UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

Desarrollo de una aplicación basada en la tecnología de Cadena de Bloques sobre la plataforma Hyperledger Fabric que permite recolectar y almacenar los datos de las transacciones en una base de datos NoSQL.

Trabajo de Seminario
presentado ante la Ilustre
Universidad Central de Venezuela
Por la bachiller:
Shaira Perez
Tutor:
Prof. Jesús Lares
Caracas, Mayo de 2019

Acta

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación, para examinar el Trabajo Especial de Grado titulado Desarrollo de una aplicación basada en la tecnología de Cadena de Bloques sobre la plataforma Hyperledger Fabric que permite recolectar y almacenar los datos de las transacciones en una base de datos NoSQL presentado por la Br. Shaira Coromoto Pérez Méndez (C.I. V-24672360) a los fines de optar al título de Licenciado en Computación, dejamos constancia de lo siguiente:

Leído el trabajo por cada uno de los miembros del jurado, se fijó el día 24 de mayo, a las 10 horas de la mañana, para que el autor lo defendiera en forma pública, haciendo una presentación oral de su contenido, luego de lo cual respondió a las preguntas formuladas.

El profesor y tutor Jesús Lares estará presente vía videoconferencia.

Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobar con la nota de 20 puntos.

En fe de lo cual se levanta la presente Acta, en Caracas el día 24 de mayo de 2019.

prof Robinson Rivas
Holly

Prof. Jesús Lares (Tutor)

Prof. Antonio Russoniello (Jurado)

Profa. Mercy Ospina (Jurado)

Resumen

Tıtulo: Desarrollo de una aplicación basada en la tecnología de Cadena de Bloques sobre la plataforma Hyperledger Fabric que permite recolectar y almacenar los datos de las transacciones en una base de datos NoSQL.

Autor: Shaira Pérez.
Tutor: Jesús Lares.

El presente TEG expone las bondades que brindan las nuevas tecnologías de cadena de bloques y contratos inteligentes sobre plataformas diseñadas para albergar aplicaciones distribuidas, adicionalmente muestra el proceso de creación de una aplicación que se vale de una interfaz web para interactuar con dichas cadenas de bloques y finalmente, albergar dichas interacciones en almacenes de datos NoSQL con la finalidad de aplicar técnicas de analítica predictiva e inteligencia artificial que ayuden a detectar patrones de comportamiento u otro tipo de información de valor que almacenen las cadenas de bloques.

Palabras clave: Cadena de bloques, Contratos inteligentes, Transacciones, Hyperledger, Aplicaciones distribuidas, NoSQL.

Algoritmos

Lista de Tablas

Lista de Figuras

Contenido

Capítulo 1 – Introducción

El crecimiento del comercio electrónico y la movilidad han alimentado el crecimiento exponencial del volumen de transacciones, y está dando señales de las complejidad, vulnerabilidad, ineficiencia y alto costo de los sistemas transaccionales actuales. Se están generando enormes cantidades de datos en industrias de todo el mundo. Pero con demasiada frecuencia estos datos no se gestionan correctamente.

Es por eso que los últimos tiempos han surgido diferentes tecnologías que buscar minimizar estos problemas, entre ellas la arquitectura de cadena de bloques, la cual permite a los participantes compartir un libro mayor actualizado, a través de replicaciones de nodo a nodo (modelo P2P) de la red, cada vez que una transacción ocurre. La replicación P2P, significa que cada participante (nodo) en la red actúa como tanto como publicador como suscriptor. Cada nodo puede recibir o enviar transacciones a otros nodos, y los datos son sincronizados a través de la red a medida que son transferidos.

Aunque en principio la blockchain fue creada para ser un gran libro de contabilidad público y accesible para cualquiera, el desarrollo de esta tecnología por parte de distintas entidades y empresas alrededor del mundo le ha dado un matiz privado que la ha dividido esencialmente en dos categorías y cuatro tipos.

Basándose en el acceso a los datos almacenados, podemos encontrarla pública o privada. En la primera, no hay ninguna restricción para la lectura de datos ni la realización de las operaciones por parte de los usuarios; en cambio, en la segunda, tanto la lectura como las operaciones se limitan a participantes determinados.

Por otro lado, basándose en la capacidad para generar bloques, se divide en aquellas sin permisos (permisionless) y con permisos (permissioned). En la primera no hay restricciones para poder realizar transacciones y crear nuevos bloques, de modo que se ofrecen monedas o activos digitales nativos de la red como recompensa a los usuarios que quieran mantener la red. Es descentralización, tal como Bitcoin. Las segundas son desarrolladas por entidades generalmente privadas, en muchos casos para uso interno, y los usuarios de estas necesitan permisos por parte de los administradores de la red para interactuar con el protocolo. Este es el tipo de blockchain que están probando los bancos: son centralizadas, es decir, controladas por la entidad y no por los usuarios.

La presente investigación realizara un análisis exhaustivo con respecto al desarrollo de aplicaciones que interactúan con la tecnología de cadena de bloques desde un enfoque privado y permisado mediante el uso de contratos inteligentes.

El capítulo II define el planteamiento del problema, justificación, el objetivo general, objetivos específicos y la arquitectura propuesta para el presente TEG.

A lo largo del capítulo III se encuentra el marco teórico, el cual define todos los conceptos teóricos y prácticos que sirven de referencia para el desarrollo planteado.

En el capítulo IV se describen las herramientas tecnológicas que se utilizaran durante el desarrollo junto con las justificaciones de uso de las mismas.

La metodología de desarrollo se describe a lo largo del capítulo V junto con su justificación y fases definidas.

El capítulo VI muestra las fases de la metodología de desarrollo aplicadas y sigue cuidadosamente el proceso del desarrollo de la herramienta, junto con pruebas y despliegues en ambiente de desarrollo.

Finalmente, el capítulo VII corresponde a las conclusiones obtenidas durante todo el desarrollo del TEG.

Capítulo 2 - Planteamiento del problema

2.1. Planteamiento del problema

Los enfoques y nuevos paradigmas que plantean las cadenas de bloques y contratos inteligentes (Smart contracts), dan solución a un gran número de problemas en todo tipo de ámbitos, desde una pequeña empresa hasta administraciones gubernamentales. Básicamente puede aplicarse a cualquier escenario donde quiera llevarse un registro, desde el origen a través del tiempo, de un bien físico o virtual, como puede ser una imagen digital o un neumático de un auto.

Cada vez más industrias están entendiendo el poder de la tecnología blockchain. Debido a su inmutabilidad y descentralización, blockchain ha hecho que el intercambio de información entre compañías sea extremadamente simple y directo.

Sin embargo, el tipo de blockchain que se ha explotado hasta ahora es el público, y este tipo de cadena de bloques no es ideal para el uso empresarial. Los dos mayores problemas con las cadenas de bloques públicas son la escalabilidad y la privacidad. La razón es bastante simple las empresas tratan con millones de transacciones cada segundo, eso está muy por encima de lo que pueden manejar las blockchains públicas en este momento, Bitcoin apenas puede gestionar 7-8 transacciones por segundo. El tiempo de confirmación del bloqueo es de 10 minutos, lo que se suma a la latencia. Las grandes empresas necesitan lidiar con millones de transacciones por día con una latencia cercana a 0. Además, las empresas manejan una gran cantidad de datos confidenciales por lo que no pueden permitir que cualquiera se una a su red.

Algunos de los inconvenientes de las blockchain públicas son resueltos con las cadenas de bloques privadas, ya que han sido diseñadas para cubrir las necesidades de las empresas.

El proyecto Hyperledger es una estrategia global con múltiples plataformas para desarrollar soluciones empresariales. Este resuelve problemas de escalabilidad y privacidad del rendimiento mediante un modo de operación autorizado y un control de acceso detallado. Además, tiene una arquitectura modular que permite que se personalice para una multitud de aplicaciones, de manera análoga a una caja de herramientas.

El presente trabajo de investigación plantea realizar un análisis certero sobre casos de uso que pueden ser resueltos mediante esta tecnología, estudiar metodologías adecuadas para la implementación de contratos inteligentes y la creación de una aplicación que, dado un caso de uso particular, conviva con la cadena de bloques definida dentro de alguna de las plataformas de Hyperledger, con la finalidad de solucionar mediante contratos inteligentes problemáticas existentes y además recopilar datos de importancia en el proceso.

2.2. Justificación

Actualmente, algo tan sencillo como el nombre de un cliente, empleado o proveedor es información que las empresas deben proteger, no por el hecho de que forma parte de los datos de la empresa, sino porque también existe un marco regulatorio referente a cada industria relacionado a la privacidad e identidad.

Por lo que es necesario estudiar el desarrollo de aplicaciones desde este requerimiento para resolver problemas que no han sido atacados y que van surgiendo a medida que las industrias evolucionan.

La vialidad, es uno de los factores más importantes a la hora de pensar en cualquier solución tecnológica, el presente TEG también plantea evaluar que tan viables son estos nuevos enfoques y desarrollos, qué tanta complejidad se despliega de este tipo de soluciones y finalmente, observar que tan eficientes y eficaces son con respecto al caso de uso seleccionado.

Adicionalmente, con el acogimiento masivo de este tipo de tecnología, viene siendo crucial encontrar maneras de recopilar los datos, generados por estas aplicaciones, de una manera eficiente ya que pueden ser de vital importancia para el crecimiento del sector. De cara a estudios futuros que se pueden realizar mediante técnicas como la analítica predictiva y la inteligencia artificial, junto con la proyección que este tipo tecnologías tienen hacia el futuro debido a su transparencia y eficiencia, la tarea de recopilación de datos para ser analizados posteriormente y detectar patrones de comportamiento se vuelve fundamental para brindar una solución completa en todos los aspectos. La posibilidad de generar estos datos estructurados de una manera eficiente para que los expertos en el área de análisis de datos puedan realizar la tarea de analizar apropiadamente dichos datos es una característica innovadora y de vital importancia para este tipo de soluciones.

2.3. Objetivo general

Desarrollar una aplicación basada en un caso de uso previamente seleccionado, sobre la tecnología de cadena de bloques (Blockchain) utilizando el framework Hyperledger Fabric y sus contratos inteligentes y adicionalmente capturar las interacciones de los usuarios con la cadena de bloques para ser almacenadas en una base de datos NoSQL, para que los expertos en el área de análisis de datos puedan realizar la tarea de analizar apropiadamente dichos datos.

2.4. Objetivos específicos

1. **Seleccionar un caso de uso:** Seleccionar una problemática existente a la cual sea viable la aplicación de las herramientas investigadas para dar con soluciones eficientes.
2. **Seleccionar una metodología de desarrollo:** Seleccionar la metodología de desarrollo

que proponga una estructura de trabajo eficiente para desarrollar una solución eficaz al caso de uso planteado.

3. **Preparar el ambiente de desarrollo para Hyperledger Fabric:** Preparación de los servidores a utilizar en el ambiente de Fabric tanto externas como de contrato, estudiar y preparar las variables necesarias para solventar la problemática y proponer una implementación de contrato inteligente.

4. **Implementar la aplicación sobre la base tecnológica de cadena de bloques y contratos inteligentes:** Implementar la aplicación de interacción con la cadena de bloques utilizando Hyperledger Fabric como herramienta desarrollo.

5. **Pruebas de funcionamiento:** Realizar pruebas a la aplicación en interacción con la infraestructura de Fabric en una red de desarrollo, con el fin de verificar su funcionamiento y capacidad para resolver la problemática seleccionada.

6. **Puesta en marcha:** Poner en marcha en un ambiente de producción la aplicación desarrollada.

2.5. Arquitectura propuesta para TEG

La presente arquitectura (Figura 2-1) presenta una estructura de tres capas principales. La primera capa se encarga de la interacción directa con la cadena de bloques de Fabric mediante contratos inteligentes, en esta capa ocurre la recopilación de datos a los cuales les serán aplicados los diferentes métodos analíticos y predictivos.

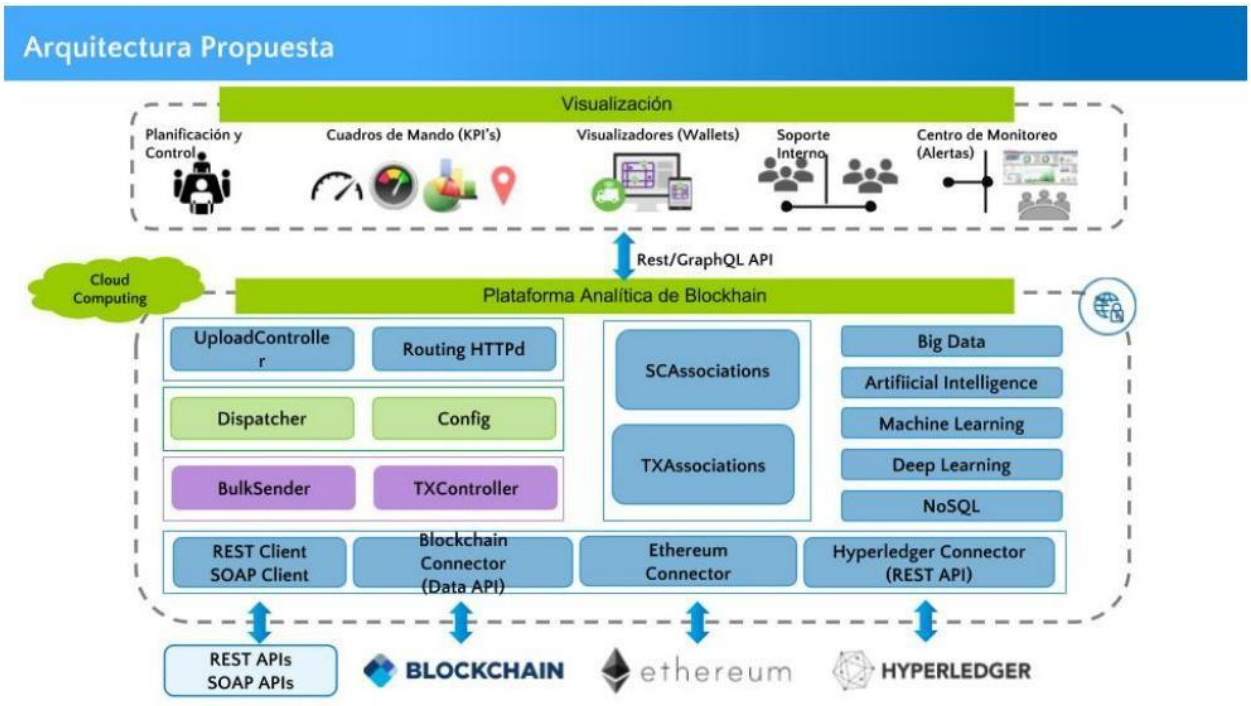

Figura 2-1: Representación gráfica de la arquitectura propuesta para el TEG.

En la segunda capa se realiza el preprocesamiento de los datos recopilados y los diferentes

algoritmos de analítica predictiva e inteligencia artificial utilizando una plataforma Big Data. Es la capa intermedia entre la recopilación de los datos y la interfaz de usuario el cual se encarga de mostrar los resultados obtenidos.

Finalmente se encuentra la capa de visualización la cual mostrará los resultados obtenidos al usuario final desde la segunda capa. Esta tercera capa está provista de distintos métodos de visualización dependiendo del tipo de indicador que se especifique.

2.6. Alcance del TEG

El TEG pretende implementar solo algunos módulos de la arquitectura planteada, los cuales serán definidos sobre la marcha a medida que se desarrolle el TEG, el resto de los módulos, debido a su amplio espectro de alcance, podrán ser implementados en otros TEG en el futuro.

La presente investigación se centra en la creación de módulos en la primera capa, con respecto al conector de Hyperledger Fabric para la recopilación de datos, la definición y llenado del almacén de datos NoSQL contemplado en la segunda capa y posiblemente un elemento de presentación de la capa de visualización.

Capítulo 3 – Marco teórico

3.1. Cadena de bloques (Blockchain)

6.3.4.1. 3.1.1. Definición

La estructura denominada cadena de bloques es un libro mayor público, distribuido, donde todos los participantes pueden inspeccionar, pero ninguno controla. Esta estructura facilita el proceso de registrar transacciones y llevar contabilidad de activos en una red de negocios.

El término cadena de bloques (blockchain) proviene de la manera en que almacena transacciones en bloques que son enlazados para formar una cadena. En los bloques se registra el detalle y confirmación del tiempo de ocurrencia y secuencia de las transacciones de activos; la cadena de bloques crece a medida que se registran transacciones.

Un activo puede ser tangible, una vivienda, un vehículo, una propiedad; o puede ser intangible, como una propiedad intelectual, patente, derechos de autor o una marca. Virtualmente cualquier valor puede ser registrado y transado en una red basada en la cadena de bloques, reduciendo el riesgo y disminuyendo costos para los actores involucrados.

El libro mayor de un activo está soportado por una cadena de bloques que comparten una red de participantes que realizan transacciones con el activo. El libro mayor se ejecuta en la red de nodos participantes, las reglas acordadas y el consenso distribuido basado en mecanismos de incentivo permite la validación de transacciones en la red de manera descentralizada. El incentivo busca mantener a los nodos honestos en la red como veremos más adelante.

La Figura 3-1 ilustra la estructura genérica de una cadena de bloques, el bloque a la izquierda representa primer bloque de la cadena o denominado bloque génesis.

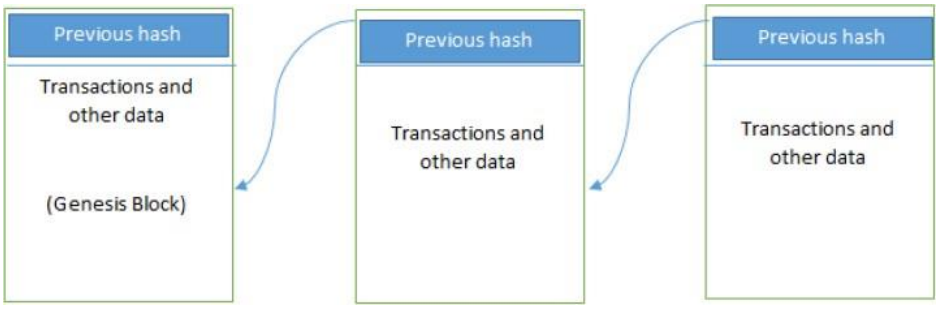

Figura 3-1: Estructura genérica de una cadena de bloques

Cada bloque contiene un hash, un grupo de transacciones validadas por la red de participantes en un tiempo dado, con su respectiva marca de tiempo y el hash del bloque previo. El hash del bloque previo enlaza los bloques y previene que cualquier bloque sea alterado o se inserte un bloque entre dos bloques existentes. De esta manera, cada bloque subsecuente fortalece la verificación del bloque previo y por lo tanto de toda la cadena de bloques.

Una red de negocios basada en blockchain es particularmente valiosa para incrementar el nivel de confianza entre los actores, ya que cada transacción está construida sobre las demás transacciones y cualquier alteración se hace evidente rápidamente, ya que todos los nodos tienen visibilidad de las transacciones. El método hace que la cadena de bloques sea evidente al ser alterado su contenido, lo que representa la propiedad inmutable de la estructura [1].

6.3.4.2. 3.1.2. Propiedades

Para construir un libro mayor digital para uso en un ambiente como la red internet, donde los participantes de este libro mayor no tienen ninguna relación de confianza establecida, la estructura de datos que soporta el libro mayor, debe contar con algunas propiedades deseables, entre ellas:

1. La estructura de datos (cadena de bloques) debe ser inmutable, es decir, la estructura de datos únicamente debe permitir agregar datos, no es posible modificar, remover o reordenar datos registrados.
2. Debe haber una manera de obtener un resumen criptográfico del estado de la estructura de datos en cualquier momento.

Un resumen criptográfico es un dato de tipo *String*, de corta longitud, que evita tener que almacenar la estructura de datos completa. Si la estructura de datos fue manipulada de alguna manera, el resumen criptográfico cambiaría y se detectaría la manipulación.

La justificación de estas propiedades, es que a diferencia de una estructura de datos regular que está almacenada en una máquina única, el blockchain es una estructura de datos global mantenida colectivamente por un conjunto de participantes sin relaciones mutuas de confianza.

Este funcionamiento contrasta con otro enfoque para descentralizar libros mayores digitales, en donde muchos participantes mantienen libros mayores locales y es responsabilidad delusuario hacer consultas en el conjunto de libros mayores para resolver cualquier conflicto [2].

La cadena de bloques tiene las siguientes propiedades:

1. Consenso: para que una transacción sea válida, un número de participantes debe estar de acuerdo en su validez.
2. Procedencia: los participantes saben de dónde viene el activo y como el propietario del activo ha cambiado en el tiempo.
3. Inmutabilidad: ningún participante puede manipular una transacción luego que ha sido registrada en la cadena de bloques.
4. Finalidad: la cadena de bloques permite determinar el propietario de un activo o la completitud de una transacción.

Las propiedades de la cadena de bloques permiten generar confianza a través de una red de negocios, de manera que las relaciones de confianza entre las partes no son necesarias al

momento de realizar una transacción.

Una red de negocios basada en blockchain además reduce la carga de un sistema regulatorio al facilitar la transparencia y auditoría de transacciones para verificar su completitud.

La estructura de blockchain permite construir confianza a través de los siguientes atributos:

1. Distribuido y sostenible: el libro mayor es compartido, actualizado con cada transacción y replicado entre los participantes en tiempo real, no depende ni es controlado por algún individuo u organización.
2. Seguro, privado e indeleble: la permisología y la criptografía previenen acceso no autorizado a través de la red y asegura que los participantes son quienes dicen ser. La privacidad es mantenida a través de técnicas criptográficas que permiten a los participantes acceso a la cadena de bloques, tanto las transacciones como la identidad de las actores de la transacción pueden ser enmascaradas.
3. Transparente y auditable: ya que los participantes en una transacción tienen acceso a los mismos registros, pueden validar transacciones y verificar identidades sin la necesidad de intermediarios o terceros. Las transacciones contienen marcas de tiempo y pueden ser verificadas prácticamente en tiempo real.
4. Basado en consenso y transaccional: todos los participantes relevantes de la red deben estar de acuerdo que una transacción es válida. Esto se logra usando algoritmos de consenso, cada red blockchain puede establecer las condiciones para realizar una transacción.
5. Orquestado y flexible: ya que las reglas de negocio y los contratos inteligentes (que se ejecutan en una o más condiciones) pueden ser construidos en la plataforma, las redes de negocios basadas en la cadena de bloques pueden evolucionar mientras maduran para soportar procesos de negocios y un amplio rango de actividades [1].

3.2. Antecedentes académicos

La estructura de la cadena de bloques de la criptomoneda Bitcoin, es tomada prestada con unas mínimas modificaciones de una serie de artículos de Stuart Haber y Scott Stornetta, escritos entre 1990 y 1997 [3].

El trabajo de Haber y Stornetta estaba relacionado con los problemas del marcado de tiempo (timestamp) de documentos, orientados a construir un servicio de tipo "notaria digital". Para casos de uso como patentes, contratos de negocios y otros documentos de valor, es deseable registrar el tiempo de creación de un documento en un momento dado y no más tarde.

La noción de documento en el trabajo de Haber y Stornetta es bastante general y podría ser de cualquier tipo de dato; aunque mencionan transacciones financieras como una potencial aplicación, no era el foco de la publicación.

En una versión simplificada de la propuesta de Haber y Stornetta, los documentos están constantemente siendo creados y transmitidos. El creador de cada documento establece un tiempo de creación, firma el documento, coloca la marca de tiempo y un apuntador al documento previamente transmitido.

Este documento previo, a su vez, apunta a su predecesor, de manera que los documentos forman una cadena con apuntadores en sentido inverso al tiempo. Un usuario externo no puede alterar el mensaje con su marcado de tiempo ya que está firmado por el creador, y el creador no puede alterar el mensaje sin también alterar la cadena entera de mensajes que le suceden.

Si se obtiene un único ítem de la cadena a través de una fuente confiable (otro usuario o un servicio especializado de marcado de tiempo), la cadena entera hasta ese punto está bloqueada, es inmutable y está ordenada en tiempo.

En artículos sucesivos, Haber y Stornetta introducen otras ideas que hacen que esta estructura de datos sea más efectiva y eficiente:

1. Los enlaces entre documentos pueden ser creados usando hashes en vez de firmas; el hash es más simple y fácil de procesar, estos enlaces son denominados apuntadores hash.
2. En vez de indexar documentos individualmente, lo cual puede ser ineficiente si muchos documentos son creados aproximadamente al mismo tiempo. La propuesta es agrupar los documentos en grupos o bloques, donde los documentos en cada bloque tienen esencialmente la misma marca de tiempo.
3. En cada bloque, los documentos pueden ser enlazados entre ellos con apuntadores hash en un árbol binario, denominado árbol Merkle, en vez de una cadena lineal.
4. De manera casual, Josh Benaloh y Michael de Mare independientemente introducen estas tres ideas en 1991, poco después del primer artículo de Haber y Stornetta [4].

3.3. Bitcoin Blockchain

La criptomoneda Bitcoin usa esencialmente la estructura de datos en los artículos de Haber y Stornetta de los años 1991 y 1997, y hacen reingeniería de sus propiedades de seguridad añadiendo el esquema de comprobación de trabajo (proof-of-work, en inglés).

Bitcoin y su cadena de bloques, puede ser la mejor instanciación conocida de las estructuras de datos de Haber y Stornetta. Sin embargo, no es la primera implementación, al menos dos empresas habían tenido una aproximación. A mediado de la década de 1990, la empresa Surety y en el año 2007, la empresa Guardtime ofrecen servicios de marcado de tiempo de documentos.

3.4. Arboles Merkle

Es una estructura de datos, que recibe el nombre por Ralph Merkle, un pionero de la

criptografía asimétrica que propuso la idea en un artículo en el año 1980. El objetivo era producir un resumen para un directorio público de certificados digitales.

Por ejemplo, un sitio web presenta al usuario con un certificado, también puede presentar una corta prueba que el certificado aparece en el directorio global. Se puede verificar eficientemente la prueba mientras se conozca el hash raíz del árbol Merkle de los certificados del directorio.

Verificación eficiente de las partes de un estado global es una de las funcionalidades claves provistas por el libro mayor en el blockchain Ethereum, una plataforma basada en blockchain para contratos inteligentes que trataremos más adelante.

La siguiente Figura 3-2 ilustra de manera simplificada la estructura de datos propuesta por Haber y Stornetta, que enlaza los bloques marcados de tiempo usando apuntadores hash.

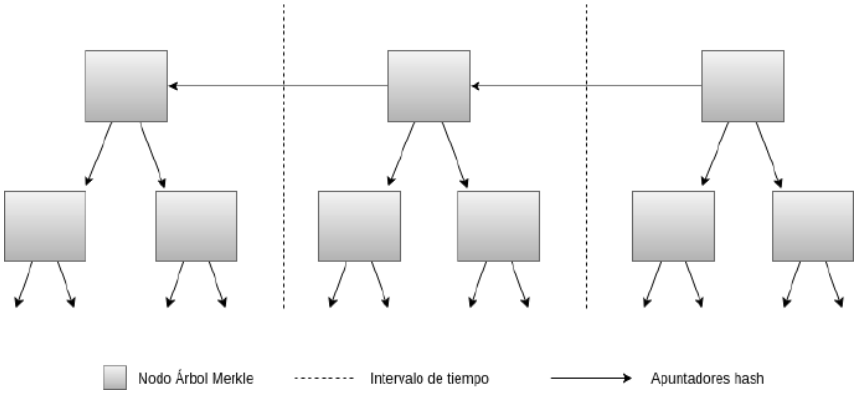

Figura 3-2: Representación simplificada del árbol Merkle en blockchain.

El árbol Merkle es un árbol binario que almacena datos en sus nodos hoja, por cada nodo hoja que contiene datos, es generado su código hash y se concatena con el código hash de su nodo hermano; a la concatenación de estos dos hash, se le vuelve a aplicar la misma función criptográfica, al cual se convierte en el nodo padre de dichos nodos hoja, este proceso se repite sucesivamente hasta llegar a la raíz del árbol.

Esta estructura es utilizada debido a que el orden de acceso de un árbol es menor que usar una estructura secuencial.

En el caso de la criptomoneda Bitcoin, el blockchain de Bitcoin emplea los árboles Merkle y las transacciones toman la forma de documentos. En el árbol Merkle de cada bloque, los nodos hoja son transacciones y cada nodo internamente contiene dos apuntadores.

Esta estructura de datos tiene dos propiedades importantes: primero, el hash del último bloque es un resumen criptográfico, algún cambio en cualquier transacción (nodos hoja) requerirá propagar los cambios hasta la raíz del bloque y las raíces de todos los bloques. Por lo tanto, si se conoce el último hash, es posible obtener el resto del libro mayor desde una fuente no confiable y verificar que no ha cambiado.

Un argumento similar establece la segunda propiedad importante de la estructura de datos, se puede probar eficientemente que una transacción particular está incluida en el libro mayor.

El usuario tendría que enviar un pequeño número de nodos en ese bloque de la transacción (este es uno de los objetivos del árbol Merkle), así como una pequeña cantidad de información para cada siguiente bloque. La habilidad para probar inclusión de transacciones eficientemente es altamente deseable por rendimiento y escalabilidad.

La Figura 3-3 ilustra el detalle de la estructura del árbol Merkle en un bloque del blockchain.

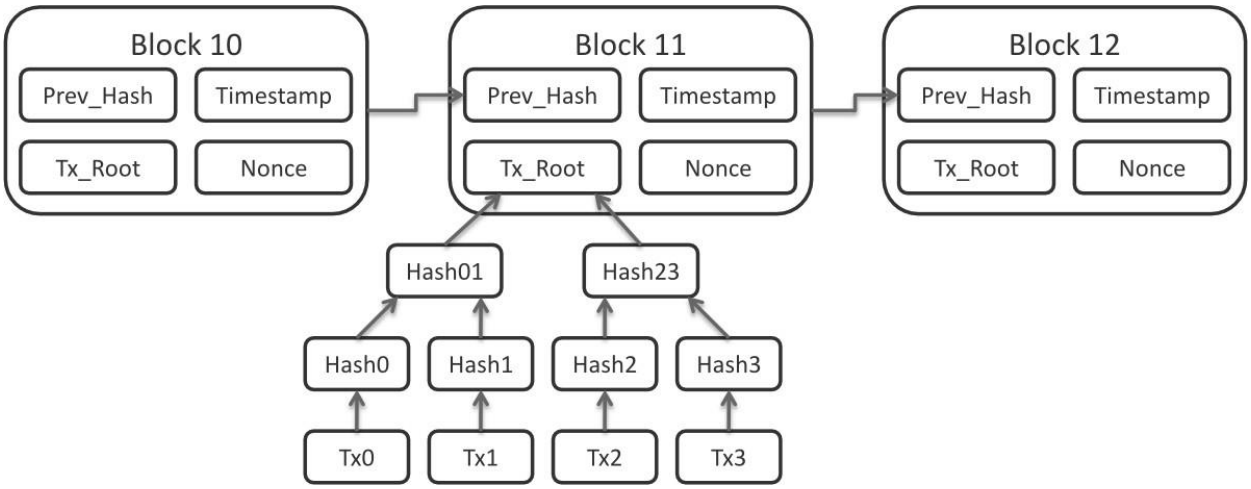

Figura 3-3: El detalle del árbol Merkle en el blockchain.
Recuperado de: https://viviendo20.wordpress.com/2018/06/08/blockchain-el-arbol-de-merkle/

En vista que una cadena de bloques irá aumentando de tamaño con el número de transacciones, la estructura basada en el árbol Merkle permite ahorrar espacio de almacenamiento, compactando el detalle de las transacciones y manteniendo el hash raíz del bloque y los hashes de las transacciones anteriores [5].

