BÚSQUEDA AUTOMÁTICA DE CONTRATOS ETHEREUM EN TIEMPO REAL

REAL-TIME SEARCH OF ETHEREUM CONTRACTS

TRABAJO FIN DE GRADO
CURSO 2023-2024

AUTOR
MANUEL DAVID PÉREZ BELIZÓN

DIRECTORES
PABLO GORDILLO ALGUACIL Y JESÚS CORREAS FERNÁNDEZ

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

BÚSQUEDA AUTOMÁTICA DE CONTRATOS ETHEREUM EN TIEMPO REAL

REAL-TIME SEARCH OF ETHEREUM CONTRACTS

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

AUTOR
MANUEL DAVID PÉREZ BELIZÓN

DIRECTOR
PABLO GORDILLO ALGUACIL Y JESÚS CORREAS FERNÁNDEZ

CONVOCATORIA: JUNIO 2024

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

27 DE MAYO DE 2024

DEDICATORIA

A mis padres que han sufrido conmigo y me han apoyado durante toda mi etapa universitaria, a mi hermano Rafa, que para mí es un modelo a seguir que siempre está ahí para echarme un cable si lo necesito, a mis lalos que espero que se sientan muy orgullos de mí y a todos mis amigos que han tenido que aguantar mis mejores y peores etapas estos años. Gracias a todos.

AGRADECIMIENTOS

Gracias a todas aquellas personas que me han acompañado durante todo este tiempo y me han hecho ser quien soy. Gracias a Jesús y a Pablo, los directores de este trabajo, que me han guiado y ayudado siempre que lo he necesitado.

RESUMEN

Búsqueda automática de contratos Ethereum en tiempo real

En la red principal de Ethereum se despliegan continuamente nuevos contratos y su código binario está disponible públicamente, lo que proporciona una enorme colección de código real de aplicaciones reales que se puede utilizar para realizar investigaciones sobre ellos. En la red de Ethereum se almacena el código compilado del contrato, aunque en algunos casos los desarrolladores de contratos publican el código fuente en repositorios públicos, como por ejemplo Etherscan.

Para realizar investigaciones sobre esta base de código, resulta de gran interés poder disponer de una herramienta que pueda cargar contratos inteligentes que estén verificados en la red de Ethereum para así buscar aquellos que cumplan condiciones de selección complejas, como, por ejemplo: tipo de licencia del código, versión del compilador, optimizaciones, así como otras condiciones complejas.

En este trabajo se introducen los conceptos fundamentales de la tecnología de cadena de bloques, se describen las principales características de Ethereum, se revisa la exploración de datos en Ethereum junto a los repositorios públicos más relevantes que almacenan código fuente de contratos inteligentes desplegados en Ethereum, y se desarrolla un prototipo de aplicación para la descarga, búsqueda y compilación de contratos inteligentes de acuerdo con diversas condiciones de selección.

Palabras clave

Cadena de bloques, Ethereum, Etherscan, aplicación web, aplicación de consola, contratos inteligentes.

ABSTRACT

Real-time search of Ethereum contracts

In Ethereum main net smart contracts are continuously being deployed with their binary code publicly accessible, this provides a huge collection of real code from real applications that can be used to research them. Ethereum main net stores the compiled code of every smart contract and in some cases, smart contract developers verify the source code in public repositories, like Etherscan.

For researching this code database, it is beneficial to have a tool that can load verified smart contracts from the Ethereum main net to query smart contracts that satisfy several complex conditions, like, for example: the license type of the code, the compiler version, optimizations, or any other complex condition.

In this thesis blockchain technology fundamentals are given, the main aspects of Ethereum are described, data exploration in Ethereum along with most relevant public repositories that store source code of smart contracts deployed in Ethereum is studied and an application prototype for downloading, querying, and compiling smart contracts that meet certain selection requirements is developed.

Keywords

Blockchain, Ethereum, Etherscan, web application, console application, smart contracts.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

Capítulo 1 - Introducción

1.1 Motivación

En la cadena de bloques de Ethereum cada transacción que modifique el estado de la cadena requiere una comisión, ya sea al interactuar con una función de un contrato inteligente o al enviar una criptomoneda. Actualmente en la red de Ethereum se realizan en torno a 1 millón de transacciones diarias[1], con una comisión media de entre 3 y 30 dólares en el último año dependiendo de la congestión de la red[2]. Esto implica que, en un entorno de alta congestión se podrían gastar en torno a 30 millones de dólares en comisiones de red.

Una gran parte de todas estas comisiones vienen de la interacción con contratos inteligentes. Un contrato inteligente es un programa que se almacena en la cadena de bloques y que se ejecuta gracias a una máquina virtual integrada en cada nodo de la red. El coste de las comisiones al interactuar con contratos inteligentes viene dado por la cantidad de instrucciones que haga falta ejecutar. Con este propósito, se han desarrollan nuevas técnicas de optimización por parte de la comunidad científica en los últimos años. Fundamentalmente por la disponibilidad de grandes conjuntos de programas reales sobre los que se pueden realizar estudios experimentales. La cadena de bloques de Ethereum es especialmente interesante, pues toda la información de las transacciones realizadas está disponible públicamente.

Sin embargo, aunque el código compilado de los contratos inteligentes desplegados es público, el código fuente no tiene por qué serlo. Algunos

[1] https://etherscan.io/chart/tx

[2] https://etherscan.io/chart/avg-txfee-usd

desarrolladores publican el código fuente de sus contratos en algunos repositorios públicos, como por ejemplo Etherscan [1], pero no es posible hacer búsquedas con condiciones complejas de selección de contratos.

Otro campo relacionado con la investigación en Ethereum es el de la seguridad, ya que al tratar de un entorno descentralizado no hay responsables a los que acudir en casos de brechas de seguridad y resulta de gran importancia tratar de asegurar en la mayor medida de lo posible la seguridad de un contrato inteligente. Por ello, también en esta área resulta fundamental contar con grandes conjuntos de programas reales para poder desarrollar nuevas técnicas que garanticen la seguridad de los contratos.

Por todo ello, la principal motivación de este proyecto es crear una herramienta que sirva de utilidad a estos investigadores, una herramienta que les permita recopilar y buscar contratos que cumplan condiciones complejas sobre diversas características y recuperar su código fuente.

1.2 Objetivos

El objetivo de este Trabajo de Fin de Grado es la implementación de un sistema que permita cargar contratos desplegados en la red principal de Ethereum, almacenarlos, buscar aquellos que cumplan con ciertos criterios de búsqueda complejos, recuperar su código fuente y las opciones de compilación que fueron utilizadas y poder compilarlos con los mismos parámetros o con otros.

De manera más específica, este proyecto tiene el fin de diseñar e implementar dos aplicaciones, una aplicación de consola y una aplicación web y que ambas permitan al usuario:

- Buscar direcciones de contratos desplegados en la red principal según criterios de búsqueda específicos.

- Descargar contratos verificados para almacenar su código fuente y opciones de compilación.
- Consultar todos los contratos previamente cargados y recuperar los contratos según criterios de búsqueda específicos.
- Compilar contratos inteligentes para obtener sus códigos compilados.

1.3 Plan de trabajo

Para conseguir los objetivos se ha elaborado un plan de trabajo con reuniones semanales con los directores del proyecto para exponer ideas y herramientas que se pudiesen ser de utilidad.

En el inicio de este trabajo se plantearon dos alternativas. Por una parte, se planteó la instalación de un nodo cliente de la red de Ethereum para obtener directamente la información en tiempo real de la cadena de bloques de Ethereum. Por otra parte, se planteó la utilización de servicios disponibles de consulta de Ethereum en tiempo real y repositorios públicos de código fuente de contratos.

La instalación de un nodo cliente plantea diversos inconvenientes, fundamentalmente la gran cantidad de contratos que se suben diariamente que se traduce en una cantidad abrumadora de datos, así como la necesidad de tener un hardware con unos requisitos mínimos junto con un software cliente ejecutándose continuamente.

Por estos motivos se descartó la instalación de un nodo cliente para centrarse en repositorios públicos de contratos.

Se realizó una investigación para poder entender y descubrir herramientas que fuesen de potencial ayuda. Las principales herramientas estudiadas son Etherscan [1], Bloxy [2] y Bitquery [3].

El desarrollo de las aplicaciones se ha llevado a cabo de manera secuencial, primero centrando el foco en la funcionalidad y luego aportando una interfaz gráfica adaptada a dicha funcionalidad.

La funcionalidad se ha ido extendiendo y cambiando acorde a los directores de este trabajo ya que aparte de directores ellos también realizan un trabajo de investigación en la materia.

Capítulo 2 - Tecnología blockchain y Smart contracts

La tecnología de cadena de bloques o tecnología *blockchain* fue por primera vez conceptualizada por Satoshi Nakamoto en su artículo llamado "Bitcoin: A Peer-to-Peer Electronic Cash System" [4]. Poco después en 2009 Bitcoin fue creado, siendo esta la primera cadena de bloques en funcionamiento con a día de hoy gran aceptación.

Satoshi Nakamoto creó un sistema en el que los usuarios pueden enviar y recibir transacciones de la criptomoneda *Bitcoin* basado en criptografía y no en confianza, lo que hace posible que funcione sin ningún banco o tercero que gestione cada movimiento financiero. Permite que los pagos en línea se envíen directamente de una parte a otra sin pasar por ninguna institución financiera.

No fue hasta 2015 con la creación de la cadena de bloques de Ethereum, junto con la criptomoneda *Ether*(ETH), que las posibilidades y aplicaciones de la tecnología se multiplicaron. Esto fue gracias a la implementación de una máquina virtual capaz de ejecutar una serie de instrucciones para realizar un amplio conjunto de operaciones, la Ethereum Virtual Machine (EVM), que dota a la cadena de bloques de Ethereum con la capacidad de construir *smart contracts* o contratos inteligentes y aplicaciones descentralizadas dentro de la propia cadena de bloques a través de lenguajes de programación como *Solidity*. [5]

La EVM se ejecuta como una máquina de pila, con una profundidad de 1024 ítems. Cada ítem es una palabra de 256 bits, que se selecciona para utilizar fácilmente con la criptografía de 256 bits. Además, la EVM tiene varias regiones en las que almacena datos:

- Región *memory*: esta es una región volátil de memoria, usada para almacenar información de manera temporal para su uso en ejecución.

- Región *storage*: esta es una región persistente de memoria y cada vez que se actualiza se debe actualizar el estado de toda la cadena de bloques.

Se puede encontrar una especificación formal con todo detalle sobre la EVM en su "*yellowpaper*" [6].

Los contratos inteligentes son programas que se almacenan y ejecutan dentro de la cadena de bloques, pero además son un tipo de cuenta de Ethereum esto quiere decir que poseen un saldo y pueden recibir transacciones. Los contratos inteligentes funcionan de manera autónoma conforme a como están programados, de forma que otros contratos y usuarios pueden interactuar con ellos utilizando las funciones definidas dentro del contrato inteligente.

Y es que, al contrario que Bitcoin, que solo tiene transacciones para el envío de su criptomoneda, Ethereum además tiene transacciones para desplegar contratos inteligentes y transacciones para interactuar con contratos inteligentes. Estas transacciones requieren una tarifa y deben incluirse en un bloque válido.

De igual manera a otros bloques de cadenas, un bloque en Ethereum es una lista de transacciones, estas pueden ser llamadas para interactuar con contratos, que está encadenado al bloque anterior mediante un *hash* obtenido de los datos de ese bloque, de aquí el concepto de cadena de bloques.

El proceso de verificación de bloques se realiza a través de un mecanismo de consenso, una serie de reglas que rigen la manera en que los nodos de la red llegan a un acuerdo sobre el estado de la cadena de bloques y la validez de las transacciones.

