

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

**CADENA DE CUSTODIA PARA PRUEBAS DIGITALES USANDO LA  
TECNOLOGÍA BLOCKCHAIN**

**BLOCKCHAIN-BASED CHAIN OF CUSTODY FOR DIGITAL  
EVIDENCES**

Realizado por

**Joaquín Trillo Escribano**

Tutorizado por

**Manuel Díaz Rodríguez**

Departamento

**Departamento de Lenguajes y Ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA

MÁLAGA, julio 2019



# RESUMEN

---

Este Trabajo de Fin de Máster describe el desarrollo e implementación de una cadena de custodia para pruebas digitales apoyándose en la tecnología blockchain.

El sistema desarrollado tiene tres componentes o subsistemas principales. Por un lado, tenemos una cadena de bloques basada en permisos (distinta de las blockchains públicas como Bitcoin o Ethereum, que no requieren permisos) en la cual se ejecuta un contrato inteligente que permite añadir los hashes de nuevas pruebas digitales a la cadena. De esta forma se puede comprobar en un futuro si las evidencias han sido o no modificadas. Para implementar la blockchain basada en permisos se ha utilizado el framework Hyperledger Fabric y para definir el contrato inteligente la herramienta Hyperledger Composer.

Por otro lado, existe un repositorio de ficheros en el que se almacenan copias de estas pruebas.

Por último, se ha desarrollado una aplicación web en Angular que permite a las distintas entidades involucradas en el proceso de la cadena de custodia intervenir en el mismo. A través de unas credenciales, cada uno de los usuarios de la aplicación web podrá visualizar un listado de casos en los que participa. Cada uno de estos casos tiene una serie de pruebas que las partes implicadas (si tienen los permisos necesarios) se van intercambiando. Todo este proceso de intercambio de evidencias se registra en la blockchain basada en permisos.

## Palabras clave

Cadena de custodia, blockchain, prueba, trusted timestamping, permisos, Hyperledger, contrato inteligente, Business Network, repositorio de ficheros, aplicación web, Angular, Firebase.

# ABSTRACT

---

This Master's Degree Thesis describes the development and implementation of a blockchain-based Chain of Custody for digital evidences.

The developed system has three main components or subsystems. On one hand, we have a permissioned blockchain (different from well-known public blockchains like Bitcoin or Ethereum, which are permissionless) in which a smart contract is executed. This smart contract allows us to add new evidences' hashes to the chain. Thanks to the hash of every proof, we can check later if the evidence has been modified or not. To implement this permissioned blockchain I have used the Hyperledger Fabric framework and to define the smart contract I have used the tool Hyperledger Composer.

On the other hand, there is a file repository where the copies of those evidences are stored.

Finally, a web application has been built using. Thanks to this one, the different entities involved in the chain of custody process can intervene in it. Using credentials, each user of the web application can visualize a list of cases in which they participate. Every case has a set of evidences and those evidences are transferred between the participants. All this process of exchange of digital evidence is registered in the permissioned blockchain.

## Keywords

Chain of Custody, blockchain, evidence, trusted timestamping, permissions, Hyperledger, smart contract, Business Network, file repository, web application, Angular, Firebase.

# ÍNDICE

---

<b>1. INTRODUCCIÓN .....</b>	<b>1</b>
1.1. Motivación .....	1
1.2. Objetivos .....	2
1.3. Estructura de la memoria .....	3
<b>2. CONTEXTUALIZACIÓN DEL TRABAJO .....</b>	<b>5</b>
2.1. Fundamentos teórico-prácticos .....	5
2.1.1. Cadena de custodia .....	5
2.1.2. Blockchain .....	6
2.1.3. Trusted timestamping .....	8
2.2. Estado del arte .....	9
2.3. Tecnologías utilizadas.....	10
2.3.1. Hyperledger Fabric .....	10
2.3.2. Hyperledger Composer.....	12
2.3.3. Angular .....	15
2.3.4. Firebase.....	16
2.4. Arquitectura del sistema .....	16
<b>3. DEFINICIÓN DE LA <i>BUSINESS NETWORK</i>.....</b>	<b>19</b>
3.1. Modelo del Sistema .....	19
3.2. Script de las transacciones .....	20
3.3. Lista de control de acceso .....	21
3.4. Consultas .....	24
<b>4. CONFIGURACIÓN DEL SERVIDOR REST HYPERLEDGER COMPOSER .....</b>	<b>27</b>
4.1. Estrategia de autenticación utilizada .....	29
4.2. Habilitando el modo multiusuario .....	32
4.3. Asegurando el servidor usando HTTPS y TLS.....	35

4.4. Segundo servidor REST desplegado .....	38
<b>5. DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN WEB PARA EL REGISTRO Y CONSULTA DE EVIDENCIAS DIGITALES .....</b>	<b>39</b>
5.1. Requisitos de la aplicación .....	39
5.1.1. Requisitos funcionales.....	39
5.1.2. Requisitos no funcionales.....	40
5.2. Casos de uso .....	41
5.3. Comunicación con el resto del sistema.....	43
<b>6. DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN WEB PARA LA ADMINISTRACIÓN DE SOLICITUDES DE NUEVOS USUARIOS.....</b>	<b>45</b>
6.1. Requisitos de la aplicación .....	45
6.1.1. Requisitos funcionales.....	45
6.1.2. Requisitos no funcionales.....	46
6.2. Casos de uso .....	46
6.3. Comunicación con el resto del sistema.....	47
<b>7. SERVICIOS DE FIREBASE USADOS .....</b>	<b>49</b>
7.1. Firebase Storage.....	49
7.2. Firebase Cloud Firestore.....	50
7.3. Firebase Auth .....	52
7.4. Firebase Hosting.....	54
<b>8. CONCLUSIONES Y LÍNEAS FUTURAS.....</b>	<b>59</b>
8.1. Conclusiones .....	59
8.2. Líneas futuras .....	60
<b>ANEXO A – MODELO DEL SISTEMA.....</b>	<b>63</b>
<b>ANEXO B – MANUAL PARA LA APLICACIÓN WEB DE REGISTRO Y CONSULTA DE EVIDENCIAS .....</b>	<b>69</b>

<b>ANEXO C – MANUAL PARA LA APLICACIÓN WEB DE GESTIÓN DE SOLICITUDES DE NUEVOS USUARIOS .....</b>	<b>79</b>
<b>REFERENCIAS Y BIBLIOGRAFÍA .....</b>	<b>85</b>

## LISTA DE FIGURAS

---

Figura 1. Estructura de una blockchain.....	6
Figura 2. Proceso de trusted timestamping (Fuente: Wikipedia).....	9
Figura 3. Creación y despliegue de una Business Network Definition. ....	13
Figura 4. Relaciones entre los tipos de participantes en Hyperledger Composer. ....	14
Figura 5. Componentes en Angular (ejemplo). ....	15
Figura 6. Flujo de datos en el sistema durante la subida de una nueva prueba. ....	17
Figura 7. Flujo de datos en el sistema durante la consulta de una prueba ya existente. ....	18
Figura 8. Ejemplo de regla para la lista de control de acceso. ....	22
Figura 9. Named Queries definidas.....	25
Figura 10. Flujo de datos usando el servidor REST de Hyperledger Composer. ....	27
Figura 11. Aplicaciones OAuth en Github. ....	30
Figura 12. Client ID y Client Secret de la aplicación OAuth registrada. ....	30
Figura 13. Configuración de la aplicación OAuth de Github registrada.....	31
Figura 14. Variable COMPOSER_PROVIDERS.....	32
Figura 15. Arquitectura del servidor REST cuando el modo multiusuario está habilitado. .....	34
Figura 16. Variable de entorno COMPOSER_DATASOURCES.....	35
Figura 17. Generación de la clave privada para la CA.....	35
Figura 18. Generación del certificado de la CA (auto firmado). ....	36
Figura 19. Generación de la clave privada para el servidor. ....	36
Figura 20. Creación del CSR para el certificado del servidor.....	37
Figura 21. Generación del certificado del servidor.....	37
Figura 22. Conversión de la clave privada y el certificado del servidor a formato PEM. ....	38
Figura 23. Diagrama de casos de uso de la aplicación de registro y consulta de evidencias digitales. ....	41



Figura 24. Comunicación de la aplicación con otros componentes del sistema. ....	43
Figura 25. Diagrama de casos de uso de la aplicación de administración de solicitudes de nuevos usuarios. ....	47
Figura 26. Comunicación de la aplicación de administración con otros componentes del sistema. ....	48
Figura 27. Una carpeta por caso en Firebase Storage. ....	50
Figura 28. Evidencias del caso MLG-001 almacenadas en Firebase Storage. ....	50
Figura 29. Colecciones existentes en Firebase Cloud Firestore. ....	51
Figura 30. Ejemplo de documento de la colección requests. ....	51
Figura 31. Ejemplo de documento de la colección admins. ....	52
Figura 32. Reglas que controlan el acceso a los documentos de Firebase Cloud Firestore. ....	52
Figura 33. Habilitar el inicio de sesión usando envío de enlaces por correo electrónico. ....	53
Figura 34. Dominios autorizados para la redirección de OAuth. ....	54
Figura 35. Mensaje de Angular CLI cuando se usa el servidor de desarrollo. ....	54
Figura 36. Firebase CLI - Sesión iniciada en Firebase. ....	55
Figura 37. Firebase CLI - Servicios de Firebase que se quieren usar. ....	56
Figura 38. Firebase CLI - Proyecto Firebase a utilizar. ....	56
Figura 39. Firebase CLI - Configuración necesaria para desplegar la aplicación en Firebase Hosting. ....	56
Figura 40. Incluir el par clave-valor señalado en rojo en el archivo firebase.json. ....	57
Figura 41. Firebase CLI - Despliegue de la aplicación. ....	57
Figura 42. Envío de solicitud. ....	69
Figura 43. Inicio de sesión en la aplicación OAuth de Github. ....	70
Figura 44. Pequeño formulario para importar una card. ....	71
Figura 45. Menú principal. ....	71

Figura 46. Barra de navegación para usuarios autenticados. ....	72
Figura 47. Abrir nuevo caso. ....	72
Figura 48. Perfil personal. ....	72
Figura 49. Búsqueda de casos y evidencias. ....	73
Figura 50. Vista Ver caso. ....	73
Figura 51. Diálogo para añadir un nuevo participante. ....	74
Figura 52. Vista Nueva evidencia. ....	75
Figura 53. Uso de la herramienta MD5 & SHA Checksum Utility. ....	75
Figura 54. Vista Ver evidencia. ....	76
Figura 55. Ventana emergente para la transferencia de evidencias. ....	77
Figura 56. Posibles mensajes al intentar iniciar sesión. ....	79
Figura 57. Mensaje recibido con el enlace de un único uso. ....	80
Figura 58. Caso en el que no hay solicitudes pendientes. ....	80
Figura 59. Tabla con las solicitudes pendientes. ....	81
Figura 60. Mensaje de Github cuando la cuenta indicada no existe. ....	81
Figura 61. Estado de los botones de procesamiento según el número de solicitudes seleccionadas. ....	82
Figura 62. Correo que debe enviar el administrador al nuevo usuario. ....	83
Figura 63. Diálogo que se abre al pulsar en los botones de rechazo de solicitudes. ....	84
Figura 64. Correo que debe enviar el administrador si se rechaza alguna petición. ....	84

## LISTA DE TABLAS

---

Tabla 1. Utilidad de las reglas definidas para la correcta ejecución de las transacciones. .....	23
Tabla 2. Servicios de Firebase utilizados.....	49
Tabla 3. URLs de las dos aplicaciones. ....	58



## 1. Introducción

---

Uno de los principales problemas en el campo de la informática forense es la gestión de las pruebas digitales. Desde que estas evidencias son recogidas o confiscadas hasta que son presentadas en un juicio, múltiples entidades participantes en la investigación necesitan acceder a las pruebas. Y, por lo tanto, ser temporalmente los propietarios de estas. A este proceso de intercambio de la propiedad temporal de evidencias se le conoce como **cadena de custodia**.

Esta cadena de custodia debe garantizar la **integridad** de las pruebas durante la investigación. Es decir, debe asegurar que las pruebas no hayan sido modificadas y puedan ser admisibles en un juicio. Pero no solo eso, también debe permitir la **trazabilidad** de estas evidencias. De esta forma la cadena de custodia nos permite conocer todos los propietarios temporales de las evidencias y el periodo de tiempo en el cual tuvieron estas pruebas en su poder [1].

En la actualidad, la cadena de custodia para evidencias digitales se gestiona de forma manual. Esto implica la intervención de entidades (algunas de ellas ajenas a la investigación) que participan en la cadena rellenando documentos que acompañan a las pruebas.

En este Trabajo de Fin de Máster propongo una **cadena de custodia usando la tecnología blockchain** (cadena de bloques) para desmaterializar este procedimiento de gestión de evidencias digitales. Usando esta tecnología se garantiza la **integridad** de las pruebas gracias a los *timestamps* (marcas temporales) que la cadena de bloques emitirá cada vez que se registre una nueva evidencia (con su correspondiente valor hash). Además, la blockchain permitirá la **trazabilidad** de los diferentes propietarios temporales de las pruebas.

### 1.1. Motivación

Con este Trabajo de Fin de Máster he pretendido poner en práctica los conocimientos que he adquirido durante este último año y medio. Más concretamente, he querido utilizar lo que he aprendido en asignaturas de la especialidad de Ciberseguridad, como Informática Forense.

Por otra parte, el uso de Hyperledger Fabric e Hyperledger Composer viene de una necesidad laboral, ya que en próximos proyectos en mi empresa implementaremos soluciones con blockchain basada en permisos para garantizar la integridad y la trazabilidad en cadenas de suministros.

En cambio, el uso del framework Angular para la implementación de la aplicación web deriva de un interés personal en conocer esta tecnología, desarrollando una primera aplicación.

## **1.2. Objetivos**

Este TFM tiene tres objetivos principales, fijados al principio del proyecto y determinados en el anteproyecto entregado:

- **Definición de la *Business Network*.** Con la blockchain basada en permisos ya montada con Hyperledger Fabric, lo primero es definir el modelo con el que queremos trabajar y el comportamiento de las transacciones que se pueden ejecutar sobre dicho modelo. Para ello se ha utilizado la herramienta Hyperledger Composer, destinada a facilitar el trabajo con Hyperledger Fabric mediante la definición de una Business Network (más información sobre este concepto en el apartado 2).
- **Diseño e implementación de la aplicación web para el registro y consulta de pruebas.** Usando el framework Angular, se ha implementado una aplicación gracias a la cual los usuarios de esta pueden registrar nuevas evidencias y consultar las ya existentes (si tienen los permisos necesarios para acceder estas pruebas).
- **Configuración del repositorio de ficheros en el que se almacenan las copias de las pruebas.** En este repositorio se almacenan las denominadas *master copies* (copias de las pruebas con las cuales se hacen otras copias) de las evidencias.

Durante el desarrollo de este Trabajo de Fin de Máster se han incluido tres objetivos adicionales, necesarios para el correcto funcionamiento del sistema de cadena de custodia propuesto. Estos nuevos objetivos son:

- **Configuración del servidor REST de Hyperledger Composer.** Este servidor es la herramienta necesaria para comunicar la aplicación web de registro y consulta de evidencias digitales con la *Business Network* desplegada en la blockchain de Hyperledger Fabric.
- **Diseño e implementación de una aplicación web para la gestión de solicitudes de nuevos usuarios.** En la mayoría de las aplicaciones web, el registro de nuevos usuarios es automático (relleno de un formulario y recepción de un correo con las credenciales de acceso o enlace de verificación). **En este sistema ese procedimiento no es viable**, ya que es necesario conocer a los participantes y otorgarles unas identidades con las que puedan ejecutar las transacciones definidas en la *Business Network*.
- **Configuración de una base de datos para el almacenamiento de las solicitudes de nuevos usuarios.** En esta base de datos se guardarán las solicitudes de las personas interesadas en participar en el sistema de cadena de custodia.

### **1.3. Estructura de la memoria**

En el **segundo apartado** de esta memoria se introducen los fundamentos teórico-prácticos de este trabajo como cadena de custodia o blockchain. También se hace un breve repaso al estado del arte en cuanto a la gestión de evidencias digital y se exponen las diversas tecnologías usadas en este proyecto. Por último, se muestra la arquitectura del sistema y como sus componentes se comunican entre ellos.

La definición de la *Business Network* con la herramienta Hyperledger Composer queda explicada el **tercer apartado** de esta memoria. En este capítulo se conocerán que tipos participantes y objetos se han especificado, que transacciones pueden ejecutarse, los eventos que se emiten, las consultas que se han definido, etc. Todos estos elementos componen el sistema de cadena de custodia.

En el **cuarto capítulo** de este documento se expone como configurar correctamente el servidor REST de Hyperledger Composer, necesario para que la aplicación web puede acceder a la información almacenada en la blockchain e interactuar con ella a través de las transacciones declaradas en la *Business Network*.

La información relacionada con el diseño y la implementación de la aplicación web para el registro y consulta de pruebas digitales se encuentra en el **quinto apartado** de la memoria. En este capítulo además se detallan los requisitos funcionales, los requisitos no funcionales y los casos de uso de la aplicación, así como la comunicación de esta aplicación con otros componentes del sistema de cadena de custodia desarrollado para este Trabajo de Fin de Máster.

De forma similar al apartado anterior, en el **capítulo sexto** se documenta la aplicación web para la gestión de solicitudes de nuevos usuarios.

En el **séptimo apartado** se habla del repositorio de ficheros elegido para almacenar las *master copies* de las evidencias y de la base de datos utilizada para guardar las solicitudes de nuevos usuarios.

En el **octavo y último capítulo** de este documento se proporcionan unas conclusiones del trabajo, así como una serie de líneas de trabajo futuras.

Finalmente se encuentran **los anexos, que en total son tres**. El anexo A muestra en detalle el modelo con el que se ha definido la Business Network, mientras que el anexo B es un manual de usuario de la aplicación para el registro. Por su parte, el anexo C es una guía de uso de la aplicación de gestión de solicitudes.



## **2. Contextualización del trabajo**

---

Antes de continuar con el desarrollo de los tres objetivos principales de este trabajo es necesario poner en contexto al lector. En este apartado de la memoria hablaremos de algunos fundamentos teórico-prácticos que aparecen en este trabajo. También veremos el estado del arte en cuanto a gestión de pruebas digitales se refiere, así como las tecnologías utilizadas en el desarrollo de esta cadena de custodia. Por último, se describirá la arquitectura del sistema y como todos los componentes de este se comunican entre sí.

### **2.1. Fundamentos teórico-prácticos**

Para poder comprender correctamente los siguientes apartados de esta memoria es necesario entender los conceptos de cadena de custodia, blockchain y trusted timestamping.

#### **2.1.1. Cadena de custodia**

Aunque ya hemos mencionado este concepto anteriormente, es preferible dar una definición de cadena de custodia para entender la utilidad de este proceso.

Una cadena de custodia es un proceso controlado que se le aplica a las pruebas relacionadas con un delito, desde su localización hasta su valoración por un especialista encargado de su análisis. La cadena de custodia debe garantizar que el procedimiento empleado haya tenido éxito y que la evidencia que se recolectó en la escena es la misma que se está presentando ante el tribunal. Toda cadena de custodia tiene una serie de etapas, que son las que siguen:

- Extracción de las pruebas
- Preservación y embalaje de la evidencia
- Transporte/traslado de la prueba
- Traspaso de la evidencia. Bien a laboratorios (para su análisis) o bien para fiscalías (para su custodia).
- Custodia y preservación final de la prueba hasta que el juicio tenga lugar.
- Devolución de la evidencia a su propietario o destrucción de esta.

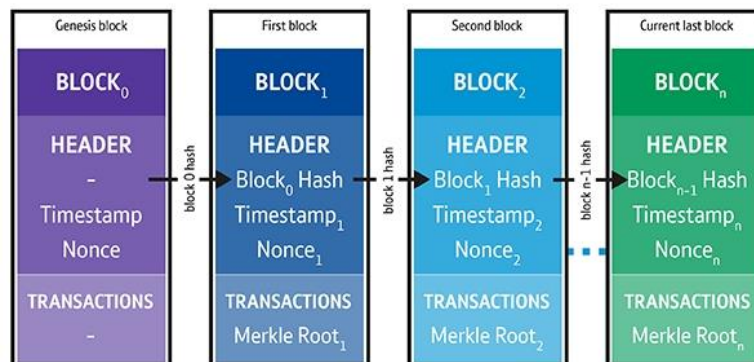
En el contexto de la informática forense el objetivo de la cadena de custodia es **garantizar la trazabilidad e integridad de la evidencia digital**. Si la integridad de dicha evidencia es cuestionada, la cadena de custodia se rompe y esa prueba puede declararse como inadmisible [1].

En el estándar internacional ISO/IEC 27037:2012 *Guidelines for identification, collection, acquisition and preservation of digital evidence* [2], se exponen buenas prácticas a la hora de recopilar pruebas digitales. Concretamente en el apartado de preservación, se indica que el uso de una cadena de custodia es esencial para la correcta conservación de las evidencias, entre otros procedimientos.

### 2.1.2. Blockchain

Hoy en día todos (o la gran mayoría de nosotros) hemos oído hablar de la tecnología blockchain pero, ¿qué es exactamente y qué aporta?

El término blockchain o cadena de bloques surge por primera vez en 2008, con la publicación del paper de Bitcoin por Satoshi Nakamoto [3]. **Una blockchain es una lista enlazada** formada por múltiples conjuntos de transacciones, que se conocen como bloques. Estos bloques se conectan unos con otros usando funciones hash criptográficas. Cada bloque contiene el valor hash del bloque anterior (permitiendo así el enlazado entre bloques), una marca temporal (timestamp) del momento exacto en el que anexa el bloque a la cadena, y la información de todas las transacciones que componen dicho bloque. En la Figura 1 se puede ver la estructura de una cadena de bloques. Al bloque inicial de toda cadena de bloques se le llama **bloque génesis**.



*Figura 1. Estructura de una blockchain.*

Las características más importantes de la tecnología blockchain son las siguientes:

- **Es distribuida.** La cadena no está almacenada en un solo equipo, sino en cientos o miles de equipos. En cada uno de los nodos que forman parte de una red blockchain existe una copia del estado actual de la cadena de bloques. Así, la amplia mayoría de estas copias contendrá la información correcta.
- **Descentralización.** No es necesaria una tercera parte confiable que valide los nuevos bloques que se intentan anexar a la cadena. En redes blockchain, la validez se otorga a través de un consenso entre los nodos que pertenece a la red blockchain. En la actualidad existen diversos algoritmos de consenso: *Proof-of-Work* (PoW), *Proof-of-Stake* (PoS), *Proof-of-Elapsed-Time* (PoET), tolerante a fallos bizantinos, etc. Por ejemplo, la red Bitcoin usa el algoritmo *Proof-of-Work* o Prueba de Trabajo.
- **Uso de hashes.** Cualquier modificación de la información almacenada en la cadena es detectada, ya que el valor hash del bloque modificado cambiaría. Por lo tanto, el uso de una blockchain permite almacenar y transmitir datos con la seguridad de que esta información no puede ser modificada.