La Figura 3-4 muestra un bloque con el detalle de las transacciones y el mismo bloque sin el detalle de las transacciones, únicamente el hash de las transacciones.

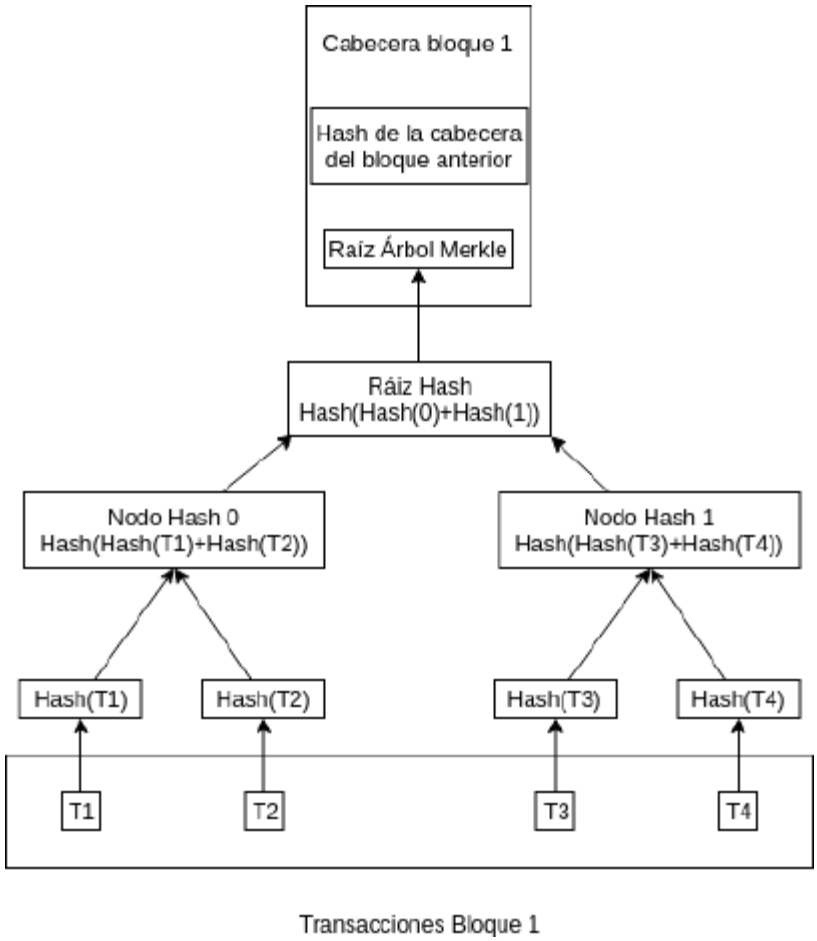

Figura 3-4: Cadena de bloques y el árbol Merkle. Recuperado y traducido de: https://bitcoin.org/bitcoin.pdf

3.5. Consenso

6.3.4.3. 3.5.1. Definición

Para que una transacción sea válida, todos los participantes deben estar de acuerdo con su validez [1]. Este proceso se conoce como "consenso" y ayuda a mantener transacciones inexactas o potencialmente fraudulentas fuera de la base de datos [6]. Esto consiste en los estándares y reglas de cómo cada nodo intercambia la información, las reglas matemáticas para todos los nodos para acordar la integridad de esos datos y el incentivo de pago para apoyar el modelo de consenso. Además de garantizar que todas las transacciones sean válidas se debe asegurar que todas las transacciones válidas se agregan una sola vez, y no se pueden omitir transacciones válidas [7].

6.3.4.4. 3.5.2. Tolerancia a fallas bizantinas (BFT)

Un problema fundamental en la computación distribuida y sistemas multi-agentes es lograr una confiabilidad general ante la presencia de un número de procesos imperfectos. Esto

generalmente requiere que los procesos se pongan de acuerdo en ciertos datos que son requeridos durante un proceso computacional. Este proceso se define como consenso, para crear un protocolo de consenso seguro, debe ser tolerante a fallas.

Los requerimientos de una moneda digital en la red internet sin una autoridad central son más rigurosos. Un libro mayor distribuido inevitablemente tendrá bifurcaciones, lo que significa que algunos nodos pensarán que un bloque A es el más reciente, mientras otros nodos pensarán que el bloque más reciente es el bloque B.

Esto puede ser porque un nodo mal intencionado intenta manipular el libro mayor o latencia de la red, como resultado se pueden generar bloques de transacciones casi simultáneamente por diferentes nodos sin tener conciencia de los bloques de otros. En el año 1998, Mike Just publicó un artículo para demostrar que las marcas de tiempo enlazadas no son suficientes para resolver bifurcaciones.

El blockchain es un libro mayor descentralizado, que por definición, no está controlado por una autoridad central. Debido al valor almacenado en estas cadenas de bloques, actores maliciosos tienen incentivos económicos para tratar de causar fallas. Por lo tanto, la tolerancia a fallas bizantinas para blockchains es fundamental.

El área de investigación de tolerancia a fallas en la computación distribuida ha estudiado este problema, que tiene varias denominaciones, incluyendo la replicación de estado. Una solución a este problema es uno que permite a un conjunto de nodos a aplicar las mismas transiciones de estado en el mismo orden, generalmente el orden preciso no importa, sino que los nodos sean consistentes. Para una moneda digital, el estado que debe replicarse es el conjunto de balances y las transacciones son transiciones de estado.

En el año 1989, Leslie Lamport propuso Paxos, una familia de protocolos para lograr consenso en una red de participantes. Estos protocolos consideran la replicación del estado cuando los canales de comunicación no son confiables o cuando una minoría de nodos exhiben ciertas fallas, tales como estar fuera de comunicación o reiniciarse enviando mensajes desactualizados.

Una línea de investigación relacionada estudia la situación donde la red es bastante confiable (los mensajes son entregados con una demora controlada), pero la definición de “falla” es expandida para manejar cualquier desviación del protocolo. Tales fallas bizantinas incluyen tanto fallas naturales como comportamientos maliciosos. Fueron publicados en un artículo también de Lamport, escrito en conjunto con Robert Shostak y Marsall Pease en el año 1982.

En el artículo original de Bitcoin de Satoshi Nakamoto, no hay citas directas a bibliografía acerca de tolerancia a fallas bizantinas, sin embargo usa algunos conceptos y se refiere al protocolo del blockchain de Bitcoin, como un mecanismo de consenso y considera “fallas” tanto los atacantes de la red, como los nodos que se unen y se van de la red.

En la bibliografía considerada para la elaboración de este seminario, encontramos que cuando le preguntan a Nakamoto por su opinión de la relación de Bitcoin con la tolerancia a fallas bizantina, la respuesta de Nakamoto es que el prueba de trabajo (proof-of-work en inglés) soluciona estos problemas.

En los años siguientes, otros académicos han estudiado el mecanismo de consenso de Nakamoto desde la perspectiva de sistemas distribuidos. Algunos consideran que las propiedades de Bitcoin son débiles, mientras que otros plantean que la perspectiva de tolerancia a fallas bizantina no se ajusta a las propiedades de consistencia de Bitcoin.

Sin embargo, todos estos trabajos de investigación asumen una conducta "honesta" entre un subconjunto de participantes, mientras que Nakamoto sugiere que no se debe asumir una conducta honesta de los nodos, sino se debe incentivar su honestidad [8].

6.3.4.5. 3.5.3. Tolerancia fallos de bloqueo (Crash Faults Tolerance)

Las fallas de bloqueo son las fallas que hacen que el sistema se detenga o se bloquee. En un sistema distribuido, hay una variedad de amenazas contra las que un algoritmo de consenso debe ser resistente. Los procesos pueden estrellarse. Las máquinas y dispositivos pueden fallar. Las conexiones de red pueden ser interrumpidas. La tolerancia a fallos de bloqueo es un nivel de resistencia, donde el sistema aún puede alcanzar correctamente el consenso si fallan los componentes [10].

6.3.4.6. 3.5.4. Tipos de consenso

3.5.4.1. Prueba de trabajo (Proof-Of-Work)

Virtualmente todos los sistemas tolerantes a fallas asumen que una estricta mayoría (más de la mitad o dos tercios) de los nodos en el sistema son tanto honestos como confiables. En una red abierta P2P, no hay registro de nodos, y los nodos pueden unirse e irse de la red. Por lo tanto, un adversario puede crear suficiente nodos ficticios para superar la garantía de consenso del sistema.

El ataque de nodos ficticios (Sybils o Sockpuppets en inglés) fue formalizado en el año 2002 por John Douceur [REF], quien usó una construcción criptográfica denominada proof-of-work (prueba de trabajo en inglés) para mitigar este problema.

La primera propuesta de proof-of-work fue creada en 1992 por Cynthia Dwork y Moni Naor, el objetivo era disminuir el spam de correos electrónicos. Es importante notar que el spam, ataques de nodos ficticios y negación de servicio son problemas similares, ya que el adversario amplifica la influencia en la red comparado con usuarios regulares. La propuesta de Proof-of-work es aplicable como mecanismo de defensa en los tres casos.

En el diseño de Dwork y Naor, los receptores de correo electrónico procesarían únicamente los correos que fueron acompañados por la prueba que el emisor había llevado a cabo una

cantidad moderada de trabajo computacional, es decir una prueba de trabajo (proof-of-work).

Calcular la prueba tomaría unos segundos en una computadora regular, por lo tanto, no representa una dificultad para usuarios regulares. Sin embargo, para un spammer que desea enviar un millón de correos requeriría varias semanas, usando el hardware equivalente.

Es importante destacar que la instancia de la prueba de trabajo debe ser específica al correo electrónico, así como para el receptor del correo. De lo contrario, un spammer tendría la habilidad de enviar múltiples mensajes al mismo receptor (o el mismo mensaje a múltiples receptores) por el costo de un mensaje al receptor.

La segunda propiedad relevante es que la prueba de trabajo no debe representar una gran carga computacional en el receptor, debe ser trivial verificarlo independientemente de la dificultad para calcularlo.

En el ámbito académico, investigadores encontraron aplicaciones a la prueba de trabajo en soluciones para spam, prevenir ataques de negación de servicio, asegurando la integridad de analítica web y limitando la rata de adivinar claves en-línea.

El término proof-of-work fue utilizado por primera vez por los autores Markus Jakobsson y Ari Juels en un artículo en el año 1999 [9].

3.5.4.2. Prueba de participación (Proof of Stake)

Para validar las transacciones, un nodo o usuario debe tener un cierto porcentaje del valor total de la red. Proof of Stake podría brindar una mayor protección contra un ataque malicioso en la red al reducir los incentivos de ataque y hacer que sea muy costoso ejecutar ataques [1]. Se basan en la suposición de que quienes han invertido más tiempo y recursos en el sistema, están especialmente interesados en la supervivencia y el buen funcionamiento de la red que otorga valor a dicho sistema y por tanto son ellos los más indicados para cargar con la responsabilidad de proteger al sistema de posibles ataques. Es por eso que el protocolo los premia con una menor dificultad para encontrar bloques (es inversamente proporcional al tiempo y recursos invertidos).

3.5.4.3. Prueba del tiempo transcurrido (Proof of Elapsed Time)

Introducido por Intel, la prueba del tiempo transcurrido está diseñada para crear un modelo de consenso justo, que se centra principalmente en la eficiencia. Gira en torno a la distribución de elecciones de líderes en la mayor cantidad posible de participantes de la red. Además, el costo de controlar este proceso debería ser proporcional al valor obtenido de él. Esto significa que las personas necesitan invertir tiempo, y posiblemente dinero, si sienten que pueden dirigir el ecosistema en una dirección particular. Por último, pero no menos importante, todos los participantes deben verificar que el líder fue seleccionado de manera legítima [11]. Usa Trusted Execution Environment (TEE: es un área segura de un procesador principal. Garantiza que el código y los datos cargados en su interior estén protegidos con respecto a la confidencialidad y la

integridad), para proporcionar aleatoriedad y seguridad en el proceso de elección del líder a través de un tiempo de espera garantizado. Requiere el procesador Intel SGX (extensiones de protección de software) para proporcionar la garantía de seguridad [12].

3.5.4.4. Firma múltiple (Multi-signature)

La mayoría de los validadores (por ejemplo, tres de cinco) deben aceptar que una transacción es válida [1].

3.5.4.5. Tolerancia de falla bizantina práctica (Practical Byzantine Fault Tolerance)

Algoritmo diseñado para resolver disputas entre nodos informáticos (participantes de la red) cuando un nodo en un conjunto de nodos genera un resultado diferente de los demás en el conjunto [1].

3.6. Tipos de Blockchain

Basados en la evolución de blockchain a través de los últimos años, algunos autores han establecido la definición de varios tipos diferentes pero en algunos casos con atributos que se intersectan. Entre los tipos de blockchain definidos se pueden encontrar [12]:

6.3.4.7. 3.6.1. Blockchain público

Este tipo de cadena de bloques está abierta al público, no requiere permiso para participar y cualquier puede participar como un nodo en el proceso de tomar la decisión, los usuarios pueden o no ser recompensados por su participación.

Todos los usuarios del blockchain que no requiere permisos mantienen una copia del libro mayor en sus nodos locales y usa el mecanismo de consenso distribuido para alcanzar la decisión del eventual estado del libro mayor. Este tipo de blockchain se conoce también como no permisado.

6.3.4.8. 3.6.2. Blockchain privado

Como su nombre lo indica, este tipo de blockchains son privados y están abiertos únicamente a un consorcio o a grupos de individuos u organizaciones que han decidido compartir el libro mayor entre ellos.

6.3.4.9. 3.6.3. Blockchains semiprivado

En este tipo, parte del blockchain es privado y otra parte es pública. La parte pública está abierta para participación de cualquiera, mientras que la parte privada es controlada por un grupo de individuos o un consorcio.

6.3.4.10. 3.6.4. Libro mayor permisado

Un libro mayor permisado es un blockchain donde los participantes de la red son conocidos y son confiables. Libros mayores permisados no requieren el uso de un mecanismo de consenso, en vez de ello un protocolo de acuerdo puede ser usado para mantener la versión compartida del estado de los registros en el blockchain.

6.3.4.11. 3.6.5. Blockchain propietario y privado

Este tipo de blockchain quizás no tiene una aplicación relevante y surgió de la idea de descentralización usando la tecnología blockchain. Sin embargo, en ambientes privados específicos dentro de una organización y existe una necesidad de compartir datos y proveer un nivel de garantía de la autenticidad de los datos.

6.3.4.12. 3.6.6. Blockchain con tokens

Este tipo de blockchain es estándar como el de Bitcoin, que genera la criptomoneda como resultado de un proceso de consenso a través del minado o distribución inicial.

6.3.4.13. 3.6.7. Blockchain sin tokens

Este tipo de blockchain no usa una unidad básica de transferencia de valor, pero son valiosos en situaciones donde no es necesario transferir valor entre nodos y únicamente requiere compartir datos entre partes ya confiables.

3.7. Casos de uso de las cadenas de bloques

6.3.4.14. 3.7.1. Criptomonedas

El caso de uso que dio luz a la cadena de bloques es la criptomoneda Bitcoin. Las criptomonedas se definen como bienes digitales que pueden ser intercambiados entre distintos entes.

Dichos bienes están sustentados bajo principios criptográficos que aseguran el flujo transaccional que genera el intercambio de estas monedas digitales, además de controlar la creación de bienes adicionales con respecto a las mismas usando la tecnología cadena de bloques. A la fecha, existe una cantidad de criptomonedas que han ido logrando aceptación a nivel global y aumentando su valor.

6.3.4.15. 3.7.2. Servicios financieros

Las empresas generalmente requieren hacer compras a crédito dentro de su cadena de suministro.

Los pagos que se realizan pueden pasar por varios intermediarios antes de que el dinero pueda

llegar a su destino final, y por cada intermediario que forma parte de la cadena, éste gana un pequeño porcentaje.

Con la cadena de bloques se forma una línea de tiempo donde puede observarse la traza de los fondos a medida que pasan por los diferentes intermediarios, de manera que una disputa con respecto a una transacción puede ser fácilmente resuelta verificando las transacciones en los bloques.

Una cadena de bloques semi-privada puede implementarse donde sólo las empresas que forman parte de la red pueden observar el flujo de los fondos que van desde una organización a otra y auditarlas en caso de algún inconveniente, todo esto ligado también al hecho que pueden generarse contratos inteligentes entre las empresas que forman parte de la cadena de bloques para que las transacciones de realicen al ocurrir ciertos eventos.

6.3.4.16. 3.7.3. Servicios aseguradores

Las empresas aseguradoras requieren una plataforma que brinde velocidad para solicitudes, verifique pólizas, procese eventos cubiertos por dicha póliza y además, ofrezca una rápida respuesta a los usuarios con el fin de cumplir con sus estándares de servicio.

La tecnología de la cadena de bloques ofrece la capacidad de realizar un procesamiento de pólizas automático mediante la implementación de contratos inteligentes, de esta manera, cuando ocurre un evento asegurado, y este proviene de una fuente confiable, la política de seguro es automáticamente activada y realiza el pago según las tarifas especificadas en la relación contractual. Este enfoque procura una plataforma estable y confiable, y provee visibilidad para el usuario, reduciendo los reclamos.

6.3.4.17. 3.7.4. Cadena de suministro

Las cadenas de suministro de las empresas se han vuelto descentralizadas y geográficamente distribuidas, cada producto es un conjunto de partes, productores, procesos y distribución que requiere orquestación desde el diseño del producto hasta la entrega al cliente.

Mientras el número de organizaciones que participan en la cadena de valor aumentan, las deficiencias en integración de información disminuyen, lo cual puede afectar la logística, tiempo y costo en la operación de la cadena de suministro.

La cadena de bloques permite llevar el seguimiento de bienes en la cadena de suministro, lo cual es muy útil para detectar condiciones de transporte, piezas de origen dudoso o inclusive eliminar intermediarios. Esto tiene redunda en la reducción de costos y riesgo, así como un impacto positivo en la marca y calidad. Se espera también que la cadena de bloques reduzca la fricción de las empresas en establecer la cadena de suministro.

6.3.4.18. 3.7.5. Salud pública y privada

La industria médica tanto pública como privada, requiere sistemas más eficientes y descentralizados para mantener los registros de sus pacientes, pre-aprobar pagos realizados para intervenciones quirúrgicas y realizar compras de medicamentos vitales para su funcionamiento, la tecnología de la cadena de bloques puede proveer soluciones eficientes para cada uno de estos aspectos. Los registros médicos electrónicos generalmente se almacenan en sistemas centralizados y su acceso suele ser limitado a la misma red hospitalaria o compañías aseguradoras.

La cadena de bloques permite guardar de manera eficiente y privada la historia médica entera de un paciente con una granularidad personalizada a la que podrá acceder cualquier centro de salud que se encuentre en la red, ahorrando así recursos a la hora de que un paciente requiera ser atendido no importa el lugar donde se encuentre, de igual forma puede observarse la evolución de cierto tipo de enfermedades, e incluso actualizar contratos inteligentes para la compra de medicinas imprescindibles para la recuperación de los pacientes de manera inmediata.

6.3.4.19. 3.7.6. Servicios gubernamentales

Una considerable parte de las instituciones gubernamentales deben realizar o supervisar grandes cantidades de transacciones las cuales implican grandes volúmenes de fondos y podría ser necesario verificar su procedencia y destino.

La tecnología de la cadena de bloques puede ser utilizada para formar una red de entidades gubernamentales confiables, que pueden verificar todo tipo de transacciones y además, llevar un seguimiento acertado de todos los eventos que pueden emerger de manera imprevista, muchos ciudadanos pierden sus identificaciones a diario, la identidad de los ciudadanos puede fácilmente representarse en una cadena de bloques y en caso de que un ciudadano pierda su identificación, puede ubicarse rápidamente su identidad siguiendo su traza en la cadena de bloques obteniendo toda la información necesaria para comprobar que dicho ciudadano es quien dice ser.

Cada vez que un ciudadano compra una valor o una propiedad y necesita registrarla localmente o regionalmente, puede asociarse este nuevo valor a la cadena de bloques de dicho ciudadano, siempre pudiendo verificar el origen de los fondos con los que lo obtuvo, si realizó un cambio de nombre de alguna de sus empresas o si quiere agregar un socio a un contrato inteligente dentro de la misma cadena.

El mismo principio puede aplicarse para certificados de residencia, partidas de nacimiento, estudios realizados y títulos de propiedad, cada uno de estos valores con su marca de tiempo.

6.3.4.20. 3.7.7. Bienes raíces

La cadena de bloques puede apoyar la creación de nuevo modelos para conectar potenciales compradores y vendedores. La automatización de los procesos relevantes y la documentación, permiten reducir costos de inspección, comisiones, alícuotas e impuestos.

La cadena de bloques apoya la toma de decisiones, donde propietarios, inquilinos y proveedores pueden interactuar con la información de propiedad o historia transaccional, actualizar registros de forma descentralizada, transparente y segura, reduciendo el uso de papel.

6.3.4.21. 3.7.8. Internet de las cosas (IoT)

La tecnología de cadena de bloques apoyará el surgimiento del internet de las cosas y las redes descentralizadas, así como nuevos aspectos de cooperación entre dispositivos y monetización.

La habilidad para las aplicaciones IoT para escribir data transaccional a las redes descentralizadas basadas en blockchain, generará la necesidad de integración a través de las industrias como transporte y distribución, cadena de suministro, seguros que pueden tener visibilidad de los eventos.

El uso de dispositivos y nuestra actividad individual, ya sea por razones personales o aplicaciones industriales, requerirá nuevas redes seguras y descentralizadas para soportar la gran cantidad de data que producirá este nuevo entorno.

3.8. Aplicaciones distribuidas (DAPP)

David Johnston y su equipo de desarrollo presentan la siguiente estructura que define una DAPP y sus características [16]:

3.8.1. El nacimiento de las DAPPs

Una nueva aproximación está surgiendo para desarrollar aplicaciones altamente escalables y potentes. Bitcoin fue el pionero en esta nueva tendencia, con sus características relacionadas a código abierto, conectividad P2P (Red entre iguales o red entre pares, por sus siglas en ingles [17]), criptografía e integración con las cadenas de bloques. Un considerable número de aplicaciones está adaptando este esquema, siendo Ethereum una de las más notables, la cual implementa su propia cadena de bloques para operar en su plataforma.

3.8.2. Definición de DAPP

Para que una aplicación pueda considerarse DAPP, debe seguir el siguiente criterio:

- La aplicación debe ser en su totalidad código abierto, debe operar de manera autónoma y sin ningún tipo de control por parte de cualquier organización. La aplicación puede adaptarse a protocolos propuestos en respuesta a las necesidades dinámicas que se presentan a lo largo de su evolución, pero todos los cambios deben realizarse en consenso con sus usuarios.
- Los registros y data de la aplicación deben ser almacenados mediante el uso de criptografía, en una cadena de bloques pública y descentralizada para evitar puntos centrales de fallo.

- La aplicación debe hacer uso de un token criptográfico el cual es necesario para acceder a la aplicación y todas las contribuciones (a mineros) de recompensa deben ser asignadas en dichos tokens.

La aplicación debe generar tokens acorde a un algoritmo criptográfico estándar que actúa como una prueba de contribución de los nodos a la plataforma de la aplicación (ya sea prueba de trabajo u otro algoritmo).

3.8.3. Bitcoin como DAPP

Bitcoin ha demostrado ser una solución eficiente a los problemas que surgen al momento de implementar una red autónoma puerto a puerto utilizando su propia cadena de bloques. Lo más importante con respecto al tema de interés de la presente investigación, es que Bitcoin se considera una DAPP, por las siguientes razones:

La plataforma y software de Bitcoin es en su totalidad código abierto, no es controlada por ninguna entidad u organización y todos los datos y registros son públicos a toda el ecosistema.
Bitcoin genera sus tokens (bitcoins), con un algoritmo predeterminado que no puede ser cambiado, y dichos tokens son necesarios para el funcionamiento de Bitcoin. Los mineros de Bitcoin son recompensados con bitcoins por su trabajo asegurando la integridad de la red.
Todos los cambios en Bitcoin deben ser aprobados mayoritariamente en un consenso utilizando el mecanismo de Prueba de trabajo.

3.8.4. Clasificación de las DAPP’s

Existen varias características que definen la clasificación de una DAPP, la clasificación dependerá mayormente de si la aplicación utiliza su propia cadena de bloques o utiliza una cadena de bloques de otra DAPP para lograr sus funciones, en base a lo anterior:

Tipo I: Aplicaciones descentralizadas las cuales implementan su propia cadena de bloques, el ejemplo primordial de este tipo de DAPP es Bitcoin, también todas las criptomonedas nativas (Lite-coin, Binance-coin, etc.) participan en esta clasificación.

Tipo II: Aplicaciones descentralizadas que hacen uso de las cadenas de bloques de las aplicaciones Tipo I para lograr sus funciones. Son protocolos que tienen acceso a tokens para poder desenvolverse en el ecosistema de la aplicación Tipo I.

Tipo III: Aplicaciones descentralizadas que hacen uso de las cadenas de bloques de las aplicaciones Tipo II para lograr sus funciones. Al igual que las Tipo II, aplican protocolos que se desenvuelven en plataformas de aplicaciones de clasificación Tipo II.

Una analogía apropiada para entender las clasificaciones de DAPP's, es la siguiente, la clasificación Tipo I equivale a un sistema operativo tal como Windows, Linux, Mac, etc. Mientras que las Tipo II podrán equivaler a un procesador de texto, o sistema de sincronización de archivos, como Dropbox. Un ejemplo de una DAPP Tipo III sería un software especializado

tal como una aplicación que una varios correos electrónicos o una plataforma de blogs que se sincronice con algún editor de texto.

Utilizando la analogía anterior, es altamente probable que existan muy pocas aplicaciones de Tipo I, una cantidad superior de DAPP Tipo II y aún más del Tipo III.

3.9. Proyecto Hyperledger

El proyecto *Hyperledger* no es una implementación de la tecnología de cadena de bloques, es un proyecto iniciado por la Fundación Linux para apoyar el avance de esta tecnología y el liderazgo en el pensamiento. Es un proyecto paragua para comunidades de desarrolladores construyendo aplicaciones basadas en blockchain y tecnologías relacionadas.

Este proyecto fue anunciado formalmente a finales del año 2015 como un esfuerzo colaborativo de 17 miembros inicialmente para construir un framework de código abierto de la tecnología de cadena que puede ser usado para implementar aplicaciones y sistemas basados en blockchain entre industrias.

Al momento de esta investigación, cuenta con más de 130 miembros en el mundo, es el proyecto que crece más rápido en la historia de la Fundación Linux.

Entre los objetivos de la iniciativa es construir y ejecutar plataformas que soporten transacciones de negocios a nivel global, así como mejorar la confiabilidad y rendimiento de sistemas basados en la tecnología de cadena de bloques.

Los proyectos bajo el paragua de Hyperledger pasan por varias etapas de desarrollo, iniciando desde la propuesta, con una transición a la incubación y graduando a un estado activo. Los proyectos pueden ser deprecados o en un estado fin de proyecto, donde ya no son activamente desarrollados. Para que un proyecto pueda moverse a una etapa de incubación, debe tener una base de código funcional en una comunidad activa de desarrolladores.

La iniciativa Hyperledger curada por la Fundación Linux, provee herramientas, entrenamiento y eventos para escalar cualquier proyecto de código abierto y cuenta con la contribución de importantes empresas comerciales a nivel mundial.

El proyecto Hyperledger es financiado con varios miembros comerciales que apoyan la tecnología de cadena de bloques, algunos de estos miembros comerciales tenían esfuerzos internos o nacientes de código abierto que podían ser curados por la Fundación Linux.

Los principios establecidos por la Fundación Linux para la industria son los siguientes:

1. Gobernabilidad abierta y transparente de los procesos de desarrollo de software para aspectos de la tecnología de bloques.
2. Protección de la propiedad intelectual de los aspectos de software.
3. Los casos de uso claves están guiados por modelos del consorcio.

¿Qué significa estar bajo el paragua de Hyperledger? La Fundación de Software Apache,

después de 20 años de existencia, es una comunidad de comunidades, hay aproximadamente más de 300 proyectos de software de alto nivel de código abierto.

Cada proyecto tiene su coordinación, comunidad de desarrolladores, mapa de evolución, proceso de desarrollo, entre otros. La mayoría usa las mismas herramientas de colaboración, los mismos procesos de reportes al grupo de dirección y deben demostrar la misma calidad de desarrollo orientado a la comunidad y evaluado por la incubadora de la Fundación Apache Software.

Basados en los antecedentes de la Fundación Apache Software, la comunidad del proyecto Hyperledger define que un proyecto Hyperledger consiste de:

1. Un conjunto identificado de desarrolladores de software que mantienen el proyecto y son responsables del proceso de desarrollo, cultura y dirección técnica general del proyecto, así como la interacción con el público. Los desarrolladores pueden votar para incluir nuevos miembros y otros miembros pueden retirarse, pero cualquier proyecto activo requiere al menos un pequeño conjunto de desarrolladores que mantengan la actividad del proyecto.
2. Un conjunto determinado de artefactos, incluyendo uno o más repositorios Git, una base de datos para seguimiento de errores, un wiki, un conjunto de listas de correo y otros recursos para mantener los desarrolladores (foros, listas de correo, canales IRC/Slack, etc.) unidos como parte de un proyecto particular; donde los que mantienen el proyecto y la comunidad son responsables de mantenerlo actualizado y activo.
3. Espacio dedicado para describir el proyecto y la comunidad con una clara definición de la relación y las relaciones con otros esfuerzos en Hyperledger, y una indicación donde cada esfuerzo es igual de importante.
4. El proyecto Hyperledger espera hacer un impacto positivo en la comunidad de la tecnología de cadena de bloque, incrementar el tráfico y la habilidad de servir en diferentes lugares del mundo como estímulo para la comunidad de esta tecnología [13].

3.9.1. Hyperledger como protocolo

El proyecto Hyperledger tiene como objetivo ser una plataforma para aplicaciones basadas en blockchain que es guiada por casos de uso de la industria.

Así como ha habido una serie de contribuciones al proyecto Hyperledger por parte de la comunidad, la plataforma de blockchain de Hyperledger está evolucionando a convertirse en un protocolo para transacciones de negocios.

El proyecto Hyperledger también está evolucionando como una especificación que puede ser usada como referencia para construir plataformas de blockchain, en vez de una solución blockchain para un solo tipo de industria o requerimiento [12].

3.9.1.1. Arquitectura de referencia

El proyecto Hyperledger ha publicado un artículo con una arquitectura de referencia que sirve como guía para construir blockchain permisados. La arquitectura de referencia consiste de dos componentes principales: a) Servicios Hyperledger y b) APIs, SDKs y CLIs.

La figura 6 muestra la versión 2.0.0 de la arquitectura de referencia de Hyperledger publicada en el artículo. El proyecto Hyperledger está evolucionando y puede cambiar rápidamente, por lo que se espera que la arquitectura de referencia mostrada pueda ser modificada.

