ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Maestría en Sistemas de Información Gerencial

“DISEÑO Y DESARROLLO DE UNA APLICACIÓN WEB QUE PERMITIRÁ CONOCER LA TRAZABILIDAD DE LOS INSUMOS USADOS EN FUMIGACIÓN, EMPLEANDO LA TECNOLOGÍA DE CONTABILIDAD DISTRIBUIDA (BLOCKCHAIN) EN UNA EMPRESA BANANERA”

TRABAJO DE TITULACIÓN

PREVIO A LA OBTENCIÓN DEL TÍTULO DE:

MAGISTER EN SISTEMAS DE INFORMACIÓN GERENCIAL

Presentado por:

CÉSAR ALBERTO VILLARROEL SAMANIEGO

DANIEL GONZALEZ CORNEJO

Guayaquil – Ecuador

2020

AGRADECIMIENTO

Agradecemos a Dios por que nos dio fuerzas para culminar esta etapa profesional. A nuestras familias por el amor, comprensión durante el tiempo que estuvimos estudiando y elaborando el presente trabajo. A nuestros padres, compañeros de trabajo y amigos que nos recordaban este pendiente, además de nuestro tutor por su apoyo en las revisiones y recomendaciones.

DEDICATORIA

El presente trabajo se lo dedicamos a nuestras familias por su apoyo y comprensión, que supieron valorar el tiempo invertido en este trabajo, en lugar de estar con ellos, tiempo que será compensado en un futuro con mucho amor. Se lo dedicamos a nuestros padres que supieron inculcarnos responsabilidad y perseverancia para cumplir con las metas establecidas.

TRIBUNAL DE SUSTENTACIÓN

Msg. Lenin Freire Cobo

DIRECTOR MSIG

Msg. Juan Carlos García

DIRECTOR DEL PROYECTO DE

GRADUACIÓN

Msg. Omar Maldonado Dañin

MIEMBRO DEL TRIBUNAL

DECLARACIÓN EXPRESA

La responsabilidad del contenido de este Trabajo de Titulación nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral.

César Alberto Villarroel Samaniego

Daniel González Cornejo

RESUMEN

El presente documento trata sobre la implementación de un sistema WEB para una empresa de plantación de banano, mismo que permite registrar y gestionar el plan de la aplicación de insumos usados en la fumigación, apoyándose en la tecnología de contabilidad distribuida mejor conocida como Blockchain (cadena de bloques), y usando como FrontEnd tecnología Web con ASP.Net.

El desarrollo tomó en cuenta los requerimientos funcionales y técnicos, tanto para la construcción de la aplicación WEB, como de los contratos inteligentes (definiciones de las estructuras que se van a intercambiar durante las transacciones), la lógica de negocio definida para el intercambio de los contratos y la definición de las transacciones en la cadena de bloques, reemplazando al registro e intercambio manual de información.

El desarrollo de esta aplicación Web permitió disponibilizar a terceros involucrados en el proceso de registro y seguimiento del plan de fumigación, y que apoyado en la tecnología Blockchain garantizó la trazabilidad, la confidencialidad y confiabilidad de la información, así como también la seguridad al acceso y la integridad de la información almacenada.

Palabras clave: Blockchain, Aplicaciones Web, Desarrollo web, ASP.NET, contratos inteligentes, HyperLedger Composer, HyperLedger Fabric. .

ÍNDICE GENERAL

ABREVIATURAS Y SIMBOLOGÍA

API	Interface para Programación de Aplicaciones
AS-IS	Estado o situación actual
ASP	Páginas de Servidor Activas (en inglés Active Server Pages)
CPU	Unidad de procesamiento central (en inglés Central Processing Unit)
CRUD	Crear, leer, actualizar y borrar (en inglés Create, read, update and delete)
EAL5	Nivel 5 de confianza en la evaluación de seguridad (en inglés Evaluation Assurance Level 5)
ESPOL	Escuela Superior Politécnica del Litoral
EU	Unión Europea
FIPS	Estándares federales de procesamiento de información (en inglés Federal Information Processing Standard)
Gb	Gigabytes
HR	Humedad Relativa
HSM	Módulo de Hardware de seguridad (en inglés Hardware Security Module)
HTML	Lenguaje de marcas de hipertexto (en inglés HyperText Markup Language)
HTTP	Protocolo de transferencia de hipertexto (en inglés Hypertext Transfer Protocol)
IBM	International Business Machines
JS	JavaScript

REST	Transferencia de estado representacional (Representational state transfer)
SDK	Herramientas de desarrollo de software (en inglés Software Development Kit)
SKU	Código de artículo (en inglés Stock-keeping unit)
SQL	Lenguaje de consultas estructuradas (en inglés Structure Query Language)
SSD	Dispositivos de Estado Sólido (en inglés Solid State Drives)
TO-BE	Estado deseado o solución planteada
USD	Dólares de Estados Unidos (United States Dollar)
vCPU	Unidad virtual de procesamiento central (en inglés virtual Central Processing Unit)
VPC	Procesador de Núcleos Virtual (en inglés Virtual processor core)
WEB	Nombramiento de una red informática, generalmente asociada a las aplicaciones que corren en un navegador sobre Internet

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El objetivo principal del presente trabajo de titulación es diseñar e implementar una aplicación Web que permita conocer la trazabilidad de los insumos usados en la fumigación, empleando la tecnología de Blockchain para una empresa bananera.

En el Capítulo 1, se explica el detalle del problema a resolver, la situación actual del proceso de registro de adquisición y aplicación de los productos de fumigación, identificando cada uno de los actores involucrados, la propuesta tecnológica que pretende resolver los problemas actuales, mencionando tanto los objetivos generales como específicos, finalizando el capítulo con la metodología a aplicar durante este trabajo para la planificación, diseño y desarrollo.

En el Capítulo 2, se especifica el fundamento teórico sobre el cual se basa la solución a diseñar e implementar, se describe y justifica a la tecnología de Blockchain como una plataforma de registro seguro y distribuido, se describe de forma breve otros conceptos asociados al uso de esta tecnología, su integración con aplicaciones Web, consideraciones sobre seguridad de información involucradas en este tipo de aplicaciones, y otras posibilidades de uso de la integración de estas tecnologías.

En el Capítulo 3, a partir de una revisión breve de la situación actual sobre las necesidades, se analiza un caso de uso para utilizar Blockchain, se profundiza en los requerimientos técnicos y funcionales que debe cumplir la solución, se levanta los

criterios de aceptación de la aplicación Web integrada al Blockchain, y se incluye resumen del alcance del proyecto.

En el Capítulo 4 se analiza los costos y las características de escalabilidad de la tecnología Blockchain escogida dentro del proyecto, así como el esquema de detallado de las herramientas para la administración y desarrollo relacionados con esa tecnología. Finalmente, con el registro de los requerimientos funcionales y técnicos, se detalle el diseño de las interfaces Web, los componentes transaccionales, los contratos inteligentes de la cadena, los eventos que se notifican a los usuarios, los casos de prueba y la lista de requerimientos que se confirmarán durante las pruebas.

El Capítulo 5 se describe el desarrollo y las pruebas de la cadena de bloques que se diseñó para el proyecto, así como la integración y pruebas de la creación y modificación de los contratos inteligentes con la aplicación Web diseñada con la plataforma de Blockchain.

En el Capítulo 6 se hace una revisión y chequeo del cumplimiento de los criterios de aceptación, las conclusiones y las lecciones aprendidas durante el desarrollo de este proyecto.

CAPÍTULO 1

1. GENERALIDADES

1.1 Antecedentes

El problema trata de una empresa bananera que dentro de su proceso de adquisición y compra de los productos para fumigación para su plantación, es afectado por una falta de transparencia entre la información que se intercambia entre los proveedores, compradores, auditores y su personal.

Comúnmente, cada una de estas entidades intercambia información a través de correo electrónico con un archivo adjunto en Excel de los productos, certificaciones y fechas de fumigación, o mediante respaldos físicos impresos. Durante este intercambio de información los archivos o documentos corren el riesgo de ser modificados y actualizados sin llevar un registro adecuado de los cambios, esta situación puede provocar la incorrecta aplicación de las fumigaciones, afectando la calidad del producto final durante la cosecha provocando un rechazo del producto durante las

visitas de inspección y en el peor de los casos llevando un mal producto al consumidor final.

1.2 Descripción del problema

Con referencia a la situación actual generada por este proceso manual, se enumeran a continuación, los principales problemas existentes en la gestión de información, entre las entidades involucradas en el proceso de gestión de productos e insumos para la fumigación de la plantación bananera:

- Falta de un correcto seguimiento a las transacciones.
- No hay controles que permitan proteger de los datos, de una modificación fraudulenta.
- Existe un alto riesgo con el acceso de información a las entidades externa sobre los aplicativos internos pueden comprometer los datos de la empresa.
- El flujo de la información entre las entidades involucradas no mantiene un orden ni un registro sobre su seguimiento, lo cual no permite realizar una trazabilidad del proceso.
- El almacenamiento en base de datos local no permite distribución de información a entidades externas.

- Actualmente la empresa no garantiza la disponibilidad de la información 24/7 debido a los altos costo de inversión que se requiere y no tiene la infraestructura adecuada.

- Dificultad de dar acceso a terceras personas a una base de datos convencional local, que cumplan las políticas y controles para garantizar la disponibilidad e integridad de información.

1.3 Solución propuesta

Se plantea crear un aplicativo Web a través del cual se integre el uso de los contratos inteligentes con la tecnología de Blockchain, como una solución al problema de la gestión en el seguimiento a las transacciones (trazabilidad), la protección de los datos e integración sobre las entidades involucradas en la producción del banano. Sólo ciertos roles dentro del sistema de intercambio de transacciones en el proceso podrán tener acceso a la información o agregar información en la cadena. La unidad básica donde se almacena la información y que es distribuida dentro de la solución se la conoce como Ledger o Libro Mayor.

Con esta tecnología, los datos una vez validados no pueden modificarse, y se cierra el registro conocido como Closed Ledger (Libro Mayor Cerrado), por lo que servirá como un gran almacenamiento distribuido histórico, del cual pueden obtenerse otra información tales como: el origen del producto, lugar, fecha de compra, fecha de elaboración, estado del producto en cada etapa, verificación por auditores, hasta el uso del producto. Al incluir esta

información en la nube, se permitirá el acceso a consultas desde la aplicación web.

La arquitectura propuesta permite de forma opcional la implementación de la plataforma Hyperledger de IBM para el soporte de Blockchain, y su integración contra la aplicación WEB mediante API Rest basada en NodeJS. La aplicación WEB se desarrollará en ASP .Net. El servidor tiene sistema operativo Windows Server 2016 y es alquilado a una empresa americana que garantiza el 99.9% de disponibilidad.

A continuación, se muestra un gráfico que representa el caso de uso donde se aplicaría la tecnología de Blockchain, respecto en el suministro de productos de fumigación que hay en la producción de banano.

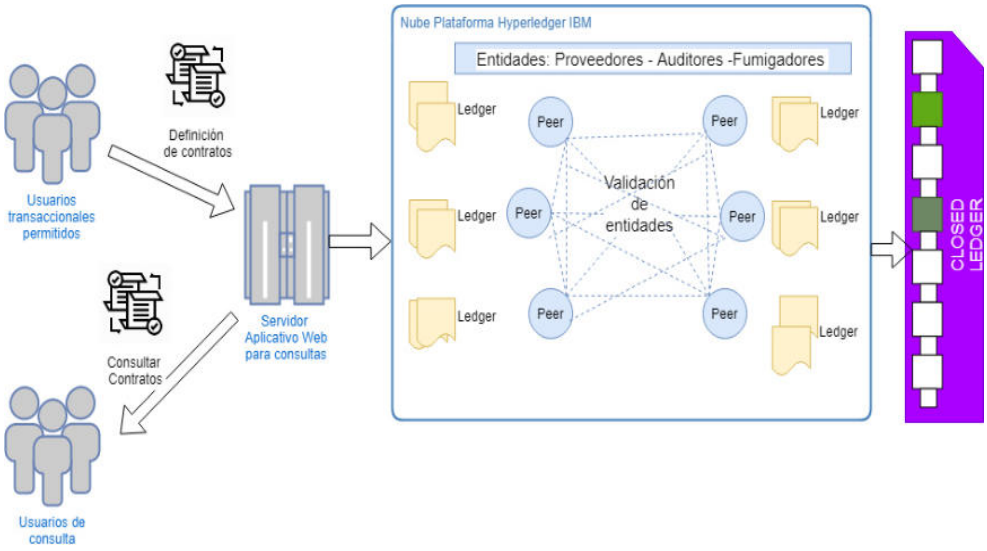


Figura 1.1 Arquitectura para el control de los productos de fumigación en la producción de banano

Los roles que se indican en el gráfico, sus funciones y la interacción que tienen con el Blockchain a través del sistema web se indican a continuación:

Tabla 1. Roles que se identifican dentro de la solución Web propuesta

Proveedor	Posterior a la compra de los insumos registra en el sistema WEB integrado con Blockchain los productos e insumos que se acordó entregar, datos tales como: SKU del producto, Fecha de Compra, Fecha de Elaboración y Valor.
Auditor de Materiales de Fumigación	Realiza las Pruebas, asociando el SKU en el aplicativo WEB con la fecha de Revisión, registrando su aceptación o rechazo en el sistema.
Empresa Fumigadora	Ingresa su recepción del producto, SKU , Nombre de la Finca, Fechas de Fumigación, Lugar establecido de la Finca y Lote.
Gerente de producción	Ingresa el plan semanal de fumigación.
Administrador financiero de la plantación bananera	Consulta información de los productos y la ejecución de los trabajos de fumigación sus novedades y el cierre exitoso o no de los mismos.

Entendiendo el ciclo de la información dentro de la cadena de Blockchain y su acceso a través de una aplicación Web, se logra obtener los siguientes beneficios:

- Acceso único a la información a través de una capa de autorización en la nube, mecanismo usado por la aplicación Web.
- La información existe dentro de la cadena de contratos para futuras consultas, sin necesidad de llevarla de forma local por la empresa, lo

cual implica un ahorro en costos de mantenimiento por infraestructura para su almacenamiento.

- Cada entidad es responsable de su parte dentro del proceso sin intervención de un tercero.
- El uso de la plataforma permite que se cierre el contrato sólo cuando todas las partes están de acuerdo, lo cual garantiza la confianza en el contenido.
- Con la tecnología Blockchain y el uso de contratos inteligentes se espera reducir los costos de intermediarios innecesarios por reducción de seguimiento y control.
- En la cadena de valor de la empresa nos enfocamos en el proceso de cosecha la cual incluye el uso de productos para la fumigación, es por esto que no entra en este alcance la integración con el proceso de compras ni con la revisión de plan de fumigación aun cuando al futuro dependiendo del éxito de la solución se pueda realizar una integración a los mismos.
- Como un beneficio adicional, el desarrollo del proyecto pretende ser un marco de referencia del uso de las Blockchain en otros modelos de negocio diferentes al financiero dentro del Ecuador.

1.4 Objetivo General

Demostrar el uso de la plataforma Blockchain, para gestionar la información de los productos químicos utilizados en la fumigación de una plantación de banano a través de un sistema basado en una aplicación Web.

1.5 Objetivos Específicos

- Levantar la información sobre las necesidades de almacenamiento seguro, que garantice la autenticación de las entidades que participan en el suministro de los productos que se aplican en la producción de banano.
- Diseñar contratos inteligentes para gestionar la adquisición y registro del uso de los productos de fumigación añadiendo transparencia y trazabilidad.
- Mejorar el proceso de auditoría y revisión de errores o causas en el proceso de adquisición de los insumos y productos en la producción de banano con el uso de una cadena de contratos inteligentes.
- Evaluar la mejora en tiempos y auditorías dentro del proceso de adquisición y aplicación de los productos e insumos para la fumigación del plátano al usar la solución planteada.

1.6 Metodología

Se usará para el presente proyecto una metodología de proyectos tradicional basado en cascada. Se realizará una revisión teórica en

resumen sobre la tecnología Blockchain, una breve descripción de las herramientas OpenSource más conocidas sobre las cuales se basa la solución de Blockchain a usar. Se realizará un diseño top-down para ir avanzando en la toma de los requerimientos funcionales y no funcionales del sistema y diseño en detalle de la solución, previo a su implementación, donde se desarrollan los contratos inteligentes y la interface WEB que permite gestionar los cambios en el Blockchain, el desarrollo de las pruebas y finalmente la revisión de los resultados.

CAPÍTULO 2

2. MARCO TEÓRICO

2.1 Tecnología de Contabilidad Distribuida (Blockchain)

En esta sección explicaremos brevemente los conceptos básicos de la tecnología Blockchain o cadena de bloques, enfocando su aplicabilidad a áreas diferentes a la financiera.

La cadena de Bloques (Blockchain) en sí un gran libro de cuentas en lo que los registros de transacciones (bloques) están en enlazados y cifrados con el fin de proteger la seguridad y privacidad de las mismas [1], permitiendo además, realizar un seguimiento de los recursos en esa red, en el sentido teórico es una gran base distribuida cuya información básica se conoce como un recurso. Un recurso (asset) puede ser algo tangible (una casa, un vehículo, dinero, una fruta, un terreno) o intangible (contrato seguro, propiedad intelectual, registro médico). Cualquier cosa que pueda ser rastreada y tratada en una cadena de bloques, permitirá reducir riesgos y recortará costos de todo lo involucrado.

En una cadena de bloques, la transferencia de información no requiere de un solo intermediario centralizado que identifique y certifique la información, sino que está distribuida en múltiples nodos independientes entre sí que validan y la registran sin necesidad de que se conozcan entre ellos [2]. La información una vez liberada no puede ser borrada, solo se podrán añadir nuevos registros y no será legítima, hasta que la mayoría de los nodos se pongan de acuerdo en hacerlo. Este concepto fue originalmente aplicado a la creación del BitCoin, que mantiene esas características de eliminar intermediarios, ser eficiente (una sola vez se registra y se dispone a todos los involucrados en la red), seguro y confiable (una transacción no puede ser cambiada, y solo se puede reversar una transacción con otra transacción, siendo ambas visibles) [3]. Vamos a corroborar el uso potencial de Blockchain bajo una red con permiso basado en usuarios que se relacionan sus permisos en dicha red, contra sus propias entidades [2].

En resumen, sobre una primera vista general de la cadena de bloques, es una base de datos distribuida, compuesta por una cadena de bloques de tamaño fijo, que incluyen de 1 a N transacciones, donde cada transacción que añade un nuevo bloque es validado y luego insertado en la cadena, manteniendo la referencia del bloque(s) anteriormente insertado(s), en pocas palabras la nueva información será insertada al final de la cadena. Viendo así hay sólo dos operaciones, lo que es diferente a las operaciones CRUD (siglas en inglés que significan: crear, leer, actualizar y borrar) que mantienen los gestores de bases de datos tradicionales.

Existen además diferentes tipos de cadenas de bloques que se describen a continuación:

Cadenas de bloques públicos: Tales como el Bitcoin, son redes distribuidas enormes que corren a través de un token nativo. Están abiertos para cualquiera que desee participar desde cualquier nivel y tiene un código abierto que su comunidad mantiene.

Cadena de bloques con permisos: Como por ejemplo Rippie, existe controles por roles que los individuos pueden asumir dentro de la red. Existen en sistemas grandes y distribuidos que usan un token nativo. Su código puede no necesariamente ser abierto. Se puede además restringir el acceso a los recursos definiendo una política en lo que se conoce como Listas de Control de Acceso (ACL en siglas en ingles de Access Control List).

Cadena de bloques privados: Estos tienden a ser pequeños y no utilizan un token. Mantiene una membresía cerrada, esos tipos de Blockchain son mantenidos por consorcios que tienen miembros confiables y manejan información confidencial.

Todos los tipos de cadenas manejan criptografía, para permitir que cada participante de una determinada red maneje el libro de cuentas en una forma segura, sin la necesidad de un ente autorizado central para reforzar o definir las reglas.

El procesamiento de la cadena de bloques pública (que servirá para explicar el proceso más general) consiste de manera resumida en estos pasos, mostrados en la siguiente imagen [4] [5]:

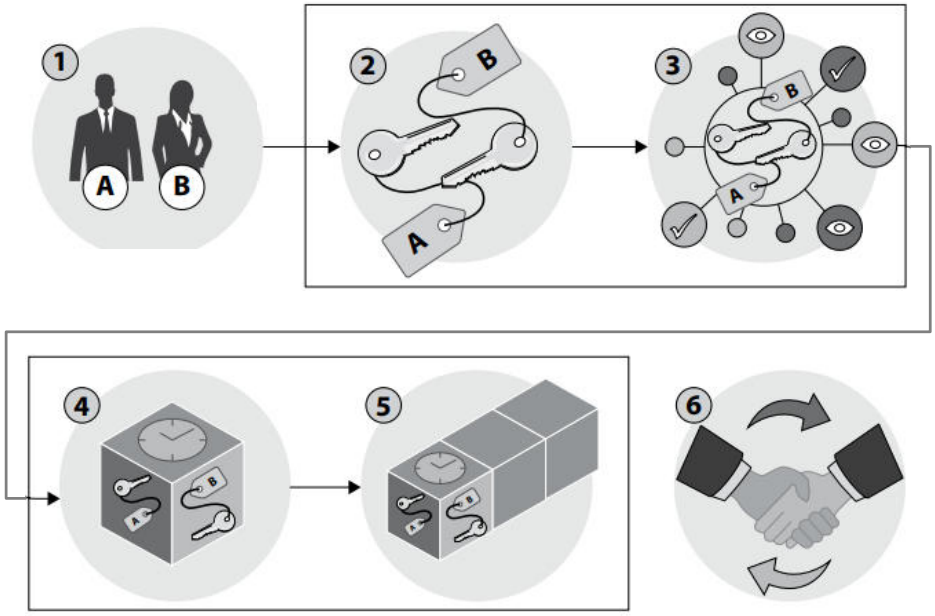


Figura 2.1 Flujo de transacción en un Blockchain público[6]

1. Añadir una nueva e imborrable transacción y organizarla dentro de bloques: Las entidades A y B desean conducir una interacción o transacción en la red.

2. Verificar la criptografía de cada transacción en el bloque: se asignan llaves criptográficas para la interacción que llevan ambos A y B.

3. La interacción es replicada y verificada por la red distribuida.

4. Una vez validado un nuevo bloque se crea.

5. Luego, se agrega el bloque al final de la cadena existente inmutable.

6. La transacción entre A y B es completada.

Para evitar que la red sea corrompida, no solamente se vale de la descentralización, se usa una criptomoneda, este es un token digital que tiene un valor en el mercado. Las criptomonedas son negociadas como intercambios como existencias.

Cada criptomoneda trabaja de forma diferente en cada Blockchain, este básicamente paga el hardware que operará las transacciones. Este software es el protocolo del Blockchain, este listado de protocolos incluye Bitcoin, Ethereum, Rippie, Hyperledger y Fatcom, mientras que el hardware consiste en los nodos llenos que aseguran los datos en la red.

2.2 Arquitectura de Blockchain

Esta plataforma encontró aplicación en las redes de negocio, en la cual los participantes tienen la posibilidad de compartir un libro diario que se actualiza durante la replicación capa a capa (peer to peer) en la cual cada participante puede publicar información o ser un subscriptor. Lo que difiere son los registros de transacciones que comparten, como vamos a ver ya en la vida real la aplicación cliente del Blockchain se encargará de definir el tipo de transacciones que trabajará cada participante.

En la siguiente tabla se indican los tres componentes principales de las cadenas de bloques (Blockchain) [6].

Tabla 2. Estructura de las Cadenas de Bloques

Resumen de componentes	
Bloques	Es una lista de transacciones registradas en un libro mayor (ledger) sobre un período dado. El tamaño, período y el evento disparador para los bloques es diferente para cadena de bloques. No todas las cadenas de bloques guardan y aseguran un registro del movimiento de su criptomoneda como objetivo principal, pero si, todas las cadenas de bloque registran el movimiento.
Cadenas	Es un código hash o algoritmo matemático que transforma el contenido de un bloque de datos en una nueva serie e caracteres de longitud fija, independiente de la longitud de los datos, que encadena un bloque con otro. El hash en cada bloque de la cadena es creado a partir de los datos del bloque previo, es una huella digital de los datos y de los bloques cerrados en orden con su marca de tiempo.
Red	La red es compuesta de nodos llenos, pensándolos como computadas corriendo algoritmos que aseguran la red, cada nodo contiene un registro completo de todas las transacciones que han sido almacenadas en la cadena de bloques.

La red ofrece las siguientes características clave, mismas que permitirán evaluar la elección del tipo de cadena de bloques a implementar.

Consenso: Para que una transacción sea válida, todos los participantes en ella deben estar de acuerdo en su validación, existen como se verá más adelante varios mecanismos para ello.

Inmutabilidad: Ningún participante puede alterar o trampear una transacción luego de que ha sido guardada en el libro mayor. Si la transacción tiene un error, una nueva transacción deberá ser usada para reversar el error, y ambas transacciones son visibles.

Finalidad o Resolución: Un sencillo, compartido libro mayor provee un lugar para determinar la procedencia o propietario de un recurso o la completitud de una transacción.

Existen dentro de los Blockchain más conocidas 2 que vamos a comparar debido a que son las que hay más información y son las más usadas Ethereum e HyperLedger.

Ethereum introdujo el concepto de contratos inteligentes por primera vez [7] llevando la implementación más allá de la criptomoneda, y agregando la característica de programar la cadena de bloques como antes no se podía. Sin embargo, aunque extensiones para lenguajes como Python[8] [9] [10] para desarrollar en Ethereum, nativamente se realizan los desarrollos usando JavaScript para la capa de FrontEnd y para el desarrollo de los contratos inteligentes (parte del back-end) el lenguaje llamado Solidity (lenguaje orientado a objetos), esta cadena de bloques se construyó basado en su implementación de la criptomoneda Ether. Los contratos inteligentes permiten operaciones de datos específicas sin

exponer los detalles de la base de datos distribuida, y permite crear una cadena de bloques privada personalizada. Estas características permitieron evidenciar el uso de Blockchain a otras áreas de negocio, sin embargo su característica de ser de consumo masivo y absolutamente transparente (es decir, visible para todos en la red), no permitió que la versión actual cumpla las expectativas de las empresas, sin embargo, para encontrar nuevos usos a la tecnología de Ethereum, hay otras iniciativas que están avanzando apoyadas en empresas como Microsoft, Accenture, J.P Morgan, etc., tales como el llamado Ethereum Alliance, cuyo fin es personalizar la red de Blockchain de Ethereum para que se aplique a las industrias, con el fin de mejorar la confianza en sus transacciones comerciales y facilitar el seguimiento a las operaciones de las cadenas de suministro o como la plataforma Quorum que se enfoca en soluciones empresariales, pero con cierto sesgo hacia el sector financiero, pretendiendo mejorar los temas de confianza, estableciendo en los usuarios mecanismos para establecer su veracidad, admite la validación de firmas.

Para cumplir la falta de funcionalidad de Ethereum surge Hyperledger, que es un sistema abierto basado en Linux, y que se usa para producir Blockchain orientadas más al ámbito empresarial. También se implementó en base al concepto de contrato inteligente de Ethereum, pero le otorga otro enfoque, usando el nuevo concepto de “Chaincode” (código encadenado), que aprovecha el concepto de consenso y la confianza de la tecnología Blockchain. Se basa en un sistema llamado Fabric, que se

mantiene por las organizaciones a nivel mundial tales como Digital Asset, IBM, Hitachi, Huawei Technologies, ImpactChoice, The Linux Fundation, etc. que crean nuevas funcionalidades y contribuyen a esta arquitectura de código abierto. Hyperledger da la flexibilidad y la seguridad de hacer transacciones visibles para ciertos grupos que tengan las llaves de encriptación correctas o necesarias para participar de esas transacciones.

Su implementación tipo framework HyperLedger Fabric, permite desarrollar aplicaciones o soluciones de una forma modular, permitiendo que los componentes, el concepto de consenso (concepto visto anteriormente) y los servicios de membresía (esquema de acceso a los servicios del Blockchain) sean plug-and-play.

Hyperledger además tiene otros proyectos y frameworks que permiten el desarrollo, tales como Hyperledger Explorer, para ver el estado de sus plataformas, HyperLedger Sawtooth soluciones para el sector financiero y para proyectos con Internet de las Cosas (IoT).

A continuación, se describe en la siguiente tabla las diferencias actuales sobre las dos implementaciones de Blockchain más conocidas.

Tabla 3. Comparación entre dos tipos de cadena de bloques

Características	Hyperledger	Ethereum
Propósito	Preferida para temas de B2B para negocios.	Plataforma para negocios B2C y aplicaciones generalizadas.
Confidencialidad	Transacciones confidenciales.	Transparentes (abiertas).
Modo de participación	Redes privadas y bajo permisos.	Redes sin Permisos y Públicas/Privadas.
Mecanismo de Consenso	Algoritmo de Consenso conectado (todos los participantes que tengan el permiso pueden validar el curso de la información en la cadena) no hay minería que realizar.	Algoritmo PoW (se usa el trabajo para decidir quién va a tener autoridad sobre el camino que tomará la cadena de bloques, se oferente a utilizar infraestructura que se necesita para minar los bloques (gestionar las transacciones), esto le da autoridad.
Lenguaje de programación	Código de la cadena escrita en lenguaje Golang, aunque también hay extensiones para otros lenguajes.	Los Contratos inteligentes son escritos en el lenguaje Solidity.
Criptomonedas	No tiene criptomonedas.	Incorpora la criptomoneda llamada Ether.

Durante el proyecto se usó durante el desarrollo HyperLedger debido a sus prestaciones y para el despliegue en un ambiente de pruebas para la

demostración se desplegó en la nube en un servidor Linux. Cabe mencionar, que también se puede usar de manera opcional la plataforma de IBM en la nube, que ofrece un ambiente administrado para un full-stack Blockchain como servicio (BaaS), a continuación, en la Figura 2.2 se resume y se explica sus características.

