



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

Aplicación Distribuida Basada en Blockchain para dar soporte a IoT

Desarrollo de un registro distribuido para IoT usando la
tecnología Blockchain.

Autor
Fernando Rafael Talavera Mendoza

Director
Pedro Ángel Castillo Valdivieso



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Noviembre de 2020



UNIVERSIDAD DE GRANADA

Aplicación Distribuida Basada en Blockchain para dar soporte a IoT

Desarrollo de un registro distribuido para IoT usando la
tecnología Blockchain.

Autor

Fernando Rafael Talavera Mendoza

Director

Pedro Ángel Castillo Valdivieso



DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Granada, Noviembre de 2020

Aplicación Distribuida Basada en Blockchain para dar soporte a IoT

Fernando Rafael Talavera Mendoza

Palabras clave: Blockchain, Raspberry Pi, Hyperledger Fabric, Docker, GraphQL, Gatsby.

Resumen

Desde el comienzo de Bitcoin, la tecnología Blockchain surgió como la siguiente tecnología revolucionaria. Aunque Blockchain comenzó como una tecnología central de Bitcoin, sus casos de uso se están expandiendo a muchas otras áreas, incluyendo finanzas, Internet de las cosas, seguridad y demás. Cabe destacar, que la tecnología del Internet de las cosas ha tenido gran influencia y aceptación, y actualmente, muchos sectores públicos y privados se están sumergiendo en la tecnología. Estos dispositivos necesitan comunicarse y sincronizarse entre sí. Pero en situaciones en las que más de miles o decenas de miles de dispositivos que se conectan, esperamos que el uso del modelo actual de cliente-servidor pueda tener limitaciones y problemas durante la sincronización.

Por tanto, se propone usar la tecnología Blockchain para controlar y configurar los dispositivos de IoT. Realizar un estudio del papel que puede tomar en dicho escenario, a nivel de soluciones, beneficios y desafíos. Y escoger las herramientas para realizar una prueba de concepto con Raspberry Pi e implementar una aplicación distribuida para facilitar el registro y la administración de los dispositivos a los usuarios en una red domótica.

Distributed Application Based on Blockchain to support IoT: Development of a distributed registry for IoT using Blockchain technology.

Fernando Rafael Talavera Mendoza

Keywords: Blockchain, Raspberry Pi, Hyperledger Fabric, Docker, GraphQL, Gatsby.

Abstract

Since the start of Bitcoin in 2008, blockchain technology emerged as the next revolutionary technology. Though blockchain started off as a core technology of Bitcoin, its use cases are expanding to many other areas including finances, Internet of Things (IoT), security and such. It is noteworthy, that IoT technology has had great influence and acceptance, and currently, many private and public sectors are diving into the technology. These IoT devices need to communicate and synchronize with each other. But in situations where more than thousands or tens of thousands of IoT devices are connected, we expect that using current model of server-client may have some limitations and issues while in synchronization.

Therefore, it is proposed to use the Blockchain technology to control and configure the IoT devices. Make a study of the role it can take in that scenario, at the level of solutions, benefits and challenges. And choose the tools to perform a proof of concept with Raspberry Pi and implement a distributed application to make it easier for users to register and manage their devices in a home automation network.

Yo, **Fernando Rafael Talavera Mendoza**, alumno de la titulación **Grado en Ingeniería Informática** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77147088M, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Fernando Rafael Talavera Mendoza

Granada a 18 de Noviembre de 2020.

D. Pedro Ángel Castillo Valdivieso, Profesor del Área Arquitectura y Tecnología de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Aplicación Distribuida Basada en Blockchain para dar soporte a IoT, Desarrollo de un registro distribuido para IoT usando la tecnología Blockchain*, ha sido realizado bajo su supervisión por **Fernando Rafael Talavera Mendoza**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 18 de Noviembre de 2020.

Los directores:

Pedro Ángel Castillo Valdivieso

Agradecimientos

A mi familia, por ser mi brújula que me guía. Mi inspiración para llegar a grandes alturas y mi consuelo ante los fallos.

A mis amigos, por estar ahí en todo momento y transmitirme buenos momentos y recuerdos.

Y gracias a mi tutor, Pedro, por animarme en este proyecto y confiar en mí.

Índice

1. Introducción.	8
1.1. Motivación.	8
1.2. Objetivos.	9
1.3. Introducción al Blockchain. Bitcoin.	10
1.3.1. La red Bitcoin. Arquitectura Peer-to-Peer.	10
1.3.2. Blockchain.	12
1.3.3. Minería y consensos.	13
1.4. Internet de las Cosas (IoT).	15
1.5. La seguridad del Blockchain en IoT.	15
1.6. ¿Qué es una Raspberry Pi?	16
1.7. Conclusión.	16
2. Herramientas para el desarrollo blockchain.	17
2.1. Tipos de redes Blockchain.	17
2.2. Plataformas Blockchain.	19
2.2.1. Plataformas de Blockchain populares.	19
2.2.2. Blockchain como plataformas de servicio (BaaS).	21
2.2.3. Otras plataformas Blockchain.	22
2.3. Smart Contracts.	23
2.4. Herramientas utilizadas para desarrollo del proyecto.	24
2.4.1. Implementación de la red Blockchain.	26
2.4.2. Herramientas para el Back end.	28
2.4.3. Herramientas para el Front end.	31
2.5. Conclusión.	32
3. Sistema propuesto.	33
3.1. Análisis de requerimientos.	33
3.1.1. Requisitos funcionales.	33
3.1.2. Requisitos no funcionales.	34
3.2. Especificación.	35
3.2.1. Actores de los casos de uso.	35
3.2.2. Diagramas de casos de uso.	36
3.2.3. Especificación de casos de uso.	38
3.2.4. Diagramas de clase.	57
3.2.5. Diagramas de secuencia.	57
3.3. Planificación y presupuesto del proyecto.	64
4. Desarrollo del sistema.	69
4.1. Configuración y creación de la topología de la red.	69
4.2. Implementación de la red Blockchain.	75
4.3. Desarrollo del Back end.	81
4.4. Desarrollo del Front end y despliegue.	83

Índice de figuras

5. Conclusiones.	85
6. Trabajos futuros.	85
7. Anexo: Vista de la aplicación web e instalación PWA.	87
7.1. Vista de la API.	87
7.2. Vista del Front end.	89
7.3. Instalación de PWA.	91
8. Bibliografía y recursos web	93

Índice de figuras

1. Un nodo de red bitcoin con las cuatro funciones. Fuente: [1]	11
2. Diferentes tipos de nodos en una red extendida bitcoin. Fuente: [1]	12
3. Tipos de redes con sus posibles aplicaciones. Fuente: [47]	17
4. Ejemplo de Smart-Contract con contenido criptográfico. Fuente: [74]	24
5. Ejemplo de instalación de una PWA en el escritorio. Fuente: [38]	26
6. Tres llamadas a endpoints de un REST API para solicitar los datos necesarios. Fuente: [35]	29
7. Una única llamada para solicitar los datos necesarios. Fuente: [35]	29
8. Diagrama de casos de uso de Gestión de usuarios.	36
9. Diagrama de casos de uso de Gestión de dispositivos.	37
10. Diagrama de casos de uso de Gestión de enlaces.	37
11. Diagrama de clases.	57
12. Diagrama de secuencia de Listar dispositivos.	57
13. Diagrama de secuencia de Consultar dispositivo.	58
14. Diagrama de secuencia de Historial dispositivo.	58
15. Diagrama de secuencia de Añadir dispositivo.	59
16. Diagrama de secuencia de Establecer valor a dispositivo.	59
17. Diagrama de secuencia de Borrar dispositivo.	60
18. Diagrama de secuencia de Listar enlaces.	60
19. Diagrama de secuencia de Consultar enlace.	61
20. Diagrama de secuencia de Historial enlace.	61
21. Diagrama de secuencia de Añadir enlace.	62
22. Diagrama de secuencia de Actualizar enlace.	62
23. Diagrama de secuencia de Borrar enlace.	63
24. Diagrama de secuencia de Habilitar enlace.	63
25. Diagrama de secuencia de Deshabilitar enlace.	64
26. Planificación general.	66