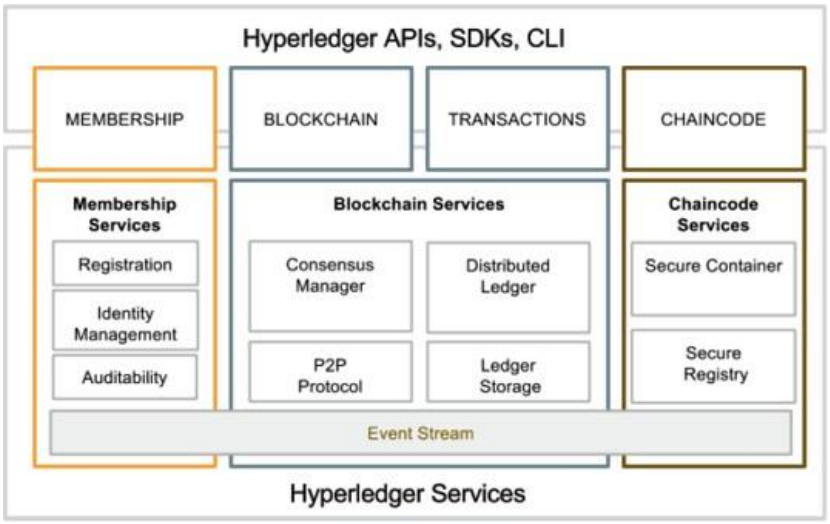

Figura 3-5: Arquitectura de referencia del proyecto Hyperledger. Recuperada de: Hyperledger white paper**.**

Los servicios Hyperledger incluyen servicios de identidad, políticas, blockchain, transacciones y contratos inteligentes. Los APIs, SDKs y CLIs proveen una interfaz para servicios de blockchain a través de interfaces de programación, componentes para desarrollo e interfaces de línea de comando.

Además, un stream de eventos (Event Stream), lo cual es básicamente un canal RPC que escucha en todos los servicios y permite invocar servicios distribuidos, recibir y enviar eventos. Los eventos pueden ser predefinidos o a la medida. Validar nodos o una aplicación basada en blockchain puede emitir eventos a los cuales aplicaciones externas pueden escuchar o responder.

Las aplicaciones basadas en blockchain y que implementan lógica de negocio para interactuar con la cadena de bloques, se denominan *Chaincode* [12].

3.9.1.2. Requerimientos del proyecto Hyperledger

Existen varios requerimientos de un servicio blockchain. La arquitectura de referencia es guiada por las necesidades y requerimientos identificados por los participantes del proyecto Hyperledger y luego de estudiar los casos de uso de las industrias.

Hay varias categorías de requerimientos que han sido deducidas al estudiar los casos de uso industriales y describimos a continuación [12].

3.9.1.3. Enfoque modular

El requerimiento principal de Hyperledger es su estructura modular. Se espera que la cadena de bloques al ser una tecnología que brinda interoperabilidad entre las industrias, será usada en muchos escenarios de negocios.

Por lo tanto, las funciones relacionadas con almacenamiento, políticas, aplicaciones de negocios (chaincode), control de acceso, consenso y muchos otros servicios deben ser integrables. Los módulos deben permitir que los usuarios puedan integrar o remover con facilidad para cumplir con los requerimientos del negocio.

Por ejemplo, si un blockchain de negocios requiere funcionar únicamente entre partes confiables y lleva a cabo operaciones de negocios muy básicas, quizás no hay necesidad de tener soporte criptográfico avanzado para confidencialidad y privacidad, por lo tanto los usuarios pueden remover la funcionalidad (módulo) y reemplazarlo con un módulo más apropiado para sus necesidades [12].

3.9.1.4. Privacidad y confidencialidad

La privacidad y confidencialidad de transacciones y contratos es de alta importancia para blockchains de negocios. Como tal, la visión de Hyperledger es proveer un amplio rango de protocolos criptográficos y algoritmos y se espera que los usuarios puedan escoger los módulos apropiados para sus requerimientos de negocios [12].

3.9.1.5. Identidad

Para proveer servicios de privacidad y confidencialidad, se requiere un modelo PKI flexible que puede ser usado para manejar el control de acceso a las funcionalidades. También se espera que La fortaleza y tipo de mecanismos criptográficos pueda variar de acuerdo a las necesidades y requerimientos de los usuarios. En algunos escenarios puede que se requiera que el usuario oculte su identidad, por lo que Hyperledger debe proveer esta funcionalidad [12].

3.9.1.6. Auditabilidad

Se espera que se mantenga un registro auditable e inmutable de todas las identidades y sus operaciones relacionadas [12].

3.9.1.7. Interoperabilidad

Actualmente hay varias soluciones de blockchain disponibles, pero no pueden comunicarse o integrarse entre ellas y esto puede ser un factor en el crecimiento del ecosistema global de negocios basado en blockchain.

Se espera que muchas redes basadas en blockchain operen en el mundo de negocios para necesidades específicas, pero es importante que sean capaces de comunicarse entre ellas.

Debe haber un conjunto común de estándares que las redes basadas en blockchain puedan seguir y permita integrarse y comunicarse entre diferentes cadenas de bloques [12].

3.9.1.8. Portabilidad

El requerimiento de portabilidad está relacionado con la habilidad para ejecutar aplicaciones a través de múltiples plataformas y ambientes sin necesidad de realizar cambios a nivel de código. El diseño de Hyperledger permite que sea portable, no únicamente a nivel de infraestructura sino también a nivel de código, librerías y APIs para que pueda soportar desarrollo uniforme a través de varias implementaciones de Hyperledger [12].

3.9.2. Plataformas del Proyecto Hyperledger

Hyperledger incuba y promueve una gama de tecnologías de cadena de bloques de negocios, que incluyen frameworks de libros de contabilidad distribuidos, motores de contratos inteligentes, bibliotecas de clientes, interfaces gráficas, bibliotecas de utilidades y aplicaciones de muestra. La estrategia general de Hyperledger fomenta la reutilización de bloques de construcción comunes y permite una rápida innovación de los componentes de DLT.

Los proyectos dentro de Hyperledger se clasifican en dos: frameworks y herramientas.

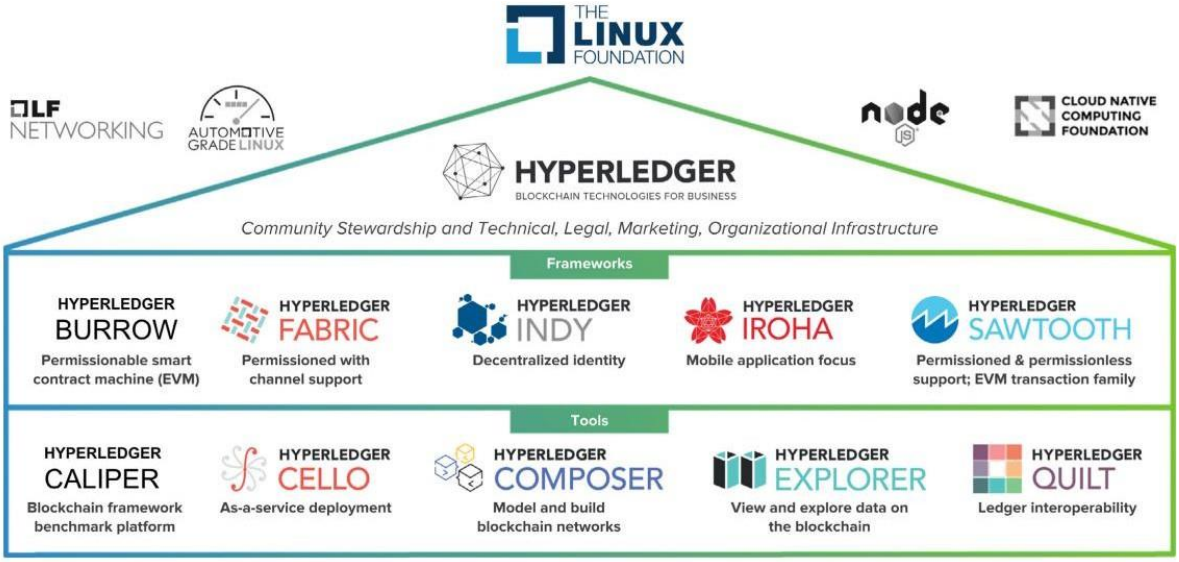

Figura 3-6: La casa verde de Hyperledger, frameworks y herramientas alojadas por Hyperledger. Recuperada de: https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf

3.9.3. Hyperledger Composer

Composer es un conjunto de herramientas y un marco de desarrollo de código abierto que acelera el tiempo que lleva escribir una aplicación de blockchain. En lugar de desarrollar

contratos inteligentes desde cero, Composer proporciona una capa de conveniencia y abstracciones a nivel de negocios para implementar contratos inteligentes en Fabric. Composer también le facilita la conexión a la red de negocio desde una aplicación web o móvil.

Ayuda a definir la red de negocios utilizando un lenguaje de modelado personalizado y un pequeño conjunto de primitivas, y luego implementa la red de negocios en Hyperledger Fabric. La red de negocios está formada por participantes, activos y las transacciones. También se incluyen listas de control de acceso que restringen el acceso a transacciones o datos [15].Construido con JavaScript, aprovechando herramientas modernas como node.js, npm, CLI y editores populares, Composer ofrece abstracciones centradas en el negocio, así como aplicaciones de muestra con procesos devops fáciles de probar para crear soluciones de cadena de bloques robustas que impulsan la alineación entre los requisitos del negocio y el desarrollo técnico [14].

3.9.3.1. Definición de red de negocios

El concepto clave para Hyperledger Composer es la definición de red de negocios (BND). Define el modelo de datos, la lógica de transacción y las reglas de control de acceso para la solución blockchain. La definición de la red se compone de los siguientes archivos [18]:

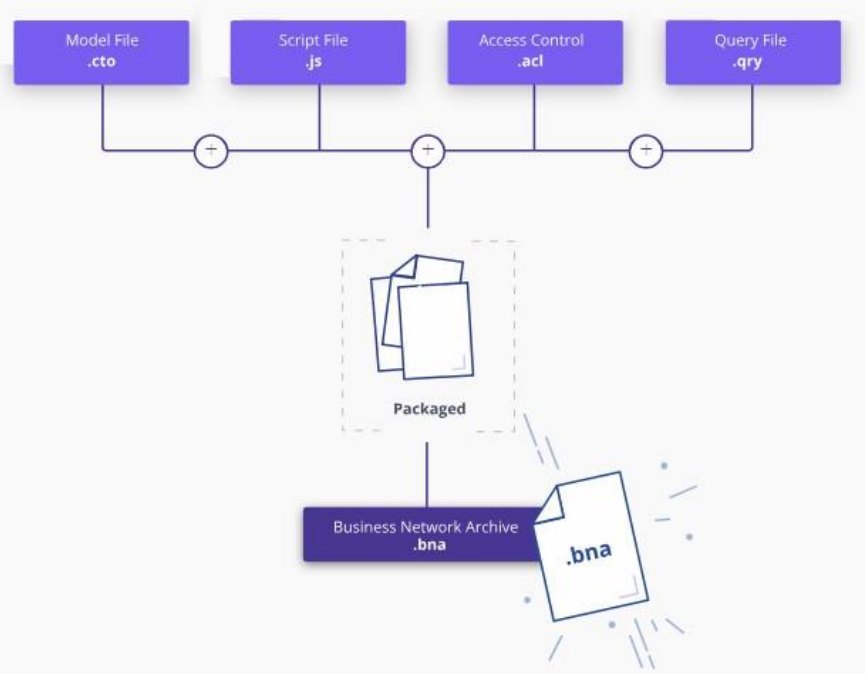

Figura 3-7: Gráfica de la composición de la definición de la red. Recuperada de: https://hyperledger.github.io/composer/latest/introduction/introduction

- **Model File:** este archivo .cto modela los activos, participantes y transacciones para la aplicación blockchain.
- **Script File:** funciones que procesan las transacciones. Estas funciones están escritas en JavaScript y representan los contratos inteligentes.

- **Access Control:** este archivo describe las reglas que rigen qué participantes de la red empresarial pueden trabajar con qué partes de la cadena de bloques, en pocas palabras manejo de permisos.
- **Query File:** Las consultas se utilizan para devolver datos sobre el blockchain world-state. Las consultas se definen dentro de una red empresarial y pueden incluir parámetros variables para una personalización simple. Las consultas se envían utilizando la API de Hyperledger Composer.

3.9.3.2. Arquitectura típica de solución de Hyperledger Composer

Hyperledger Composer utiliza el marco generador de código **Yeoman** de código abierto para crear esqueletos de proyectos:

- Aplicación web angular.
- Aplicación Node.js
- Esqueleto de Red de Negocios.

El **SDK de JavaScript de Hyperledger Composer** es un conjunto de API Node.js que permite a los desarrolladores crear aplicaciones para administrar e interactuar con redes empresariales desplegadas.

El servidor **REST Hyperledger Composer** genera automáticamente una Open API (Swagger) REST API para una red empresarial. El servidor REST (basado en la tecnología LoopBack) convierte el modelo Composer para una red empresarial en una definición Open API, y en tiempo de ejecución implementa el soporte crear, leer, actualizar y eliminar para activos y participantes y permite que las transacciones se envíen para su procesamiento o recuperación [19].

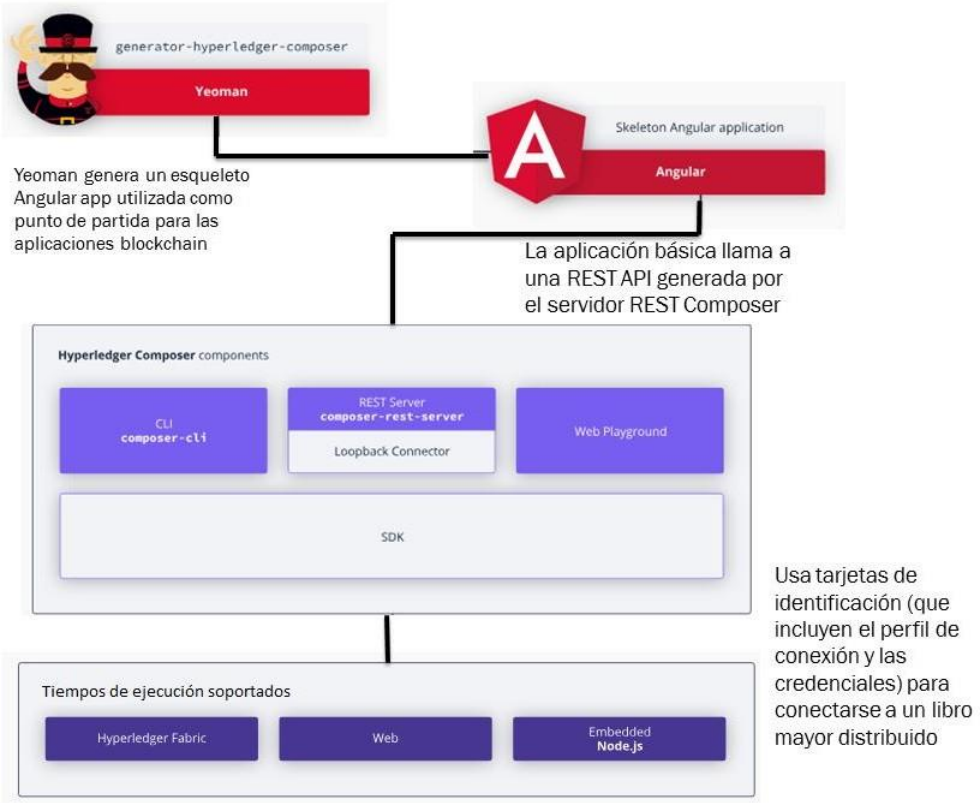

Figura 3-8: Arquitectura típica de una solución con Hyperledger Composer. Recuperada y traducida de: https://hyperledger.github.io/composer/latest/introduction/solution-architecture

3.9.4. Hyperledger Fabric

Fabric es la contribución original realizada por IBM al proyecto Hyperledger. El objetivo de esta contribución es habilitar un enfoque abierto, modular y flexible para construir redes blockchain.

Varias funciones en Fabric son integrables y permite el uso de cualquier lenguaje para desarrollar contratos inteligentes, esto es posible porque está basado en tecnología de contenedor que puede albergar cualquier lenguaje [20].

Los contratos inteligentes (denominados chaincode en Fabric) son ejecutados en un contenedor seguro que incluye un sistema operativo seguro, ambiente de ejecución, lenguaje de soporte del chaincode y APIs de desarrollo para los lenguajes Go, Java y Node.js, otros lenguajes pueden ser soportados de ser requeridos [21]. Esto significa que la mayoría de las empresas ya tienen el conjunto de habilidades necesario para desarrollar contratos inteligentes y no se necesita capacitación adicional para aprender un nuevo idioma o DSL [22].

Esto es una característica muy poderosa comparada con el lenguaje específico del dominio (Solidity) en Ethereum o el muy limitado lenguaje de scripting en Bitcoin.

Fabric es una red permisada que tiene como objetivo mejorar aspectos tales como escalabilidad, privacidad y confidencialidad. La idea clave es la tecnología modular, que permite

flexibilidad en diseño e implementación, esto puede resultar en alcanzar los objetivos descritos.

Las transacciones en Fabric son privadas, confidenciales y anónimas para los usuarios en general, pero pueden trazadas y enlazadas a los usuarios por auditores autorizados. Al ser una red permisada, todos los participantes deben registrarse con los servicios de identidad y membresía para acceder a la red blockchain, también provee funcionalidad de auditoria para cumplir con requerimientos regulatorios y compatibilidad [20].

Uno de los diferenciadores más importantes de la plataforma es su compatibilidad con protocolos de consenso conectables que permiten que la plataforma se personalice de manera más efectiva para adaptarse a casos de uso particulares y modelos de confianza. Por ejemplo, cuando se implementa dentro de una sola empresa, o es operado por una autoridad confiable, el consenso tolerante a fallas totalmente bizantino podría considerarse innecesario y un arrastre excesivo en el rendimiento. En situaciones como esta, un protocolo de consenso tolerante a fallos de bloqueo (CFT) podría ser más que adecuado, mientras que, en un caso de uso descentralizado y multipartidista, podría ser necesario un protocolo de consenso tolerante a fallos bizantinos (BFT) más tradicional [22].

3.9.4.1. Arquitectura Hyperledger Fabric

Una característica importante de la arquitectura Hyperledger Fabric son los canales; dado que una empresa puede formar parte de diferentes redes comerciales, es necesario utilizar canales seguros e independientes en los que pueda fluir la información. Cada canal es también el espacio protegido donde las credenciales y autorizaciones son válidas, limitadas a esa red comercial. Se puede pensar en los canales como grupos de Whatsapp separados en los que se puede hablar con algunas personas pero no con otras. Un participante también puede ser un endosante en una red de negocios, un cliente en otra, un ordenador en otra [26].

Está lógicamente organizado en tres categorías principales basadas en el tipo de servicio provisto: servicios de membresía, servicios blockchain y servicios chaincode, siguiendo la definición del Proyecto Hyperledger.

3.9.4.1.1. Servicios de membresía

Estos servicios son usados para proveer capacidad de control de acceso para los usuarios de la red Fabric. Las funciones de los servicios de membresía son:

- Registro de usuario.
- Validación de la identidad del usuario.
- Asignar permisos apropiados para los usuarios dependiendo de sus roles.

Los servicios de membresía usan la infraestructura de llave pública (PKI) para soportar las operaciones de gestión de identidad y autorización.

3.9.4.1.2. Servicios blockchain

Los servicios blockchain están en el núcleo de Hyperledger Fabric. Componentes en esta

categoría son los siguientes:

1 Gestor de consenso es responsable por proveer la interfaz al algoritmo de consenso, esto sirve como un adaptador que recibe una transacción de otras entidades Hyperledger y la ejecuta bajo el criterio acordado por el tipo de algoritmo seleccionado. El consenso es integrable y actualmente hay tres tipos de algoritmos de consenso disponibles en Fabric.
2 Libro mayor distribuido soportado por la cadena de bloques es almacenado en una base de datos de tipo llave/valor. Esta base de datos es usada por los contratos inteligentes para almacenar los estados relevantes durante la ejecución de las transacciones.
3 Protocolo P2P en Hyperledger Fabric es construido usando Google RPC (gRPC), que usa buffers del protocolo para definir la estructura de los mensajes. Los mensajes son pasados entre los nodos para llevar a cabo varias funciones. Hay cuatro tipos principales de mensajes en Hyperledger Fabric: descubrimiento, transacción, sincronización y consenso.

3.9.4.1.3. Servicios chaincode

Estos servicios permiten la creación de contenedores seguros que son usados para ejecutar el chaincode. Los componentes en esta categoría son los siguientes:

- Contenedor seguro: el chaincode es desplegado en contenedores Docker que proveen una ambiente restringido para ejecución del contrato inteligente.
- Registro seguro provee un registro de todas las imágenes que contienen contratos inteligentes.

3.9.4.2. Nodos en Hyperledger Fabric

Los nodos son las entidades de comunicación de la cadena de bloques. Un "nodo" es solo una función lógica en el sentido de que múltiples nodos de diferentes tipos pueden ejecutarse en el mismo servidor físico. Lo que cuenta es cómo los nodos se agrupan en "dominios de confianza" y se asocian a entidades lógicas que los controlan. Básicamente hay tres tipos de nodos [23]:

- Peers, nodos que mantienen copia de la cadena de bloques y pueden validar transacciones (endosantes). Propiedad de las organizaciones, organizan el libro mayor y los contratos inteligentes y conforman la estructura física de una red Fabric.
- Ordenadores, estos nodos forman el servicio de pedido, tratan con las comunicaciones proporcionando garantías de entrega. El servicio de pedidos proporciona un canal de comunicación compartido a clientes y peers, ofreciendo un servicio de difusión para mensajes que contienen transacciones.
- Clientes, que envían las propuestas para que las transacciones sean aprobadas (por los endosantes). Los clientes no poseen una copia de la blockchain. El cliente representa la entidad que actúa en nombre de un usuario final. Debe conectarse a un peer para comunicarse con la cadena de bloques. El cliente puede conectarse a cualquier peer de su elección.

3.9.4.3. Flujo de las transacciones

Este flujo asume que un canal está configurado y funcionando. El usuario de la aplicación se ha registrado e inscrito en la autoridad de certificación (CA) de la organización y ha recibido el material criptográfico necesario, que se utiliza para autenticarse en la red [24].

Fabric introduce una nueva arquitectura para transacciones que llamamos execute-order-validate (ejecutar–ordenar-validar). Aborda los desafíos de resiliencia, flexibilidad, escalabilidad, rendimiento y confidencialidad que enfrenta el modelo de orden de ejecución al separar el flujo de transacciones en tres pasos [22]:

> ***Ejecutar*** una transacción, los ***endorsing peers*** (endosantes) verifican (1) que la propuesta de transacción está bien formada, (2) no se ha enviado ya en el pasado (protección contra ataques de repetición), (3) la firma es válida (usando MSP) y (4) que el remitente (el Cliente A, por ejemplo) está debidamente autorizado para realizar la operación propuesta en ese. Los peers que respaldan toman las entradas de la propuesta de transacción como argumentos de la función del código de cadena invocado. Luego, el código de cadena se ejecuta en la base de datos del estado actual para producir resultados de transacción que incluyen un valor de respuesta, un conjunto de lectura y un conjunto de escritura. No se realizan actualizaciones en el libro mayor en este momento. El conjunto de estos valores, junto con la firma del peer que avala, se devuelve como una "respuesta de propuesta" al SDK que analiza la carga útil para que la aplicación consuma.
>
> La aplicación verifica las firmas de los peers que respaldan y compara las respuestas de la propuesta para determinar si las respuestas de la propuesta son las mismas. Si el código de la cadena solo consultó el libro mayor, la aplicación inspeccionaría la respuesta de la consulta y, por lo general, no enviaría la transacción al Servicio de pedidos (***Ordering Service***). Si la aplicación cliente pretende enviar la transacción al Servicio de pedidos para actualizar el libro mayor, la aplicación determina si la política de consenso especificada se ha cumplido antes de enviar. La arquitectura es tal que, incluso si una aplicación decide no inspeccionar las respuestas o reenvía una transacción sin aprobación, la política de aprobación todavía será aplicada por sus pares y se mantendrá en la fase de validación de confirmación.
>
> ***Ordenar*** transacciones, el Servicio de Pedidos recibe transacciones de todos los canales en la red, las ordena cronológicamente por canal y crea bloques de transacciones por canal.
>
> ***Validar***, los bloques de transacciones se “entregan” a todos los peers en el canal. Las transacciones dentro del bloque se validan para garantizar que se cumpla la política de consenso y para garantizar que no se hayan producido cambios en el estado del libro mayor para las variables del conjunto de lectura desde que la ejecución de la transacción generó el conjunto de lectura. Las transacciones en el bloque se etiquetan como válidas o no válidas. Cada peer agrega el bloque a la cadena del canal y, para cada transacción válida, los

conjuntos de escritura se confirman en la base de datos del estado actual. Se emite un evento para notificar a la aplicación cliente que la transacción (invocación) se ha agregado de manera inmutable a la cadena, así como una notificación de si la transacción fue validada o no.

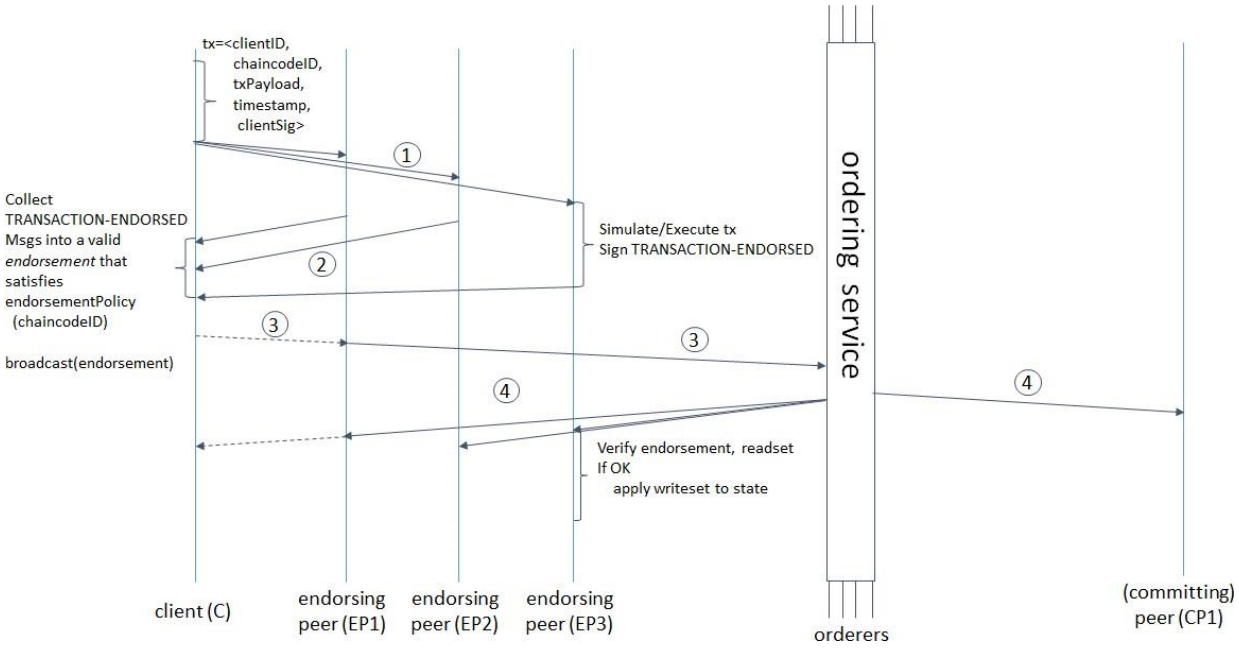

Figura 3-9: Ilustración de un posible flujo de transacciones (ruta de un caso común). Recuperada de: https://hyperledger-fabric.readthedocs.io/en/release-1.4/arch-deep-dive.html

3.9.5. Contratos Inteligentes (Smart Contracts)

De todos los datos que pueden guardarse en una cadena de bloques, los contratos inteligentes vienen siendo una de las adiciones más atractivas para la mayoría de los casos de uso. Un contrato inteligente, haciendo una analogía con su contraparte física, es un acuerdo automatizado y ejecutable.

Son una serie de promesas que son especificadas de manera digital, que incluye los protocolos mediante los cuales las partes involucradas deben realizar dichas promesas. Mientras que es automatizado por algún elemento computacional, puede o no requerir la intervención de control humano. Dicho contrato representa una relación entre dos o más entes que se realiza bajo ciertas condiciones, la ventaja en el uso de contratos inteligentes, radica en que están enteramente supervisados tanto por entidades legales que los construyen, así como cuerpos computacionales que los ejecutan, haciéndolos a prueba de fraudes [24].

En Hyperledger Fabric un contrato inteligente se define dentro de un *chaincode*. Se pueden definir varios contratos inteligentes dentro del mismo *chaincode*. Cuando se implementa un *chaincode*, todos los contratos inteligentes dentro de él se ponen a disposición de las aplicaciones. Cada contrato inteligente tiene una política de aprobación asociada con él. Esta política de aprobación identifica qué organizaciones deben aprobar las transacciones generadas

por el contrato inteligente antes de que esas transacciones puedan identificarse como válidas [25].

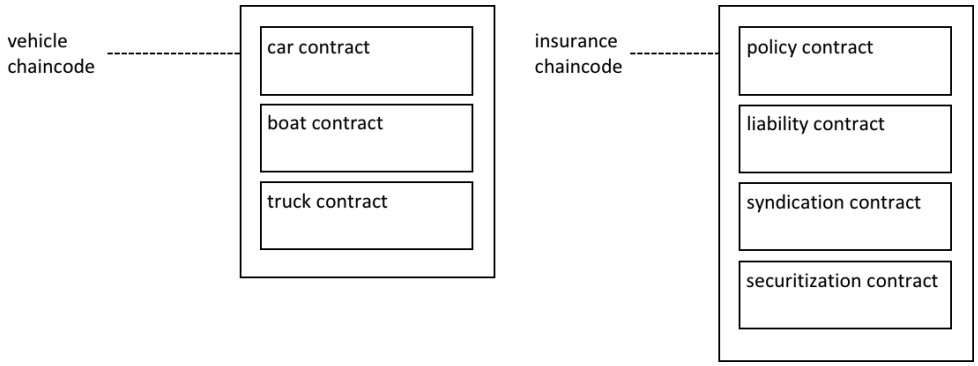

Figura 3-10: Ejemplo de chaincodes en Fabric. Recuperado de: https://hyperledger-fabric.readthedocs.io/en/release-1.4/smartcontract/smartcontract.html

3.9.5.1. Contratos Inteligentes: Casos de uso

La Cámara Digital de Comercio ya se encuentra trabajando en el uso de contratos inteligentes, la misma, comenta lo siguiente [27]: "Los contratos inteligentes permiten a individuos poseer y controlar su identidad digital, conteniendo reputación, datos y bienes digitales. Esto permite a los individuos decidir qué datos compartir a sus contrapartes, dando a las empresas la oportunidad de utilizar únicamente los datos necesarios".