Gracias a que es distribuida, es descentralizada y usa funciones hash criptográficas, los datos almacenados en una blockchain no pueden ser modificados. Así es como esta tecnología **garantiza la integridad de los datos almacenados en la cadena.**

Otra característica habitualmente ligada a blockchain y que no he mencionado es el acceso público a la cadena. Pero esto no es siempre así, puesto que existen tres grandes tipos de blockchains: públicas, privadas o de consorcio.

- **Blockchains públicas.** No tienen permisos y están abiertas a todo el mundo, por lo que no necesitan control de acceso. Por el contrario, requieren que el consenso se lleve a cabo antes de anexar nuevos bloques en la cadena, lo cual no es inmediato. El tiempo medio que tarda en añadirse un nuevo bloque se conoce como *block time*, y en Bitcoin es actualmente de 12 minutos y 38 segundos, mientras que en Ethereum es de unos 15 segundos [4].

- **Blockchains privadas.** Están basadas en permisos, por lo que para participar en este tipo de redes blockchain se necesita una invitación de los administradores de la red en cuestión. Además, todos los nodos de este tipo de blockchain pertenecen a una misma empresa. A cambio, el tiempo que tarda un nuevo bloque en añadirse a la cadena es insignificante, ya que no necesitan realizar el protocolo de consenso.
- **Blockchains de consorcio.** También llamadas semi-descentralizadas. Están basadas en permisos como las blockchains privadas, pero en este caso los nodos pertenecen a varias organizaciones (de ahí lo de consorcio). Al igual que en las privadas, los administradores del consorcio son los que otorgan los permisos de participación en la red. Como aquí no solo hay una compañía involucrada, los nodos deben realizar un protocolo de consenso.

### 2.1.3. Trusted timestamping

De acuerdo al estándar RFC 3161 [5], un sellado de tiempo confiable o *trusted timestamping* es un *timestamp* emitido por una tercera parte confiable que actúa como una autoridad de sellado de tiempo (en inglés, Timestamping Authority, TSA).

Estos *timestamps* son usados para demostrar la existencia de alguna información antes de cierta fecha (información de investigación, contratos, etc.) sin la posibilidad de que el dueño pueda cambiar el sellado de tiempo. Tal y como podemos observar en Figura 2, el proceso de *trusted timestamping* tiene varios pasos:

- **Cálculo del hash.** El usuario debe calcular el hash del documento que quiera sellar.
- **Envío del hash.** Una vez el usuario calcula el hash de ese documento, debe enviar dicho hash a una autoridad de sellado de tiempo.
- **Concatenación del hash recibido y del *timestamp*.** La TSA concatena el hash recibido y el *timestamp* de ese preciso instante.
- **Cálculo del hash de esta concatenación.**
- **Firma digital del hash de la concatenación junto al *timestamp*.** La TSA aplica su clave privada al hash de la concatenación y al *timestamp* generado previamente.

- **Envío del hash y *timestamp* firmados.** Por último, la TSA envía al usuario el hash y el *timestamp* firmados.

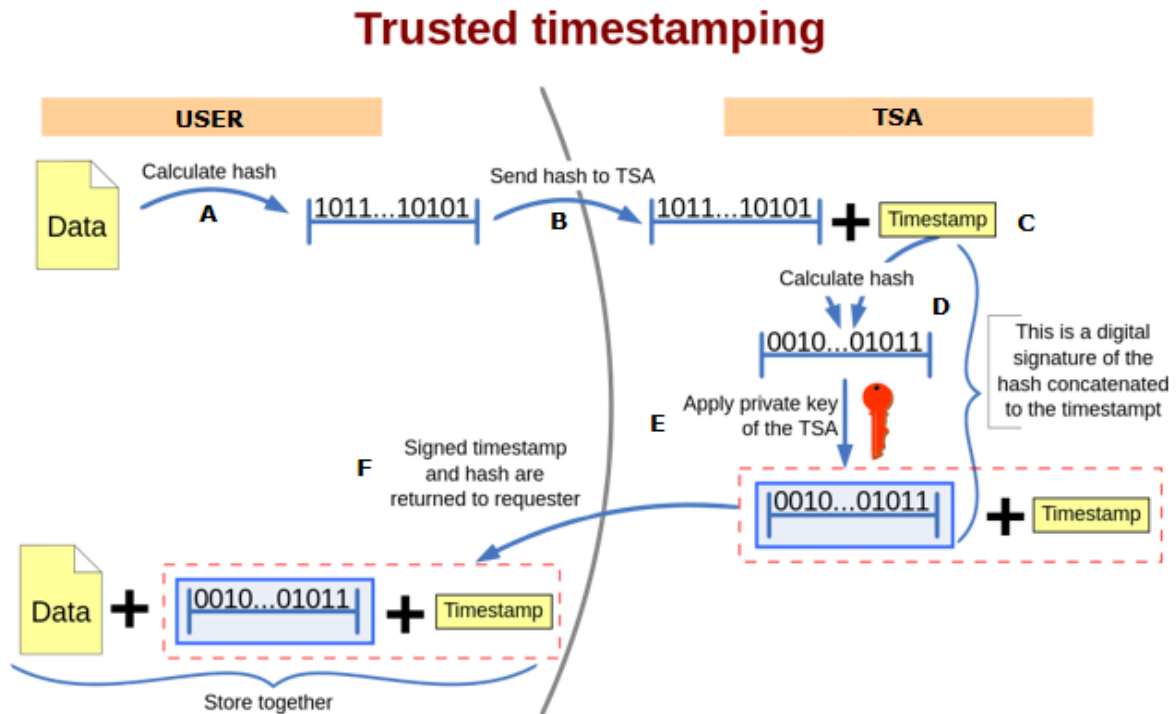


Figura 2. Proceso de trusted timestamping (Fuente: Wikipedia).

## 2.2. Estado del arte

Tal y como ya se ha mencionado en este documento, a día de hoy el procedimiento de gestión de evidencias digitales se lleva a cabo de forma manual. Y, por lo tanto, tiene el inconveniente de que terceras partes ajenas a la investigación (agentes de los depósitos de pruebas) intervienen en el proceso de la cadena de custodia, bien entregando y recibiendo pruebas, o bien, rellenando documentos que registren el intercambio de evidencias.

Pero el hecho es que en la actualidad no existe un sistema de gestión de pruebas digitales en el que no tengan que intervenir esas terceras partes ajenas para la cumplimentación de documentos de la cadena de custodia. Un primer acercamiento a esta idea de gestión descentralizada de evidencias digitales llega con el prototipo de cadena de custodia usando una blockchain privada propuesto por S. Bonimi et al [6].

En este artículo publicado en julio de 2018, se utiliza Geth [7], la implementación escrita en Go (la más popular de hecho) de un nodo completo de Ethereum. Geth permite montar una red privada desde cero y configurar todos los aspectos de una blockchain, incluido el protocolo de consenso empleado. En este prototipo propuesto se ejecuta un *smart contract* de Ethereum en la cadena de bloques. Este contrato inteligente permite crear, transferir, consultar y eliminar pruebas, si el usuario que ejecute dichas transacciones tiene el permiso necesario.

Tal y como hemos visto y veremos a lo largo de esta memoria, este TFM tiene varias similitudes con el prototipo de S. Bonimi et al, pero también existen diferencias notables. La más significativa, el uso de Hyperledger Fabric para la implementación de la cadena de bloques en lugar de Geth. Otro punto de divergencia notorio está en el contrato inteligente. Mientras que el *smart contract* de S. Bonimi está escrito en Solidity (lenguaje de Ethereum para la redacción de contratos inteligentes), en este Trabajo de Fin de master se ha definido una *Business Network* de Hyperledger Composer. A diferencia de Solidity, una *Business Network Definition* no solo permite la implementación de transacciones como ocurre con Solidity, sino que también posibilita la creación de recursos, participantes y eventos, definición de listas de control de acceso, etc. Gracias a esto, una *Business Network Definition* de Hyperledger Composer será más completa y estará mejor estructurada que un contrato inteligente de Solidity.

### **2.3. Tecnologías utilizadas**

Antes de empezar un proyecto es muy importante conocer bien las tecnologías que se van a emplear. En este Proyecto de Fin de Máster se han utilizado una gran variedad de tecnologías y por ello en este subapartado de la memoria vamos a exponer las más significativas para este trabajo: Hyperledger Fabric, Hyperledger Composer y Angular.

#### **2.3.1. Hyperledger Fabric**

Hyperledger Fabric [8] es una plataforma open-source que permite la creación de nuevas redes blockchain basadas en permisos. Diseñada para el uso en contextos empresariales, Fabric es uno de los 10 proyectos Hyperledger que aloja The Linux Foundation [9]. Además, ofrece capacidades que hacen que este framework se distinga de otras plataformas blockchain. Algunas de estas características son:

- **Open source.** Al ser un proyecto de código abierto en el que contribuyen múltiples empresas, el crecimiento y mantenimiento de Fabric no depende de una sola entidad (como si ocurre con Bitcoin o Ethereum). Actualmente su comunidad de desarrollo ha crecido hasta 35 organizaciones y casi 200 desarrolladores.
- **Modular y configurable.** La arquitectura de Fabric es modular y altamente configurable, lo que permite la innovación, versatilidad y optimización para un amplio rango de casos de usos empresariales, incluyendo banca, finanzas, seguros, recursos humanos, cadenas de suministros e incluso plataformas de música digital.
- **Soporta contratos inteligentes en lenguajes de programación de propósito general.** Fabric es la primera plataforma blockchain que permite la redacción de smart contracts en lenguajes de programación de propósito general como Java, Go y Node.js. Esto significa que las empresas tienen ya el conocimiento necesario para desarrollar contratos inteligentes, ya que no tienen la necesidad de aprender un nuevo lenguaje de propósito específico como ocurre en Ethereum con su lenguaje Solidity.
- **Está basado en permisos.** Esto quiere decir que, al contrario que en las redes blockchain públicas, los participantes se conocen los unos a los otros. Pero esto no significa que dichos participantes confíen plenamente entre sí (por ejemplo, competidores en una misma industria), por lo que la red basada en permisos actúa como un acuerdo legal o un marco para la gestión de disputas entre las partes implicadas.
- **No requiere una criptomoneda.** Al estar dentro de una red de consorcio (varias organizaciones) o red privada (una única organización), no es necesario incentivar a los mineros (Bitcoin) o pagar por la ejecución del contrato inteligente (uso de gas en Ethereum), como ocurre en las redes públicas.
- **Mejor rendimiento.** En términos de latencia en el procesamiento y confirmación de las transacciones.
- **Privacidad y confidencialidad.** La falta de confidencialidad ligada a los algoritmos de las redes blockchain públicas como *Proof-of-Work* puede ser problemática para muchos casos de usos empresariales. En Hyperledger Fabric se soluciona el problema de la confidencialidad con los llamados *channels*, que

permiten a un subconjunto de participantes ver un conjunto concreto de transacciones. De esta manera, solo esos nodos que participen en el *channel* tendrán acceso a dichas transacciones.

### 2.3.2. Hyperledger Composer

Hyperledger Composer [10] es un amplio conjunto de herramientas que hace más rápido y sencillo el desarrollo de aplicaciones con blockchain. Su objetivo principal es acelerar y simplificar la integración de nuevas soluciones blockchain con los sistemas de negocio ya existentes en las empresas, permitiendo el desarrollo de casos de usos y el despliegue de aplicaciones blockchain en semanas (en lugar de meses, como ocurre con otras plataformas).

**Composer utiliza** la infraestructura y el runtime de la blockchain que proporciona el **framework Hyperledger Fabric**. Gracias a Fabric las transacciones se validan de acuerdo con la política diseñada por los participantes de la red empresarial.

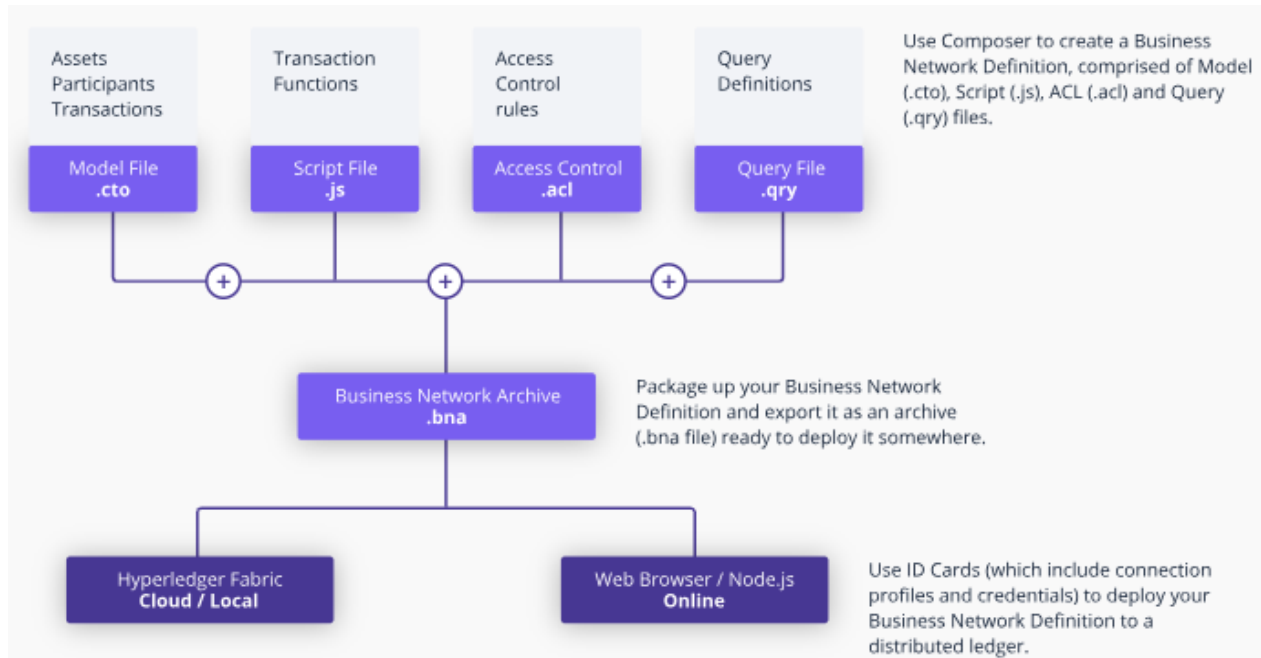
En resumen, usar Hyperledger Composer significa definir una red de negocio, denominadas ***Business Network Definitions*** en esta herramienta. Tal y como se puede observar en la Figura 3, una ***Business Network Definition*** tiene 4 componentes: modelo, script, lista de control de acceso y consultas. Una vez estos 4 componentes se crean y se definen de acuerdo con la política diseñada, se empaquetan y se obtiene un ***Business Network Archive*** (archivo .bna). Con el archivo empaquetado lo único que queda es desplegarlo en la blockchain que proporciona Hyperledger Fabric.

Entrando un poco más en detalle, los cuatro archivos que componen una *Business Network Definition* se encargan de:

- **Modelo (.acto)**. Aquí es donde se definen todos los recursos del sistema: activos/bienes, los participantes, las transacciones y los eventos.
- **Script (.js)**. Se usa el lenguaje de programación JavaScript para especificar el comportamiento de las transacciones definidas en el modelo.
- **Lista de control de acceso (.acl)**. Este archivo regula que participantes pueden acceder a qué recursos y con qué tipo de permisos (sin permisos, lectura, escritura, lectura/escritura, borrado, etc.) acceden a dichos recursos.



- **Consultas (.qry).** Aquí se definen consultas (como en las bases de datos tradicionales) para consultar la información almacenada en los registros de los recursos (bienes y participantes).



*Figura 3. Creación y despliegue de una Business Network Definition.*

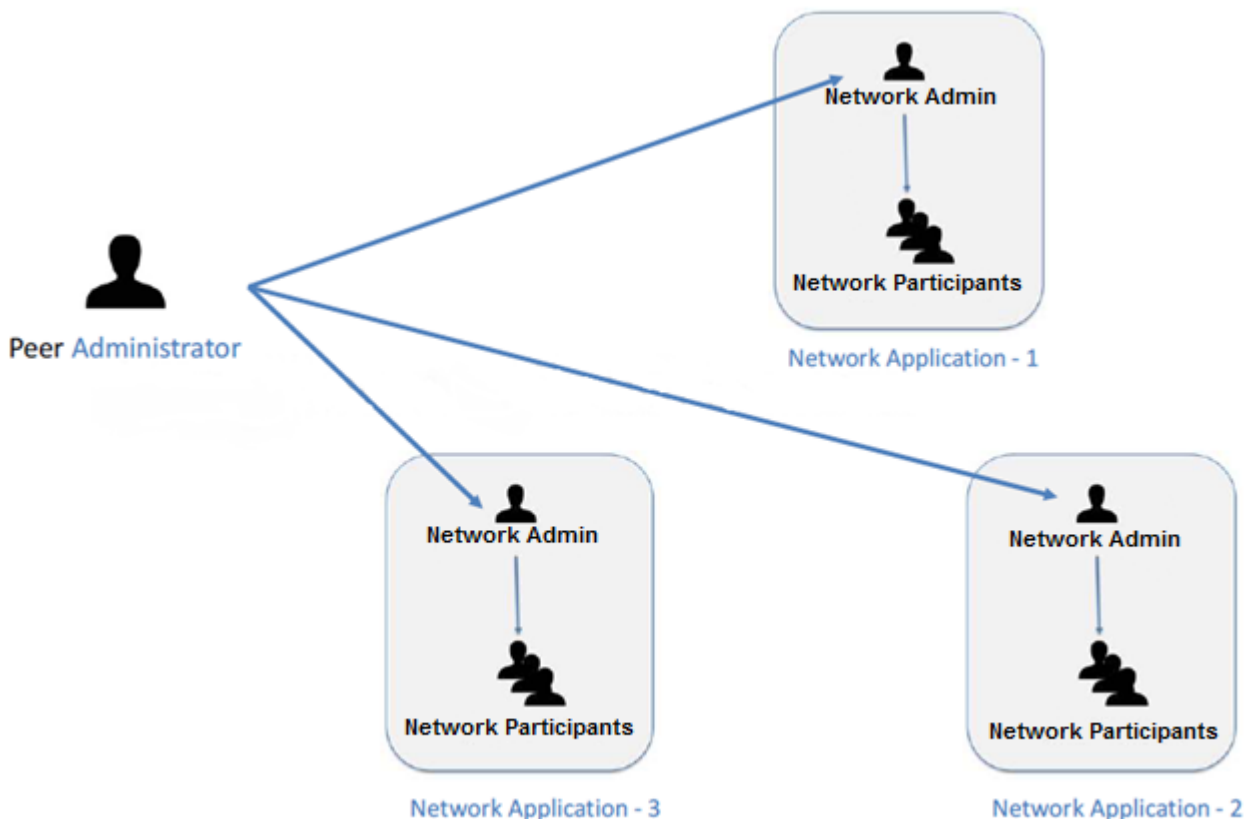
Para desplegar estos archivos e interactuar con la blockchain mediante las transacciones definidas, se utilizan las denominadas **Business Network Cards** (también conocidas como **ID Cards**). Una *Business Network Card* está asociada a un participante y proporciona toda la información necesaria para conectarse a una *Business Network*: identificador del participante, permisos, credenciales, etc. Todas estas *Cards* pueden exportarse y compartirse con otras personas, permitiendo que estas personas se conecten a la *Business Network* usando la identidad de la *Card* en cuestión.

En Composer existen tres tipos de participantes principales: *Peer Administrator*, *Network Administrator* y *Network Participants*.

- **Peer Administrator.** Responsable de llevar a cabo actividades a nivel de nodo y de infraestructura. Se crea durante la instalación de Fabric. Entre otras funciones, se encarga de desplegar nuevas *Business Networks* y de actualizar las definiciones de estas redes.

- **Network Administrator.** Responsable de llevar a cabo actividades administrativas a nivel de aplicación. Cada vez que se despliega una nueva red, Composer genera automáticamente una Business Network Card para poder ejecutar tareas administrativas. Las tareas principales de administración de una red son crear nuevos participantes en la red y emitir identidades para estos.
- **Network Participants.** Son los usuarios de las aplicaciones de estas redes. Pueden tener diversos roles, definidos como parte del modelo. Por lo tanto, las acciones que pueden llevar a cabo los participantes dependen del rol que tengan. Además, pueden crear otros participantes si el *Network Administrator* les concedió ese permiso cuando los creó.

En la Figura 4 podemos observar como los diferentes tipos de participantes en Hyperledger Composer se relacionan entre sí. Un **Peer Administrator** puede desplegar varias redes con sus respectivos **Network Administrators**, que a su vez crean **Network Participants** para que estos últimos interactúen con la red mediante las transacciones definidas en el modelo.



*Figura 4. Relaciones entre los tipos de participantes en Hyperledger Composer.*

### 2.3.3. Angular

Angular [11] es un framework que se utiliza para crear y mantener aplicaciones web de una sola página (en inglés, *Single-page application*). Basado en TypeScript y de código abierto, este marco de trabajo está mantenido por el Angular Team de Google y por su comunidad de desarrolladores individuales y corporaciones. Este equipo también fue el autor del framework AngularJS y fueron los encargados de reescribir Angular basándose en su predecesor.

Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo por hacer que el desarrollo y las pruebas sean más fáciles. Su primera versión, Angular 2.0, fue lanzada el 14 de septiembre de 2016. El lanzamiento de la última versión estable, Angular 7.0, tuvo lugar el 18 de octubre de 2018.

Una aplicación realizada en Angular está formada por un conjunto de componentes. Un componente es una clase TypeScript (con el decorador **@Component**) que cumple una tarea específica dentro de la aplicación. Por ejemplo, un componente sería el menú de navegación o la barra lateral de una aplicación web, tal y como aparece en la Figura 5. De este modo, si se necesitara hacer algún cambio de en el menú de navegación, simplemente tendríamos que modificar la clase del componente en cuestión.



*Figura 5. Componentes en Angular (ejemplo).*

#### 2.3.4. Firebase

Firebase [12] es una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles desarrollada por James Tamplin y Andrew Lee en 2012. 2 años más tarde, en 2014, Firebase fue adquirida por Google. A fecha de octubre de 2018, esta plataforma cuenta con 18 productos, los cuales son usados por más de un millón y medio de aplicaciones.

Sus productos (o servicios) se dividen en 4 grandes ramas: desarrollo, calidad, analíticas y crecimiento.

- **Servicios de desarrollo:** autenticación, bases de datos, almacenamiento, hosting o ML Kit (sistema de machine learning para dispositivos móviles).
- **Servicios de calidad:** controlan la calidad de la aplicación. Los productos más destacados son **Crashlytics** (genera informes detallados de errores en la aplicación), **Firebase Performance** (proporciona información del rendimiento de las aplicaciones) y **Firebase Test Lab** (para el testing de aplicaciones móviles como iOS o Android).
- **Servicios de analíticas:** proporcionan una visión profunda sobre el uso de la aplicación por parte de los usuarios. Producto principal de esta rama: **Firebase Analytics**.
- **Servicios de crecimiento:** esta rama de los productos de Firebase ayuda al desarrollador de la aplicación a comprender el comportamiento que tendrán los usuarios de esta (**Firebase Predictions**, que usa los datos de Analytics) o a sacar un mayor rendimiento económico a la aplicación a través de anuncios (**AdMob**).