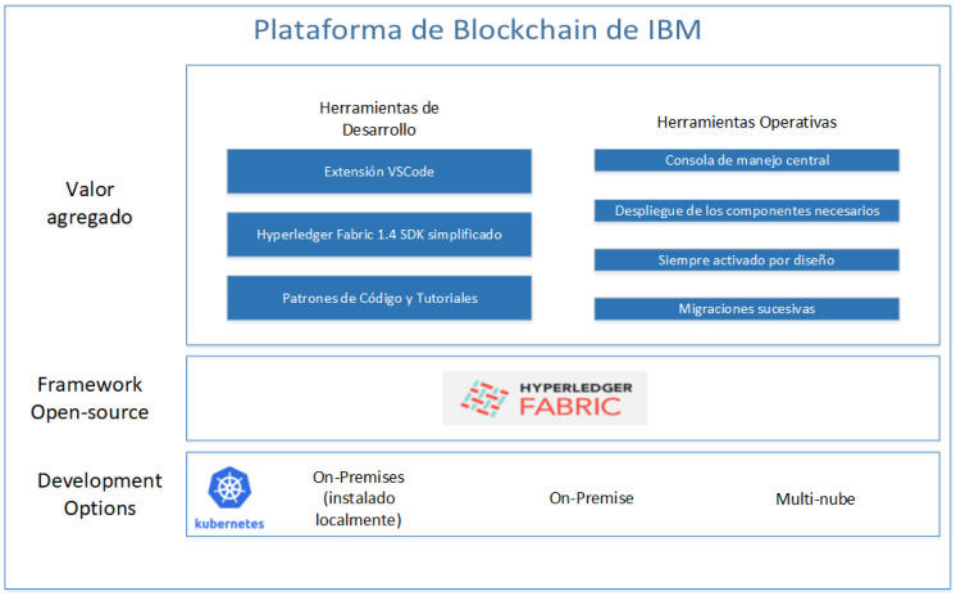


Figura 2.2 Arquitectura de la plataforma de IBM con HyperLedger [11]

- El framework está construido sobre Hyperledger Fabric 1.4, que incluye un SDK simplificado [11]. Incorpora herramientas de despliegue, paneles para administración y otro conjunto de herramientas, además de una extensión para Visual Studio Code.

- Facilita la creación de nuevos miembros o participantes en minutos, y crea canales seguros y privados con la misma facilidad.

- Soporta múltiples lenguajes para el desarrollo de los contratos inteligentes incluyendo: Node.js, Go, Java, Solidity y más.
- Permite correr migraciones de actualizaciones con un bajo tipo de espera en la red.
- Existen dos modelos la nube y la multi-nube, permite la apertura tanto a soluciones IBM como de terceros, provee soporte 24x7x365, así como otras características.

2.3 Contratos Inteligentes

A pesar de que los contratos inteligentes es un término que se ha difundido mayoritariamente con la introducción de la tecnología de la cadena de Bloques (Blockchain), realmente se los puede considerar como “cualquier acuerdo en el existen cláusulas definidas mediante scripts o pequeños programas, cuyo efecto es que una vez concluido el acuerdo y señalados uno o varios eventos desencadenantes, esa producción de eventos conlleva a la ejecución automática del resto del contrato, sin que ocurra modificación, bloqueo o la no ejecución de la prestación debida” [12].

Esos mecanismos de ejecución no dependerán de la voluntad de las partes sino de lo que se haya definido en el programa de ejecución. Para permitir que esto ocurra, estos contratos se apoyan en la tecnología de cadenas de bloques, adicionalmente, al usar esa tecnología los scripts pueden programarse en serie y usan los datos enviados en las transacciones de las cadenas, ejecutando tanto tareas sencillas como tareas complejas,

tanto reactivas como activar garantías por un préstamo o proactivas tal como liberar una subvención a un préstamo [12].

Los casos de uso son diversos van desde el sector financiero tales como Préstamos, Depósitos en garantía, Compras en línea de vehículos, así como la industria referente a la Gestión de la cadena de suministros para Productos Manufacturados que abarca Salud, Alimentos, Telecomunicaciones, etc.

2.4 Aplicaciones WEB

Para la implementación de aplicaciones WEB hay varias opciones de patrones de diseño y arquitectura que se pueden seguir.

En un principio el desarrollo de la aplicación web se trabajó bajo una arquitectura de tipo Monolítica para realizar el trabajo en ambiente de desarrollo, así como durante la ejecución de pruebas básicas de integración con Blockchain en un entorno virtual, en un proyecto de ASP.NET de un solo proyecto. Sin embargo, el despliegue fue realizado en un entorno por capas, usando la tecnología de ASP.NET 4.7 Web Forms, usando C# como lenguaje de programación.

Esta tecnología de ASP .NET Web Forms permite usar, de cierta manera, un modelo orientado a eventos diseñando en un ambiente arrastrar y soltar, que es familiar para los desarrolladores de escritorio. Este se basa en el Microsoft .NET Framework, lo que proporciona las ventajas que posee

entre ellas un entorno administrador, seguridad de tipos y herencia. Se puede agregar controles creados por el usuario y por controles de terceros.

Durante el desarrollo se trabajó en carpetas independientes, así como carpetas adicionales para los datos y servicios de ser el caso.

Las tareas de presentación fueron puestas en la carpeta vista, los detalles de acceso a datos en la carpeta datos, y la lógica de negocios de la aplicación web, residió en los servicios y las clases en las carpetas de Modelos, la capa de acceso a Datos usó las Apis desarrolladas en NodeJs para exponer los contratos de Blockchain.

La aplicación web que ya se encontraba dentro de un servidor en la nube contratado para este propósito, logró extender su funcionalidad, agregándole una interacción contra una API generada en HyperLedger, para gestionar la información del plan de fumigación. Cabe mencionar que aunque no se implementó por temas de costos, es posible usar para esta integración, una subscripción en la plataforma Blockchain de IBM. A continuación se muestra una gráfica que describe la arquitectura WEB planteada y su relación con Blockchain [11].

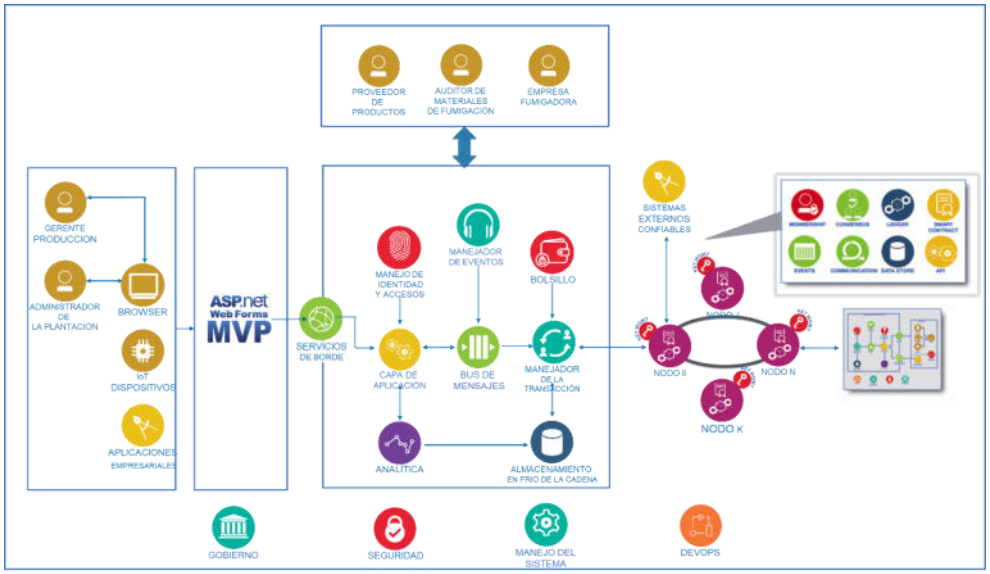


Figura 2.3 Arquitectura con ASP Web Forms Web y Blockchain [11]

Los formularios WEB usarán las Apis de la plataforma de Blockchain, para escribir en la cadena los bloques cuya actualización será realizada por usuarios con roles específicos a través de la aplicación WEB.

El gerente de producción y el administrador de la plantación accederán a una instancia local de la aplicación Web, mientras que los participantes externos como la Empresas Proveedoras de Productos, Empresa Fumigadora y la Empresa Auditora, accederán mediante micro sitios web de acceso autorizado externo.

La aplicación de FrontEnd Web, se basó en un formulario Web Asp.net con los siguientes componentes mostrados en la siguiente imagen:

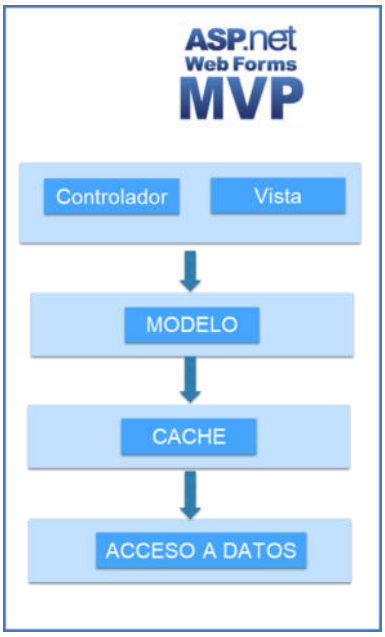


Figura 2.4 **Arquitectura de los Formularios ASP.Net usar en la aplicación**

El uso de esta tecnología tiene las siguientes ventajas [13] :

- Separar el código HTML y otro código de interfaz de usuario de la lógica de la aplicación.
- Un conjunto de controles de servidor para tareas habituales como el acceso a datos.
- Enlace de datos, con compatibilidad con herramientas excelentes.
- Compatibilidad con scripting del lado cliente que se ejecuta en el explorador.

- Compatibilidad con una variedad de otras funciones, incluido el enrutamiento, seguridad, rendimiento, internacionalización, pruebas, depuración, control de errores y administración de Estados.

2.5 Seguridad de la información

Dentro de la aplicación se contempló tratar de cubrir los aspectos básicos descritos en la literatura [14].

Confidencialidad: Se plantea que ambos servicios tanto la aplicación WEB como la Cadena de Bloques con sus contratos inteligentes y transacciones generadas, sean accedidos sólo a personal autorizado, al no estar disponible en un centro de cómputo local, se minimiza el riesgo de acceso. Las tareas de respaldos son llevadas a cabo como parte de la prestación de servicios seguros.

Autenticación: Del lado de la aplicación WEB se garantiza que la identidad del creador de los pedidos de información y actualizaciones sea legítima, a través de controles de ASP.Net, también se puede manejar variables de cookies, en nuestro caso se usará formularios Web Forms de ASP.NET con registro de usuario, confirmación por correo electrónico y restablecimiento de contraseña con C#. Referente a la interfaz para trabajar la cadena de bloques que brinda IBM se rige y mantiene altos estándares federales de procesamiento de información y los niveles de evaluación de seguridad (FIPS 140-2 y EAL respectivamente), en la interfaz web que provee IBM Blockchain, permite así como en Hyperledger

crear perfiles con roles para cada ente creador o consultor dentro del sistema.

Integridad: Para garantizar que ningún archivo o mensaje, no se haya modificado desde la creación en la cadena, se incluye mecanismos de auditoria interna sobre el proceso de tal manera que el sistema no registre cambios en el BlockChain. A su vez, el ambiente del Blockchain creado como una cadena privada, no permitirá que ningún usuario ajeno entre sin cumplir las herramientas de autenticación, primeramente.

No repudio: Otro objetivo que cumple la cadena de bloques es que la información que cumple todas las condiciones entra en la cadena, misma que no puede ser alterada gracias a que entre otras cosas arrastra el Hash desde el primer nodo de la cadena, información que es contrastada por la misma plataforma, lo cual mitiga el repudio de algún reporte, consulta o notificación notificada por email.

Disponibilidad: Tanto el sistema Web en la nube como la cadena de bloques, cuenta cubre algunos aspectos de disponibilidad, cubriéndose de ataques de denegación de servicio.

Auditabilidad: Los servicios tienen como registrar los eventos generados por los usuarios autorizados y autenticados que se suscitan de forma de forma histórica, con los cuales se puede identificar algún comportamiento anómalo y riesgoso en de los usuarios. Además, es posible obtener reportes de rendimiento del sistema tanto para la aplicación WEB como para la cadena de Bloques.

Autorización: Se crean listas de acceso basados en roles (grupos de usuarios) los cuales tendrán permisos a ciertas opciones de los sistemas.

Confirmación de la realización de una transacción: Los usuarios recibirán de forma visual y por correo la retroalimentación del éxito o fallo en las transacciones realizadas.

CAPÍTULO 3

3. DEFINICIÓN DE REQUERIMIENTOS

3.1 Resumen de la situación actual

La fumigación de plantaciones de Banano en la Empresa siempre ha sido uno de los grandes problemas, la participación de entidades externa ha hecho que cada una de ellas cree su propia forma de registro y que los tiempos de entrega de los mismos sean diferentes.

Los tipos de archivos con los que se trabajan son tablas en hojas de Excel, las cuales no poseen seguridad alguna de ser manipulable por la entidad que la crea como la entidad que recibe, lo que dificultad la seguridad sobre los datos ingresados.

Las entidades que participan en el proceso de fumigación son:

Proveedor de Productos: Empresa a la cual se le compra los productos de fumigación en base a las certificaciones que posee y las características necesarias para el mismo.

Auditor de material de fumigación: Pertenece a la empresa consultora que provee el uso de los productos comprados, define la dosis a utilizar.

Gerente de Producción: Pertenece a la empresa que contrata el servicio de fumigación y que compra los productos a aplicar, define el plan semanal para el uso de los productos, con la dosis recomendada y los lugares donde se va a aplicar.

Empresa Fumigadora: Empresa contratada para realizar fumigaciones aéreas como terrestres cumpliendo los estándares de seguridad laboral y protección contra químicos.

Administrador Financiero de la plantación: Realiza las labores de compra y pago de los servicios y productos en la plantación.

Se requiere encontrar una alternativa que permita que la transacción fluya entre las entidades, a través de un sistema que minimice la necesidad de controles manuales y personas que no sean parte de la cadena de valor y que actúen a manera de intermediarios, proveyendo una interfaz que permita el ingreso y seguimiento de las transacciones, aislando la modificación por parte de entidades ajenas al proceso.

3.2 Definir el caso de uso para un modelo de Blockchain

En la agricultura el uso del Blockchain ha ido evolucionando en los últimos años, la mayoría se enfoca en la calidad del producto final. Empresas en Ecuador como Dolé, Nestlé y Unilever desean utilizar la trazabilidad y la inalterabilidad de los datos con el fin de proveer al consumidor final la opción de saber de dónde se originó, así como los procesos que transcurrieron hasta llegar a sus manos.

Gracias a la tecnología de Blockchain, que permiten que la información sea manejada de forma segura y confiable, se identifica la posibilidad de desarrollar, en el área de producción de banano escenarios de uso orientado al control de productos de fumigación utilizados. Esta información es muy relevante para sus consumidores, debido a que esta tecnología nos permite mantener la disponibilidad y transparencia todo el tiempo ya que se va a encontrar distribuida en la nube (internet). En un futuro casi inmediato, otras aplicaciones podrán añadirse otros escenarios de uso a esta cadena de bloque, como por ejemplo dentro de la cadena de suministros, llevar información del transporte y manejo de las cajas de banano hasta los supermercados.

3.3 Definir los requerimientos de seguridad de acceso sobre la información

En base a la teoría indicada dentro del Capítulo 2, se identifica que la solución a los temas de uso de Blockchain, es diseñar la cadena como privada, dando acceso sólo a personal perteneciente las entidades

autorizadas con el rol definido y con sus niveles de acceso en cada nodo de la cadena.

Blockchain traba la información distribuida en cada uno de los nodos con su libro mayor. Esta información solo va a ser accedida por los entes que se registren en la aplicación para su manejo no público.

De igual forma la solución Web tendría ya definidos roles y usuarios de aplicación con su respectiva autorización para registrar, consultar y recibir tanto reportes como notificaciones provenientes de la cadena de bloques.

A continuación, un resumen de las opciones con su nivel de acceso:

Aplicativo web (Privado), aplica para las entidades Gerente de Producción, y Administrador Financiero de la plantación.

- Formulario de usuario y contraseña.
- Plan semanal (Compartida con Blockchain por servicio web).
- Fitosanitario (Compartida con Blockchain por servicio web).
- Formulario ingreso de productos comprados por registro en el plan semanal.

Aplicativo NodeJS Blockchain (Privado entre Entidades), aplica para las entidades Auditor, Empresa Proveedora y Empresa Fumigadora.

- Formulario entidades.
- Formulario Ingreso Productos.

- Formulario Aplicación de dosis.
- Creación de Contratos inteligentes.

3.4 Revisar los requisitos de infraestructura: servidores, comunicación y redes

A continuación, se listan los componentes a nivel de Aplicación WEB (legado y Blockchain).

Servidor Aplicativo Web (Hosting Compartido):

- Sistema Operativo Windows Server 2016.
- Soporta ASP .NET , ASP Core, Php 5.0, NodeJS.
- Base de datos Microsoft Sql Server Versión 2012 hasta 2017.
- Espacio en Web ilimitado.
- Ancho de Banda Ilimitado.
- Conexiones concurrentes ilimitadas.
- Espacio de base de datos hasta 10gb y se puede expandir.
- Los discos duros son del tipo estado solidos o SSD.

Plataforma IBM Blockchain cumple con estas características:

- HSM certificado FIPS 140-2 Nivel 4 (módulo de hardware de criptografía de información integrado y certificado).

- Auditorías de seguridad por empresas terceras independientes.
- Contenedores de Servicio seguro: no se da acceso al súper usuario (root).
- Cumplimiento estándares y regulaciones sobre protección de datos tales como la EU Managed (estándar a nivel de la Unión Europea).
- Privacidad de datos a través de encriptación de datos, cumplimiento del nivel de evaluación de aseguramiento EAL5 que se basa en criterios estándar de evaluación de seguridad, en este caso nivel 5 (el software ha sido semi formalmente diseñado y probado).
- Soporte a la Tecnología Kubernets – Gestionadores de clúster, tecnología para publicar aplicaciones de gran escala.
- Servicios Web API que permiten intercambiar información entre las aplicaciones.
- Soporte a la creación de Contratos Inteligentes, que habilitan generar un flujo de proceso con el cumplimiento por parte de entidades.

Referentes al ambiente y lenguajes para el desarrollo de las aplicaciones tenemos lo siguiente:

Aplicativo Web desarrollada con Visual Studio de Microsoft:

- Tecnología Formularios Web Asp .Net.
- .Net Framework 4.7.

- Lenguaje de Programación C#.
- Soporte para Visual Studio Code de Microsoft.
- Aplicativo Cliente NodeJS para Blockchain IBM.
- Lenguaje de programación JavaScript, HTML 5.
- Lenguaje Go, Javascript para la elaboración de contratos.

3.5 Definir los criterios de aceptación

Los criterios de aceptación se basan en el cumplimiento de la entrega de información, la cual se procesa en su mayoría de forma tardía y con la probabilidad de que la información pueda ser modificada o alterada fácilmente.

A continuación, se indica la situación actual (AS-IS) en el siguiente gráfico:

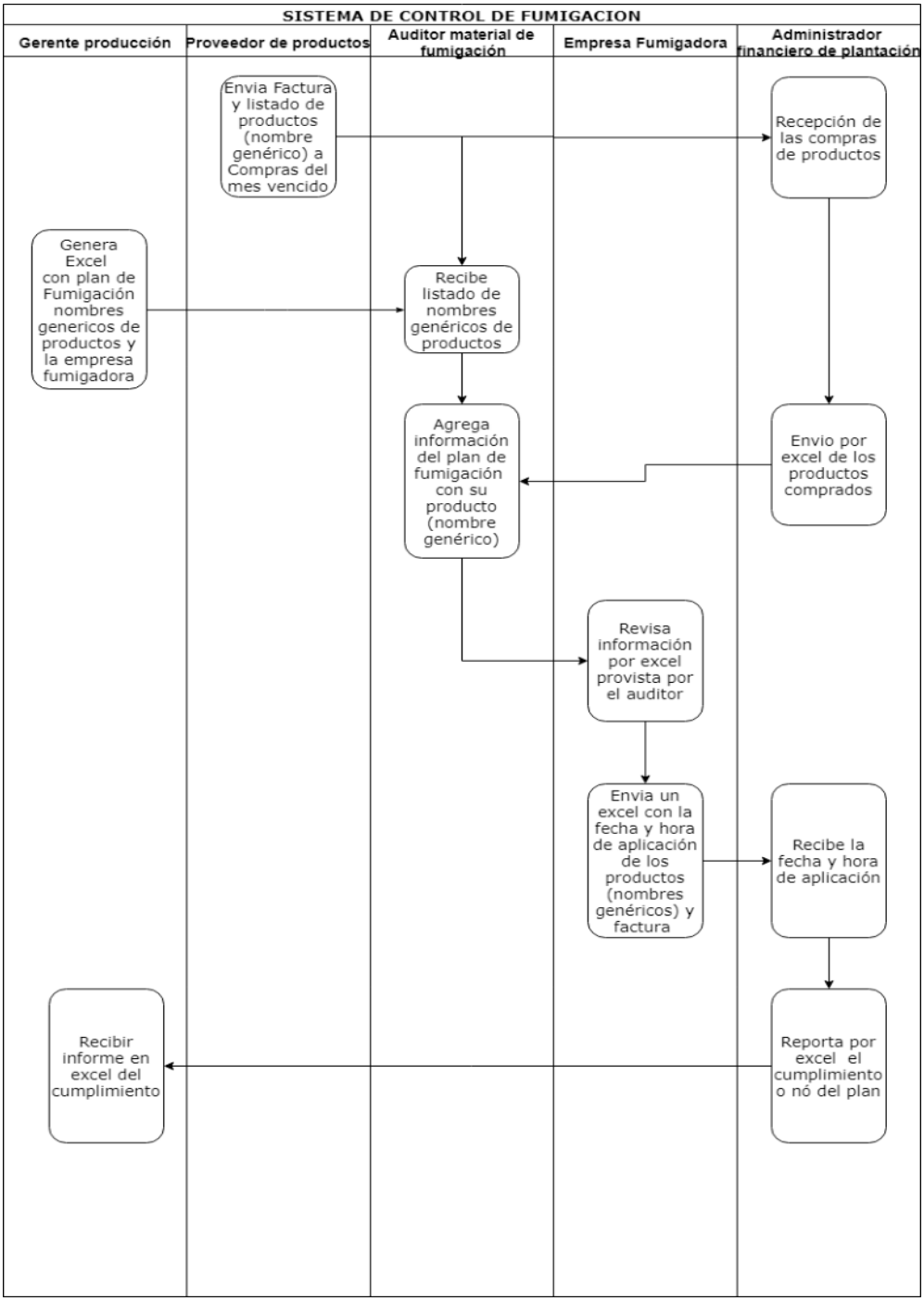


Figura 3.1 Flujo de proceso actual manual (AS-IS)

Datos de Productos comprados (Proveedor). - Esta información se la entrega con la factura al fin de mes (Hoja de Excel).

Datos de Productos a utilizarse (Auditor de material de fumigación). - La información del uso de los productos del proveedor, validación de certificados y sus respectivas dosis en base a un plan de aplicación. Dicho plan se trabaja con el gerente de producción de la empresa.

Datos de Fumigación en base a plan (Empresa Fumigadora). - La información provista por el Fitosanitario debe cumplir la fumigadora. Una vez realizada la fumigación lo reportan mediante una hoja de Excel al fin de mes.

Una vez cumplida la fumigación se procede a notificar la aprobación de los valores a cancelar por la adquisición de los productos (hacia el proveedor) y por el cumplimiento del plan de fumigación (hacia la empresa fumigadora).

A continuación, se muestra en el siguiente gráfico los pasos que se van a automatizar con la aplicación WEB y la cadena de bloques (TO-BE):

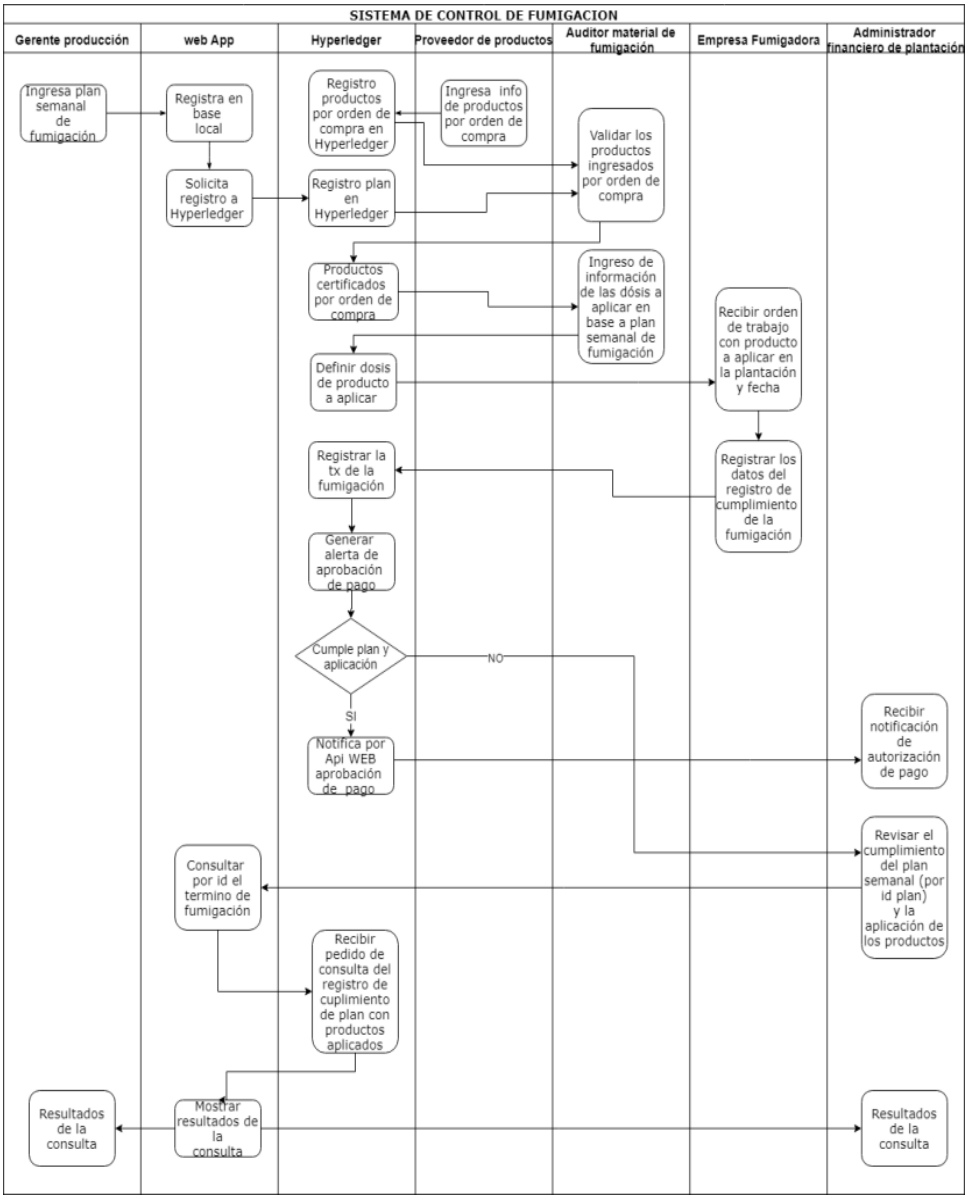


Figura 3.2 Flujo de proceso para propuesta de solución (TO-BE)

Bajo este escenario se generan las siguientes mejoras para cada entidad que se traducen en requerimientos funcionales descritos como casos de uso descritos a continuación:

Tabla 4. Caso de uso de ingreso del Plan semanal de fumigación

Autor(es)	César Villarroel y Daniel Gonzalez C.	**Fecha:**	15/12/2019
		Version:	1.0

Nombre del caso de uso:	Generar plan semanal de fumigación	Casos de uso del tipo requerimientos de la aplicación
Id del caso de uso:	GSICO-APP-00001	
Prioridad:	Alta	
Fuente:		
Actor primario de negocios:	Generente o Administrador de la plantación	
Otros actores participantes		
Descripción	Esta información se la ingresa en el aplicativo Blockchain a través de la aplicación local de la Empresa bananera, se ingresa los datos de los productos genéricos a aplicar, la fecha tentativa, el área y lotes así como alguna observación complementaria	
Precondición:	Debe estar definido un plan de fumigación	
Ocasionador:		

Curso Típico de eventos:	**Acción del actor:**	**Respuesta de la aplicación**
	Paso 1: El administrador registra el detalle de una tarea de fumigación	Paso 2: El registro de la tarea del plan de fumigación se guarda en el Blockchain, como participante la Empresa bananera

Tabla 5. Caso de uso del Ingreso de productos de fumigación comprados

Autor(es)	César Villarroel y Daniel Gonzalez C.	**Fecha:**	15/12/2019
		Version:	1.0

Nombre del caso de uso:	Registrar datos de la compra de productos de fumigación en micrositio Web externo	Casos de uso del tipo requerimientos de la aplicación
Id del caso de uso:	GSICO-APP-00002	
Prioridad:	Alta	
Fuente:		
Actor primario de negocios:	Empresa proveedora de productos de fumigación	
Otros actores participantes	Area de compras de la empresa bananera (paso previo durante la compra)	
Descripción	Asocia el numero de orden de compra, con los productos e insumos de fumigación por su código SKU, y su nombre comercial. Esos datos serán ingresados en un paso posterior por su orden de compra a la ejecución del plan de fumigación	
Precondición:	Orden de compra facturada por la empresa proveedora de productos de fumigación	
Ocasionador:	Area de Compras	
Curso Típico de eventos:	**Acción del actor:**	**Respuesta de la aplicación**
	Paso 1: La empresa registra el detalle de la orden de compra recibida dentro del sistema	Paso 2: La información se registra en la estructura del blockchain.