Tomando en cuenta lo mencionado anteriormente, algunos de los casos de uso de los contratos inteligentes más innovadores son:

3.9.5.1.1. Contratos inteligentes para Seguridad Financiera

Los contratos inteligentes pueden manejar automáticamente el pago de dividendos, divisiones de acciones y gestión de pasivos, mientras que reduce los riesgos operacionales de confiar operaciones a terceros. Entre las bondades que brindan podemos nombrar:

- Digitalizar flujos de trabajo debido a la confianza del almacenamiento en una cadena de bloques.
- Reducir el riesgo de intromisión de terceros en manejo de capital.
- Ciclos de pagos asegurados gracias a las marcas de tiempo.

3.9.5.1.2. Contratos inteligentes para Procesos Derivados

Los procesos de post-negociación, pueden ser manejados vía contratos inteligentes, eliminando procesos duplicativos de recursos en ambas entidades luego de realizar un acuerdo comercial. Basta con observar los beneficios:

- Asignación de obligaciones de manera automática, mientras que se ejecutan eventos repetitivos una vez terminada una negociación (por ejemplo, ciclos de pago periódicos).
- Procesamiento de eventos externos o de sucesión (aprobación de un crédito, por

ejemplo).

- Permite la evolución de movimientos financieros en tiempo real, el cual aumenta el nivel de seguridad y confianza, mientras que minimiza errores y disputas.

3.9.5.1.3. Contratos inteligentes para la Captación de Datos

Organizaciones financieras pueden hacer uso de contratos inteligentes para la recopilación de datos de manera eficaz, eficiente y transparente. Reducción de procesos de auditoría, datos uniformes entre organizaciones y máximos niveles de trasparencia son algunas de las bondades a explotar, algunas otras se enumeran como:

- Integridad de los datos y transparencia, lo cual genera un mercado con más confianza y estabilidad.
- Al ser almacenados en una cadena de bloques, los datos son almacenados en estructuras accesibles para todos los entes que vayan a utilizar estos datos.

3.9.5.1.4. Contratos inteligentes para Hipotecas

Los contratos inteligentes pueden automatizar los procesos manuales y tediosos que ocurren tras un contrato hipotecario. En este caso el contrato inteligente brinda una conexión entre todas las entidades relacionadas con la hipoteca, lo cual puede generar un proceso a prueba de errores que sea confiable y transparente. Entre otras ventajas se observan:

- Liberación de contratos a compradores una vez que haya pagado la deuda hipotecaria de manera automática.
- Costos y errores minimizados debido a la eliminación de procesos manuales.
- Verificación de pagos y seguimiento de crédito tanto al comprador como al arrendador.

3.9.5.1.5. Contratos inteligentes en el Ámbito de Seguros

Actualmente, el proceso de asegurar cualquier bien, está sujeto a muchos procesos manuales y tediosos, los cuales pueden agilizarse de una manera óptima mediante el uso de contratos inteligentes. El contrato inteligente, puede guardar los registros de cada póliza, riesgos cubiertos y, combinado con herramientas del tipo Internet de las cosas, puede emitirse una señal de manera automática cuando ocurre algún tipo de accidente, además ofrece:

- Almacenamiento de historia de un asegurado, conteniendo bienes, pólizas, accidentes cubiertos y tipo de bien asegurado.
- Ahorro de recursos debido a la automatización de procesos de verificación.

3.9.5.1.6. Contratos inteligentes en el Ámbito Médico

El manejo del historial de un paciente, el cual puede ser almacenado en una cadena de bloques y ser leído por cualquier institución médica con acceso al contrato inteligente para realizar sus pagos de manera automática es un aditivo atractivo a todos los hospitales u organizaciones

médicas del planeta, adicionalmente:

- Reducción de costos debido al ahorro de recursos invertidos en instituciones médicas en mantener el historial de un paciente.
- Acceso a datos íntegros de pacientes los cuales al ser almacenados en una cadena de bloques se vuelven confiables, transparentes y seguros.
- Al estar en la cadena de bloques del paciente, su privacidad aumenta de manera considerable.

3.9.5.1.7. Contratos inteligentes junto a Internet de las cosas

Probablemente uno de los casos más atractivos. La conexión que puede establecerse entre un dispositivo IOT y un contrato inteligente puede disparar alertas, pagos de consumo y monitoreo en tiempo real de dispositivos de una manera segura, confiable y transparente. Entre otras características, también puede evaluarse:

- Integridad en los datos de consumo, asegurando siempre un registro actualizado en caso de que se haya presentado una falla en algún dispositivo IOT conectado al contrato inteligente.
- Previsión de diferentes escenarios que podrían presentarse dependiente del dispositivo, siempre que esté contemplado en contrato inteligente (por ejemplo, qué hacer si un hogar está produciendo exceso de electricidad).
- Un historial confiable, transparente y seguro de las actividades de diferentes dispositivos pueden quedar almacenadas mediante la vigilancia del contrato inteligente.

Capítulo 4 – Tecnologías utilizadas

4.1. Lenguajes de programación

4.1.1. JavaScript

Según la fundación Mozilla [44], JavaScript es un lenguaje ligero, interpretado, orientado a objetos con funciones de primera clase, más conocido como el lenguaje de script de páginas web, pero también, utilizado en muchos entornos tales como NodeJS.

4.1.2. Python

Python es un lenguaje de programación interpretado. Cuenta con estructuras de datos de alto nivel eficientes y un enfoque simple pero efectivo para la programación orientada a objetos. La elegante sintaxis y escritura dinámica de Python, junto con su naturaleza interpretada, lo convierten en un lenguaje ideal para la creación de scripts y el rápido desarrollo de aplicaciones en muchas áreas en la mayoría de las plataformas.

4.1.3. Lenguaje de modelado de Hyperledger Composer

Es un lenguaje de modelado orientado a objetos que se utiliza para definir el modelo de dominio para una definición de red empresarial. La extensión de los archivos de este tipo es .cto. Un archivo CTO de Hyperledger Composer se compone de los siguientes elementos [46]:

- Un solo espacio de nombres. Todas las declaraciones de recursos dentro del archivo están implícitamente en este espacio de nombres.
- Un conjunto de definiciones de recursos, que abarca activos, transacciones, participantes y eventos.
- Declaraciones de importación opcionales que importan recursos de otros espacios de nombres.

4.1.4. Lenguaje de control de acceso Hyperledger Composer

Hyperledger Composer incluye un lenguaje de control de acceso (ACL) que proporciona control de acceso declarativo sobre los elementos del modelo de dominio. Al definir las reglas de ACL, puede determinar qué usuarios / roles pueden crear, leer, actualizar o eliminar elementos en el modelo de dominio de una red empresarial [47].

4.2. Hyperledger Composer Playground

Hyperledger Composer Playground se utiliza para crear y probar redes de negocios, y para crear y cambiar entre identidades. Se puede utilizar para interactuar con un Blockchain real o en modo web, donde el usuario interactúa con un blockchain simulado [48].

4.3. Django

Django es un framework de desarrollo web de alto nivel escrito en Python, el cual se inventó para cumplir con los plazos de las salas de redacción de forma rápida, al tiempo que satisfacía los exigentes requisitos de los desarrolladores web experimentados. [53].

4.4. NodeJS

Es un entorno de ejecución para JavaScript, está diseñado para construir aplicaciones de red escalables. Fue desarrollado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web 2.0. Fue creado por Ryan Dahl en 2009.

4.4.1. Mocha

Mocha es un framework de NodeJS el cual permite realizar pruebas asíncronas de manera rápida y segura [45].

4.4.2. Loopback

LoopBack es un framework para crear APIs de forma avanzada y conectarlas con fuentes de datos backend, proporcionando una gran capa de abstracción a la hora de trabajar con bases de datos. Construido sobre Express (Node.js), puede tomar una definición de modelo de datos y generar fácilmente una API REST de extremo a extremo totalmente funcional que puede ser llamada por cualquier cliente [49].

4.5. Cucumber

Cucumber es una herramienta que admite el desarrollo guiado por el comportamiento (Behaviour-Driven Development en inglés, BDD).

Cucumber lee especificaciones ejecutables escritas en texto plano y valida que el software haga lo que dichas especificaciones dicen [51]. Esta herramienta permite realizar pruebas de la definición de la red.

4.6. Karma

Karma es esencialmente una herramienta que genera un servidor web que ejecuta código fuente contra código de prueba para cada uno de los navegadores conectados. Los resultados de cada prueba en cada navegador se examinan y se muestran mediante la línea de comandos al desarrollador para que puedan ver qué exploradores y pruebas pasaron o fallaron [52].

4.7. Docker

Docker proporciona una manera de ejecutar aplicaciones aisladas de forma segura en un

contenedor, empaquetadas con todas sus dependencias y bibliotecas [53]. Haciendo que las aplicaciones software sean portables ya que pueden ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

4.8. Yeoman

Yeoman es un sistema de andamios genérico que permite la creación de cualquier tipo de aplicación. Permite iniciar rápidamente nuevos proyectos y agiliza el mantenimiento de proyectos existentes.

Yeoman es un lenguaje agnóstico. Puede generar proyectos en cualquier idioma (Web, Java, Python, C #, etc.)

Yeoman por sí mismo no toma ninguna decisión. Cada decisión es tomada por generadores que son básicamente complementos en el entorno Yeoman. Hay muchos generadores disponibles al público y es fácil crear uno nuevo que se adapte a cualquier flujo de trabajo. Yeoman es siempre la elección correcta para sus necesidades de andamios [54].

4.9. Bootstrap

Es un framework CSS desarrollado inicialmente (en el año 2011) por Twitter que permite dar forma a un sitio web mediante librerías CSS que incluyen tipografías, botones, cuadros, menús y otros elementos que pueden ser utilizados en cualquier sitio web. Aunque el desarrollo del framework Bootstrap fue iniciado por Twitter, fue liberado bajo licencia MIT en el año 2011 y su desarrollo continua en un repositorio de GitHub [54].

4.10. Justificación de tecnologías

4.10.1. Hyperledger Composer

Hyperledger Composer se posicionó como la opción estándar para desarrollar redes de cadenas de bloques sobre Hyperledger Fabric debido a que permite crear aplicaciones descentralizadas abstrayéndose de las configuraciones de bajo nivel. Adicionalmente, se mantiene al día con actualizaciones regulares.

4.10.2. Django + Loopback + Bootstrap

Hyperledger Composer permite a los arquitectos y desarrolladores crear rápidamente soluciones de cadena de bloques "fullstack". Es decir. la lógica de negocios que se ejecuta en la cadena de bloques, las API REST que exponen la lógica de la cadena de bloques a aplicaciones web o móviles, así como la integración de la cadena de bloques con los sistemas de registro empresariales existentes.

Siguiendo esta idea utiliza Loopback Connector para generar una API REST completa. Dada la arquitectura planteada, el uso de este elemento viene siendo la parte intermedia de la arquitectura, el puente que une la base de la red de Fabric con la interfaz de usuario final, como se puede apreciar en la Figura 4-1.

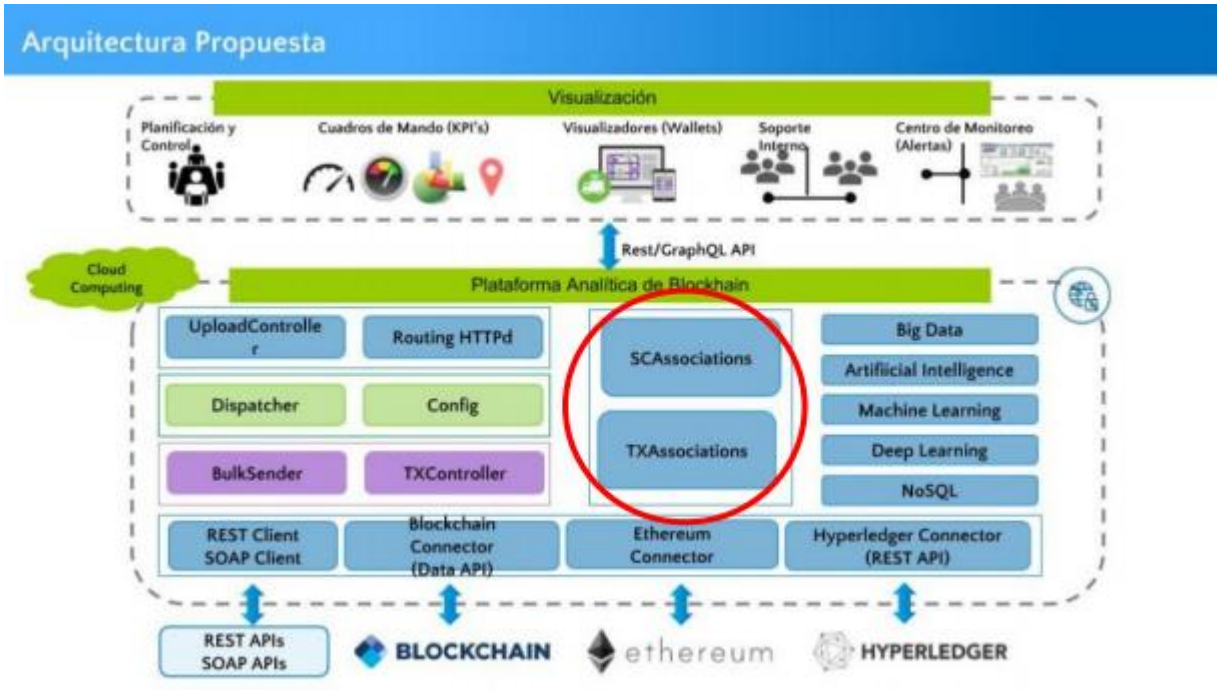

Figura 4-1: Capa intermedia planteada en la arquitectura.

Al generar una API REST completa es posible conectarse a la red con cualquier tecnología por lo que la conexión con Django resulta sencilla. Django al estar basado en Python ofrece una amplia gama de librerías que vuelven más sencillo el desarrollo. De cierta forma, Fabric vendría a reemplazar lo que sería un servidor en una aplicación web convencional, en vista de este cambio de paradigma, Django, viene siendo una solución que cubre todas las necesidades que puede presentar esta versión de cliente web.

Todo el trabajo de crear la interfaz de usuario, habilitar el intercambio de información con Fabric mediante un cliente web y la interacción final del usuario final con la herramienta corresponde a la capa superior a la arquitectura planteada (Figura 4-2), la cual fue desarrollada utilizando Bootstrap.

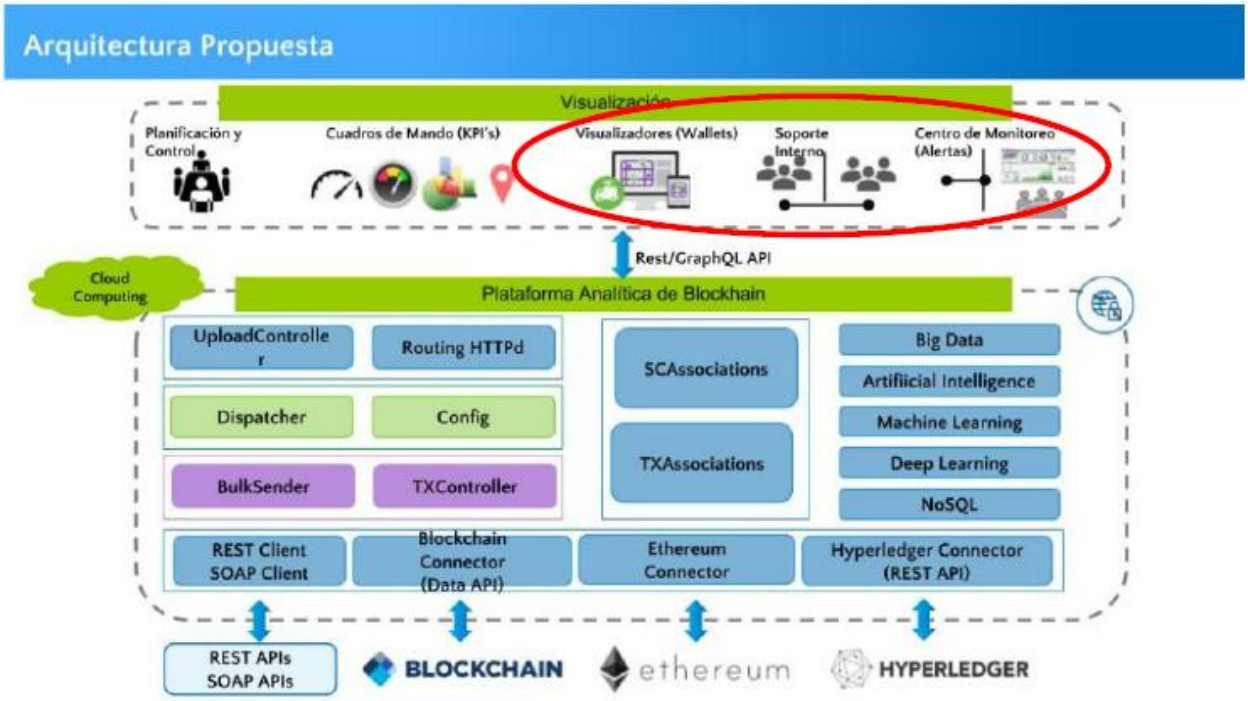

Figura 4-2: Capa final planteada en la arquitectura.

4.11. Almacenamiento

Una de las tareas planteadas para el presente TEG es el de almacenar las interacciones de los usuarios con la cadena de bloques, para ello, es necesario definir las propiedades de algunas bases de datos que puedan ser de utilidad para realizar dicha tarea de forma eficiente.

4.11.1. Bases de Datos NoSQL

Son un enfoque hacia la gestión de datos y el diseño de base de datos útil para grandes conjuntos de datos distribuidos. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad).

Son especialmente útiles cuando se necesita acceder y analizar grandes cantidades de datos no estructurados o datos que se almacenan de forma remota en varios servidores virtuales en la nube [31].

4.10.1.2. Tipos de Bases de Datos NoSQL

- Orientada a Columnas: Este tipo de bases de datos están pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos. Funcionan de forma parecida a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros. Algunos ejemplos de base de datos orientada a columnas: Cassandra [32], HBase [35].

- Orientadas a Clave-Valor: Son sencillas de entender. Simplemente guardan tuplas que contienen una clave y su valor. Cuando se quiere recuperar un dato, simplemente se busca por su clave y se recupera el valor. Algunos ejemplos de base de datos clave/valor: DynamoDB [34], Redis [39].
- Orientadas a Documentos: Son aquellas que gestionan datos semi estructurados. Es decir documentos. Estos datos son almacenados en algún formato estándar como puede ser XML, JSON o BSON. Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales. Algunos ejemplos de base de datos orientada a documentos: MongoDB [37], CouchDB [33].
- Orientada a Grafos: Basadas en la teoría de grafos utilizan nodos y aristas para representar los datos almacenados. Son muy útiles para guardar información en modelos con muchas relaciones, como redes y conexiones sociales. Algunos ejemplos de base de datos orientada a grafos: Infinite Graph [36], Neo4j [38].

4.10.1.2. MongoDB

Es una base de datos ágil que permite a los esquemas cambiar rápidamente a medida que las aplicaciones evolucionan, proporcionando siempre la funcionalidad que los desarrolladores esperan de las bases de datos tradicionales, tales como índices secundarios, un lenguaje completo de búsquedas y consistencia estricta [37].

La misma pertenece a la familia de las bases de datos NoSQL orientadas a documentos, donde se pueden almacenar archivos en formato JSON donde se permite la anidación de los mismos.

Posee un lenguaje de consulta propio del sistema manejador de base de datos, el cual permite realizar cualquier tipo de consulta sin importar el esquema del documento almacenado.

Entre las bondades que brinda MongoDB destacan:

1 MongoDB almacena sus documentos en formatos JSON flexibles, haciendo que los documentos puedan variar a dependiendo de lo que se quiera almacenar.
2 Trabaja con modelos definidos dentro de la aplicación, lo que permite simpleza al momento de modelos los datos a almacenar.
3 Consultas a la medida (Ad hoc), indexación y agregaciones en tiempo real proveen de herramientas flexibles y potentes para analizar los datos.
4 Es código abierto lo que viene siendo una parte fundamental en la creación de DAPP's.

4.10.1.3. Justificación de uso MongoDB

MongoDB al ser una base de datos orientada a documentos, que además, se vale del formato JSON para manejar los modelos, viene siendo la convención perfecta para trabajar junto a tecnologías como NodeJS, gracias a su base en JavaScript, el manejo de objetos no tiene variación alguna con respecto a trabajar con los objetos del lenguaje, y luego almacenar dichos

objetos en la base de datos.

Adicionalmente, la flexibilidad con la que se almacenan los datos, resulta extremadamente ventajoso, a la hora de cambiar el modelo de los datos que se desea almacenar, si a partir de cierto momento, un tipo de dato no contemplado resulta ser bastante útil, puede realizarse una mínima modificación en el modelo de datos sin romper la integración del sistema.

4.11. Inteligencia Artificial

4.11.1. Analítica Predictiva

La analítica predictiva es una forma de análisis avanzado que utiliza datos nuevos e históricos para predecir la actividad futura, el comportamiento y las tendencias. Implica la aplicación de técnicas de análisis estadístico, consultas analíticas y algoritmos automáticos de aprendizaje automático a conjuntos de datos para crear modelos predictivos que sitúen un valor numérico o puntuación en la probabilidad de que ocurra un evento particular.

Las aplicaciones de software de análisis predictivo utilizan variables que pueden medirse y analizarse para predecir el comportamiento probable de individuos, maquinaria u otras entidades [40].

4.11.2. Aprendizaje Automático (Machine Learning)

El aprendizaje automático (Machine Learning, en inglés) es una rama de las ciencias de la computación que es usada para describir algoritmos tanto de aprendizaje supervisado (predicción y clasificación) como aprendizaje no supervisado (clusterizar y detección de comportamiento). Es una rama derivada de la Inteligencia Artificial, que busca desarrollar en las computadoras habilidades de aprendizaje sin ser explícitamente programadas para ello [41].

El aprendizaje automático no es un dominio específico sino un conjunto de dominios que ofrecen diferentes enfoques para resolver problemas complejos. El propósito general del aprendizaje automático es el desarrollo de algoritmos.

Junto a la inteligencia artificial, se enfocan en el desarrollo de algoritmos que pueden enseñarse a sí mismos para adaptarse iterativamente al momento de exponerse a nuevos datos y continuamente mejorar sus resultados.

4.11.3. Aprendizaje profundo (Deep Learning)

El aprendizaje profundo (Deep Learning, en inglés), es una sub-rama del aprendizaje automático que está basado en aprender en diferentes niveles de representación correspondientes a una jerarquía con respecto a características o conceptos, donde los conceptos de alto nivel son definidos en base a los conceptos de bajo nivel y los mismos conceptos de bajo nivel ayudan a construir conceptos de alto nivel.

El aprendizaje profundo es una parte más amplia de los métodos de aprendizaje automático basado en aprender representaciones. Una observación (por ejemplo, una imagen) puede ser representada de varias maneras (como un vector de pixeles), pero algunas representaciones pueden hacer más sencillas algunas tareas de interés (Como el reconocimiento facial en imágenes).

Esta rama del aprendizaje automático se encarga de definir que representaciones son mejores para cierta tarea y como aprender a interpretarla [43].

4.11.4. Minería de Datos (Data Mining)

Surgió como una tecnología que busca ayudar a comprender el contenido de una base de datos, resaltando la importancia de la manipulación de los datos, debido al valor que _estos pueden aportar.

En el momento en que el usuario les atribuye algún significado especial, pasan a convertirse en información. Cuando los especialistas elaboran o encuentran un modelo, haciendo que la interpretación obtenida entre la información y ese modelo represente un valor agregado, es cuando se obtiene el conocimiento. La minería de datos busca patrones, comportamientos, agrupaciones, secuencias, tendencias o asociaciones, que puedan generar algún modelo que permita comprender mejor el dominio para ayudar en una posible toma de decisión. Con respecto a un problema [42].

4.12. Alcance de la Investigación

Definiciones más específicas de los elementos anteriormente descritos se escapan del alcance de la presente investigación debido a que el tema principal es la implementación de una aplicación distribuida que pueda desenvolverse en el ambiente de Hyperledger Fabric y la tarea de obtener datos de interés de la misma.

La metodología con respecto al procesamiento e interpretación de los datos que puedan ser obtenidos a partir del contenido de las cadenas de bloques queda en responsabilidad de futuras investigaciones con respecto a los datos una vez que ya han sido extraídos de la cadena de bloques.

Capítulo 5 Capítulo 5 – Marco metodológico

5.1. Metodología de desarrollo y justificación

La selección de una metodología de desarrollo viene siendo una tarea fundamental a la hora de crear cualquier solución de la índole de la presente investigación.

En un mundo tan cambiante, dónde las tecnologías son cada vez más diversas y especializadas, las metodologías ágiles brillan por su rápida adaptación a los diferentes paradigmas que nacen a diario en el mundo tecnológico. Sin embargo, el enfoque tradicional, brinda muchas herramientas ya establecidas las cuales gozan de validación a nivel mundial, permitiendo crear soluciones de una manera eficiente.

Con el objetivo de explotar lo mejor de ambos mundos, la adaptabilidad de las metodologías ágiles, y lo certero de metodologías tradicionales, la propuesta para la presente investigación es la metodología Proceso Ágil Unificado [28].

5.2. Proceso Ágil Unificado

Los primeros pasos que concibieron el nacimiento de esta metodología fueron en el año 1999 por Scott W Ambler, como una solución para mejorar la metodóloga RUP. La idea principal, era actualizar RUP [28] para que fuera más ágil debido a todos los cambios tecnológicos que nacían en la época.

En esencia, es una versión simplificada de RUP. Describe de una manera simple los acercamientos para crear software orientado a las soluciones de negocios utilizando herramientas ágiles, pero, basándose en conceptos utilizados en RUP.

Se siguen conservando elementos de RUP tales como los modelos, pero ahora no de una manera tan dominante, está más enfocado a prácticas ágiles, como por ejemplo: El uso de modelos sigue siendo fundamental para simular las soluciones, pero ahora el desarrollo se divide en iteraciones ‘Lo suficientemente buenas’, según Ambler [29].

5.2.1. Fases

1 Inicio (*Inception*): La meta de esta fase, es definir el alcance del proyecto, potenciales arquitecturas para una solución viable y definir los costos asociados a la solución y su vialidad.
2 Elaboración (*Elaboration*): Definir la arquitectura de la solución.
3 Construcción (*Construction*): La meta de esta fase es construir software de una manera simple, pero incremental, que solucione las problemáticas bases planteadas en la fase de inicio y luego, mediante iteraciones ágiles, ir puliendo problemáticas secundarias.

4 Transición (*Transition*): Esta fase representa la validación del sistema en un ambiente de producción.

5.2.2. Disciplinas y actividades

Las disciplinas son desarrolladas de una manera iterativa, definiendo al equipo de desarrollo actividades que ayuden a validar, probar y entregar software que satisfaga la necesidad de la solución planteada, las disciplinas son [30]:

- Modelado: El objetivo de esta disciplina es entender el negocio de la organización, el dominio del problema e identificar potenciales soluciones.
- Implementación: Transformar los modelos en código ejecutable y realizar pruebas básicas, en particular pruebas unitarias.
- Pruebas: Definir pruebas que aprueben o rechacen el software implementado, asegurando estándares de calidad. Esto incluye encontrar defectos, y verificar cumplimiento de requerimientos.
- Despliegue: Planear la entrega del sistema y ejecutar un plan para que los usuarios finales puedan acceder a la solución.
- Gestión de configuración: Gestionar el acceso a los artefactos del proyecto. Esto incluye, además de la traza de versiones de los artefactos, el control de cambios y la gestión de los mismos.
- Gestión de proyectos: Dirigir las actividades que tomarán acción dentro del proyecto, esto incluye la interacción con entes externos para asegurar una entrega del producto a tiempo y dentro del presupuesto acordado.
- Ambiente: La tarea de esta disciplina es asegurar un ambiente adecuado para maximizar las tareas de desarrollo de la solución.

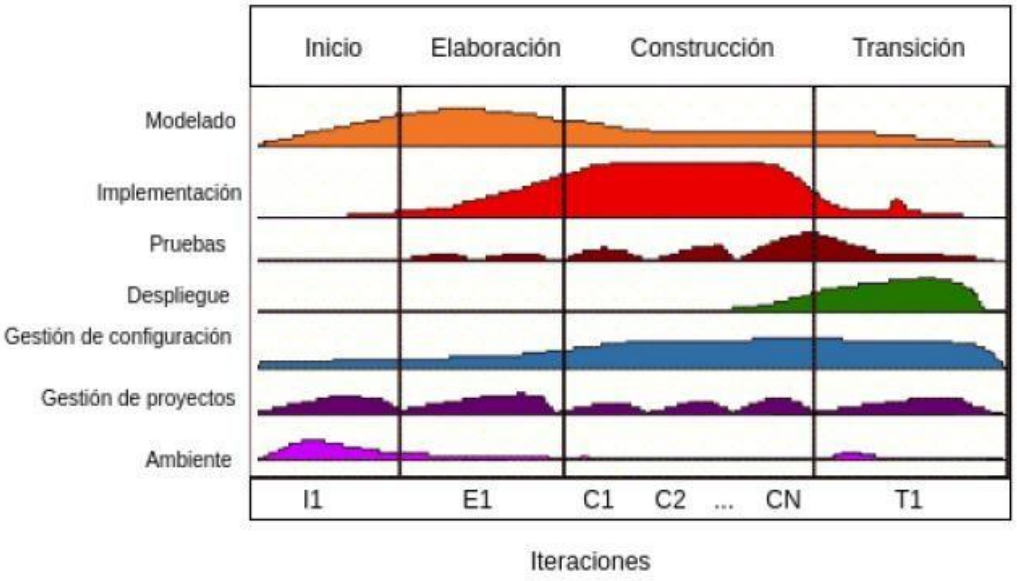

Figura 5-1: Ciclo de vida del Proceso Ágil Unificado.

5.2.3. Iteraciones alrededor del tiempo

En lugar de entregar un sólo proyecto final, se realizan entregas en porciones también conocidas como versiones de prueba, las cuales van evaluando partes del problema que van siendo resueltas, una vez que una versión de prueba es liberada, va a una área de prueba (conocida también como ambiente de calidad).

La idea de este acercamiento, es ir realizando pre-entregas en intervalos de tiempo cada vez más cortos, con el fin de crear un hábito en el equipo que garantice la agilidad y la robustez del desarrollo a medida que se refinan las pre-entregas, luego de cada cierto número de pre-entregas, se lanza una versión de producción, la cual servirá de base para las siguientes pre-entregas, se puede apreciar un flujo de trabajo en la Figura 5-2.

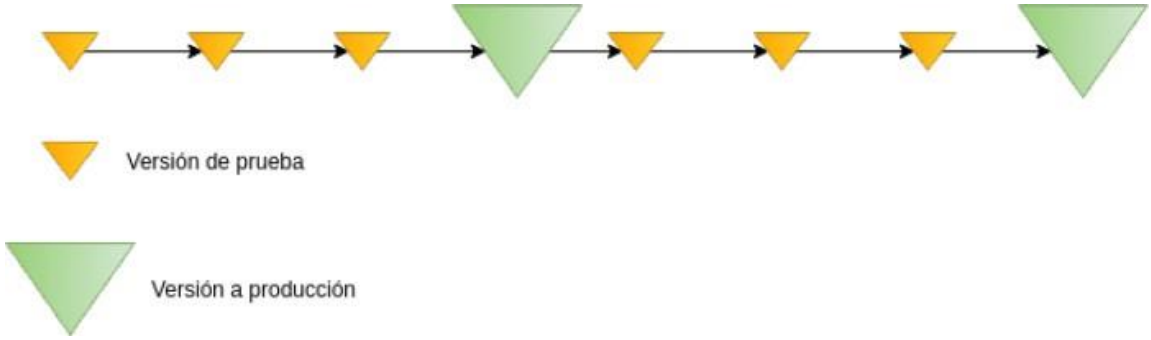

Figura 5-2: Iteraciones de liberación de versiones.

