



UNIVERSIDAD DE MÁLAGA



Graduado en Ingeniería Informática

Trazabilidad de acciones de control en estaciones de carga de vehículos eléctricos mediante aplicación móvil y blockchain

Traceability of control actions in electric vehicle charge stations through mobile application and blockchain

Realizado por
Antonio Lara Gutiérrez

Tutorizado por
Rodrigo Román Castro
María Cristina Alcaraz Tello

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, junio de 2022



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

**Trazabilidad de acciones de control en estaciones de carga de
vehículos eléctricos mediante aplicación móvil y blockchain**

**Traceability of control actions in electric vehicle charge
stations through mobile application and blockchain**

Realizado por
Antonio Lara Gutiérrez

Tutorizado por
Rodrigo Román Castro
María Cristina Alcaraz Tello

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2022

Fecha defensa: julio de 2022

Resumen

Este TFG consiste en la creación de un ecosistema de trazabilidad de acciones de control para el uso en cualquier empresa de mantenimiento o gestor de estaciones de carga de vehículos eléctricos. Este permite implementar y llevar un control sobre todos los reportes enviados por el personal operario de dichas organizaciones, creando para ello un sistema novedoso de recogida y tratamiento de la información mediante una tecnología que nos permite realizar operaciones mucho más complejas que una solución basada en una base de datos convencional.

La tecnología empleada para el almacenamiento y visualización de los reportes enviados por los operarios se basa en una red de *blockchain* o red de cadena de bloques, específicamente, una red de *blockchain* permissionada, ya que su uso estará limitado a una única organización. El usar una tecnología de este tipo nos permite llevar a cabo operaciones desde una lógica distinta, manteniendo en todo momento un registro inmutable de reportes enviados, así como tener la seguridad de contar con un registro íntegro y protegido ante ataques. Todo ello es posible gracias al uso de esta tecnología de cadena de bloques, a la que tendremos acceso mediante dos aplicaciones creadas para tal efecto.

Estas dos aplicaciones consisten en una aplicación móvil para el sistema operativo *Android*, y en una aplicación web. La primera de ellas es la encargada de recoger los informes enviados por los operarios de campo para introducirlos en la *blockchain*, mientras que la segunda de ellas es usada por administradores, y es la encargada de administrar y visualizar toda la información recogida en la red.

Palabras clave: Vehículos eléctricos, *Blockchain*, Trazabilidad, Aplicación *Android*, Aplicación Web

Abstract

This Bachelor Thesis consists of the creation of an ecosystem of traceability of control actions for use in any maintenance company or manager of electric vehicle charging stations. This allows us to implement and keep track of all the reports sent by the operating personnel of these organisations, creating a novel system for collecting and processing information using a technology that allows us to perform much more complex operations than a solution based on a conventional database.

The technology used for the storage and visualisation of the reports sent by the operators is based on a blockchain network, specifically, a permissive blockchain network, as its use will be limited to a single organisation. Using a technology of this type allows us to carry out operations from a different logic, maintaining at all times an immutable record of reports sent, as well as having the security of having a complete record and being protected against attacks. All this is possible thanks to the use of this blockchain technology, to which we will have access through two applications created for this purpose.

These two applications consist of a mobile application for the Android operating system, and a web application. The first one is in charge of collecting the reports sent by the field operators to introduce them in the blockchain, while the second one is used by the administrators, and is in charge of managing and visualising all the information collected in the network.

Keywords: Electric vehicles, Blockchain, Traceability, Android Application, Web Application

Índice

1. Introducción	7
1.1. Motivación	7
1.2. Objetivos	9
1.3. Metodología	10
1.4. Estructura de la memoria	12
1.5. Conceptos clave	12
1.6. Acrónimos	13
2. Estudio Preliminar. Ecosistema de estaciones de carga	15
2.1. Vehículos eléctricos	15
2.2. Situación actual y futura	20
2.3. Infraestructura hardware y software de las estaciones de carga	21
2.4. Protocolo de comunicación OCPP	25
2.5. Otros protocolos de comunicación	27
3. Estudio Preliminar. Blockchains permissionadas	29
3.1. Redes de <i>blockchain</i>	29
3.2. Tipos de redes de <i>blockchain</i>	32
3.3. El contrato inteligente	35
3.4. Tecnologías <i>blockchain</i> permissionadas disponibles	36
3.4.1. <i>Ethereum</i>	37
3.4.2. <i>Corda</i>	38
3.4.3. <i>Hyperledger Fabric</i>	38
3.4.4. Otras tecnologías	40
3.5. Viabilidad en la prueba de concepto	40
4. Prueba de concepto	45
4.1. Introducción	45
4.2. Tecnologías utilizadas	48

4.2.1. Tecnologías <i>Blockchain</i>	48
4.2.2. Tecnologías <i>Android</i>	56
4.2.3. Tecnologías Web	63
4.3. Contrato inteligente	71
4.4. Aplicación móvil cliente	73
4.5. Aplicación web administrador	79
5. Conclusiones y Líneas Futuras	85
Apéndice A. Bibliografía	87
Apéndice B. Manual de Uso e Instalación	93
B.1. Creación de cuenta Kaleido y despliegue de red <i>blockchain</i>	93
B.2. Creación de cuenta en <i>Firebase</i>	99
B.3. Creación de claves API de <i>Google Maps Platform</i>	102
B.4. Preparación de las aplicaciones <i>Android</i> y web	103
B.5. Pasos posteriores	105
Apéndice C. Casos de Uso en el protocolo OCPP	107
Apéndice D. Listado de variables en el protocolo OCPP	109
Apéndice E. Documentación de Casos de Uso	113
Apéndice F. Documentación API REST Kaleido	121
F.1. Ejemplos prácticos	123
Apéndice G. Documentación del código fuente del contrato inteligente	125
G.1. Dependencias y método ejecutable	125
G.2. Estructura de datos	126
G.3. Métodos diseñados	127

Introducción

Sirva este apartado de introducción para presentar la motivación a la hora de realizar este TFG, los principales objetivos que se pretenden alcanzar con su ejecución, y la metodología usada para ello. Además, se recopila una serie de conceptos clave y acrónimos que servirán de soporte para la lectura de esta memoria.

1.1. Motivación

Este TFG surge como propuesta de caso de uso de aplicación de una de las últimas tecnologías exploradas en el campo informático y tecnológico: las redes de cadena de bloques o *blockchains*. Se trata de una tecnología novedosa que, aún en fase de desarrollo y crecimiento, se puede definir como una gran estructura de almacenamiento de datos inmutable, distribuida, segura y trazable [52].

Es esta última propiedad la que, en mayor medida, se ha querido sacar partido durante la ejecución de este TFG. La tecnología de *blockchains* tienen multitud de aplicaciones y casos de uso, siendo la implementación de tokens digitales o criptomonedas su aplicación más conocida (*Ethereum, Bitcoin, Solana...*), pero es un gran error limitar su uso a esta única aplicación, ya que el almacenamiento seguro e íntegro de la información que proporciona ofrece una gran ventaja con respecto a los sistemas gestores de bases de datos tradicionales, además de una infinidad de propuestas de uso [52].

Esta tecnología se aplicará sobre los ecosistemas de carga de los vehículos eléctricos. En los últimos años, el sector de los vehículos eléctricos en el mundo, así como en España, se encuentra en expansión. Se prevé que en 2023 haya más de 250.000 vehículos eléctricos matriculados en España, y todo el sector generaría más de 140.000 puestos de trabajo, directos e indirectos, contribuyendo más del 1 % al PIB español [21]. Además, desde la Unión Europea, en el programa *Objetivo 55. El plan de la UE para una transición ecológica* [61], se pretende “lograr

la neutralidad climática de aquí a 2050”, mediante la reducción de las emisiones contaminantes y la promoción del vehículo eléctrico para ello. Se prevé la creación de más de 3,5 millones de estaciones de carga de vehículos eléctricos para 2030, por lo que el caso de uso que se trata estará cada vez más presente en la sociedad, en los próximos años y décadas.

En la Figura 1 se puede observar el crecimiento exponencial de los últimos años en la matriculación de vehículos eléctricos en España, aunque se necesitaría alcanzar cifras muy superiores a lo largo de esta década.

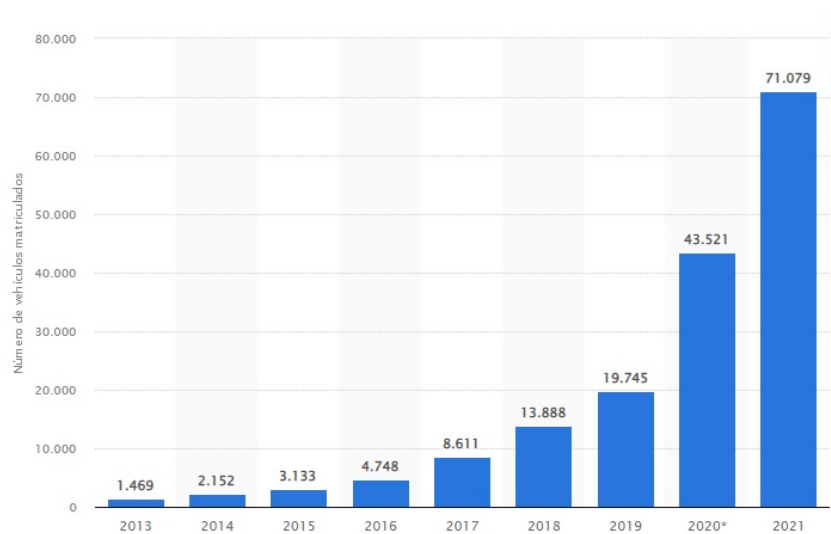


Figura 1: Evolución de las matriculaciones de vehículos eléctricos en España, a partir de los datos de la DGT. Fuente: [4].

Este TFG se centra en las estaciones de carga que se han desplegado, y que se desplegarán, sobre el territorio español, con el fin de cubrir la creciente demanda de vehículos eléctricos. Su misión es la de aportar un sistema de trazabilidad basado en una red de *blockchain* permissionada que permita trazar todas las operaciones de mantenimiento y gestión que el personal operario pudiera realizar sobre estas estaciones de carga.

En el escenario que se propone se distinguen dos tipos de operarios: los operarios de campo, que enviarán reportes a través de una aplicación móvil diseñada para ello directamente a la red, y los operarios del sistema central, que consultarán toda la información enviada a través de una aplicación web, garantizando la trazabilidad de todo el sistema. El uso de una red de *blockchain* permissionada en este caso resuelve problemas como la falta de confianza en la introducción de datos por parte de los operarios y su posibilidad de modificación si se emplease una base de

datos, y mejoraría la seguridad de dichas estaciones evitando fallos futuros de funcionamiento o de seguridad tanto para las organizaciones como para los propios usuarios de vehículos eléctricos.

Los reportes que los operarios deben enviar a la red se han elaborado en base a un estudio de la especificación de las estaciones de carga, y el diseño de su red de funcionamiento, todo ello recogido en un protocolo estándar de código abierto, OCPP (*Open Charge Point Protocol*) [40], que ha ayudado a definir los distintos casos de uso que los operarios pueden encontrarse como posibles tareas de gestión y mantenimiento.

1.2. Objetivos

En esta sección se presentan los objetivos principales del TFG. En primer lugar, realizar un estudio de las opciones actuales de implementación de la tecnología *blockchain* en el mercado, así como su viabilidad de uso en los ecosistemas de estaciones de carga de vehículos eléctricos, con un estudio sobre su funcionamiento y las tareas de mantenimiento y gestión que los operarios deben realizar sobre estas. En segundo lugar, se pretende implementar un caso práctico en el que se diseñará una aplicación móvil para el sistema operativo *Android* que permita enviar reportes de gestión y mantenimiento a los operarios de estaciones de carga de vehículos eléctricos, y una aplicación web que servirá como administrador, y que gestionará toda la información recibida por la aplicación móvil, aplicando trazabilidad sobre la red de *blockchain* permissionada implementada, así como otros aspectos relacionados con la gestión y mantenimiento de estaciones de carga de vehículos eléctricos.

En la Figura 2, confeccionada a partir de la documentación del estándar OCPP [40] y de los objetivos del trabajo, se puede ver un esquema del escenario que se propone como caso práctico en este TFG.

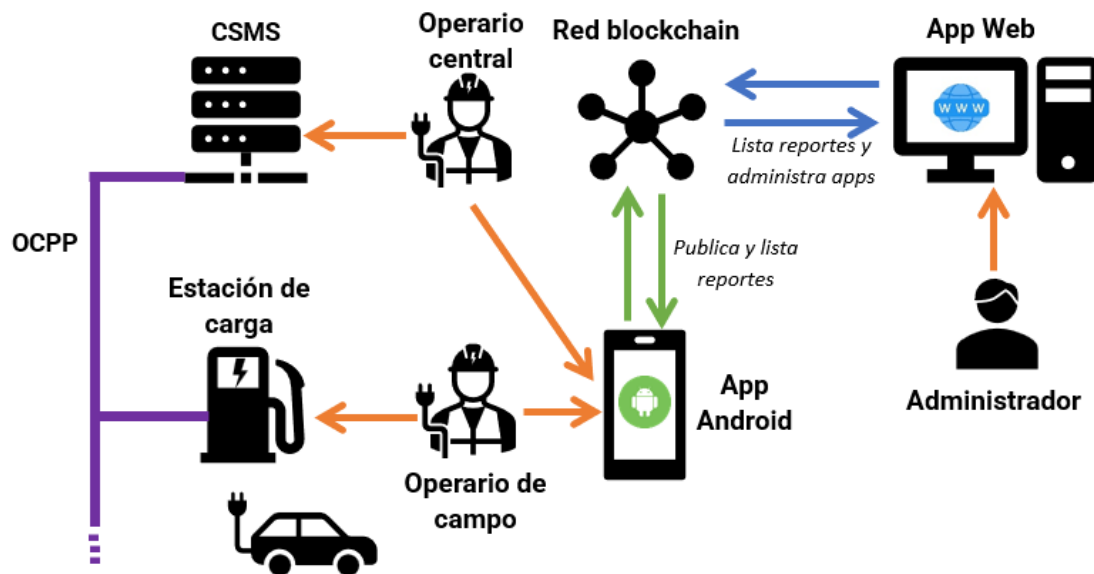


Figura 2: Escenario planteado en el caso práctico de este TFG.

En dicha Figura aparecen los distintos actores y tecnologías que se tendrán en cuenta para el caso práctico. Se puede observar que los dos operarios, tanto central como de campo, usarán la aplicación *Android* para publicar y listar sus propios reportes, mientras que el administrador encargado podría listar dichos reportes mediante una aplicación web y administrar ambas. Los operarios principales realizan tareas de gestión y/o mantenimiento de manera local o remota, y rellenan los reportes de su trabajo a continuación.

1.3. Metodología

En esta sección se detalla la forma de proceder a la hora de realizar este trabajo. Concretamente, se ha seguido un proceso iterativo de desarrollo software siguiendo las operaciones que conforma el ciclo de vida tradicional del software: análisis de requisitos, diseño, implementación y validación, y en base a un estudio sólido del problema y del estado del arte para conocer de primera mano los problemas a abordar y las tecnologías que se deben considerar.

Como se ha podido apreciar, hay dos partes bien diferenciadas dentro de este TFG, el primero corresponde con el estudio del estado del arte del despliegue de vehículos eléctricos y estaciones de carga. En él se presenta todo lo necesario para conocer toda la infraestructura y protocolos de comunicación que existen en el ecosistema de estaciones de carga de vehículos

eléctricos. Se realizará un estudio sobre su situación actual y futura tanto en Europa como en España, así como un estudio sobre la infraestructura hardware y software diseñada para su funcionamiento, especialmente el protocolo de comunicación OCPP [40], que se usa principalmente en estos sistemas para la comunicación de las estaciones con las centrales gestoras de estaciones de carga. En el segundo estudio preliminar, se introduce una breve historia sobre las redes *blockchain*, principalmente las redes permissionadas, que se tratarán en este TFG, así como el funcionamiento de estas y la principal tecnología escogida y su implementación.

Para el desarrollo del caso práctico y la implementación tanto de la aplicación móvil como de la aplicación web, se ha optado por seguir una metodología iterativa, preferida en todos los proyectos de Ingeniería del Software, para conseguir un buen resultado en la aplicación y evitar redundancias que puede provocar no seguir una metodología en ciclo para el desarrollo software. Se sigue, por tanto, un esquema de recogida de requisitos tanto funcionales como no funcionales, casos de uso, implementación por iteraciones y pruebas de funcionamiento.

Para llevar un mayor control y para comprobar que se han cumplido las distintas fases citadas de ciclo de vida de un proyecto software, se ha seguido una programación de tareas, que se puede visualizarse en el diagrama de Gantt de la Figura 3.

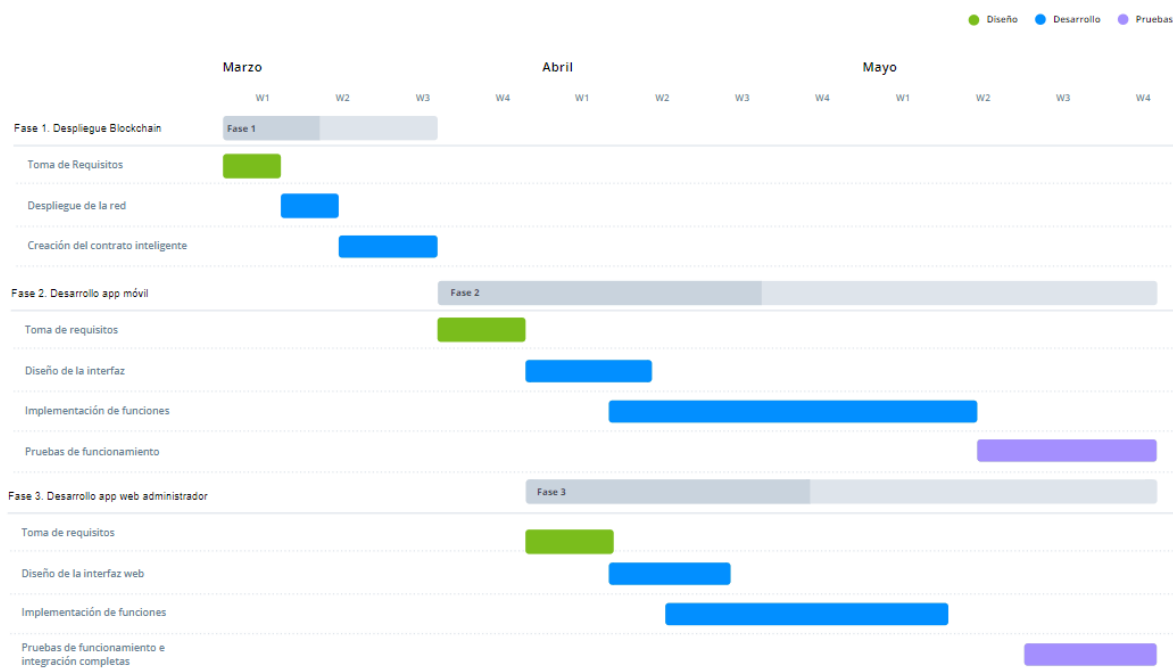


Figura 3: Diagrama de Gantt seguido para el desarrollo del caso práctico e implementación de ambas aplicaciones.

1.4. Estructura de la memoria

La memoria queda estructurada en capítulos, donde los Capítulos 2 y 3 se refieren a los dos estudios preliminares propuestos en los objetivos, respectivamente.

A continuación, en el Capítulo 4 se detalla la prueba de concepto, con la descripción de las tecnologías utilizadas y las decisiones de diseño en la implementación de cada aplicación.

Posteriormente, se incluye el Capítulo 5, el cual abarca las conclusiones y líneas futuras del proyecto.

En la última parte de la memoria se incluyen los Apéndices, los cuales contienen información y documentación útil y desarrollada de algunas partes clave de la memoria, así como la Bibliografía en el Apéndice A, y el Manual de Uso e Instalación, desarrollado en el Apéndice B.

1.5. Conceptos clave

En este apartado, se listan los principales conceptos usados para el desarrollo de este TFG, y a lo largo de esta memoria.

- **Vehículo eléctrico:** cualquier tipo de vehículo que base su funcionamiento en un motor alimentado por energía eléctrica, que la transforma en energía cinética para el movimiento de las ruedas [26].
- **Vehículo híbrido:** vehículo que cuenta tanto con un motor de combustión interna como un motor que funciona con energía eléctrica. Este último se encarga de asistir al primero, reduciendo la cantidad de gases contaminantes que expulsa. Sin embargo, pueden existir vehículos híbridos que funcionen con cierta autonomía eléctrica [26].
- **Transacción:** traspase de energía desde un EVSE a un vehículo eléctrico a través de un conector adecuado. El usuario del vehículo eléctrico inicia la transacción, y esta puede finalizar automáticamente o a petición de dicho usuario. La transacción también incluye el abono de precios por el servicio [40].
- **Smart Charging:** implementación de sistema de carga basado en distintos perfiles que la propia estación de carga escoge en base a ciertos parámetros como prioridad de uso, planificación, limitación actual de la energía..., etc. [18].

- **Heartbeat:** mensaje enviado por una CSMS a una estación de carga con cierta frecuencia, y este debe ser respondido para comprobar que la estación se encuentra activa y funcionando en la red. Un fallo en este mensaje puede indicar algún problema en la estación [40].
- **Blockchain:** conjunto de tecnologías descentralizadas y accedidas por los nodos de una red. La información es inmutable y segura, y se almacena en estructuras de datos conocidas como bloques [8].
- **Blockchain permissionada:** tipo de *blockchain*, en la que un nodo central se encarga de controlar las transacciones en la red, en la que sólo pueden participar nodos autorizados [8].
- **Contrato inteligente:** conjunto de reglas introducido en una *blockchain* y compartido por todos los nodos de la red. Permite definir instrucciones, funciones, constantes... que los nodos de la red pueden usar para facilitar y asegurar la introducción de datos [29].

1.6. Acrónimos

En este apartado, se presentan los distintos acrónimos usados en la redacción de esta memoria:

- **API:** Application Programming Interface
- **APK:** Android Application Package
- **BaaS:** Blockchain as a Service
- **BEV:** Battery Electric Vehicle
- **CS:** Charge Station
- **CSMS:** Charge Station Management System
- **CSS:** Cascading Style Sheets
- **DGT:** Dirección general de tráfico

- **EREV:** Extended Range Electric Vehicle
- **ERP:** Enterprise Resource Planning
- **EV:** Electric Vehicle
- **EVSE:** Electric Vehicle Supply Equipment
- **FCEV:** Fuel Cell Electric Vehicle
- **HEV:** Hybrid Electric Vehicle
- **HMI:** Human Machine Interface
- **IDE:** Integrated Development Environment
- **JDBC:** Java Database Connectivity
- **JPA:** Java Persistence API
- **JS:** JavaScript
- **JSON:** JavaScript Object Notation
- **MSP:** Membership Service Provider
- **OCA:** Open Charge Alliance
- **OCPP:** Open Charge Point Protocol
- **OSCP:** Open Smart Charging Protocol
- **PHEV:** Plug-in Hybrid Electric Vehicle
- **PIB:** Producto Interior Bruto
- **SaaS:** Software as a Service
- **SAVE:** Sistema de Alimentación del Vehículo Eléctrico
- **SCADA:** Supervisory Control And Data Acquisition
- **SIE:** Sistemas de Información Empresarial
- **ZBE:** Zona de Bajas Emisiones

Estudio Preliminar. Ecosistema de estaciones de carga

En este estudio preliminar, centrado en el ecosistema de las estaciones de carga de vehículos eléctricos, se investigará sobre la situación actual y futura de los vehículos eléctricos, la infraestructura hardware y software de las estaciones de carga, así como los principales protocolos de comunicación usados.

2.1. Vehículos eléctricos

Los vehículos a tracción convencionales usan motores de combustión interna, provocando una reacción entre un combustible inflamable y una chispa, y el gas generado por dicha combustión es el encargado de mover los pistones del motor, movimiento que posteriormente se transferirá a las ruedas.

Los vehículos eléctricos, por el contrario, no cuentan con el combustible en dicha reacción, para ello usan la energía eléctrica, que se transforma mediante una serie de mecanismos en energía cinética. Puede parecer que esta invención, que cada vez aparece como sustituta de los vehículos tradicionales de combustión interna, ha sido de nueva creación durante este siglo. Sin embargo, su invención se remonta a poco menos de dos siglos. Fue el inventor escocés Robert Anderson quien creó el primer vehículo que funcionaba con celdas eléctricas, como se puede ver en la Figura 4, y que fue desarrollado entre 1832 y 1839, mucho antes de la invención del motor de gasolina, por Nicolaus Otto en 1876.



Figura 4: Primer coche eléctrico, diseñado por Robert Anderson. Fuente: [14].

El vehículo eléctrico original tenía una autonomía de hasta 100 km, y una velocidad de 25 km/h. ¿Que ocurrió entonces para que, aún habiéndose inventado el motor de combustión en una fecha más tardía, este lograra la hegemonía del mercado hasta estos días? Básicamente, las mejoras introducidas por Henry Ford en la cadena de montaje, y la baja eficiencia y costes elevados que tenían los coches eléctricos, hicieron que se desterraran al olvido durante la mayor parte del siglo XX [9].

Sin embargo, a raíz de los movimientos sostenibles y de protección del medio ambiente, se ha intentado buscar soluciones a las emisiones contaminantes de los automóviles de combustión interna, optando por vehículos de emisiones reducidas o, directamente, sin emisiones. Así, los vehículos eléctricos volvieron a introducirse con fuerza en el mercado, y se podría decir que gozan de numerosos privilegios por parte de empresas y gobiernos, con el fin de incentivar su uso, en la búsqueda de un horizonte sostenible en el terreno de los desplazamientos por carretera.



Figura 5: Tesla Model S. Vehículo eléctrico moderno. Fuente: [53].

Los vehículos eléctricos actuales (véase un ejemplo en la Figura 5) se pueden clasificar en cinco tipos, según el tipo de motor o motores con los que cuentan [26]:

1. **Vehículos eléctricos de baterías (BEV):** son los coches eléctricos por excelencia. Funcionan exclusivamente con una batería de ion litio, que puede ser recargada directamente con energía eléctrica, a través de un punto de carga instalado en el vehículo. No emiten ningún gas contaminante a la atmósfera. Modelos de coches BEV son Opel Corsa o Renault ZOE.
2. **Vehículos eléctricos con pilas de combustible (FCEV):** para funcionar deben desencadenar una reacción de electrólisis, haciendo que el hidrógeno que contiene su pila de combustible se oxide, perdiendo electrones que posteriormente son capturados para la generación de la energía eléctrica que necesita el motor. No son muy frecuentes actualmente debido al alto coste energético y presupuestario de obtener hidrógeno puro, ya que este se presenta naturalmente como compuesto con otros elementos. El único residuo que emiten es el resultante de la oxidación de hidrógeno con oxígeno: agua. Modelos de coches FCEV son Toyota Mirai o Hyundai Nexo.
3. **Vehículo eléctrico de batería extendida (EREV):** funcionan con una batería eléctrica, al igual que los BEV, pero añadiendo un motor de combustión interna encargado de funcionar como generador para la batería, aumentando la autonomía del vehículo. No llega a ser considerado un vehículo híbrido, ya que, en ningún caso, el motor de combus-

ción interna es el encargado de ejecutar el movimiento de las ruedas. Modelos de coches EREV son Opel Ampera o BMW i3 Range Extender.

4. **Vehículo híbrido enchufable (PHEV):** funcionan con un motor clásico de combustión interna y un motor eléctrico. En este caso, uno no funciona como generador del otro, ya que el propio conductor puede elegir si usar el modo eléctrico o el modo combustible. Modelos de coches PHEV son Renault Captur E-Tech o SEAT León e-HYBRID.

5. **Vehículo híbrido no enchufable (HEV):** fueron los primeros coches eléctricos en reaparecer en el siglo XXI. Tienen un motor eléctrico y otro de combustión interna, funcionando principalmente con este último, y sirviéndose del eléctrico para asistirlo, reduciendo las emisiones contaminantes en cierta medida. Existen dos tipos:

- a) Híbrido completo (*full hybrid*): funciona con ambos motores a la vez, o con solo uno de ellos.
- b) Híbrido suave (*mild hybrid*): necesita siempre del motor de combustión interna para funcionar.

Modelos de coches HEV son Honda HR-V o Hyundai Kona.

Por otro lado, la DGT ha creado un distintivo ambiental con el fin de establecer una clasificación de los vehículos matriculados en territorio español según unos criterios de contaminación. Este distintivo se usa principalmente en políticas locales para restringir el acceso de automóviles en lugares altamente contaminantes como el centro de las ciudades (Madrid 360 o la ZBE del Área Metropolitana de Barcelona).

Sólo el 50 % de los vehículos del parque automovilístico español estarán identificados con un distintivo ambiental entre los disponibles, dejando al otro 50 % sin este. Existen cuatro tipos de distintivos regulados, dependiendo del impacto medioambiental de los motores de los vehículos a los que son concedidos [13]:

- **Etiqueta 0 emisiones. Color azul:** aquellos vehículos que no producen ningún tipo de emisiones contaminantes a la atmósfera. Se otorga a los vehículos BEV, EREV, FCEV y PHEV (estos últimos siempre que el motor eléctrico tenga una autonomía de, al menos, 40km).

- **Etiqueta ECO. Colores azul y verde:** principalmente para vehículos híbridos o con baja autonomía eléctrica. Se otorga a los vehículos HEV, PHEV (si tienen una autonomía eléctrica de menos de 40km), y vehículos propulsados por gas natural o gas licuado del petróleo.
- **Etiqueta C. Color verde:** vehículos de combustión interna que cumplen las últimas normativas de emisiones europeas EURO [11].
- **Etiqueta B. Color amarillo:** vehículos de combustión interna que no cumplen las últimas normativas de emisiones europeas EURO [11], pero sí cumplen las previas.



Figura 6: Imágenes gráficas de las etiquetas por distintivo ambiental. Fuente: [13].

En la Figura 6 se observan las representaciones gráficas de los distintivos ambientales presentados.

En el Cuadro 1 se sintetiza todo lo presentado hasta ahora.

Tipo	Carga	Distintivo ambiental	Reducción de emisiones CO2
BEV	Eléctrica	0	100 %
EREV		0	Depende del uso. Hasta 100 %
PHEV		Autonomía > 40 km: 0 Autonomía < 40 km: ECO	50-75 %
HEV	Combustible	ECO	<i>Mild hybrid:</i> 10-20 % <i>Full hybrid:</i> 20-40 %
FCEV	Hidrógeno	0	100 %

Cuadro 1: Comparativa entre tipos de vehículos eléctricos. Fuente: [34].

2.2. Situación actual y futura

Una vez presentada la tecnología y la historia que ha hecho olvidar y resucitar los vehículos eléctricos, en esta sección se tratará la situación tanto actual como futura del vehículo eléctrico en España. Se tomarán como referencia los contenidos del informe *El vehículo eléctrico en España. Situación actual, objetivos y retos a abordar* [34].