El mecanismo de consenso más utilizado en las distintas cadenas de bloques se basa en una "prueba de trabajo". En Ethereum este método se ha usado hasta el 15 de septiembre de 2022, tras este día Ethereum pasó a utilizar un mecanismo de consenso basado en "prueba de participación". Este proceso se denominó "The *merge*" [7].

Ambas tienen la misma finalidad la de ayudar a alcanzar el consenso en la cadena de bloques de forma segura, pero con diferencias notables, la Prueba de Trabajo se basa en una competición entre los nodos validadores, llamados mineros, estos tratan de resolver problemas que requieren una carga computacional muy elevada y el que primero resuelva el problema consigue una recompensa. Para la red, la cadena más larga es la válida ya que es la que más carga computacional había necesitado para crearla. Este es un mecanismo muy eficaz para resistir a los denominados ataques *Sybil* de los que se hablarán un poco más adelante.

El mecanismo de Prueba de Participación ya no utiliza mineros para validar bloques, ahora los nodos validadores son aquellos que tienen una participación económica en Ethereum. Para poder participar como validador, un nodo tiene que depositar 32 ETH en un contrato inteligente, entonces este se convierte en un validador y en responsable de verificar y propagar bloques. El mecanismo de consenso es el encargado de evitar la infiltración de nodos maliciosos en la red. Todos los nodos verifican el comportamiento de los demás. Si se detecta una acción maliciosa en contra de la red por parte de un nodo validador, es

expulsado y perdera sus 32 ETH depositados, equivalentes en el momento de escribir este documento a aproximadamente 110000 euros.

Un ataque *Sybil* [8] es aquel en el que un nodo o grupo de nodos se hace pasar por una cantidad masiva de nodos para poder ganar influencia en las decisiones de la red. Por ejemplo, si un nodo pudiese él solo ampliar la cadena, podría crear nuevos bloques maliciosos que tengan transacciones falsas. Para que este tipo de ataque pueda darse en una cadena de bloques que utilice Prueba de Trabajo, se debería de tener más del 51 % de toda la capacidad de cómputo de la red para poder imponer un bloque fraudulente a todos los demás nodos. Esa cantidad de trabajo requiere una gran cantidad de energía de computación y la energía gastada podría incluso haber superado los beneficios obtenidos en un ataque.

Algo muy similar pasa con la Prueba de Participación, donde el usuario malvado tendría que depositar y exponer el 51% del total de Ether depositado, el equivalente a una verdadera fortuna.

Además de Bitcoin y Ethereum existen otras muchas otras cadenas de bloques. A día de hoy hay cientos de cadenas de bloques públicas, algunas como la cadena de bloques de Bitcoin, no Turing completas, y otras como la cadena de bloques de Ethereum, Turing completas.

Sin entrar demasiado en detalle, la cadena de bloques de Ethereum es Turing completa ya que la EVM es una máquina virtual Turing completa, es decir, puede realizar cualquier computación pues permite ejecutar contratos inteligentes que contienen bucles y condicionales. Aunque normalmente se define de esta manera, realmente la EVM no es del todo Turing completa, es *cuasi*-Turing completa, esto se debe a que

la computación que está máquina virtual puede realizar está limitada por un parámetro de la máquina virtual llamado *gas*.

El gas se puede entender como una comisión que el usuario tiene que pagar para para que los validadores de la red ejecuten las instrucciones de la EVM que forman el código del contrato.

Esta comisión sirve principalmente para añadir seguridad a la red, ya que al tener que pagar una comisión por cada transacción se evitan posibles ataques de denegación de servicios basados en la ejecución de bucles infinitos y posibles ataques para saturar la red con cantidades masivas de información inútil.

2.1 Principales problemas de Ethereum

Definitivamente el sistema de Ethereum ha supuesto una revolución dentro de la tecnología de cadena de bloques, esto ha supuesto un gran auge en la popularidad y usa de la cadena de bloques de Ethereum. Sin embargo, todo el éxito que ha tenido también ha traído una serie de problemas que afectan a las cadenas de bloques que usan este sistema, especialmente la red principal de Ethereum.

El problema principal desde el punto de vista del usuario es que el gas que se tiene que pagar por cada transacción puede ser bastante elevado.

En la figura 2-1 se puede apreciar un gráfico con el coste promedio de transacción en el último año, que alcanzó el 5 de marzo de 2024 un coste de casi 30$.

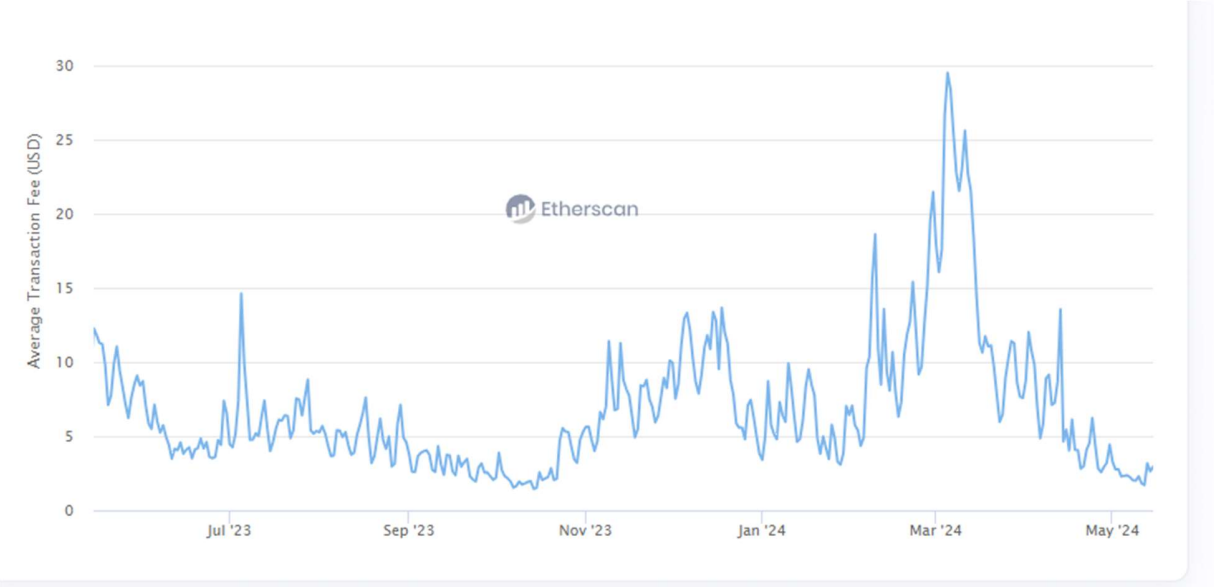

Figura 2-1. Coste promedio de transacción. [3]

Hay que tener en cuenta que este es el coste promedio, si la transacción requiere un esfuerzo computacional superior al promedio este precio podría ser muy superior.

El segundo gran problema que presenta la cadena de bloques de Ethereum es su problema de escalabilidad. Este problema tiene dos aspectos principales: el tamaño de la cadena de bloques y la congestión de la red.

El problema del tamaño de la cadena de bloques de Ethereum surge ya que la red está en constante crecimiento. El problema con una cadena de bloques muy grande es el riesgo de centralización ya que llegados a un tamaño muy grande los usuarios regulares dejarían de validar y solo validarían unos pocos nodos empresariales.

[3] https://etherscan.io/chart/avg-txfee-usd

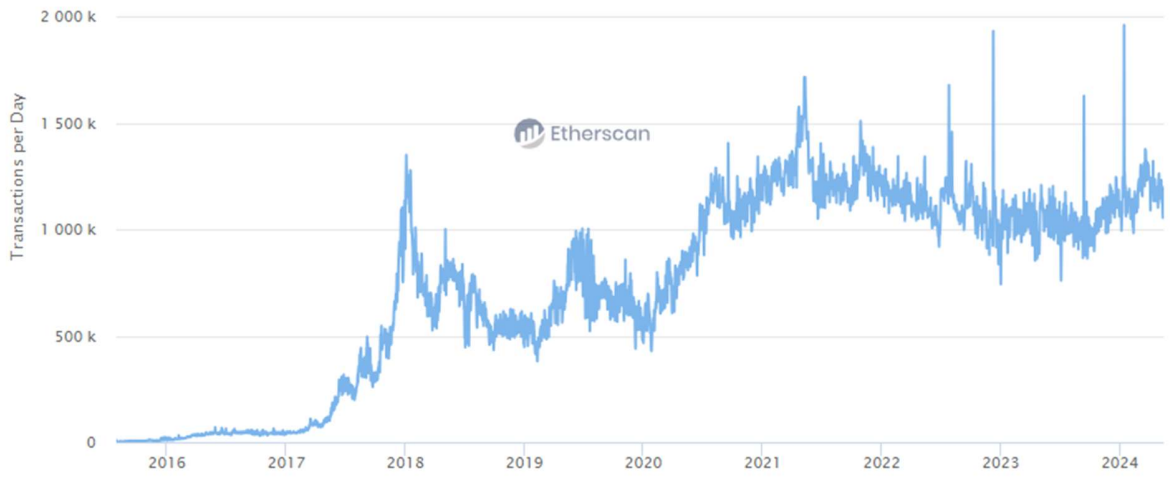

Figura 2-2. Cantidad de transacciones. [4]

El segundo problema se debe a que Ethereum, a día de hoy puede procesar en torno a 15 transacciones por segundo[5], esto da aproximadamente un millón de transacciones al día como muestra la figura 2-2. Esto provoca lentitud y comisiones muy altas para el usuario.

2.2 Layer 2

Los proyectos de capa 2 o más conocidos como "*Layer 2*" surgen como una solución a estos problemas, estos proyectos tratan de reducir las comisiones procesando transacciones fuera de Ethereum, pero aprovechándose de la seguridad de Ethereum.

Una capa 2 es una cadena de bloques separada que extiende el rendimiento de Ethereum.

[4] https://etherscan.io/chart/tx

[5] https://chainspect.app/chain/ethereum

Hay muchos tipos diferentes de capa 2. Cada uno tiene sus propias ventajas y desventajas, así como sus propios modelos de seguridad. Los proyectos de capa 2 liberan de carga transaccional a la capa 1, reduciendo su congestión y haciéndola más escalable.

Los *rollups* se presentan como uno de los principales tipos de soluciones de capa 2, los *rollups* agrupan lotes de transacciones que se comprimen y almacenan fuera de la cadena de bloques. Luego, se adjunta una versión empaquetada de las transacciones en la cadena de bloques principal, que sirve como mecanismo de seguridad para el *rollup*.

La figura 2-3 muestra un ejemplo que combina una solución de capa 2 con un *rollup* para descongestionar Ethereum, las transacciones se ejecutan en la cadena de bloques de la capa 2, se empaquetan y se envían en una sola transacción a la red de Ethereum. De esta manera el coste de la transacción se divide entre todas las transacciones comprimidas, esto implica menores comisiones para el usuario y descongestión para la cadena de bloques de Ethereum.

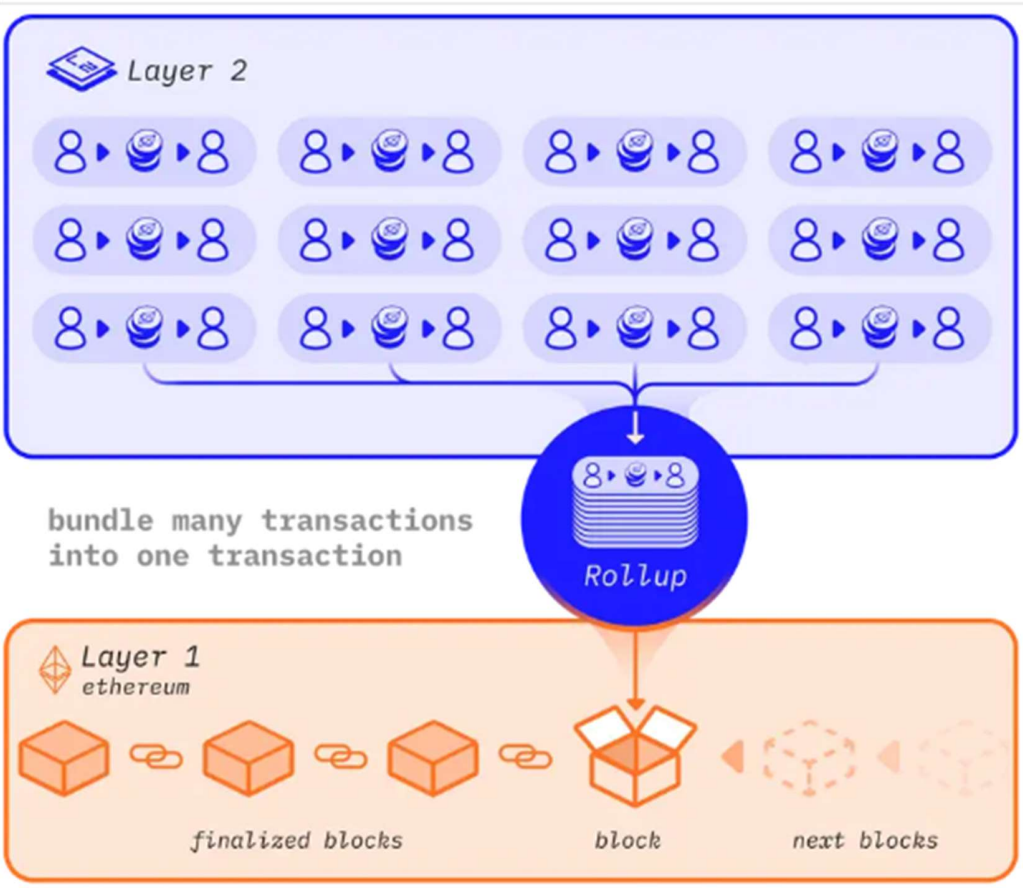

Figura 2-3. Ejemplo de capa 2. [6]

En esencia estas cadenas de bloques de capa 2 se comportan igual que Ethereum, pero con costes de comisiones mucho más baratos y ayudan a descongestionar a Ethereum.

[6] https://ethereum.org/en/layer-2/

2.3 Optimizaciones en el código

A nivel de un desarrollador, aunque en mucha menor medida, también hay soluciones o buenas prácticas que pueden reducir las comisiones producidas al interactuar con el contrato desarrollado.

Como el gas varía en función de la carga computacional que cueste la transacción, la utilización de técnicas de optimización del código de los contratos puede reducir el consumo de gas de las transacciones. Algunas de las optimizaciones más comunes son:

- Minimizar el uso de *storage*, si se puede siempre se debe intentar usar variables locales o *memory* en vez de *storage*.
- Optimizar el uso de bucles en el código.
- Utilizar librerías.

2.4 Optimizaciones en el compilador

El bytecode compilado del contrato inteligente se ejecuta a través de códigos de operación de la EVM, que realizan operaciones estándar de pila como XOR, AND, ADD, SUB, etc. La EVM también implementa varias operaciones de pila específicas de las cadenas de bloques, como ADDRESS, BALANCE, BLOCKHASH, etc. [9]

Cada una de estas operaciones tiene un coste en gas asociado, por ejemplo, la instrucción ADDRESS tiene un gas asociado de 2.

Las optimizaciones a nivel de compilador tratan de reducir el gas, por ejemplo, sustituyendo un *bytecode* por otro que sea equivalente en cuanto a funcionalidad, pero con un consumo de gas más reducido.

Capítulo 3 - Exploración de datos en la red principal de Ethereum

Este proyecto va a centrarse en la cadena de bloques de Ethereum, que es la segunda cadena de bloques con más usuarios solo por detrás de la cadena de bloques de *Bitcoin*. Aunque la información que reside en la cadena de bloques es inmutable y pública para cualquier usuario, la labor de recuperar cierta información de la cadena de bloques puede resultar ser bastante tediosa para el usuario.

Es posible recuperar toda la información de la cadena de bloques ejecutando un nodo a través de un software de gestión de nodos como *Geth* [10]. Aunque luego también sería necesario programar herramientas para poder recuperar adecuadamente la información de las estructuras internas del programa.

Actualmente existen herramientas que permiten al usuario extraer información de la cadena de bloques de Ethereum como los exploradores de bloques. Los exploradores de bloques son herramientas que permiten al usuario navegar por la cadena y ver el contenido de la misma, por ejemplo, ver las transacciones realizadas por una dirección, ver los contenidos de un bloque o el saldo de tokens de una dirección.

En los exploradores de bloques podemos consultar información de contratos inteligentes asociados a una dirección, como el código de byte compilado del contrato, pero el código fuente no se guarda en la cadena de bloques. Al interactuar con un contrato el usuario se expone a un posible código malicioso o una vulnerabilidad explotable que pueda potencialmente hacer perder al usuario parte de su capital.

Para garantizar transparencia, en algunos casos el creador provee el código fuente de su contrato desplegado, de forma que el usuario puede comprobar si efectivamente el contrato desplegado se corresponde con el código fuente proporcionado por el creador mediante un proceso de verificación de código fuente.

Para poder verificar un contrato desplegado en la cadena de bloques de Ethereum, dada la dirección del mismo, un código fuente en teoría asociado al contrato desplegado y los parámetros de compilación, el usuario debe realizar los siguientes pasos, representados gráficamente en la figura 3-1:

1. Compilar el código fuente con los parámetros de compilación, de aquí el usuario obtiene el *bytecode* asociado al contrato inteligente.

2. Obtener el *bytcode* asociado al contrato inteligente ya desplegado en la cadena de bloques.

3. Comparar los dos *bytecodes*, si coinciden entonces ese contrato inteligente desplegado se da por verificado para el código fuente y los parámetros de compilación dados.

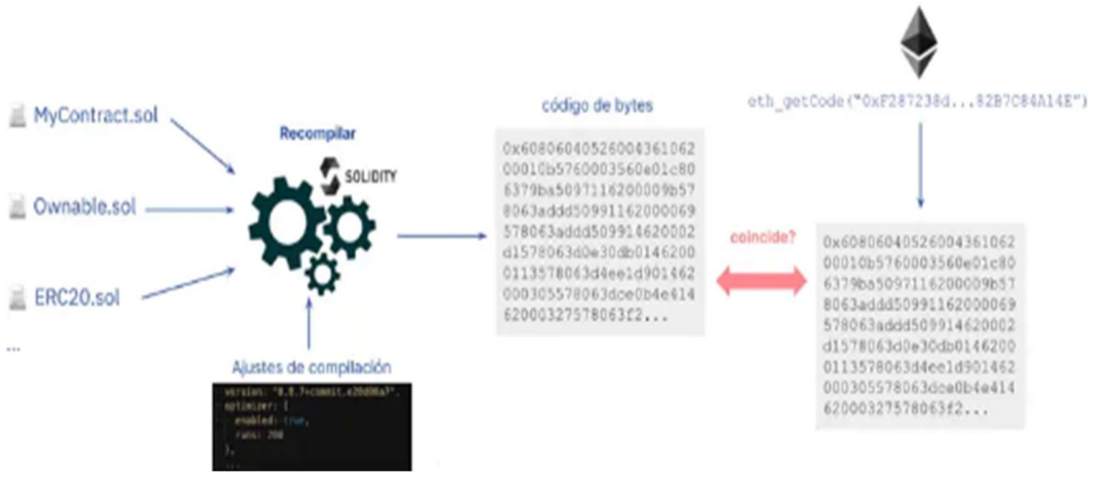

Figura 3-1. Verificación de contratos[7].

Esto realmente es un resumen simplificado del proceso de verificación de contratos, este proceso al completo puede llegar a ser bastante tedioso, por suerte existen herramientas que lo realizan. *Sourcify* [11] es un ejemplo de herramienta para verificación de contratos.

Etherscan [1] es un explorador de bloques que a su vez ofrece una herramienta para verificación de contratos, además de permitir al usuario la descarga de las direcciones de los últimos 5000 contratos verificados. Aunque *Etherscan* sea una herramienta de análisis realmente poderosa y útil para la cadena de bloques de Ethereum, cuando se trata de analizar una cantidad masiva de contratos es muy ineficiente, no tiene ningún sistema que permita, por ejemplo, filtrar los contratos cuyo creador sea una dirección o los contratos compilados en una versión concreta. Esto ralentiza muchísimo el trabajo de un investigador, y es por eso que surge la necesidad de la aplicación desarrollada en este proyecto. Esta aplicación trata de aportar facilidad para aquellos

[7] https://ethereum.org/es/developers/docs/smart-contracts/verifying/

usuarios que quieran almacenar y buscar contratos que cumplan ciertos criterios e incluso compilarlos con la versión del compilador deseada.

3.1 Etherscan

Etherscan es un explorador de bloques de Ethereum, es decir, permite de manera relativamente sencilla explorar toda la cadena de bloques de Ethereum, desde ver cada bloque con todas sus transacciones hasta ver direcciones de usuario o de contratos.

Además, Etherscan actúa como un repositorio de contratos verificados, de hecho, el propio proceso de verificación se puede realizar desde Etherscan.

Aun así, la mayoría de los contratos inteligentes que se despliegan en Ethereum, no son verificados o bien porque no interesa compartir el código fuente o porque directamente no es necesario verificar un contrato para poder interaccionar con él.

Como se puede ver en las figuras 3-2 y 3-3, el porcentaje de contratos verificados es muy pequeño: más del 90% de contratos desplegados no están verificados.

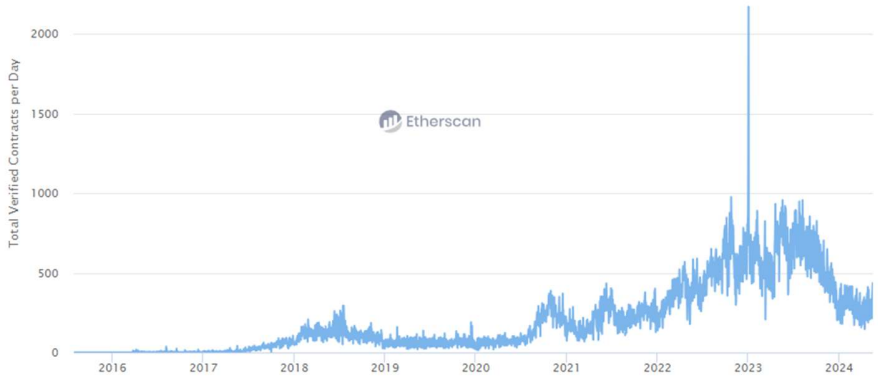

Figura 3-2. Contratos verificados diariamente[8]

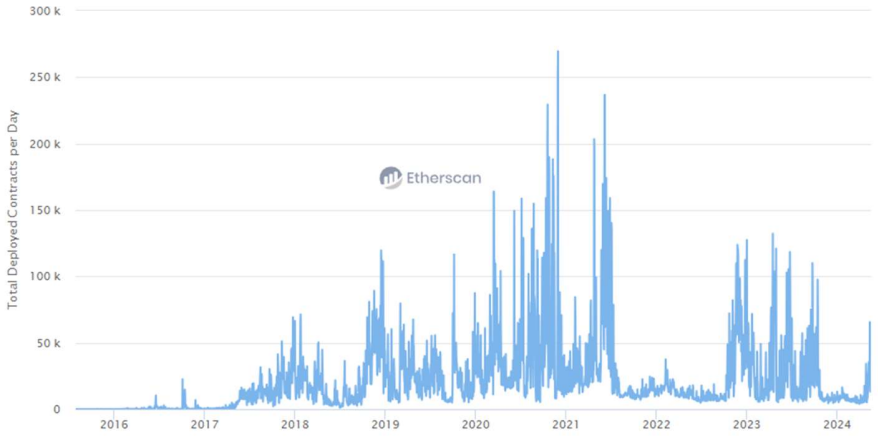

Figura 3-3. Contratos desplegados diariamente[9]

Etherscan ha resultado ser de gran utilidad por su API de uso gratuito, aunque limitado, la cual permite de forma sencilla descargar el código fuente y los parámetros de compilación asociados a la dirección de un contrato inteligente. [12]

[8] https://etherscan.io/chart/verified-contracts

[9] https://etherscan.io/chart/deployed-contracts

3.2 Otros repositorios

Para este proyecto se ha tratado de encontrar otros repositorios de contratos verificados, aunque ninguno ha sido de tanta utilidad como Etherscan.

Bloxy [2] es una herramienta de análisis de cadena de bloques que parece tener una API muy completa. Aunque por desgracia dada la alta demanda, esta herramienta suspendió nuevos registros.

Bitquery [3] es una plataforma con gran cantidad de herramientas en diversas cadenas de bloques, algunas prometedoras sobre todo una herramienta para realizar consultas realmente complejas dentro de la cadena de bloques. El inconveniente principal con esta herramienta es su sistema de uso. Funciona con un sistema de puntos y en la versión gratuita esos puntos se traducen en no más de 10 consultas complejas al mes.

Por último, Bigquery [13] es una herramienta de Google Cloud que permite realizar consultas en sus diferentes bases de datos. Se encontró una base de datos sobre Ethereum que se actualiza en tiempo real, en la figura 3-4 se muestra el diseño de esta base de datos. El problema con esta herramienta es que no almacena los códigos fuente. Por este motivo, en este trabajo se va a integrar Bigquery y su API junto con Etherscan y su API, lo que daría una herramienta capaz de descargar fuentes y hacer consultas complejas en una base de datos que se actualiza en tiempo real.

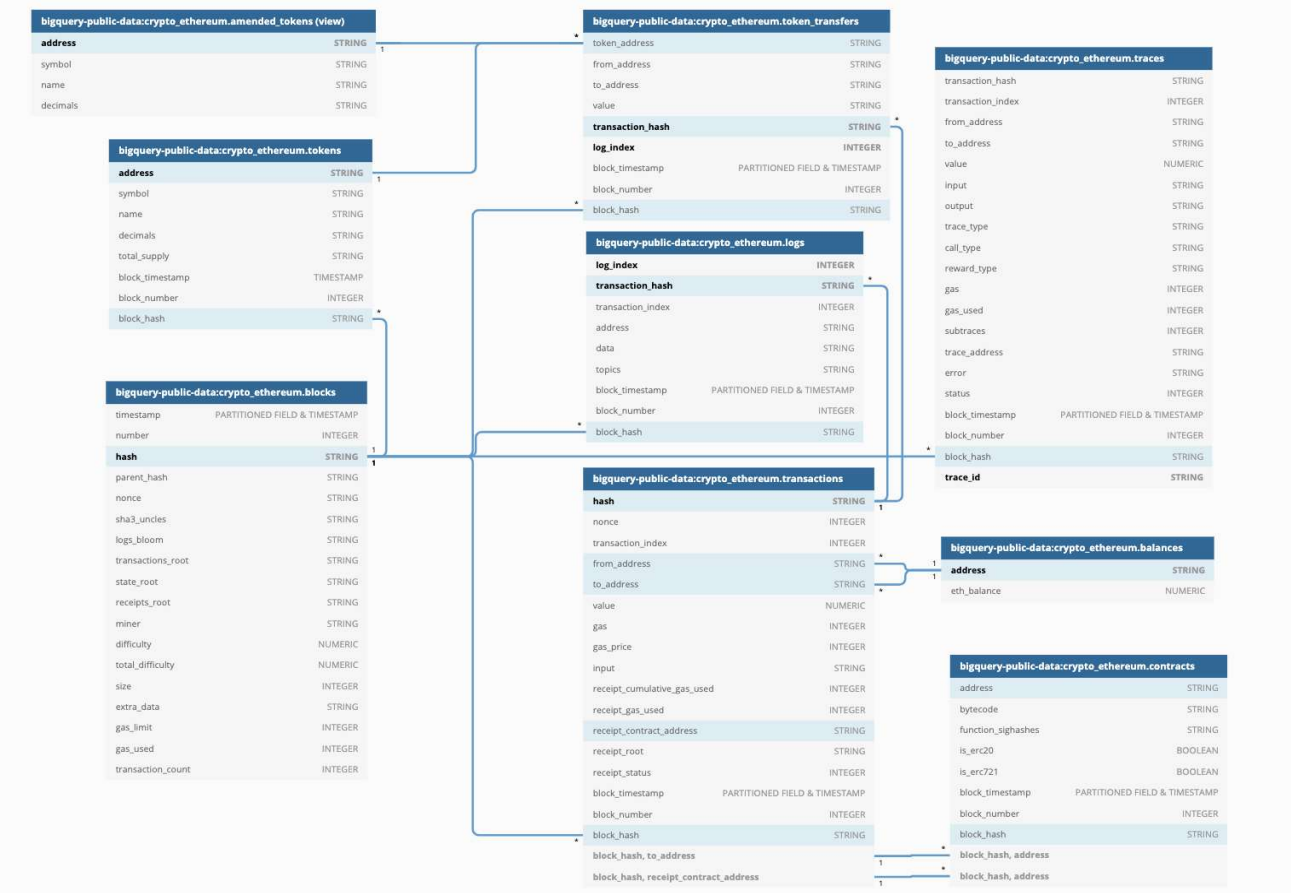

Figura 3-4. Diseño de la base de datos de Bigquery.[10]

[10] https://medium.com/google-cloud/full-relational-diagram-for-ethereum-public-data-on-google-bigquery-2825fdf0fb0b

Capítulo 4 - Diseño de la aplicación

En este capítulo se describe la arquitectura de la aplicación y los diseños de cada una de las partes que la integran.

4.1 Arquitectura de la aplicación

La arquitectura de esta aplicación se puede dividir en distintos módulos o sistemas que se comunican entre ellos creando la aplicación en sí como se muestra en la figura 4-1.

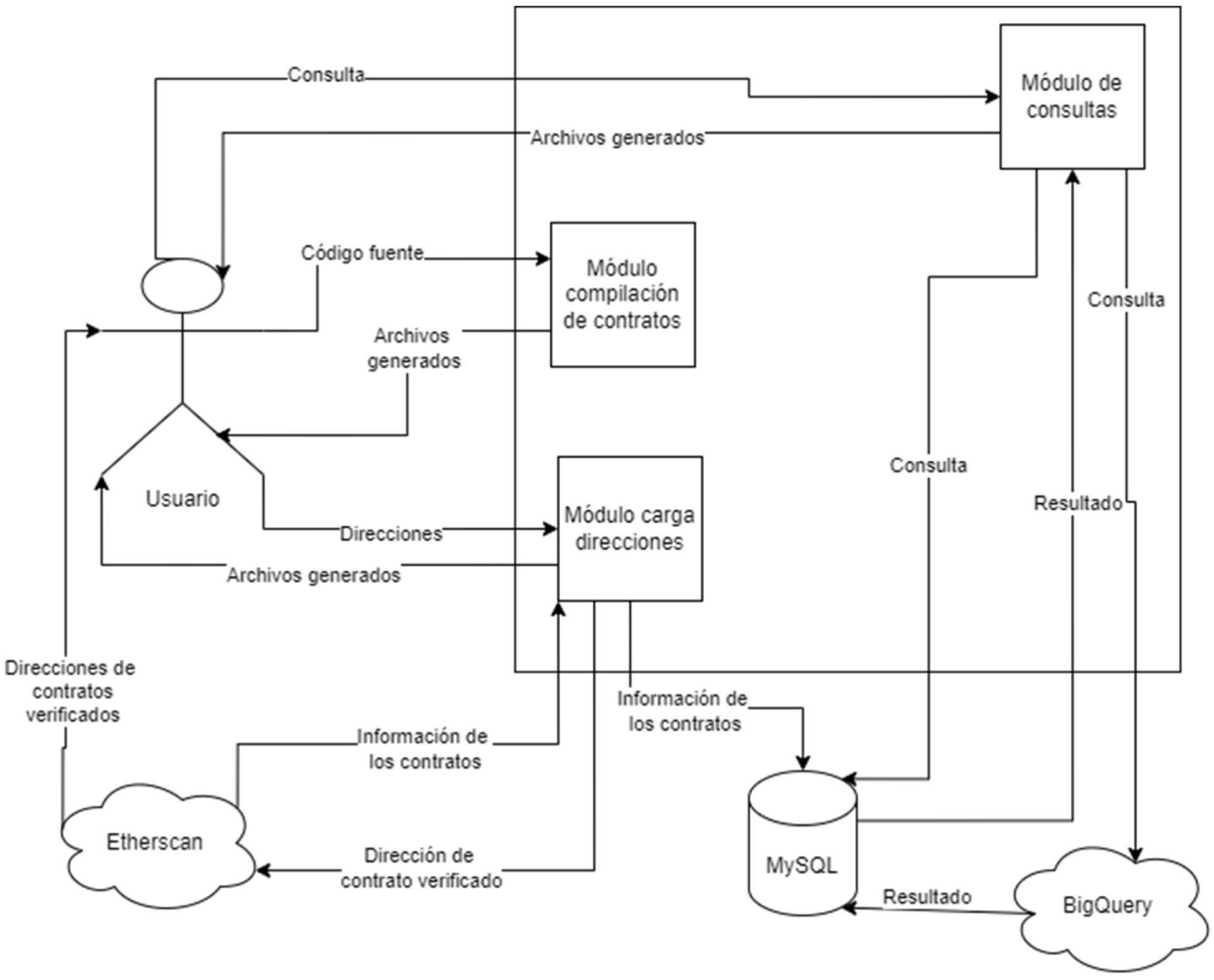

Figura 4-1. Arquitectura de la aplicación

El módulo de carga de direcciones hace uso de los anteriormente explicados repositorios de contratos inteligentes como Etherscan, que a través del usuario alimentan con direcciones de contratos a la aplicación. Mediante dos llamadas a la API proporcionada por Etherscan y un conjunto de direcciones de contratos, se obtiene el código fuente de los contratos inteligentes que estén verificados y los parámetros de compilación. Este módulo además es el encargado de generar y poblar las tablas de la base de datos MySQL según el repositorio fuente del que proceden las direcciones que el usuario carga, guardando los siguientes parámetros de compilación:

- SourceCode: el código fuente del contrato.
- CompilerVersion: la versión del compilador que se usó para compilar el contrato.
- OptimizationUsed: parámetro usado a la hora de la compilación.
- Runs: parámetro usado a la hora de la compilación.
- EVMVersion: la versión de la EVM.
- LicenseType: el tipo de licencia que tiene el contrato.

Por último, se genera el archivo CSV en local, en el directorio *csv_out*, que almacena la misma información guardada en la tabla de la base de datos.

El módulo de consultas está integrado principalmente por una base de datos MySQL que nos permite hacer consultas de todos los contratos guardados en ella. Además de esta base de datos la aplicación cuenta con una conexión al servicio de Bigquery, sobre el que se pueden realizar consultas directamente a su repositorio. Tras ejecutar la consulta sobre la base de datos

local o sobre el repositorio de Bigquery, guarda los resultados en un fichero csv en el directorio local *consultas_out*.

Por último, está el módulo de compilación de contratos, que permite al usuario proporcionar el código fuente de un contrato y la versión del compilador deseada para compilarlo.

La compilación del contrato se hace gracias a dos comandos que se ejecutan tras saber la versión del compilador:

solc-select use versión --always-install

solc -o compilados/contrato --bin –asm --opcodes --overwrite ruta

El primer comando utiliza el módulo de solc-select para cambiar la versión local del compilador de Solidity a la versión indicada por el usuario, gracias a la opción --always-install si el usuario no tiene instalada la versión indicada del compilador, se le instalará.

El segundo comando compila el código fuente ubicado en la ruta indicada por el usuario y los siguientes archivos de salida en la el directorio *compilados*:

- Archivos binarios .bin del contrato en hexadecimal.
- Archivos con el código en ensamblador del contrato.
- Archivos con los *opcodes* del contrato.

4.2 Diseño de datos

En esta sección se describen los componentes de almacenamiento de los datos usados en la aplicación. En la primera subsección se explica qué se guarda en la base de datos local, en la segunda subsección se explica qué datos se usan como entrada para la aplicación y en la última qué datos se guardan en los archivos locales.

4.2.1 Diseño de la base de datos

Hay que destacar que las tablas de la base de datos no tienen relación entre sí. Hay dos tipos de tablas en la base de datos: las tablas de contratos y la tabla de consultas.

Las tablas de contratos siempre tienen los siguientes campos que son obligatorios:

- *compilerversion*: versión con la que se compiló el contrato.
- *optimization*: parámetro de compilación.
- *runs*: parámetro de compilación.
- *evmversion*: versión de la Ethereum Virtual Machine.
- *licensetype*: tipo de licencia usada en la compilación.
- *fuente*: procedencia del contrato.
- *contractcreator*: dirección creadora del contrato.
- *ruta*: ruta en la que se almacena el código fuente del contrato.
- *address*: dirección del contrato.

Además, si el usuario indica que la fuente es Etherscan la tabla asociada siempre tiene los campos siguientes:

- *tx*: hash de la transacción de la creación del contrato.
- *name*: nombre indicado en la creación del contrato.

El resto de tablas de contratos siempre tendrán los campos obligatorios más los adicionales que indique el usuario.

La tabla de consultas es una tabla usada para almacenar todas las consultas que el usuario haga, con el fin de repetir una consulta de manera rápida. Sus campos son:

- *consulta*: consulta SQL.
- *nombre_consulta*: nombre de la consulta indicado por el usuario.
- *fecha*: fecha en la que se realizó la consulta.

4.2.2 Archivos de entrada

Para el módulo de carga de direcciones, el usuario debe proveer un CSV con direcciones de contratos, si son de Etherscan, el CSV debe seguir el mismo formato que se encuentra al descargar el CSV de los últimos 5000 contratos verificados de Etherscan [14].

En el módulo de consultas, si el usuario elige la opción de realizar la consulta directamente al repositorio de Bigquery, debe proveer un JSON asociado a su cuenta de Bigquery.

En el módulo de compilación de contratos, el usuario debe proporcionar el archivo de código fuente de un contrato, un archivo ".sol".

4.2.3 Archivos generados

El uso de la aplicación genera diferentes archivos en local, aunque ya se han mencionado en la descripción de la funcionalidad de cada módulo, aquí se mostrarán todos los posibles archivos generados juntos.

- *contracts*: carpeta en la que se guardan los códigos fuentes de los contratos cargados.
- *csv_out*: carpeta en la que se guardan en formato csv información importante sobre los contratos cargados como el hash de la transacción o la versión del compilador.
- *consultas_out*: carpeta en la que se guardan los resultados de las consultas realizadas.
- *compilados*: carpeta en la que se guardan los resultados del compilado de los contratos.

Cabe destacar que estas carpetas son alimentadas por ambas aplicaciones.

4.3 Diseño de la aplicación web

La aplicación web ha sido desarrollada con el framework Next.js, aunque este framework usa una arquitectura cliente servidor donde el servidor es el encargado de recibir y procesar las peticiones del cliente, la aplicación web está dotada de un segundo servidor web creado con el framework Flask, encargado de asociar funciones a rutas específicas, es decir, es el encargado de procesar las peticiones del cliente.

Esta decisión fue tomada por comodidad ya que el desarrollo de las funcionalidades ha sido programado en Python mientras que el desarrollo de la interfaz web ha sido programado en Typescript, HTML y TailwingCSS.

Todas las herramientas mencionadas se recogen en la sección 1 del capítulo 5.

4.4 Diseño de la interfaz de usuario

Se han diseñado dos interfaces de usuario: un interfaz gráfico basado en web y un interfaz de consola para interactuar con la aplicación desde un terminal de texto.

4.4.1 Diseño de la interfaz web

El desarrollo de la interfaz de usuario se inició con un prototipo creado en *Figma* [15] para tener una visión inicial de la aplicación a la que poco a poco se irían agregando nuevas pantallas a medida que se iban implementando las diferentes funcionalidades.

La aplicación web tiene cinco páginas por las que el usuario puede navegar además de otros elementos importantes que se verán a continuación.

4.4.1.1 NavBar

La barra superior de navegación o *NavBar* es un componente creado para poder navegar a cualquier parte de la aplicación desde cualquier parte de la aplicación ya que está fija siempre en la parte superior de la pantalla como se muestra en la figura 4-2.

Figura 4-2. NavBar

Concretamente el botón *Home* lleva al usuario a la página de inicio, el botón *Cargar csv* lleva al usuario a la página de carga de direcciones que, dependiendo de la fuente, llevará al usuario a la carga de direcciones de Etherscan o a la carga de direcciones de otra fuente, el botón de *Búsqueda en tiempo real* lleva al usuario a la página de búsqueda, el expandible *Fuentes* permite al usuario indicar la fuente para la carga de direcciones, el botón *Compilar* lleva al usuario a la página de compilar, *Etherscan contratos verificados* este botón lleva al usuario a la URL en la que se pueden descargar los últimos 5000 contratos verificados publicados en Etherscan.

4.4.1.2 Página de inicio

La página de inicio es una página sencilla que incluye accesos a los distintos módulos de la aplicación, como se muestra en la figura 4-3.

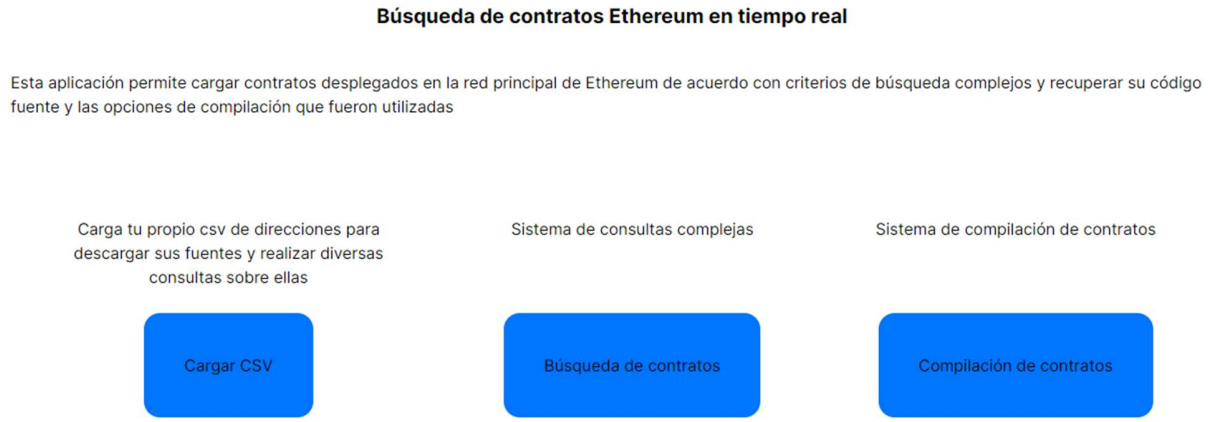

Figura 4-3. Página de inicio.

4.4.1.3 Página de carga de direcciones de Etherscan

En esta página primero se pide al usuario que seleccione el CSV que desee cargar y si el archivo CSV tiene la línea de formato o no. Una vez seleccionado el archivo, se le muestran las primeras líneas del propio CSV en pantalla.

A continuación, se muestra un ejemplo de uso donde se enseñan las pantallas para aportar mayor detalle a la funcionalidad. La figura 4-5, muestra la pantalla sin haber introducido el archivo CSV y la figura 4-6 muestra la pantalla habiendo introducido el archivo CSV, se puede ver la previsualización del archivo en modo tabla. Suponiendo el archivo CSV "contratos_verificados1.csv" mostrado en la figura 4-4, archivo recién descargado de Etherscan que contiene 5 direcciones y tiene la línea del formato tal y como viene tras la descarga.

Figura 4-4. Archivo CSV recién descargado de Etherscan.

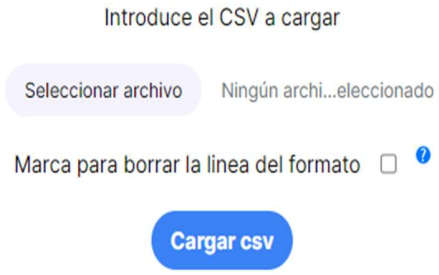

Figura 4-5. Carga de direcciones de Etherscan sin CSV.

Figura 4-6. Carga de direcciones de Etherscan con CSV.

Mientras se realizan las descargas se muestra el botón con un símbolo girando como se muestra en la figura 4-7.

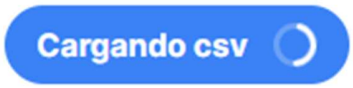

Figura 4-7. Cargando CSV.

En caso de que la operación haya sido exitosa se muestra por pantalla un mensaje de éxito y en caso contrario un mensaje de error.

Por último, el usuario puede ver en su carpeta "*contracts*" los códigos fuentes descargados, figura 4-8, y tanto en su base de datos como en su carpeta "*csv_out*" los parámetros de compilación como los de la figura 4-9, asociados a estos contratos inteligentes en formato CSV.

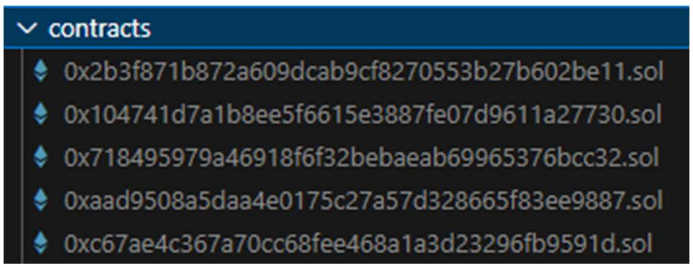

Figura 4-8. Códigos fuente generados.

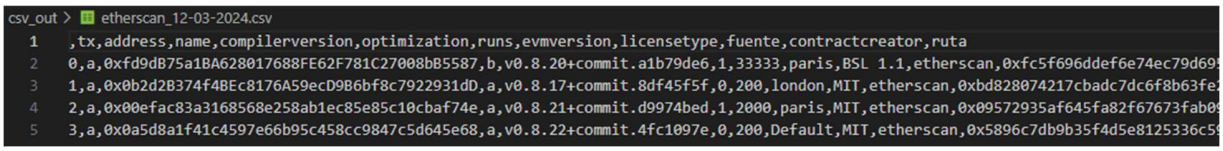

Figura 4-9. Parámetros de compilación.

4.4.1.4 Página de carga de direcciones de otras fuentes

Esta página es muy parecida a la página anterior pero el usuario tiene que indicar el formato que tiene el CSV que va a cargar. La figura 4-10 muestra la pantalla sin el archivo CSV seleccionado y la figura 4-11 la pantalla con el archivo CSV seleccionado donde se puede ver el previsualizado de ese

archivo. Este formato representará las columnas de la tabla en la base de datos. El único campo obligatorio en el formato es el de "*address*".

A continuación, se muestra un ejemplo de uso solo en el que solo se mostrarán las pantallas ya que tanto el CSV de entrada como la carpeta donde se guardan el código fuente sería igual. Suponiendo un CSV muy similar al anterior, "contratos_verificados2" proveniente de Bigquery y con dos campos diferentes, "*Bloque*" y "*Time*", el usuario introduce la fuente con los campos adicionales y el formato del CSV.

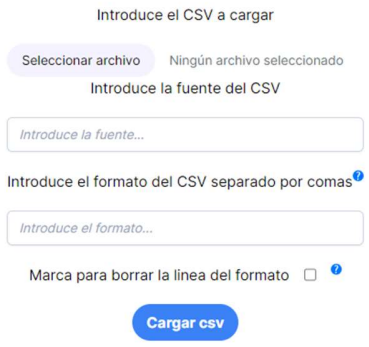

Figura 4-10.Carga de direcciones de otras fuentes sin CSV.

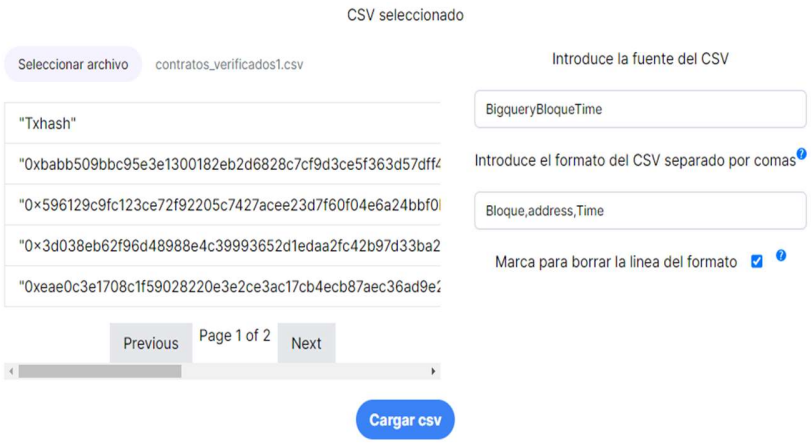

Figura 4-11. Carga de direcciones de otras fuentes con CSV.

Como se ha mencionado anteriormente los archivos de código fuente descargados se guardan igual que antes, pero aquí cabe destacar que se crea una nueva tabla en la base de datos del usuario con el nombre "*contracts_bigquerybloquetime*" y con dos columnas diferentes dadas en el formato del CSV.

4.4.1.5 Página de búsqueda

Esta página permite al usuario realizar consultas a su base de datos local, al repositorio de Bigquery o realizar consultas anteriormente realizadas.

Esta página tiene tres menús expandibles con diferentes opciones: el primero contiene las columnas a las que el usuario puede realizar la consulta, la segunda contiene las todas las tablas que hay en la base de datos y la tercera contiene las consultas anteriores junto a dos opciones para crear una consulta propia y hacer una consulta a Bigquery, respectivamente.

Si la opción seleccionada es la de *consulta propia,* figura 4-12, el usuario tiene que indicar un nombre de consulta y la propia consulta SQL. Si la opción es consultas anteriores, figura 4-13, se muestra otra lista desplegable que contiene las consultas anteriormente realizadas, las que se guardan en la tabla *consultas* de la base de datos. Por último, si se selecciona *Bigquery*, figura 4-14, el usuario debe proveer un token de Bigquery necesario para usar la API de Bigquey.

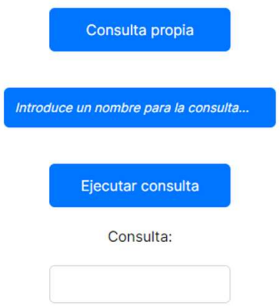

Figura 4-12. Página de búsqueda, consulta propia.

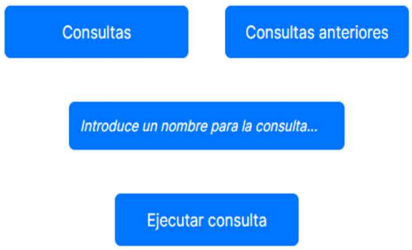

Figura 4-13. Página de búsqueda, consultas anteriores.

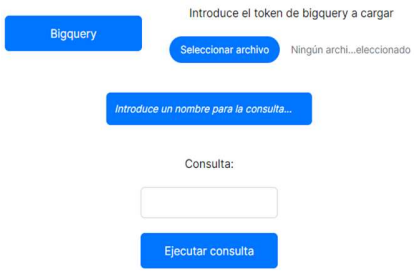

Figura 4-14. Página de búsqueda.

Para detallar bien esta funcionalidad se mostrarán varios ejemplos de uso. En el primer ejemplo de uso, el usuario trata de hacer una consulta ya hecha, para ello indica la columna, la tabla, la consulta y la condición como se muestra en la figura 4-15. También puede ver su consulta antes de ejecutarla para ver si es correcta.

Figura 4-15. Pantalla de consulta.

Una vez se ejecuta la consulta, si se ejecuta con éxito, se genera el archivo CSV correspondiente y se puede previsualizar en la propia pantalla como se muestra en la figura 4-16. Además de un mensaje donde se indica que el CSV se guarda en la carpeta "*consultas_out*" con el nombre proporcionado a la consulta y la fecha de la consulta.

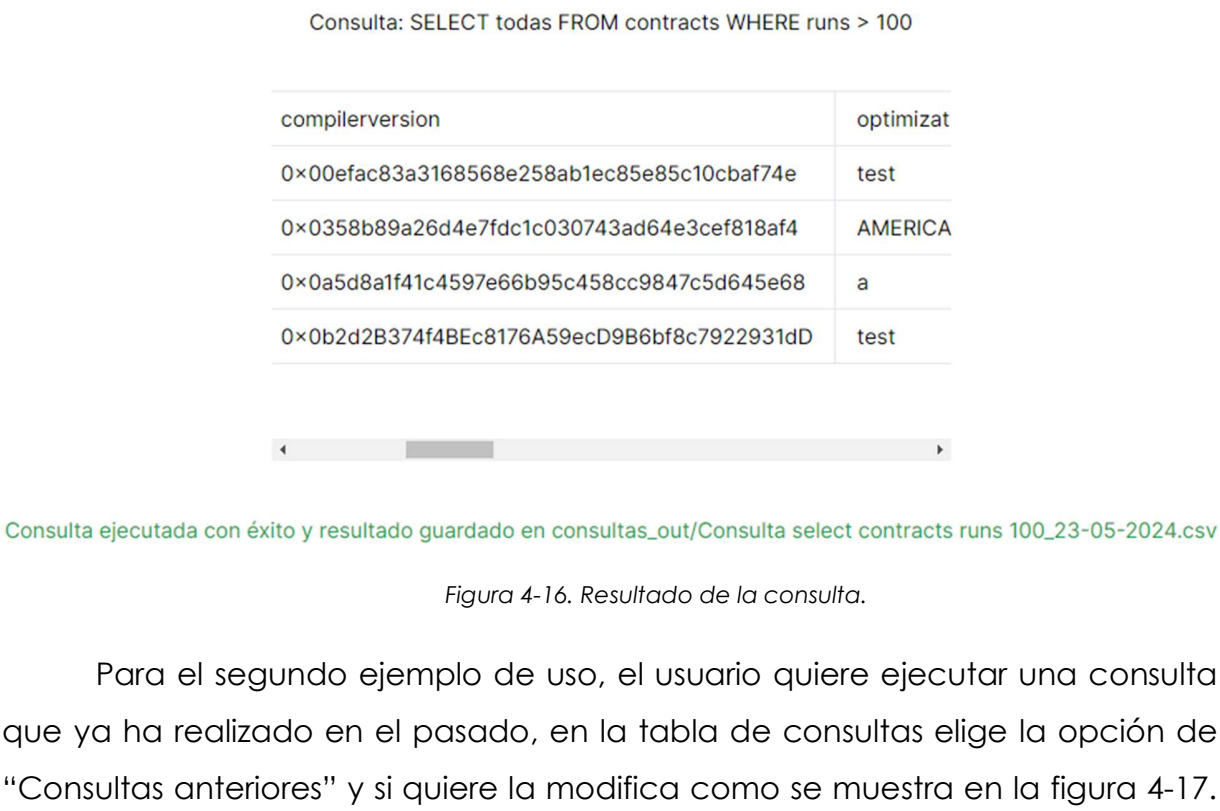

Figura 4-16. Resultado de la consulta.

Para el segundo ejemplo de uso, el usuario quiere ejecutar una consulta que ya ha realizado en el pasado, en la tabla de consultas elige la opción de "Consultas anteriores" y si quiere la modifica como se muestra en la figura 4-17. El resultado se muestra igual que en el ejemplo de uso anterior.

Figura 4-17. Consultas anteriores.

En el último caso de ejemplo el usuario quiere realizar una consulta a través de Bigquery. Para ello en la tabla de consultas selecciona la opción de Bigquery, el usuario debe proporcionar un token de Bigquery, la consulta a realizar y un nombre para ella. En la figura 4-18, una vez seleccionada la opción de Bigquery, se ve el *input* que permite al usuario subir el token de Bigquery, más abajo el campo azul para darle un nombre a la consulta, la consulta en un componente modificables y por último el botón para ejecutar la consulta.

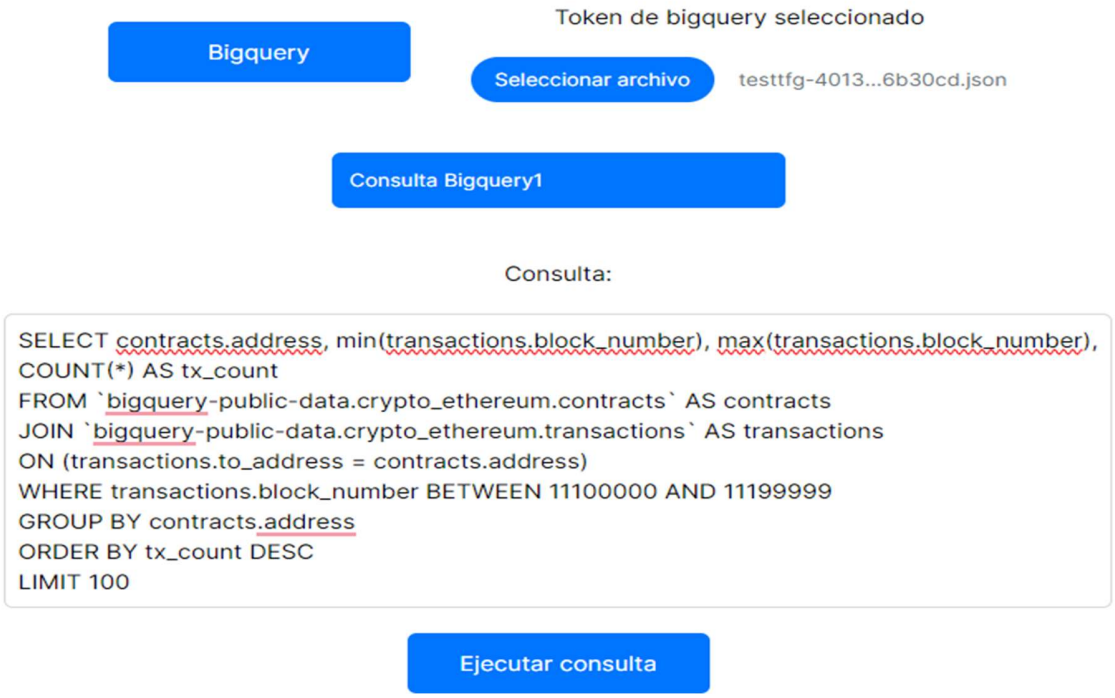

Figura 4-18. Pantalla consulta de Bigquery.

Para construir una consulta de Bigquery el usuario debe utilizar el diseño proporcionado en la figura 3-4, las tablas más interesantes son:

- *Contracts*: que recoge información de contratos como direcciones, código compilado o el número de bloque en el que se desplegó.

- *Transactions*: que recoge información de transacciones como el gas usado en la transacción, el *hash* de la transacción o el número de bloque al que pertenece.
- *Blocks*: que recoge información de los bloques como su *hash* o el *hash* del bloque anterior, la fecha en la que se añadió a la cadena de bloques o la cantidad de transacciones que tiene.

4.4.1.6 Página de compilar

Esta página permite al usuario subir un archivo .sol que debe contener el código fuente de un contrato, en la figura 4-19 se puede ver la pantalla sin el archivo seleccionado. Una vez seleccionado se visualiza en un componente modificable, es decir, que el contrato se puede modificar antes de compilarlo. La aplicación extrae automáticamente las directivas pragma del contrato para que el usuario pueda seleccionar la versión del compilador que se va a utilizar. El usuario también puede elegir una versión del compilador específica. En la figura 4-20 se puede ver la pantalla una vez se selecciona el archivo, en ella aparecen las directivas pragma extraídas del contrato y el componente modificable con el código fuente del contrato.

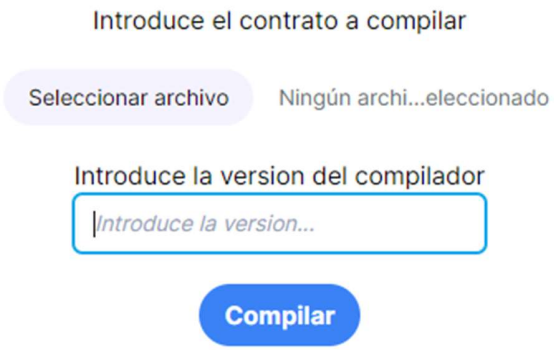

Figura 4-19. Página de compilar contratos sin archivo seleccionado.

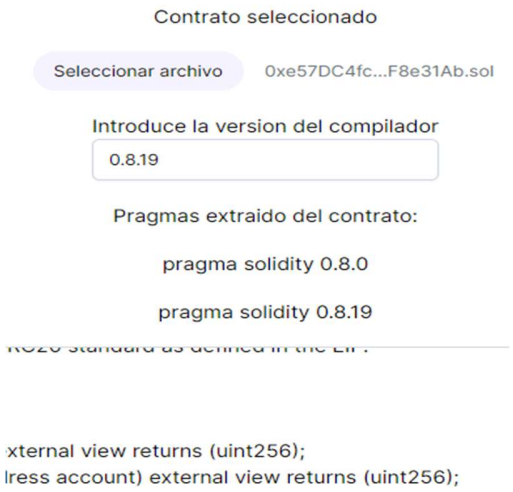

Figura 4-20. Página de compilar contratos con archivo seleccionado.

4.4.2 Diseño de la aplicación de consola

La aplicación de consola fue pensada para un usuario más experto ya que es sacrifica el aspecto visual que es más descriptivo a cambio de un posible uso de la herramienta de manera más eficaz.

Esta aplicación fue diseñada con un menú principal que permite al usuario elegir la funcionalidad deseada y se comunica con él pidiendo el input necesario como rutas a un archivo o la consulta SQL específica.

El menú principal tiene seis opciones como se muestra en la figura 4-21, el usuario introduce el número asociado a la función que quiere usar.

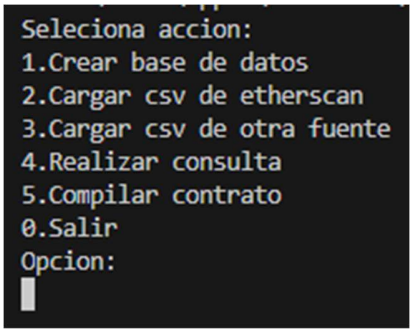

Figura 4-21. Menú consola.

La primera opción crea la base de datos MySQL.

La segunda opción carga direcciones que vienen de Etherscan, también descarga los archivos de código fuente de las direcciones cargadas y guarda toda la información en la tabla *contracts*. Por lo que el usuario debe proporcionar un CSV que tenga el formato de Etherscan. El usuario también debe especificar el repositorio fuente del que vienen las direcciones y si el CSV proporcionado está preprocesado o no como se muestra en la figura 4-22.

Figura 4-22. Opción carga CSV de Etherscan.

La tercera opción permite al usuario cargar direcciones igual que con la segunda opción con la diferencia de que el CSV proporcionado ya no tiene que seguir el formato de Etherscan como se muestra en la figura 4-23. Además, ya no se almacena la información en la tabla *contracts* sino que se guardan en una tabla que representa ese mismo formato, esta tabla se crea con el nombre

del repositorio fuente concatenado con *contracts_*, por ejemplo, si el repositorio del que vienen las direcciones es *Bitquery* la tabla se llamará *contracts_bitquery*.

```
Opcion:
3
Ya existe el subdirectorio o el archivo contracts.
Introduce la fuente de las addresses:
bitquery
Introduce el formato del csv:
tx,address,name
Introduce la ruta del csv:
C:\Users\rpper\Documents\UCM\23-24\TFG\bitquery.csv
¿Contiene el csv el formato?(Si es que si se asume en la primera linea)
n
  tx                                      address name
0  a  0xfd9dB75a1BA628017688FE62F781C27008bB5587    b
1  a  0x0b2d2B374f4BEc8176A59ecD9B6bf8c7922931dD    a
2  a  0x00efac83a3168568e258ab1ec85e85c10cbaf74e    a
3  a  0x0a5d8a1f41c4597e66b95c458cc9847c5d645e68    a
Contract 0xfd9dB75a1BA628017688FE62F781C27008bB5587 OK
Contract 0x0b2d2B374f4BEc8176A59ecD9B6bf8c7922931dD OK
Contract 0x00efac83a3168568e258ab1ec85e85c10cbaf74e OK
Ya existe el subdirectorio o el archivo contracts\0x00efac83a3168568e258ab1ec85e85c10cbaf74e.
Contract 0x0a5d8a1f41c4597e66b95c458cc9847c5d645e68 OK
Ya existe el subdirectorio o el archivo csv_out.
Generado bitquery_15-05-2024.csv
```

Figura 4-23. Opción carga CSV de otras fuentes.

La cuarta opción es la encargada de realizar consultas en la base de datos donde el usuario puede elegir entre una serie de consultas ya hechas en las que tiene que indicar los campos que faltan como las columnas o el nombre de la tabla como se muestra en la figura 4-24. El usuario también puede proporcionar su propia consulta SQL.

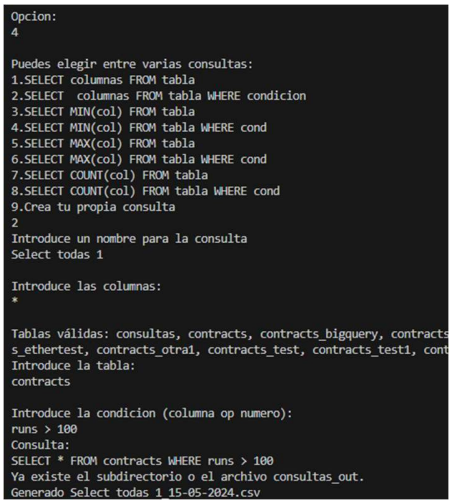

Figura 4-24. Opción consulta.

La quinta opción se encarga de compilar un contrato inteligente, el usuario tiene que introducir la ruta donde tiene el código fuente del contrato inteligente y tiene que introducir la versión del compilador como se muestra en la figura 4-25.

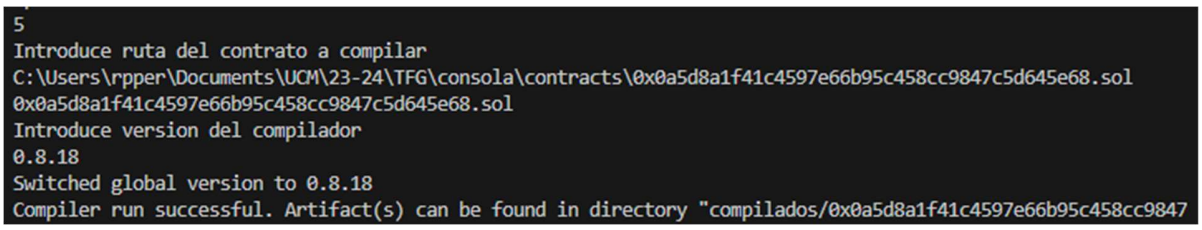

Figura 4-25. Opción compilar.

La última opción simplemente termina la ejecución de la aplicación.

El desarrollo de esta aplicación se realizó en paralelo junto con la aplicación web ya que al final la funcionalidad es prácticamente la misma.

Capítulo 5 - Implementación del sistema

5.1 Herramientas para la implementación

Herramientas necesarias para ejecutar ambas aplicaciones junto con las versiones que se han utilizado para el desarrollo:

- Node.js v19.1.0: herramienta de código abierto y multiplataforma para crear aplicaciones web [16]. Herramienta necesaria para crear la aplicación web junto con el framework de Next.js.
- Next.js: framework para el desarrollo web de código abierto. Herramienta usada para la creación de la aplicación web [17].
- Python 3.11.9: lenguaje usado para crear toda la funcionalidad de la aplicación [18].
- MySQL 8.0.35: sistema de gestión de bases de datos, usado para gestionar las bases de datos locales [19].
- Yarn 1.22.19: gestor de paquetes y de dependencias [20].
- TailwindCSS 3.4.1: framework de CSS de código abierto para diseño de la aplicación web [21].
- Solc-select: herramienta para cambiar he instalar versiones del compilador de Solidity [22].
- Solc: compilador de Solidity usada a la hora de compilar código fuente de contratos inteligentes [23].
- Flask: framework usado para la creación del servidor web que maneja la funcionalidad [24].

- Typescript: lenguaje con el que se ha programado la lógica de la interfaz web [25].
- HTML: lenguaje usado en la creación de la interfaz web.

5.2 Instrucciones de uso

Para ejecutar la aplicación de consola basta con tener instalado Python, MySQL y el módulo solc-select también habría que editar el archivo config.py.

El módulo de solc-select se puede instalar con el comando:

pip3 install solc-select

En este archivo tenemos diferentes campos:

- apikey, se refiere a la clave para acceder a la API de Etherscan.
- dir, se refiere al directorio en el que se guardarán los códigos fuentes de los contratos.
- namedb, se refiere al nombre de la base de datos.
- hostdb, se refiere al nombre o dirección IP del servidor MySQL.
- userdb, el nombre de usuario para autentificarse en el servidor MySQL.
- passdb, la contraseña para autentificarse en el servidor MySQL.
- portdb, el puerto TCP/IP del servidor MySQL.

Para iniciar la aplicación basta con ejecutar el comando:

python main.py

Sustituyendo cualquier otro archivo .py para ejecutar otro módulo de manera individual.

Para ejecutar la aplicación de escritorio es necesario instalar todas las herramientas mencionadas en el punto 5.1, además hay que instalar todos los módulos para ello ejecutar el siguiente comando desde un terminal estando situado en la carpeta web:

yarn install

También tenemos que iniciar tanto el servidor web de Next.js como el de Flask, para el primero tenemos que situarnos en la carpeta web y para el segundo dentro de la carpeta api que está dentro de la carpeta app dentro de la carpeta web:

yarn dev

python script.py

Una vez iniciados los dos servidores web podemos dirigirnos a través de un navegador web a la dirección http://localhost:3000/

5.3 Ubicación y distribución

Todo el código relacionado con el proyecto está publicado en el siguiente repositorio de GitHub: https://github.com/Manudpb/tfg-manuel

Dentro de este repositorio se encuentran dos carpetas:

- consola: aquí se encuentran todos los ficheros asociados con la aplicación de consola, teniendo la aplicación en sí y cada componente de la misma de manera individual.
    - main.py: aplicación de consola completa.

- cargaddresses_ether.py: módulo de carga de direcciones con el formato de Etherscan.
- cargaaddresses_otros.py: módulo de carga de direcciones con cualquier formato.
- consulta.py: módulo de consultas.
- compilar_contratos.py: módulo de compilación de contratos.
- config.py: archivo con los parámetros de configuración.

- web: aquí se encuentran todos los ficheros asociados con la aplicación web.

  La base de la aplicación web se generó con el siguiente comando:

  npx create-next-app@latest web --typescript --eslint

  Y posteriormente marcando las siguientes respuestas a estas preguntas:

  ¿Quieres usar Tailwind CSS? Sí.

  ¿Quieres usar un directorio 'src/'? No.

  ¿Quieres usar el App Router? Sí.

  ¿Quieres usar import alias @? No.

Además, dentro de este proyecto de Next.js se pueden encontrar tres carpetas importantes:

- app: contiene tanto la lógica como la interfaz de usuario de la aplicación.
  - api: contiene el servidor web construido con Flask.

    - El resto de carpetas representan cada una de las páginas de la aplicación.

- components: contiene los componentes customizados.
- context: contexto para manejar las fuentes.

Capítulo 6 - Conclusiones y trabajo futuro

El objetivo principal de este Trabajo de Fin de Grado surge de la necesidad de disponer de una herramienta de descarga de código fuente de contratos mediante consultas de selección complejas para realizar estudios sobre contratos desplegados en la red principal de Ethereum.

Concretamente se han desarrollado dos aplicaciones, una aplicación de consola y una aplicación con interfaz de usuario web que permiten realizar búsqueda de direcciones de contratos desplegados utilizando consultas complejas en Bigquery, así como descargar el código fuente de contratos verificados en el repositorio proporcionado por Etherscan. Además, estas aplicaciones permiten realizar consultas sobre los contratos según criterios de búsqueda específicos. Las consultas realizadas hasta el momento se pueden almacenar en la base de datos local de forma que se pueden repetir sobre otros conjuntos de contratos. Por último, se permite la compilación de los contratos descargados con distintas configuraciones de compilación.

Estas aplicaciones han sido desarrolladas e implementadas siguiendo las especificaciones dadas por los directores de este Trabajo de Fin de Grado. Aplicación web como la aplicación de consola han sido implementadas con toda la funcionalidad prevista, por lo que se puede decir que los objetivos de este Trabajo de Fin de Grado se han cumplido.

Aunque los objetivos se han cumplido aún hay mucho trabajo que se puede hacer para mejorar este proyecto:

- Un aspecto de mejora que se investigó fue el de automatizar la carga de direcciones desde Etherscan, mejora que tuvo que ser descartada por el hecho de la existencia de un *captcha* que exige la descarga manual. Se puede explorar la posibilidad de utilizar

algún servicio de pago para la descarga de direcciones y códigos fuente de contratos. Tanto Etherscan como Bigquery tienen versiones de pago con menos limitaciones y más funcionalidades.

- Otro aspecto a mejorar sería el de migrar ambas aplicaciones a un servidor para no tener que ejecutarlas en local. Este trabajo se ha centrado en implementar la funcionalidad de la aplicación, pero se puede adaptar sin grandes cambios para instalarla en un servidor. Para hacer dicha adaptación podría ser necesario añadir un módulo de control de acceso de usuarios.
- Siguiendo una de las propiedades más importantes del paradigma de las cadenas de bloques, la descentralización, un trabajo a futuro podría ser el de adaptar las aplicaciones a un sistema de archivos descentralizado como *IPFS* [26].

A partir de aquí las sugerencias de extensión del trabajo deberían surgir del propio uso de más clientes junto con el propio avance de la tecnología.

Es algo indiscutible que el ecosistema de criptomonedas está en auge, cada día hay más conocedores de este ecosistema y usuarios que quieren formar parte de este ecosistema, pero también es verdad que las barreras de entrada son grandes para los usuarios. Por este motivo es importante contribuir a una constante mejora de esta tecnología, y para ello es necesario la utilización de herramientas como la desarrollada en este Trabajo de Fin de Grado.

Introduction

Motivation

Within the Ethereum blockchain, every transaction that modifies the blockchain state requires a fee, whether it interacts with a smart contract function or when sending a cryptocurrency. Currently, about 1 million[11] transactions are executed within the Ethereum network daily, with a fee between 3 and 30 dollars during the last year, depending on the network congestion[12].

A big part of all these fees comes from the interactions with smart contracts. A smart contract is a program stored in the blockchain and is executed thanks to a virtual machine integrated within each node of the network. The cost of these fees when interacting with smart contracts comes from the amount of instructions needed to be executed. With this goal in mind, new optimization techniques have been developed by the scientific community in the past years. This was fundamentally possible due to the availability of large sets of programs that can be used to carry out experimental studies. The Ethereum blockchain is especially interesting as all the information on the transactions executed is available to the public.

On the other hand, even though the compiled code of the deployed smart contracts is public, the source code does not necessarily have to be so. Some developers publish the source code of their contracts in certain public

[11] https://etherscan.io/chart/tx

[12] https://etherscan.io/chart/avg-txfee-usd

repositories such as Etherscan [1], but it is not possible to do searches with complex select conditions for contracts.

Another field related to research on Ethereum is security, as when working with a decentralized environment there are not any responsible parties to go to when security breaches occur. As such, it is of extreme importance to try to ensure the security of smart contracts as much as possible. Consequently, it is also fundamental to have large sets of real programs to develop new techniques that guarantee the security of the smart contracts.

Therefore, the main motivation behind this project is to create a tool that is useful for said researchers, a tool that allows them to gather and search contracts that match the complex requirements of different characteristics and retrieve their source code.

Goals

The objective of this thesis is the implementation of a system that allows loading deployed contracts displayed on the Ethereum main net, storing them, searching those that meet specific complex requirements, retrieving their source code and compilation options that were used, and compiling them with the same or different parameters.

More specifically, this project aims to design and implement two applications, a console and a web application that allow the user to:

- Search contract addresses deployed on the Ethereum main net according to specific search criteria.
- Download verified contracts to store their source code and compilation options.

- Query all the previously loaded contracts and retrieve their contracts according to specific search criteria.
- Compile smart contacts to obtain their compiled bytecode.

Work plan

To achieve the objectives, a work plan, consisting of weekly meetings with the project leaders to present ideas and tools that might be useful, has been developed.

At the start of this thesis, two alternatives were proposed. On one hand, the installation of an Ethereum client node was suggested to directly obtain the information of the Ethereum blockchain in real-time. On the other hand, using the available real-time Ethereum searching services and public source code repositories of contracts was considered.

The installation of a client node presents various drawbacks, the main one being that the large quantity of smart contracts that are uploaded daily translates into an overwhelming amount of data, as well as the need to have hardware with specific requirements and client software constantly being executed.

Because of this, the idea of the installation of a client node was discarded to focus on the public source code repositories of contracts approach.

An investigation was carried out to understand and discover tools that could help. The main tools studied are Etherscan [1], Bloxy [2] and Bitquery [3].

The development of the applications has been carried out sequentially, firstly focusing on functionality and then providing a graphic interface to said functionality.

The functionality has been expanding and changing according to the directors of this thesis as a part of being the directors, they are also doing a research project on the matter.

Conclusions and future work

The main goal of this thesis comes from the need for a tool that allows one to download the source code of smart contracts that match complex selection criteria to research the smart contracts deployed on the Ethereum main net.

More specifically, two applications have been developed, a console one and one with a web user interface that allows for searching the addresses of deployed contracts using complex searches in Biquery as well as downloading the source code of verified contracts in the repository given by Etherscan. Furthermore, these applications allow to do queries about smart contracts according to complex and specific search criteria. All the queries done until now can be stored in the local database, making it possible to repeat them on other smart contracts. Lastly, it allows the compilation of downloaded contracts with different compilation configurations.

These applications have been developed and implemented following the specifications given by the thesis directors. Both the web application and the console application have been implemented with all the planned functionality and therefore can be said that the objectives of this thesis have been met.

Even though the objectives of the thesis have been completed, there still is a lot of work that can be put into this project to make it better:

- An aspect that was studied and can be improved was the automatization of the loading of addresses from Etherscan, an improvement that had to be discarded due to the existence of a captcha that requires a manual download. The possibility of using paid services for the download of addresses and contract source codes can be explored. Both Etherscan and Bigquery have paid versions with fewer limitations and more functionalities.

- Another aspect to improve would be to migrate both applications to a server to avoid executing them locally. This project has focused in implementing the functionality of the application but can be adapted to install it on a server without needing major changes. To do said adaptation, it could be necessary to add a module to control user access.
- Following one of the most important properties of the paradigm of blockchain, decentralization, a future project could be to adapt the applications to a decentralized archive system like *IPFS* [24].

From here on, the suggestions for the expansion of the thesis should come from the use of the clients and the advancements of the technology itself.

It is indisputable that the cryptocurrency ecosystem is booming, every day more and more connoisseurs of the ecosystem and users want to be a part of it, but it is also true that there are high entry barriers for them. Because of this, it is important to contribute to a constant improvement of this technology and for that, the utilization of tools like the one developed in this thesis is essential.

Bibliografía

[1] Etherscan, Explorador de bloques de Ethereum, «https://etherscan.io/».

[2] Bloxy, «https://bloxy.info/».

[3] Bitquery, «https://bitquery.io/».

[4] S. Nakamoto, «https://bitcoin.org/bitcoin.pdf,» Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[5] Solidity, «https://soliditylang.org/».

[6] Ethereum Yellow Paper, «https://ethereum.github.io/yellowpaper/paper.pdf».

[7] Ethereum foundation, «https://ethereum.org/es/roadmap/merge/,» La fusión, The merge..

[8] Ataque Sybil, «https://en.wikipedia.org/wiki/Sybil_attack».

[9] Ethereum.org, «https://ethereum.org/es/developers/docs/evm/,» Máquina virtual de Ethereum.

[10] GETH, «https://geth.ethereum.org/».

[11] Sourcify, «https://sourcify.dev/#/verifier,» Herramienta web de verificación de contratos.

[12] EtherscanAPI, «https://docs.etherscan.io/api-endpoints/contracts,»

Etherscan API contracts endpoint.

[13] Bigquery, «https://cloud.google.com/bigquery?hl=es».

[14] Etherscan CSV 5000 contratos verificados, «https://etherscan.io/exportData?type=open-source-contract-codes».

[15] Figma, «https://www.figma.com/».

[16] Node.js, «https://nodejs.org/en».

[17] Next.js, «https://nextjs.org/».

[18] Python, «https://www.python.org/».

[19] MySQL, «https://www.mysql.com/».

[20] Yarn, «https://yarnpkg.com/».

[21] TailwindCSS, «https://tailwindcss.com/».

[22] Solc-select, «https://github.com/crytic/solc-select».

[23] Solc, «https://binaries.soliditylang.org/».

[24] Flask, «https://flask.palletsprojects.com/en/3.0.x/».

[25] Typescript, «https://www.typescriptlang.org/».

[26] IPFS, «https://ipfs.tech/,» Sistema de archivos descentralizados.