5.2.4. Filosofía del Proceso Ágil Unificado

1. El equipo sabe lo que está haciendo: El equipo no requiere una documentación detallada todos los das, realizar guías concretas y cortas es suficiente para asegurar que el equipo vaya en la dirección correcta.
2. Simplicidad: Todos los requerimientos y documentaciones están descritos en páginas cortas al alcance de la mano.
3. Agilidad: Mantener los principios ágiles, pero conservando conceptos básicos de RUP.
4. Mantener el foco en actividades de alto valor: Enfocarse principalmente en actividades que cuentan y son importantes para llegar a la solución.
5. Independencia de herramientas: Dentro de lo necesario para el desarrollo, permitir que el equipo trabaje con las herramientas con las que más se sientan a gusto.

Capítulo 6 – Marco aplicativo

6.1. Inicio (Inception)

6.1.1. Propuesta de caso de uso para el TEG

6.1.1.1. Solución para la compra y venta de datos personales

6.1.1.2. ¿Qué problemas presentan actualmente las recaudaciones de datos?

Las compañías de tecnología gigantes han utilizado inteligentemente los protocolos técnicos subyacentes de Internet para crear capas de aplicaciones propietarias que capturan y controlan grandes cantidades de datos personales. En la economía de hoy, los datos equivalen a dinero.

En consecuencia, cientos de millones de datos personales de ciudadanos se han comprometido en los últimos años. La reciente violación de la seguridad de Equifax es solo uno de los muchos incidentes que resaltan el problema con los datos personales de las personas. Bajo el sistema actual, las personas no tienen absolutamente ningún control sobre cómo Equifax y otros intermediarios de datos recopilan, protegen y utilizan sus datos personales.

Aunque los organismos gubernamentales y las organizaciones de derechos de los consumidores, especialmente en la Unión Europea, están tratando de mantener un equilibrio adecuado de transparencia, uso y acceso, cuando se trata de datos personales, están luchando en una batalla cuesta arriba.

Este ecosistema de datos rotos asigna erróneamente el valor de los datos a su propietario legítimo, el individuo, y evita que la sociedad aborde de manera efectiva muchos de sus mayores desafíos.

Es el momento de que los ciudadanos recuperen el control sobre sus datos y comiencen a participar activamente en el valor generado por sus datos.

6.1.1.3. ¿Cómo pueden Hyperledger Fabric y los contratos inteligentes atacar estos problemas?

Mediante el uso de la cadena de bloques, los vendedores y compradores de datos pueden ejecutar contratos inteligentes y transferir directamente los datos asociados en una transacción fuera de la cadena cifrada. Como resultado, las identidades del vendedor nunca se revelan a los compradores de datos sin el consentimiento explícito del vendedor. A su vez, los compradores obtienen datos de alta calidad directamente de la fuente y ya no tienen que depender de inferencias de baja calidad de fuentes inexactas e indirectas.

6.1.1.4. Solución

La propuesta plantea crear una aplicación web que se apoya en el uso de Hyperledger Fabric y

contratos inteligentes para hacer más transparentes y confiables la compra y venta de datos.

En un primer acercamiento, un usuario, se encarga de crear un dataset mediante el cliente web.

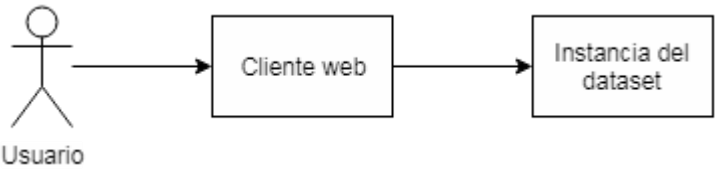

Figura 6-1: Usuario crea una instancia de un dataset mediante un cliente web.

En el cliente web, en la ventana de creación de datasets, el usuario puede parametrizar su dataset y establecer el monto de venta para su dataset.

Una vez creado el dataset, los demás usuarios pueden acceder al cliente web y seleccionar el dataset que desean comprar y seguidamente, se comprueba que los compradores disponen del monto necesario para la compra y en caso afirmativo el correo de los compradores es almacenado en el dataset con el fin de que formen parte ahora de la lista especial de autorizados a leer dicho dataset.

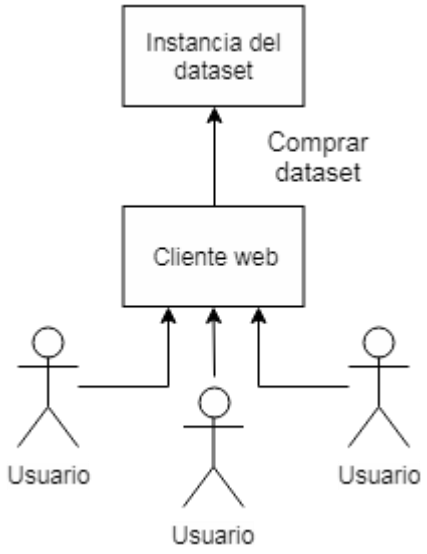

Figura 6-2: Usuarios compran un dataset mediante cliente web.

Automáticamente los compradores pueden ver el contenido completo del dataset que han comprado.

Adicionalmente, con el fin de detectar patrones de comportamiento y encontrar datos relevantes en este tipo de emprendimientos, todas las interacciones que se realicen con la cadena de bloques (Crear datasets, compras, modificaciones) quedan registradas en una instancia de MongoDB para llevar un registro detallado de las direcciones que interactúan con los datasets y cómo son dichas interacciones.

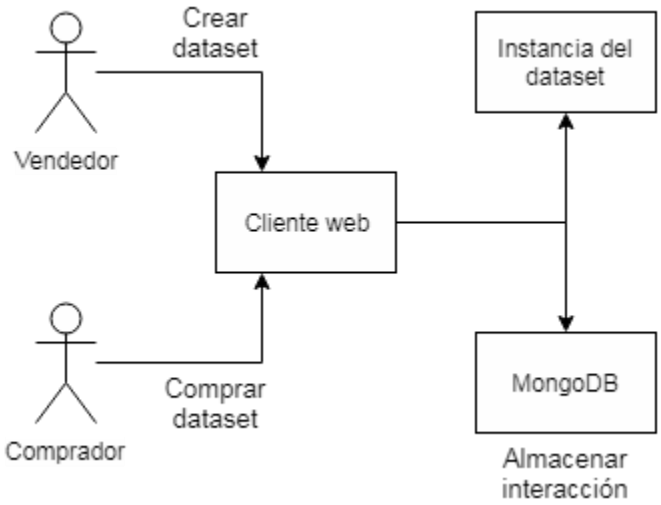

Figura 6-3: Todas las interacciones de parte de los usuarios con el dataset son registradas en una instancia de MongoDB.

Finalmente, se presentan los casos de uso para los usuarios.

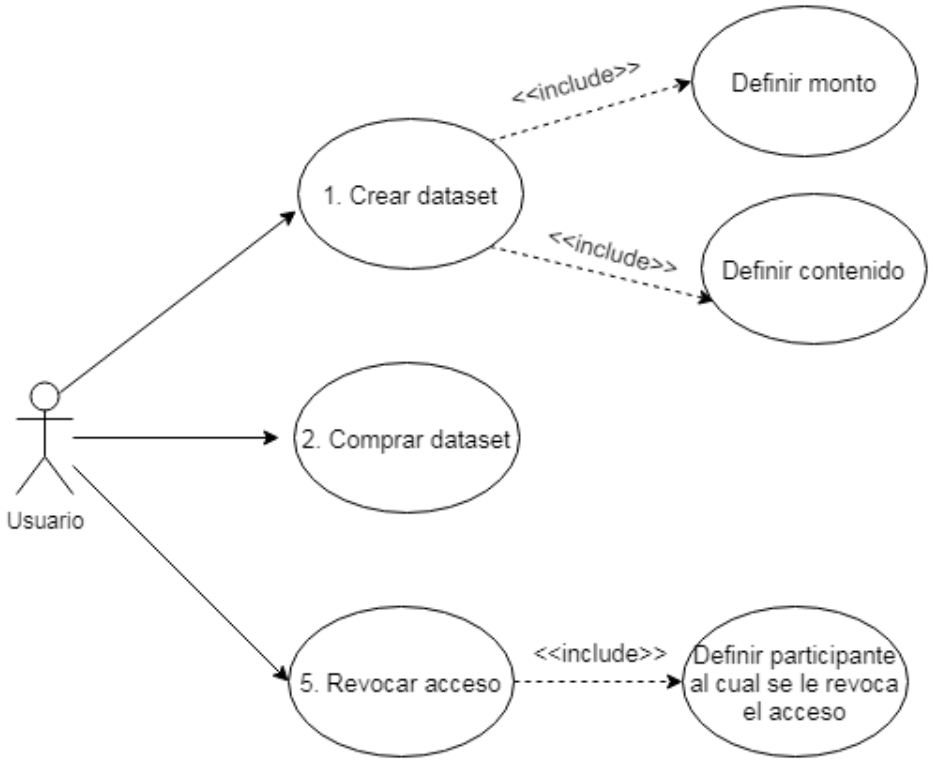

Figura 6-4: Diagrama de caso de uso de los usuarios.

6.2. Elaboración (Elaboration)

La arquitectura planteada para la solución del caso de uso fue planteada en la introducción del presente trabajo y sus componentes fueron identificados en los capítulos 3 y 4.

6.3. Construcción (Construction)

En esta sección se definirán las iteraciones que darán forma a la solución.

6.3.1. Iteración 0: Acercamiento inicial

En esta primera iteración se realizó el primer prototipo del código fuente, el cual ataca las problemáticas fundamentales descritas en el caso de uso.

El concepto clave para Hyperledger Composer es la definición de red de negocios (BND). Define el modelo de datos, la lógica de transacción y las reglas de control de acceso para su solución de blockchain. Para crear un BND, necesitamos crear una estructura de proyecto adecuada en el disco.

Una red de negocios está formada por activos, participantes, transacciones, reglas de control de acceso y, opcionalmente, eventos y consultas. En la red de negocios de esqueleto creada en el paso anterior, hay un archivo modelo (.cto) que contendrá las definiciones de clase para todos los activos, participantes y transacciones en la red de negocios. La red empresarial principal también contiene un documento de control de acceso (permissions.acl) con reglas de control de acceso básicas, un archivo de script (logic.js) que contiene funciones de procesador de transacciones y un archivo package.json que contiene metadatos de la red empresarial.

Debido a este temprano enfoque para una solución, el desarrollo de la red y las pruebas se realizaron de manera manual mediante la herramienta Hyperledger Composer Playground.

El primer paso es crear una red de negocio vacía, esto se realiza de la siguiente manera:

1. Se selecciona la opción ‘Deploy a new business network’.

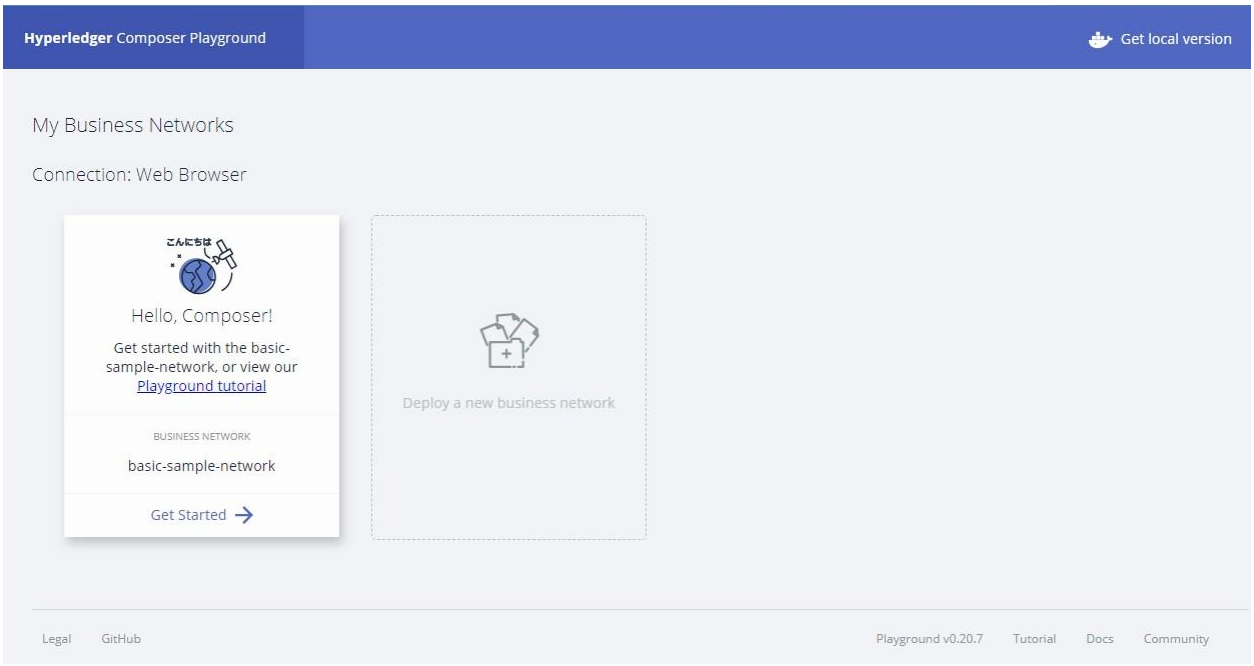

Figura 6-5: Interfaz inicial de Hyperledger Composer Playground.

2. Aparecen platillas de diferentes modelos de red de negocio, para el efecto de este desarrollo se selecciona desplegar un modelo de red vacío 'empty-business-network', y se acciona el botón 'Deploy'.

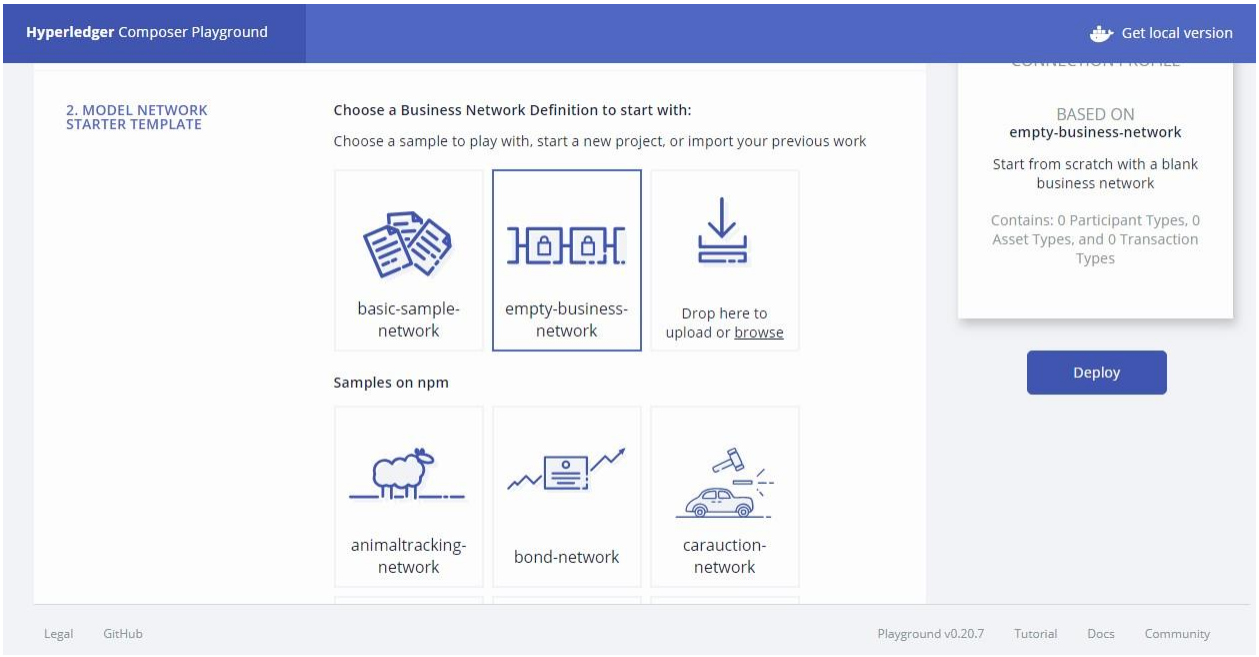

Figura 6-6: Definición inicial de la red utilizando plantillas predefinidas.

3. Luego se procede a conectarse a la red haciendo click en 'Connect now'.

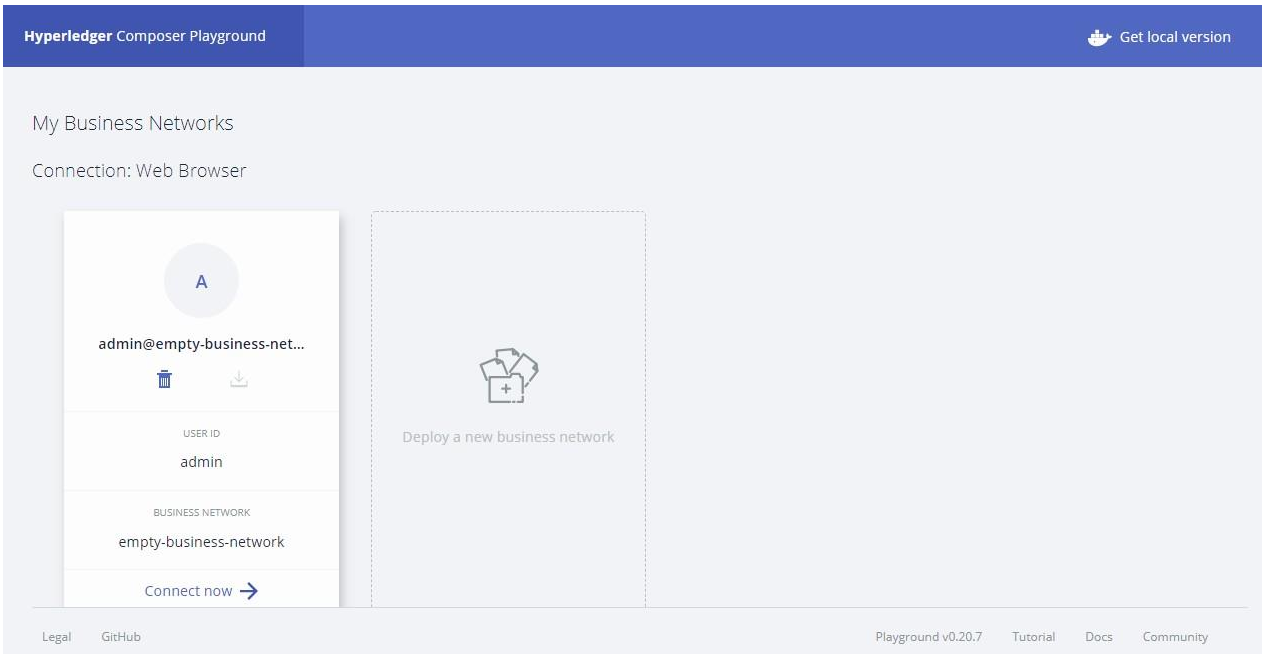

Figura 6-7: Conexión a la red creada.

4. Se despliega una interfaz para realizar la definición de la red en los archivos: README.md, model.cto y permissions.acl

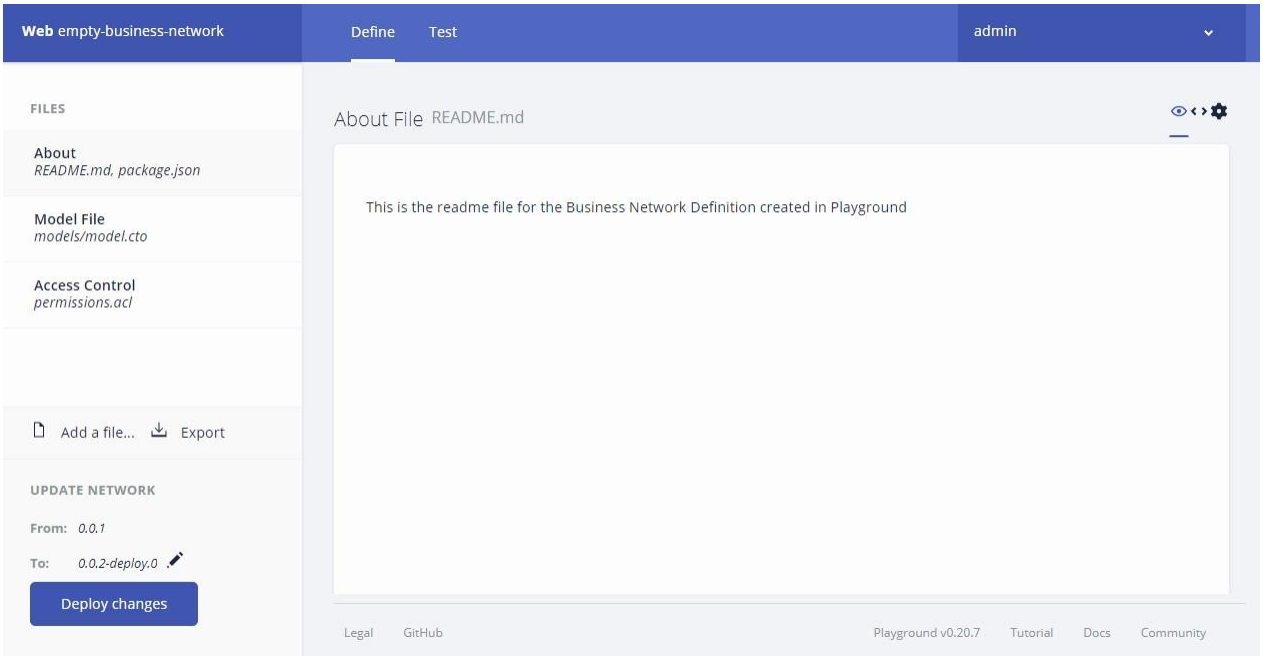

Figura 6-8: Definición de la red en Hyperledger Composer Playground.

El primer documento para actualizar es el archivo modelo (.cto). Este archivo se escribe utilizando el lenguaje de modelado de Hyperledger Composer. Se definen un solo tipo de participante 'Person', tres diferentes tipos de activos: Dataset que representa el conjunto de datos, Persona_data es un tipo de conjunto de datos enfocado solo a datos personales y

Medical_data un tipo de conjunto de datos con atributos enfocados al área médica. Las transacciones definidas son: Buy que consiste en el contrato inteligente para la comprar un Dataset y RevokeAccess otro contrato inteligente para revocar el acceso a un Dataset, también se define un evento llamado AuthorizedAccess el cual notifica cuando se ejecuta correctamente una transacción del tipo Buy y se autoriza a un nuevo participante a leer un Dataset.

```
19  namespace org.redv2
20
21  enum Sexo {
22    o FEMENINO
23    o MASCULINO
24  }
25
26  enum DatasetDesc {
27    o DATOS_PERSONALES
28    o DATOS_MEDICOS
29    o DATOS_PERSONALES_Y_DATOS_MEDICOS
30  }
31
32  //Definición de participantes
33  participant Person identified by email {
34    o String email
35    o Double coins
37  }
```

```
38
39    //Definición de activos
40    //Dataset principal
41    asset Dataset identified by datasetId {
42      o String datasetId
43      o Double price
44      o DatasetDesc description
45      --> Person owner
46      --> Personal_data perData optional
47      --> Medical_data medData optional
48    }
49
50    //Definición de atributos para el conjunto de datos personales
51    asset Personal_data identified by dataId {
52      o String dataId
53      o String firstName optional
54      o String lastName optional
55      o DateTime birth_date
56      o Sexo gender
57      o String origin_country
58      o String[] authorized optional
59      --> Person owner
60    }
61
62    //Definición de atributos para el conjunto de datos médicos
63    asset Medical_data identified by dataId {
64      o String dataId
65      o Double height optional
66      o Double weight optional
67      o Double blood_pressure optional
68      o Double glucose_level optional
69      o String residence_country optional
70      o String[] authorized optional
71      --> Person owner
72    }
73
74  //Definición de transación
75  //Transacción para realizar la compra
76  transaction Buy {
77    --> Dataset dataset
78    --> Person buyer
79  }
80
81  //Transacción para revocar accesos a un dataset
82  transaction RevokeAccess{
83    o String personId
84    --> Dataset dataset
85  }
86
87  //Definición de evento:
88  //Este evento se genera para indicar que se agregó a un participante a la lista de autorizados a ver un dataset
89  event AuthorizedAccess {
90    --> Dataset dataset
91    o String owner
92    o String newAuthorized
93  }
94
```

Algoritmo 6-1: Definición de los elementos de la red.

Ahora que el modelo de dominio se ha definido, se puede definir la lógica de transacción para

la red de negocios. Composer expresa la lógica de una red de negocios utilizando funciones de JavaScript. Estas funciones se ejecutan automáticamente cuando se envía una transacción para su procesamiento. Esta lógica se plasma en un archivo llamado logic.js.

La primera transacción en definir es RevokeAccess.

```
/**
 * A Person revokes access to its data set to another person.
 * @param {org.redv2.RevokeAccess} revoke - the RevokeAccess to be processed
 * @transaction
 */
async function revokeAccess(revoke) {
    //Buscar el participante actual, quien ejecuta la transacción.
    const me = getCurrentParticipant();
    //Se almacena en una constante el valor del dataset pasado por parámetro.
    const dataset = revoke.dataset;
    console.log('**** REVOKE: ' + me.getIdentifier() + ' revoking access to ' + revoke.personId + ' in ' + dataset.da

    //Si es participante no existe o no está certificado se cancela la transacción.
    if(!me) {
        throw new Error('A participant/certificate mapping does not exist.');
    }

    // Si la persona está autorizada se remueve de las listas de autorizados
    // Si el dataset contiene un conjunto de datos personales se remueve al autorizado de la lista de este conjunto
    if (dataset.perData) {
        //Almacenar en una constante el dataset de datos personales
        const perData = dataset.perData;
        //Verificar si el participante al cual se le revocará el acceso efectivamente está en la lista de autorizados
        //Se almacena en la constante index1 la posición del email a remover en la lista
        const index1 = perData.authorized ? perData.authorized.indexOf(revoke.personId) : -1;

        //Si el participante está en la lista, entonces se procede a eliminar de la lista
        if(index1>-1) {
            //Se saca de la lista de autorizados el email en la posición que indica la constante index1
            perData.authorized.splice(index1, 1);

            //Se obtienen los registros de activos del tipo org.redv2.Personal_data
            const datasetRegistry = await getAssetRegistry('org.redv2.Personal_data');
            //Se actualiza el activo especifico
            await datasetRegistry.update(perData);
        }
    }
    // Si el dataset contiene un conjunto de datos médicos se remueve al autorizado a la lista de este conjunto
    if (dataset.medData) {
        //Almacenar en una constante el dataset de datos médicos
        const medData = dataset.medData;
        //Verificar si el participante al cual se le revocará el acceso efectivamente está en la lista de autorizados
        //Se almacena en la constante index2 la posición del email a remover en la lista
        const index2 = medData.authorized ? medData.authorized.indexOf(revoke.personId) : -1;

        //Si el participante está en la lista, entonces se procede a eliminar de la lista
        if(index2>-1) {
            //Se saca de la lista de autorizados el email en la posición que indica la constante index2
            medData.authorized.splice(index2, 1);

            //Se obtienen los registros de activos del tipo org.redv2.Medical_data
            const datasetRegistry = await getAssetRegistry('org.redv2.Medical_data');
            //Se actualiza el activo especifico
            await datasetRegistry.update(medData);
        }
    }
}
```

Algoritmo 6-2: Definición de la transacción RevokeAccess Luego se crea la segunda función makeBuyout para la transacción Buy

```
/**
 * Buy a Dataset
 * @param {org.redv2.Buy} buyout - the buyout of a dataset
 * @transaction
 */
async function makeBuyout(buyout) {
    const dataset = buyout.dataset;

    const buyer = buyout.buyer;
    const seller = dataset.owner;

    if  (buyer.coins >= dataset.price){
        //Actualizar las monedas del vendedor.
        console.log('#### seller coins before: ' + seller.coins);
        seller.coins += dataset.price;
        console.log('#### seller coins after: ' + seller.coins);
        //Actualizar las monedas del comprador.
        console.log('#### buyer coins before: ' + buyer.coins);
        buyer.coins -= dataset.price;
        console.log('#### buyer coins after: ' + buyer.coins);
        // Dar acceso al comprador.
        console.log('**** AUTH: ' + seller.email + ' granting access to ' + buyer.email + ' to dataset ' + dataset.da
        // Si la persona aún no ha sido autorizada se agrega a la lista de autorizados.
        // Se define una variable index para determinar si el participante está o no en la lista.
        // -1 si no está en la lista, cualquier otro número significa la posición que se encuentra en la lista.
        let index = -1;

        // Si el dataset contiene un conjunto de datos personales se agrega el email a la lista de este conjunto.
        if (dataset.perData){
            // El campo autorizados es opcional puede que aún no haya sido creado, por eso se verifica si no es null.
            if (!dataset.perData.authorized) {
                // Si es null entonces se inicializa el arreglo.
                dataset.perData.authorized = [];
            }
            else {
                // Si ya fue creado, se busca en el arreglo el email del comprador.
                // En index se guardará la posición donde se encuentra el email, si no está retorna -1.
                index = dataset.perData.authorized.indexOf(buyer.email);
            }

            // Si el email no está el index será -1 y se agrega el email a la lista de autorizados.
            if(index < 0) {
                dataset.perData.authorized.push(buyer.email);
            }

            // Se obtienen los registros de activos del tipo org.redv2.Personal_data.
            const datasetRegistry = await getAssetRegistry('org.redv2.Personal_data');
            // Se actualiza el activo especifico
            await datasetRegistry.update(dataset.perData);
        }

        // Si el dataset contiene un conjunto de datos médicos se agrega el email a la lista de este conjunto.
        if (dataset.medData){
            // El campo autorizados es opcional puede que aún no haya sido creado, por eso se verifica si no es null.
            if (!dataset.medData.authorized) {
                // Si es null entonces se inicializa el arreglo.
                dataset.medData.authorized = [];
            }
            else {
                // Si ya fue creado, se busca en el arreglo el email del comprador.
                // En index se guardará la posición donde se encuentra el email, si no está retorna -1.
                index = dataset.medData.authorized.indexOf(buyer.email);
            }

            // Si el email no está el index será -1 y se agrega el email a la lista de autorizados.
            if(index < 0) {
                dataset.medData.authorized.push(buyer.email);
            }

            // Se obtienen los registros de activos del tipo org.redv2.Medical_data
            const datasetRegistry = await getAssetRegistry('org.redv2.Medical_data');
            // Se actualiza el activo especifico
            await datasetRegistry.update(dataset.medData);
        }
```

```
152
158
159         // Se emite un evento para notificar que un dataset fue modificado.
160         let event = getFactory().newEvent('org.redv2', 'AuthorizedAccess');
161         event.dataset = dataset;
162         event.owner = seller.email;
163         event.newAuthorized = buyer.email;
164         emit(event);
165     }
166     else{
167         throw new Error('The buyer doesnt have enough coins to buy');
168     }
169 }
```

