1.1 Criptografía

Durante el siglo XX se gestó y desarrolló la criptografía: el estudio de esquemas y técnicas utilizados para transformar un mensaje legible en uno distinto para el enmascaramiento del mensaje, proceso conocido como *encriptación*.

2.1.1.1 Encriptación Simétrica

Las primeras técnicas desarrolladas por la criptografía son conocidas como *encriptación simétrica* debido a que tanto el emisor como el receptor del mensaje deben conocer un "secreto" o *llave* compartida para su encriptación y desencriptación. Un modelo de encriptación simétrica tiene cinco componentes fundamentales [10]:

1. El mensaje original, o ***plaintext***, sobre el que se aplica el algoritmo.
2. El algoritmo de encriptación, que realiza operaciones de sustitución y tranformación sobre el plaintext.
3. Una llave secreta, un valor independiente del plaintext que determina las operaciones de sustitución y transformación que el algoritmo producirá, modificando por lo tanto el resultado de la encriptación.
4. El mensaje encriptado, o ***ciphertext***, que aparentará un set aleatorio de datos.
5. El algoritmo de desencriptación, que revertirá las operaciones del algoritmo de encriptación para hacer el mensaje legible nuevamente.

Más formalmente, sea $X = [X_1, X_2, \ldots, X_M]$ el plaintext que se quiere encriptar de largo M elementos de un alfabeto finito, por ejemplo, el dominio de 26 letras o el alfabeto binario {0,1}. Se genera una llave $K = [K_1, K_2, \ldots, K_J]$ de largo J para la encriptación. Esta llave debe ser compartida por el emisor y el receptor del mensaje. Usando el mensaje X y la llave K como entradas del algoritmo de encriptación se obtiene el ciphertext $Y = [Y_1, Y_2, \ldots, Y_N]$ de largo N. Simplificando en la siguiente notación:

$$\boldsymbol{Y = E(K, X)}$$

que indica que el mensaje Y es producido utilizando el algoritmo de encriptación E utilizando el plaintext X y la llave K como entradas. El receptor del mensaje, por su parte recuperará el mensaje con el algoritmo de desencriptación:

$$\boldsymbol{X = D(K, Y)}$$

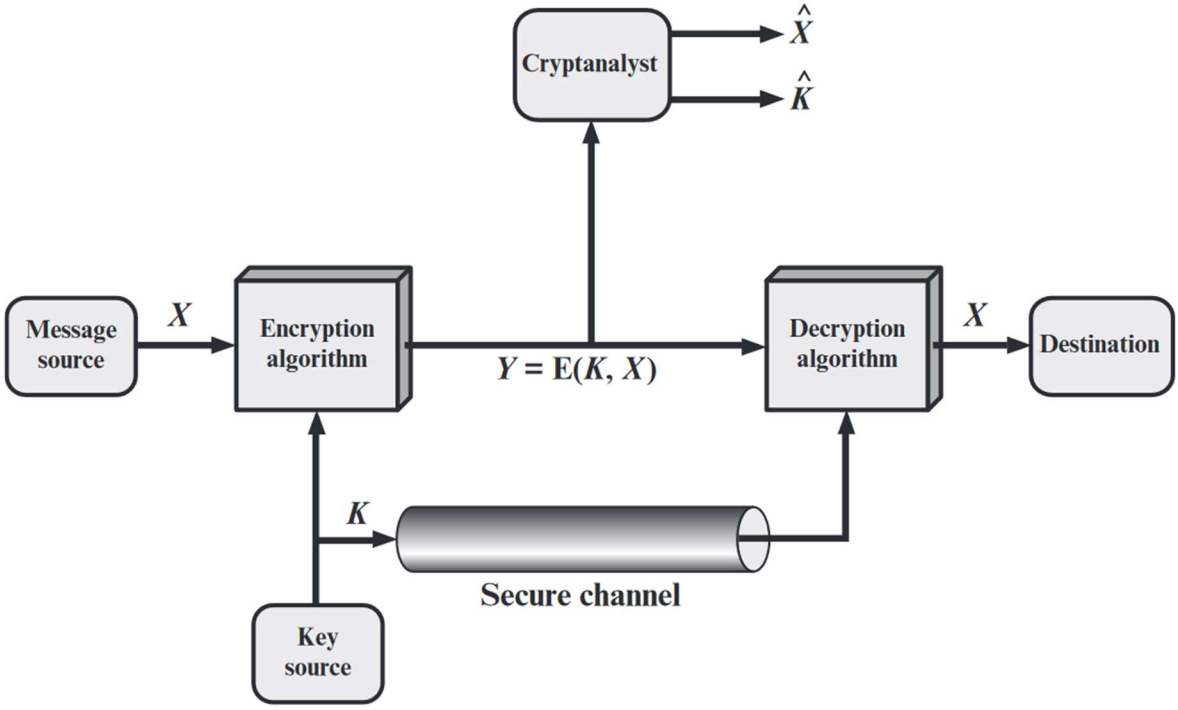


Figura 1: Modelo de encriptación simétrica [10]

Existen dos requisitos de vital importancia para el uso seguro de encriptación simétrica: un algoritmo de encriptación fuerte, es decir, que aun conociendo el algoritmo e incluso teniendo acceso a pares de mensajes y sus encriptaciones un oponente no pueda desencriptar mensajes ni determinar cuál fue la llave utilizada para su encriptación y, en segundo lugar, que tanto el emisor como el receptor tengan copias de la llave secreta, la cual ha de ser compartida a través de un canal seguro.

Esta dependencia de la encriptación simétrica de un canal seguro a través del cual se comparta la llave de encriptación representa su mayor vulnerabilidad, en particular cuando se requiere de encriptación para la comunicación por canales digitales.

2.1.1.2 Encriptación Asimétrica

Dichas vulnerabilidades motivan el desarrollo de técnicas de *encriptación asimétrica* [10], llamada de dicha manera debido a la utilización de dos llaves para la encriptación y desencriptación de mensajes, en lugar de solo una. También se conoce como *encriptación de llave pública*, y parte de su innovación respecto a modelos de encriptación simétrica radica en el uso de funciones matemáticas en lugar de sólo permutación y sustitución.

La encriptación asimétrica busca resolver dos grandes problemáticas en la encriptación simétrica: la distribución de llaves, la que a falta de un consenso previo sería distribuida a través de un centro distribuidor de llaves, vulnerando potencialmente el algoritmo; y la creación de firmas digitales:

el uso de criptografía para el cifrado de mensajes demandaría a sus usuarios métodos por los cuales un emisor pueda firmar digitalmente un documento encriptado y que el receptor del mensaje pueda verificar la identidad de quien recibe el mensaje.

Los algoritmos asimétricos de encriptación utilizan una *llave de encriptación* o *llave pública* para el cifrado y firma del mensaje, y una *llave de desencriptación* o *llave privada*, distinta pero relacionada con la llave de encriptación, para la decodificación. Se dice de estos algoritmos que es computacionalmente inviable determinar la llave privada conociendo sólo detalles del algoritmo y la llave de encriptación. Además, algoritmos como RSA permiten el uso de cualquiera de las dos llaves para encriptación, y luego su par sería la llave para la desencriptación.

Las partes de un sistema de encriptación asimétrica se modelan muy similarmente a su contraparte simétrica, sin embargo, existen distinciones claves en su funcionamiento. En general, el algoritmo puede describirse de la siguiente manera:

1. Cada usuario genera un par de llaves, una para la encriptación de un mensaje y su contraparte para desencriptar.
2. El usuario comparte o anuncia una de las llaves en un registro o canal público, esta llave pasa a ser conocida como la llave pública. Su contraparte debe mantenerse privada para no vulnerar la encriptación. Usuarios podrán guardar una colección de llaves públicas de otros usuarios para comunicarse con ellos por medio de encriptación.
3. Si un usuario A desea enviar un mensaje secreto al usuario B, entonces puede hacer uso de la llave pública de B (PU_b) para encriptar el mensaje.
4. Cuando el usuario B recibe el mensaje, este podrá desencriptarlo haciendo uso de su llave privada (PR_b). B será el único en poder desencriptar el mensaje, siempre y cuando haya mantenido su llave privada en secreto.

Más formalmente, sea $X = [X_1, X_2, \ldots, X_M]$ el plaintext de largo M que el usuario A desea encriptar, cuyo dominio es algún alfabeto finito. Sea B el destinatario del mensaje, que genera un par de llaves pública PU_b, y privada PR_b. B hace pública PU_b, información que A puede utilizar en el algoritmo de encriptación E para producir el ciphertext $Y = [Y_1, Y_2, \ldots, Y_N]$ de largo N:

$$\boldsymbol{Y = E(PU_b, X)}$$

Luego, el receptor del mensaje, B, podrá desencriptar el ciphertext enviado por A localmente por medio del algoritmo de desencriptación D y haciendo uso de su llave privada PR_b:

$$\boldsymbol{X = D(PR_b, Y)}$$

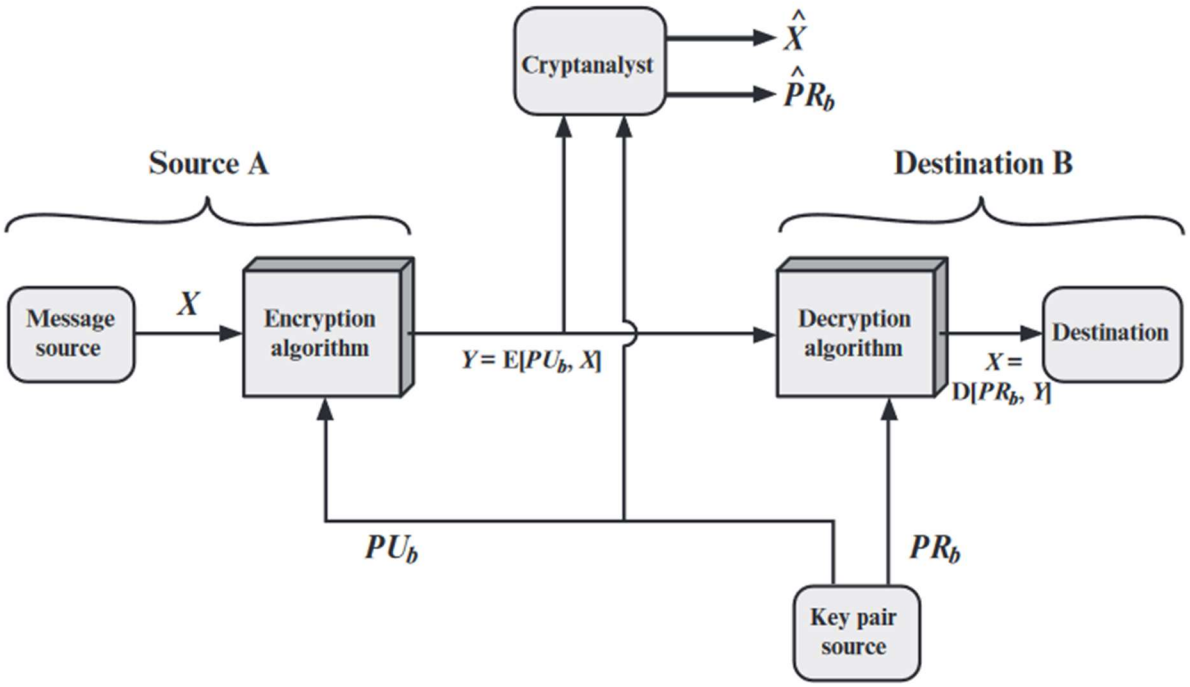


Figura 2: Modelo de encriptación asimétrica [10]

2.1.1.3 Firmas Digitales

Sin embargo, este no es el único esquema de operación para la encriptación asimétrica [10]. Como se mencionó brevemente, es posible utilizar cualquiera de las dos llaves tanto para encriptar como para desencriptar mensajes. Esto posibilita el uso de estas técnicas como una firma digital.

Formalmente, los algoritmos de encriptación y desencriptación para firmas digitales se pueden representar de la siguiente manera:

$$Y = E(PR_a, X)$$

$$X = D(PU_a, Y)$$

Donde A encripta el mensaje haciendo uso de su llave privada, y luego B sólo podrá desencriptar el mensaje haciendo uso de la llave pública de A, efectivamente operando como una *firma digital* de A en el mensaje, el que es inalterable sin conocer la llave privada utilizada para su encriptación. Es entonces un esquema que permite la *autenticación* de usuarios, y la inmutabilidad del mensaje encriptado.

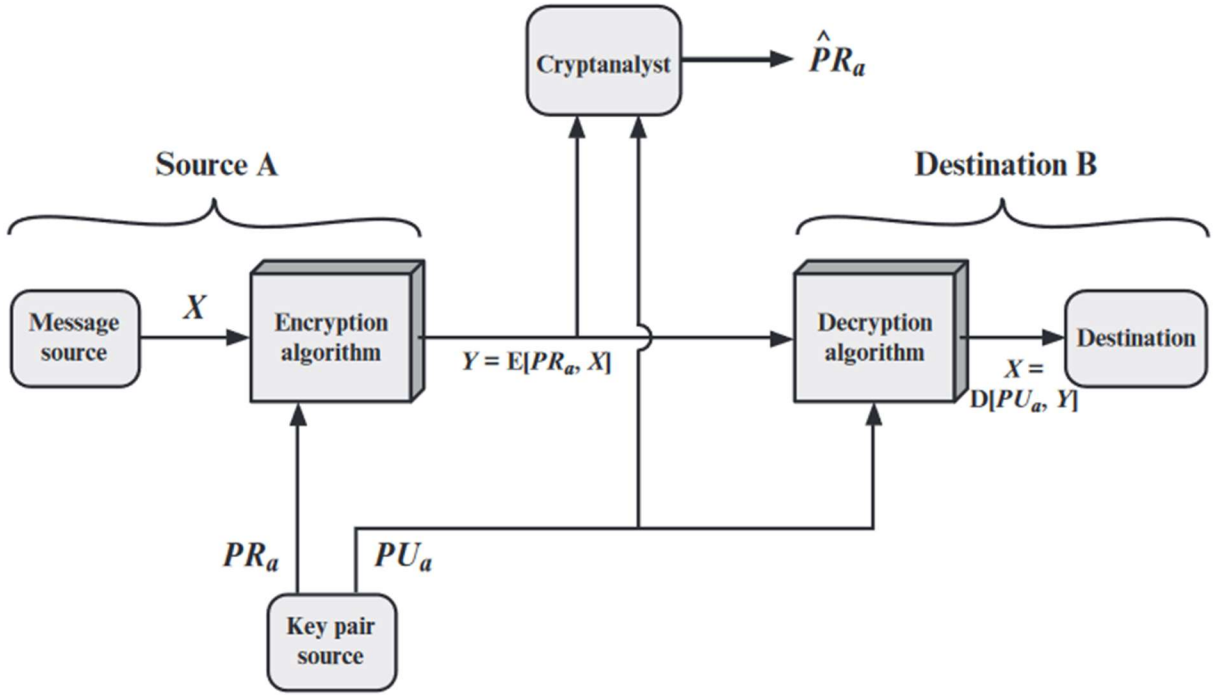


Figura 3: Modelo de autenticación utilizando encriptación asimétrica [10]

Este esquema, sin embargo, hace públicos los mensajes y se asume que cualquiera podrá desencriptarlos, dado que la llave utilizada para la desencriptación es de conocimiento público. Esto es solucionado encriptando primero el mensaje con la llave privada de A (PR_a) y el resultado de dicha encriptación puede ser encriptado nuevamente con la llave pública de B (PU_b). Formalmente, sea Z el *ciphertext* generado por la doble encriptación descrita:

$$Z = E(PU_b, Y = E(PR_a, X))$$

$$X = D(PU_a, Y = D(PR_b, Z))$$

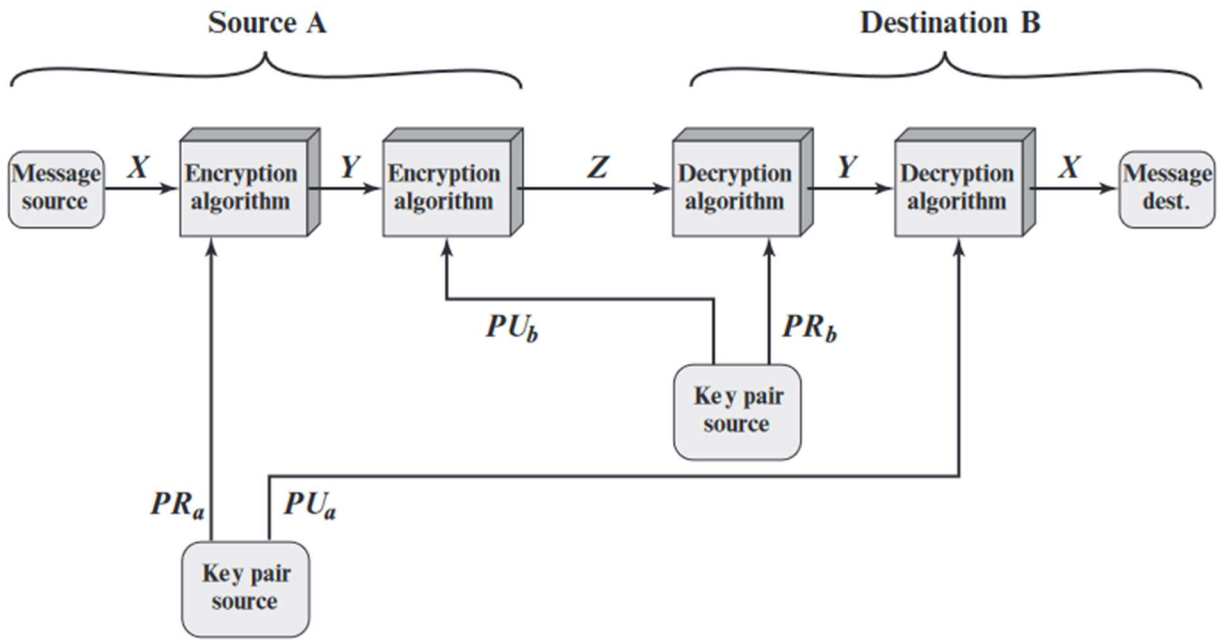


Figura 4: Modelo de autenticación y confidencialidad en un esquema de encriptación asimétrica [10]

Sin embargo, este esquema presenta la desventaja de producir un incremento sustancial en el costo computacional de la comunicación cifrada, puesto que el algoritmo de encriptación/desencriptación es ejecutado cuatro veces.