Tabla 6. Caso de uso de ingreso de datos para control fitosanitario productos y condiciones previas

Autor(es)	César Villarroel y Daniel Gonzalez C.	**Fecha:** 15/12/2019
		Version: 1.0
Nombre del caso de uso:	Registrar la información de las condiciones y los productos que se van a aplicar dentro del plan de fumigación	Casos de uso del tipo requerimientos de la aplicación
Id del caso de uso:	GSICO-APP-00003	
Prioridad:	Alta	
Fuente:		
Actor primario de negocios:	Empresa auditora de fumigación o un auditor fitosanitario de la empresa bananera	
Otros actores participantes		
Descripción	Con la Información ingresada de Productos y el plan semanal desde el aplicativo WEB se registra y gestiona la transacción de dosis a aplicar en fumigación. La revisión e ingreso de esta información ocurre al momento de recibir la notificación de las dos áreas	
Precondición:	Ingreso de los productos de fumigación de cada compra, e ingreso de la tarea del plan de fumigación	
Ocasionador:	Administrador de la Plantación	
Curso Típico de eventos:	**Acción del actor:**	**Respuesta de la aplicación**
	Paso 1: El auditor fitosanitario ingresa los productos que se aplicarán en la tarea de fumigación, la humedad y condiciones de viento que se deben de cumplir	Paso 2: La información actualiza el registro del plan de fumigación dentro de la estructura del blockchain.

Tabla 7. Caso de uso de ingreso de ejecución de las tareas de fumigación

Autor(es)	César Villarroel y Daniel Gonzalez C.	**Fecha:**	15/12/2019
		Version:	1.0

Nombre del caso de uso:	Registrar la ejecución de las tareas de fumigación del plan de fumigación	Casos de uso del tipo requerimientos de la aplicación
Id del caso de uso:	GSICO-APP-00004	
Prioridad:	Alta	
Fuente:		
Actor primario de negocios:	Empresa fumigadora	
Otros actores participantes		
Descripción	La Empresa Fumigadora recibe la notificación por parte de las entidades. Y una vez ejecutado el plan, se ingresa la información respectiva con una interfaz web hacia la plataforma de Blockchain	
Precondición:	Plan de fumigación actualizado con los productos que van a ser aplicados y la fecha tentativa para su ejecución	
Ocasionador:	Administrador de la Plantación	
Curso Típico de eventos:	**Acción del actor:**	**Respuesta de la aplicación**
	Paso 1: Empresa fumigadora registra la ejecución de las tareas de fumigación.	Paso 2: La información actualiza el registro del plan de fumigación dentro de la estructura del blockchain.

Tabla 8. Caso de uso de reporte de tareas pendientes de fumigación

Autor(es)	César Villarroel y Daniel Gonzalez C.	**Fecha:**	15/12/2019
		Version:	1.0

Nombre del caso de uso:	Mostrar un listado del reporte de las tareas pendientes de fumigación	Casos de uso del tipo requerimientos de la aplicación
Id del caso de uso:	GSICO-APP-00005	
Prioridad:	Alta	
Fuente:		
Actor primario de negocios:	Administrador de compras	
Otros actores participantes		
Descripción	El Administrador de compras perteneciente a la empresa bananera, accede a la aplicación Web local y revisa el estado e información de las tareas pendientes de fumigación	
Precondición:	Registro de las tareas de fumigación	
Ocasionador:	Auditor de fumigación	
Curso Típico de eventos:	**Acción del actor:**	**Respuesta de la aplicación**
	Paso 1: El administrador de compras revisa las tareas pendientes de fumigación	Paso 2: Se muestra la información por cada tarea pendiente de fumigación a través de la aplicación Web.

Tabla 9. Revisión de tareas completadas de fumigación

Autor(es)	César Villarroel y Daniel Gonzalez C.	**Fecha:**	15/12/2019
		Version:	1.0
Nombre del caso de uso:	Revisar tareas ejecutadas de los planes de fumigación	Casos de uso del tipo requerimientos de la aplicación	
Id del caso de uso:	GSICO-APP-00006		
Prioridad:	Alta		
Fuente:			
Actor primario de negocios:	Administrador de compras		
Otros actores participantes			
Descripción	El Administrador de compras es notificado de la finalización de una tarea de fumigación y revisa en la aplicación Web un reporte de las tareas cerradas		
Precondición:	Registro de la ejecución de una tarea de fumigación		
Ocasionador:	Empresa fumigadora		
Curso Típico de eventos:	**Acción del actor:**	**Respuesta de la aplicación**	
	Paso 1: El administrador de compras revisa las tareas de fumigación finalizadas o ejecutadas	Paso 2: Se muestra la información por cada tarea ejecutada de fumigación a través de la aplicación Web.	

Se definen a continuación, los requerimientos técnicos que se deben de aplicar para implementar esta solución:

Disponibilidad a través de un aplicativo Web conectado en la nube, al cual solo pueda acceder el personal autorizado (Gerente de producción y Administrador financiero).

Disponibilidad en la nube de la cadena de bloques usando para esto una aplicación montada en la plataforma IBM, a la cual tendrán acceso las entidades externas (Proveedor Productos, Auditor de Material de Fumigación, Empresa Fumigadora).

Permitir la integración entre la aplicación Web y la plataforma Blockchain para realizar consultas del personal interno y recibir las notificaciones y reportes.

Las notificaciones provenientes del Blockchain serán canalizadas vía correo electrónico.

Ambas plataformas estarán protegidas a personal no autorizado, manteniendo la integridad de la información.

3.6 Resumen del alcance del Proyecto

De forma resumida el alcance acordado para el cumplimiento del proyecto se resume en tres puntos, mencionados a continuación:

a) Diseñar una solución de software que permita resolver las necesidades de transparencia con el uso de productos químicos en la producción de banano, utilizando tecnologías como cadena de bloques (Blockchain) y aplicaciones web. Las cuales nos permiten garantizar confianza, autenticidad y disponibilidad de la información.

b) Implementar el piloto de la solución contra el Blockchain, para lo cual es necesario crear las facilidades tecnológicas y uso de las plataformas como cadena de bloque para las transacciones de las entidades (Proveedor, Auditor, Empresa Fumigadora). También se requiere un alojamiento en la nube para el aplicativo web que usara los servicios de la plataforma de la cadena de bloque para transacciones en contratos inteligentes.

c) Poner a prueba la implementación y revisar los resultados con respecto a los criterios de aceptación funcionales y técnicos.

CAPÍTULO 4

4. ESTUDIO COMPARATIVO Y DE FACTIBILIDAD

4.1 Estudio de costos y escalabilidad de una solución basada en la tecnología de Blockchain

Dentro de las diferentes alternativas se evaluó poner la solución en una máquina virtual en la nube como opción para la empresa bananera, a continuación, se realiza un análisis comparativo de beneficio y costos, con la finalidad de escoger la mejor opción tomando en consideración el tipo de inversión y la empresa cliente.

Respecto a la solución integral de poner la solución en la nube de IBM, se resume a continuación, los beneficios y precios referenciales [15].

La plataforma de IBM Blockchain presenta un nuevo modelo de precios por hora que se basa en el uso de núcleos de procesador virtual (VPC). Este modelo simplificado se basa en la cantidad de CPU (o VPC) que consumen

los nodos de IBM Blockchain Platform en cada hora, con una tarifa plana de $0,29 USD/VPC-hora.

Una VPC es una unidad de medida que se utiliza para determinar el coste de la licencia de los productos de IBM. Se basa en el número de núcleos virtuales (vCPU) que están disponibles para el producto. Una vCPU es un núcleo virtual que se asigna a una máquina virtual o a un núcleo de procesador físico. Para fines de estimación del coste de IBM Blockchain Platform, 1 VPC = 1 CPU = 1 vCPU = 1 Núcleo.

Opciones de precios** (1 VPC = 1 CPU)	Red de prueba	Unirse a una red de producción
Asignación de CPU	1,65 CPU Incluye: - 1 igual (1,1 CPU) - 2 CA (0,1 CPU x 2) - 1 nodo de ordenación (0,35 CPU)	4,5 CPU Incluye: - 2 iguales (para HA) **(2x cálculo predeterminado = 2 x 1,1 x 2)** - 1 CA (0,1)
Coste por hora: IBM Blockchain Platform	$0,48 USD (1,65 CPU x $0,29 USD/VPC-hora)	$1,31 USD (4,5 CPU x $0,29 USD/VPC-hora)
Coste por hora: clúster Kubernetes de IBM Cloud	$0,27 USD (Cálculo: 4 x 16 nivel más bajo; 1 nodo trabajador; 1 zona) (Asignación de IP: $16 USD/mes)	$0,46 USD (Cálculo: 8 x 32 nivel más bajo; 1 nodo trabajador; 1 zona) (Asignación de IP: $16 USD/mes)
Coste por hora: almacenamiento	$0,07 USD 340GB Bronce 2 IOPS/GB	$0,13 USD 420GB Plata 4 IOPS/GB
Coste total por hora	**$0,82 USD**	**$1,90 USD**

Figura 4.1 Lista de precios para ambiente a medida [15]

- El caso de ejemplo **Red de prueba** es adecuado para empezar y para probar contratos inteligentes sobre un equipo manualmente creado en la nube sin necesidad de invertir aún en la plataforma IBM.
- El caso de ejemplo **unirse a una red de producción** incluir equipos con iguales características, lo cual se recomienda para la alta disponibilidad, y una entidad emisora de certificados (CA) necesaria para la pertenencia a la organización.
- Esta solución se puede implementar a una red de producción de IBM Blockchain Platform.
- Los nodos siempre pueden volver a un estado de utilización mínima (0,001 CPU) cuando no estén en uso para reducir el coste.
- Debido a que este caso de ejemplo está pensado para un entorno de **producción**:
    - Los recursos de cálculo predeterminados se han duplicado para proporcionar una mayor capacidad.
    - En el ejemplo se ha elegido la clase de almacenamiento Plata para aumentar el rendimiento.

A continuación, se revisa la propuesta de la compra de un servidor virtual en la nube, con las características para una empresa con bajo presupuesto:

Tabla 10. Características del servidor para Blockchain y costos asociados al Hosting

CARACTERISTICA	DETALLE
Sistema Operativo	Ubuntu 18-04.23
Instalación	Free
Ip estática	SI
Tipo de servidor	VPS
Discos	SSD (State Solid Drive)
Memoria	16 Gb
VCPU	4 cores
Costo total	$15 / mes

Se realizó la adquisición del Hosting a través de la empresa SmarterASP.NET, usando la tarifa del plan .NET Premium.

A continuación, se detallan las características del servidor virtual, para el desarrollo de la aplicación WEB:

Tabla 11. Características del servidor virtual Web y costos asociados

CARACTERISTICA	DETALLE
Sistema Operativo	Windows Server 2016
Instalación	Free
Tipo de plan	Premium
Espacio Web	Ilimitado
Número de Sitios	Ilimitado
Nombres de Dominio	Ilimitado
Ancho de Banda	Ilimitado
Ip estática	$2 / mes
Permite encriptación SSL	SI
Acceso a los logs planos	SI
Panel de control Web	Si
FTP	SI
IIS	10x
Visual Studio	2017
MSSQL 2014	Ilimitado
SQL Backup and SQL Restore	SI
Tecnología Disco Duro	SSD
Soporte software	ASP .NET , ASP Core, Php 5.0, NodeJS
Costo total	$8.25 / mes = $99 / año

En conclusión, el caso de uso a implementar es adecuado para empezar y para probar contratos inteligentes sobre un equipo manualmente creado en la nube sin necesidad de invertir aún en la plataforma IBM.

4.2 Esquema detallado de las herramientas en la nube para desarrollo y administración de Blockchain.

El ambiente en el que se desarrolla la solución de Blockchain se basa en el uso de HyperLedger Fabric e HyperLedger Composer, ya tratado en capítulos anteriores y que se resume en la siguiente figura:

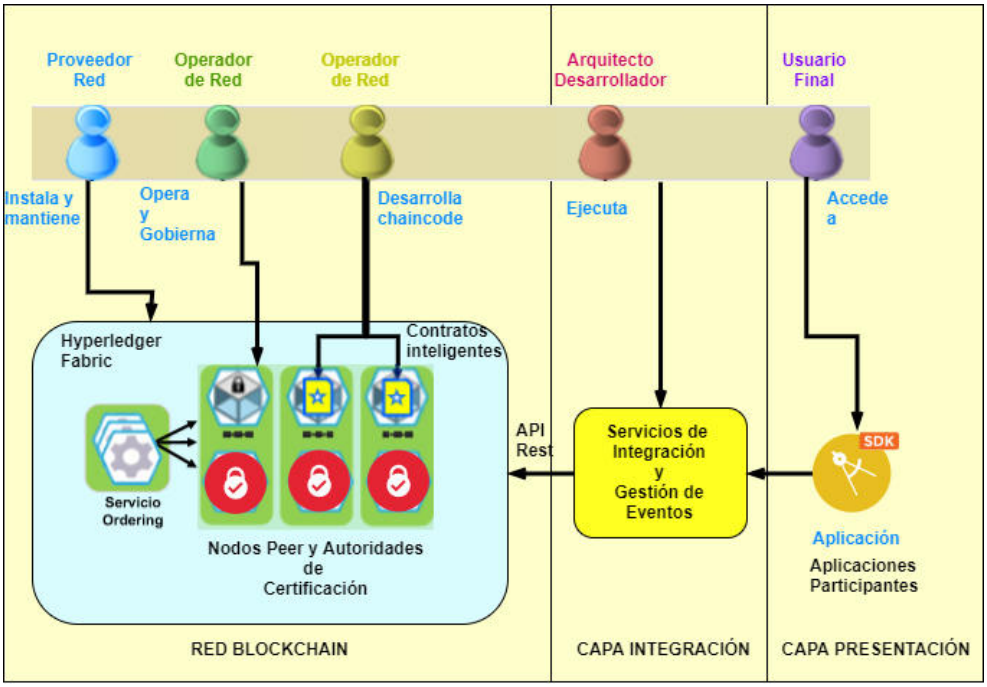


Figura 4.2 Ambiente de operación y desarrollo para la tecnología Blockchain usando HyperLedger Fabric [16]

HyperLedger Fabric, permite la construcción y operación de aplicaciones soportadas en Blockchain, los operadores de red administran los servicios

de ordenamiento, que validan el orden de las transacciones y las distribuye entre los bloques a los nodos, que no son más que, servicios de red que mantienen el estado del ledger y ejecutan los contratos inteligentes, que son creados por los desarrolladores de Blockchain, el subconjunto de redes de nodos que comparten un ledger y que ejecutan los contratos inteligentes, dependiendo de un alcance determinado se lo conoce como canales. Las autoridades certificadoras proporcionan los servicios de identidad a los participantes de la red, que pueden acceder mediante aplicaciones que se comunican vía APIs Rest, previamente desarrollados por Arquitectos y Desarrolladores de software.

Para el desarrollo de las aplicaciones con Blockchain se hace uso de las siguientes herramientas:

HyperLedger Composer, cuyo detalle se muestra en el Capítulo 2, el mismo que permite mediante lenguajes de scripts crear tanto las estructuras como la lógica de negocio. En resumen, esta herramienta permite construir sobre el **HyperLedger Fabric** de forma más amigable el desarrollo de las cadenas de bloque, los participantes, los tipos de acceso y los contratos inteligentes. A continuación, se muestra un resumen gráfico del listado de herramientas y lenguajes que se usaron en el proyecto para el desarrollo de la Cadena de Bloques.

Node Js: Se usa Node js para proveer el backend, y JavaScript para desarrollar la capa que exponer los servicios Rest hacia la aplicación Web.

Visual Studio Code: Se lo uso como editor de código.

Figura 4.3 Ecosistema de lenguajes y herramientas para el desarrollo de aplicaciones Blockchain [11]

Se modelaron haciendo uso de las herramientas las entidades tales como participantes y activos los mismos que se van a ver en el siguiente capítulo.

4.3 Diseño de Componentes Transaccionales

Como la estructura se basa en un libro diario (ledger), las entidades tomarán el nombre de Participantes, Activos, Operaciones, Las transacciones que operan sobre esas entidades se definen como parte de la lógica de negocio, mediante el cual se puede añadir, cambiar los nodos dentro de la cadena de bloques, añadiendo nueva información dentro de la estructura. Se crean además los métodos para mantener cada entidad.

Se dividen los participantes en tres tipos, Productor de activos dentro de un espacio de trabajo (namespace), con el cual se comparte la información de la cadena de bloques:

Para el rol perteneciente al Proveedor de Productos de Fumigación.

Administrador de producción: ProductionManager.

Proveedor de los productos: Provider.

Se define los activos a los que tiene acceso esto roles:

Productos: Products.

Record de Historia de transacciones: HistorianRecord.

El código del diseño del resto de estas entidades se encuentra en el ANEXO 2.

El código Javascript que implementa los métodos de las transacciones de mantenimiento para cada participante dentro de la cadena de bloques se encuentra en ANEXO 3, ANEXO 4,y ANEXO 5, para la actualización de los recursos (assets) tales como el Plan de Fumigación se encuentra en el ANEXO 8, este permite crear, modificar o actualizar el plan durante todo el proceso de seguimiento y registro por parte de los participantes tales como Gerente de la Plantación, Auditor de Fumigación (o Auditor Fitosanitario) y la Empresa Fumigadora. Se creó un método de filtro, que permite dar seguimiento a los participantes y a los recursos, buscando la entidad que tiene un particular valor en alguno de sus atributos (detalle en el ANEXO 6). También se creó un método para obtener el historial de transacciones o acciones realizadas sobre la cadena de bloques (ANEXO 7).

4.4 Diseño de Contratos Inteligentes

La lógica de negocio se la construye como contratos inteligentes, todas las transacciones se basan en el seguimiento del plan de fumigación, creado desde la interfaz por el Gerente de Producción, a la cual tendrá acceso los participantes el Auditor y la Empresa Fumigadora, estos contratos se exponen como servicios REST (GET, POST, PUT) tal como se muestra en el ANEXO 9:

Fumigationcompany: Lógica de transacciones de la compañía de fumigación:

- **addFumigationPlan**: Añadir el plan de fumigación.
- **getFumigationPlan**: Obtener todos los registros de datos del plan de fumigación.
- **getFumigacionPlan (por Id de plan de fumigación)**: Obtener el registro de un plan de fumigación por el plan ID.
- **addFumigacionData**: Añadir datos de la fumigación, usado por la empresa fumigadora.
- **getFumigacionData**: Obtener los datos de la ejecución fumigación, usado por el usuario Administrador de la Empresa bananera.
- **addFumigacionCompany**: Añadir una nueva empresa de Fumigación.

- **getFumigacionData (por el id)**: Obtener la información de los datos de fumigación que ingresan al Blockchain por el Id de fumigación, esos datos son ingresados por la empresa fumigadora.
- **updateFumigacionData**: Actualiza la observación sobre la tarea de fumigación, la fecha de dicha observación, haciendo referencia al id interno de la compañía de fumigación.
- **add**: Permite añadir una empresa fumigadora.

Product: Lógica de transacciones para mantener los productos, crear el Proveedor de productos y Administrador de Producción.

- **addProduct**: Ingresa los datos de los productos.
- **list**: Lista los productos registrados en la cadena (la aplicación la filtra por el proveedor correspondiente).
- **List (por el id de producto generado por el Blockchain)**: Lista de las características de un producto consultado por su código de producto.
- **List (por el id de producto generado por el Blockchain)**: Lista de las características de un producto consultado por su código de producto.
- **addProductionManager**: Permite ingresar los datos del Administrador de Producción (de la empresa bananera).
- **updateProducts (por el idProduct)**: Permite actualizar los datos ingresados de los productos, la usa el Proveedor de los productos.

- **addProvider**: Permite ingresar el proveedor de los Productos de Fumigación.

Auditor: Lógica de transacciones para mantener los productos.

- **saveAuditorData**: Permite los datos de las condiciones para aplicar la fumigación, ingresado por el Auditor de Fumigación y la fecha donde se acuerda aplicar la fumigación.

- **getAuditorData**: Permite consultar todos los datos ingresados por los Auditores, sobre las condiciones para aplicar la fumigación y la fecha tentativa para la ejecución de la misma, por cada transacción se crea un idFito, que es el código de control FitoSanitario.

- **getAuditorData (por el idFito)**: Permite consultar por un id del control Fito sanitario, los datos ingresados sobre las condiciones al aplicar la fumigación y la fecha de ejecución.

- **add:** Permite consultar por un id del control Fito sanitario, los datos ingresados sobre las condiciones al aplicar la fumigación y la fecha de ejecución.

- **updateAuditorData (por idFito):** Permite actualizar la información de las condiciones de fumigación y la fecha ingresada por un Auditor de Fumigación, por el idFito (id de registro fitosanitario).

- **add**: Permite ingresar los datos de un auditor para el mantenimiento de este participante.

- **list**: Permite consultar todos los datos de todos los Auditores de Fumigación.
- **updateAuditor (por auditorId):** Permite actualizar los datos de un auditor especificado en el url.

Métodos para el contrato TransactionHistory: Genera el historial de transacciones.

- **transaction/logs**: Permite obtener el historial de todas las transacciones realizadas desde los logs del Blockchain.

Metodos de filtrado: Permite consultar recurso o un participante, a través de uno de los atributos que lo componen.

- **getFilterData:** Sacar las consultas de recursos (assets) y participantes (participants), por los valores de cualquier atributo que pertenezca a ese recurso o participante.

Metodos para Empresa (Enterprise): Metodos para actualizar y consultar los datos de la empresa bananera.

- **add**: Permite ingresar los datos de referencia de una empresa.
- **list**: Lista totas las empresas ingresadas en la cadena de bloques
- **(sin método- consulta por enterpriseId)**: Permite consultar por el id de una empresa los datos de referencia.

4.5 Diseño del Aplicativo Web

Se construyen 3 aplicaciones web:

1.- Aplicación de la empresa bananera, que accede al sistema y base de datos local, así como al Blockchain para la creación del plan y seguimiento del mismo (ANEXO 10 al ANEXO 14 y del ANEXO 30 al ANEXO 35). El conjunto de opciones que tiene acceso al Blockchain y que es implementado dentro de este proyecto se muestra en la Figura 4.4.

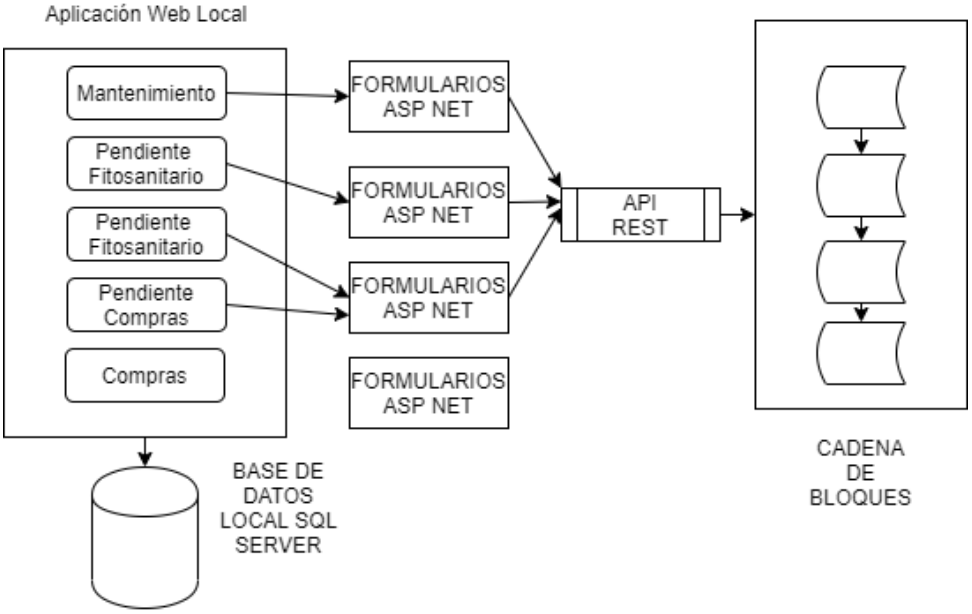


Figura 4.4 Aplicación para la empresa bananera mantenimento y actualización del plan de fumigación

La opción de mantenimiento de Plan, es donde se genera el plan, es el primer paso dentro del flujo de información, que va a ser registrado en la cadena de bloques.

El pendiente Fitosanitario, es actualizado por el auditor de fumigación, al relacionar y registrar los productos con nombre genérico ingresados en el Plan de fumigación, con el nombre de producto comercial y su código SKU obtenido durante el proceso de compras, además se ingresa la fecha estimada de aplicación de los productos y las condiciones en un campo de observación.

La opción Fitosanitario, es una pantalla que sería llevado por un colaborador de la empresa bananera, que funge el rol del participante de la empresa fumigadora dentro de la cadena de bloques, este será encargado de registrar la fecha efectiva de aplicación de los productos de fumigación y las condiciones con las que se aplicó llenando un campo de observación en el formulario, este acceso es para los colaboradores de la empresa bananera.

La opción de compras sobre el cálculo financiero del trabajo realizado no es propósito de este proyecto, esta parte es parte del sistema actual, pero si se apoya en la última transacción generada desde el Blockchain respecto a la actualización de la tarea de fumigación y la fecha realizada.

2.- Aplicación web de Blockchain para mantenimiento de los participantes en la cadena de bloques, esta aplicación consume directamente los servicios rest de mantenimiento de participantes (ANEXO 15 al ANEXO 26).

3.- Aplicación Web de acceso para la empresa de Productos de Fumigación, en esta aplicación el usuario generará la lista de productos por

órdenes de Compra, incluyendo el SKU del producto (ANEXO 27 al ANEXO 29).

4.- Aplicación Web para el registro de la ejecución de los trabajos de fumigación (ANEXO 30 al ANEXO 34).

5.- Aplicación Web para la revisión de los planes pendientes de fumigación y los planes que ya fueron ejecutados sus tareas de fumigación (ANEXO 33 al ANEXO 35).

4.6 Diseño de casos de pruebas y definición de las métricas basado en los criterios de aceptación

A continuación, se listan los escenarios de pruebas.

Escenario 1: Ingreso de órdenes de compra por la compañía proveedora, realizado por el Proveedor de Productos.

Escenario 2: Ingreso de Plan de Fumigación por parte del Gerente de producción.

Escenario 3: Ingreso de la relación de productos contra el plan, basado en la orden de compra.

Escenario 4: Actualización del plan, con la ejecución del trabajo de fumigación.

Escenario 5: Consulta para revisión del cumplimiento del plan por el Administrador Financiero (Compras), a través de la aplicación Web local de la empresa bananera.

Las métricas de evaluación para los escenarios de prueba son:

a) Disminución del tiempo de revisión respecto a la integridad y trazabilidad sobre las transacciones y controles realizados.

b) Disminución en el tiempo de ingreso que se realiza por cada uno de los actores involucrados.

CAPÍTULO 5

5. PRUEBAS E IMPLEMENTACION

5.1 Desarrollo y pruebas de la plataforma blockchain

El desarrollo se realizó contra NodeJS, y aplicando el lenguaje de modelado Hyperledger Composer (CTO) y el HyperLedger Composer Playground para la configuración, del canal del blockchain.

A continuación, se describen los datos de una de las pruebas realizadas:

Escenario 1: Ingreso por Orden de compra 0005872 de la compañía Diximant S.A. por el Proveedor de Productos, para aplicarlo en un plan de fumigación a la plantación “Maria Cecilia”.

La Orden de compra 00004002 tiene 1 item:

1 Odeon.

Este ingreso se lo realiza en la Pantalla de Mantenimiento de productos que se encuentra en el ANEXO 29.

Escenario 2: Ingreso de Plan de Fumigación por Parte Participante Gerente de Producción.

Ingresamos la información del Plan de fumigación:

- Fecha: 2019-08-13.
- Finca: Maria Cecilia.
- Lote: Todos.
- Ciclo: 10 Selección del Combo (indicar) con los productos y su cantidad.
- Combo 14.

Selección de la lista desplegable tipo combo con los productos y su cantidad, los datos ingresados de ejemplo a continuación:

Producto 1: Odeon. Dosis: 1.3 Litros/Hectárea, producto con SKU Odeon-00004001-Dixi.

Escenario 3: Ingreso del Participante Auditor

Selecciona el Plan de Fumigación y hace la relación con los productos de la orden de compra.

Selecciona Plan.

Selecciona los Productos en Base a la orden de compra.

Fecha de registro dentro del plan: 2019-08-24.

Observaciones: “” (campo no obligatorio).

HR (humedad relativa) inicial: 9.

HR (humedad relativa) final: 11.

Vientos (velocidad): 0 (el valor cero indica que es opcional).

Escenario 4: Ingreso de la Empresa Fumigadora

Selecciona el Plan.

Ingresa la Fecha de fumigación: 2019-09-21.

Ingresa Observación (Opcional): Sin novedad.

Escenario 5: Ingreso de revisión de compra en la Aplicación Interna

Selecciona el plan.

Valida los productos y costos.

Autoriza Pago por correo.

5.2 Desarrollo y pruebas de los contratos inteligentes con tecnología Web, en la plataforma Blockchain

Las pruebas sobre los contratos inteligentes fueron realizadas usando la herramienta Postman, para validar la funcionalidad de los contratos a través de requerimientos Http Rest enviados a través de esa herramienta, los resultados de otras pruebas con se adjuntan en el ANEXO 9.

5.3 Resultado y análisis las pruebas

El proceso de registro a través de las aplicaciones web que consumía cada participante, fue realizado con éxito, se identificaron cambios en los formatos de fecha (del tipo estampado de tiempo o timestamp) que se debieron incluir producto de las pruebas realizadas.