Algoritmo 6-3: Definición de la función para la transacción Buy

Por último se definen las reglas de acceso a la red, modificando el archivo permissions.acl

```
23  rule EveryoneCreateReadDatasets{
24    description: "Allow all participants create and read access to datasets"
25      participant: "org.redv2.Person"
26      operation: READ
27      resource: "org.redv2.Dataset"
28      action: ALLOW
29  }
30
31  rule OwnDatasetFullAccess {
32      description: "Allow all participants full access to their own dataset"
33      participant(p): "org.redv2.Person"
34      operation: ALL
35      resource(r): "org.redv2.Dataset"
36      condition: (r.owner.getIdentifier() === p.getIdentifier())
37      action: ALLOW
38  }
39
40  rule OwnRecordFullAccess {
41      description: "Allow all participants full access to their own record"
42      participant(p): "org.redv2.Person"
43      operation: ALL
44      resource(r): "org.redv2.Person"
45      condition: (r.getIdentifier() === p.getIdentifier())
46      action: ALLOW
47  }
48
49  rule CreateDatasetRegistry {
50      description: "Allow all participants to create Personal_data asset"
51      participant(p): "org.redv2.Person"
52      operation: CREATE
53      resource(r): "org.redv2.Dataset"
54      condition: (r.owner.getIdentifier() === p.getIdentifier())
55      action: ALLOW
56  }
57
58  rule OwnPersonalDataFullAccess {
59      description: "Allow all participants full access to their own Personal_data asset"
60      participant(p): "org.redv2.Person"
61      operation: ALL
62      resource(r): "org.redv2.Personal_data"
63      condition: (r.owner.getIdentifier() === p.getIdentifier())
64      action: ALLOW
65  }
66
67  rule OwnMedicalDataFullAccess {
68      description: "Allow all participants full access to their own Medical_data asset"
69      participant(p): "org.redv2.Person"
70      operation: ALL
71      resource(r): "org.redv2.Medical_data"
72      condition: (r.owner.getIdentifier() === p.getIdentifier())
73      action: ALLOW
74  }
```

```
75
76  rule CreateBuyTransaction {
77      description: "Allow all participants to submit Buy transactions"
78      participant: "ANY"
79      operation: CREATE
80      resource: "org.redv2.Buy"
81      action: ALLOW
82  }
83
84  rule RevokeAccessTransaction {
85      description: "Allow all participants to submit RevokeAccess transactions"
86      participant(p): "org.redv2.Person"
87      operation: CREATE
88      resource(r): "org.redv2.RevokeAccess"
89      condition: (r.dataset.owner.getIdentifier() === p.getIdentifier())
90      action: ALLOW
91  }
92
93  //It allows those who bought the dataset to see all their information
94  rule ForeignRecordPersonalDataConditionalAccess {
95      description: "Allow participants access to other people's datasets if granted"
96      participant(p): "org.redv2.Person"
97      operation: READ
98      resource(r): "org.redv2.Personal_data"
99      condition: (r.authorized && r.authorized.indexOf(p.getIdentifier()) > -1)
100     action: ALLOW
101 }
102
103 rule DatasetPersonalDataUpdateAccessWithTransaction {
104     description: "Allow all participants update access dataset when they do a buyout"
105     participant(p): "org.redv2.Person"
106     operation: READ,UPDATE
107     resource: "org.redv2.Personal_data"
108     transaction (tx): "org.redv2.Buy"
109     condition: (tx.buyer.getIdentifier() === p.getIdentifier())
110     action: ALLOW
111 }
112
113 rule ForeignRecordMedicalDataConditionalAccess {
114     description: "Allow participants access to other people's datasets if granted"
115     participant(p): "org.redv2.Person"
116     operation: READ
117     resource(r): "org.redv2.Medical_data"
118     condition: (r.authorized && r.authorized.indexOf(p.getIdentifier()) > -1)
119     action: ALLOW
120 }
121
122 rule DatasetMedicalDataUpdateAccessWithTransaction {
123     description: "Allow all participants update access dataset when they do a buyout"
124     participant(p): "org.redv2.Person"
125     operation: READ,UPDATE
126     resource: "org.redv2.Medical_data"
127     transaction (tx): "org.redv2.Buy"
128     condition: (tx.buyer.getIdentifier() === p.getIdentifier())
129     action: ALLOW
130 }
131
132 rule PersonUpdateAccessWithTransaction {
133     description: "Allow all participants update access person when they buy a dataset"
134     participant(p): "org.redv2.Person"
135     operation: READ,UPDATE
136     resource: "org.redv2.Person"
137     transaction (tx): "org.redv2.Buy"
138     condition: (tx.buyer.getIdentifier() === p.getIdentifier())
139     action: ALLOW
140 }
141
142 rule SystemACL {
143     description:  "System ACL to permit all access"
144     participant: "org.hyperledger.composer.system.Participant"
145     operation: ALL
146     resource: "org.hyperledger.composer.system.**"
147     action: ALLOW
148 }
```

```
149
150 rule NetworkAdminUser {
151     description: "Grant business network administrators full access to user resources"
152     participant: "org.hyperledger.composer.system.NetworkAdmin"
153     operation: ALL
154     resource: "**"
155     action: ALLOW
156 }
157
158 rule NetworkAdminSystem {
159     description: "Grant business network administrators full access to system resources"
160     participant: "org.hyperledger.composer.system.NetworkAdmin"
161     operation: ALL
162     resource: "org.hyperledger.composer.system.**"
163     action: ALLOW
164 }
```

Algoritmo 6-4: Definición de reglas de acceso a la red

Después de realizar todas las definiciones se realiza el despliegue y se pasa al área de Test para realizar las pruebas de funcionamiento de la red. Como se mencionó anteriormente para efectos de este primer acercamiento las pruebas se realizaron manuales.

Tipo de Prueba	Funcional
Entorno de ejecución	Navegador Web Chrome
Ejecutante	Desarrollador de la herramienta
Herramienta usada	Hyperledger Composer Playground
Versión ejecutada	1
Resultados	100% de ´ exito

Tabla 6-1: Especificaciones de la prueba para la versión 0.1

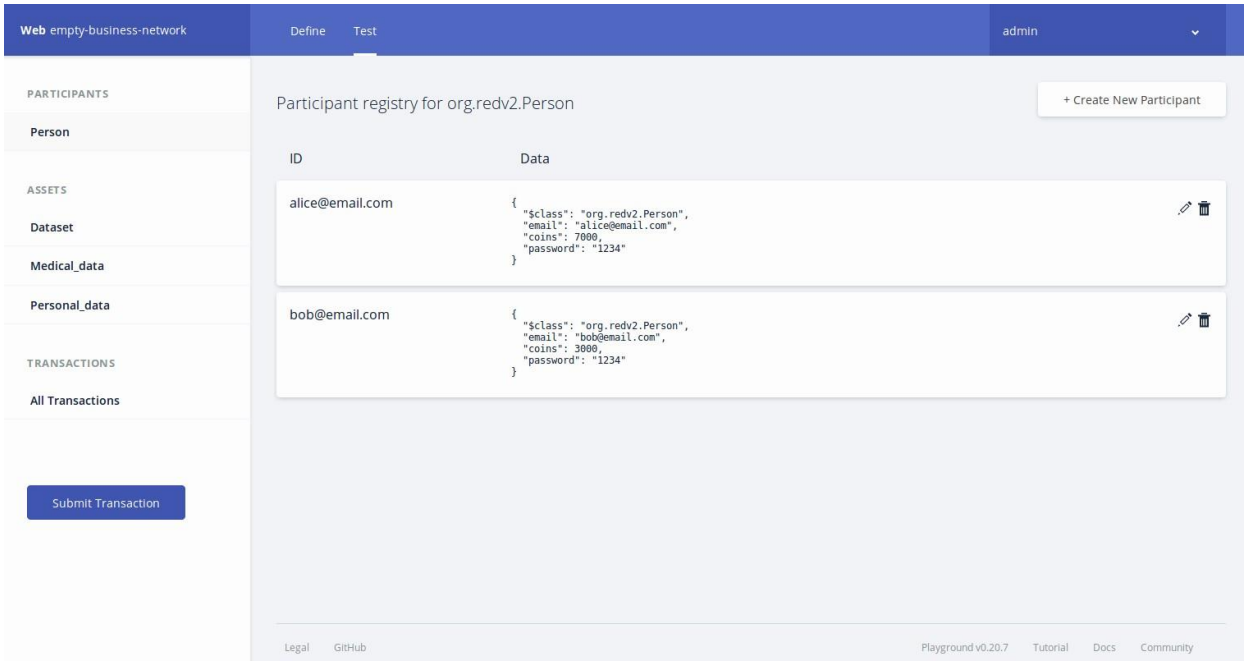

Figura 6-9: Agregar participantes del tipo Person.

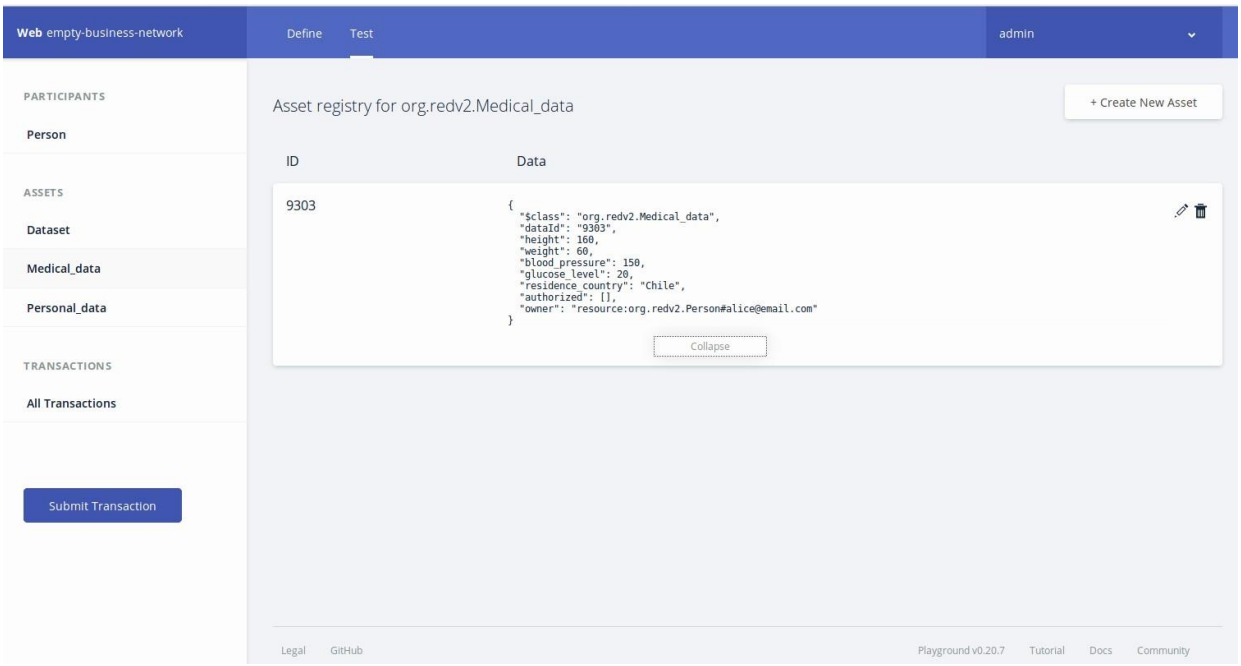

Figura 6-10: Lista de los activos del tipo Medical_data registrados en la red.

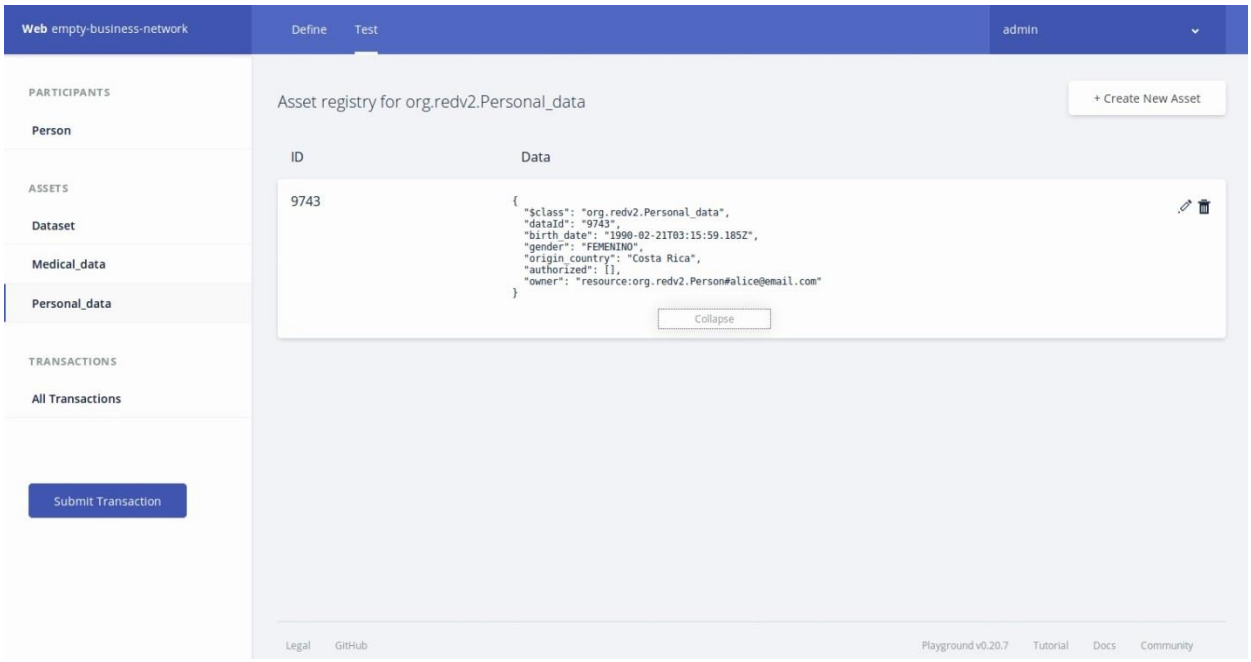

Figura 6-11: Lista de activos del tipo Personal_data registrados en la red.

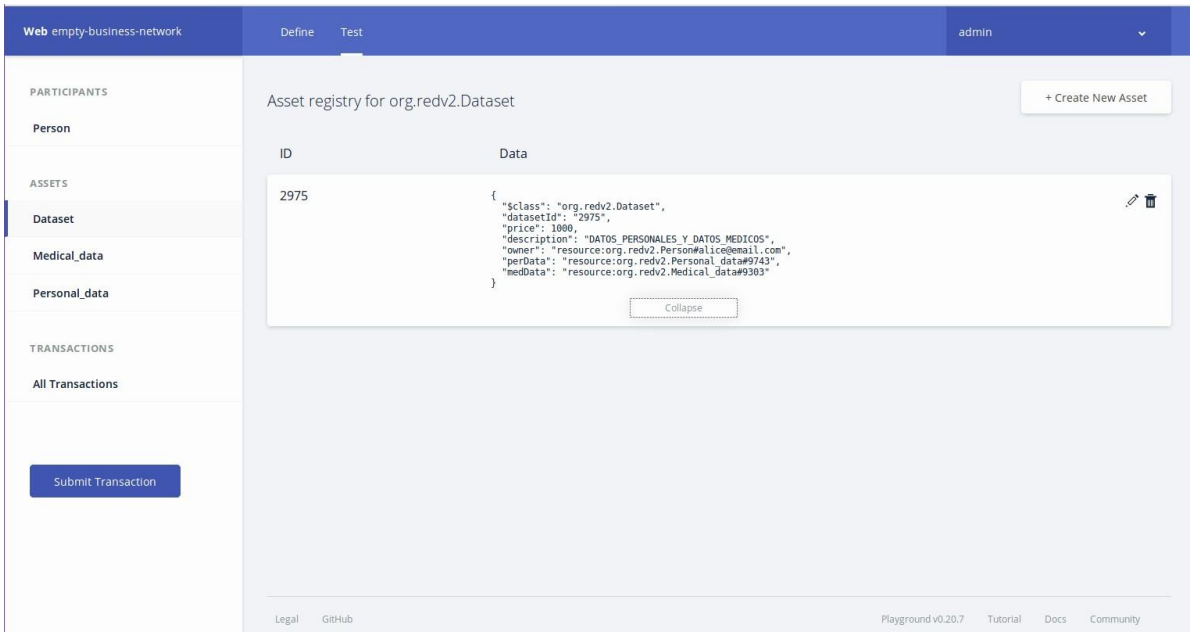

Figura 6-12: Todos los activos Dataset registrados en la red.

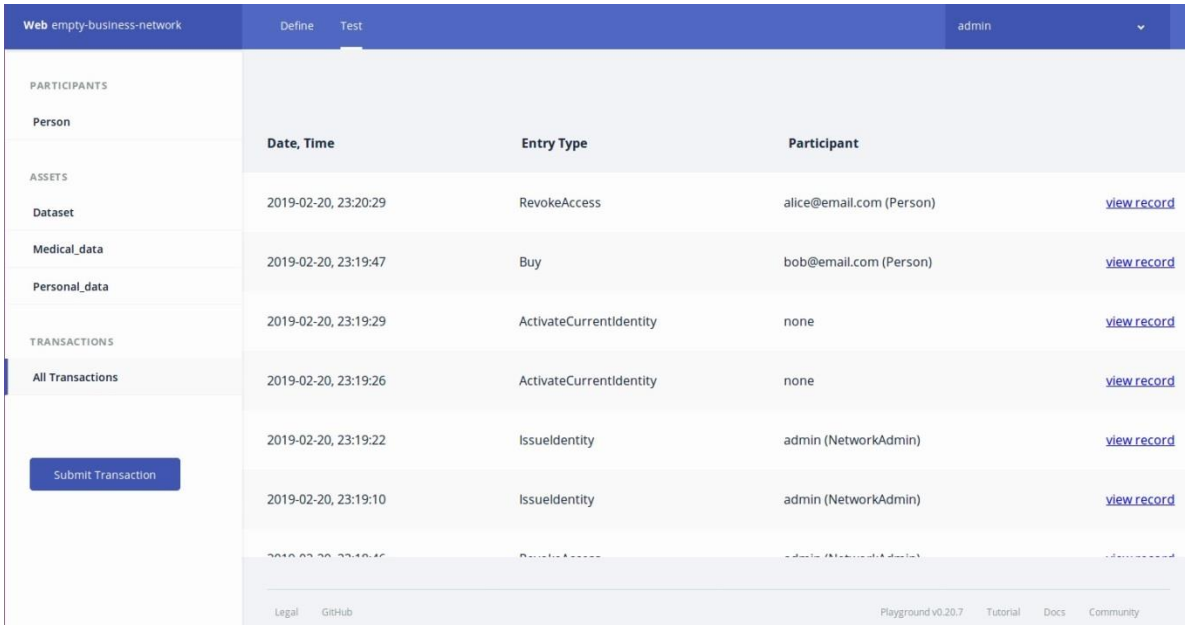

Figura 6-13: Lista de todas las transacciones realizadas en la red.

6.3.2. Iteración 1: Compilación y primera prueba automática

La presente iteración presenta la compilación y despliegue del contrato en redes locales para realizar pruebas automáticas que garanticen su correcto funcionamiento.

Para desplegar la red localmente, la primera tarea que se realiza es la ejecución de Hyperledger Fabric.

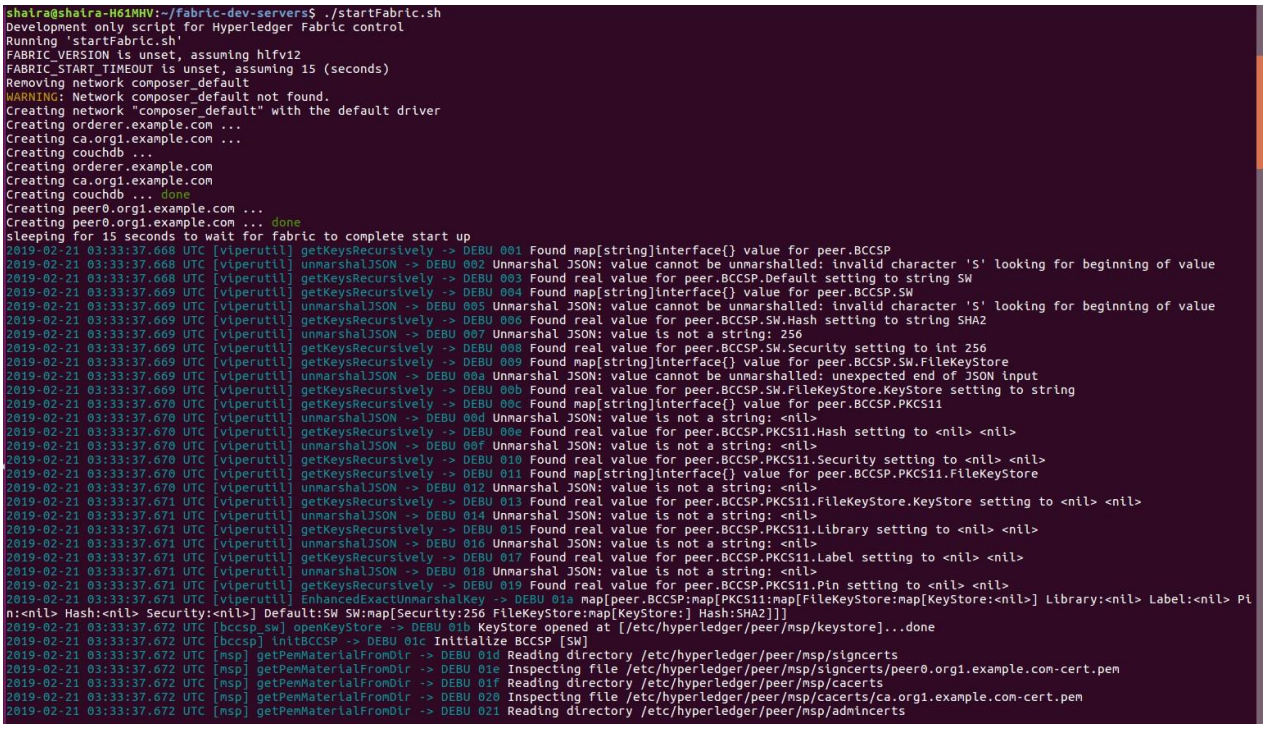

Figura 6-14: Ejecución de Hyperledger Fabric localmente.

Luego que Fabric está corriendo, se crean los diferentes Peers de la organización.

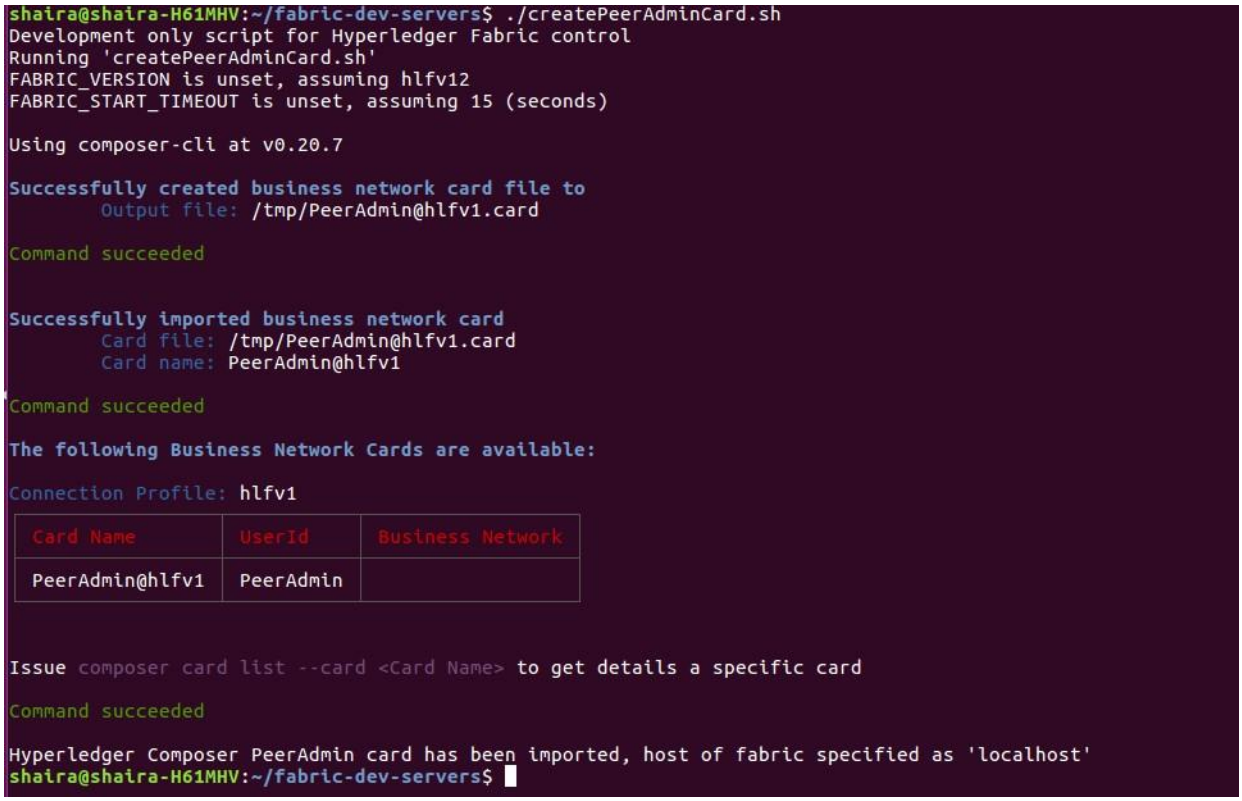

Figura 6-15: Creación de Peers para la organización.

Una vez Fabric ya está ejecutándose y configurado se procede a la creación de la definición de la red. La forma más fácil de comenzar es usar el generador de Yeoman para crear el esqueleto de una red empresarial. Esto creará un directorio que contiene todos los componentes de una red de negocios.

```
shaira@shaira-H61MHV:~/fabric-dev-servers$ yo hyperledger-composer:businessnetwo
rk
Welcome to the business network generator
? Business network name: data-exchange
? Description: Red de compra y venta de datos
? Author name:  Shaira Pérez
? Author email: shairaperez.sp@gmail.com
? License: Apache-2.0
? Namespace: org.redv2
? Do you want to generate an empty template network? Yes: generate an empty temp
late network
   create package.json
   create README.md
   create models/org.redv2.cto
   create permissions.acl
   create .eslintrc.yml
shaira@shaira-H61MHV:~/fabric-dev-servers$
```

Figura 6-16: Generar plantilla de red de negocio.

Se puede generar una red de ejemplo o una vacía, para los efectos de este desarrollo se genera una plantilla de red vacía.

Para lograr esta tarea, se maneja la primera versión de la red contraída en la iteración 0, por lo que se actualizan todos los archivos con las definiciones realizadas en la iteración anterior.

Ahora que la red empresarial se ha definido, se debe empaquetar en un archivo de archivo de red empresarial desplegable (.bna). Para esta tarea se utiliza un comando de composer.

```
shaira@shaira-H61MHV:~/fabric-dev-servers/data-exchange$ composer archive create -t dir -n .
Creating Business Network Archive


Looking for package.json of Business Network Definition
        Input directory: /home/shaira/fabric-dev-servers/data-exchange

Found:
        Description: Red de compra y venta de datos
        Name: redv2
        Identifier: redv2@0.0.1

Written Business Network Definition Archive file to
        Output file: redv2@0.0.1.bna

Command succeeded
```

Figura 6-17: Ejecución de comando para generar archivo .bna

Después de crear el archivo .bna, la red empresarial se puede implementar en la instancia de Hyperledger Fabric. Normalmente, la información del administrador de Fabric se requiere para crear una identidad PeerAdmin, con privilegios para instalar chaincode en el par, así como para iniciar chaincode en el canal composerchannel. Sin embargo, como parte de la instalación del entorno de desarrollo, ya se ha creado una identidad PeerAdmin.

Para las mejores prácticas, se debe crear una nueva identidad para administrar la red de

negocios después de la implementación. Esta identidad se conoce como un administrador de red.

La implementación de una red de negocios en el Hyperledger Fabric requiere que la red de negocios de Hyperledger Composer se instale en el *peer*, luego se puede iniciar la red de negocios y se debe crear una nueva tarjeta de participante, identificada y asociada para que sea el administrador de la red. Finalmente, la tarjeta de red empresarial del administrador de red debe importarse para su uso, y luego se puede hacer ping a la red para verificar que está respondiendo.