#### 2.4. Arquitectura del sistema

Para terminar este apartado de la memoria, vamos a hablar de la arquitectura de este sistema de cadena de custodia, y de cómo los tres componentes de dicho sistema interaccionan entre ellos.

Por una parte, tenemos la **blockchain de Hyperledger Fabric**, en la cual se despliega la **Business Network Definition** modelada y empaquetada con la herramienta

Hyperledger Composer. Esta definición está formada por los ficheros de modelo, script, lista de control de acceso y consultas (ver Figura 3).

Por otra parte, está la **aplicación web** que permite a las entidades participantes en el proceso de la cadena de custodia intervenir en él. Por último, tenemos el **repositorio de ficheros** en el que se almacenan las *master copies* de las pruebas.

Cuando un usuario sube una nueva evidencia al sistema, la aplicación web calcula el hash de dicha prueba y, con las credenciales del usuario en cuestión, trata de añadirla a la cadena de bloques. Si esta anexión tiene éxito, la aplicación web envía la prueba (*master copy*) al repositorio de ficheros para almacenarla allí. Este flujo se puede observar en la Figura 6.

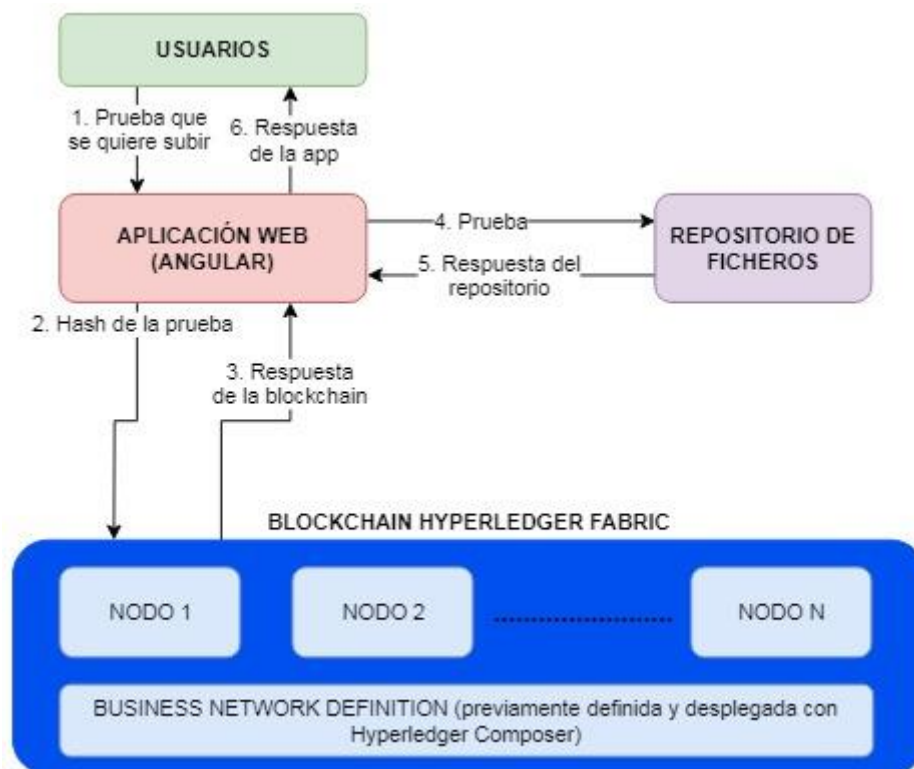
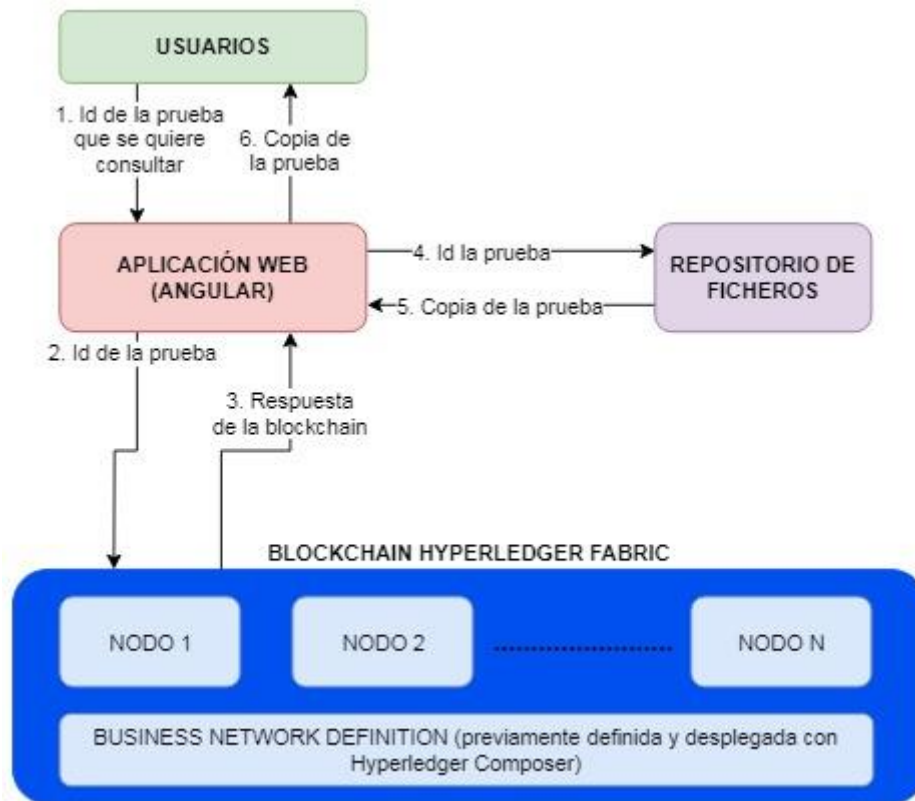


Figura 6. Flujo de datos en el sistema durante la subida de una nueva prueba.

En el caso de que el usuario desee consultar una evidencia ya existente en el sistema, el flujo de datos es diferente, tal y como se ve en la Figura 7. En primer lugar, la aplicación web consulta en la cadena de bloques si existe o no esa prueba a partir de su identificador. Si existe y el usuario tiene permisos para su consulta, la aplicación

requiere una copia de la evidencia al repositorio de ficheros. Una vez se recibe la copia, se descarga automáticamente en el navegador web del usuario.



*Figura 7. Flujo de datos en el sistema durante la consulta de una prueba ya existente.*

### 3. Definición de la **Business Network**

---

En este apartado se profundizará en la definición de los cuatro archivos que forman una *Business Network* de Hyperledger Composer. Como ya sabemos por lo explicado anteriormente, estos archivos son el modelo del sistema en cuestión (proceso de cadena de custodia), el script en el que se codifica el comportamiento de las transacciones definidas en el modelo, la lista de control de acceso y el archivo de consultas.

#### 3.1. **Modelo del Sistema**

Para especificar este modelo se utiliza el **Modeling Language de Hyperledger Composer** [13], que es un lenguaje orientado a objetos cuyo único propósito es permitir la definición de modelos para una Business Network. Este tipo de archivos (con extensión .cto) tiene los siguientes elementos: un único namespace y un conjunto de definiciones de participantes, activos, transacciones y eventos que pertenecen a dicho namespace. Además, es posible utilizar tipos enumerados y conceptos (clases abstractas que suelen ir contenidas en la definición de un participante, un activo o una transacción). Opcionalmente, se pueden declarar importaciones de recursos de otros namespaces.

En este caso se ha optado por definir el modelo en un único fichero CTO con su correspondiente namespace, **uma.coc.network**. En el Anexo A se encuentran todos los detalles a nivel de código del modelo, con algunas figuras que permiten clarificar la implementación. La definición de este modelo para el proceso de cadena de custodia cuenta con los siguientes:

- **Participantes.** Por un lado, tenemos la clase abstracta **CoCParticipant** con un único atributo, *participantId*, para identificar al participante dentro de este sistema. En el modelo diseñado existen dos clases que heredan de **CocParticipant**, **Agent** y **Deposit**. La clase **Agent** reúne todos los datos sobre un agente: nombre, apellidos, fecha de nacimiento, sexo, puesto (que puede ser detective, oficial o técnico forense), formación y oficina en la que trabaja. La clase **Deposit** por su parte representa un depósito de pruebas y tiene sólo un atributo, la oficina a la que pertenece.

- **Activos.** Se han definido dos tipos de activos, uno para representar pruebas (**Evidence**) y otro para representar casos (**Case**), ambos identificados por identificadores únicos de prueba y caso, respectivamente. Una prueba, además de su identificador, incluye el hash de la prueba, el tipo de hash, una breve descripción, la fecha de creación, el propietario actual de la prueba, los propietarios anteriores y el caso al que pertenece. Por su parte, la clase **Case** tiene como atributos un identificador, una descripción, las fechas de apertura y cierre del caso, una resolución, el estatus actual (abierto o cerrado), la referencia al participante que abrió el caso y una lista de participantes.
- **Transacciones y Eventos.** Las transacciones son las interacciones que los participantes tienen con la blockchain y gracias a las cuales pueden consultar o modificar los activos existentes en la red. Aunque se definen en el modelo, es en el script donde se define el comportamiento de las transacciones. Los eventos se definen también en el modelo y son emitidos por las transacciones en el script donde estas últimas se implementan. Es por esto por lo que las transacciones y eventos se han definido por parejas y permiten: abrir y cerrar un caso, incluir nuevos participantes a un caso, añadir pruebas a un caso y transferir estas evidencias de un participante a otro.

### **3.2. Script de las transacciones**

En este fichero JavaScript se codifica el comportamiento de las cinco transacciones definidas en el modelo. A continuación, se van a exponer las singularidades de cada una de estas transacciones.

- **OpenCase.** Permite a un detective o a un oficial abrir un nuevo caso. Se crea un activo **Case** y añade al agente que abre el caso y al depósito de la oficina de este agente como participantes involucrados en el caso. Si el participante que ejecuta esta transacción no es un agente (clase **Agent**) cuyo puesto sea detective u oficial, se lanza una excepción.
- **CloseCase.** Esta transacción permite cerrar un caso ya abierto, transfiriendo todas sus evidencias al depósito de pruebas. Si cualquier otro participante que no sea el agente que abrió el caso ejecuta esta transacción, se lanza una excepción.



- **AddParticipant.** El agente que abrió el caso puede añadir otros participantes para que puedan participar en él. Si el caso ya está cerrado o el participante que ejecuta esta transacción no es el agente que abrió el caso, se lanza una excepción.
- **AddEvidence.** Cualquier agente previamente involucrado en un caso puede añadir pruebas ligadas a dicho caso, si este aún está abierto. Además, este agente se convierte en el primer propietario temporal de la evidencia que se registra. Si el participante que ejecuta esta transacción no es un agente o es un agente que no participa en el caso, se lanza una excepción.
- **TransferEvidence.** El propietario temporal de una prueba puede transferirla a otro participante (agente o depósito) que también participe en el caso que atañe a la evidencia. Si la transacción es ejecutada por un participante que no es el propietario actual de la prueba o el destinatario no está involucrado en el caso en cuestión, se lanza una excepción.

### 3.3. *Lista de control de acceso*

Dado que la blockchain utilizada está basada en permisos, es necesario definir una lista de control de acceso para limitar la interacción del usuario con la cadena de custodia implementada. Al igual que en otros sistemas informáticos, la lista de control de acceso de Hyperledger Composer permite determinar los permisos de acceso apropiados a un determinado registro de un recurso (tanto de participantes como de activos).

Esta lista de control de acceso está formada por un conjunto de reglas. Además, cada regla tiene **cinco atributos obligatorios**, que son los siguientes:

- **description:** una breve descripción textual de la regla.
- **participant:** participante al que se le otorga o deniega el permiso.
- **resource:** recurso al que se quiere acceder.
- **operation:** tipo de acceso al recurso en cuestión (creación, lectura, modificación, borrado o todos).
- **action:** permitir o denegar el acceso.

Adicionalmente, algunas reglas pueden tener algunos de los **dos atributos opcionales**:

- **transaction:** si este atributo aparece en la regla, solo se permite el tipo de acceso al recurso al participante si dicho participante ejecuta la transacción especificada en este atributo.
- **condition:** se trata de una expresión booleana escrita en JavaScript que cuando es verdadera accede o deniega el acceso al recurso al participante.

Si no hay una regla definida en la lista de control de acceso, **Hyperledger Composer deniega por defecto el acceso a los recursos**. Si existiesen dos o más reglas que involucrasen al mismo participante y recurso, se tendrá en cuenta la primera y se otorgará o denegará el acceso al recurso según lo definido en dicha regla. En la Figura 8 se adjunta una de las reglas definidas para la lista de control de acceso. El resto de las reglas se pueden consultar en el código adjunto en el disco entregado con esta memoria.

```
rule AgentsCanOpenCaseRule2 {  
  description: "Allow agents (officers or detectives) to create an asset of type Case when executing tx OpenCase"  
  participant(p): "uma.coc.network.Agent"  
  operation: CREATE  
  resource: "uma.coc.network.Case"  
  transaction: "uma.coc.network.OpenCase"  
  condition: (p.job === 'OFFICER' || p.job === 'DETECTIVE')  
  action: ALLOW  
}
```

*Figura 8. Ejemplo de regla para la lista de control de acceso.*

En total se han definido 16 reglas, que se van a explicar a continuación:

- **MandatoryRule:** regla obligatoria que permite a todos los participantes leer los recursos del sistema (versión de las *Business Networks*, sus identidades en la blockchain, etc.)
- **SystemResourcesControlPermission:** permite al administrador de la *Business Network* llevar a cabo la gestión de identidades y operaciones administrativas en la red.
- **NetworkControlPermission:** permite al administrador de la *Business Network* la gestión de los recursos de la red de negocios.
- **ParticipantsCanReadRule:** otorga a los participantes **CocParticipant** el acceso de lectura a los recursos de la *Business Network*.

- **ParticipantsCanExecuteTxRule**: permite a los participantes **CocParticipant** la ejecución de transacciones.

Las siguientes reglas que aparecen en la Tabla 1 son las necesarias para que los participantes puedan ejecutar las transacciones definidas, así como puedan crear y actualizar *assets* durante la ejecución de dichas transacciones.

*Tabla 1. Utilidad de las reglas definidas para la correcta ejecución de las transacciones.*

Transacción	Regla	Utilidad
OpenCase	<b>AgentsCanOpenCaseRule</b>	Permitir a detectives u oficiales ejecutar la transacción OpenCase.
	<b>AgentsCanOpenCaseRule2</b>	Permitir a detectives u oficiales crear assets de tipo Case ejecutando OpenCase.
CloseCase	<b>AgentsCanCloseCaseRule</b>	Permite a agentes ejecutar la transacción CloseCase.
	<b>AgentsCanCloseCaseRule2</b>	Permite al agente que abrió el caso modificarlo (cambiando su estado a cerrado) ejecutando CloseCase.
	<b>AgentsCanCloseRule3</b>	Permite a los agentes modificar assets de tipo Evidence durante la ejecución de CloseCase.
AddParticipant	<b>AddParticipantRule</b>	Permite a agentes ejecutar la transacción AddParticipant.
	<b>AddParticipantRule2</b>	Permite al agente que creó el caso modificarlo (añadiendo nuevos participantes) ejecutando AddParticipant.
AddEvidence	<b>AddEvidenceRule</b>	Permite a agentes ejecutar la transacción AddEvidence.
	<b>AddEvidenceRule2</b>	Permite a los agentes involucrados en el caso crear assets de tipo Evidence cuando ejecutan AddEvidence.

TransferEvidence	TransferEvidenceRule	Permite a participantes de tipo CoCParticipant ejecutar la transacción TransferEvidence.
	TransferEvidenceRule2	Permite al propietario temporal de la evidencia modificar dicho asset de tipo Evidence al ejecutar TransferEvidence.

### 3.4. Consultas

En Hyperledger Composer existen dos tipos de consultas: las **Named Queries** y las **Dynamic Queries**. La diferencia entre ambas es que las primeras se definen como parte de la Business Network en el fichero de consultas, mientras que las segundas se construyen de manera dinámica, durante la ejecución de las transacciones.

Como el código de las transacciones ya es de por sí bastante amplio, se ha optado por utilizar **Named Queries**. De esta forma en el script de transacciones se llama a consultas existentes en el archivo **queries.qry**. En total se han definido cuatro consultas (ver Figura 9):

- **EvidencesByCase**: lista todas las pruebas pertenecientes a un caso.
- **EvidencesByParticipant**: recupera todas las pruebas cuyo propietario temporal es el participante en cuestión.
- **CasesByParticipant**: lista todos los casos en los que está involucrado un participante, estén o no cerrados los casos.
- **GetDepositByOffice**: recupera el participante de tipo **Deposit** cuya oficina es la que se pasa por parámetro.

```
query EvidencesByCase {  
  description: "Select all evidences of one single case"  
  statement:  
    SELECT uma.coc.network.Evidence  
    WHERE (caso == _$case_fqi)  
}  
  
query EvidencesByParticipant {  
  description: "Select all evidences of one participant"  
  statement:  
    SELECT uma.coc.network.Evidence  
    WHERE (owner == _$owner_fqi)  
}  
  
query CasesByParticipant {  
  description: "Select all cases in which a participant is involved"  
  statement:  
    SELECT uma.coc.network.Case  
    WHERE (participants CONTAINS _$participant_fqi)  
}  
  
query GetDepositByOffice {  
  description: "Get a deposit by office"  
  statement:  
    SELECT uma.coc.network.Deposit  
    WHERE (office == _$office)  
}
```

*Figura 9. Named Queries definidas.*



## 4. Configuración del servidor REST Hyperledger Composer

El servidor REST de Hyperledger Composer es un **proceso NodeJS autónomo** que expone la *Business Network* (es decir, expone sus recursos: participantes, queries, assets, etc.) a través una API REST. De esta forma el desarrollador de aplicaciones web puede obtener información de la Business Network accediendo a los diversos recursos de la API, además de poder invocar las transacciones definidas en el modelo.

Tal y como se observa en la Figura 10, las aplicaciones pueden conectarse directamente al servidor REST en lugar de conectarse con el runtime de Hyperledger Fabric (la blockchain en sí). La ventaja de este enfoque reside en que el desarrollador web no necesita depender de ninguna librería específica de Hyperledger Fabric. De esta forma el código resultante es mucha más simple y sostenible.



*Figura 10. Flujo de datos usando el servidor REST de Hyperledger Composer.*

**Este servidor REST se configura usando variables de entorno.** A continuación, se van a detallar las variables utilizadas en este caso para la configuración del servidor REST:

- **COMPOSER\_CARD.** En Hyperledger Composer una card es un archivo en el que se indica con qué identidad y a qué Business Network se debe conectar un usuario. Por lo tanto, esta variable de entorno sirve para saber que queries, recursos, transacciones, etc., exponer en la API REST. **El valor de esta variable es admin@cocv2**, la card generada automáticamente al desplegar la Business Network cov2, y que corresponde a la identidad del administrador de esta red.

- **COMPOSER\_NAMESPACES.** Sirve para especificar si el servidor REST debe generar la API con namespaces o no. Los posibles valores son `always`, `required` y `never`. En este caso se necesita que se usen los namespaces, por lo que **el valor de esta variable es `always`.**
- **COMPOSER\_AUTHENTICATION.** Sirve para especificar si el servidor REST debe habilitar autenticación en la API o no. Los posibles valores son `true` y `false`. En el caso de que la variable tenga valor `true`, es necesario definir la variable **COMPOSER\_PROVIDERS.** Como se quiere habilitar la autenticación en la API REST, **el valor de esta variable es `true`.**
- **COMPOSER\_PROVIDERS.** Esta variable especifica las estrategias de autenticación de **Passport.js** que el servidor REST debería usar para autenticar a los usuarios de la API REST. En el subapartado 4.1 de esta memoria se profundizará en la estrategia de autenticación utilizada, así como se expondrá el valor que se le ha otorgado a esta variable de entorno.
- **COMPOSER\_MULTUSER.** Sirve para especificar si el servidor REST debe habilitar el modo multi usuario o no. Los posibles valores son `true` y `false`. Si no se habilita este modo, todas las transacciones se ejecutarán con la identidad de la card indicada en la variable de entorno **COMPOSER\_CARD.** Este no es el comportamiento deseado para nuestro sistema, por lo que **el valor de esta variable es `true`. Para habilitar el modo multi usuario es necesario haber configurado previamente la autenticación en el servidor REST.**
- **COMPOSER\_DATASOURCES.** Esta variable indica al servidor REST la configuración a una base de datos, en la que se almacenan las credenciales de los diferentes usuarios identificados en el servidor. De este modo, cada vez que un usuario se autentique en el servidor REST, este sabe que identidad debe utilizar para ejecutar las transacciones. En el subapartado 4.2 se profundiza en este aspecto.



- **COMPOSER\_TLS.** Esta variable indica al servidor REST si debe o no habilitar HTTPS y TLS. Los posibles valores son true y false. En el caso de que la variable tenga valor true, es recomendable definir las variables que hacen referencia al certificado y a la clave usadas para asegurar el servidor REST de Composer, **COMPOSER\_TLS\_CERTIFICATE** y **COMPOSER\_TLS\_KEY**. Como se desea habilitar HTTPS y TLS, **el valor de esta variable es true**. En el subapartado 4.3 de esta memoria se detallan las particularidades de esta variable, así como la generación del certificado y clave privada usados.
- **COMPOSER\_TLS\_CERTIFICATE.** En esta variable se especifica la ruta en la que se encuentra el certificado que debe usar el servidor REST cuando HTTPS y TLS están habilitados.
- **COMPOSER\_TLS\_KEY.** En esta variable se especifica la ruta en la que se encuentra la clave privada que debe usar el servidor REST cuando HTTPS y TLS están habilitados.

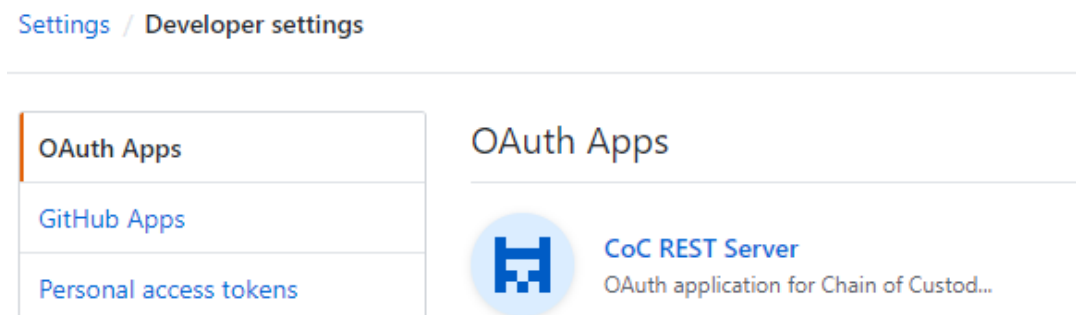
Junto con la memoria de este TFM se adjuntan los scripts desarrollados para ejecutar el servidor REST con los valores deseados en las variables de entorno que se acaban de exponer. Estos scripts se han nombrado como **rest-server-auth.sh**, **multi-user.sh** y **secure-server.sh**.