Muchos de los objetivos actuales de desarrollo y promoción de los vehículos eléctricos en España, con el fin de reducir las emisiones contaminantes que los vehículos tradicionales emitían, son dictadas en el marco de la Unión Europea, y adoptadas por sus estados miembros. La base de legitimación y puesta en marcha de objetivos nace a partir del Pacto Verde Europeo, o *Green Deal* [60], que establece distintos horizontes para el cumplimiento de medidas de ahorro energético y contra la contaminación por parte de los países de la Unión. La medida que afecta a España, viene detallada, enmarcada como una política dentro del Pacto Verde Europeo, en el programa *Objetivo 55. El plan de la UE para una transición ecológica* [61].

En dicho plan, se pretende “lograr la neutralidad climática de aquí a 2050”, en base a una serie de objetivos y/o metas bien definidos:

- Objetivos de reducción de emisiones de CO_2 : para turismos, se busca una reducción del 15 % en 2025, del 55 % en 2030, y del 100 % en 2035.
- Objetivos de estaciones de recarga eléctrica públicas. Se debe asegurar una red de estaciones “ampliamente disponible, interoperable, fácil de usar y a intervalos fijos a lo largo de los principales corredores de transporte de Europa”. Para ello, se fija como objetivo tener operativas 1 millón de estaciones en 2025, 3,5 millones en 2030, hasta alcanzar más de 16 millones de estaciones en 2050.

Estos objetivos se basan en las previsiones de existencias de vehículos eléctricos en las próximas décadas. La Unión Europea prevé, para el año 2030, contar con más de 40 millones de vehículos eléctricos en la región [61].

Si se trasladan estos objetivos a España, estos deben ser adoptados de igual manera por todos los estados miembros. El Gobierno de España, con la Ley 7/2021, de 20 de mayo, de cambio climático y transición energética [32], recoge los objetivos a medio y largo plazo de implantación de las medidas recogidas en el Pacto Verde Europeo. Se establece que en el país

debe contar con 5 millones de vehículos eléctricos para 2030, y más de 2 millones de puntos de recarga para dichos vehículos.

Para cumplir estos objetivos, el Gobierno ha puesto en marcha varios planes de ayudas económicas para la adquisición de modelos de vehículos eléctricos, como el “*PERTE para el desarrollo del vehículo eléctrico y conectado*”, aprobado el 13 de julio de 2021 [43], con el objetivo de “*crear en España el ecosistema necesario para el desarrollo y fabricación de vehículos eléctricos y conectados a la red y convertir a España en el Hub Europeo de electromovilidad*”. Se pretende dedicar 24.000 millones de euros hasta 2023 para la persecución de este plan.

2.3. Infraestructura hardware y software de las estaciones de carga

Uno de los puntos clave a la hora de implantar el vehículo eléctrico en el día a día pasa por mantener una red de estaciones de carga que, al igual que las actuales gasolineras, permita a los usuarios de vehículos eléctricos poder recargar sus baterías.

En España existe, a fecha del 21 de junio de 2022, 11.136 estaciones de puntos de carga a lo largo del territorio nacional, con un total de 29.516 plazas o conectores únicos disponibles [16].

Para empezar a ver cómo funcionan las estaciones o puntos de carga, es necesario conocer primero su clasificación. Esta puede ser según la velocidad de carga, el modo de recarga del vehículo eléctrico, y el tipo de conector que usa:

1. **Velocidad de carga:** dependiendo de las necesidades del usuario del vehículo eléctrico, los puntos de carga pueden suministrar diferentes niveles de energía para realizar la carga de tres tipos distintos:
 - **Carga lenta:** utilizada cuando no se pretende hacer uso del vehículo, normalmente en espacios domésticos, ya que solo requieren de un conector e infraestructura convencional. La carga completa puede durar hasta 6 horas en este modo.
 - **Carga semi-rápida:** utilizada en zonas públicas y privadas, requiriendo de conectores e infraestructuras dedicadas. La carga completa puede durar un par de horas.
 - **Carga rápida:** utilizada en cuando se pretende seguir haciendo uso del vehículo, como una parada en una estación de servicio. Pueden permitir recargar el vehículo en unos 30 minutos. Al requerir de infraestructura especial y más compleja, se limita su uso en estaciones de servicio o *electrolineras*.

2. **Modo de recarga:** dependiendo de la conexión física que el vehículo acepta para realizar la recarga, se distinguen 4 modos:

- **Modo 1:** la recarga se realiza directamente mediante corriente alterna monofásica, con una intensidad máxima de 16 amperios.
- **Modo 2:** al igual que en modo 1, pero permitiendo una intensidad máxima de 32 amperios.
- **Modo 3:** la recarga se realiza mediante un Sistema de Alimentación del Vehículo Eléctrico (SAVE), y este incluye funcionalidades extra como protección ante subidas de tensión, administración de la energía... Es el modo más usado.
- **Modo 4:** la recarga se realiza mediante corriente continua, en estaciones de carga rápida, donde las funciones del SAVE quedan en manos del punto de recarga.

3. **Tipo de conector:** existen numerosos conectores estándar en el mercado, al no existir un estándar único, y cada punto de carga puede ofrecer una o varias conexiones con uno o varios tipos de conectores:

- **Tipo 1 (SAE J1772):** conector más usado en América y Asia (véase Figura 7).



Figura 7: Conector Tipo 1 SAE J1772. Fuente: [57].

- **Tipo 2 (IEC 62196):** conector más usado en Europa (véase Figura 8).



Figura 8: Conector Tipo 2 IEC 62196. Fuente: [57].

- **Schuko:** conector doméstico convencional, usado para el Modo 1 de carga (véase Figura 9).



Figura 9: Conector convencional Schuko. Fuente: [57].

- **Carga inductiva:** no presenta conector físico, y la carga se realiza mediante inducción. Actualmente se encuentra en investigación y se suele usar principalmente en autobuses o furgonetas (véase Figura 10).



Figura 10: Representación de la carga inductiva. Fuente: [49].

Las estaciones que se tratarán en este estudio son aquellas desplegadas con titularidad pública o privada a lo largo del territorio. El diseño de estas estaciones o puntos de carga sigue una topología en cuatro capas [42]:

1. En el **nivel 1** se encuentran las CSMS (sistema de control de las estaciones de carga). Estos sistemas son los encargados de controlar y monitorizar de forma remota distintas estaciones de carga.
2. En el **nivel 2** aparecen las propias CS (estaciones de carga). Las estaciones de carga engloban todos los equipamientos y conectores inferiores, y proporciona la infraestructura necesaria para ello.
3. El **nivel 3** lo ocupan los EVSE (equipamiento de suministro de energía). Se refiere a cada uno de los puestos que proporcionan energía dentro de una estación de carga. Deben adaptarse al modo y tipo de carga que cada vehículo eléctrico necesita.
4. Finalmente, en el **nivel 4** se encuentran los conectores. Los conectores se enchufan directamente al vehículo, y proporcionan la carga necesaria para este.

De este modo, una CSMS controla varias CS, que estas controlan uno o varios EVSE, y estos pueden disponer de uno o varios tipos de conectores disponibles para el usuario (dependiendo del modelo que use su dispositivo). En Figura 11 se resume toda esta información:

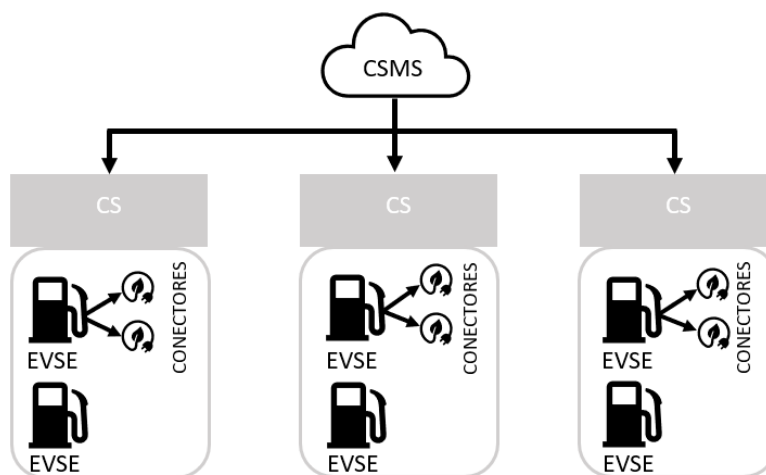


Figura 11: Esquema con topología de cuatro niveles del ecosistema de estaciones de carga.

Ya se ha alcanzado el punto de aplicación de este trabajo. En este proyecto se tratarán las tareas de administración y gestión que se realizan desde los sistemas de control (CSMS) y en las propias estaciones de carga, que los operarios deben ir registrando, haciendo uso de la prueba de concepto que se ha diseñado para ello. Para finalizar con el estudio sobre los ecosistemas de estaciones de carga, es deber conocer cómo se realiza la comunicación entre la CSMS y las diferentes estaciones de carga (CS). El protocolo de comunicación entre la CSMS y el CS es realizado a través del protocolo de control industrial: OCPP.

2.4. Protocolo de comunicación OCPP

El protocolo de comunicación OCPP (véase Figura 12) ha sido desarrollado por la Open Charge Alliance (OCA), que se trata de un consorcio global que pretende estandarizar las comunicaciones entre las infraestructuras dedicadas a los vehículos eléctricos [41].



Figura 12: Logotipo de la *Open Charge Alliance*. Fuente: [41].

La versión que se ha usado para la realización de este proyecto ha sido la 2.0.1 [40], lanzada originalmente en abril de 2018. La comunicación está basada en mensajes JSON, y desde esta puede manejarse cualquier tipo de parámetros que las distintas estaciones de carga y demás niveles inferiores manejan. Esta última actualización, una de las más importantes, incorpora nuevos mecanismos de seguridad: notificación de eventos de seguridad e incorporación de tres perfiles de seguridad:

- **Perfil 1:** Transporte no seguro con autenticación HTTP básica. Solo debe ser usado en redes de producción.
- **Perfil 2:** Transporte con TLS usando autenticación básica.

- **Perfil 3:** Transporte con TLS y autenticación con certificados en el lado del cliente.

Además, agrega soporte a funciones inteligentes de carga, buscando mayor compatibilidad con la gama de vehículos eléctricos actualmente en el mercado.

El uso del protocolo OCPP en una red industrial se ubica en la Figura 13.

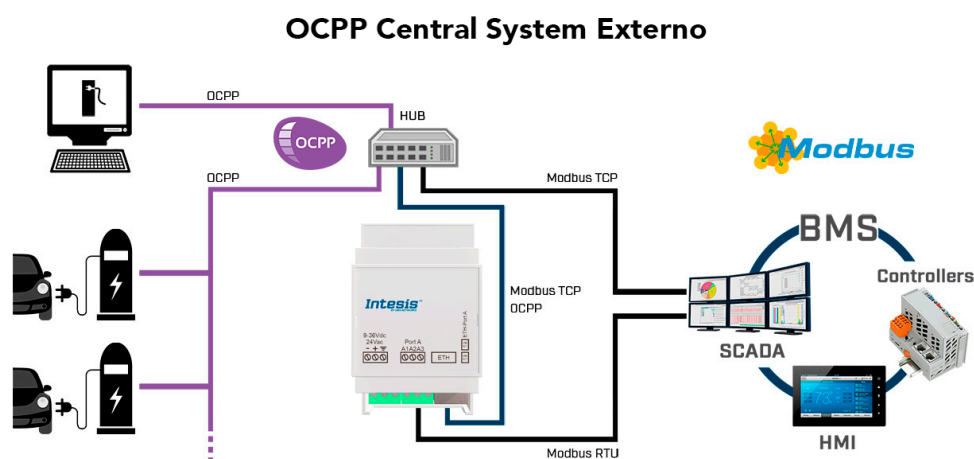


Figura 13: Esquema de conexiones de un sistema control de estaciones de carga de vehículos eléctricos. Fuente: [15].

El protocolo OCPP está presente en la comunicación entre la CSMS y las estaciones de carga. La información que se envía y recibe (al tratarse de un protocolo bidireccional) puede ser recogida a través de un *hub*, y esta ser transmitida al área de gestión de la empresa administradora. El área de gestión puede integrar sistemas SCADA, de supervisión y recogida de datos, junto a Interfaces Hombre-Máquina (HMI) para la visualización y administración de todo el flujo de datos.

Para este trabajo se han recopilado los distintos casos de uso que involucran operaciones de gestión y administración y que están definidos en el protocolo. Estos casos de uso formarán parte de los supuestos incluidos en los reportes que los operarios podrán enviar a la red a través de la aplicación móvil.

Uno de los casos de uso que se contemplan recoge la posibilidad de modificar variables por parte del operario. Se ha realizado otro estudio exhaustivo de recogida y recopilación de datos sobre las distintas variables que pueden ser modificadas según el estándar.

Por simplicidad, se remite al lector a los apéndices C y D, en los que se presenta una tabla con los casos de uso y las variables disponibles para modificar, respectivamente.

2.5. Otros protocolos de comunicación

Aunque este proyecto se ha centrado en la recopilación de datos en base a la especificación del estándar OCPP, las estaciones de carga no sólo cuentan con este protocolo de comunicación para comunicarse con las centrales de control.

Antes de su implantación, el mercado estaba controlado por redes y protocolos de comunicación propietarios, que no facilitaban la implantación masiva de estaciones de carga, ya que estas deberían adaptarse a los requisitos hardware y software de cada protocolo.

Ahora, gracias a la aparición de este protocolo, las estaciones de carga pueden adaptarse fácilmente a nuevos cambios, y no necesitan de software ni hardware propietario para funcionar, facilitando su expansión.

Sin embargo, aún existen otros protocolos de comunicación, menos conocidos, que pueden ser usados para estos fines. Normalmente corresponden con redes cerradas de comunicación, como es el caso de la red de supercargadores de Tesla (*Tesla Supercharger*, representado en la Figura 14).



Figura 14: Fotografía de una estación de carga *Tesla Supercharger*. Fuente: [54].

Estudio Preliminar. Blockchains permissionadas

Este segundo estudio preliminar se enfoca en las redes de *blockchain*, y para ello documentará qué son las redes de *blockchain* actualmente y lo que se espera que puedan lograr en el futuro de su desarrollo, los diferentes tipos de redes de *blockchain* de tipo permissionadas que existen actualmente. Finalmente, se analizará la viabilidad de la tecnología *blockchain* y la solución escogida para el desarrollo de la prueba de concepto.

3.1. Redes de *blockchain*

El concepto de red de *blockchain* engloba, a su vez, muchos otros conceptos de distintas áreas de la tecnología y de la informática. El inicio de este se remonta a 2008, con la publicación del informe “*Bitcoin: A Peer-to-Peer Electronic Cash System*”, por uno o varios autores bajo el pseudónimo de Satoshi Nakamoto [48]. Dicho informe sienta las bases sobre un nuevo mecanismo de pagos online que evita la necesidad de contar con instituciones financieras. Con este nuevo sistema, los usuarios de internet podrán enviar y recibir dinero a través de tokens digitales, más conocidos como criptomonedas, con cualquier otro usuario de la red. Este procedimiento estará basado en pruebas criptográficas, en vez de depender de un tercero, que normalmente se trata de una institución financiera, ya que este incrementa los gastos de la transacción y puede limitar a los usuarios en las transacciones que estos deseen realizar.

Bitcoin (véase Figura 15) permite, a través de un mecanismo de **clave pública-privada**, controlar todas las criptomonedas existentes en la web, permitir su traspase de un usuario a otro, y evitar que estas puedan ser copiadas.



Figura 15: Logo de la criptomoneda *Bitcoin*, junto a una estatua dedicada al autor de la misma, Satoshi Nakamoto, en Budapest.

La propuesta que daría lugar al funcionamiento de *Bitcoin* y, posteriormente, a las redes de *blockchain* en general, trata de una cadena de bloques de información conectados uno detrás de otro y distribuidos por toda la red. Cada bloque de información, además de otros campos, cuenta con un campo de **hash actual** y otro con el *hash* del bloque que lo precede, pudiendo formar así una serie de bloques encadenados, como se puede observar en la Figura 16.

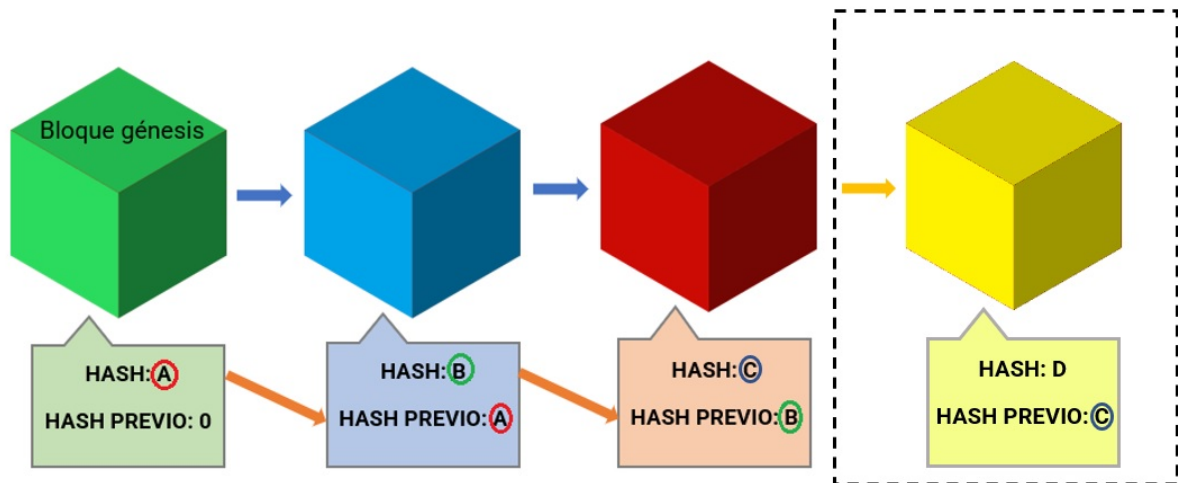


Figura 16: Esquema de bloques de información en una red de *blockchain*.

En dicha Figura, cada bloque contiene información sobre su *hash*, y el *hash* del bloque inmediatamente anterior en la red, a excepción del primer bloque, conocido como **bloque génesis**. Este bloque marca el inicio de una cadena.

Se supone ahora que un nuevo bloque, el bloque amarillo, ha sido creado por un usuario y va a ser introducido en la red. Antes de introducirse, el bloque se envía a todos los nodos activos de la red, y este espera a que validen su transacción. Una vez que todos los nodos han confirmado la transacción, este se inserta en la red, tomando como *hash* previo el valor del *hash* del último nodo de dicha red (en este caso, C).

Este modo de funcionamiento proporciona un registro de todas las transacciones de la red con tres características fundamentales, que diferencian esta tecnología de todas las soluciones anteriores [25]:

- **Seguridad:** las redes de *blockchain* utilizan en gran medida elementos criptográficos ampliamente usados como *hashes* o criptografía de clave pública, que, junto con la validación necesaria de todos los nodos de la red en cada transacción, certifican que todas ellas se llevarán acabo de manera segura.
- **Eficiencia:** al contar un registro compartido entre todos los nodos, no es necesario ejecutar tareas de sincronización entre registros, registros duplicados o alterados en un nodo... etc.
- **Inmutabilidad:** todas las transacciones son registradas en la red, y la información no puede ser alterada, ya que inmediatamente el resto de nodos puede verificar que la información que un nodo alterado emite es incorrecta para esa red.

Desde la introducción de esta estructura de datos, los casos de uso y aplicación de esta no han hecho más que aparecer. Principalmente, puede ser usada en aquellos supuestos en los que una base de datos centralizada es ineficiente y costosa y no posee de mecanismos de seguridad y recuperación ante posibles ataques. Las redes de *blockchain* permiten llevar un registro de todas las transacciones, pudiendo almacenar en ellas todo tipo de información.

Sin embargo, también existe una serie de problemas que el uso de esta tecnología genera, como es el gran consumo energético que en ciertos casos¹ se necesita para la introducción de

¹Dependiendo de la red de *blockchain* implementada y de sus algoritmos de consenso.

un bloque en la red (ya que este debe ser comunicado a todos los nodos, y, al ser un sistema altamente escalable, puede multiplicar exponencialmente el envío de mensajes entre nodos de la red), la falta de interoperabilidad entre sistemas que implementen *blockchain* (al ser una estructura altamente personalizable), o su escalabilidad con manejo de datos a nivel global [25].

Si se observan las tendencias de búsqueda en Google para el término “*blockchain*”, accesibles desde [Google Trends](#) y representadas en la Figura 17, la introducción de la tecnología no causó prácticamente ningún interés en internet. Sin embargo, este interés fue creciendo desde 2014, unos 5 años después, para alcanzar un pico de interés en diciembre de 2017, que coincide con una gran revalorización de *Bitcoin* en el mercado, alcanzando un valor de más de 15.000 € por *Bitcoin*. Posteriormente, llegó a revalorizarse en casi un 20.000 % de su valor original.

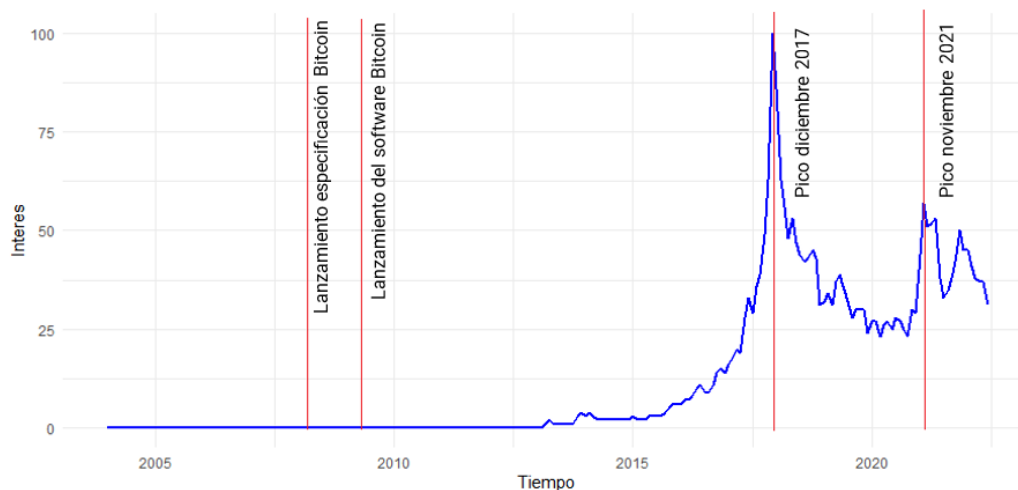


Figura 17: Análisis de tendencia de búsqueda del término “*blockchain*” en todo el mundo.

Elaboración propia usando R a partir de los datos de [Google Trends](#).

3.2. Tipos de redes de *blockchain*

En esta sección se presentarán y estudiarán los distintos tipos de redes de *blockchain* que se han ido implementando desde la concepción de esta tecnología [29].

A gran escala, existen cuatro tipos de redes:

1. **Redes de *blockchain* públicas o no permissionadas:** son aquellas redes en las que cualquier usuario de internet puede participar, sin necesidad de identificación previa.

Deben tener mayor capacidad computacional, y en contrapartida, las transacciones son públicas, y cualquier usuario que conozca nuestra dirección o clave pública puede obtener todo el historial de nuestra actividad en la red, aunque esta sea anónima. Ejemplos de redes de *blockchain* públicas son *Bitcoin* o *Ethereum*.

2. **Redes de *blockchain* privadas:** son aquellas redes en las que usuarios autenticados pueden participar en la red. Necesitan identificación previa, que suele ser otorgada por una organización central. Esta organización se encarga del mantenimiento exclusivo de la red, de los miembros autorizados para hacer uso de ella, de los registros compartidos... Estas redes son comúnmente implementadas en empresas, requiriendo de un equipo que prepare y configure el acceso a los empleados, por lo que estos tipos de redes aseguran una mayor eficiencia y seguridad que las redes públicas. Ejemplos de redes de *blockchain* privadas que pueden implementarse son *Hyperledger* o *Corda*.
3. **Redes de *blockchain* federadas o de consorcio:** son aquellas redes privadas en las que la administración de la red no recae sobre una única organización. Varias organizaciones comparten las tareas de gestión y mantenimiento de la red, decidiendo quiénes pueden pertenecer a estas. Este tipo de redes se suele usar cuando se necesitan de varios actores en la actividad empresarial, como por ejemplo, en la trazabilidad de envíos, donde aparecen productores, proveedores, empresas de envío... Este tipo de redes también pueden implementarse, al igual que las privadas, haciendo uso de *Hyperledger* o *Corda*.
4. **Redes *blockchain* híbridas:** son aquellas redes privadas que permiten acceso a ciertos componentes a usuarios no autorizados. Solo una parte de la red tiene acceso limitado, y es la organización encargada de administrarla la que debe decidir sobre las operaciones que los usuarios autenticados y no autenticados pueden llevar a cabo.

Una vez presentados los distintos tipos de redes de *blockchain* que existen, se procede a comparar las características que ofrecen cada una de ellas. Esto puede ser útil a la hora de escoger el mejor tipo de red para el desarrollo de la prueba de concepto.

Característica	Red pública	Red privada	Red federada	Red híbrida
Acceso libre	✓	✗	✗	●
Transparencia	✓	●	●	●
Eficiencia	●	✓	✓	✓
Administrador único	✗	✓	✗	●
Sin administrador	✓	✗	✗	✗
Igualdad	✓	✗	✗	✗
Permite ejecutar contratos inteligentes	✓	✓	✓	✓
Mecanismos de falta de confianza	✓	✗	●	●
Protocolos de consenso	✓	✗	●	●
Aislamiento	✗	✓	✓	●

Cuadro 2: Comparación por características entre los tipos de redes de *blockchain* existentes. Donde un círculo verde es un resultado positivo, rojo es un resultado negativo, y amarillo es intermedio.

La información recogida en el Cuadro 2 provee de datos para buscar cuál es la mejor opción a la hora de escoger una red de *blockchain* para un caso de uso concreto. Por ejemplo, si se busca una red accesible y abierta a todos los usuarios de la red (características de *Acceso libre* y *Transparencia*), descentralizada (característica de *Sin Administrador*) y en la que todos los nodos posean los mismos privilegios (característica de *Igualdad*), se estará hablando de una **red pública**.

Por el contrario, si se quiere limitar el acceso a la red a un cierto grupo de usuarios y que estos desempeñen roles distintos, se necesitará de una **red privada o federada**, donde la diferencia entre ambas radica en la existencia de un administrador único o no, respectivamente. Para el último caso, si se busca fusionar las características tanto de redes públicas como permissionadas (privadas y federadas) se hará uso de una **red híbrida**.

3.3. El contrato inteligente

Un contrato, en su definición original y según el artículo 1254 del Código Civil [46], “*existe desde que una o varias personas consienten en obligarse, respecto de otra u otras, a dar alguna cosa o prestar algún servicio*”. Un **contrato inteligente** también implica a una o varias partes en la prestación de un servicio, pero este está definido usando un lenguaje de programación, y puede ejecutarse y cumplirse autónomamente. Los principales objetivos que se persiguen con la creación de un contrato inteligente es el de reforzar la confianza y la seguridad entre las partes implicadas, evitando interpretaciones humanas e injerencias de terceras personas [27].

Casi todas las redes *blockchain* existentes permiten la implementación de un *smart contract* o contrato inteligente. Se propone un ejemplo para entender su funcionamiento:

Un usuario de internet desea realizar un depósito de dinero. Este dinero no tiene por qué ser dinero real, puede tratarse de tokens digitales, monedas dentro de una aplicación, puntos de fidelización..., etc. Dicha transferencia va a estar gobernada por una red de *blockchain* pública. Cuando dicho usuario quiere realizar el depósito, indica a la aplicación (que se llamará *EtherBank*) que desea introducir cierta cantidad. La aplicación propone a la red la transacción y esta la acepta. El contrato inteligente es el encargado de tomar la dirección del usuario que desea realizar el depósito, verificar que el valor es positivo y actualizar el monto total asociado a su cuenta correctamente. En la Figura 18 se presenta el flujo de información de este caso.



Figura 18: Flujo de información del ejemplo propuesto.

Este contrato inteligente se puede definir haciendo uso de un lenguaje de programación. En la Figura 19 aparece una posible especificación del contrato inteligente aplicado en el ejemplo, y programado en *Solidity*, que es el lenguaje usado para la definición de contratos inteligentes en *Ethereum*. Dicho contrato ejecutará la función `deposit()` en el ejemplo, que se encarga de verificar el saldo y actualizar su valor con la cantidad introducida.

```
1  pragma solidity >=0.4.22 <0.7.0;
2  contract EtherBank {
3      mapping (address => uint256) public balances;
4      function deposit() external payable {
5          require(balances[msg.sender] + msg.value >= balances[msg.
            sender]);
6          balances[msg.sender] += msg.value;
7      }
8      function withdraw(uint256 amount) external {
9          require(amount <= balances[msg.sender]);
10         balances[msg.sender] -= amount;
11         msg.sender.transfer(amount);
12     }
13 }
```

Figura 19: Ejemplo de contrato inteligente para *Ethereum* escrito en *Solidity*. Fuente: [33].

Un contrato inteligente puede adaptarse a cualquier caso de uso, al igual que puede hacerlo la informática. Por ejemplo, se pueden definir contratos inteligentes para el almacenamiento y mantenimiento de registros de una empresa, asegurar la cadena de suministro, gestionar operaciones de ingreso y retirada de dinero virtual (como se ha presentado en el ejemplo), autenticar certificados..., etc.

3.4. Tecnologías *blockchain* permissionadas disponibles

En esta sección se presentan las distintas tecnologías *blockchain* permissionadas disponibles en el mercado. Posteriormente, se abordará la viabilidad del uso de estas redes en el proyecto, así como la tecnología que finalmente será usada, mediante un análisis funcional de las tecnologías propuestas a continuación.

3.4.1. *Ethereum*

Ethereum [51] es la implementación *blockchain* más popular actualmente en el mercado, siendo esta lanzada en 2015, por el joven programador ruso-canadiense Vitalik Buterin, que sólo tenía 17 años por entonces. Fue creada con el fin de sacar mayor partido a la tecnología subyacente introducida en Bitcoin: las redes de *blockchains*.

Sirve a su vez como soporte para la criptomoneda homónima *ETH*, que es la segunda criptomoneda más importante por volumen de mercado, justo después de *Bitcoin*. Como *Bitcoin*, es un servicio descentralizado, pero este cuenta con acceso abierto tanto a consumidores como a productores. Se puede entender, por tanto, como una evolución de esta, ya que introduce todos los conceptos básicos de redes de cadenas de bloques, junto con la posibilidad de replicar y ejecutar código en los nodos de la red.

Estas piezas de código, escritas en el lenguaje de programación *Solidity*, son los contratos inteligentes que anteriormente se introdujeron, y son ejecutados en cada nodo bajo un sistema operativo ligero denominado *Ethereum Virtual Machine* [17]. Originalmente la red *Ethereum* estaba pensada como una red pública, pero recientemente ha logrado soporte para su implantación en redes permissionadas.

Los usuarios de la red pueden valerse de un cliente Ethereum para participar en esta, pudiendo explorar la *blockchain*, crear nuevas transacciones en la red, ejecutar contratos inteligentes... Las principales diferencias con *Bitcoin* radica en la mejora en el tiempo de procesamiento de las transacciones, y en la tamaño máximo de información que puede alojar cada bloque.