Índice de figuras

27.	Planificación expandida.	66
28.	Topología de la red.	70
29.	Captura de las Raspberry Pi.	70
30.	Binarios de Hyperledger Fabric.	73
31.	Imágenes Docker de Hyperledger Fabric.	73
32.	Nodos de Docker Swarm.	74
33.	Tree de la carpeta infraestructure.	75
34.	ficheros de configuración de la red Blockchain.	76
35.	Arquitectura de la red Blockchain.	77
36.	Opción generate del script networkiot.	78
37.	Salidas del comando up.	79
38.	Salida de contenedores en el nodo maestro.	80
39.	Salida de contenedores en el nodo worker 1.	80
40.	Salida de contenedores en el nodo worker 2.	80
41.	Opción remove del script networkiot.	80
42.	Tree de la carpeta server.	81
43.	Ejecución del script launch_api.	82
44.	Tree de la carpeta web.	83
45.	Arquitectura de la aplicación.	84
46.	GraphQL API endpoint.	87
47.	Consulta a GraphQL API endpoint.	87
48.	Información de GraphQL API.	88
49.	Página inicial.	89
50.	Vista dispositivos.	89
51.	Añadir dispositivo.	90
52.	Editar dispositivo.	90
53.	Vista enlaces.	90
54.	Añadir enlace.	91
55.	Editar enlace.	91
56.	Instalación PWA en portatil.	91
57.	App PWA en portatil.	92
58.	Instalación PWA en móvil.	92
59.	Aplicación móvil.	92

Índice de cuadros

1.	Caso de uso 1: Solicitar dar de alta a un usuario.	38
2.	Caso de uso 2: Solicitar dar de baja a un usuario.	39
3.	Caso de uso 3: Dar de alta a un usuario.	40
4.	Caso de uso 4: Dar de baja a un usuario.	41
5.	Caso de uso 5: Renovar el certificado de autoridad del usuario.	42
6.	Caso de uso 6: Añadir un dispositivo.	43
7.	Caso de uso 7: Modificar el valor de entrada de un sensor. . .	44
8.	Caso de uso 8: Borrar un dispositivo.	45
9.	Caso de uso 9: Consultar dispositivos.	46
10.	Caso de uso 10: Consultar los datos de un dispositivo. . . .	47
11.	Caso de uso 11: Consultar el historial de un dispositivo. . .	48
12.	Caso de uso 12: Añadir un enlace.	49
13.	Caso de uso 13: Modificar un enlace.	50
14.	Caso de uso 14: Borrar un enlace.	51
15.	Caso de uso 15: Habilitar un enlace.	52
16.	Caso de uso 16: Deshabilitar un enlace.	53
17.	Caso de uso 17: Consultar enlaces.	54
18.	Caso de uso 18: Consultar los datos de un enlace.	55
19.	Caso de uso 19: Consultar el historial de un enlace. . . .	56
20.	Coste de recursos humanos. Fuente: [41].	67
21.	Coste total de los recursos	68

1. Introducción.

Con el crecimiento de dispositivos inteligentes y las redes de alta velocidad, el Internet de las Cosas se ha convertido en una tecnología con gran influencia, aceptación e impacto dentro de mercados del sector [5].

Ello representa una red interconectada donde dispositivos incorporados con sensores, están interconectados a través de una red privada o pública. Inter cambiando datos comprometedores, críticos para la seguridad y protección, así como información sensible a la privacidad y, por lo tanto, son objetivos atractivos para diversos ciberataques [2].

Un claro ejemplo fue el famoso ciberataque hacia el proveedor de DNS estadounidense Dyn. El ataque se originó de decenas de millones de direcciones IP, y que al menos parte del tráfico provenía de dispositivos IoT. Estos últimos habían sido infectados con un malware llamado Mirai, que se encargaba de controlar los dispositivos en linea y lanzar ataques de denegación de servicio distribuido (DDoS) [4].

Los dispositivos IoT son limitados en cuanto a capacidad computacional, almacenamiento y conectividad, y en consecuencia son más vulnerables a ataques. Por ello, en este proyecto vamos a indagar, sobre cómo la tecnología emergente Blockchain, puede aportar beneficios con el objetivo de fortalecer la seguridad del Internet de las Cosas (IoT). Diversas compañías están tomando iniciativas para integrar la tecnología Blockchain en su producción y suministro (p. ej., IBM, Cisco, Bosch, etc) [4].

1.1. Motivación.

La finalidad de este proyecto es realizar una labor de investigación sobre la tecnología Blockchain, su funcionalidad, tipos de herramientas y capacidades de desarrollo. Para elaborar un prototipo funcional de infraestructura distribuida de gestión de dispositivos IoT (Raspberry Pi) dentro de una red domótica local [3]. La arquitectura distribuida asegurará la comunicación y sincronización de los datos entre los dispositivos, con el fin de llevarlo a dominios que están adaptando la tecnología IoT en sus labores de negocio.

Todo ello regido por contratos inteligentes para permitir realizar cualquier operación, de una forma segura, inmutable y trazable de los dispositivo dentro de la red.

1 Introducción.

Como prueba de concepto, se utilizarán 5 Raspberry Pi:

- 3 Raspberries Pi 4 Modelo B+ 2GB, que servirán como nodos principales de la red Blockchain.
- 2 Raspberry Pi 2 Modelo B+ que tomará el papel del actuador y un sensor (Raspberry Pi 3 Modelo B+), ambos para simular la red domótica.

La configuración, establecida por el usuario, se encargará de indicar políticas entre los sensores y actuadores. El primero recogerá los valores del entorno y el otro efectuará la acción pertinente. Así por ejemplo, el sensor puede detectar una temperatura determinada dentro de la habitación, y en el caso de que sobre pase un umbral, se mandaría una señal al actuador, para encender el aire acondicionado. [3]

Entonces, todos los registros que se han llevado a cabo, se introducirán en la red Blockchain, haciendo que los datos sean inmutables, donde se podrán ver toda la trazabilidad de las transacciones que se han aceptado en la red de forma satisfactoria hasta el momento.

1.2. Objetivos.

Los objetivos del trabajo son los siguientes:

- Realizar un estudio de la tecnología Blockchain y el papel que puede tomar en el escenario del Internet de las cosas, a nivel de soluciones, beneficios y desafíos [5, 7].
- Seleccionar las herramientas existentes en el mercado para desarrollar la infraestructura, y la propia aplicación.
- Elaborar un prototipo para implementar una red distribuida de gestión de dispositivos IoT y una aplicación para su gestión.

1.3. Introducción al Blockchain. Bitcoin.

El Bitcoin fue inventado en 2008 por Satoshi Nakamoto, que combinó la tecnología b-money¹ y HashCash² para crear un sistema completamente descentralizado de dinero electrónico, que no depende de una autoridad central para la emisión de moneda o liquidación y validación de las transacciones [1, 10, 36].

Los usuarios pueden transferir y almacenar bitcoins entre los participantes en la red Bitcoin usando el protocolo bitcoin. El protocolo bitcoin está disponible como software de código abierto, y se puede ejecutar en cualquier dispositivo, lo que hace que la tecnología sea fácilmente accesible [1].

A diferencia de las divisas tradicionales, los bitcoins son enteramente virtuales. Los usuarios poseen una clave virtual que le permite verificar la propiedad las transacciones en la red de bitcoin, desbloqueando el valor para gastarlo y transferirlo a un nuevos destinatarios [1]. Esas llaves se guardan en una cartera digital.

Además, al ser un sistema totalmente distribuido peer-to-peer, no existe una única entidad central que se encargue de validar todas las transacciones realizadas entre los participantes. Si no que son varias entidades, conocidas como “mineros”, que mediante la potencia de procesamiento de su ordenador, intentarán encontrar soluciones a un problema difícil, para validar las transacciones.

Después de esta breve introducción, en las siguientes secciones empezaremos a desenvolver las capas de tecnología que hacen que bitcoin sea posible y examinar el funcionamiento interno de la red y el protocolo de bitcoin.

1.3.1. La red Bitcoin. Arquitectura Peer-to-Peer.

La red Bitcoin está estructurada como una arquitectura de red peer-to-peer. El término peer-to-peer o P2P significa que los participantes, en este caso computadoras, son iguales entre sí y que la carga de proporcionar servicios de red se comparte entre los nodos. No existe un sistema centralizado, son participantes igualmente privilegiados y equipotentes en la aplicación. Los nodos de la red proveen y consumen servicios al mismo tiempo, con

¹B-money fue una de las primeras propuestas creadas por Wei Dai para un “sistema electrónico de dinero en efectivo anónimo y distribuido”.

²Hashcash es un sistema de prueba de trabajo que se utiliza para limitar el spam de correo electrónico y los ataques de denegación de servicio, y más recientemente se ha dado a conocer por su uso en bitcoin (y otras criptodivisas) como parte del algoritmo de minería. Hashcash fue propuesto en 1997 por Adam Back.

1 Introducción.

reciprocidad que actúa como incentivo para la participación. Las redes peer-to-peer son intrínsecamente resistentes, descentralizadas y abiertas [1, 55].