2.1.1.4 Funciones criptográficas de Hash

Una función criptográfica de hash es un algoritmo de encriptación que *mapea* (asigna) datos de largo arbitrarios a cadenas de datos (*strings*) de largo determinado. El resultado de aplicar una función de hash sobre un conjunto de datos se conoce como valor del hash, o más simplemente solo ***hash***. Dichos valores son usualmente indexados en una tabla, las que se conocen como *hash tables.*

Se dice de las funciones de hash que necesitan dos requerimientos de seguridad importantes:

- Que sea *unidireccional*, es decir, que la función de hash sea no invertible (en la práctica) y
- Que sea *resistente a colisiones*, es decir, que sea prácticamente imposible encontrar un par de datos que produzcan el mismo *hash*.

De estos requerimientos de seguridad se pueden calcular que, para garantizar la seguridad de una función de hash en al menos 80 bits de datos, se debe utilizar un resultado de largo de al menos 160 bits [11].

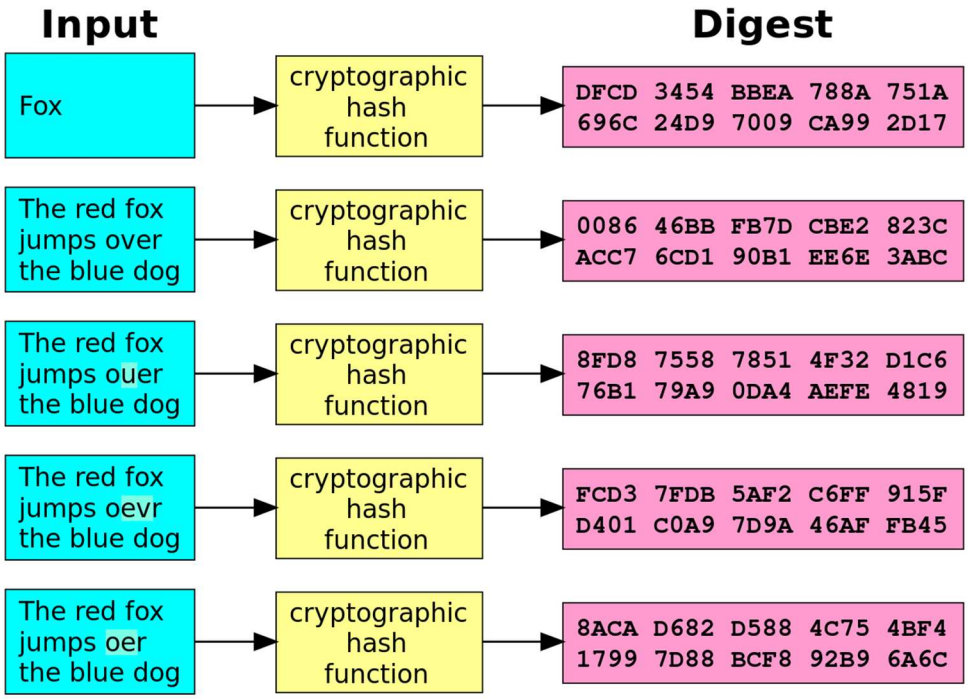


Figura 5:Diagrama simplificado de aplicación de la función de hash SHA-1. Variaciones de un sólo caracter en el input cambian totalmente el output [12].

En este documento nos referiremos en múltiples ocasiones a funciones de hash y sus resultados, más específicamente, nos estaremos refiriendo típicamente a la función SHA256, un algoritmo de *hashing* estándar, parte de la familia de *Secure Hash Algorithms* (SHA) estipulados por el Federal Information Processing Standard (FIPS) de Estados Unidos.

Las técnicas descritas anteriormente son fundamentales en la operación de sistemas distribuidos de carácter público, y en particular a lo largo de este documento nos referiremos a ellas en sus usos para la autenticación de usuarios y resolución de puzles criptográficos.

2.1.2 Distributed Ledger Technology (DLT)

Un *Sistema DLT* describe un sistema de registros electrónicos que permite a usuarios independientes mantener un consenso sobre un registro persistente, gracias a la redundancia de datos en múltiples nodos participantes de la red o sistema, de transacciones firmadas criptográficamente. Este registro goza de inmutabilidad debido a la dependencia de los registros nuevos de los antiguos: se vinculan todos los registros entre sí por medio de *hashes* [13].

Si bien las DLT no necesariamente garantizan a todos los usuarios los permisos para modificar los registros compartidos, estas presentan un nivel de transparencia y trazabilidad deseable en aplicaciones de registros que tengan como componente fundamental la verificación o consulta de estos datos públicamente.

2.2 BLOCKCHAIN

Una *Blockchain* es una DLT que incorpora operaciones criptográficas como medida de seguridad y para otorgarle un grado importante de inmutabilidad al registro compartido. Es, en esencia, un libro de contabilidad digital que requiere que se realice un cálculo complejo antes de poder registrar nuevas entradas [14].

En lo que sigue de este documento nos referiremos principalmente a los detalles de las dos *Blockchain* más populares a la fecha: *Bitcoin* y *Ethereum*.

2.2.1 Origen

En 1977 el *National Bureau of Standards* (NBS) de Estados Unidos publica su "*Data Encryption Standard*", donde se describen algoritmos de encriptación y desencriptación de datos desarrollados a comienzos de la década por ingenieros de IBM y propuestos a petición del NBS, que buscaba establecer un estándar para la protección de datos gubernamentales electrónicos con información sensible o clasificada [15].

Tres años más tarde, en una publicación relacionada, el NBS incorpora en el estándar el proceso de *Cipher Block Chaining* (CBC), que sentaría las primeras bases de la tecnología subyacente de la *Blockchain*. Este algoritmo describe la encriptación de un mensaje dividido en bloques, los cuales se vinculan entre sí en una cadena por medio de operaciones lógicas *XOR* [16].

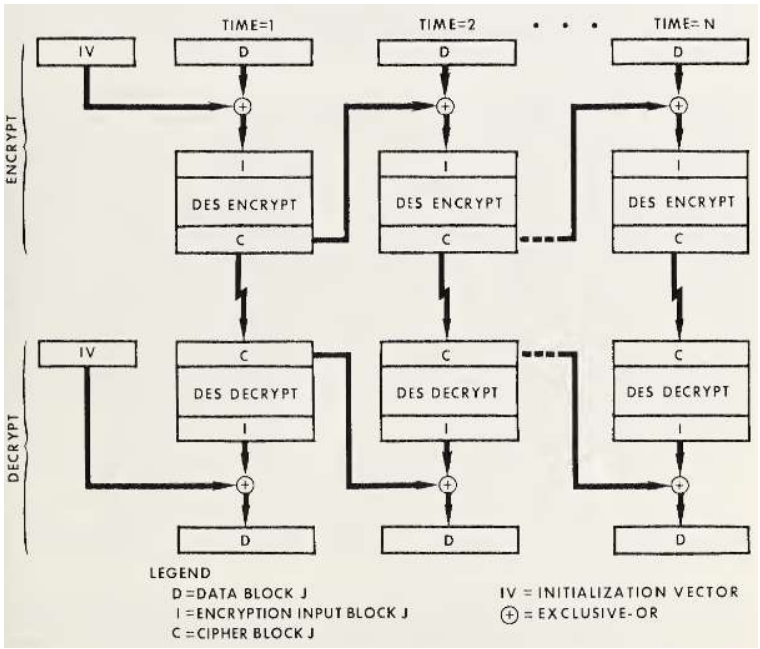


Figura 6: Diagrama de algoritmo de Cipher Block Chaining (CBC) [16].

Este algoritmo ha sido de vital importancia en el desarrollo de las tecnologías de la información, y ha sido utilizado transversalmente por décadas. No es de sorprenderse, entonces, que estos

conceptos vieran una nueva innovación aplicando este principio de vinculación de bloques a un registro inmutable.

En 2008, un individuo o grupo bajo el pseudónimo de Satoshi Nakamoto publica el paper "*Bitcoin: A Peer-to-Peer Electronic Cash System*" [17] donde se detalla un sistema descentralizado para una moneda virtual que prescinde de terceros de confianza para el procesamiento de transacciones, el que por su arquitectura se conoce como *Blockchain* (cadena de bloques).

Nakamoto propone la extensión de la idea del CBC a una moneda virtual, la que define como una cadena de firmas digitales: para transferir una moneda se firman digitalmente el hash de la transacción que le precede y la llave pública de quién la recibirá y se agregan al final de la cadena, declarando así la voluntad de transferir su propiedad [17]. Sin embargo, esto no garantiza a quien recibe la moneda que esta no ha sido gastada en otras transacciones paralelas o pasadas, un problema conocido como *double-spending* [18].

Este problema es solucionado con la introducción de una autoridad centralizada que valide las transacciones y en la que el sistema confíe para determinar el orden en que se ejecutaron las transacciones, previniendo así el double-spending. Sin embargo, esta solución depende de un alto nivel de confianza en dicha autoridad, por lo que Nakamoto propone una red donde toda transacción es anunciada públicamente, validada y propagada por los nodos de la red, y existe un mecanismo por el cual todos los participantes pueden acordar el orden en que las transacciones son recibidas por la red [17].

En esencia, lo que se requiere para la solución del problema de double-spending es un servicio de *timestamp* que aplique a cada transacción una marca temporal. Para lograr llegar a un consenso en el *timestamp* que las transacciones deban llevar en un esquema Peer-to-Peer (P2P), una arquitectura de computación distribuida, Bitcoin propone la utilización de un *protocolo de consenso* conocido como *Proof of Work* [17].

Antes de continuar con las definiciones del protocolo de consenso, veamos algunos conceptos básicos:

2.2.2 Transacción

Una *blockchain* puede ser modelada como un sistema de transición de estado distribuido en el que existe un *estado***,** y una *función de transición de estado***,** que recibe un estado y una transacción como entradas para entregar un nuevo estado en su salida [9] [19]. En otras palabras, es análoga a una máquina de estado distribuida, con las transacciones siendo solicitudes de cambio de estado.

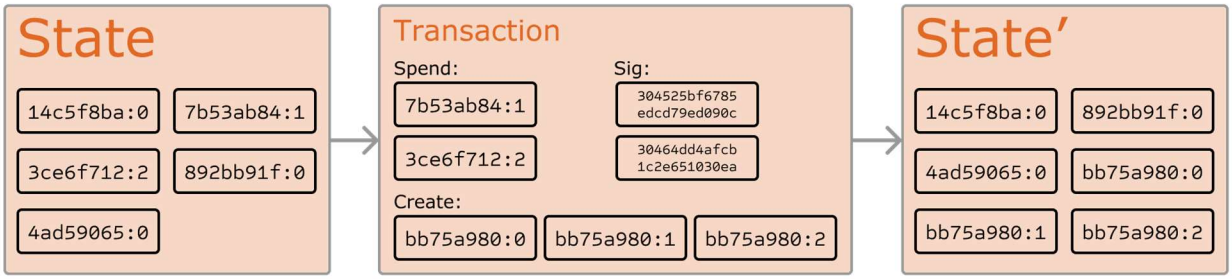


Figura 7:Transacciones como operaciones de transición de estado en la blockchain [9].

Bitcoin (y Ethereum) usan encriptación asimétrica para asignar a usuarios una identidad o *address*, la que es análoga a la llave pública, y una *llave privada* para firmar digitalmente sus transacciones. Los usuarios firman las transacciones, validando su identidad y dando a la transacción acceso a los fondos del usuario, y las anuncian a la red. Los nodos que conforman la red de la *blockchain* verifican su validez y las propagan, descartándolas en caso de que existan discrepancias en las firmas digitales, por lo que un atacante no podrá gastar los fondos de un usuario de la red sin conocer la llave privada que utiliza para su autenticación.

Estas transacciones pasan a formar parte de un conjunto de transacciones (*transaction pool, mempool* en Bitcoin) a la espera de ser confirmadas y añadidas al registro por un nodo validador. Bitcoin (y Ethereum) incorpora tarifas arbitrarias de transacción (*transaction fees*) como medida *anti-spam* para la red y como un incentivo para los validadores [20].

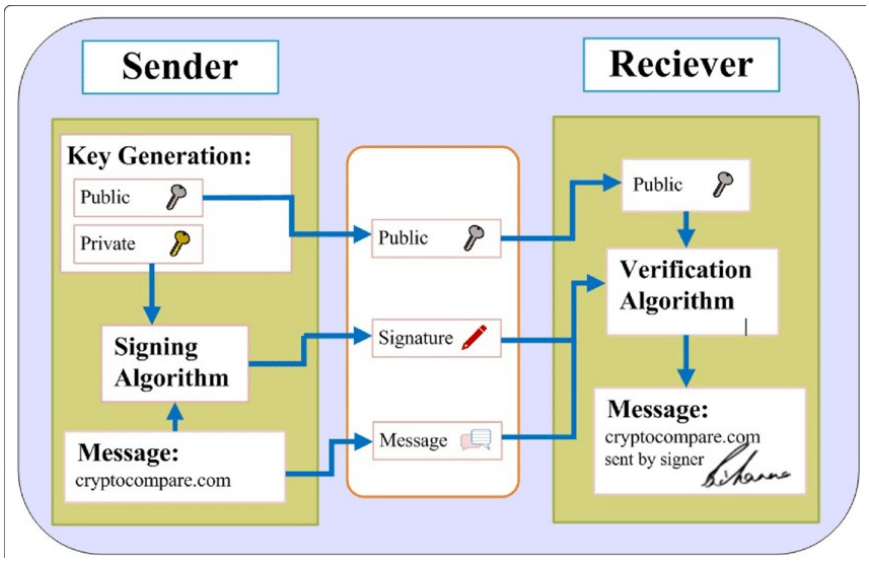


Figura 8: Esquema del proceso de firma y verificación de una transacción en Bitcoin [21].

Los usuarios pueden escoger cuánto pagar por la transacción, pero depende de los validadores aceptar esa tarifa, e incluso es posible que incluyan transacciones sin tarifa alguna. En la generalidad los validadores aceptarán las transacciones con las tarifas más altas, por lo que en la

práctica se utiliza la tarifa como una forma de pedir prioridad a la red para la inclusión de la transacción en el registro [20].

2.2.3 Bloque

Un bloque en una *blockchain* es un conjunto de transacciones. Son creados por nodos especializados en la red, coloquialmente conocidos como "mineros", para agregarlos a la cola de la cadena de bloques.