#### **4.1. Estrategia de autenticación utilizada**

El servidor REST de Hyperledger Composer puede ser configurado para autenticar clientes. Cuando esta opción está habilitada, los clientes deben autenticarse en el servidor REST antes de poder realizar llamadas a la API REST.

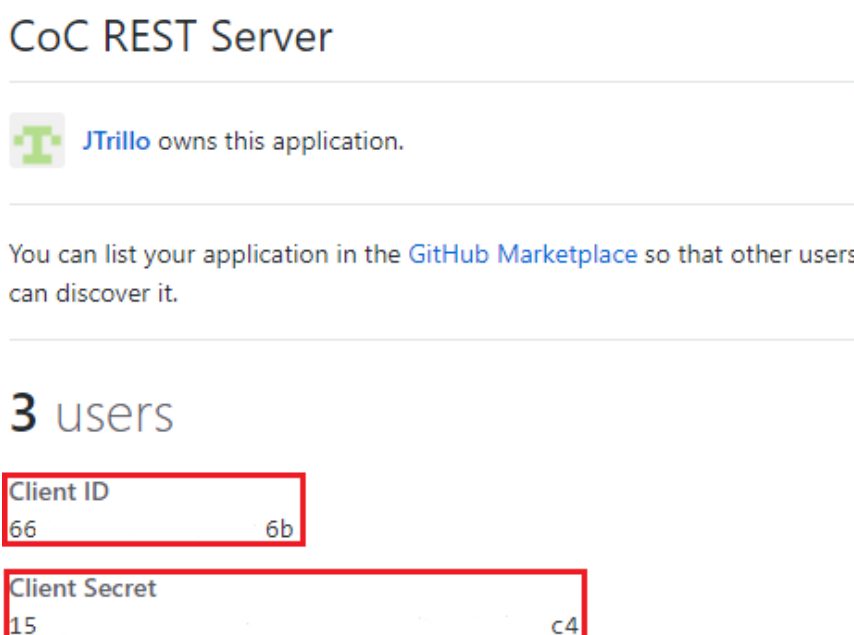
Para poder llevar a cabo estas tareas de autenticación, el servidor REST **usa el middleware de autenticación Passport.js**. Este middleware cuenta con un gran número de estrategias de autenticación (más de 300 cuando se redactó esta memoria) que se pueden encontrar en [14]. **En este proyecto se ha optado por utilizar la estrategia de Github**. Los pasos que se deben seguir para configurar correctamente esta estrategia de autenticación son los siguientes:

- **Registrar una nueva aplicación de OAuth en Github.** OAuth es un estándar que permite flujos simples de autorización para sitios web o aplicaciones informáticas. Github permite de forma rápida y sencilla crear nuevas aplicaciones OAuth, solicitando unos pocos parámetros para la configuración de la mencionada aplicación. En las opciones de desarrollador de Github (Developer settings, ver Figura 11) es posible crear nuevas aplicaciones OAuth.



*Figura 11. Aplicaciones OAuth en Github.*

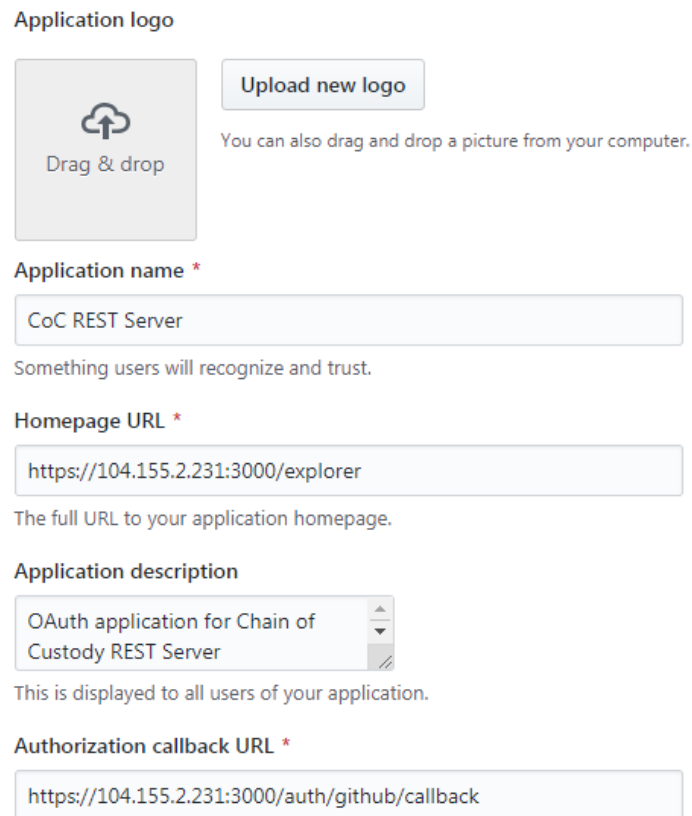
Esta aplicación genera un **client ID** y un **client secret**, los cuales son parámetros necesarios a la hora de configurar la estrategia de autenticación de Passport.js. En la Figura 12 se observan ambos parámetros de la aplicación OAuth de Github registrada para el servidor REST de Hyperledger Composer.



*Figura 12. Client ID y Client Secret de la aplicación OAuth registrada.*

- **Configurar los parámetros de la aplicación OAuth de Github.** Una vez se ha creado la aplicación OAuth, hay que cumplimentar 5 campos, que son los que se enumeran a continuación: logo de la aplicación, nombre de la aplicación, URL de la página de inicio, descripción de la aplicación y URL del callback de autorización.

Ni el logo ni la descripción son campos obligatorios, aunque en este caso se ha optado por incluir una breve descripción para que cuando los nuevos usuarios inicien sesión con su cuenta de Github sepan cual es el propósito de la aplicación que están utilizando. En la Figura 13 se puede comprobar la configuración escogida para la aplicación OAuth de Github.



Application logo

Upload new logo

Drag & drop

You can also drag and drop a picture from your computer.

Application name \*

CoC REST Server

Something users will recognize and trust.

Homepage URL \*

https://104.155.2.231:3000/explorer

The full URL to your application homepage.

Application description

OAuth application for Chain of Custody REST Server

This is displayed to all users of your application.

Authorization callback URL \*

https://104.155.2.231:3000/auth/github/callback

*Figura 13. Configuración de la aplicación OAuth de Github registrada.*

- **Informar la variable de entorno COMPOSER\_PROVIDERS.** En este último paso solo queda darle valor a la variable de entorno COMPOSER\_PROVIDERS. Esta variable es un JSON que cuenta con los campos que vemos en la Figura 14. A continuación, se van a describir brevemente algunos de estos campos:
  - **provider** → nombre del proveedor de autenticación.

- **module** → es el nombre del módulo de Passport.js que se encarga de realizar la autenticación.
- **clientID** → client ID de la aplicación OAuth de Github.
- **clientSecret** → client secret de la aplicación OAuth de Github.
- **authPath** → ruta del servidor REST que se debe de llamar para realizar la autenticación.
- **callbackURL** → dirección del callback de autenticación. Esta dirección debe ser la misma que la indicada en la aplicación OAuth de Github.
- **successRedirect** → dirección a la que debe redirigirse si la autenticación ha tenido éxito. En este caso es la dirección en la que se encuentra la aplicación web para el registro y consulta de evidencias digitales.

```
export COMPOSER_PROVIDERS='{
  "github": {
    "provider": "github",
    "module": "passport-github",
    "clientId": "66",
    "clientSecret": "15",
    "authPath": "/auth/github",
    "callbackURL": "/auth/github/callback",
    "successRedirect": "https://coc-app.firebaseio.com/login",
    "failureRedirect": "/"
  }
}'
```

**Figura 14. Variable COMPOSER PROVIDERS.**

## 4.2. *Habilitando el modo multiusuario*

Por defecto, **el servidor REST de Hyperledger Composer utiliza la identidad con la que se arranca (COMPOSER\_CARD) para firmar todas las transacciones**, lo que implica que la Business Network no distinguirá entre los diferentes clientes que se comuniquen con el servidor REST. Esto puede ser aceptable en ciertos casos de uso, si la identidad con la que se arranca el servidor REST solo tiene permisos de lectura. Pero en este sistema de cadena de custodia **es necesario que todos los usuarios ejecuten transacciones y, por tanto, que firmen con sus identidades dichas transacciones.**

Este es el motivo por el cual es necesario arrancar el servidor REST en modo multiusuario. Para habilitar este modo es necesario seguir una serie de pasos, como ocurre si se quiere habilitar la autenticación.

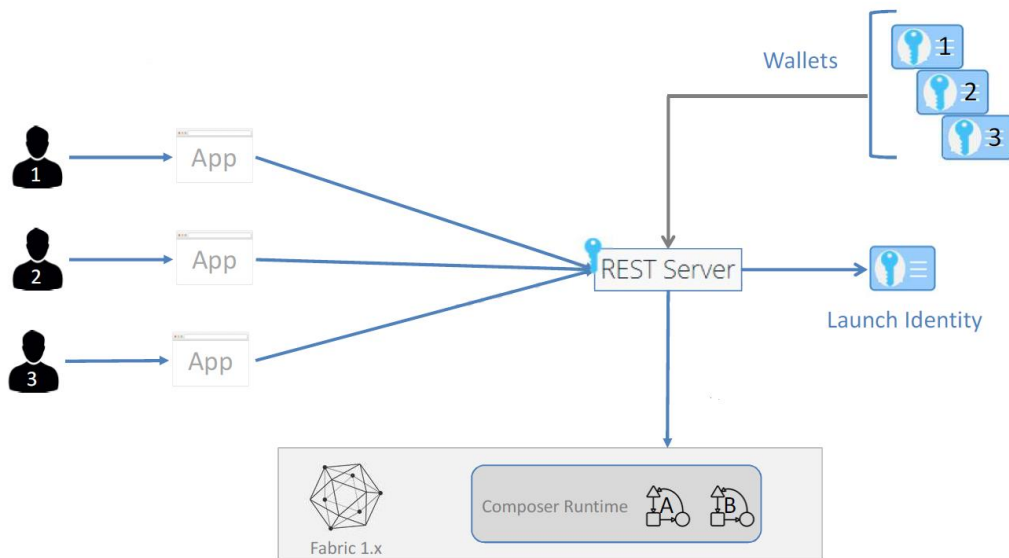
- En primer lugar, **es necesario** seguir los pasos indicados en el subapartado 4.1 de esta memoria para **habilitar la autenticación** en el servidor REST.
- En segundo lugar, **es necesario configurar un sistema de almacenamiento persistente para la gestión de wallets**. En Hyperledger Composer, cada usuario autenticado tiene un **wallet**, que es un conjunto de **cards** (identidades, una por cada *Business Network* en la que dicho usuario participe).

**Son por tanto estas cards las que se han de guardar de forma persistente.**

De esta forma, cada vez que un usuario se autentique con su cuenta de Github en el servidor REST, este último asociará al usuario con su wallet, y de su wallet cogerá la card perteneciente a la *Business Network* del sistema de cadena de custodia desarrollado. Con esta card se firmarán las transacciones que ejecute dicho usuario en la *Business Network*.

**Es importante resaltar que la primera vez que un usuario se autentica en el servidor REST no tiene ninguna card almacenada, por lo que en esta primera conexión es necesario guardar en el almacenamiento persistente una card.**

En la Figura 15 se presenta la arquitectura que sigue el servidor REST cuando el modo multiusuario está habilitado, en donde la card (identidad) utilizada para arrancar el servidor deja de ser la encargada para firmar todas las transacciones que se ejecuten.



*Figura 15. Arquitectura del servidor REST cuando el modo multiusuario está habilitado.*

El servidor REST de Hyperledger Composer permite usar cualquier base de datos NoSQL para el almacenamiento persistente de las cards, aunque las bases de datos NoSQL recomendadas para llevar a cabo esta tarea son MongoDB y CouchDB. En este caso **se ha optado por utilizar MongoDB, concretamente MongoDB Atlas**, un servicio cloud de este tipo de base de datos NoSQL.

- A continuación, hay que **instalar el módulo conector loopback**, necesario para que el servidor REST se conecte a la base de datos en la que están guardadas las cards. En este caso se usará el módulo conector para MongoDB, que se instala con el sistema de gestión de paquetes por defecto para Node.js, npm. El comando para instalar el módulo conector es el siguiente:

```
npm install -g loopback-connector-mongodb
```

- Por último, hay que otorgarle un valor a **la variable de entorno que gestiona la conexión a la base de datos, COMPOSER\_DATASOURCES**. Esta variable también es un JSON con dos parámetros, **url**, que es la dirección del almacenamiento persistente, y **connector**, que es el conector que utiliza el servidor REST para conectarse al almacenamiento persistente. En la Figura 16 se muestra el valor otorgada a la variable COMPOSER\_DATASOURCES.

```
export COMPOSER_DATASOURCES='{
  "db": {
    "url": "mongodb+srv://[redacted]:[redacted]@cluster0-mmujw.gcp.mongodb.net/test",
    "connector": "loopback-connector-mongodb"
  }
}
```

*Figura 16. Variable de entorno COMPOSER\_DATASOURCES.*

### 4.3. Asegurando el servidor usando HTTPS y TLS

A la hora de desplegar el servidor REST de Hyperledger Composer en un entorno de producción, dicho servidor debe asegurarse con HTTPS y TLS. Una vez el servidor REST haya sido configurado para usar HTTPS y TLS, **toda la información transferida entre el servidor y los clientes estará encriptada.**

Para conseguir esta configuración **es necesario proveer al servidor de un certificado y de una clave privada.** Por defecto, el servidor REST de Hyperledger Composer incluye un certificado y una clave privada que facilita el uso durante la etapa inicial de desarrollo. Pero este par certificado-clave **no debe usarse bajo ningún concepto en un entorno de producción.**

Aunque en este Trabajo de Fin de Máster se intenta conseguir un sistema lo más realista posible, **el certificado y clave usados por el servidor han sido generados con openssl.** En las próximas líneas se exponen los pasos seguidos para la generación del certificado y la clave privada que el servidor REST usará para poder habilitar HTTPS y TLS.

- **Generar una autoridad de certificación auto firmada.** Todo certificado debe ser firmado por una autoridad de certificación (de ahora en adelante, CA, del inglés *Certificate Authority*). En este caso, se ha creado una CA auto firmada, primero obteniendo la clave privada (Figura 17), y luego generando un certificado autofirmado (Figura 18).

```
jokinator20@ubuntu-16-04-lts-desktop:~/Escritorio/Certificates$ openssl genrsa -des3 -out myCA.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for myCA.key:
Verifying - Enter pass phrase for myCA.key:
```

*Figura 17. Generación de la clave privada para la CA.*

```
openssl req -x509 -new -nodes -key myCA.key -sha256 -days 730 -out myCA.pem

Enter pass phrase for myCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Málaga
Locality Name (eg, city) []:Málaga
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UMA
Organizational Unit Name (eg, section) []:Ertis
Common Name (e.g. server FQDN or YOUR name) []:My CA
Email Address []:trillo@uma.es
```

*Figura 18. Generación del certificado de la CA (auto firmado).*

- **Generar una clave privada para el servidor.** De forma similar a como se hizo para genera la clave de la CA, se genera una clave privada para el servidor REST (Figura 19).

```
jokinator20@ubuntu-16-04-lts-desktop:~/Escritorio/Certificates$ openssl genrsa -out rest_server.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
...+++
e is 65537 (0x10001)
```

*Figura 19. Generación de la clave privada para el servidor.*

- **Creación de la petición de firma de certificado.** A diferencia del certificado de la CA, el certificado del servidor no es auto firmado. Es la CA la encargada de firmar el certificado del servidor y para ello es necesario crear una petición de firma de certificado (a partir de ahora CSR, siglas inglesas de *Certificate Signing Request*). En la Figura 20 se muestra el comando usado para crear la CSR, así como la información deseada para el certificado (common name, país, provincia, email, etc.).



```
openssl req -new -key rest_server.key -out rest_server.csr

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Málaga
Locality Name (eg, city) []:Málaga
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UMA
Organizational Unit Name (eg, section) []:Ertis
Common Name (e.g. server FQDN or YOUR name) []:104.155.2.231
Email Address []:trillo@uma.es

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

*Figura 20. Creación del CSR para el certificado del servidor.*

- **Generación del certificado del servidor.** Una vez se tiene el CSR, la CA debe generar el certificado firmado para el servidor. Para ello se deberá ejecutar el comando que aparece en la Figura 21. Es importante resaltar que la opción -**Ccreateserial** solo se debe usar la primera vez que nuestra CA firme un certificado. Esta opción lo que hace es crear el archivo **myCA.srl**, que contiene un número de serie. Para próximas firmas de certificados por parte de esta CA, se deberá usar la opción -**Cserial**, seguida del nombre del archivo contenedor del número de serie.

```
openssl x509 -in rest_server.csr -CA myCA.pem -CAkey myCA.key \
> -Ccreateserial -out rest_server.crt -days 730 -sha 256
Signature ok
subject=/C=ES/ST=M\xC3\x83\xC2\xA1laga/L=M\xC3\x83\xC2\xA1
Getting CA Private Key
Enter pass phrase for myCA.key:
```

*Figura 21. Generación del certificado del servidor.*

- **Pasar clave y certificado a formato PEM.** El servidor REST de Hyperledger Composer requiere que tanto la clave privada como el certificado del servidor estén en formato PEM (Figura 22).

```
jokinator20@ubuntu-16-04-lts-desktop:~/Escritorio/Certificates$ \
> openssl x509 -in rest_server.crt -out certificate.pem -outform PEM
jokinator20@ubuntu-16-04-lts-desktop:~/Escritorio/Certificates$ \
> openssl rsa -in rest_server.key -out key.pem -outform PEM
writing RSA key
```

*Figura 22. Conversión de la clave privada y el certificado del servidor a formato PEM.*

#### **4.4. Segundo servidor REST desplegado**

Para poder crear nuevos participantes y poder generar las identidades de estos es necesario disponer de un segundo servidor REST, el cual **permita el acceso a los recursos de la API REST sin tener que realizar la autenticación**. Para arrancar este servidor se utilizará la identidad del administrador de la Business Network del sistema de cadena de custodia, **admin@cocv2**, ya que es el único usuario con permiso para crear nuevos participantes y generar nuevas identidades.

Además, para paliar la ausencia de autenticación, se iniciará el servidor con el comando **-y**, el cual indica la clave que la API REST debe aceptar en todas las peticiones que reciba (en el header **x-api-key**). También se usará la opción **-p**, para arrancar el servidor en puerto diferente al servidor REST principal (que por defecto se inicia en el puerto 3000). En este caso y para simplificar las cosas, se usarán el par clave-certificado por defecto para habilitar HTTPS y TLS en el servidor, usando la opción **-t**. El comando que hay que ejecutar para arrancar este servidor REST secundario es el siguiente:

```
composer-rest-server -c admin@cocv2 -p 3001 -y YOUR_API_KEY -t
```

## 5. Diseño e implementación de la aplicación web para el registro y consulta de evidencias digitales

---

Para poder interactuar con la Business Network desplegada (que se ha detallado en el tercer capítulo de esta memoria) se ha diseñado e implementado una aplicación web. Esta aplicación permite a los diferentes usuarios del sistema de cadena de custodia ejecutar las diversas transacciones establecidas. Gracias a estas transacciones se posibilita el registro, consulta y transferencias de evidencias entre los participantes de un caso.

Tal y como se ha mencionado a lo largo de este documento, se ha utilizado el framework Angular para la implementación de esta aplicación web. También se ha usado la biblioteca Bootstrap 4, que permite la creación de interfaces limpias y con un diseño responsive.

En este capítulo de la memoria se exponen los requisitos de la aplicación, los diferentes casos de uso y las comunicaciones de esta aplicación con otros elementos del sistema implementado.

### 5.1. Requisitos de la aplicación

Antes de comenzar con el diseño, se establecieron los requisitos de la aplicación para el registro y consulta de evidencias digitales. Se distinguen dos grupos principales de requisitos: funcionales y no funcionales.

#### 5.1.1. Requisitos funcionales

Los requisitos funcionales definen las funciones del sistema software. Estas funciones además están fuertemente influenciadas por la Business Network desplegada en el blockchain de Fabric. Estos requisitos funcionales, que quedan reflejados en los casos de uso, son los siguientes:

- **RF1. Envío de solicitudes de registro.** No cualquier persona puede participar en el sistema. Todo interesado deberá rellenar un formulario que posteriormente será evaluado por un administrador del sistema, el cual aceptará o rechazará las solicitudes de registro.

- **RF2. Apertura de nuevos casos.** Los participantes de tipo 'Agent' con puesto de detective u oficial (más información en el apartado 3 y en el Anexo A) podrán abrir nuevos casos.
- **RF3. Acceso a los casos en los que participa.** Los participantes de tipo 'Agent' tendrán acceso a la información de todos los casos en los que participan (casos abiertos) o en los que han participado (casos cerrados).
- **RF4. Consulta de evidencias registradas.** El propietario temporal de la prueba podrá consultarla, así como descargar una copia de dicha evidencia.
- **RF5. Acceso a su perfil personal.** Todos los participantes de tipo 'Agent' podrán visualizar sus datos personales, con los que es posible identificarlo en el sistema.
- **RF6. Inclusión de nuevos participantes a casos ya creados.** La persona que abrió el caso en la aplicación podrá incluir nuevos participantes al mencionado caso.
- **RF7. Registro de nuevas evidencias en casos ya creados.** Cualquier participante podrá registrar nuevas pruebas relacionadas con un caso.
- **RF8. Cierre de casos.** La persona que abrió el caso podrá cerrar el caso en la aplicación.
- **RF9. Transferencias de evidencias entre participantes de un mismo caso.** El propietario temporal de la prueba podrá transferir dicha evidencia a otro participante del caso en cuestión.

### 5.1.2. Requisitos no funcionales

Los requisitos no funcionales se refieren a todos aquellos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento y de diseño. Se han definido los siguientes requisitos no funcionales:

- **RNF1. Apariencia de la aplicación.** La apariencia de la aplicación deberá ser sencilla, clara y con colores simples, de forma que facilite al usuario el manejo de esta.
- **RNF2. Autenticación.** La aplicación debe ofrecer un modo de verificar la identidad de cada uno de los participantes en el sistema. **En esta aplicación la autenticación se realiza gracias al middleware Passport.js, existente en el servidor REST de Hyperledger Composer desplegado.** Tal y como se ha

expuesto en el apartado 4 de esta memoria, Passport cuenta con multitud de estrategias para llevar a cabo el proceso de autenticación, aunque para esta aplicación se ha optado por utilizar **la estrategia de Github**.

- **RNF3. Multidispositivo.** La aplicación debe visualizarse correctamente en cualquier tipo de dispositivo, por lo que el diseño web tendrá que ser *responsive* o adaptable (del inglés, responsive web design). De este modo los contenidos a visualizar se adaptarán al tamaño de la pantalla para maximizar la experiencia de los usuarios.

## 5.2. Casos de uso

Un caso de uso es una secuencia de interacciones que se desarrollan entre un sistema (la aplicación web) y sus actores (los participantes en el sistema de cadena de custodia) en respuesta a un evento que inicia un actor principal sobre el propio sistema. Los casos de uso sirven para especificar el comportamiento de un sistema mediante su relación con los usuarios y/u otros sistemas.

En la Figura 23 se observa el diagrama de casos de uso de la aplicación de registro y consulta de evidencias digitales.

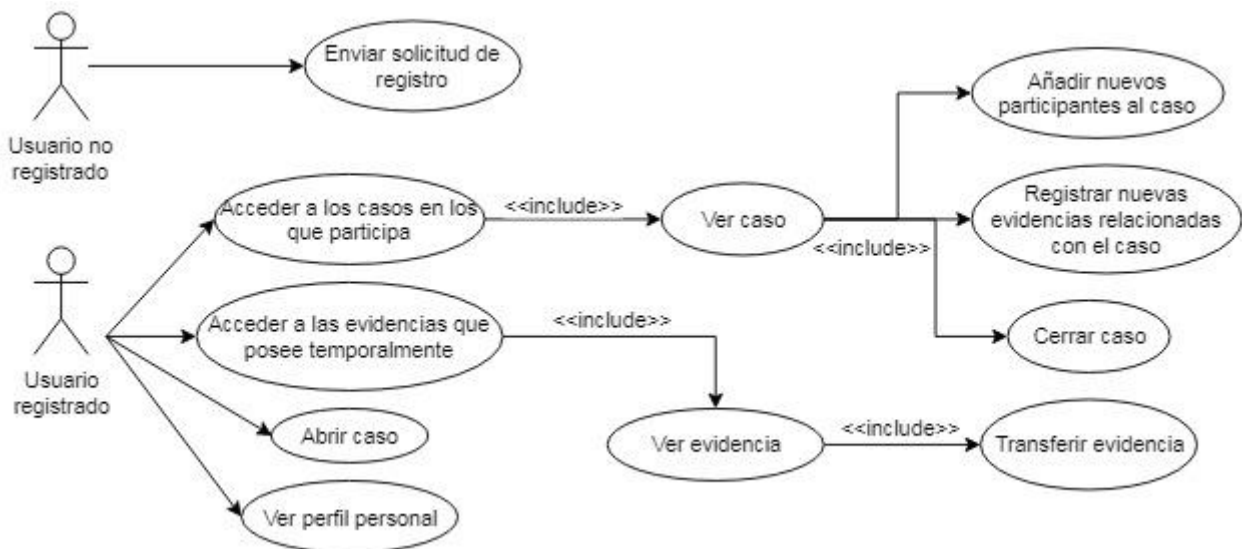


Figura 23. Diagrama de casos de uso de la aplicación de registro y consulta de evidencias digitales.

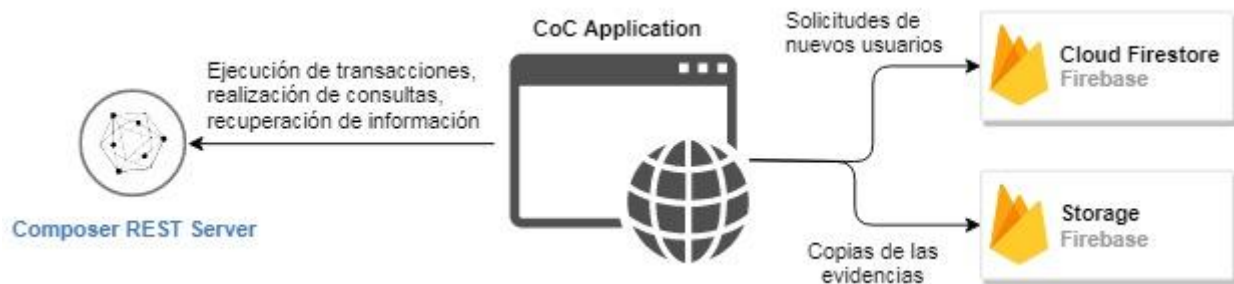
A continuación, se explican brevemente cada uno de los casos de uso de la aplicación descritos en el diagrama.

- **Enviar solicitud de registro.** Es el único caso de uso en el que intervienen los usuarios no registrados. Se les permite rellenar un formulario con sus datos personales, el cual será evaluado posteriormente por un administrador del sistema.
- **Abrir caso.** Los agentes cuyo puesto sea el de detective o el de oficial podrán abrir nuevos casos. Se necesitará incluir una descripción durante esta apertura.
- **Acceder a los casos en los que participa de casos.** Se le muestran al usuario un listado con los casos en los que está involucrado.
- **Acceder a las evidencias que posee temporalmente.** Se le muestran al usuario las pruebas que tiene en ese momento.
- **Ver perfil personal.** Los usuarios podrán visualizar sus datos personales, tales como su puesto de trabajo o la oficina en la que trabajan.
- **Ver caso.** Se muestra toda la información relacionada con el caso: fecha de apertura, participantes, pruebas registradas, descripción, etc.
- **Añadir nuevos participantes al caso.** El agente que abrió el caso puede incluir nuevos participantes a dicho caso.
- **Registrar nuevas evidencias relacionadas con el caso.** Cualquier agente involucrado en el caso puede registrar nuevas evidencias relacionadas con este. Se tendrá que informar del hash de la evidencia, el tipo de hash, una breve descripción de la prueba, así como se tendrá que subir al repositorio de ficheros una copia de la evidencia.
- **Cerrar caso.** El agente que abrió el caso puede cerrarlo, proporcionando previamente una conclusión.
- **Ver evidencia.** El propietario temporal de la prueba podrá chequear toda la información de esta. Además, podrá descargar en su equipo una copia de la evidencia.
- **Transferir evidencia.** El propietario de la evidencia podrá traspasarla a otro agente que participe en el caso al que la prueba en cuestión pertenece.

En el Anexo B de esta memoria se adjuntan capturas de esta aplicación, a modo de guía para los usuarios de esta.

### 5.3. Comunicación con el resto del sistema

Entre los diferentes componentes que forman el sistema de cadena de custodia desarrollado, **esta aplicación web se comunica con el servidor REST de Hyperledger Composer, así como con la plataforma de Google Firebase**, tal y como se observa en la Figura 24.



*Figura 24. Comunicación de la aplicación con otros componentes del sistema.*

Por un lado y gracias al servidor REST de Hyperledger Composer, la aplicación web **puede solicitar datos de los Assets existentes en la Business Network, consultar información específica gracias a las Named Queries definidas** (por ejemplo, obtener todas las evidencias pertenecientes a un caso), **o solicitar la ejecución de las diferentes transacciones** especificadas.

Por otro lado, se usan dos servicios de desarrollo de la plataforma Firebase:

- **Firebase Storage.** Este servicio permite el almacenamiento de las master copies de las evidencias digitales.
- **Firebase Cloud Firestore.** Este servicio permite el almacenamiento de las solicitudes de nuevos usuarios.





## 6. Diseño e implementación de la aplicación web para la administración de solicitudes de nuevos usuarios

---

Esta segunda aplicación desarrollada sirve para que los administradores del sistema puedan visualizar las solicitudes de nuevos usuarios y gestionar estas peticiones, ya sea aceptándolas o rechazándolas. En el caso de que la solicitud sea aceptada, se interactuará con la *Business Network* desplegada, creando un nuevo participante y generando una identidad (card) para dicho participante.

Del mismo modo que ocurre con la aplicación de registro y consulta de evidencias, se ha utilizado el framework Angular para la implementación de esta aplicación web, junto con la biblioteca Bootstrap 4, que permite un diseño responsive. En este caso además se ha empleado **Angular Material**, que es una librería de componentes visuales (UI components) la cual está optimizada para usarse junto con Angular. Esta librería ha sido desarrollada por el Angular team por lo que se integra perfectamente con este framework

En este apartado de la memoria se exponen los requisitos de la aplicación, los diferentes casos de uso y las comunicaciones de esta aplicación con otros elementos del sistema implementado.

### 6.1. Requisitos de la aplicación

Antes de comenzar con el diseño, se establecieron los requisitos de la aplicación para la administración de solicitudes de nuevos usuarios. Se distinguen dos grupos principales de requisitos: funcionales y no funcionales.

#### 6.1.1. Requisitos funcionales

Los requisitos funcionales son los que definen las funciones de la aplicación de administración y quedan reflejados en los casos de uso. Son los siguientes:

- **RF1. Gestión de solicitudes.** El administrador que inicie sesión en la aplicación visualizará en forma de listado todas las solicitudes pendientes de personas que quieren participar en el sistema de custodia. A juicio de este administrador las peticiones de inclusión serán aceptadas o rechazadas.

- **RF2. Envío de notificaciones.** Sean aceptadas o no las peticiones de inclusión en el sistema, la aplicación deberá notificar por correo electrónico a los solicitantes de la resolución. En el caso de que el administrador apruebe la petición, la notificación deberá incluir la identidad/card del nuevo participante en el sistema, para que el usuario pueda importar dicha identidad en su wallet.

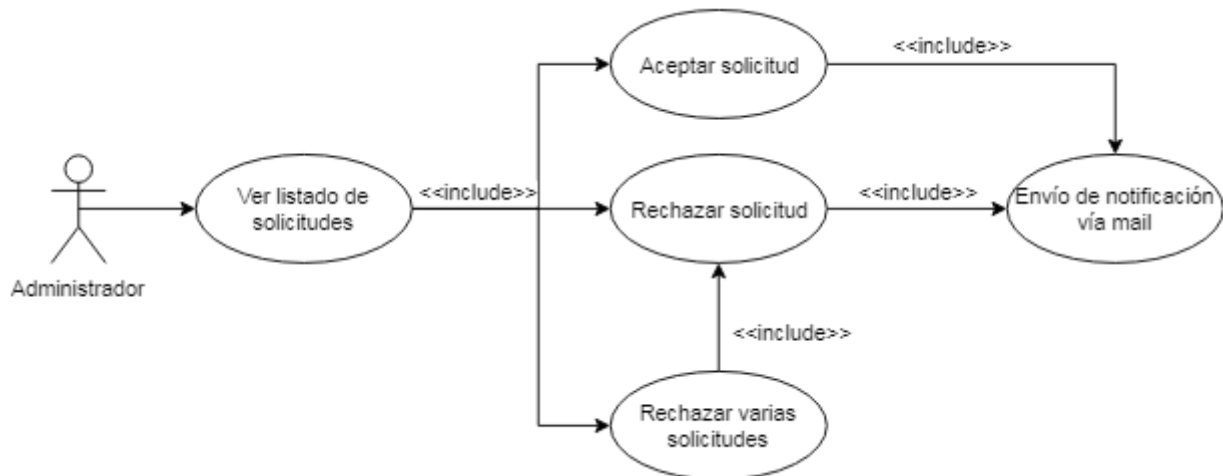
### 6.1.2. Requisitos no funcionales

Los requisitos no funcionales se refieren a todos aquellos que nos describen información a guardar, ni funciones a realizar, sino características de funcionamiento y de diseño. Se han definido los siguientes requisitos no funcionales:

- **RNF1. Apariencia de la aplicación.** La apariencia de la aplicación deberá ser sencilla, clara y con colores simples, de forma que facilite al usuario el manejo de esta.
- **RNF2. Autenticación.** La aplicación debe ofrecer un modo de verificar la identidad de cada uno de los participantes en el sistema. En esta aplicación la autenticación se realiza gracias a Firebase Auth. Este servicio de Firebase permite autenticar a los usuarios gracias a los múltiples proveedores de autenticación con los que cuenta.
- **RNF3. Multidispositivo.** La aplicación debe visualizarse correctamente en cualquier tipo de dispositivo, por lo que el diseño web tendrá que ser *responsive* o adaptable (del inglés, responsive web design). De este modo los contenidos a visualizar se adaptarán al tamaño de la pantalla para maximizar la experiencia de los usuarios

## 6.2. Casos de uso

Dado que esta aplicación tiene menos requisitos funcionales que la otra aplicación, también existen menos casos de uso. En la Figura 25 se muestra el diagrama de casos de uso de la aplicación de administración de solicitudes de nuevos usuarios. A diferencia de lo que ocurría con la aplicación de registro y consulta de evidencias, aquí un usuario no autenticado no puede interactuar con la aplicación, ya que todos los casos de uso están destinados para los administradores (que deben iniciar sesión en esta aplicación) del sistema.



*Figura 25. Diagrama de casos de uso de la aplicación de administración de solicitudes de nuevos usuarios.*

A continuación, se explican brevemente cada uno de los casos de uso que aparecen en el diagrama:

- **Ver listado de solicitudes.** Tras iniciar sesión, el administrador podrá visualizar todas las solicitudes pendientes.
- **Aceptar solicitud.** El administrador podrá aceptar la petición seleccionada.
- **Rechazar solicitud.** Del mismo modo, se podrá rechazar una solicitud.
- **Envío de notificación vía mail.** Se acepte o se rechace la petición, se abrirá el cliente de correo predeterminado en el ordenador del administrador, con un mensaje por defecto. Ya es decisión del administrador el modificar o no dicho correo.
- **Rechazar varias solicitudes.** También existe la opción de eliminar varias solicitudes a la vez.

En el Anexo C de este documento se adjunta capturas de esta aplicación, a modo de breve guía de uso.

### **6.3. Comunicación con el resto del sistema**

Entre los diferentes componentes que forman el sistema de cadena de custodia desarrollado, esta aplicación web se comunica con el servidor REST de Hyperledger Composer secundario, así como con la plataforma de Google Firebase, tal y como se observa en la Figura 26.



*Figura 26. Comunicación de la aplicación de administración con otros componentes del sistema.*

Por un lado, esta aplicación se comunica con el servidor REST secundario, el cual se usa únicamente por los administradores, para la creación de nuevos participantes y para la generación de identidades para estos participantes.

Por otro lado, la aplicación de administración de solicitudes hace uso de dos servicios de la plataforma Firebase:

- **Firebase Cloud Firestore.** Se utiliza la misma colección que se usa en la otra aplicación para el almacenamiento de solicitudes. En este caso, se lee las solicitudes y se eliminan tras su aceptación o denegación. Además, se hace uso de una segunda colección para comprobar que el correo introducido durante el proceso de autenticación pertenece a uno de los administradores del sistema.
- **Firebase Auth.** Este servicio se utiliza para poder autenticar a los administradores del sistema. En el apartado 7 de esta memoria se profundiza en los detalles de este mecanismo de autenticación.

## 7. Servicios de Firebase usados

Tal y como ya se ha mencionado durante los apartados 6 y 7 de esta memoria, se han utilizados varios servicios pertenecientes a la plataforma Firebase. **Se han usado hasta un total de cuatro servicios de la rama de desarrollo: Storage, Cloud Firestore, Auth y Hosting.** En la Tabla 2 se resume la utilidad de cada uno de los servicios de Firebase utilizados en este proyecto.

*Tabla 2. Servicios de Firebase utilizados*

Servicio de Firebase	Utilidad
<b>Storage</b>	Repositorio de ficheros
<b>Cloud Firestore</b>	Almacenamiento de las peticiones de nuevos usuarios. Almacenamiento de los correos de los administradores.
<b>Auth</b>	Autenticación de la aplicación de administración de solicitudes.
<b>Hosting</b>	Despliegue de las dos aplicaciones

En las siguientes secciones de este capítulo se profundiza en cada uno de estos servicios empleados.

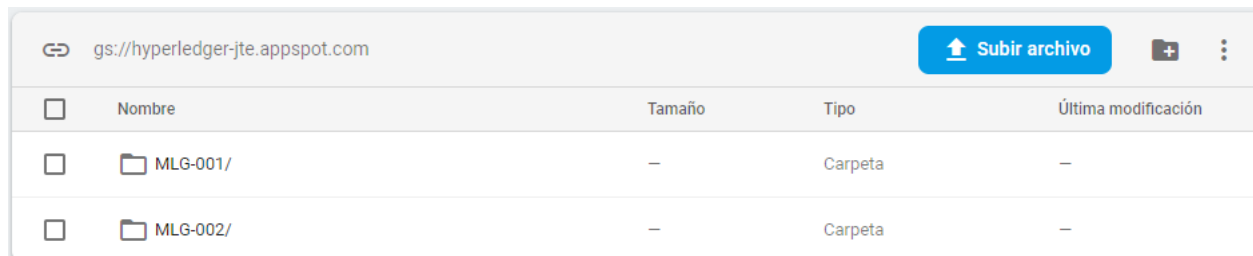
### 7.1. Firebase Storage

Este servicio es el que **permite implementar el repositorio de ficheros, en el que se almacenan las master copies** de las evidencias digitales. Se trata de **un servicio que permite la subida y descarga** de imágenes, audio, videos o cualquier otro contenido generado por el usuario.

Cada vez que un participante del sistema registra una nueva evidencia, además de registrarla en la *Business Network* desplegada en la blockchain de Hyperledger Fabric, **se sube al servicio de Firebase Storage una copia de la prueba**, en donde se guarda de forma indefinida.

Para el almacenamiento de estas copias, se ha implementado la siguiente jerarquía en el repositorio de ficheros:

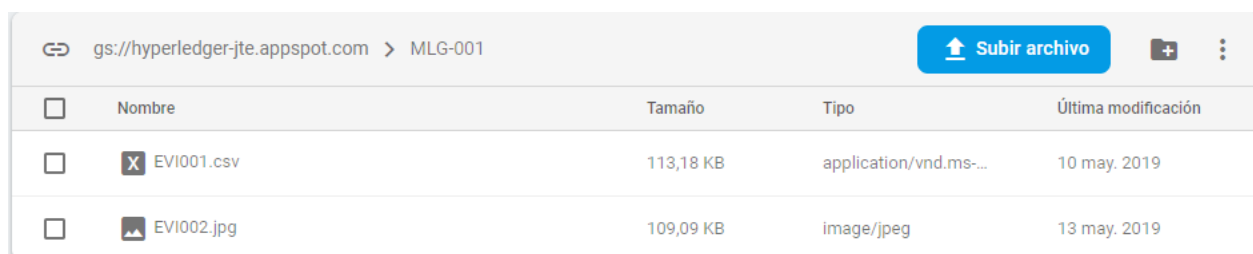
- En la carpeta raíz se encuentran una serie de carpetas. **Cada caso tiene una carpeta asociada**, cuyo nombre es el identificador del caso (Asset Case de la *Business Network*) en cuestión (ver Figura 27).



<input type="checkbox"/>	Nombre	Tamaño	Tipo	Última modificación
<input type="checkbox"/>	MLG-001/	—	Carpeta	—
<input type="checkbox"/>	MLG-002/	—	Carpeta	—

*Figura 27. Una carpeta por caso en Firebase Storage.*

- **Dentro de cada carpeta se encuentran las evidencias registradas para dicho caso.** Las evidencias tienen el mismo identificador que los assets Evidence existentes en la Business Network. En la Figura 28 se observan las evidencias registradas para el caso MLG-001.



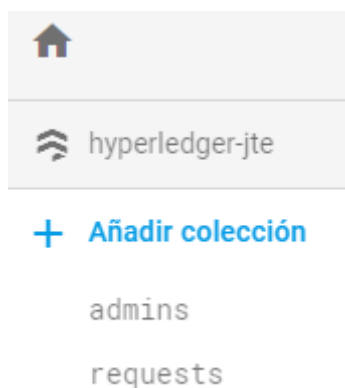
<input type="checkbox"/>	Nombre	Tamaño	Tipo	Última modificación
<input type="checkbox"/>	EVI001.csv	113,18 KB	application/vnd.ms-...	10 may. 2019
<input type="checkbox"/>	EVI002.jpg	109,09 KB	image/jpeg	13 may. 2019

*Figura 28. Evidencias del caso MLG-001 almacenadas en Firebase Storage.*

## 7.2. Firebase Cloud Firestore

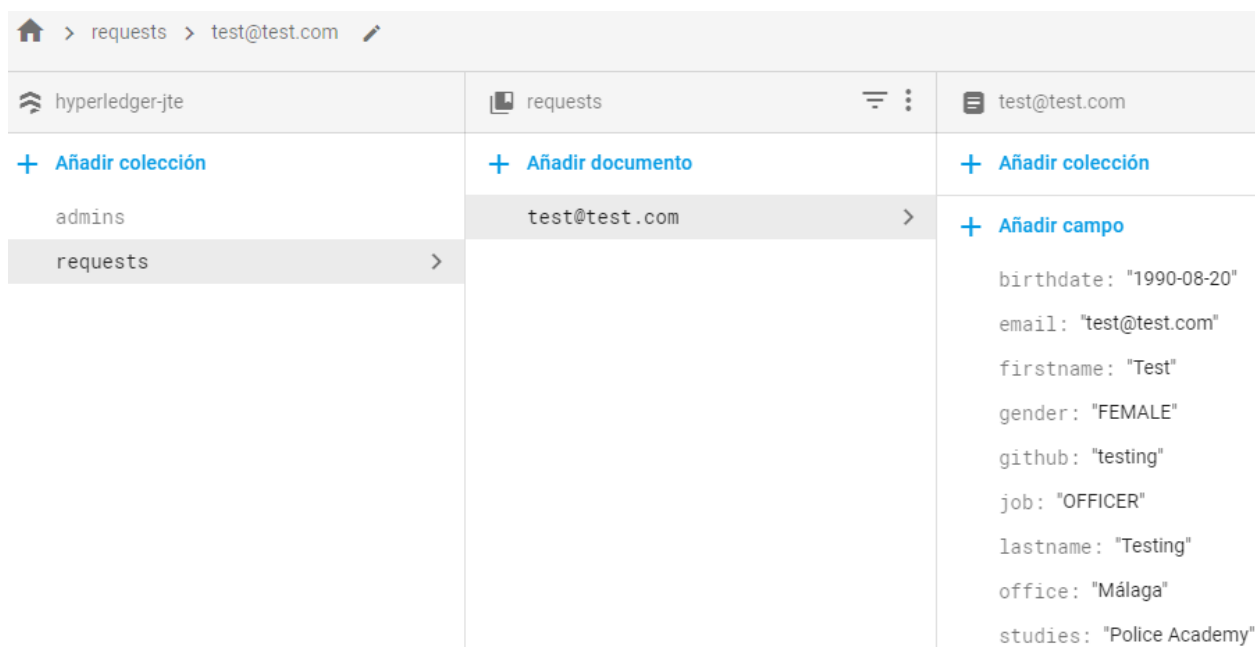
Se trata de una base de datos NoSQL basada en colecciones y documentos, de forma similar a MongoDB o CouchDB. Es por ello por lo que **se han definido dos colecciones**, una para la administración de las solicitudes de nuevos usuarios, y otra para registrar los emails de los administradores del sistema.

Por un lado, la aplicación de registro y consulta de evidencias crea documentos en la **colección requests**, y la aplicación de administración los consulta y elimina. Por otro lado, la aplicación de administración consulta los documentos de la **colección admins**. En la Figura 29 se observan las colecciones en la consola de Firebase Cloud Firestore.

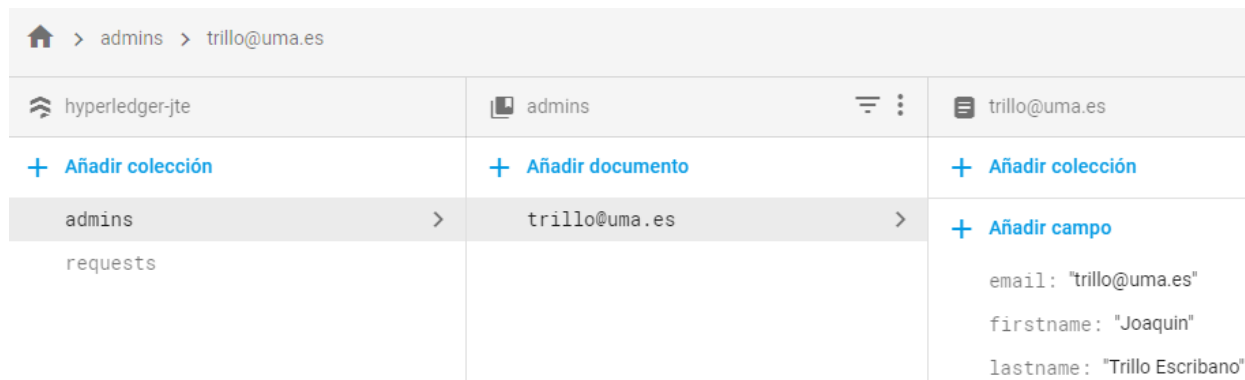


*Figura 29. Colecciones existentes en Firebase Cloud Firestore.*

El identificador de los documentos en ambas colecciones es el correo electrónico del solicitante/administrador (esto facilita las búsquedas, en el caso de que haya muchos administradores o peticiones sin revisar). En las Figuras Figura 30 y Figura 31 se adjuntan ejemplos de documentos pertenecientes a ambas colecciones. Tal y como se observa **los documentos de la colección *admins* cuentan con el nombre completo del administrador**, mientras que **los documentos de la colección *requests* incluyen todos los datos del posible nuevo participante** en el sistema de cadena de custodia.



*Figura 30. Ejemplo de documento de la colección requests.*



The screenshot shows a web application interface with a breadcrumb trail at the top: 'home > admins > trillo@uma.es'. Below this is a table with three columns. The first column is labeled 'hyperledger-jte' and contains a '+ Añadir colección' button and a list item 'admins' with a right arrow. The second column is labeled 'admins' and contains a '+ Añadir documento' button and a list item 'trillo@uma.es' with a right arrow. The third column is labeled 'trillo@uma.es' and contains a '+ Añadir colección' button, a '+ Añadir campo' button, and a list of fields: 'email: "trillo@uma.es"', 'firstname: "Joaquin"', and 'lastname: "Trillo Escribano"'.

hyperledger-jte	admins	trillo@uma.es
+ Añadir colección	+ Añadir documento	+ Añadir colección
admins >	trillo@uma.es >	+ Añadir campo
requests		email: "trillo@uma.es" firstname: "Joaquin" lastname: "Trillo Escribano"

*Figura 31. Ejemplo de documento de la colección admins.*

Además, para garantizar acceso indebidos a esta base datos, se han incluido reglas de acceso a las documentos (ver Figura 32). Por un lado, se garantizan permisos de lectura y escritura sobre los documentos de la colección **requests**. Por otro lado, solo se otorga acceso de lectura a los documentos de la colección **admins**.

```
service cloud.firestore {
  match /databases/{database}/documents {
    //Matches any document in the 'admins' collection.
    match /admins/{admin} {
      allow read;
    }

    //Matches any document in the 'requests' collection.
    match /requests/{request} {
      allow read, write;
    }
  }
}
```

*Figura 32. Reglas que controlan el acceso a los documentos de Firebase Cloud Firestore.*

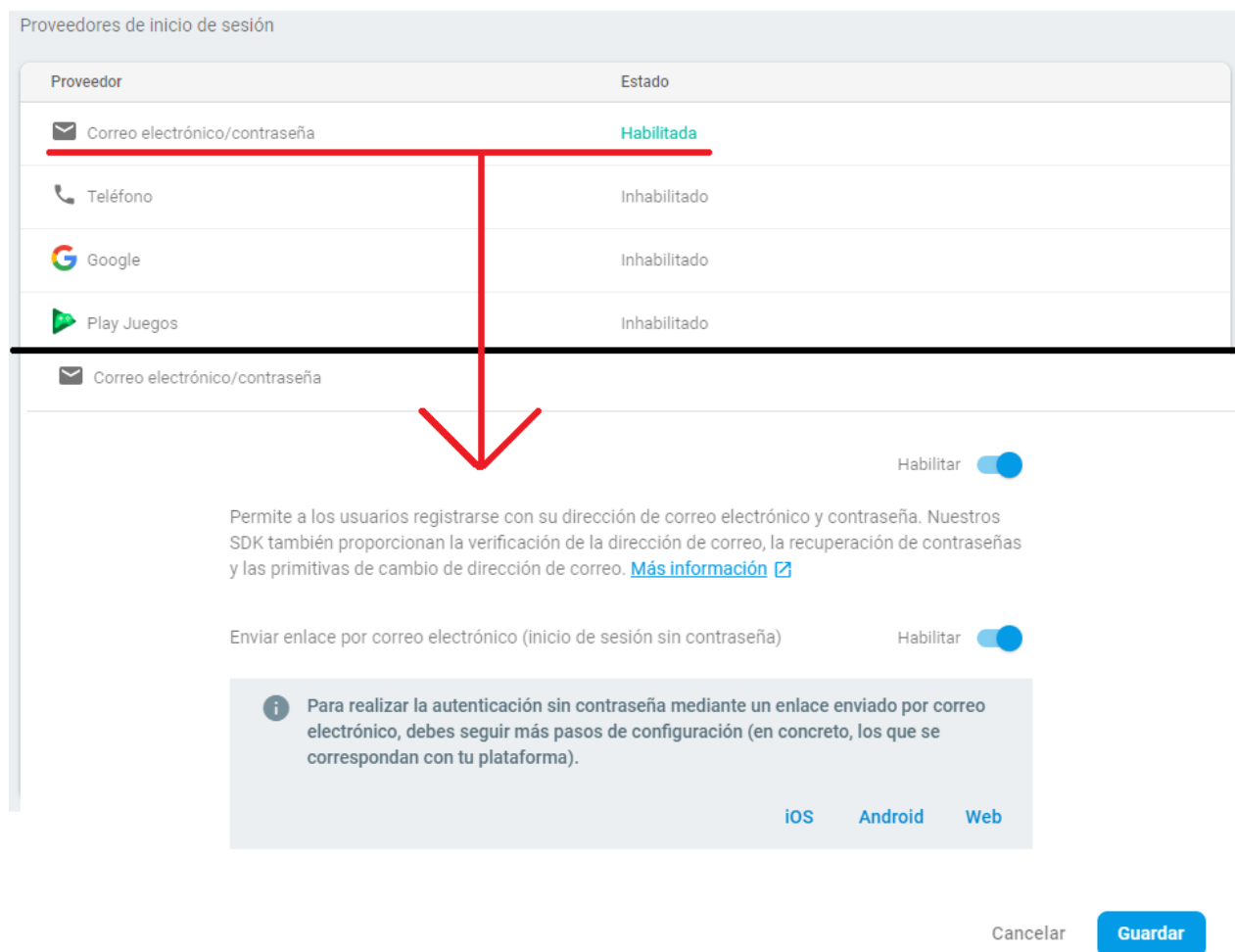
### 7.3. Firebase Auth

Este servicio permite añadir un mecanismo de autenticación a las aplicaciones desarrolladas. Cuenta con múltiples proveedores de inicio de sesión: Google, Facebook, Twitter, GitHub, Yahoo, Microsoft, por correo electrónico, por teléfono... En este caso se ha optado por utilizar como proveedor de inicio de sesión – para la aplicación de administración – **el envío de un enlace de único al correo electrónico**.



Para configurar este tipo de configuración basta con seguir los siguientes pasos:

- **Dirigirse a la consola de administración de Firebase.**
- **Acceder al servicio de autenticación (Authentication).**
- **Seleccionar Método de inicio de sesión.** Aquí aparecerá un listado con todos los proveedores de inicio de sesión que soporta Firebase Auth.
- **Habilitar el proveedor 'Correo electrónico/contraseña'.** En este último paso hay que habilitar tanto el proveedor como el envío de enlace por correo electrónico, marcando los dos botones de palanca (ver Figura 33).



**Figura 33. Habilitar el inicio de sesión usando envío de enlaces por correo electrónico.**

Además, **es necesario añadir el dominio de la aplicación de administración** (la cual hace uso de este servicio) **a la lista de dominios autorizados**, para la redirección que hace OAuth. El dominio por defecto del proyecto y localhost ya vienen incluidos en esta

lista y se pueden añadir todos los que se consideren oportunos. En la Figura 34 de muestran los dominios que se han autorizado en el servicio Auth de Firebase.

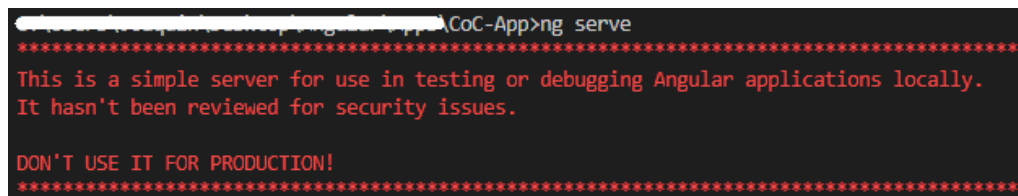


Dominio autorizado	Tipo
localhost	Predeterminado
hyperledger-jte.firebaseio.com	Predeterminado
<b>coc-admin-app.firebaseio.com</b>	Personalizado

*Figura 34. Dominios autorizados para la redirección de OAuth.*

## 7.4. Firebase Hosting

Durante el desarrollo de ambas aplicaciones se utilizó el servidor incluido en la herramienta Angular CLI. Pero este servidor es de uso exclusivo para testeo y depuración de las aplicaciones en un entorno local, tal y como se puede observar en la Figura 35.



```
ng serve
*****
This is a simple server for use in testing or debugging Angular applications locally.
It hasn't been reviewed for security issues.

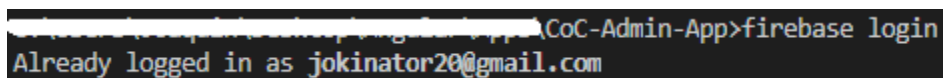
DON'T USE IT FOR PRODUCTION!
*****
```

*Figura 35. Mensaje de Angular CLI cuando se usa el servidor de desarrollo.*

Este es el motivo por el cual se han tenido que desplegar ambas aplicaciones. Y como en este proyecto ya se está usando la plataforma Firebase, se decidió desplegar ambas aplicaciones en esta plataforma, usando el servicio de Hosting. Para ello se deben utilizar las herramientas de línea de comandos de Firebase (**Firebase CLI**), que se instalan a través del sistema de gestión de paquetes de Node.js, npm (**npm install -g firebase-tools**).

Los pasos seguidos para desplegar las dos aplicaciones son los mismos, por lo que se va a detallar sólo uno de estos dos despliegues (concretamente el de la aplicación de administración de solicitudes). Los pasos seguidos son:

- **Generar la versión de producción de la aplicación.** Con la herramienta Angular CLI es posible generar los ficheros necesarios para desplegar la aplicación en un entorno de producción. A diferencia de los ficheros que se usan durante el desarrollo de la aplicación, estos están optimizados para ser usados en un entorno de producción. El comando **ng build -prod** genera la carpeta **dist** con los archivos mencionados.
- **Iniciar sesión en Firebase.** Mediante la herramienta de línea de comandos de Firebase, iniciar sesión en esta plataforma con el comando **firebase login**. Si ya está iniciada la sesión aparecerá el mensaje de la Figura 36.



```
CoC-Admin-App>firebase login
Already logged in as jokinator20@gmail.com
```

*Figura 36. Firebase CLI - Sesión iniciada en Firebase.*

- **Iniciar el proyecto de Firebase.** Usando el comando **firebase init** comenzará este proceso. En primer lugar, la herramienta pedirá que servicios de la plataforma se quieren usar (en este caso **solo el servicio de Hosting**, Figura 37). En segundo lugar, se demanda el proyecto a utilizar (Figura 38). Por último, se pide la carpeta en la que se encuentran los ficheros de la aplicación, si esta aplicación es de una sola página y si desea sobrescribir el fichero index.html (en la Figura 39 se puede ver la configuración escogida).

```

C:\Users\user> cd C:\Users\user\Documents\CoC-Admin-App> firebase init

#####  ###  #####  #####  ###  #####  #####
##      ##  ##  ##  ##  ##  ##  ##  ##  ##
#####  ##  #####  #####  #####  #####  #####
##      ##  ##  ##  ##  ##  ##  ##  ##  ##
##      #####  ##  #####  #####  ##  ##  #####  #####

You're about to initialize a Firebase project in this directory:

C:\Users\user\Documents\CoC-Admin-App

? Are you ready to proceed? Yes
? Which Firebase CLI features do you want to set up for this folder? Press Space to select features, then Enter to confirm your choices.
  ( ) Database: Deploy Firebase Realtime Database Rules
  ( ) Firestore: Deploy rules and create indexes for Firestore
  ( ) Functions: Configure and deploy Cloud Functions
> (*) Hosting: Configure and deploy Firebase Hosting sites
  ( ) Storage: Deploy Cloud Storage security rules

```

**Figura 37. Firebase CLI - Servicios de Firebase que se quieren usar.**

```
? Select a default Firebase project for this directory:
[don't setup a default project]
> hyperledger-jte (Hyperledger-JTE)
[create a new project]
```

**Figura 38. Firebase CLI - Proyecto Firebase a utilizar.**

```
? What do you want to use as your public directory? dist/CoC-Admin-App
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? File dist/CoC-Admin-App/index.html already exists. Overwrite? No
i Skipping write of dist/CoC-Admin-App/index.html

i Writing configuration info to firebase.json...
i Writing project information to .firebaserc...

+ Firebase initialization complete!
```

**Figura 39. Firebase CLI - Configuración necesaria para desplegar la aplicación en Firebase Hosting.**

- **Desplegar la aplicación.** En el paso anterior se habrá generado el archivo **firebase.json**. Es necesario modificar este fichero y añadir dentro de la clave **hosting** el par clave valor **target-admin** antes de desplegar la aplicación, tal y como se observa en la Figura 40. Esta modificación es necesaria para que se desplieguen los archivos generados en la aplicación de Firebase que deseamos.

```
{
  "hosting": {
    "target": "admin",
    "public": "dist/CoC-Admin-App",
    "ignore": [
      "firebase.json",
      "**/.*",
      "**/node_modules/**"
    ],
    "rewrites": [{
      "source": "**",
      "destination": "/index.html"
    }]
  }
}
```

*Figura 40. Incluir el par clave-valor señalado en rojo en el archivo firebase.json.*

Una vez se modifica el archivo `firebase.json` hay que ejecutar el comando **firebase target:apply hosting admin coc-admin-app**. Este comando informa a Firebase CLI de que en el próximo despliegue deberá desplegar el contenido del target **admin** (en este caso la carpeta **dist/CoC-Admin-App**) en la aplicación **coc-admin-app**. Tras aplicar el target, solo queda ejecutar el comando para desplegar la aplicación en Firebase: **firebase deploy --only hosting**. En la Figura 41 se muestra la respuesta de Firebase CLI tras la ejecución de los dos últimos comandos mencionados.

```
C:\coc-admin-app>firebase target:apply hosting admin coc-admin-app
+ Applied hosting target admin to coc-admin-app

Updated: admin (coc-admin-app)

C:\coc-admin-app>firebase deploy --only hosting
=== Deploying to 'hyperledger-jte'...

i  deploying hosting
i  hosting[coc-admin-app]: beginning deploy...
i  hosting[coc-admin-app]: found 10 files in dist/CoC-Admin-App
+  hosting[coc-admin-app]: file upload complete
i  hosting[coc-admin-app]: finalizing version...
+  hosting[coc-admin-app]: version finalized
i  hosting[coc-admin-app]: releasing new version...
+  hosting[coc-admin-app]: release complete

+ Deploy complete!

Project Console: https://console.firebase.google.com/project/coc-admin-app/overview
Hosting URL: https://coc-admin-app.firebaseio.com
```

*Figura 41. Firebase CLI - Despliegue de la aplicación.*

Las URL de las aplicaciones desplegadas en Firebase Hosting son las que aparecen en la Tabla 3.

*Tabla 3. URLs de las dos aplicaciones.*

Aplicación	URL
Registro y consulta evidencias	<a href="https://coc-app.firebaseio.com">https://coc-app.firebaseio.com</a>
Administración solicitudes	<a href="https://coc-admin-app.firebaseio.com">https://coc-admin-app.firebaseio.com</a>

## 8. Conclusiones y líneas futuras

---

En este último capítulo se explica cómo surge la idea para este proyecto, lo que ha supuesto su realización y su posible despliegue para un caso de uso real. También se comentan posibles mejoras del sistema, principalmente en la definición de la Business Network.

### 8.1. Conclusiones

La idea de crear este sistema de cadena de custodia para evidencias digitales surge a comienzos del tercer cuatrimestre de este máster (septiembre/octubre de 2018), cuando empezamos a ver el temario de la asignatura **Informática Forense**. Concretamente en el primer tema **aparece el concepto de cadena de custodia** (término que personalmente desconocía hasta esa fecha) como mecanismo para garantizar la trazabilidad y la integridad de las evidencias digitales.

Por otra parte, **la tecnología blockchain**, con la que llevo trabajando aproximadamente un año, permite crear sistemas que aseguran la integridad de la información (al existir múltiples equipos, no es posible modificar los datos existentes) y que posibilitan identificar el origen y diferentes etapas de dicha información (es decir, se obtienen sistemas con trazabilidad de los datos).

Como acabamos de ver, **blockchain es una tecnología idónea para implementar con ella un sistema de cadena de custodia**, y es por esto por lo que decidí realizar este Trabajo de Fin de Máster.

Respecto a lo que ha supuesto la realización de este proyecto destacaría tres puntos. En primer lugar, **he podido aplicar conocimientos aprendidos en las asignaturas del Máster**, no solo de *Informática Forense*, también de otras asignaturas como *Desarrollo de Aplicaciones en la Nube*, *Infraestructura para Computación en la Nube* o *Seguridad y Privacidad en Entornos de Aplicaciones*, entre otras.

En segundo lugar, **he conseguido aprender a trabajar con el framework Hyperledger Fabric y su herramienta asociada, Hyperledger Composer**. Esta ha sido tal vez la parte más difícil de este trabajo, ya que, aunque existe bastante documentación de Hyperledger, para casos de uso algo más complejos como es el caso, la documentación en mi opinión se queda corta en ciertos aspectos. Además, la

comunidad existente no es demasiado grande hoy en día, por lo que encontrar respuestas en los foros de preguntas se hace complicado. La parte positiva de ese esfuerzo es que **estos conocimientos adquiridos me están sirviendo en la actualidad** para diseñar un nuevo sistema a partir del cual (junto con mis compañeros de mi grupo de investigación) vamos a tratar de realizar y publicar un artículo científico.

En tercer y último lugar, **pude aprender a manejar el framework de diseño de aplicaciones web Angular**. Gracias a las dos aplicaciones que se han diseñado e implementado para este proyecto, he logrado cumplir el interés personal que tenía en conocer este framework. Además, me he forzado a ser constante en el **uso de un sistema de control de versiones (git) durante la implementación de ambas aplicaciones**. Así que aparte de realizar mi TFM, cuento en mi repositorio con dos aplicaciones Angular funcionales y desarrolladas íntegramente por mí.

Para finalizar, es importante remarcar que **la cadena de custodia es un proceso fundamental para la preservación y el procesamiento de las evidencias** relacionadas con un presunto hecho delictivo. Tal es su importancia que ni el órgano jurisdiccional ni la defensa podrán poner en duda la integridad y la admisibilidad de la evidencia. Desde el momento en que la evidencia se extrae hasta que se presenta al Tribunal, la cadena de custodia **establece la posesión de dicha prueba**.

## **8.2. Líneas futuras**

Siempre es posible mejorar, y este sistema no va a ser una excepción. A pesar de que el trabajo realizado es un proyecto innovador, considero que hay ciertos puntos que son claramente mejorables de cara a un caso de uso real. Por ejemplo, si la Policía Nacional decidiese usar este sistema, se deberían llevar a cabo las siguientes modificaciones:

- **Mejorar la definición de la *Business Network***. Este es sin duda la parte del sistema en el que se pueden dar más modificaciones. La *Business Network* ha sido definida sin consultar con ningún implicado en un sistema real de cadena de custodia (agentes, jueces, fiscales, etc.) y es altamente probable que dicha definición no sea operativa. Por lo tanto, **antes de desplegar el sistema para la Policía Nacional, es prácticamente obligatorio delimitar junto a ellos la definición del sistema**: que participantes existen y que permisos tienen, que



activos hay, cuantas transacciones son necesarias y que comportamiento tienen, etc.

- **Desplegar todos los componentes en una red interna.** Todos los componentes del sistema se han desplegado en la nube de Google y se han usado servicios cloud como Firebase y MongoDB Atlas para el almacenamiento. Pero de cara a un sistema real, los componentes, los datos y las copias de las pruebas deben alojarse en la red interna de la Policía Nacional. Este sistema trata información muy sensible, que no debería almacenarse en una plataforma cloud, si no en unos servidores propios de la Policía. Y a estos servidores solo debería tener acceso las personas que participen en el sistema de cadena de custodia.
- **Cambiar los mecanismos de autenticación.** Por un lado, para acceder a la aplicación de registro y consulta de evidencias digitales se usa **el middleware Passport.js con Github como proveedor de autenticación**. Por otro lado, en la autenticación a la aplicación de administración de solicitudes utiliza **el servicio Firebase Auth, enviando al correo del administrador un enlace de un único uso para el inicio de sesión**. En el caso de uso real, utilizar una red social como proveedor de autenticación no es coherente. A priori, lo ideal sería usar el correo interno de la Policía para iniciar sesión en la aplicación de registro y consulta de evidencias digitales. Por otra parte, el mecanismo de autenticación de la aplicación de administración si pudiera mantenerse.



## Anexo A – Modelo del sistema

---

Este anexo complementa el apartado 3.1 de la memoria, en el que se explica el modelo del sistema definido para implementar el proceso de la cadena de custodia. En dicho modelo, codificado con el lenguaje **Hyperledger Composer Modeling Language**, se han especificado participantes, bienes, transacciones, eventos, conceptos y tipos enumerados.

Los participantes, bienes, transacciones, eventos y conceptos **tienen una serie de atributos**, como cualquier clase de los lenguajes de programación orientados a objetos. Se anota con la letra **o** antes del tipo que sean (por ejemplo, **o String office**).

Los participantes, bienes y conceptos además **pueden tener relaciones hacia otros recursos**. Estas relaciones son básicamente punteros a instancias de otros recursos y se anotan con **-->** antes del tipo de recurso hacia el que apuntan (por ejemplo, la relación existente de una prueba hacia su propietario, **--> CoCParticipant owner**).

Además, los participantes y bienes **deben identificarse mediante uno de los atributos que lo componen**. Este identificador debe ser único entre objetos/instancias del mismo tipo (Por ejemplo, no puede haber dos objetos de tipo **Evidence** con identificador EVD001).

### Participantes

Para definir un tipo de participante nuevo se usa la palabra reservada **participant**. El modelo dispone del tipo de participante **CoCParticipant**, identificado por el atributo **participantId**. De este tipo abstracto heredan otros dos **Agent** y **Deposit**.

```
abstract participant CoCParticipant identified by participantId {
  o String participantId
}
participant Agent extends CoCParticipant {
  o String firstName
  o String lastName
  o DateTime birthdate
  o Gender
  o Job job
  o String studies
  o String office
}
```

```
participant Deposit extends CoCParticipant {  
  o String office  
}
```

## Activos

Para definir un tipo de activo nuevo se utiliza la palabra reservada **asset**. Se han especificado dos tipos de activos: caso y prueba. El activo caso (**Case**) tiene como identificador el atributo **caseId**, y tiene dos relaciones:

- **openedBy**. Participante de tipo **CoCParticipant** que abrió el caso.
- **participants**. Conjunto de participantes que están involucrados en el caso.

Por su parte, los activos de tipo prueba (**Evidence**) son identificados por el atributo **evidenceId** y también tienen dos relaciones:

- **owner**. Participante de tipo **CoCParticipant** el cual es el propietario temporal de la prueba en cuestión.
- **caso**. Caso al que la evidencia pertenece.

```
asset Case identified by caseId {  
  o String caseId  
  o String description  
  o DateTime openingDate  
  o String resolution optional  
  o DateTime closureDate optional  
  o CaseStatus status  
  --> CoCParticipant openedBy  
  --> CoCParticipant[] participants  
}  
  
asset Evidence identified by evidenceId {  
  o String evidenceId  
  o String hash  
  o HashType hash_type  
  o String description  
  o String extension  
  o DateTime additionDate  
  --> CoCParticipant owner  
  o Owner[] olderOwners  
  --> Case caso //case is a reserved word, so I must use another word to name the  
attribute  
}
```

## Concepto

Para definir un nuevo concepto se usa la palabra reservada **concept**. Los conceptos son clases abstractas que suelen ir contenidas en la definición de un participante, un activo, una transacción o un evento. En este modelo sólo se ha definido un concepto, **Owner**.

```
concept Owner {  
  --> CoCParticipant owner  
  o DateTime till  
}
```

## Tipos enumerados

Idénticos a los existentes en otros lenguajes de programación. Para definir un tipo enumerado se usa la palabra reservada **enum**. En este modelo se han definido 5 enumerados: género (**Gender**), puesto de trabajo de los agentes (**Job**), estado actual del caso (**CaseStatus**), tipo de participante (**ParticipantType**) y tipo de función hash usada para garantizar la integridad de las pruebas (**HashType**).

```
enum Gender {  
  o MALE  
  o FEMALE  
}  
enum Job {  
  o DETECTIVE  
  o OFFICER  
  o FORENSICS_TECHNICIAN  
}  
enum CaseStatus {  
  o OPENED  
  o CLOSED  
}  
enum ParticipantType {  
  o AGENT  
  o DEPOSIT  
}  
enum HashType {  
  o MD5  
  o SHA1  
  o SHA256  
  o SHA512  
}
```

## Transacciones y eventos

Habitualmente las transacciones y eventos de Hyperledger Composer se definen en tuplas, ya que cada vez que se ejecuta una transacción se suele emitir un evento para informar a los usuarios de la compleción de la mencionada transacción. Para definir una transacción se utiliza la palabra reservada **transaction** y para los eventos se usa la palabra reservada **event**. Como ya se ha explicado en el tercer apartado de la memoria de este proyecto, se han especificado un total de cinco parejas de transacciones y eventos.

La implementación del comportamiento de cada una de estas cinco transacciones tiene lugar en el script que forma parte de la *Business Network Definition*. Dado que se trata de un archivo de más de 500 líneas de código no se ha copiado en este anexo, pero se ha adjuntado en la entrega de este proyecto (archivo **script.js**). A continuación, se adjunta el código de las parejas de transacción-evento existentes en el modelo.

```
//1. Open a case by an agent
transaction OpenCase {
  o String id
  o String description
}
event CaseOpened {
  o String case_id
  o String openedBy_participant_id
}
//2. Close a case (only the agent who opened it can close the case)
transaction CloseCase {
  o String id
  o String resolution
}
event CaseClosed {
  o String case_id
}
//3. The agent who opened the case can add other participants to that case. Only if
the case is still opened
transaction AddParticipant {
  o String case_id
  o ParticipantType participant_type
  o String participant_id
}
event ParticipantAdded {
  o String case_id
  o ParticipantType participant_type
  o String participant_id
}
```

```
//4. Add a new evidence and link it with a case opened
transaction AddEvidence {
  o String evidence_id
  o String hash
  o HashType hash_type
  o String description
  o String extension
  o String case_id
}

event EvidenceAdded {
  o String evidence_id
  o String case_id
  o String participant_id
}

//5. The temporal owner of the evidence transfers it to another agent
transaction TransferEvidence {
  o String evidence_id
  o ParticipantType participant_type
  o String participant_id //new owner id
}

event EvidenceTransferred {
  o String evidence_id
  o String old_owner_id
  o String new_owner_id
}
```





## Anexo B – Manual para la aplicación web de registro y consulta de evidencias

---

En este anexo se va a describir en detalle la funcionalidad de la aplicación web diseñada e implementada en este proyecto para el registro y la consulta de evidencias digitales. Esta aplicación distingue entre usuarios no registrados y usuarios registrados (es decir, los participantes en el sistema de cadena de custodia).

Por un lado, **los usuarios no registrados solo pueden enviar una solicitud de registro**. La vista que permite llevar a cabo esta acción (ver Figura 42) es además la vista que aparece cuando se accede a la aplicación. Tras rellenar todos los campos, el botón '**Send**' se habilitará. Pulsar dicho botón enviará la solicitud de registro, **a menos que exista ya una solicitud con ese mismo correo electrónico**.

### Chain of Custody for Digital Evidences

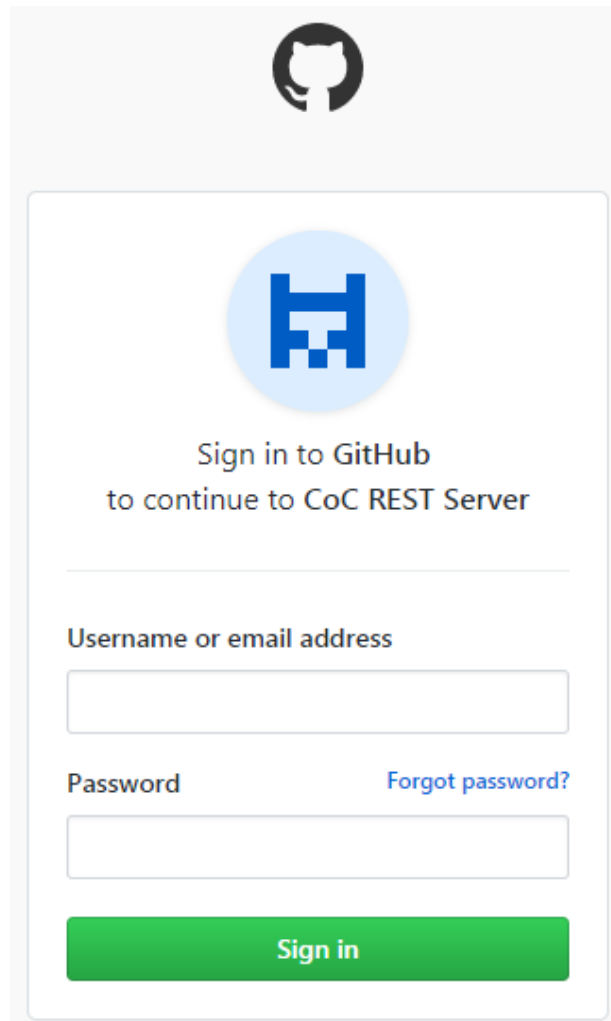
---

Welcome to the web application for the management of digital evidences. If you want to participate in the Chain of Custody system, please fill this form. As soon as possible, it will be checked by an administrator and you will receive an e-mail with your credentials. If you are already inscribed in the system please [click here](#).

Email	<input type="text" value="peter.dun@test.com"/>		
Github ID	<input type="text" value="Your Github ID"/>	First name	<input type="text" value="Peter"/>
Last name	<input type="text" value="Dun"/>	Birthdate	<input type="text" value="dd/mm/aaaa"/>
Gender	<input type="text" value="-- Select one --"/>	Job Position	<input type="text" value="-- Select one --"/>
Studies	<input type="text" value="Software Engineer"/>	Office	<input type="text" value="-- Select one --"/>
<input type="button" value="Send"/>			

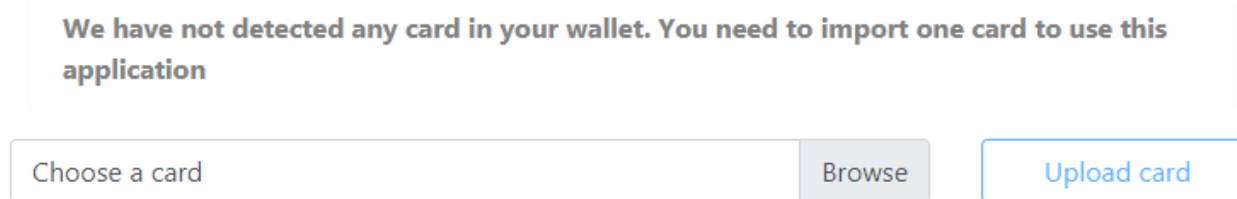
*Figura 42. Envío de solicitud.*

Por otro lado, si el usuario ya está registrado y es un participante en el sistema de cadena de custodia, deberá hacer click en el enlace '**click here**'. De esta forma, se redirigirá al inicio de sesión de la aplicación OAuth de Github (**CoC REST Server**, Figura 43), donde el usuario deberá autenticarse con sus credenciales de esta plataforma.



*Figura 43. Inicio de sesión en la aplicación OAuth de Github.*

Si es la primera vez que inicia sesión, la aplicación detecta que **el wallet del usuario** (conjunto de identidades/cards) **está vacío**. De este modo, se le presenta al usuario un pequeño formulario similar al de la Figura 44 que le permite importar en su wallet la card que recibió en su dirección de correo electrónico. El botón '**Upload card**' se habilitará cuando se seleccione un archivo. Solo si este archivo termina en **@cocv2.card** se importará en el wallet del usuario.



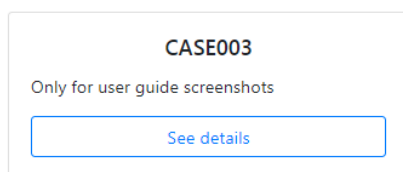
We have not detected any card in your wallet. You need to import one card to use this application

Choose a card Browse Upload card

Figura 44. Pequeño formulario para importar una card.

Si por el contrario el usuario ya tiene importada la card en su wallet, **se cargan los casos en los que este participa, las evidencias que posee y su perfil personal**. Tras obtener toda esta información, el flujo de la aplicación conduce al usuario hasta el menú principal. En esta vista se muestran hasta tres casos abiertos y hasta 3 evidencias. En la Figura 45 se observa un ejemplo de lo que se podría encontrar el usuario: participa en 1 caso y en ese momento no es el propietario temporal de ninguna prueba.

## My opened cases



CASE003

Only for user guide screenshots

See details

## My evidences

You are not the temporary owner of any evidence.

Figura 45. Menú principal.

Además, **la barra de navegación también cambia si el usuario está identificado**. Tal y como se aprecia en la Figura 46, desde esta barra de navegación **el usuario puede moverse por las vistas del menú principal** (Figura 45), **de apertura de nuevos casos** (Figura 47) y **del perfil personal del participante** (Figura 48). También existe en esta barra de navegación un pequeño campo de texto junto a un botón que permite realizar búsquedas de casos y evidencias. **En la vista de resultados de búsqueda** (Figura 49) **aparecerán casos en los que participa el usuario y evidencias que este posee**, cuyos identificadores contienen el término introducido en el campo de texto.

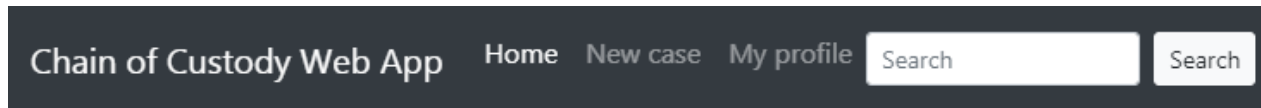


Figura 46. Barra de navegación para usuarios autenticados.

En el caso de la apertura de un caso nuevo, el usuario deberá introducir un identificador y una breve descripción para dicho caso. Solo si estos dos campos están rellenos se habilitará el botón '**Add new case**' y desaparecerá la alerta que indican que todos los campos se deben cumplimentar.

Al pulsar este botón **se ejecutará la transacción OpenCase** de la *Business Network*. Si esta transacción tiene éxito, el caso se abrirá y se le notificará al usuario mediante un mensaje. En este mensaje además aparecerá el id de la transacción con la que se ha abierto el caso.

### New case

---

Case identifier	<input type="text" value="Enter identifier"/>
Case description	<div><input type="text" value="Enter description"/></div>

Add new case

All fields must be filled

Figura 47. Abrir nuevo caso.

### My Profile (identifier in the system: JTrillo)

Full Name:	Joaquin Trillo	Birthdate:	23/11/1994
Gender:	MALE	Job Position:	DETECTIVE
Studies:	Software Engineer	Office:	Málaga

Figura 48. Perfil personal.

## Search results for MLG

Case ID	Status	Opened/Close since
<a href="#">MLG-001</a>	CLOSED	13/05/2019 15:45:53 GMT+2

Any of your evidences match the term **MLG**

Figura 49. Búsqueda de casos y evidencias.

### Vista Ver caso

La vista que recoge toda la información de un caso (ver Figura 50) es la más compleja de la aplicación. En primer lugar, **se muestran todos los datos del caso**: el identificador del propio caso, el identificador del participante que lo abrió, el estatus del caso, la fecha de apertura y la descripción. **Si el caso está cerrado también se muestran fecha de cierre y la resolución dada.**

### Case CASE003

Opened by	JTrillo	Status	OPENED
Opening date	03/06/2019 15:56:42 GMT+2	Closure date	04/06/2019
Description	Only for user guide screenshots		
Resolution	Enter a resolution		

#### Evidences

Evidence id	Info	Current owner id
<a href="#">EVI003</a>	Ertis logo	JTrillo

Upload evidence

#### Participants

Participant identifier
<a href="#">JTrillo</a>
0001 (Deposit)

Add participant

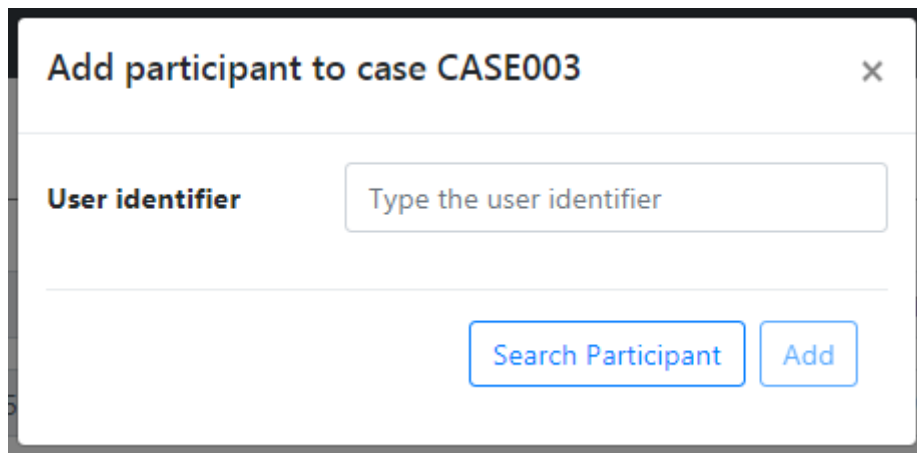
Close case

Go back

Figura 50. Vista Ver caso.

Si el caso está abierto y se quiere cerrar, estos dos últimos campos se deben cumplimentar y pulsar el botón '**Close case**'. Al pulsar este botón **se ejecutará la transacción CloseCase** de la *Business Network*. Si esta transacción tiene éxito, el caso se cerrará, **asignando la propiedad de todas las pruebas relacionadas al participante de tipo Deposit** y notificando al usuario de la compleción de la operación mediante un mensaje.

En segundo lugar, se muestran dos tablas: **una con las evidencias ligadas al caso y otra con los identificadores de los participantes involucrados en el caso**. La tabla de participantes permite clicar en los identificadores para acceder a los perfiles de estos usuarios. También es posible añadir nuevos participantes al caso, pulsando el botón '**Add participant**'. Esta acción abrirá un diálogo como el de la Figura 51, gracias al cual es posible buscar usuarios por identificador. Si el usuario existe, se habilitará el botón '**Add**', el cual **permite la ejecución de la transacción AddParticipant** de la *Business Network*. Si esta operación tiene éxito, se habrá añadido el usuario al caso.



*Figura 51. Diálogo para añadir un nuevo participante.*

Por su parte, la tabla de evidencias muestra el identificador de la prueba, información de esta y su propietario actual. **Si el usuario de la aplicación es además el propietario de alguna de las evidencias, podrá hacer click en el identificador de estas para ver todos los datos de dichas pruebas**. También existe la posibilidad de añadir nuevas evidencias gracias al botón '**Upload evidence**', que redirige al usuario a la vista 'Nueva evidencia'.

## Vista Nueva evidencia

Esta vista es un formulario en el que **se registra toda la información de una nueva prueba**: identificador de la evidencia, valor hash, tipo de hash utilizado, breve descripción de la evidencia y la prueba en sí. En la Figura 52 se muestra la vista para añadir nuevas evidencias.

### New evidence for case CASE003

Evidence ID:  Hash Type:

Hash Value \*:

Description:

Evidence:

\* If you have not any tool to calculate the hash value, we recommend this one --> [MD5 & SHA Checksum Utility](#)

All fields must be filled

*Figura 52. Vista Nueva evidencia.*

De cara a facilitar al usuario el cálculo del hash de la prueba, **se adjunta un enlace para descargar la herramienta MD5 & SHA Checksum Utility**. En la Figura 53 se muestra como generar los diversos hashes con la mencionada herramienta.



*Figura 53. Uso de la herramienta MD5 & SHA Checksum Utility.*

De forma similar a como funcionan otros formularios de esta aplicación, hasta que todos los campos de este formulario no estén rellenos, el botón '**Add evidence**' no se habilitará. Pulsar este botón **ejecutará la transacción AddEvidence** de la *Business Network*. Tras recibir una respuesta positiva de la blockchain, automáticamente **comienza la subida de la copia de la evidencia al repositorio de ficheros**, alojado en la nube de Google.

### Vista Ver evidencia

Esta última vista de la aplicación (ver Figura 54) presenta la información registrada de una evidencia al propietario temporal de esta. Esta vista también muestra los diferentes participantes que han tenido bajo su poder la prueba, así como el periodo de tiempo que la poseyeron.

### Evidence EVI003

Case ID	CASE003	Addition date	04/06/2019 13:47:23 GMT+2
Hash	4AE64D3B7045FAE94D7C67B0EC8C5E4078CC5E5CCCF A23FD30E668349A5024F5 (SHA256)		<a href="#">Copy</a>
Description	Ertis logo		

#### Owners (evidence current owner in bold)

Owner id	Since	Until
JTrillo	04/06/2019 13:47:23 GMT+2	...

[Transfer evidence](#)[Download a copy](#)[Go back](#)

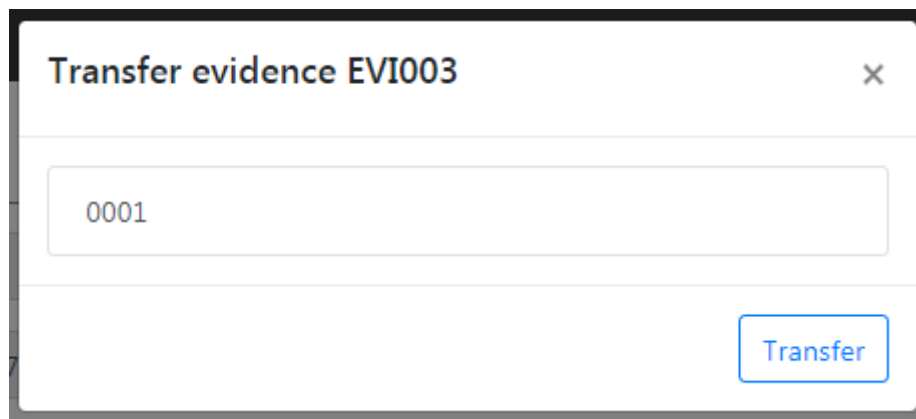
*Figura 54. Vista Ver evidencia.*

Tal y como se puede observar en la imagen adjunta, la vista cuenta con varios botones, cuyas funciones se describen a continuación:

- **Botón 'Copy'**. Copia en el portapapeles del usuario el valor hash de la evidencia. Esto es útil de cara a verificar que la copia descargada sea la misma que la que se subió al añadir la prueba al sistema.
- **Botón 'Download a copy'**. Descarga en el navegador del usuario una copia de la evidencia almacenada en el repositorio de ficheros.



- **Botón ‘Transfer evidence’**. Se abre una ventana emergente, mostrando una lista con los identificadores de los otros usuarios que participan en el caso al que la prueba pertenece (ver Figura 55). Seleccionando uno de estos participantes y pulsando el botón ‘Transfer’, **se ejecutará la transacción *TransferEvidence* de la Business Network**. Tras recibir un mensaje de compleción de la operación, se redirige al usuario al menú principal, **en donde ya no verá la prueba que acaba de transferir**.



*Figura 55. Ventana emergente para la transferencia de evidencias.*



## Anexo C – Manual para la aplicación web de gestión de solicitudes de nuevos usuarios

A diferencia de la aplicación para el registro y consulta de evidencias digitales, la funcionalidad de esta aplicación es muy simple. De hecho, solo cuenta con dos vistas: una para iniciar sesión y otra para que los administradores del sistema puedan aceptar o rechazar las solicitudes de nuevos usuarios.

Para iniciar sesión (ver Figura 56), el administrador únicamente deberá escribir su correo electrónico y pulsar el botón '**Send**'. Si ese correo electrónico no pertenece a ningún administrador del sistema, aparecerá un mensaje de error. Si por el contrario el correo electrónico pertenece a un administrador, aparecerá un mensaje de éxito y se le enviará a dicho email un correo para completar el proceso de inicio de sesión.





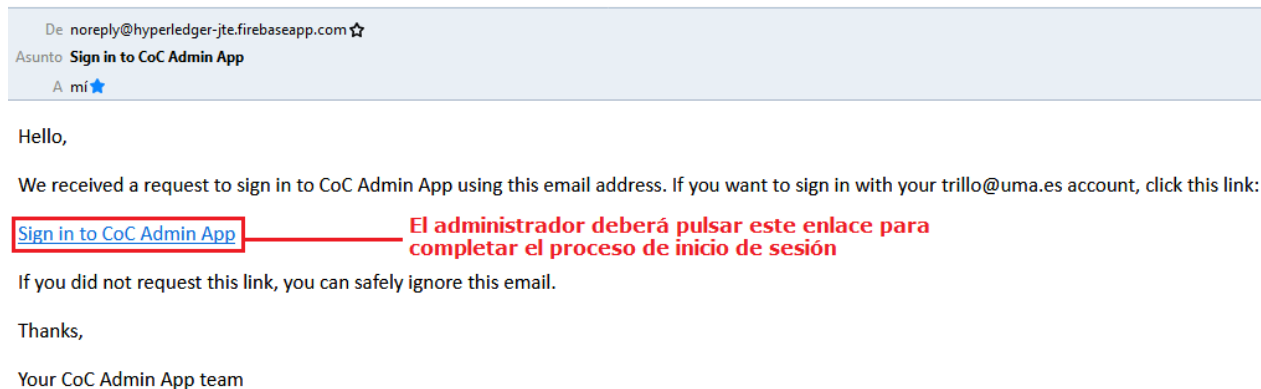
Chain of Custody Administration App	Chain of Custody Administration App
Email test@test.com You must enter an email	Email trillo@uma.es You must enter an email
<b>Send</b>	<b>Send</b>
Email <b>test@test.com</b> does not belong to an admin	Email sent to <b>trillo@uma.es</b>
© 2019 Joaquín Trillo Escribano. Grupo de investigación ERTIS. Universidad de Málaga	© 2019 Joaquín Trillo Escribano. Grupo de investigación ERTIS. Universidad de Málaga
  UNIVERSIDAD DE MÁLAGA	  UNIVERSIDAD DE MÁLAGA

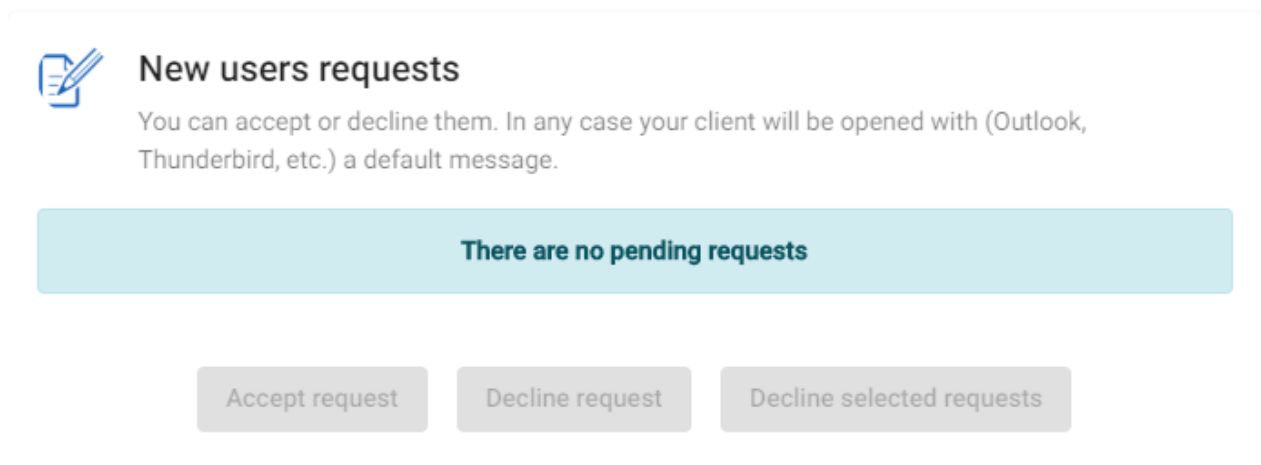
Figura 56. Posibles mensajes al intentar iniciar sesión.

El correo que recibirá el administrador incluirá **un enlace de un único uso que permite completar el proceso de inicio de sesión**, tal y como se observa en la Figura 57. El remitente de estos correos siempre será [noreply@hyperledger-jte.firebaseio.com](mailto:noreply@hyperledger-jte.firebaseio.com). Si se recibe un correo similar pero el remitente no es el indicado, **se recomienda no pulsar el enlace**.




*Figura 57. Mensaje recibido con el enlace de un único uso.*

Una vez que se pulsa el enlace, se abrirá de nuevo en el navegador web la aplicación. Pero en esta ocasión aparecerá una tabla con todas las solicitudes pendientes de nuevos usuarios. Si no hay ninguna solicitud pendiente el administrador verá la tabla vacía, como ocurre en el ejemplo mostrado en la Figura 58.



*Figura 58. Caso en el que no hay solicitudes pendientes.*

Aunque lo habitual es que haya varias peticiones sin revisar, por lo que el administrador visualizará estas solicitudes en forma de tabla, tal y como se muestra en la Figura 59.

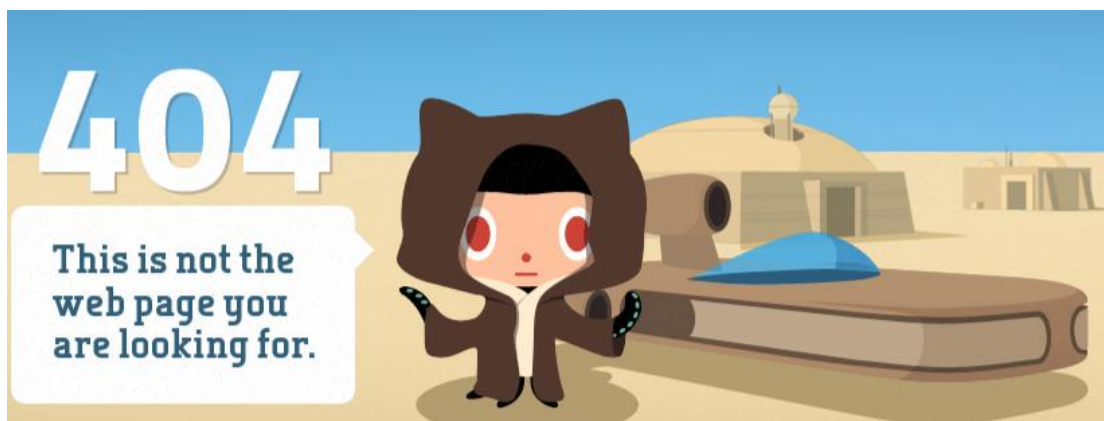
 **New users requests**  
You can accept or decline them. In any case your client will be opened with (Outlook, Thunderbird, etc.) a default message.

Email	Github ID	Firstname	Lastname	Gender	Birthdate	Job Position	Studies	Office	Select
test2@test.com	<a href="#">Test2</a>	Test2	Testing	FEMALE	1993-12-07	FORENSICS_TECHNICIAN	Software Engineer	Málaga	<input type="checkbox"/>
test@test.com	<a href="#">Test</a>	Test	Testing	MALE	1991-04-22	DETECTIVE	Law	Málaga	<input type="checkbox"/>

*Figura 59. Tabla con las solicitudes pendientes.*

Tal y como se ve, **cada entrada de la tabla muestra todos los datos del posible nuevo usuario**: su email, su nombre completo, su fecha de nacimiento, su puesto de trabajo, en qué oficina trabaja, etc. Aunque en realidad **hay dos columnas que son más interesantes que el resto: la columna ‘Github ID’ y la columna ‘Select’**.

La primera de estas dos columnas clave proporciona un enlace a la cuenta de Github del posible nuevo usuario, abriendo en el navegador web del administrador una nueva pestaña. **Esto es útil para verificar si la cuenta existe o no, y de esta forma descartar rápidamente esa solicitud**. Si la cuenta de Github no existe, la nueva pestaña que se abrirá en el navegador mostrará algo similar a lo que vemos en la Figura 60.

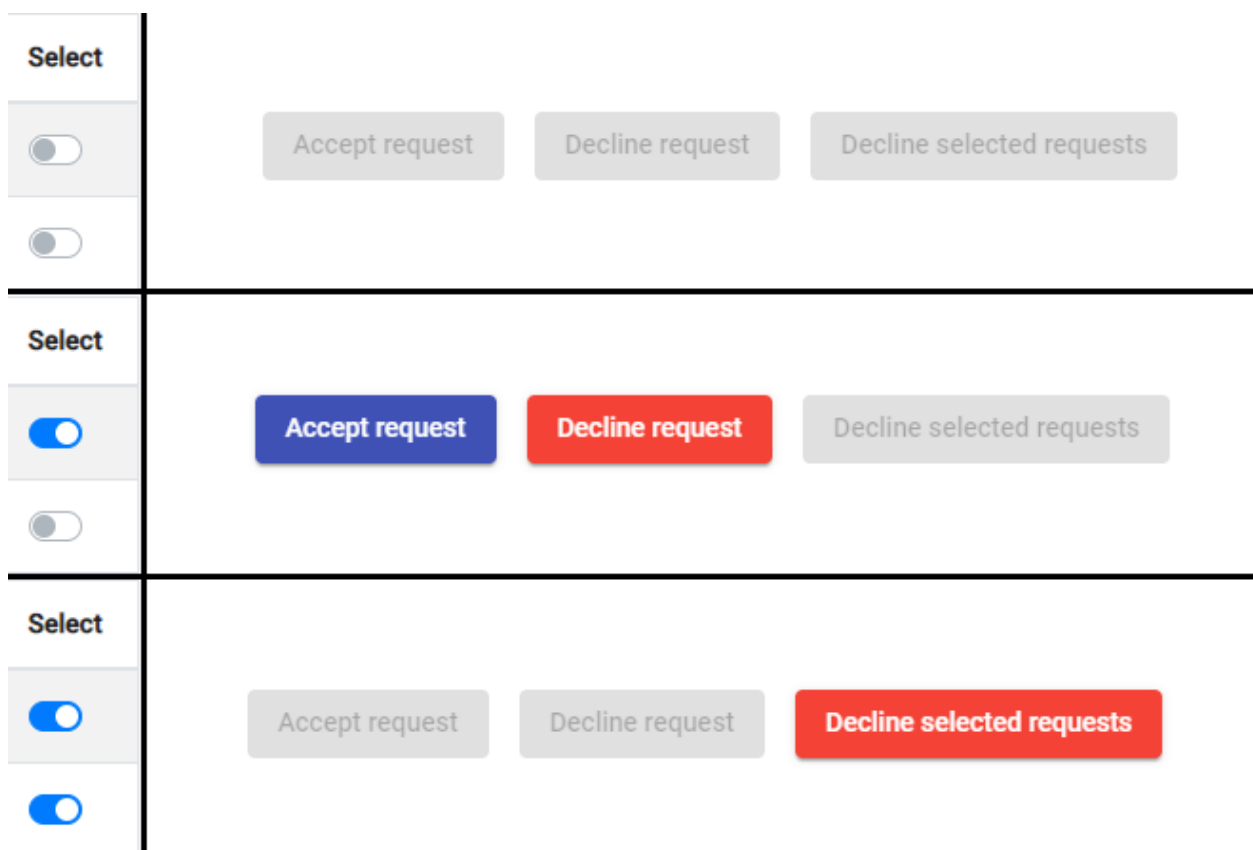


*Figura 60. Mensaje de Github cuando la cuenta indicada no existe.*

Como ya se ha dicho, la otra columna importante es '*Select*'. Esta columna permite seleccionar una o varias solicitudes para su procesamiento (aceptación o denegación). Según el número de peticiones seleccionadas, se habilitan o deshabilitan ciertos botones para ejecutar el procesamiento de esta solicitudes. Las posibles opciones son:

- **Ninguna solicitud seleccionada.** En este caso ningún botón estará habilitado.
- **Una única solicitud seleccionada.** Se habilitan los botones para aceptar esa solicitud ('***Accept request***') o para rechazarla ('***Decline request***').
- **Más de una solicitud seleccionada.** Se habilita el botón para rechazar varias solicitudes ('***Decline selected requests***').

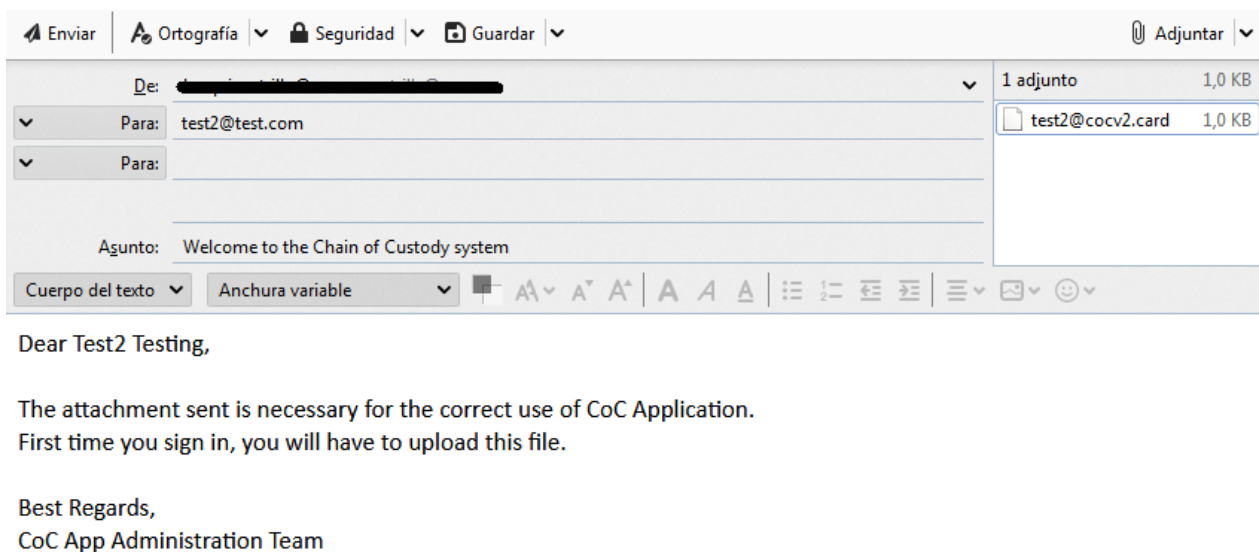
En la Figura 61 se muestra cómo se visualizarían las posibles opciones que se acaban de explicar.



*Figura 61. Estado de los botones de procesamiento según el número de solicitudes seleccionadas.*

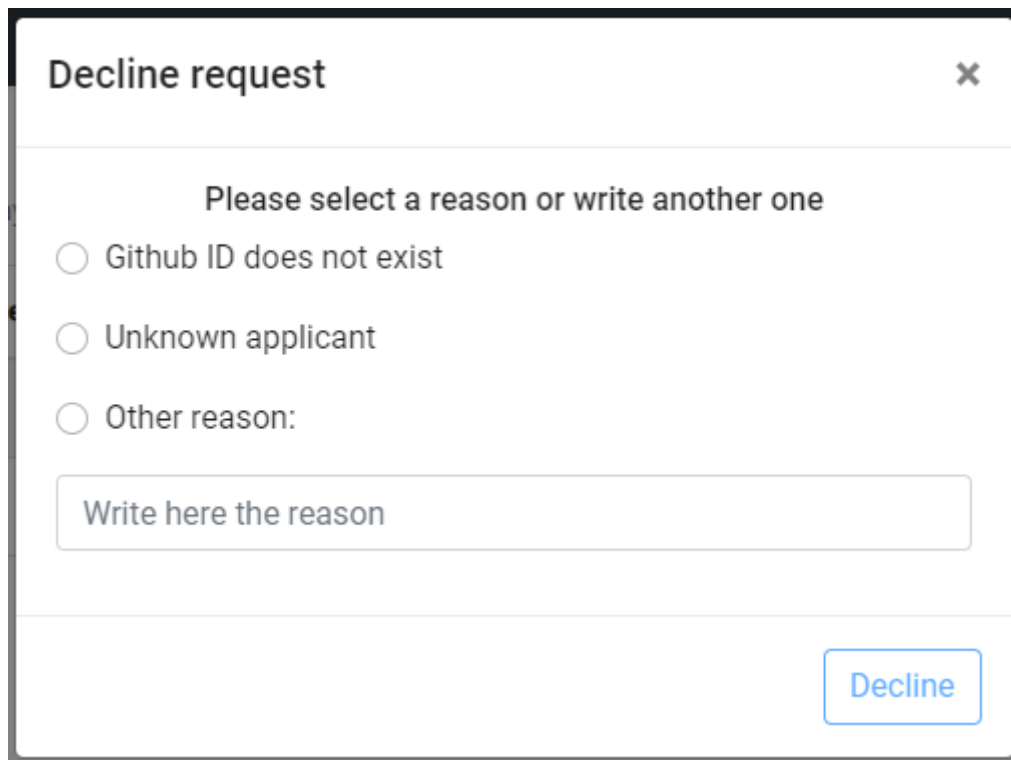
Por último, queda exponer el funcionamiento de cada uno de los botones.

Por un lado, el **botón de aceptación de solicitud** crea un participante en la *Business Network* y genera una identidad para dicho participante. Esta identidad es descargada en el navegador del administrador. **Se trata de un archivo con extensión .card**, con el siguiente formato: **ID-Github-User@cocv2.card**. Por ejemplo, en el caso de que el id de Github del nuevo usuario sea Test2, el nombre del archivo descargado será **Test2@cocv2.card**. Además, se abrirá el cliente de correo predeterminado del equipo del administrador con una plantilla de mensaje. El administrador deberá adjuntar en dicho mensaje la identidad descargada, tal y como se observa en la Figura 62.



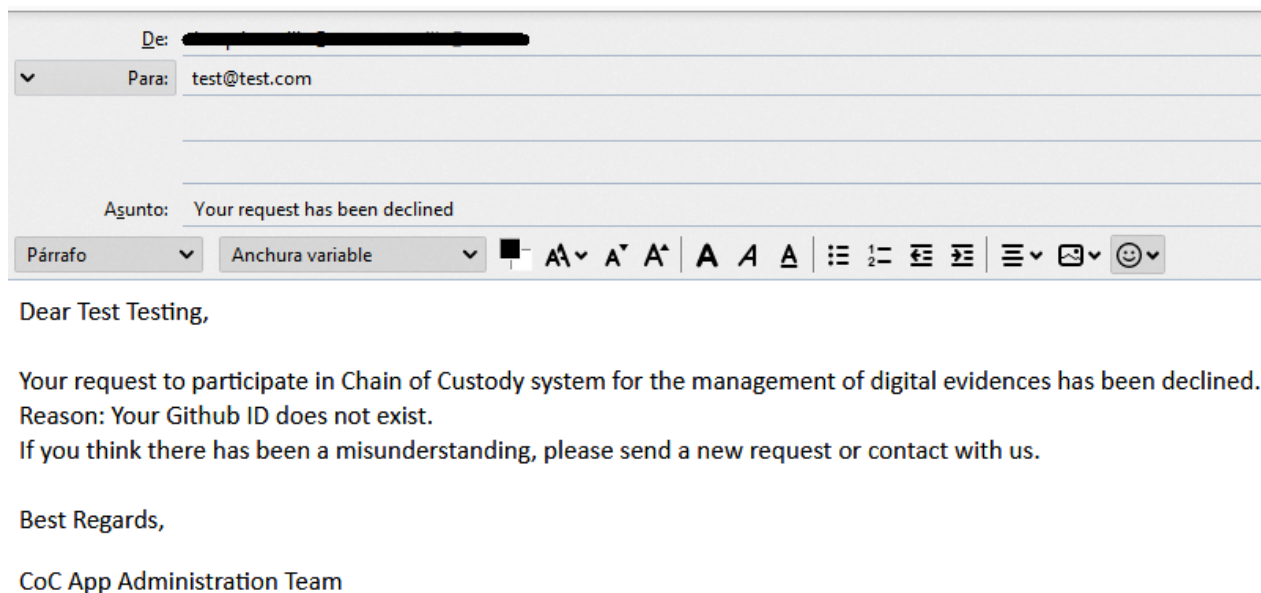
*Figura 62. Correo que debe enviar el administrador al nuevo usuario.*

Por otro lado, **los botones de rechazo de solicitudes** abren un diálogo como el que se ve en la Figura 63. Este diálogo requiere que el administrador seleccione un motivo por el cual se ha rechazado la solicitud (o solicitudes) o él indique un motivo distinto a los que se le muestran. Una vez se seleccione el motivo de rechazo, el botón '**Decline**' del diálogo se habilitará. Al pulsar dicho botón, la solicitud o solicitudes se eliminarán del sistema y se abrirá el cliente de correo predeterminado en el ordenador del administrador con un mensaje que incluye el motivo de rechazo (ver Figura 64).



A dialog box titled "Decline request" with a close button (X) in the top right corner. The main text says "Please select a reason or write another one". There are three radio button options: "Github ID does not exist", "Unknown applicant", and "Other reason:". Below these is a text input field with the placeholder "Write here the reason". At the bottom right is a blue button labeled "Decline".

*Figura 63. Diálogo que se abre al pulsar en los botones de rechazo de solicitudes.*



An email template with a header section containing "De:" (redacted), "Para: test@test.com", and "Asunto: Your request has been declined". Below the header is a rich text editor toolbar with options for paragraph, variable width, bold, italic, underline, link, unlink, list, and image. The body of the email contains the following text:

Dear Test Testing,

Your request to participate in Chain of Custody system for the management of digital evidences has been declined.  
Reason: Your Github ID does not exist.  
If you think there has been a misunderstanding, please send a new request or contact with us.

Best Regards,

CoC App Administration Team

*Figura 64. Correo que debe enviar el administrador si se rechaza alguna petición.*



## Referencias y bibliografía

---

- [1] A. Nieto y J. López, «Apuntes de la asignatura informática forense,» Universidad de Málaga, Málaga, 2018.
- [2] ISO/IEC JTC 1/SC 27 - IT Security techniques, *ISO/IEC 207037:2012 - Guidelines for identification, collection, acquisition and preservation of digital evidence*, 2012.
- [3] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.
- [4] BitinfoCharts.com, [En línea]. Available: <https://bitinfocharts.com/comparison/confirmationtime-btc-eth.html>. [Último acceso: 3 Diciembre 2018].
- [5] IETF, *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*, 2001.
- [6] S. Bonomi, M. Casini y C. Ciccotelli, «B-Coc: A Blockchain-based Chain of Custody for Evidences Management in Digital Forensics,» *arXiv:1807.10359*, 2018.
- [7] Geth, [En línea]. Available: <https://github.com/ethereum/go-ethereum/wiki/geth>. [Último acceso: 10 Diciembre 2018].
- [8] Hyperledger Fabric, [En línea]. Available: <https://www.hyperledger.org/projects/fabric>.
- [9] Proyectos Hyperledger, [En línea]. Available: <https://www.hyperledger.org/projects>.
- [10] Hyperledger Composer, [En línea]. Available: <https://www.hyperledger.org/projects/composer>.
- [11] Angular, [En línea]. Available: <https://angular.io/>.

[12] Firebase, [En línea]. Available: <https://firebase.google.com/>.

[13] Hyperledger Composer Modeling Language, [En línea]. Available: [https://hyperledger.github.io/composer/v0.19/reference/cto\\_language.html](https://hyperledger.github.io/composer/v0.19/reference/cto_language.html). [Último acceso: 10 Diciembre 2018].

[14] Passport.js, «Estrategias de autenticación,» [En línea]. Available: <http://www.passportjs.org/packages/>.