La red bitcoin está formada por una red de nodos P2P. Los nodos pueden asumir diferentes roles dependiendo de la funcionalidad que estén llevando a cabo. Un nodo bitcoin realiza las siguientes funciones (ver fig.1):

- Enrutamiento
- Base de datos de Blockchain
- Minería
- Servicios de cartera

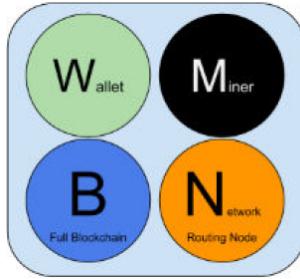


Figura 1: Un nodo de red bitcoin con las cuatro funciones. Fuente: [1]

Todos los nodos incluyen funciones de enrutamiento, validan y propagan transacciones y bloques. Los nodos completos, también mantienen una copia completa y actualizada de la Blockchain. Se encargan también de verificar de forma autónoma y autorizada cualquier transacción sin referencia externa. Los nodos cartera pueden ser parte de un nodo completo, en este caso suele ser un cliente de bitcoin de escritorio.

Algunos nodos mantienen sólo un subconjunto de la cadena de bloque y verifican las transacciones mediante un método llamado verificación de pago simplificado. Estos nodos se conocen como SPV o nodos de peso ligero.

Los nodos mineros compiten para crear nuevos bloques ejecutando hardware especializado para resolver el algoritmo de Proof-of-Work. Pueden ser completos o ligeros.

Además de los principales tipos de nodos del protocolo P2P de bitcoin, hay servidores y nodos que ejecutan otros protocolos, como los protocolos de piscinas mineras especializadas y los protocolos de acceso a clientes ligeros.

1 Introducción.

Aquí están los tipos de nodos más comunes en la red extendida de bitcoin:

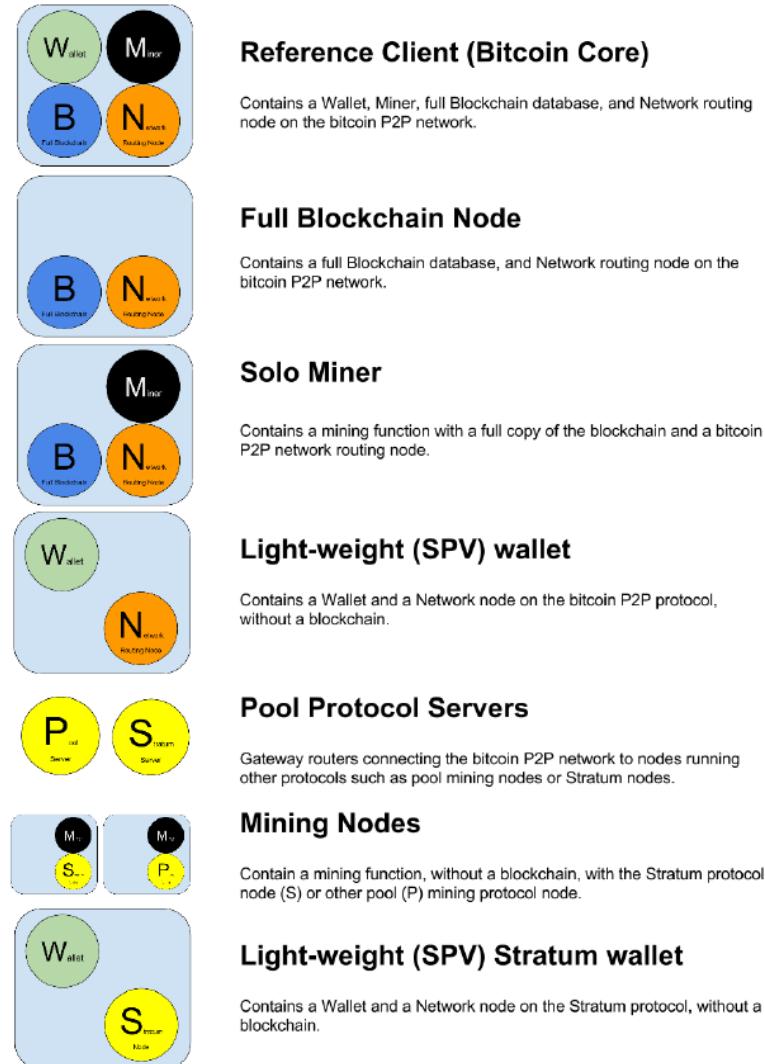


Figura 2: Diferentes tipos de nodos en una red extendida bitcoin. Fuente: [1]

1.3.2. Blockchain.

La tecnología Blockchain, o también conocido como Distributed Ledger Technology, es una lista ordenada de bloques de transacciones vinculados hacia atrás. Cada bloque es identificado por un hash, generado por el algoritmo criptográfico SHA256, almacenado en la cabecera del bloque. Cada bloque está enlazado con el anterior bloque (bloque padre) a través de su hash en la cabecera del bloque. Dicha secuencia, crea una cadena hasta llegar al primer bloque, conocido como el bloque génesis [1].

1 Introducción.

Puede darse el caso de que un bloque tenga varios hijos. Los hijos múltiples surgen cuando se produce un fork del Blockchain, se debe a que diferentes bloques son descubiertos casi simultáneamente por diferentes mineros. Finalmente, un bloque hijo pasa a formar parte del Blockchain y la división se resuelve.

La DLT es inmutable debido a que, en la cabecera del bloque, contiene el hash del padre y, por lo tanto, afecta al bloque actual. La identidad del hijo cambia si la del padre también lo hace. Cuando cambia el padre, el hash de su padre cambia, y por tanto hay que modificar la cabecera del bloque. Esto a su vez cambia el hash del hijo y por tanto requiere un cambio en el puntero del nieto y así sucesivamente [1].

Este efecto de cascada asegura que una vez que un bloque tiene muchas generaciones que lo siguen, no puede ser cambiado sin forzar un recálculo de todos los bloques subsiguientes. La existencia de una larga cadena Blockchain hace que la historia profunda de la cadena de bloques sea inmutable.

1.3.3. Minería y consensos.

La minería es la encargada de validar nuevas transacciones, comprobando si son fraudulentas, y se registran en el libro de contabilidad global. Asimismo, se ocupa de añadir nuevos bitcoins a la oferta monetaria, ya que los mineros proporcionan potencia de procesamiento a la red de bitcoin a cambio de la oportunidad de ser recompensados [1].

Cada 10 minutos se “extrae” un nuevo bloque que contiene las transacciones que se han producido desde el último, añadiendo así, esas transacciones a la cadena de bloques. Las transacciones que pasan a formar parte de un bloque y se añaden a la cadena de bloques se consideran “confirmadas”, lo que permite a los nuevos propietarios de bitcoin gastar el bitcoin recibido en esas transacciones [1].

Los mineros pueden recibir dos tipos de recompensas: las creadas por nuevos bloques y tasas de transacciones de todas las transacciones dentro del bloque. Para ello, tiene que competir contra otros mineros, para encontrar la solución a un problema difícil basado en un algoritmo criptográfico de hash. La solución del problema, llamada Proof-of-Work, se incluye en el nuevo bloque y actúa como prueba de que el minero gastó un esfuerzo de computación significativo. La competencia por resolver el algoritmo de PoW para ganar recompensas y el derecho a registrar las transacciones en la cadena de bloques, es la base del modelo de seguridad de Bitcoin.

La minería es el principal proceso de la cámara de compensación descentra-

lizada, por el cual las transacciones son validadas y autorizadas. La minería asegura el sistema de bitcoin y permite el surgimiento de consenso en toda la red sin una autoridad central [1].

Consenso descentralizado.

A diferencia de los sistemas de pagos tradicionales, el Bitcoin no tiene una autoridad central, sino que cada nodo tiene una copia completa de un libro de cuentas público como registro de autoridad. Cada nodo de la red comparte la información transmitida y, en conclusión, ensamblará una copia del mismo libro público de todos los demás [1].

El consenso descentralizado de Bitcoin surge de la interacción de cuatro procesos que se producen de forma independiente en los nodos de la red:

- Verificación independiente de cada transacción, por cada nodo completo, sobre la base de una lista completa de criterios.
- La agregación independiente de esas transacciones en nuevos bloques por nodos mineros, junto con un cálculo demostrado a través de un algoritmo de prueba de trabajo.
- Verificación independiente de los nuevos bloques por cada nodo y ensamblaje en una cadena.
- Selección independiente, por cada nodo de la cadena, con el cálculo más acumulativo demostrado a través del PoW.

Algoritmo Proof-of-Work.

La idea de PoW fue publicada por primera vez en 1993 por Cynthia Dwork y Moni Naor, y posteriormente fue aplicada por Satoshi Nakamoto en el Bitcoin en 2008 [39].

El problema matemático se puede describir de forma abstracta de la siguiente forma:

Dados los datos A, encuentra un número x como el que el hash de x adjunto a los resultados de A es un número menor que el de B.

La respuesta al problema debe ser un número menor que el hash del bloque para que sea aceptado, conocido como el “hash del objetivo”. Un hash del objetivo es un número que el encabezamiento de un bloque de hash debe ser igual o menor que el de un nuevo bloque, junto con la recompensa, para ser otorgado a un minero. Cuanto más bajo es un objetivo, más difícil es generar un bloque [39].