Figura 20: Logo de *Ethereum*.

3.4.2. Corda



Figura 21: Logo de Corda.

Corda [12] es una red de *blockchain* permissionada creada en 2019 por la empresa R3. Para poder usar una red basada en Corda, los usuarios de la red deben estar correctamente identificados y validados en esta. A diferencia de las tecnologías como *Bitcoin* o *Ethereum*, no posee soporte para ninguna criptomoneda, ya que su uso es puramente empresarial.

Esta red fue creada inicialmente para construir una red empresarial donde sus integrantes pudiesen comunicarse fácilmente a través de contratos inteligentes, poniendo un enfoque más restrictivo en la privacidad de las transacciones que el resto de sus competidoras. Estos contratos inteligentes están programados en lenguajes más habituales como *Java* o *Kotlin*, ya que se ejecutan en una *Java Virtual Machine*.

3.4.3. Hyperledger Fabric

Hyperledger Fabric [55] es un proyecto de *The Linux Foundation* que comenzó a desarrollarse a finales de 2015. Se basa en una red privada propia creada para brindar apoyo a empresas de una o múltiples organizaciones, y con opciones interesantes de privacidad. La

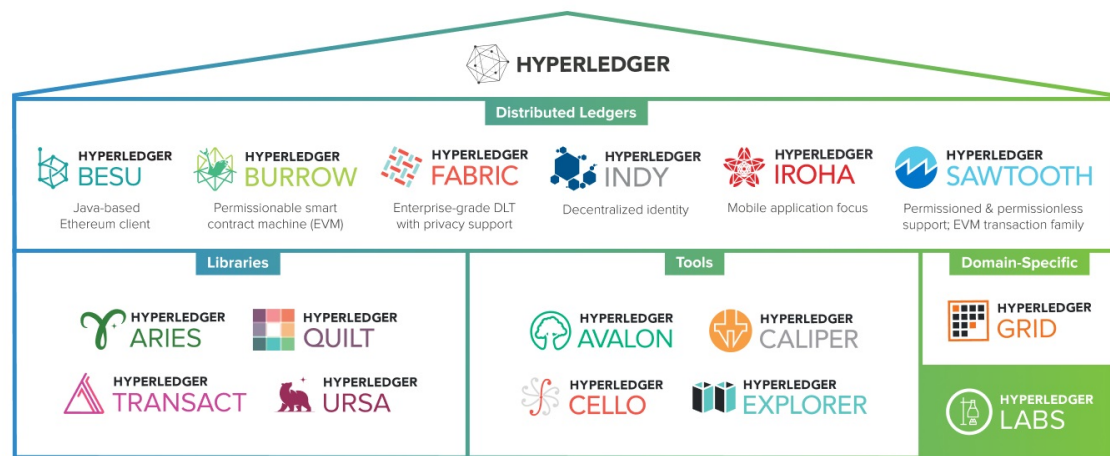
red mantiene certificados digitales tanto de las organizaciones como de los usuarios que tienen permiso para escribir transacciones, visualizar el contenido... Todo ello gracias a un contrato inteligente, conocido en *Hyperledger Fabric* como **chaincode**, escrito en lenguajes de propósito general como *Go*, *Node.js*, *Java*...

Hyperledger engloba multitud de aplicaciones y soluciones para el entorno de redes de cadenas de bloques, que se pueden ver representadas en la Figura 23.



Figura 22: Logo de Hyperledger.

Companies that are part of Hyperledger Greenhouse



 | cointelegraph.com

source: *Linux Foundation*

Figura 23: Aplicaciones y soluciones informáticas pertenecientes al ecosistema de *Hyperledger*. Fuente: [55].

Establece distinción entre redes de *blockchain*, librerías y herramientas, todas ellas integradas en el ecosistema. Estas son las soluciones *blockchain* que propone:

- **Hyperledger BESU:** se trata de un cliente *Ethereum* diseñado para manejar redes de *blockchain* basadas en esta tecnología en entornos empresariales, tanto para redes públicas como privadas.
- **Hyperledger BURROW:** se trata de una red de *blockchain* permissionada basada en *Ethereum*.
- **Hyperledger FABRIC:** se trata de una red de *blockchain* permissionada orientada al entorno empresarial, que asegura las transacciones.
- **Hyperledger INDY:** se trata de una red de *blockchain* que permite administrar identidades digitales dentro de otra cadena de bloques.
- **Hyperledger IROHA:** se trata de una red de *blockchain* orientada a poder utilizarse en múltiples plataformas, como son los dispositivos móviles.

- **Hyperledger SAWTOOTH:** se trata de una red de *blockchain* pública o privada basada en *Ethereum*, con la diferencia de que permite una configuración más exhaustiva de la *blockchain* diseñada. Puede trabajar conjuntamente con *Hyperledger BURROW*.

Para el caso práctico, es de mayor utilidad seleccionar *Hyperledger Fabric* entre todas las soluciones del ecosistema. Al tratarse de una de las más utilizadas, cuenta con una extensa documentación (que puede ser consultada a través de este [enlace](#), de donde también se puede extraer toda la especificación de la red) con multitud de casos de uso y ejemplos, así como un gran soporte entre desarrolladores de esta tecnología.

3.4.4. Otras tecnologías

En el mercado, y cada vez más frecuentemente, van apareciendo nuevas tecnologías y herramientas para el desarrollo de redes de *blockchain* privadas tanto para el ámbito empresarial como para ámbito financiero, personal..., etc. y cada vez se está extendiendo a más casos de uso. Algunas de estas tecnologías menos conocidas, o que se encuentran en fase actual de desarrollo, son las siguientes [44]:

- **Quorum:** *framework* de desarrollo *blockchain* basado en *Ethereum*, diseñado principalmente para la industria financiera, por el gigante financiero JP Morgan. Soporta tanto redes privadas como públicas, y la ejecución de contratos inteligentes, escritos en *Solidity*.
- **Multichain:** se trata de una de las pocas redes disponibles basadas en *Bitcoin*. Añade funcionalidades interesantes a la red como los permisos para el acceso a esta, privacidad, elegir tamaño de bloque... La versión 2.0 incluye el soporte para ejecutar contratos inteligentes, siendo uno de los más versátiles, al incluir *Python*, *Ruby*, o *C++* entre los lenguajes de programación disponibles.

3.5. Viabilidad en la prueba de concepto

Una vez presentadas los *frameworks* disponibles para el desarrollo empresarial de una red *blockchain*, se procede a analizar si estas pueden o no ser adecuadas en el escenario que se trata en este TFG.

Para ello, es necesario establecer unos requisitos preliminares a gran escala que se pretenden cumplir al implementar la prueba de concepto:

- **R1. Escalabilidad:** como se detalló en el estudio preliminar referido a los ecosistemas de estaciones de carga, en el Capítulo 2, la transición a un parque móvil plenamente eléctrico es cuestión de años, y todos los sistemas que se desarrollen en estos tiempos van a sufrir un aumento exponencial de usuarios próximamente. Por ello, el sistema que se pretende desarrollar debe contar con una alta escalabilidad, con tal de que, a medida que el total de usuarios vaya creciendo, la implementación no debe verse afectada ni en rendimiento ni en prestaciones.
- **R2. Privacidad:** el registro de los operarios que pueden pertenecer a la red, y tener permiso para consultar, añadir transacciones..., etc. debe darse con ciertas garantías de seguridad, pues se debe proteger la red contra terceras personas y usuarios sin autenticación que pretendan hacer uso de la misma.
- **R3. Seguridad:** toda la información recogida debe estar asegurada ante posibles ataques informáticos, cada vez más frecuentes en la sociedad y mucho más en servicios críticos, como es la energía.
- **R4. Latencia:** una latencia alta indica un gran retardo entre la información enviada y/o recibida dentro de una red. Para la prueba de concepto, no solo es necesaria una latencia decente, se busca ofrecer la menor posible, ya que esto mejora el funcionamiento y la integridad de la red.
- **R5. Interoperabilidad:** se busca una solución que permita conectar la información en varios ecosistemas, como es el ecosistema *Android* y el ecosistema Web. Además, si este ecosistema se expandiera, debería adaptarse a nuevas formas de tecnología sin perder su funcionalidad.
- **R6. Adopción por la comunidad:** a la hora de desarrollar una solución software ante un problema, se debe tener en cuenta la adopción de una tecnología por parte de la comunidad mundial de desarrolladores. Al tratar con opciones de código abierto, siempre se prefiere usar una tecnología ampliamente documentada y desarrollada, a una que

apenas tiene uso y cuyos problemas posteriores pueden ocasionar que termine su soporte.

Habiendo presentado los requisitos a gran escala que se buscan para realizar este TFG, casi todos son ampliamente tratados en las redes de *blockchain*. Como alternativa a esta tecnología se encuentran las bases de datos convencionales.

Las bases de datos sí que garantizan una escalabilidad exponencial (R1) como lo hacen las redes de cadenas de bloques, pudiendo ser esta vertical u horizontal (mejorando las prestaciones del servidor o añadiendo nuevos servidores, respectivamente), pero las *blockchain* ganan en cuanto a descentralización y escalabilidad horizontal. Otro de los puntos fuertes de las redes de *blockchain* es la seguridad y la privacidad (R2 y R3), pues los datos introducidos son inmutables y cada usuario puede usar la red mediante un sistema seguro de criptografía clave público-privada. La latencia (R4) es un punto fuerte de las bases de datos, pues bien es sabido que el costo de las transacciones en una red de *blockchain* es extremadamente elevado, aunque cada vez se consiguen sistemas que logran retardos muy inferiores a los vistos en esta tecnología de manera primigenia.

En resumen, por la **inmutabilidad** que ofrecen las redes de *blockchain*, por la necesidad de implementar una aplicación **descentralizada**, junto con las necesidades de mantener un sistema **trazable** que sea fácilmente **supervisado**, se opta por usar esta tecnología de red de cadena de bloques para la realización de la prueba de concepto enmarcada en este proyecto.

Ahora es el turno de comparar las distintas propuestas de redes de *blockchain* privadas presentadas previamente. Para comparar entre todas las opciones posibles, es necesario establecer un sistema de evaluación que otorgará más o menos puntos a cada solución dependiendo del grado de cumplimiento de cada requisito a grandes rasgos, usando para ello los siguientes atributos [44]:

- **Adopción (R6):** se obtiene un valor en base a las contribuciones realizadas en el proyecto *GitHub* de cada *framework*, el número de *commits*, así como los seguidores en redes sociales como *Twitter* y los *tweets* publicados.
- **Privacidad y seguridad (R2-R3):** se obtiene un valor en base a las características de privacidad y seguridad ofertadas por cada *framework*, y los mecanismos usados para ello.

- **Escalabilidad y latencia (R1-R4):** esta información se obtiene analizando los resultados extraídos de numerosos desarrolladores que han testeado y probado cada solución.

Una vez establecido el sistema de puntuaciones y calculado la puntuación para cada *framework* de desarrollo, los resultados se resumen en el Cuadro 3.

<i>Blockchain</i>	<i>Adopción</i>	<i>Privacidad</i>	<i>Seguridad</i>	<i>Escalabilidad</i>	<i>Latencia</i>	<i>Total</i>
<i>Ethereum</i>	5	0	5	5	2	17/25
<i>Corda</i>	2	3	5	1	4	15/25
<i>Fabric</i>	3	5	5	4	5	22/25

Cuadro 3: Resultados de la evaluación según los criterios definidos y evaluados. Fuente: [44].

Y que se pueden representar de una forma más gráfica en la Figura 24.

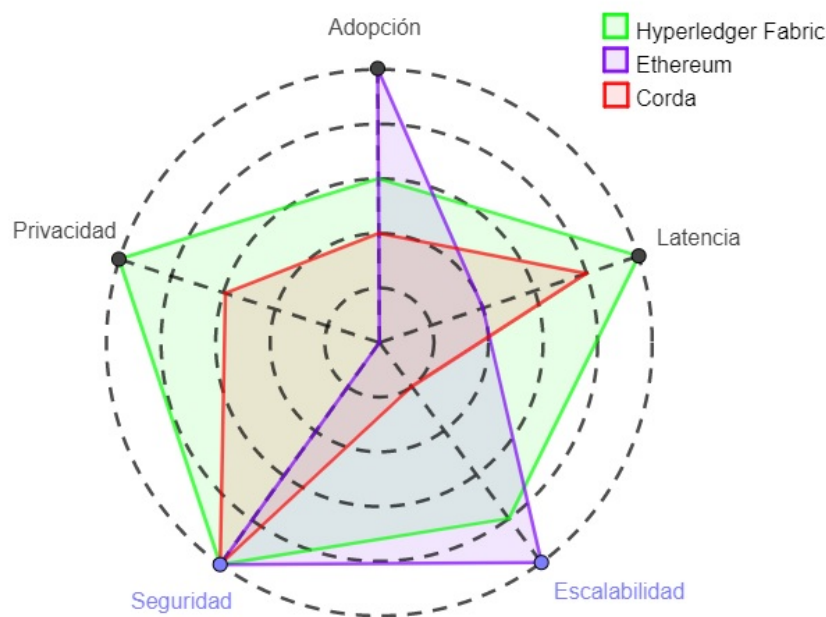


Figura 24: Comparación gráfica resultado de la evaluación de las tecnologías *blockchain* propuestas.

Hyperledger Fabric cuenta con las mejores prestaciones en base a los requisitos que se pretenden alcanzar en el proyecto, y es la tecnología que se usará para implementar la prueba de concepto. El próximo capítulo tratará sobre dicha prueba, donde se presentará con más detalle el funcionamiento de este *framework*.

Prueba de concepto

Tras presentar los dos estudios previos, se ha logrado comprender tanto el ecosistema de los sistemas de carga como la tecnología en la que se basa las redes *blockchain*, específicamente, las redes *blockchain* permissionadas, que son las que pondremos en práctica.

En esta prueba de concepto se incluye el despliegue y el desarrollo de tres tecnologías:

1. Red *blockchain*: se desplegará una red *blockchain* permissionada basada en *Hyperledger Fabric* que dará soporte para poder enviar y consultar los reportes que los operarios de campo y centrales envíen.
2. Aplicación móvil: se implementará una aplicación móvil que será usada tanto por los operarios de campo como los operarios centrales para enviar y listar los reportes que creen durante su trabajo, siendo estos procesados y almacenados en la red de *blockchain*.
3. Aplicación web: se implementará una aplicación web local administrador que servirá de centro de control de todos los reportes enviados, así como de administración para funcionalidades de la aplicación.

4.1. Introducción

Retomando el escenario planteado para esta prueba de concepto, que ya se presentó en la Figura 2, el objetivo de este es el de integrar dos aplicaciones en una red *blockchain*, y cada una contará con distintos requisitos y funcionalidades.

Uno de los grandes retos de esta prueba de concepto es el de interconectar la información y el de crear una red segura e inmutable, fácilmente manejable y gestionable por los distintos actores.

Existe un total de tres actores distintos que acceden a alguna parte del sistema:

1. **Los operarios centrales:** encargados de administrar múltiples estaciones de carga desde la CSMS gracias al protocolo de comunicación OCPP. Los operarios centrales pueden iniciar transacciones remotas, verificar el estado de las estaciones y conectores conectados, modificar variables..., etc. Estos operarios usan la aplicación móvil diseñada para enviar los reportes con los cambios realizados.
2. **Los operarios de campo:** encargados de la reparación y la administración de estaciones de carga *in situ*. Los operarios de campo pueden instalar nuevos conectores, verificar el correcto funcionamiento de estos, configurar el *smart meter*..., etc. Estos operarios también hacen uso de la misma aplicación móvil.
3. **Los administradores de la aplicación:** encargados de procesar los reportes enviados por el resto de los actores, así como administrar la información de la aplicación, usuarios disponibles y registrados en la red..., etc. Estos hacen uso de la aplicación web, que se conecta también a la red para sus propósitos.

Cuando cualquier integrante de alguna de las dos clases de operarios necesita enviar un informe o reporte a la red, accede a la aplicación *Android* creada para tal efecto, a través de un usuario y contraseña creados previamente por el administrador. Este usuario y contraseña garantiza que el operario que use la aplicación se encuentra autorizado en la red. Una vez que el operario ha iniciado sesión correctamente, accede a todo el contenido de la aplicación, que incluye una página principal de información que permite el acceso tanto a la vista de creación de reportes como al listado de reportes previamente enviados por dicho usuario.

En la vista de creación de reportes, el operario se encuentra con ciertos campos que puede rellenar (según los casos de uso recogidos en los apéndices **C** y **D** de esta memoria), así como la posibilidad de incluir la localización de una estación de carga concreta (mantenida en una base de datos no relacional), o la propia localización del operario, opción usada principalmente por los operarios de campo. Una vez rellenado el reporte, este se confirma y se envía a la red, pudiendo luego ser visualizado desde la vista de listado de reportes en la misma aplicación.

Por otra parte, el administrador hace uso de la aplicación web para acceder a un listado de reportes enviados por todos los usuarios de la red, así como la información adjuntada en cada uno. Además, puede configurar el acceso a la aplicación a los usuarios, visualizar en un mapa

interactivo la localización de los reportes enviados, y administrar las estaciones de carga fijas proporcionadas en la aplicación.

En la Figura 25 se recogen los distintos casos de uso que se han presentado y que deben tratarse en el sistema.

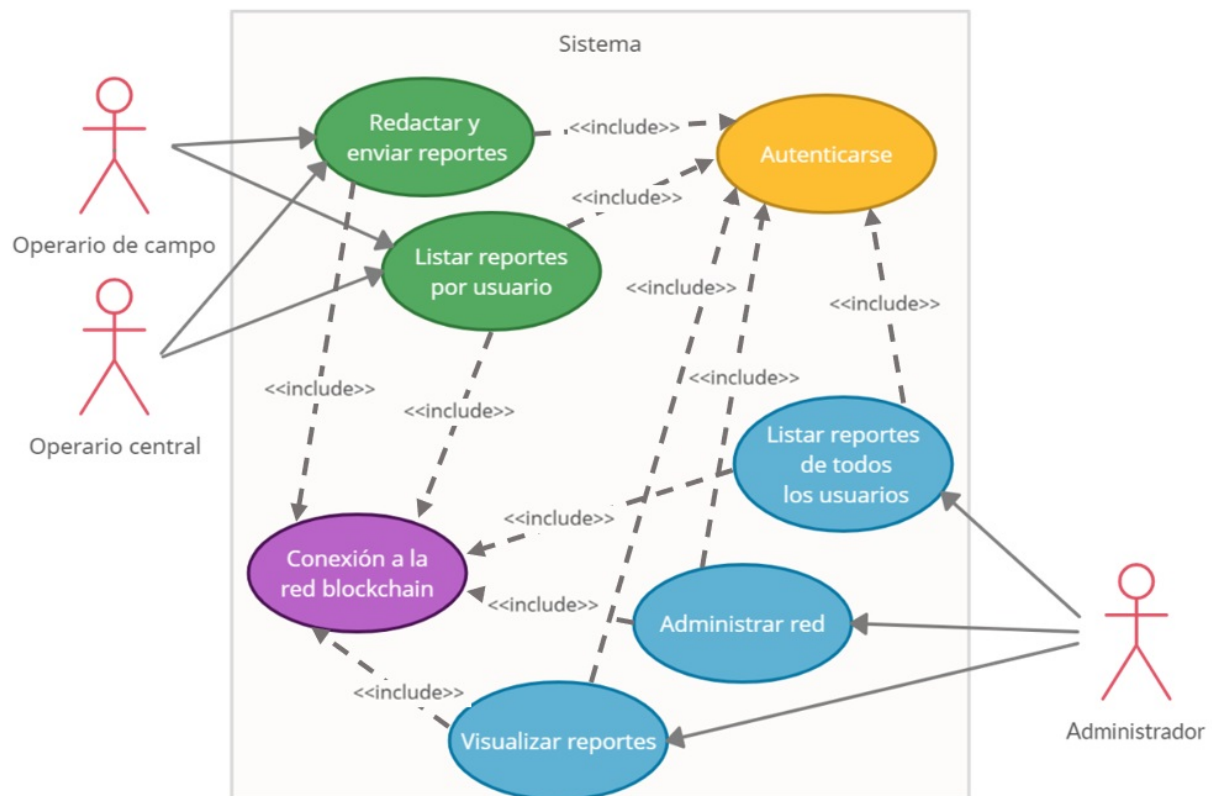


Figura 25: Casos de uso del sistema propuesto.

Se puede observar que existen tres tipos de casos de uso distintos, correspondiendo a su extensión a un único tipo de tecnología implementada. Los casos de uso verdes corresponden a casos de uso dentro de la aplicación web, los azules, a la aplicación web, y el morado, a la *blockchain*, que hemos resumido en la conexión a esta.

Para más información, el anexo E recoge la documentación de todos los casos de uso especificados en la Figura 25.

Una vez presentada la lógica principal de la prueba de concepto, se enumeran a continuación las tecnologías usadas e integradas en el sistema.

4.2. Tecnologías utilizadas

Durante la realización de la prueba de concepto se ha entrado en contacto con multitud de tecnologías. Como se observó en el diagrama de casos de la uso de la Figura 25, la prueba de concepto necesita de tres grupos de tecnologías bien diferenciados, y que se enumeran de esta forma:

1. **Tecnologías blockchain:** aquellas tecnologías que se han usado para el despliegue y configuración tanto de la red *blockchain* como del contrato inteligente al que se suscribe.
2. **Tecnologías Android:** aquellas tecnologías que se han usado para el desarrollo de una aplicación funcional en el sistema operativo *Android*, para dispositivos móviles.
3. **Tecnologías Web:** aquellas tecnologías que se han usado para el despliegue, implementación y verificación de la aplicación administrador web.

En los apartados siguientes se abordará más en detalle qué son y cómo hemos usado las distintas tecnologías propuestas.

4.2.1. Tecnologías Blockchain

Como se pudo comprobar en el estudio preliminar del Capítulo 3, donde se sentaron las bases de la tecnología *blockchain*, se escogió *Hyperledger Fabric* como la herramienta que se usará para desplegar la red *blockchain* que servirá para esta prueba de concepto.

Es por ello que en este primer apartado se explicará cómo funciona *Hyperledger Fabric* por dentro, como se ha adaptado y configurado para su uso en el proyecto y cómo se puede implementar el contrato inteligente con la lógica de negocio del sistema (este será explicado y visto con mayor detalle en siguientes apartados, este paso se centra en la tecnología usada).

■ Hyperledger Fabric

Este framework de desarrollo de redes *blockchain* creado por *The Linux Foundation* a finales de 2015 se usa principalmente para la creación y el despliegue de redes *blockchain* en un entorno empresarial con tres características principales [55]:

- Se trata de una **red permissionada**, por lo que sólo aquellas personas que la organización permita pueden acceder a todos los contenidos de la red.
- Las transacciones son privadas y no son accesibles desde ningún punto, otorgando **confidencialidad** a la red.
- La lógica de negocio se implementa mediante **chaincodes**, que corresponden a los clásicos *smart contracts*. Estos *chaincodes* son programables en lenguajes como *Node.js*, *Go* o *Java*.

Al tratarse de una red privada, todos los usuarios deben contar con un certificado otorgado por una autoridad certificadora de confianza para hacer uso de la red. En *Hyperledger Fabric*, además de los usuarios, todos los nodos existentes cuentan con un certificado de autenticación, otorgado a través de un servicio de **MSP**.

La estructura de la red que *Hyperledger Fabric* propone pasa por una estructura modular de nodos interconectados y autorizados en la red. Todos estos nodos comparten el registro de la red, donde se almacenan todas las transacciones y estados actuales de los *assets* (activos que se buscan representar. En este caso, reportes).

Cada vez que se realiza una nueva transacción en la red, esta debe encargarse de mantener el registro actualizado en todos los nodos de la red. Para ello se sigue lo que se conoce como **protocolo de consenso**. Este asegura que únicamente las transacciones válidas se añaden al registro, y que estas aparecerán en el mismo orden en el que han sido confirmadas en todos los nodos.

La red, a su vez, se puede dividir en organizaciones, donde cada una mantiene la certificación de sus propios nodos. En este caso, hablaremos de una única organización. Además, los nodos se agrupan en torno a un **canal**, que es una subestructura que garantiza la privacidad dentro de ella. Así, podemos tener una red con distintos canales, a los que se conectan distintos usuarios de distintos departamentos sin conexión directa entre ellos y, por tanto, sus registros.

Una vez presentadas las características de la red, es necesario conocer cómo se estructura la red [6]. Como ya se presentó previamente, *Hyperledger Fabric* cuenta con una serie de nodos pertenecientes a una organización y autorizados a participar en la red por esta, mediante la certificación por un MSP.

Existen un total de tres nodos distintos en la red:

1. **Nodos *orderer***: se encargan de mantener los registros de la red actualizados en todo momento y de manera correcta, aplicando el protocolo de consenso especificado. Se puede considerar el nodo principal de la red pues, aunque no desempeña una función activa, es el encargado de mantener la consistencia de esta en todo momento. El nodo *orderer*, al recibir una transacción por parte de otro nodo, crea un bloque, lo inserta en la red, y lo comunica al resto de nodos. Se pueden aplicar tres tipos distintos de protocolos de consenso:

- a) *Solo*: protocolo de consenso que solo cuenta con un único nodo *orderer* en la red, encargado de aplicar y mantener toda la consistencia de la red. Es un protocolo que solo debe usarse en redes de prueba, pues no es tolerante a fallos.
- b) *Kafka*: mediante este protocolo de consenso se forma un cluster de nodos *orderer*. Es un escenario adecuado para redes de producción, pues se trata de un protocolo tolerante a fallos que aporta concurrencia y estabilidad a la red. Sin embargo, al tratarse de un protocolo mantenido por una empresa externa (*Apache*) puede no adaptarse completamente al funcionamiento de la red.
- c) *Raft*: también se trata de un protocolo basado en un cluster de nodos *orderer*, pero con la salvedad de que se trata de una implementación más sencilla que el protocolo *Kafka*. Este protocolo selecciona dinámicamente a un líder dentro del cluster de nodos disponibles, y este es el encargado de aplicar el protocolo. Si este nodo fallase, el control pasaría a cualquier otro nodo activo del cluster.

Raft es el protocolo recomendado por *Hyperledger*, y será el único que se mantendrá en el futuro, dejando sin soporte a *Kafka* y sustituyendo el protocolo *Solo* por una implementación de *Raft* con un único nodo.

2. **Nodos *peer***: son los encargados de mantener los datos del registro actualizados, y de ejecutar el *chaincode* en cada transacción. Existen cuatro tipos distintos de nodos *peer*:

- a) *Anchor peers*: nodos principales y accesibles por elementos fuera de la organización. Es necesario contar con al menos un *peer* de este tipo.
- b) *Leader peers*: reciben los bloques ordenados por el nodo *orderer* y los transmiten al resto de nodos. Existe un nodo de este tipo por cada canal.

c) *Regular peers*: no realizan ninguna función especial aparte de mantener los datos del registro actualizados.

d) *Endorsing peers*: reciben y ejecutan transacciones antes de ser enviadas al nodo *orderer*, pero sin tener efecto en el registro. Se usan para evitar errores de funcionamiento o ataques a la red.

3. **Nodos cliente**: son aquellos nodos que quieren comunicarse con la red para enviar transacciones. En este caso corresponden con la aplicación *Android* y la aplicación Web. Los clientes mandan la transacción inicialmente al *endorsing peer*, que la verifica y responde al cliente con un mensaje de confirmación. Finalmente, este ya envía la transacción al *orderer*, que se encarga de crear un bloque para ella y actualizar el resto de *peers*.

En la Figura 26 se pueden comprobar como interactúan todos los nodos previamente introducidos. Para ello, se ha considerado que existen dos organizaciones en el sistema, que cuentan con una serie de *peers* correctamente certificados, y un nodo *orderer*, común a todas las organizaciones (en este caso, se supondrá que sigue un protocolo de consenso *Solo*). El cliente puede tratarse de cualquier aplicación que desea enviar alguna transacción a la red. Por simplicidad, cada nodo estará suscrito a un mismo canal, y el *chaincode* se encuentra también desplegado en este para todos los nodos que lo conforman.

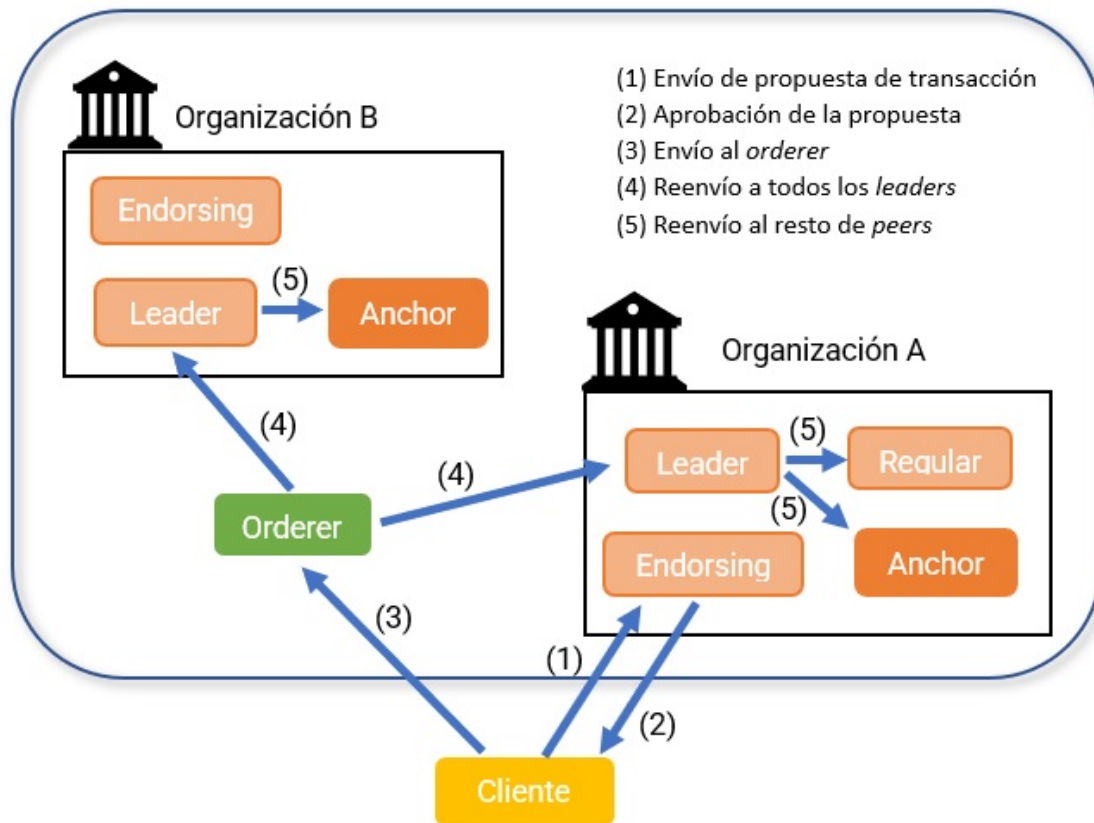


Figura 26: Esquema resumen de la comunicación entre nodos a la hora de enviar una transacción por parte de un cliente. Elaboración propia a partir de [55].