La estructura de un bloque varía de acuerdo a la implementación, pero en general cumplen con la condición de vincular cada nuevo bloque de la cadena con el que le precede a través de funciones de hash: cada nuevo bloque contiene dentro el hash del bloque (o la cabecera del bloque) anterior, por lo que mientras más crece la cadena más se refuerza la confiabilidad de los bloques más antiguos debido a que modificarlos involucraría la modificación de todos los bloques que le siguen para que el hash sea correcto, aumentando exponencialmente el costo computacional de modificarlos [17].

Bitcoin hace uso de Árboles de Merkle, una estructura de datos que utiliza hashes para disminuir el tamaño de la estructura, y facilitar su verificación y descarte de entradas. Esta estructura se construye calculando el hash de todas las transacciones que se quieran agrupar independientemente. Luego, se concatenan los hashes en grupos y se calculan los hashes de dichas concatenaciones. El proceso se repite hasta llegar a un solo hash, el que referencia la totalidad de las transacciones involucradas. Cuando el tamaño de los grupos de hash concatenados es dos, hablamos de un Árbol Binario de Merkle.

A continuación, se presenta una figura ilustrando la estructura simplificada de un bloque de acuerdo a las especificaciones de Nakamoto, y de cómo facilita el proceso de poda (*pruning*) de transacciones que ya han sido gastadas:

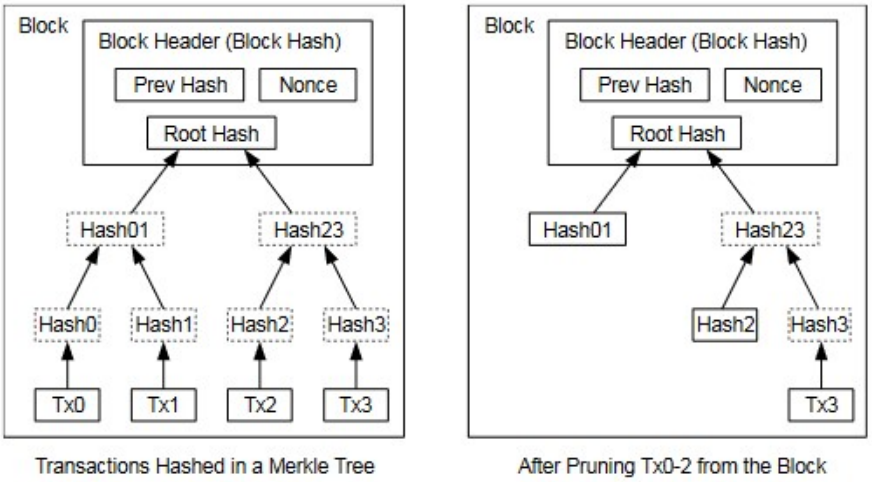


Figura 9:Estructura simplificada de bloques en Bitcoin [17].

Además, la construcción de un bloque busca satisfacer las condiciones establecidas por el protocolo de consenso específico de la implementación.

2.2.4 Protocolos de Consenso

Parecerá lógico que, dado el planteamiento de una red descentralizada que mantenga un registro compartido, se deba establecer un mecanismo que permita a los usuarios de la red llegar a un consenso sobre el estado actual de dicho registro; esto es lo que se conoce como un protocolo o algoritmo de consenso.

Estos protocolos son las reglas bajo las cuales los nodos deben operar en la red, y establece las condiciones que un minero debe cumplir para proponer un nuevo bloque a la red.

2.2.4.1 Proof of Work

En Bitcoin y *blockchain*s derivadas de dicho proyecto el protocolo de consenso que mantiene la integridad de la red es conocido como *Proof of Work*. Este protocolo deriva su nombre de su principal característica: para proponer un nuevo bloque a la red este debe cumplir con ciertas condiciones que demandan trabajo computacional intenso.

En particular, la propuesta de Bitcoin es exigir a los mineros que resuelvan un puzzle criptográfico: la construcción del bloque debe ser tal que el hash del bloque contenga un número mínimo de bits en cero al comienzo del resultado, según la *dificultad* de la red que es determinada por el protocolo para mantener un número determinado de bloques por hora. Bitcoin reconfigura la dificultad cada 2016 bloques, o aproximadamente 2 semanas, para mantener en promedio 6 nuevos bloques por hora. Mientras mayor sea la dificultad, más bits en cero debe tener el hash o, en otras palabras, menor debe ser el resultado [20].

La manera en que los mineros pueden variar el resultado del hash del bloque propuesto es a través de un número que se incluye en el bloque, el que puede modificarse arbitrariamente, conocido como *nonce* [22]. Debido a la aleatoriedad del resultado de las funciones de hash (en la práctica) este proceso sólo puede ser resuelto con fuerza bruta y por ende un mayor poder computacional, que permite calcular en menos tiempo el hash, tiene el efecto de incrementar linealmente las probabilidades de un validador de encontrar un nuevo bloque válido [23].

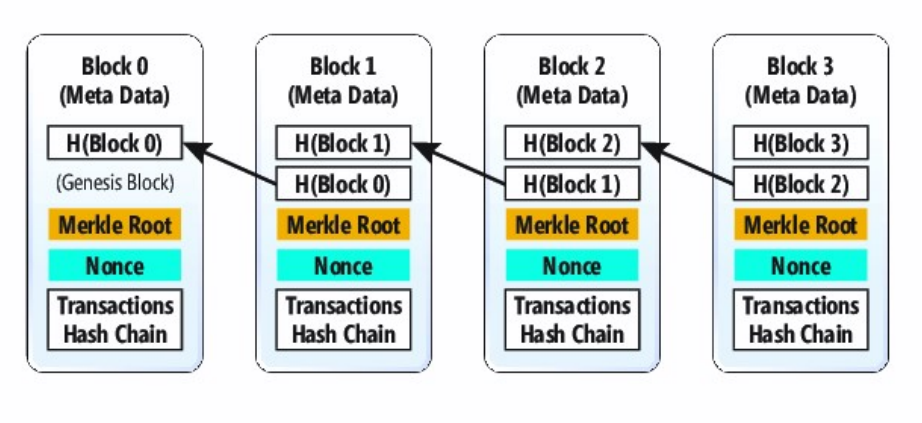


Figura 10: Diagrama simplificado de una cadena de bloques [24].

La construcción y validación de bloques es incentivada con la creación de nuevos criptoactivos; quien sea el primero en proponer un bloque válido recibirá los criptoactivos recién acuñados, además de todas las tarifas de las transacciones del bloque. Este incentivo además juega el rol de mantener a los nodos validadores honestos, puesto que un nodo malicioso que cuenta con suficiente capacidad computacional para vulnerar la red revirtiendo sus transacciones estaría devaluando el criptoactivo con su ataque, asumiendo la pérdida de confianza del público en el sistema; mientras que manteniéndose honesto puede seguir generando nuevas criptomonedas y cobrando las tarifas de transacción [17].

Los nodos de la red considerarán la cadena más larga anunciada como el estado actual del registro compartido, y a los bloques que ya son parte de dicha cadena como *confirmados* [22]. Se dice del último bloque en ser incorporado a la cadena que tiene una confirmación, del que le precede que tiene dos confirmaciones, del bloque siguiente tres confirmaciones, y así sucesivamente. El número de confirmaciones es entonces de vital importancia a la hora de considerar la permanencia de las transacciones, considerándose 6 confirmaciones un número prudente, de acuerdo a los cálculos de Nakamoto [17].

2.2.5 Ethereum

A lo largo de este documento se han descrito las especificaciones de una *blockchain* principalmente enfocándose en su primer y más popular iteración, Bitcoin. Sin embargo, de la radical innovación introducida en el trabajo de Nakamoto derivan un centenar de *blockchain*s que utilizan el protocolo Proof-of-Work. Entre ellas destaca Ethereum [9], desarrollada en 2013 por Vitalik Buterin, es la *blockchain* open-source que procesa la mayor cantidad de transacciones diarias [25].

2.2.6 Ether

De manera similar a Bitcoin, Ethereum habilita un sistema descentralizado para el acuñamiento y distribución de una moneda virtual, el *Ether* (ETH). Utiliza un protocolo Proof-of-Work y una arquitectura similar a la de Bitcoin en la construcción de sus bloques, sin embargo, *su diseño está*

orientado a la ejecución de código computacional de manera distribuida. El Ether es usado como moneda de incentivo para que los validadores mantengan la cadena y ejecuten el código computacional alojado en todos los nodos de la red, los que se conocen como *Smart Contracts*.

2.2.7 Smart Contracts

El término *Smart Contract* es acuñado por el criptógrafo Nick Szabo en 1990 y posteriormente formalizado en 1997 [26]. Originalmente el término describe técnicas para embeber cláusulas contractuales en hardware y software, permitiendo así una definición más concreta de los términos que rigen la propiedad de sistemas y relaciones humanas con *Smart Contracts* embebidos. Szabo afirma que la visión bajo la que define los *Smart Contracts* busca la definición explícita de, por ejemplo, la relación del dueño o director y sus empleados en un contexto profesional, reduciendo costos mentales y computacionales de formalizar y asegurar dichas relaciones digitalmente.

Andreas Antonopoulos, autor de "*Mastering Bitcoin"* y "*Mastering Ethereum*", describe en este último una concepción post-Bitcoin de *Smart Contracts* en el contexto de Ethereum [27]:

> *"...we use the term "smart contracts" to refer to immutable computer programs that run deterministically in the context of an Ethereum Virtual Machine as part of the Ethereum network protocol—i.e., on the decentralized Ethereum world computer."* ***– A. Antonopoulos.***

> *"...usamos el término "smart contracts" para referirnos a programas computacionales inmutables que son ejecutados deterministamente en el contexto de una Máquina Virtual de Ethereum como parte del protocolo de red de Ethereum, es decir, en el computador global descentralizado de Ethereum."*

Antonopoulos continúa explicando los puntos clave de esta definición:

1. Un *Smart Contract* en Ethereum es simplemente un *programa computacional* sin un significado legal intrínseco, a pesar de su nombre.
2. Es *inmutable*, el código que rige el comportamiento del *Smart Contract* no puede ser modificado una vez este es desplegado en la red, por lo que la única manera de alterar el comportamiento de un *Smart Contract* es por medio del despliegue de una nueva instancia.
3. Es *determinista*, significando que el resultado de la ejecución de un *Smart Contract* es la misma para cualquiera que lo corra, dado el mismo contexto que inicia su ejecución y el estado de la *blockchain* en dicho momento.
4. Son ejecutados en un *contexto limitado* dentro de la *Ethereum Virtual Machine (EVM)*, la máquina virtual que ejecutan todos los nodos de la red de Ethereum para el despliegue y ejecución del código de los *Smart Contracts*. Tienen acceso a su propio estado, al contexto de las transacciones que realizan llamados al *Smart Contract*, e información limitada de los bloques más recientes.

5. Debido a estos últimos dos puntos podemos decir que, dado a que la EVM es ejecutada como una instancia local en todos los nodos de Ethereum y que todas las instancias de la EVM operan con los mismos estados iniciales y producen el mismo estado final, el sistema es en la práctica un computador global, o *computador distribuido*.

Los detalles de desarrollo e implementación de *Smart Contracts* en Ethereum más relevantes a este proyecto serán ahondados en las definiciones y explicación de la composición de Aplicaciones Descentralizadas en las secciones que siguen.

2.3 APLICACIONES DESCENTRALIZADAS: ĐAPPS

Una aplicación descentralizada (*Decentralized Application, Đapp* o *dApp)* es un sistema computacional que incorpora la lógica interna de una aplicación tradicional en un *backend* alojado y operado en una red descentralizada, típicamente haciendo uso de *Smart Contracts* en una *Blockchain*, en conjunto con componentes de la arquitectura de la aplicación como el *frontend* e incluso partes del *backend* que pueden ser centralizadas [27].

Las principales ventajas que una aplicación descentralizada tiene sobre una aplicación tradicional son [27]:

- **Resiliencia:** la implementación de la lógica interna de la aplicación en *Smart Contracts* alojados en una red descentralizada le permite mantenerse operacional siempre que la red se mantenga en línea.
- **Transparencia:** toda interacción con la Đapp que se realiza *on-chain*, es decir, haciendo uso de los *Smart Contracts* en la *blockchain*, es pública. Se registran estas interacciones inmutablemente, otorgando un alto grado de trazabilidad a la aplicación.
- **Resistencia a Censura:** un usuario solo requiere de acceso a un nodo de la *blockchain* en la que los *Smart Contracts* se han desplegado para interactuar con la Đapp. En caso de requerirlo puede ejecutar su propio nodo con hardware de bajo costo. Además, el código de un *Smart Contract* no puede ser modificado una vez desplegado en la red.

2.3.1 Smart Contracts en Ethereum

Ethereum es, a la fecha, la *blockchain* con la mayor cantidad de Đapps desplegadas en su red, con cerca de 2900 listadas y 1930 activas en el sitio web *State of the Dapps*, que mantiene estadísticas de aplicaciones descentralizadas en múltiples plataformas [28]. El sitio registra en Ethereum sobre 90 mil usuarios de Đapps diarios y 180 mil transacciones durante las últimas 24 horas, con un volumen de cerca de 53 mil ETH siendo transferidos a *Smart Contracts* de alguna Đapp registrada en el sitio, los que representan un valor de casi US$215 millones diarios, al 11 de diciembre de 2021.

Debido al tamaño de su red, su popularidad y capitalización de mercado (volumen en circulación * precio promedio) del ETH, la comunidad de desarrollo de Ethereum ha crecido aceleradamente,

haciendo disponible una gran cantidad de recursos, documentación y herramientas desarrolladas en torno y dentro de la *blockchain*. Además, es posible monetizar Đapps de manera sencilla y confiable en comparación con otras *blockchain*, si bien volátil en comparación con alternativas centralizadas existen mecanismos para disminuir estos riesgos lo que hace la hace una plataforma atractiva.

2.3.2 Estructura

Por conveniencia y como simplificación conceptual podemos modelar una Đapp en cuatro componentes esenciales [27]:

2.3.2.1 Backend

Uno de los objetivos en el diseño de una aplicación descentralizada en una *blockchain* como la de Ethereum es alocar la lógica de negocios de la aplicación en *Smart Contracts*. Esta lógica interna podemos considerarla análoga a las componentes alojadas en servidores en una aplicación tradicional, concepto conocido como *backend*.

El diseño del *backend* en una Đapp presenta un desafío dado lo costosas que son las operaciones ejecutadas en la red, demandando poner especial atención a la minimización de redundancias y a la correcta identificación de los sistemas que ameritan ser públicos y descentralizados. Más aún, procesos críticos y de resolución rápida del sistema diseñado pueden no ser aptos para su ejecución descentralizada en *Smart Contracts* puesto que una Đapp segura debe esperar a un cierto número mínimo de confirmaciones antes de considerar una transacción como permanente, proceso que no puede ser acelerado.

Ethereum permite a los *Smart Contracts* interactuar entre sí, permitiendo a los desarrolladores atomizar la arquitectura del *backend*, diseñando *Smart Contracts* para realizar funciones especializadas y disminuyendo los costos de gas (costos de transacción) del sistema. Sin embargo, debido a que en Ethereum el código de un *Smart Contract* no puede ser modificado una vez ha sido desplegado, se debe tener especial cuidado cuando se diseñan sistemas muy fragmentados.

Un *Smart Contract* desplegado en la *blockchain* tendrá un espacio de almacenamiento (*storage*) y un espacio de memoria asignados, ambos dinámicos y divididos en secciones (*slots*) de 32 Bytes. Ejecutar operaciones de escritura en el *storage* del contrato o en memoria tiene un costo asociado al número de slots que se necesita modificar, y por ende la optimización del uso de estos recursos es imperante para el diseño de una aplicación descentralizada.

2.3.2.2 Frontend

De la misma manera que una aplicación tradicional, una Đapp requiere de una interfaz de usuario, la componente *client-sided* de la aplicación que se conoce como el *frontend*. Estas interfaces son diseñadas con tecnologías web estándar como HTML, CSS y JavaScript, en conjunto con el uso de librerías que las vinculan con la *blockchain* de Ethereum. La librería de JavaScript web3.js es frecuentemente utilizada en la integración de los servicios *on-chain* en el *frontend* [29].

Una aplicación flexible incorporará gran parte de sus interfaces de usuario en un sitio web, el que frecuentemente es alojado en un servidor tradicional pero no necesariamente, donde los usuarios podrán interactuar con la *blockchain* con extensiones en sus navegadores que facilitan esta comunicación, como *MetaMask* [30].