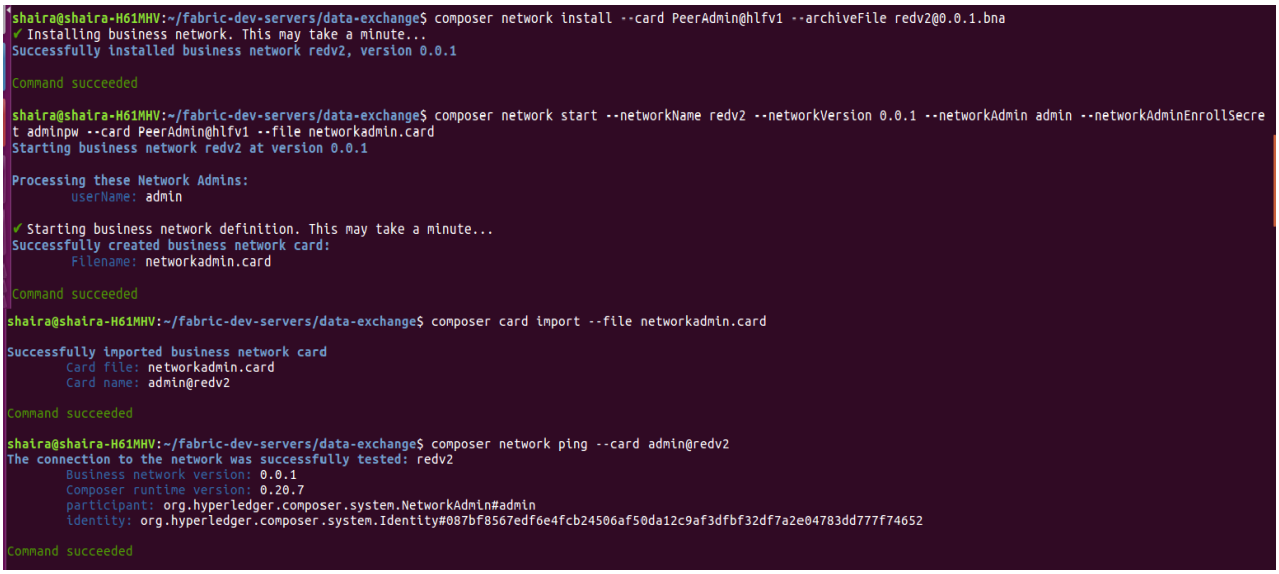

Figura 6-18: Despliegue de la red en Fabric

6.3.2.1. Prueba

Tipo de prueba	Funcional
Entorno de ejecución	Ubuntu 16.04 LTS
Ejecutante	Desarrollador de la herramienta
Herramienta usada	Mocha
Veces ejecutada	1
Resultados	100% de los requerimientos estipulados

Tabla 6-2: Especificaciones de la prueba automática para la versión 0.1

Es importante acotar la información con respecto al equipo de hardware utilizado para realizar dicha prueba, el cual es:

Componente	Especificación
Sistema Operativo	Ubuntu 16.04 LTS
Memoria	4GB
Procesador	Intel Core i5-3470 CPU @ 3.20GHz x 4
Disco Duro	500GB

Tabla 6-3: Especificaciones de hardware para pruebas en Mocha

Una vez definidas las pruebas para la red, se hace uso del comando 'npm test', provisto por Mocha en el terminal para ejecutar el código anterior y que se realicen las pruebas de manera automática:

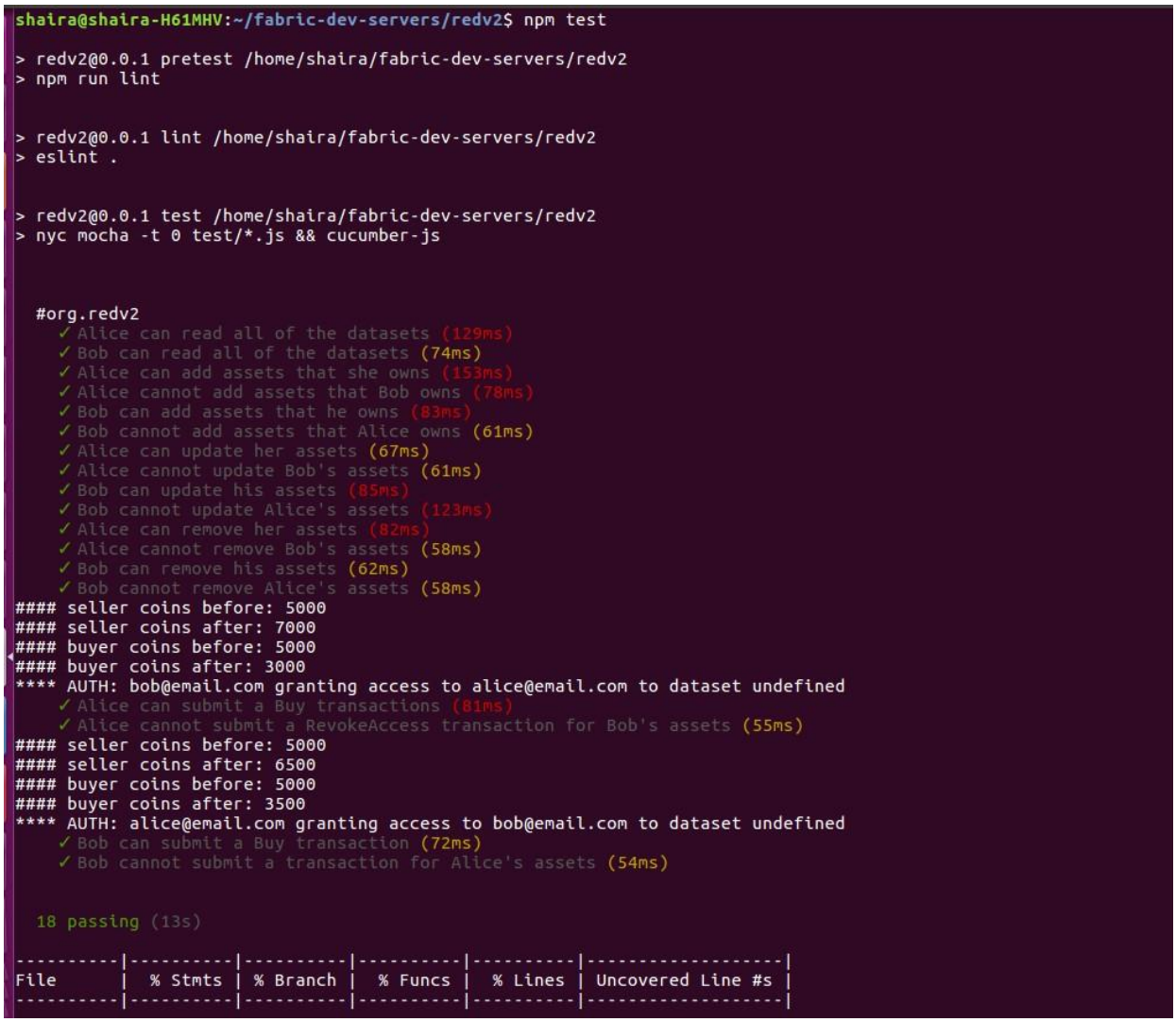

Figura 6-19: Ejecución de pruebas automáticas mediante el uso de la biblioteca Mocha.

Como se pude observar en la Figura 6-19, las pruebas se realizaron de manera exitosa, se logró hacer el despliegue de la definición de la red empresarial en una red local Fabric y se validó el correcto funcionamiento.

6.3.3. Iteración 2: Generación de API REST

Hyperledger Composer puede generar una API REST a medida basada en una red empresarial. Para desarrollar una aplicación web, la API REST proporciona una capa útil de abstracción independiente del lenguaje.

```
shaira@shaira-H61MHV:~/fabric-dev-servers/data-exchange$ composer-rest-server
? Enter the name of the business network card to use: admin@redv2
? Specify if you want namespaces in the generated REST API: never use namespaces


? Specify if you want to use an API key to secure the REST API: No
? Specify if you want to enable authentication for the REST API using Passport:
No
? Specify if you want to enable the explorer test interface: Yes
? Specify a key if you want to enable dynamic logging: blockchain
? Specify if you want to enable event publication over WebSockets: Yes
? Specify if you want to enable TLS security for the REST API: No

To restart the REST server using the same options, issue the following command:
   composer-rest-server -c admin@redv2 -n never -u true -d blockchain -w true

Discovering types from business network definition ...
Discovering the Returning Transactions..
Discovered types from business network definition
Generating schemas for all types in business network definition ...
Generated schemas for all types in business network definition
```

Figura 6-20: Generación de API REST a partir de la definición de la red.

Para esta primera versión del API no se habilitó la autenticación ni el acceso a múltiples usuarios, esto se agregará en iteraciones siguientes.

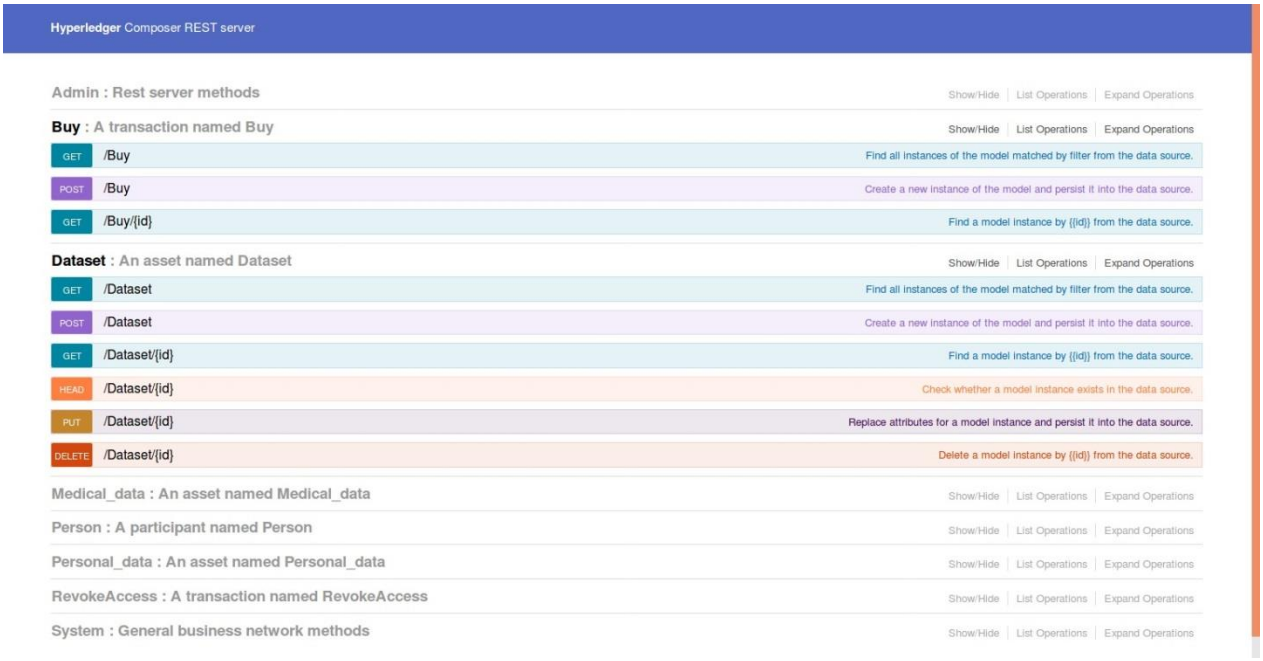

Figura 6-21: Explorador web del API.

6.3.5. Iteración 3: Construcción cliente web

La iteración 3 presenta el desarrollo que permite a un cliente web interactuar con diferentes instancias de la red de negocio, este desarrollo presenta una serie de pasos previos que serán explicados en detalle a medida que se vaya avanzando en el despliegue en la red, nótese que se hará uso de la API generada en la Iteración 2. El cliente web se encargará de generar los componentes de la red de negocio, así como las transacciones que constituyen los contratos inteligentes.

6.3.5.1. Cliente web

La construcción de la cliente web se realiza utilizando el framework Django, tal cual como fue especificado en el Capítulo 4 de la presente investigación. El mismo presenta una estructura de directorios integrada de la siguiente manera:

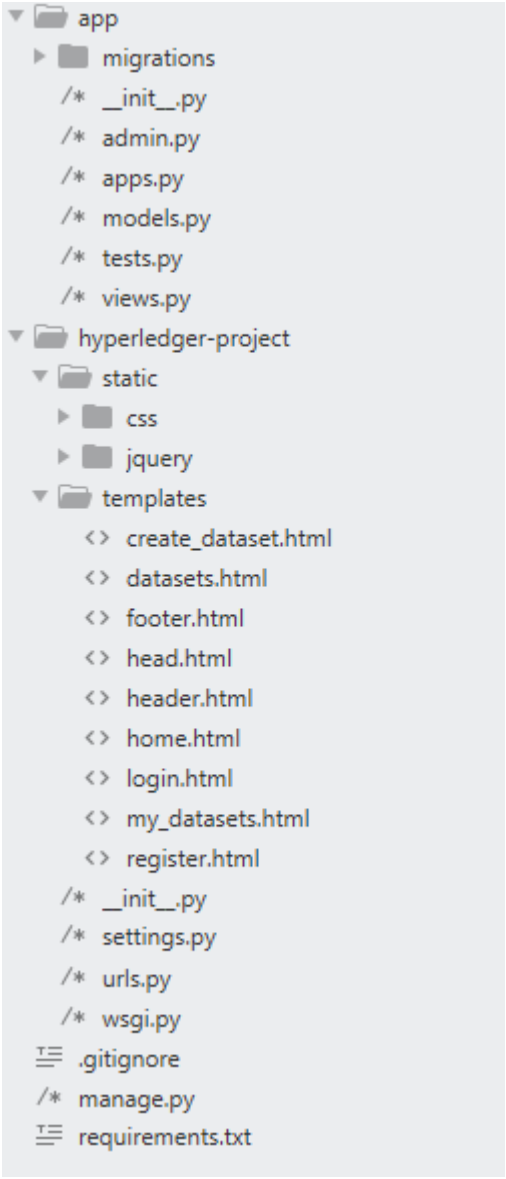

Figura 6-22: Directorio del proyecto para la aplicación web

Los diferentes directorios están organizados de la siguiente manera:

- En interior del directorio hyperledger-project/ es el propio paquete de Python para el proyecto. Su nombre es el nombre del paquete de Python a utilizar para importar todo dentro de este (por ejemplo, hyperledger-project.urls).

- **hyperledger-project/__init__.py**: Un archivo vacío que le indica a Python que este directorio debería ser considerado como un paquete Python.
- **hyperledger-project/settings.py**: Ajustes/configuración para este proyecto Django.
- **hyperledger-project/urls.py**: Las declaraciones URL para este proyecto Django; una «tabla de contenidos» del sitio basado en Django.
- **hyperledger-project/wsgi.py**: Un punto de entrada para que los servidores web compatibles con WSGI puedan servir su proyecto.
- **hyperledger-project/templates:** Contiene todas las vistas ("*templates*") del cliente web.
- **hyperledger-project/static:** Contiene todos los archivos estáticos del proyecto como archivos de estilo (CSS) y JQuery.
- **app/view.py:** Una vista es una función de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP. Las vistas acceden a los datos que necesitan para satisfacer las peticiones por medio de *modelos*, y delegan el formateo de la respuesta a las *plantillas* (*"templates"*).
- **app/model.py:** En este archivo se definen los modelos de la aplicación. Los modelos son objetos de Python que definen la estructura de los datos de una aplicación y proporcionan mecanismos para gestionar (añadir, modificar y borrar) y consultar registros en la base de datos.

Una vez descrita la estructura de los directorios de la aplicación web, se comprueba que el proyecto Django funciona, utilizando el archivo manage.py, este archivo lo crea Django al crear el proyecto:

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'tesis.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()

```

Algoritmo 6-5: Módulo que inicia en una red local la aplicación web (manage.py)

Una vez corriendo el servidor, se configura el manejo de rutas mediante el siguiente módulo:

```python
from django.conf.urls import include, url
from hyperledger import views
from django.contrib.auth import views as auth_views

urlpatterns = [
    #Index
    url(r'^$', views.home),
    url(r'home$', views.home, name='home'),
    url(r'login$', views.login, name='login'),
    url(r'logout$', views.logout, name='logout'),
    url(r'registro$', views.create_user, name='registro'),
    url(r'crear-dataset$', views.create_dataset, name='crear-dataset'),
    url(r'filtrar-datasets$', views.filter_datasets, name='filtrar-datasets'),
    url(r'comprar-dataset$', views.buy_dataset, name='comprar-dataset'),
    url(r'mis-datasets$', views.my_datasets, name='mis-datasets'),
]
```

Algoritmo 6-6: Módulo que configura las rutas del servidor web (urls.py)

Además se realiza la configuración de las bases de datos a utilizar dentro del archivo settings.py:

```python
# Database
# https://docs.djangoproject.com/en/2.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'hyperledger',
        'USER': 'postgres',
        'PASSWORD': '08012018',
        'HOST': 'tesis.cojlwmboclia.us-east-2.rds.amazonaws.com',
        'PORT': '5432'
    }
}

USERNAME_MONGODB = 'admin'
PASSWORD_MONGODB = '08012018'

```

Algoritmo 6-7: Sección que define la conexión a las base de datos dentro del módulo de configuración del proyecto (settings.py)

6.3.5.2. Elementos de la interfaz de usuario

Siguiendo la práctica de reutilización del código, se crean las siguientes plantillas que formarán parte de todas las vistas.

```
{% load staticfiles %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Intercambio de datos</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="
    sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/
    X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
    <script src="{% static "jquery/dist/jquery.min.js" %}"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="
    sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="
    sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <link href="{% static "css/default.css" %}" rel="stylesheet">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.1/css/all.css" integrity="
    sha384-50oBUHEmvpQ+1lW4y57PTFmhCaXp0ML5d60M1M7uH2+nqUivzIebhndOJK28anvf" crossorigin="anonymous">
    <link href="https://fonts.googleapis.com/css?family=Raleway" rel="stylesheet">
    <script src="{% static "js/buy.js" %}"></script>
</head>
<body>
```

Algoritmo 6-8: HTML HEAD, configuración de los metadatos (head.html)

```
{% load staticfiles %}
<header>
    <nav class="navbar navbar-expand-md navbar-dark fixed-top">
        <div class="container">
            <!-- Navbar links -->
            <div class="collapse navbar-collapse" id="collapsibleNavbar">
                <ul class="navbar-nav ml-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="{% url 'home' %}">Market</a>
                    </li>
                    {% if not is_user %}
                    <li class="nav-item">
                        <a class="nav-link" href="{% url 'login' %}">Iniciar sesión</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link btn btn-secondary" href="{% url 'registro' %}">Registrarse</a>
                    </li>
                    {% endif %}
                    {% if is_user %}
                    <li class="nav-item">
                        <a class="nav-link" href="{% url 'mis-datasets' %}">Mis Datasets</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{% url 'logout' %}">Desconectarse</a>
                    </li>
                    {% endif %}
                </ul>
            </div>
        </div>
    </nav>
</header>
```

Algoritmo 6-9: Plantilla que configura el menú de navegación en la aplicación web (header.html)

```
<footer>
    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12 mt-2 mt-sm-2 text-center text-white">
            <p>Shaira Pérez</p>
            <p class="h6">&copy All right Reversed.</p>
        </div>
        </hr>
    </div>
</footer>
</body>
</html>
```

Algoritmo 6-10: Plantilla que configura el pie de la página (footer.html)

6.3.5.3. Servicio de conexión al API

El acceso a la red Blockchain se realiza a través de una API REST como se explica en el capítulo 4, es por ello que se crean varias funciones que permiten interactuar con la misma de manera sencilla.

```
503     """Funciones para trabajar con el API del Blockchain.
504         La URL de acceso al API se compone de la IP del servidor, el puerto de Loopback: 3000 y el endpoint /api/
505         El header es el mismo para todas las solicitudes, indica que la respuesta del API es un JSON.
506     """
507     def get_all_records(ns):
508         """Obtener todos los registros de un modelo."""
509         url = "http://23.23.61.193:3000/api/"+ns
510         headers = {"Accept": "application/json",
511                    "Content-Type": "application/json"}
512         response = requests.get(url)
513         if response.ok:
514             data = json.loads(response.text)
515             return data, True
516         else:
517             return [], False
518 
519     def get_single_record(ns, id):
520         """Obtener un registro especifico de un modelo"""
521         url = "http://23.23.61.193:3000/api/"+ns+"/"+id
522         headers = {"Accept": "application/json",
523                    "Content-Type": "application/json"}
524         response = requests.get(url)
525         if response.ok:
526             data = json.loads(response.text)
527             return data, True
528         else:
529             return [], False
530 
531     def add_record(ns, asset):
532         """Agregar un registro a un modelo"""
533         url = "http://23.23.61.193:3000/api/"+ns
534         headers = {"Accept": "application/json",
535                    "Content-Type": "application/json"}
536         response = requests.post(url, data=asset)
537         if response.ok:
538             return True
539         else:
540             return False
541 
542     def update_record(ns, id, item):
543         url = "http://23.23.61.193:3000/api/"+ns+"/"+id
544         headers = {"Accept": "application/json",
545                    "Content-Type": "application/json"}
546         response = requests.put(url, data=item)
547         if response.ok:
548             return True
549         else:
550             return False
551 
552     def delete_record(ns, id):
553         url = "http://23.23.61.193:3000/api/"+ns+"/"+id
554         headers = {"Accept": "application/json",
555                    "Content-Type": "application/json"}
556         response = requests.delete(url)
557         if response.ok:
558             return True
559         else:
560             return False
```

Algoritmo 6-11: Funciones de interacción con el API (views.py)

6.3.5.4. Creación de un participante

Para esta acción se provee un formulario que permite al usuario registrar sus datos de identificación dentro de la red (correo electrónico). Este formulario tiene la siguiente plantilla:

```
<!-- Cargar metadatos -->
{% include 'head.html' %}
<div class="login-register">
  <div class="container">
    <div id="login-box" class="col-md-12">
        <!-- Mostrar mensajes de respuesta del controlador en caso de que hayan errores -->
        {% if messages %}
        <div class="alert alert-danger alert-dismissable">
            <button type="button" class="close" data-dismiss="alert" aria-hidden="true"> &times;</button>
            <ul class="messages">
                {% for message in messages %}
                    <p>{{message}}</p>
                {% endfor %}
            </ul>
       </div>
        {% endif %}
        <!-- Formulario de registro -->
        <form class="form" id="loginform" method="POST"> {% csrf_token %}
            <h3 class="text-center text-info">Registrarse</h3>
            <div class="form-group">
                <label for="email" class="text-info">Email:</label><br>
                <input name="email" type="text" class="form-control" placeholder="Correo" required>
            </div>
            <div class="form-group">
                <label for="password" class="text-info">Password:</label><br>
                <input name="password" type="password" class="form-control" placeholder="Contraseña">
            </div>
            <div align="center" class="form-group">
              <button type="submit" class="btn btn-info">Confirmar</button>
            </div>
        </form>
    </div>
  </div>
</div>

</body>
</html>
```

Algoritmo 6-12: Plantilla de registro de un participante en la red (register.html)

El controlador recibe la solicitud y procede a resolver la misma. La creación de un participante se divide en 2 tareas: crear el participante en la red de negocios y crear un nuevo usuario en la base de datos de la aplicación. Esta función sigue la siguiente estructura:

```
def create_user(request):
    """Crear un nuevo usuario en la aplicación:
        Crear partipante en Blockchain.
        Crear usuario en PostgreSQL.
    """
    if request.method == "POST":
        """Recibir datos del formulario de registro"""
        email = request.POST['email']
        password = make_password(request.POST['password'])
        """Verificar que el participante no existe
        La función get_single_record recibe como parámetros:
        ns = el nombre del modelo
        id = el ID a buscar en dicho modelo
        En el caso del modelo Person, el ID es el email"""
        resonse, status_code = get_single_record(ns= "Person", id=email)
        if status_code:
            messages.error(
            request,
            "El email ya existe",
            fail_silently=True
            )
            return redirect(create_user)
```

```
212             else:
213                 """Crear participante: Person en Blockchain"""
214                 participant = {
215                   "$class": "org.redv2.Person",
216                   "email": email,
217                   "coins": 5000
218                 }
219                 response = add_record("Person", participant)
220                 if response:
221                     """Crear usuario en PostgreSQL"""
222                     person = AuthUser(
223                             username= email,
224                             password= password,
225                             last_login= datetime.now(),
226                             is_superuser= True,
227                             first_name= '',
228                             last_name= '',
229                             email= request.POST['email'],
230                             is_staff= True,
231                             is_active= True,
232                             date_joined= datetime.now(),
233                         )
234                     person.save()
235                     """Iniciar sesión en la app"""
236                     a = EmailAuthBackend()
237                     user = a.authenticate(
238                             username = email,
239                             password = request.POST['password']
240                         )
241                     if user is not None:
242                         auth_login(request, user)
243                         return redirect(home)
244                     """Registrar transacción en MongoDB"""
245                     """Crear conexión con servidor de MopngoDB"""
246                     client = MongoClient('mongodb://%s:%s@54.89.16.99' % (settings.USERNAME_MONGODB, settings.PASSWORD_MONGODB))
247                     """Conectarse con base de datos"""
248                     db = client["blockchain"]
249                     """Buscar las transacciones en el Blockchain para registrar las transacciones en MongoDB"""
250                     response, status_code = get_all_records("system/historian")
251                     """Definir colección a utilizar"""
252                     transactions = db["transaction"]
253                     """Registrar la última transacción realizads: Comprar"""
254                     for transaction in response[:2]:
255                         """Agregar registro en la colección transacción"""
256                         transaction = {
257                                 "class": transaction['$class'],
258                                 "transactionId": transaction['transactionId'],
259                                 "transactionType": transaction['transactionType'],
260                                 "transactionInvoked": transaction['transactionInvoked'],
261                                 "participantInvoking": transaction['participantInvoking'],
262                                 "identityUsed": transaction['identityUsed'],
263                                 "eventsEmitted": transaction['eventsEmitted'],
264                                 "transactionTimestamp": transaction['transactionTimestamp']
265                         }
266                         transactions.insert_one(transaction)
267                 else:
268                     messages.error(
269                             request,
270                             "No se pudo crear el usuario vuelva a intentarlo",
271                             fail_silently=True
272                             )
273                     return redirect(create_user)
274         else:
275             return render(request, 'register.html', {})
276
```

Algoritmo 6-13: Función que crea un participante dentro de la red y un usuario en la base de datos que permite entrar y salir de la aplicación (views.py)

6.3.5.5. Creación de datasets

Para la creación de un dataset se provee al usuario de un formulario que le permite ingresar todos los datos que desea vender con respecto al dataset que está por crear.

La estructura de la plantilla es la siguiente:

```
<!-- Cargar HEAD y HEADER -->
{% include 'head.html' %}
{% include 'header.html' %}
<div class="page-wrapper" id="page-wrapper">
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <div class="white-box">
                <!-- Mostrar mensajes para indicar si la transacción fue exitosa o no -->
                {% if messages %}
                <div class="alert alert-danger alert-dismissable">
                    <button type="button" class="close" data-dismiss="alert" aria-hidden="true"> &times;</button>
                    <ul class="messages">
                        {% for message in messages %}
                            <p>{{message}}</p>
                        {% endfor %}
                    </ul>
                </div>
                {% endif %}
                <!-- Formulario de registro -->
                <form class="form" id="loginform" method="POST"> {% csrf_token %}
                    <h3 class="text-center text-info">Crear dataset</h3>
                    <div class="row">
                        <div class="col-lg-6">
                            <div class="form-group">
                                <label for="first_name" class="text-info">Nombre:</label><br>
                                <input name="first_name" type="text" class="form-control" placeholder="Nombre" required>
                            </div>
                        </div>
                        <div class="col-lg-6">
                            <div class="form-group">
                                <label for="last_name" class="text-info">Apellido:</label><br>
                                <input name="last_name" type="text" class="form-control" placeholder="Apellido" required>
                            </div>
                        </div>
                    </div>
                    <div class="row">
                        <div class="col-lg-4">
                            <div class="form-group">
                                <label for="birth_date" class="text-info">Fecha de nacimiento:</label><br>
                                <input name="birth_date" type="date" class="form-control" required>
                            </div>
                        </div>
                        <div class="col-lg-8">
                            <label class="text-info">Sexo:</label><br>
                            <div class="form-check form-check-inline">
                                <input class="form-check-input" type="radio" name="sex" id="female" value="FEMENINO" checked>
                                <label class="form-check-label" for="female">FEMENINO</label>
                            </div>
                            <div class="form-check form-check-inline">
                                <input class="form-check-input" type="radio" name="sex" id="male" value="MASCULINO" checked>
                                <label class="form-check-label" for="male">MASCULINO</label>
                            </div>
                        </div>
                    </div>
                    <div class="row">
                        <div class="col-lg-3">
                            <div class="form-group">
                                <label for="height" class="text-info">Estatura (cm):</label><br>
                                <input name="height" type="number" class="form-control" min="0" required>
                            </div>
                        </div>
                        <div class="col-lg-2">
                            <div class="form-group">
                                <label for="weight" class="text-info">Peso:</label><br>
                                <input name="weight" type="number" class="form-control" min="0" required>
                            </div>
                        </div>
                        <div class="col-lg-3">
                            <div class="form-group">
                                <label for="blood_pressure" class="text-info">Presión arterial:</label><br>
                                <input name="blood_pressure" type="number" class="form-control" min="0" required>
                            </div>
                        </div>
                        <div class="col-lg-4">
                            <div class="form-group">
                                <label for="glucose_level" class="text-info">Nivel de Glucosa:</label><br>
                                <input name="glucose_level" type="number" class="form-control" min="0" required>
                            </div>
                        </div>
                    </div>
                    <div class="row">
                        <div class="col-lg-6">
                            <div class="form-group">
                                <label for="origin_country" class="text-info">País de nacimiento:</label><br>
                                <input name="origin_country" type="text" class="form-control" placeholder="Venezuela" required>
                            </div>
                        </div>
```

```
                        </div>
                        <div class="col-lg-6">
                            <div class="form-group">
                                <label for="residence_country" class="text-info">País de residencia:</label><br>
                                <input name="residence_country" type="text" class="form-control" placeholder="Venezuela" required>
                            </div>
                        </div>
                    </div>
                    <div class="row">
                        <div class="col-lg-12">
                            <div class="form-group">
                                <label for="price" class="text-info">Precio:</label><br>
                                <input name="price" type="number" class="form-control" min="0" required>
                            </div>
                        </div>
                    </div>
                    <div align="center" class="form-group">
                      <button type="submit" class="btn btn-info">Confirmar</button>
                    </div>
                </form>
            </div>
          </div>
        </div>
    </div>
</div>


</body>
</html>
```

Algoritmo 6-14: Formulario de creación de un dataset (créate_dataset.html)

El dataset está compuesto por datos personales (Personal_data) y datos médicos (Medical_data) es por ello que antes de crear un dataset se crean estos activos primero. En caso que alguno de estos dos no pueda ser creado, el dataset tampoco se creará y se eliminarán todos los modelos que se hayan creado hasta el momento del error.

Para el posterior análisis de los datos, los datasets también son guardados dentro de la base de datos NoSQL.