1 Introducción.

El consenso de prueba de trabajo más utilizado se basa en el SHA-256 y se introdujo como parte de Bitcoin.

Características del algoritmo:

- Es difícil encontrar una solución para el problema matemático
- Es fácil verificar la corrección de esa solución

1.4. Internet de las Cosas (IoT).

El Internet de las Cosas, o también conocido como Internet of Things, consiste en dispositivos que generan procesos, e intercambian grandes cantidades de información a través de Internet. Recolectan y comparten datos sobre cómo son utilizados y sobre el medio que los rodea. Esta información puede ser utilizada para detectar patrones, hacer recomendaciones y detectar posibles problemas antes de que ocurran [14, 6].

Desde usos personales como zapatos inteligentes para proveer apoyo al seguimiento y análisis de los datos de actitud física, como servicios a la comunidad: control de la cirugía en los hospitales, detectar condiciones meteorológicas y proporcionar rastreo y conectividad en los automóviles [5].

Los objetos y sistemas inteligentes permiten automatizar ciertas tareas, especialmente cuando éstas son repetitivas, mundanas, requieren mucho tiempo o incluso son peligrosas.

1.5. La seguridad del Blockchain en IoT.

El uso de la tecnología Blockchain dentro del ámbito de los dispositivos IoT, ha tomado un papel importante en la gestión, control y seguridad. En esta sección, vamos a describir como la tecnología Blockchain puede ser clave para proporcionar soluciones de seguridad a los problemas actuales del Internet de las Cosas [5, 7].

- **Espacio de direcciones:** Blockchain tiene 160-bit de espacio de direcciones, a diferencia de IPv6 que solo tiene 128-bit. Esto es debido a que el hash de la clave pública generada por ECDSA (Elliptic Curve Digital Algorithm) es de 20 bytes o 160-bits. Por lo tanto, podemos alojar alrededor de $1,46 * 10^{48}$ direcciones para dispositivos IoT. Lo que hace que sea más escalable a diferencia de IPv6.
- **Identidad y administración:** Blockchain tiene la capacidad de resolver estos desafíos de manera fácil, segura y eficiente. Se ha utilizado

1 Introducción.

ampliamente para proporcionar un registro de identidad fiable y autorizado, el seguimiento de la propiedad y la supervisión de productos, bienes y activos.

- **Autenticación e integridad de los datos:** Por su diseño, los datos transmitidos siempre estarán criptográficamente probados y firmados, garantizando así la autenticación e integridad de los datos.
- **Autenticación, autorización y privacidad:** Provee una solución efectiva en autorización, privacidad en los datos y la habilidad de suministrar reglas de autenticación decentralizada.
- **Comunicaciones seguras:** Los protocolos que utilizan las IoT son HTTP, MQTT, CoAP o XMPP, y no son seguros por diseño y por tanto, son envueltos por una capa DTLS o TLS. Con Blockchain, la gestión y distribución de claves se elimina totalmente, y cada dispositivo IoT tendrá su par de claves simétricas, para conectarse a la Blockchain.

1.6. ¿Qué es una Raspberry Pi?

La Rasberry Pi es un ordenador de bajo coste hecho por la Raspberry Pi Foundation, del tamaño de una tarjeta de crédito, permite aprender programación, construir proyectos de hardware, hacer automatización del hogar, e incluso a nivel industrial. Además, tiene la capacidad de interactuar con el mundo exterior, y se ha utilizado en una amplia gama de proyectos de creación digital, desde máquinas de música hasta estaciones meteorológicas [76].

El primer modelo lanzado al mercado, tenía una CPU de un solo núcleo de 700MHz y 256MB de RAM [77], hasta hoy en día que tenemos el modelo 4 con Quad core de 1.5GHz y con diferentes versiones de 2GB, 4GB y 8GB DDR4 de RAM [58].

Aunque existen diferentes alternativas, hemos utilizado la Rasberry Pi para este proyecto por comodidad, ya que he podido trabajar antes con diferentes modelos de la marca.

1.7. Conclusión.

En este capítulo hemos dado una introducción al proyecto. Hemos hablando sobre la tecnología Blockchain, apoyandonos con el concepto de Bitcoin. Para pasar ha comentar sobre el Internet de las Cosas, y su combinación con el Blockchain, También hemos comentado sobre los dispositivos que voy a utilizar como prueba de concepto. A continuación, indagaremos más sobre otras tecnologías que han aplicado el Blockchain y plataformas.

2. Herramientas para el desarrollo blockchain.

2.1. Tipos de redes Blockchain.

Cuando la tecnología Blockchain fue introducida al mundo, era del tipo público en el caso de uso de criptomonedas, pero con el paso del tiempo y las posibles aplicaciones que se podían llevar a cabo, dió lugar a diferentes clasificaciones comúnmente aceptadas. Las redes blockchain se pueden clasificar entre públicas o privadas, y dos modos diferentes: abiertas o cerradas. Dando el resultado de cuatro posibles configuraciones [47].

La comparación de estos dos últimos va asociado a quién es capaz de leer esos datos de la Blockchain. Y así, podemos hablar de soluciones que son públicas y abiertas, públicas y cerradas, privadas y abiertas y privadas y cerradas.

Un ejemplo se puede ver a continuación en la siguiente figura (ver fig.3):

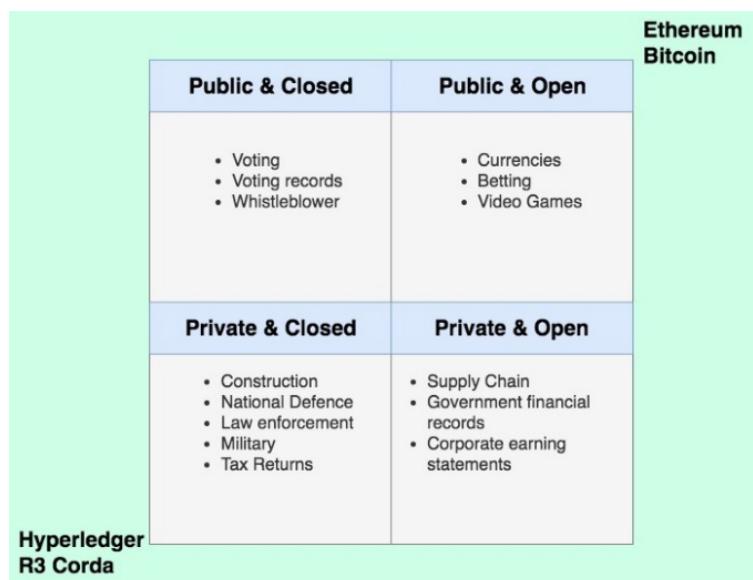


Figura 3: Tipos de redes con sus posibles aplicaciones. Fuente: [47]

En esta sección vamos a definir los diferentes tipos de redes y sus posibles prácticas.

Redes públicas (permissionless).

Cuando hablamos de redes públicas, nos referimos a redes públicas abiertas. Es un tipo de Blockchain que permite que cualquiera pueda unirse a la

red, lo que significa que puede leer, escribir o participar en ella. Las Blockchain públicas están descentralizadas, nadie tiene control sobre la red, y están seguras de que los datos no pueden ser cambiados una vez validados. Este tipo de Blockchain tiende a centrarse más en un escenario B2C o Business to Customer [47].

Las plataformas que podemos encontrar son Bitcoin, Ethereum entre otras criptomonedas. Al no haber permisos, este tipo de redes se centran en proteger el anonimato de los usuarios y por tanto, no se pueden establecer roles y controles en qué datos se pueden leer o escribir.

Redes privadas (permissioned).

En contraparte, las redes Blockchain privadas son del tipo cerradas, ya que queremos controlar quien escribe y lee los datos de la Blockchain, y por ende tenemos que establecer entidades. Se definen reglas sobre qué datos pueden añadir al libro mayor y qué datos pueden consumir del libro mayor. La mayoría de veces, vienen incorporados con herramientas de gestión de identidades (OAuth) [47].

Las Blockchain privadas también son conocidas como Enterprise Blockchains, ya que son las más utilizadas por empresas. Pueden ser Open Source, de consorcio o desarrolladas de forma privada. Las opciones más comunes son Hyperledger, R3 Corda y Quorum. Por esta razón, están estructuradas a un modelo B2B o Business to Business.

Las transacciones son procesadas por una cantidad mínima y conocida de nodos de la Blockchain, en comparación con las redes públicas, como los 8000 nodos en el caso de Ethereum [22]. Hay una clara diferencia en rendimiento entorno a la latencia y velocidades de transmisión.

Beneficios de una red pública.