Para lanzar una red basada en esta tecnología, además de tener que desplegar los distintos nodos necesarios sobre un servidor, es conveniente contar con una **API REST** para poder realizar consultas más fácilmente. Esta API se encargará de tomar el rol de nodo cliente, y realizar consultas de diversa índole a la red.

En resumen, se puede lanzar una red *Hyperledger Fabric* siguiendo los siguientes pasos [1]:

1. **Preparar el entorno.** Se debe crear, como mínimo, un nodo *orderer* y un nodo *peer* dentro de una organización, y estos deben estar suscritos a un canal. Estos deberían estar correctamente certificados por una autoridad certificadora, haciendo uso de **certificados X.589**.
2. **Preparar la lógica de negocio.** Se debe programar el contrato inteligente, el *chaincode*, y desplegarlo sobre el canal.

3. **Preparar API REST.** Se debe crear un *endpoint* al que acceder a la red, o *backend*, a través de una aplicación, o *frontend*, que realice consultas a dicha API.

En la introducción de la siguiente tecnología se explicará como hemos desplegado y configurado los nodos de la red, así como la API REST a la que consultaremos desde las aplicaciones creadas.

■ Kaleido

Kaleido es un sitio web que ofrece servicios SaaS, específicamente BaaS (Blockchain as a Service) [30]. Busca simplificar el despliegue de redes de producción privadas basadas en *blockchain* usando para ello tecnologías como Ethereum, Corda o Hyperledger Fabric. Provee además mecanismos de creación de certificados para los distintos nodos de la red y una API REST con credenciales de acceso para que las aplicaciones creadas se conecten y envíen consultas a la red. Desde el propio servicio no se incluye la creación rápida de *chaincodes*, por lo que debemos diseñarlo, programarlo y compilarlo por autónomamente.



Figura 27: Logo de *Kaleido*.

Kaleido se puede entender como un servicio de procesamiento en la nube, o *cloud computing*, al estilo de [AWS](#) o [Microsoft Azure](#).

Para el despliegue de la red se usará la versión de prueba de la aplicación. Esta permite tener desplegado un servidor con dos nodos: un nodo *orderer*, que seguirá un algoritmo de consenso *Raft* y un *peer*, que se encargará de recibir consultas a través de la API REST incluida con la aplicación y de ejecutar el *chaincode* con cada transacción. Ambos nodos están suscritos al mismo canal por defecto. Además, el servicio incluye una autoridad certificadora encargada de proveer el acceso al resto de nodos.

En la Figura 28 se puede observar un resumen de la red desplegada.

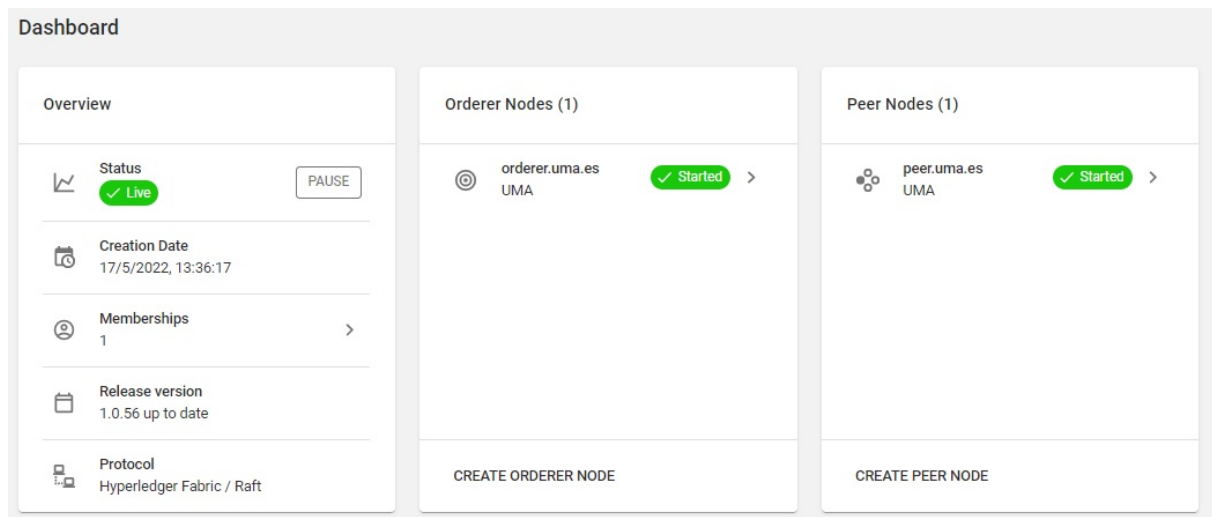


Figura 28: Dashboard de la red *blockchain* creada para la prueba de concepto usando Kaleido.

Además de la API REST, el nodo *peer* cuenta con uno de los servicios que se presentaron cuando se introdujo el ecosistema de *Hyperledger*, *Hyperledger Explorer*, el cual permite conocer los *chaincode* instalados, los bloques y transacciones creadas y añadidas a la red, y más información estadística.

En la Figura 29 imagen aparece la pantalla de inicio de dicha herramienta para la red desplegada.

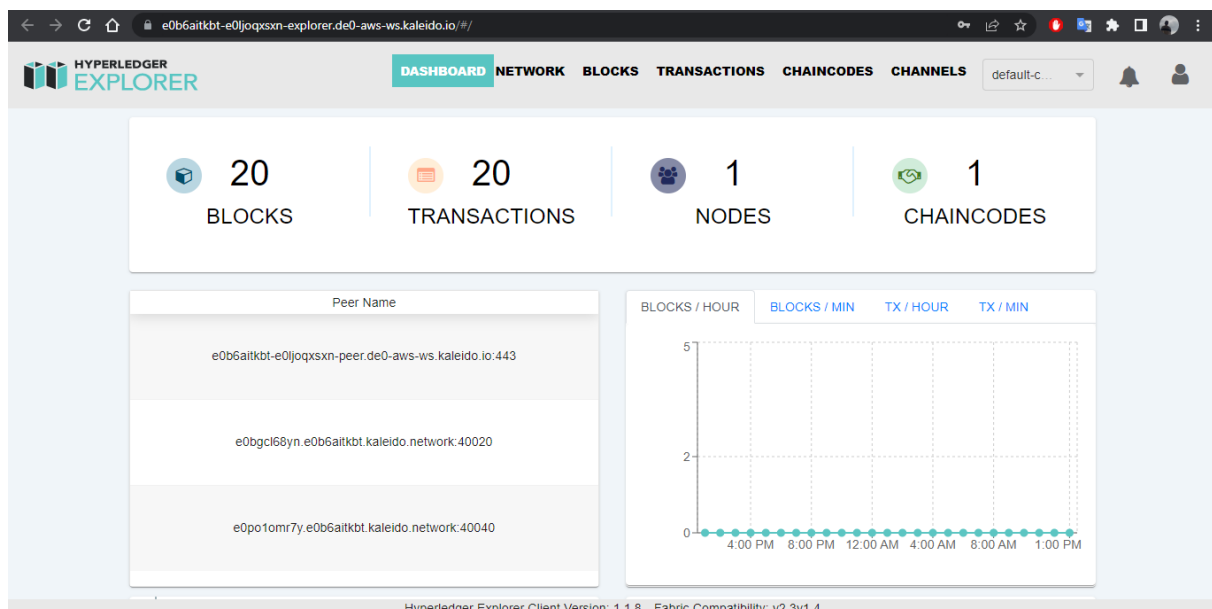


Figura 29: Dashboard de la herramienta *Hyperledger Explorer* de la red desplegada.

Para mayor información sobre cómo desplegar la red y configurar el acceso a esta para los usuarios, se remite al lector al Apéndice B, donde se detalla la información para instalar y usar las distintas aplicaciones.

■ Go



Figura 30: Logo de Go

Go es un lenguaje de programación compilado creado y soportado por Google en 2007, siendo este lanzado al mercado a finales de 2009. Se trata de un lenguaje con una sintaxis bastante fácil de entender y usar, pues se basa en Python y C++ (además de otros como Pascal u Oberon) [5].

Al compilarse, Go se transforma en un archivo de código binario, que puede ser usado para su ejecución en cualquier sistema operativo, además de otras ventajas como su rápido procesamiento y compilación. La documentación del lenguaje puede consultarse en este [enlace](#).

Se puede observar en la Figura 31 un ejemplo de un pequeño programa escrito en lenguaje Go, el cual imprime “Hola Mundo”.

```
package main

import "fmt"

func main() {
    fmt.Println("Hola, mundo.")
}
```

Figura 31: Ejemplo de programa básico en Go.

Su versatilidad, su capacidad de usarse en cualquier plataforma, y su novedad, al tratarse de un lenguaje de programación muy joven, han sido algunas de las razones por las que se ha escogido para implementar el contrato inteligente que se ejecutará en la red *blockchain*. Solo se necesita de las librerías básicas de Go para poder compilarlo y desplegarlo en dicha plataforma. El contenido del contrato se detalla en los siguientes apartados.

4.2.2. Tecnologías *Android*

Una vez presentada la tecnología *blockchain* que se usará, es necesario conocer las tecnologías que se han necesitado para la creación de los clientes que se conectarán a la red. En primer lugar, se presentarán las tecnologías usadas para el desarrollo e implementación de la aplicación *Android*.

■ **Android**

Android es, actualmente, el sistema operativo para plataformas móviles más usado del mundo. Es un sistema basado en Linux, y funciona en dispositivos táctiles como teléfonos móviles, tabletas y otros dispositivos inteligentes [2].

Fue desarrollado inicialmente por la empresa *Android Inc.*, en 2003, compitiendo directamente con los sistemas operativos *Windows Mobile* y *Symbian*, y mucho antes de la

aparición del *iPhone* de *Apple* junto con *iOS*, su sistema operativo, y principal rival de *Android*.

En 2005 fue vendida a *Google Inc.*, que continuó activamente su desarrollo, hasta estos días. El primer teléfono móvil basado en *Android*, HTC Dream G1, se lanzó en octubre de 2008, en Estados Unidos. Desde entonces, multitud de dispositivos han adoptado y/o adaptado este sistema operativo en sus terminales.

En la Figura 33 se observa la cuota de mercado que ocupa cada sistema operativo móvil en el mercado mundial.



Figura 32: Logo de *Android*.

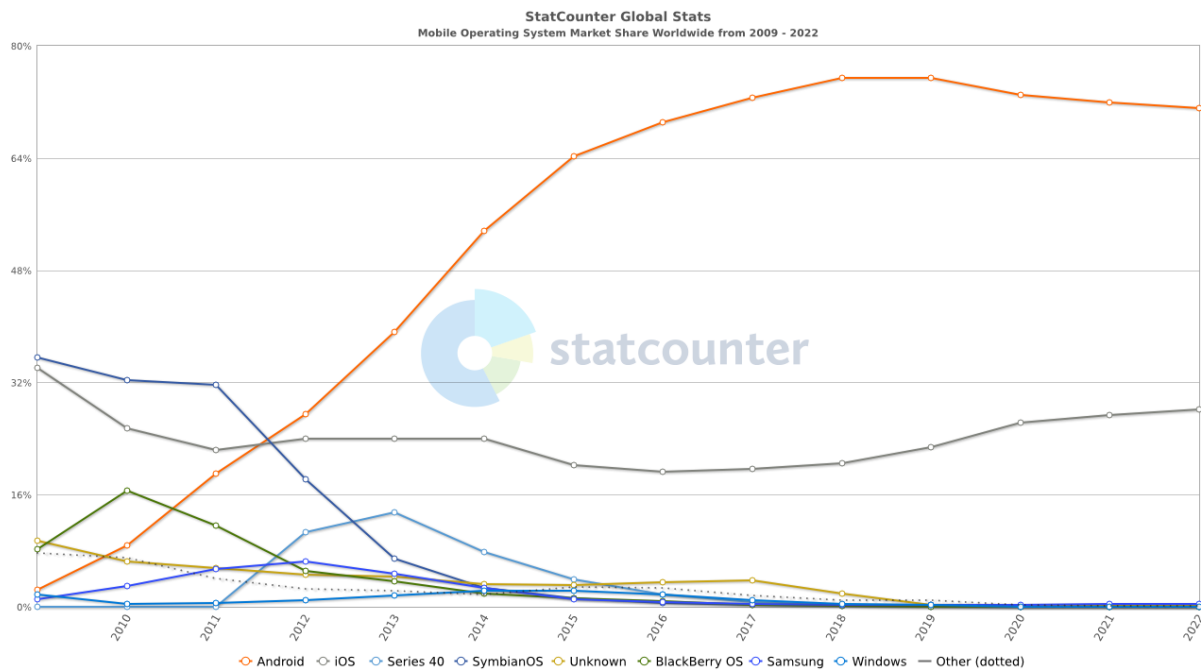


Figura 33: Evolución de la cuota de mercado de los sistemas operativos móviles desde 2009.
Fuente: [50].

Con el paso de los años, interesantes funcionalidades han sido añadidas a los teléfonos móviles gracias a *Android* y a nuevos componentes hardware como el pago móvil, conexión *Bluetooth*, carga inalámbrica...

En el ecosistema *Android* existen aplicaciones, que pueden ser desarrolladas tanto en *Java* como en *Kotlin*, y que son las que el usuario ejecuta para realizar alguna actividad. Estas pueden ser del sistema (cámara, calendario, reloj, ajustes...), o creadas por otros desarrolladores (*Google Chrome*, *Twitter*, *Whatsapp*...). En esta prueba de concepto se desarrolla una de estas aplicaciones, que podrá ser instalada y usada en un dispositivo *Android*.

Es necesario conocer cómo funciona el ciclo de vida de las aplicaciones *Android*, conocimiento que será usado a la hora de desarrollar la aplicación [59].

Una aplicación *Android* está formada por elementos o ventanas que conocemos como **Actividades**, algo muy similar a las vistas que tenemos en una aplicación Web. Todo lo que se ejecuta en una aplicación debe formar parte de una Actividad. Las Actividades son ejecutadas a petición del usuario, mientras que los servicios son ejecutados por el propio sistema.

En su ciclo de vida, una Actividad puede pasar por los siguientes estados: *Running* (se encuentra ejecutándose y el usuario puede interactuar activamente con ella), *Paused* (se en-

cuenta ejecutándose y es parcialmente visible para el usuario, pues otra ha tomado parte de su lugar en pantalla), *Stopped* (se encuentra parada pues otra ha tomado su lugar visible en la pantalla) o *Destroyed* (deja de estar disponible para el usuario, por decisión de este o por decisión del sistema).

Además de las Actividades, es necesario conocer otros componentes de *Android* que serán de utilidad. Se trata de los **permisos de la aplicación**. Estos permisos te dan la posibilidad de usar **servicios** del sistema, y algunos de ellos necesitan una confirmación fehaciente por parte del usuario para su uso, como el acceso al micrófono del dispositivo, a la localización del GPS..., etc.

Una vez presentada la tecnología y la arquitectura de las aplicaciones *Android*, es el turno de conocer la tecnología para poder desarrollar aplicaciones propias.

■ *Android Studio*



Figura 34: Logo de *Android Studio*.

Android Studio es una IDE gratuito y oficial mantenido por la comunidad de desarrolladores de *Android*, accesible en <https://developer.android.com/> y que es usado para la programación y el desarrollo de aplicaciones *Android*.

Su núcleo se basa en el IDE *IntelliJ Idea* de *JetBrains*, con la salvedad de que este es gratuito para todos los usuarios (usa una licencia Apache 2.0). Permite crear aplicaciones tanto en *Java*, *Kotlin* o incluso C++. En esta prueba de concepto se ha usado por el uso de *Java*, pues es un lenguaje de programación conocido previamente.

En la Figura 35 se puede observar el IDE con el proyecto de la aplicación abierto.

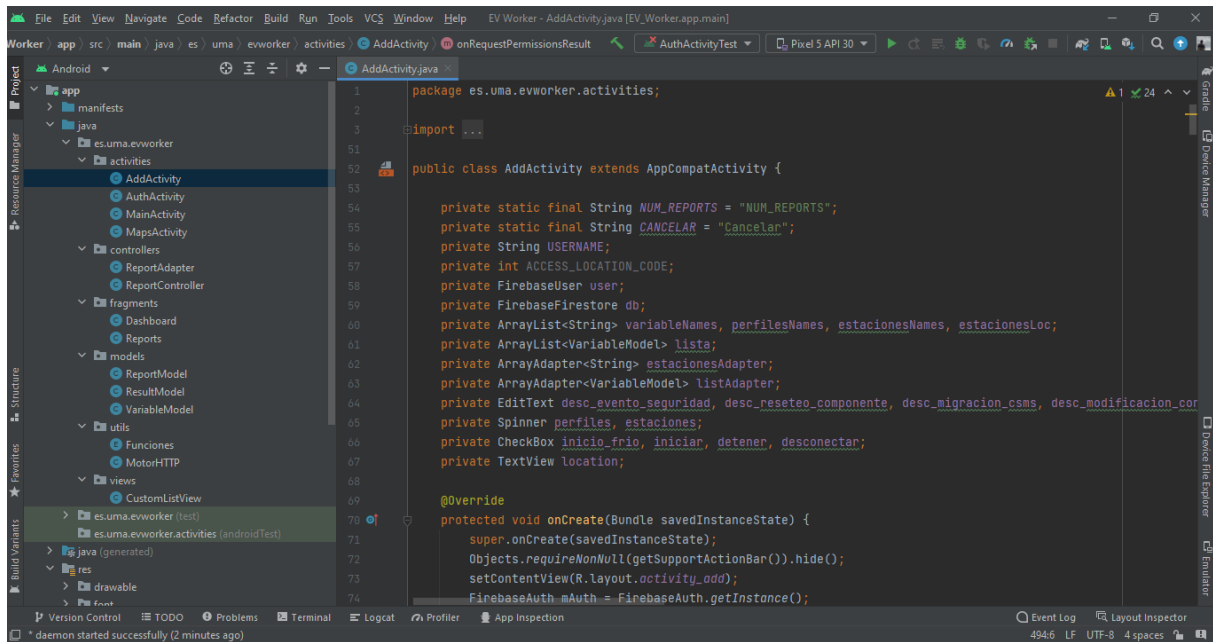


Figura 35: Captura de pantalla de la ventana de *Android Studio* abierta con el proyecto de la aplicación *Android*.

Las aplicaciones en *Android* están compuestas de numerosos componentes. Todos estos se compilan en un paquete conocido como APK, que es reconocido e instalado en el sistema operativo. Para el desarrollo de la aplicación se deben crear los siguientes elementos:

1. **Archivo *Manifest*:** contiene metainformación de la aplicación. En este archivo se declara la estructura de la aplicación, los permisos necesarios para su ejecución...
2. **Actividades:** las actividades se crean como clases *Java*, y extienden de la clase *AppCompatActivity*. Esta incluye los métodos *onCreate()*, *onStart()*, *onResume()*... que se ejecutan en el ciclo de vida de la actividad.
3. **Clases *Java*:** contienen todo tipo de información y procesamiento para llevar a cabo las funcionalidades de la aplicación.
4. **Recursos:** archivos generalmente de extensión XML, o lenguaje marcado extensible, que corresponden a recursos gráficos de la aplicación. Pueden ser:
 - *Layout*: modifican la disposición de los elementos en la pantalla, así como su integración en las Actividades. Cada Actividad se basa en un *Layout* para mostrarse por pantalla. Estos a su vez se organizan en *layouts* y componentes.

- *Drawable*: indican ficheros de imágenes, vectores, iconos...
- *Mipmap*: modifican la visualización del icono de la aplicación, con diferentes tamaños para adaptarse a los dispositivos que lo usen.
- *Values*: contienen ficheros de definición de colores, temas, cadenas de texto contenidas en la aplicación, traducciones...

La integración, el empaquetado y la resolución de dependencias se realiza mediante **Gradle** [24]. Este sistema permite automatizar la compilación, dejando al desarrollador únicamente la parte de programación. Para la compilación de una aplicación *Android* se siguen los siguientes pasos, resumidos visualmente en la Figura 36.

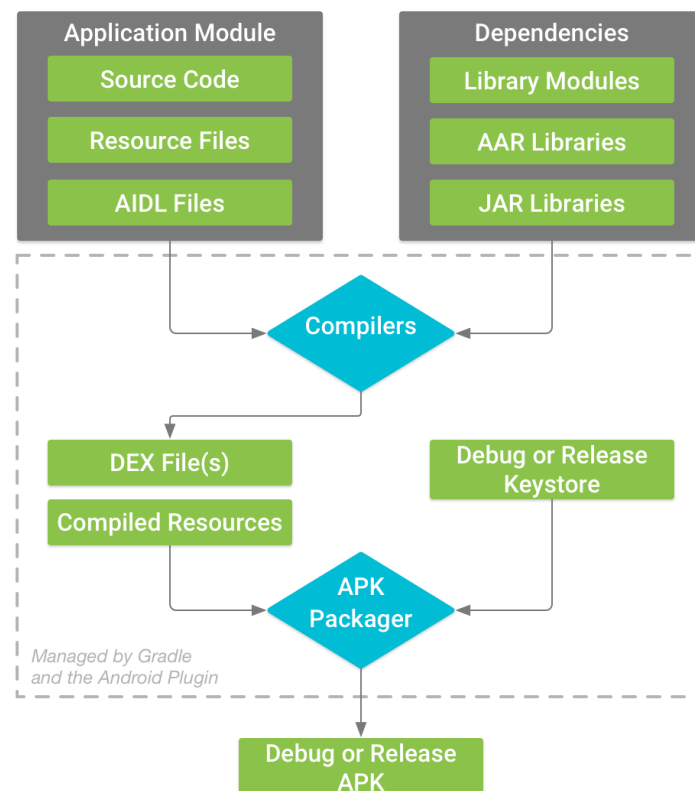


Figura 36: Proceso de compilación de una aplicación *Android* usando *Android Studio*.

Fuente: [3].

1. Los compiladores toman el código fuente en Java, los archivos de recursos y las librerías necesarias y lo compilan en un archivo DEX, que incluye código en bytes, ejecutable en *Android*.

2. El empaquetador de aplicaciones toma los archivos DEX con código compilado y los recursos compilados para generar un archivo APK, que podrá ser instalado en *Android*. Además, este empaquetador puede firmar la aplicación de dos formas distintas.

- a) Si se encuentra en una etapa de **depuración**, se firma como tal usando el almacén de claves de depuración.
- b) Si se encuentra en una etapa de **producción**, se firma usando tu almacén de claves de lanzamiento.

■ **Firestore**

Firestore es una herramienta web propiedad de Google pensada para el desarrollo e interoperabilidad de aplicaciones desarrolladas en plataformas diversas: *Android*, *iOS*, web, videojuegos...

En este caso, se integrará en una aplicación *Android* y una aplicación Web, que se encuentran conectadas como se observa en la Figura 38, que muestra una captura de la pantalla inicial de la herramienta.



Figura 37: Logo de *Firestore*.

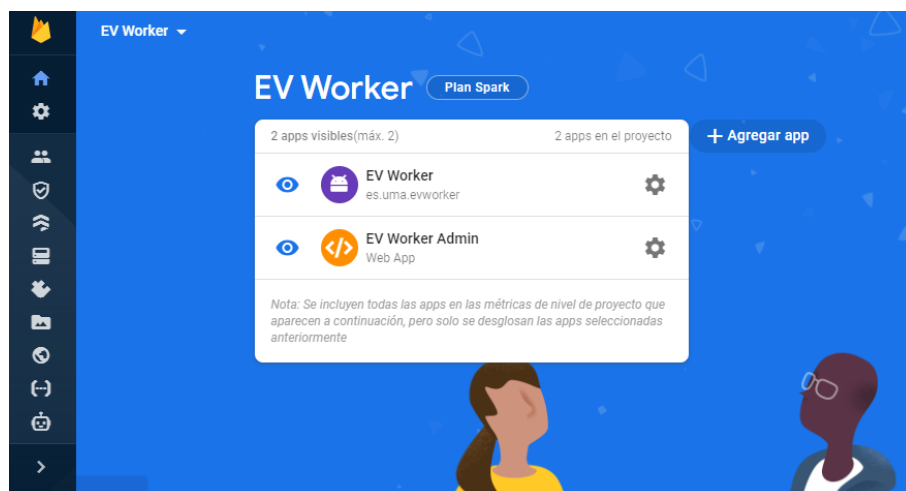


Figura 38: Página de inicio de *Firestore*, mostrando las aplicaciones actualmente conectadas al proyecto.

Comenzó ofreciendo únicamente una base de datos, pero ahora se centra en simplificar el proceso de creación de aplicaciones mediante distintas herramientas que integra [19]. Estas herramientas se dividen en tres grupos:

- **Herramientas de desarrollo:** contribuyen a un proceso de desarrollo más sencillo. Para ello, incluye herramientas como autenticación centralizada, bases de datos en tiempo real, almacenamiento en la nube, reporte de fallos o alojamiento de páginas web estáticas mediante un servicio de *hosting* gratuito.
- **Herramientas de crecimiento:** ayuda a dar a conocer la aplicación mediante anuncios, promociones, invitaciones, indexación en buscadores web..., etc.
- **Herramientas de monetización:** permite la inclusión de publicidad para generar ingresos fácilmente desde tu aplicación. Incluye también un servicio de análisis de toda la información de usuarios activos, registrados..., etc. para ayudar en este aspecto.

Los servicios de *Firebase* pueden ser usados por cualquier aplicación para simplificar su desarrollo. En este caso, se hará uso de dos de sus herramientas de desarrollo que se integrarán tanto en la aplicación *Android*, como Web: autenticación de usuarios y una base de datos en tiempo real.

Se usará la autenticación de usuarios para controlar el acceso a la aplicación. El servicio proporciona herramientas para el registro, confirmación de registro, recuperación de contraseñas por distintos métodos, inicio de sesión mediante redes sociales... Sin embargo, solo se hará uso de la herramienta para controlar el inicio de sesión de usuarios previamente registrados por el administrador. Este servicio central proporciona seguridad ante ataques y delega el almacenamiento de información sensible a una aplicación que cumple con todos los estándares de seguridad, como es *Firebase*.

Además, también implementará una base de datos en tiempo real para mantener sincronizados entre ambas aplicaciones el registro de estaciones de carga disponibles a través de un listado. La base de datos que proporciona *Firebase* es una base de datos no relacional, que contendrá información sobre el nombre de la estación de carga y la localización de esta (dada su latitud y longitud). Desde la aplicación *Android* se podrá seleccionar de un listado a la hora de rellenar un reporte, y desde la aplicación web se podrá editar, borrar y añadir nuevas estaciones de carga.

■ *Google Maps Platform*



Figura 39: Logo de *Google Maps*.

Google Maps [23] es bien conocido por todos como el servicio de mapas de *Google*, que ofrece servicios de ubicación, trazado de rutas, control de tráfico, navegación en 360° (*Google Street View*)...

Para este proyecto, se usará la herramienta *Google Maps Platform*, herramienta creada para incorporar servicios de mapas de *Google Maps* a cualquier aplicación que diseñemos. Para ello, se realizan consultas a una API privada con una clave que se debe solicitar según la [documentación](#). En el apéndice B se presenta en detalle como obtener una clave API para poder instalar y ejecutar las aplicaciones, ya que usaremos una distinta para la aplicación móvil y web.

4.2.3. Tecnologías Web

Por último, se analizarán las tecnologías que serán de utilidad para el desarrollo de la aplicación Web administrador.

Dicha aplicación se despliega en una red local, y que solo puede ser accedida desde un computador. Para usarla, por tanto, no se necesitan de credenciales, por lo que es necesario asegurar el ordenador en el que se instala.

■ *IntelliJ IDEA*

IntelliJ IDEA es un IDE de desarrollo disponible tanto para Windows, MacOS y Linux. Se trata de uno de los entornos de desarrollo integrado más completos actualmente. Permite la programación en multitud de lenguajes como *Java*, *Kotlin* (lenguaje basado en *Java* desarrollado por *JetBrains*), *Haskell*, *Python*, *Ruby*...

Además, cuenta con sincronización en *Git* e incluye la posibilidad de trabajar con distintos *frameworks* de desarrollo como *Google App Engine*, *Spring*, *Android* (como se explicó anteriormente, *Android Studio* está basado en este IDE).

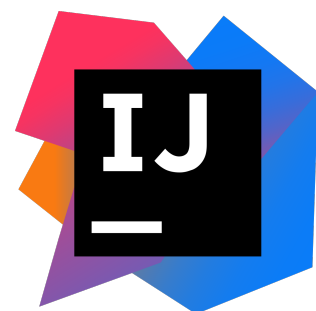


Figura 40: Logo de *IntelliJ IDEA*.

En este caso, y para el desarrollo de la aplicación web, se usará este IDE para la programación y lanzamiento de esta. Para ello, se basará *Spring*, que se detalla a continuación, junto con *Maven*, que ayudará a la hora de compilar, configurar y preparar el proyecto y todas las dependencias que este necesite.

■ Spring Boot



Figura 41: Logo de *Spring*.

Spring es un *framework* de desarrollo Web que surge como simplificación de aplicaciones Web basadas en *Java EE* (*Enterprise Edition*).

Java EE es una especificación de Java para el desarrollo de aplicaciones empresariales [28]. Propone un esquema de desarrollo en capas o *tiers*, que constituyen áreas funcionales aisladas. *Java EE* propone un modelo en cuatro capas:

- **Capa cliente:** son aquellas aplicaciones que interactúan con páginas web para mandar peticiones al servidor, y obtener una respuesta. Un cliente puede ser una aplicación, un navegador web...
- **Capa web:** se encarga de procesar la información solicitada en la capa cliente hacia la capa de negocio, para ser procesada. Para ello, genera páginas webs de manera dinámica, recoge información del cliente, mantiene sesiones... alguna de las tecnologías presentes en esta capa son las *Java Server Pages* o los *Servlets*.
- **Capa de negocio:** provee el código que otorga la funcionalidad a la aplicación².
- **Capa de Sistema de Información Empresarial (SIE):** incluye el acceso a bases de datos mediante un motor de persistencia, el acceso a sistemas ERP... Normalmente accede a servidores web externos mediante APIs como JDBC o JPA.

²Se podría comparar esta capa con los contratos inteligentes de las redes *blockchain*.

Se pueden diferenciar todas estas capas relacionadas en la Figura 42.

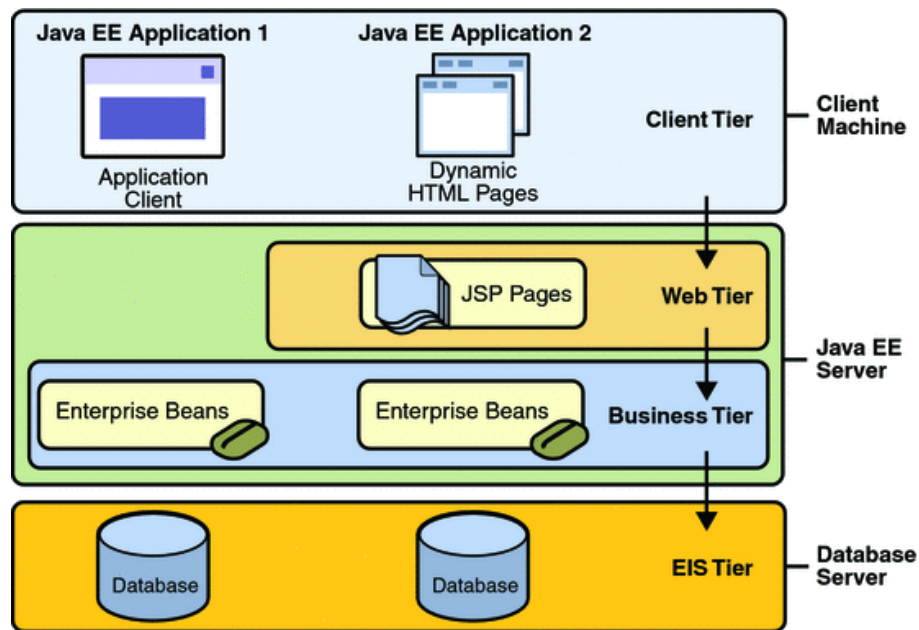


Figura 42: Organización de la estructura en capas o *tiers* de Jave EE. Fuente: [58].