Como resultado de las pruebas con usuarios, se pudo evidenciar un acorte de tiempo de semanas a minutos, debido a que las actualizaciones al plan de fumigación tales como: la definición inicial del plan ejecutado por el Administrador de la Empresa Bananera, el registro de los productos comerciales contra el plan ingresado por el representante del proveedor de productos de fumigación, el registro de las fechas para aplicación y validación de los productos comerciales ingresadas por el Auditor de Fumigación, las tareas de fumigación registrada por la empresa de fumigación, y las verificaciones de la ejecución por parte del personal financiero de Compras, se realizaban con transacciones que se ejecutaban en el momento, y que al estar validados que la persona autorizada ingresa la información en el momento, sin perjuicio de pérdida de integridad o manipulación, cumplía el objetivo planteado inicialmente.

Se realizó un ingreso por cada uno de los participantes tanto en el aplicativo web local, así como las subaplicaciones web que tienen acceso a la cadena de bloques (Blockchain), todas corriendo en un servidor en la nube de internet.

CAPÍTULO 6

6. ANÁLISIS DE RESULTADOS

6.1 Revisión del cumplimiento de los criterios de aceptación

Revisando los diferentes requerimientos de la aplicación, al revisar con la empresa bananera, se obtuvo los resultados esperados, detallados a continuación:

Tabla 12. Tabla de cumplimiento de requerimientos funcionales

Código del requerimiento funcional	Nombre del caso de uso:	Observación
GSICO-APP-00001	Generar plan semanal de fumigación	Desarrollado con éxito en el sitio Local de la empresa bananera.
GSICO-APP-00002	Registrar datos de la compra de productos de fumigación en micrositio Web externo	Micrositio de acceso a la Empresa Proveedora, para facilitarle el ingreso de los productos provistos para las tareas de fumigación, sin depender de intercambio manual de información, basta con la confirmación del área de compras
GSICO-APP-00003	Registrar la información de las condiciones y los productos que se van a aplicar dentro del plan de fumigación	Acceso vía micrositio WEB para el Auditor de Fumigación, que le permitió actualizar la tarea de fumigación con los datos requeridos para su ejecución, en cualquier momento al disponer acceso a internet
GSICO-APP-00004	Registrar la ejecución de las tareas de fumigación del plan de fumigación	El encargado o la empresa contratada para fumigacion puede cerrar la tarea registrando en ese mismo instante la labor ejecutada de fumigación
GSICO-APP-00005	Mostrar un listado del reporte de las tareas pendientes de fumigación	Mostrar en línea el estado de las tareas del plan de fumigación pendientes, conforme se van ingresando más planes por ejecutar
GSICO-APP-00006	Revisar tareas ejecutadas de los planes de fumigación	Revisar el detalle de las tareas ejecutadas para calcular los costos asociados a la aplicación de los productos de fumigación, esta tarea se la lleva a cabo en el sitio web local de la empresa bananera

En resumen, por consecuencia de la implementación del software se pudo solucionar los problemas que existían antes de la aplicación del nuevo software, y que se muestran en la siguiente tabla:

Tabla 13. Mediciones antes y después de aplicar el nuevo software

PROBLEMA EXISTENTE ANTES DE LA APLICACIÓN DEL SOFTWARE	REPORTADOS	CANTIDAD CASOS ANTES A LA APLICACIÓN DEL SOFTWARE	RESOLUCION	CANTIDAD CASOS POSTERIOR A LA APLICACIÓN DEL SOFTWARE
Falta de un correcto seguimiento a las transacciones.	Como cada participante maneja un archivo de Excel. Cada quien puede modificar la información y se pierde trazabilidad tanto en el flujo de ingreso como su edición.	9	Uso de Aplicativo Web con Blockchain con un log de transacciones.	0
No hay controles que permitan proteger los datos, de una modificación fraudulenta.	Por 3 meses hubo una colusión entre el personal de la empresa productora con una empresa fumigadora la cual permitía cambiar la fecha de ejecución de fumigación. Esto hacia que se pudiera pagar anticipadamente aunque no se haya ejecutado.	6	El Aplicativo Web permite el control del flujo por estados, lo cual controla el ingreso de los participantes y registra el log en cada transacción.	0
Existe un alto riesgo con el acceso de la información a las entidades externas sobre los aplicativos internos pueden comprometer los datos de la empresa.	Si se le permite el acceso a terceros al servidor de aplicaciones puede ocurrir que modifiquen información importante de la empresa. Con el aplicativo de recursos humanos se hizo una interface para que un contratista ingrese información y tuvo opción de ver información empresarial por falta de seguridad	1	Con la tecnología de cadena de bloques que permite que la información sea compartida entre los participantes, esta información se maneja con la visibilidad que permite el Aplicativo Web	0
El flujo de Información entre las entidades involucradas no mantienen un orden ni un registro sobre su seguimiento, lo cual no permite una trazabilidad del proceso.	Como cada entidad maneja su proceso en forma independiente con archivo de Excel, primero puede llegar la información del auditor sin haber ingresado información de productos. O también llegar la información de la Empresa Fumigadora sin que les haya proporcionado los datos del auditor.	7	El aplicativo Web permite controlar el flujo por estados y la cadena de bloques el log transaccional de los ingresos respectivo.	0
El almacenamiento en base de datos local no permite distribución de información a entidades externas.	En una bases de datos local no se le puede dar acceso a proveedores externos. Número de casos de pedido acceso a nuestra información el cual es restringido.	5	La información en Blockchain permite que sea compartida y no es necesario que sea dependiente de la empresa.	0

Estos cambios permiten a la empresa tener información relevante para la toma de decisiones respecto a los planes de fumigación de forma más ágil.

6.2 Resumen de las lecciones aprendidas

Como recomendación durante el desarrollo, se debe tomar en cuenta las salidas y entradas de todo el proceso de principio a fin, de manera que el

diseño de la solución permita aprovechar la característica de seguimiento que provee Blockchain, a la información una vez ingresada.

El aplicativo web como herramienta de ingreso de información permite que distintos participantes ubicados en áreas geográficas diferentes, puedan llevar a cabo el registro en un menor tiempo, al evitar temas de logística como envió de documentación física, validación manual, autorizaciones formales, etc.

Fomentar el uso de Blockchain, en otros tipos de procesos de negocio dentro de sectores diferentes al financiero, aprovechando la seguridad, inviolabilidad, autointegridad, que posee esta tecnología disruptiva.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

A través de los resultados obtenidos durante la implementación de la aplicación web de apoyo a la gestión del registro y seguimiento del plan de fumigación de los cultivos que realiza una empresa productora de banano, descrito en el presente trabajo de titulación, se pudo evidenciar y concluir, que el uso combinado de las tecnologías de desarrollo web y Blockchain, permite crear nuevas aplicaciones que garanticen la veracidad, autenticidad e integridad de la información ingresada, de manera oportuna, en diferentes modelos de negocio, tanto para empresas grandes como medianas. Se concluye que debe masificarse el uso de esta tecnología para aprovechar sus ventajas y generar mayor competitividad en las empresas transformando sus procesos de operación.

Recomendaciones

Durante el desarrollo e implementación de aplicaciones basados en tecnologías web y Blockchain, se debe tomar en cuenta as características que ofrece el Blockchain respecto a la integridad, al seguimiento que se le puede dar a la información. Tener cuidado con las características tales como la inmutabilidad (un bloque no puede ser cambiado), que conllevan a que en ese sentido la información mal ingresada, debe ser invalidada por la aplicación web que usa los servicios de actualización de la cadena de bloques.

Adquirir nuevos onocimientos sobre mejores prácticas en el desarrollo de aplicaciones web y de aplicaciones Blockchain, estar pendiente de las actualizaciones de las

diferentes versiones de las herramientas de desarrollo y marcos de trabajo, que se generen durante el período de desarrollo e implementación.

BIBLIOGRAFÍA

[1] M. Gupta, *Blockchain For Dummies®, 2nd IBM Limited Edition*, 2nd IBM Limited Edition. US: Wiley, For Dummies, 2018.

[2] Arun Jai, Jerry Cuomo, Nitin Gaur, *Blockchain for Business*. Addison Wesley, 2019.

[3] "¿Qué es el Blockchain? - Diccionario de Marketing 40deFiebre," *40deFiebre*. [Online]. Available: https://www.40defiebre.com/que-es/blockchain. [Accessed: 28-Jul-2019].

[4] Michael Wuehler, Solomon Lederer, Rene Madsen, and Joseph J. Bambara, *Blockchain: a practical guide to developing business, law, and technology solutions*, 1st ed., vol. 20. McGraw-Hill, 2018.

[5] O. Blancarte, "Escalabilidad Horizontal y Vertical," *Oscar Blancarte Blog*, 07-Mar-2017.

[6] T. Laurence, *Blockchain For Dummies*, 1st ed. For Dummies, 2017.

[7] C. Dannen, Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners, 1st ed. Berkely, CA, USA: Apress, 2017.

[8] "Ethereum Smart Contracts in Python: a comprehensive(ish) guide." [Online]. Available: https://hackernoon.com/ethereum-smart-contracts-in-python-a-comprehensive-ish-guide-771b03990988. [Accessed: 10-Aug-2019].

[9] A python interface for interacting with the Ethereum blockchain and ecosystem.: ethereum/web3.py. ethereum, 2019.

[10] Ethereum.org. "Que es Ethereum". [Online]. Disponible en https://ethereum.org/es/what-is-ethereum/.

[11] "Iniciación a IBM Blockchain Platform." [Online]. Disponible: https://cloud.ibm.com/docs/services/blockchain?topic=blockchain-get-started-ibp. [Accessed: 11-Aug-2019].

[12] M. Echebarría Sáenz and Universidad de Valladolid, “Contratos electronicos autoejecutables (smart contract) y pagos con tecnología blockchain,” 2017.

[13] Rick-Anderson, “¿Qué es Web Forms.” [Online]. Disponible: https://docs.microsoft.com/es-es/aspnet/web-forms/what-is-web-forms. [Accessed: 16-Aug-2019].

[14] Á. G. Vieites, Enciclopedia de la Seguridad Informática. 2ª edición. Grupo Editorial RA-MA.

[15] “Precios de IBM Blockchain Platform for IBM Cloud.” [Online]. Disponible: https://cloud.ibm.com/docs/services/blockchain?topic=blockchain-ibp-saas-pricing. [Accessed: 03-Sep-2019].

[16] “IBM Blockchain Platform.” [Online]. Disponible: http://www.blockchaincon.es/assets/presentaciones/IBM_Blockchain_Platform.pdf. [Accessed: 03-Oct-2019].

ANEXOS

ANEXO 1 – DISEÑO DE LAS ENTIDADES

Participantes externos:

Empresa Proveedora de Productos de Fumigación, llamada en ingles Provider.

Empresa Auditora Fitosanitaria o de Fumigación (esta puede o no pertenecer a la empresa productora de banano) llamada en inglés en el desarrollo como Auditor.

Empresa Fumigadora llamada en ingles dcomo FumigationCompany.

Participante interno:

Administrador de Producción llamado en el desarrollo como ProductionManager.

Se crea como activos:

Empresa llamado Enterprise.

Productos (de Fumigación) llamado Products.

Datos ingresados por el Auditor Fitosanitario llamado en el código como DataAuditor.

Plan de fumigación llamado en el código como PlanFumigacion.

Datos de la Fumigación llamado como DataFumigacion.

Transacciones

Una transacción parametrizada dependiendo del recurso a actualizar llamado FumigacionPlan.

Eventos

Un evento llamado changeFumigacionPlan.

Y un último activo para bitacorizar las transacciones, llamado en el desarrollo como HistorianRecord.

ANEXO 2 – ARCHIVO DEL MODELO

(FUMIGACIONPLAN.CTO)

```
namespace org.hyperledger_composer.fumigacion


participant ProductionManager identified by productionId {
  o String productionId
  o String email regex=/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/
  o String firstName
  o String lastName
  o String transactionId optional
}
participant FumigacionCompany identified by companyId {
  o String companyId
  o String email regex=/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/
  o String firstName
  o String lastName
  o String transactionId optional
}
participant Auditor identified by auditorId {
  o String auditorId
  o String email regex=/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/
  o String firstName
```

```
  o String lastName
  o String transactionId optional
}
participant Provider identified by providerId {
  o String providerId
  o String email regex=/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/
  o String firstName
  o String lastName
  o String transactionId optional
}
asset Enterprise identified by enterpriseId {
  o String enterpriseId
  o String email regex=/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/
  o String Name
  o String transactionId optional
}
asset Products identified by productId {
  o String productId
  o String descripcion
  o String SKU
  o String purchase_order
  o String status
  o Double quantity
  o String transactionId optional
```

```
    --> Provider provider
    --> Enterprise enterprise
}
asset DataAuditor identified by idFito {
  o String idFito
  o DateTime realDate
  o Double hrInitial
  o Double hrFinal
  o Double wind
  o String observation
  o String transactionId optional
    --> Auditor auditor
}
asset PlanFumigacion identified by idPlan{
  o String idPlan
  o DateTime date
  o Double cycle
  o String Farm
  o Double product_id1
  o String product_tx1
  o Double product_id2
  o String product_tx2
  o Double product_id3
  o String product_tx3
```

```
  o Double product_id4
  o String product_tx4
  o String idFito optional
  o String fumigacionId optional
  o String auditorId optional
  o String status optional
  o String transactionId optional
    --> Products[] products optional
    --> DataAuditor dataauditor optional
    --> DataFumigacion datafumigacion  optional
    --> Enterprise enterprise
}
asset DataFumigacion identified by fumigacionId {
  o String fumigacionId
  o DateTime  dateExecution
  o String observation
  o String transactionId optional
   --> FumigacionCompany fumigacioncompany
}
transaction FumigacionPlan {
    o String email optional
    o String [] assetType optional
    --> PlanFumigacion planfumigacion optional
    --> Products[] products optional
```

```
    --> DataAuditor dataauditor optional
    --> DataFumigacion datafumigacion optional
}
event changeFumigacionPlan {
}
asset HistorianRecord identified by transactionId {
  o String      transactionId
  o String      transactionType
  o DateTime      transactionTimestamp
}
```

ANEXO 3 – CODIGO LOGICA DE NEGOCIO DE PARTICIPANTE AUDITOR

Código que hace referencia a las transacciones que se pueden hacer sobre el recurso (asset) llamado Auditor (el Auditor de aplicación de los productos Fitosanitarios).

auditor.js

```
const uuid = require('uuid');
//const config = require('../config')
const _ = require('underscore');
const microservice = require('../microservices/service');
const { businessNetworkConnection, fabricClient, channel, peer, event_hub } = require('../app.js');

let assetType = "DataAuditor";

exports.saveAuditorData = (data) => {
   return new Promise(async (resolve, reject) => {
      try {
         var idFito = uuid();
         let auditorData = {
            idFito: idFito.toString(),
```

```
            realDate: new Date(data.realDate),
            hrInitial: data.hrInitial,
            hrFinal: data.hrFinal,
            wind: data.wind,
            observation: data.observation,
            auditor: data.auditor.toString(),
            transactionId: ''
        };
        let Data = await microservice.insertAsset(businessNetworkConnection, auditorData, ["Auditor"], assetType, idFito.toString(), event_hub);
        resolve({ "code": 201, "sucess": true, "message": 'Auditor Data saved in blockchain Successfully', 'idFito': idFito })

    } catch (err) {
        console.log("err.message----------", err.message);
        errMessage = typeof err == 'string' ? err : err.message;
        resolve({ "code": 500, sucess: false, message: errMessage })
    }
  });
}
exports.getAuditorData = () => {
  return new Promise(async (resolve, reject) => {
    try {
```

```
            let Data = await microservice.getAllAssets(businessNetworkConnection,
assetType, null);
            resolve(Data);
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.saveAuditor = (data) => {
    return new Promise(async (resolve, reject) => {
        try {
            let auditorId = uuid();
            let participantData = {
                email: data.email,
                firstName: data.firstName,
                lastName: data.lastName,
                transactionId: ""
            }
            let Data = await microservice.insertParticipant(businessNetworkConnection,
participantData, "Auditor", auditorId, event_hub);
            resolve({ "code": 201, "sucess": true, "message": 'Auditor saved in blockchain
Successfully', "auditorId": auditorId })
```

```
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.getAuditorDataById = (idFito) => {
    return new Promise(async (resolve, reject) => {
        try {
            let Data = await microservice.getAssetById(businessNetworkConnection,
assetType, idFito);
            resolve(Data);
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.updateAuditorDataById = (req) => {
    const assetType = "DataAuditor";
    return new Promise(async (resolve, reject) => {
```

```
        try {
            let AuditorData = {
                realDate: new Date(req.body.realDate),
                observation: req.body.observation,
                hrInitial: req.body.hrInitial,
                hrFinal: req.body.hrFinal,
                wind: req.body.wind,
                auditor: req.body.auditorId
            }
            let data = await microservice.updateAsset(businessNetworkConnection, req.body.fumigationCompanyEmail, assetType, AuditorData, null, ["Auditor"], req.params.idFito, event_hub);
            resolve({ "code": 201, "sucess": true, "message": 'Auditor Data updated' })
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.getAuditors = () => {
    return new Promise(async (resolve, reject) => {
        try {
```

```
            let Data = await microservice.getAllParticipants(businessNetworkConnection,
"Auditor", null);
            resolve(Data);
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.updateAuditorById = (req) => {
    const participantType = "Auditor";
    return new Promise(async (resolve, reject) => {
        try {
            let participantData = {
                firstName: req.body.firstName,
                lastName: req.body.lastName,
                email: req.body.email
            }
            let data = await microservice.updateParticipant(businessNetworkConnection,
participantType, participantData, req.params.auditorId);
            resolve({ "code": 201, "sucess": true, "message": 'Auditor updated' })
        } catch (err) {
            console.log("err.message----------", err.message);
```

```
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
```

ANEXO 4 – CODIGO JAVASCRIPT DE LAS TRANSACCIONES PARA LA EMPRESA BANANERA

Código que hace referencia a las transacciones que se pueden hacer sobre el recurso (asset) llamado Empresa (Enterprise).

enterprise.js

```
const uuid = require('uuid');
const _ = require('underscore');
const microservice = require('../microservices/service');
const { businessNetworkConnection, fabricClient, channel, peer, event_hub } = require('../app.js');

let assetType = "Enterprise";

exports.saveEnterpriseData = (data) => {
    return new Promise(async (resolve, reject) => {
        try {
            var enterpriseId = uuid();
            let enterpriseData = {
                enterpriseId: enterpriseId.toString(),
                email: data.email,
```

```
            Name: data.Name,
        };
        let Data = await microservice.insertAsset(businessNetworkConnection, enterpriseData, [], assetType, enterpriseId.toString(), event_hub);
        resolve({ "code": 201, "sucess": true, "message": 'EnterPrise Data saved in blockchain Successfully', "enterpriseId": enterpriseId })

    } catch (err) {
        console.log("err.message----------", err.message);
        errMessage = typeof err == 'string' ? err : err.message;
        resolve({ "code": 500, sucess: false, message: errMessage })
    }
  });
}


exports.getEnterpriseData = () => {
  return new Promise(async (resolve, reject) => {
    try {
        let Data = await microservice.getAllAssets(businessNetworkConnection, assetType, null);
        resolve(Data);
    } catch (err) {
        console.log("err.message----------", err.message);
```

```
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    });
}

exports.updateEnterpriseData = (req) => {
    return new Promise(async (resolve, reject) => {
        try {
            let EnterpriseData = {
                email: req.body.email,
                Name: req.body.Name
            }
            let Data = await microservice.updateAsset(businessNetworkConnection, null,
assetType, EnterpriseData, null, [], req.params.enterpriseId, event_hub);
            resolve({ "code": 201, "sucess": true, "message": 'Enterprise Data updated' })
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
```

ANEXO 5 – CODIGO JAVASCRIPT DE LAS TRANSACCIONES

Código que hace referencia a las transacciones que se pueden hacer sobre el recurso (asset) llamado Producto (que es usado vía interfaza por el Proveedor de Productos de Fumigación).

product.js

```
const uuid = require('uuid');
const config = require('../config')
const _ = require('underscore');
const { businessNetworkConnection, fabricClient, channel, peer, event_hub } = require('../app.js');
const microservice = require('../microservices/service');
let assetType = "Products";

exports.addProducts = (data) => {
    return new Promise(async (resolve, reject) => {
        try {
            let ProductId = uuid();
            let products = {
                productId: ProductId.toString(),
```

```
            descripcion: data.descripcion,
            SKU: data.SKU,
            purchase_order: data.purchase_order,
            status: data.status,
            quantity: data.quantity,
            transactionId: '',
            provider: data.provider,
            enterprise:data.enterprise
        };
        let Data = await microservice.insertAsset(businessNetworkConnection, products, ["Provider","Enterprise"], assetType, ProductId.toString(), event_hub);
        resolve({ "code": 201, "sucess": true, "message": 'Products saved in blockchain Successfully',"productId":ProductId })

    } catch (err) {
        console.log("err.message----------", err.message);
        errMessage = typeof err == 'string' ? err : err.message;
        resolve({ "code": 500, sucess: false, message: errMessage })
    }
  });
}
exports.fetchAllProducts = () => {
  return new Promise(async (resolve, reject) => {
    try {
```

```
            let Data = await microservice.getAllAssets(businessNetworkConnection,
assetType, null);
            resolve(Data);
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.fetchProductsById = (productId) => {
    return new Promise(async (resolve, reject) => {
        try {
            let Data = await microservice.getAssetById(businessNetworkConnection,
assetType, productId);
            resolve(Data);

        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
```

```
exports.addProductionManager = (data) => {
    return new Promise(async (resolve, reject) => {
        try {
            let productionId = uuid();
            let participantData = {
                email: data.email,
                firstName: data.firstName,
                lastName: data.lastName,
                transactionId: ""
            }
            let Data = await microservice.insertParticipant(businessNetworkConnection, participantData, "ProductionManager", productionId, event_hub);
            resolve({ "code": 201, "sucess": true, "message": 'ProductionManager saved in blockchain Successfully',"productionId":productionId })
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.updateProducts = (req) => {
    return new Promise(async (resolve, reject) => {
        try {
```

```
            let productData = {
                descripcion: req.body.descripcion,
                SKU: req.body.SKU,
                purchase_order: req.body.purchase_order,
                status: req.body.status,
                quantity: req.body.quantity,
                provider: req.body.provider,
                enterprise:req.body.enterprise
            }
            let data = await microservice.updateAsset(businessNetworkConnection, req.body.auditorEmail, assetType, productData, null, ["Provider","Enterprise"], req.params.productId, event_hub);
            resolve({ "code": 201, "sucess": true, "message": 'Product updated' })
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.getProductionManagerById = (productionId) => {
    return new Promise(async (resolve, reject) => {
        try {
            assetType = "ProductionManager"
```

```
        let blockList = [], cnt = 0;
        let ParticipantRegistry = await businessNetworkConnection.getParticipantRegistry(`${config.ns}.${assetType}`);
        let productionData = await ParticipantRegistry.get(productionId.toString());
        if (productionData.length == 0) {
            resolve({ "code": 200, "sucess": true, "payload": blockList })
        }
        let transactionDetails = {};
        let response_payload = await channel.queryBlockByTxID(productionData.transactionId, peer);
        let assets = response_payload.data.data[0].payload.data.actions[0].payload.action.proposal_response_payload.extension.results.ns_rwset[0].rwset.writes;
        let blockChainData = JSON.parse(assets[2].value);
        transactionDetails.block_data = {
            email: blockChainData.resources[0].email,
            firstName: blockChainData.resources[0].firstName,
            lastName: blockChainData.resources[0].lastName
        }
        transactionDetails.previous_hash = response_payload.header.previous_hash;
        transactionDetails.current_hash = response_payload.header.data_hash;
        transactionDetails.blockNumber = response_payload.header.number;
        blockList.push(transactionDetails);
```

```
            resolve({ "code": 200, "sucess": true, "payload": blockList })


        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 404, sucess: false, message: errMessage })
        }
    });
}
exports.addProvider = (data) => {
    return new Promise(async (resolve, reject) => {
        try {
            let providerId = uuid();
            let participantData = {
                email: data.email,
                firstName: data.firstName,
                lastName: data.lastName,
                transactionId: ""
            }
            let Data = await microservice.insertParticipant(businessNetworkConnection,
participantData, "Provider", providerId, event_hub);
            resolve({ "code": 201, "sucess": true, "message": 'Provider saved in blockchain
Successfully',"providerId":providerId })
        }
```

```
        catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    })
}
exports.getProductionManagers = () => {
    return new Promise(async (resolve, reject) => {
        try {
            let Data = await microservice.getAllParticipants(businessNetworkConnection,
"ProductionManager", null);
            resolve(Data);
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.getProviderData = () => {
    return new Promise(async (resolve, reject) => {
        try {
```

```
            let Data = await microservice.getAllParticipants(businessNetworkConnection,
"Provider", null);
            resolve(Data);
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    });
}

exports.updateProductionManager = (req) => {
    const participantType = "ProductionManager";
    return new Promise(async (resolve, reject) => {
        try {
            let participantData = {
                firstName: req.body.firstName,
                lastName: req.body.lastName,
                email: req.body.email
            }
            let data = await microservice.updateParticipant(businessNetworkConnection,
participantType, participantData, req.params.productionId);
            resolve({ "code": 201, "sucess": true, "message": 'ProductionManager
updated' })
```

```
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}

exports.updateProvider = (req) => {
    const participantType = "Provider";
    return new Promise(async (resolve, reject) => {
        try {
            let participantData = {
                firstName: req.body.firstName,
                lastName: req.body.lastName,
                email: req.body.email
            }
            let data = await microservice.updateParticipant(businessNetworkConnection,
participantType, participantData, req.params.providerId);
            resolve({ "code": 201, "sucess": true, "message": 'Provider updated' })
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
```

```
        }
    });
}
```

ANEXO 6 – CODIGO JAVASCRIPT PARA FILTRADO DE DATOS

Código que hace referencia los filtros que se pueden hacer contra los datos.

filter.js

```
const uuid = require('uuid');
const _ = require('underscore');
const microservice = require('../microservices/service');
const { businessNetworkConnection, fabricClient, channel, peer, event_hub } = require('../app.js');
exports.getFilterData = async (req) => {
    return new Promise(async (resolve, reject) => {
        let keys, values, Data;
        keys = _.keys(req.query);
        values = _.values(req.query);
        keys[0] == "assetName" ? Data = await microservice.getAllAssets(businessNetworkConnection, values[0], req) : Data = await microservice.getAllParticipants(businessNetworkConnection, values[0], req)
        resolve(Data)
```

```
    })
}                                                   let Data = await
microservice.insertPlan(businessNetworkConnection,FumigacionPlan,assetType,idP
lan,event_hub);
            resolve({ "code": 201, "sucess": true, "message": 'Fumigacion Plan saved in
blockchain Successfully',"planId":idPlan })
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.getFumigacionPlan = () => {
    let assetType = "PlanFumigacion";
    return new Promise(async (resolve, reject) => {
        try {
            let Data = await microservice.getAllAssets(businessNetworkConnection,
assetType, null);
            resolve(Data);
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
```

```
        }
    });
}
exports.getFumigacionPlanById = (planId) => {
    let assetType = "PlanFumigacion";
    return new Promise(async (resolve, reject) => {
        try {
            let Data = await microservice.getAssetById(businessNetworkConnection,
assetType, planId);
            resolve(Data);
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.addFumigacionData = (data) => {
    let assetType = "DataFumigacion", realtionshipType = "FumigacionCompany";
    return new Promise(async (resolve, reject) => {
        try {
            let fumigacionId = uuid();
            let FumigacionData = {
                fumigacionId: fumigacionId.toString(),
```

```
            dateExecution: new Date(data.dateExecution),
            observation: data.observation,
            transactionId: '',
            fumigacioncompany: data.fumigacioncompany
        };
        let Data = await microservice.insertAsset(businessNetworkConnection, FumigacionData, ["FumigacionCompany"], assetType, fumigacionId, event_hub);
        resolve({ "code": 201, "sucess": true, "message": 'Fumigacion Data saved in blockchain Successfully',"fumigacionId":fumigacionId })

    } catch (err) {
        console.log("err.message----------", err.message);
        errMessage = typeof err == 'string' ? err : err.message;
        resolve({ "code": 500, sucess: false, message: errMessage })
    }
  });
}
exports.getFumigacionData = () => {
  let assetType = "DataFumigacion";
  return new Promise(async (resolve, reject) => {
    try {
      let Data = await microservice.getAllAssets(businessNetworkConnection, assetType, null);
      resolve(Data);
```

```
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.addFumigacionCompany = (data) => {
    return new Promise(async (resolve, reject) => {
        try {
            let companyId = uuid();
            let participantData = {
                email: data.email,
                firstName: data.firstName,
                lastName: data.lastName,
                transactionId: ""
            }
            let Data = await microservice.insertParticipant(businessNetworkConnection,
participantData, "FumigacionCompany", companyId, event_hub);
            resolve({ "code": 201, "sucess": true, "message": 'FumigacionCompany saved
in blockchain Successfully',"companyId":companyId })
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
```

```
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.getFumigacionDataById = (fumigacionId) => {
    let assetType = "DataFumigacion";
    return new Promise(async (resolve, reject) => {
        try {
            let Data = await microservice.getAssetById(businessNetworkConnection,
assetType, fumigacionId);
            resolve(Data);
        } catch (err) {
            //console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.updateFumigacionData = (req) => {
    const assetType = "DataFumigacion";
    return new Promise(async (resolve, reject) => {
        try {
            let FumigacionData = {
                dateExecution: new Date(req.body.dateExecution),
```

```
            observation: req.body.observation,
            fumigacioncompany: req.body.fumigacioncompany,
        }
        let data = await microservice.updateAsset(businessNetworkConnection, req.body.productionManagerEmail, assetType, FumigacionData, null, ["FumigacionCompany"], req.params.fumigacionId,event_hub);
        resolve({ "code": 200, sucess: true, message: "Fumigacion Data updated" })
    } catch (err) {
        console.log("err.message----------", err.message);
        errMessage = typeof err == 'string' ? err : err.message;
        resolve({ "code": 500, sucess: false, message: errMessage })
    }
  });
}
exports.updateFumigacionPlan = (req) => {
  let assetType = "PlanFumigacion";
  return new Promise(async (resolve, reject) => {
    try {
        if(req.body.dataauditor){
            req.body.dataauditor.realDate =new Date()
        }
        if(req.body.datafumigacion){
            req.body.datafumigacion.dateExecution =new Date()
        }
```

```
let FumigacionPlan = {
   // date: new Date(),
   date: new Date(req.body.date),
   cycle: req.body.cycle,
   Farm: req.body.Farm,
   product_id1: req.body.product_id1,
   product_tx1: req.body.product_tx1,
   product_id2: req.body.product_id2,
   product_tx2: req.body.product_tx2,
   product_id3: req.body.product_id3,
   product_tx3: req.body.product_tx3,
   product_id4: req.body.product_id4,
   product_tx4: req.body.product_tx4,
   enterprise:req.body.enterprise,
   idFito:req.body.idFito,
   fumigacionId:req.body.fumigacionId,
   auditorId:req.body.auditorId,
   status:req.body.status,
    products: req.body.products,
    dataauditor: req.body.dataauditor,
    datafumigacion: req.body.datafumigacion
}
```

```
        let data = await microservice.updatePlan(businessNetworkConnection, null, assetType, FumigacionPlan, null, ["Products", "DataAuditor", "DataFumigacion"], req.params.planId,req.body.emails,event_hub);
        //let data = await microservice.updatePlan(businessNetworkConnection, null,FumigacionPlan,["Products","DataAuditor","DataFumigacion"],assetType,req.params.planId);
        resolve({ "code": 201, "sucess": true, "message": 'Fumigacion Plan updated' })

      } catch (err) {
        console.log("err.message----------", err);
        errMessage = typeof err == 'string' ? err : err.message;
        resolve({ "code": 500, sucess: false, message: errMessage })
      }
    });
}
exports.getFumigacionCompanies = () => {
  return new Promise(async (resolve, reject) => {
    try {
      let Data = await microservice.getAllParticipants(businessNetworkConnection, "FumigacionCompany", null);
      resolve(Data);
    } catch (err) {
      console.log("err.message----------", err.message);
      errMessage = typeof err == 'string' ? err : err.message;
```

```
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
exports.updateFumigacionCompany = (req) => {
    const participantType = "FumigacionCompany";
    return new Promise(async (resolve, reject) => {
        try {
            let participantData = {
                firstName: req.body.firstName,
                lastName: req.body.lastName,
                email: req.body.email
            }
            let data = await microservice.updateParticipant(businessNetworkConnection,
participantType, participantData, req.params.companyId);
            resolve({ "code": 201, "sucess": true, "message": 'FumigacionCompany
updated' })
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
```