- **Abierto a lectura y/o escritura:** Cualquiera puede participar enviando transacciones al Blockchain, como Ethereum o Bitcoin; las transacciones pueden verse en el explorador del Blockchain.
- **El libro mayor es distribuido:** No está centralizado como en un enfoque cliente-servidor, y todos los nodos participan en la validación de la transacción.
- **Inmutable.**
- **Seguro gracias a la minería (regla del 51 %):** Por ejemplo, con Bitcoin, la obtención de la mayor parte de la potencia de la red podría

permitir una duplicación masiva de los gastos, y la posibilidad de evitar confirmaciones de transacciones, entre otros actos potencialmente malintencionados.

Beneficios de una red privada.

- **Permisos empresariales:** La empresa controla los recursos y el acceso, por lo tanto, es privada y/o autorizada.
- **Transacciones más rápidas:** Cuando distribuyes los nodos localmente, y tienes una cantidad mínima de nodos, el rendimiento es mejor a la hora de validar las transacciones.
- **Mejor escalabilidad:** El hecho de poder añadir nodos y servicios a petición puede ser una gran ventaja para la empresa.
- **Soporte de cumplimiento:** Como empresa, tiene que satisfacer con requisitos de cumplimiento, y tener el control de su infraestructura, esto permite que se cumpla más fácilmente.
- **Consenso más eficiente (menos nodos):** Las Blockchain empresariales o privadas tienen menos nodos y suelen tener un algoritmo de consenso diferente, como BFT vs PoW.

2.2. Plataformas Blockchain.

Conocemos de antemano, que la tecnología Blockchain surgió en 2008 con la plataforma Bitcoin, y ha evolucionado hasta convertirse en una tecnología dominante, utilizada también a nivel empresarial, y en consecuencia, han surgido nuevas y diversas plataformas de Blockchain. En la sección anterior, hemos comentado algunas plataformas, según su tipo de red. Es el momento de dar paso a definir la lista de las plataformas más conocidas de Blockchain que hay actualmente.

2.2.1. Plataformas de Blockchain populares.

Hyperledger.

Hyperledger es conocida por ser una de las mejores plataformas blockchain de Open-source. Está alojada por la Linux Fundation y permite a los usuarios crear y desarrollar frameworks distribuidos de libro de contabilidad a nivel empresarial [65]. Se creó en 2015 y hay una amplia gama de frameworks dirigidos a diferentes casos de uso [64]. Cabe mencionar los siguientes:

- **Hyperledger Fabric:** es un framework de Blockchain para las empresas. Ofrece una amplia gama de características, incluyendo servicios

de membresía y consenso. Además, también puede alojar contratos inteligentes llamados “chaincode”. IBM y Digital Asses son los primeros en contribuir al proyecto.

- **Hyperledger Sawtooth:** es una plataforma modular, diseñada para desplegar y ejecutar DLT sin una autoridad central [69].
- **Hyperledger Iroha:** es una plataforma de cadena de bloques para la construcción de aplicaciones descentralizadas confiables, seguras y rápidas [69].

Este tipo de plataforma es permissioned [47, 69], por tanto la entrada a la red de un nuevo participante ha de ser aprobada por el resto de nodos, se crean canales de comunicación, donde se implementan la privacidad de transacciones. Se puede catalogar como subredes dentro de la propia red.

R3 Corda.

R3 Corda es una plataforma de cadena de bloques de código abierto. Es una tecnología de libro mayor distribuido [64].

Su enfoque es bastante diferente de la cadena de bloques tradicional, donde una transacción necesita ser verificada por muchos nodos. Su enfoque es reducir el número de validaciones y mejorar la eficiencia mientras se mantiene intacto todo el conjunto de características de la cadena de bloques [64].

Las tres características clave de R3 Corda son el código, diseño y desarrollo abiertos. Cualquiera puede usar R3 Corda y utilizar la plataforma de cadena de bloques de código abierto según sus necesidades y requerimientos [64].

Ethereum.

Ethereum es una de las plataformas blockchain más importantes, destacamos de ella ser pionera en los Smart Contract así como la ejecución simple de los mismos [64]. Creada por Vitalik Buterin en 2013, es una plataforma de código abierto y pública. Su moneda es el Ether. Sigue un mecanismo de consenso PoW, que es comparativamente más lento en velocidad, pero se va a modificar por el algoritmo a prueba de participación o Proof-of-Stake³ [29].

Ethereum tiene, también, un apartado para empresas conocido como Enterprise Ethereum [65]. Ofrece una cadena de bloqueo flexible, segura y escalable dirigida a los negocios. Ofrece tres tipos de implementación, incluyendo

³Proof-of-Stake (PoS) establece que una persona puede minar o validar transacciones dependiendo de la cantidad de monedas que tenga. Cuantas más monedas posea un minero, más poder de minería tendrá.

2 Herramientas para el desarrollo blockchain.

privada, híbrida y de consorcio. Cualquier empresa puede aprovechar las ventajas del Ethereum adaptándolo a sus necesidades. Con las herramientas proporcionadas, puede crear aplicaciones descentralizadas (dApp⁴).

Quorum.

Quorum es una plataforma de código abierto diseñada para mejorar la privacidad. Es una innovación basada en Ethereum, desarrollada en asociación con J.P. Morgan y EthLab, una startup de Ethereum . Modifica el núcleo de Ethereum y, por lo tanto, tiene muchas más características excepcionales añadidas que la tecnología tradicional de cadenas de bloques [64].

Esta plataforma permite implementaciones de consensos diferentes según las necesidades. Ofrece privacidad a nivel de transacción, junto con transparencia en toda la red, y no soporta la política global de intercambio de datos. Actualmente, la plataforma de Quorum se está convirtiendo en una parte indispensable de organizaciones como bancos e instituciones financieras en las que la privacidad de los datos es crucial [64].

Ripple.

Es una plataforma de cadena de bloques de código abierto que está diseñada para permitir transacciones rápidas y baratas. Su criptodivisa es conocida como XRP, pero también permite a la gente crear su propia moneda a través de RippleNet. RippleNet permite a los usuarios encontrar nuevos clientes en nuevos mercados, ampliar los servicios y ofrecer la mejor experiencia en pagos globales. A diferencia de los métodos de transacción tradicionales, esta plataforma tiene por objeto facilitar y agilizar el proceso de transacción, especialmente en el caso de los pagos transfronterizos, creando así un mejor ecosistema de crecimiento y desarrollo [64].

2.2.2. Blockchain como plataformas de servicio (BaaS).

Blockchain-as-a-Service o BaaS, es la creación y gestión por parte de terceros de redes basadas en la nube para empresas que se dedican a la construcción de aplicaciones Blockchain. Estos servicios de terceros son un desarrollo relativamente nuevo en el creciente campo de la tecnología de las Blockchain, El negocio de la tecnología ha superado con creces su uso más conocido en las transacciones en criptografía y se ha ampliado para abordar las transacciones seguras de todo tipo [28, 65]. Las empresas basadas en la nube que implementan la tecnología Blockchain tenemos:

⁴Aplicación software que está construida en una red descentralizada [11].

IBM.

IBM es la empresa pionera en aventurarse en la creación de una plataforma Blockchain para operaciones comerciales transparentes. Posee una división encargada de crear aplicaciones basada en Blockchain [65].

Ofrece una red permissioned, y asegura a las empresas que puedan construir, dirigir, operar y crecer usando la tecnología de IBM. Soporta Golang y Java.

Amazon.

Amazon también ofrece su solución BaaS con el uso de Amazon Managed Blockchain. Con él, puedes crear y gestionar la Blockchain. Funciona con redes de cadenas de bloques populares como Ethereum y Hyperledger Fabric [65].

Tiene los siguientes beneficios:

- Completamente administrado.
- Se puede escoger entre Ethereum o Hyperledger Fabric.
- Seguro y escalable.
- Confiable.

Ofrece una amplia gama de soluciones, incluyendo Amazon QLDB (Quatum Ledger Database), Amazon Managed Blockchain, AWS Blockchain Templates, y AWS Blockchain Partners [65].

2.2.3. Otras plataformas Blockchain.

OpenChain.

Es de código abierto y se dirige principalmente a organizaciones que quieren asegurarse de que sus activos digitales se gestionan y emiten de forma segura, robusta y escalable.

Es fácil de usar y administrar, pudiendo establecer reglas para los usuarios finales que participen e intercambien el valor definido por esas reglas. Cada transacción está firmada digitalmente para garantizar la seguridad. Utiliza el consenso particionado. Cada instancia de Openchain tiene una autoridad para la validez de la transacción, en la que la Blockchain utiliza las instancias de Openchain con fines de descentralización. Esto significa que la transacción debe ser validada por diferentes instancias de OpenChain [65].

2 Herramientas para el desarrollo blockchain.

Otras características fundamentales de OpenChain son una mayor eficiencia gracias al uso de una arquitectura cliente-servidor y la ausencia de minadores. Por tanto, las transacciones no necesitan ningún tipo de tasas y son instantáneas [69].

Stellar.

Stellar es un libro de contabilidad distribuido basado en cadenas de bloques que se utiliza para facilitar las transferencias cruzadas de valor. De manera similar a Ripple, también puede tratar con intercambios entre criptomonedas. Es posible construir herramientas bancarias, dispositivos inteligentes y carteras móviles utilizando la red Stellar [69].

El Protocolo de Consenso Estelar (SCP) permite llegar a un consenso sin depender de un sistema cerrado de registro de las transacciones financieras. Al tener un conjunto de propiedades de seguridad comprobables, el SCP optimiza la seguridad deteniendo el progreso de la red hasta que se pueda llegar a un consenso en caso de que los nodos o la partición se comporten mal [69].

En comparación con los algoritmos descentralizados de prueba de trabajo y prueba de toma, SCP tiene requisitos financieros e informáticos modestos, lo que reduce la barrera de entrada y abre el sistema financiero a nuevos participantes [65, 69].

2.3. Smart Contracts.

Un contrato inteligente es un acuerdo incrustado en un código informático gestionado por una cadena de bloqueo. El código contiene un conjunto de reglas bajo las cuales las partes de ese contrato inteligente acuerdan interactuar entre sí. Si se cumplen las reglas predefinidas, el acuerdo se cumple automáticamente. Los contratos inteligentes proporcionan mecanismos para gestionar eficazmente los activos simbólicos y los derechos de acceso entre dos o más partes [74].

Pueden formalizar las relaciones entre personas e instituciones y los bienes que poseen a través de Internet, totalmente P2P, sin necesidad de intermediarios de confianza. La tecnologías de Blockchain son impulsores del concepto para la implementación de contratos inteligentes.

Los contratos inteligentes, proporcionan una forma pública y verificable de integrar las reglas de gobierno y la lógica empresarial en unas pocas líneas de código, que pueden ser auditadas y aplicadas por el consenso mayoritario de una red P2P [74].

Smart Contracts

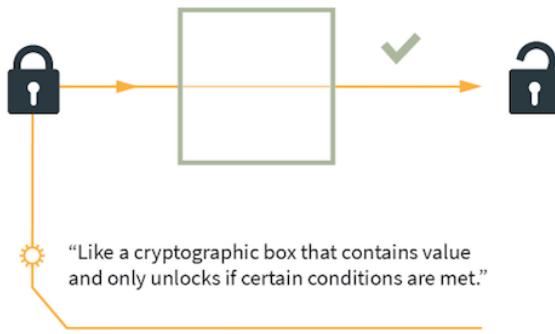


Figura 4: Ejemplo de Smart-Contract con contenido criptográfico. Fuente: [74]

Un contrato inteligente puede ser invocado desde entidades dentro (otros contratos inteligentes) y fuera (fuentes de datos externas) de la Blockchain. Los contratos inteligentes podrían proporcionar una seguridad en las transacciones superior a la del derecho contractual tradicional, con lo que se reducirían los costos de coordinación de la auditoría y el cumplimiento de esos acuerdos [74].

Casos de uso.

Los casos de uso de contratos inteligentes van de simples a complejos. Pueden utilizarse para transacciones económicas sencillas como el envío de dinero de A a B. También pueden utilizarse para registrar cualquier tipo de propiedad y de derechos de propiedad, como los registros de tierras y la propiedad intelectual, o para gestionar el control de acceso inteligente para la economía de reparto. Se pueden encontrar casos de uso en la banca, los seguros, la energía, el gobierno electrónico, las telecomunicaciones, la industria musical, el arte, la movilidad, la educación y muchos más [74].

2.4. Herramientas utilizadas para desarrollo del proyecto.

En esta sección vamos a comentar las herramientas utilizadas en el proyecto y el motivo de por qué he utilizado dichas herramientas para cada apartado de la prueba de concepto. Como ya conocemos en la sección 1.1, comenté sobre los dispositivos que voy a utilizar, que son Raspberry Pi para simular la red Blockchain y los dispositivos IoT que se encargarán de interactuar con la red. Y para la elaboración de la aplicación he decidido implementar una aplicación web que constará de dos partes principalmente. Una API⁵

⁵Una interfaz de programación de aplicaciones es una interfaz informática que define las interacciones entre múltiples intermediarios de software. Define los tipos de llamadas o

2 Herramientas para el desarrollo blockchain.

para hacer consultas directamente a la red Blockchain y un Front end para la visualización de los datos obtenidos desde la API hasta el cliente.

Las aplicaciones basadas en la web son un tipo de software que permite a los usuarios interactuar con un servidor remoto a través de una interfaz de navegador web [75]. Con el paso de los años, ha tomado mucha relevancia y popularidad, sustituyendo a las aplicaciones de escritorio. Tienen varias ventajas sobre las aplicaciones de escritorio tradicionales, una de ellas es la portabilidad. Con las aplicaciones basadas en la web, los usuarios no tienen que instalar software adicional y los desarrolladores no tienen que escribir múltiples versiones de la misma aplicación para diferentes sistemas operativos o plataformas [75], tan sólo se deben de preocupar de las compatibilidades de los diferentes navegadores que hay actualmente (Chrome, Mozilla Firefox, Safari, Opera, etc).

Otras ventajas que ofrece las aplicaciones basadas en web son:

- **Multiplataforma y acceso universal**, para todos los sistemas operativos y móviles
- **Accesibles** en cualquier momento, siempre y cuando se tenga acceso a internet. Aunque como veremos más adelante, podremos interactuar con la aplicación de manera **offline**, gracias a PWA (Progressive Web Application).
- **Ahorro en tiempo y dinero**, al no tener que elaborar la aplicación para múltiples plataformas y lenguajes.
- Son **altamente escalables**. Es fácil aumentar rápidamente la cantidad de usuarios activos, ya que no tiene que ser instalado y configurado. También, gracias a balanceadores de carga para administrar clústeres, permite alta disponibilidad y sin tiempos de inactividad [49].
- Son excelentes para almacenar datos. Los datos son almacenados en la nube, y es el servidor el que se encarga de gestionar la información y distribuirla a los usuarios.
- Son **seguras**.
- El despliegue es fácil, rápido y fáciles de actualizar y mantener.

Estas ventajas de las aplicaciones web para empresas han permitido que el software basado en la web gane popularidad tanto en las corporaciones multinacionales como en las pequeñas empresas. Por tanto, son buenos los

solicitudes que pueden hacerse, cómo hacerlas, los formatos de datos que deben utilizarse, las convenciones a seguir, etc [9].

2 Herramientas para el desarrollo blockchain.

motivos para utilizar las aplicaciones web, para la implementación del proyecto.

Anteriormente, hemos mencionado el concepto de PWA. Se trata de un tipo de aplicación software que se entrega a través de la web, y diversas compañías como Google, Microsoft, Mozilla y Apple, están implementando soporte para ello. Estas aplicaciones son aplicaciones web, pero se comportan más como aplicaciones nativas. Al instalar una aplicación web progresiva (ver fig.5), se podrá tener un acceso directo desde la pantalla de inicio, en la barra de tareas o en el escritorio. La aplicación se cargará rápidamente e incluirá soporte fuera de línea, notificaciones y soporte de sincronización en segundo plano [38].

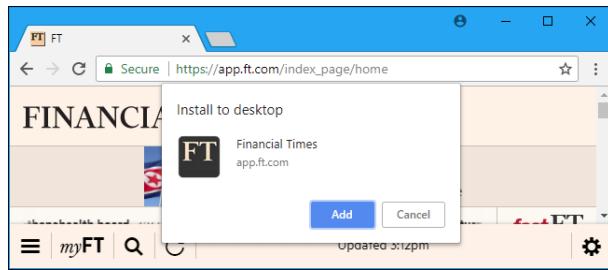


Figura 5: Ejemplo de instalación de una PWA en el escritorio. Fuente: [38]

Este último concepto, ha permitido que aparte de elaborar una aplicación web, poder tener una aplicación tanto para el escritorio como para el móvil⁶ y poder utilizarla sin necesidad de un navegador, ni instalaciones y con todas las ventajas que ofrece las aplicaciones web progresivas.

2.4.1. Implementación de la red Blockchain.

Teniendo en cuenta los objetivos del proyecto para la gestión y administración de dispositivos IoT de una red domótica, era necesario utilizar una tecnología Blockchain privada, para identificar qué dispositivos están escribiendo o leyendo datos y llevar un control de todos ellos. Por tanto, la tecnología que hemos escogido para el proyecto es Hyperledger Fabric, ya que se basa en una red autorizada, y por tanto los participantes han de estar autenticados. También posibilita la creación de contratos inteligentes (chaincode), para poder implementar la lógica de la red. También consta de la creación de canales privados entre los nodos seleccionados para ejecutar la lógica del chaincode sobre el canal, protegiendo los datos [80].

⁶La aplicación ha de ser responsive, para que se pueda adaptar a cualquier tamaño de pantalla.

Hyperledger Fabric ofrece SDKs⁷ para apoyar el desarrollo de contratos inteligentes en varios lenguajes de programación. Hay tres disponibles: Go, Node.js y Java. En cuanto al desarrollo de aplicaciones soporta los lenguajes de Node.js y Java pero actualmente, aunque en desarrollo, tienen Python y Go [8, 40].

En esta ocasión, hemos escogido Go, para la creación de los chaincode y para la parte de la elaboración de la API, nos hemos decantado por Node.js, que más adelante comentaremos en la siguiente sección. Los chaincode se ejecutan en un contenedor Docker seguro y aislado del proceso de aprobación de los peers. Inicializa y gestiona el estado del libro mayor a través de las transacciones presentadas por las solicitudes. Se puede invocar un chaincode dentro de un mismo canal para actualizar y consultar el libro mayor, con los permisos correspondientes.

Hyperledger Fabric también necesita que tengamos instalado Docker y Docker Compose, para la gestión de los diferentes componentes que podemos crear en nuestra red privada. Al ser un arquitectura modular, dichos componentes toman diferentes roles dentro de la red privada [8]. Vamos a comentar uno por uno de los elementos y su labor dentro de una red:

- **Peer:** Mantiene el libro mayor y ejecuta contenedores de chaincode para realizar operaciones de lectura/escritura.
- **Ordering Service (Servicio de pedidos):** El servicio de pedidos se encarga de mandar los bloques de las transacciones para todos los canales de la red y que sean consumidos por los peers. También genera el bloque génesis. Esto lo rige el componente **Orderer**.
- **CA:** La autoridad certificadora de Hyperledger Fabric es el componente predeterminado de la Autoridad de Certificación, que emite certificados basados en la PKI a las organizaciones miembros de la red y a sus usuarios. La CA emite un certificado raíz (rootCert) a cada miembro y un certificado de inscripción (ECert) a cada usuario autorizado.
- **CLI (Command Line Interface):** Hyperledger Fabric posee un CLI para realizar operaciones establecidas en los chaincode dentro de nuestra red. Requiere que esté vinculado a un peer de la red.

A la hora de guardar las transacciones, Hyperledger Fabric ofrece dos tipo de bases datos, LevelDB y CouchDB. LevelDB es la base de datos de estado predeterminada e incrustada en el nodo peer y almacena los datos del

⁷SDK significa software development kit o devkit. Es un conjunto de herramientas de software y programas utilizados por los desarrolladores para crear aplicaciones para plataformas específicas [72].

chaincode como pares clave-valor simples y sólo admite consultas de clave, rango clave y clave compuesta. En cambio, CouchDB es opcional y soporta consultas ricas ya que los datos de los chaincode están en formato JSON. Los datos se encuentran indexados, lo que hace que las consultas sean flexibles y eficientes [71].

Antes de la creación de la red, es necesario establecer qué base de datos se quiere utilizar, en mi caso, me he decantado por utilizar CouchDB por todos los beneficios mencionados anteriormente.

Con Docker y Docker Compose, nos permite crear contenedores ligeros para ejecutar cada componente de la red, con la ventaja de ser cada uno portable y seguro por la capacidades de aislamiento que tienen. Con Docker Compose podemos definir y ejecutar aplicaciones de varios contenedores Docker. Y mediante comandos, podemos crear e iniciar todos los servicios que estén configurados.

Al utilizar varias Raspberries como nodos principales de la red Blockchain, como un clúster, voy a utilizar Docker Swarm para la orquestación de múltiples contenedores dentro de múltiples máquinas anfitrionas. Dentro de un clúster hay varios nodos trabajadores y un nodo administrador, que se encarga de manejar los recursos de los nodos trabajadores de manera eficiente y de asegurar que el grupo funcione con eficacia. También, hace que la aplicación tenga un alto nivel de disponibilidad [78].

2.4.2. Herramientas para el Back end.

GraphQL.

En esta sección hablaremos sobre la creación de la API con Node.js y utilizando el estandar de GraphQL. GraphQL es un lenguaje de código abierto de consulta y manipulación de datos para APIs. Fue desarrollado por Facebook en 2012 y publicado en 2015 [34]. Hoy en día REST se ha convertido en una estandar para el diseño de web APIs. Sin embargo, es inflexible a la hora de obtener los datos según lo que solicita el cliente. Con REST es necesario realizar varias llamadas a diferentes endpoints con estructura de datos fijas (ver fig.6), y por tanto se produce insuficiencia o exceso de los datos que realmente se requieren [35].

Gracias a GraphQL, basta con mandar solo una consulta con los datos que se quiera en concreto, y el servidor responde con un objeto JSON con los requerimientos llenos (ver fig.7) [35].

2 Herramientas para el desarrollo blockchain.



Figura 6: Tres llamadas a endpoints de un REST API para solicitar los datos necesarios. Fuente: [35]

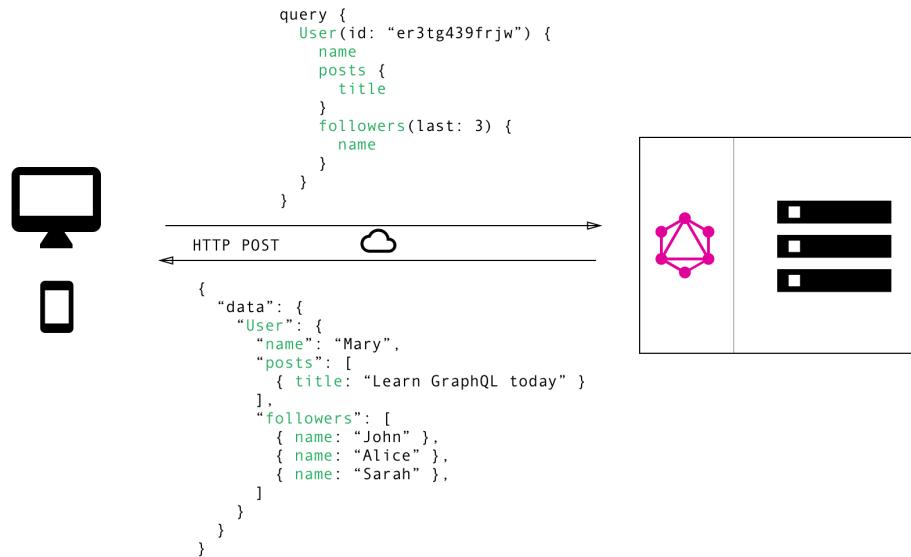


Figura 7: Una única llamada para solicitar los datos necesarios. Fuente: [35]

En GraphQL es necesario definir esquemas para indicar las capacidades de una API. Una vez definido el esquema, tanto el Front como el Back son

conscientes de la estructura definitiva de los datos.

Node.js

Node.js es un entorno de ejecución que permite ejecutar código JavaScript fuera del navegador, utilizando Chrome V8 como motor de JavaScript. Este entorno permite escribir comandos de línea en una terminal y “scripting” en el lado del servidor [54].

Usa un gestor de paquetes JavaScript por defecto, llamado NPM (Node Package Manager). Fue fundada en 2014 y adquirida por Github en 2020. Es un proyecto de código abierto para ayudar a los desarrolladores de JavaScript a compartir fácilmente módulos de código empaquetado. Mantiene una colección pública de paquetes de código abierto para Node.js, aplicaciones web de front-end, aplicaciones para móvil, robots, enruteadores, etc. El cliente de línea de comandos permite a los desarrolladores instalar y publicar esos paquetes.

La razón por el cual he escogido Node.js es por utilizar el SDK que ofrece Hyperledger Fabric para la creación de aplicaciones sobre redes Blockchain, por la buena comunidad que tiene el lenguaje y la cantidad de paquetes que ofrece, lo que facilita el desarrollo de la aplicación web.

Entre los paquetes que he instalado en el lado del servidor caben destacar:

- **fabric-network:** envio de transacciones y consultas de contratos inteligentes.
- **fabric-ca-client:** contiene los servicios para la administración de usuarios, añadir, revocar por ID o por certificado, y se puede personalizar la persistencia.
- **graphql:** necesario para la implementación de la API de lenguaje de consulta de JavaScript.
- **Apollo Server:** creación de servidor GraphQL y es compatible con Apollo Client. Permite definir esquemas con literales de GraphQL [44]. Es un complemento para el middleware Node.js como Express. Tiene una buena documentación.
- **Express:** es framework web ligero para ayudar a organizar una aplicación web en una arquitectura MVC en el lado del servidor [53].

SDK Node.js de Hyperledger Fabric.

El SDK que ofrece Hyperledger Fabric de Node.js, permite interactuar con la red Blockchain de Hyperledger utilizando los certificados TLS para asegurar

2 Herramientas para el desarrollo blockchain.

el envío de solicitudes. La seguridad de la red se hace con firmas digitales. Un usuario se considera válido, si está firmado por un CA (autoridad de certificación) de confianza.