Spring nace como una simplificación de esta estructura, dada la gran complejidad que las aplicaciones multicapa poseen en su desarrollo y conectividad con el resto de componentes [22].

Así, *Spring* se encarga de la gestión de dependencias (conexión a base de datos, paquetes, librerías..., etc.), así como de desplegar la aplicación en un servidor de aplicaciones (por defecto se despliega en Tomcat [45]).

La arquitectura de *Spring* simplificada que se pondrá en práctica en la prueba de concepto se resume en la Figura 43.



Figura 43: Flujo de información en la aplicación *Spring* desarrollada.

Se puede observar que existen tres componentes, que se detallan a continuación:

- **Cliente:** corresponde con al capa cliente de *Java EE*. Los clientes envían peticiones HTTP al servidor *Tomcat*, y este las procesa analizando la ruta de la petición en el *Controller*. El cliente puede realizar tanto cualquier tipo de petición HTTP, y recibir respuestas con páginas webs creadas dinámicamente por parte del servidor.
- **Controller:** recibe las peticiones del cliente, las procesa y delega en el *Service* el procesamiento de los datos recibidos. El *Controller* también se encarga de interactuar con las vistas que serán enviadas al cliente.
- **Service:** clase *Java* típica que contiene la lógica de negocio de la aplicación. En el *Service* se incluyen las operaciones que se deben llevar a cabo y pueden ser solicitadas por el *Controller*.

Para poner en marcha una aplicación de *Spring*, es interesante visitar la página de [Spring Initializr](#), donde se permite seleccionar el tipo de proyecto, lenguaje usado y su versión, versión de *Spring*, listado de dependencias...

▪ HTML

HTML [36] es el lenguaje de etiquetas usado para la creación de páginas web. Es el componente básico de la web y es interpretado por todos los navegadores web para mostrar información. Una página web no solo cuenta con una página HTML, pues se suele usar con otros componentes como CSS para añadir estilos, JS para añadir interacción en la máquina cliente...

La versión actualmente usada de HTML, HTML5, fue lanzada el 28 de octubre de 2014. Añade y elimina elementos del lenguaje para adaptarlo al nuevo esquema de la web.

Todos los documentos HTML se componen de una etiqueta “html” (engloba todo el documento), una etiqueta “head” (metainformación, enlaces a recursos...) y una etiqueta “body” (donde se detalla el contenido que se muestra en pantalla). En la Figura 45 se presenta un ejemplo de página web creada con HTML.



Figura 44: Logo de HTML5.



Figura 45: Ejemplo de página web en HTML y visualización mediante un navegador web.

■ Thymeleaf



Figura 46: Logo de Thymeleaf.

Thymeleaf [56] es un motor de plantillas que permite generar páginas HTML dinámicas mediante sencillos comandos. Fue desarrollado por el ingeniero de software español Daniel Fernández (GitHub) con el propósito de crear un software de código abierto que permita crear páginas web dinámicas de manera fácil y rápida.

Se integra perfectamente en *Spring* por lo que no es difícil usarlo para presentar información de cualquier tipo, añadiendo atributos y etiquetas desde el *Controller*.

Thymeleaf trabaja con distintos tipos de expresiones, aunque solo se detallaran las más importantes [35]:

- **Variables:** se representan mediante `${...}`. Representan una variable pasada como atributo a la vista. Por ejemplo, para representar la variable “var”, se usará `${var}`, o para el atributo “nombre” del objeto “autor”, `${autor.nombre}`.
- **Enlaces:** permiten definir enlaces con parámetros o variables. Se representan mediante `@{...}`. Por ejemplo, si quiere crear un formulario cuya acción haga una petición a la ruta “/form/id” puedo usar la expresión `@{/form/id(id=autor.id)}`.
- **Literales:** las expresiones literales pueden ser cadenas de texto, números, expresiones booleanas..., etc.
- **Operadores:** se pueden utilizar operadores unarios y binarios como la concatenación, comparación, igualdad, aritmética..., etc.

- **Atributos:** para componer la página HTML usando *Thymeleaf* debemos especificar el atributo que se desea procesar con la expresión en Thymeleaf deseada. Por ejemplo, si queremos crear una etiqueta “a” con un atributo “href” creado a partir de un enlace, lo definiremos añadiendo el prefijo “th:” al atributo: “<a th:href=@{...}>”.

Se presenta a continuación un ejemplo explicado de una pieza de código desarrollada en *Thymeleaf*:

```
<a th:with="baseUrl=${user.admin}? '/admin/home' : ${user.homeUrl}" th:href="@{{{baseUrl}}}">
```

Se busca crear una etiqueta “a” de HTML con un atributo “href” que sea un enlace definido por la variable de nombre “baseUrl”. Dicha variable se ha definido mediante el atributo “th:with”, que asigna a “baseUrl” el valor “/admin/home” o el valor del atributo “homeUrl” del objeto “user” dependiendo si el valor de “admin” del mismo objeto devuelve True o False, usando para ello un operador ternario.

Si “user.admin” es True, *Thymeleaf* renderizará la etiqueta del siguiente modo:

```
<a href="/admin/home">
```

Si “user.admin” es False y “user.homeUrl” tiene como valor “/user/home”, *Thymeleaf* renderizará la etiqueta del siguiente modo:

```
<a href="/user/home">
```

▪ JavaScript

JavaScript es un lenguaje de programación de alto nivel que permite añadir funciones interactivas a páginas webs. Funciona junto con HTML y CSS para la creación de dichas páginas.

JavaScript [31] no se debe confundir con Java pues la sintaxis es distinta y el propósito también lo es. No usa una estructura tan formal como Java y no se trata de un lenguaje compilado, pues este se ejecuta en la máquina del cliente en tiempo de ejecución. Así se consigue delegar parte del procesamiento a la máquina del cliente, normalmente un navegador web, para aliviar la carga del servidor.



Figura 47: Logo de *JavaScript*.

El código en *JavaScript* se puede incluir en una página HTML usando la etiqueta `<script>`, que permite incluir directamente piezas de código, o enlazar un archivo de extensión JS.

En el proyecto se hará uso de *JavaScript* para controlar la verificación de formularios, accediendo y controlando la información antes de ser enviada al servidor. Además, se usará para representar visualmente los reportes enviados mediante un mapa de *Google Maps*, haciendo uso de *Google Maps Platform* para tecnologías Web.

■ CSS



Figura 48: Logo de CSS

CSS, o *Cascading Style Sheets* es la última tecnología del trío formado junto con HTML y JS para la creación universal de páginas web. Esta ayuda en la personalización y en los estilos que se añaden a cada elemento definido en HTML.

Previamente, los estilos se especificaban usando el atributo “style” dentro de cada etiqueta de HTML. Sin embargo, esta tarea es tediosa y repetitiva, pues se debe replicar en cada nuevo elemento que se cree, problema que iba aumentando como así lo hacía la complejidad de las páginas web.

Así, se creó esta tecnología que separaba la estructura del documento de su personalización. Además, permite agregar estilos a grupos de etiquetas, a grupos de elementos dados sus identificadores o clases... acelerando el desarrollo de la página y mejorando a su vez el rendimiento de estas [47].

■ Bootstrap

Bootstrap es un *framework* de CSS desarrollado por la empresa Twitter Inc. en 2010. Se trata de un software de código abierto para el diseño de páginas rápido y simple, y que responden al tamaño de multitud de dispositivos con configuraciones de pantalla muy distintas (grandes monitores, portátiles, teléfonos móviles, tabletas...) [10].

Este *framework* también hace uso de *scripts* JS para el desarrollo más rápido e intuitivo de una página web, así como la



Figura 49: Logo de *Bootstrap*

incorporación de elementos más avanzados como menús de navegación, barras de carga, control de páginas..., etc.

La última versión de *Bootstrap*, la versión 5.2, incluye en su [documentación](#) cómo usar cada componente, con multitud de ejemplos, así como el proceso de instalación, que es tan sencillo como incluir un enlace en cada página HTML que creemos.

■ Selenium



Figura 50: Logo de *Selenium*

Selenium es una herramienta usada para crear pruebas automatizadas funcionales de aplicaciones web. Esto es, la herramienta recoge las acciones y comportamientos esperados de la aplicación, y las ejecuta y comprueba en tiempo de ejecución, para así simular la acción de un usuario [38].

La herramienta es de código abierto, permite su ejecución en los navegadores y sistemas operativos más conocidos del mercado, y permite exportar los *tests* realizados a código *Java*, *Python*, *Ruby*... para ser ejecutados en otras plataformas.

Se usará esta herramienta para realizar pruebas en la aplicación web creada. En concreto, se utilizará *Selenium IDE*, que se instala como una extensión del navegador web. Esta permite crear proyectos con *tests* unitarios mediante la grabación de acciones directamente en la aplicación web que se desea probar.

4.3. Contrato inteligente

Una vez presentadas de manera exhaustiva todas las tecnologías web que se usarán en esta prueba de concepto, se procede a relatar las implementaciones realizadas.

En primer lugar, se abordará el contrato inteligente. Dicho contrato se enmarca dentro de una red *blockchain*, que en este caso desplegaremos una basada en *Hyperledger Fabric* sobre el servicio BaaS de *Kaleido*.

Se debe proporcionar un fichero compilado en código binario del contrato inteligente programado en *Go*. En el contrato tenemos que especificar tanto la estructura de datos que usaremos para almacenar la información, así como las funciones disponibles que podrán ser llamadas a través de la API REST.

Se creará para para ello un objeto *QueryResult*, que tendrá especificado un par clave-valor, donde clave es el identificador del reporte, y valor es el propio reporte, encapsulado en el objeto *Report*. La información del reporte se almacena en un archivo JSON que contiene toda la información necesaria para que las aplicaciones puedan procesarlo.

En la Figura 51 se presenta su estructura mediante un diagrama UML.

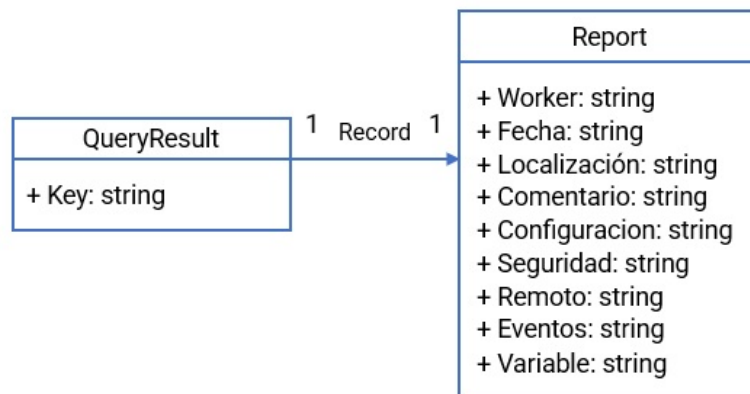


Figura 51: Diagrama UML de la estructura de datos creada para almacenar información de los reportes en la red.

En la red se pueden ejecutar un total de tres funciones que se han diseñado para cumplir su funcionalidad:

- **CreateReport**: esta función tiene como argumentos cadenas de texto con la información

enviada por los operarios a través de la aplicación móvil. Esta información, que será introducida a través de un formulario que visualizaremos en el siguiente apartado, es manipulada y enviada a la red haciendo una petición POST a la API llamando a esta función. A continuación, realizada la llamada `PutState(Key,Record)` para introducir en el registro de la red dicha información.

- **QueryReport** y **QueryAllReports**: estas funciones realizan una llamada a `GetState(Key)` y a `GetStateByRange(startKey,endKey)` para consultar del registro la información solicitada. Generalmente, se usará la segunda función y se filtrarán aquellos reportes encontrados según el usuario que ha realizado la petición.

Se puede obtener más información del código fuente programado para este contrato inteligente en el Apéndice [G](#).

Cada vez que un operario prepara un reporte y lo envía a la red, este se procesa y ocupa un bloque en esta. Este bloque se añade al final de esta y contiene la información suministrada siguiendo el siguiente formato:

```
key:"Report1-user1"
value:{"worker":"user1","fecha":"17-05-2022 11:47:43","
  localizacion":"1,1","comentario":"primer reporte","
  configuracion":"","seguridad":"","remoto":"","eventos":"Daño
  en un conector","variables":""}
```

Los parámetros introducidos en el reporte se procesan en los valores de `configuracion`, `seguridad`, `remoto`, `eventos` y `variables`. Los datos se introducen como cadenas de texto dependiendo de lo especificado por el operario. Por ejemplo, en el valor de `seguridad` existen dos posibles eventos a notificar³). Si el operario rellena ambos, la cadena de texto que se envía a la red en el campo “seguridad” estará compuesta de una serie de apartados identificados con `[x]` y seguidos del valor introducido en dicho apartado, de la siguiente forma:

```
"seguridad":"[1]1[2]Nuevo evento de seguridad"
```

Así, se consigue un registro temporal de todos los reportes enviados por todos los operarios, pudiendo acceder cada uno de ellos a los reportes que han enviado, o pudiendo el administrador visualizar todos. Gracias a las capacidades de la red *blockchain* desplegada, los

³Según la documentación recogida en el Apéndice [C](#)

reportes no van a poder ser manipulados en el futuro, consiguiendo así la **trazabilidad** que se busca en la implementación de la prueba de concepto.

4.4. Aplicación móvil cliente

En este apartado se presentará la aplicación móvil cliente desarrollada. Si se revisa de nuevo el escenario planteado para esta prueba de concepto, presentado en la Figura 2, la aplicación *Android* es la encargada de la publicación y el listado de reportes por parte de los operarios centrales y de campo.

La aplicación *Android* desarrollada implementa el diagrama de actividades representado en la Figura 52.

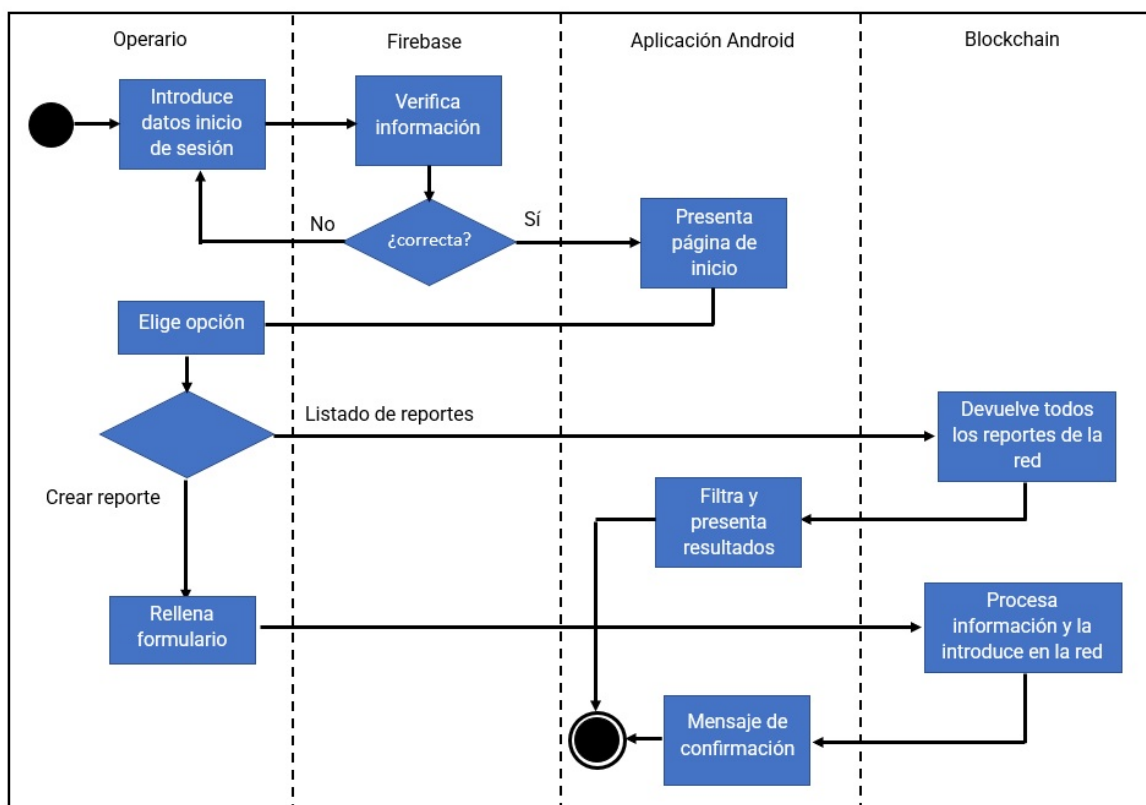


Figura 52: Diagrama de las actividades principales y la interacción de cada componente.

Para ello se han creado cuatro actividades en la aplicación:

1. **Actividad de inicio de sesión:** implementa el caso de uso UC-02 (documentado en el Cuadro 7). La actividad se conecta por medio de Internet a los servidores de *Firebase* del proyecto para comprobar la identidad de los usuarios que buscan iniciar sesión. Si la identidad no es correcta, se informa con un pop-up en pantalla y se invita al usuario a intentar de nuevo la autenticación (véase Figura 53).

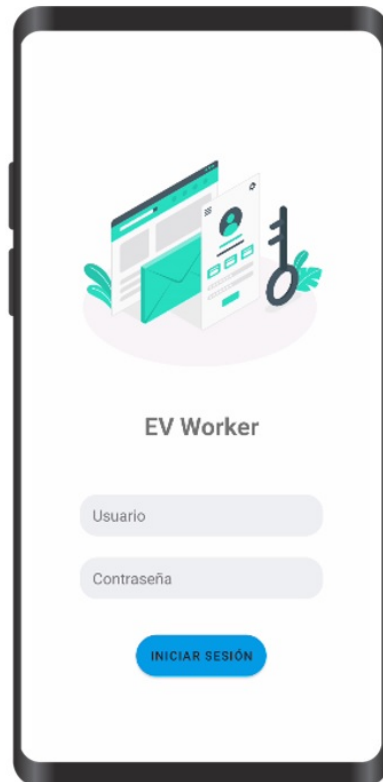


Figura 53: Representación de la pantalla de inicio con la actividad de inicio de sesión ejecutándose.

2. **Actividad principal:** implementa el caso de uso UC-04 (documentado en el Cuadro 9). En esta actividad se presenta una página de inicio con información y accesos directos relevantes para el operario, así como una comprobación del estado de la red. Esta actividad está formada por dos fragmentos, que se comportan como componentes independientes dentro de una actividad. En el menú inferior se puede navegar entre la pantalla principal, el listado de reportes y un botón para cerrar sesión en la aplicación. En el apartado de listado de reportes, el operario puede consultar la información enviada en cada uno, así como filtrar por nombre, fecha o comentario (véase Figura 54).

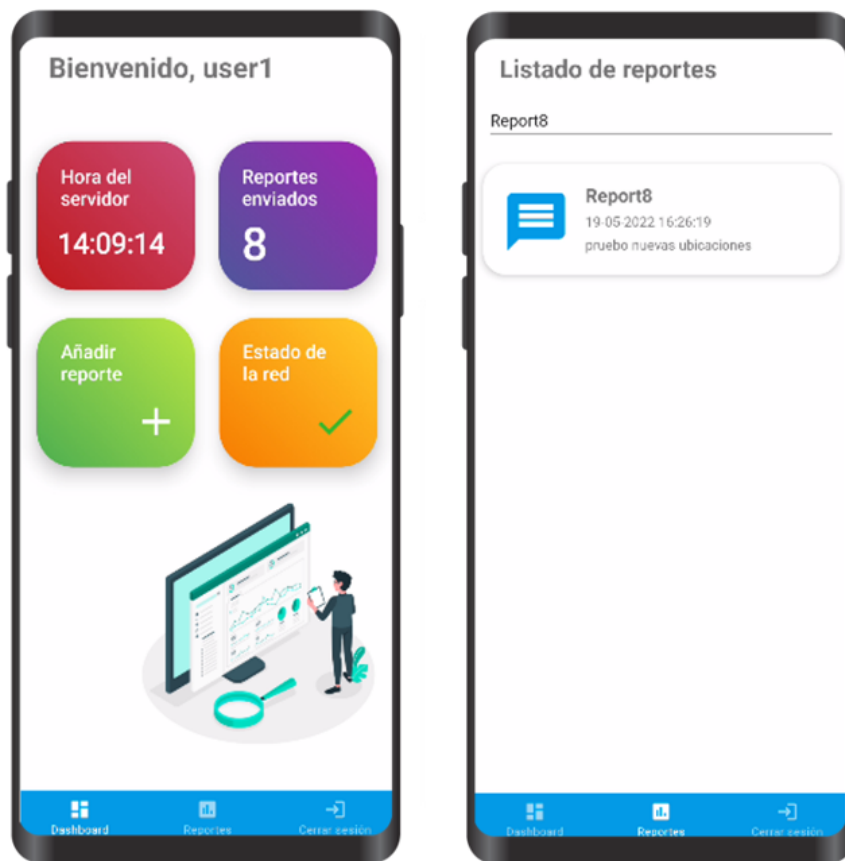


Figura 54: Representación de la actividad principal, que incluye las vistas de resumen y de listado de reportes.

3. **Actividad de creación de reporte:** implementa el caso de uso UC-03 (documentado en el Cuadro 8). El formulario permite introducir datos de distintos campos, seleccionados a partir de los casos de uso del protocolo OCPP extraídos y documentados en los apéndices C y D. En el caso de las variables, el sistema cuenta con un listado que permite agregar pares (variable, valor), y poder modificar y eliminar las variables provisionalmente añadidas en el reporte. El operario puede también incluir desde aquí la localización de una estación de carga precargada, o escoger su propia ubicación, accediendo a una nueva actividad. Una vez que el formulario ha sido rellenado, el operario lo envía a la red (véase Figura 55).

Nuevo reporte
Rellene todas las características que deben constar en el reporte

USAR MI UBICACIÓN ACTUAL

Estación de carga:
Iberdrola Charging Station Malaga

Localización: 36.727805,-4.417920

Seguridad
Actualización perfil de seguridad
Evento de seguridad
Descripción

Configuración
Inicio en frío ☐
Reseteo de central o componente
Componentes reseteados
Migración a nueva CSMS
Nueva CSMS
Modificación red de conexión
Detalles del cambio
Instalación de componentes

Figura 55: Representación de la actividad de creación de reporte por parte del operario.

4. **Actividad de selección de ubicación:** Si el operario desea incluir su propia ubicación, debe seleccionar el botón “Usar mi ubicación actual” presente en la actividad de creación de reporte. Este redirige a una vista en la que se puede comprobar si la ubicación del operario es correcta, y puede incluirla en el reporte pulsando “Incluir ubicación”. Para la inclusión del mapa, se usa la API de *Google Maps Platform*⁴ (véase Figura 56).

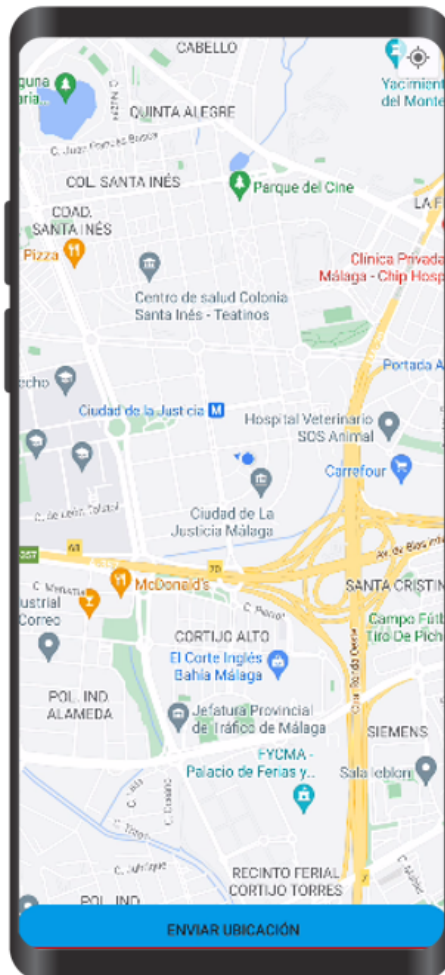


Figura 56: Representación de selección de ubicación ya posicionada, accesible desde la actividad de creación de reporte.

⁴Para el uso de esta API en la aplicación, es necesario añadir en el archivo de manifiesto los permisos de ubicación `ACCESS_COARSE_LOCATION` y `ACCESS_FINE_LOCATION`, así como el solicitarlos personalmente al usuario al intentar acceder a la actividad.

En *Android Studio* se deben definir las **actividades**, el resto de **clases Java** y los ficheros de **recursos**.

Los ficheros de recursos no son relevantes a la hora de documentar la memoria, basta con saber que cada actividad tiene asociado un *Layout*, donde se presenta su interfaz. La funcionalidad de la aplicación, por tanto, se estructura en el organigrama representado en la Figura 57.

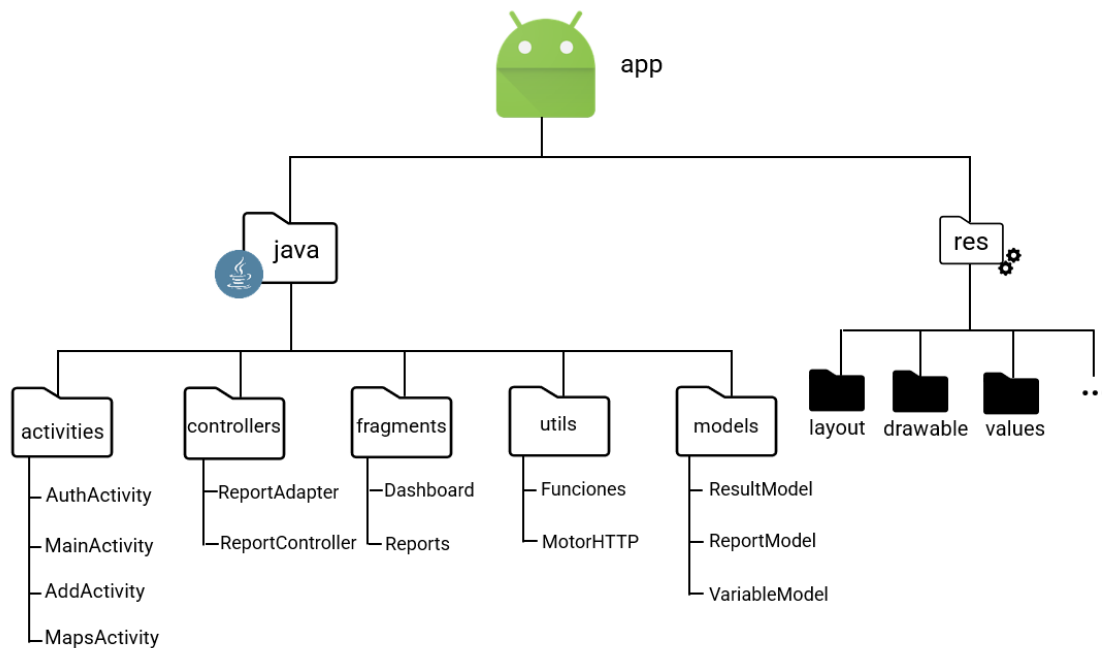


Figura 57: Organigrama de archivos creados para el desarrollo de la aplicación.

Se pueden observar que existen cinco paquetes principales dentro de la carpeta Java, donde cada uno contiene la siguiente información:

1. **Activities:** contiene cuatro clases con las distintas actividades implementadas (en el mismo orden en el que se han presentado anteriormente). Dichas clases heredan de `AppCompatActivity` y deben implementar, como mínimo, el método `onCreate()`, donde se asigna el fichero XML correspondiente al *layout* con el método `setContentView()`.
2. **Controllers:** se tratan de clases de soporte para el manejo de los reportes. En `ReportAdapter` se implementa el listado y filtrado de reportes, mientras que en `ReportController` se maneja la información recibida de la red.
3. **Fragments:** son clases complementarias a la actividad `MainActivity`. La clase está

compuesta de dos vistas, que se definen como fragmentos de ella en estas dos clases, `Dashboard` y `Reports`.

4. **Utils:** clases que contienen información de apoyo a los controladores. La clase `Funciones` define un objeto de tipo *Enum* en Java que representa las funciones del contrato inteligente⁵. La clase `MotorHTTP` contiene la URL del *endpoint* de la REST API generada para el nodo *peer* al que busca conectarse, y un token de autenticación único para la aplicación.
5. **Models:** contiene una representación de los modelos de datos existentes que se manejan. `ResultModel` contiene el identificador de un reporte, y un objeto `ReportModel`, mientras que `VariableModel` representa una asociación de una variable del sistema (ver Apéndice D) con un valor dado.

Finalmente, para completar la definición de todas las funcionalidades que se implementan, es necesario conocer cómo son las peticiones que se realizan a la red desde la aplicación. Para ello, se remite al lector al Apéndice F, donde se detalla el mecanismo de envío de mensajes, así como la cabecera que se envía en cada petición.

4.5. Aplicación web administrador

En este último apartado de la documentación de la prueba de concepto se presenta toda la información relevante sobre la implementación de la aplicación web administrador.

Esta será creada usando *Spring*, por ello, sigue el flujo de información que se representó en la Figura 43. La aplicación contará con un total de cuatro vistas a las que el administrador puede acceder fácilmente gracias a un menú de navegación en la parte superior. La estructura de la aplicación, como también se presentó en la aplicación *Android*, se representa en la Figura 58.

⁵Puede tomar por tanto, los valores de `QueryAllReports`, `CreateReport` o `QueryReport`.