ANEXO 7 – CODIGO JAVASCRIPT PARA CONSULTA DE HISTORIAL DE TRANSACCIONES

Servicio para obtener histórico de transacciones (no expuesto como parte de los servicios Web).

transactionHistory.js

```
const { businessNetworkConnection } = require('../app.js');
const microservice = require('../microservices/service');
exports.getTransactionHistory = () => {
    return new Promise(async (resolve, reject) => {
        try {
            let transactions = await
microservice.getTransactionHistory(businessNetworkConnection);
            resolve(transactions);
        } catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    });
}
```

ANEXO 8 – CODIGO JAVASCRIPT DE LAS TRANSACCIONES GENERALES CONTRA LOS ACTIVOS

Funciones de acceso a través de un servicio rest.

```
const config = require('../config');
const microservice = require('../microservices/service.js');
const Email = require('../email.service.js');
const _ = require('underscore');
const request = require('request-promise');

exports.updatePlan = (businessNetworkConnection, email, assetType, data, realtionshipType, key, id, emails, event_hub) => {
   return new Promise(async (resolve, reject) => {
      try {
         let factoryAssign = businessNetworkConnection.getBusinessNetwork().getFactory();
         let mainAssetRegistry = await businessNetworkConnection.getAssetRegistry(`${config.ns}.${assetType}`);
         let assetData = await mainAssetRegistry.get(id.toString());
         let keys = Object.keys(data);
```

```
        String.prototype.capitalize = function () {
            return this.charAt(0).toUpperCase() + this.slice(1);
        }
        // let factoryAssign =
businessNetworkConnection.getBusinessNetwork().getFactory();
        let set = 0;
        // console.log("DataAuditor==========", keys);
        let transactionData = {};
        await keys.map(async (k, index) => {
            if (data[k] != undefined) {
                let assetType = k == 'products' ? 'Products' :
                    k == 'dataauditor' ? 'DataAuditor' :
                        k == 'enterprise' ? 'Enterprise' : 'DataFumigacion';
                // let assetRealation = k == 'products' ? 'provider' :
                //     k == 'dataauditor' ? 'auditor' : 'fumigacioncompany';
                // let newAssetId = uuid();

                // let capitalAssetRealtion = assetRealation.capitalize();

                // capitalAssetRealtion = capitalAssetRealtion == 'Auditor' ? 'Auditor'
                //                 :    capitalAssetRealtion    ==    'Fumigacioncompany'    ?
'FumigacionCompany'
                //         : capitalAssetRealtion;
```

```
                // //       console.log("all==========", assetType, assetRealation,
capitalAssetRealtion);
                // let assetRegistry = await
businessNetworkConnection.getAssetRegistry(`${config.ns}.${assetType}`);
                // // let Data = await
microservice.insertAsset(businessNetworkConnection, data[k],
[capitalAssetRealtion], assetType, newAssetId.toString(),event_hub);
                // // console.log("DDD",Data);
                //  resource = factoryAssign.newResource(config.ns, assetType,
newAssetId);
                //  let relationAssign = factoryAssign.newRelationship(config.ns,
capitalAssetRealtion, data[k][assetRealation]);
                // delete data[k][assetRealation]
                // resource[assetRealation] = relationAssign
                // resource = Object.assign(resource, data[k]);
                // let assetAdded = await assetRegistry.add(resource);
                //  let Event = await microservice.triggerEvent(event_hub, newAssetId,
"asset", assetRegistry);
                //  relationAssign = factoryAssign.newRelationship(config.ns, assetType,
newAssetId);
                if (k == 'products') {
                    let Relation = [];
                    data[k].map((productId) => {
```

```
                    let relationAssign = factoryAssign.newRelationship(config.ns,
assetType, productId);
                    if (assetData[k]) {
                        assetData[k].push(relationAssign);
                    } else {
                        Relation.push(relationAssign);
                        assetData[k] = Relation;
                    }
                })
                transactionData[k] = assetData[k]

            } else if (k == 'dataauditor' || k == 'datafumigacion' || k == 'enterprise') {
                let relationAssign = factoryAssign.newRelationship(config.ns,
assetType, data[k]);
                assetData[k] = relationAssign;
                if (k != 'enterprise') {
                    transactionData[k] = assetData[k];
                }
            } else {
                assetData[k] = data[k];
            }
        }
        set++;
        if (set == keys.length) {
```

```
                await mainAssetRegistry.update(assetData);
            // microservice.triggerEvent(event_hub, id, "asset", mainAssetRegistry);
                const newTransaction = factoryAssign.newTransaction(config.ns, "FumigacionPlan");
                trans = Object.assign(newTransaction, transactionData);
                let res = businessNetworkConnection.submitTransaction(trans);
                let eventCount = 0;
                businessNetworkConnection.on('event', async (event) => {
                    console.log("email ecvenr====================")
                    eventCount++;
                    let message;
                    if (eventCount == 1) {
                        emails.map((email) => {
                            email.type == "product" ? message = "Hello Product information in Fumigacion Plan" : email.type == "auditor" ? message = "Hello Auditor information in Fumigacion Plan" : message = "Hello Fumigacion Company information in Fumigacion Plan"
                            Email.sendEmail(email.Email, message)
                        })
                    }
                })
                resolve(true);
            }
        });
```

```
        }
        catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    })
}
exports.updateAsset = (businessNetworkConnection, email, assetType, data, realtionshipType, key, id, event_hub) => {
    return new Promise(async (resolve, reject) => {
        try {
            let factory = businessNetworkConnection.getBusinessNetwork().getFactory();
            let assetRegistry = await businessNetworkConnection.getAssetRegistry(`${config.ns}.${assetType}`);
            let assetData = await assetRegistry.get(id.toString());
            let keys = Object.keys(data);
            keys.map(async (i) => {
                let relationtype = null, productRelations = [];
                await key.find((relation) => {
                    if (relation.toUpperCase() == i.toUpperCase() && data[i] != undefined) {
                        relationtype = factory.newRelationship(config.ns, relation, data[i]);
                    }
                })
```

```
            assetData[i] = i == 'products' ? productRelations :
                relationtype ? relationtype : data[i]
        })
        let assetUpdated = assetRegistry.update(assetData)
        //   microservice.triggerEvent(event_hub, id, "asset", assetRegistry);
        resolve(assetUpdated);
    }
    catch (err) {
        console.log("err.message----------", err.message);
        errMessage = typeof err == 'string' ? err : err.message;
        reject({ "code": 500, sucess: false, message: errMessage })
    }
  })
}
exports.insertPlan = (businessNetworkConnection, data, assetType, id, event_hub)
=> {
  return new Promise(async (resolve, reject) => {
    try {
        let resource, relation;
        let factory = businessNetworkConnection.getBusinessNetwork().getFactory();
        let assetRegistry = await
businessNetworkConnection.getAssetRegistry(`${config.ns}.${assetType}`);
        resource = factory.newResource(config.ns, assetType, id);
        relation = factory.newRelationship(config.ns, "Enterprise", data.enterprise);
```

```
            delete data.enterprise;
            data.enterprise = relation;
            resource = Object.assign(resource, data);
            let assetAdded = await assetRegistry.add(resource);
            microservice.triggerEvent(event_hub, id, "asset", assetRegistry);
            const newTransaction = factory.newTransaction(config.ns,
"FumigacionPlan");
            relationtype = factory.newRelationship(config.ns, "PlanFumigacion", id);
            let transactionData = {
                planfumigacion: relationtype
            }
            trans = Object.assign(newTransaction, transactionData);
            let res = businessNetworkConnection.submitTransaction(trans);
            resolve(assetAdded);
        }
        catch (err) {
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    })
}
exports.insertAsset = (businessNetworkConnection, data, key, assetType, id,
event_hub) => {
    return new Promise(async (resolve, reject) => {
```

```
        try {
            let resource;
            let factory = businessNetworkConnection.getBusinessNetwork().getFactory();
            let assetRegistry = await
businessNetworkConnection.getAssetRegistry(`${config.ns}.${assetType}`);
            let keys = Object.keys(data);
            resource = factory.newResource(config.ns, assetType, id);
            await keys.map(async (i) => {
                let relationtype = null;
                await key.find((relation) => {
                    if (relation.toUpperCase() == i.toUpperCase() && data[i] != undefined) {
                        relationtype = factory.newRelationship(config.ns, relation, data[i]);
                        delete data[i];
                        resource[i] = i == 'products' ? productRelations :
                            relationtype ? relationtype : data[i]
                    }
                })
            })
            resource = Object.assign(resource, data);
            let assetAdded = await assetRegistry.add(resource);
            microservice.triggerEvent(event_hub, id, "asset", assetRegistry);
            resolve(assetAdded);
        }
        catch (err) {
```

```
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    })
}
exports.insertParticipant = (businessNetworkConnection, data, participantType, id, event_hub) => {
    return new Promise(async (resolve, reject) => {
        try {
            let factory = businessNetworkConnection.getBusinessNetwork().getFactory();
            let resource = factory.newResource(config.ns, participantType, id);
            let participantRegistry = await businessNetworkConnection.getParticipantRegistry(`${config.ns}` + '.' + participantType);
            resource = Object.assign(resource, data);
            let participantAdded = await participantRegistry.add(resource);
            microservice.triggerEvent(event_hub, id, "participant", participantRegistry, null, null, null);
            resolve(participantAdded);
        }
        catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
```

```
        }
    })
}
exports.triggerEvent = (event_hub, id, registrytype, Registry) => {
    return new Promise(async (resolve, reject) => {
        try {
            let eventCount = 0
            event_hub.connect();
            event_hub.registerBlockEvent(async (block) => {
                if (registrytype == 'participant') {
                    let participantdata = await Registry.get(id.toString());
                    participantdata.transactionId = block.filtered_transactions[0].txid;
                    Registry.update(participantdata)
                }
                else {
                    let assetData = await Registry.get(id.toString());
                    assetData.transactionId = block.filtered_transactions[0].txid;
                    Registry.update(assetData)

                }
                event_hub.disconnect();
            })

        }
```

```
        catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    })
}
exports.getAllAssets = (businessNetworkConnection, assetType, req) => {
    return new Promise(async (resolve, reject) => {
        try {
            let blockList = [], cnt = 0, assetList = [], response = [];
            let assetRegistry = await businessNetworkConnection.getAssetRegistry(`${config.ns}.${assetType}`);
            let fumigacionAssetList = await assetRegistry.getAll();
            assetList = fumigacionAssetList;
            let allAssetString = JSON.stringify(assetList);
            let allAssetArray = JSON.parse(allAssetString);

            if (allAssetArray.length == 0) {
                resolve({ "code": 200, "sucess": true, "payload": blockList })
            }
            if (req != null) {
                let filterCounter = 0;
                // console.log("inside");
```

```
            let Keys = Object.keys(req.query);
            await Keys.map(async (key) => {
                filterCounter++;
                let filteredData = _.filter(allAssetArray, function (filterData) { return
filterData[key] == req.query[key] });
                if (filterCounter == Keys.length) {
                    allAssetArray = filteredData


                }
            })
        }
        if (allAssetArray.length) {
            for (j = 0; j < allAssetArray.length; j++) {
                let fumigacionData = [], products = [], auditor = [], response = [];
                let transactionDetails = {};
                if (allAssetArray[j].products) {
                    for (i = 0; i < allAssetArray[j].products.length; i++) {
                        let productResources = allAssetArray[j].products[i];
                        let productIds = productResources.split('#');
                        let productId = productIds[1];
                        await        microservice.getAssetById(businessNetworkConnection,
"Products", productId).then((Data) => {
                            if (Data.payload) {
                                products.push(Data.payload[0].block_data);
```

```
                    }
                })


            }
        }
        if (allAssetArray[j].dataauditor) {
            auditorResources = allAssetArray[j].dataauditor;
            let auditorIds = auditorResources.split('#');
            let auditorId = auditorIds[1];
            await microservice.getAssetById(businessNetworkConnection,
"DataAuditor", auditorId).then((Data) => {
                if (Data.payload) {
                    auditor.push(Data.payload[0].block_data);
                }
            });
        }
        if (allAssetArray[j].datafumigacion) {
            let fumigacionResources = allAssetArray[j].datafumigacion;
            auditorResources = allAssetArray[j].dataauditor;
            let fumigacionIds = fumigacionResources.split('#');
            let fumigacionId = fumigacionIds[1];
            await microservice.getAssetById(businessNetworkConnection,
"DataFumigacion", fumigacionId).then((Data) => {
                if (Data.payload) {
```

```
                    fumigacionData.push(Data.payload[0].block_data);
                }
            })
        }
        let keys = Object.keys(allAssetArray[j]);
        transactionDetails.block_data = {}
        await keys.map(async (i) => {
            transactionDetails.block_data[i] =
                i == 'products' ? products :
                    i == 'dataauditor' ? auditor :
                        i == 'datafumigacion' ? fumigacionData :
                            allAssetArray[j][i];
        })
        blockList.push(transactionDetails);
        cnt++;
        if (allAssetArray.length == cnt) {
            resolve({ "code": 200, "sucess": true, "payload": blockList })
        }
    }
}
else {
    resolve({ "code": 200, "sucess": true, "payload": blockList })
}
}
```

```
        catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    })
}
exports.getAllParticipants = (businessNetworkConnection, participantType, req) => {
    return new Promise(async (resolve, reject) => {
        try {
            let blockList = [], cnt = 0, assetList = [], response = [];
            let participantRegistry = await businessNetworkConnection.getParticipantRegistry(`${config.ns}.${participantType}`);
            let participantList = await participantRegistry.getAll();
            let allParticipantString = JSON.stringify(participantList);
            let allParticipantArray = JSON.parse(allParticipantString);

            if (allParticipantArray.length == 0) {
                resolve({ "code": 200, "sucess": true, "payload": blockList })
            }
            if (req != null) {
                let filterCounter = 0;
                let Keys = Object.keys(req.query);
```

```
            await Keys.map(async (key) => {
                filterCounter++;
                let filteredData = _.filter(allParticipantArray, function (filterData) { return
filterData[key] == req.query[key] });
                if (filterCounter == Keys.length) {
                    allParticipantArray = filteredData


                }
            })
        }
        if (allParticipantArray.length) {
            for (j = 0; j < allParticipantArray.length; j++) {
                let transactionDetails = {};
                let keys = Object.keys(allParticipantArray[j]);
                transactionDetails.block_data = {}
                await keys.map(async (i) => {
                    transactionDetails.block_data[i] = allParticipantArray[j][i];
                })
                blockList.push(transactionDetails);
                cnt++;
                if (allParticipantArray.length == cnt) {
                    resolve({ "code": 200, "sucess": true, "payload": blockList })
                }
            }
```

```
            }
            else {
                resolve({ "code": 200, "sucess": true, "payload": blockList })
            }
        }
        catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    })
}
exports.getAssetById = async (businessNetworkConnection, assetType, assetId) => {
    return new Promise(async (resolve, reject) => {
        try {
            let blockList = [], cnt = 0;
            let fumigacionData = [], products = [], auditor = []
            let assetRegistry = await businessNetworkConnection.getAssetRegistry(`${config.ns}.${assetType}`);
            let fumigacionAssetList = await assetRegistry.get(assetId.toString());
            let stringifyFumigacionAsset = JSON.stringify(fumigacionAssetList);
            let fumigacionAssetById = JSON.parse(stringifyFumigacionAsset);
            let transactionDetails = {};
            if (fumigacionAssetById.products) {
```

```
            for (i = 0; i < fumigacionAssetById.products.length; i++) {
                let productResources = fumigacionAssetById.products;
                let productIds = productResources[i].split('#'),
                    productId = productIds[1];
                await microservice.getAssetById(businessNetworkConnection,
"Products", productId).then((Data) => {
                    if (Data.payload) {
                        products.push(Data.payload[0].block_data);
                    }
                })
                // let productAssetList = await assetRegistry.get(productId.toString());
                // products = productAssetList.length ? products.push(productAssetList) :
[];

            }
        }
        if (fumigacionAssetById.dataauditor) {
            auditorResources = fumigacionAssetById.dataauditor;
            let auditorIds = auditorResources.split('#');
            auditorId = auditorIds[1];
            await microservice.getAssetById(businessNetworkConnection,
"DataAuditor", auditorId).then((Data) => {
                if (Data.payload) {
                    auditor.push(Data.payload[0].block_data);
```

```
            }
        });
        // let auditorAssetList = await assetRegistry.get(auditorId.toString());
        // auditor = auditorAssetList.length ? auditor.push(auditorAssetList) : [];
    }
    if (fumigacionAssetById.datafumigacion) {
        fumigacionResources = fumigacionAssetById.datafumigacion;
        let fumigacionId = fumigacionResources.split('#')[1];
        await microservice.getAssetById(businessNetworkConnection,
"DataFumigacion", fumigacionId).then((Data) => {
            if (Data.payload) {
                fumigacionData.push(Data.payload[0].block_data);
            }
        })
        // let fumigacionAssetList = await
assetRegistry.get(fumigacionId.toString());
        // fumigacionData = fumigacionAssetList.length ?
fumigacionData.push(fumigacionAssetList) : [];
    }
    let keys = Object.keys(fumigacionAssetById);
    transactionDetails.block_data = {}
    await keys.map(async (i) => {
        transactionDetails.block_data[i] =
            i == 'products' ? products :
```

```
                    i == 'dataauditor' ? auditor :
                        i == 'datafumigacion' ? fumigacionData :
                            fumigacionAssetById[i];
            })
            blockList.push(transactionDetails);
            resolve({ "code": 200, "sucess": true, "payload": blockList })
        }
        catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            resolve({ "code": 500, sucess: false, message: errMessage })
        }
    })
}
exports.apiRealtion = (data, url) => {
    return new Promise(async (resolve, reject) => {
        let id = data.split('#')[1];
        await request(url + id).then(function (response, err) {
            let getInfo = JSON.parse(response);
            if (getInfo.payload) {
                resolve(getInfo.payload[0].block_data);
            }
            else {
                resolve(null);
```

```
            }
        })
    })
}
exports.getTransactionHistory = (businessNetworkConnection) => {
    return new Promise(async (resolve, reject) => {
        let historian = await businessNetworkConnection.getHistorian();
        let historianRecords = await historian.getAll();
        resolve(historianRecords)
    })
}
exports.apiUrlCall = (url) => {
    return new Promise(async (resolve, reject) => {
        await request(url).then(function (response, err) {
            let getInfo = JSON.parse(response);
            if (getInfo) {
                resolve(getInfo);
            }
            else {
                resolve([]);
            }
        })
    })
}
```

```
exports.updateParticipant = (businessNetworkConnection, participantType, data, id)
=> {
    return new Promise(async (resolve, reject) => {
        try {
            let participantRegistry = await
businessNetworkConnection.getParticipantRegistry(`${config.ns}.${participantType}`
);
            let participantData = await participantRegistry.get(id.toString());
            let keys = Object.keys(data);
            keys.map(async (i) => {
                participantData[i] = data[i];
            })
            let participantUpdated = participantRegistry.update(participantData)
            resolve(participantUpdated);
        }
        catch (err) {
            console.log("err.message----------", err.message);
            errMessage = typeof err == 'string' ? err : err.message;
            reject({ "code": 500, sucess: false, message: errMessage })
        }
    })
}
```

ANEXO 9 – SERVICIOS HTTP REST, REQUEST Y RESPONSE

Métodos para el contrato Fumigacioncompany.

addFumigationPlan: Añadir el plan de fumigación.

Metodo Put

http://164.68.126.181/rest/fumigacioncompany/addFumigacionPlan

{

"date": "2020-01-12T22:42:34.200Z",

"cycle": 0,

"Farm": "ttt",

"product_id1": 0,

"product_tx1": "666",

"product_id2": 0,

"product_tx2": "234",

"product_id3": 0,

"product_tx3": "555",

"product_id4": 0,

"product_tx4": "678",

"enterprise":"fg700a78-244d-42aa-84a2-36444ff4e880",

"idFito":"h8e0m8a0n3t0",

"fumigacionId":"f7r8t5y55u6r6y5u5i",

"auditorId":"newfb78458798855",

```
  "status":"ACTIVE"
}
```


getFumigationPlan Obtener todos los datos históricos del plan de fumigación

Metodo Get

http://164.68.126.181/rest/fumigacioncompany/getFumigacionPlan


```
{
   "code": 200,
   "sucess": true,
   "payload": [
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
            "idPlan": "064e6e53-0c61-4824-929b-8b03a997196e",
            "date": "2020-01-11T05:00:00.000Z",
            "cycle": 10,
            "Farm": "Clemencia",
            "product_id1": 5,
            "product_tx1": "Paladium",
            "product_id2": 2,
            "product_tx2": "Dithane",
            "product_id3": 3,
            "product_tx3": "Emulad",
            "product_id4": 11,
            "product_tx4": "Aceite",
```

```
            "fumigacionId": "44db83a6-01df-4f13-9595-0aec4ae96a78",
            "status": "ACTIVE",
            "transactionId": "3dc40cbe3ab42a77ac55aa3db6113539fcde9cc956ea5df1
ce5bf7888a80adb6",
             "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
            "idPlan": "0f4d1ff2-abf6-416a-8fd5-c048e7368721",
            "date": "2020-01-11T05:00:00.000Z",
            "cycle": 6,
            "Farm": "1",
            "product_id1": 5,
            "product_tx1": "Paladium",
            "product_id2": 2,
            "product_tx2": "Dithane",
            "product_id3": 3,
            "product_tx3": "Emulad",
            "product_id4": 11,
            "product_tx4": "Aceite",
            "fumigacionId": "44db83a6-01df-4f13-9595-0aec4ae96a78",
```

```
            "status": "ACTIVE",
             "transactionId": "bcf33e8abe994de2887bf33d0f894fb67dfccd9208d13b6fc
d43bd8cd0388d30",
             "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
          }
       },
       {
          "block_data": {
             "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
             "idPlan": "3a99a765-b9ed-445e-8668-216229e2648a",
             "date": "2020-01-11T05:00:00.000Z",
             "cycle": 10,
             "Farm": "Clemencia",
             "product_id1": 4,
             "product_tx1": "Bravo",
             "product_id2": 2,
             "product_tx2": "Dithane",
             "product_id3": 13,
             "product_tx3": "Ninguno",
             "product_id4": 12,
             "product_tx4": "Agua",
             "fumigacionId": "44db83a6-01df-4f13-9595-0aec4ae96a78",
             "status": "ACTIVE",
```

```
            "transactionId": "1a0893c32119da0b3f62b5bc426ac62f2c0616084283869
77e3c6ac17e641e26",
            "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
            "idPlan": "4677d3d1-392f-4778-90a3-0ca8980b71a1",
            "date": "2020-01-11T05:00:00.000Z",
            "cycle": 5,
            "Farm": "Clemencia",
            "product_id1": 1,
            "product_tx1": "Cumora",
            "product_id2": 2,
            "product_tx2": "Dithane",
            "product_id3": 3,
            "product_tx3": "Emulad",
            "product_id4": 11,
            "product_tx4": "Aceite",
            "fumigacionId": "44db83a6-01df-4f13-9595-0aec4ae96a78",
            "status": "ACTIVE",
```

```
            "transactionId": "c1eb51d79ee56d7fd6eba06546ac48d38047f776ea9f9f9fa
c1e9b2a85567804",
             "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
            "idPlan": "5ee26437-645b-4669-a5d1-19ed1a30d21b",
            "date": "2020-01-11T05:00:00.000Z",
            "cycle": 1,
            "Farm": "Clemencia",
            "product_id1": 1,
            "product_tx1": "Cumora",
            "product_id2": 2,
            "product_tx2": "Dithane",
            "product_id3": 3,
            "product_tx3": "Emulad",
            "product_id4": 11,
            "product_tx4": "Aceite",
            "fumigacionId": "44db83a6-01df-4f13-9595-0aec4ae96a78",
            "status": "ACTIVE",
```

```
            "transactionId": "be3a84e2a079a1dcf1c89ecfae1460d290809cd9a93bb519b4b2bcb221f133df",
            "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#3413973d-c324-43fd-b3ba-5a82913160cf"
        }
    },
    {
        "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
            "idPlan": "a4ace63b-664e-46b7-b5fd-a64450715c0c",
            "date": "2020-01-11T05:00:00.000Z",
            "cycle": 1,
            "Farm": "Clemencia",
            "product_id1": 1,
            "product_tx1": "Cumora",
            "product_id2": 2,
            "product_tx2": "Dithane",
            "product_id3": 3,
            "product_tx3": "Emulad",
            "product_id4": 11,
            "product_tx4": "Aceite",
            "fumigacionId": "44db83a6-01df-4f13-9595-0aec4ae96a78",
            "status": "ACTIVE",
```

```
            "transactionId": "1dd4547797804dd02f88a7e72ac39e03dd6d4e283085884
058b2eb098f1a41ee",
             "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
            "idPlan": "b1a10a7f-fcc6-4b6b-9908-8a15a7a23d09",
            "date": "2020-01-12T22:42:34.200Z",
            "cycle": 0,
            "Farm": "ttt",
            "product_id1": 0,
            "product_tx1": "666",
            "product_id2": 0,
            "product_tx2": "234",
            "product_id3": 0,
            "product_tx3": "555",
            "product_id4": 0,
            "product_tx4": "678",
            "idFito": "h8e0m8a0n3t0",
            "fumigacionId": "f7r8t5y55u6r6y5u5i",
            "auditorId": "newfb78458798855",
```

```
            "status": "ACTIVE",
            "transactionId": "3f97e5e60315dbc0445378c3cbd75a7b8121c87ed4fc198c
e0e7e50c3bcade34",
             "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#f
g700a78-244d-42aa-84a2-36444ff4e880"
          }
       },
       {
          "block_data": {
             "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
             "idPlan": "b9147ec2-8bc4-4e0b-b476-d42ed2d31418",
             "date": "2019-05-31T01:00:00.000Z",
             "cycle": 10,
             "Farm": "Maria Cecilia",
             "product_id1": 9,
             "product_tx1": "Bumper",
             "product_id2": 2,
             "product_tx2": "Dithane",
             "product_id3": 3,
             "product_tx3": "Emulad",
             "product_id4": 11,
             "product_tx4": "Aceite",
             "idFito": "e4dda2e4-face-440b-bb60-c33bf75d340c",
             "fumigacionId": "44db83a6-01df-4f13-9595-0aec4ae96a78",
```

```
            "auditorId": "e4dda2e4-face-440b-bb60-c33bf75d340c",
            "status": "PROCESS AUDITOR",
             "transactionId": "cee60c4d733ef95ee5eb8a7f0936de935da2bc0985bde3a
e70ab73ac5424a249",
            "dataauditor": [
               {
                  "$class": "org.hyperledger_composer.fumigacion.DataAuditor",
                  "idFito": "e4dda2e4-face-440b-bb60-c33bf75d340c",
                  "realDate": "2019-05-30T05:00:00.000Z",
                  "hrInitial": 9,
                  "hrFinal": 12.33,
                  "wind": 7.89,
                  "observation": "listox",
                   "transactionId": "d2345881c88280ef357441a0dc33f3267bd6097c67af
f5c868a523a26f841c14",
                    "auditor": "resource:org.hyperledger_composer.fumigacion.Auditor#c
a700708-01a2-415e-adbc-5b1cbf55a4b8"
               }
            ],
             "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
         }
      },
      {
```

```
"block_data": {
    "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
    "idPlan": "d8c191e5-5575-4bfe-8511-dd2487c60c09",
    "date": "2019-09-23T05:42:34.200Z",
    "cycle": 40,
    "Farm": "ttt",
    "product_id1": 5,
    "product_tx1": "123",
    "product_id2": 0,
    "product_tx2": "234",
    "product_id3": 0,
    "product_tx3": "456",
    "product_id4": 0,
    "product_tx4": "678",
    "idFito": "h8e0m8a0n3t0",
    "fumigacionId": "f7r8t5y55u6r6y5u5i",
    "auditorId": "newfb78458798855",
    "status": "ACTIVE",
     "transactionId": "789e1eacc7a27cb99f01d4ba8eb7845e8c64d258b9cd8c0
03694be40f5368d4a",
    "products": [],
    "dataauditor": [],
     "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#f
g700a78-244d-42aa-84a2-36444ff4e880"
```

```
        }
    },
    {
        "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
            "idPlan": "fdce572a-acc4-4337-8d1b-166198dddb3c",
            "date": "2020-01-11T05:00:00.000Z",
            "cycle": 3,
            "Farm": "Catay",
            "product_id1": 7,
            "product_tx1": "Siganex",
            "product_id2": 2,
            "product_tx2": "Dithane",
            "product_id3": 3,
            "product_tx3": "Emulad",
            "product_id4": 11,
            "product_tx4": "Aceite",
            "fumigacionId": "44db83a6-01df-4f13-9595-0aec4ae96a78",
            "status": "ACTIVE",
            "transactionId": "e75149dcaea601349124d444c58eaa7d60999df27c15e99
08f939dffbfc24ce1",
            "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
        }
```

```
      }
   ]
}
```


getFumigacionPlan (por Id de plan de fumigación): Obtener el registro de un plan de fumigación por el plan ID.

http://164.68.126.181/rest/fumigacioncompany/getFumigacionPlan/fdce572a-acc4-4337-8d1b-166198dddb3c

Body request


```
{
   "code": 200,
   "sucess": true,
   "payload": [
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
            "idPlan": "fdce572a-acc4-4337-8d1b-166198dddb3c",
            "date": "2020-01-11T05:00:00.000Z",
            "cycle": 3,
            "Farm": "Catay",
            "product_id1": 7,
            "product_tx1": "Siganex",
            "product_id2": 2,
            "product_tx2": "Dithane",
            "product_id3": 3,
```

```
            "product_tx3": "Emulad",
            "product_id4": 11,
            "product_tx4": "Aceite",
            "fumigacionId": "44db83a6-01df-4f13-9595-0aec4ae96a78",
            "status": "ACTIVE",
            "transactionId": "e75149dcaea601349124d444c58eaa7d60999df27c15e99
08f939dffbfc24ce1",
            "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
        }
    }
  ]
}
```


Body response


```
{
   "code": 201,
   "sucess": true,
   "message": "Fumigacion Plan saved in blockchain Successfully",
   "planId": "b1a10a7f-fcc6-4b6b-9908-8a15a7a23d09"
}
```


getFumigacionData: Obtiene los datos de la ejecución de la fumigación.

Metodo Get

http://164.68.126.181/rest/fumigacioncompany/getFumigacionData

```
{
   "code": 200,
   "sucess": true,
   "payload": [
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.DataFumigacion",
            "fumigacionId": "d5f7f2ad-1f58-4d21-a780-30cf067c5b64",
            "dateExecution": "2019-09-23T10:04:22.821Z",
            "observation": "additya",
            "transactionId": "7a85dbd4c9d6b9cc318c1d5e999d71a7859dbbfcc2e5dbf8
ad390044a403cef6",
             "fumigacioncompany": "resource:org.hyperledger_composer.fumigacion.F
umigacionCompany#b265d3d3-252e-4f35-8655-5b4d1588ee48"
         }
      }
   ]
}
```


addFumigacionData: Permite añadir datos de la fumigación, usado por la empresa fumigadora.

Metodo Post

http://164.68.126.181/rest/fumigacioncompany/addFumigacionData

Body request


```
{
```

```
  "dateExecution": "2019-12-01T22:39:22.821Z",
  "observation": "ingrsado por cesar v2",
  "fumigacioncompany": "b265d3d3-252e-4f35-8655-5b4d1588ee48"
}
```


Body response


```
{
    "code": 201,
    "sucess": true,
    "message": "Fumigacion Data saved in blockchain Successfully",
    "fumigacionId": "cb66e72a-e163-4894-8b16-29869a1fa073"
}
```


updateFumigacionData (por el fumigacionId): Actualiza la observación sobre la tarea de fumigación, la fecha de dicha observación, haciendo referencia al id interno de la compañía de fumigación.

Metodo Put

http://164.68.126.181/rest/fumigacioncompany/updateFumigacionData/cb66e72a-e163-4894-8b16-29869a1fa073

Request Body


```
{
  "dateExecution": "2020-01-12T23:18:22.821Z",
  "observation": "actualizado por cesar villarroel",
  "fumigacioncompany": "b265d3d3-252e-4f35-8655-5b4d1588ee48"
}
```


Response Body

```
{
    "code": 200,
    "sucess": true,
    "message": "Fumigacion Data updated"
}
```


Comprobación con el getFumigationData (por el id)

Metodo Get

http://164.68.126.181/rest/fumigacioncompany/getFumigacionData/cb66e72a-e163-4894-8b16-29869a1fa073


```
{
    "code": 200,
    "sucess": true,
    "payload": [
        {
            "block_data": {
                "$class": "org.hyperledger_composer.fumigacion.DataFumigacion",
                "fumigacionId": "cb66e72a-e163-4894-8b16-29869a1fa073",
                "dateExecution": "2020-01-12T23:18:22.821Z",
                "observation": "actualizado por cesar villarroel",
                "transactionId": "6779043e870b689dfbbd537bff7e9252285a2135711bca3b70b9852cb0e5ae8b",
                 "fumigacioncompany": "resource:org.hyperledger_composer.fumigacion.FumigacionCompany#b265d3d3-252e-4f35-8655-5b4d1588ee48"
            }
        }
```

```
    ]
}
```


add: Permite añadir una empresa fumigadora.

Método POST

http://164.68.126.181/rest/fumigacioncompany/add

Request Body


```
{
  "email": "victor2005@gmail.com",
  "firstName": "Victor",
  "lastName": "Villarroel"
}
```


Response Body


```
{
    "code": 201,
    "sucess": true,
    "message": "FumigacionCompany saved in blockchain Successfully",
    "companyId": "66c58bf0-74cb-4441-b9b2-c5345381dc1b"
}
```


getFumigacionData (por el id de fumigación): Los datos de la ejecución de la fumigación por el id de fumigación.

Metodo Get

http://164.68.126.181/rest/fumigacioncompany/getFumigacionData/d5f7f2ad-1f58-4d21-a780-30cf067c5b64


```
{
```

```
    "code": 200,
    "sucess": true,
    "payload": [
        {
            "block_data": {
                "$class": "org.hyperledger_composer.fumigacion.DataFumigacion",
                "fumigacionId": "d5f7f2ad-1f58-4d21-a780-30cf067c5b64",
                "dateExecution": "2019-09-23T10:04:22.821Z",
                "observation": "additya",
                "transactionId": "7a85dbd4c9d6b9cc318c1d5e999d71a7859dbbfcc2e5dbf8
ad390044a403cef6",
                 "fumigacioncompany": "resource:org.hyperledger_composer.fumigacion.F
umigacionCompany#b265d3d3-252e-4f35-8655-5b4d1588ee48"
            }
        }
    ]
}
```


updateFumigacionPlan (por el id de plan de fumigacion): Actualizar el plan de fumigación, por el id de un plan de fumigación.

Métodos para el contrato Product

add: Ingreso de los productos de fumigación con su SKU relacionado a la orden de compra.

http://164.68.126.181/rest/product/add

Método Post

Request Body


```
{
  "descripcion": "Dithane",
  "SKU": "3435",
  "quantity": 2,
  "purchase_order":"1",
  "status":"active",
  "provider":"caf95479-5ad9-4322-a7b7-cc322d4def24",
  "enterprise":"3b884c19-0925-4697-9254-e93fecb17b72"
}
```


Response Body


```
{
    "code": 201,
    "sucess": true,
    "message": "Products saved in blockchain Successfully",
    "productId": "30dd7ccc-bfe6-4201-abe4-14f0455cb9d3"
```

}

list: Listado de todos los productos de fumigación con su SKU relacionado a la orden de compra.

Método get

http://164.68.126.181/rest/product/list

Response Body


```
{
   "code": 200,
   "sucess": true,
   "payload": [
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Products",
            "productId": "112a3e59-4003-4e19-be91-1761d47c3c2b",
            "descripcion": "Triazol SAP",
            "SKU": "TRIAZSP2912",
            "purchase_order": "0005872",
            "status": "ACTIVO",
            "quantity": 5,
             "transactionId": "e5f051315b987bd3555f619394e1765e596cab4588eb8bd
d9c7d067b90435bcb",
              "provider": "resource:org.hyperledger_composer.fumigacion.Provider#a97
8efb7-5f86-425b-a985-1ff5b9ca8deb",
```

```
            "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Products",
            "productId": "1effec4e-a5f1-4a94-99f8-7030d8370013",
            "descripcion": "Ditane Forte",
            "SKU": "DIYAFO4522",
            "purchase_order": "0005872",
            "status": "ACTIVO",
            "quantity": 5,
            "transactionId": "242ca31fedceb1bb482dc0306289994b96b592b5dfdb17ef
e52c6c5d33357bcb",
            "provider": "resource:org.hyperledger_composer.fumigacion.Provider#a97
8efb7-5f86-425b-a985-1ff5b9ca8deb",
            "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Products",
```

```
            "productId": "30dd7ccc-bfe6-4201-abe4-14f0455cb9d3",
            "descripcion": "Dithane",
            "SKU": "3435",
            "purchase_order": "1",
            "status": "active",
            "quantity": 2,
             "transactionId": "daa04a4fa3d63d28eea30a86819eb9d04fd33279707c7298ae87db95d8ca25a7",
             "provider": "resource:org.hyperledger_composer.fumigacion.Provider#caf95479-5ad9-4322-a7b7-cc322d4def24",
              "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#3b884c19-0925-4697-9254-e93fecb17b72"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Products",
            "productId": "4c9c45a4-f5f2-449e-980a-e5d2cb7304b9",
            "descripcion": "Triazol CAB",
            "SKU": "TRIAZCB8712",
            "purchase_order": "0005872",
            "status": "ACTIVO",
            "quantity": 3,
```

```
            "transactionId": "a3071f6cd4cb90327cb9dad2a7b01361f8bde4cd8ccaa29b
75d8799853124baa",
             "provider": "resource:org.hyperledger_composer.fumigacion.Provider#a97
8efb7-5f86-425b-a985-1ff5b9ca8deb",
             "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Products",
            "productId": "8012c4ef-a80b-46d6-a6d4-80fd158ea84a",
            "descripcion": "SIGATOKA TEMPOX",
            "SKU": "SIKA56GBHT78",
            "purchase_order": "2867",
            "status": "ACTIVO",
            "quantity": 24,
             "transactionId": "32c928638e82d04d27c4461c65bfc07322afda2bb5785fa0
db2009e3c6e411b3",
             "provider": "resource:org.hyperledger_composer.fumigacion.Provider#a97
8efb7-5f86-425b-a985-1ff5b9ca8deb",
             "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3413973d-c324-43fd-b3ba-5a82913160cf"
         }
```

```
      }
   ]
}
```


list (por el idProduct): Lista de las características de un producto consultado por su código de producto.

Método get

http://164.68.126.181/rest/product/list/30dd7ccc-bfe6-4201-abe4-14f0455cb9d3

Response Body


```
{
   "code": 200,
   "sucess": true,
   "payload": [
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Products",
            "productId": "30dd7ccc-bfe6-4201-abe4-14f0455cb9d3",
            "descripcion": "Dithane",
            "SKU": "3435",
            "purchase_order": "1",
            "status": "active",
            "quantity": 2,
             "transactionId": "daa04a4fa3d63d28eea30a86819eb9d04fd33279707c729
8ae87db95d8ca25a7",
```

"provider": "resource:org.hyperledger_composer.fumigacion.Provider#caf95479-5ad9-4322-a7b7-cc322d4def24",
"enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#3b884c19-0925-4697-9254-e93fecb17b72"
}
}
]
}

addProductionManager: Permite ingresar los datos del Administrador de Producción (de la empresa bananera).

Método post

http://164.68.126.181/rest/product/addProductionManager

{
"email": "cvillarr123@gmail.com",
"firstName": "Cesar",
"lastName": "Villarroel"
}

Response body

{
"code": 201,
"sucess": true,
"message": "ProductionManager saved in blockchain Successfully",
"productionId": "49e27de4-9000-4863-a30e-4b85316f2179"
}

updateProducts (por el idProduct): Permite los datos de las condiciones para aplicar la fumigación, ingresado por el Auditor de Fumigación.

Método put

http://164.68.126.181/rest/product/updateProducts/30dd7ccc-bfe6-4201-abe4-14f0455cb9d3

Request body


```
{
  "descripcion": "Dithane_v2",
  "SKU": "3435",
  "quantity": 4,
  "purchase_order":"1",
  "status":"active",
  "provider": "caf95479-5ad9-4322-a7b7-cc322d4def24",
  "enterprise":"3b884c19-0925-4697-9254-e93fecb17b72"
}
```


Response body


```
{
    "code": 201,
    "sucess": true,
    "message": "Product updated"
}
```


Validamos con el método list para obtener el producto que actualizamos

http://164.68.126.181/rest/product/list/30dd7ccc-bfe6-4201-abe4-14f0455cb9d3

Response Body


```
{
    "code": 200,
```

```
   "sucess": true,
   "payload": [
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Products",
            "productId": "30dd7ccc-bfe6-4201-abe4-14f0455cb9d3",
            "descripcion": "Dithane_v2",
            "SKU": "3435",
            "purchase_order": "1",
            "status": "active",
            "quantity": 4,
             "transactionId": "daa04a4fa3d63d28eea30a86819eb9d04fd33279707c729
8ae87db95d8ca25a7",
             "provider": "resource:org.hyperledger_composer.fumigacion.Provider#caf9
5479-5ad9-4322-a7b7-cc322d4def24",
              "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
3b884c19-0925-4697-9254-e93fecb17b72"
         }
      }
   ]
}
```


addProvider: Permite ingresar el proveedor de los Productos de Fumigación.

Método post

http://164.68.126.181/rest/product/addProvider

Request body


```
{
  "email": "cvillarr123@hotmail.com",
  "firstName": "Alberto",
  "lastName": "Samaniego"
}
```


Response body


```
{
    "code": 201,
    "sucess": true,
    "message": "Provider saved in blockchain Successfully",
    "providerId": "8e622120-7794-451c-a8fb-19af15ac9ff6"
}
```


Métodos para el contrato auditor

saveAuditorData: Permite los datos de las condiciones para aplicar la fumigación, ingresado por el Auditor de Fumigación.

Método POST

http://164.68.126.181/rest/auditor/saveAuditorData

Request body


```
{
  "realDate": "2020-01-18T11:29:55.063Z",
  "hrInitial": 2,
  "hrFinal": 2,
  "wind": 1,
```

```
  "observation": "Prueba datos Auditor",
  "auditor": "f652b95a-9db5-4100-b15a-a6d141515402"
}
```


Response Body


```
{
    "code": 201,
    "sucess": true,
    "message": "Auditor Data saved in blockchain Successfully",
    "idFito": "aa206fb5-e156-4142-8e16-f38c55b26d62"
}
```


getAuditorData: Permite consultar todos los datos ingresados por los Auditores, sobre las condiciones para aplicar la fumigación.

Método GET

http://164.68.126.181/rest/auditor/getAuditorData

Response Body


```
{
    "code": 200,
    "sucess": true,
    "payload": [
        {
            "block_data": {
                "$class": "org.hyperledger_composer.fumigacion.DataAuditor",
                "idFito": "21f35f10-ad62-41ad-822e-8c3b05c4928d",
                "realDate": "2019-05-30T05:00:00.000Z",
```

```
            "hrInitial": 5,
            "hrFinal": 7,
            "wind": 6,
            "observation": "listo",
             "transactionId": "d7b584c1b0822c1ccc3ceb25757e1e552c2419658d38cf8
e379294c2b1ad7dd5",
              "auditor": "resource:org.hyperledger_composer.fumigacion.Auditor#ca700
708-01a2-415e-adbc-5b1cbf55a4b8"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.DataAuditor",
            "idFito": "54cee8c0-d2c1-4347-b37c-185af3bfa5a2",
            "realDate": "2019-05-30T05:00:00.000Z",
            "hrInitial": 5,
            "hrFinal": 7,
            "wind": 3,
            "observation": "listo",
             "transactionId": "ececce7af41ecf0740dc734ac6a2edcf823f0df938d64e604
aeeba761cb0e79b",
              "auditor": "resource:org.hyperledger_composer.fumigacion.Auditor#ca700
708-01a2-415e-adbc-5b1cbf55a4b8"
         }
```

```
    },
    {
      "block_data": {
        "$class": "org.hyperledger_composer.fumigacion.DataAuditor",
        "idFito": "aa206fb5-e156-4142-8e16-f38c55b26d62",
        "realDate": "2020-01-18T11:29:55.063Z",
        "hrInitial": 2,
        "hrFinal": 2,
        "wind": 1,
        "observation": "Prueba datos Auditor",
         "transactionId": "29d6a153d9c2ae7bff34912fd45147b69ebbfe9cc945cee9
a4c0f8469e486ded",
         "auditor": "resource:org.hyperledger_composer.fumigacion.Auditor#f652b9
5a-9db5-4100-b15a-a6d141515402"
      }
    }
  ]
}
```


getAuditorData (por idFito): Permite consultar por un id del control Fito sanitario, los datos ingresados sobre las condiciones al aplicar la fumigación y la fecha de ejecución.

Método GET

http://164.68.126.181/rest/auditor/getAuditorData/aa206fb5-e156-4142-8e16-f38c55b26d62

Response Body


```
{
    "code": 200,
    "sucess": true,
    "payload": [
        {
            "block_data": {
                "$class": "org.hyperledger_composer.fumigacion.DataAuditor",
                "idFito": "aa206fb5-e156-4142-8e16-f38c55b26d62",
                "realDate": "2020-01-18T11:29:55.063Z",
                "hrInitial": 2,
                "hrFinal": 2,
                "wind": 1,
                "observation": "Prueba datos Auditor",
                 "transactionId": "29d6a153d9c2ae7bff34912fd45147b69ebbfe9cc945cee9
a4c0f8469e486ded",
                 "auditor": "resource:org.hyperledger_composer.fumigacion.Auditor#f652b9
5a-9db5-4100-b15a-a6d141515402"
            }
        }
    ]
}
```

updateAuditorData (por idFito): Permite actualizar la información de las condiciones de fumigación y la fecha ingresada por un Auditor de Fumigación, por el idFito (id de registro fitosanitario).

Método PUT

http://164.68.126.181/rest/auditor/updateAuditorData/aa206fb5-e156-4142-8e16-f38c55b26d62

Request Body


```
{
  "realDate": "2020-01-19T11:09:55.063Z",
  "hrInitial": 1,
  "hrFinal": 1,
  "wind": 1,
  "observation": "prueba cvi",
  "transactionId":
"29d6a153d9c2ae7bff34912fd45147b69ebbfe9cc945cee9a4c0f8469e486ded",
  "auditorId": "b3162171-0630-4d2f-9009-97a4f427c052"
}
```


Response Body


```
{
    "code": 201,
    "sucess": true,
    "message": "Auditor Data updated"
}
```


Verificamos con el método **getAuditorData**

http://164.68.126.181/rest/auditor/getAuditorData/aa206fb5-e156-4142-8e16-f38c55b26d62


```
{
```

```
   "code": 200,
   "sucess": true,
   "payload": [
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.DataAuditor",
            "idFito": "aa206fb5-e156-4142-8e16-f38c55b26d62",
            "realDate": "2020-01-19T11:09:55.063Z",
            "hrInitial": 3,
            "hrFinal": 1,
            "wind": 1,
            "observation": "prueba cvi",
             "transactionId": "29d6a153d9c2ae7bff34912fd45147b69ebbfe9cc945cee9
a4c0f8469e486ded",
             "auditor": "resource:org.hyperledger_composer.fumigacion.Auditor#b3162
171-0630-4d2f-9009-97a4f427c052"
         }
      }
   ]
}
```


add: Permite ingresar los datos de un auditor para el mantenimiento de este participante.

Método POST

http://164.68.126.181/rest/auditor/add

Request Body


```
{
  "email": "auditor_externo@gruposico.com",
  "firstName": "Auditor",
  "lastName": "Externo Diximant"
}
```


Request Body


```
{
    "code": 201,
    "sucess": true,
    "message": "Auditor saved in blockchain Successfully",
    "auditorId": "bbf5fde9-dd6f-4a92-be1d-b29e5187cba6"
}
```


list: Permite consultar todos los datos de todos los Auditores de Fumigación.

Método GET

http://164.68.126.181/rest/auditor/list

Response Body


```
{
    "code": 200,
    "sucess": true,
    "payload": [
        {
            "block_data": {
                "$class": "org.hyperledger_composer.fumigacion.Auditor",
```

```
            "auditorId": "a836e2ca-7b42-4821-accc-5967a4a5f248",
            "email": "auditor_externo@gruposico.com",
            "firstName": "Auditor Externo",
            "lastName": "Auditor Externo",
            "transactionId": "27b0056c04edcb8d4f65503f54f7c585a688d625a9d98819
8068a8bbc1a2b76f"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Auditor",
            "auditorId": "b3162171-0630-4d2f-9009-97a4f427c052",
            "email": "cvillarr@hotmail.com",
            "firstName": "Cesar",
            "lastName": "Villarroel",
            "transactionId": "e533eef19a15cfb91eaf70347f571666344a5a42d67ecaf86
2c1347fde97f434"
         }
      },
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Auditor",
            "auditorId": "ca700708-01a2-415e-adbc-5b1cbf55a4b8",
            "email": "auditror_interno@gruposico.com",
```

"firstName": "Auditor Interno",

"lastName": "Auditor Interno",

"transactionId": "c1516c69794e2115bbb595f6931f01b798e711c3d0712c623f5ba026e3d3d76f"

}

}

]

}

(por auditorId): Permite actualizar los datos de un auditor especificado en el url.

Método PUT

http://164.68.126.181/rest/auditor/b3162171-0630-4d2f-9009-97a4f427c052

Request Body

{

"email": "cvillarr123@hotmail.com",

"firstName": "Alberto",

"lastName":"Samaniego"

}

Response Body

{

"code": 201,

"sucess": true,

"message": "Auditor updated"

}

Consultamos con el listado para ver los cambios usando el método getFilterData consultando el participante Auditor.

http://164.68.126.181/rest/filter/getFilterData?participant=Auditor&auditorId=b3162171-0630-4d2f-9009-97a4f427c052


```
{
   "code": 200,
   "sucess": true,
   "payload": [
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Auditor",
            "auditorId": "b3162171-0630-4d2f-9009-97a4f427c052",
            "email": "cvillarr123@hotmail.com",
            "firstName": "Alberto",
            "lastName": "Samaniego",
            "transactionId": "e533eef19a15cfb91eaf70347f571666344a5a42d67ecaf862c1347fde97f434"
         }
      }
   ]
}
```


Métodos para el contrato TransactionHistory

transaction/logs: Permite obtener el historial de todas las transacciones realizadas.

Método GET

http://164.68.126.181/rest/transactionhistory/logs

Request Request


```
[
  {
    "$class": "org.hyperledger.composer.system.HistorianRecord",
    "transactionId": "3dd7f1ed8ed55151aad58ca13989ace23e1f050ffad77ae29f5dc0b78833e54a",
    "transactionType": "org.hyperledger.composer.system.ActivateCurrentIdentity",
    "transactionInvoked": "resource:org.hyperledger.composer.system.ActivateCurrentIdentity#3dd7f1ed8ed55151aad58ca13989ace23e1f050ffad77ae29f5dc0b78833e54a",
    "identityUsed": "resource:org.hyperledger.composer.system.Identity#cd8a9c35bf9ed1eb473710f204bc45ed86203aa0f87bba4c56420e04d7dd74a2",
    "eventsEmitted": [],
    "transactionTimestamp": "2020-01-20T12:12:42.404Z"
  },
  {
    "$class": "org.hyperledger.composer.system.HistorianRecord",
    "transactionId": "4485cd818141e76ee18afcc6942c695d01d4a355435637ce96fce7cbac79a2f8",
    "transactionType": "org.hyperledger.composer.system.UpdateParticipant",
```

"transactionInvoked": "resource:org.hyperledger.composer.system.UpdateParticipant#4485cd818141e76ee18afcc6942c695d01d4a355435637ce96fce7cbac79a2f8",
"participantInvoking": "resource:org.hyperledger.composer.system.NetworkAdmin#fumigacionplan",
"identityUsed": "resource:org.hyperledger.composer.system.Identity#cd8a9c35bf9ed1eb473710f204bc45ed86203aa0f87bba4c56420e04d7dd74a2",
"eventsEmitted": [],
"transactionTimestamp": "2020-01-25T14:07:30.634Z"
},
{
"$class": "org.hyperledger.composer.system.HistorianRecord",
"transactionId": "4b139495cf94b9fcf522610c3e3f35fe71c19c74c8b234440e579e9a0c8b95de",
"transactionType": "org.hyperledger.composer.system.AddParticipant",
"transactionInvoked": "resource:org.hyperledger.composer.system.AddParticipant#4b139495cf94b9fcf522610c3e3f35fe71c19c74c8b234440e579e9a0c8b95de",
"participantInvoking": "resource:org.hyperledger.composer.system.NetworkAdmin#fumigacionplan",
"identityUsed": "resource:org.hyperledger.composer.system.Identity#cd8a9c35bf9ed1eb473710f204bc45ed86203aa0f87bba4c56420e04d7dd74a2",
"eventsEmitted": [],
"transactionTimestamp": "2020-01-25T14:02:00.583Z"
},

```
  {
    "$class": "org.hyperledger.composer.system.HistorianRecord",
     "transactionId": "5a12452de5c1db01e5ef0f185b6410cacc90c22ed1a2defeb1f6
ad0276a73dc5",
    "transactionType": "org.hyperledger.composer.system.AddAsset",
     "transactionInvoked": "resource:org.hyperledger.composer.system.AddAsset#5
a12452de5c1db01e5ef0f185b6410cacc90c22ed1a2defeb1f6ad0276a73dc5",
     "participantInvoking": "resource:org.hyperledger.composer.system.NetworkAdm
in#fumigacionplan",
    "identityUsed": "resource:org.hyperledger.composer.system.Identity#cd8a9c35b
f9ed1eb473710f204bc45ed86203aa0f87bba4c56420e04d7dd74a2",
    "eventsEmitted": [],
    "transactionTimestamp": "2020-01-25T14:00:51.571Z"
  }
]
```


Métodos para el contrato TransactionHistory

getFilterData: Sacar las consultas de recursos (assets) y participantes (participants), por los valores de cualquier atributo que pertenezca a ese recurso o participante.

Método GET

Ejemplo cuando se define por participante

http://164.68.126.181/rest/filter/getFilterData?participant=Auditor&auditorId=bbf5fde9-dd6f-4a92-be1d-b29e5187cba6

Response Body


```
{
  "code": 200,
```

```
   "sucess": true,
   "payload": [
      {
         "block_data": {
            "$class": "org.hyperledger_composer.fumigacion.Auditor",
            "auditorId": "bbf5fde9-dd6f-4a92-be1d-b29e5187cba6",
            "email": "auditor_externo@gruposico.com",
            "firstName": "Auditor",
            "lastName": "Externo Diximant",
             "transactionId": "2de0123e2ee13a347ac17636bc2509b45c4dce899de180
4d7959323cccc7d5fe"
         }
      }
   ]
}
```


Consultar los datos de un recurso a través de algún(nos) atributos de ese recurso.

Método GET

Ejemplo:

Cuando se define por asset, en este caso para consultar un plan que tiene el atributo, status=PA_AUDITADO_PO_FUMIGAR, fumigacionId=4ce80916-7a4f-4536-b075-69c1f1afb0b9 se ingresaría un url como este:

http://164.68.126.181/rest/filter/getFilterData?assetName=PlanFumigacion&status=PA_AUDITADO_PO_FUMIGAR&fumigacionId=4ce80916-7a4f-4536-b075-69c1f1afb0b9

Response Body

```
{
    "code": 200,
    "sucess": true,
    "payload": [
        {
            "block_data": {
                "$class": "org.hyperledger_composer.fumigacion.PlanFumigacion",
                "idPlan": "6fc05a74-670d-47a5-8b6b-a0f4ff2115e5",
                "date": "2019-08-13T05:00:00.000Z",
                "cycle": 14,
                "Farm": "Maria Cecilia",
                "product_id1": 1.3,
                "product_tx1": "Odeon",
                "product_id2": 0,
                "product_tx2": "Ninguno",
                "product_id3": 0,
                "product_tx3": "Ninguno",
                "product_id4": 0,
                "product_tx4": "Ninguno",
                "tx_lotes": "Todos",
                "id_combo": "Combo 14",
                "idFito": "194c4f89-5d32-4549-8c72-57d396e39f99",
                "fumigacionId": "4ce80916-7a4f-4536-b075-69c1f1afb0b9",
                "auditorId": "3ad6424a-d357-4144-a6b1-e9954025894e",
```

```
"status": "PA_AUDITADO_PO_FUMIGAR",
 "transactionId": "25b93bcfb3a46317761dc86848d1e1f6cd25e038eb581c8a5da893e444882230",
"products": [
  {
    "$class": "org.hyperledger_composer.fumigacion.Products",
    "productId": "8dddf53a-81fb-493f-b8ff-c6b2b96aff85",
    "descripcion": "Odeon",
    "SKU": "Odeon-00004001-Dixi",
    "purchase_order": "00004001",
    "status": "ACTIVO",
    "quantity": 1,
    "transactionId": "117dd27fb858b1365ea155574452a0f4bab76bf2edca4b0cd809a83bccf07e2c",
     "provider": "resource:org.hyperledger_composer.fumigacion.Provider#a258ad70-5bf4-4652-b4e6-575ce0b57302",
     "enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#c1db93fd-47c2-46fd-81a7-cac1da3f5e28"
  }
],
"dataauditor": [
  {
    "$class": "org.hyperledger_composer.fumigacion.DataAuditor",
    "idFito": "194c4f89-5d32-4549-8c72-57d396e39f99",
```

"realDate": "2019-08-24T05:00:00.000Z",

"hrInitial": 9,

"hrFinal": 11,

"wind": 0,

"observation": "testx",

"transactionId": "206936f6b826a42041ed4d46185a2bcc413c309a564
79f1d83288b73af63f1f2",

"auditor": "resource:org.hyperledger_composer.fumigacion.Auditor#3
ad6424a-d357-4144-a6b1-e9954025894e"

}

],

"enterprise": "resource:org.hyperledger_composer.fumigacion.Enterprise#
c1db93fd-47c2-46fd-81a7-cac1da3f5e28"

}

}

]

}

Métodos para el participante Empresa (empresa productora de banano).

add: Permite ingresar los datos de un auditor para el mantenimiento de este participante.

Método POST

http://164.68.126.181/rest/enterprise/add

Request Body

{
"email": "ventas@gruposico.com",
"Name": "Diximant S.A."
}

Response Body

{
"code": 201,
"sucess": true,
"message": " EnterPrise Data saved in blockchain Successfully",
"enterpriseId": "bbf5fde9-dd6f-4a92-be1d-b29e5187cba6"
}

list: Lista los datos de las empresas bananeras registradas en el blockchain.

Método GET

http://164.68.126.181/rest/enterprise/list

Response Body

{
"code": 200,
"sucess": true,
"payload": [
{
"block_data": {
"$class": "org.hyperledger_composer.fumigacion.Enterprise",
"enterpriseId": "c1db93fd-47c2-46fd-81a7-cac1da3f5e28",
"email": "ventas@gruposico.com",

"Name": "Diximant S.A.",

"transactionId": "5a12452de5c1db01e5ef0f185b6410cacc90c22ed1a2defe
b1f6ad0276a73dc5"

}

}

]

}

(actualizar Empresa por el EnterpriseId): Actualiza datos de la empresa bananera.

Método PUT

http://164.68.126.181/rest/enterprise/c1db93fd-47c2-46fd-81a7-cac1da3f5e28

Request Body

{

"email": "ventas@gruposico.com",

"Name": "Diximant S.A."

}

Response Body

{

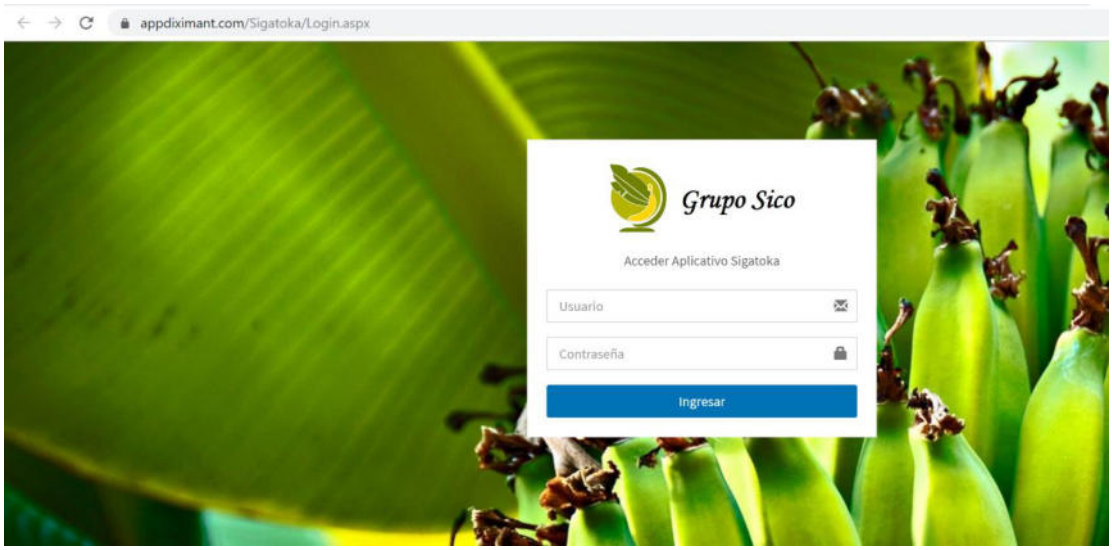
"code": 201,

"sucess": true,

"message": "Enterprise Data updated"

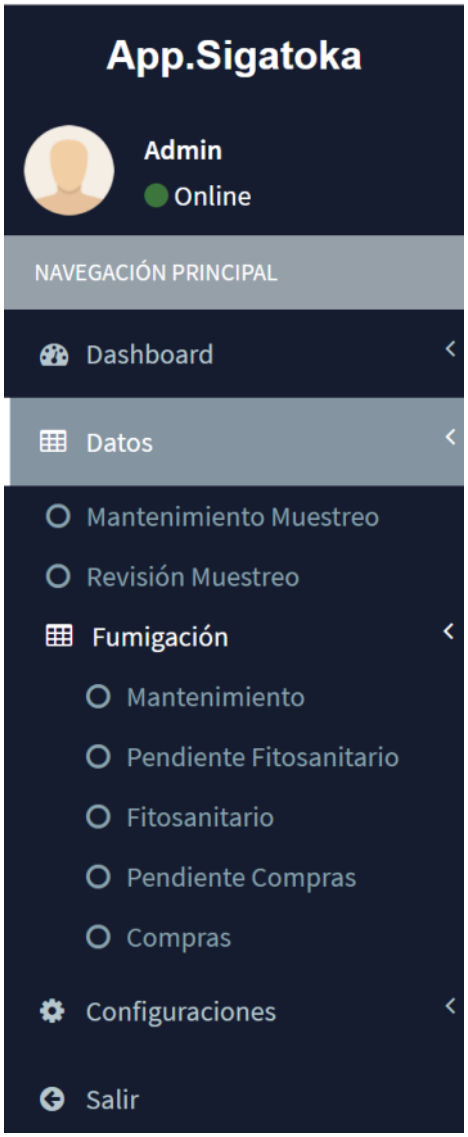
}

ANEXO 10 – APLICACIÓN WEB (PANTALLA INICIAL)



Esta pantalla provee el acceso al personal de la Plantación bananera para los roles del Administrador Financiero encargado de las compras y el Gerente Propietario que delega o genera la planificación. El ingreso es por el usuario de correo.

ANEXO 11 – APLICACIÓN WEB (MENU ACCESOS)



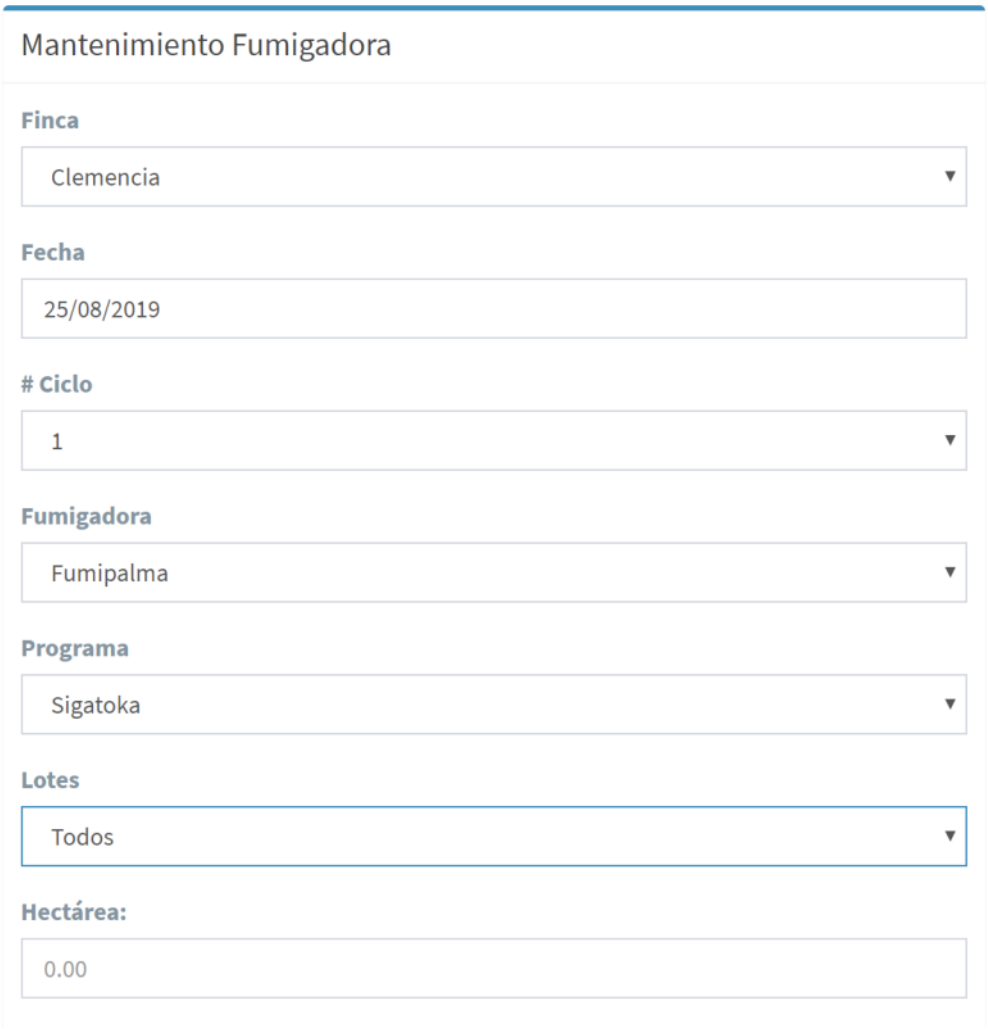
Este es el menú de las opciones descritas a continuación:

Mantenimiento. - La generación del Plan de Fumigación (Mantenimiento) realizado actualmente por el Gerente Propietario o a quien delegue. Mismo que se asigna a una determinada empresa Fumigadora.

Pendiente Fitosanitario. - Complementa las fechas de inicio y fin de la fumigación, así como las condiciones de viento suscitadas durante la aplicación, y observaciones o comentarios adicionales.

Fitosanitario. - Ingresa la información sobre la ejecución efectiva de las tareas de Fumigación.

ANEXO 12 – INTERFACE WEB CREACION PLAN FUMIGACIÓN (INTERFASE PARTE 1)



A continuación, la descripción de los controles para la aplicación web de la empresa productora. En esta parte se genera el plan de mantenimiento que será enviado a la cadena de Blockchain, para agregar las subsiguientes transacciones que actualizarán el plan con la aplicación de los productos comprados y su fecha de ejecución real del

plan, en esta etapa no necesariamente se usarán los nombres comerciales de los productos:

Finca. - Nombre de la finca donde se planifica la aplicación de los productos de fumigación.

Fecha. - Fecha de ingreso del registro (referencial).

Ciclo: Ciclo de fumigación (lista de valores pre-definidos para la aplicación web).

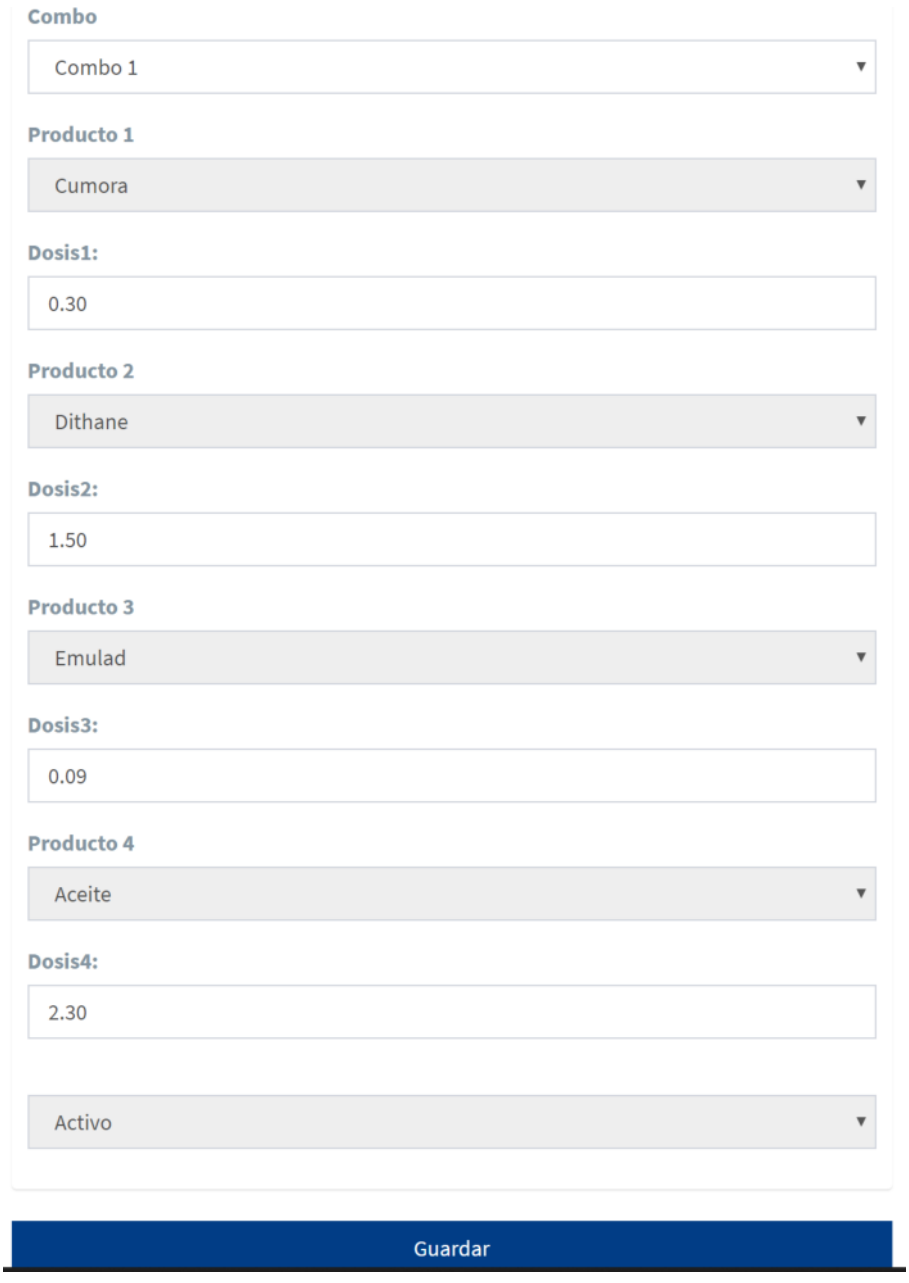
Fumigadora. - Nombre la empresa fumigadora.

Programa. - Nombre relacionado al plan, puede ser relativo a la plaga que se pretende controlar.

Lotes. - Selección del lote o todos los lotes donde ese van a aplicar el producto.

Hectárea. - Definición del área donde se va a aplicar el mantenimiento (fumigación de productos).

ANEXO 13 – INTERFACE WEB CREACION PLAN FUMIGACIÓN (PARTE 2)



A continuación, una descripción de la parte final de la pantalla de planificación:

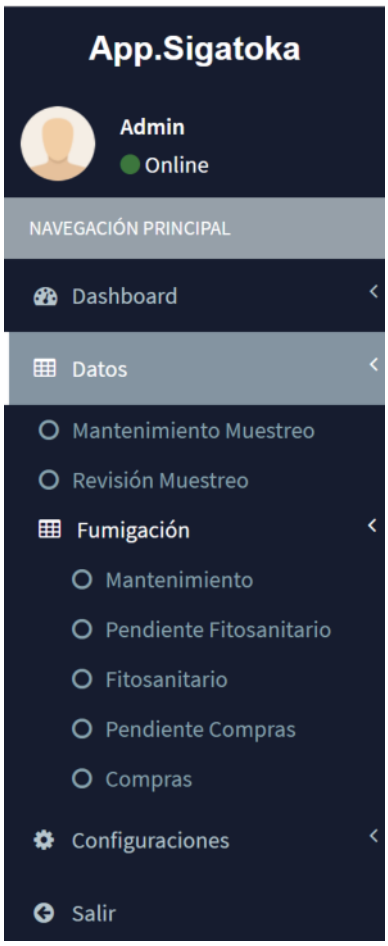
Combo. - Nombre de la combinación de productos químicos a aplicar a la plantación.

Producto #. - Nombre del producto genérico que forma parte del combo.

Dosis#. - Cantidad del producto a aplicar en litros.

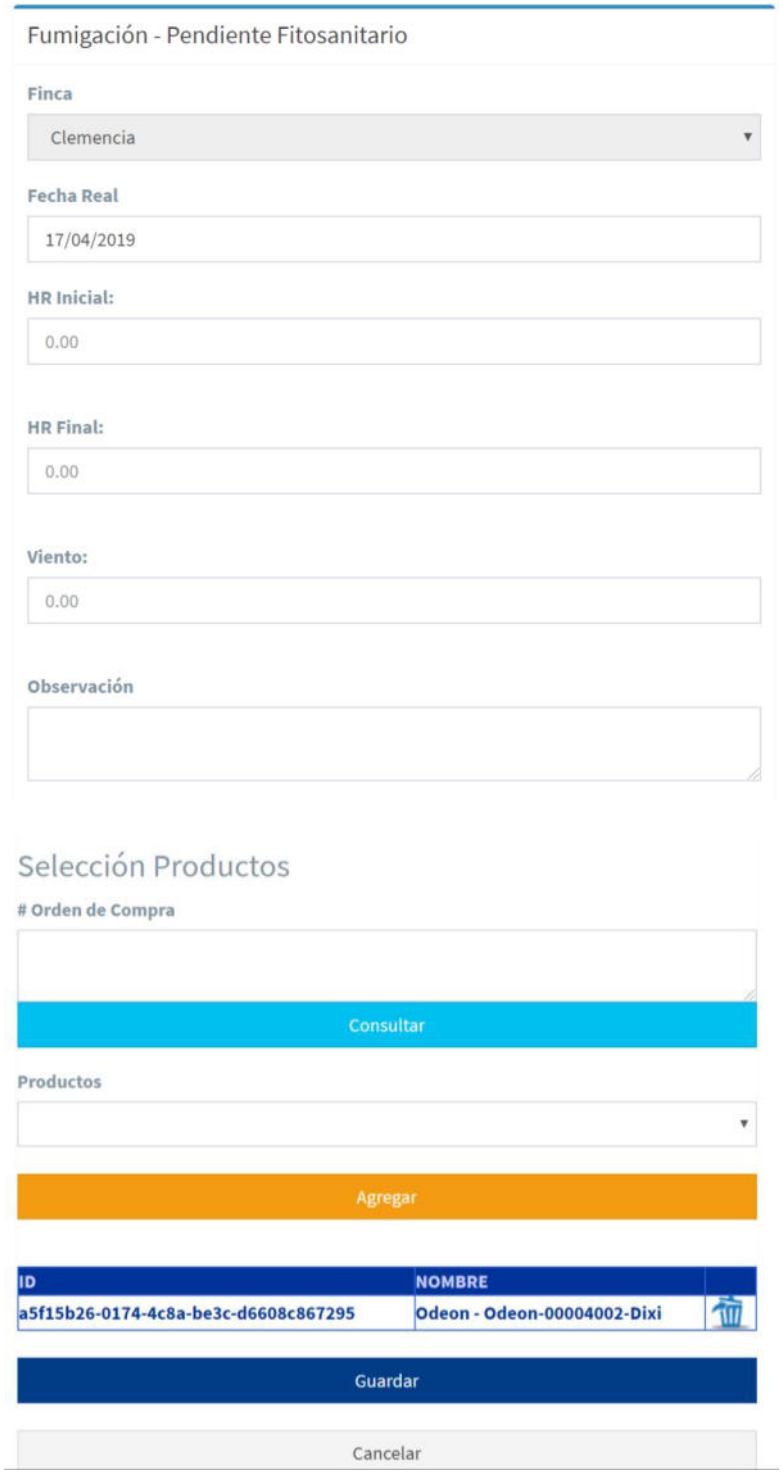
Activo. - Estatus del combo que es manejado por el sistema.

ANEXO 14 – AUDITOR DE PRODUCTOS DE FUMIGACIÓN



El auditor al ingresar a la opción **Pendiente Fitosanitario** se le desplegará la lista de los pendientes.

Al dar click en **Seleccionar** se le abrirá la siguiente pantalla:



A continuación, una breve explicación de los campos del formulario web:

Finca. - Nombre corto de la finca donde se aplicará el mantenimiento o fumigación.

Fecha real. - Fecha real de cuando se aplicó el control fitosanitario (fumigación).

HR inicial. - Humedad relativa inicial en porcentaje.

HR final. - Humedad relativa inicial en porcentaje.

Viento. - Velocidad del viento durante el día de la aplicación.

Observación. - Campo del formulario usado para agregar comentarios adicionales u observaciones a novedades sobre la tarea de control fitosanitario.

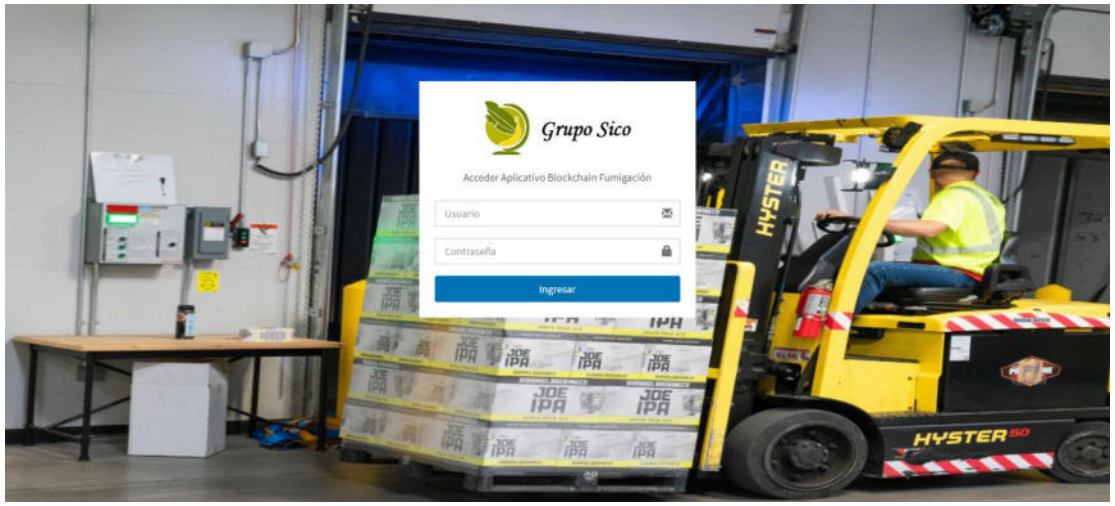
Al dar click en 

le permitirá asociar los productos comerciales sobre el requerimiento genérico ingresado en el plan de fumigación.

Finalmente al dar click en 

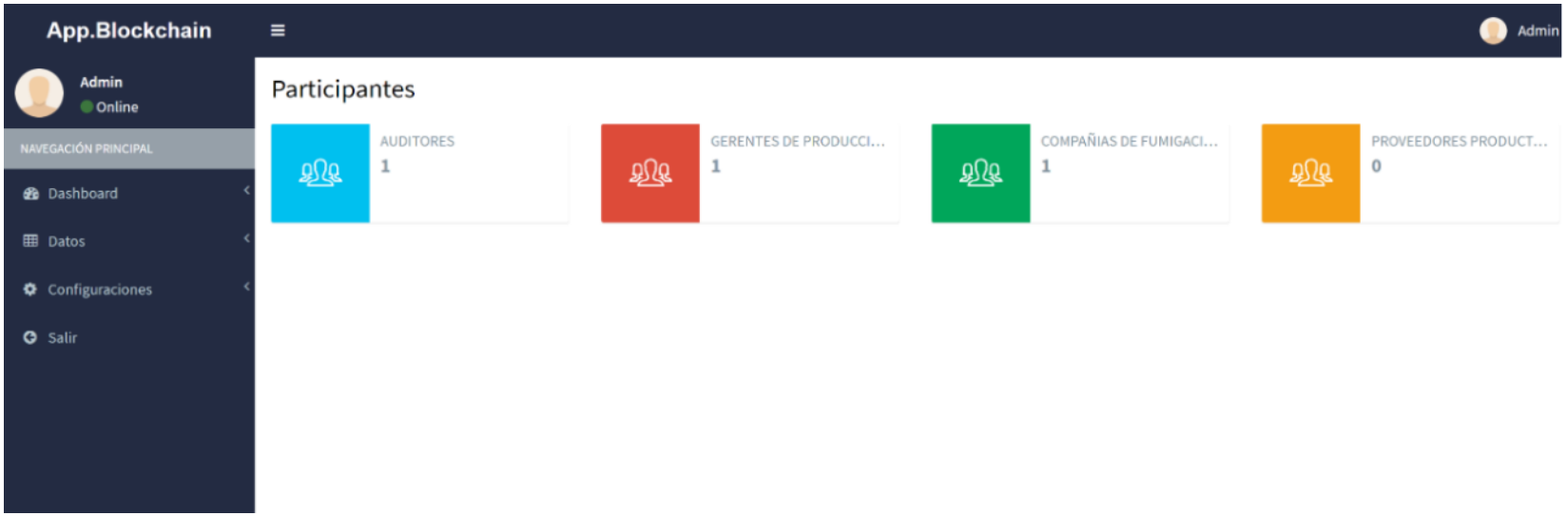
guardará el registro.

ANEXO 15 – ADMINISTRADOR BLOCKCHAIN



El usuario ingresará con su cuenta de correo para el login.

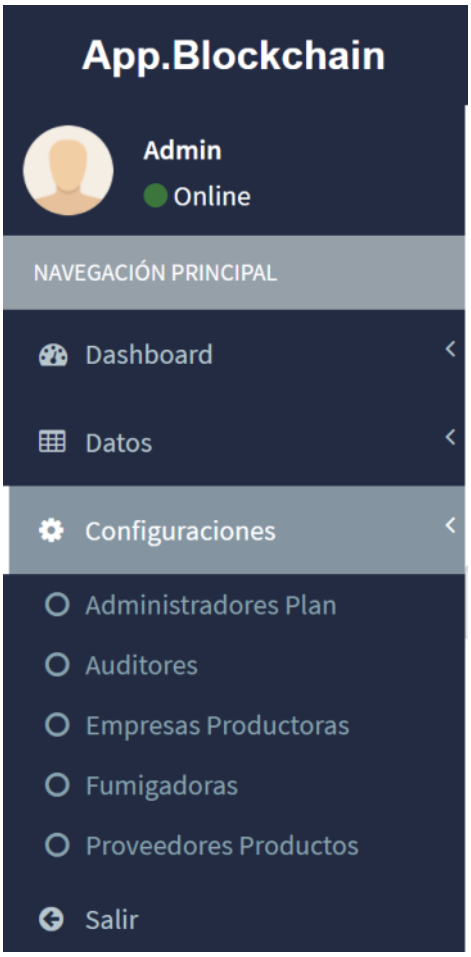
ANEXO 16 – ADMINISTRADOR BLOCKCHAIN



El dashboard indica los participantes (auditores, gerentes de producción, compañías de fumigación) y activos (planes de fumigación, productos, Datos de fumigación, empresa, registros históricos de transacciones, planFumigación, productos).

ANEXO 17 – MENU MANTENIMIENTO

PARTICIPANTES



ANEXO 18 – APLICACION WEB (MANTENIMIENTO ADMINISTRADOR PLAN)

La pantalla de mantenimiento de administrador de Plan presenta los siguientes campos:

Clase. - Código interno generado por el Blockchain del Administrador del Plan, se usa para referenciar el objeto generado en la clase.

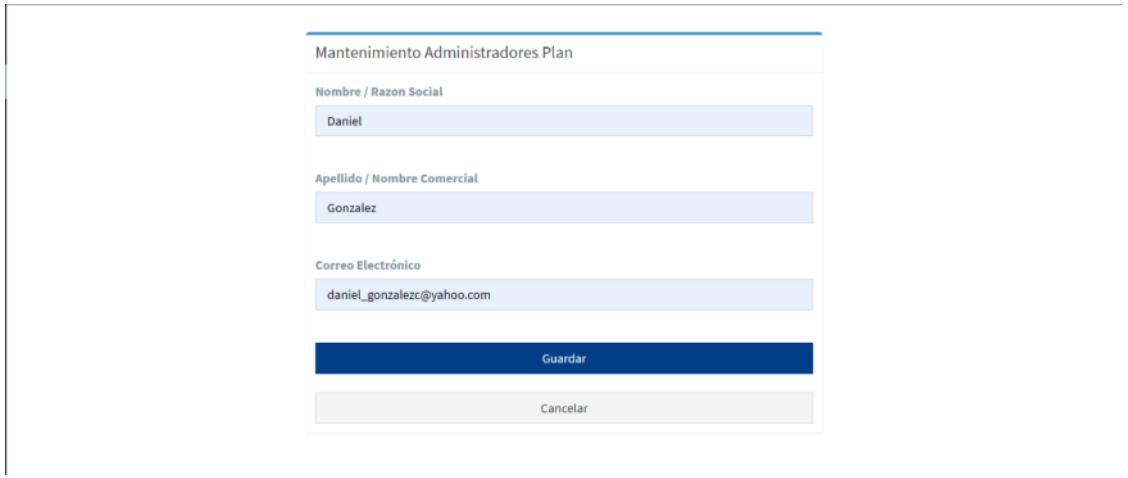
AdminPlanID. - Tiene el valor de los últimos caracteres de la Clase código generado por Blockchain para el Administrador del Plan, usado para visualizar.

Nombre / Razón Social. - El nombre o Razón Social del Administrador del Plan (que en el caso es el Gerente de Producción).

Apellido / Nombre Comercial. - Apellido o Nombre Comercial del Administrador del Plan (que en este caso es el Gerente de Producción).

Correo electrónico.- Correo electrónico del representante del Administrador del Plan.

ANEXO 19 – APLICACION WEB (ADMINISTRADORES PLAN)



Es la pantalla para modificar los valores ingresados del Administrador del Plan:

Nombre / Razón Social. - El nombre o Razón Social del Administrador del Plan (que en el caso es el Gerente de Producción).

Apellido / Nombre Comercial. - Apellido o Nombre Comercial del Administrador del Plan (que en este caso es el Gerente de Producción).

Correo electrónico.- Correo electrónico del representante del Administrador del Plan.

ANEXO 20 – LISTA DE MANTENIMIENTO DE AUDITORES

La pantalla de mantenimiento del Auditor (o representante de la empresa Auditora de productos de fumigación):

Clase. - Código interno generado por el Blockchain del Auditor o Empresa Auditora, se usa para referenciar el objeto generado en la clase.

AdminPlanID. - Tiene el valor de los últimos caracteres generados en el Blockchain para la entidad Auditor, usado para visualizar los datos.

Nombre / Razón Social. - El nombre o Razón Social del Auditor.

Apellido / Nombre Comercial. - Apellido o Nombre Comercial del Auditor o empresa Auditora.

Correo electrónico.- Correo electrónico del representante del Administrador del Plan.

ANEXO 21 – LISTA PARA EDICION MANTENIMIENTO AUDITORES

Es la pantalla para modificar los valores ingresados del Auditor o Empresa Auditora de productos agrícolas (fumigación):

Clase. - Código interno generado por el Blockchain de la empresa Auditora, se usa para referenciar el objeto generado en la clase.

AuditorID. - Tiene el valor de los últimos caracteres del código generado por Blockchain para la Empresa Auditora, usado para visualizar y corroborar valores.

Nombre / Razón Social. - El nombre o Razón Social del Auditor o empresa Auditora.

Apellido / Nombre Comercial. - Apellido o Nombre Comercial del Auditor o Empresa Auditora.

Correo electrónico.- Correo electrónico del representante del Auditor o Empresa Auditora.

ANEXO 22 – LISTA PARA MANTENIMIENTO EMPRESA PRODUCTORA

La pantalla que permite seleccionar un registro para mantenimiento de la Empresa Productora de banano (o de la plantación):

Clase. - Código interno generado por el Blockchain de la empresa productora de Banano, se usa para referenciar el objeto generado en la clase.

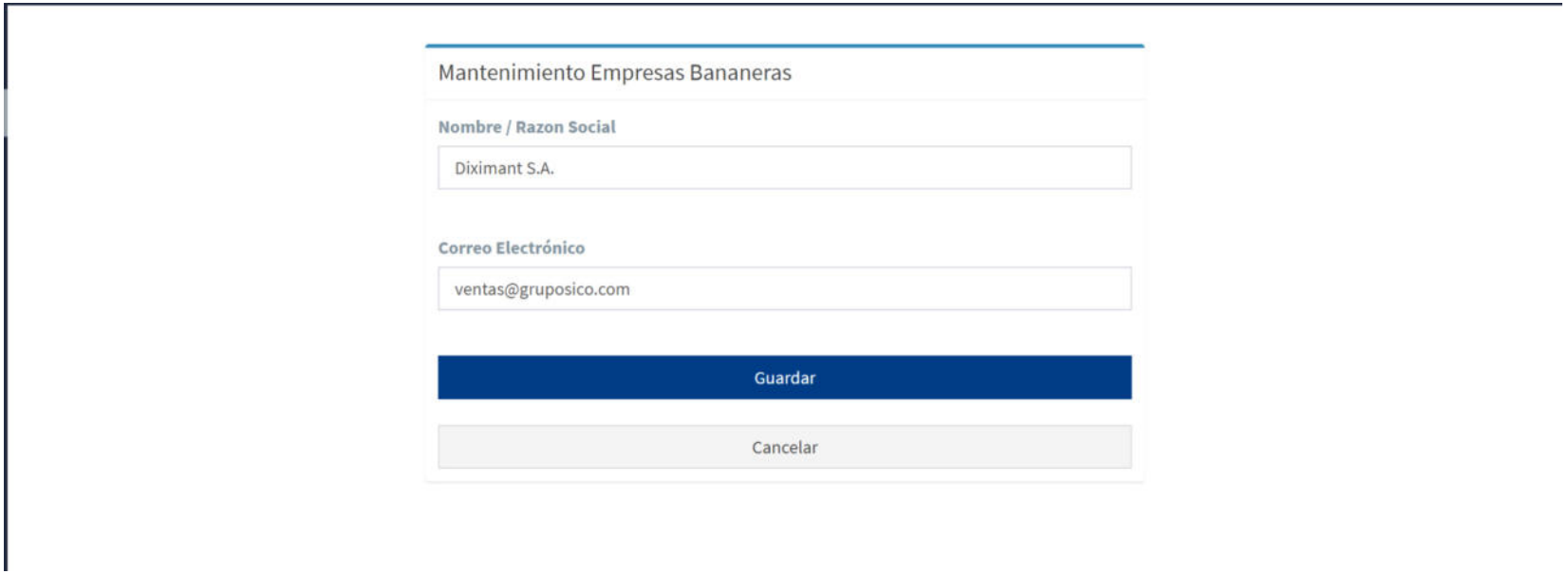
EmpresaID. - Tiene el valor de los últimos caracteres del código generado por Blockchain (Clase), usado para visualizar y comprobar el registro.

Nombre / Razón Social. - El nombre o Razón Social de la Empresa Productora de banano.

Apellido / Nombre Comercial. - Apellido o Nombre Comercial de la Empresa Productora de banano.

Correo electrónico.- Correo electrónico del representante de la Empresa Productora de banano.

ANEXO 23 – MANTENIMIENTO EMPRESAS PRODUCTORAS



Es la pantalla para modificar los valores ingresados del Producto o Empresa Productora de productos agrícolas (fumigación):

Nombre / Razón Social. - El nombre o Razón Social de la Empresa Productora de banano.

Apellido / Nombre Comercial. - Apellido o Nombre Comercial de la Empresa Productora de banano.

Correo electrónico.- Correo electrónico del representante de la Empresa Productora de banano

ANEXO 24 – LISTA MANTENIMIENTO FUMIGADORAS

Pantalla para modificar los valores ingresados de la empresa que aplica los productos agrícolas (Empresas Fumigadoras):

Clase. - Código interno generado por el Blockchain de la empresa productora de Banano, se usa para referenciar el objeto generado en la clase.

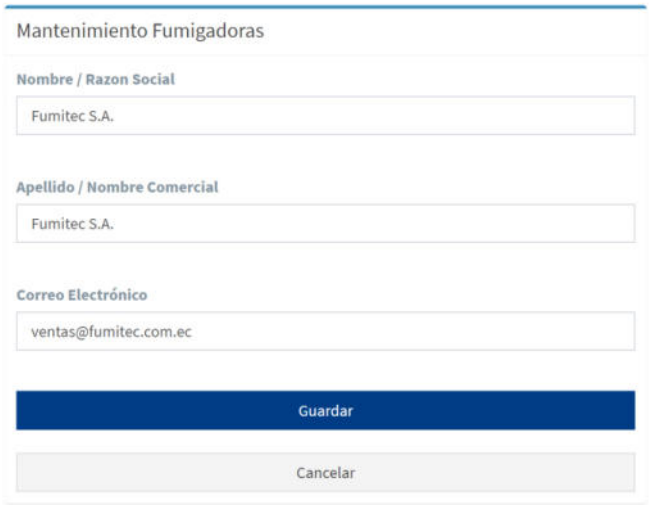
CompañiaID. - Tiene el valor de los últimos caracteres del código generado por Blockchain para la entidad Empresa Fumigadora, usado para visualizar y corroborar valores.

Nombre / Razón Social. - El nombre o Razón Social de la Empresa Fumigadora.

Apellido / Nombre Comercial. - Apellido o Nombre Comercial de la Empresa Fumigadora.

Correo electrónico.- Correo electrónico del representante de la Empresa Fumigadora

ANEXO 25 – EDITA MANTENIMIENTO EMPRESAS FUMIGADORAS



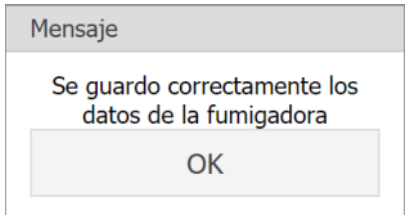
Es la pantalla para modificar los valores ingresados de la Empresa Fumigadora:

Nombre / Razón Social. - El nombre o Razón Social de la Empresa Fumigadora (que en el caso es el Gerente de Producción).

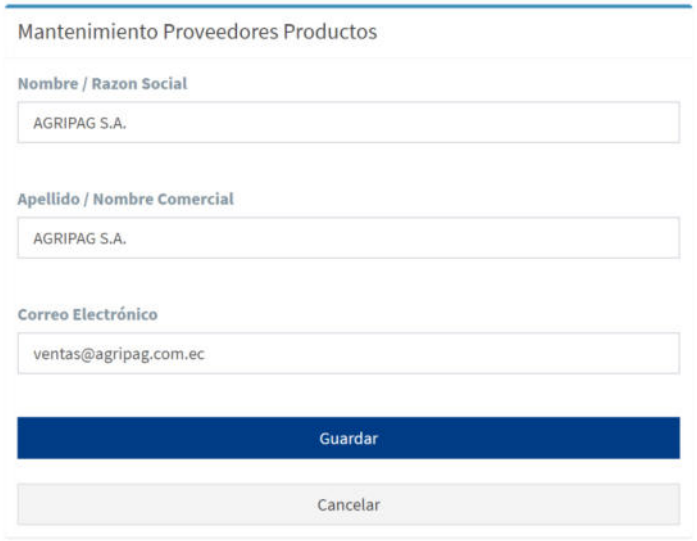
Apellido / Nombre Comercial. - Apellido o Nombre Comercial de la Empresa Fumigadora (que en este caso es el Gerente de Producción).

Correo electrónico. - Correo electrónico del representante de la Empresa Fumigadora.

Cuando los datos se ingresan correctamente, se genera un mensaje de confirmación.



ANEXO 26 – EDITA MANTENIMIENTO PROVEEDORES PRODUCTOS



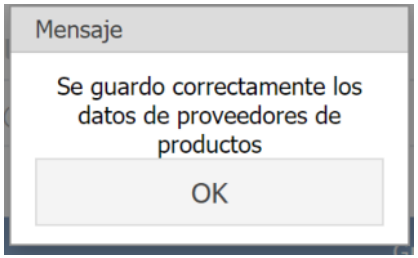
Es la pantalla para modificar los valores ingresados de los proveedores de Productos de Fumigación:

Nombre / Razón Social. - El nombre o Razón Social de los Proveedores de Productos de Fumigación (que en el caso es el Gerente de Producción).

Apellido / Nombre Comercial. - Apellido o Nombre Comercial de los Proveedores de Productos de Fumigación (que en este caso es el Gerente de Producción).

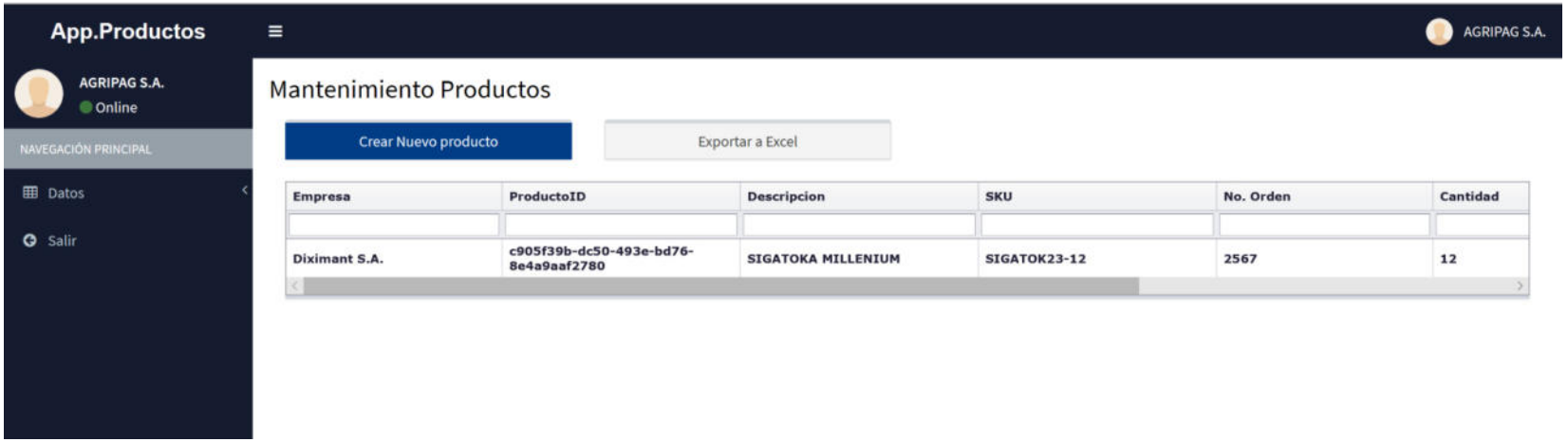
Correo electrónico. - Correo electrónico del representante de los Proveedores de Productos de Fumigación.

Cuando los datos se ingresan correctamente, se genera un mensaje de confirmación.



ANEXO 27 – ADMINISTRADOR APLICATIVO PRODUCTOS BLOCKCHAIN

ANEXO 28 – LISTADO DE PRODUCTOS



Esto es accedido por las empresas proveedoras de productos, lista los productos (referenciados por el SKU) asociados con la orden de compra, que fue generada para la empresa productora agrícola, los campos se describen a continuación:

Empresa. - Empresa proveedora de los productos Agrícolas (en este ejemplo Diximant S.A.).

ProductoID. - Tiene el valor de los últimos caracteres del código interno generado por el Blockchain para el Producto, usado para visualizar y corroborar valores.

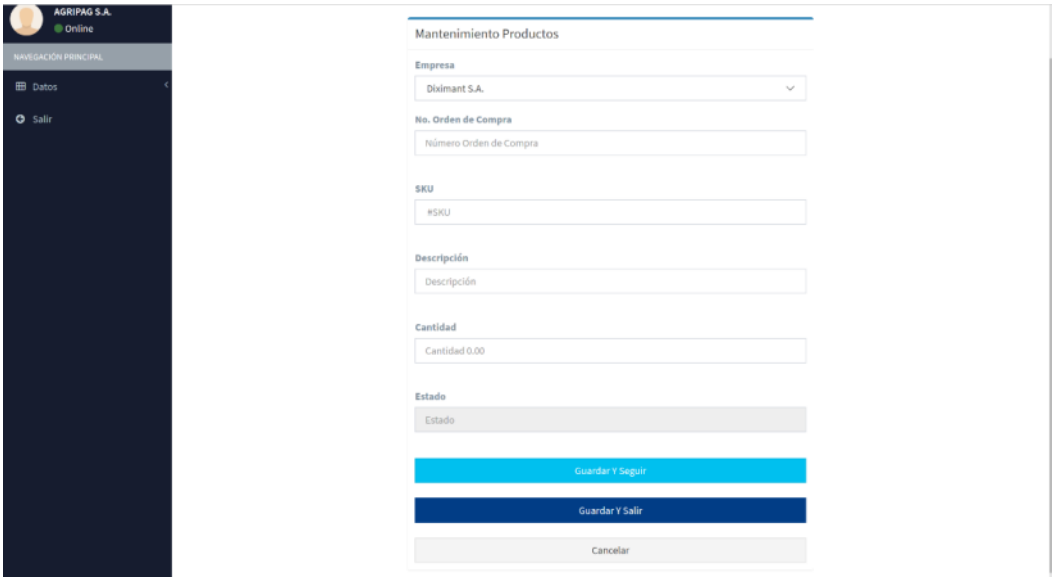
Descripción. - Nombre descriptivo del producto (no genérico).

SKU. - Código único que consiste en letras y números que identifican características de cada producto, como su fabricación, marca, estilo, color y talla. Las compañías productoras de productos emiten su propio y único código SKU que son específicos para el producto que vende.

No Orden. - Código de la orden de compra generada por la empresa proveedora de productos de fumigación hacia la empresa productora de banano, este campo es ingresado manualmente.

Cantidad. - Cantidad de producto (la unidad está relacionada al producto por su SKU).

ANEXO 29 – MANTENIMIENTO DE PRODUCTOS



El usuario asocia el número de orden de compra que el departamento de Compras ingreso hacia el proveedor, con su SKU de cada producto para que se agregue a la cadena.

Empresa. - Empresa proveedora de los productos Agrícolas (en este ejemplo Diximant S.A.).

No Orden de Compra. - Código de la orden de compra generada por la empresa proveedora de productos de fumigación hacia la empresa productora de banano, este campo es ingresado manualmente.

SKU. - Código único que consiste en letras y números que identifican características de cada producto, como su fabricación, marca, estilo, color y talla. Las compañías productoras de productos emiten su propio y único código SKU que son específicos para el producto que vende.

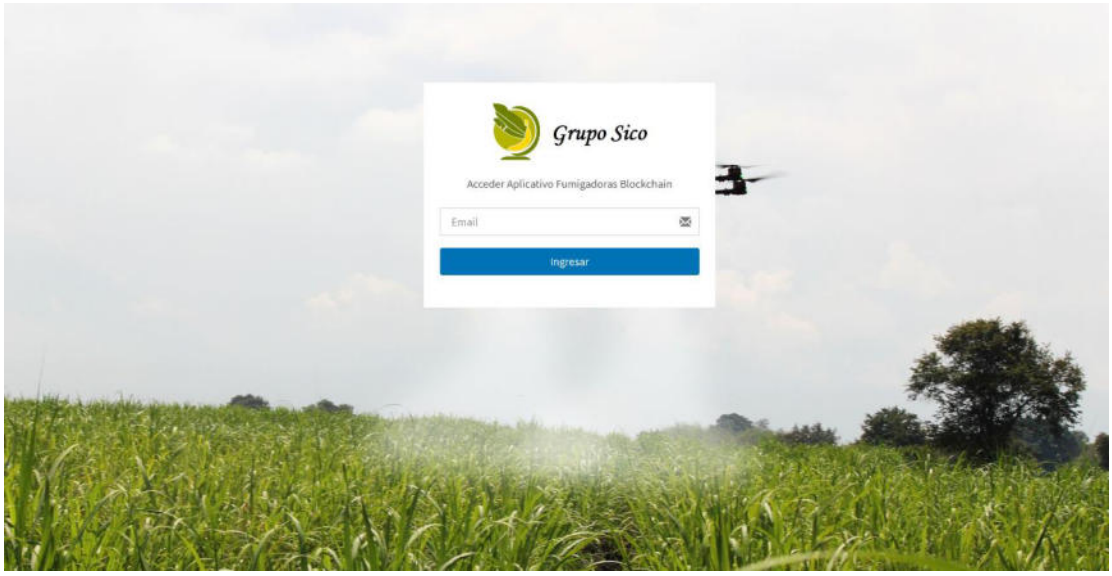
Descripción. - Nombre descriptivo del producto (no genérico).

Cantidad. - Cantidad de producto (la unidad está relacionada al producto por su SKU).

Estado. - Estado interno el producto Activo/ No Activo.

.

ANEXO 30 – APLICATIVO WEB FUMIGADORAS BLOCKCHAIN



Esta es la pantalla de login para las empresas de fumigación que incluirán información de las tareas de ejecución de fumigación.

ANEXO 31 – APP WEB LISTA DE PLANES DE FUMIGACION PENDIENTES

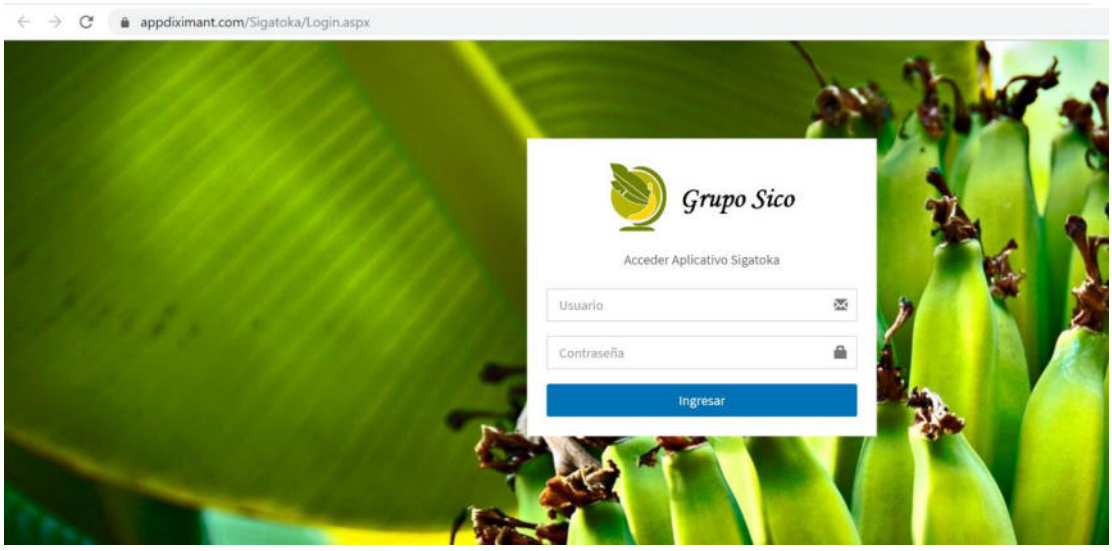
En esta pantalla se ve el listado de Plan de fumigaciones pendientes, al seleccionar uno de los pendientes se puede actualizar los datos sobre la ejecución de la fumigación.

ANEXO 32 – FUMIGACIÓN / TERMINADA

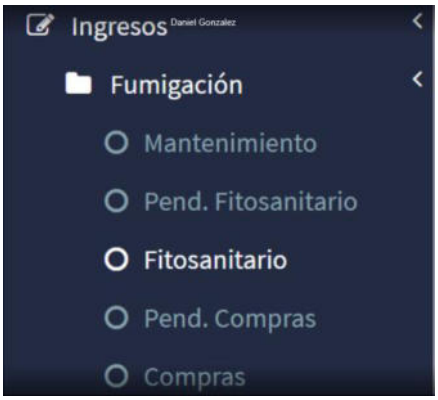
Al seleccionar en la lista de planes de fumigación pendientes ingresados por el Auditor de fumigación (Anexo 31), el audtori puede ver los datos ingresados y los productos relacionados, ingresa la fecha y una observación para el cierre, una vez ingresado puede entrar solo para lectura.

Una vez finalizado el Auditor de fumigación puede ver en otro listado los planes de fumigación finalizados.

ANEXO 33 – PANTALLA DE LOGIN PARA COMPRAS

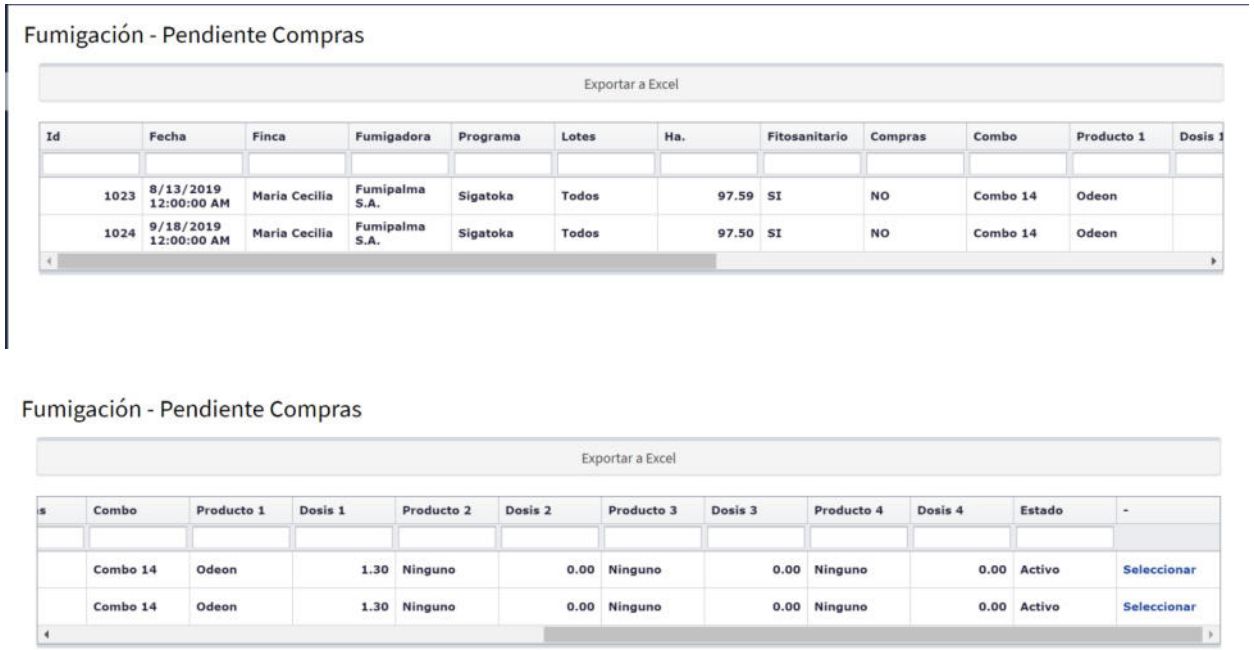


Esta es la pantalla de login para Compras:



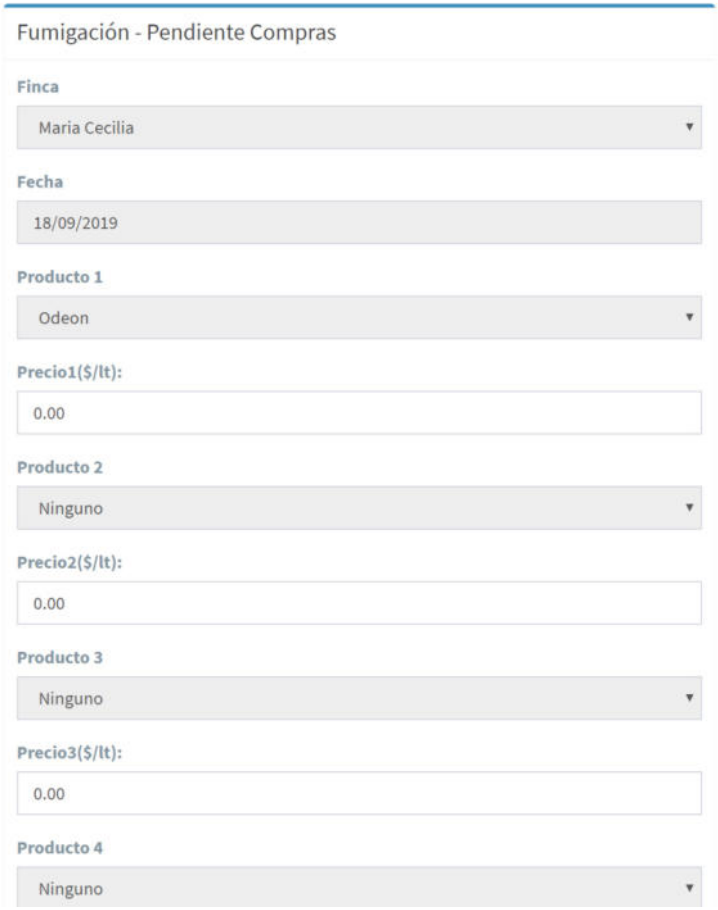
El usuario deberá dar en el enlace de **Pend. de Compras**

ANEXO 34 – FUMIGACIÓN / PENDIENTE DE COMPRAS

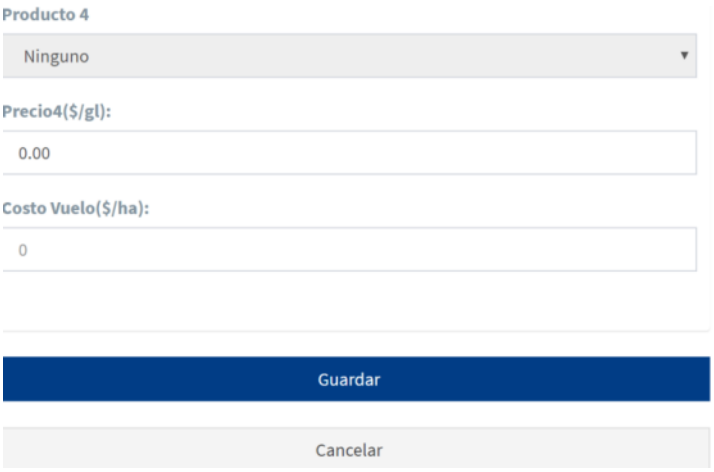


En esta pantalla el Administrador Financiero (Compras) verá todos los planes ejecutados la fumigación, a manera de consulta, para verificar un registro en particular deberá dar click en **Seleccionar.** En el anexo 34 se describen los campos de esa pantalla.

ANEXO 35 – PANTALLA DE CONSULTA DE REGISTRO PENDIENTE DE COMPRAS



En esta pantalla el personal de compras puede consultar las tareas de fumigación aplicadas en cada plan, ingresando 2 variables (precio por galon de producto de fumgiación aplicado y costo de los vuelos realizaods), estas variables se registran por fuera de la cadena de bloques, se las ingresan en la base local de la empresa bananera.



Internamente se realiza un cálculo que se registra en el sitema de la empresa bananera, con esto finaliza un flujo de proceso.