2.3.2.3 Almacenamiento de Datos

Por diseño, la *blockchain* de Ethereum tiene altos costos sobre la administración, procesamiento y almacenamiento de grandes volúmenes de datos. Por ello, un gran número de Đapps optan por el uso de servicios tradicionales de almacenamiento de datos como la nube (*cloud storage*) para datos pesados. Sin embargo, existen alternativas P2P descentralizadas para distribuir los recursos de la interfaz web y recursos estáticos como imágenes y vídeos.

Una de estas alternativas es *Inter-Planetary File System* [31], un protocolo que distribuye archivos en una red P2P para su futuro acceso a través de una referencia a su hash. Podría considerarse análogo a un servicio de nube distribuido en múltiples servidores a la vez, los nodos de IPFS, permitiendo a los usuarios el acceso a los archivos a través de cualquier nodo de IPFS y ofreciendo una potencial disminución en el ancho de banda requerido para la operación de aplicaciones.

Otra alternativa es *Swarm* [32], desarrollado por la *Ethereum Foundation*. Al igual que IPFS es una red P2P descentralizada diseñada explícitamente para su uso en conjunto con aplicaciones on-chain en Ethereum. Cuenta con un sistema de incentivos que utiliza la *blockchain* de Ethereum para promover la distribución, replicación y disponibilidad de los archivos en la red.

2.3.2.4 Protocolo Descentralizado de Comunicación por Mensajes

Por último, una aplicación descentralizada requerirá, frecuentemente en su diseño, métodos para intercambiar mensajes entre aplicaciones, entre distintas instancias de una misma aplicación o entre usuarios de la aplicación. Esta necesidad es usualmente resuelta centralizadamente, pero es posible hacer uso de sistemas distribuidos para incorporar un protocolo de mensajería.

Whisper [33] es una de las alternativas disponibles: es un sistema de mensajes basados en identidad (*identity-based messaging system*) que provee una API de bajo nivel para comunicación P2P asíncrona entre y para aplicaciones descentralizadas, que evite las limitaciones de bajo nivel del hardware, particularmente el que sus puertos de comunicaciones sean singulares. Está específicamente diseñado para el *multi-casting* y *broadcasting* de mensajes enfocado en la protección de la privacidad de sus usuarios.

2.3.3 Solidity y Despliegue en la Red

El diseño de *Smart Contracts* típicamente ocurre con lenguajes de programación de alto nivel como *Solidity* [34], un lenguaje imperativo, orientado a objetos, de alto nivel y open-source para la implementación de *Smart Contracts* creado por Dr. Gavin Wood. Su sintaxis esta influenciada por lenguajes de programación como C++, Python y JavaScript, y está diseñado específicamente para interacciones con la EVM [35].

Un *Smart Contract* en Solidity es luego compilado a *bytecode* de bajo nivel ejecutable en la EVM y desplegado a través de una transacción de creación de contrato que le otorgará una dirección (*address*) en la red. Usando la dirección del contrato es posible enviar transacciones para depositar fondos en ether (ETH), e incluir un *payload* con instrucciones para la ejecución de algún método programado. El creador del contrato no recibe ningún privilegio especial a nivel de protocolo de red, el *Smart Contract* opera autónomamente según el código compilado donde, sin embargo, se podrían declarar privilegios tanto para el creador del contrato como para otros usuarios de manera arbitraria.

Solidity es el lenguaje de alto nivel más popular para programar *Smart Contracts*, sin embargo, existen múltiples alternativas que presentan distintos paradigmas de programación y sintaxis, algunas de ellas son:

- *LLL*: lenguaje funcional con sintaxis similar a Lisp.
- *Serpent:* lenguaje procedimental con sintaxis similar a Python. Permite además programación funcional.
- *Vyper:* alternativa a Serpent con sintaxis parecida a Python.

2.3.4 Costos de Despliegue y Operacionales

En Ethereum toda transacción que resulta en un cambio de estado de la red tiene un costo asociado en forma de una tarifa que se paga a los nodos validadores. Su principal finalidad es evitar abusos de la red como ataques DDoS (Distributed Denial-of-Service) mediante un desincentivo económico para el *spam* de transacciones [36].

Los pagos de las tarifas de transacción se hacen con unidades de *gas.* Se calcula cuanto gas se necesita para la ejecución de la transacción en base a las operaciones de escritura que se ejecutarán. Cada tipo de computación que se realiza en la *blockchain* tiene un costo en gas definido por el protocolo [36].

En una transacción el gas se paga con Ether, y el precio por unidad de gas es determinado por el usuario que anuncia la transacción a la red. Además, toda transacción tiene un parámetro de límite de gas, que determina la cantidad máxima de gas que puede consumir la transacción. Si la ejecución de la transacción supera el límite, esta es revertida [36].

Los mineros pueden decidir arbitrariamente qué transacciones se incorporan en el nuevo bloque que construyen, por lo que un usuario interesado en que su transacción sea procesada rápidamente deberá ofrecer un mayor precio a pagar por unidad de gas en su transacción para hacerla más atractiva. En resumidas cuentas, el precio del gas y, en consecuencia, el costo de la transacción estará dado por la oferta y demanda de gas en la red [36].

Como se menciona brevemente en la sección anterior, el despliegue de *Smart Contracts* compilados es ejecutado mediante una transacción en la red. El costo de dicha transacción en gas

está compuesto por un costo fijo de la transacción de despliegue, correspondiente a 53000 gas, y por un costo variable que depende de las operaciones de escritura que la transacción realiza en *storage* y *memory*.

2.3.5 Entorno de Desarrollo

Para el desarrollo de aplicaciones descentralizadas y, en particular, para el diseño y despliegue de *Smart Contracts* existen múltiples alternativas de entornos de desarrollo integrados (*IDE*), para los distintos lenguajes de programación de Ethereum. En este trabajo nos interesa principalmente el IDE de Ethereum *Remix IDE* [37], proyecto dirigido por la Ethereum Foundaiton, el cual puede encontrarse en su página web https://remix.ethereum.org, y opcionalmente como una aplicación de escritorio o extensión para el IDE VSCode.

Remix incorpora en su IDE un compilador de Solidity y distintas alternativas para el entorno (*environment*) de despliegue del contrato: puede desplegarse el contrato en una simulación de la *blockchain* local en el navegador (*Javascript VM*), a través de un nodo externo ejecutado localmente, o a través de un nodo externo siendo ejecutado remotamente. Estos dos últimos pueden estar conectados a una red de prueba local, a la red de prueba pública de Ethereum, o a la *blockchain* misma, la *mainnet*.

Este entorno de desarrollo es de vital importancia para el desarrollo de aplicaciones descentralizadas, ofreciendo la posibilidad de desplegar iteraciones del contrato para la verificación de su funcionamiento y la estimación de costos. *Remix* indica el costo en gas de la ejecución de transacciones, e incluso provee una interfaz gráfica simple para la interacción con los contratos desplegados.

En el Anexo A de este informe se incluye una descripción gráfica superficial del proceso de despliegue utilizando *Remix.*

Capítulo 3. DISEÑO PROPUESTO

En la siguiente sección se describe el diseño final propuesto y se justifican sus características esenciales.

3.1 OPEN SOURCE

El diseño propuesto a continuación es de carácter Open Source, siendo todo el código sujeto a la licencia MIT.

Esta característica es clave puesto que la solución pretende establecer un estándar para la estructuración de los datos de rutas de escaldada y su despliegue en la *blockchain* de Ethereum.

Los *Smart Contracts* que conforman la plataforma están diseñados con múltiples instancias de estos coexistiendo en Ethereum en mente, y más aún, incluso instancias con modificaciones al código fuente de los *Smart Contracts* que gestionan los datos.

Luego, la licencia MIT es adecuada para permitir que existan innovaciones futuras sobre el trabajo propuesto e instancias del sistema que se acomoden específicamente a las necesidades de las comunidades locales de escalada.

3.2 ESTÁNDAR DE INFORMACIÓN DE RUTAS DE ESCALADA EN ETHEREUM

La base de este proyecto es el *Smart Contract* que define una ruta de escalada, al que llamaremos *RouteData*. Este *Smart Contract* es presentado como un nuevo estándar para almacenar los datos de una ruta de escalada de manera inmutable en la *blockchain* de Ethereum.

Cada instancia de dicho *Smart Contract* desplegada en la red de Ethereum representará una ruta de escalada, identificada a través de un nombre y su ubicación mediante coordenadas geográficas expresadas decimalmente con 5 dígitos decimales para un nivel de precisión de ±11cm.

Además, el *Smart Contract* contiene datos de la disciplina a la que corresponde la ruta, distinguiendo entre *free climbing* y *bouldering*, el grado de dificultad propuesto por su creador y la escala utilizada para dicha graduación. Este último set de datos (disciplina – escala de graduación – grado de dificultad) es codificado en un solo byte de información para la minimización de costos del sistema.

La siguiente figura ilustra los datos contenidos en un *Smart Contract* de tipo RouteData:

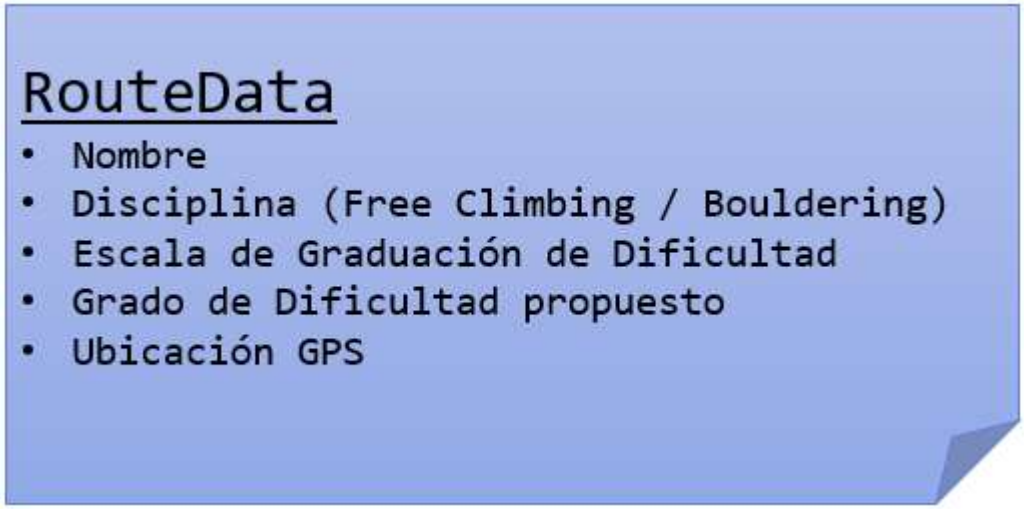


Figura 11: Simplificación de Smart Contract RouteData, usados como representación de rutas de escalada en la blockchain.

Los detalles de la codificación de dichos parámetros son descritos en el anexo del presente informe y desplegados en la *blockchain* de Ethereum en forma de un *Smart Contract* interfaz *ScalesData* que permite a los usuarios decodificar la información registrada en la *blockchain*.

3.3 Sistema de Creación y Registro de Rutas de Escalada en Ethereum

La solución que propone el presente trabajo se concentra en proponer una infraestructura para la catalogación de rutas de escalada en una *blockchain* descentralizada, específicamente, en Ethereum.

Si bien la información de las rutas debe poder registrarse en la *Blockchain* sin la aprobación de una entidad central, es conveniente que los *Smart Contract* de catalogación de rutas cuenten con controles de acceso de escritura como una opción de operación. De esta manera se pueden implementar infraestructuras similares a las plataformas de registro digital de rutas de escalada ya existentes, pero que incorporen datos desplegados de manera descentralizada y cuyo almacenamiento es distribuido en la *blockchain*, manteniendo el registro incluso si dichas plataformas cesan sus operaciones.

La incorporación de una opción de control de acceso de escritura es además conveniente para efectos de la organización de comunidades locales de escaladores, en particular en situaciones donde grupos organizados hagan mantención y desarrollo de rutas en una región determinada.

Estas funcionalidades se implementan en *Smart Contracts* de gestión e indexación de datos en los que se pueden registrar las *addresses* de múltiples *RouteData* para su catalogación. Se utilizan dos tipos de *Smart Contracts* para la gestión de datos: *RouteCatalog* y *SectorData.*

RouteCatalog es un *Smart Contract* donde se podrán registrar múltiples *RouteData* e indexarlos por nombre para facilitar su búsqueda y, además, de manera opcional, restringir la creación e indexación de *RouteData* en este contrato a un *Smart Contract* fábrica *RouteFactory,* es decir, un *Smart Contract* diseñado para la creación correcta de *Smart Contracts* del tipo *RouteData.*

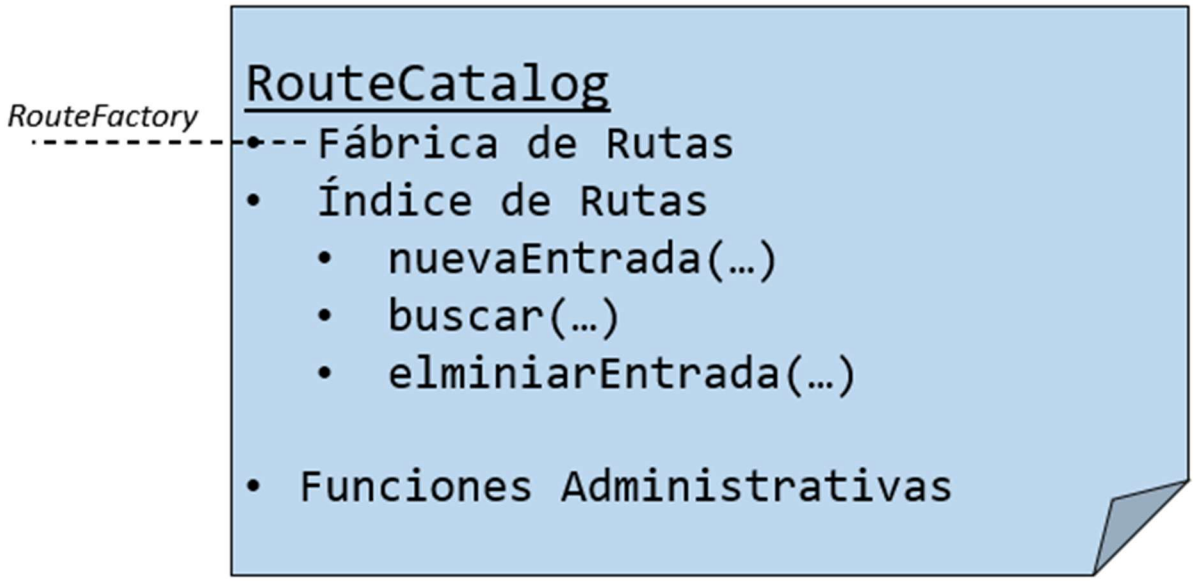


Figura 12:Simplificación de Smart Contract RouteCatalog, usados para la indexación de rutas de escalada desplegadas en la blockchain

Por otro lado, *SectorData* es un *Smart Contract* similar a *RouteCatalog*, pero sin capacidad de creación de *RouteData*. En su lugar, se pueden registrar *RouteData* que ya se encuentrn desplegados en la *blockchain* siempre que estos se encuentren en un *RouteCatalog* asociado, el cual es declarado en la instanciación del *Smart Contract*. En otras palabras, un *SectorData* contendrá un subconjunto del conjunto de *RouteData* que se encuentran en un *RouteCatalog* determinado, y se utilizará para representar un sector geográfico donde se concentran rutas de escalada.

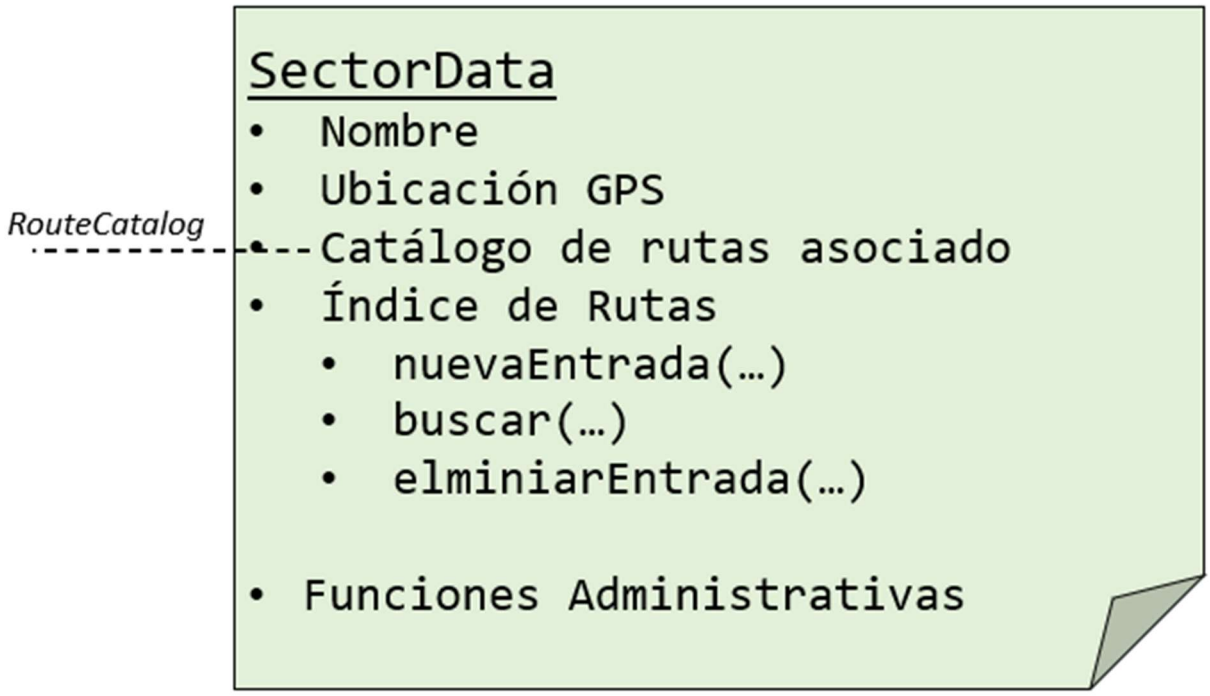


Figura 13:Simplificación de Smart Contract SectorData, usados para la indexación de rutas de escalada desplegadas en la blockchain asociadas a un sector geográfico determinado

La finalidad de este esquema es que los usuarios que deseen desplegar una plataforma de indexación de rutas lo puedan hacer utilizando todos estos componentes: restringiendo la creación y registro de *RouteData* en su *RouteCatalog* a un *RouteFactory* para garantizar la integridad de los datos y la utilización de una codificación estándar detallada en un *ScalesData* ya sea desplegado por el mismo usuario, o un tercero. Luego se pueden desplegar cuantos *SectorData* se desee asociados a un *RouteCatalog* ya desplegado, limitando así las rutas que pueden ser catalogadas a aquellas provenientes de un *RouteCatalog* de confianza.

El despliegue de *SectorData* asociados a un *RouteCatalog* no tiene requisitos de permisos especiales, por lo que múltiples individuos y comunidades pueden desplegar sus propias instancias de *SectorData* vinculadas a un mismo *RouteCatalog* administrado por un tercero. Dicho de otra manera, los usuarios pueden usar cualquier *RouteCatalog* que se encuentre desplegado en la red de Ethereum para validar las rutas registradas al sector geográfico que administran por medio de *SectorData*.

Este modo de operación permitiría la creación de un gran catálogo público (en conjunto con una fábrica asociada) donde cualquiera pueda registrar rutas para su indexación. Mientras que el despliegue y administración de un *SectorData* que represente un sector geográfico puede ser gestionado por el usuario que desee hacerlo, independiente de los permisos de escritura que tenga en el *RouteCatalog* asignado al sector.

Esta forma de organización resulta beneficiosa para escenarios donde, por ejemplo, terceros pueden ofrecer el despliegue íntegro y correcto del sistema y la implementación de un *frontend* que haga uso de estas componentes como un servicio comercial sin mermar la alternativa que un usuario con el conocimiento técnico de la *blockchain* podría tomar para el despliegue de su propio sistema, o para el ingreso de nuevas entradas a sistemas ya desplegados.

3.4 INDEXACIÓN DE SECTORES Y FRONTEND

Finalmente, para la practicidad de esta plataforma se debe contar con un *frontend* que haga de interfaz entre el usuario y la aplicación en la *blockchain*. El sistema no ha sido diseñado con una solución *frontend* determinada, más bien detalla el proceso mediante el cual los usuarios pueden diseñar e implementar sus propias *frontend* que satisfagan las necesidades de su comunidad, así como minimizar los costos del sistema.

Particularmente se hace énfasis en la catalogación de sectores: esta funcionalidad, a diferencia de lo que hace *RouteCatalog* análogamente con *RouteData*, es relegada al *frontend* de la aplicación. Las motivaciones para esta decisión son el costo de dicha implementación y la redundancia innecesaria que un catálogo de sectores implica.

El uso de *RouteCatalog* está motivado por la necesidad de contar con datos correctamente formateados y una minimización de datos espurios en un catálogo indexado; debido a que los *SectorData* ya se vinculan con *RouteCatalog* entonces la finalidad de un posible *SectorCatalog* sería únicamente la catalogación de sectores. Se estima que esto no justifica el costo de despliegue y operación de un *Smart Contract* catálogo de sectores, sin embargo, esta alternativa siempre es posible dada la modularidad del proyecto y la naturaleza de la *blockchain*.

La sugerencia de diseño de este proyecto es la indexación *off-chain* de sectores en el servidor donde se desee desplegar una instancia del sistema. Es decir, utilizando un nodo de Ethereum se pueden consultar los datos de uno o más *SectorData* conociendo sus *address*, estos datos luego son registrados en una base de datos off-chain que incluya su ubicación para poblar un mapa de sectores, como por ejemplo en un servidor proveído por *Amazon Web Services* (AWS).

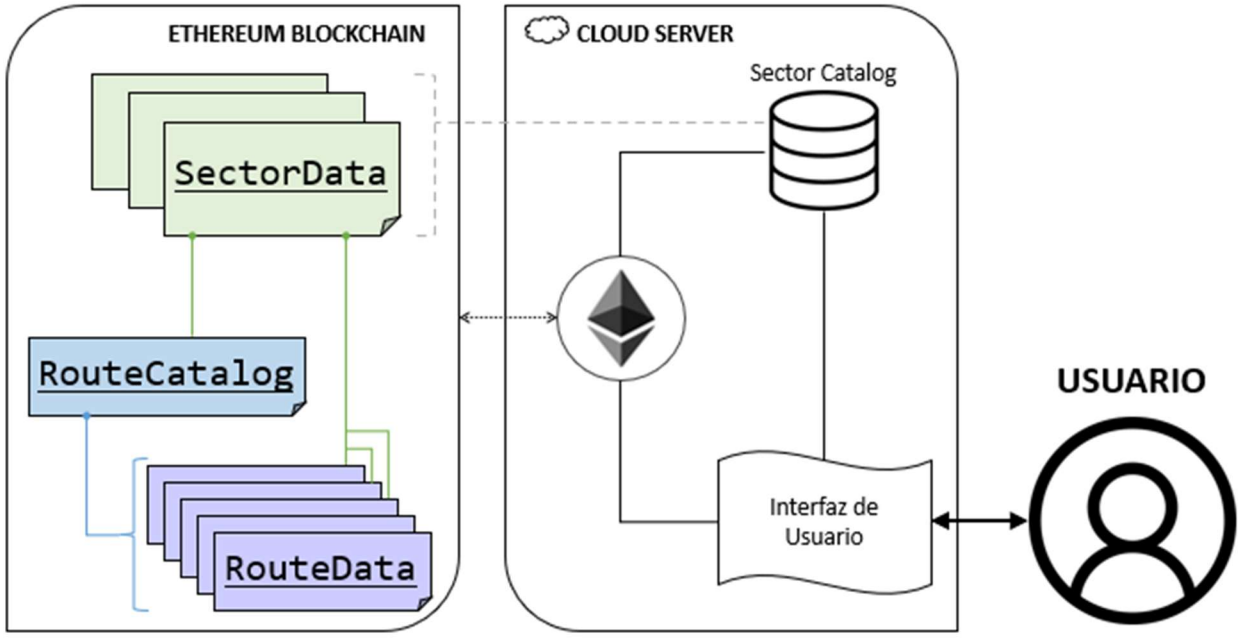


Figura 14:Descripción gráfica de la implementación de una interfaz de usuario con un catálogo de sectores incorporado

Esta base de datos puede ser tan sencilla como un objeto de JavaScript en .JSON que liste un conjunto de sectores y sus datos más importantes, o tan compleja como se desee, como por ejemplo, haciendo uso de gestores de bases de datos relacionales (RDBMS).

Las recomendaciones de este proyecto son la utilización de JavaScript para el desarrollo del *frontend*, haciendo uso de *frameworks* como React o Vue, en conjunto con la librería Web3.js para la interfaz frontend-*blockchain*.

La razón por la que se recomiendan estas herramientas en particular es debido a su ubicuidad en el desarrollo de la web: el lenguaje de programación JavaScript es utilizado como motor de *frontend* (*client-side*) en el 98% de las páginas web en existencia [38], mientras que React y Vue son los *frameworks* líderes en el mercado para trabajar con JS en el diseño de *frontends* [39].

3.5 CAPAS DE OPERACIÓN

Para efectos ilustrativos, dividiremos el diseño de la aplicación descentralizada propuesta en 4 capas de operación:

- **Capa 1:** Datos de Ruta – capa fundamental del sistema correspondiente a información codificada y desplegada en Ethereum, en conjunto con los métodos implementados para la extracción de estos datos. Las instancias desplegadas de *RouteData* son parte de esta capa.
- **Capa 2:** Catalogación – capa de indexación de rutas desplegadas en Ethereum, diseñada para la gestión de referencias a los datos desplegados en la capa 1. Definida por los métodos que incorpora para la creación de nuevas rutas haciendo uso de fábricas, y la indexación

automática (o manual, optativamente) de dichas rutas en un catálogo. Las instancias desplegadas de los contratos *ScalesData*, *RouteCatalog* y *RouteFactory* operan en este nivel.

- **Capa 3:** Geografía – capa que define un sector geográfico determinado y referencia los datos desplegados de las rutas de escalada de dicho sector a través de catálogos de la capa 2. Las instancias del contrato *SectorData* representan la información de estos sectores y, por ende, conforman esta capa.
- **Capa 4:** Interfaz de Usuario – esta última capa es definida como la interfaz entre el usuario final del sistema, es decir, los escaladores que quieran consultar o registrar datos de una o más rutas de escalada desplegados en el sistema, y la *blockchain*. Las implementaciones de *frontend* que interactúan con los contratos de la capa 2 y 3 conforman este último nivel.

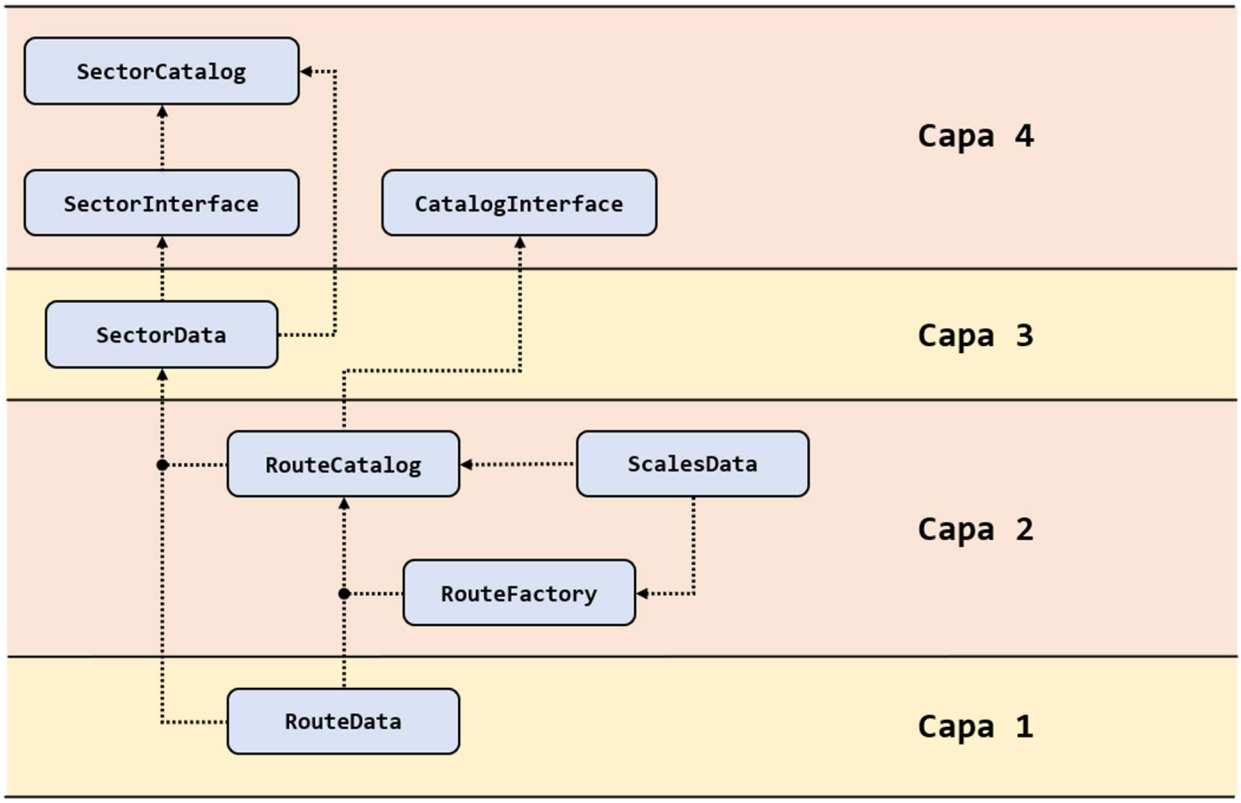


Figura 15: Capas operacionales del sistema propuesto.

3.6 EXPERIENCIA DE USUARIO

Se distinguen cuatro perfiles de usuario no excluyentes de interés para el diseño propuesto y se detalla la experiencia de usuario (UX) diseñada para cada uno de ellos. Se subcategorizan, además, en proveedores y *end-users*.

3.6.1 Administrador de Catálogo (*CatalogAdmin*)

Individuos u organizaciones que gestionan el registro de rutas de escalada en un catálogo indexado del registro distribuido. Este tipo de usuario es el proveedor del servicio fundamental para la organización del sistema distribuido, y provee dicho servicio a otros usuarios en la *blockchain* y frontends.

Opera primordialmente en la capa 2 de operación, definida en la sección anterior y, dependiendo de la configuración de control de acceso dada al catálogo desplegado, en la capa 4: el usuario puede desplegar su propia interfaz de usuario para el catálogo, o delegar dicha función a un tercero que cuente con los permisos de escritura necesario.

3.6.2 Administrador de Sector (*SectorAdmin*)

Individuos u organizaciones que gestionan el registro de rutas de escalada correspondientes a un sector geográfico determinado en el registro distribuido. Se da por supuesto que existe un contrato *RouteCatalog* desplegado en la *blockchain* y que el *SectorAdmin* cuenta al menos con permisos de escritura en dicho catálogo, ya sea porque ha sido configurado para ser público, porque *SectorAdmin* ha sido asignado los permisos requeridos o porque *SectorAdmin* es el mismo usuario que ha desplegado el contrato *RouteCatalog* a vincular.

El rango de operación de este segmento de usuarios corresponde a la capa 3, operando como proveedor del servicio de administración de *SectorData* y, opcionalmente, la capa 4 cuando se incorpora una interfaz de usuario para *SectorData*.

3.6.3 Creador de rutas (*Setter*)

Individuos que registran rutas de escalada en el registro distribuido. Se da por supuesto que quien registra la ruta es al menos parte del grupo que la ha desarrollado, sin embargo, esto no es un requerimiento impuesto por el sistema.

Este segmento de usuarios corresponde a uno de los *end-user* del sistema y su experiencia de usuario dependerá fuertemente de la calidad del servicio que proveen los otros usuarios que actúan en las capas 2 y 3. Su rango de operación corresponde a la capa 4, donde interactúa con las interfaces de usuario proveídas por otros usuarios o, alternativamente pero poco recomendado, a través de una interfaz de usuario local.

3.6.4 Consultores

Individuos que consultan el registro distribuido a través de las interfaces de usuario desplegadas por los usuarios proveedores. Su experiencia de usuario es similar a la de los *setters* en su dependencia de los proveedores de las capas 2 y 3. De la misma manera que los *setters*, su rango de operación corresponde a la capa 4.

3.7 DESPLIEGUE Y OPERACIÓN

A lo largo de esta sección se denominará como "dueño" de un *Smart Contract* a la *address* de Ethereum que realiza la transacción de despliegue del contrato.

3.7.1 RouteData

El despliegue de este tipo de contratos es, por diseño, gestionado por los contratos de fábrica y no se espera que el usuario los despliegue manualmente, sin embargo, el despliegue de estos contratos y su subsecuente registro en un catálogo es posible si se cuentan con los permisos de escritura correspondientes. A continuación, se listan, en orden de aparición, los parámetros de su despliegue, los cuales son requeridos incluso cuando se despliega a través de una fábrica:

1. `_name`: nombre de la ruta. Máximo 32 caracteres.
2. `_discipline`: disciplina de la ruta. Se ingresa el valor 0 si esta es *Free Climbing*, y el valor 1 en caso de tratarse de una ruta de *Bouldering*.
3. `_scale`: escala de graduación de dificultad usada. Se ingresa el valor 0 si se usa el estándar estadounidense (YDS para *Free Climbing*, Hueco para *Bouldering*) o el valor 1 en caso de usar el estándar francés (*French Decimal System* para *Free Climbing*, *Fontaine Bleau* para *Bouldering*).
4. `_grade`: la dificultad propuesta por el creador de la ruta, en la escala designada. Mientras mayor el valor de este parámetro, mayor la dificultad de la ruta. El contrato *ScalesData* referenciado por el catálogo contiene la información necesaria para encontrar la equivalencia entre la dificultad y el valor de este parámetro. Por ejemplo, una dificultad de V6 en la escala Hueco corresponderá a un valor de 7 para este parámetro.
5. `_latitude, _longitude`: la ubicación geográfica de la ruta, expresada en formato decimal.

3.7.2 ScalesData

El despliegue del *Smart Contract ScalesData,* que hace de interfaz para la decodificación del parámetro que agrupa la información de la disciplina, escala y grado de dificultad en un *RouteData*, es trivial y corresponde al primer paso necesario para el montaje de un sistema de catálogo.

Una vez desplegado, un contrato de *ScalesData* puede ser utilizado para desplegar múltiples contratos de catálogo y fábrica de rutas como se desee, sólo debe pasarse la *address* de *ScalesData* como parámetro de despliegue de dichos contratos.

3.7.3 RouteFactory

Antes de desplegar un contrato de catálogo es recomendado desplegar una fábrica de rutas en una instancia del contrato *RouteFactory*, la que podrá ser vinculada a múltiples instancias de *RouteCatalog* para la creación de contratos *RouteData* correctamente formateados.

El único parámetro de despliegue de este tipo de contratos es la *address* de un contrato del tipo *ScalesData* que ya se encuentre desplegado en la *blockchain*. Esta referencia es luego utilizada para confirmar el uso de la misma codificación para la información `[ disciplina : escala : dificultad ]` en el proceso de vinculación de un *RouteCatalog* con la fábrica.

3.7.4 RouteCatalog

El despliegue de un catálogo tiene como único parámetro de despliegue, de la misma manera que *RouteFactory*, la *address* de un contrato *ScalesData*.

A continuación, se presenta un flujograma que detalla el proceso de despliegue de una instancia del *Smart Contract RouteCatalog*, correspondiente a un sistema de catalogación de rutas de escalada. Este catálogo puede tener permisos de escritura públicos o privados, según sea la preferencia del usuario.

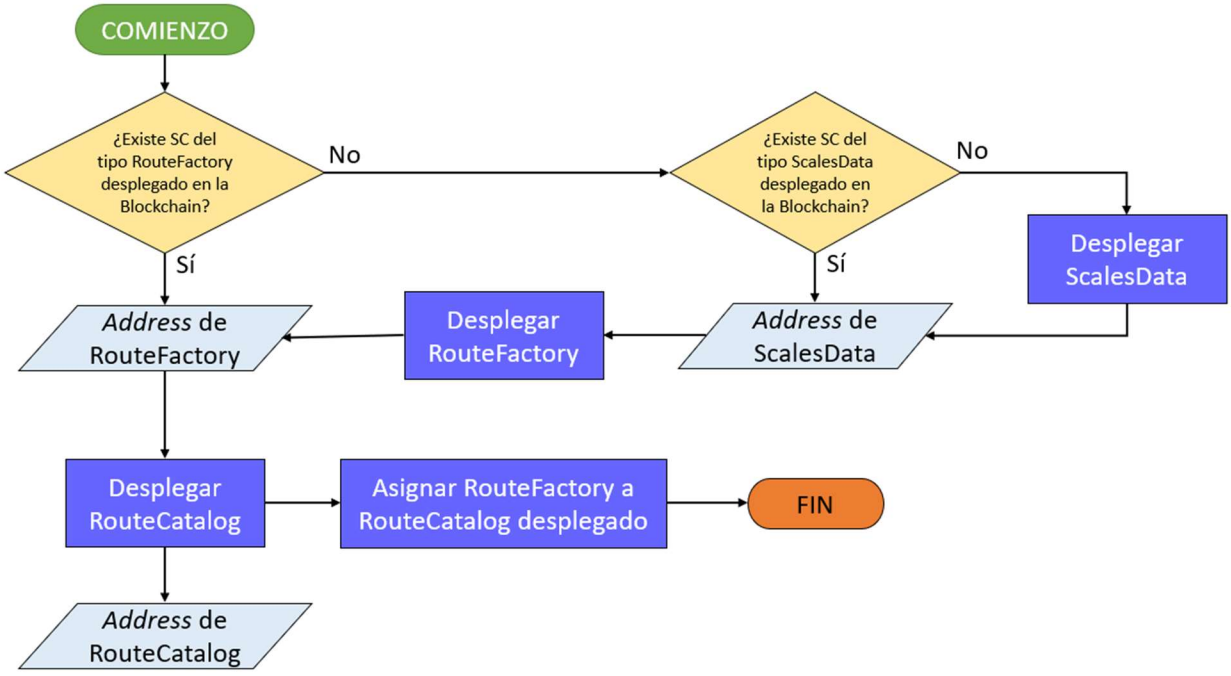


Figura 16: Flujograma del proceso de despliegue de un catálogo descentralizado de rutas de escalada en el sistema propuesto.

Una vez se ha desplegado este *Smart Contract* podemos interactuar con la instancia desplegada a través de transacciones de Ethereum para hacer llamados a las funciones del mismo. A continuación, se detallan las funciones básicas disponibles en dicho *Smart Contract*:

- `newEntry(address _routeData)` : ingresa los datos de la ruta de escalada desplegados en la *address* `_routeData` al catálogo.
- `delEntry(address _routeData)` : elimina los datos de la ruta de escalada desplegados en la *address* `_routeData` del catálogo. Solo el usuario que ingresó la ruta

al catálogo, el dueño del catálogo y administradores designados pueden realizar esta operación.

- `newFactoryEntry(bytes _name, uint8 _discipline, uint8 _scale, uint8 _grade, int24 _latitude, int32 _longitude)` : crea una nueva instancia del contrato *RouteData* a través de la *RouteFactory* asignada y es ingresada inmediatamente al catálogo. Esta función recibe como parámetros los mismos parámetros de instanciación de *RouteData*, señalados anteriormente.
- `getRoutesByName(bytes _name)` : retorna una lista de *address*es con todas las rutas que tengan el nombre `_name`. Con esta referencia luego se puede acceder a los datos alojados en dichas *address*es.

Este *Smart Contract* incluye una variable booleana que determina si los permisos de escritura en el catálogo son públicos o designados por el dueño y sus administradores. Por defecto el contrato está configurado para que los permisos de escritura sean designados individualmente, es decir privado, pero esta configuración puede ser modificada cambiando el valor de este booleano en el código fuente del contrato.

A continuación, se presentan las funciones de administración de los permisos del *Smart Contract* de catálogo, ejecutables por los administradores designados:

- `assignAdmin(address _user)` : solo ejecutable por el dueño del contrato. Designa al usuario con la *address* indicada como administrador del catálogo.
- `removeAdmin(address _user)` : solo ejecutable por el dueño del contrato. Revoca los permisos de administrador del usuario con la *address* indicada.
- `assignFactory(address _factory, bool _restrict)` : asigna un *Smart Contract* del tipo *RouteFactory* desplegado en la *address* `_factory` al catálogo. Además, recibe un parámetro booleano que cuando es `true` restringe el registro de nuevas rutas, de manera que sólo las rutas creadas por la *RouteFactory* referenciada sean opciones válidas.
- `toggleRestrictFactory()` : alterna la configuración que restringe el ingreso de entradas al catálogo a la fábrica designada o permite que los usuarios ingresen rutas al catálogo manualmente.
- `toggleUserAccess(address _user)` : alterna los permisos de escritura en el catálogo del usuario con la *address* indicada. Si el usuario cuenta con permisos de escritura la ejecución de esta función los revocará, mientras que en el caso contrario otorgará permisos al usuario.

3.7.5 SectorData

Los parámetros de despliegue de *SectorData* son los siguientes:

1. `_name`: nombre del sector. Máximo 32 caracteres.

2. `_latitude, _longitude:` la ubicación geográfica del sector, expresada en formato decimal.
3. `_catalog:` la *address* del contrato catálogo que usará el sector.

Una vez desplegado un contrato de sector este cuenta con métodos análogos a los de catálogo para la gestión del índice de rutas asignadas a dicho sector geográfico. A continuación, se señalan cuales son estos métodos:

- `addRoute(`*`address`* `_routeData)` : ingresa los datos de la ruta de escalada desplegados en la *address* `_routeData` al índice del sector. Si el catálogo vinculado no contiene esta ruta se rechaza y no se registra en el índice. Solo usuarios con permisos administrativos pueden realizar esta operación.
- `delRoute(`*`address`* `_routeData)` : elimina los datos de la ruta de escalada desplegados en la *address* `_routeData` del índice. Solo usuarios con permisos administrativos pueden realizar esta operación.
- `delRoute_i(uint256 _index)` : idéntico al método anterior, pero se referencia el índice bajo el cual se ha indexado la ruta que se desea eliminar. Solo usuarios con permisos administrativos pueden realizar esta operación.
- `findRoute(`*`address`* `_routeData)` : busca la ruta desplegada en la *address* indicada en el índice del sector. Si se encuentra, retorna el índice correspondiente a la ruta.
- `getRoutes()` : retorna una con referencias a todas las rutas indexadas en este sector. Con estas referencias luego se puede acceder a los datos alojados en dichos *RouteData*.

Por último, se tienen sólo dos funciones de carácter administrativo que son restringidas al dueño del contrato y análogas a las funciones administrativas del mismo nombre encontradas en *RouteCatalog*:

- `assignAdmin(`*`address`* `_user)` : Designa al usuario con la *address* indicada como administrador del sector.
- `removeAdmin(`*`address`* `_user)` : Revoca los permisos de administrador del usuario con la *address* indicada.

Capítulo 4. RESULTADOS Y ANÁLISIS

4.1 RESULTADOS

En la presente sección se muestra el funcionamiento de todas las funciones descritas en el diseño propuesto y se detalla la estimación de costos del despliegue de los contratos diseñados y de su uso en distintos escenarios en la *testnet* proveída por *Remix*. Además, se muestran los resultados para las primeras iteraciones con la finalidad de comparar con el diseño final e ilustrar el efecto que las optimizaciones implementadas tienen en los costos asociados.

4.1.1 Precio del Gas

A lo largo de esta sección y para el análisis de estos resultados se utiliza como precio mínimo del gas un valor de $20*10^{-9}$ ETH, expresado alternativamente como 20 Gwei. Este precio es determinado haciendo uso de la herramienta online EthGasStation [40], que mantiene estadísticas y analiza las transacciones que se validan en la red para entregar un tiempo estimado de confirmación según el precio que ofrezca el usuario por el gas a pagar de su transacción.

Dado a que la aplicación descentralizada no requiere de una confirmación rápida de sus transacciones, podemos disminuir costos si operamos con un tiempo promedio de confirmación de aproximadamente 8 horas y 15 minutos, el que se logra ofertando 20 Gwei por unidad de gas.

Predictions: Gas Used = 1; Gas Price = 20 gwei

Outcome	
% of last 200 blocks accepting this gas price	1.6393442623
Transactions At or Above in Current Txpool	546
Mean Time to Confirm (Blocks)	1939.1
Mean Time to Confirm (Seconds)	29824
Transaction fee (ETH)	0
Transaction fee (Fiat)	$0

Figura 17: Estadísticas del precio de gas proveídas por la plataforma EthGasStation

Se mostrarán resultados haciendo uso de distintos precios de gas, que impondrán los siguientes tiempos de confirmación aproximados:

gasPrice [Gwei]	Tiempo promedio de confirmación [s]
20	29824
22	16354
24	5294
26	1090
28	240

Tabla 2: Tiempos promedio de confirmación según precio ofertado por gas (gasPrice) en una transacción

Estos tiempos de confirmación variarán de acuerdo a la demanda de la red. Se utilizará como precio de referencia del Ether (ETH), para la conversión de costos a dólares, un valor de 1350 USD/ETH, consultado el día 17 de Julio de 2022.

4.1.2 RouteData

El contrato fundamental del sistema es desplegado con los siguientes parámetros:

- _name : 0x546865204e6f7365 (representación en bytes de “The Nose”)
- _discipline: 0
- _scale: 0
- _grade: 25
- _latitude: 3773292 (37.73292)
- _longitude: -11964025 (-119.64025)

Se despliegan dos instancias: la primera corresponde a la versión *naive* del contrato sin optimizaciones, mientras que la segunda implementa optimizaciones para la minimización de costos.

El costo de despliegue de la primera implementación de este contrato corresponde a 1073059 unidades de gas, mientras que la implementación final de *RouteData* tiene un costo de despliegue de 262385 unidades de gas. Los costos en dólares se detallan en la siguiente tabla:

gasPrice [Gwei]	Costo de Despliegue Naive [USD]	Costo de Despliegue Optimizado [USD]
20	28.97	7.08
22	31.87	7.79
24	34.77	8.50
26	37.66	9.21
28	40.56	9.92

Tabla 3: Costos de despliegue, expresados en dólares (USD), de RouteData en su versión inicial y final, según precio por gas ofertado en la transacción.

4.1.3 ScalesData

El despliegue del contrato *ScalesData* incurre en un costo de 2936320 unidades de gas para su versión no optimizada, y 1355590 unidades de gas para la versión final. Los costos en dólares se detallan a continuación:

gasPrice [Gwei]	Costo de Despliegue Naive [USD]	Costo de Despliegue Optimizado [USD]
20	79.28	36.60
22	87.21	40.26
24	95.14	43.92
26	103.06	47.58
28	111.00	51.24

Tabla 4:Costos de despliegue, expresados en dólares (USD), de ScalesData en su versión inicial y final, según precio por gas ofertado en la transacción.

4.1.4 RouteFactory

Se despliega un contrato del tipo *RouteFactory* pasándole como parámetro la *address* del contrato *ScalesData* optimizado desplegado anteriormente. Este proceso tiene un costo de transacción de 1011260 unidades de gas.

gasPrice [Gwei]	Costo de Despliegue [USD]
20	27.30
22	30.03
24	32.76
26	35.50
28	28.23

Tabla 5:Costos de despliegue, expresados en dólares (USD), de RouteFactory según precio por gas ofertado en la transacción.

Luego, se ejecuta la función *newRoute(...)* para la creación y despliegue de una nueva ruta (*RouteData*) independiente de un catálogo. Se utiliza la misma información de ruta que ha sido señalada anteriormente, correspondiente a los datos de la ruta "The Nose" en el Yosemite National Park de Estados Unidos.

Por último, se ejecuta la función *newCatalogEntry(...)* la cual es idéntica a la anterior, con la salvedad de que además registrará la ruta desplegada en un catálogo que se indique como parámetro de la función. Esto quiere decir que además del costo de gas de la creación de ruta se incurrirá en costos de gas en la ejecución de las funciones de registro contenidas en el *RouteCatalog* referenciado.

Función	Costo [GAS]
newRoute(…)	236,337
newCatalogEntry(…)	383,874

Tabla 6:Costos de transacción en unidades de gas para las funciones del contrato RouteFactory

gasPrice	20 [Gwei]	22 [Gwei]	24 [Gwei]	26 [Gwei]	28 [Gwei]
Costo newRoute [USD]	6.38	7.02	7.66	8.30	8.93
Costo newCatalogEntry [USD]	10.36	11.40	12.44	13.47	14.51

Tabla 7: Costos de transacción, expresados en dólares (USD), de las funciones del contrato RouteFactory, de acuerdo al precio por gas ofertado en la transacción.

4.1.5 RouteCatalog

Se despliega un contrato del tipo *RouteCatalog* con su parámetro de despliegue siendo la *address* del contrato *ScalesData* desplegado anteriormente. Este proceso tiene un costo de transacción de 2676089 unidades de gas.

gasPrice [Gwei]	Costo de Despliegue [USD]
20	72.25
22	79.48
24	86.71
26	93.93
28	101.16

Tabla 8:Costos de despliegue, expresados en dólares (USD), de RouteCatalog según precio por gas ofertado en la transacción.

Tras el despliegue se ejecutan cada una de las funciones disponibles en este contrato, utilizando como parámetros de creación de ruta en los métodos que ejecuten esta tarea el mismo set de datos indicado en la sección de *RouteData*.

Se asigna la fábrica desplegada y se utiliza para crear rutas y registrarlas en el catálogo. Esta operación de escritura será la más usada del sistema, y tiene un costo en dólares entre USD $10.51 y USD $14.72, dependiendo del tiempo de confirmación que queramos tener que esperar.

En la gran mayoría de los casos, el registro de una ruta no es cuestión de suma urgencia, y por lo tanto podemos utilizar el sistema con una oferta de precio del gas de sólo 20 Gwei, minimizando el costo de esta operación.

Por último, se destaca que la ejecución de la función que alterna el estado de los permisos de escritura de un usuario determinado *toggleUserAccess(...)* tiene un menor costo de ejecución cuando se está reiniciando esta variable a su estado OFF, es decir, un valor booleano False.

A continuación, se desglosan los costos de cada una de las funciones de este contrato:

Función	Costo [GAS]
assignAdmin(…)	46,624
assignFactory(…)	81,733
delEntry(…)	49,874
newEntry(…)	169,552
newFactoryEntry(…)	389,336
removeAdmin(…)	24,655
toggleRestrictFactory(…)	28,755
toggleUserAccess(…) ON	46,624
toggleUserAccess(…) OFF	24,724

Tabla 9:Costos de transacción en unidades de gas para las funciones del contrato RouteCatalog

gasPrice	20 [Gwei]	22 [Gwei]	24 [Gwei]	26 [Gwei]	28 [Gwei]
Costo assignAdmin [USD]	1.26	1.38	1.51	1.64	1.76
Costo assignFactory [USD]	2.21	2.43	2.65	2.87	3.09
Costo delEntry [USD]	1.35	1.48	1.62	1.75	1.89
Costo newEntry [USD]	4.58	5.04	5.49	5.95	6.41
Costo newFactoryEntry [USD]	10.51	11.56	12.61	13.67	14.72
Costo removeAdmin [USD]	0.67	0.73	0.80	0.87	0.93
Costo toggleRestrictFactory [USD]	0.78	0.85	0.93	1.01	1.09
Costo toggleUserAccess ON [USD]	1.26	1.38	1.51	1.64	1.76
Costo toggleUserAccess OFF [USD]	0.67	0.73	0.80	0.87	0.93

Tabla 10:Costos de transacción, expresados en dólares (USD), de las funciones del contrato RouteCatalog, de acuerdo al precio por gas ofertado en la transacción.

Se confirma con estas pruebas que la ejecución de aquellas funciones que requieren permisos especiales (*owner*, *admin, access*) se encuentra restringida como se detalla en el diseño propuesto.

4.1.6 SectorData

Se despliega un contrato del tipo *SectorData* con los siguientes parámetros:

- _name: 0x596f73656d697465204e6174696f6e616c205061726b (representación en bytes de “Yosemite National Park”)
- _latitude: 3784412 (37.84412)
- _longitude: -11955900 (-119.55900)
- _catalog: la *address* de un contrato de catálogo desplegado que tenga rutas registradas

Este proceso tiene un costo de transacción de 1493485 unidades de gas.

gasPrice [Gwei]	Costo de Despliegue [USD]
20	40.32
22	44.36
24	48.39
26	52.42
28	56.45

Tabla 11:Costos de despliegue, expresados en dólares (USD), de SectorData, según precio por gas ofertado en la transacción.

Tras el despliegue del contrato se prueban todas las funciones de este y se verifica el control de acceso. Además, se constata que sólo pueden ser añadidas rutas que se encuentran indexadas en el catálogo vinculado.

A continuación, se presenta el detalle de costos de transacción para cada función del contrato:

Función	Costo [GAS]
addRoute(…)	79,844
assignAdmin(…)	46,560
delRoute(…)	31,013
delRoute_i(…)	30,388
removeAdmin(…)	24,680

Tabla 12:Costos de transacción en unidades de gas para las funciones del contrato SectorData

gasPrice		20 [Gwei]	22 [Gwei]	24 [Gwei]	26 [Gwei]	28 [Gwei]
Costo addRoute	**[USD]**	2.16	2.37	2.59	2.80	3.02
Costo assignAdmin	**[USD]**	1.26	1.38	1.51	1.63	1.76
Costo delRoute	**[USD]**	0.84	0.92	1.00	1.09	1.17
Costo delRoute_i	**[USD]**	0.82	0.90	0.98	1.07	0.87
Costo removeAdmin	**[USD]**	0.67	0.73	0.80	0.87	0.93

Tabla 13:Costos de transacción, expresados en dólares (USD), de las funciones del contrato SectorData, de acuerdo al precio por gas ofertado en la transacción.

Se confirma que el acceso a los datos indexados no se encuentra restringido de ninguna manera y que cualquier usuario de la red de Ethereum puede interactuar con estos *Smart Contract*. Además, se confirma con estas pruebas que la ejecución de aquellas funciones que requieren permisos especiales (*owner*, *admin*) se encuentra restringida como se detalla en el diseño propuesto.

4.2 ANÁLISIS DE RESULTADOS

4.2.1 Optimizaciones

A lo largo de este trabajo se desarrollan iterativamente los diseños de los *Smart Contract* que definen y sustentan el sistema. Este proceso, como se muestra en los resultados, deriva en una reducción de costos de transacción considerable: se requiere 25% del gas requerido originalmente para el despliegue de *RouteData*, mientras que en *ScalesData* la reducción es del 54%.

Esta disminución de costos se justifica por los siguientes detalles de diseño en su implementación:

- Eliminación de datos redundantes para la identificación de rutas de escalada, como la descripción, la que puede incorporarse luego a nivel de *frontend* sin mayores problemas de centralización de datos.
- Cambiar la declaración de variables a tipos (*data types*) de menor tamaño. Puntualmente, las coordenadas GPS eran almacenadas como un string de 10 bytes para la latitud y otros 10 bytes para la longitud. Se puede almacenar esta información en 3 bytes para la latitud, lo que resulta en un espacio de coordenadas entre -83.88608 y +83.88608 que, si bien no cubre todas las coordenadas posibles para la latitud, tiene un rango adecuado para abarcar el territorio entre el punto más al norte y más al sur alcanzables por tierra (excluyendo los polos). Se utilizan 4 bytes para el almacenamiento de la longitud, que cubre todo el espacio de coordenadas posibles. Así, se reduce el espacio de almacenamiento requerido para la ubicación de una ruta o sector desde 20 bytes a sólo 7.

- El orden de la declaración de variables importa: Ethereum trata de agrupar datos para registrarlos en el espacio de almacenamiento de los *Smart Contracts* en *slots* de 32 bytes. Esto quiere decir que la declaración de una variable de 3 bytes como la latitud, seguida de una de 32 bytes como el nombre, y luego los 4 bytes de la longitud ocuparían 3 slots de 32 bytes distintos, en lugar de los sólo dos que se asignan ubicando la declaración de latitud y longitud contiguamente.
- Se codifica y agrupa información en variables de menor tamaño. Específicamente para los datos de ruta se opta por incorporar la información de disciplina, escala de dificultad y grado de dificultad en una misma variable, en lugar de tres. Con ello se reduce el almacenamiento requerido para registrar estos datos de 3 bytes a solo 1.

4.2.2 Modularidad integrada

Se incorporan métodos en *RouteCatalog* que hacen llamados directamente a los métodos de creación de rutas de *RouteFactory* de manera que se automatiza el proceso de creación y el posterior registro de esta ruta en el índice del catálogo. Esto tiene como finalidad minimizar el número de transacciones requeridas para este proceso y resulta en una reducción del 4.28% en los costos de registro de rutas cuando se crean a través de una fábrica, y un 10.95% cuando desplegamos manualmente los datos de la ruta en un contrato *RouteData* y registramos su dirección en el catálogo.

Método	Costo [USD]
RouteCatalog.newFactoryEntry	10.51
RouteCatalog.newEntry(RouteFactory.newRoute)	10.96
RouteCatalog.newEntry(RouteData)	11.66

Tabla 14:Comparación de costos del proceso integrado de creación y registro de rutas vs el registro manual de rutas creadas en una RouteFactory

4.2.3 Viabilidad

El diseño presentado, por limitaciones del protocolo de la *blockchain* de Ethereum, hace responsable del pago de gas para la ejecución de las funciones del sistema al usuario que las ejecuta, sin embargo, este costo puede ser abordado por comunidades locales de escaladores e incluso, por ejemplo, entidades gubernamentales que administren parques nacionales donde se halle un foco de rutas de escalada.

Veamos el ejemplo del Yosemite National Park en Estados Unidos, uno de los lugares más confluidos en el mundo para practicar la escalada en la naturaleza [41]. Se estima que el parque nacional recibe entre 25 mil y 50 mil escaladores-día al año, de acuerdo a números del *National Park Service* (NPS) de Estados Unidos [42].

En *theCrag.com*, una de las plataformas estudiadas, se registran más de 3100 rutas de escalada agrupadas en unos 200 sectores dentro del parque nacional. Si trabajamos bajo el supuesto de que esos 200 sectores podemos incorporarlos en 100 contratos de *SectorData* y diferenciarlos *off-chain* en su implementación, los costos de registrar esta estructura en una plataforma de catálogo de rutas del parque nacional serán los siguientes:

Número de Despliegues	**Contrato**	**Costo por despliegue**		**Costo Total**	
		[GAS]	**[USD]**	**[GAS]**	**[USD]**
1	ScalesData	2,936,320	36.60	2,936,320	36.60
1	RouteFactory	1,011,260	27.30	1,011,260	27.30
1	RouteCatalog	2,676,089	72.25	2,676,089	72.25
100	SectorData	1,493,485	40.32	149,348,500	4,032.41
3100	RouteData via RouteCatalog	389,336	10.51	1,206,941,600	32,587.42
COSTO TOTAL DE DESPLIEGUE					
[gas]			**[USD]**		
1,362,913,769			36,798.67		

Tabla 15: Desglose de costos de despliegue de una implementación de la plataforma diseñada para acomodar las rutas de escalada y sectores de Yosemite National Park

Además, se incurrirá en costos adicionales de registro con la ejecución de las funciones que incorporan las rutas ya desplegadas en el catálogo en los *SectorData*, y costos administrativos de asignar la fábrica y un cuerpo de administradores tanto para el catálogo como para los sectores.

Número de Operaciones	**Operación**	**Costo por operación**		**Costo Total**	
		[GAS]	**[USD]**	**[GAS]**	**[USD]**
1	RouteCatalog .assignFactory	81,733	2.21	81,733	2.21
3100	SectorData .addRoute	79,844	2.16	247,516,400	6,682.94
10	RouteCatalog .assignAdmin	46,624	1.26	466,240	12.59
200	SectorData .assignAdmin	46,560	1.26	9,312,000	251.42
COSTO TOTAL DESPLIEGUE ADMINISTRATIVO ADICIONAL					
[gas]			**[USD]**		
257,376,373			6,949.16		

Tabla 16: Desglose de costos por operación requeridos para finalizar el despliegue de la implementación del sistema diseñado para Yosemite National Park

Con ello vemos que el costo total del despliegue en la *blockchain* del sistema que describe al área del Yosemite National Park, subdividido en 200 sectores representados por 100 contratos *SectorData* en la *blockchain,* conteniendo la información de un total de 3100 rutas de escalada indexadas en un catálogo privado, es de alrededor de USD $43,750.

Consideremos el registro de 24 nuevas rutas anualmente, un estimado un tanto elevado para un área bien establecida, pero una buena representación de lo que podría ser un área donde se está desarrollando y creciendo la cultura del deporte. El costo de estos nuevos registros será de USD $12.67 por ruta, es decir, un total de USD $304 anuales.

Si consideramos los costos de hosting de un nodo de Ethereum, por ejemplo, utilizando los servicios de AWS que ofrece planes *on-demand* [43], podemos reducir el uso de un nodo para que sea desplegado y utilizado en promedio 2 horas diarias y mantener el sistema operacional. Esto tendría un costo de mantención anual de alrededor de USD $100.

A esto se le debe sumar el costo de mantener un servidor en un servicio de *cloud storage* para proveer una interfaz de usuario mediante las librerías web3.js para la interacción con el nodo de Ethereum desplegado, y React.js para el diseño de la interfaz web. Tomando como ejemplo AWS, que ofrece un plan de costos *on-demand* para sus servidores EC2 con un costo de USD $0.0042 por hora [44], entonces el costo total anual de mantener la interfaz de usuario operacional es de aproximadamente USD $37.

Finalmente, el costo total de mantener este sistema operacional en la blockchain es de aproximadamente USD $450 anuales si contabilizamos cierta holgura. Además, se deberá incurrir en gastos en recursos humanos para la administración y mantención del sistema; estos costos serán determinados dependiendo de la manera en la que se implementa el sistema.

Dependiendo del nivel de desarrollo del deporte y del contexto económico en el que se encuentra la zona que se desea representar con esta plataforma, se estima que este proyecto podría tener viabilidad económica para su implementación si se incorpora ya sea un esquema de *crowdfunding* organizado por las comunidades locales, o una propuesta comercial similar a las presentadas por las plataformas de referencia, con un servicio de suscripción. La investigación de alternativas para ello se propone como trabajo futuro.

Capítulo 5. CONCLUSIONES Y TRABAJO FUTURO

5.1 CONCLUSIONES

El tamaño total de una base de datos que represente el ejemplo ilustrado del Yosemite National Park con la información de 3100 rutas de escalada, registradas en 200 sectores, codificados como se indica en este informe, es de aproximadamente 200 kB. El costo anual de almacenar este volumen de datos en una aplicación tradicional, como por ejemplo utilizando los servicios de Amazon AWS S3 o Google Cloud Storage, es marginal [45] y, asumiendo que tanto una aplicación tradicional como una implementación integrada de este proyecto en la web, mediante *frontends* en servidores centralizados, tienen un costo de *hosting* similar, la gran diferencia entre estos dos paradigmas de diseño está principalmente en el almacenamiento y la accesibilidad de los datos.

El costo de mantención de los datos una vez son desplegados en Ethereum es cero, y el acceso a estos datos es completamente abierto a cualquier individuo o aplicación dentro o fuera de la *blockchain* que tenga el código fuente proveído de manera libre por este trabajo bajo la licencia MIT. Esto quiere decir que un esquema como el presentado promueve el desarrollo de herramientas y plataformas construidas incorporando las funcionalidades implementadas en el diseño de este proyecto.

Por lo tanto, *se concluye que el trabajo realizado sienta las bases del desarrollo de aplicaciones descentralizadas que requieran acceder a información pública de rutas de escalada en Ethereum*.

En cuanto a la viabilidad económica del montaje de esta plataforma, como se ha mostrado en el análisis de los resultados obtenidos de la estimación de costos del sistema, se concluye que *podría ser viable establecer y mantener un sistema de registro descentralizado de rutas de escalada haciendo uso del diseño propuesto,* sin embargo se requieren estudiar los distintos contextos y circunstancias que determinarán las estructuras de costos específicas para el despliegue de este sistema, lo que queda propuesto como trabajo futuro.

Finalmente, con este trabajo se demuestra la viabilidad técnica y el libre acceso a los datos que se despliegan en la red de Ethereum, se logran mostrar los mecanismos internos mediante los cuales los usuarios pueden modificar el registro compartido, y se demuestra que estos mecanismos indexan correctamente la información en Smart Contracts. Con ello, *se concluye que el trabajo realizado y presentado en el presente informe cumple con los objetivos generales y específicos propuestos.*

5.2 TRABAJO FUTURO

Habiendo comprobado la viabilidad técnica de este proyecto y las primeras exploraciones de su viabilidad económica, queda propuesto como trabajo futuro el realizar la evaluación de alternativas comerciales o de *crowdfunding* para la implementación real de este proyecto, y con ello la reiteración del diseño de este proyecto de acuerdo a las nuevas necesidades delineadas por dicho trabajo futuro.