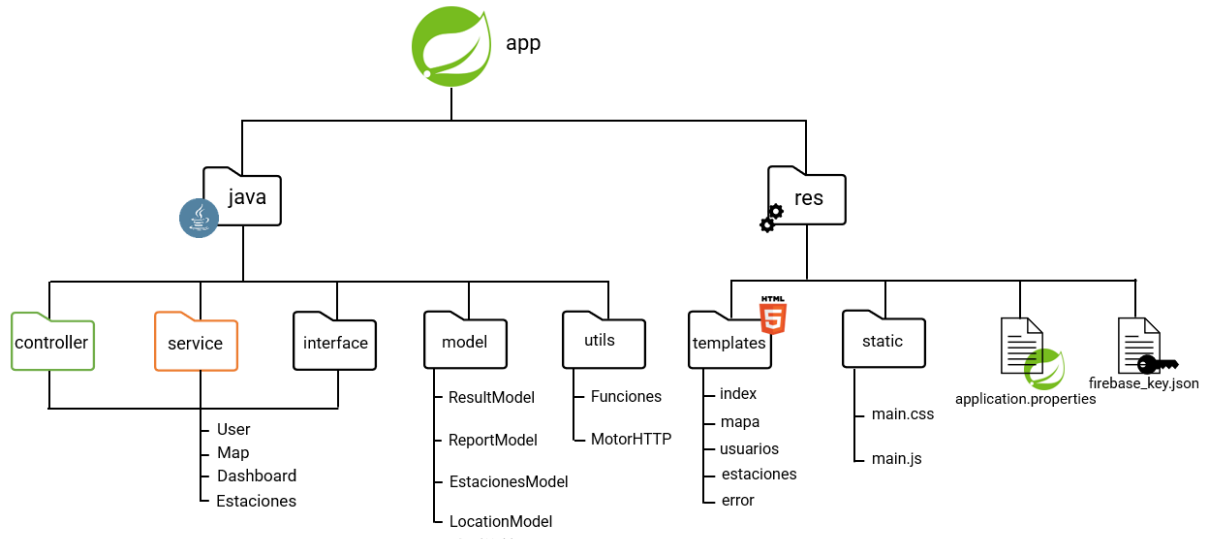


Figura 58: Organigrama de archivos creados para el desarrollo de la aplicación web.

La estructura es bastante similar a la usada en la aplicación *Android*, pues también cuenta con una carpeta con todos los archivos *Java*, y otra carpeta de recursos. En este caso, si los trataremos:

1. **Carpeta Java:** contiene los paquetes *Controller*, *Service* e *Interface* sobre las cuatro vistas diseñadas en la aplicación. Así, el *UserController* responde a las peticiones realizadas sobre la página de administración de usuarios, que a su vez delega las operaciones en *UserService*, cuyos métodos están declarados y documentados en *UserInterface*. En los paquetes *model* y *utils* se almacena la misma información que en la aplicación móvil.
2. **Carpeta Resources:** contiene los recursos principales de la aplicación. Destaca el archivo *application.properties*, con configuración del servidor de aplicaciones y de la aplicación Spring en general, y el archivo *firebase_key.json*, que contiene una clave privada para la conexión de la aplicación al servidor de *Firebase* ⁶. En la carpeta *templates* se almacenan las páginas HTML que devolverán cada controlador respectivamente, mientras que en *static* se encuentran los archivos de estilos CSS y de *scripts* JS.

A continuación se presenta la aplicación web con sus funcionalidades implementadas a través de las cuatro vistas con las que cuenta:

⁶En este caso, se usa para la administración de estaciones, pues la aplicación no cuenta con mecanismos de autenticación.

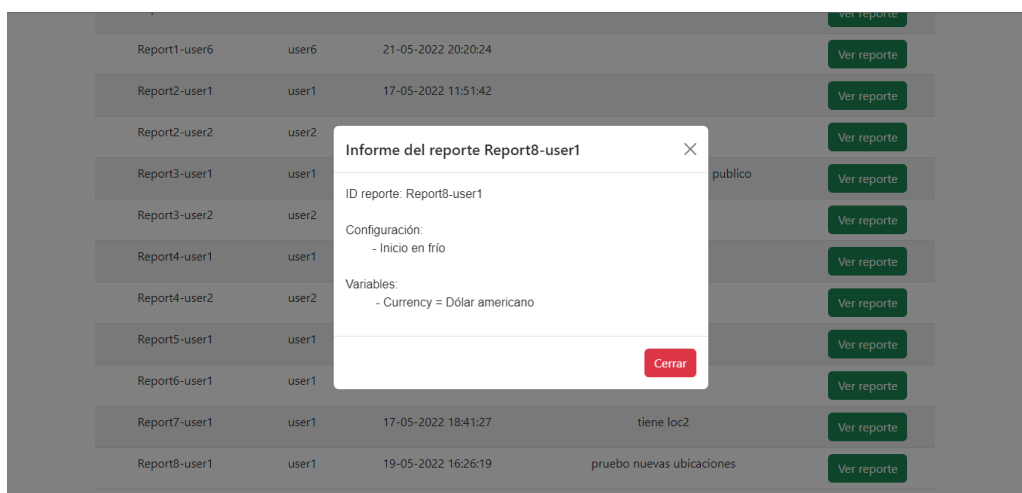
1. **Vista de *Dashboard*:** corresponde con la vista predeterminada de la aplicación, usando la plantilla `index.html`. Implementa el caso de uso UC-05 (documentado en el Cuadro 10), en el que el administrador puede consultar información sobre todos los reportes enviados, así como filtrar por identificador del reporte, fecha o comentario (véanse Figuras 59 y 60).



The screenshot shows the 'EV Worker Admin' dashboard with a navigation bar containing 'Dashboard', 'Mapa', 'Usuarios', and 'Estaciones'. The main section is titled 'Listado de reportes en la red'. It features a search bar with a 'Búsqueda' input, a 'Buscar' button, and a 'Resetear' button. Below the search bar is a table with the following columns: ID, Operario, Fecha, and Comentario. Each row represents a report and includes a 'Ver reporte' button.

ID	Operario	Fecha	Comentario	
Report1-operario2	operario2	21-05-2022 21:43:42		Ver reporte
Report1-user1	user1	17-05-2022 11:47:43	primer reporte	Ver reporte
Report1-user2	user2	17-05-2022 12:07:48	hola	Ver reporte
Report1-user6	user6	21-05-2022 20:20:24		Ver reporte
Report2-user1	user1	17-05-2022 11:51:42		Ver reporte
Report2-user2	user2	19-05-2022 12:38:35		Ver reporte
Report3-user1	user1	17-05-2022 11:56:07	segundo comentario que publico	Ver reporte
Report3-user2	user2	19-05-2022 12:38:35		Ver reporte

Figura 59: Captura de pantalla del *Dashboard* de la aplicación, con el listado de todos los reportes de la red.



The screenshot shows the same dashboard as Figure 59, but with a modal window open for 'Informe del reporte Report8-user1'. The modal displays the report ID, configuration, and variables.

ID	Operario	Fecha	Comentario	
Report1-user6	user6	21-05-2022 20:20:24		Ver reporte
Report2-user1	user1	17-05-2022 11:51:42		Ver reporte
Report2-user2	user2			Ver reporte
Report3-user1	user1		publico	Ver reporte
Report3-user2	user2			Ver reporte
Report4-user1	user1			Ver reporte
Report4-user2	user2			Ver reporte
Report5-user1	user1			Ver reporte
Report6-user1	user1			Ver reporte
Report7-user1	user1	17-05-2022 18:41:27	tiene loc2	Ver reporte
Report8-user1	user1	19-05-2022 16:26:19	pruebo nuevas ubicaciones	Ver reporte

Informe del reporte Report8-user1

ID reporte: Report8-user1

Configuración:

- Inicio en frío

Variables:

- Currency = Dólar americano

Cerrar

Figura 60: Información detallada mostrada por la aplicación web de cada reporte enviado.

2. **Vista de Mapa:** presenta la visualización en un mapa interactivo usando *Google Maps Platform* de los reportes enviados. La vista está contenida en la plantilla `mapa.html` y esta implementa el caso de uso UC-07 (documentado en el Cuadro 12), en el que el administrador puede visualizar en un mapa los reportes contenidos en la red. Además, puede filtrar para conseguir una mejor visualización, al igual que se hacía en la vista del *Dashboard* (véanse Figuras 61 y 62).

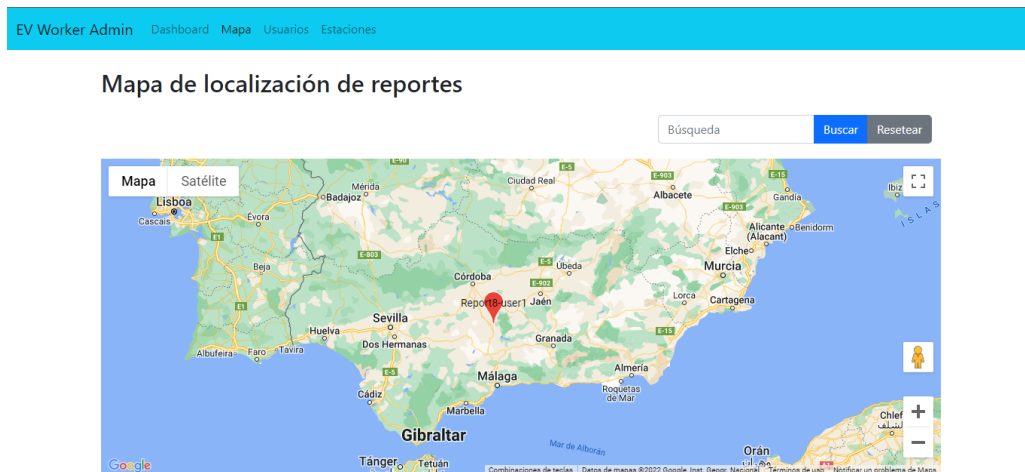


Figura 61: Captura de pantalla de la pestaña Mapa de la aplicación.

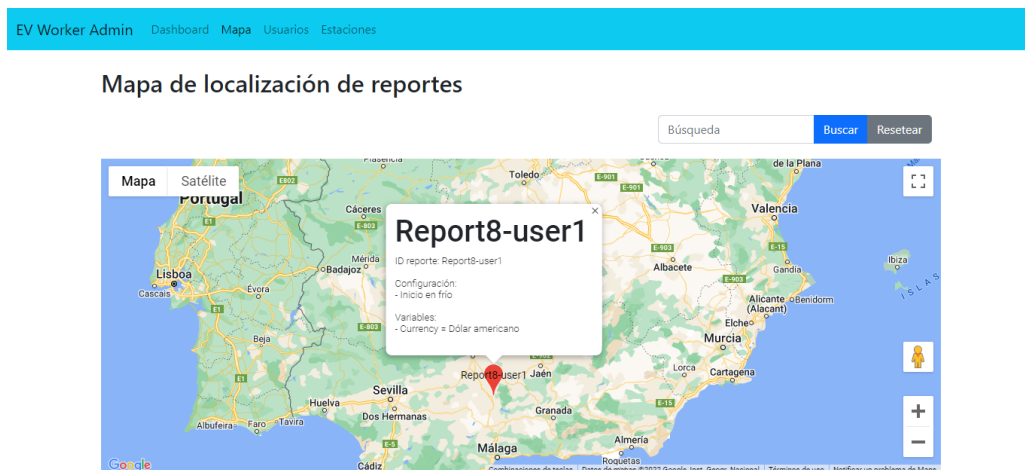
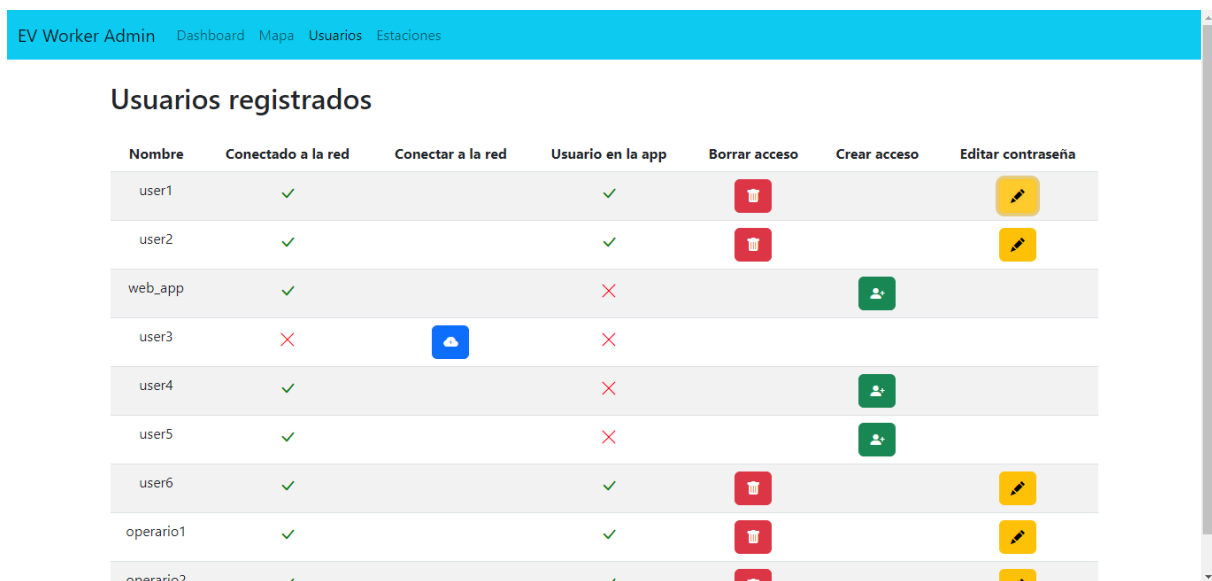


Figura 62: Información detallada mostrada por la aplicación web de cada reporte enviado en el mapa interactivo.

3. **Vista de Usuarios:** en esta pantalla se presenta la administración de los usuarios tanto en la red como en la aplicación móvil, está contenida en la plantilla `usuarios.html` y cumple el caso de uso UC-06 (documentado en el Cuadro 11). Los usuarios que aparecen listados son aquellos que se encuentran actualmente disponibles en la red, aunque no necesariamente conectados y autorizados para participar en ella (son aquellos dados de alta en el servicio MSP que proporciona la red). El administrador puede autorizarlos para participar en la red (introduciendo para ello su contraseña), así como administrar su acceso a la aplicación. Es condición necesaria que el usuario esté conectado y autorizado en la red para que el administrador pueda crearle un usuario en la aplicación. Dicho usuario tendrá la contraseña creada por el administrador, pudiendo este modificarla por cualquier motivo, o eliminar el acceso a la aplicación al usuario (bien por la finalización de la relación empresarial, o por otros motivos). Todas estas configuraciones se pueden realizar mediante los distintos botones que aparecen en la vista (véase Figura 63).



Nombre	Conectado a la red	Conectar a la red	Usuario en la app	Borrar acceso	Crear acceso	Editar contraseña
user1	✓		✓			
user2	✓		✓			
web_app	✓		✗			
user3	✗		✗			
user4	✓		✗			
user5	✓		✗			
user6	✓		✓			
operario1	✓		✓			
operario2	✓		✓			

Figura 63: Captura de pantalla de la pestaña Usuarios de la aplicación.

4. **Vista de Estaciones:** en esta última vista, que se obtiene de la plantilla `estaciones.html`, el administrador puede listar, editar, borrar y añadir direcciones de estaciones de carga predeterminadas. Las estaciones de carga y su localización estarán disponibles en el formulario de creación de reporte de la aplicación móvil. Esto es posible a que los datos se almacenan en la base de datos de *Firebase*, que es accesible a ambas tecnologías al encontrarse ambas localizadas en el mismo proyecto de *Firebase* (véase Figura 64).

Estaciones de carga disponibles				Crear
Nombre	Localizacion			
Endesa Charging Station Granada	37.148280,-3.66270211			
Fenie Energía Charging Station Lucena	37.430867,-4.474913			
Iberdrola Charging Station Malaga	36.727805,-4.417920			
Tesla Supercharger Aguadulce	37.256941,-4.986003			

Figura 64: Captura de pantalla de la pestaña Estaciones de la aplicación.

La aplicación web ha sido sometida a numerosos *tests* desarrollados usando la herramienta *Selenium IDE*, para probar el correcto funcionamiento de los cuatro módulos desarrollados. Los *tests* cuentan con comprobaciones del diseño de la página, resultados esperados y cambios en la página a partir de ciertas acciones automatizadas.

Finalmente, al igual que para la aplicación móvil, se remite al lector al apéndice F, para consultar la referencia de las direcciones de la API REST usadas para el desarrollo de ambas aplicaciones.

Con esto finalizaría la presentación de las aplicaciones, tecnologías y funcionalidades desarrolladas e implementadas para este proyecto.

5

Conclusiones y Líneas Futuras

Este TFG ha sido un proyecto muy interesante en el que he podido ampliar mis conocimientos en el desarrollo de las redes *blockchain*, las distintas clases de redes de este tipo que existen en el mercado, y todo el proceso de creación, despliegue y uso de estas.

Las redes *blockchain* son un tema que siempre me ha despertado cierto interés, y mucho más desde la introducción de nuevos casos de uso para estas. Bien es cierto que estas redes van ocupando cada año una mayor cuota de mercado, aunque de momento su uso es discreto en la mayor parte de empresas, bien por tratarse de una tecnología recién creada, o bien por pensar que el único valor que puede tener estas redes es en el terreno económico y especulativo.

Una de las mayores ventajas es la descentralización y la escalabilidad que pueden ofrecer, pero en contrapartida se encuentra la dificultad de su adopción, pues es más fácil recurrir a métodos tradicionales como las bases de datos, antes de invertir en infraestructura, mecanismos de autenticación..., etc.

Otro de los temas en los que he podido profundizar más es el referente a la automoción eléctrica, pues es un tema bastante técnico (y amplio) que cuenta con grandes aspiraciones de suplantar a los vehículos tradicionales por parte de las empresas y los gobiernos. Es una gran noticia pues es otro indicativo de cómo la tecnología evoluciona a pasos agigantados, y esta siempre debe buscar mejorar la sociedad y nuestro modo de vida.

En la prueba de concepto que se ha implementado y documentado en esta memoria, se han utilizado una gran variedad de tecnologías y conceptos de muchos campos de la informática, lo que ha servido para obtener una visión general de todo lo estudiado y asimilado a lo largo del grado.

El desarrollo de software novedoso siempre es un proceso complicado y desarrollado nor-

malmente por un equipo. En este caso, aún contando con un único autor, los resultados obtenidos con la puesta en marcha de la prueba de concepto son bastante satisfactorios.

Cabe destacar algunos planteamientos que se pueden seguir como **líneas futuras** en el desarrollo de este campo.

Una de ellas es la de integrar todo el sistema para que únicamente se use la red *blockchain* en todas las gestiones. El apartado de introducción de estaciones de carga precargadas en el sistema usa la base de datos de *Firebase*, y esta puede sustituirse para funcionar independientemente sin necesidad de contar con un servicio externo de almacenamiento de información.

Otra de las líneas que se plantean es el de añadir inteligencia artificial a un supuesto administrador para que pueda evaluar los reportes enviados por los operarios en base a buenas prácticas de mantenimiento, gestión y seguridad definidas en su evaluación. Así, con un procedimiento automatizado se puede filtrar la gran cantidad de información que puede llegar a la aplicación de manera rápida y sencilla.

Para concluir, destacar de nuevo la importancia de la tecnología y el desarrollo tecnológico, y cómo están evolucionando para los principales temas que se han cubierto en este proyecto: *redes blockchain* y movilidad sostenible. Estoy seguro que en los próximos años tendremos grandes avances en estos campos, pues todavía nos encontramos en la casilla de salida de ambos.

Apéndice A

Bibliografía

- [1] 101 Blockchains. (2021, 5 noviembre). *Hyperledger Fabric Tutorial: Getting Started Guide*. <https://101blockchains.com/hyperledger-fabric-tutorial/>
- [2] *Android*. (2016). Ediciones ENI.
- [3] Android Developers. (s. f.). *Configura tu compilación*. <https://developer.android.com/studio/build?hl=es-419>
- [4] ANFAC. (s. f.). *Matriculaciones de vehículos electrificados*. <https://anfac.com/matriculaciones-vehiculos-electrificados>
- [5] Apuntes de Programador. (2022, 20 mayo). *Introducción a Golang*. <https://apuntes.de/golang/introduccion-a-golang/#gsc.tab=0>
- [6] Araujo, G. (2020, 7 octubre). *Hyperledger Fabric: Conceptos y tipos de nodos*. BABEL Sistemas de Información. <https://www.babelgroup.com/es/Media/Blog/Agosto-2019/Hyperledger-Fabric-Conceptos-y-tipos-de-nodos>
- [7] Arroyo Guardado, Díaz Vico, J., & Hernández Encinas, L. (2019). *Blockchain*. Editorial CSIC.
- [8] Baset, Desrosiers, L., Gaur, N., & Novotny, P. (2019). *Blockchain development with hyperledger: build decentralized applications with hyperledger fabric and composer* (1st edition). Packt.
- [9] BBVA. (2022, 4 abril). *¿Qué es el automóvil eléctrico?* BBVA NOTICIAS. <https://www.bbva.com/es/sostenibilidad/que-es-el-automovil-electrico/>
- [10] *Bootstrap: guía para principiantes de qué es, por qué y cómo usarlo*. (2021, 12 febrero). Rock Content - ES. <https://rockcontent.com/es/blog/bootstrap/>

- [11] Comisión Europea. (s. f.). *Emissions in the automotive sector*. Internal Market, Industry, Entrepreneurship and SMEs. https://ec.europa.eu/growth/sectors/automotive-industry/environmental-protection/emissions-automotive-sector_en
- [12] Corda. (2022, 23 junio). *Leading DLT Platform for Regulated Industries*. <https://www.corda.net/>
- [13] dgt, www.dgt.es. (s. f.). *Consulta del distintivo ambiental*. CONSULTA DEL DISTINTIVO AMBIENTAL. <https://sede.dgt.gob.es/es/vehiculos/distintivo-ambiental/#>
- [14] eCar Selection. (2021, 13 abril). *La historia del coche eléctrico*. <https://ecarselection.com/la-historia-del-coche-electrico>
- [15] eEnergie Shop. (2021, 19 mayo). *Pasarelas protocolos Ocpp - Modbus*. Ocpp Modbus. <https://ocpp-modbus.com/>
- [16] Electromaps. (s. f.). *Listado de puntos de recarga en España*. <https://www.electromaps.com/puntos-de-recarga/espana>
- [17] Ethereum. (2022, 19 junio). *Ethereum Virtual Machine (EVM)*. Ethereum.Org. <https://ethereum.org/en/developers/docs/evm/>
- [18] EVBox. (s. f.). *¿Cómo funciona el Smart Charging?* | EVBox. <https://evbox.com/es-es/sobrenosotros/faq/como-funciona-smart-charging>
- [19] *Firebase: qué es, para qué sirve, funcionalidades y ventajas*. (2020, 17 mayo). DIGITAL55. <https://digital55.com/que-es-firebase-funcionalidades-ventajas-conclusiones/>
- [20] Fundación HABITEC & Ayuntamiento de Málaga (2017). *Guía de infraestructuras de carga de Vehículos Eléctricos*. Primera Edición
- [21] Gobierno de España (2021). *Plan de Recuperación, Transformación y Resiliencia*. Presidencia del Gobierno de España
- [22] Gonzalez, M. V. (2021, 17 marzo). *¿Qué es Spring Boot?* Codmind. <https://blog.codmind.com/que-es-spring-boot/>

- [23] Google. (s. f.). *Google Maps Platform Documentation*. Google Maps Platform . <https://developers.google.com/maps/documentation?hl=es>
- [24] Gradle. (s. f.). *Gradle Documentation*. <https://docs.gradle.org/current/userguide/userguide.html>
- [25] Hirsh. (2019). *Blockchain*. American Library Association.
- [26] Hita, M. Á. (2021, 12 noviembre). *Estos son los seis tipos de coches eléctricos y electrificados, por si estás buscando uno de segunda mano*. Motorpasión. shorturl.at/bpsNW
- [27] Iberdrola. (2021, 22 abril). *Smart contracts: contratos inteligentes para formalizar acuerdos en la era digital*. <https://www.iberdrola.com/innovacion/smart-contracts>
- [28] IBM. (2021, 5 marzo). *Aplicaciones Java SE y Java EE*. © Copyright IBM Corp. 2013. <https://www.ibm.com/docs/es/odm/8.5.1?topic=application-java-se-java-ee-applications>
- [29] IBM. (s. f.). *¿Qué es la tecnología de blockchain?* IBM Blockchain. <https://www.ibm.com/es-es/topics/what-is-blockchain>
- [30] Kaleido. (s. f.). *What is Kaleido?* Kaleido Docs <https://docs.kaleido.io/kaleido-platform/foundational-concepts/>
- [31] Kinsta. (2022, 23 enero). *¿Qué Es el JavaScript? Un Repaso al Lenguaje de Programación de Scripts Más Popular de la Web*. <https://kinsta.com/es/base-de-conocimiento/que-es-javascript/>
- [32] Ley 7/2021, de 20 de mayo, de cambio climático y transición energética. *Boletín Oficial del Estado*, 121, de 21 de mayo de 2021. <https://www.boe.es/eli/es/l/2021/05/20/7>
- [33] Liu, Chao & Gao, Jianbo & Li, Yue & Wang, Huihui & Chen, Zhong. (2020). *Studying gas exceptions in blockchain-based cloud applications*. Journal of Cloud Computing. <https://doi.org/10.1186/s13677-020-00176-9>
- [34] López, M. (2021, septiembre). *El vehículo eléctrico en España. Situación actual, objetivos y retos a abordar*. OBS Business School. <https://marketing.onlinebschool.es/Prensa/Informes/Informe%20OBS%20Vehiculos%20eco.pdf>

- [35] Magaña, L. M. L. (2021, 30 agosto). Qué es Thymeleaf. OpenWebinars.net. <https://openwebinars.net/blog/que-es-thymeleaf/>
- [36] MDN. (2021, 20 abril). *HTML: Lenguaje de etiquetas de hipertexto*. MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/HTML>
- [37] MDN. (2022, 30 mayo). *JavaScript*. MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/JavaScript>
- [38] Moreno, O. (2020, 5 mayo). *Introducción a Selenium*. Oscar Moreno. <http://oscarmoreno.com/selenium/>
- [39] OMG. (2017, diciembre). *OMG® Unified Modeling Language*. An OMG® Unified Modeling Language® Publication. <https://www.omg.org/spec/UML/2.5.1/PDF>
- [40] Open Charge Alliance (2018). *Open Charge Point Protocol 2.0.1*. <https://www.openchargealliance.org/protocols/ocpp-201/>
- [41] Open Charge Alliance. (s. f.). *Open Charge Alliance*. <https://www.openchargealliance.org/>
- [42] Open Charge Alliance. (2018, abril). *OCPP 2.0 Part 2 - Specification*. https://smartcharge.com.br/artigos/ocpp/OCPP-2.0_part2_specification.pdf
- [43] Orden PCM/756/2021, de 16 de julio, por la que se publica el Acuerdo del Consejo de Ministros de 13 de julio de 2021, por el que se declara como Proyecto Estratégico para la Recuperación y Transformación Económica el desarrollo de un ecosistema para la fabricación del Vehículo Eléctrico y Conectado. *Boletín Oficial del Estado*, 170, de 17 de julio de 2021. <https://www.boe.es/eli/es/o/2021/07/16/pcm756>
- [44] Polge, J., Robert, J., & le Traon, Y. (2021). Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, 7(2), 229–233. <https://doi.org/10.1016/j.icte.2020.09.002>
- [45] Project, A. T. (s. f.). *Welcome! Apache Tomcat®*. <https://tomcat.apache.org/>

- [46] Real Decreto de 24 de julio de 1889 por el que se publica el Código Civil. *Gaceta de Madrid*, 206, de 25 de julio de 1889. Artículo 1254. [https://www.boe.es/eli/es/rd/1889/07/24/\(1\)/con](https://www.boe.es/eli/es/rd/1889/07/24/(1)/con)
- [47] Robledano, A. (2021, 6 septiembre). *Qué es CSS y para qué sirve*. OpenWebinars.net. <https://openwebinars.net/blog/que-es-css/>
- [48] Satoshi Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. <https://bitcoin.org/bitcoin.pdf>
- [49] Sauras, Á. (2022, 12 abril). *Coches eléctricos: así funciona la recarga inalámbrica*. Autofaciles. <https://www.autofacil.es/coches-electricos-e-hibridos/recarga-inalambrica-coches-electricos/350339.html>
- [50] StatCounter Global Stats. (2022). *Mobile Operating System Market Share Worldwide*. <https://gs.statcounter.com/os-market-share/mobile/worldwide/>
- [51] Solomon. (2019). *Ethereum* (1st edition). John Wiley and Sons.
- [52] Tapscott, D., & Tapscott, A. (2017). *La revolución blockchain. Descubre cómo esta nueva tecnología transformará la economía global*. Ediciones Deusco. Séptima Edición.
- [53] Tesla. (s. f.). *Model S*. https://www.tesla.com/es_es/models
- [54] Tesla. (s. f.). *Supercargador*. https://www.tesla.com/es_es/supercharger
- [55] The Linux Foundation. (s. f.). *Key Concepts — hyperledger-fabricdocs main documentation*. Key Concepts. https://hyperledger-fabric.readthedocs.io/en/release-2.2/key_concepts.html
- [56] The Thymeleaf Team. (s. f.). *Thymeleaf Docs*. Thymeleaf. <https://www.thymeleaf.org/>
- [57] The Wallbox Store. (s. f.). *Tipos de conectores de Vehículos Eléctricos*. <https://thewallboxstore.com/es/info/tipos-de-conectores-de-vehiculos-electricos.html>
- [58] Universidad de Alicante. (2008). *¿Qué es Java Enterprise?* Especialista Universitario Java Enterprise. <http://www.jtech.ua.es/j2ee/2007-2008/jee.html>

- [59] Universidad Politécnica de Valencia. (s. f.). *Ciclo de vida de una actividad*. Máster en Desarrollo de Aplicaciones Móviles. <http://www.androidcurso.com/index.php/tutoriales-android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-actividad>
- [60] Unión Europea. (2019, 11 diciembre). Comunicación COM/2019/640 de la Comisión al Parlamento Europeo, al Consejo Europeo, al Consejo, al Comité Económico y Social Europeo y al Comité de las Regiones sobre *El Pacto Verde Europeo*. https://eur-lex.europa.eu/resource.html?uri=cellar:b828d165-1c22-11ea-8c1f-01aa75ed71a1.0004.02/DOC_1&format=PDF
- [61] Unión Europea. (2022, 25 marzo). *Objetivo 55*. European Council. <https://www.consilium.europa.eu/es/policies/green-deal/fit-for-55-the-eu-plan-for-a-green-transition/>

Apéndice B

Manual de Uso e Instalación

En esta sección se presenta un manual para la réplica de un sistema funcional igual al desarrollado en el caso práctico.

Para ello, se deben de seguir los siguientes pasos, que se detallan posteriormente:

1. Crear cuenta de *Kaleido* y desplegar red *blockchain*.
2. Crear cuenta de *Firebase* y asociar proyectos.
3. Crear cuenta de *Google Cloud Platform* y obtener claves API de *Goole Maps Platform*.
4. Preparar aplicaciones *Android* y web.

B.1. Creación de cuenta Kaleido y despliegue de red *blockchain*

En primer lugar, se habilitará y configurará la red de *blockchain* en el entorno de *Kaleido*. Para ello, hay que acceder a la consola desde este [enlace](#) (Figura 65). Basta con crear una cuenta gratuita (Figura 66), suficiente para desplegar los requisitos de este proyecto.

