UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Blockchain como herramiento en la trazabilidad de un producto

Trabajo Fin de Grado

Grado en Tecnologías Interactivas

AUTOR/A: Chova Aparisi, Pablo

Tutor/a: Marín-Roig Ramón, José

CURSO ACADÉMICO: 2023/2024

RESUMEN

El proyecto consiste en un prototipo que simula la trazabilidad de un producto alimentario mediante el uso de un Arduino con un sensor de humedad y temperatura, cuyos datos se certifican en una blockchain. El proyecto aborda la necesidad de una trazabilidad efectiva de productos para garantizar calidad, seguridad y confianza en la industria alimentaria. Mediante la integración de tecnología blockchain y dispositivos IoT, se asegura una cadena de suministro transparente y segura, permitiendo la verificación constante del estado del producto desde su origen hasta el consumidor final.

Palabras clave: Blockchain, Arduino, IOT, Trazabilidad, Node-Red, Hedera.

Abstract

The project consists of a prototype that simulates the traceability of a food product using an Arduino with a humidity and temperature sensor, with data certified on a blockchain. The project addresses the need for effective product traceability to ensure quality, safety, and trust in the food industry. By integrating blockchain technology and IoT devices, a transparent and secure supply chain is ensured, allowing constant verification of the product's status from its origin to the final consumer.

Keywords: Blockchain, Arduino, IOT, Traceability, Node-Red, Hedera.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo I: Introducción

1.1. Presentación

La trazabilidad de un producto se logra mediante la recolección de datos a través de sensores IoT (Internet de las Cosas), que monitorean y registran en tiempo real diversas condiciones del producto, tales como la temperatura, humedad y ubicación. Estos datos, una vez capturados, son certificados y registrados en la blockchain, una tecnología que garantiza su integridad, seguridad y transparencia a lo largo de toda la cadena de suministro, desde la producción hasta el consumidor final.

En este proyecto se presenta un prototipo, en el cual, se ha utilizado un Arduino M5STACK junto a un sensor DH11 para la captura de datos. Estos datos han sido enviados mediante el protocolo MQTT a NODE-RED, donde esta tecnología ha convertido los datos en un formato idóneo para su introducción en HEDERA y mediante un smartcontract programado en Solidity se ha conseguido certificar los datos capturados en la blockchain. Esto tiene como objetivo saber el origen y el recorrido del producto, para así optimizar los procesos logísticos reduciendo los errores y retrasos en las industrias como la alimentaria o farmacéutica.

1.2. Motivación

La motivación para adoptar una solución basada en IoT y blockchain surge de las claras debilidades observadas en las tecnologías de trazabilidad tradicionales, donde la mayoría de los registros se realizan manualmente, lo que los hace propensos a errores humanos y manipulaciones. Incluso si estos procesos fueran automatizados, seguirían careciendo de una garantía de fiabilidad. La combinación de la sensorización IoT con la blockchain no solo automatiza los procesos de registro, eliminando la intervención humana y sus posibles errores, sino que también asegura la integridad y autenticidad de los datos. Esta innovadora integración permite una trazabilidad precisa y confiable, previniendo fraudes y asegurando la transparencia en toda la cadena de suministro.

1.3. Objetivos del Proyecto

El **principal objetivo** es realizar una prueba piloto o prototipo simulando la trazabilidad de un producto en la cadena de valor.

Los **objetivos secundarios** incluyen:

- Analizar la teoría del blockchain
- Analizar las tecnologías utilizadas en este prototipo

1.4. Estructura del proyecto

El contenido restante de la memoria está estructurado de la siguiente manera:

Capítulo II: Marco teórico

En el segundo capítulo, se realiza una recopilación de información e investigación respecto a la trazabilidad, del mismo modo que la historia y la definición de la tecnología blockchain. Además, se indaga el uso y la implementación de los sensores IoT en la industria alimentaria.

Capítulo III: Tecnologías

En este capítulo se desarrolla las principales tecnologías utilizadas para el desarrollo e implementación del proyecto.

Capítulo IV: Desarrollo del proyecto

En el cuarto capítulo se explica desde el diseño de las conexiones electrónicas al Arduino hasta la explicación detallada de todo el software desarrollado y necesario para la certificación de los datos de la cadena alimentaria en la tecnología de blockchain.

Capitulo V: Resultados y discusión

Se evalúan las fortalezas que pueden aportar la tecnología blockchain a la cadena alimentaria y se nombran ejemplos de especificaciones de comida donde su principal característica y valor cobran un sentido cuando la transparencia de la procedencia de estos alimentos está garantizada, así pues, se conecta directamente los beneficios de la tecnología con la necesidad de estas garantías.

Capítulo VI: Conclusiones y trabajo futuro

Capítulo VII: Github
Capítulo VIII: Bibliografía

En el último capítulo se nombran las referencias utilizadas para realizar este documento

Capítulo II: Marco teórico

En el segundo capítulo, se realiza una recopilación de información e investigación respecto a la situación del proyecto. En primer lugar, se efectúa un análisis de la tecnología blockchain y la sensorización IoT, explorando su implementación en la trazabilidad de productos alimentarios. Además, se examina la situación actual de la trazabilidad en la cadena de suministro alimentaria, identificando las debilidades y desafíos de los métodos tradicionales. Este análisis incluye una revisión de casos de estudio y ejemplos de proyectos exitosos que han integrado blockchain e IoT para mejorar la transparencia, seguridad y eficiencia en la gestión de la trazabilidad. La investigación también aborda aspectos regulatorios y normativos relevantes, así como las tendencias emergentes en el sector. En conjunto, este capítulo proporciona una base sólida para comprender el contexto tecnológico y situacional del proyecto, estableciendo los fundamentos necesarios para el desarrollo y la implementación de la solución propuesta.

2.1. Trazabilidad

2.1.1 Definición

La trazabilidad se refiere a la capacidad de seguir el rastro de un producto a lo largo de todas las etapas de su producción, transformación y distribución. Esta práctica es esencial para garantizar la calidad, la seguridad y la autenticidad de los productos, permitiendo identificar el origen de las materias primas, los procesos de fabricación y las condiciones de almacenamiento y transporte. La trazabilidad asegura que cada etapa del ciclo de vida de un producto esté documentada y pueda ser verificada de manera precisa y confiable.

2.1.2 Historia

La historia de la trazabilidad se remonta a tiempos antiguos, cuando los comerciantes utilizaban métodos básicos para rastrear sus bienes y asegurar la calidad y autenticidad de los productos que vendían. Con la Revolución Industrial en el siglo XIX, la necesidad de sistemas de trazabilidad más precisos se hizo evidente, especialmente en industrias como la alimentaria y la farmacéutica, donde la seguridad del consumidor era una preocupación primordial. En el siglo XX, la introducción de tecnologías como los códigos de barras y los sistemas informáticos permitió una gestión más eficiente y detallada de la trazabilidad. La globalización y la complejidad creciente de las cadenas de suministro modernas han incrementado aún más la importancia de contar con sistemas robustos de trazabilidad.

2.1.3 Futuro

El futuro de la trazabilidad está vinculado al desarrollo y adopción de tecnologías avanzadas como el Internet de las Cosas (IoT) y la blockchain. Los sensores IoT permitirán la recolección de datos en tiempo real y la monitorización continua de las condiciones de los productos a lo largo de toda la cadena de suministro. La tecnología blockchain proporcionará una capa adicional de seguridad y transparencia, registrando de manera inmutable toda la información relevante. Estas innovaciones tienen el potencial de transformar la trazabilidad, haciéndola más precisa, fiable y accesible, lo cual será esencial para satisfacer las demandas de un mercado globalizado y cada vez más exigente.

2.2. Blockchain

2.2.1 Definición

La blockchain, o cadena de bloques, es una tecnología de registro descentralizado que permite almacenar datos de manera segura y transparente a través de una red de nodos distribuidos. Cada bloque de la cadena contiene un conjunto de transacciones verificadas y un hash del bloque anterior, creando una cadena inmutable y segura de información. Esta tecnología es conocida por su capacidad para garantizar la integridad de los datos sin necesidad de intermediarios centralizados, lo que la hace ideal para aplicaciones donde la transparencia y la seguridad son fundamentales.

2.2.2 Historia

La historia de la blockchain comienza con la publicación del libro blanco de Bitcoin en 2008 por un autor bajo el seudónimo de Satoshi Nakamoto. Este documento describía un sistema de dinero digital descentralizado basado en la tecnología blockchain. Bitcoin fue la primera implementación práctica de esta tecnología y demostró su potencial para transformar la manera en que se realizan y verifican las transacciones. Desde entonces, la blockchain ha evolucionado significativamente, con el desarrollo de múltiples plataformas y aplicaciones más allá de las criptomonedas, incluyendo contratos inteligentes, cadenas de suministro, y sistemas de votación, entre otros.

2.2.3 Tipos de Redes

Existen varios tipos de redes blockchain, cada una con características y aplicaciones específicas:

- Blockchain pública: Es una red abierta y descentralizada donde cualquiera puede unirse y participar. Bitcoin y Ethereum son ejemplos de blockchains públicas. Estas redes son transparentes y seguras, pero pueden ser menos eficientes debido a la necesidad de consenso masivo.

- Blockchain privada: Es una red cerrada y controlada por una sola organización. Solo los participantes autorizados pueden unirse y realizar transacciones. Estas redes son más rápidas y eficientes, pero ofrecen menos transparencia que las públicas.

- Blockchain de consorcio: Es una red gestionada por un grupo de organizaciones que trabajan juntas. Combina las ventajas de las redes públicas y privadas, proporcionando eficiencia y cierta transparencia, al tiempo que mantiene un control restringido.

- Blockchain híbrida: Combina características de blockchains públicas y privadas, permitiendo que algunos datos sean públicos y otros privados. Este tipo de red ofrece flexibilidad y control, adecuándose a diversas necesidades empresariales.

2.2.4 Smart Contracts

Los smart contracts (contratos inteligentes, en inglés) son programas informáticos diseñados para ejecutarse automáticamente a medida que las personas o empresas involucradas en un acuerdo van cumpliendo con las cláusulas del mismo. Están basados en la tecnología blockchain y prometen transformar en un futuro no muy lejano la forma tradicional de hacer negocios, eliminando la necesidad de interpretar si una cláusula se ha ejecutado o no, haciendo, por tanto, que los smart contracts sean deterministas y se ejecuten autónomamente.En la trazabilidad de un producto, los smart contracts pueden ser utilizados para verificar y certificar cada etapa del ciclo de vida del producto. Por ejemplo, un smart contract puede activar automáticamente un pago al productor una vez que un sensor IoT registre que el producto ha sido entregado en las condiciones adecuadas y en el lugar correcto. Del mismo modo, cuando el producto llega al almacén, otro smart contract puede registrar automáticamente esta información en la blockchain, garantizando que todos los datos son precisos e inmutables.

Además, los smart contracts pueden gestionar el cumplimiento de regulaciones y normativas. Por ejemplo, en la industria alimentaria, los contratos inteligentes pueden verificar automáticamente que los productos han sido mantenidos a la temperatura adecuada durante el transporte y almacenamiento. Si alguna condición no se cumple, el smart contract puede emitir alertas y tomar medidas correctivas, como detener la distribución del producto hasta que se resuelva el problema.

Los smart contracts también facilitan la colaboración entre diferentes actores de la cadena de suministro, asegurando que cada parte cumpla con sus obligaciones antes de proceder a la siguiente etapa. Esto aumenta la eficiencia y la confianza entre los participantes, ya que cada transacción y movimiento del producto está registrado y es verificable en la blockchain.

En resumen, los smart contracts en la trazabilidad de productos permiten una automatización precisa y segura de los procesos, garantizando que cada paso se registre de manera inmutable y transparente. Esto no solo mejora la eficiencia operativa, sino que también aumenta la confianza en la integridad del producto y la cadena de suministro en su totalidad.

2.2.5 Funcionamiento de una red Blockchain

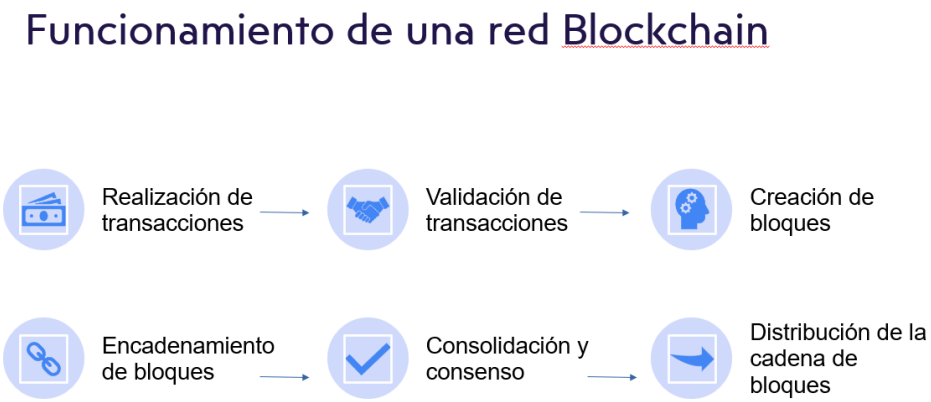

Figura 1: Funcionamiento de una red Blockchain

2.2.6 Beneficios económicos

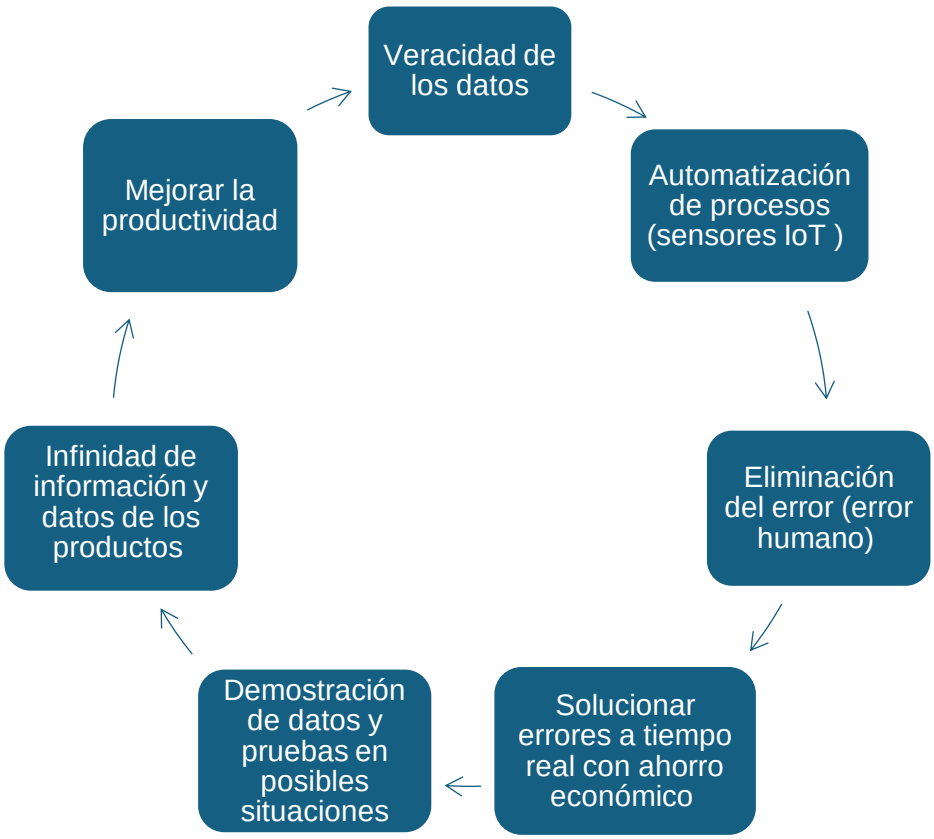

Figura 2: Beneficios económicos del Blockchain

2.2.7 Casos de Uso

Walmart y la Trazabilidad con Blockchain

Walmart, uno de los minoristas más grandes del mundo, ha adoptado la tecnología blockchain para mejorar la trazabilidad de sus productos alimenticios. Esta iniciativa no solo busca aumentar la eficiencia en la cadena de suministro, sino también garantizar la seguridad alimentaria y aumentar la confianza del consumidor. A continuación, se describe cómo Walmart ha implementado la blockchain tanto a nivel técnico como desde la perspectiva del usuario final.

Implementación Técnica de la Blockchain en Walmart

- Plataforma Utilizada: Walmart ha colaborado con IBM para utilizar la plataforma IBM Food Trust, una solución basada en la tecnología blockchain Hyperledger Fabric. Esta plataforma permite a los participantes de la cadena de suministro registrar y compartir información de manera segura y transparente.
- Integración con la Cadena de Suministro: La implementación técnica de la blockchain en Walmart involucra la integración de todos los participantes de la cadena de suministro, desde productores hasta minoristas. Cada participante tiene un nodo en la red blockchain, lo que les permite registrar datos en tiempo real sobre el movimiento de los productos.
- Registro de Datos: Los datos registrados en la blockchain incluyen información sobre el origen de los productos, condiciones de transporte, fechas de recolección y entrega, y cualquier proceso de manipulación que los productos hayan experimentado. Por ejemplo, en el caso de una caja de mangos, se registran datos desde la granja donde se cultivaron hasta el supermercado donde se venden.
- Smart Contracts: Se utilizan contratos inteligentes (smart contracts) para automatizar y garantizar el cumplimiento de los acuerdos entre las partes. Por ejemplo, un contrato inteligente puede liberar el pago al productor una vez que los mangos han sido entregados y verificados en el almacén de Walmart.
- Escaneo y Verificación: Los productos son etiquetados con códigos QR que pueden ser escaneados en cada punto de la cadena de suministro. Cada escaneo actualiza la información en la blockchain, proporcionando una trazabilidad completa y en tiempo real.

Beneficios para el Usuario Final

1. **Transparencia y Confianza:** Los consumidores pueden escanear el código QR en el empaque de los productos para acceder a toda la información registrada en la blockchain. Esto incluye detalles sobre el origen, las condiciones de transporte y los controles de calidad. Por ejemplo, un consumidor que compra mangos en Walmart puede escanear el código QR y ver la granja de donde provienen, la fecha en que fueron recolectados y el recorrido hasta llegar al supermercado.

2. **Seguridad Alimentaria:** En caso de un brote de contaminación

alimentaria, la blockchain permite a Walmart rastrear rápidamente el origen del problema. Esto agiliza las retiradas de productos y minimiza el impacto en la salud pública. Por ejemplo, si se detecta una contaminación en un lote de lechugas, Walmart puede identificar rápidamente la granja de origen y retirar solo los productos afectados, evitando retiradas masivas e innecesarias.

3. **Sostenibilidad:** La trazabilidad completa también permite a los consumidores tomar decisiones más informadas sobre la sostenibilidad de los productos que compran. Pueden verificar si los productos han sido producidos y transportados de manera sostenible, lo cual es cada vez más importante para los consumidores conscientes del medio ambiente.

4. **Eficiencia en la Cadena de Suministro:** La blockchain reduce la necesidad de intermediarios y papeleo, lo que agiliza los procesos y reduce costos. Esto puede traducirse en precios más competitivos para los consumidores. Además, la eficiencia mejorada en la cadena de suministro asegura que los productos frescos lleguen más rápidamente a las tiendas, mejorando la calidad y la frescura de los alimentos.

5. **Protección contra el Fraude:** La inmutabilidad de la blockchain protege contra la falsificación de productos y garantiza la autenticidad de la información. Esto es especialmente importante para productos de alto valor y con certificaciones específicas, como los alimentos orgánicos. Los consumidores pueden estar seguros de que los productos etiquetados como orgánicos o libres de pesticidas realmente cumplen con esos estándares.

Ejemplo Práctico: El Caso de los Mangos

Etapa 1: Producción

- **Registro en la Granja:** Los mangos son recolectados en una granja y se registra la información en la blockchain, incluyendo la ubicación de la granja, la fecha y la hora de la cosecha.

- **Primera Inspección:** Se realiza una inspección de calidad en la granja. Los resultados de la inspección se registran en la blockchain.

Etapa 2: Transporte

- **Etiquetado y Escaneo:** Los mangos son empaquetados y etiquetados con un código QR único. El escaneo del código QR registra el inicio del transporte en la blockchain.
- **Monitoreo en Tiempo Real:** Sensores en el contenedor registran y transmiten datos sobre las condiciones de transporte (temperatura, humedad) a la blockchain.

Etapa 3: Recepción y Almacenamiento

- **Llegada al Almacén:** Los mangos llegan al almacén de Walmart, donde se escanea el código QR para registrar su llegada.

- **Inspección de Calidad:** Se realiza otra inspección de calidad en el almacén. Los resultados se registran en la blockchain.

Etapa 4: Distribución y Venta

- **Transporte a la Tienda:** Los mangos se distribuyen a las tiendas minoristas. Cada movimiento es registrado en la blockchain mediante el escaneo del código QR.

- **Disponibilidad en la Tienda:** Una vez en la tienda, los mangos están disponibles para los consumidores, quienes pueden escanear el código QR para acceder a toda la información de trazabilidad.

- **Perspectiva del usuario final:** un consumidor en una tienda de Walmart decide comprar un paquete de mangos. Al escanear el código QR con su smartphone, accede a una interfaz de usuario amigable que muestra:

    - Origen del Producto: Información sobre la granja de origen, incluyendo ubicación y prácticas agrícolas.
    - Historial de Transporte: Datos sobre las condiciones de transporte, incluyendo temperatura y humedad.
    - Inspecciones de Calidad: Resultados de las inspecciones de calidad realizadas en diferentes etapas.
    - Fecha de Recolección y Entrega: Fechas exactas de recolección y entrega, asegurando frescura del producto.
    - Certificaciones: Verificación de cualquier certificación específica (orgánico, comercio justo, etc.).

Esta transparencia no solo fortalece la confianza del consumidor en la marca Walmart, sino que también permite a los consumidores tomar decisiones informadas sobre los productos que compran, promoviendo la seguridad alimentaria y la sostenibilidad.
En resumen, la implementación de blockchain en Walmart para la trazabilidad de productos alimenticios mejora significativamente la eficiencia y la transparencia en la cadena de suministro, proporcionando beneficios tangibles tanto a nivel técnico como para el usuario fin.

Ejemplo de Uso de Blockchain en Navidul

Navidul, una de las marcas líderes en el sector cárnico en España, ha adoptado la tecnología blockchain para mejorar la trazabilidad de sus productos. La implementación de blockchain permite a Navidul ofrecer a sus clientes una transparencia sin precedentes sobre el origen y el recorrido de sus productos, desde la granja hasta la mesa del consumidor. A continuación, se detalla cómo Navidul utiliza la blockchain para garantizar la calidad y la seguridad de sus productos.

Implementación de Blockchain en Navidul

Navidul ha integrado la tecnología blockchain para registrar y certificar cada etapa del proceso de producción de sus productos. Este sistema de registro inmutable y transparente asegura que toda la información relevante sobre los productos esté disponible para los consumidores, distribuidores y reguladores.

- Registro de Datos en la Granja: En la fase inicial, se recopilan datos sobre el origen de los animales, incluyendo información genética, alimentación y condiciones de vida. Estos datos son registrados en la blockchain, asegurando que la información sea precisa y verificable.

- Sensorización IoT en la Producción: Durante el proceso de producción, Navidul utiliza sensores IoT para monitorear diversas condiciones, como la temperatura y la humedad en las instalaciones de procesamiento. Estos sensores capturan datos en tiempo real, que son automáticamente registrados en la blockchain. Esto garantiza que cada etapa del procesamiento se realice en condiciones óptimas, manteniendo la calidad y la seguridad del producto.

- Transporte y Almacenamiento: La blockchain se utiliza para rastrear el transporte y el almacenamiento de los productos. Sensores IoT instalados en los vehículos de transporte monitorean las condiciones ambientales, asegurando que los productos sean transportados en condiciones adecuadas. Los datos de ubicación y condiciones de almacenamiento se registran en la blockchain, proporcionando una visibilidad completa del recorrido del producto.

- Distribución y Venta: En la fase final, cuando los productos llegan a los puntos de venta, la blockchain permite a los distribuidores y minoristas acceder a toda la información registrada sobre el producto. Los consumidores pueden escanear un código QR en el empaque del producto para obtener información detallada sobre su origen, procesamiento y transporte.

Beneficios de la Blockchain para Navidul

La adopción de blockchain por parte de Navidul ha proporcionado varios beneficios significativos:

1. **Transparencia y Confianza del Consumidor**: La capacidad de los consumidores para acceder a información detallada y verificable sobre el origen y el recorrido del producto aumenta la confianza en la marca Navidul. Los consumidores pueden estar seguros de que los productos que compran cumplen con los más altos estándares de calidad y seguridad.

2. **Mejora en la Gestión de la Cadena de Suministro**: La blockchain facilita una gestión más eficiente de la cadena de suministro al proporcionar una visibilidad completa de cada etapa del proceso. Esto permite a Navidul identificar y resolver rápidamente cualquier problema que pueda surgir, minimizando las interrupciones y garantizando la entrega oportuna de productos frescos y seguros.

3. **Cumplimiento Normativo**: La blockchain ayuda a Navidul a cumplir con las regulaciones y normativas de seguridad alimentaria al proporcionar un registro inmutable de todos los datos relevantes. Esto facilita las auditorías regulatorias y asegura que la empresa cumpla con todas las leyes y regulaciones aplicables.

4. **Prevención de Fraudes**: La inmutabilidad de la blockchain previene la manipulación y el fraude de datos. Esto asegura que la información sobre los productos sea precisa y confiable, protegiendo tanto a los consumidores como a la marca Navidul.

Futuras Mejoras y Aplicaciones

Navidul continúa explorando nuevas formas de mejorar su sistema de trazabilidad con blockchain. Algunas de las mejoras futuras incluyen:

- **Integración con Inteligencia Artificial**: La integración de IA con blockchain podría proporcionar análisis predictivos y recomendaciones proactivas para mejorar la eficiencia operativa y la calidad del producto. Por ejemplo, la IA podría analizar los datos de los sensores IoT para predecir posibles problemas y sugerir medidas correctivas antes de que ocurran.

- **Expansión a Nuevos Productos y Mercados**: Navidul planea expandir el uso de blockchain a otros productos y mercados, ofreciendo la misma transparencia y confianza en toda su línea de productos. Esto podría incluir nuevos productos cárnicos y la expansión a mercados internacionales.

- **Colaboración con Otros Actores de la Cadena de Suministro**: Navidul está explorando la posibilidad de colaborar con otros actores de la cadena de suministro, como proveedores y minoristas, para crear un ecosistema de trazabilidad más integrado y eficiente. Esto podría incluir la creación de

consorcios blockchain para compartir datos y mejorar la eficiencia de la cadena de suministro en toda la industria.

Conclusión

La implementación de la tecnología blockchain por parte de Navidul es un ejemplo destacado de cómo esta tecnología puede mejorar la trazabilidad y la transparencia en la industria alimentaria. Al proporcionar un registro inmutable y verificable de cada etapa del proceso de producción, la blockchain ayuda a Navidul a garantizar la calidad y la seguridad de sus productos, aumentar la confianza del consumidor y cumplir con las normativas de seguridad alimentaria. A medida que Navidul continúa innovando y mejorando su sistema de trazabilidad, la blockchain seguirá siendo una herramienta esencial para mantener la integridad y la transparencia en toda la cadena de suministro.

Figura 3: Diagrama de trazabilidad Blockchain de Navidul

2.3. Sensores IoT en la Trazabilidad

2.3.1 Definición

El Internet de las Cosas (IoT) se refiere a la red de dispositivos físicos que están conectados a Internet, recopilando y compartiendo datos. En el contexto de la trazabilidad, los sensores IoT pueden monitorear en tiempo real diversas condiciones ambientales y de manejo, como la temperatura, la humedad y la ubicación geográfica, asegurando que los productos se mantengan en condiciones óptimas durante su tránsito a lo largo de la cadena de suministro.

2.3.2 Implementación en la Industria Alimentaria

En la industria alimentaria, la sensorización IoT permite un control más estricto de las condiciones de almacenamiento y transporte. Por ejemplo, los sensores de temperatura y humedad se pueden colocar en almacenes y vehículos de transporte para asegurar

que los alimentos se mantengan frescos y seguros. Si las condiciones se desvían de los parámetros aceptables, los sensores pueden enviar alertas en tiempo real, permitiendo acciones correctivas inmediatas.

2.3.3 Beneficios

La implementación de sensores IoT en la trazabilidad alimentaria ofrece numerosos beneficios.

Primero, proporciona datos precisos y en tiempo real sobre las condiciones de almacenamiento y transporte, mejorando la gestión de la cadena de suministro. Segundo, ayuda a prevenir la pérdida de productos y asegura que los alimentos lleguen a los consumidores en condiciones óptimas. Tercero, aumenta la confianza del consumidor al proporcionar transparencia sobre las condiciones de manejo de los productos. Finalmente, facilita el cumplimiento normativo al asegurar que los productos se almacenen y transporten según las regulaciones de seguridad alimentaria.

2.3.4 Desafíos

A pesar de sus beneficios, la implementación de sensores IoT también presenta desafíos. La integración de sensores en la infraestructura existente puede requerir inversiones significativas y cambios en los procesos operativos. Además, la gestión y el análisis de grandes volúmenes de datos generados por los sensores requieren soluciones avanzadas de procesamiento y almacenamiento de datos. Sin embargo, con el desarrollo continuo de tecnologías y la reducción de costos, estos desafíos pueden superarse, permitiendo una adopción más amplia de la sensorización IoT en la trazabilidad.

Capítulo III: Tecnologías

3.1. Tecnologías utilizadas

Para el desarrollo del proyecto se hace uso de diferentes tecnologías, las cuales han hecho posible la creación de la aplicación. A continuación, se enumeran cada una de ellas y su respectiva función durante la implementación.

JavaScript

Es un lenguaje de programación interpretado, de alto nivel y dinámico, que se ejecuta en el lado del cliente. Fue desarrollado originalmente por Netscape como un medio para agregar programas dinámicos y ejecutables a las páginas web. Permite a los desarrolladores crear contenido interactivo en las páginas web, como animaciones, control de multimedia, formularios dinámicos y aplicaciones complejas. JavaScript es un lenguaje basado en prototipos, multiparadigma, que soporta estilos de programación funcional, orientada a objetos e imperativa. Es una parte fundamental del núcleo de tecnologías web junto con HTML y CSS, y es utilizado en una variedad de frameworks y bibliotecas como React, Angular y Vue.js. Además, gracias a entornos de ejecución como Node.js, también puede ser utilizado en el desarrollo del lado del servidor.

Arduino

Es una plataforma de hardware y software de código abierto que permite a los usuarios crear proyectos de electrónica interactivos. La plataforma consta de una serie de placas de desarrollo (microcontroladores) que se pueden programar utilizando el lenguaje de programación Arduino, basado en Wiring, y el entorno de desarrollo Arduino IDE. Las placas de Arduino pueden leer entradas (como luz en un sensor, un dedo en un botón, o un mensaje de Twitter) y convertirlas en salidas (activando un motor, encendiendo un LED, publicando algo en línea).

Hedera

Hedera Hashgraph es una tecnología de libro mayor distribuido que utiliza un método llamado "gossip about gossip" y gráficos acíclicos dirigidos (DAG) para procesar transacciones de manera rápida y eficiente. Esto permite que las transacciones se realicen con alta capacidad de procesamiento y costos de transacción bajos,proporciona una plataforma segura y descentralizada para realizar transacciones y almacenar datos. Es capaz de manejar muchas transacciones por segundo, lo que la hace ideal para aplicaciones que requieren alta velocidad y seguridad, como pagos, contratos inteligentes y otras aplicaciones empresariales.

Solidity

Es un lenguaje de programación de alto nivel, orientado a contratos inteligentes, que se

utiliza principalmente para desarrollar aplicaciones descentralizadas (DApps) en la plataforma de blockchain de Ethereum. Fue creado por Gavin Wood, Christian Reitwiessner, Alex Beregszaszi y varios ex desarrolladores principales de Ethereum. Solidity es un lenguaje de tipo estático, similar en su sintaxis a JavaScript y C++, lo que facilita su aprendizaje para los desarrolladores familiarizados con estos lenguajes. Los contratos inteligentes escritos en Solidity son autoejecutables y se ejecutan en la Ethereum Virtual Machine (EVM).

Node-RED

Es una herramienta de programación basada en flujo, diseñada para la integración de dispositivos, API y servicios en línea. Desarrollado originalmente por IBM, Node-RED está construido sobre Node.js y permite a los usuarios conectar nodos (representando funciones) en un editor visual basado en web para crear flujos de datos que interactúan entre sí. Esta herramienta es especialmente útil para aplicaciones de Internet de las Cosas (IoT), automatización del hogar y la integración de sistemas. Los flujos de Node-RED pueden ser exportados e importados fácilmente, lo que facilita la colaboración y el intercambio de soluciones. La comunidad de Node-RED ha desarrollado una amplia gama de nodos adicionales que permiten la integración con numerosos servicios, plataformas y dispositivos. Su interfaz intuitiva y su capacidad para gestionar flujos complejos hacen de Node-RED una herramienta poderosa para desarrolladores y no desarrolladores por igual.

Node.js

Node.js es un entorno de ejecución de JavaScript del lado del servidor que permite a los desarrolladores ejecutar código JavaScript en el servidor. Fue creado por Ryan Dahl en 2009, con la intención de construir aplicaciones de red escalables y de alto rendimiento. Node.js utiliza un modelo de E/S no bloqueante y orientado a eventos, lo que lo hace muy eficiente y adecuado para aplicaciones que requieren un alto rendimiento y capacidad de respuesta, como servidores web, aplicaciones en tiempo real, y servicios API. lenguaje de programación, simplificando el proceso de desarrollo y mantenimiento.

MQTT

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería ligero y de publicación-suscripción, diseñado para dispositivos con recursos limitados y redes con ancho de banda limitado. Fue creado por IBM y Arlen Nipper en 1999, y se ha convertido en un estándar de facto para la comunicación en aplicaciones de Internet de las Cosas (IoT). MQTT es extremadamente eficiente en el uso del ancho de banda y en el manejo de conexiones intermitentes, lo que lo hace ideal para dispositivos IoT que requieren una comunicación fiable y eficiente. El protocolo utiliza un modelo de publicación-suscripción, donde los dispositivos pueden publicar mensajes en "temas" específicos y suscribirse a esos temas para recibir mensajes. Un broker MQTT gestiona la distribución de mensajes entre los clientes.

Capítulo IV: Desarrollo del proyecto

4.1. Diseño

4.1.1. Diseño de conexiones electrónicas

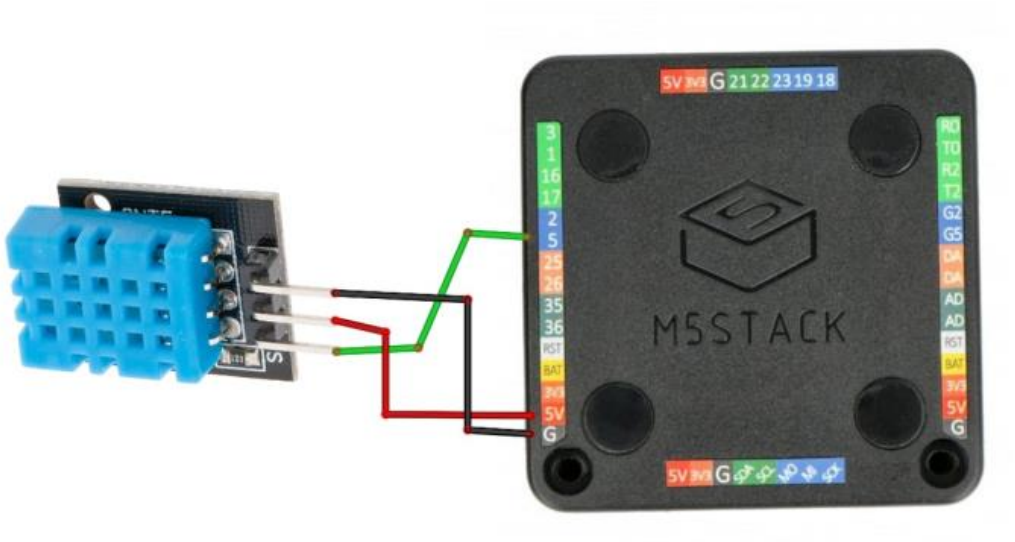

Figura 4: Dibujo de conexiones

Tabla 1: Diseño de conexiones electrónicas

Sensor DH11	M5STACK
VCC	5V
GND	GND
DATA	GPIO 22

Conexión del Sensor al M5Stack

1. **VCC (Voltaje de Alimentación)**
   - **Conexión:** VCC del sensor a 5V del M5Stack.
   - **Motivo:** El pin VCC del sensor debe conectarse al pin de alimentación de 5V del M5Stack para proporcionar la energía necesaria para el funcionamiento del sensor. El pin de 5V en el M5Stack suministra un voltaje constante y adecuado que el sensor necesita para operar correctamente.

2. **GND (Tierra)**
   - **Conexión:** GND del sensor a GND del M5Stack.
   - **Motivo:** El pin GND del sensor debe conectarse al pin GND del M5Stack para completar el circuito eléctrico. La conexión a tierra es esencial para estabilizar el circuito y proporcionar una referencia de voltaje común. Esto asegura que la diferencia de potencial (voltaje) sea medida

correctamente y que el sensor funcione adecuadamente.

3. **DATA (Línea de Datos)**
    - **Conexión:** DATA del sensor a GPIO 22 del M5Stack.
    - **Motivo:** El pin DATA del sensor se conecta a uno de los pines GPIO (General Purpose Input/Output) del M5Stack, en este caso, el GPIO 22. Este pin se utiliza para la comunicación de datos entre el sensor y el microcontrolador. La elección del pin GPIO 22 puede estar determinada por varios factores:
        - **Disponibilidad:** GPIO 22 puede estar libre y disponible para su uso sin conflictos con otras conexiones o funciones del dispositivo.
        - **Funciones Especiales:** Algunos pines GPIO tienen funciones especiales o características que pueden ser útiles para ciertos tipos de sensores, como interrupciones o capacidades de comunicación específica.
        - **Conveniencia Física:** La ubicación física de GPIO 22 en la placa puede ser conveniente para la conexión física del sensor, minimizando la longitud de los cables y reduciendo el desorden.

4.1.2. Diseño de flujo de datos

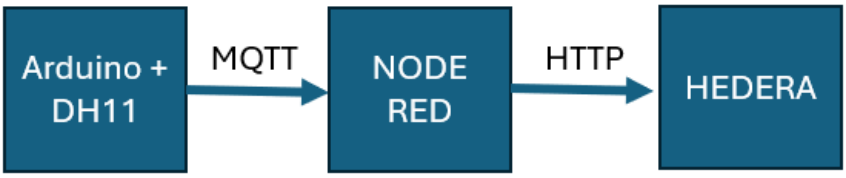

Figura 5: Flujo de datos

En el flujo de datos del proyecto, primero se capturan los datos de humedad y temperatura con un M5Stack y un sensor DHT11. Estos datos se envían a través del protocolo MQTT.

Luego, en Node-RED, un suscriptor MQTT recoge los datos y, mediante el protocolo HTTP, se introducen en funciones de JavaScript. Posteriormente, estos datos se envían a un contrato programado en Solana y se certifican en el sistema Hashgraph de Hedera

4.2. Implementación

4.2.1. Smart Contract en Solidity

Un Smart Contract (contrato inteligente) en Solidity es un programa que se ejecuta en la blockchain de Ethereum. Estos contratos son auto-ejecutables con las condiciones del acuerdo entre comprador y vendedor directamente escritas en líneas de código. Los contratos inteligentes permiten realizar transacciones y acuerdos confiables entre partes anónimas sin la necesidad de un sistema legal, evitando el tiempo y el costo de intermediarios.

El smart contract proporcionado se llama “baseDeDatos” y tiene la finalidad de almacenar y recuperar datos relacionados con humedad y temperatura.

```
pragma solidity ^0.8.0;

contract baseDeDatos {
    struct data{
        string humedad;
        int temperatura;
    }
    mapping(uint => data) private registro;
    event registroAnyadido (uint idRegister, string  humedad, int temperatura);

    function addRegister(uint _idRegister,string memory  _humedad, int _temperatura) public {    infinite gas
        registro[_idRegister] = data(_humedad,_temperatura);
        emit registroAnyadido(_idRegister, _humedad, _temperatura);

    }
    function getRegister(uint _idRegister) public view returns (string memory ,int) {    infinite gas
            require(bytes(registro[_idRegister].humedad).length>0,"el registro no existe");

            return (registro[_idRegister].humedad,registro[_idRegister].temperatura);
    }
}
```

Figura 6: Código Smart Contract Solidity

4.2.1.1. Explicación de los componentes y funciones

1. **Estructura data**: Define un tipo de dato llamado data que contiene dos campos: humedad (de tipo string) y temperatura (de tipo int).
2. **Mapping registro**: Declara una variable registro que es un mapping (mapeo) de un entero uint a una estructura data. Este mapping se utiliza para almacenar los registros de datos asociados a un identificador único (ID).
3. **Evento registroAnyadido**:Declara un evento llamado registroAnyadido que se emite cuando se añade un nuevo registro. Este evento incluye el ID del registro, la humedad y la temperatura.
4. **Función addRegister**:
    a. Esta función pública permite añadir un nuevo registro. Recibe tres parámetros: el identificador del registro (_idRegister), la humedad (_humedad) y la temperatura (_temperatura).

    b. Almacena los datos recibidos en el mapping registro utilizando _idRegister como clave.

c. Emite el evento registroAnyadido con los datos del nuevo registro.

5. **Función getRegister**:

   a. Esta función pública y de vista (no modifica el estado) permite recuperar un registro almacenado.
   b. Recibe un parámetro: el identificador del registro (_idRegister).
   c. Verifica que el registro exista comprobando que la longitud del string humedad sea mayor que 0.
   d. Si el registro existe, retorna los valores de humedad y temperatura asociados a _idRegister.
   - Si el registro no existe, lanza un error con el mensaje "el registro no existe"

4.2.2. Prueba de Smartcontract en Remix IDE

Remix IDE es una herramienta basada en web proporcionada por Ethereum que permite a los desarrolladores escribir, compilar, desplegar y depurar smart contracts en el lenguaje de programación Solidity. Remix ofrece una interfaz de usuario intuitiva y una variedad de características que facilitan el desarrollo de contratos inteligentes, lo que lo convierte en una herramienta esencial para desarrolladores de Ethereum.

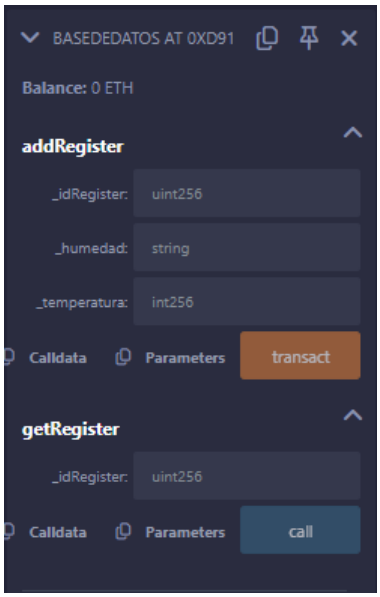

Figura 7: Remix IDE prueba Smartcontract

4.2.2.1. Funcionalidades principales de Remix IDE

1. **Editor de Código**:
   - Permite escribir y editar código en Solidity con resaltado de sintaxis y autocompletado.
   - Ofrece la capacidad de gestionar múltiples archivos dentro del entorno del proyecto.

2. **Compilador**:
    - Compila el código de Solidity y proporciona información detallada sobre errores y advertencias.
    - Genera el bytecode y la interfaz de aplicación binaria (ABI) necesarios para desplegar el contrato en la blockchain.

3. **Despliegue y Ejecución**:

    - Permite desplegar contratos inteligentes en varias redes de prueba (testnets) y en la red principal de Ethereum.
    - Proporciona una interfaz para interactuar con las funciones del contrato desplegado.
    -
4. **Depuración**:

    - Ofrece herramientas de depuración para analizar el comportamiento de los contratos inteligentes.
    - Permite realizar un seguimiento paso a paso de la ejecución del contrato para identificar y corregir errores.

Ejemplo Práctico

1. Añadir un Registro:
    - _idRegister: 1
    - _humedad: "50%"
    - _temperatura: 22

Haz clic en "transact" para añadir este registro al contrato.

2. Recuperar un Registro:

    - _idRegister: 1

Haz clic en "call" para recuperar los valores de humedad y temperatura para el registro con identificador 1.

4.2.3. Arduino

```
#include <M5Stack.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
```

Figura 8: Librerías Arduino

<M5Stack.h>

- **Descripción:** Esta librería es específica para el desarrollo con el módulo

M5Stack, una plataforma de desarrollo basada en ESP32 que incluye pantalla, botones y otros periféricos integrados.

- **Función:** Proporciona funciones y métodos para interactuar con el hardware del M5Stack, como la pantalla LCD, los botones, y otros componentes integrados. Facilita la programación y el control de estos componentes sin tener que manejar directamente los registros del hardware.
- **Uso:** Se utiliza para inicializar y controlar los diversos elementos del M5Stack, permitiendo, por ejemplo, mostrar información en la pantalla o leer el estado de los botones.

<WiFi.h>

- **Descripción:** Librería estándar para la conexión WiFi en placas basadas en ESP32.
- **Función:** Permite conectar el dispositivo a una red WiFi, gestionar conexiones y realizar operaciones de red como HTTP y MQTT. Proporciona métodos para conectarse a redes WiFi, manejar eventos de red y obtener información de la red.
- **Uso:** Utilizada para establecer una conexión a Internet, lo que es crucial para aplicaciones que requieren comunicación remota o acceso a servicios en la nube.

<PubSubClient.h>

- **Descripción:** Librería para el protocolo MQTT (Message Queuing Telemetry Transport).
- **Función:** Facilita la implementación de la comunicación basada en el protocolo MQTT, que es ligero y adecuado para dispositivos IoT. Permite al dispositivo publicar (enviar) y suscribirse (recibir) mensajes a/de un broker MQTT.
- **Uso:** Utilizada para enviar y recibir datos de manera eficiente entre el dispositivo y otros sistemas a través de MQTT, un protocolo de mensajería popular en aplicaciones de IoT por su bajo consumo de ancho de banda y recursos.

<DHT.h>

- **Descripción:** Librería para sensores de temperatura y humedad DHT (DHT11, DHT22, etc.).
- **Función:** Proporciona métodos para interactuar con los sensores DHT, facilitando la lectura de la temperatura y la humedad ambiental. Incluye funciones para inicializar el sensor y obtener lecturas de temperatura y humedad de manera precisa.
- **Uso:** Se utiliza para leer datos de sensores DHT conectados al dispositivo. Esta librería simplifica el proceso de comunicación con el sensor y la interpretación de sus datos.

```
// Configuración del broker MQTT
const char* mqtt_server = "194.30.15.135";
const int mqtt_port = 1883;
const char* mqtt_user = "user";
const char* mqtt_password = "user";
```

Figura 9: Configuración MQTT

const char* mqtt_server = "194.30.15.135";

- **Descripción:** Esta línea define la dirección IP del broker MQTT al que se conectará el dispositivo.
- **Función:** mqtt_server es un puntero constante a una cadena de caracteres que contiene la dirección IP del servidor MQTT. En este caso, la dirección IP es 194.30.15.135.
- **Uso:** Esta dirección es utilizada por la librería MQTT (PubSubClient) para establecer la conexión con el broker. Cambiar esta dirección permite conectar a diferentes brokers MQTT.

const int mqtt_port = 1883;

- **Descripción:** Esta línea define el número de puerto que el cliente MQTT utilizará para conectarse al broker.
- **Función:** mqtt_port es una constante entera que representa el puerto del broker MQTT. El puerto 1883 es el puerto estándar para conexiones MQTT sin cifrar.
- **Uso:** La librería MQTT utiliza este número de puerto para establecer la conexión al broker en la dirección IP especificada.

const char* mqtt_user = "user";

- **Descripción:** Esta línea define el nombre de usuario que se usará para autenticarse con el broker MQTT.
- **Función:** mqtt_user es un puntero constante a una cadena de caracteres que contiene el nombre de usuario. En este caso, es "user".
- **Uso:** Si el broker MQTT requiere autenticación, este nombre de usuario se enviará durante el proceso de conexión. Este valor debe coincidir con el nombre de usuario configurado en el broker MQTT.

const char* mqtt_password = "user";

- **Descripción:** Esta línea define la contraseña que se usará para autenticarse con el broker MQTT.
- **Función:** mqtt_password es un puntero constante a una cadena de caracteres que contiene la contraseña. En este caso, es "user".

- **Uso:** Si el broker MQTT requiere autenticación, esta contraseña se enviará junto con el nombre de usuario durante el proceso de conexión. Este valor debe coincidir con la contraseña configurada en el broker MQTT.

```
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  Serial.print("Humedad: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperatura: ");
  Serial.print(t);
  Serial.println(" *C");

  // Convertir los datos a cadenas de caracteres
  char tempString[8];
  char humString[8];
  dtostrf(t, 1, 2, tempString);
  dtostrf(h, 1, 2, humString);

  // Publicar los datos en los topics de MQTT
  client.publish("sensor/temperatura", tempString);
  client.publish("sensor/humedad", humString);
```

Figura 10: Main code

Verificación de Conexión MQTT:

- El código comprueba si el cliente MQTT está conectado. Si no lo está, llama a la función reconnect() para reestablecer la conexión.
- Usa client.loop(); para mantener activa la conexión y procesar mensajes entrantes.

Lectura de Datos del Sensor:

- Lee los valores de humedad y temperatura del sensor DHT.
- Verifica si las lecturas son válidas. Si no lo son, imprime un mensaje de error y finaliza la ejecución de la función loop().

Impresión de Datos:

- Muestra los valores de humedad y temperatura en el monitor serial.

Publicación de Datos en MQTT:

- Convierte los valores de temperatura y humedad a cadenas de caracteres.
- Publica estos valores en los tópicos sensor/temperatura y sensor/humedad del

broker MQTT.

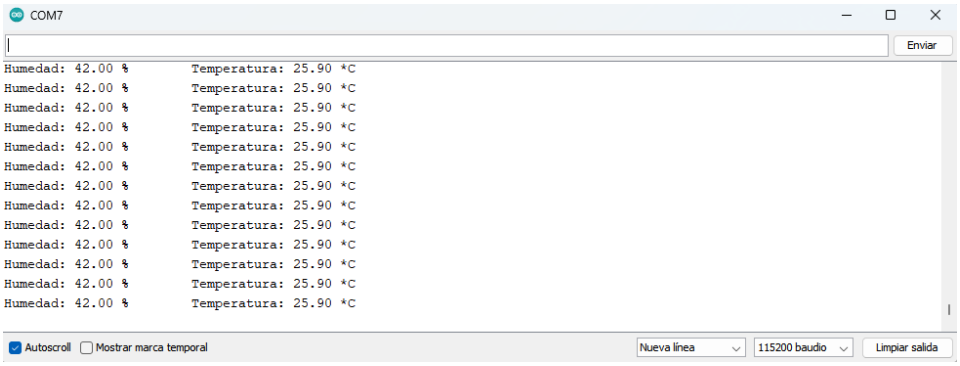

Figura 11: Output Arduino

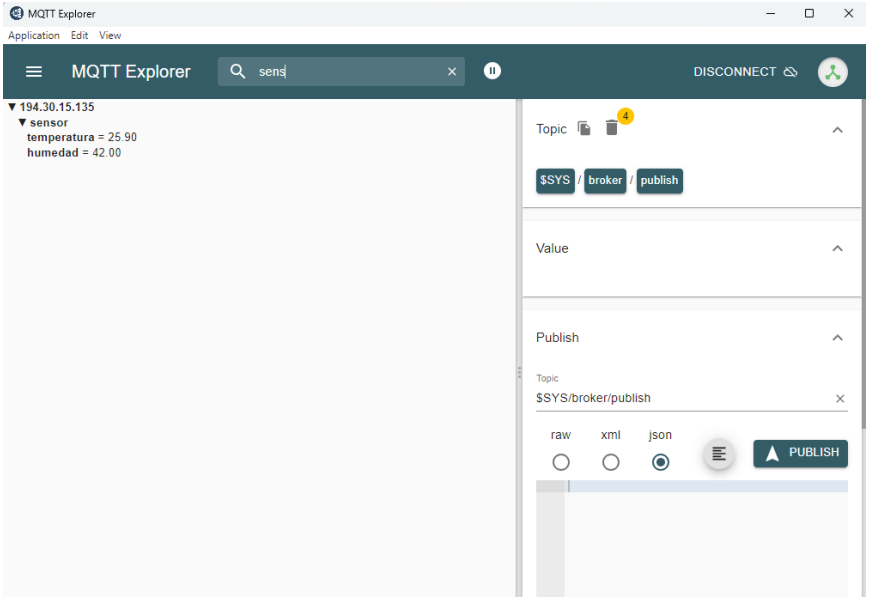

Figura 12: MQTT Explorer

Verificación de que está publicando los datos en el topic y bróker correctos.

4.2.4. Node-Red

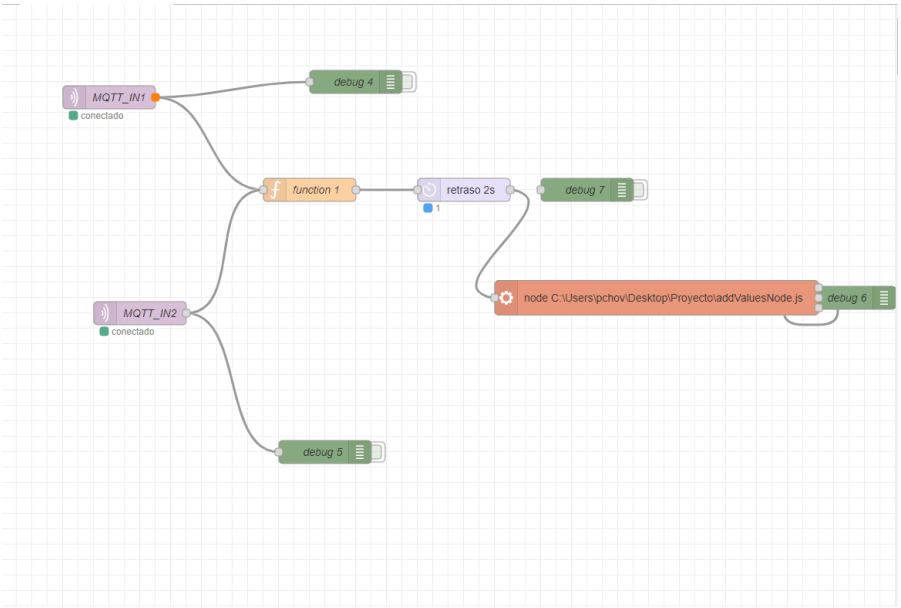

Figura 13: Diseño de Node-Red

El diseño de Node-RED incluye los siguientes componentes:

1. **Dos suscriptores MQTT:**
   - **Suscriptor de Humedad:** Se suscribe al tópico MQTT específico para la humedad.
   - **Suscriptor de Temperatura:** Se suscribe al tópico MQTT específico para la temperatura.
2. **Función de Procesamiento:**
   - Ambos suscriptores están conectados a una función que procesa los datos recibidos. La función se detalla más adelante.
3. **Delay de 2 Segundos:**
   - Un nodo de retraso (delay) de 2 segundos se coloca después de la función de procesamiento para espaciar la ejecución.
4. **Ejecución de Script para Blockchain:**
   - Tras el delay, se ejecuta un script que agrega los valores de humedad y temperatura en la blockchain.

Este flujo asegura que los datos de los sensores sean procesados, temporizados adecuadamente, y finalmente registrados en la blockchain para su seguimiento y verificación.

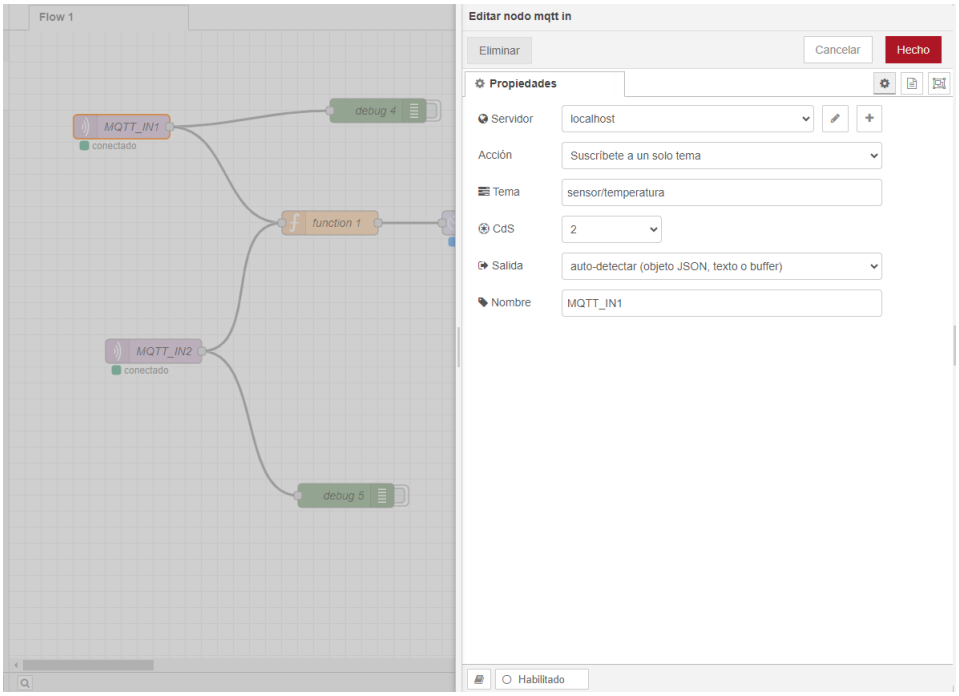

Figura 14: Suscriptor MQTT temperatura Node-Red

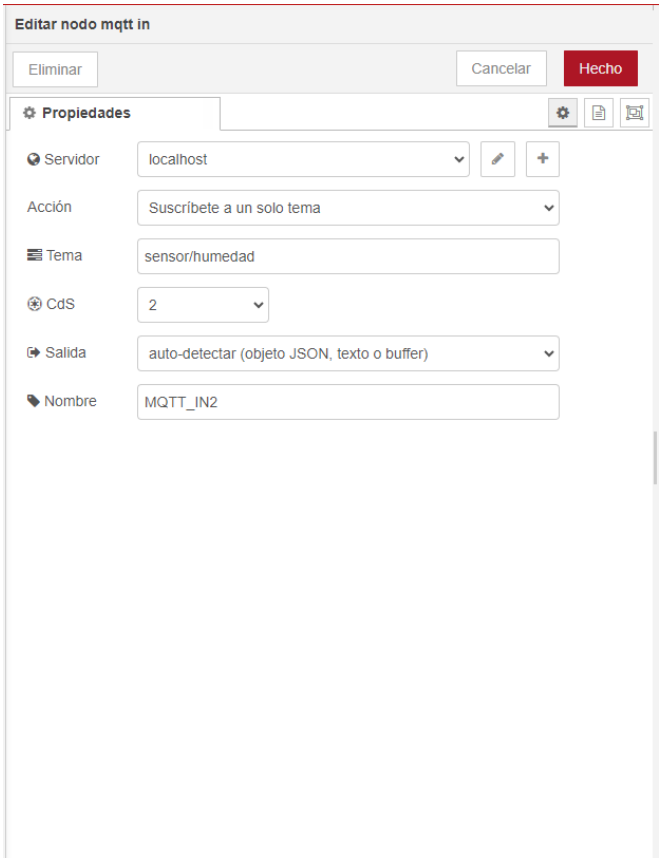

Figura 15: Suscriptor MQTT humedad Node-Red

```
// Inicializa el contexto de flujo si aún no existe
flow.set('temperatura', flow.get('temperatura') || null);
flow.set('humedad', flow.get('humedad') || null);

// Verifica cuál entrada MQTT está llegando
if (msg.topic === 'sensor/temperatura') {
    flow.set('temperatura', String(msg.payload)); // Casteo a
} else if (msg.topic === 'sensor/humedad') {
    flow.set('humedad', String(msg.payload)); // Casteo a str
}

// Recupera los valores de ambos MQTT
var temperatura = flow.get('temperatura');
var humedad = flow.get('humedad');

// Si ambos valores están disponibles, combina y envía el nue
if (temperatura !== null && humedad !== null) {
    msg.payload = `${temperatura} ${humedad}`;

    // Resetea los valores en el contexto de flujo
    flow.set('temperatura', null);
    flow.set('humedad', null);

    return msg;
}

// Si uno de los valores aún no está disponible, no devuelve
return null;
```

Figura 16: Script function Node-Red

La función implementada en Node-RED tiene como objetivo manejar y procesar datos provenientes de dos tópicos MQTT diferentes, uno para temperatura y otro para humedad. A continuación, se detalla el proceso que sigue esta función:

1. **Inicialización del Contexto de Flujo:**
    - La función comienza asegurándose de que las variables de contexto para temperatura y humedad estén inicializadas. Si estas variables no existen, se inicializan con un valor nulo (null). Esto es necesario para evitar errores en la recuperación de datos más adelante.
2. **Verificación de la Entrada MQTT:**
    - La función verifica cuál de los dos tópicos MQTT está recibiendo datos en ese momento. Si el tópico es de temperatura, el valor del mensaje recibido se guarda en la variable de contexto temperatura. De manera similar, si el tópico es de humedad, el valor del mensaje se guarda en la variable de contexto humedad.
3. **Recuperación de Valores:**
    - Una vez que se han almacenado los datos en las variables de contexto, la función recupera los valores actuales de temperatura y humedad.
4. **Combinación y Envío de Datos:**
    - La función comprueba si ambos valores, temperatura y humedad, están disponibles (es decir, no son nulos). Si ambos valores están presentes, se combinan en un solo mensaje.
    - El mensaje combinado se configura como el nuevo payload del mensaje de salida.
5. **Reinicio de Variables:**
    - Después de enviar el mensaje combinado, las variables de contexto para temperatura y humedad se reinician a null para prepararse para la

próxima ronda de datos entrantes.

6. **Manejo de Datos Incompletos:**
    - Si uno de los valores (temperatura o humedad) aún no está disponible, la función no envía ningún mensaje y simplemente termina. Esto asegura que sólo se envíen mensajes completos y válidos.

4.2.4.1. Objetivo de la función

El propósito principal de esta función es asegurarse de que los datos de temperatura y humedad se envíen juntos como un solo mensaje, proporcionando así una actualización completa y coherente cada vez que ambos valores estén disponibles. Este enfoque es crucial para aplicaciones donde la correlación entre temperatura y humedad es importante y se necesita evitar el procesamiento de datos parciales o incompletos.
Esta funcionalidad es especialmente útil en sistemas de monitoreo ambiental, donde los valores de temperatura y humedad deben ser monitoreados y registrados simultáneamente para análisis precisos y acciones basadas en condiciones ambientales.

4.2.4.2. Archivo Contract.js

```javascript
const {
    Client, ContractCreateTransaction, FileCreateTransaction, Hbar, PrivateKey} = require("@hashgraph/sdk");

const fs = require("fs");

async function deployContract(){
    const client = Client.forTestnet();

    const accountID = "0.0.4515903";
    const PrivateKeyHex = "0xd36e3b1a69b19071146b158789200c275e0c713545387a2561066773bd78bb4f";
    const privateKey = PrivateKey.fromStringED25519(PrivateKeyHex);


    client.setOperator(accountID,privateKey);
    const bytecode = fs.readFileSync("trybin.bin");


    const fileCreateTx = new FileCreateTransaction()
        .setContents(bytecode)
        .setMaxTransactionFee(new Hbar(10)); // Ajusta la tarifa de transacción según sea necesario
    const submitFileCreateTx = await fileCreateTx.execute(client);
    const fileCreateRx = await submitFileCreateTx.getReceipt(client);
    const bytecodeFileId = fileCreateRx.fileId;//Hash de vuelta

    const contractInstantiateTx = new ContractCreateTransaction()
        .setBytecodeFileId(bytecodeFileId)
        .setGas(100000) // Ajusta el gas según sea necesario
        .setMaxTransactionFee(new Hbar(10)); // Ajusta la tarifa de transacción según sea necesario
    const submitInstantiateTx = await contractInstantiateTx.execute(client);
    const contractInstantiateRx = await submitInstantiateTx.getReceipt(client);
    const contractId = contractInstantiateRx.contractId;

    console.log(`Contrato desplegado con éxito en el ID: ${contractId}`);

    // Finaliza el proceso con éxito
    process.exit(0);
}

deployContract().catch(error => {
    console.error("Error al desplegar el contrato:", error);
    process.exit(1); // Finaliza el proceso con un código de error
});
```

Figura 17: Deploy Smart Contract code

El proceso de despliegue de un smart contract en la red de prueba de Hedera Hashgraph incluye varias etapas clave, desde la configuración inicial hasta el manejo de errores. A continuación, se explica cada parte del proceso y se proporciona información sobre cómo obtener las credenciales necesarias (Account ID, PrivateKey, y PrivateKeyHex).

Importación de Módulos y Configuración Inicial

1. **Importaciones y Configuración Inicial:**
    - Se importan los módulos necesarios del SDK de Hedera Hashgraph y el módulo para manejo de archivos. Estos módulos permiten conectar con la red de Hedera, crear y gestionar transacciones de contratos y archivos, y manipular archivos que contienen el bytecode del contrato.

e

Credenciales Necesarias

1. **Account ID (ID de Cuenta)**

    - **Descripción:** El Account ID es un identificador único para una cuenta en la red de Hedera Hashgraph, con un formato típico como 0.0.4515903.

    - **Obtención:**
        - **Creación de Cuenta:** Al crear una cuenta en la red principal o en una red de prueba de Hedera, se asigna un Account ID único.
        - **Servicios de Terceros:** Usando servicios como MyHbarWallet o el portal de desarrolladores de Hedera, obtendrás un Account ID al registrar una nueva cuenta.
        - **Registro en Testnet:** Para trabajar en una testnet, puedes registrarte en el Portal de Desarrolladores de Hedera y obtener tu Account ID.

2. **PrivateKey (Clave Privada)**

    - **Descripción:** La PrivateKey es una clave criptográfica utilizada para firmar transacciones y acceder de manera segura a tu cuenta de Hedera.

    - **Obtención:**
        - **Creación de Cuenta:** Cuando creas una cuenta en Hedera, se te proporciona una clave privada que debes guardar en un lugar seguro.
        - **Servicios de Terceros:** Servicios como MyHbarWallet te proporcionan una clave privada al crear tu cuenta.

3. **PrivateKeyHex (Clave Privada en Formato Hexadecimal)**
    - **Descripción:** La PrivateKeyHex es la representación hexadecimal de tu clave privada.

    - **Conversión:** La clave privada obtenida puede ser convertida a formato hexadecimal utilizando herramientas específicas o funciones dentro del SDK de Hedera.

Función Asíncrona para Desplegar el Contrato

1. **Configuración del Cliente:**

    - Se crea una instancia del cliente para la red de prueba (Testnet) de Hedera, permitiendo ejecutar transacciones en un entorno de prueba. Luego, se configuran el ID de la cuenta y la clave privada, estableciendo la cuenta operadora que firmará las transacciones.

2. **Lectura del Bytecode del Contrato:**

- Se lee el bytecode del contrato desde un archivo binario. Este bytecode es esencialmente el programa del contrato que se desplegará en la red de Hedera.

3. **Creación y Ejecución de la Transacción de Archivo:**

    - Se crea una transacción de archivo para subir el bytecode del contrato a la red de Hedera. Esto implica establecer el contenido del archivo y ajustar la tarifa de transacción. Una vez ejecutada la transacción, se obtiene un recibo que incluye el ID del archivo creado, necesario para los siguientes pasos.

4. **Creación y Ejecución de la Transacción de Contrato:**

    - Utilizando el ID del archivo obtenido anteriormente, se crea una transacción de contrato. Se especifica el ID del archivo que contiene el bytecode, la cantidad de gas necesaria para la ejecución, y se ajusta la tarifa de transacción. La transacción se ejecuta en la red, y se obtiene un recibo que incluye el ID del contrato desplegado.

5. **Manejo de Errores:**

    - Se implementa una captura de errores para manejar cualquier problema que pueda ocurrir durante el despliegue del contrato. Si ocurre un error, se imprime un mensaje adecuado y el proceso se termina de manera segura.

4.2.4.3. Archivo addValuesNode.js

addValuesNode.js

C: > Users > pchov > Desktop > Proyecto > addValuesNode.js > addUserToContract

```
const {
    Client,
    ContractExecuteTransaction,
    Hbar,
    ContractId,
    PrivateKey,
    ContractFunctionParameters,
} = require("@hashgraph/sdk");

async function addUserToContract(temperatura, humedad) {
    const client = Client.forTestnet();

    // Datos de la cuenta HBAR
    const accountID = "0.0.4515903";
    const PrivateKeyHex = "0xd36e3b1a69b19071146b15878920 0c275e0c713545387a2561066773bd78bb4f";
    const privateKey = PrivateKey.fromStringED25519(PrivateKeyHex);

    client.setOperator(accountID, privateKey);

    // El Contract ID es fijo según la modificación solicitada
    const contractId = "0.0.4612570";

    // El ID de usuario es fijo o puede ser otro argumento si lo necesitas
    const idRegistro = 1; // Puedes cambiar esto si necesitas un ID de usuario dinámico

    const contractIdObj = ContractId.fromString(contractId);
    const gas = 100000; // Ajusta el gas según sea necesario

    try {
        // Prepara los parámetros de la función
        const functionParameters = new ContractFunctionParameters()
            .addUint256(parseInt(idRegistro, 10))
            .addString(humedad)
            .addInt256(parseInt(temperatura, 10));
        // Crea y envía la transacción para llamar a la función del contrato
        const contractExecTx = await new ContractExecuteTransaction()
            .setContractId(contractIdObj)
            .setGas(gas)
            .setFunction("addRegister", functionParameters)
            .setMaxTransactionFee(new Hbar(10)) // Ajusta según sea necesario
            .execute(client);

        const receipt = await contractExecTx.getReceipt(client);
        console.log(`Estado de la transacción: ${receipt.status.toString()}`);
    } catch (error) {
        console.error("Error al ejecutar la transacción:", error);
    } finally {
        // Finalizamos el proceso
        process.exit(0);
    }
}

const args = process.argv.slice(2);
const temperatura = args[0];
const humedad = args[1];

if (temperatura === undefined || humedad === undefined) {
    console.error("Error: los argumentos de temperatura o humedad no están definidos.");
    process.exit(1);
}

console.log(`Temperatura: ${temperatura}, Humedad: ${humedad}`);

addUserToContract(temperatura, humedad).catch(error => {
    console.error("Error:", error);
    process.exit(1); // Salida por error
});
```

Figura 18: Añadir valores code

Utiliza el SDK de Hedera Hashgraph para interactuar con un contrato inteligente desplegado en la red de prueba. El objetivo del script es añadir registros de temperatura y humedad al contrato inteligente. A continuación se describe cada sección del script y su función:

Importación de Módulos

1. **Importaciones:**

    - Se importan varios componentes del SDK de Hedera Hashgraph que son necesarios para crear y ejecutar transacciones de contrato, manejar claves privadas y definir parámetros de función.

Función Asíncrona Principal

2. **Función Principal addUserToContract:**
    - Esta función asíncrona toma dos argumentos (temperatura y humedad) y realiza una serie de pasos para agregar estos valores al contrato inteligente en Hedera.

3. **Configuración del Cliente:**
    - Se crea una instancia del cliente para la red de prueba (Testnet) de Hedera, utilizando las credenciales de cuenta y clave privada que se establecen previamente.

4. **Identificación del Contrato y Parámetros:**
    - **Contract ID:** Se define el ID del contrato al que se desea interactuar.
    - **ID de Registro:** Se establece un ID de usuario fijo para el registro; puede modificarse si se necesita un ID dinámico.
    - **Gas:** Se ajusta el gas necesario para la transacción del contrato.

5. **Preparación de Parámetros de Función:**
    - Se preparan los parámetros necesarios para llamar a la función del contrato (addRegister) utilizando el objeto ContractFunctionParameters. Estos parámetros incluyen el ID del registro, la humedad y la temperatura.

6. **Creación y Ejecución de la Transacción de Contrato:**
    - Se crea una transacción de contrato (ContractExecuteTransaction) configurando el ID del contrato, el gas, la función a llamar y los parámetros de la función.
    - La transacción se ejecuta y se obtiene un recibo para verificar el estado de la transacción.

7. **Manejo de Errores:**
    - El bloque try-catch se utiliza para manejar cualquier error que pueda ocurrir durante la ejecución de la transacción. Si ocurre un error, se imprime un mensaje de error y el proceso se termina adecuadamente.

8. **Manejo de Argumentos de Línea de Comandos:**
    - Se extraen los argumentos de línea de comandos para temperatura y humedad. Si no se proporcionan estos argumentos, se imprime un mensaje de error y se termina el proceso.

9. **Llamada a la Función Principal:**
    - Se llama a la función addUserToContract pasando los valores de temperatura y humedad. Si ocurre un error durante la ejecución, se maneja adecuadamente e imprime un mensaje de error.

4.2.4.4. Archivo getValues.js

```
const {
    Client,
    ContractExecuteTransaction,
    ContractCallQuery,
    Hbar,
    ContractId,
    PrivateKey,
    ContractFunctionParameters, // Asegúrate de incluir esto en tus importaciones
} = require("@hashgraph/sdk");

// Función para solicitar información al usuario
const askQuestion = (question) => {
    const readline = require("readline").createInterface({
        input: process.stdin,
        output: process.stdout
    });

    return new Promise(resolve => readline.question(question, ans => {
        readline.close();
        resolve(ans);
    }));
}

const getValues = async () => {
    const client = Client.forTestnet();

        //Datos de la propia HBAR ACCOUNT!
        const accountID = "0.0.4515903";
        const PrivateKeyHex = "0xd36e3b1a69b19071146b158789280c275e0c713545387a2561066773bd78bb4f";
        const privateKey = PrivateKey.fromStringED25519(PrivateKeyHex) //<-- Escoger la encriptación correcta :)

        client.setOperator(accountID, privateKey);

    client.setOperator(accountID, privateKey);

    // El Contract ID es fijo según la modificación solicitada
    const contractId = "0.0.4612570";
    const idRegistro = await askQuestion("Ingrese el ID de Registro a consultar: ");

    try {
        // Crear y configurar la consulta para llamar a la función del contrato
        const query = new ContractCallQuery()
            .setContractId(contractId)
            .setGas(100000) // Ajusta el gas según sea necesario
            .setFunction("getRegister", new ContractFunctionParameters().addUint256(parseInt(idRegistro)))
            .setQueryPayment(new Hbar(2)); // Ajusta el pago de la consulta según sea necesario, más bajo para pruebas

        // Ejecutar la consulta
        const result = await query.execute(client);

        // Decodificar el resultado de la llamada
        const humedad = result.getString(0);
        const temperatura = result.getInt256(1); // Asegúrate de usar el índice correcto para obtener la nota

        console.log(`Humedad: ${humedad}, Temperatura: ${temperatura}`);
        // Finalizamos el proceso correctamente después de imprimir los resultados
        process.exit(0);
        return { humedad, temperatura };
    } catch (error) {
        console.error("Error retrieving data from contract:", error);
        throw error;
    }

};

getValues().catch(error => {
    console.error("Error:", error);
    process.exit(1); // Salida por error
});
```

Figura 19: Obtener valores code

El script proporcionado está escrito en Node.js y utiliza el SDK de Hedera Hashgraph para interactuar con un contrato inteligente desplegado en la red de prueba. El objetivo del script es leer registros de temperatura y humedad al contrato inteligente. A continuación se describe cada sección del script y su función:

Importación de Módulos

1. **Importaciones:**

   Se importan varios componentes del SDK de Hedera Hashgraph que son necesarios para crear y ejecutar transacciones de contrato, manejar claves privadas y definir parámetros de función.

Función Asíncrona Principal

2. **Función Principal getValues:**
   - Esta función asíncrona se encarga de obtener valores de temperatura y humedad desde un contrato inteligente en Hedera, utilizando el ID de un registro específico.
     .

3. **Configuración del Cliente:**
   Se crea una instancia del cliente para la red de prueba (Testnet) de Hedera, utilizando las credenciales de cuenta y clave privada que se establecen previamente

4. **Identificación del Contrato y Parámetros:**
   - **Contract ID:** Se define el ID del contrato al que se desea interactuar.
   - **ID de Registro:** Se establece un ID de usuario fijo para el registro; puede modificarse si se necesita un ID dinámico.
   - **Gas:** Se ajusta el gas necesario para la transacción del contrato.

5. **Preparación de Parámetros de Función:**
   - Se preparan los parámetros necesarios para llamar a la función del contrato (getRegister) utilizando el objeto ContractFunctionParameters. Estos parámetros incluyen el ID del registro.

6. **Creación y Ejecución de la Transacción de Contrato:**
   - Se crea una consulta de contrato (ContractCallQuery) configurando el ID del contrato, el gas, la función a llamar y los parámetros de la función..

   - La transacción se ejecuta y se obtiene un recibo para verificar el estado de la transacción.

7. **Manejo de Errores:**
   - El bloque try-catch se utiliza para manejar cualquier error que pueda ocurrir durante la ejecución de la transacción. Si ocurre un error, se imprime un mensaje de error y el proceso se termina adecuadamente.

8. **Llamad a la función principal:**
   - Se llama a la función `getValues` para iniciar el proceso de consulta. Si ocurre un error durante la ejecución, se maneja adecuadamente e imprime un mensaje de error.

```
PS C:\Users\pchov\desktop\proyecto> node .\getValues.js
Ingrese el ID de Registro a consultar: 1
Humedad: 54, Temperatura: 27
```

Figura 20: getValues.js output

4.3. Validación del proceso

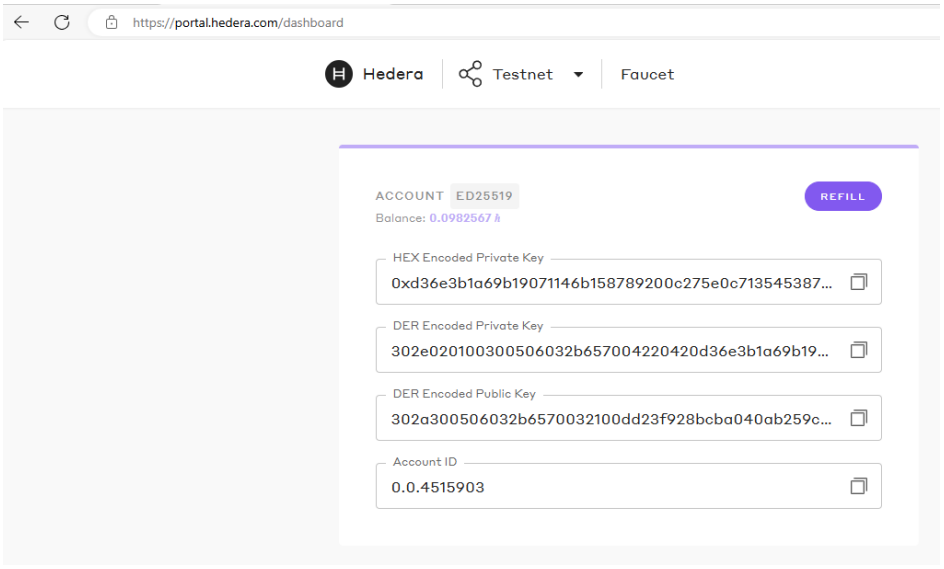

Figura 21: Datos de la cuenta utilizados como ejemplo, importados del portal de Hedera (Cuenta ED25519)

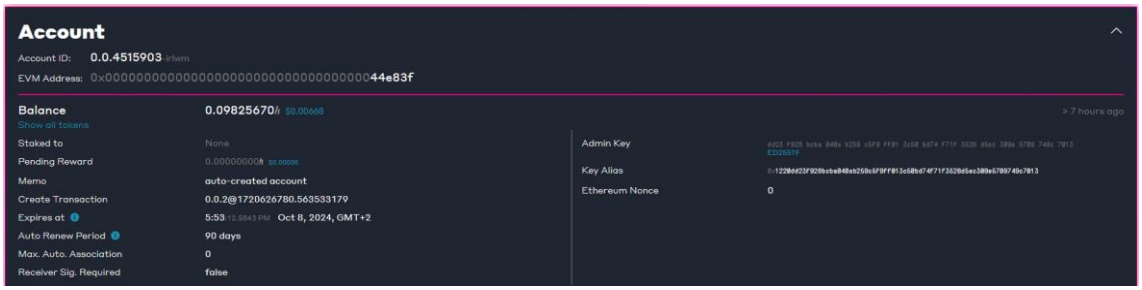

Figura 22: Cuenta Hedera

Visión detallada de una cuenta en la red de Hedera Hashgraph, incluyendo su balance, información de staking, detalles de la creación de la cuenta, y configuraciones de seguridad. Estos detalles son cruciales para la gestión de la cuenta, permitiendo realizar transacciones, verificar el historial y asegurar que la cuenta esté configurada y operando correctamente en la red de Hedera.

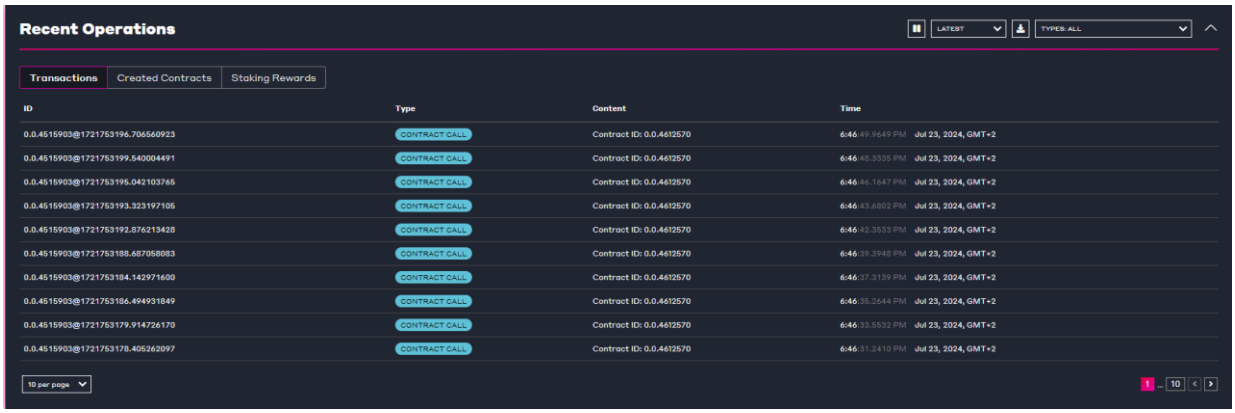

Figura 23: Operaciones recientes en Hedera

El panel de "Recent Operations" proporciona una visión integral de las actividades recientes de una cuenta en la red de Hedera Hashgraph. Permite a los usuarios revisar y gestionar sus transacciones, contratos creados y recompensas de staking. Esta funcionalidad es crucial para mantener un seguimiento detallado y transparente de todas las acciones realizadas por la cuenta, facilitando tanto la auditoría como la gestión eficiente de los recursos en la red de Hedera.

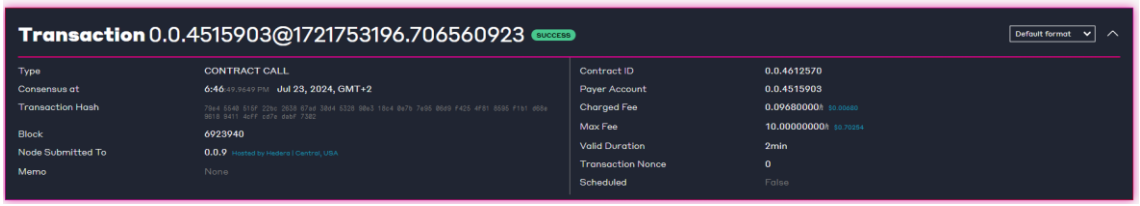

Figura 24: Detalles de la transacción Hedera

El panel de "Transaction Details" en Hedera Hashgraph proporciona una vista integral de todos los aspectos de una transacción específica. Incluye detalles esenciales como el tipo de transacción, la confirmación del consenso, el hash de la transacción, los costos asociados, y la validez. Esta información es crucial para los desarrolladores, auditores y usuarios que necesitan verificar y entender el comportamiento y los costos de las transacciones en la red de Hedera.

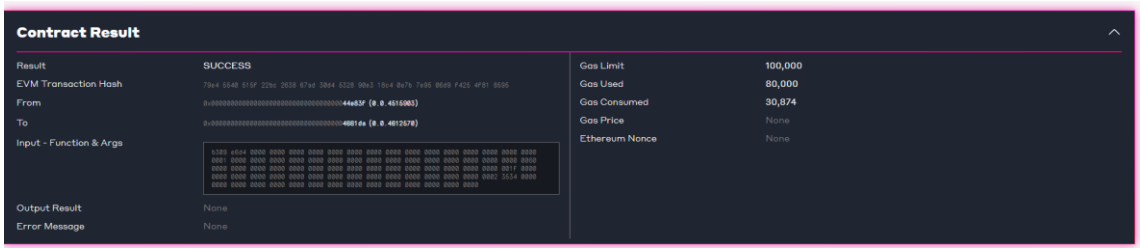

Figura 25: Resultado de la transacción del contrato Hedera

El panel “Contract Result” muestra los resultados de una transacción de contrato en la red de Hedera Hashgraph. Este panel proporciona detalles importantes sobre la ejecución de la transacción. A continuación, se describe cada sección y lo que representa:

Información del Resultado del Contrato

1. **Result (Resultado):** indica el estado final de la transacción. En este caso, el resultado es SUCCESS, lo que significa que la transacción se completó con éxito.

2. **EVM Transaction Hash:** el hash de la transacción en la Máquina Virtual de Ethereum (EVM). Este hash es un identificador único para la transacción, que puede ser usado para rastrear y verificar la transacción en la red. Su función es proporcionar un medio para referenciar y buscar la transacción específica en la blockchain.

3. **From (De):** dirección de la cuenta que envió la transacción. En este caso, es la cuenta 0.0.4515903. Indica quién inició la transacción.

4. **To (A):** dirección del contrato que recibió la transacción. En este caso, es el contrato 0.0.4612570. Indica el destino de la transacción.

5. **Input - Function & Args (Entrada - Función y Argumentos):** muestra los datos de entrada enviados al contrato, incluyendo la función llamada y los argumentos proporcionados. Este es un conjunto de datos en formato hexadecimal que representa los parámetros de la función llamada. Su función es proporcionar detalles sobre qué función del contrato fue invocada y con qué argumentos.

6. **Gas Limit (Límite de Gas):** es el límite máximo de gas asignado para esta transacción, en este caso 100,000. Define la cantidad máxima de gas que la transacción está permitida consumir.

7. **Gas Used (Gas Utilizado):** se trata de la cantidad de gas realmente utilizada por la transacción, en este caso 80,000. Muestra cuántas unidades de gas se necesitaron para completar la transacción.

8. **Gas Consumed (Gas Consumido):** muestra el gas consumido durante la ejecución de la transacción, en este caso 30,874. Indica el consumo efectivo de

gas para llevar a cabo la ejecución del contrato.

9. **Gas Price (Precio del Gas):** no se especifica en la captura de pantalla, pero usualmente representa el costo por unidad de gas. Determina el costo total de la transacción basada en el gas utilizado.

10. **Ethereum Nonce:** utilizado en Ethereum para evitar la duplicación de transacciones y asegurar el orden correcto de ejecución. No se especifica en la captura de pantalla.

El panel de "Contract Result" en Hedera Hashgraph proporciona una vista detallada de la ejecución de una transacción de contrato, incluyendo el estado del resultado, los detalles de la transacción, los parámetros de entrada y el consumo de gas. Esta información es crucial para los desarrolladores y usuarios que necesitan verificar el comportamiento y la eficiencia de sus contratos inteligentes, asegurando que las transacciones se realicen de manera correcta y eficiente en la red de Hedera.

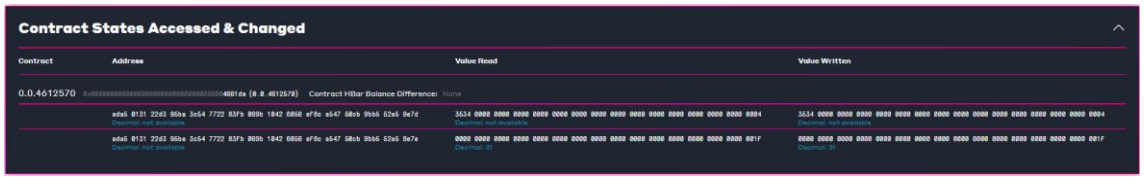

Figura 26: Estado del contrato en Hedera

El panel de "Contract States Accessed & Changed" en Hedera Hashgraph. Este panel proporciona una visión detallada sobre los estados del contrato que han sido leídos y escritos durante la ejecución de una transacción de contrato. A continuación, se describe cada sección y lo que representa:

Información General del Contrato

1. **Contract (Contrato):** muestra el ID del contrato involucrado en la transacción. En este caso, el contrato es 0.0.4612570. Identifica cuál contrato inteligente fue accedido y modificado durante la transacción.

2. **Address (Dirección):** la dirección del contrato en la red de Hedera Hashgraph. Proporciona una referencia específica al contrato en la blockchain.

Valores Leídos y Escritos

1. **Values Read (Valores Leídos):** muestra los valores del estado del contrato que fueron leídos durante la ejecución de la transacción. Además, indica qué datos almacenados en el contrato fueron accedidos para la lectura. Estos valores se muestran en formato hexadecimal.

2. **Values Written (Valores Escritos):** muestra los valores del estado del contrato que fueron modificados o escritos durante la transacción e indica qué datos almacenados en el contrato fueron actualizados. Estos valores también se muestran en formato hexadecimal.

Detalles Específicos

1. **Estado del Contrato:**
    - **Dirección y Valores:** Cada fila debajo del ID del contrato muestra una dirección específica del contrato y los valores leídos y escritos en esa dirección.
    - **Formato Hexadecimal:** Los valores se presentan en formato hexadecimal para precisión y compatibilidad con el manejo de datos en la blockchain.

2. **Contract State Difference (Diferencia en el Estado del Contrato):** indica si hubo diferencias significativas en los estados del contrato antes y después de la transacción. Y ayuda a identificar los cambios específicos realizados en el estado del contrato debido a la transacción.

El panel de "Contract States Accessed & Changed" en Hedera Hashgraph proporciona información detallada sobre cómo los estados del contrato inteligente fueron accedidos y modificados durante la ejecución de una transacción. Esto es crucial para los desarrolladores y auditores que necesitan entender el impacto exacto de las transacciones en los contratos inteligentes, asegurando transparencia y precisión en la gestión de estados en la blockchain de Hedera. Este nivel de detalle ayuda a verificar la integridad y el correcto funcionamiento de los contratos desplegados.

Figura 27: Eventos Hedera

Este panel proporciona detalles sobre los eventos que ocurrieron durante la ejecución de una transacción de contrato inteligente. A continuación, se describe cada sección y lo que representa:

Información de la Transacción

- **Transaction Hash (Hash de la Transacción):** el hash de la transacción es un

identificador único generado para la transacción específica. Este hash se utiliza para rastrear y verificar la transacción en la red. Permite a los usuarios buscar y referenciar la transacción en la blockchain.

Información del Bloque

- **Block (Bloque):** indica el número de bloque en el que la transacción fue incluida. En este caso, el bloque es 49793419. Proporciona un contexto temporal y secuencial para la transacción dentro de la blockchain.

Información del Contrato

- **Address (Dirección):** muestra la dirección del contrato inteligente involucrado en la transacción. En este caso, es 0.0.4612570, e identifica el contrato específico que emitió los eventos.

Logs de Eventos

- **Logs (Registros):** contiene los datos emitidos por el contrato durante la transacción. Estos logs son generalmente utilizados para emitir eventos específicos desde el contrato inteligente. Los logs pueden incluir diversos tópicos (en este caso, Topic 0), que ayudan a categorizar y estructurar los datos emitidos.
    - **Ejemplo de Log:** En la captura, se muestra un log hexadecimal que representa los datos emitidos por el evento.

El panel de "Events" en Hedera Hashgraph proporciona una vista detallada de los eventos generados durante la ejecución de una transacción de contrato inteligente. Incluye información esencial como el hash de la transacción, el bloque en el que se incluyó la transacción, la dirección del contrato que emitió los eventos, y los registros detallados de los eventos. Esta información es crucial para los desarrolladores y auditores que necesitan entender y verificar el comportamiento de los contratos inteligentes en la red de Hedera, asegurando transparencia y precisión en la gestión de eventos y transacciones en la blockchain.

Capítulo V: Resultados y Discusión

5.1. Evaluación de la implementación

En el desarrollo de este Trabajo de Fin de Grado (TFG), hemos logrado certificar datos de temperatura y humedad provenientes de un dispositivo Arduino utilizando una tecnología de libro mayor distribuido (DLT). Este proceso asegura la integridad, transparencia y seguridad de los datos en la cadena de suministro de productos alimentarios.

La certificación de datos es un paso crucial para garantizar que la información registrada sea inmutable y verificable. En este proyecto, hemos utilizado Hedera para certificar los datos recolectados.

Captura de Datos: Los datos de temperatura y humedad se capturan utilizando un sensor DHT11 conectado a un M5Stack, un potente microcontrolador compatible con Arduino. Este dispositivo recolecta datos en tiempo real, monitoreando las condiciones ambientales de los productos alimentarios.

Los datos capturados se envían y certifican en la red Hedera Hashgraph. Este proceso se realiza a través de un protocolo seguro y eficiente, garantizando que la información se almacene de manera inmutable.

Certificación mediante Hedera Hashgraph: Hedera Hashgraph proporciona una plataforma segura y descentralizada para la certificación de datos. La tecnología de "gossip about gossip" y gráficos acíclicos dirigidos (DAG) utilizada por Hedera permite que las transacciones se procesen rápidamente y con alta eficiencia, asegurando que los datos de temperatura y humedad sean accesibles y verificables en todo momento.

La implementación de getValues.js permite recuperar la información certificada directamente desde la red Hedera. Este script de JavaScript se conecta a la API de Hedera, consulta los datos certificados y los recupera para su análisis y uso posterior. Esto asegura que los datos presentados sean auténticos y no hayan sido alterados desde su certificación.

La integración de sensores IoT con tecnologías DLT como Hedera Hashgraph permite una trazabilidad precisa y segura de las condiciones ambientales de los productos alimentarios. Certificar los datos de temperatura y humedad asegura que toda la información registrada sea inmutable y verificable, proporcionando una garantía adicional de calidad y seguridad en la cadena de suministro.

Este proyecto demuestra cómo las tecnologías avanzadas pueden integrarse para mejorar la gestión y la trazabilidad en la industria alimentaria, promoviendo prácticas más responsables y eficientes.

Capítulo VI: Conclusiones y trabajo futuro

6.1. Conclusiones

La conclusión de este proyecto subraya la importancia crítica de la tecnología blockchain en la trazabilidad de productos alimentarios. Respaldada por numerosas instituciones, la tecnología blockchain ofrece una solución altamente escalable una vez que se ha diseñado e implementado para un proyecto específico.

Una de sus principales ventajas es la eliminación de errores humanos, lograda mediante la automatización de la captura de datos con sensores IoT. Este enfoque no solo facilita la recolección de datos, sino que también permite automatizar su análisis, garantizando así la transparencia y la integridad de los datos recopilados.

En una industria tan crucial como la alimentaria, que tiene un impacto directo en todos los seres humanos, la integración de la tecnología blockchain es esencial. La capacidad de asegurar una cadena de suministro transparente y segura desde el origen del producto hasta el consumidor final es un avance significativo. La implementación de blockchain en la trazabilidad de productos alimentarios proporciona una garantía adicional de calidad y seguridad, aspectos fundamentales para mantener la confianza de los consumidores y cumplir con las regulaciones sanitarias.

El hecho de que los supermercados líderes en los Estados Unidos ya hayan adoptado esta tecnología para la trazabilidad de sus productos alimentarios es un testimonio de su viabilidad y beneficios. Estos establecimientos han demostrado que la blockchain puede mejorar significativamente la eficiencia operativa y la transparencia en la cadena de suministro. Al eliminar intermediarios y proporcionar un registro inmutable de todas las transacciones y movimientos del producto, se reduce el riesgo de fraudes y errores, y se aumenta la confianza en la integridad de los productos alimentarios.

Por lo tanto, la integración de blockchain en la trazabilidad de productos alimentarios no solo es una innovación tecnológica, sino una necesidad imperativa para mejorar la seguridad alimentaria, la transparencia y la eficiencia en la industria. Este proyecto destaca cómo la adopción de tecnologías avanzadas puede transformar y mejorar la gestión de la cadena de suministro, ofreciendo beneficios tangibles y sostenibles tanto para los productores como para los consumidores.

6.2. Trabajo futuro

5.2.1 Mejoras:

En futuros desarrollos, se plantea la implementación de un sistema basado en códigos QR que permitirá al usuario final, específicamente al consumidor de alimentos, acceder a toda la información registrada en la blockchain. Este código QR estará asociado a cada producto alimenticio y, mediante su escaneo con un dispositivo móvil, el consumidor podrá obtener datos detallados sobre el origen, procesamiento, transporte y condiciones de almacenamiento del producto. La transparencia y trazabilidad proporcionadas por la blockchain asegurarán la autenticidad y la integridad de la información, incrementando la confianza del consumidor en la calidad y seguridad de los productos alimenticios adquiridos.

Capitulo VII: Github

Aquí podemos encontrar todo el código

Capitulo VIII: Bibliografía

[1] Food Retail & Shoppers. (2021, noviembre 10). Navidul implanta la tecnología blockchain en la trazabilidad de sus jamones. Food Retail & Shoppers. URL: https://www.foodretail.es/food/navidul-blockchain-jamon-tecnologia-alimentacion_0_1610838933.html

[2] Santander. (s.f.). Smart Contracts. URL: https://www.santander.com/es/stories/smart-contracts

[3] GitHub. URL: https://github.com/

[4] JavaScript. URL: https://www.javascript.com/

[5] Hedera. URL: https://docs.hedera.com/hedera/

[6] Solidity. URL: https://docs.soliditylang.org/

[7] Node-RED. (s.f.). Creating your first flow. URL: https://nodered.org/docs/tutorials/first-flow

[8] Electromaker. (s.f.). Weather Station using DHT11 Sensor & M5Stack Core ESP32. URL: https://www.electromaker.io/project/view/weather-station-using-dht11-sensor-amp-m5stack-core-esp32