Otro punto importante es el del diseño de las interfaces de usuario en el *frontend*, el cual se menciona que escapa de los alcances de este proyecto. Un trabajo futuro importante es el de diseñar interfaces de usuario compactas y claras que aprovechen las funcionalidades establecidas por este trabajo. Con el diseño de interfaces de usuario viene además incorporado el estudio de los contextos más específicos para los cuales se diseña esta plataforma, motivando potencialmente el diseño de nuevos *Smart Contract* para la gestión e integración prolija de las nuevas herramientas.

Se propone también el diseño de mecanismos que, mediante el menor número de transacciones posible, permitan registrar simultáneamente un volumen importante de rutas en este sistema. Como se demostró en el análisis de resultados de este trabajo, el proceso de registro de rutas en sectores en los que se han desarrollado un gran número de rutas, como Yosemite National Park, resulta costoso dada la gran cantidad de transacciones que han de realizarse. Se estima posible reducir estos costos a cerca de la mitad si se diseñan mecanismos para codificar y desplegar este gran volumen de datos en la red, en menos transacciones.

Una exploración más a cabalidad de las vulnerabilidades de este sistema, en particular para el control de permisos de escritura, es importante y debiese realizarse antes de implementar esta plataforma. Se debe comprobar con este estudio también la practicidad de utilizar cuentas de Ethereum como credenciales de control de acceso, o si debiese implementarse un sistema ad-hoc para la autenticación de usuarios.

El constante cambio que tiene la tecnología *blockchain* motiva, además, a realizar una reevaluación de los costos de transacción una vez se realicen cambios en el protocolo. Actualmente la red de Ethereum se encuentra en un proceso de transición de un protocolo de consenso *Proof of Work* a uno *Proof of Stake*, que podría significar un cambio importante en los costos de transacción. Se teoriza un aumento sustancial del volumen de transacciones, y por ende una menor competitividad por gas. Luego resulta importante volver a evaluar el factor económico de este proyecto con un nuevo precio de gas e incluso incorporar más funcionalidades para compensar.

Por último, se propone volver a iterar sobre el diseño propuesto y proponer alternativas que cumplan con los mismos requisitos y funcionalidades: si bien se ha procurado a lo largo de este trabajo implementar optimizaciones y se logra una reducción de costos considerable, pueden existir aún alternativas a los mecanismos incorporados en el sistema que incurran en menos costos, sin sacrificar demasiado la practicidad de la plataforma.

BIBLIOGRAFÍA

[1] International Federation of Sport Climbing, «IFSC Annual Report 2021,» 2021. [En línea]. Available: https://reports.ifsc-climbing.org/2021/72-1.

[2] International Federation of Sport Climbing, «IFSC Annual Report 2019,» 2019. [En línea]. Available: https://cdn.ifsc-climbing.org/images/ifsc/Footer/Annual_Reports/ifsc-annual-report-2019-final-website.pdf.

[3] S. A. Cornejo, «Fanáticos de la escalada se han duplicado en los últimos cinco años en Chile - EyN,» 2018. [En línea]. Available: http://www.economiaynegocios.cl/noticias/noticias.asp?id=491835.

[4] theCrag, «theCrag: the largest collaborative rock climbing & bouldering platform,» 2022. [En línea]. Available: https://www.thecrag.com.

[5] UKClimbing, «UKClimbing,» 2022. [En línea]. Available: https://www.ukclimbing.com/.

[6] Similarweb LTD, «Top Climbing Websites Ranking in July 2022 | Similarweb,» 2022. [En línea]. Available: https://www.similarweb.com/top-websites/category/sports/climbing/. [Último acceso: July 2022].

[7] Similarweb LTD, «About Us | Our Mission, Company, and History | Similarweb,» 2022. [En línea]. Available: https://www.similarweb.com/corp/about/.

[8] B. J. Sbarra, «What It Takes To Develop A New Climbing Route,» Recreational Equipment, Inc., May 2019. [En línea]. Available: https://www.rei.com/blog/climb/what-it-takes-to-develop-a-new-climbing-route.

[9] V. Buterin, «Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform,» 2013.

[10] W. Stallings, Cryptography and Network Security: Principles and Practice, Prentice Hall, 1999.

[11] L. Wang, X. Shen, J. Li, J. Shao y Y. Yang, «Cryptographic primitives in blockchains,» *Journal of Network and Computer Applications,* vol. 127, pp. 43-58, 2019.

[12] J. Stolfi, «Cryptographic Hash Function,» 2008. [En línea]. Available: https://en.wikipedia.org/wiki/Cryptographic_hash_function#/media/File:Cryptographic_Hash_Function.svg.

[13] M. Rauchs, A. Glidden, B. Gordon, G. C. Pieters, M. Recanatini, F. Rostand, K. Vagneur y B. Z. Zhang, «Distributed Ledger Technology Systems: A Conceptual Framework,» *SSRN Electronic Journal,* 2018.

[14] BBVA Communications, «What is the difference between DLT and blockchain?,» May 2018. [En línea]. Available: https://www.bbva.com/en/difference-dlt-blockchain/.

[15] National Bureau of Standards, «Federal Information Processing Standards Publication: Data Encryption Standard,» Gaithersburg, MD, 1977.

[16] National Bureau of Standards, «Federal Information Processing Standards Publication: DES modes of operation,» Gaithersburg, MD, 1980.

[17] S. Nakamoto, «Bitcoin: a peer-to-peer electronic cash system,» 2008. [En línea]. Available: http://bitcoin.org/bitcoin.pdf.

[18] C. Pinzón y C. Rocha, «Double-spend Attack Models with Time Advantange for Bitcoin,» *Electronic Notes in Theoretical Computer Science,* vol. 329, pp. 79-103, 2016.

[19] T. Takenobu, «Ethereum EVM Illustrated,» Github Pages, 2018. [En línea]. Available: https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf.

[20] A. M. Antonopoulos, Masterign Bitcoin, O'Reilly Media, 2017.

[21] W. Fang, W. Chen, W. Zhang, J. Pei, W. Gao y G. Wang, «Digital signature scheme for information non-repudiation in blockchain: a state of the art review,» *EURASIP Journal on Wireless Communications and Networking,* vol. 2020, nº 1, 2020.

[22] A. Bolfing, Cryptographic Primitives in Blockchain Technology, Oxford University Press, 2020.

[23] H. Jung y H.-N. Lee, «ECCPoW: Error-Correction Code based Proof-of-Work for ASIC Resistance,» *Symmetry,* vol. 12, nº 6, p. 988, 2020.

[24] J. Waldman, «Blockchain Fundamentals,» Microsoft, 2018. [En línea]. Available: https://docs.microsoft.com/en-us/archive/msdn-magazine/2018/march/blockchain-blockchain-fundamentals.

[25] N. Conte, «Visualizing the Rise of Cryptocurrency Transactions,» September 2021. [En línea]. Available: https://www.visualcapitalist.com/visualizing-the-rise-of-cryptocurrency-transactions/.

[26] N. Szabo, «Formalizing and Securing Relationships on Public Networks,» *First Monday,* vol. 2, September 1997.

[27] A. M. Antonopoulos, Mastering Ethereum, O'Reilly Media, 2018.

[28] State of the Dapps, «Dapp Statistics,» [En línea]. Available: https://www.stateofthedapps.com/stats. [Último acceso: 11 December 2021].

[29] Ethereum Foundation, «Web3 Javascript API,» [En línea]. Available: https://github.com/ethereum/wiki/wiki/JavaScript-API.

[30] Consensys Software Inc., «MetaMask,» [En línea]. Available: https://metamask.io/.

[31] Protocol Labs, «IPFS,» [En línea]. Available: https://ipfs.io.

[32] Ethereum Foundation, «Swarm,» [En línea]. Available: https://www.ethswarm.org/.

[33] Ethereum Foundation, «Whisper | Ethereum Wiki,» [En línea]. Available: https://eth.wiki/concepts/whisper/whisper.

[34] Ethereum Foundation, «Solidity Programming Language,» [En línea]. Available: https://soliditylang.org/.

[35] Ethereum Foundation, «Solidity Documentation,» [En línea]. Available: https://docs.soliditylang.org/.

[36] G. Wood, «Ethereum: A Secure Decentralised Generalised Transaction Ledger,» 2022.

[37] Ethereum Foundation, «Remix Project,» [En línea]. Available: https://remix-project.org/.

[38] W3Techs, «Usage statistics of JavaScript as client-side programming language on websites,» [En línea]. Available: https://w3techs.com/technologies/details/cp-javascript.

[39] devographics, «State of JavaScript - Front End Frameworks,» [En línea]. Available: https://2019.stateofjs.com/front-end-frameworks/.

[40] Concourse Open Community, «ETH Gas Station,» [En línea]. Available: https://legacy.ethgasstation.info/calculatorTxV.php.

[41] Elevated Adventurer, «Best Rock Climbing Routes & Locations in the World in 2022,» 2022. [En línea]. Available: https://elevatedadventurer.com/what-are-the-best-rock-climbing-cliffs-in-the-world/.

[42] National Park Service U.S. Department of Interior, «Climbing Safety,» 2022. [En línea]. Available: https://www.nps.gov/yose/planyourvisit/climbing_safety.htm.

[43] AWS, «Amazon Managed Blockchain for Ethereum pricing,» [En línea]. Available: https://aws.amazon.com/managed-blockchain/pricing/ethereum/.

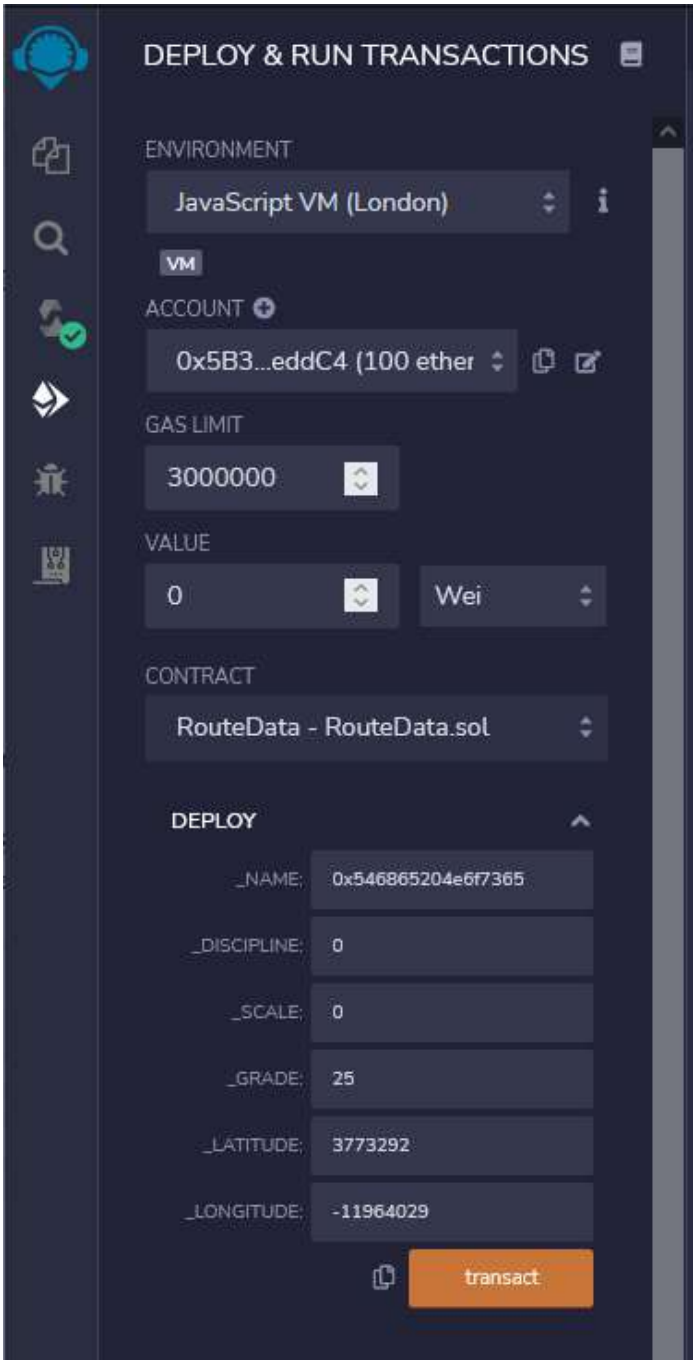
[44] AWS, «Amazon EC2 On-Demand Pricing,» [En línea]. Available: https://aws.amazon.com/ec2/pricing/on-demand/.

[45] Google LLC, «Cloud Storage pricing,» [En línea]. Available: https://cloud.google.com/storage/pricing.

[46] Semrush, «ukclimbing.com Overview, Traffic Analytics,» 2022. [En línea]. Available: https://www.semrush.com/analytics/traffic/overview/ukclimbing.com.

ANEXOS

ANEXO A. REMIX IDE



> NODO VIRTUAL

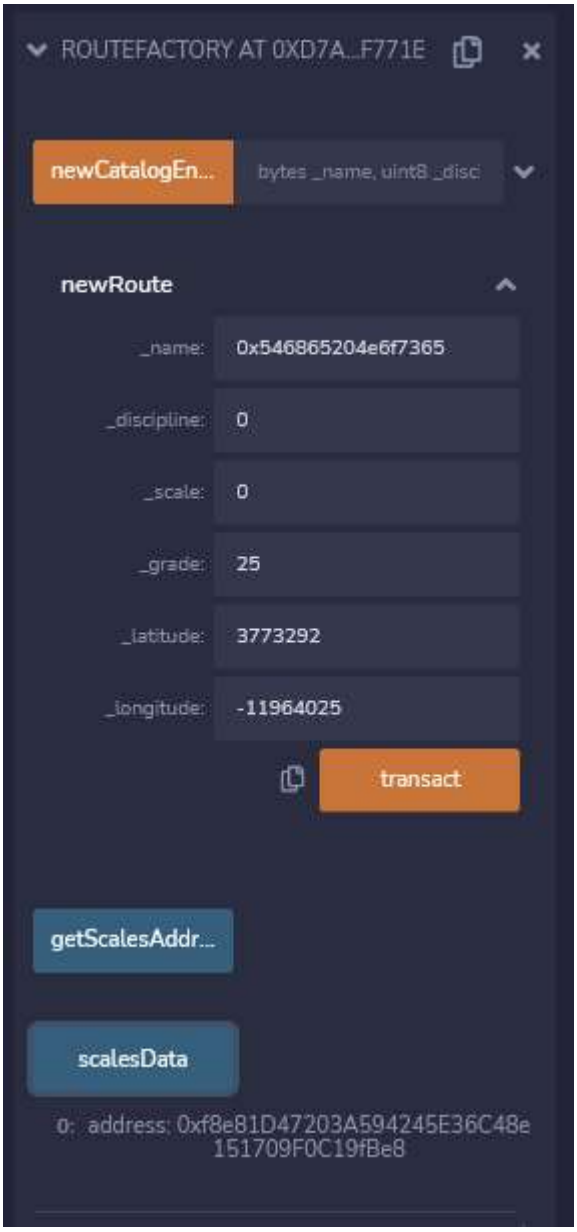
> CUENTA DE ORIGEN DE TRANSACCIÓN
Remix proporciona 15 cuentas desplegadas en el nodo virtual

> MÁXIMO GAS A GASTAR

> ETHER A ENVIAR – excluye pago de gas.

> CONTRATO A DESPLEGAR

> PARÁMETROS DE DESPLIEGUE



> INSTANCIA DE CONTRATO CON LA QUE SE INTERACTÚA

> FUNCIÓN PARA AGREGAR NUEVA ENTRADA A CATÁLOGO

> FUNCIÓN PARA DESPLEGAR UNA NUEVA RUTA (*RouteData*) EN LA BLOCKCHAIN

Se incluyen sus parámetros de despliegue

> OUTPUT DE UNA FUNCIÓN DE CONSULTA A STORAGE

ANEXO B. CÓDIGO Y DOCUMENTACIÓN

Todo el código desarrollado y la documentación generada en este proyecto puede encontrarse en el repositorio público de GitHub señalado a continuación:

https://github.com/Tino-HG/climbing-route-data-dapp

ANEXO C. CODIFICACIÓN DE DISCIPLINA Y GRADO DE DIFICULTAD