GOBIERNO DEL PRINCIPADO DE ASTURIAS
CONSEJERÍA DE ECONOMÍA Y EMPLEO

Unión Europea
Fondo Europeo
de Desarrollo Regional

RESULTADOS BCCB

Blockchain for Consumers and Businesses

En Gijón , a 31 de diciembre de 2017

1. **MEMORIA TÉCNICA**

Introducción

El proyecto BCCB (*Empleo de tecnologías de registro distribuido / blockchain en diversas aplicaciones de mercado*) es ejecutado por Fundación CTIC – Centro Tecnológico entre el 1 de enero de 2016 y el 31 de diciembre de 2017.

El objetivo del proyecto BCCB es generar recomendaciones de diseño y aplicación para aplicaciones de mercado basadas en tecnologías de registro distribuido (*Distributed Ledger – Blockchain*), inferidas gracias al desarrollo de dos aplicaciones piloto, enfocadas por un lado al registro de información generada por transacciones originadas por personas (mediante interfaces de usuario), y por otro lado al registro de transacciones originadas por máquinas (mediante dispositivos inteligentes / IoT).

Así, este proyecto se divide en dos bloques principales, el segundo dividido a su vez en dos sub-bloques diferenciados:

- implantación de un sistema de registro distribuido para registrar de forma segura y confiable transacciones críticas
- y diseño de un sistema de captura de información/datos compuesto a su vez de un subsistema de captura (manual) vía software y de un subsistema de captura (automática) vía hardware.

El sistema de registro distribuido interactúa con la plataforma de captura y presentación de datos, y ésta a su vez con usuarios y máquinas.

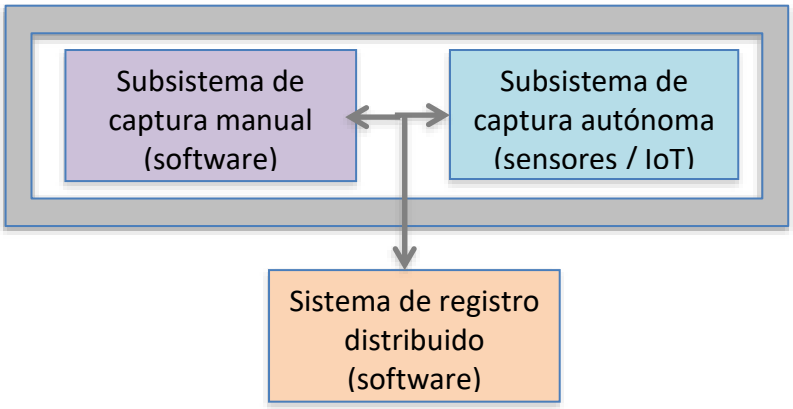


Figura 1. Bloques principales del Proyecto BCCB

La ejecución del proyecto BCCB se divide en cinco (5) hitos claramente diferenciados:

- Hito 1: Diseño del sistema completo
- Hito 2: Implementación del sistema – plataforma de registro distribuido (blockchain)
- Hito 3: Desarrollo y despliegue del sistema de captura manual de datos (software)
- Hito 4: Despliegue del sistema de captura autónoma de datos (hard + soft)
- Hito 5: Adaptación a casos reales mediante la creación de dos aplicaciones piloto

2016

Hito 1: Diseño

El objetivo principal de este hito fue realizar un amplio análisis de la situación actual de las tecnologías y principales plataformas de registro distribuido actualmente existentes y más prometedoras, para seguidamente adoptar la más adecuada para el proyecto BCCB y realizar para ella el diseño de la arquitectura y requisitos/funcionalidades deseados.

Dentro de este hito se identificaron tres (3) tareas principales, y dos de ellas han sido ejecutadas parcialmente en paralelo, y al 100% todas ellas durante el periodo que cubre este informe:

- Tarea 1.1: Análisis plataformas DL existentes (3).
- Tarea 1.2: Análisis de casos de uso DL (vehículos; IoT).
- Tarea 1.3: Diseño arquitectura general, funcionalidades y requisitos (hardware y software).

T1.1. Análisis de plataformas DL

Se identificaron veinte (20) plataformas o productos DL mediante búsquedas en Internet y análisis de numerosos artículos, noticias, así como agendas de eventos y conferencias relacionadas con la temática. Este proceso de vigilancia sistemática de numerosas fuentes de información dio como resultado el siguiente listado de empresas e iniciativas, plataformas y/o productos con tecnologías DL:

Tabla 1. Listado de plataformas DL identificadas

Nombre/URL	**Descripción** (del proveedor)
BigchainDB https://www.bigchaindb.com/	BigchainDB looks, acts and feels like a database with added blockchain characteristics.
Bitcoin https://bitcoin.org	Bitcoin is an innovative payment network and a new kind of money.
Bitproof https://bitproof.io/	Bitproof.io aims to provide everyone a way to protect their files rights.

Blockstack https://blockstack.io/	Provides Infrastructure to build applications quickly and easily on blockchains, also private hosted development platforms.
Blockstream https://blockstream.com/	Blockstream's primary area of innovation is in sidechains. Sidechains allow digital assets to be moved from one blockchain to another.
Chain https://chain.com/	Chain is blockchain infrastructure that enables organizations to build better financial services from the ground up.
Colu http://www.colu.co/	Colu provides a solution for creating local currencies for smart economies using blockchain: the Colored-Coins protocol for digitial assets on top of the Bitcoin blockchain.
Ethereum https://www.ethereum.org/	Ethereum is the first World Computer. It's a decentralized platform that runs smart contracts.
Eris https://erisindustries.com/	An open platform to build, ship, and run blockchain-based applications for business ecosystems.
Factom http://factom.org/	Factom is a service for distributed record storage. Businesses and governments can use it to simplify records management, record business processes, and address security and compliance issues.
Hyperledger Project https://www.hyperledger.org/	An open source collaborative effort created to advance cross-industry blockchain technologies
IBM Blockchain https://developer.ibm.com/blockchain/	Your own private blockchain network on IBM Bluemix
IPFS https://ipfs.io/	A peer-to-peer hypermedia protocol to make the web faster, safer, and more open
MaidSafe http://maidsafe.net/	The SAFE (Secure Access For Everyone) Network is made up of the unused hard drive space, processing power and data connection of its users.
Namecoin http://namecoin.info/	Namecoin is an experimental open-source technology that improves decentralization, security, and speed of components of the Internet such as DNS and identities.

Openchain https://www.openchain.org/	Openchain is an open source distributed ledger technology. It is suited for organizations wishing to issue and manage digital assets in a robust, secure and scalable way.
Ripple https://ripple.com/	Ripple's distributed financial technology enables banks to send real-time international payments across networks.
Stellar https://www.stellar.org/	Stellar is a platform that connects banks, payments systems, and people.
Storj https://storj.io/	Blockchain-based, end-to-end encrypted, distributed object storage, where only you have access to your data.
Tierion https://tierion.com/	Create a verifiable record of any data or business process on the blockchain.

Son numerosas las plataformas DL o productos que inicialmente fueron identificados, por lo que se procedió a la definición de una serie de **criterios de análisis**, que fueron consensuados entre todo el equipo del proyecto BCCB, para elegir acertadamente la mejor plataforma para utilizar en el mismo. Por otro lado, también se tuvo en cuenta que la plataforma a elegir se ajustase al tipo el proyecto que se iba a desarrollar, que no se trata de un proyecto de pura investigación carente de aplicación a corto plazo.

Mediante la lectura de la **información técnica** de instalación y las guías de utilización ofrecidas por las empresas u organizaciones creadoras de las plataformas de blockchain identificadas, y tras realizar pruebas de instalación y ejecución del software ofrecido por algunas de éstas, se concluyó lo siguiente:

- que varias plataformas están enfocadas casi en exclusiva al desarrollo de soluciones para el sector financiero;
- que cada empresa u organización ofrece una plataforma que opera únicamente en una red propia, formada por todos los equipos (nodos) conectados entre sí a través de un software también propio (cliente de la red);
- y que la mayoría de las redes proporcionan una serie de servicios (un protocolo), que, aparte del cliente habitualmente ofrecido (normalmente una *wallet* o "cartera" para gestionar la cuenta de acceso y el saldo de la misma), no ofrecen funciones para la implementación de otro tipo de aplicaciones, lo que implica desarrollar ciertas funciones básicas de interacción con la red que normalmente ya vienen dadas en otro tipo de tecnologías, lo cual es un importante hándicap para cualquier desarrollo ágil. Esto denota la inmadurez de muchas de estas plataformas.

Así, del primer análisis general realizado sobre las plataformas identificadas, se seleccionaron las tres mejores candidatas para el proyecto, al objeto de efectuar un análisis más profundo de las mismas y concluir con una elección final de una de ellas. Estas tres plataformas candidatas fueron: Openchain,

IBM Blockchain (Hyperledger Project), y Ethereum Project.

Tras el análisis y la aplicación de los criterios de evaluación fijados, la plataforma blockchain finalmente elegida para el proyecto BCCB fue **Ethereum Project**.

Teniendo en cuenta las particularidades, características y grado actual de evolución de sus herramientas, se inicia la fase de diseño de la arquitectura y funcionalidades del proyecto BCCB (Tarea 1.3).

T1.2 Análisis de casos de uso DL

Dado que el proyecto busca explorar la aplicación práctica de las tecnologías blockchain en la realidad, buscando acercar lo más posible su utilización en un caso real que ayude a resolver una problemática social o de negocio existente, el equipo del proyecto hace un **brainstorming** para proponer unos primeros casos de uso donde se considera factible aplicar tecnología de blockchain, que estén alineados con los focos de actividad de CTIC Centro Tecnológico, y que puedan convertirse en un caso de uso con potencial aplicación para resolver un problema real, social y/o de negocio.

De este ejercicio surgieron las siguientes primeras propuestas:

- Homogeneización de datos públicos de valor y acceso universal y confiable a los mismos.
- Preservación segura de la información genética animal.
- Trazabilidad de la cadena de distribución de alimentos.
- Autenticación descentralizada de títulos académicos.
- Autenticación descentralizada de certificados púbicos.
- Historial médico interoperable.
- Registro de la distancia recorrida de vehículos y otros parámetros relacionados con sostenibilidad.
- Transporte compartido descentralizado e interoperable.
- Distribución de contenidos de forma confiable en entornos IoT y autenticación de "*things*"
- Registro distribuido de logros deportivos.

También se buscaron e identificaron propuestas de aplicación ya conocidas o divulgadas y casos prácticos o primeras pruebas de aplicación, que se pudieron identificar de manera conjunta al proceso de búsqueda de plataformas DL (T1.1).

Así pues, se recopilan diversos casos de uso desde publicaciones especializadas, portales web, blogs y

redes sociales. Dichos ejemplos se clasificaron por temáticas sectoriales, para mejor organización y claridad, pero también para tener una visión conjunta del alcance que cubren actualmente las soluciones basadas en blockchain para cada sector.

Además de esta información sectorial más tecnológica, se consultaron y recopilaron otro tipo de informaciones que se deben considerar por tener impacto a la hora de trabajar con soluciones de aplicación real, como pueden ser cuestiones de **seguridad**, **privacidad**, y también de **legislación**.

Atendiendo al conocimiento obtenido de la revisión de los casos de uso detectados y propuestos, y los focos de actividad de CTIC, **finalmente se consideraron para incluir en el proyecto BCCB estas dos aplicaciones** o casos de estudio concretos:

- uno para el registro y control de diversos datos relacionados con el **ciclo de vida útil de los vehículos**,
- y otro relacionado con **entornos IoT** para la gestión automatizada del mantenimiento de maquinaria.

Teniendo en cuenta la plataforma DL elegida para el proyecto en la tarea T1.1 y los dos casos de uso elegidos en la tarea T1.2, se pasa a realizar la fase de diseño (T1.3) de todo el sistema necesario para la ejecución del proyecto según los objetivos previstos.

T1.3. Diseño arquitectura general, funcionalidades y requisitos (hard&soft)

Arquitectura general

La arquitectura general del proyecto está formada por dos sistemas principales, uno para la captura de datos o información y el segundo para el registro de la misma. El primer sistema de captura de datos está compuesto por dos subsistemas diferentes según cuál sea el método de captura de datos:

- Sistema de captura de información/datos:
    - Subsistema de captura manual (vía software): captura de información/datos generados por transacciones originadas por personas (mediante interfaces de usuario).
    - Subsistema de captura automática (vía hardware): captura de información/datos generados por transacciones originadas por máquinas (mediante dispositivos inteligentes / IoT).
- Sistema de registro distribuido para registrar de forma segura y confiable transacciones críticas. Este sistema de registro distribuido interactuará con la plataforma de captura y presentación de datos para que ésta lo haga a su vez con usuarios y máquinas.