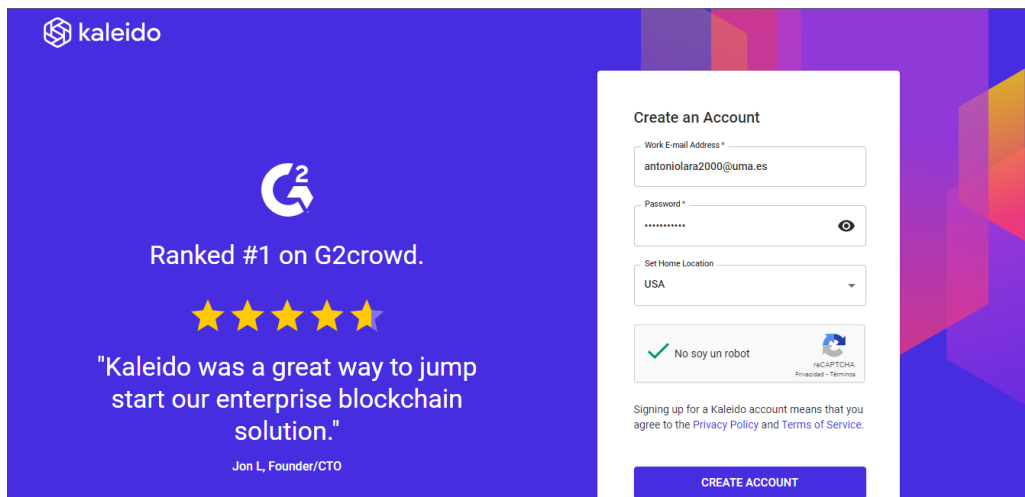


Figura 65: Captura de pantalla de la vista de registro.

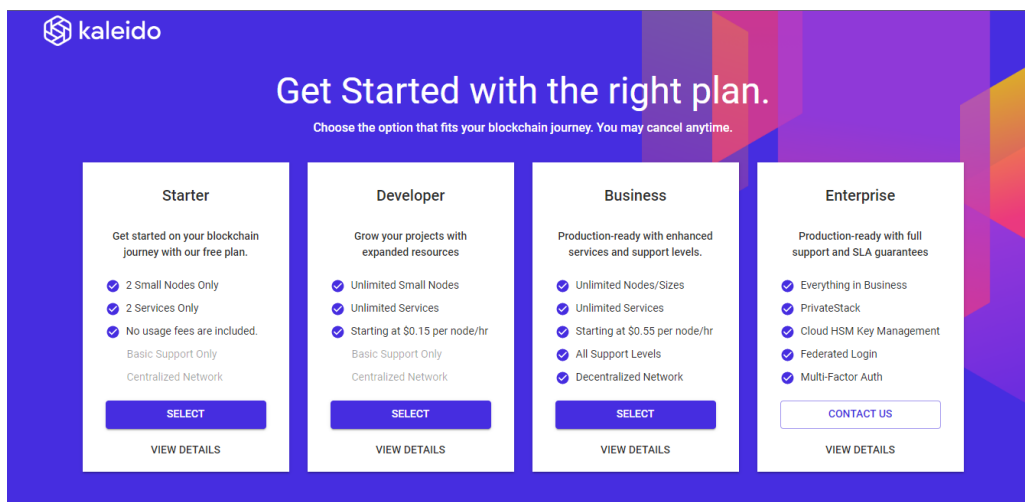


Figura 66: Captura de pantalla con los planes disponibles.

Una vez en la pantalla de inicio de la aplicación, se debe crear una red, pulsado para ello en el botón “*Create Network*”. Es necesario escoger un nombre y una ubicación (Figura 67) donde alojarse (es recomendable escoger aquel servidor más próximo).

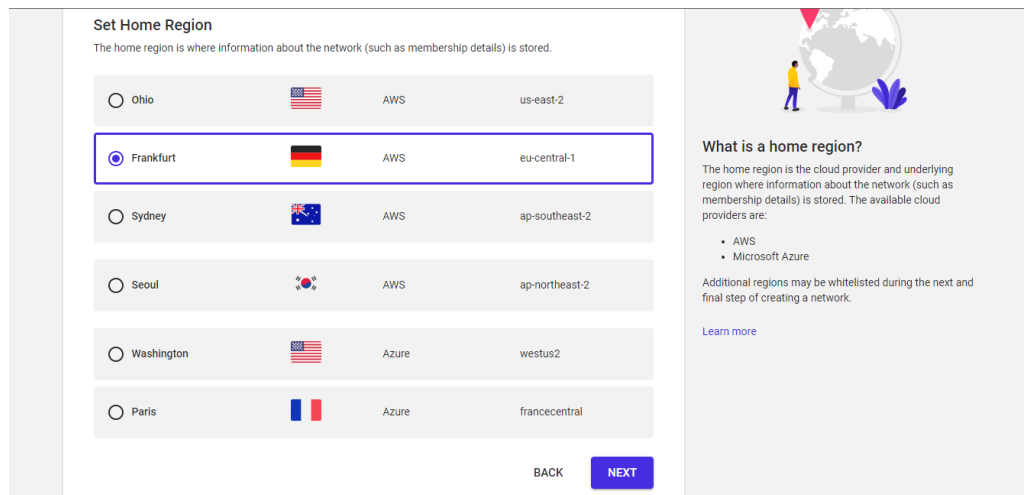


Figura 67: Captura de pantalla con las ubicaciones de red disponibles en *Kaleido*.

Dentro de la red, es necesario crear un entorno, donde se instalarán los nodos de la red. Para ello, hay que hacer click sobre “*Create an Environment*” en la nueva página. Esta preguntará sobre el tipo de red que se desea desplegar. Se escoge “*Standard Blockchain Service*” e “*Hyperledger Fabric*” (Figura 68).

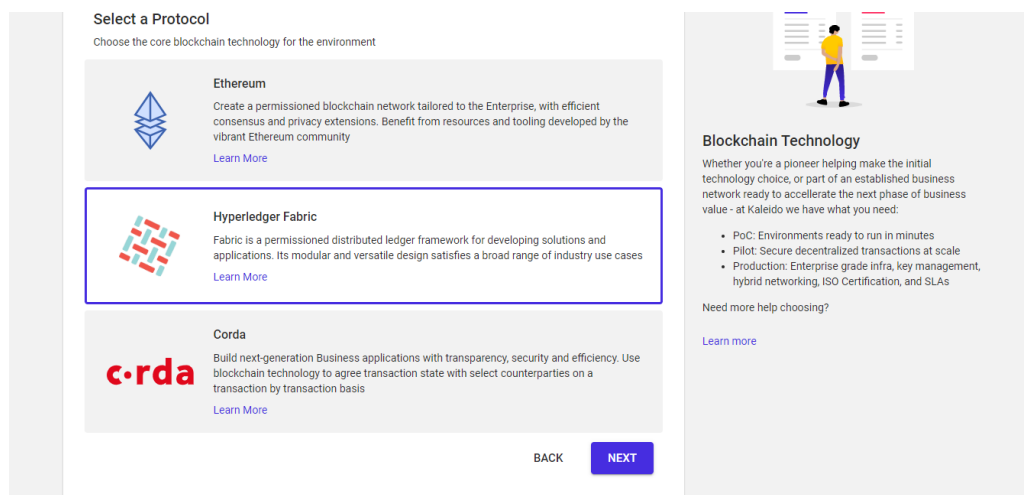


Figura 68: Captura de pantalla con los tipos de redes disponibles.

Ya se cuenta con la red creada, ya solo falta crear su estructura. Se debe desplegar un nodo *orderer* y un nodo *peer*. Para ello, en el menú de navegación de la izquierda, se seleccionará, dentro del *environment* creado, la opción *Manage Resources >Blockchain >Nodes >Add Orderer Node / Add Peer Node*. Para la creación de ambos (Figura 69) solicitan el nombre de dominio del nodo y el proveedor de servicios de autenticación, que será usado el creado por defecto

para la cuenta.

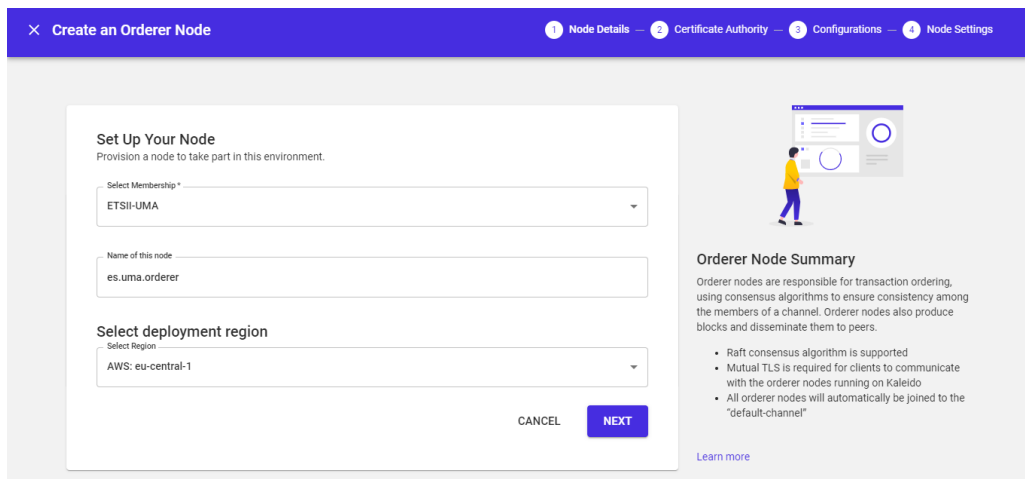


Figura 69: Captura de pantalla de la vista de creación del nodo *orderer*.

Una vez se hayan creado ambos nodos y se hayan inicializado correctamente, la siguiente información (Figura 70) aparecerá en el menú de *Dashboard* de la red.

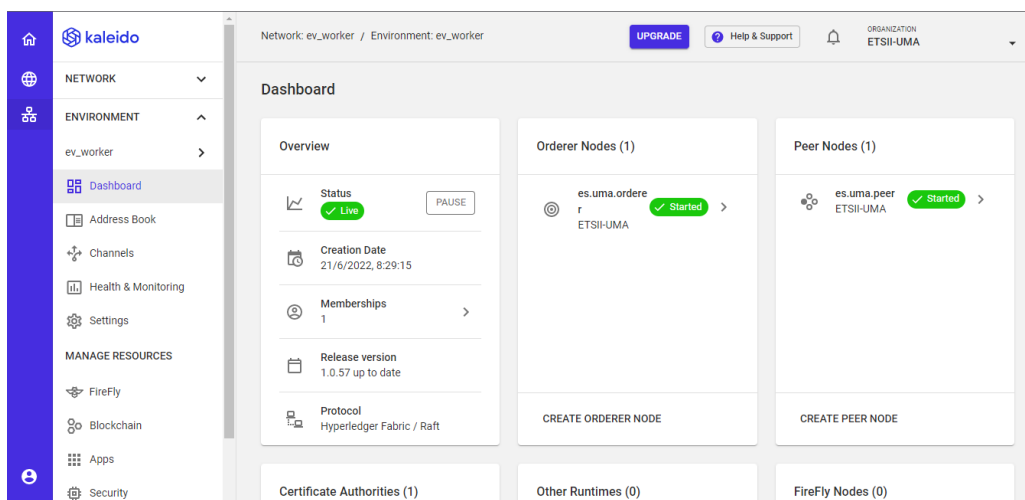


Figura 70: Captura de pantalla del dashboard una vez creados y configurados ambos nodos.

A continuación, es hora de desplegar el contrato inteligente. Para ello es necesario contar el fichero binario que se encuentra en la carpeta *chaincode* del código fuente del proyecto. Para instalar el contrato, es necesario viajar a la vista *Channel*, usando de nuevo el menú de navegación izquierda. Dentro del canal por defecto, se seleccionará *Chaincodes >Promote a Chaincode* (Figura 71).

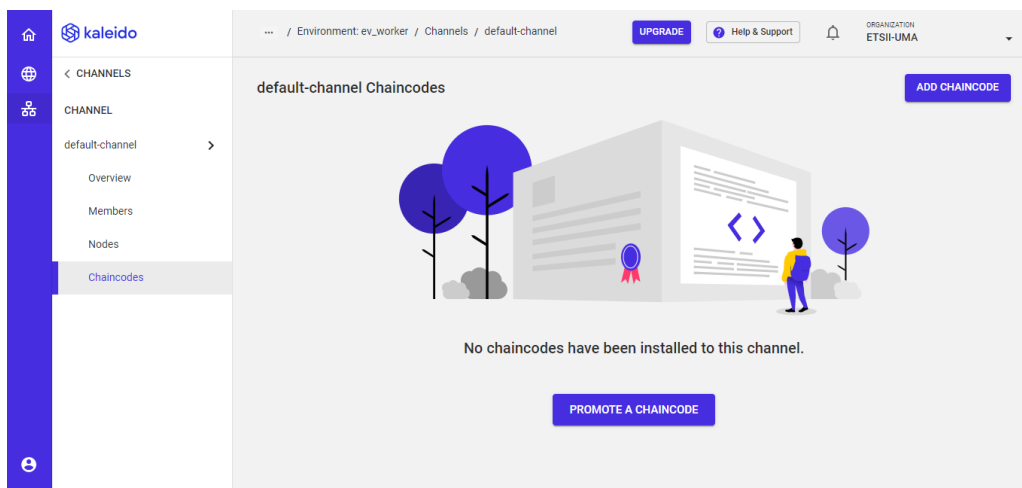


Figura 71: Captura de pantalla del acceso para desplegar el *chaincode*.

Figura 72: Captura de pantalla del formulario de creación del *chaincode*.

Primeramente, hay que crear una aplicación sobre la que instalar el *chaincode*. Se escoge de nuevo *Hyperledger Fabric* y el tipo de código válido ***Golang Executable*** (Figura 72). Una vez creada la app, se añade el contrato haciendo click en *Create New Version* en la nueva página, donde se podrá adjuntar el archivo mencionado anteriormente⁷ (Figura 73).

⁷Si se ha realizado algún cambio en el código fuente del *chaincode*, puede recompilarse usando el comando `go install`.

Enter Chaincode Details
Select the Membership who will own this version and give this specific version a description.

Select Membership *
ETSII-UMA

Description *
1.0.0

Select a file
Select the file to upload

ev_worker.bin **SELECT FILE**

☐ Require initialization before invocation
This adds the "--init-required" parameter to the lifecycle commands, which requires the chaincode to be initialized before it can process transactions.

CANCEL **FINISH**

Your chaincode version
Set the details for

- Owner - the membership you are acting under when uploading this subscription.
- Description - Give this version a description so you will recognize it later.
- Binary file - upload the linux compiled go binary for distribution to nodes in the network

[Learn more](#)

Figura 73: Captura de pantalla de la subida del fichero binario como *chaincode*.

Ahora, solo queda desplegarlo en la red haciendo click en *Promote to environment* y *Deploy instance to channel*. Debe aparecer todo correcto (Figura 74).

ENVIRONMENT

ev_worker

Dashboard

Address Book

Channels

Health & Monitoring

Settings

MANAGE RESOURCES

FireFly

Blockchain

Apps

peer nodes below:

	Owner	Installed Status
monitor	System	✓ Installed
er	ETSII-UMA	✓ Installed

following channels:

Memberships	Nodes	Status	Chaincode Approved	Membership Approvals	Chaincode Committed	Latest Sequence/Version
	2	✓ Live	✓ Yes	System ✓ Yes	✓ Yes	✓ Yes
				ETSII-UMA ✓ Yes		Sequence 1

Rows per page: 10 1-1 of 1

Figura 74: Captura de pantalla con instalación y despliegue correcto del *chaincode*.

Finalmente, para poder añadir usuarios (que son los que estarán disponibles desde el panel de administración), hay que dirigirse a *Address Book >Certificate Authorities >View Certificate Details >Create new identity*. Esto permitirá introducir un nombre para devolver posteriormente una contraseña, que es la que se debe suministrar en la pestaña de administración correspondiente de la aplicación web desarrollada.

Ya se cuenta con todos los componentes de la red activos y correctamente configurados.

B.2. Creación de cuenta en *Firebase*

En este paso, se configurará una cuenta en *Firebase* para poder habilitar tanto el acceso y registro de usuarios en la red como el mantenimiento de la base de datos con las estaciones precargadas en el sistema y administradas desde la aplicación web.

Para ello, es necesario visitar el siguiente [enlace](#). En la pantalla de inicio de la consola, se escoge “Crear un proyecto” (Figura 75), donde hay que otorgarle un nombre común.

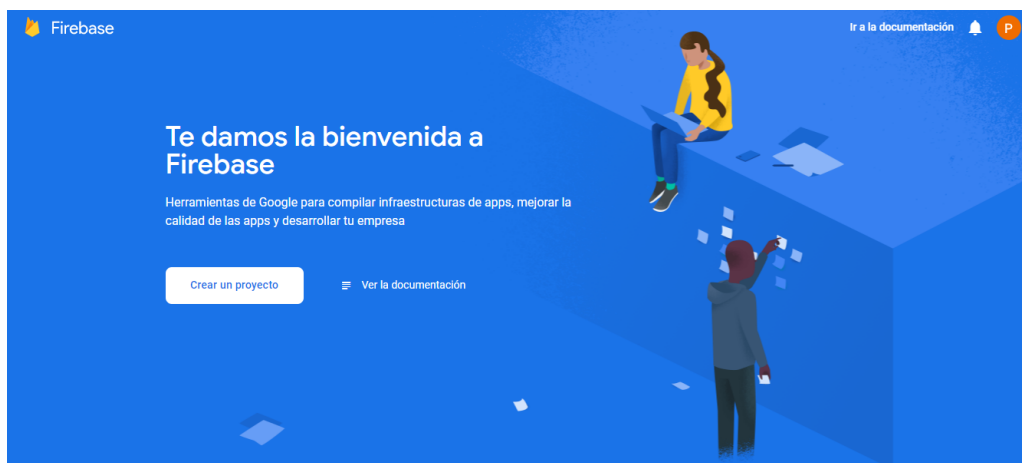


Figura 75: Captura de pantalla de inicio de la consola de *Firebase*.

A continuación, hay que habilitar el módulo de autenticación desde el panel de navegación izquierda, haciendo click en “*Authentication*” (Figura 76), así como el módulo de la base de datos, en “*Firestore Database*” (Figura 77).

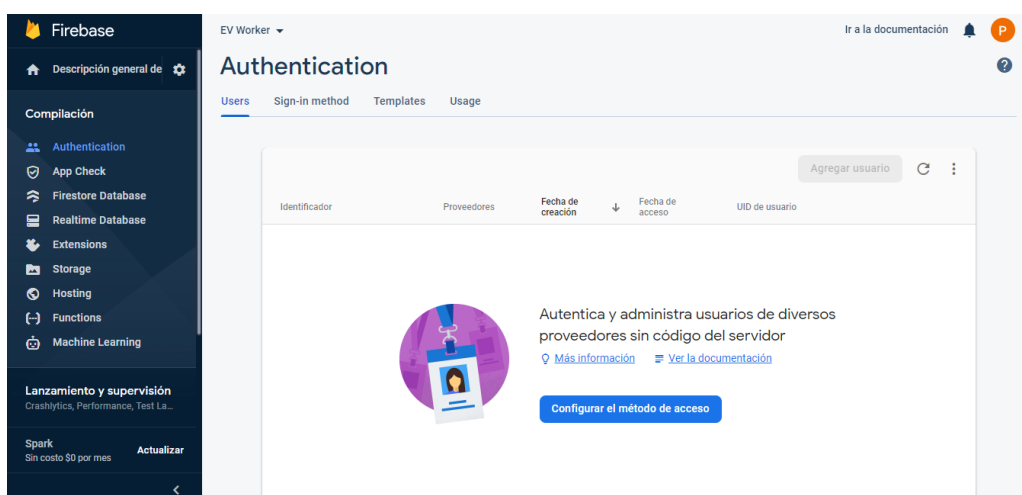


Figura 76: Captura de pantalla del módulo de autenticación habilitado.

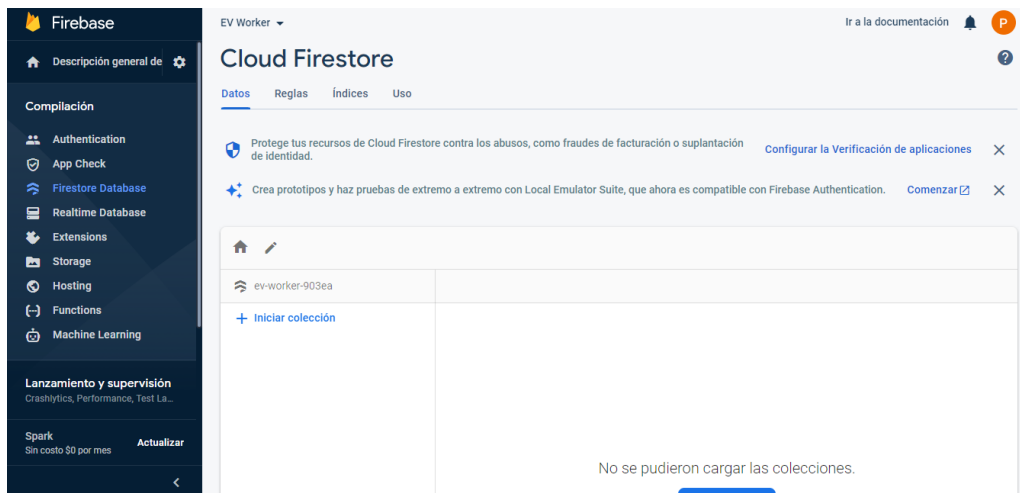


Figura 77: Captura de pantalla del módulo de base de datos habilitado.

En *Firestore Database* únicamente se debe crear una colección con el nombre **estaciones**, haciendo click en “Iniciar colección” (Figura 78).

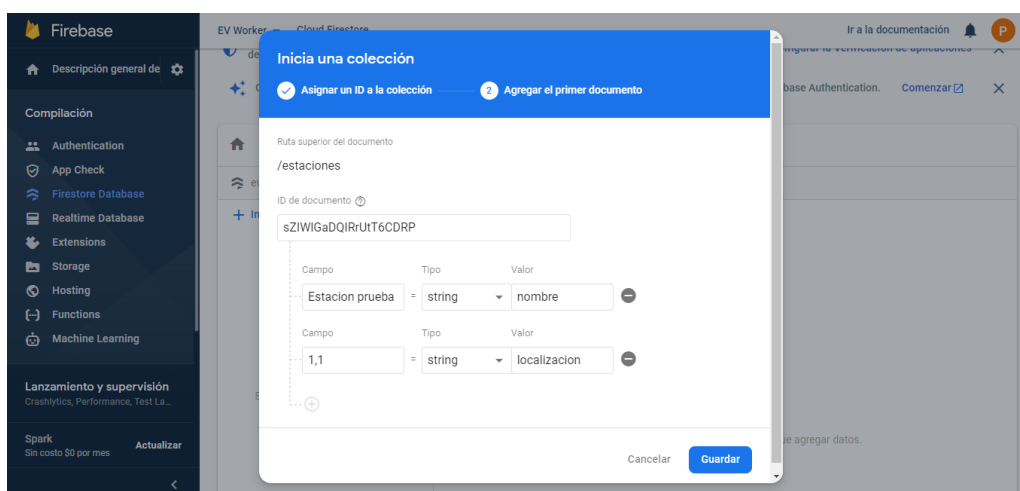


Figura 78: Captura de pantalla de creación de la colección y un valor de prueba inicial.

Ahora, es el turno de conectar el proyecto de *Firebase* a las aplicaciones creadas:

- Para la aplicación móvil, desde la ventana inicial de la consola de *Firebase*, se seleccionará agregar una aplicación *Android* (Figura 79). Hay que indicar el nombre del paquete *Android*, que puede ser consultado en el archivo de manifiesto. En este caso, es `es.uma.evworker`. A partir de ahí, el propio *Firebase* muestra los pasos a dar, teniendo que instalar un archivo JSON con las claves API, y añadir a las dependencias de *gradle* algunos paquetes necesarios.

[illegible]

- Para la aplicación web, igual que antes, se seleccionará la aplicación web para instalar (Figura 80). A continuación, preguntará por un sobrenombre y de nuevo el propio *Firebase* indicará los pasos a dar.

×

Agrega Firebase a tu app web

1

Registrar app

Sobrenombre de la app ⓘ

Mi app web

☐ Configurar **Firebase Hosting** para esta app también.
[Más información](#) ⓘ

Hosting también se puede configurar más tarde. Puedes comenzar en cualquier momento, no hay ningún costo.

Registrar app

2

Agrega el SDK de Firebase

Figura 80: Captura de pantalla del formualrio de conexión con aplicación web.

Con esto, ya se encuentran conectadas ambas aplicaciones con los servicios de *Firebase*.

B.3. Creación de claves API de *Google Maps Platform*

Ambas aplicaciones desarrolladas hacen uso de la plataforma de mapas de *Google*, por lo que es necesario obtener un par de claves API ⁸, que es necesario para poder hacer uso de la visualización de los mapas.

Para ello, se debe iniciar sesión en la consola de *Google Cloud*, desde este [enlace](#) (Figura 81). Hay que desplazarse al apartado de “API y Servicios” del menú de navegación izquierda.

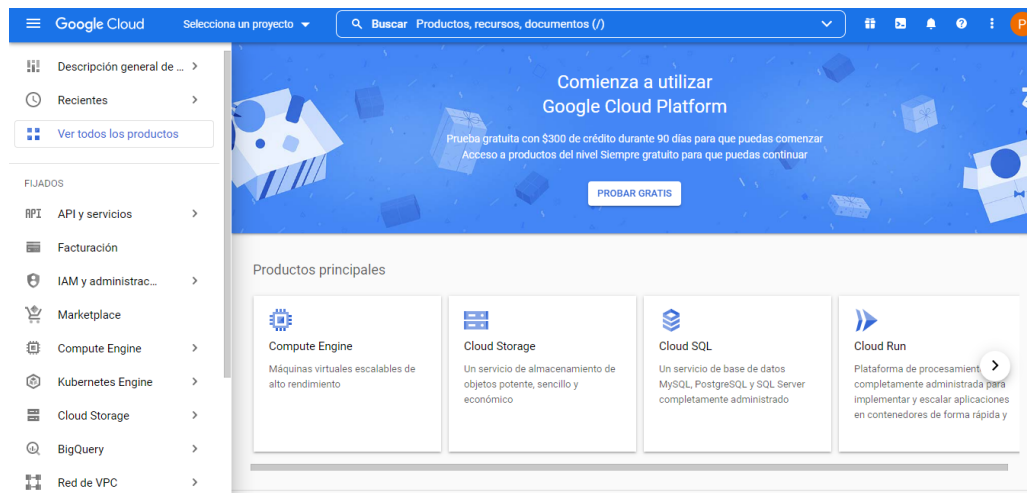


Figura 81: Captura de pantalla de la ventana de inicio de la consola de *Google Cloud*.

En dicho apartado, se selecciona y habilita en “Biblioteca” (Figura 82) las API necesarias para ambas aplicaciones:

- Maps SDK for Android, utilizada a través de la aplicación *Android*.
- Maps JavaScript API, usada por la aplicación Web.

⁸Es recomendable usar una clave distinta para cada aplicación, y limitar su alcance a cada una.

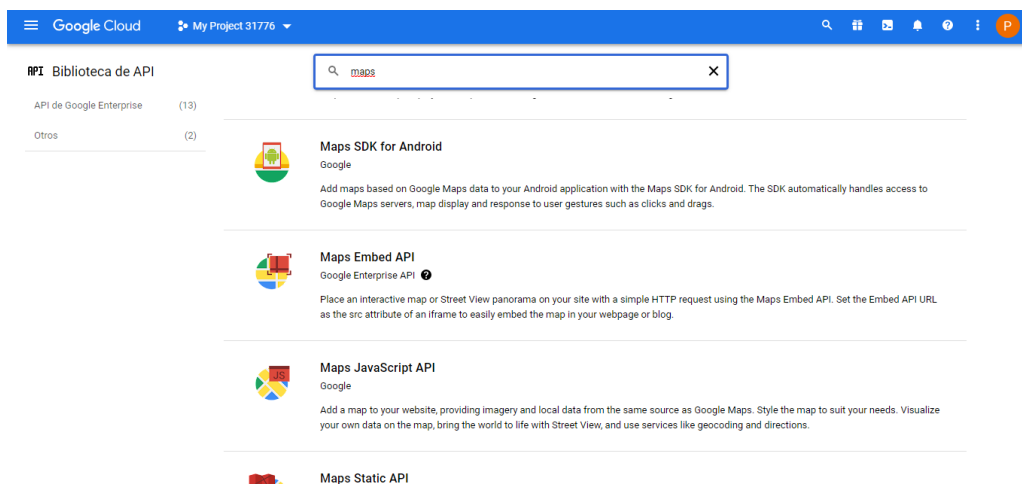


Figura 82: Captura de pantalla de la biblioteca de APIS.

A continuación, en el subapartado de “Credenciales” se podrán crear las dos claves API necesarias (Figura 83). Es de buena práctica de seguridad limitar el uso de cada clave a una única aplicación. Se puede obtener más información de como hacerlo en el siguiente [enlace](#) de la documentación de *Google Maps Platform*.

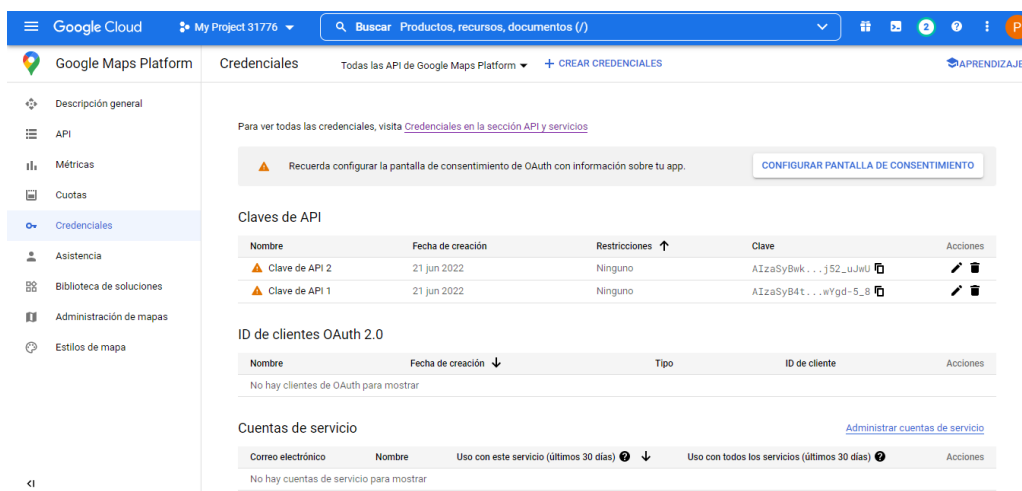


Figura 83: Captura de pantalla del apartado de “Credenciales”, con información de las claves APIS generadas.

B.4. Preparación de las aplicaciones *Android* y web

Una vez que se han realizado todos los pasos previos, ya solo queda terminar de configurar las aplicaciones *Android* y web. En ambas es necesario modificar los siguientes parámetros:

- Dirección URL de conexión con el nodo *peer* de la red *blockchain*
- Token de autenticación para conexión a la URL anterior
- Clave API correspondiente a cada aplicación, obtenida de *Google Maps Platform*

Para modificar los primeros dos parámetros, hay que terminar la clase `MotorHTTP` en ambas aplicaciones, completando el siguiente TODO (Figura 84).

```
MotorHTTP.java
1 package es.uma.evworker.utils;
2
3 /**
4  * Parametros de conexión y autenticación para la app web
5  */
6 public class MotorHTTP {
7
8     //TODO: obtener URL de conexión con el peer de la blockchain y token de autenticación
9     public static final String MAIN_URL = "";
10    public static final String TOKEN = "";
11 }
```

Figura 84: Clase `MotorHTTP` a modificar en ambas aplicaciones.