```
def create_dataset(request):
    """Crear dataset"""
    if request.user.is_authenticated:
        if request.method == "POST":
            """Recibir datos del formulario"""
            first_name = request.POST['first_name']
            last_name = request.POST['last_name']
            birth_date = request.POST['birth_date']
            sex = request.POST['sex']
            origin_country = request.POST['origin_country']
            height = request.POST['height']
            weight = request.POST['weight']
            blood_pressure = request.POST['blood_pressure']
            glucose_level = request.POST['glucose_level']
            residence_country = request.POST['residence_country']
            price = request.POST['price']
            email = request.user.email
            owner = "org.redv2.Person#" + email
            """Crear modelo: Personal_data"""
            personal_data_id = random.randint(0, 99999999)
            personal_data = {
              "$class": "org.redv2.Personal_data",
              "dataId": personal_data_id,
              "firstName": first_name,
              "lastName": last_name,
              "birth_date": birth_date,
              "gender": sex,
              "origin_country": origin_country,
              "authorized": [],
              "owner": owner
            }
```

```
            """Registrar modelo en el Blockchain"""
            response = add_record("Personal_data",personal_data)
            if response:
                """Crear modelo: Medical_data"""
                medical_data_id = random.randint(0, 99999999)
                medical_data = {
                  "$class": "org.redv2.Medical_data",
                  "dataId": medical_data_id,
                  "height": height,
                  "weight": weight,
                  "blood_pressure": blood_pressure,
                  "glucose_level": glucose_level,
                  "residence_country": residence_country,
                  "authorized": [],
                  "owner": owner
                }
            """Registrar modelo en el Blockchain"""
            response = add_record("Medical_data",medical_data)
            if response:
                """Registrar Dataset"""
                dataset = {
                  "$class": "org.redv2.Dataset",
                  "datasetId": str(random.randint(0, 99999999)),
                  "price": int(price),
                  "description": "DATOS_PERSONALES_Y_DATOS_MEDICOS",
                  "owner": owner,
                  "perData": "org.redv2.Personal_data#" + str(personal_data_id),
                  "medData": "org.redv2.Medical_data#" + str(medical_data_id)
                }
                """Registrar modelo en el Blockchain"""
                response = add_record("Dataset",dataset)
                if response:
                    """Registrar datos en MongoDB"""
                    """Crear conexión con servidor de MopngoDB"""
                    client = MongoClient('mongodb://%s:%s@54.89.16.99' % (settings.USERNAME_MONGODB, settings.PASSWORD_MONGODB))
                    """Conectarse con base de datos"""
                    db = client["blockchain"]
                    """Buscar las transacciones en el Blockchain para registrar las transacciones en MongoDB"""
                    response, status_code = get_all_records("system/historian")
                    """Definir colección a utilizar"""
                    transactions = db["transaction"]
                    """Registrar las últimas 3 realizadas: Agregar Persona_data, Medical_data y Dataset"""
                    for transaction in response[:3]:
                        """Agregar registro en la colección transacción"""
                        transaction = {
                                "class": transaction['$class'],
                                "transactionId": transaction['transactionId'],
                                "transactionType": transaction['transactionType'],
                                "transactionInvoked": transaction['transactionInvoked'],
                                "participantInvoking": transaction['participantInvoking'],
                                "identityUsed": transaction['identityUsed'],
                                "eventsEmitted": transaction['eventsEmitted'],
                                "transactionTimestamp": transaction['transactionTimestamp']
                        }
                        transactions.insert_one(transaction)
                    """Registrar Dataset en MongoDB"""
                    """Definir colección a utilizar"""
                    datasets = db["dataset"]
                    dataset = {
                        "owner": email,
                        "dataset": [
                                    {
                                        "type": "personal_data",
                                        "first_name": first_name,
                                        "last_name": last_name,
                                        "birth_date": birth_date,
                                        "sex": sex,
                                        "origin_country": origin_country
                                    },
                                    {
                                        "type": "medical_data",
                                        "height": height,
                                        "weight": weight,
                                        "blood_pressure": blood_pressure,
                                        "glucose_level": glucose_level,
                                        "residence_country": residence_country
                                    }
                                ]
                            }
```

```
                            """Agregar dataset en la colección"""
                            datasets.insert_one(dataset)
                            return redirect(home)
                        else:
                            """Error, no se creó el registro en Dataset"""
                            messages.error(
                            request,
                            "No se pudo crear el dataset vuelva a intentarlo",
                            fail_silently=True
                            )
                            """Eliminar el registro en Personal_data"""
                            response = delete_record("Personal_data",email)
                            """Eliminar el registro en Medical_data"""
                            response = delete_record("Medical_data",email)
                            return redirect(create_dataset)
                    else:
                        """Error, no se creó el registro en Medical_data"""
                        messages.error(
                            request,
                            "No se pudo crear el dataset vuelva a intentarlo",
                            fail_silently=True
                            )
                        """Eliminar el registro en Personal_data"""
                        response = delete_record("Personal_data",email)
                        return redirect(create_dataset)
                else:
                    """Error, no se creó el registro en Personal_data"""
                    messages.error(
                            request,
                            "No se pudo crear el dataset vuelva a intentarlo",
                            fail_silently=True
                            )
                    return redirect(create_dataset)
            else:
                return render(request, 'create_dataset.html', {'user': request.user, 'is_user': True})
        else:
            return redirect(login)

```

Algoritmo 6-15: Función que crea un dataset (views.py)

6.3.5.6. Visualizar los detalles de datasets

Cada vez que se crea un dataset, la vista de inicio (home) se recarga para mostrar la lista de datasets disponibles. Sólo los dueños y compradores autorizados pueden ver los datos completos de los datasets, los demás usuarios verán sólo la parte general de los mismos (ID, dueño y precio), el resto de los datos se verán borrosos, ya que no han sido comprados. Dicha vista se divide en dos plantillas: *home.html* que es la página inicial de la aplicación, por lo que contiene una sección de bienvenida además de la lista de datasets, y la plantilla *datasets.html* la cual se encarga del listado. Estas tienen la siguiente composición:

```
<!-- Cargar HEAD y HEADER -->
{% include 'head.html' %}
{% include 'header.html' %}
    <!-- Sección de bienvenida -->
    <div class="banner" id="banner">
        <div class="container" style="margin-left: 50px;">
            <div class="col-lg-6 col-md-6" style="padding:200px 0 150px 0;">
                <h1>Mercado de datos impulsado por Blockchain</h1>
                <h5> Transacciones seguras basadas en blockchain y contratos inteligentes automatizados para vender y comprar datasets.</h5
                >
                {% if not is_user %}
                <a class="btn btn-info" style="margin-top:15px" href="{% url 'registro' %}">Registrarse</a>
                {% endif %}
            </div>
        </div>
    </div>
    <!-- Listar datasets -->
    {% include 'datasets.html' %}
    <!-- Pie de página -->
{% include 'footer.html' %}
```

Algoritmo 6-16: Plantilla de vista de inicio (home.html)

```
<div class="container" id="datasets">
    <div class="row">
        <div class="col-md-12">
            {% if datasets_size > 0 %}
            <scroll-container class="scroll-container">
                <table class="table" id="table">
                    <thead>
                        <tr>
                            <th></th>
                            <th>ID</th>
                            <th>Precio</th>
                            <th>Dueño</th>
                            <th>Nombre</th>
                            <th>Apellido</th>
                            <th>Fecha de nacimiento</th>
                            <th>Sexo</th>
                            <th>País de origen</th>
                            <th>Estatura</th>
                            <th>Peso</th>
                            <th>Presión arterial</th>
                            <th>Nivel de glucosa</th>
                            <th>País de residencia</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for dataset in datasets%}
                            <tr>
                                {% if not user.email in dataset.personal_data.authorized and not user.email == dataset.owner%}
                                <td><input type="checkbox" class="check" align="center" id="{{dataset.datasetId}}" value="
                                {{dataset.datasetId}}"></td>
                                {% else %}
                                <td></td>
                                {% endif %}
                                <td>{{dataset.datasetId}}</td>
                                <td>{{dataset.price}}</td>
                                <td>{{dataset.owner}}</td>
                                {% if user.email in dataset.personal_data.authorized or user.email == dataset.owner%}
                                    <td>{{dataset.personal_data.firstName}}</td>
                                    <td>{{dataset.personal_data.lastName}}</td>
                                    <td>{{dataset.personal_data.age}}</td>
                                    <td>{{dataset.personal_data.gender}}</td>
                                    <td>{{dataset.personal_data.origin_country}}</td>
                                    <td>{{dataset.medical_data.height}}</td>
                                    <td>{{dataset.medical_data.weight}}</td>
                                    <td>{{dataset.medical_data.blood_pressure}}</td>
                                    <td>{{dataset.medical_data.glucose_level}}</td>
                                    <td>{{dataset.medical_data.residence_country}}</td>
                                {% else %}
                                    <td class="borroso">XXXXX</td>
                                    <td class="borroso">XXXXX</td>
                                    <td class="borroso">XXXXX</td>
                                    <td class="borroso">XXXXX</td>
                                    <td class="borroso">XXXXX</td>
                                    <td class="borroso">XXXXX</td>
                                    <td class="borroso">XXXXX</td>
                                    <td class="borroso">XXXXX</td>
                                    <td class="borroso">XXXXX</td>
                                {% endif %}
                            </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </scroll-container>
            {% else %}
            <div class="col-lg-12">
                <h3>No hay datasets con esos parámetros de búsqueda</h3>
            </div>
            {% endif %}
        </div>
    </div>

```

Algoritmo 6-17: Plantilla que lista los datasets disponibles (datasets.html)

```python
def home(request):
    """Cargar página de bienvenida junto con la lista de datasets"""
    """Consultar al API para obtener todos los datasets registrados en la red"""
    records, status_code = get_all_records("Dataset")
    """Si la respuesta es exitosa se estructura la respuesta en la variable dataset"""
    if status_code:
        """Lista de datasets"""
        datasets = []
        for record in records:
            dataset = {}
            dataset['datasetId'] = record['datasetId']
            dataset['owner'] = record['owner'].replace('resource:org.redv2.Person#','')
            dataset['price'] = record['price']
            dataset['personal_data'], status_code = get_single_record("Personal_data", record['perData'].split('#')[1])
            """La fecha manejada por la red es con zona horaria, por lo que para trabajar comodamente se pasa a un formato AÑO-MES-DIA"""
            birth_date = dataset['personal_data']['birth_date'].split('T')[0]
            birth_date = datetime.strptime(birth_date, '%Y-%m-%d')
            dataset['personal_data']['birth_date'] = birth_date
            """A partir de la fecha de nacimiento se calcula el dato edad"""
            dataset['personal_data']['age'] = datetime.today().year - birth_date.year
            dataset['medical_data'], status_code = get_single_record("Medical_data", record['medData'].split('#')[1])
            """Se agrega el dataset a la lista"""
            datasets.append(dataset)
    if not status_code:
        """En caso de una respuesta errónea por parte del API no se lista ningún dataset"""
        datasets = []
    """Verificar si el usuario está autenticado, para así mostrarle los datos pertenecientes a sus datasets"""
    if request.user.is_authenticated:
        return render(request, 'home.html', {'user':request.user, 'is_user': True, 'datasets': datasets, 'datasets_size': len(datasets)})
    else:
        return render(request, 'home.html', {'is_user': False, 'datasets': datasets, 'datasets_size': len(datasets)})
```

Algoritmo 6-18: Función que lista todos los datasets registrados en la red (views.py)

6.3.5.7. Comprar un dataset

Desde la lista de datasets el usuario tiene la posibilidad de comprar un dataset. Esta compra es la ejecución de un contrato inteligente dentro de la cadena de bloques, y es este quien se encarga de ver el estado de cuenta del usuario, para chequear que tenga crédito disponible para la compra. La función comprar dentro del cliente web solo se encarga de enviar la solicitud al API para que el contrato sea desplegado.

```python
def buy_dataset(request):
    """Comprar dataset"""
    if request.user.is_authenticated:
        try:
            """
            Obtener lista de datasets a comprar:
            request.POST['datasets'] es un string con los Ids de los dataset separados por coma. Ejemplo:
            ,1234,5678
            """
            datasets = request.POST['datasets']
            """El string se transforma en un arreglo, como el string inicia con una ',' se omite el 1er elemento del string"""
            datasets = datasets[1:].split(',')
            print(datasets)

            """Se realiza la compra de cada uno de los datasets en el arreglo"""
            for dataset in datasets:
                """Definir transacción"""
                transaction = {
                      "$class": "org.redv2.Buy",
                      "dataset": "org.redv2.Dataset#" + dataset,
                      "buyer": "org.redv2.Person#" + request.user.email
                    }
                """Registrar transacción en el Blockchain"""
                response = add_record("Buy", transaction)
                if response:
                    print("success",dataset)
                    """Registrar transacción en MongoDB"""
                    """Crear conexión con servidor de MopngoDB"""
                    client = MongoClient('mongodb://%s:%s@54.89.16.99' % (settings.USERNAME_MONGODB, settings.PASSWORD_MONGODB))
                    """Conectarse con base de datos"""
                    db = client["blockchain"]
                    """Buscar las transacciones en el Blockchain para registrar las transacciones en MongoDB"""
                    response, status_code = get_all_records("system/historian")
                    """Ordenar por fecha de ejecución, desde la más reciente hasta la más vieja"""
                    response = sorted(response, key = lambda k:k['transactionTimestamp'], reverse=True)
                    """Definir colección a utilizar"""
                    transactions = db["transaction"]
```

```
                    """Registrar la última transacción realizads: Comprar"""
                    for transaction in response[:2]:
                        """Agregar registro en la colección transacción"""
                        if len(transaction['eventsEmitted']) > 0:
                            eventsEmitted = [{
                                        'class': transaction['eventsEmitted'][0]['$class'],
                                        'dataset': transaction['eventsEmitted'][0]['dataset'],
                                        'eventId': transaction['eventsEmitted'][0]['eventId'],
                                        'newAuthorized': transaction['eventsEmitted'][0]['newAuthorized'],
                                        'owner': transaction['eventsEmitted'][0]['owner'],
                                        'timestamp': transaction['eventsEmitted'][0]['timestamp']
                                        }]
                        else:
                            eventsEmitted = transaction['eventsEmitted']
                        transaction = {
                                "class": transaction['$class'],
                                "transactionId": transaction['transactionId'],
                                "transactionType": transaction['transactionType'],
                                "transactionInvoked": transaction['transactionInvoked'],
                                "participantInvoking": transaction['participantInvoking'],
                                "identityUsed": transaction['identityUsed'],
                                "eventsEmitted": eventsEmitted,
                                "transactionTimestamp": transaction['transactionTimestamp']
                        }
                        transactions.insert_one(transaction)
                else:
                    """Error, el comprador no pudo adquirir el dataset"""
                    messages.error(
                            request,
                            "Transacción denegada para el dataset " + dataset +", confirme que dispone de coins suficientes para
                            fail_silently=True
                            )
                    return redirect(home)
            """Si la transacción es exitosa se muestra un mensaje de exito al usuario y se redirige al inicio"""
            messages.success(
                    request,
                    "Transacción exitosa",
                    fail_silently=True
                    )
            return redirect(home)
        except:
            datasets = []
    else:
        return redirect(login)

```

Algoritmo 6-19: Función comprar dataset (views.py)

6.4. Iteración 4: Almacenamiento de interacciones

Mientras se desarrollaba el cliente web, se tomó en cuenta el almacenamiento de las interacciones en la base de datos mediante solicitudes HTTP, las interacciones a ser almacenadas son:

- Crear participante.
- Crear dataset.
    - Crear personal_data.
    - Crear medical_data
- Comprar dataset.

6.4.1. Servidor MongoDB

La instancia de MongoDB se desplegó en un servidor de AWS, y la conexión al mismo se realiza utilizando una librería de Python llamada *pymongo*, la cual permite realiza todas las tareas CRUD (crear, leer, actualizar y eliminar) de manera sencilla, sin necesidad de crear servicios extras.

```python
from pymongo import MongoClient

def testing_mongo(request):
    client = MongoClient('mongodb://%s:%s@54.89.16.99' % (settings.USERNAME_MONGODB, settings.PASSWORD_MONGODB))
    db = client["blockchain"]
    transactions = db["transaction"]
    transaction = {
                        "class": "org.hyperledger.composer.system.HistorianRecord",
                        "transactionId": "171ecd31cee30edffa6af0a144e35ecb58c89d3fb683a24c244328dd63b1447b",
                        "transactionType": "org.hyperledger.composer.system.AddAsset",
                        "transactionInvoked": "resource:org.hyperledger.composer.system.AddAsset#171ecd31cee30edffa6af0a144e35ecb58c89d3f
                        "participantInvoking": "resource:org.hyperledger.composer.system.NetworkAdmin#admin",
                        "identityUsed": "resource:org.hyperledger.composer.system.Identity#08e5ca9c727ad43d49bc517415c805b27d45ccc145237
                        "eventsEmitted": [],
                        "transactionTimestamp": "2019-05-05T23:03:59.125Z"
                    }
    t = transactions.insert_one(transaction)
    datasets = db["dataset"]
    dataset = {
            "owner": "larramita@yopmail.com",
            "dataset": [
              {
                "type": "personal_data",
                "first_name": "Larra",
                "last_name": "Mita",
                "birth_date": "2019-05-05T23:03:59.125Z",
                "sex": "Femenino",
                "origin_country": "Mexico"
        },
        {
            "type": "medical_data",
            "height": 0,
            "weight": 0,
            "blood_pressure": 0,
            "glucose_level": 0,
            "residence_country": "Guadalajara",
        }
        ]
    }
    datasets.insert_one(dataset)

    """udpating someone"""
    my_query = { "transactionId": "171ecd31cee30edffa6af0a144e35ecb58c89d3fb683a24c244328dd63b1447b" }
    new_values = { "$set": { "transactionType": "Beta" } }
    transactions.update_one(my_query, new_values)
```

Algoritmo 6-20: Ejemplo de interacción con la base de datos MongoDB (views.py)

6.4.2. Modelos

A continuación se definen los modelos para el almacenamiento en la base de datos NoSQL:

Colección Dataset:

```
{
   "owner": "string",
   "dataset": [
      {
         "type": "string",
         "first_name": "string",
         "last_name": "string",
         "birth_date": "timezone",
         "sex": "string",
         "origin_country": "string"
      },
      {
         "type": "string",
         "height": "number",
         "weight": "number",
         "blood_pressure": "number",
```

```
            "glucose_level": "number",
            "residence_country": "string",
        }
    ]
}
```

Colección Transaction:

```
{
    "class": "string",
    "transactionId": "string",
    "transactionType": "string",
    "transactionInvoked": "string",
    "participantInvoking": "string",
    "identityUsed": "string",
    "eventsEmitted": [],
    "transactionTimestamp": "timestamp"
}
```

Una vez implementadas todas las interacciones con la base de datos, se da por terminada la fase de *Construcción*, ya que fueron resueltas todas las problemáticas planteadas en el caso de uso y se da cabida a la ´última fase de la metodología en la cual se realizan pruebas de interfaz en ambiente de producción.

6.5. Transición (Transition)

Esta última fase de la metodología propone realizar la validación de la solución construida en la fase de construcción. A lo largo de esta fase, se mostrarán las interfaces construidas con el fin de mostrar su usabilidad y validar la propuesta que se mostrará a los usuarios finales.

6.5.1. Iteración 0: Pruebas de interfaz de usuario

La presente iteración muestra las interfaces de usuario implementadas con el fin de mostrar la usabilidad y funcionalidades, además de describir el comportamiento de la aplicación en un ambiente de producción.

Para iniciar la aplicación web, se ejecuta el comando ‘python manage.py runserver’ desde la consola, situándose en el directorio de la herramienta.

En vista de comprobar el correcto funcionamiento de las interfaces, se hizo un sondeo a 15 personas con un perfil de usuario final para que pudieran evaluar las interfaces creadas con el fin de comprobar la correcta transmisión de la información. Se muestra una tabla compartida con los usuarios que interactuaron con la aplicación para que pudieran evaluar las interfaces con una calificación del 1 al 5, donde la puntuación representa:

1 Muy mal.
2 Mal
3 Regular.

4 Bien

5 Excelente.

En la siguiente tabla puede observarse la descripción del tipo de prueba a realizar:

Tipo de prueba	Aceptación
Ejecutantes	Usuarios finales potenciales
Herramienta usada	Preguntas evaluativas
Veces ejecutada	15 por interfaz seleccionada
Resultados esperados	90 % de resultados mayores a la media evaluativa (Mayores a 2.5)

Tabla 6-4: Especificaciones de la prueba de aceptación para interfaces graficas

La próxima tabla contiene las preguntas evaluativa:

Pregunta	Puntos
¿Considera que la interfaz es agradable a la vista?	
¿La información se entiende correctamente?	
¿Comprende el objetivo de la interfaz?	
¿La funcionalidad de los botones queda clara?	
¿Se siente ubicado en la pantalla actual de la interfaz?	

Tabla 6-5: Tabla para evaluación de interfaces graficas

Al momento de evaluar una interfaz, se muestran las preguntas realizadas a los usuarios, luego se encuentra el promedio de las respuestas recibidas por los mismos.

A continuación se muestra la pantalla principal de la herramienta:

Figura 6-23: Pantalla principal del cliente web.

Pregunta	Puntos
¿Considera que la interfaz es agradable a la vista?	5
¿La información se entiende correctamente?	4,5
¿Comprende el objetivo de la interfaz?	4
¿La funcionalidad de los botones queda clara?	4,5
¿Se siente ubicado en la pantalla actual de la interfaz?	4,8

Tabla 6-6: Tabla para evaluación de la pantalla principal del cliente web.

6.5.1.1. Crear dataset

La creación de datasets se muestra como un formulario que llena el usuario, finalmente se presenta el botón para crear el dataset, una vez creado se mostrará en la pantalla principal de la aplicación:

Figura 6-24: Pantalla para la creación de un dataset

Pregunta	Puntos
¿Considera que la interfaz es agradable a la vista?	5
¿La información se entiende correctamente?	5
¿Comprende el objetivo de la interfaz?	5
¿La funcionalidad de los botones queda clara?	4,8
¿Se siente ubicado en la pantalla actual de la interfaz?	5

Tabla 6-7: Tabla para evaluación de la pantalla crear dataset.

6.5.1.2. Visualizar un dataset/ Comprar un dataset

Las funcionalidades de visualizar un dataset y comprar un dataset vienen una misma vista, debido a que un usuario querrá tener a su alcance todos los detalles del dataset que comprará:

Figura 6-25: Selección compra de múltiples datasets

Pregunta	Puntos
¿Considera que la interfaz es agradable a la vista?	4,9
¿La información se entiende correctamente?	4,5
¿Comprende el objetivo de la interfaz?	5
¿La funcionalidad de los botones queda clara?	4,8
¿Se siente ubicado en la pantalla actual de la interfaz?	5

Tabla 6-8: Tabla para evaluación de la pantalla comprar dataset.

Figura 6-26: Detalles de un dataset una vez es comprado

Pregunta	Puntos
¿Considera que la interfaz es agradable a la vista?	5
¿La información se entiende correctamente?	5
¿Comprende el objetivo de la interfaz?	5
¿La funcionalidad de los botones queda clara?	4,1
¿Se siente ubicado en la pantalla actual de la interfaz?	5

Tabla 6-9: Tabla para evaluación de la pantalla detalle de dataset.

6.3.5.8.

Capítulo 7 – Conclusiones

El presente TEG muestra las bondades que brindan estos nuevos enfoques para resolver problemáticas complejas que cada vez requieren más confianza y transparencia a la hora de manejar bienes valiosos como datos personales y cualquier tipo de propiedad que represente un valor para el que la posea.

Si bien la solución planteada no es infalible, es un paso más cerca a eliminar las malversaciones de datos en sistemas de uso cotidiano.

Se logró crear una solución que, apoyada en la criptografía, las cadenas de bloques y los contratos inteligentes, puede brindar confianza a sus usuarios. Este tipo de soluciones, son sólo el inicio de una tecnología muy joven que cada día se refina un poco más y promete revolucionar como se manejan la propiedad del individuo y la confianza que este le tiene a los sistemas distribuidos.

7.0.1. Seleccionar Caso de Uso

Se logró seleccionar un caso de uso que ilustraba de una manera evidente una problemática a resolver, y además, muestra como la criptografía, apoyada en la cadena de bloques y los contratos inteligentes, puede ser una solución eficiente a dichas problemáticas con la finalidad de generar confianza y transparencia a la hora de traspasar fondos.

7.0.2. Seleccionar una metodología de desarrollo

Se planteó una metodología que logró adaptarse y ser eficiente en los enfoques ágiles actuales sin dejar de lado prácticas tradicionales. Finalmente la metodología resultó un pilar fundamental a la hora de definir las etapas en las que se construía la herramienta y la forma en que fueron ejecutadas.

7.0.3. Preparar el ambiente de desarrollo de la plataforma Hyperledger Fabric

Se hizo uso del framework Hyperledger Composer como pieza fundamental a la hora de preparar ambiente para el uso de contratos inteligentes, ya sea de manera local o de producción. Estas herramientas permitieron la interacción con la red y los contratos inteligentes que en ella residen de una manera rápida y sencilla, se puede concluir que la preparación del ambiente para la herramienta desarrollada fue exitoso gracias al uso de esta herramienta.

7.0.4. Recopilación de datos

La recopilación de las interacciones del usuario con el contrato fueron almacenadas de manera exitosa en la herramienta propuesta, MongoDB probó ser una herramienta versátil para realizar este tipo de tareas. Su similitud con el uso de objetos de JavaScript hizo que su implementación se lograra de manera sencilla y eficiente.

7.0.5. Definir almacén de datos

El alcance de la investigación no pudo cubrir una definición que pueda considerarse eficiente del punto de vista de un analista de datos. Se logró recopilar ciertos datos de importancia, pero no se pudo asegurar que la forma en que fueron almacenados fue la más eficiente o que fue utilizada la herramienta que mejor se ajuste a este tipo de análisis futuros. El uso, transformación y carga de los datos en plataformas eficientes para el procesamiento de grandes volúmenes de datos se releva a futuras investigaciones.

7.0.6. Implementar la aplicación sobre la base tecnológica de cadena de bloques y contratos inteligentes

La combinación de las tecnologías utilizadas, junto con la metodología seleccionada dieron pie a varias iteraciones que permitieron implementar en fases concretas una solución al caso de uso planteado, la solución permite una interacción directa tanto como con la red Fabric como con los contratos inteligentes que viven en la misma.

7.0.7. Pruebas de funcionamiento

Se realizaron pruebas de tipo funcional las cuales validaron el correcto funcionamiento de la aplicación en cada una de las versiones, se realizó una prueba por cada versión desplegada y las mismas ayudaron a detectar errores en cada una de las diferentes iteraciones.

Adicionalmente se realizaron pruebas de aceptación las cuales fueron realizadas por potenciales usuarios finales con la finalidad de validar la correcta transferencia de información desde la aplicación a los usuarios finales, los resultados de estas pruebas de validación arrojaron en su totalidad resultados positivos lo cual reafirmó su correcto funcionamiento.

7.0.8 Puesta en marcha

Finalmente, la puesta en marcha en un ambiente de producción se logró exitosamente mediante el uso de una instancia de AWS el cual permitió que la herramienta previamente desarrollada y probada pudiera ser desplegada para que los usuarios finales pudieran interactuar con la herramienta.

7.1. Contribuciones

- Se sentaron las bases para el desarrollo de aplicaciones que se basen en tecnologías asociadas a las cadenas de bloques y los contratos inteligentes, el presente TEG puede sirve de guía para atacar diferentes casos de uso asociados a los enfoques seleccionados.
- Dada la arquitectura planteada, se desarrollaron varios módulos de importancia los cuales sirven de utilidad para trabajos futuros que puedan tomar los datos generados por el presente TEG para aplicar diferentes técnicas de analítica predictiva e inteligencia artificial.

7.2. Recomendaciones

Para hacer pruebas con este tipo de tecnologías, la mejor opción es utilizar la herramienta Hyperledger Composer Playground, ya que permite diseñar y desplegar una red blockchain independientemente del hardware que se tenga para desarrollar.

7.3. Trabajos futuros

Como se contempló en la arquitectura propuesta para este TEG, los datos y la recopilación de los mismos mediante esta herramienta quedan abiertos y estructurados de manera que sea más sencillo para los expertos en el área de minería de datos o tecnologías afines, tomar esos datos y aplicar algoritmos de inteligencia artificial sobre ellos y detectar patrones de comportamiento u otros patrones útiles para el estudio de estas herramientas.

Bibliografía

[1]Gupta. M. 2017. Blockchain for Dummies, IBM Limited Edition, John Wiley & Sons, Inc.

[2]Nakamoto, Satoshi. 2008. Bitcoin: a peer-to-peer electronic cash system; http://bitcoin.org/bitcoin.pdf.

[3]Stuart Haber, Scott S.: How to Write Time-Stamp a Digital Document

[4]Josh Benaloh, Michael de M.: One-way accumulators: a decentralized alternative to digital signatures, 1993

[5]Muhammad Saqib Niaz, Gunter S.: Merkle Hash Tree based Techniques for Data Integrity of Outsourced Data, 2015

[6] Pattison, I. (2017). 4 characteristics that set blockchain apart. USA: IBM. Recuperada de: https://www.ibm.com/blogs/cloud-computing/2017/04/characteristics-blockchain/

[7] DTCC CONNECTION. (2016). Eight Key Features of Blockchain and Distributed Ledgers Explained. USA: DTCC CONNECTION. Recuperada de: http://www.dtcc.com/news/2016/february/17/eight-key-features-of-blockchain-and-distributed- ledgers-explained

[8] Lamport, L., et al. 1982. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems 4(3): 382-401; https://dl.acm.org/citation.cfm?id=357176.

[9]https://kb.osu.edu/bitstream/handle/1811/73299/ISJLP_V3N2_337.pdf?sequence=1

[10] https://medium.com/kokster/understanding-hyperledger-fabric-byzantine-fault-tolerance- cf106146ef43

[11] JP Buntix. (2017).What is Proof of Elapsed Time?. The Merkle. Recuperada de: https://themerkle.com/what-is-proof-of-elapsed-time/

[12]Bashir, I. (2017), Mastering Blockchain, Birmingham, UK: Packt Publishing Ltd

[13] Behlendorf, B. 2016. Hyperledger: An “Umbrella” for Open Source Blockchain & Smart Contract Technologies; https://www.hyperledger.org/blog/2016/09/13/meet-hyperledger-an-umbrella-for-open-source-blockchain-smart-contract-technologies.

[14]https://www.hyperledger.org/projects/composer

[15]Developing a Blockchain Business Network with Hyperledger Composer using the IBP Starter Plan.

[16]The General Theory of Decentralized Applications, Dapps. https://github.com/DavidJohnstonCEO/DecentralizedApplications

[17]Steinmetz, Wehrle K.: What Is This Peer-to-Peer About?

[18]Introducción Hyperledger Composer, https://hyperledger.github.io/composer/latest/introduction/introduction

[19]Arquitectura típica Hyperledger Composer, https://hyperledger.github.io/composer/latest/introduction/solution-architecture

[20] Rilee, K.2018. Understanding Hyperledger Fabric; https://medium.com/kokster/understanding-hyperledger-fabric-chaincode-e7767d50f0b4

[21] Hyperledger Fabric. 2017. What is Chaincode?; https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode.html

[20]https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatis.html

[21]https://hyperledger-fabric.readthedocs.io/en/release-1.4/arch-deep-dive.html

[22]Christopher D. Clack, Lee B.: Smart Contract Templates: foundations, design landscape and research directions, August 4, 2016

[23]https://hyperledger-fabric.readthedocs.io/en/release-1.4/smartcontract/smartcontract.html

[24]https://medium.com/swlh/how-to-talk-to-a-company-about-blockchain-the-ibm-hyperledger-example-2-2-fb680e9ee346

[25] Smart Contracts: 12 Use Cases for Business and Beyond. https://gallery.mailchimp.com/a87f67248663abe55ad9325d6/files/Smart_Contracts_12_Use_Cases_for_Business_Beyond.pdf

[26]Agile. http://www.ambysoft.com/unifiedprocess/agileUP.html

[27]RUP. http://www.usmp.edu.pe/publicaciones/boletin/fia/info49/articulos/RUP%20vs.%20XP.pdf

[28]http://nosolopau.com/2012/06/07/mas-sobre-el-proceso-unificado-agil-fases-y-disciplinas/

[29]Base de datos no relacionales. http://nosql-database.org/

[30]Cassandra. http://cassandra.apache.org/doc/latest/

[31]CouchDB. http://docs.couchdb.org/en/2.1.1/

[32]Dynamo. https://aws.amazon.com/es/documentation/dynamodb/

[33]HBase. https://hbase.apache.org/book.html

[34]InfiniteGraph. http://www.objectivity.com/products/infinitegraph/

[35]MongoDB. https://www.mongodb.com/what-is-mongodb

[36]Neo4j. https://neo4j.com/docs/developer-manual/current/

[37]Redis. https://redis.io/documentation

[38]Nyce, Charles: Predictive Analytics White Paper, 2007

[39]Saarenvirta, Gary: Machine Learning in Retail, 2010

[40]L, Molina: Data mining: torturando los datos., 2016

[41]Li Deng, Dong Y.: Deep Learning Methods and Applications, 2014

[42]Javascript. https://developer.mozilla.org/es/docs/Web/JavaScript

[43]Mocha. https://mochajs.org/

[44]Lenguaje de modelado, https://hyperledger.github.io/composer/v0.19/reference/cto_language

[45]Lenguaje de control de acceso, https://hyperledger.github.io/composer/v0.19/reference/acl_language

[46]Composer Playground, https://github.com/hyperledger/composer/wiki/Composer-Playground

[47]https://www.toptal.com/nodejs/let-loopback-do-it-a-walkthrough-of-the-node-api-framework-you-ve-been-dreaming-of

[48]Cucumber, https://docs.cucumber.io/guides/overview/

[49]Django, https://www.djangoproject.com/

[50] Karma, https://karma-runner.github.io/3.0/intro/how-it-works.html

[51] Docker, https://docs.docker.com/

[52] Yeoman, https://yeoman.io/learning/

[53] Python, https://docs.python.org/3/tutorial/index.html

[54] Fontanela, A. (2015). ¿Qué es Bootstrap? [blog].Disponible en: https://raiolanetworks.es/blog/que-es-bootstrap/