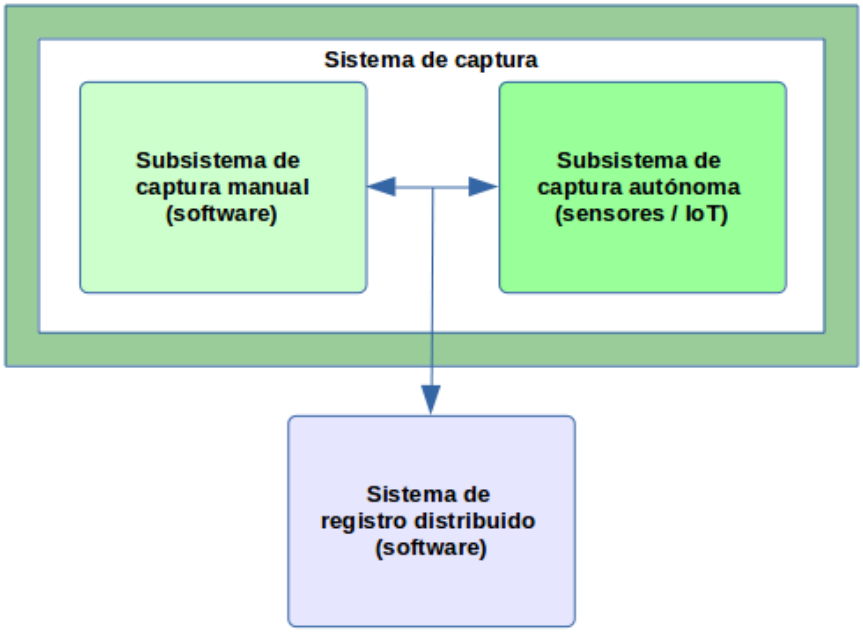


Figura 1. Arquitectura general

Sistema de registro distribuido

Este sistema es el que empleará las tecnologías de registro distribuido (*Distributed Ledger – Blockchain*) objeto de estudio del proyecto. La plataforma basada en tecnología blockchain que se ha seleccionado para este proyecto es la de Ethereum. Esta plataforma no sólo permite manejar transacciones (almacenar información), sino que mediante el uso de entidades denominadas *smart contracts* o contratos inteligentes permite crear el código de aplicaciones distribuidas de uso genérico.

En la red Ethereum, al tratarse de una red distribuida, todos los nodos son iguales entre sí sin haber nodos más importantes que otros. Aunque un nodo falle el resto de nodos siguen estando interconectados entre sí haciendo a este tipo de redes más resistente a fallos y ataques que las redes centralizadas o que las redes descentralizadas que no llegan a ser realmente distribuidas.

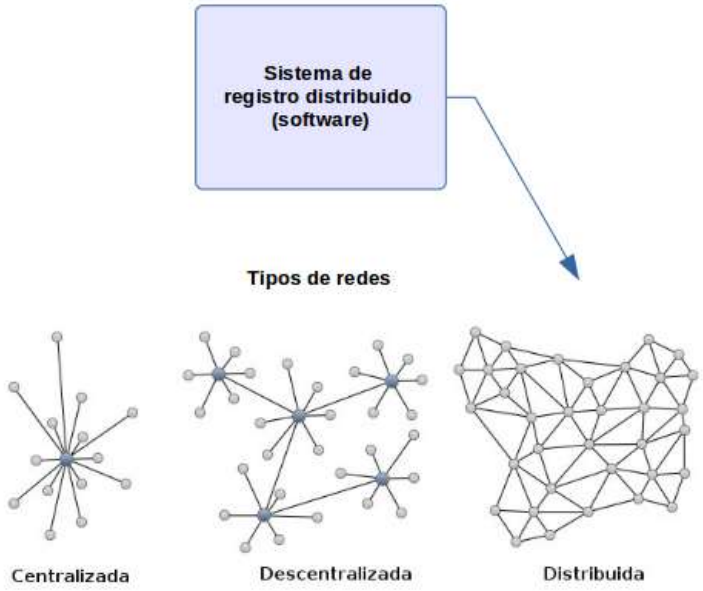


Figura 2. Diferentes tipos de redes. El sistema de registro opera sobre una red distribuida.

Además, en un sistema de registro distribuido basado en blockchain, como en el caso de la red Ethereum, si un nodo falla tampoco se produce pérdida de información ya que ésta se encuentra replicada en todos los nodos. Cada nodo contiene una copia de todo el blockchain, es decir, de todo el registro y por tanto de todos los datos.

Una particularidad de la red Ethereum, fundamental para el desarrollo del proyecto, es que ésta no sólo es capaz de almacenar datos o información, sino que también puede almacenar código y ejecutarlo: los denominados contratos inteligentes o *smart contracts*. De esta forma la red de Ethereum sirve como plataforma para la creación de aplicaciones distribuidas, o aplicaciones que se almacenan y ejecutan desde cualquier nodo y no desde un único servidor central. Los contratos serían las piezas de código que se encargan de gestionar la lógica de negocio de las aplicaciones.

Cada nodo no sólo contiene la cadena de bloques, almacenando datos y código, sino que además también dispone de una máquina virtual (EVM, Ethereum Virtual Machine) para poder ejecutar el código de los contratos inteligentes.

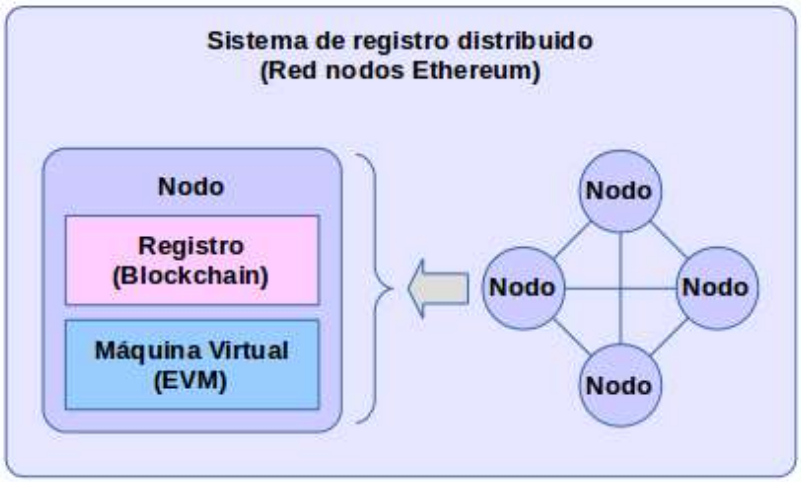


Figura 3. Cada nodo del sistema de registro distribuido contiene toda la cadena de bloques (datos y código) y una máquina virtual para ejecutar el código de los contratos inteligentes.

Entrando algo más en detalle, los contratos son un tipo de cuenta en la red Ethereum. Existen dos tipos de cuentas, por un lado, los contratos inteligentes y por otro las cuentas de usuario:

- Externally Owned Accounts (EOAs): cuentas controladas por una clave privada y quien posea dicha clave (normalmente un usuario final) tendrá la capacidad de realizar transacciones.
- Contratos: cuentas que tienen su propio código, y que son controladas por dicho código.

En la cadena de bloques se almacenará, por tanto, las cuentas de usuario, los contratos, los datos controlados por los contratos y todas las transacciones realizadas en la red.

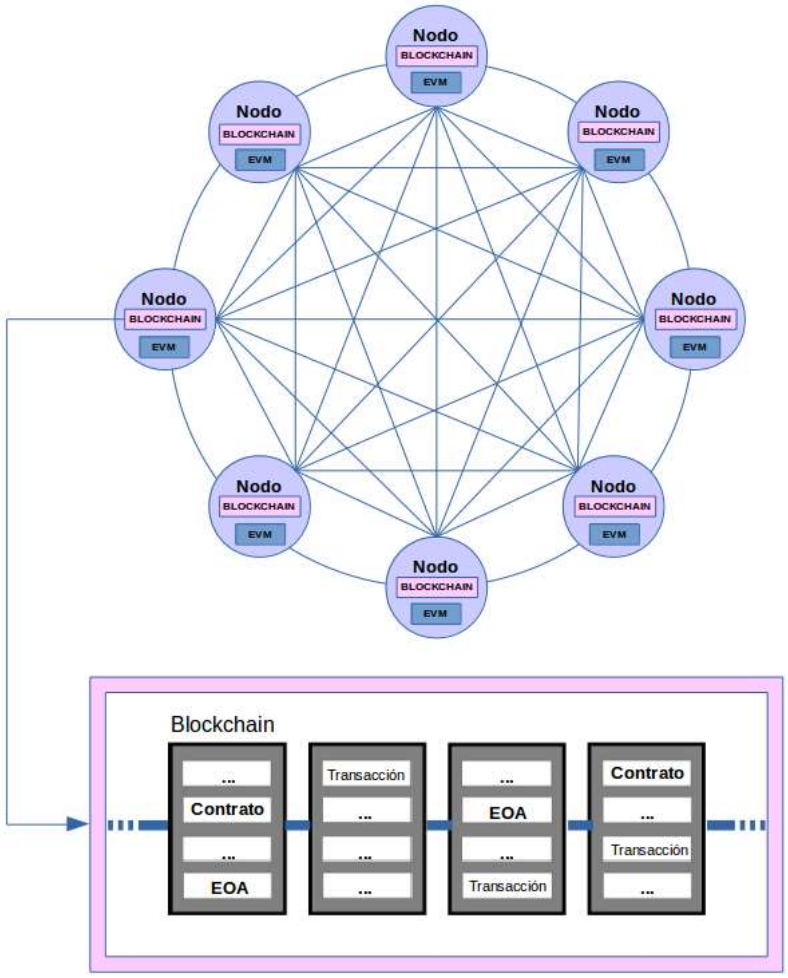


Figura 4. En la cadena de bloques de cada nodo se almacenan todas las cuentas de usuario, todos los contratos y todas las transacciones realizadas

Sistema de captura

El sistema de captura de datos, como se ha comentado, está formado por dos subsistemas diferentes según cuál sea el método de captura de datos: un subsistema de captura manual donde las personas introducen los datos mediante un interfaz de usuario y un subsistema de captura automática mediante dispositivos inteligentes (IoT).

Ambos subsistemas se comunican con el sistema de registro distribuido a través de un interfaz (API) con el nodo local empleado por el sistema. Podemos considerar así que el sistema de registro distribuido está dividido en dos partes: una local con el nodo controlado por la aplicación del proyecto y otra remota con el resto de nodos de la red.

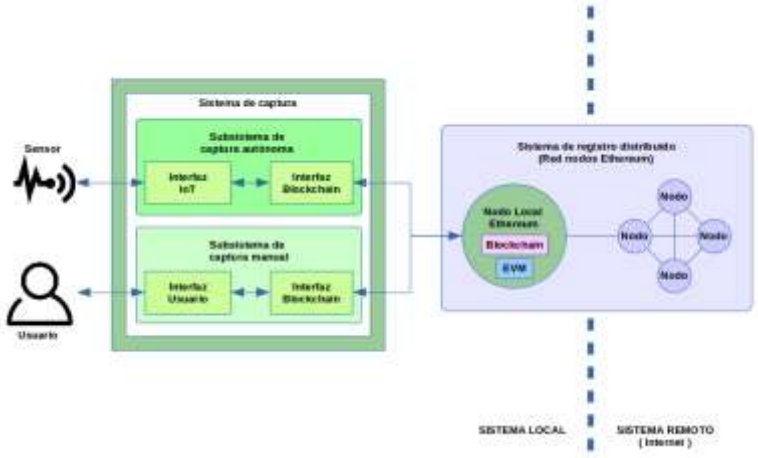


Figura 5. Esquema general del sistema de captura y su interacción con el registro distribuido

Sin embargo, y en la práctica, al realizar las pruebas iniciales para montar la infraestructura necesaria de los subsistemas de captura según el esquema anterior **se han encontrado limitaciones técnicas** que sugieren variar esta arquitectura inicialmente planteada para que sea técnicamente viable.

Subsistema de captura manual

En la arquitectura inicial, el equipo local contiene tanto el subsistema de captura manual como el nodo local con el que se comunica dicho sistema de captura y que hace de intermediario con el resto de la red de Ethereum.

Si bien esta arquitectura es la ideal y es técnicamente posible, la necesidad de que el equipo local contenga toda la información almacenada en la cadena de bloques aumenta considerablemente los requisitos hardware y software del sistema local, tanto en capacidad de almacenamiento, como en capacidad de cómputo y en conectividad.

Estos elevados requisitos suponen una barrera de acceso importante si se desea que el sistema sea empleado por usuarios finales de forma transparente y sencilla. Con ese objetivo de mejora de la usabilidad del sistema **se plantea una arquitectura alternativa empleando el modelo de *cliente ligero***. En este modelo el nodo de Ethereum se elimina del equipo local, que únicamente contendrá el sistema de captura, y se lleva a un servidor que realizará toda la carga pesada de trabajo y actuará de intermediario con el resto de la red Ethereum.

Subsistema de captura autónoma

La arquitectura inicial del subsistema de captura autónoma es similar al subsistema manual, donde el equipo local contiene el nodo Ethereum con el que se comunica el sistema de captura.

En este caso, al tratarse de un sistema autónomo, los requisitos relativos a la usabilidad del sistema no tienen carácter crítico. Sin embargo, durante las primeras pruebas de concepto se encontraron dificultades de carácter técnico. Si bien se consiguió **su implementación en un sistema** de pruebas ya

disponible y **basado en un microcomputador de placa reducida** (SBC, Single Board Computer) de bajo coste, una Raspberry Pi 3 Modelo B, los altos requisitos de hardware necesarios para ejecutar un nodo Ethereum de forma local lo hicieron, desde un punto de vista técnico, **no inviable pero sí escasamente práctico**. La capacidad de cómputo necesaria y la necesidad de conexión ininterrumpida y sincronización continua con la red Ethereum, o re-sincronización en caso de pérdida de conexión, fueron las principales limitaciones técnicas.

Para salvar estas limitaciones, **se optará por un modelo en el que el equipo IoT (Rasperry Pi) actúe de forma remota** realizando únicamente las tareas de captura de datos (sensórica) y envío de dicha información a través de la red **hasta el equipo con la aplicación y el nodo local**. A este equipo se le denomina "***IoT Gateway***[1]" en el ecosistema del IoT.

Funcionalidades

Tal y como se señalaba anteriormente, el sistema debe registrar de forma segura y confiable los datos capturados. Al basarse en una arquitectura distribuida, ha de disponer de tecnologías y protocolos de comunicación entre los nodos de la red y el almacenamiento de los datos y la información se ha de realizar de forma distribuida por todos los nodos.

Asimismo, para dar libertad en el tipo de transacciones y operaciones que se pueden realizar en el sistema, éste no ha de estar limitado a un conjunto de transacciones predefinidas sino que debe permitir la ejecución de código (contratos inteligentes) de forma que el sistema sirva como plataforma de aplicaciones distribuidas. También ha de aportar mecanismos para maximizar la seguridad del sistema, como el uso de criptografía para el firmado de transacciones o sistemas de consenso para la validación de cualquier transacción antes ser registrada en el sistema.

A nivel no funcional se han identificado tanto requisitos técnicos como requisitos de usuario. A nivel técnico destacan principalmente dos requisitos de alta prioridad: seguridad (RT1) e inmutabilidad (RT2) de los datos. Ambos requisitos son necesarios para desarrollar un sistema que funcione con la fiabilidad necesaria para que los usuarios confíen tanto en el sistema como en la veracidad e integridad de los datos almacenados en el mismo.

Los requisitos de usuario cobran gran importancia para minimizar las barreras técnicas y de usabilidad que puedan disuadir del uso del sistema. Para ello, se ha trabajado en detalle en este aspecto identificando todos los requisitos de usuario que se pueden resumir en la búsqueda de un sistema que sea fácil de usar y no invasivo. En la misma línea, también destacan los requisitos para aumentar la confianza en el sistema, como la transparencia de su funcionamiento y el carácter público de la información almacenada.

[1] Intel Developer Zone – IoT Documentarion - What Is the Gateway and Why Should I Care?
https://software.intel.com/en-us/articles/what-is-the-gateway-and-why-should-i-care

Por otra parte, el sistema de captura debe de proporcionar la funcionalidad necesaria para permitir al usuario la introducción de datos y la consulta de la información ya registrada en el sistema. De igual forma, el sistema también debe poder capturar y registrar datos de forma automática desde sensores u otros dispositivos de IoT.

Igual que en el caso del sistema de registro, para el sistema de captura también se ha llevado a cabo el análisis de los requisitos no funcionales desde el punto de vista técnico y desde el punto de vista de los usuarios.

Todo el conjunto de requisitos recolectados durante la ejecución del Hito 1 han sido identificados por dos medios principales:

- *Análisis del estado del arte*. Se han analizado las principales referencias científicas encontradas y relacionadas con el estudio del tipo de sistemas objetivo de este proyecto.
- *Análisis de aplicaciones piloto existentes*. Se han analizado diversas aplicaciones distribuidas (Dapps) relacionadas con el registro y almacenamiento confiable de información crítica.

Requisitos hardware y software

Dado que la arquitectura general está compuesta por varios subsistemas diferentes interrelacionados entre sí, pero potencialmente independientes en cuanto a requisitos hardware y software necesario, no se pueden establecer unos requisitos mínimos únicos, sino que será necesario diferenciarlos para cada caso.

Podemos identificar los siguientes sistemas para cubrir las diferentes arquitecturas y configuraciones planteadas:

- Sistema de captura de datos (sin nodo)
- Sistema de captura de datos (con nodo)

En cuanto a **requisitos de software para un cliente sin nodo** (sistema de captura de datos sin nodo) el principal es disponer del framework Meteor[2], empleado para la creación de aplicaciones distribuidas. Meteor es una plataforma para crear aplicaciones web en tiempo real construida sobre Node.js. Se trata de un framework *full-stack* cuya principal ventaja es la incorporación de todas las herramientas necesarias para la creación de aplicaciones web SPA (Single Page Applications) reactivas. Se trata de la tecnología principal para la creación de dapps o aplicaciones distribuidas en la red de Ethereum. Adicionalmente, para permitir la funcionalidad de los clientes ligeros, es necesario un navegador que permita el acceso a nodos Ethereum remotos, bien mediante el propio navegador como es el caso de Mist (navegador de Ethereum) o bien mediante extensiones de otros navegadores (p. ej. Metamask para Chrome).

[2] Meteor https://www.meteor.com/

En cuanto a los **requisitos de hardware para un cliente sin nodo**, y teniendo en cuenta lo anterior, estos serán los necesarios para poder ejecutar un navegador web moderno con amplio soporte de tecnologías actuales como HTML5 (requisitos propios del framework Meteor). Así, los requisitos mínimos identificados son los requeridos por el navegador Chrome sobre un sistema operativo Windows 7, o similar.

Por otro lado, los **requisitos de software para un cliente con nodo** (sistema de captura de datos con nodo) son los mismos que para un cliente sin nodo, pero añadiendo la necesidad de un cliente de la red Ethereum, como Geth[3] (cliente principal de la red Ethereum implementado en el lenguaje Go).

Los **requisitos hardware para un cliente con nodo** aumentan para poder ejecutar dicho cliente (requisitos de CPU y Memoria), así como para almacenar toda la cadena de bloques (requisito de almacenamiento).

Otros requisitos

Además de los requisitos de hardware y software mencionados, existen otros requisitos de naturaleza diferente debidos al funcionamiento de la plataforma de aplicaciones descentralizadas Ethereum.

Esta red blockchain dispone de su propio token criptográfico (moneda/divisa digital) denominado Ether[4]. Esta moneda, que tiene un valor real y hasta puede intercambiarse por dinero fiduciario según la cotización en cada momento, se emplea en la plataforma como método de pago en transacciones y para pagar las cuotas de carga computacional (gas) necesaria para realizar transacciones. Toda transacción tiene un coste computacional y por tanto requiere cierta cantidad de gas el cual tiene un valor que ha de pagarse con Ether.

Como consecuencia de lo anterior, para poder trabajar sobre la red de Ethereum es necesario disponer de al menos una cantidad mínima de Ethers, que puede conseguirse de alguna de estas dos formas:

- Cambiando moneda fiduciaria, como Euros, por Ethers. Es decir, comprar monedas digitales con dinero real según la cotización.
- Creando Ethers (minado) desde la propia red Ethereum. Sin entrar en detalles técnicos, la minería es el proceso por el cual un equipo conectado a la red realiza el trabajo de cómputo necesario para que funcione el sistema (validar transacciones, generación de bloques, etc.) y recibe Ether a modo de recompensa por el trabajo aportado. El trabajo necesario para conseguir Ether es muy intensivo y costoso desde el punto de vista computacional, requiriendo de elevados recursos en cuanto a hardware.

En cualquiera de los dos casos, es necesaria una inversión con un coste económico para la obtención de Ether, bien mediante compra directa con dinero, o bien mediante la inversión inicial en equipos de minería y en gasto energético (consumo eléctrico) causado por el proceso computacional.

[3] Go Ethereum (Geth) https://geth.ethereum.org/

[4] Ethereum Project. What is Ether? https://www.ethereum.org/ether

Para el proyecto BCCB optaremos inicialmente por efectuar el proceso de minado, utilizando para ello si es posible el equipo Gateway IoT propuesto.

Hito 2: Implementación del sistema – plataforma de registro distribuido (blockchain)

T2.1. Despliegue de red BlockChain

Para proceder al despliegue de una red blockchain, en nuestro caso basada en la plataforma DL ofrecida por Ethereum Project, es necesario primeramente **instalar un software cliente** de la red (cliente Ethereum) en todos los equipos que vayan a formar parte de la red como nuevos nodos de la misma. Después hay que proceder a la **sincronización de estos nodos con la red Ethereum**, proceder a la **creación de cuentas**, y dotarlas de saldo para poder realizar operaciones tales como la **creación de contratos inteligentes**, o la ejecución de los mismos y los procesos asociados a las **aplicaciones (Đapp) de usuario final**.

Despliegue de cliente Geth

Se realizó la instalación del cliente denominado Geth (seleccionado entre los disponibles por su popularidad y documentación disponible) en todos los ordenadores (3 equipos) empleados en el entorno de desarrollo (con sistema operativo Linux) asignado al proyecto BCCB, y se procedió a la sincronización de dichos equipos con la red blockchain de Ethereum, convirtiéndose así en nodos de la red Ethereum, tanto de la principal o de "producción" como de la red de test utilizada para pruebas.

Sincronización con la plataforma Ethereum

Con el objeto de acelerar el proceso de sincronización inicial con la red principal se usaron unos parámetros en cada nodo (el tag –fast) que hace que no se conserven las transacciones pasadas.

Aún así, el proceso de sincronización de los nodos resultó bastante lento y en muchos casos bloqueante, impidiendo a veces el uso de la conexión a Internet e incluso el uso de cualquier aplicación, sobre todo en aquellos equipos con menor capacidad de proceso.

Creación de cuentas

Empleando el cliente Geth, se creó una cuenta de usuario, del tipo *externally owned accounts*, en la plataforma Ethereum.

Con el fin de poder realizar pruebas de la aplicación sin consumir Ethers reales, se decidió hacer uso de la red blockchain de prueba, en la que se lanzó un proceso de minería de Ether que permitió acumular suficientes recursos para los siguientes pasos a realizar.

Publicación de contratos inteligentes

Se procedió a publicar y ejecutar en la plataforma un primer contrato inteligente (smart contract) de prueba, lo cual se trató de realizar usando el cliente Geth, aunque no fue posible, ni tampoco empleando el entorno de desarrollo integrado (IDE) proporcionado por la plataforma Mix (Mix IDE). Finalmente se recurrió a otro cliente como es la Wallet Ethereum, que instalada en uno de los nodos permitió publica el primer contrato con éxito, y observar los resultados de su ejecución.

Con esto se dio por completada y probada la interacción mínima necesaria con la plataforma de registro distribuido (DL) desde los equipos (nodos) de desarrollo asignados al proyecto BCCB, un paso fundamental para poder continuar desplegando el resto de sistemas que componen el proyecto. Así **las actividades pendientes tanto de la tarea T2.1 como de la tarea T2.2 dentro de este Hito 2 seguirán su curso en los primeros meses del año 2017**.

2017

Hito 1: Diseño

El objetivo principal de este hito fue realizar un amplio análisis de la situación actual de las tecnologías y principales plataformas de registro distribuido actualmente existentes y más prometedoras, para seguidamente adoptar la más adecuada para el proyecto BCCB y realizar para ella el diseño de la arquitectura y requisitos/funcionalidades deseados.

Dentro de este hito se identificaron tres (3) tareas principales, y dos de ellas han sido ejecutadas parcialmente en paralelo, y al 100% todas ellas durante el periodo que cubre este informe:

- Tarea 1.1: Análisis plataformas DL existentes (3). (2016)
- Tarea 1.2: Análisis de casos de uso DL (vehículos; IoT). (2016)
- Tarea 1.3: Diseño arquitectura general, funcionalidades y requisitos (hardware y software). (2016 y 2017)

T1.3. Diseño arquitectura general, funcionalidades y requisitos (hard&soft)

Arquitectura general

La arquitectura general del proyecto está formada por dos sistemas principales, uno para la captura de datos o información y el segundo para el registro de la misma. El primer sistema de captura de datos está compuesto por dos subsistemas diferentes según cuál sea el método de captura de datos:

- Sistema de captura de información/datos:
  - Subsistema de captura manual (vía software): captura de información/datos generados por transacciones originadas por personas (mediante interfaces de usuario).
  - Subsistema de captura automática (vía hardware): captura de información/datos generados por transacciones originadas por máquinas (mediante dispositivos inteligentes / IoT).
- Sistema de registro distribuido para registrar de forma segura y confiable transacciones críticas. Este sistema de registro distribuido interactuará con la plataforma de captura y presentación de datos para que ésta lo haga a su vez con usuarios y máquinas.

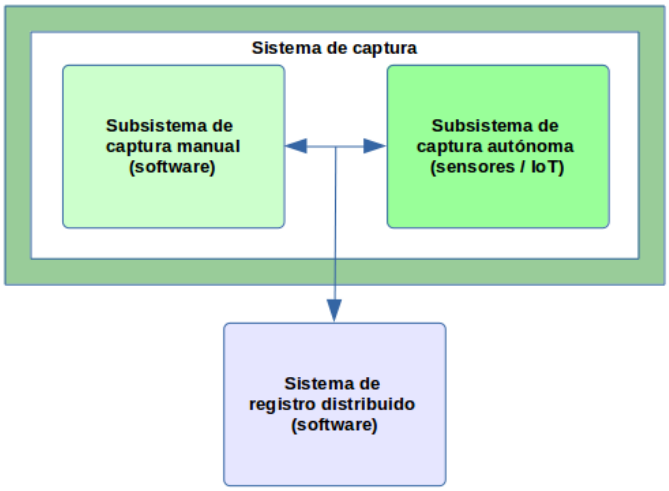


Figura 3. Arquitectura general

Sistema de registro distribuido

Este sistema es el que emplea las tecnologías de registro distribuido (*Distributed Ledger – Blockchain*) objeto de estudio del proyecto. La plataforma basada en tecnología blockchain que se ha seleccionado para este proyecto es la de Ethereum. Esta plataforma no sólo permite manejar transacciones (almacenar información), sino que mediante el uso de entidades denominadas *smart contracts* o contratos inteligentes permite crear el código de aplicaciones distribuidas de uso genérico.

En la red Ethereum, al tratarse de una red distribuida, todos los nodos son iguales entre sí sin haber nodos más importantes que otros. Aunque un nodo falle el resto de nodos siguen estando interconectados entre sí haciendo a este tipo de redes más resistente a fallos y ataques que las redes centralizadas o que las redes descentralizadas que no llegan a ser realmente distribuidas.

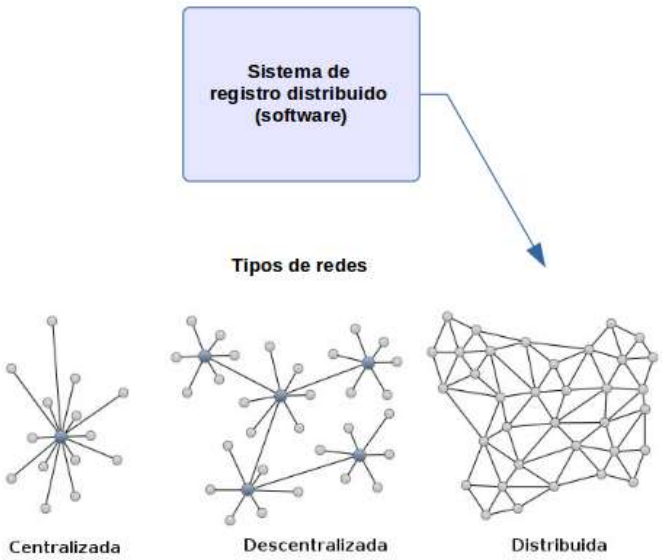


Figura 4. Diferentes tipos de redes. El sistema de registro opera sobre una red distribuida.

Además, en un sistema de registro distribuido basado en blockchain, como en el caso de la red Ethereum, si un nodo falla tampoco se produce pérdida de información ya que ésta se encuentra replicada en todos los nodos. Cada nodo contiene una copia de todo el blockchain, es decir, de todo el registro y por tanto de todos los datos.

Una particularidad de la red Ethereum, fundamental para el desarrollo del proyecto, es que ésta no sólo es capaz de almacenar datos o información, sino que también puede almacenar código y ejecutarlo: los denominados contratos inteligentes o *smart contracts*. De esta forma la red de Ethereum sirve como plataforma para la creación de aplicaciones distribuidas, o aplicaciones que se almacenan y ejecutan desde cualquier nodo y no desde un único servidor central. Los contratos serían las piezas de código que se encargan de gestionar la lógica de negocio de las aplicaciones.

Cada nodo no sólo contiene la cadena de bloques, almacenando datos y código, sino que además también dispone de una máquina virtual (EVM, Ethereum Virtual Machine) para poder ejecutar el código de los contratos inteligentes.

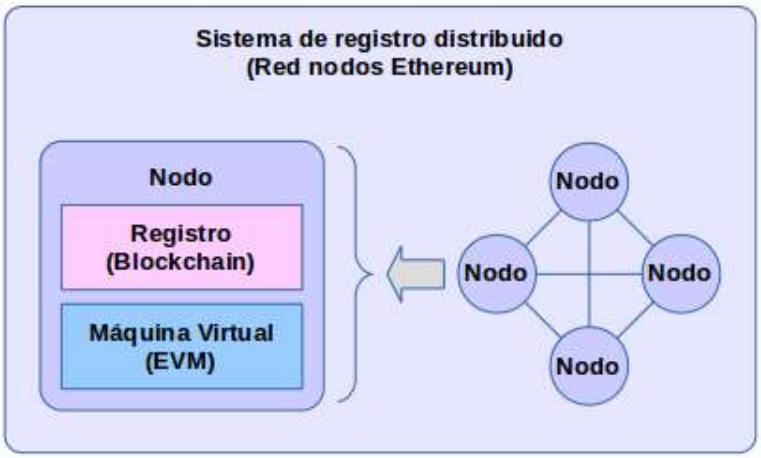


Figura 5. Cada nodo del sistema de registro distribuido contiene toda la cadena de bloques (datos y código) y una máquina virtual para ejecutar el código de los contratos inteligentes.

Entrando algo más en detalle, los contratos son un tipo de cuenta en la red Ethereum. Existen dos tipos de cuentas, por un lado los contratos inteligentes y por otro las cuentas de usuario:

- Externally Owned Accounts (EOAs): cuentas controladas por una clave privada y quien posea dicha clave (normalmente un usuario final) tendrá la capacidad de realizar transacciones.
- Contratos: cuentas que tienen su propio código, y que son controladas por dicho código.

En la cadena de bloques se almacenará, por tanto, las cuentas de usuario, los contratos, los datos controlados por los contratos y todas las transacciones realizadas en la red.

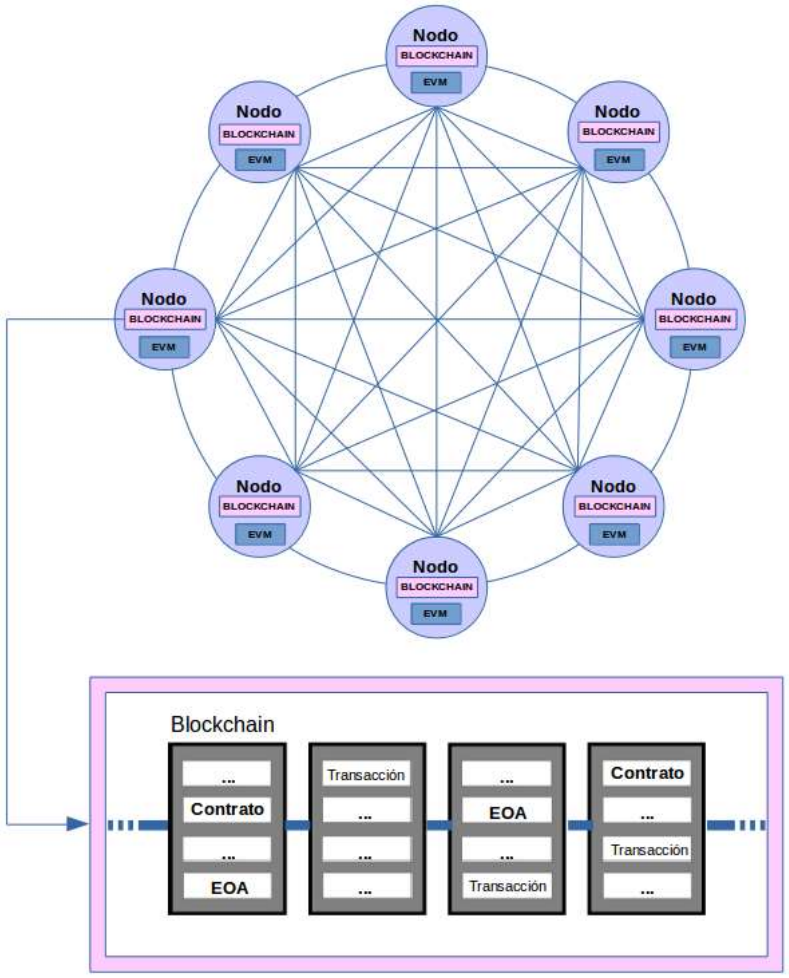


Figura 6. En la cadena de bloques de cada nodo se almacenan todas las cuentas de usuario, todos los contratos y todas las transacciones realizadas

Sistema de captura

El sistema de captura de datos, como se ha comentado, está formado por dos subsistemas diferentes según cuál sea el método de captura de datos: un subsistema de captura manual donde las personas introducen los datos mediante un interfaz de usuario y un subsistema de captura automática mediante dispositivos inteligentes (IoT).

Ambos subsistemas se comunican con el sistema de registro distribuido a través de un interfaz (API) con el nodo local empleado por el sistema. Podemos considerar así que el sistema de registro distribuido está dividido en dos partes: una local con el nodo controlado por la aplicación del proyecto y otra remota con el resto de nodos de la red.

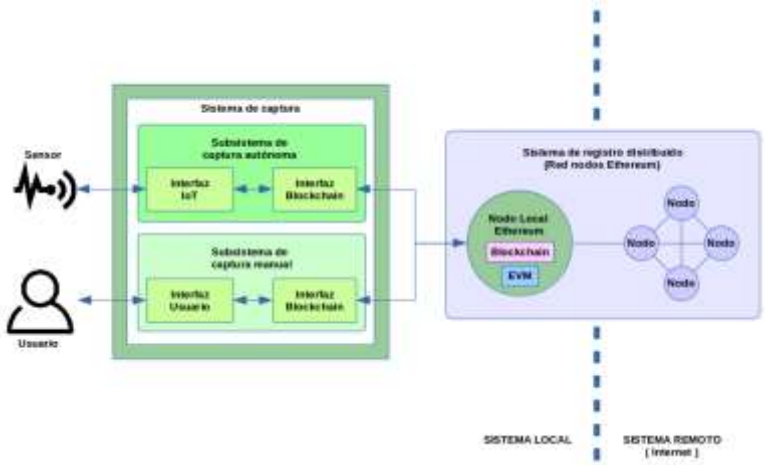


Figura 7. Esquema general del sistema de captura y su interacción con el registro distribuido

Sin embargo, y en la práctica, al realizar las pruebas iniciales para montar la infraestructura necesaria de los subsistemas de captura según el esquema anterior **se han encontrado limitaciones técnicas** que sugieren variar esta arquitectura inicialmente planteada para que sea técnicamente viable.

Subsistema de captura manual

En la arquitectura inicial, el equipo local contiene tanto el subsistema de captura manual como el nodo local con el que se comunica dicho sistema de captura y que hace de intermediario con el resto de la red de Ethereum.

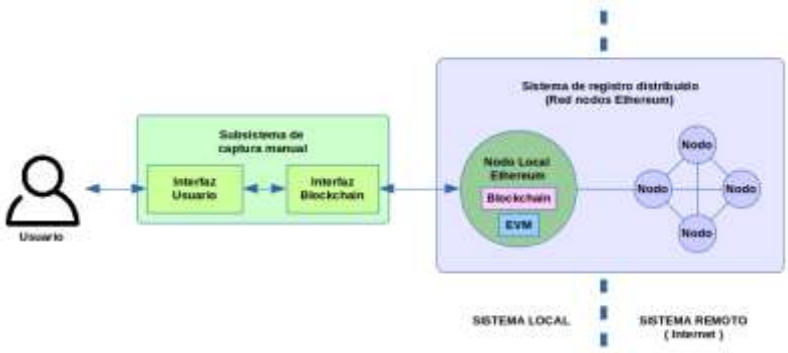


Figura 8. Esquema del subsistema de captura manual y su interacción con el registro distribuido

Si bien esta arquitectura es la ideal y es técnicamente posible, la necesidad de que el equipo local contenga toda la información almacenada en la cadena de bloques aumenta considerablemente los requisitos hardware y software del sistema local, tanto en capacidad de almacenamiento, como en capacidad de cómputo y en conectividad.

Estos elevados requisitos suponen una barrera de acceso importante si se desea que el sistema sea empleado por usuarios finales de forma transparente y sencilla. Con ese objetivo de mejora de la

usabilidad del sistema **se plantea una arquitectura alternativa empleando el modelo de *cliente ligero***. En este modelo el nodo de Ethereum se elimina del equipo local, que únicamente contendrá el sistema de captura, y se lleva a un servidor que realizará toda la carga pesada de trabajo y actuará de intermediario con el resto de la red Ethereum.

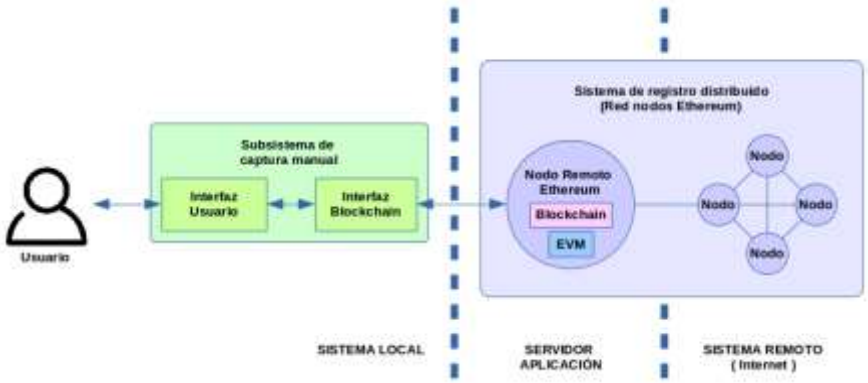


Figura 9. Esquema alternativo del subsistema de captura manual accediendo a un nodo Ethereum remoto

Subsistema de captura autónoma

La arquitectura inicial del subsistema de captura autónoma es similar al subsistema manual, donde el equipo local contiene el nodo Ethereum con el que se comunica el sistema de captura.

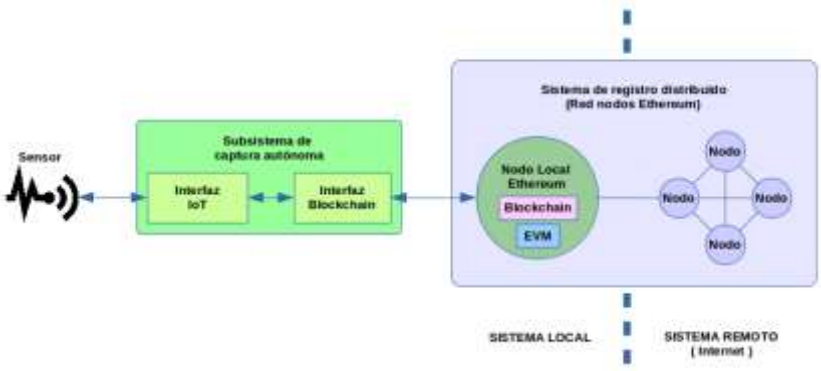


Figura 10. Esquema del subsistema de captura autónoma y su interacción con el registro distribuido

En este caso, al tratarse de un sistema autónomo, los requisitos relativos a la usabilidad del sistema no tienen carácter crítico. Sin embargo, durante las primeras pruebas de concepto se encontraron dificultades de carácter técnico. Si bien se consiguió **su implementación en un sistema** de pruebas ya disponible y **basado en un microcomputador de placa reducida** (SBC, Single Board Computer) de bajo coste, una Raspberry Pi 3 Modelo B, los altos requisitos de hardware necesarios para ejecutar un nodo Ethereum de forma local lo hicieron, desde un punto de vista técnico, **no inviable pero sí escasamente práctico**. La capacidad de cómputo necesaria y la necesidad de conexión ininterrumpida y sincronización continua con la red Ethereum, o re-sincronización en caso de pérdida de conexión, fueron las principales

limitaciones técnicas.

Para salvar estas limitaciones, **se optará por un modelo en el que el equipo IoT (Rasperry Pi) actúe de forma remota** realizando únicamente las tareas de captura de datos (sensórica) y envío de dicha información a través de la red **hasta el equipo con la aplicación y el nodo local**. A este equipo se le denomina "***IoT Gateway***[5]" en el ecosistema del IoT.

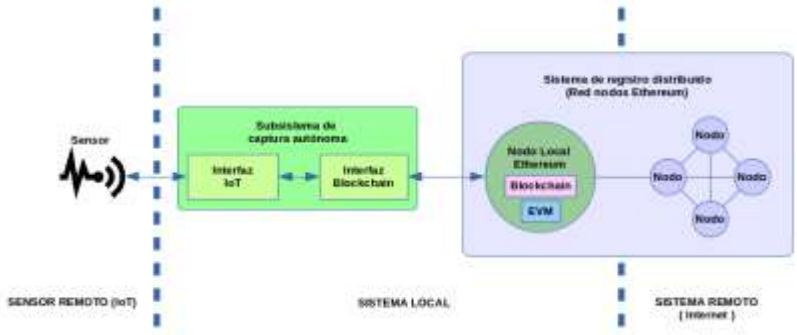


Figura 11. Esquema alternativo del subsistema de captura autónoma actuando de forma remota

Funcionalidades

Tal y como se señalaba anteriormente, el sistema debe registrar de forma segura y confiable los datos capturados. Al basarse en una arquitectura distribuida, ha de disponer de tecnologías y protocolos de comunicación entre los nodos de la red y el almacenamiento de los datos y la información se ha de realizar de forma distribuida por todos los nodos.

Asimismo, para dar libertad en el tipo de transacciones y operaciones que se pueden realizar en el sistema, éste no ha de estar limitado a un conjunto de transacciones predefinidas, sino que debe permitir la ejecución de código (contratos inteligentes) de forma que el sistema sirva como plataforma de aplicaciones distribuidas. También ha de aportar mecanismos para maximizar la seguridad del sistema, como el uso de criptografía para el firmado de transacciones o sistemas de consenso para la validación de cualquier transacción antes ser registrada en el sistema.

A nivel no funcional se han identificado tanto requisitos técnicos como requisitos de usuario. A nivel técnico destacan principalmente dos requisitos de alta prioridad: seguridad (RT1) e inmutabilidad (RT2) de los datos. Ambos requisitos son necesarios para desarrollar un sistema que funcione con la fiabilidad necesaria para que los usuarios confíen tanto en el sistema como en la veracidad e integridad de los datos almacenados en el mismo.

Los requisitos de usuario cobran gran importancia para minimizar las barreras técnicas y de usabilidad

[5] Intel Developer Zone – IoT Documentarion - What Is the Gateway and Why Should I Care?
https://software.intel.com/en-us/articles/what-is-the-gateway-and-why-should-i-care

que puedan disuadir del uso del sistema. En la misma línea, también destacan los requisitos para aumentar la confianza en el sistema, como la transparencia de su funcionamiento y el carácter público de la información almacenada.

Por otra parte, el sistema de captura debe de proporcionar la funcionalidad necesaria para permitir al usuario la introducción de datos y la consulta de la información ya registrada en el sistema. De igual forma, el sistema también debe poder capturar y registrar datos de forma automática desde sensores u otros dispositivos de IoT.

Igual que en el caso del sistema de registro, para el sistema de captura también se ha llevado a cabo el análisis de los requisitos no funcionales desde el punto de vista técnico y desde el punto de vista de los usuarios

Todo el conjunto de requisitos recolectados durante la ejecución del Hito 1 han sido identificados por dos medios principales:

- *Análisis del estado del arte*. Se han analizado las principales referencias científicas encontradas y relacionadas con el estudio del tipo de sistemas objetivo de este proyecto.
- *Análisis de aplicaciones piloto existentes*. Se han analizado diversas aplicaciones distribuidas (Dapps) relacionadas con el registro y almacenamiento confiable de información crítica.

Requisitos hardware y software

Dado que la arquitectura general está compuesta por varios subsistemas diferentes interrelacionados entre sí, pero potencialmente independientes en cuanto a requisitos hardware y software necesario, no se pueden establecer unos requisitos mínimos únicos, sino que será necesario diferenciarlos para cada caso.

Podemos identificar los siguientes sistemas para cubrir las diferentes arquitecturas y configuraciones planteadas:

- Sistema de captura de datos (sin nodo)
- Sistema de captura de datos (con nodo)

En cuanto a **requisitos de software para un cliente sin nodo** (sistema de captura de datos sin nodo) el principal es disponer del framework Meteor[6], empleado para la creación de aplicaciones distribuidas. Meteor es una plataforma para crear aplicaciones web en tiempo real construida sobre Node.js. Se trata de un framework *full-stack* cuya principal ventaja es la incorporación de todas las herramientas necesarias para la creación de aplicaciones web SPA (Single Page Applications) reactivas. Se trata de la tecnología principal para la creación de dapps o aplicaciones distribuidas en la red de Ethereum. Adicionalmente, para permitir la funcionalidad de los clientes ligeros, es necesario un navegador que

[6] Meteor https://www.meteor.com/

permita el acceso a nodos Ethereum remotos, bien mediante el propio navegador como es el caso de Mist (navegador de Ethereum) o bien mediante extensiones de otros navegadores (p. ej. Metamask para Chrome).

En cuanto a los **requisitos de hardware para un cliente sin nodo**, y teniendo en cuenta lo anterior, estos serán los necesarios para poder ejecutar un navegador web moderno con amplio soporte de tecnologías actuales como HTML5 (requisitos propios del framework Meteor). Así, los requisitos mínimos identificados son los requeridos por el navegador Chrome sobre un sistema operativo Windows 7, o similar.

Por otro lado, los **requisitos de software para un cliente con nodo** (sistema de captura de datos con nodo) son los mismos que para un cliente sin nodo, pero añadiendo la necesidad de un cliente de la red Ethereum, como Geth[7] (cliente principal de la red Ethereum implementado en el lenguaje Go).

Los **requisitos hardware para un cliente con nodo** aumentan para poder ejecutar dicho cliente (requisitos de CPU y Memoria), así como para almacenar toda la cadena de bloques (requisito de almacenamiento).

Otros requisitos

Esta red blockchain dispone de su propio token criptográfico (moneda/divisa digital) denominado Ether[8]. Esta moneda, que tiene un valor real y hasta puede intercambiarse por dinero fiduciario según la cotización en cada momento, se emplea en la plataforma como método de pago en transacciones y para pagar las cuotas de carga computacional (gas) necesaria para realizar transacciones. Toda transacción tiene un coste computacional y por tanto requiere cierta cantidad de gas el cual tiene un valor que ha de pagarse con Ether.

Como consecuencia de lo anterior, para poder trabajar sobre la red de Ethereum es necesario disponer de al menos una cantidad mínima de Ethers, que puede conseguirse de alguna de estas dos formas:

- Cambiando moneda fiduciaria, como Euros, por Ethers. Es decir, comprar monedas digitales con dinero real según la cotización.
- Creando Ethers (minado) desde la propia red Ethereum. Sin entrar en detalles técnicos, la minería es el proceso por el cual un equipo conectado a la red realiza el trabajo de cómputo necesario para que funcione el sistema (validar transacciones, generación de bloques, etc.) y recibe Ether a modo de recompensa por el trabajo aportado. El trabajo necesario para conseguir Ether es muy intensivo y costoso desde el punto de vista computacional, requiriendo de elevados recursos en cuanto a hardware.

En cualquiera de los dos casos, es necesaria una inversión con un coste económico para la obtención de Ether, bien mediante compra directa con dinero, o bien mediante la inversión inicial en equipos de

[7] Go Ethereum (Geth) https://geth.ethereum.org/
[8] Ethereum Project. What is Ether? https://www.ethereum.org/ether

minería y en gasto energético (consumo eléctrico) causado por el proceso computacional.

Hito 2: Implementación del sistema – plataforma de registro distribuido (blockchain)

El objetivo principal de este hito es realizar el despliegue y puesta en marcha de la plataforma DL seleccionada para el proyecto como resultado de la tarea T1.1 del Hito 1, es decir de la plataforma Ethereum.

T2.1. Despliegue de red BlockChain

Para proceder al despliegue de una red blockchain, en nuestro caso basada en la plataforma DL ofrecida por Ethereum Project, es necesario primeramente **instalar un software cliente** de la red (cliente Ethereum) en todos los equipos que vayan a formar parte de la red como nuevos nodos de la misma. Después hay que proceder a la **sincronización de estos nodos con la red Ethereum**, proceder a la **creación de cuentas**, y dotarlas de saldo para poder realizar operaciones tales como la **creación de contratos inteligentes**, o la ejecución de los mismos y los procesos asociados a las **aplicaciones (Đapp) de usuario final**.

En el caso del segundo piloto "Mantenimiento predictivo con blockchain" se optó por utilizar una Blockchain privada.

Despliegue del cliente de red Ethereum

Inicialmente se realizó la instalación del cliente principal para la red Ethereum, denominado Geth, en todos los ordenadores del entorno de desarrollo asignado al proyecto BCCB (3 equipos con sistema operativo Linux).

Geth es un cliente multipropósito por línea de comandos que ejecuta un nodo Ethereum completo. Geth ofrece tres interfaces: los subcomandos y opciones por línea de comando, un servidor JSON-RPC, y una consola interactiva. Este cliente se seleccionó entre los disponibles debido a su popularidad y a tener disponible una mejor documentación, lo que haría más fiable su actualización y resolución de incidencias.

Desplegado el cliente Geth se procedió a la sincronización de los equipos de desarrollo y pruebas con la red blockchain de Ethereum, convirtiéndose así en nodos de dicha red, tanto de la principal o de "producción" como de las redes de test utilizadas para pruebas.

Para el segundo piloto se simuló una red privada global desplegando varios nodos de distinta naturaleza utilizando contenedores Docker. Estos contenedores utilizan geth parametrizado con un fichero génesis ad-hoc que define las características de la red.

Sincronización con la plataforma Ethereum

Con el objeto de acelerar el proceso de sincronización inicial con la red principal se usaron unos parámetros en cada nodo (el tag –fast) que hace que no se conserven las transacciones pasadas.

Aun así, el proceso de sincronización de los nodos resultó bastante lento y en muchos casos bloqueante, impidiendo a veces el uso de la conexión a Internet e incluso el uso de cualquier aplicación, sobre todo en aquellos equipos con menor capacidad de proceso.

En el caso del segundo piloto la sincronización no ha resultado un problema puesto que al ser una red privada el número de transacciones es muy reducido y por tanto la sincronización es generalmente inmediata

Creación de cuentas

Empleando el cliente Geth, se creó una cuenta de usuario, del tipo *externally owned accounts*, en la plataforma Ethereum.

Con el fin de poder realizar pruebas de la aplicación sin consumir Ethers reales, se decidió hacer uso de la red blockchain de prueba, en la que se lanzó un proceso de minería de Ether que permitió acumular suficientes recursos para los siguientes pasos a realizar.

Publicación de contratos inteligentes

Se procedió a publicar y ejecutar en la plataforma un primer contrato inteligente (smart contract) de prueba, lo cual se trató de realizar usando el cliente Geth, aunque no fue posible, ni tampoco empleando el entorno de desarrollo integrado (IDE) proporcionado por la plataforma Mix (Mix IDE). Finalmente se recurrió a otro cliente como es la Wallet Ethereum, que instalada en uno de los nodos permitió publica el primer contrato con éxito, y observar los resultados de su ejecución.

T2.2. Sistema de tokens, ID, seguridad

Con la plataforma Ethereum apareció la posibilidad de crear **tokens** que representan derechos sobre bienes (materiales o inmateriales) y sobre servicios, y de forma que dichos tokens pueden ser objeto de comercio (transferirse, subdividirse, etc.).

Dentro del proyecto se valoró la posibilidad de crear un token funcional propio utilizando el estándar conocido como ERC20. El objetivo era emplear dicho token en el primer piloto (vida útil de vehículos) para articular a través del mismo el “modelo de negocio” de la dApp, pensando en su aplicación en la vida real.

Figura 14. Diseño circulación de tokens (modelo negocio 1er piloto)

Aunque en las primeras tentativas se comprobó la viabilidad y que la creación del propio token no presenta una especial dificultad técnica, pues se realiza mediante un relativamente sencillo contrato inteligente (código), finalmente se ha descartado incluir en los pilotos del proyecto el empleo de un token propio. El motivo es que añadiría una complejidad funcional grande a las dApp, por la intrincada trama de circulación posible de tokens entre diversos actores en el sistema, mientras que desde el punto de vista de los usuarios principales (propietarios de vehículos y posibles compradores interesados, en el caso del 1er piloto) no aporta ventajas de cara al objetivo principal. Así pues, creemos que la implantación de un token utilitario podría ser algo a considerar, pero ya fuera del ámbito de los pilotos, en caso de una evolución futura de los mismos hacia una escalabilidad para salir a mercado.

Aunque la autenticación de usuarios se basa en el propio sistema de **identidad** de la plataforma Ethereum, en cuanto a la privacidad y seguridad se han establecido controles de acceso vinculados a dicha identidad. En los contratos inteligentes se almacena qué cuentas/usuarios tienen acceso a realizar ciertas operaciones (creación y modificación) y son luego esos mismos contratos inteligentes los que verifican en tiempo de ejecución si el usuario que solicita una operación pertenece a la lista de autorizados.

Esto se visibiliza en el propio piloto incluyendo mensajes de advertencia al usuario presentados desde la dApp cuando intenta realizar alguna operación no autorizada.

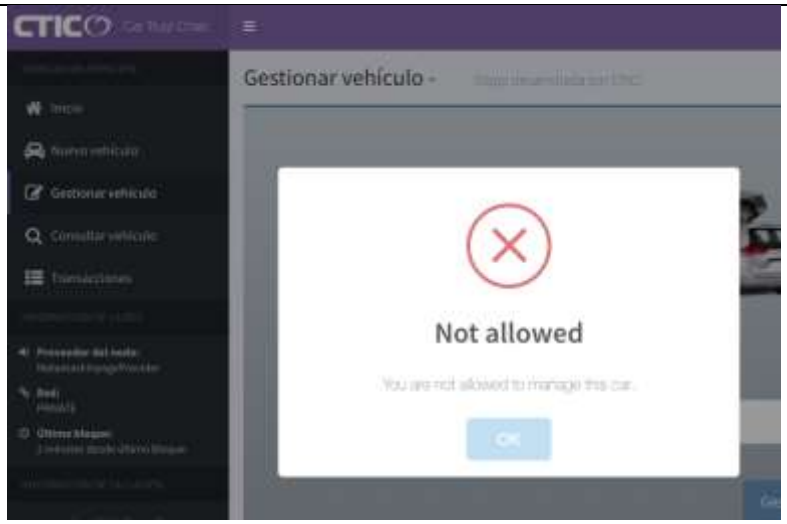


Figura 15. Mensaje de advertencia a usuario no autorizado (1er piloto)

Así mismo, y dado que los datos de la blockchain son públicos, se ha implementado un **mecanismo de anonimización para datos sensibles** consistente en la utilización de funciones criptográficas de hashing (Keccak-256). Cuando se identifica un dato que es a la vez privado y necesario para la operativa de la aplicación se realiza un hashing previo a la grabación del mismo en la blockchain. Dado que estos algoritmos no son reversibles es imposible obtener el dato original a partir del hash presente en la blockchain. Pero por otra parte si es posible realizar una verificación puesto que es trivial a partir de un dato presentado comprobar si su hash coincide con el supuestamente homólogo en la blockchain.

Esto se visibiliza (en parte) en el propio piloto generando códigos QR para el usuario, suministrados desde la dApp, con los que poder ocultar información sensible y a la vez dar un medio controlado de acceso a la misma cuando se requiera.

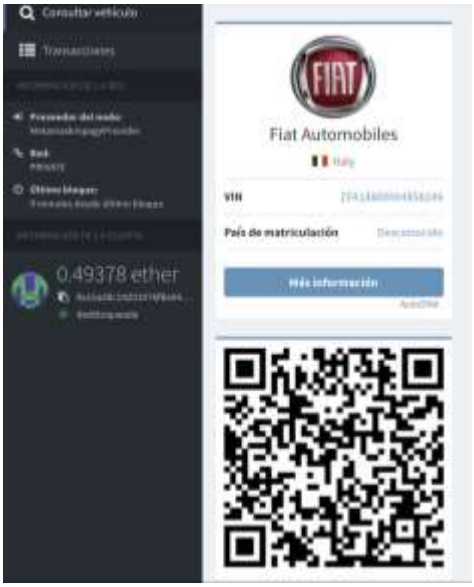


Figura 16. Código QR generado para ocultar y dar acceso a información sensible (1er piloto)

Hito 3: Captura y gestión de datos (software)

El objetivo principal de este hito es realizar el desarrollo de un subsistema que permita a los usuarios intercambiar elementos de valor según unas reglas de negocio establecidas por software (T3.1) y a través de un interfaz amigable y usable (T3.2).

T3.1. Desarrollo de contratos inteligentes

La plataforma seleccionada (Ethereum) permite utilizar contratos inteligentes (Smart Contracts) para la interacción de los actores externos con la Blockchain. Estos contratos no son más que pequeños programas que se ejecutan en la EVM (Ethereum Virtual Machine) que cada nodo completo de la red mantiene operativa. En estos contratos es posible implementar una lógica sencilla que gobierne el flujo de ciertas transacciones, así como almacenar/recuperar datos.

Es importante recalcar que dada la naturaleza distribuida de la red estos contratos se ejecutan en todos los nodos de la misma, y deben pues cumplir ciertas caracteristicas principales:

- **Determinismo**: el flujo de datos debe ser consistente, para unas determinadas entradas y estados iniciales el resultado debe ser siempre el mismo.
- **Simplicidad**: la complejidad, tanto a nivel computacional como de datos, tiene un coste proporcional en la red Ethereum. Es posible incurrir en costes operacionales muy elevados si no se prima la sencillez.
- **Autosuficiencia**: su posibilidad de comunicación con el mundo exterior es muy limitada y por tanto es preciso diseñar contratos que no dependan de interacciones con agentes externos.

Aunque la red siempre ejecuta los contratos en su forma “bytecode” (compilados) el desarrollo se realiza en un lenguaje de alto nivel. Son varios los lenguajes que disponen de compiladores a bytecode EVM, sin embargo, para este proyecto se ha seleccionado Solidity por las siguientes razones:

- **Adecuación**: Es un lenguaje diseñado específicamente para el desarrollo de contratos inteligentes en Ethereum, y por tanto dispone de ciertas características especiales que permiten sacar todo el rendimiento a la plataforma.
- **Adopción**: Es, por un amplio margen, el lenguaje más utilizado en el ecosistema Ethereum, y por tanto existen más recursos que facilitan la labor de desarrollo (documentación, herramientas, frameworks).
- **Simplicidad**: En su forma básica es un lenguaje sencillo, con similitudes con algunos de los más populares, y que no precisa de una formación costosa.

Para el desarrollo de los contratos específicos para cada piloto se ha utilizado el módulo de contratos del

framework Truffle, que permite simplificar las tareas de compilación, despliegue y pruebas. Las pruebas y depuración se han realizado utilizando la herramienta ReMix.

Los contratos inteligentes de ambos pilotos siguen un esquema general similar: un contrato principal que ejerce de interfaz con el usuario y varios auxiliares cuya misión es facilitar la gestión de los datos persistentes, permisos de acceso y los eventos.

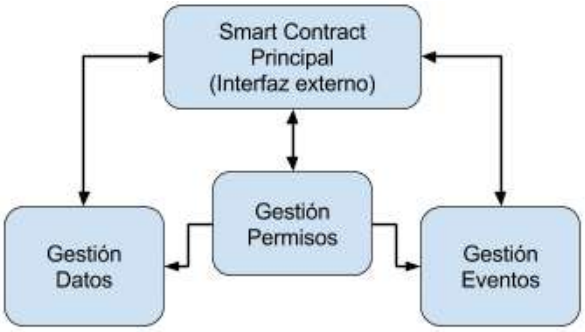


Figura 17. Arquitectura general de Smart Contracts en Pilotos

Esta arquitectura ha sido el fruto de un proceso iterativo, al que se ha llegado después de pasar por varias etapas:

- Monolítico: toda la lógica y datos se almacenaba en un solo contrato. Si bien esto simplificó la labor de desarrollo, se comprobó posteriormente que era muy ineficiente para la realización de nuevos despliegues, puesto que exigía migrar los datos de forma completa en cada operación.
- Multiobjeto: se creó un nuevo contrato para cada nuevo elemento a modelar (p.ej: vehículo). En las pruebas realizadas se observó que el coste de operaciones era muy elevado, puesto que la creación de nuevos contratos es una de las operaciones más costosas en la red Ethereum.
- Multicapa: se separó la lógica de la aplicación de la gestión de datos, eventos y permisos, de tal manera que es posible actualizar cada una de las piezas de forma independiente. Se constató que este modelo ofrece una gran flexibilidad sin elevar en exceso la complejidad ni los costes operacionales. Este fué el modelo finalmente utilizado en la versión final de los pilotos.

T3.2. Desarrollo de la interfaz de usuario

Una aplicación distribuida o DApp de Ethereum permite a los usuarios interactuar a través de una aplicación, normalmente web (HTML/Javascript), y una API de JavaScript con la cadena de bloques. Una DApp tiene asociado su propio contrato o conjunto de contratos desplegados en la cadena de bloques. Estos contratos forman la lógica de negocio o el *backend* de dicha aplicación, permitiendo el almacenamiento persistente de su estado de forma consensuada, aprovechando así las características y ventajas que aporta la cadena de bloques.

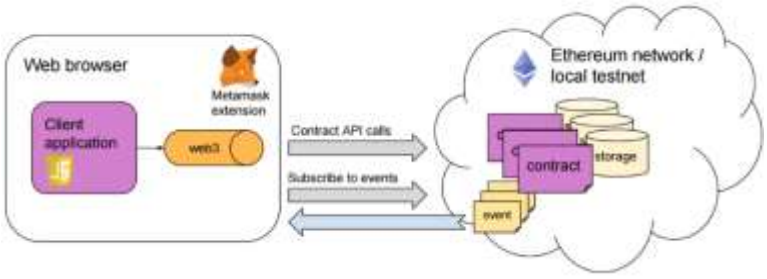


Figura 18. Esquema clásico de interacción del frontend con contratos en la red Ethereum

Aunque pueden emplearse diferentes tecnologías para la creación del *frontend* de la DApp, para el proyecto se ha optado por utilizar las tecnologías más habituales y recomendadas en la plataforma Ethereum, como son las tecnologías web (HTML/Javascript) junto con diversos *frameworks* de desarrollo.

Figura 19. Ejemplo: frontend básico HTML vs. frontend rico piloto vida útil vehículos (1ª versión)

Para el desarrollo del interfaz de usuario de la aplicación, y con el objetivo de probar diferentes soluciones técnicas, en cada uno de los prototipos del proyecto se ha empleado un *framework* de desarrollo diferente.

Así, en el primer prototipo (vida útil de vehículos) se optó por el uso de **Meteor**, un framework de desarrollo de aplicaciones web con carácter generalista.

Figura 20. IDE con el proyecto en Meteor (ej.: dApp CarTurstChain)

Sin embargo, para el segundo prototipo (mantenimiento de máquinas) se empleó **Truffle**, un framework específico para DApps de Ethereum.

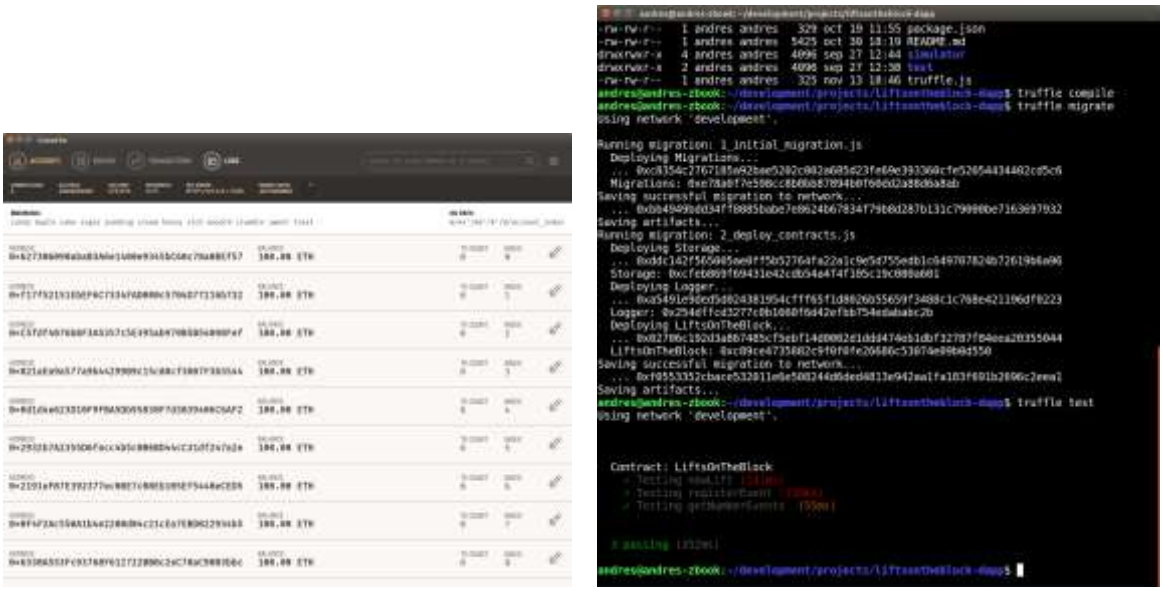


Figura 21. GUI testrpc + Truffle línea de comando (ej. dApp LiftsOnTheBlock)

Este cambio en las tecnologías de desarrollo empleadas fue posible gracias a la evolución de las herramientas del ecosistema de Ethereum en este último año, tanto la creación de nuevas herramientas como a la evolución y mejora de las ya existentes.

El uso de Truffle y la no dependencia de Meteor permitió además eliminar la necesidad de un servidor web (Node en el caso de Meteor) para alojar y servir el frontend de la aplicación, pudiendo así explorar otras soluciones propias del ecosistema Ethereum como el sistema de archivos distribuidos **IPFS**. Empleando el sistema IPFS se ha logrado obtener una DApp funcional completamente distribuida,

siendo este el objetivo fundamental de toda DApp.

Como se ha indicado, la comunicación entre el frontend y un nodo de Ethereum con la cadena de bloques se realiza empleando la API de Javascript Web3.js. El interfaz web intenta establecer en primer lugar la comunicación con un nodo de Ethereum detectando la presencia de un proveedor de un objeto Web3.

Hito 4: Adquisición automática de datos (hardware)

T4.1. Despliegue de sensores IoT

En el contexto del piloto "Mantenimiento predictivo con Blockchain" los sensores IoT serían los encargados de suministrar datos referentes al estado del equipamiento que monitorizan, así como diversas incidencias que se pudieran producir.

Dada la dificultad de acceder directamente a sistemas propietarios se optó por utilizar un dispositivo IoT genérico (RPi3) operando en un modo simulador.

Asimismo, se ha simulado un hub IoT mediante un equipo mini-PC, cuya misión es servir de nodo de acceso a la red Ethereum para un conjunto de sensores.

En este caso concreto son los sensores simulados en la RPi3 los que se conectan al hub.

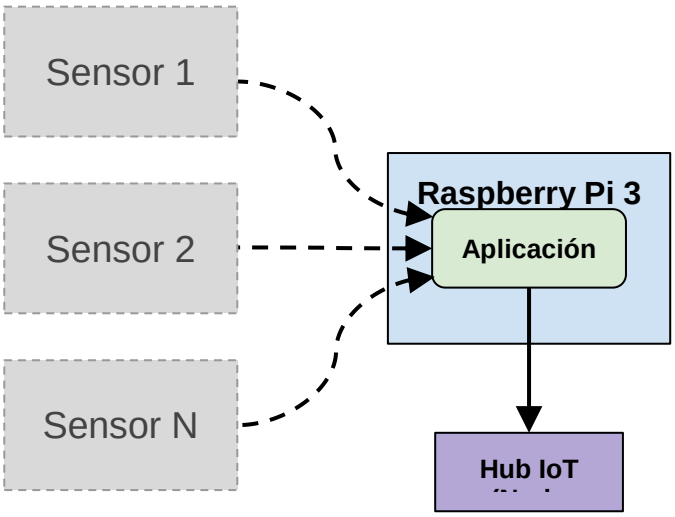


Figura 23. Esquema general de la red de sensores simulada

Todas estas simulaciones se han realizado manteniendo la operativa que seguiría un sensor real (luz, movimiento, etc.), de tal forma que el esfuerzo para adaptar cualquier nuevo sensor sea mínimo.

Se ha desarrollado una sencilla aplicación que gestiona los datos de los sensores y los encamina al hub

IoT donde reside la dApp que interactúa con la blockchain.

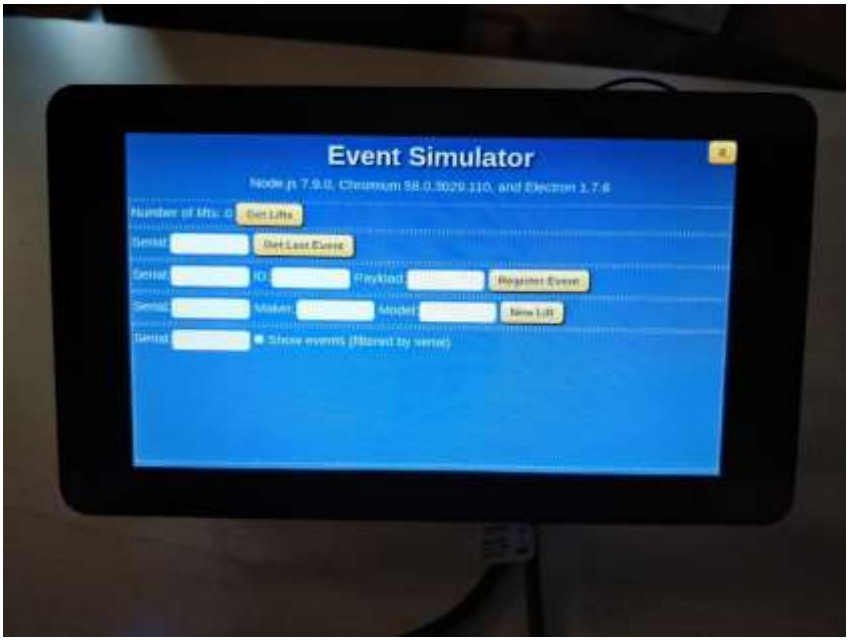


Figura 25. Imagen de la aplicación ejecutándose en la Raspberry Pi

T4.2. Adaptación al sistema

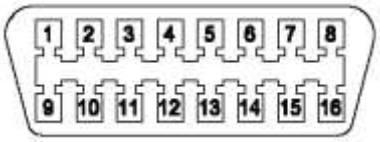
La adquisición por las aplicaciones distribuidas (las dApp) de los datos capturados por los sensores se materializa a través de la aplicación que se ejecuta en la Raspberry Pi (dispositivo IoT genérico), a la cual hemos dotado de un interfaz de usuario mostrado en una pantalla táctil de reducidas dimensiones, simulando así un panel de control.

De esta forma, y en concreto para el segundo piloto “Mantenimiento predictivo con blockchain” (véase más adelante en Hito 5), además de simular el procesamiento y envío de diversos tipos de alertas y eventos generados de forma automática por una máquina, la aplicación habilita a los posibles usuarios de la misma que interactúen con ella para generar alertas de forma manual (p. ej. botón de alerta).

En este caso es el propio sistema de captura el que se encarga de registrar los datos directamente en la blockchain conectándose a un nodo (mini-PC con hub IoT) de una red privada creada de forma específica para el piloto.

Adicionalmente para el primero piloto “Vida útil de vehículos”, cuya dApp recibe los datos del sistema de captura (manual vía interfaz de usuario) para posteriormente grabarlos en la blockchain, nuestra intención era también recuperar los datos del kilometraje acumulado por un coche real, desde su centralita vía puerto OBD-II (On Board Diagnostic Second Generation), pero se comprobó que dicho

parámetro no forma parte de los recuperables[9].



Esquema genérico del puerto OBD-II

De hecho, los únicos parámetros disponibles relacionados con la distancia son el PID 33 (distancia recorrida con la luz de aviso de fallo encendida) y el PID 49 (distancia recorrida desde el último borrado de códigos), ninguno de los cuales resulta de utilidad para nuestro propósito. Es muy probable que el parámetro de distancia total acumulada solo esté a disposición del fabricante, ya que además de los parámetros estándar, cada fabricante puede añadir los suyos propios, y la recuperación de éstos sólo es posible mediante herramientas propietarias del mismo.

Así pues y debido a estas dificultades técnicas, cuya resolución no forma parte del objetivo principal del proyecto, para el piloto "Vida útil de vehículos" se ha simulado la captura de datos automática mediante un servicio web (REST), que publica en la red los Km virtuales recorridos por un vehículo tipo, en forma de un coche que circula a una velocidad constante de 60 Km/h.

Hito 5: Pruebas piloto

T5.1. Vida útil de vehículos

A continuación, se describe el proceso seguido para construir el modelo de aplicación y realizar las pruebas piloto correspondientes.

Análisis del caso de uso

Primeramente, se llevó a cabo una fase de **investigación documental** para recuperar información sobre el ciclo de vida de los vehículos (automóviles y vehículos ligeros), así como su vida útil, desde numerosas fuentes de información, principalmente online, y también de algunas publicaciones en papel.

Entre dichas fuentes consultadas están:

- sitios web de algunos fabricantes de automóviles;
- sitios web de asociaciones empresariales: fabricantes, talleres, otros servicios relacionados;
- sitios web de asociaciones de consumidores y clubes de automovilistas;
- sitios web de AA.PP. y agencias de control o empresas públicas: ITVs, DGT, DG MOVE, etc.

Adicionalmente también mantuvimos **entrevistas** con representantes de alguna de las anteriores organizaciones.

[9] OBD-II Parameter IDs https://en.wikipedia.org/wiki/OBD-II_PIDs

Toda esta información hizo posible acotar el ámbito del **caso de uso**, seleccionando una importante problemática a la que dar uso mediante nuestro piloto, como es el **fraude del kilometraje en la compra-venta de vehículos de segunda mano**.

Una vez acotado el ámbito, se procedió a determinar los principales **actores** que pueden interactuar con el sistema, siendo los mismos:

- Fabricantes de vehículos.
- Propietarios de vehículos.
- Vendedores profesionales de vehículos.
- Talleres de mantenimiento y reparación.
- Organismos públicos / entidades autorizadas de control.
- Desguaces de vehículos.
- Aseguradoras de vehículos.

El actor más importante es el propietario del vehículo o titular del mismo (sea persona física o jurídica), siendo el resto actores que para interactuar con un vehículo, suelen tener que contar con al menos el conocimiento, y a veces autorización, de su titular. Por lo tanto, para nuestro caso de uso y por simplificación únicamente contemplamos dos tipos de usuarios: propietario y (tercero) autorizado.

Finalmente se dedujeron las **variables** a tratar en el caso de uso y a considerar en los requisitos funcionales de la aplicación distribuida a desarrollar (dApp):

- Número o Código VIN: es el número de chasis o número de bastidor, denominado internacionalmente Vehicle Identification Number (VIN), único para cada vehículo fabricado, que consiste en un código específico formado por una secuencia de dígitos que identifica los vehículos de motor de cualquier tipo, y los remolques a partir de un cierto peso. La norma ISO 3779:2009[10] especifica el contenido y estructura del código VIN.

Figura 27. Ejemplo de localización del Código VIN en un vehículo (*Fuente: SEAT*)

- Matrícula: la matrícula de un vehículo es una combinación de caracteres alfabéticos o numéricos que identifica e individualiza el vehículo respecto a los demás para un determinado territorio.
- País: ámbito geográfico de aplicación de la matrícula de un vehículo. Esta variable es necesaria para que la dApp pueda tener uso no solo en España, sino en Europa, siendo el problema a

[10] ISO 3779:2009, Road vehicles – Vehicle identification number (VIN) – Content and structure https://www.iso.org/news/2011/04/Ref1603.html

atajar de magnitud Comunitaria.

- Fecha de matriculación: corresponde a la fecha de alta de la matrícula de un vehículo en el Registro de Vehículos oficial, gestionado por la entidad responsable en cada país, siendo la DGT en el caso de España.
- Kilómetros: distancia acumulada por el vehículo desde el inicio de su vida útil hasta el momento actual. Dado el ámbito europeo se contempla únicamente el sistema métrico.
- Tercero autorizado: cualquier actor no titular del vehículo que necesita interactuar con el mismo para registrar alguna información de entre las anteriores. En el piloto desarrollado queda restringido únicamente a la variable kilómetros.
- Tiempo de autorización: es el tiempo, expresado en minutos, del que dispone el tercero autorizado para efectuar el registro de información, a contar a partir de la concesión de la autorización.

Figura 28. Logotipo diseñado para la dApp CarTrustChain

Implementación

Para la implementación de este caso de uso se ha realizado una particularización de los componentes descritos con anterioridad en el Hito 3, es decir contratos inteligentes e interfaz de usuario.

En las primeras versiones de los contratos (monolítico, es decir, un solo contrato) se incluyeron en el mismo los métodos **`newCar`**, **`setKms`** y **`getKms`** básicos para dar de alta un registrar y consultar sus kilómetros.

En siguientes evoluciones (multiobjeto y multicapa) se incluyeron nuevos métodos hasta completar la arquitectura completa que da soporte a todas las funcionalidades de CarTrustChain en su estado final de desarrollo en el marco del proyecto BCCB.

Para el desarrollo de la interfaz de usuario se empleó el framework de desarrollo Meteor, y se partió de un diseño inicial conceptual (maquetas o mockups) a partir del cual se siguieron realizando mejoras y correcciones tras las primeras pruebas de funcionalidad.

Figura 30. Maquetas iniciales del front-end para la dApp CarTrustChain

Pruebas

Las primeras **pruebas funcionales** consistieron en:

- La creación de un nuevo contrato;
- La creación de un nuevo vehículo para ese contrato;
- El registro de kilómetros para ese vehículo;
- Consulta de últimos km registrados para ese vehículo.

Con posterioridad y una vez se añadieron nuevas funcionalidades y con ellas nuevas variables, se completaron las anteriores pruebas funcionales con:

- La creación de un usuario autorizado y un tiempo de autorización;
- El registro de kilómetros para un vehículo por un usuario autorizado para el mismo;
- La lectura (automática) de kilómetros de un vehículo al habilitar la opción “coche conectado”.

Se procedió al despliegue de la aplicación CarTrustChain en uno de los ordenadores de la empresa, conectado y sincronizado con la misma red Ethereum empleada en CTIC. Posteriormente personal de la empresa, por un periodo de alrededor de dos semanas procedieron al registro de varios de sus vehículos, sin observar ninguna incidencia negativa de funcionamiento, salvo algún problema de sincronización debido a cambios en la propia red Ethereum, pero que pudimos solventar de forma remota en poco tiempo.

T5.2. Mantenimiento predictivo 4.0

A continuación, se describe el proceso seguido para construir el modelo de aplicación y realizar las pruebas piloto correspondientes.

Análisis del caso de uso

De forma similar al otro piloto, primeramente, se llevó a cabo una fase de **investigación documental** para recuperar información sobre el mantenimiento predictivo de máquinas, desde numerosas fuentes de información, principalmente online, y también de algunas publicaciones en papel.

Entre dichas fuentes consultadas están:

- sitios web de algunos destacados fabricantes de sistemas de automatización industrial;
- sitios web de asociaciones empresariales: Industria 4.0, máquina herramienta, etc.;
- sitios web de AA.PP. y/o agencias de verificación y control: Ministerio de Industria, AENOR, etc.

Toda esta información hizo posible acotar el ámbito del **caso de uso**, seleccionando una máquina muy extendida y universal a la que monitorizar con nuestro piloto, como son los **ascensores o elevadores**, presentes tanto en la industria como en nuestros hogares.

Una vez acotado el ámbito, se procedió a determinar los principales **actores** que pueden interactuar con el sistema, siendo los mismos:

- Fabricantes de ascensores;
- Instaladores de ascensores;
- Empresas conservadoras;
- Organismos de control;
- Comunidades de vecinos;
- Usuarios particulares y empresas;
- Aseguradoras de bienes inmuebles.

Finalmente se dedujeron las **variables** a tratar en el caso de uso y a considerar en los requisitos funcionales de la aplicación distribuida a desarrollar (dApp):

- Número de serie del ascensor;
- Nombre (o ID) del fabricante;
- Modelo del ascensor;
- Tipo de evento detectado;
- Payload o valor asociado al evento según su tipo.
-

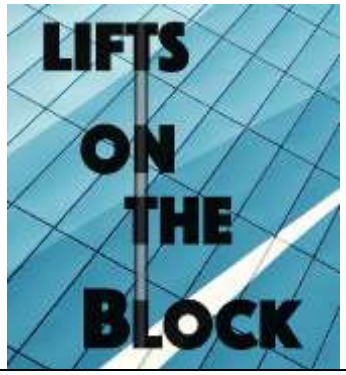


Figura 31. Logotipo diseñado para la dApp LiftsOnTheBlock

Implementación

Para la implementación de este caso de uso se ha realizado una particularización de los componentes descritos con anterioridad en el Hito 3, es decir contratos inteligentes e interfaz de usuario.

En un primer diseño de la arquitectura de contratos particularizada se pensó en la posibilidad de un edificio con varios ascensores, que incluso podían estar separadamente destinados a diferentes fines (personas, montacargas, otros…) y asociados a distintos propietarios.

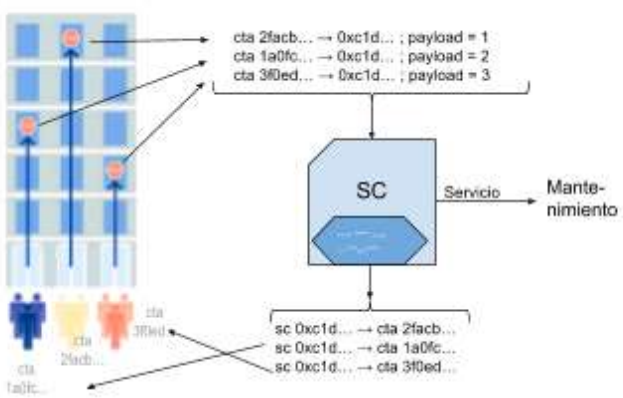


Figura 32. Diseño inicial del flujo de eventos en la dApp LiftsOnTheBlock

Pero el sistema “multipropietario” podía complicar en exceso el caso de uso y sobre todo la seguridad y mantenimiento de los contratos inteligentes, por lo que para el piloto se limitó el diseño de contratos e implementación a un único propietario (cuenta principal) y múltiples ascensores gestionados por el mismo.
En este piloto hay una gran parte de automatización, donde no es necesario un front-end de usuario para el registro de alertas de servicio, ya que los parámetros de funcionamiento de la máquina (ascensor) son capturados por sensores y enviados a un sistema de proceso (reactivo o predictivo) que automáticamente genera las alertas, que a su vez son registradas directamente por la dApp LiftsOnTheBlock en la blockchain, y todo ello sin necesidad de intervención humana.

Pero consideramos que el caso de uso no debe quedarse aquí, sobre todo para integrarlo en un proceso de mantenimiento real que pueda ser de utilidad para diversos actores, como empresas mantenedoras o comunidades de vecinos. Estos actores son, en definitiva, personas que deberán comprobar el estado real del ascensor, de su histórico de incidencias producidas, e incluso registrar a través de una interfaz amigable las tareas de mantenimiento necesarias para corregir las incidencias producidas. Es por ello que también se procedió al diseño de una interfaz de usuario.

Para este segundo prototipo se empleó Truffle, un framework específico para DApps de Ethereum, que nos permitió prescindir de un servidor web que alojase y sirviese el frontend de la aplicación. En su defecto pudimos emplear el sistema IPFS[11] para obtener una dApp funcional completamente distribuida, lo cual representa una mejora tecnológica respecto al primer piloto.

Pruebas

Las primeras **pruebas funcionales** consistieron en:

- La creación de un nuevo ascensor;
- La lectura automática de diversos eventos y su registro en blockchain;
- La consulta del histórico de eventos y estado del ascensor;
- El registro de la resolución de incidencias y mantenimientos programados.

Comprobado el correcto funcionamiento de todas las funcionalidades en nuestro entorno de laboratorio, se procedió a la demostración como **prueba real** de la dApp LiftsOnTheBlock en un entorno de pruebas simulado.

2. RESULTADOS PREVISTOS Y RESULTADOS CONSEGUIDOS

Durante el periodo de ejecución del proyecto BCCB (2016 - 2017) se han conseguido todos los resultados planificados, tal y como se había planteado al inicio del proyecto y que se detallan a continuación:

- Diseño e implementación de un sistema capaz de registrar y recuperar transacciones en una red blockchain
- Puesta en marcha de una red blockchain
- Puesta en marcha de un sistema de registro y consulta de información y datos de valor proporcionados por usuarios (humanos) y máquinas (sensores)
- Validación de las técnicas generadas mediante dos casos-estudio (pilotos)

[11] InterPlanetary File System (IPFS) https://ipfs.io/