Para obtener dichos parámetros, se accede a la consola de *Kaleido*, a la información del nodo *peer*, y, posteriormente, a la opción “*Connect app*” (Figura 85). Se debe elegir un nombre para las credenciales, lo que devolverá la siguiente información:

App Cred information		
ID	e0vafibbyf	
Password	_B8N0-w7SPMLEIGwvIAik9aWEfSCkgutb7mPUtO2SNU	
Basic auth	e0vafibbyf:_B8N0-w7SPMLEIGwvIAik9aWEfSCkgutb7mPUtO2SNU	
Authorization header	Authorization: Basic ZTB2YWZpYnliZjpfQjhOMC13N1NQTUxFeEd3dk lBaWs5YVdFZINDaZd1dGI3bVBVdE8yU05V	
Owner	ETSII-UMA	
Created at	21/6/2022, 10:55:22	


REST API Gateway
<https://e0bk6toqr5-e0ar0zldar-connect.de0-aws-ws.kaleido.io/>


Figura 85: Credenciales de la aplicación generadas y URL principal a la que conectarse para acceder a la API REST.

Hay que repetir este proceso para cada aplicación, e incluir dichos valores en la clase `MotorHTTP` de cada una⁹.

Una vez se cuenta con todo esto configurado, solo queda añadir los valores de la API REST de *Google Maps Platform*.

- En la aplicación móvil, es necesario modificar el archivo `local.properties`, añadiendo el valor de la clave API de ***Maps SDK for Android*** al parámetro `MAPS_API_KEY`.
- En la aplicación web, se debe modificar el valor del *script* de la línea 64 en la plantilla `mapa.html`. El valor de `src` debe contener la clave API correspondiente justo después de `key=`.

B.5. Pasos posteriores

En este punto del manual, las aplicaciones ya funcionan correctamente y el sistema se encuentra totalmente sincronizado. Como pasos posteriores, se deben lanzar las aplicaciones para poder usarlas.

Para lanzar la aplicación web, basta con ejecutar la clase `EVWorkerApplication`, que se encarga de lanzar correctamente en el puerto 8080 la aplicación (se puede acceder mediante <http://localhost:8080>).

Para lanzar la aplicación *Android*, se puede compilar la aplicación y extraer el fichero instalable APK (ver más información en este [enlace](#)), o instalarla en modo depuración a través de un cable USB o Wi-Fi directamente con *Android Studio* (ver más información en este [enlace](#)).

⁹En la variable `TOKEN` solo es necesario incluir el valor del token, la cabecera `Authorization: Basic` no es necesaria.

Apéndice C

Casos de Uso en el protocolo OCPP

En este apéndice se recogen los distintos casos de uso que se han tomado en consideración para la realización de este proyecto. Para ello, se ha tomado la especificación del protocolo OCPP [41] en su versión 2.0.1 (disponible desde este [enlace](#)), y se ha realizado un estudio para seleccionar aquellos casos de uso que se enmarquen en un supuesto de gestión o administración por parte de un operario u que, por tanto, son susceptibles a ser notificados por este en el envío de un reporte. En el Cuadro 4 se recoge una clasificación por secciones, como viene dispuesto en la aplicación móvil, el identificador del caso de uso al que corresponde en el estándar, un nombre, una descripción, el mensaje enviado a través del protocolo (si lo hubiese) y la acción que debe realizar el operario en cada supuesto.

Seccion	Caso de Uso	Evento	Descripción	Acción del protocolo	Acción del operario
Seguridad	A05	Actualizar perfil de seguridad de la estación de carga	El operador cambia el perfil de seguridad para incrementar la seguridad	SetVariablesRequest	Elegir perfil escogido (1,2 o 3)
	-	Notificar un evento de seguridad	Permite notificar que ha ocurrido un evento: actualización de software, vaciado de registros...	-	Indicar código de evento y descripción
Configuración	B01	Inicio en frío de la estación de carga	Arranca una CS estando apagada	BootNotificationRequest	Seleccionar si se ha ejecutado
	B11	Resetear un componente de la estación de carga	Reinicia una CS o un EVSE	ResetRequest	Indicar componentes reseteados
	B05	Modificar variables	El operario quiere modificar una variable de la estación de carga	SetVariablesRequest	Elegir variable y nuevo valor
	B10	Migrar a una nueva CSMS	La CS cambia de CSMS	Change Network Config	Indicar nueva CSMS
	-	Modificación red de conexión	Se modifica la conexión por red	-	Indicar detalles del cambio
	-	Instalación de nuevos conectores o EVSE	Se instala en la CS nuevos puntos de carga o nuevos conectores en los existentes	-	Indicar detalles de la instalación
Control remoto	F02	Iniciar transacción remota	Permite al operario ayudar a un conductor que tiene problemas en iniciar una transacción (funciona como external trigger)	remote start()	Indicar si se ha realizado
	F03	Detener transacción remota	Permite al operario ayudar a un conductor que tiene problemas en detener una transacción (funciona como external trigger)	remote stop()	Indicar si se ha realizado
	F05	Desconectar remotamente un conector	Desbloquea un conector que no funciona correctamente tras terminar una transacción en la CS	UnlockConnectorRequest	Indicar si se ha realizado
Eventos	-	Notificar un error/ataque/vulnerabilidad	Permite notificar de algún error/ataque/vulnerabilidad, o daño al equipamiento de la CS	-	Descripción detallada de la notificación

Cuadro 4: Tabla modelo para los casos de uso que un operario puede rellenar en el reporte enviado. Elaboración propia a partir de la especificación del estándar OCPP.

Apéndice D

Listado de variables en el protocolo OCPP

En este apéndice se recogen las distintas variables en uso que se han tomado en consideración para la realización de este proyecto. Para ello, se ha tomado la especificación del protocolo OCPP [41] en su versión 2.0.1 (disponible desde este [enlace](#)), y se ha realizado un estudio para seleccionar aquellas variables que se enmarquen en un supuesto de gestión o administración por parte de un operario y que, por tanto, son susceptibles a ser notificadas por este en el envío de un reporte. Completan la definición del caso de uso de OCPP B05, especificado en el apéndice C. En el Cuadro 5 se indica el grupo en el que se enmarca cada variable, el nombre, descripción y el tipo de dato que acepta (y las unidades en el que se debe introducir).

Variable	Descripción	Dato
<i>HeartBeatInterval</i>	Tiempo entre mensajes Heartbeat	Integer
<i>NetworkProfile</i> <i>ConnectionAttempts</i>	Numero de intentos de conexión que la CS realiza antes de cambiar a otro perfil de seguridad	
<i>OfflineThreshold</i>	Tiempo de espera en el que la CS está offline y debe comunicar su estado a los conectores	
<i>QueueAllMessages</i>	Indica si los mensajes deben encolarse antes de ser enviados a la CSMS	Bool
<i>MessageAttempts</i>	Numero de mensajes que la CS debe enviar a la CSMS después de un fallo	Integer
<i>MessageAttemptInterval</i>	Intervalo entre mensajes que la CS debe enviar a la CSMS después de un fallo	
<i>ResetRetries</i>	Numero de intentos de reseteo que la CS debe ejecutar	
<i>DateTime</i>	Modificar fecha y hora	Date
<i>NtpSource</i>	Para implementación de NTP, elegir conexión DHCP o manual	OptionList
<i>NtpServerUri</i>	Dirección del servidor NTP	String
<i>TimeOffset</i>	Offset del tiempo local con respecto a GMT	
<i>NextTimeOffset</i> <i>TransitionDateTime</i>	Fecha de la próxima actualización del offset	Date
<i>TimeOffsetNextTransition</i>	Offset del siguiente tiempo local con respecto a GMT	String
<i>TimeSource</i>	Configura la opción de tiempo que se desea implementar	SequenceList
<i>TimeZone</i>	Configura zona horaria	String
<i>Identity</i>	Establece el nombre identificador de la CS	
<i>OrganizationName</i>	Nombre de la organización del operario (campo Subject del certificado del cliente)	
<i>AuthEnabled</i>	Indica si existe autorización en cada transacción	Bool
<i>AuthCacheEnabled</i>	Habilita o no caché de autorización	

<i>AuthCacheLifeTime</i>	Tiempo de expiración del token desde su último uso	Integer
<i>AuthCachePolicy</i>	Algoritmo que se usa para el reemplazo de la caché	OptionList
<i>LocalAuthListEnabled</i>	Habilita o no autorización local	Bool
<i>EVConnectionTimeOut</i>	Tiempo de espera hasta considerar que no se ha realizado la conexión correcta entre el EV y el conector	Integer
<i>TxBeforeAcceptedEnabled</i>	La CS puede empezar a funcionar sin tener una respuesta satisfactoria después de su arranque (Boot)	Bool
<i>TxStartPoint</i>	Define cuando la CS empieza una nueva transacción	MemberList
<i>TxStopPoint</i>	Define cuando la CS termina una transacción	
<i>MaxEnergyOnInvalidId</i>	Energía máxima que se suministra cuando se desautoriza un ID	Integer
<i>StopTxOnInvalidId</i>	Indica si la CS puede desautorizar una transacción al recibir un mensaje de rechazo	Bool
<i>StopTxOnEVSideDisconnect</i>	Define si cuando se desconecta el conector, la transacción finaliza	
<i>TariffEnabled</i>	Indica si la tarificación está habilitada	
<i>TariffFallBackEnabled</i>	Mensaje que se muestra al conductor si no existe información tarifaria sobre su vehículo	String
<i>CostEnabled</i>	Indica si el coste total está habilitado	Bool
<i>TotalCostFallBackEnabled</i>	Mensaje que se muestra al conductor si no se ha podido calcular coste total de la tx	String
<i>Currency</i>	Divisa usada en ISO 4217	
<i>MonitoringEnabled</i>	Indica si diagnóstico está habilitado	Bool
<i>OfflineQueuingSeverity</i>	Si la CS está offline, esta encola los mensajes que cumplan cierta severidad, indicada en esta variable	Integer
<i>PhaseRotation</i>	Indica rotación de fase para el equipamiento indicado	String

<i>Capacity</i>	Indica capacidad máxima	Decimal
<i>Power</i>	Limite máximo de energía que el EVSE puede suministrar	
<i>ChargeProtocol</i>	Protocolo de carga seguido por el conector	String
<i>ConnectorType</i>	Tipo de conector	
<i>SmartChargingEnabled</i>	Indica si Smart Charging está habilitado	Bool
<i>ACPhaseSwitchingSupported</i>	Para un EVSE, indica si soporta la selección de uso de fase para la carga	
<i>ExternalControlSignalsEnabled</i>	Indica si la CS debe responder a señales externas de control	
<i>Phases3to1</i>	Indica si la CS soporta cambios en las fases usadas durante la carga	
<i>ReservationEnabled</i>	Indica si las reservas están habilitadas	
<i>NonEvseSpecific</i>	Indica si la CS soporta reservas sin especificar EVSE	
<i>AllowNewSessionsPendingFirmwareUpdate</i>	Indica si la CS debe permitir nuevas sesiones cuando tiene pendiente una actualización de firmware	
<i>DisplayMessageEnabled</i>	Habilita mensajes	

Cuadro 5: Tabla modelo para las variables que un operario puede especificar en el reporte.

Elaboración propia a partir de la especificación del estándar OCPP.

Apéndice E

Documentación de Casos de Uso

En este apéndice se recoge la documentación de los distintos casos de uso propuestos para la implementación de la prueba de concepto. Se enumeran y se documentan uno detrás de otro siguiendo las convenciones y normas de documentación de casos de uso UML [39].

UC-01	Conexión a la red <i>blockchain</i>	
Actores	Operario central, Operario de campo, Administrador	
Descripción	Cualquier acción realizada por cualquier aplicación necesita conectarse a la red <i>blockchain</i> para consultar o enviar información como transacciones.	
Subsistema	Red <i>blockchain</i>	
Precondición	Cualquier aplicación implementada desea conectarse a la red para enviar o recibir información	
Secuencia normal	Pasos	Acción
	1	Una aplicación prepara consulta HTTP GET para realizar a la API de la red
	2	La aplicación envía la petición y espera respuesta de la red
	3	La red <i>blockchain</i> procesa la petición a través del <i>chaincode</i> instalado y envía la respuesta a la aplicación
	4	La aplicación procesa la respuesta recibida y se la muestra al actor
Secuencia alternativa	Pasos	Acción
	3.a	La red no puede procesar correctamente la petición y envía respuesta de error
	4.a	La aplicación muestra el error recibido y las posibles causas de este
Excepciones	Pasos	Acción
	3.a	Si algún campo enviado no es correcto o la red está inactiva, esta no puede procesar la petición. Se debe volver al paso 1 y realizar de nuevo la petición
Postcondición	La respuesta de la red es procesada y se informa al actor	

Cuadro 6: Documentación del Caso de Uso 01: Conexión a la red *blockchain*.

UC-02	Autenticarse	
Actores	Operario central, Operario de campo, Administrador	
Descripción	Cualquier operario que busque acceder a la aplicación debe autenticarse. Para ello usa un nombre de usuario y contraseña proporcionados. El administrador inicia sesión localmente en su computador	
Subsistema	Aplicación <i>Android</i> , Aplicación Web	
Precondición	El operario inicia la aplicación y se encuentra en la pantalla de inicio de sesión.	
Secuencia normal	Pasos	Acción
	1	El operario introduce usuario y contraseña
	2	La aplicación comprueba en el servidor los datos introducidos
	3	La aplicación se conecta a la red como el usuario indicado
Secuencia alternativa	Pasos	Acción
	1.a	El administrador inicia sesión en su computador y accede a la vista de la aplicación Web local.
	3.a	La contraseña o nombre de usuario no corresponden a ningún usuario registrado y habilitado en la red. Vuelve a paso 1
Excepciones	Pasos	Acción
	3.a	Si la contraseña o usuario son incorrectos, se muestra un mensaje y el operario debe reintroducir los datos. El acceso a la aplicación no se concede
Postcondición	El operario accede a todos los contenidos disponibles en la aplicación	

Cuadro 7: Documentación del Caso de Uso 02: Autenticarse.

UC-03	Redactar y enviar reportes	
Actores	Operario central, Operario de campo	
Descripción	Cualquier operario puede redactar un informe a través de un formulario provisto en la aplicación, y enviarlo a la red <i>blockchain</i>	
Subsistema	Aplicación <i>Android</i>	
Precondición	El operario correctamente identificado quiere redactar un informe. Accede a la aplicación y selecciona la actividad de “Añadir reporte”.	
Secuencia normal	Pasos	Acción
	1	El operario visualiza el contenido de todo el formulario preparado.
	2	El operario rellena todos los campos que considere oportunos.
	3	Una vez rellenos y comprobados, pulsa sobre el botón “Enviar reporte”.
	4	La aplicación informa sobre el envío correcto del informe y adjunta ID de la transacción en la red.
Secuencia alternativa	Pasos	Acción
	3.a	El operario ya no desea enviar el reporte. El sistema confirma y le devuelve a la página principal
	4.a	La aplicación informa sobre el envío incorrecto del informe y no se envía nada a la red. Volver a paso 2.
Excepciones	Pasos	Acción
	4.a	La aplicación no puede conectarse a la red. No puede enviarse ningún reporte
Postcondición	El reporte se envía correctamente a la red y se añade a los otros reportes enviados.	
Inclusión	Este Caso de Uso incluye UC-01, de conexión a la red <i>blockchain</i> y UC-02, de autenticación para el uso autorizado de la aplicación.	

Cuadro 8: Documentación del Caso de Uso 03: Redactar y enviar reportes.

UC-04	Listar reportes por usuario	
Actores	Operario central, Operario de campo	
Descripción	El usuario de la aplicación puede listar los reportes que él mismo ha enviado previamente, junto con la información enviada	
Subsistema	Aplicación <i>Android</i>	
Precondición	El operario se autentica en la aplicación y accede a la vista de “Reportes”	
Secuencia normal	Pasos	Acción
	1	El operario accede a la vista correcta de la aplicación
	2	El operario puede visualizar por orden de envío todos los reportes que ha enviado
	3	El operario puede hacer click sobre alguno para visualizar su contenido
Excepciones	Pasos	Acción
	2.a	Si la conexión a la red no es correcta, se informa del error y el operario no puede visualizar ningún reporte.
	2.b	Si el usuario no ha publicado ningún reporte aún, se informa de la situación y el operario no puede visualizar ningún reporte
Postcondición	El operario puede visualizar y conocer todos los reportes que ha enviado	
Inclusión	Este Caso de Uso incluye UC-01, de conexión a la red <i>blockchain</i> y UC-02, de autenticación para el uso autorizado de la aplicación.	

Cuadro 9: Documentación del Caso de Uso 04: Listar reportes por usuario.

UC-05	Listar reportes de todos los usuarios	
Actores	Administrador	
Descripción	El administrador accede a la aplicación para visualizar y filtrar los reportes enviados por todos los operarios a la red	
Subsistema	Aplicación Web	
Precondición	El administrador accede a la aplicación web administrador local	
Secuencia normal	Pasos	Acción
	1	El administrador accede a la vista de <i>Dashboard</i>
	2	Se visualizan todos los reportes y su contenido
Secuencia alternativa	Pasos	Acción
	2.a	El administrador puede filtrar mediante un campo de búsqueda todos los reportes enviados por nombre, fecha o comentario
Excepciones	Pasos	Acción
	2.a	Si la conexión a la red no es correcta, se informa del error y el administrador no puede visualizar ningún reporte.
	2.b	Si no se ha publicado ningún reporte aún, se informa de la situación y el administrador no puede visualizar ningún reporte
Postcondición	El administrador puede puede visualizar y filtrar todos los reportes de la red	
Inclusión	Este Caso de Uso incluye UC-01, de conexión a la red <i>blockchain</i>	

Cuadro 10: Documentación del Caso de Uso 05: Listar reportes de todos los usuarios.

UC-06	Administrar red	
Actores	Administrador	
Descripción	El administrador accede a la aplicación para modificar parámetros de la red	
Subsistema	Aplicación Web	
Precondición	El administrador accede a la aplicación web administrador local	
Secuencia normal	Pasos	Acción
	1	El administrador accede a la vista de <i>Usuarios</i> o <i>Estaciones</i>
	2	El administrador modifica los datos pertinentes con los campos y botones creados para ello
Excepciones	Pasos	Acción
	2.a	Si alguna modificación no se puede realizar, la aplicación informará al administrador sobre el error, y vuelve al paso 2
Postcondición	El administrador ha modificado algún parámetro, administrando así el funcionamiento de la red	
Inclusión	Este Caso de Uso incluye UC-01, de conexión a la red <i>blockchain</i>	

Cuadro 11: Documentación del Caso de Uso 06: Administrar la red.

UC-07	Visualizar reportes	
Actores	Administrador	
Descripción	El administrador busca visualizar en un mapa interactivo los reportes enviados por los operarios	
Subsistema	Aplicación Web	
Precondición	El administrador accede a la aplicación web administrador local	
Secuencia normal	Pasos	Acción
	1	El administrador accede a la vista de <i>Mapa</i>
	2	El administrador visualiza en el mapa los reportes enviados
	3	El administrador puede ver el contenido de cada reporte haciendo click sobre su marca en el mapa
Secuencia alternativa	3.a	El administrador puede filtrar los reportes que se mostrarán en el mapa, con el fin de conseguir una mejor visualización
Excepciones	Pasos	Acción
	2.a	Si no se puede conectar a la red, se mostraría un mensaje de error en la aplicación, y no se mostraría ningún reporte en el mapa
Postcondición	El administrador ha podido visualizar y consultar el contenido de los reportes que contienen información de geolocalización en su contenido	
Inclusión	Este Caso de Uso incluye UC-01, de conexión a la red <i>blockchain</i>	

Cuadro 12: Documentación del Caso de Uso 07: Visualizar reportes.

Apéndice F

Documentación API REST Kaleido

En este apéndice se presenta toda la información relacionada con la API REST proporcionada por *Kaleido* para la comunicación de terceras aplicaciones con los nodos *peer* desplegados en su red. Este apartado es quizás el más importante pues es el punto de enlace entre las aplicaciones desarrolladas y la propia red. Dicha red cuenta con un *chaincode* (más información sobre la estructura y contenidos del contrato inteligente en el Apartado 4.3, en la documentación de la prueba de concepto) instalado, al que se debe llamar cada vez que se desee realizar una transacción o una búsqueda.

El esquema que se sigue en cada comunicación viene representado en la Figura 86.

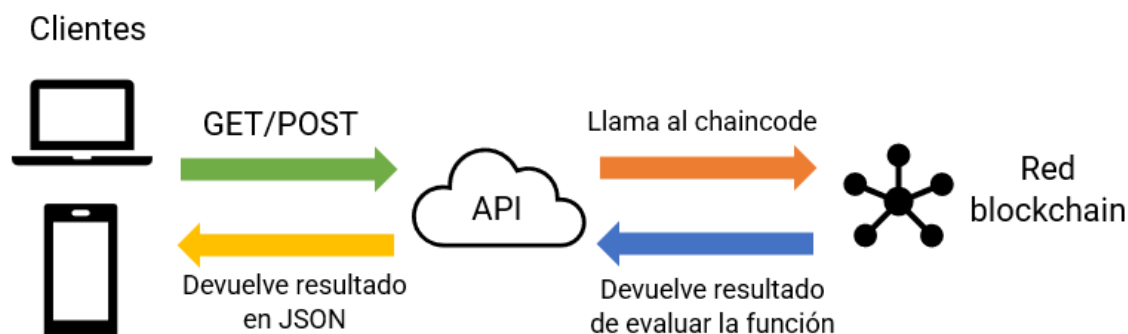


Figura 86: Flujo de comunicación entre los clientes, la API REST y la red de *blockchain*.

La API se encarga de recibir peticiones de parte de los clientes, llamar al chaincode de la red, recibir el resultado de esta y devolverlo al cliente con un formato específico.

Los clientes se encargan de preparar la petición **GET** o **POST** y enviarla a la dirección de la API, proporcionada por Kaleido para cada aplicación. Además, deben recibir la respuesta y procesarla correctamente.

Antes de ver cómo preparar las peticiones, es necesario conocer las rutas que se disponen. Solo se presentarán aquellas de las que se hacen uso por parte de las aplicaciones desarrolladas.

Vienen recogidas en el Cuadro 13.

Tipo	Ruta	Descripción	Parámetros	Cuerpo	Respuesta
GET	/identities	Devuelve identidades registradas en el MSP	-	-	JSONArray con identidades registradas
POST	/query	Realiza una consulta a la red mediante una función del <i>chaincode</i>	-	<pre>{ "headers": { "signer": "string", "channel": "string", "chaincode": "string" }, "func": "string", "args": ["string"], "strongread": true }</pre>	JSONArray con resultados obtenidos
POST	/transactions	Envía una propuesta de transacción para ser incluida en la red	fly-sync	<pre>{ "headers": { "type": "SendTransaction", "signer": "string", "channel": "string", "chaincode": "string" }, "func": "string", "args": ["string"], "init": false }</pre>	Confirmación del envío o error

Cuadro 13: Puntos API usados en el desarrollo de las aplicaciones para la comunicación con la red.

A la hora de realizar una petición, se debe tener en cuenta los siguientes aspectos básicos del protocolo HTTP:

- **URL:** combinación de la URL base de conexión a la API junto con la ruta deseada (se incluye también el flag *fly-sync* en el caso de **transaction**).
- **Cabecera:** en el caso de una petición POST, se incluye la cabecera **Content-Type: application/json**, que indica el formato de la respuesta. Además, hay que añadir el token de autenticación con la cabecera **Authorization: Basic #####**.
- **Cuerpo:** se envían en formato JSON con la especificación indicada en el cuadro anterior, donde “signer” indica el usuario de la red, “channel” y “chaincode” el nombre de ambos elementos, “func” indica el nombre de la función del *chaincode* a la que se desea llamar, y “args” un array con los argumentos que esta requiera (si no tuviese, se deja un array vacío).

F.1. Ejemplos prácticos

Se proponen algunos ejemplos de casos de uso de alguna aplicación a continuación:

1. El usuario “user1” se ha autenticado en la aplicación móvil y quiere listar sus reportes.

La petición que se debe enviar debe ir dirigida a la ruta `query`, y debe contener la siguiente información (se simulará haciendo uso del comando `curl`):

```
curl -X 'POST' \
  'https://e0b6aitkbt-e0ljoqxsxn-connect.de0-aws-ws.kaleido.io/
  query' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Basic #####' \
  -d '{
    "headers": {
      "signer": "user1",
      "channel": "default-channel",
      "chaincode": "app"
    },
    "func": "QueryAllReports",
    "args": [],
    "strongread": true
  }'
```

2. El usuario “user2” se ha autenticado en la aplicación móvil y ha rellenado convenientemente un reporte. En este caso, la petición que se debe enviar debe ir dirigida a la ruta `transactions`, y debe contener la siguiente información:

```
curl -X 'POST' \
  'https://e0b6aitkbt-e0ljoqxsxn-connect.de0-aws-ws.kaleido.io/
  transactions?fly-sync=true' \
-H 'Content-Type: application/json' \
-H 'Authorization: Basic #####' \
-d '{
  "headers": {
    "type": "SendTransaction",
    "signer": "user2",
    "channel": "default-channel",
    "chaincode": "app"
  },
  "func": "CreateReport",
  "args": [
    "Report1-user2","user2","1,1","envio prueba","","","","",""
  ],
  "init": false
}'
```

Apéndice G

Documentación del código fuente del contrato inteligente

En este apéndice se recoge la documentación del código fuente que ha sido desarrollado para la implementación de la prueba de concepto.

El contrato inteligente se ha desplegado en una red *blockchain* basada en *Hyperledger Fabric* por lo que se debe escoger entre los lenguajes *Go*, *Java* o *Node.js*. En este caso, se ha programado haciendo uso de *Go*.

G.1. Dependencias y método ejecutable

En primer lugar, el código define las dependencias a usar en el proyecto, que se realizan con la palabra reservada `import`:

```
package main

import (
    "encoding/json"
    "fmt"
    "time"

    "github.com/hyperledger/fabric-contract-api-go/contractapi"
)
```

Cabe destacar la librería `contractapi`, ya que es la que proporciona los métodos y objetos necesarios para comunicarse con la red y obtener información de ella.

El método ejecutable o función `main` se ejecuta cada vez que se instancia el contrato, y esta llama al método `NewChaincode` de la librería `contractapi`:

```

func main() {

    chaincode, err := contractapi.NewChaincode(new(SmartContract))

    if err != nil {
        fmt.Printf("Error creating chaincode: %s", err.Error())
        return
    }

    if err := chaincode.Start(); err != nil {
        fmt.Printf("Error starting chaincode: %s", err.Error())
    }
}

```

G.2. Estructura de datos

La estructura de datos se encarga de almacenar la información que contiene un reporte. Está compuesta de una clave y un valor, donde el valor corresponde al tipo estructurado `Report`. Se especifica al lado del tipo de cada variable el nombre que lleva en el JSON creado del objeto:

```

// QueryResult structure used for handling result of query
type QueryResult struct {
    Key      string `json:"ReportID"`
    Record   *Report
}

// Report describes what a record can contain
type Report struct {
    Worker      string `json:"worker"`
    Fecha       string `json:"fecha"`
    Localizacion string `json:"localizacion"`
    Comentario  string `json:"comentario"`
    Configuracion string `json:"configuracion"`
    Seguridad   string `json:"seguridad"`
    Remoto      string `json:"remoto"`
    Eventos     string `json:"eventos"`
    Variables   string `json:"variables"`
}

```

G.3. Métodos diseñados

Existen tres métodos diseñados en el contrato inteligente:

1. CreateReport

```
func (s *SmartContract) CreateReport(ctx contractapi.  
    TransactionContextInterface, reportId string, worker string,  
    localizacion string, comentario string,  
    configuracion string, seguridad string, remoto string, eventos  
    string, variables string) error {  
    report := Report{  
        Worker:      worker,  
        Fecha:       time.Now().Format("02-01-2006 15:04:05"),  
        Localizacion: localizacion,  
        Comentario:   comentario,  
        Configuracion: configuracion,  
        Seguridad:    seguridad,  
        Remoto:       remoto,  
        Eventos:      eventos,  
        Variables:    variables,  
    }  
  
    reportAsBytes, _ := json.Marshal(report)  
  
    return ctx.GetStub().PutState(reportId, reportAsBytes)  
}
```

Esta función propone a la red una nueva transacción con un nuevo reporte. Esta tiene como parámetros todos los atributos que el operario ha rellenado en el informe, a excepción de la fecha, que se introduce la correspondiente al sistema en el tiempo en el que se ejecutó el contrato. Se crea el tipo estructurado `Report`, que se transforma a bytes y se envía como transacción con la función `PutState(reportId,reportAsBytes)`.

2. QueryReport

```
// QueryReport return report by given id report
func (s *SmartContract) QueryReport(ctx contractapi.
    TransactionContextInterface, reportId string) (*Report, error) {
    reportAsBytes, err := ctx.GetStub().GetState(reportId)

    if err != nil {
        return nil, fmt.Errorf("Failed to read from world state. %s", err.
            Error())
    }

    if reportAsBytes == nil {
        return nil, fmt.Errorf("%s does not exist", reportId)
    }

    report := new(Report)
    _ = json.Unmarshal(reportAsBytes, report)

    return report, nil
}
```

La función devuelve de la red el reporte requerido (objeto `Report`), pasando como parámetro su identificador. Se hace uso de la función `GetState(reportId)` para obtener una representación en bytes de la información del reporte asociada. Posteriormente, se transforma a un objeto JSON y se devuelve al cliente.

Además, se añade mecanismo de errores para mostrar si ocurre algún problema (error de conexión o reporte no encontrado en la red).

3. QueryAllReports

```
// QueryAllReports returns all reports
func (s *SmartContract) QueryAllReports(ctx contractapi.
    TransactionContextInterface) ([]QueryResult, error) {
    startKey := ""
    endKey := ""

    resultsIterator, err := ctx.GetStub().GetStateByRange(startKey,
        endKey)

    if err != nil {
        return nil, err
    }
    defer resultsIterator.Close()

    results := []QueryResult{}

    for resultsIterator.HasNext() {
        queryResponse, err := resultsIterator.Next()

        if err != nil {
            return nil, err
        }

        report := new(Report)
        _ = json.Unmarshal(queryResponse.Value, report)

        queryResult := QueryResult{Key: queryResponse.Key, Record: report}
        results = append(results, queryResult)
    }

    return results, nil
}
```

Esta última función devuelve todos los reportes almacenados en la red (en un array de objetos `QueryResult`), desde el primero hasta el último.

Para ello, se hace uso de la función `GetStateByRange(startKey, endKey)`. Esto devuel-

ve un *array* con todos los reportes de la red. La función se encarga de iterar sobre este para ir transformando cada uno a un objeto `QueryResult` y añadirlo al *array* que se devolverá finalmente.



UNIVERSIDAD
DE MÁLAGA

| **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga