



UNIVERSIDAD
DE
CÓRDOBA

Instituto de Estudios de Posgrado
Universidad de Córdoba

Máster Universitario en Inteligencia Computacional e
Internet de las Cosas

CIBERSEGURIDAD

PROYECTO:

BLOCKCHAIN EN PYTHON. COMPARATIVA ENTRE
HYPERLEDGER BY BESU Y ETHEREUM SOLIDITY PARA LA
IMPLEMENTACIÓN DE UNA RED BLOCKCHAIN PRIVADA

Autora:

Alba Márquez-Rodríguez

Profesores:

Ezequiel Herruzo Gómez

Juan Carlos Gámez

Córdoba, Febrero 2024

Índice

1. Introducción	4
2. Fundamentos de Ciberseguridad	5
2.1. Amenazas Comunes en Entornos Digitales	5
2.2. Técnicas de Ataque	5
2.3. Medidas de Seguridad	6
3. <i>Blockchain</i> y Seguridad	7
3.1. Definición y Principios Básicos de Blockchain	7
3.1.1. <i>Distributed ledger</i>	8
3.1.2. Almacenamiento Inmutable	8
3.1.3. Algoritmo de Consenso	9
3.2. Mejora de la Seguridad mediante Blockchain	9
3.3. Casos de Uso en Ciberseguridad	10
4. <i>Ethereum Solidity</i>	11
4.1. Contratos Inteligentes	11
4.2. Ethereum Virtual Machine (EVM)	11
4.3. Diferenciadores Clave de Ethereum	11
4.3.1. Versatilidad:	11
4.3.2. Programabilidad:	11
4.3.3. Turing Completeness:	11
4.3.4. Estandarización de Tokens:	12
4.3.5. Crecimiento de la Comunidad:	12
4.4. Solidity y Desarrollo de Contratos Inteligentes	12
5. <i>Hyperledger by Besu</i>	13
5.1. Hyperledger Besu: Un Enfoque en la Implementación Empresarial	13
5.1.1. Características Clave de Hyperledger Besu	13
5.1.2. Proof of Work (PoW) en Hyperledger Besu	13
5.1.3. Proof of Authority (PoA) en Hyperledger Besu	13
5.2. Implementación Práctica con Hyperledger Besu	14
6. Implementación de Blockchain con Python	15
6.1. Clase Block	15
6.2. Clase Blockchain	17
6.3. Fichero <code>gui_server.py</code>	19
7. Implementación de Ethereum Solidity	27
8. Caso de Uso	32
8.1. Cadenas de Suministro Sostenibles	32
9. Comparación	35

10. Retos y Consideraciones	36
10.1. Retos en <i>Blockchain</i>	36
10.1.1. Escalabilidad y Rendimiento	36
10.1.2. Interoperabilidad	36
10.1.3. Seguridad y Privacidad	36
10.1.4. Adopción y Conciencia	36
10.2. Retos en la Implementación de Blockchain en python	36
10.3. Retos en la Implementación de Blockchain con Ethereum Solidity	37
11. Conclusiones	39

Índice de figuras

1.	Representación Imaginaria de Blockchain por Microsoft Copilot	4
2.	Proceso del <i>Blockchain</i>	7
3.	Interacción con ChatGPT para recibir ayuda en la implementación de la interfaz gráfica	20
4.	Servidor Blockchain	22
5.	Cliente Blockchain	23
6.	Lista de cuentas de Ganache	27
7.	Datos de una de las cuentas de Ganache	28
8.	Blockchain con bloques en Ganache	31
9.	Ejemplo de mejoras con la aplicación de <i>Blockchain</i> a cadenas de suministros	33

1. Introducción

En la era digital actual, la ciberseguridad se ha convertido en un aspecto fundamental para salvaguardar la integridad, confidencialidad y disponibilidad de la información en entornos tecnológicos. El incremento de amenazas cada vez más sofisticadas y los constantes desafíos que enfrentan las organizaciones para proteger sus activos digitales resaltan la necesidad de enfoques innovadores y robustos [1].

Este trabajo se centra en explorar la convergencia de dos campos cruciales: la ciberseguridad y la tecnología *blockchain*. La ciberseguridad, como disciplina, se ocupa de salvaguardar sistemas, redes y datos contra amenazas maliciosas [2]. Por otro lado, la tecnología *blockchain*, popularizada inicialmente por ser la infraestructura detrás de las criptomonedas, ofrece una nueva perspectiva en cuanto a la seguridad, gracias a su naturaleza descentralizada e inmutable [3].

La implementación de *blockchain* en entornos de ciberseguridad presenta un potencial significativo para mejorar la resistencia a ataques, la transparencia y la confianza en los sistemas digitales [4]. Este trabajo busca explorar cómo la aplicación práctica de *blockchain*, utilizando el lenguaje de programación Python, puede fortalecer las medidas de seguridad existentes.

A lo largo de este documento, se examinarán los fundamentos de la ciberseguridad, los principios básicos de la tecnología *blockchain* y se presentará una implementación práctica utilizando Python. Además, se analizará un caso de uso específico en el que la integración de *blockchain* demuestra ser especialmente beneficiosa en el ámbito de la ciberseguridad.

La comprensión de cómo la descentralización, la criptografía y la inmutabilidad inherentes a *blockchain* pueden mitigar riesgos y fortalecer la seguridad es esencial en el panorama actual de amenazas digitales. Este trabajo ayudará a la comprensión de la aplicación de *blockchain* en ciberseguridad, proporcionando una perspectiva práctica y herramientas concretas implementadas en Python.

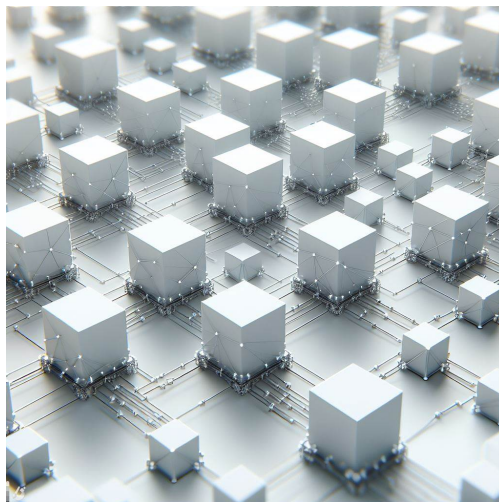


Figura 1: Representación Imaginaria de Blockchain por Microsoft Copilot

2. Fundamentos de Ciberseguridad

La ciberseguridad es una disciplina muy importante en el mundo digital actual, donde la protección de la información sensible y la prevención de amenazas maliciosas son de gran importancia. Comprender los conceptos básicos de la ciberseguridad es esencial para abordar los desafíos constantes que enfrentan los sistemas y redes informáticas [5]. A continuación, se presentan algunos aspectos fundamentales de la ciberseguridad:

Los tipos de amenazas se pueden agrupar en:

- **Robo de información**
- **Pérdida y manipulación de datos**
- **Robo de identidad**
- **Interrupción de servicio**

2.1. Amenazas Comunes en Entornos Digitales

Las amenazas cibernéticas abarcan una amplia gama de actividades maliciosas diseñadas para comprometer la seguridad de sistemas y datos [6]. Entre las amenazas comunes se incluyen:

- **Malware:** Software malicioso diseñado para dañar o comprometer sistemas y datos.
- **Phishing:** Técnicas para engañar a usuarios y obtener información confidencial, como contraseñas o datos financieros.
- **Ataques de Denegación de Servicio (DDoS):** Intentos de saturar un sistema, servicio o red para que no esté disponible para usuarios legítimos.
- **Ingeniería Social:** Manipulación psicológica para obtener información confidencial de individuos o acceso no autorizado a sistemas.
- **Fugas de Datos:** Divulgación no autorizada de información sensible.

2.2. Técnicas de Ataque

Los atacantes emplean diversas técnicas para comprometer la seguridad [7]. Algunas de las técnicas de ataque más comunes incluyen:

- **Inyección de Código:** Introducción de código malicioso en sistemas para ejecutar acciones no autorizadas.
- **Intercepción de Datos:** Captura no autorizada de información durante su transmisión. Ya sea simplemente la intercepción de estos datos o la modificación.
- **Suplantación de Identidad:** Pretender ser otra entidad para obtener privilegios no autorizados.
- **Exploración de Vulnerabilidades:** Identificación de debilidades en sistemas para explotarlas.

La comprensión de estas amenazas y técnicas es crucial para desarrollar estrategias efectivas de ciberseguridad y proteger activos digitales de manera adecuada.

2.3. Medidas de Seguridad

Para mitigar las amenazas cibernéticas, se deben implementar medidas de seguridad adecuadas [8]. Estas pueden incluir:

- **Sistemas IDS / IPS:** Sistemas de monitorización, detección y/o prevención de accesos no permitidos en una red.
- **Honeypot:** Equipos aparentemente vulnerables diseñados para atraer y detectar a los atacantes, protegiendo los sistemas realmente críticos.
- **SIEM:** Sistemas de correlación de eventos y generación de alertas de seguridad.
- **Firewalls:** Dispositivos que controlan el tráfico de red y aplican reglas de seguridad.
- **Antivirus y Antimalware:** Software diseñado para detectar y eliminar software malicioso.
- **Cifrado de Datos:** Protección de la información mediante algoritmos criptográficos.
- **Autenticación Multifactor:** Uso de múltiples métodos para verificar la identidad de usuarios.

La combinación de estas medidas contribuye a fortalecer la postura de seguridad de un sistema o red.

En el próximo apartado, se explorará cómo la tecnología *blockchain* puede proporcionar una capa adicional de seguridad en este contexto.

3. *Blockchain* y Seguridad

La tecnología *blockchain* ha emergido como un componente revolucionario en la era digital, inicialmente conocida por respaldar las criptomonedas como Bitcoin y Ethereum. Sin embargo, su impacto se extiende más allá de este tipo de transacciones e intereses financieros, ofreciendo un enfoque innovador para abordar desafíos de seguridad en diversos dominios. En esta sección, se explorarán los fundamentos de *blockchain* y cómo su implementación puede fortalecer la seguridad en entornos cibernéticos.

3.1. Definición y Principios Básicos de Blockchain

Blockchain se describe como una base de datos utilizada como almacenamiento para una red descentralizada [9]. Se ve comúnmente en su uso popularizado en criptomonedas como Bitcoin, Ethereum, Litecoin y Dogecoin. Sin embargo, Blockchain no se limita al ámbito financiero, ya que puede expandirse para abarcar otros tipos de sistemas y aplicaciones, constituyendo una red descentralizada [10]. La criptografía asimétrica y los algoritmos de consenso distribuido forman parte de los sistemas dentro de Blockchain, proporcionando seguridad al usuario y consistencia en el libro mayor [11]. En resumen, Blockchain es una base de datos descentralizada e inmutable que facilita su red de cadenas con nodos participantes a través de un esquema de votación.

Como se muestra en la Figura 2, que ilustra el proceso general de Blockchain. El proceso comienza con la solicitud de una transacción desde un nodo, que se empaquetaría en un bloque. Luego, transmitiría el bloque a otros nodos dentro de la red Blockchain para su validación y verificación. Cuando ese bloque se ha verificado correctamente, se agregaría al final de la Blockchain para su almacenamiento y finalizaría la transacción.

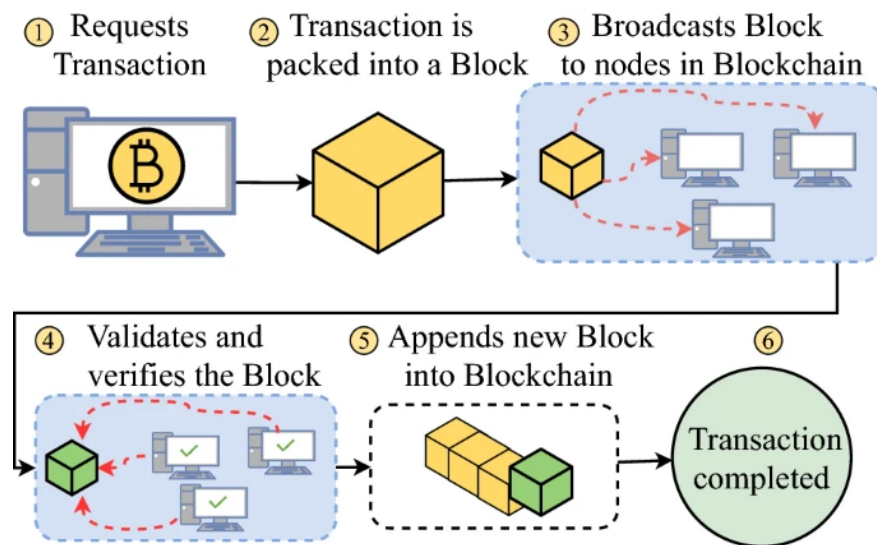


Figura 2: Proceso del *Blockchain*

Blockchain presenta las siguientes características clave [10] [11]:

- **Descentralización:** Cada transacción en la red se realiza solo entre dos nodos a la vez y no requiere validación de un tercero. La descentralización permite que *Blockchain* no dependa de una autoridad central, lo que permite que los nodos tengan derechos de voto iguales en la red, utilizando un algoritmo de consenso para dictar la *Blockchain*.
- **Persistencia:** Cada transacción debe ser validada por mineros de confianza. La persistencia se vincula con la tecnología de inmutabilidad para garantizar que los libros de contabilidad almacenados en los nodos sean absolutos y no se puedan modificar ni eliminar.
- **Anonimato:** Cada minero utiliza una dirección generada como identificación única. Aunque no todas las Blockchains son completamente anónimas y algunas practican la pseudoanonimidad, como Ethereum y Bitcoin, donde se generan direcciones para cada transacción en la *Blockchain*, el principio fundamental se mantiene para garantizar que los mineros en la red puedan permanecer anónimos.
- **Auditabilidad:** Se refiere a tener un punto de referencia para cada transacción dentro de la *Blockchain*, que también se imprime en los nodos de la *Blockchain*. Estos puntos de referencia se utilizan para permitir que cada transacción verificada y realizada en la *Blockchain* sea rastreable. Esta *Auditabilidad* se puede ver como la caracterización de la verificación y dejar un rastro de la transacción en la red *Blockchain*.

Tres componentes principales operan dentro del sistema Blockchain. Todos estos componentes deben trabajar juntos, ya que proporcionan los pilares de soporte para garantizar la descentralización de la red Blockchain [12].

3.1.1. *Distributed ledger*

Los *distributed ledger* ofrecen una base de datos distribuida, que forma una conexión de red entre usuarios, y las computadoras utilizadas por estos usuarios para conectarse se denominan nodos. Dentro de estos nodos hay libros mayores, que son listas ordenadas de transacciones con marcas de tiempo. Estos *distributed ledger* solo se pueden agregar a la base de datos, garantizando una forma segura de rastrear transacciones sin necesidad de una figura central para la verificación. El proceso inicial de las transacciones se realizaba de manera P2P, solo para ser facilitado por Contratos Inteligentes durante la segunda generación de *Blockchains*. Los Contratos Inteligentes son protocolos de transacción que controlan la transmisión de los *distributed ledger* entre nodos.

3.1.2. Almacenamiento Inmutable

El Almacenamiento Inmutable es un componente que se refiere a la capacidad de los nodos de ser inalterables. Cada base de datos se retiene en cada nodo y tiene una referencia de sí misma en la Blockchain como un historial inmutable [10]. El Almacenamiento Inmutable proporciona la función de cifrado para mantener la integridad de los *distributed ledger* dentro de los nodos. Garantiza que ningún otro medio altere el contenido de la transacción, estableciendo un aumento de incentivos y confianza dentro de la *Blockchain*.

3.1.3. Algoritmo de Consenso

Los algoritmos de consenso desempeñan un papel fundamental en la operación de las redes *blockchain*. Estos algoritmos se utilizan para lograr consenso entre los nodos para la alteración o modificación de los *distributed ledger* existentes, solo para agregarlos a un nuevo bloque al final de la cadena dentro de la *Blockchain*. El algoritmo de consenso modera la *Blockchain* al dictar a los nodos cómo lograr un acuerdo y actualizar la red *Blockchain* [12]. Dos de los enfoques más comunes son Proof of Work (PoW) y Proof of Authority (PoA).

3.1.3.1. Proof of Work (PoW) El algoritmo Proof of Work (PoW) es conocido por ser utilizado en la red Bitcoin y otras criptomonedas. En este sistema, los participantes, llamados mineros, compiten para resolver problemas matemáticos complejos. El primero en resolver el problema tiene el derecho de agregar un bloque a la *blockchain* y es recompensado con nuevas unidades de la criptomoneda.

PoW garantiza la seguridad de la red a través de la inversión de recursos computacionales para resolver los desafíos. Sin embargo, este método consume una cantidad considerable de energía y ha llevado al desarrollo de alternativas más sostenibles.

3.1.3.2. Proof of Authority (PoA) Proof of Authority (PoA) es un enfoque que otorga el derecho de validar bloques a entidades específicas consideradas autoridades. Estas autoridades son conocidas y de confianza, eliminando la necesidad de tener una gran capacidad computacional.

PoA ofrece eficiencia y escalabilidad al minimizar los recursos necesarios para validar transacciones. Hyperledger Besu es un ejemplo de una plataforma que admite PoA, lo que lo convierte en una opción atractiva para redes empresariales y permissionadas.

Estos algoritmos de consenso son elementos clave en la arquitectura *blockchain* y desempeñan un papel crucial en la determinación de cómo se llega a un acuerdo sobre la validez de las transacciones.

3.2. Mejora de la Seguridad mediante Blockchain

La aplicación de *blockchain* en entornos de ciberseguridad ofrece varias mejoras significativas:

- **Resistencia a la Alteración:** La estructura inmutable de la cadena de bloques dificulta la manipulación de datos, proporcionando un nivel adicional de seguridad contra ataques de alteración.
- **Transparencia y Auditoría:** La transparencia inherente de la cadena de bloques permite la auditoría en tiempo real de todas las transacciones, facilitando la detección temprana de actividades sospechosas.
- **Descentralización:** Al eliminar la dependencia de una entidad central, blockchain reduce los puntos únicos de falla y mejora la resistencia ante ataques dirigidos.

3.3. Casos de Uso en Ciberseguridad

Existen diversos casos de uso donde la implementación de blockchain puede fortalecer la seguridad en ciberseguridad:

- **Gestión de Identidad:** Utilizar *blockchain* para garantizar la integridad y la gestión segura de identidades digitales.
- **Registro de Eventos:** Mantener un registro inmutable de eventos y acciones para una trazabilidad completa.
- **Protección contra Ransomware:** Utilizar la descentralización y copias distribuidas para prevenir o mitigar el impacto de ataques de ransomware.

La implementación práctica de *blockchain* en Python será abordada en una próxima sección, donde se explorarán herramientas y scripts que demuestren su aplicabilidad en el contexto de la ciberseguridad.

En este trabajo, se han explorado los fundamentos de *blockchain*, destacando sus características clave y su aplicación en ciberseguridad. A continuación, se estudiarán dos tecnologías específicas que desempeñan un papel crucial en el desarrollo y la implementación de soluciones *blockchain*: Hyperledger Besu y Ethereum Solidity.

Ethereum Solidity es un lenguaje de programación específico para contratos inteligentes utilizado en la plataforma Ethereum. Permite la creación de contratos inteligentes que se ejecutan en la máquina virtual Ethereum (EVM). Los contratos inteligentes son protocolos que facilitan y automatizan la ejecución de acuerdos en la *blockchain* de Ethereum.

Hyperledger Besu es un cliente de *blockchain* de código abierto desarrollado bajo la iniciativa Hyperledger de la Fundación Linux. Es compatible con la especificación Ethereum y ofrece características como la capacidad de ejecutar contratos inteligentes y soporte para algoritmos de consenso como Proof of Work (PoW) y Proof of Authority (PoA). Su flexibilidad lo hace adecuado para implementar redes *blockchain* privadas y permissionadas.

En las secciones siguientes, se explorarán con más detalle estas tecnologías y cómo se utilizan para desarrollar soluciones prácticas en el contexto de la ciberseguridad.

4. *Ethereum Solidity*

Ethereum, a menudo considerada como la segunda generación de *blockchain*, es una plataforma descentralizada que va más allá de la simple transferencia de valor, como lo hace Bitcoin. Fue propuesta por Vitalik Buterin en 2013 y desarrollada en 2014, con el objetivo de permitir la ejecución de contratos inteligentes y aplicaciones descentralizadas (DApps) en su red [13].

4.1. Contratos Inteligentes

El concepto central de Ethereum es el de contratos inteligentes. Estos son programas informáticos autoejecutables que operan en la *blockchain* de Ethereum. Los contratos inteligentes están diseñados para ejecutar automáticamente y hacer cumplir acuerdos cuando se cumplen ciertas condiciones, sin necesidad de intermediarios.

La creación y ejecución de contratos inteligentes se realiza a través del lenguaje de programación Solidity. Solidity es un lenguaje específicamente diseñado para escribir contratos inteligentes en la Ethereum Virtual Machine (EVM), la máquina virtual que ejecuta el código en la red Ethereum.

4.2. Ethereum Virtual Machine (EVM)

La Ethereum Virtual Machine (EVM) es una máquina virtual Turing completa que ejecuta el código de los contratos inteligentes en la red Ethereum. Cada nodo en la red ejecuta la EVM, lo que garantiza la consistencia en la ejecución de contratos en toda la red.

4.3. Diferenciadores Clave de Ethereum

Ethereum se destaca por varios diferenciadores clave:

4.3.1. Versatilidad:

Ethereum es conocida por su versatilidad, ya que permite el desarrollo de una amplia gama de aplicaciones descentralizadas y contratos inteligentes. Esto ha llevado a la creación de un ecosistema diverso que va más allá de las transacciones de criptomonedas.

4.3.2. Programabilidad:

La capacidad de programar contratos inteligentes en Solidity brinda a los desarrolladores la flexibilidad de crear soluciones personalizadas y automatizadas en la red Ethereum.

4.3.3. Turing Completeness:

Ethereum se enorgullece de ser una plataforma Turing completa, lo que significa que puede realizar cualquier tarea computacional que pueda ser descrita de manera algorítmica. Esto abre un amplio abanico de posibilidades para el desarrollo de aplicaciones.

4.3.4. Estandarización de Tokens:

Ethereum ha sido pionera en la creación de estándares como ERC-20 y ERC-721, que han facilitado la emisión de tokens y la creación de activos digitales únicos, respectivamente.

4.3.5. Crecimiento de la Comunidad:

La red Ethereum ha experimentado un crecimiento significativo de su comunidad de desarrolladores y usuarios, respaldado por su capacidad para impulsar la innovación en el espacio *blockchain*.

4.4. Solidity y Desarrollo de Contratos Inteligentes

Solidity es el lenguaje de programación utilizado para escribir contratos inteligentes en Ethereum. Es un lenguaje estáticamente tipado diseñado para compilar código que se ejecuta en la Ethereum Virtual Machine (EVM). A continuación, en la siguiente sección, se explorarán algunos conceptos clave relacionados con Solidity y cómo se utilizan para desarrollar contratos inteligentes en Ethereum.

5. *Hyperledger by Besu*

5.1. Hyperledger Besu: Un Enfoque en la Implementación Empresarial

Hyperledger Besu es un cliente de *blockchain* de código abierto desarrollado bajo la iniciativa Hyperledger de la Fundación Linux. A diferencia de Ethereum, que es conocida por su enfoque más general y descentralizado, Hyperledger Besu se ha diseñado específicamente para aplicaciones empresariales y consorcios.

5.1.1. Características Clave de Hyperledger Besu

5.1.1.1. Compatibilidad con Ethereum: Hyperledger Besu es compatible con la especificación Ethereum, lo que significa que puede ejecutar contratos inteligentes escritos en Solidity y es interoperable con la red Ethereum. Esta característica facilita la migración de aplicaciones entre redes y permite la flexibilidad en la elección de la red.

5.1.1.2. Soporte para Algoritmos de Consenso: Hyperledger Besu ofrece soporte para varios algoritmos de consenso, incluyendo Proof of Work (PoW) y Proof of Authority (PoA). Esto permite a las organizaciones elegir el mecanismo de consenso que mejor se adapte a sus necesidades de seguridad y rendimiento.

5.1.1.3. Privacidad y Confidencialidad: Besu incorpora características diseñadas para abordar preocupaciones de privacidad empresarial. Puede implementar redes privadas, lo que significa que solo los participantes autorizados tienen acceso a la información y la capacidad de validación.

5.1.1.4. Gestión de Permisos: Hyperledger Besu permite la configuración de permisos a nivel de nodo y cuenta, brindando control sobre quién puede unirse a la red, proponer bloques y realizar transacciones.

5.1.1.5. Integración con Herramientas Empresariales: Al estar diseñado para aplicaciones empresariales, Hyperledger Besu se integra bien con herramientas y procesos empresariales existentes, facilitando la adopción en entornos corporativos.

5.1.2. Proof of Work (PoW) en Hyperledger Besu

El algoritmo de consenso Proof of Work (PoW) es utilizado por Hyperledger Besu para validar y agregar bloques a la *blockchain*. En este sistema, los nodos, llamados "mineros", compiten para resolver problemas matemáticos complejos. El primer nodo en encontrar la solución valida el bloque y lo agrega a la cadena. Aunque PoW es conocido por su seguridad, también es intensivo en recursos y energía.

5.1.3. Proof of Authority (PoA) en Hyperledger Besu

Hyperledger Besu también admite el algoritmo de consenso Proof of Authority (PoA), que es más eficiente en términos de recursos. En PoA, los nodos validadores son seleccionados y

autorizados por una entidad central de confianza. Estos nodos tienen el derecho exclusivo de proponer y validar bloques. Este enfoque es adecuado para entornos empresariales donde la confianza entre los participantes está establecida.

5.2. Implementación Práctica con Hyperledger Besu

A continuación, en la siguiente sección, se explorarán detalles sobre la implementación práctica de Hyperledger Besu, destacando sus ventajas en escenarios empresariales y proporcionando ejemplos específicos de su aplicación en ciberseguridad y otros casos de uso relevantes.

6. Implementación de Blockchain con Python

Para comenzar se realizará una implementación básica de *blockchain* en python para entender desde el nivel más básico su funcionamiento e implementación, ayudando así a la consolidación de los conceptos estudiados en apartados anteriores.

La elección de Python como lenguaje de implementación para este proyecto de *blockchain* se fundamenta en su versatilidad, sintaxis clara y la disponibilidad de bibliotecas que facilitan el desarrollo de diversos proyectos. A continuación, se abordará el proceso de implementación.

Python destaca por su facilidad de aprendizaje y la rapidez con la que se puede desarrollar software. Su comunidad activa y la amplia gama de bibliotecas hacen que sea una elección natural para implementar proyectos basados en cualquier técnica, como *blockchain*.

Para lograr la seguridad necesaria en *blockchain*, se hará uso de bibliotecas en Python, como `hashlib`, que proporcionan funciones esenciales para la generación de firmas digitales, siendo las funciones hash una parte fundamental de este proceso.

A continuación, se presentarán los códigos realizados y sus respectivas explicaciones y utilidades. Para dar inicio a la codificación de la *Blockchain* en Python, se ha utilizado como guía la referencia de la página *Building a Blockchain from Scratch with Python!* [14].

El código se puede encontrar en el repositorio de python de la autora de este proyecto. El repositorio *Blockchain* del usuario *GrunCrow* [15].

La estructura del proyecto se compone de dos clases principales:

- **Block:** Esta clase representa los bloques individuales de la cadena *blockchain*. Cada bloque contiene información como su índice, el hash del bloque anterior, el tiempo y los datos asociados.
- **Blockchain:** Esta clase gestiona la cadena de bloques y contiene métodos para añadir nuevos bloques, verificar la integridad de la cadena y otras operaciones relacionadas con la *blockchain*.

6.1. Clase Block

La clase **Block** representa la unidad básica de una cadena de bloques (*blockchain*). Cada bloque contiene información clave, incluyendo datos, el hash del bloque actual y el hash del bloque anterior.

El constructor de la clase recibe los parámetros esenciales como el índice del bloque, el hash del bloque anterior, la marca de tiempo de creación, los datos almacenados en el bloque y opcionalmente el hash actual y la dirección del minero (Proof of Stake). El hash actual se calcula automáticamente si no se proporciona.

La clase **Block** incluye métodos para calcular el hash del bloque, convertir el bloque a un diccionario para facilitar la representación en formato JSON, y verificar la validez del bloque mediante la comparación de su hash calculado con el hash actual almacenado.

El código proporcionado incluye también funciones para comparar, obtener y establecer

elementos, así como representaciones en formato cadena para facilitar la comprensión y depuración del código.

La integración de un consenso se logra mediante la introducción del atributo `miner_address`, que almacena la dirección del minero responsable de la creación del bloque. Esta adición es fundamental para implementar un sistema de consenso basado en la participación del minero en lugar de la potencia de cálculo.

```
1 import hashlib
2 import time
3
4 '''
5 This is the basic unit of a blockchain. Each block contains data, the hash
6   of the block, and the hash of the previous block.
7 '''
8 class Block:
9     def __init__(self, index, previous_hash, timestamp, data, current_hash
10 =None, miner_address=None):
11         self.index = index
12         self.previous_hash = previous_hash
13         self.timestamp = timestamp
14         self.data = data
15         self.current_hash = current_hash or self.calculate_hash()
16         # Consensus
17         self.miner_address = miner_address
18
19     def __str__(self):
20         return f"Block Hash: {self.current_hash}\nPrevious Hash: {self.
21 previous_hash}\nData: {self.data}"
22
23     def __repr__(self):
24         return str(self)
25
26     def __eq__(self, other):
27         return self.current_hash == other.current_hash and \
28             self.previous_hash == other.previous_hash and \
29             self.data == other.data
30
31     def __ne__(self, other):
32         return not self.__eq__(other)
33
34     def __hash__(self):
35         return hash(self.current_hash + self.previous_hash + self.data)
36
37     def __len__(self):
38         return len(self.data)
39
40     def __getitem__(self, key):
41         return self.data[key]
42
43     def __setitem__(self, key, value):
44         self.data[key] = value
45
46     def calculate_hash(self):
```

```

45     data_string = str(self.index) + str(self.previous_hash) + str(self
    .timestamp) + str(self.data)
46     return hashlib.sha256(data_string.encode()).hexdigest()
47
48     def to_dict(self):
49         block_dict = {
50             "index": self.index,
51             "previous_hash": self.previous_hash,
52             "timestamp": self.timestamp,
53             "data": self.data,
54             "current_hash": self.current_hash,
55             "miner_address": self.miner_address
56         }
57         return block_dict
58
59     def is_valid(self):
60         return self.current_hash == self.calculate_hash()

```

Listing 1: Clase Block en Python

6.2. Clase Blockchain

La clase `Blockchain` representa una cadena de bloques, donde cada bloque está vinculado al anterior a través de su hash. Esta clase proporciona las funciones esenciales para gestionar la cadena de bloques, incluyendo la creación de bloques, la adición de nuevos bloques, y la validación de la integridad de la cadena.

El constructor inicializa la cadena con el bloque génesis, que se crea mediante la función `create_genesis_block`. La cadena es representada como una lista de objetos de la clase `Block`. La función `create_genesis_block` crea el primer bloque de la cadena con valores predeterminados y un minero asociado denominado "Genesis Miner".

La clase `Block` incluye funciones básicas y necesarias para realizar una cadena de bloques. La función `add_block` se utiliza para añadir un nuevo bloque a la cadena. Recibe datos y la dirección del minero como parámetros y genera un nuevo bloque con estos datos. La función `to_dict` convierte la cadena de bloques a una lista de diccionarios para facilitar su representación en formato JSON.

Por último, la clase `Blockchain` incluye métodos y algoritmos básicos pero típicos de *Blockchain*, como la validación de bloques y un algoritmo de consenso, en este caso, como se ha indicado ya anteriormente, un algoritmo muy básico de consenso. La función `is_valid` verifica la validez de la cadena de bloques, asegurándose de que cada bloque sea válido y que los enlaces entre bloques sean correctos. La función `select_miner` implementa un mecanismo de consenso muy simple seleccionando aleatoriamente un minero de entre los participantes actuales. Los participantes se definen como direcciones en la lista `participants`.

```

1 from block import Block
2 import time
3 import random
4
5 ...

```

```

6 This is a chain of blocks, where each block is linked to the previous one
  via its hash.
7 '''
8
9 class Blockchain:
10     def __init__(self):
11         self.chain = [self.create_genesis_block()]
12
13     def __str__(self):
14         return str(self.chain)
15
16     def __repr__(self):
17         return str(self)
18
19     def __eq__(self, other):
20         return self.chain == other.chain
21
22     def __ne__(self, other):
23         return not self.__eq__(other)
24
25     def __hash__(self):
26         return hash(str(self))
27
28     def __len__(self):
29         return len(self.chain)
30
31     def __getitem__(self, key):
32         return self.chain[key]
33
34     def __setitem__(self, key, value):
35         self.chain[key] = value
36
37     def create_genesis_block(self):
38         # create the first block of the blockchain
39         return Block(0, "0", int(time.time()), data="Genesis Block",
miner_address="Genesis Miner")
40
41     def get_latest_block(self):
42         return self.chain[-1]
43
44     def add_block(self, data, miner_address):
45         index = len(self.chain)
46         previous_block = self.get_latest_block()
47         previous_hash = previous_block.current_hash
48         timestamp = int(time.time())
49         new_block = Block(index, previous_hash, timestamp, data,
miner_address=miner_address)
50         self.chain.append(new_block)
51
52     def to_dict(self):
53         return [block.to_dict() for block in self.chain]
54
55     def is_valid(self):
56         for i in range(1, len(self.chain)):
57             current_block = self.chain[i]

```

```

58         previous_block = self.chain[i - 1]
59
60         if not current_block.is_valid() or \
61             current_block.previous_hash != previous_block.current_hash
62     :
63         return False
64
65     return True
66
67     def select_miner(self):
68         # Simple PoS mechanism: Randomly select a miner from the current
69         participants
68         participants = ["Local Miner", "Server Miner"] # Add your
69         participant addresses
69         return random.choice(participants)

```

Listing 2: Clase Blockchain en Python

6.3. Fichero `gui_server.py`

La implementación de la interfaz del servidor se aborda mediante la creación de clases auxiliares destinadas a representar el funcionamiento de *blockchain*. Inicialmente, se han diseñado dos clases específicas: **server** y **client**.

Estas clases están destinadas a desempeñar roles distintos en la comunicación y funcionamiento del sistema. Sin embargo, durante el desarrollo, se ha decidido consolidar la interfaz gráfica en un único archivo **GUI Server**. Esta decisión se toma con el objetivo de simplificar la estructura del proyecto y mejorar la cohesión del código. Ya que de la forma anterior había que ejecutar los dos *scripts* de forma paralela.

En la fase inicial del desarrollo, se utiliza la combinación de las librerías *Flask* y *Tkinter* para implementar la interfaz gráfica. Aunque estas herramientas son conocidas y han sido utilizadas en proyectos anteriores, el enfoque principal del proyecto no es la creación de una interfaz gráfica compleja. Por esta razón, se recurre a herramientas de Inteligencia Artificial, como *ChatGPT* y *Copilot*, para obtener ayuda en la implementación inicial.

Particularmente, *ChatGPT* desempeña un papel crucial al ser la ayuda principal sobre la creación de la página de *Flask* mucho más visual. La Figura 3 muestra parte del chat con *ChatGPT* para recibir orientación sobre la implementación de la interfaz gráfica.

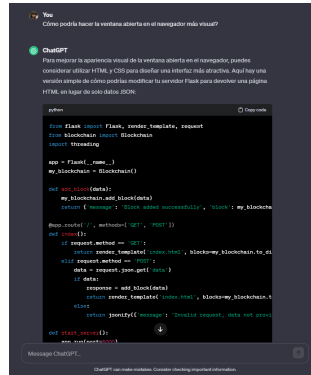


Figura 3: Interacción con ChatGPT para recibir ayuda en la implementación de la interfaz gráfica

Siguiendo las indicaciones proporcionadas por *ChatGPT*, se procede a la creación de una plantilla HTML. El contenido final de la plantilla se ajusta y modifica para adaptarlo al contenido del proyecto y se presenta en la implementación de la interfaz gráfica.

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Blockchain Server</title>
8     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
9     <style>
10         body {
11             font-family: Arial, sans-serif;
12             margin: 20px;
13         }
14
15         h1 {
16             color: #333;
17         }
18
19         #blocks {
20             margin-top: 20px;
21             border: 1px solid #ccc;
22             padding: 10px;
23         }
24
25         #message {
26             margin-top: 10px;
27             color: green;
28         }
29
30         #add-block {
31             margin-top: 20px;
32         }
33
34         #shutdown {

```

```

35         margin-top: 20px;
36     }
37     </style>
38 </head>
39 <body>
40     <div class="container">
41         <h1 class="mt-4">Blockchain Server</h1>
42
43
44         <div id="add-block" class="mb-3">
45             <form method="POST" action="/">
46                 <div class="form-group">
47                     <label for="block_data">Enter Block Data:</label>
48                     <input type="text" class="form-control" id="block_data"
49 " name="block_data" required>
50                 </div>
51                 <button type="submit" class="btn btn-primary">Add Block</
52 button>
53                 <!--<button type="button" class="btn btn-danger" onclick="
54 shutdownServer()">Shut Down Server</button>-->
55             </form>
56         </div>
57
58         <div id="message" class="mb-3">{{ message }}</div>
59
60         <div id="blocks">
61             {% for block in blocks %}
62                 <div>Index: {{ block.index }}</div>
63                 <div>Miner Address: {{ block.miner_address }}</div>
64                 <div>Previous Hash: {{ block.previous_hash }}</div>
65                 <div>Timestamp: {{ block.timestamp }}</div>
66                 <div>Data: {{ block.data }}</div>
67                 <div>Current Hash: {{ block.current_hash }}</div>
68                 <hr>
69             {% endfor %}
70         </div>
71     </div>
72
73     <!--
74     <script>
75         function shutdownServer() {
76             fetch('/shutdown', { method: 'POST' })
77             .then(response => {
78                 if (response.ok) {
79                     alert('Server shutting down...');
80                 } else {
81                     alert('Error while shutting down the server.');

```

```

87     -->
88 </body>
89 </html>

```

Listing 3: Plantilla HTML del proyecto implementada por ChatGPT

La implementación de la interfaz del servidor y del cliente local se basa en la utilización de las clases previamente explicadas, **Block** y **Blockchain**. Se han desarrollado interfaces gráficas tanto para el servidor como para el cliente local.

El código presentado utiliza las librerías **Flask** y **Tkinter** para crear estas interfaces gráficas. La interfaz del servidor se muestra en una ventana del navegador, mientras que la del cliente local se presenta en una ventana de ejecución local en la máquina del usuario.

En el servidor, se define una ruta principal ("/") que maneja las solicitudes GET y POST. Cuando se realiza una solicitud GET desde el navegador, se verifica si la solicitud proviene del navegador o del cliente local implementado en **Tkinter**. En el cliente local, se devuelve la representación de la cadena JSON de la **Blockchain** y en el servidor se renderiza una plantilla HTML que muestra los bloques en la cadena. Así se mantienen actualizados tanto la visualización del cliente local como del servidor cada vez que se añade un bloque. En la Figura 4 se muestra la ventana del navegador tras agregar diferentes bloques.

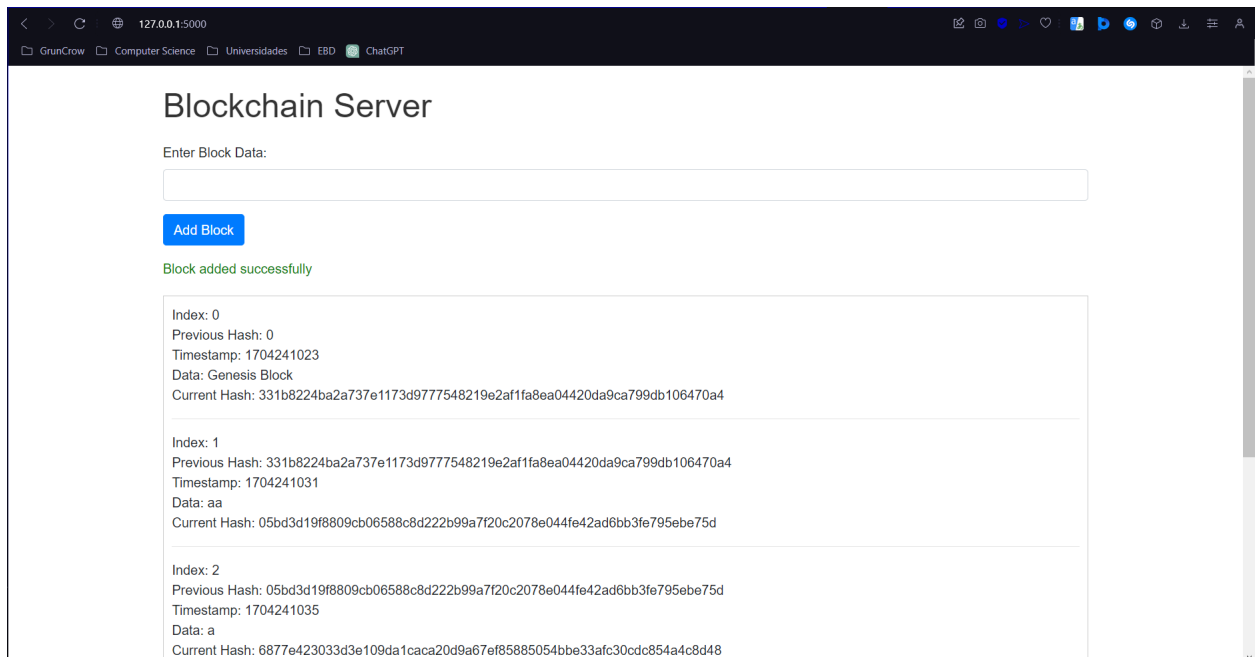


Figura 4: Servidor Blockchain

Cuando se realiza una solicitud POST, se agrega un nuevo bloque a la **Blockchain** con los datos proporcionados. Se actualiza el cuadro de texto con la información de los bloques y se valida la integridad de la cadena, también se ejecuta el consenso de los mineros implementado en la clase **Blockchain**.

El cliente local, implementado en **Tkinter**, presenta una interfaz gráfica con un cuadro de

texto para ingresar datos y un botón para agregar bloques localmente. Al hacer clic en el botón, se selecciona el minero local y se envía la solicitud al servidor. La interfaz también se actualiza periódicamente para mostrar los bloques existentes y validar la cadena. En la Figura 5 se muestra la ventana local con diferentes bloques añadidos.

También se utiliza ChatGPT y Copilot para recibir ayuda sobre la implementación de la interfaz gráfica de esta parte.

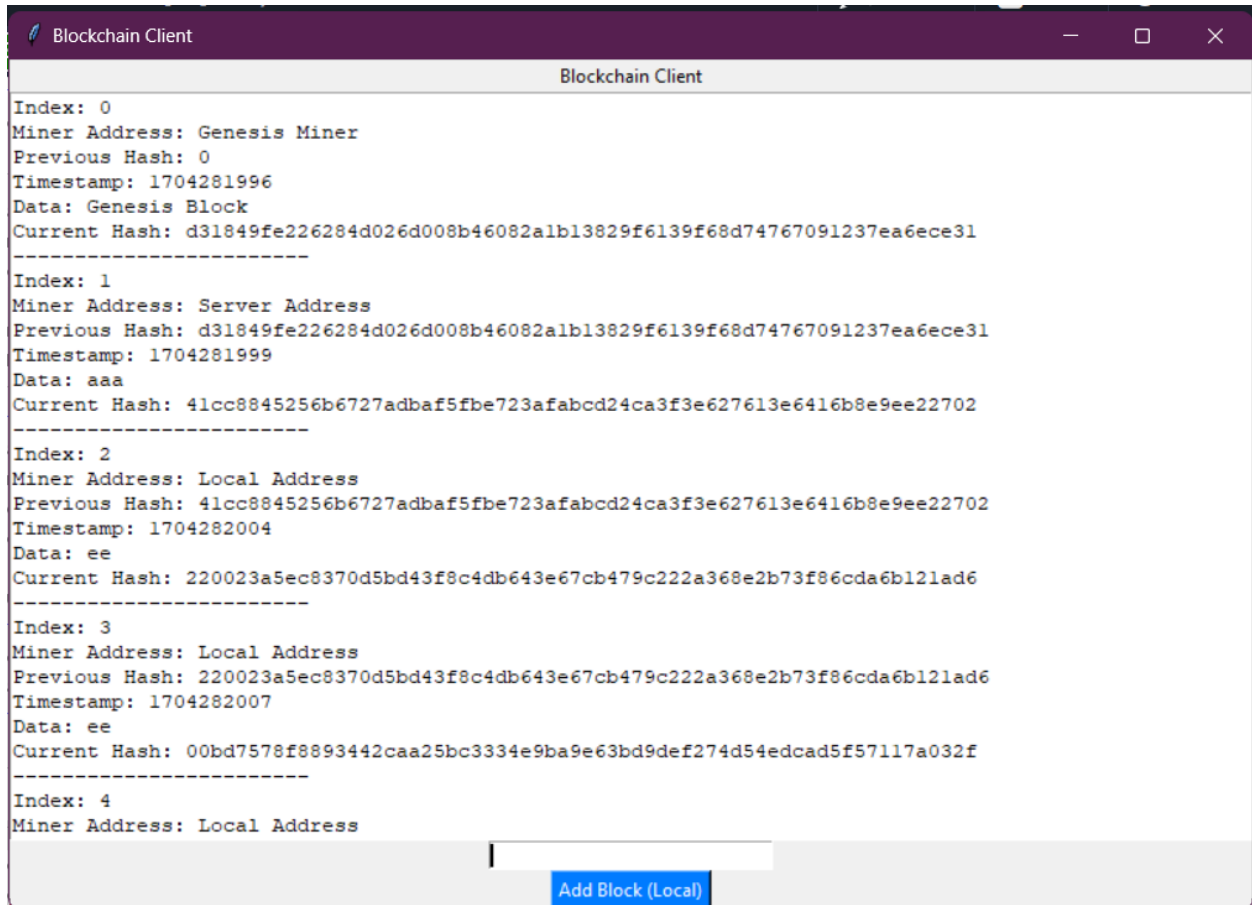


Figura 5: Cliente Blockchain

```

1 from flask import Flask, render_template, request, jsonify
2 from blockchain import Blockchain
3 import threading
4 from tkinter import Tk, Label, Button, Text, END, Entry
5 import requests
6 import webbrowser
7 import time
8
9 LOCAL_ADDRESS = "Local Miner"
10 SERVER_ADDRESS = "Server Miner"
11
12 app = Flask(__name__)
13 my_blockchain = Blockchain()
14

```



```

15 def add_block(data, miner_address):
16     my_blockchain.add_block(data, miner_address)
17     return {'message': 'Block added successfully', 'block': my_blockchain.
18         get_latest_block().to_dict()}
19
20 last_server_response = None # Store last server response to avoid
21     updating the textbox when the server responds with the same data
22
23 def update_textbox():
24     textbox.delete(1.0, END)
25     response = requests.get('http://127.0.0.1:5000/?source=tkinter') #
26     Agregate ?source=tkinter al URL
27     try:
28         response.raise_for_status()
29         blocks = response.json()
30         for block in blocks:
31             textbox.insert(END, f"Index: {block['index']}\n")
32             textbox.insert(END, f"Miner Address: {block['miner_address']}\n")
33             textbox.insert(END, f"Previous Hash: {block['previous_hash']}\n")
34             textbox.insert(END, f"Timestamp: {block['timestamp']}\n")
35             textbox.insert(END, f>Data: {block['data']}\n")
36             textbox.insert(END, f"Current Hash: {block['current_hash']}\n")
37         )
38         textbox.insert(END, "-----\n")
39     except requests.exceptions.RequestException as e:
40         print(f"Error while updating the textbox: {e}")
41
42 def add_block_local():
43     data = entry.get()
44     if data:
45         miner_address = my_blockchain.select_miner()
46         response = add_block(data, LOCAL_ADDRESS)
47         update_textbox()
48         entry.delete(0, END)
49     else:
50         print("Invalid request, data not provided")
51
52 @app.route('/', methods=['GET', 'POST'])
53 def index():
54     global last_server_response
55     if request.method == 'GET':
56         # Verify if the request comes from the browser or from the tkinter
57         client
58         if request.args.get('source') == 'tkinter':
59             return jsonify(my_blockchain.to_dict())
60         else:
61             # Render HTML template
62             return render_template('index.html', blocks=my_blockchain.
63                 to_dict())
64     elif request.method == 'POST':
65         data = request.form.get('block_data')
66         if data != last_server_response:

```

```

62         response = add_block(data, SERVER_ADDRESS)
63         update_textbox()
64         last_server_response = data
65         return render_template('index.html', blocks=my_blockchain.
to_dict(), message=response['message'])
66     else:
67         update_textbox()
68         return render_template('index.html', blocks=my_blockchain.
to_dict())
69
70 @app.route('/validate', methods=['GET'])
71 def validate_blockchain():
72     if my_blockchain.is_valid():
73         return {'message': 'Blockchain is valid'}
74     else:
75         return {'message': 'Blockchain is not valid'}, 500
76
77 def start_server():
78     app.run(port=5000)
79
80 def open_browser():
81     webbrowser.open('http://127.0.0.1:5000/')
82
83 def scheduled_update():
84     update_textbox()
85     validate_blockchain()
86     root.after(1000, scheduled_update) # Schedule the next update after
1000 ms (1 second)
87
88 if __name__ == "__main__":
89     server_thread = threading.Thread(target=start_server)
90     server_thread.start()
91
92     # Iniciar la interfaz local de Tkinter
93     root = Tk()
94     root.title("Blockchain Client")
95
96     # Label
97     label = Label(root, text="Blockchain Client")
98     label.pack()
99
100     # Textbox
101     textbox = Text(root, height=30, width=100)
102     textbox.pack()
103
104     # Entry to add new blocks
105     entry = Entry(root, width=30)
106     entry.pack()
107
108     # Button to add new blocks locally
109     button = Button(root, text="Add Block (Local)", command=lambda:
add_block_local(), bg="#007BFF", fg="white", relief="raised")
110     button.pack()
111
112     # Iniciar la actualizaci n programada

```

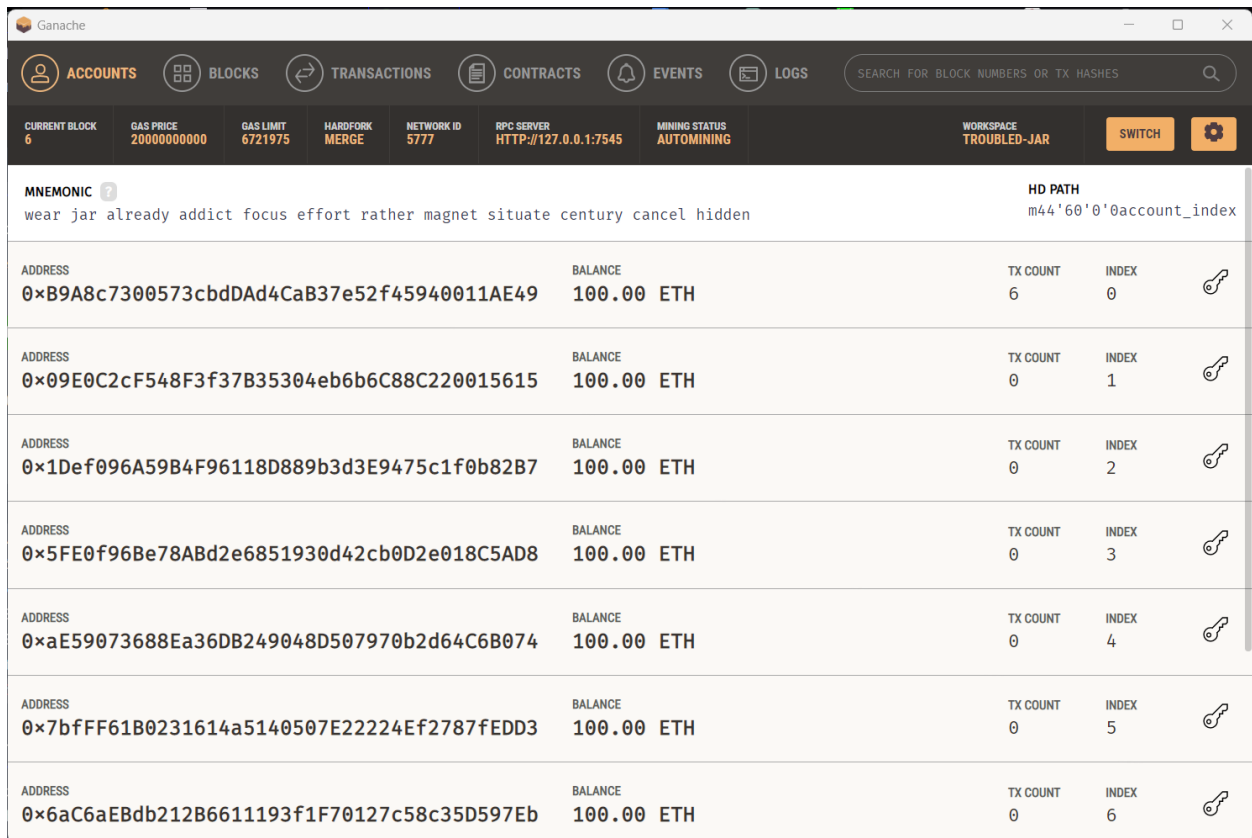
```
113     root.after(1000, scheduled_update)
114
115     # Abrir el navegador web
116     open_browser()
117
118     # Iniciar la interfaz local de Tkinter
119     root.mainloop()
```

Listing 4: Código de gui_server con ayuda de ChatGPT

7. Implementación de Ethereum Solidity

Para implementar contratos inteligentes en Ethereum utilizando Solidity, se requiere el uso de herramientas externas como Ganache.

Ganache es un framework que permite iniciar y probar una blockchain local personalizada. Proporciona 10 cuentas preconfiguradas para interactuar con la blockchain, cada una con su propia clave privada, dirección de Ethereum, saldo y registro de transacciones [16].



MNEMONIC ?		HD PATH	
wear jar already addict focus effort rather magnet situate century cancel hidden		m44'60'0'0account_index	
ADDRESS	BALANCE	TX COUNT	INDEX
0xB9A8c7300573cbdDAd4CaB37e52f45940011AE49	100.00 ETH	6	0
0x09E0C2cF548F3f37B35304eb6b6C88C220015615	100.00 ETH	0	1
0x1Def096A59B4F96118D889b3d3E9475c1f0b82B7	100.00 ETH	0	2
0x5FE0f96Be78ABd2e6851930d42cb0D2e018C5AD8	100.00 ETH	0	3
0xaE59073688Ea36DB249048D507970b2d64C6B074	100.00 ETH	0	4
0x7bFF61B0231614a5140507E22224Ef2787fEDD3	100.00 ETH	0	5
0x6aC6aEBdb212B6611193f1F70127c58c35D597Eb	100.00 ETH	0	6

Figura 6: Lista de cuentas de Ganache

Al seleccionar una de las direcciones proporcionadas por Ganache, se pueden obtener los datos necesarios para conectarla con la blockchain en el código Python:

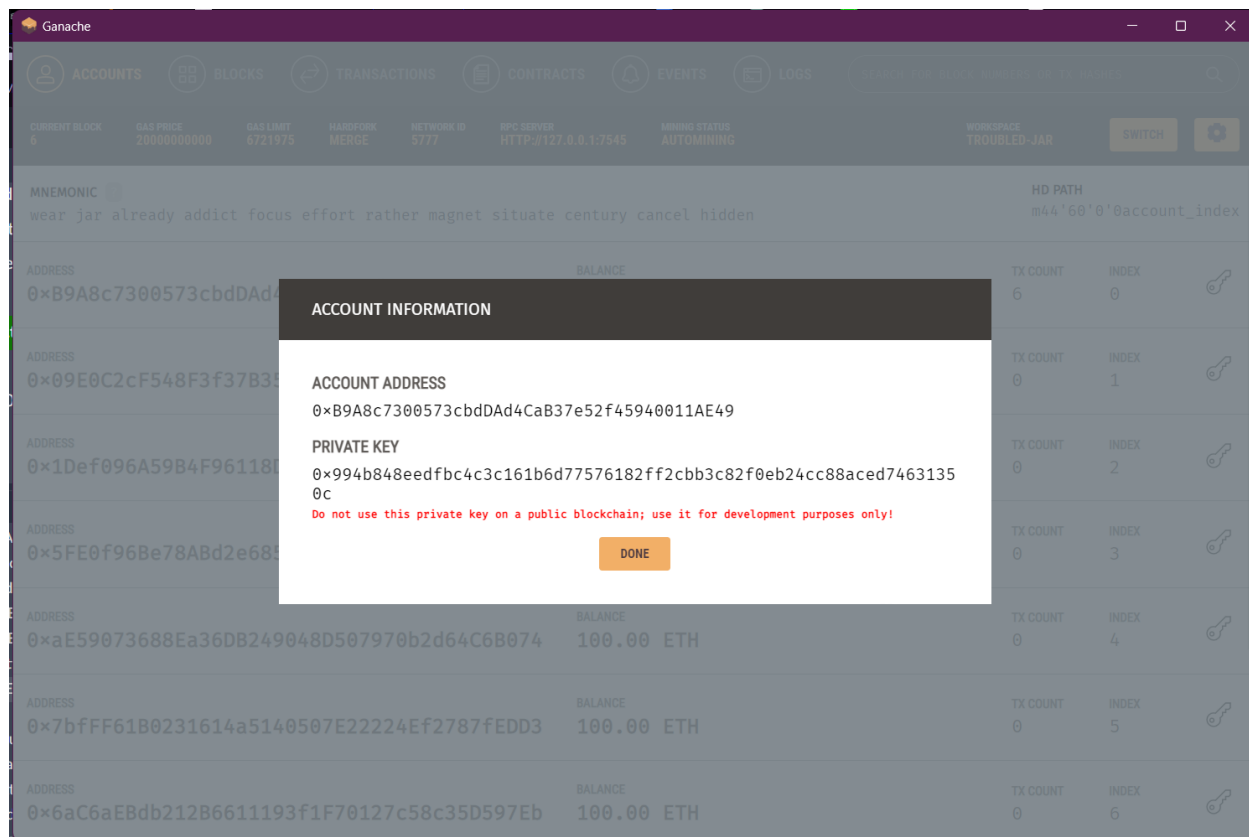


Figura 7: Datos de una de las cuentas de Ganache

Además de configurar Ganache, se requiere desplegar un contrato inteligente. A continuación, se presenta un ejemplo de un contrato `SimpleStorage.sol`:

```

1 // SimpleStorage.sol
2 // SPDX-License-Identifier: MIT
3 pragma solidity ^0.8.0;
4
5 contract SimpleStorage {
6     uint256 public data;
7
8     function setData(uint256 _data) public {
9         data = _data;
10    }
11 }

```

Listing 5: Smart Contract

Este contrato debe compilarse utilizando el compilador Solidity (`solc`) en la terminal:

```
1 solc SimpleStorage.sol --bin --abi --optimize -o ./build
```

Listing 6: Compilación del contrato inteligente

Para desplegarlo en este caso se va a utilizar *Truffle*. Una vez instalado localmente se debe inicializar en el directorio:

```
1 truffle init
```

Listing 7: Inicializar proyecto de Truffle

Se coloca el contrato inteligente, en este caso el fichero `SimpleStorage.sol` dentro del directorio `contracts` del proyecto Truffle que se acaba de crear. Y se configura el archivo de despliegue. En el directorio `migrations` del proyecto Truffle, se crea un archivo de migración. En este archivo, define el proceso de despliegue del contrato inteligente. En este caso su contenido es el siguiente:

```
1 const SimpleStorage = artifacts.require("SimpleStorage");
2
3 module.exports = function(deployer) {
4   deployer.deploy(SimpleStorage);
5 };
```

Listing 8: Configuración del archivo de despliegue

Se debe verificar que el archivo `truffle-config.js` esté configurado correctamente para conectarse a la red Ganache local.

```
1 module.exports = {
2   /**
3    * Networks define how you connect to your ethereum client and let you
4    * set the
5    * defaults web3 uses to send transactions. If you don't specify one
6    * truffle
7    * will spin up a managed Ganache instance for you on port 9545 when you
8    * run `develop` or `test`. You can ask a truffle command to use a
9    * specific
10   * network from the command line, e.g
11   *
12   * $ truffle test --network <network-name>
13   */
14   networks: {
15     // Useful for testing. The `development` name is special - truffle
16     // uses it by default
17     // if it's defined here and no other network is specified at the
18     // command line.
19     // You should run a client (like ganache, geth, or parity) in a
20     // separate terminal
21     // tab if you use this network and you must also set the `host`, `port`
22     // and `network_id`
23     // options below to some value.
24     //
25     development: {
26       host: "127.0.0.1",      // Localhost (default: none)
27       port: 7545,            // Standard Ethereum port (default: none)
28       network_id: "*",       // Any network (default: none)
29       gas: 6721975           // Gas limit used for deploys (default:
30                             6721975)
31     },
32   },
33 }
```

Listing 9: Configuración Truffle

Despliegue del contrato inteligente: Con Ganache ejecutándose y conectado a través del puerto 7545, se debe ejecutar el siguiente comando para desplegar el contrato inteligente:

```
1 truffle migrate --network development
```

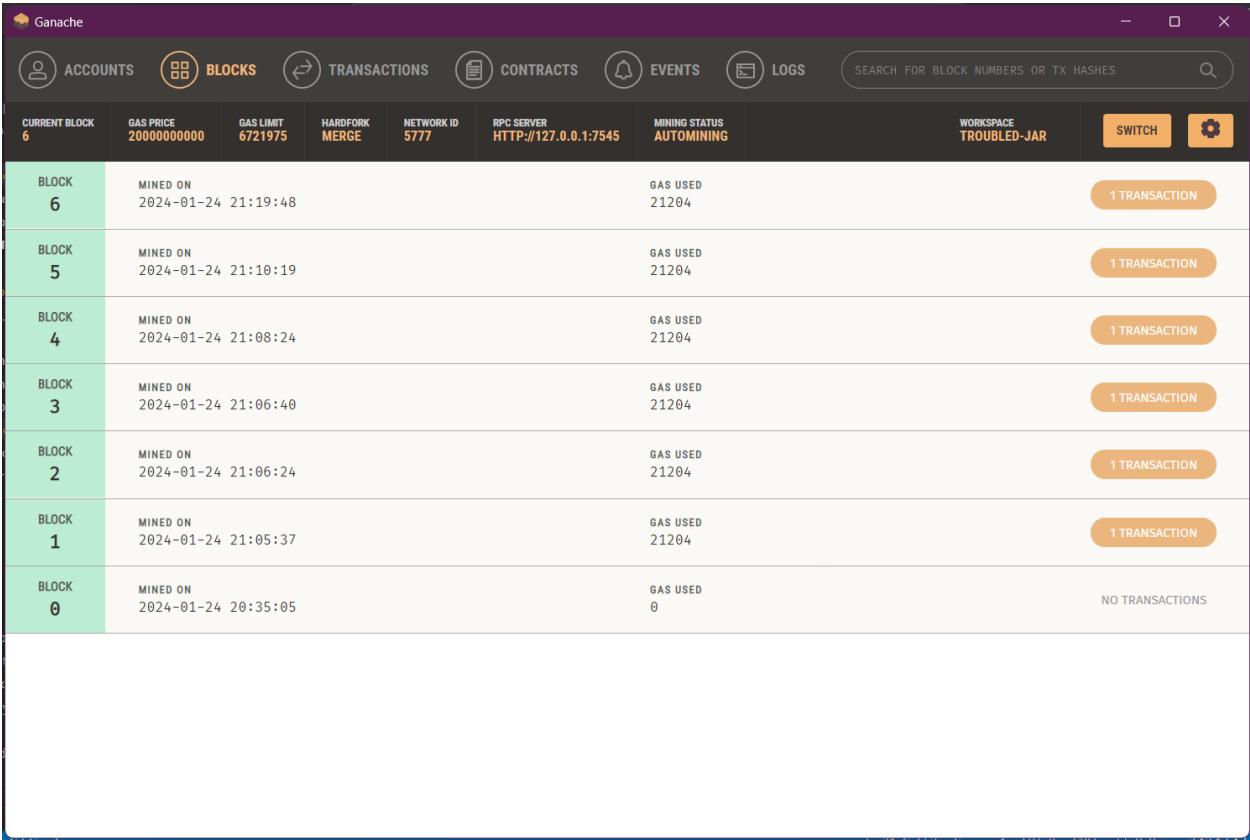
Listing 10: Inicializar proyecto de Truffle

Por otro lado, se debe elaborar un código en Python que conecte con Ganache y que sea el encargado de interactuar, intentando añadir o borrar bloques de la Blockchain de Ganache, el código utilizado es el siguiente:

```
1 from web3 import Web3
2 from eth_account import Account
3 import random
4
5 # Con ctate a tu nodo Ethereum (puedes usar un nodo local o Infura)
6 w3 = Web3(Web3.HTTPProvider('http://localhost:7545'))
7
8 # Define tu clave privada y la direcci n del contrato compilado
9 private_key = '0
   x994b848eedfbc4c3c161b6d77576182ff2cbb3c82f0eb24cc88aced74631350c '
10 contract_address = '0xB9A8c7300573cbdDAd4CaB37e52f45940011AE49 '
11
12 # Crea una instancia del contrato inteligente
13 contract_abi = [{"inputs": [], "name": "data", "outputs": [{"internalType": "
   uint256", "name": "", "type": "uint256"}], "stateMutability": "view", "type": "
   function"}, {"inputs": [{"internalType": "uint256", "name": "_data", "type": "
   uint256"}], "name": "setData", "outputs": [], "stateMutability": "nonpayable"
   , "type": "function"}]
14 contract = w3.eth.contract(address=contract_address, abi=contract_abi)
15
16 # Establece la cuenta desde la clave privada
17 account = Account.from_key(private_key)
18
19 # Genera un valor de gas aleatorio entre 0 y 100000
20 gas = random.randint(5000, 100000)
21
22 # Transacci n: Establece el valor del contrato
23 tx_hash = contract.functions.setData(42).transact({'from': account.address
   , 'gas': gas})
24
25 # Espera a que la transacci n sea minada
26 transaction_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
27
28 # Lectura: Obtiene el valor del contrato
29 current_data = contract.functions.data().call()
30
31 print(f'Valor actual del contrato: {current_data}')
```

Listing 11: Ethereum Solidity Code

Al ejecutar este código, se pueden agregar bloques a la blockchain según lo especificado en el script. Después de ejecutarlo varias veces, se pueden observar diferentes bloques en la ventana de Ganache:



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE	SWITCH	SETTINGS
6	20000000000	6721975	MERGE	5777	HTTP://127.0.0.1:7545	AUTOMINING	TROUBLED-JAR		
BLOCK 6	MINED ON 2024-01-24 21:19:48					GAS USED 21204		1 TRANSACTION	
BLOCK 5	MINED ON 2024-01-24 21:10:19					GAS USED 21204		1 TRANSACTION	
BLOCK 4	MINED ON 2024-01-24 21:08:24					GAS USED 21204		1 TRANSACTION	
BLOCK 3	MINED ON 2024-01-24 21:06:40					GAS USED 21204		1 TRANSACTION	
BLOCK 2	MINED ON 2024-01-24 21:06:24					GAS USED 21204		1 TRANSACTION	
BLOCK 1	MINED ON 2024-01-24 21:05:37					GAS USED 21204		1 TRANSACTION	
BLOCK 0	MINED ON 2024-01-24 20:35:05					GAS USED 0		NO TRANSACTIONS	

Figura 8: Blockchain con bloques en Ganache

Esta implementación ha sido mucho más sencilla al no haber sido necesaria la implementación de algoritmos sino el uso de librerías ya existentes con sus funciones ya implementadas y con algoritmos de consenso también ya incluidos. Por otro lado, la dificultad ha residido en el hecho de no conocer las herramientas de las que se ha hecho uso, ni el compilador del Smart Contract que ha tenido que ser instalado, ni Ganache ni las librerías utilizadas. Pero una vez se conocen estas herramientas es un método mucho más seguro de implementar una blockchain privada.

8. Caso de Uso

La tecnología *blockchain*, inicialmente popularizada por su aplicación en las criptomonedas, ha demostrado ser una herramienta versátil en diversas aplicaciones de diferentes sectores. Su capacidad para proporcionar trazabilidad, transparencia y seguridad en las transacciones ha llevado a su adopción en áreas más allá de este ámbito financiero. Mientras las criptomonedas continúan siendo el caso de uso más expandido, conocido y destacado, existen otras aplicaciones quizás menos conocidas pero igualmente interesantes de la tecnología *blockchain*.

La característica distintiva de *blockchain* que ha impulsado su adopción en diversos campos es la trazabilidad. La capacidad de rastrear y verificar de manera segura y transparente la procedencia y los eventos asociados a un elemento específico a lo largo de su ciclo de vida tiene un impacto significativo en la confianza y la autenticidad de la información.

El caso más conocido de aplicación de *blockchain* es, como se ha mencionado anteriormente, en el ámbito de las criptomonedas. Bitcoin y Ethereum han liderado el camino al cambiar fundamentalmente la forma en tradicional de entender y realizar transacciones financieras. La descentralización, inmutabilidad y transparencia inherentes a *blockchain* han eliminado la necesidad de intermediarios centralizados en las transacciones, proporcionando seguridad y confiabilidad.

Más allá de las criptomonedas, existen aplicaciones menos conocidas pero igualmente relevantes de la tecnología *blockchain*. Un caso de uso interesante es su implementación en cadenas de suministro sostenibles, donde se aprovecha la capacidad de *blockchain* para garantizar la transparencia en la producción y distribución de productos *ecofriendly*, como alimentos orgánicos o productos reciclados.

En esta sección, se explorará una aplicación menos conocida de la tecnología *blockchain*, destacando cómo estas tecnologías pueden llegar a abordar problemas específicos en diversos sectores y ofrecer soluciones tangibles para mejorar la eficiencia, la autenticidad y la sostenibilidad en diversas áreas de estudio. Al ir más allá de las criptomonedas, se puede descubrir cómo la aplicación de *blockchain* puede transformar sectores enteros, proporcionando soluciones innovadoras y sostenibles a problemas contemporáneos.

8.1. Cadenas de Suministro Sostenibles

Así, además del ámbito financiero, *blockchain* ha demostrado ser una herramienta valiosa en la creación de cadenas de suministro sostenibles. En este caso, se estudiará cómo la tecnología *blockchain* puede asegurar la transparencia y sostenibilidad en las cadenas de suministro, particularmente en la industria de productos ecológicos, como alimentos orgánicos o productos reciclados.

Por ejemplo, una cadena de suministro que produce y distribuye alimentos orgánicos certificados. En este escenario, la autenticidad y la sostenibilidad de los productos son fundamentales para los consumidores sensibles con el respeto con el medio ambiente y el cambio climático. Sin embargo, en la cadena de suministro tradicional, a menudo existe opacidad en cuanto al origen exacto de los productos y las prácticas utilizadas en su producción [17].

Las cadenas de suministro tradicionales pueden enfrentar desafíos relacionados con la au-

tenticidad de los productos ecológicos. La falta de visibilidad en los procesos de producción y distribución puede dar lugar a que se hagan reclamaciones falsas o expongan información inexacta sobre la sostenibilidad de los productos.

La implementación de *blockchain* en esta cadena de suministro puede abordar estos problemas de manera efectiva. Al utilizar la tecnología *blockchain*, cada etapa del proceso, desde la siembra hasta la entrega, puede registrarse de manera inmutable en la cadena de bloques. Esto permite a los consumidores y partes interesadas rastrear la procedencia de cada producto, asegurando su autenticidad y sostenibilidad.

Además de garantizar la transparencia en la cadena de suministro, la implementación de *blockchain* también puede proporcionar beneficios adicionales, como la reducción del riesgo de fraude, la mejora de la eficiencia operativa y el fortalecimiento de la confianza entre los consumidores y los productores ecológicos.

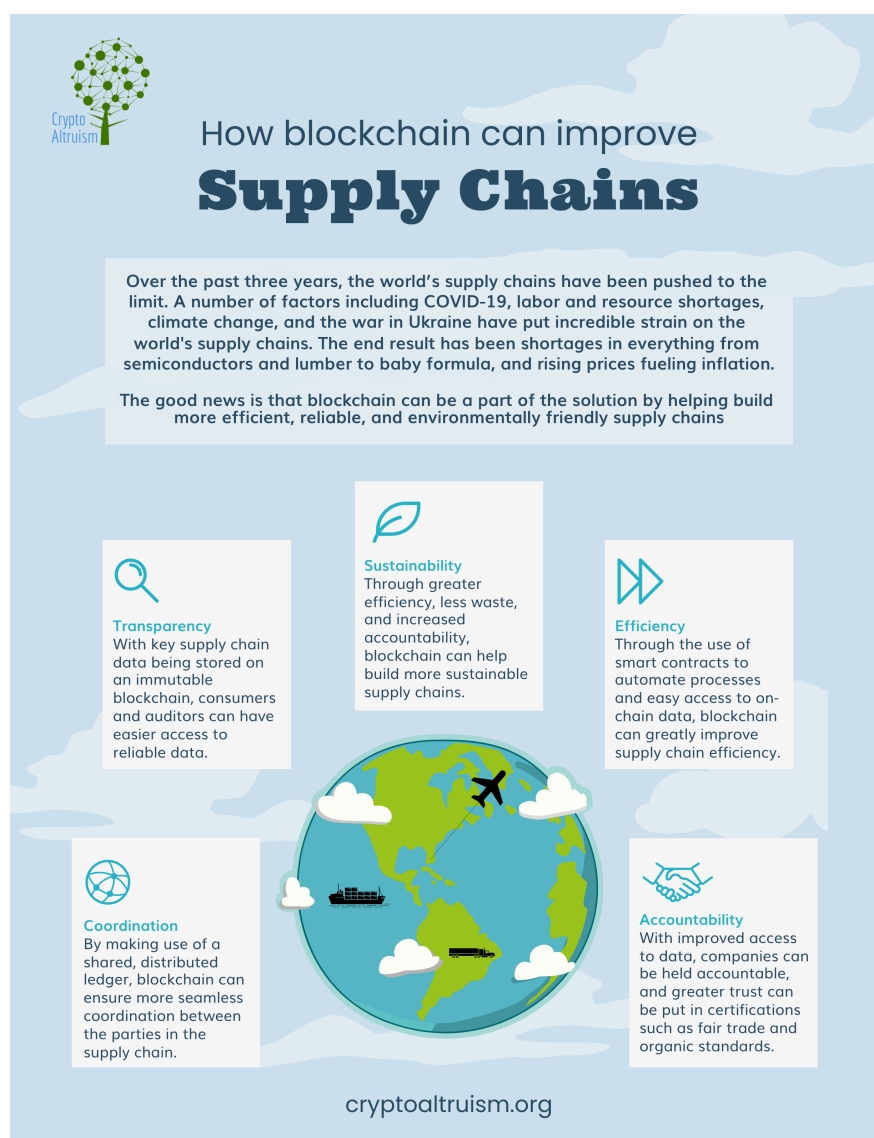


Figura 9: Ejemplo de mejoras con la aplicación de *Blockchain* a cadenas de suministros

Este caso de uso destaca cómo la tecnología *blockchain* va más allá de su aplicación en criptomonedas y demuestra su capacidad para abordar problemas específicos en industrias como la de productos ecoamigables, ofreciendo soluciones tangibles para mejorar la transparencia y sostenibilidad en las cadenas de suministro.

9. Comparación

A continuación se comparan las diferentes implementaciones y enfoques que se han considerado en este trabajo:

En la implementación en Python, se destaca su facilidad de comprensión y flexibilidad para crear una blockchain desde cero. Este enfoque implica la creación de objetos de bloque y blockchain, así como el uso de algoritmos de consenso específicamente programados para el proyecto. Además, se pueden crear puntos de acceso para diversas funciones del blockchain, como añadir transacciones, mediante el uso de frameworks como Flask, que ha sido el utilizado en este caso. Es importante mencionar que esta implementación se ejecuta en un servidor local, lo que puede limitar su escalabilidad y disponibilidad en comparación con otras soluciones.

Por otro lado, Ethereum Solidity se ha elaborado con ayuda de frameworks como Ganache que se presenta como una opción para el desarrollo y prueba de contratos inteligentes en Ethereum. Esta herramienta proporciona una blockchain personalizada que facilita el despliegue y la prueba de contratos escritos en Solidity. Ganache genera automáticamente 10 cuentas con sus respectivas claves privadas y direcciones de Ethereum, lo que simplifica el proceso de interacción con la cadena de bloques. Además, ofrece funcionalidades para visualizar el saldo y las transacciones de cada cuenta. En el proyecto mencionado, se utilizó un código Python para introducir bloques en la cadena gestionada por Ganache, lo que demuestra su compatibilidad con este entorno de desarrollo.

Finalmente, Hyperledger Besu se presenta como un cliente Ethereum de código abierto diseñado especialmente para entornos empresariales, tanto en redes públicas como privadas. Besu ofrece una variedad de algoritmos de consenso, incluyendo Proof of Stake, Proof of Work y Proof of Authority, lo que permite adaptarse a diferentes requisitos y escenarios de uso. Además, sus esquemas de permisos están diseñados específicamente para entornos de consorcio, lo que garantiza un control adecuado sobre la participación en la red y el acceso a los recursos. Aunque este enfoque no se ha probado en el proyecto mencionado, su potencial para aplicaciones empresariales lo convierte en una opción interesante a considerar.

En resumen, cada enfoque tiene sus propias características y ventajas, y la elección entre ellos dependerá de los requisitos específicos del proyecto y las necesidades del negocio. La implementación en Python es adecuada para proyectos que requieren un alto nivel de personalización y flexibilidad, mientras que Ganache es ideal para el desarrollo y prueba de contratos inteligentes en Ethereum. Por su parte, Hyperledger Besu se destaca por su enfoque empresarial y sus capacidades avanzadas de gestión de permisos y consenso.

10. Retos y Consideraciones

La implementación de la tecnología *blockchain* presenta diversos retos y consideraciones clave. En concreto su implementación en python para la realización de este proyecto ha supuesto unos retos que se han podido experimentar de primera mano, por otro lado también se ha implementado la opción de Ethereum Solidity haciendo uso de frameworks y librerías que facilitan la implementación en python aunque requieran la instalación de otras herramientas. En esta sección, se identifican algunos de estos retos a nivel general y los experimentados durante el desarrollo del proyecto.

10.1. Retos en *Blockchain*

A nivel general la adopción de blockchain como una parte de las tecnologías que pueden contribuir a la ciberseguridad no está exenta de obstáculos. Algunos de los retos más relevantes son:

10.1.1. Escalabilidad y Rendimiento

El aumento en la cantidad de transacciones puede resultar en problemas de escalabilidad y afectar el rendimiento de la red blockchain. Resolver estos problemas es esencial para garantizar una ciberseguridad efectiva en entornos de alta demanda.

10.1.2. Interoperabilidad

La falta de estándares comunes entre diferentes plataformas blockchain puede dificultar la interoperabilidad, lo que se traduce en la integración efectiva de sistemas en entornos ciberseguros. Establecer protocolos comunes es esencial para una colaboración fluida entre diversas infraestructuras.

10.1.3. Seguridad y Privacidad

A pesar de su reputación de seguridad, blockchain no está exento de retos. Proteger contra ataques y vulnerabilidades sigue siendo un área crítica, al igual que abordar preocupaciones sobre la privacidad de los datos almacenados en la cadena de bloques.

10.1.4. Adopción y Conciencia

La falta de comprensión generalizada y la resistencia al cambio pueden obstaculizar la adopción de blockchain en estrategias de ciberseguridad. Educar a los profesionales y crear conciencia sobre los beneficios de esta tecnología son retos continuos.

10.2. Retos en la Implementación de Blockchain en python

Durante el desarrollo del proyecto, se enfrentaron diversos retos que influyeron en la implementación y la experiencia general. A continuación, se describen los principales dificultades encontrados y las estrategias adoptadas para superarlas.

Uno de los retos fundamentales fue la implementación general del proyecto. Inicialmente, la percepción del problema era de una complejidad significativa, especialmente debido al desconocimiento de las posibilidades y la implementación de la tecnología blockchain. Sin embargo, a medida que se exploró la documentación disponible en Internet y se revisaron enfoques previos de otros usuarios, la implementación se simplificó mediante el uso de objetos y una estructura más clara.

Otro desafío destacado fue diseñar una interfaz de usuario sencilla y atractiva para facilitar la interacción con el programa. El uso de herramientas como *ChatGPT* y *Copilot* permitió superar este desafío, proporcionando una base sólida sobre la cual agregar las funcionalidades necesarias.

En el ámbito de la implementación de *blockchain* en python, se enfrentaron retos específicos relacionados con la validación de bloques y el consenso. La validación asegura que cada bloque tenga el hash correcto y que el bloque anterior sea válido. Aunque durante las pruebas con dos clientes no surgieron grandes problemas, se reconoce que la técnica de validación puede resultar más beneficiosa cuando se enfrenta a una mayor cantidad de datos que ingresan simultáneamente.

En cuanto al consenso, se implementó una técnica sencilla debido a la naturaleza limitada de los clientes durante las pruebas. Al ampliar el número de clientes y la cantidad de bloques a agregar, se reconoce la necesidad de desarrollar un algoritmo más complejo y adaptable a las necesidades específicas.

Finalmente, se identificó un reto no directamente relacionado con *blockchain*, pero muy importante para la visualización del proyecto. Coordinar la actualización y mostrar simultáneamente los bloques agregados en ambas pantallas de los clientes representó un desafío de sincronización que requirió una atención especial para garantizar la coherencia en la visualización de la cadena de bloques compartida.

10.3. Retos en la Implementación de Blockchain con Ethereum Solidity

La implementación de contratos inteligentes en Ethereum Solidity presentó varios desafíos, especialmente si no se tiene un buen conocimiento del desarrollo de blockchain y contratos inteligentes. Algunos de los retos encontrados durante el proceso incluyeron el aprendizaje de Solidity, un lenguaje de programación específico de Ethereum utilizado para escribir contratos inteligentes. Además, desplegar un contrato inteligente en la blockchain de Ethereum requiere seguir varios pasos, incluida la compilación del contrato, la gestión de las direcciones y claves privadas, y la interacción con la red Ethereum. Esto implicó comprender los procedimientos y herramientas necesarios para el despliegue correcto del contrato.

Una vez desplegado el contrato inteligente, fue muy importante realizar pruebas para garantizar su funcionalidad y seguridad. La depuración de los contratos inteligentes en una red en vivo puede ser un reto, ya que cualquier error podría tener consecuencias graves. La seguridad de los contratos inteligentes es de gran importancia en Ethereum.

Además, la implementación de contratos inteligentes a menudo requiere integrarse con otras herramientas y servicios, como proveedores de infraestructura blockchain, exploradores de

bloques y bibliotecas de desarrollo. Coordinar estas integraciones puede ser un desafío técnico adicional. En este caso se ha hecho uso de Ganache y de la librería de ethereum para Python.

En el caso anterior se realizó una interfaz gráfica para facilitar el añadir bloques a la cadena y visualizarlos. En este último caso no se ha elaborado ninguna interfaz gráfica concreta para el proyecto. Ya que en el caso de Ganache ya consta de una interfaz gráfica para el servidor donde visualizar los bloques añadidos y sus datos. Sin embargo, el cliente y la forma de añadir los datos no tiene ningún tipo de interfaz gráfica, se debe realizar directamente desde el código de python. En este proyecto se ha añadido un ejemplo de añadir bloques a la blockchain.

11. Conclusiones

A pesar de los retos y consideraciones identificados, la implementación de blockchain ofrece oportunidades significativas para fortalecer la seguridad digital. Al abordar estos retos de manera proactiva, es posible construir sistemas robustos que aprovechen el potencial de esta tecnología innovadora.

Además, leer a niveles más profundos sobre *blockchain* ha permitido conseguir un mayor conocimiento sobre esta técnica y sus posibilidades de uso. Por otro lado, implementar por métodos propios *blockchain* ha permitido terminar de afianzar los conocimientos adquiridos en la parte de estudio teórico. Ha sido un proyecto muy interesante de realizar.

Posteriormente se han estudiado diferentes enfoques más conocidos y robustos que la implementación desde cero en Python. Los casos estudiados son Hyperledger by Besu y Ethereum Solidity. La implementación realizada se ha hecho sólo de uno de estos casos, el de Ethereum Solidity. Que ha demostrado ser una muy buena opción y solución, pues asegura métodos robustos para la implementación de una blockchain privada.

Una vez realizadas las implementaciones se ha elaborado una comparación. A nivel práctico sólo se pueden comparar la elaboración de la blockchain desde 0 en python y de Ethereum Solidity pero no de la opción de Hyperledger by Besu. Aún así ha sido muy interesante comparar ambos métodos y el esfuerzo que requiere implementar ambos así como conocer la flexibilidad que ambos permiten.

Por otro lado, el posterior pensamiento de posibilidades de aplicar esta tecnología en algún caso de uso no tan conocido como las criptomonedas ha permitido conocer mejor la flexibilidad de uso de esta tecnología y que no se limita al ámbito financiero.

Referencias

- [1] S. Sadik, M. Ahmed, L. F. Sikos, and A. N. Islam, “Toward a sustainable cybersecurity ecosystem,” *Computers*, vol. 9, no. 3, p. 74, 2020.
- [2] J. C. Romero, “Ciberseguridad: Evolución y tendencias,” *bie3: Boletín IEEE*, no. 23, pp. 460–494, 2021.
- [3] W. Z. Umpiri, “Blockchain y la innovación en las tecnologías,” *TecnoHumanismo*, vol. 2, no. 2, pp. 117–125, 2022.
- [4] N. Kshetri, “Blockchain’s roles in strengthening cybersecurity and protecting privacy,” *Telecommunications policy*, vol. 41, no. 10, pp. 1027–1038, 2017.
- [5] M. Templ and M. Sariyar, “A systematic overview on methods to protect sensitive data provided for various analyses,” *International Journal of Information Security*, vol. 21, no. 6, pp. 1233–1246, 2022.
- [6] M. Mijwil, O. J. Unogwu, Y. Filali, I. Bala, and H. Al-Shahwani, “Exploring the top five evolving threats in cybersecurity: An in-depth overview,” *Mesopotamian journal of cybersecurity*, vol. 2023, pp. 57–63, 2023.
- [7] H. Al-Mohannadi, Q. Mirza, A. Namanya, I. Awan, A. Cullen, and J. Disso, “Cyber-attack modeling analysis techniques: An overview,” in *2016 IEEE 4th international conference on future internet of things and cloud workshops (FiCloudW)*, pp. 69–76, IEEE, 2016.
- [8] S. F. Aboelfotoh and N. A. Hikal, “A review of cyber-security measuring and assessment methods for modern enterprises,” *JOIV: International Journal on Informatics Visualization*, vol. 3, no. 2, pp. 157–176, 2019.
- [9] M. Di Pierro, “What is the blockchain?,” *Computing in Science & Engineering*, vol. 19, no. 5, pp. 92–95, 2017.
- [10] M. Sopek, P. Gradzki, W. Kosowski, D. Kuziski, R. Trójezak, and R. Trypuz, “Graph-chain: a distributed database with explicit semantics and chained rdf graphs,” in *Companion Proceedings of the The Web Conference 2018*, pp. 1171–1178, 2018.
- [11] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An overview of blockchain technology: Architecture, consensus, and future trends,” in *2017 IEEE international congress on big data (BigData congress)*, pp. 557–564, Ieee, 2017.
- [12] J. Zarrin, H. Wen Phang, L. Babu Saheer, and B. Zarrin, “Blockchain for decentralization of internet: prospects, trends, and challenges,” *Cluster Computing*, vol. 24, no. 4, pp. 2841–2866, 2021.
- [13] V. Buterin *et al.*, “Ethereum white paper,” *GitHub repository*, vol. 1, pp. 22–23, 2013.
- [14] A. Islam, “Building a blockchain from scratch with python,” 2023. Medium.
- [15] GrunCrow, “Blockchain implementation.” <https://github.com/GrunCrow/blockchain>, 2024. GitHub repository.

- [16] Izertis, “Cómo arrancar y testear tu propia blockchain con Ganache.” <https://www.izertis.com/es/-/blog/como-arrancar-y-testear-tu-propia-blockchain-con-ganache>.
- [17] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, “Blockchain technology and its relationships to sustainable supply chain management,” *International journal of production research*, vol. 57, no. 7, pp. 2117–2135, 2019.