Con SDK de Node.js se pueden ejecutar las siguientes acciones:

- Crear canales.
- Pedir a los nodos iguales que se unan al canal.
- Instalar chaincodes en pares.
- Instanciar códigos de cadena en un canal.
- Invocar transacciones llamando al chaincode.
- Consultar el libro mayor para transacciones o bloques.

2.4.3. Herramientas para el Front end.

Tras hablar sobre las herramientas y frameworks utilizados en el lado del servidor, en esta sección vamos a hablar sobre las herramientas utilizadas en el lado del cliente.

Gatsby.

Gatsby es un generador de sitios estáticos basado en React, alimentado por GraphQL. Utiliza las mejores partes de React, webpack, react-router, GraphQL, y otras herramientas de cliente [31, 37].

Utiliza una poderosa preconfiguración para construir un sitio web que utiliza sólo archivos estáticos para cargas de página increíblemente rápidas, trabajadores de servicio, división de código, renderizado del lado del servidor, carga inteligente de imágenes, optimización de activos y pre-recolocación de datos [37, 50].

Gatsby está muy bien integrado con GraphQL para construir su capa de datos. Permite recopilar datos desde donde sea que estén: Markdown, JSON, CMS, APIs de terceros, etc. Y en el momento de la construcción, crea un servidor interno de GraphQL de todos estos datos. Todos los componentes de React se construyen mediante consultas a través de GraphQL.

Cuenta con una gran documentación y riqueza de plugins que puedes añadir a tu aplicación web. Está dedicado al rendimiento y a la accesibilidad, permitiendo crear PWA de manera sencilla y fácil [50].

2 Herramientas para el desarrollo blockchain.

React es una biblioteca creada por Facebook, basada en JavaScript, que permite construir interfaces de usuario mediante componentes y dispone de una gran cantidad de documentación con una buena comunidad por detrás [59].

Existen diversas tecnologías similares a React, como Angular o Vue. Angular es un framework más maduro, creada por Google que tiene un buen respaldo en términos de contribuyentes, sin embargo, su curva de aprendizaje es empinada y más compleja que React. Vue no tiene respaldo de ninguna compañía importante, se fundamenta por contribuyentes al proyecto. Posee menos documentación y es menos conocido [18].

También se ha utilizado Apollo Client, que es una biblioteca completa de gestión de estado para JavaScript que permite gestionar datos locales y remotos con GraphQL. Sirve para obtener, almacenar y modificar datos de aplicaciones, mientras se actualiza la interfaz de usuario. El núcleo de la biblioteca de Apollo Client, proporciona una integración con React y Apollo Server [43].

Bootstrap.

Bootstrap es un framework código abierto y gratuito de CSS dirigido al desarrollo web móvil. Contiene plantillas de diseño basadas en CSS y (opcionalmente) en JavaScript para la tipografía, los formularios, los botones, la navegación y otros componentes de la interfaz [12].

Se ha utilizado la biblioteca Reactstrap que contiene componentes de React Bootstrap 4 que favorecen la composición y el control [60].

Para este proyecto, se ha elegido Bootstrap porque tiene un patrón más dirigido al diseño responsive, adaptado a dispositivos móviles, que frameworks de su competencia como Materialize.

2.5. Conclusión.

En este capítulo se han revisado los tipos de redes Blockchain, dependiendo del uso que se quiere llevar a cabo, y también se ha indagado en las plataformas más conocidas que se utilizan hoy en día basadas en la tecnología Blockchain, tanto para uso personal como empresarial. Se ha tratado sobre el concepto de Smart Contract, como sistema sustituto a los métodos tradicionales. Finalmente, hemos comentado sobre las herramientas que se han escogido para la elaboración de la red Blockchain y para la aplicación web.

3. Sistema propuesto.

En esta sección vamos a exponer los requisitos extraídos del análisis de la descripción del proyecto, que se van a especificar en los requisitos funcionales y no funcionales, los caso de uso, sus actores y diagramas de clase y secuencia. Se comentará sobre la planificación y presupuesto que se ha llevado a cabo.

3.1. Análisis de requerimientos.

3.1.1. Requisitos funcionales.

RF-1. Gestión de usuarios. Habrá dos actores principales: los usuarios finales y el administrador.

RF-1.1. El usuario podrá solicitar darse de alta.

RF-1.2. El usuario podrá solicitar darse de baja.

RF-1.3. El administrador podrá dar de alta a un usuario en la red Blockchain.

RF-1.4. El administrador podrá revocar el certificado de un usuario en la red Blockchain.

RF-1.5. El administrador podrá renovar el certificado de un usuario en la red Blockchain.

RF-2. Gestión de dispositivos. El sistema permitirá la gestión de dispositivos IoT de los usuarios, dentro de la aplicación. Podemos catalogar los dispositivos entre: sensores y actuadores.

RF-2.1. El usuario tendrá permitido añadir un dispositivo indicando un nombre, su número serial y su dirección IP.

RF-2.2. El usuario podrá establecer un valor numérico al sensor como valor de entrada y se comprobará las condiciones de sus enlaces en el cual se encuentre involucrado.

RF-2.3. El usuario no podrá establecer un valor numérico al actuador, se dejará por defecto el -1.

RF-2.4. El usuario podrá borrar el dispositivo de la red, indicando el número serial.

RF-2.5. El usuario podrá consultar todos los dispositivos añadidos a la aplicación.

RF-2.6. El usuario podrá consultar los datos del dispositivo indicando el número serial.

3 Sistema propuesto.

RF-2.7. El usuario podrá consultar el historial de un dispositivo indicando el número serial.

RF-3. Gestión de enlaces. El sistema permitirá la gestión de enlaces de dispositivos IoT dentro de la aplicación. Los enlaces vienen a representar la relación que hay entre un sensor y un actuador, mediante una condición de activación.

RF-3.1. El usuario tendrá permitido añadir un enlace indicando los dispositivos que lo forman (el número serial del sensor y el número serial del actuador), la región donde se encuentre el dispositivo, y una condición, formada por una variable, acompañado de un operador de comparación y un valor numérico. Al añadirse, tendrá su estado activado por defecto.

RF-3.2. El usuario podrá actualizar un enlace, indicando su ID (hash del número serial del sensor y el actuador). Podrá cambiar la condición y la región.

RF-3.3. El usuario podrá borrar un enlace, indicando su ID.

RF-3.4. El usuario podrá habilitar un enlace, y se mandará una señal de activación al actuador.

RF-3.5. El usuario podrá deshabilitar un enlace, y se mandará una señal de desactivación al actuador.

RF-3.6. El usuario podrá consultar todos los enlaces añadidos a la aplicación.

RF-3.7. El usuario podrá consultar los datos del enlace indicando el ID del mismo.

RF-3.8. El usuario podrá consultar el historial de un enlace indicando el ID del mismo.

3.1.2. Requisitos no funcionales.

RNF-1. La aplicación ha de ser accesible desde fuera de la red local, con cualquier dispositivo, smartphone, table, portátil o sobremesa

RNF-2. La aplicación tiene que ser compatible con diferentes navegadores.

RNF-3. Se establecerán conexiones seguras dentro de la aplicación, cifrando las peticiones HTTP.

3 Sistema propuesto.

RNF-4. La aplicación tiene que cumplir los criterios de PWA, tiene que trabajar de forma offline, mandar notificaciones y soportar la sincronización en segundo plano.

RNF-5. La aplicación tiene que adaptarse a cualquier dimensión de pantalla, es decir, tiene que ser responsive.

RNF-6. La aplicación tiene que cumplir niveles de rendimiento, accesibilidad y SEO⁸ aceptables, utilizando la aplicación LightHouse⁹.

RNF-7. La interfaz ha de ser clara e intuitiva.

RNF-8. Se realizarán test a la aplicación.

RNF-9. No se requerirá de conocimientos previos para el uso de la aplicación.

RNF-10. La API tiene que ser accesible desde la aplicación, para realizar cualquier petición directamente a la red Blockchain.

3.2. Especificación.

3.2.1. Actores de los casos de uso.

- **Entorno:** es el actor vinculado al medio ambiente, se incluye lo referente al aire, la temperatura, la humedad, el tiempo atmosférico, etc. Los sensores se van a encargar de recoger esos datos del exterior y manejarlos.
- **Dispositivo:** es el actor que realiza un mecanismo con una función específica. Podemos diferenciar entre sensores y actuadores. Nos referimos a los dispositivos de IoT.
- **Enlace:** es el actor que está formado por dos dispositivos, un sensor y un actuador. Un enlace va a estar condicionado por el cumplimiento de la condición establecida por el usuario. Por defecto, el enlace creado va a estar habilitado. Cuando un sensor recibe un valor, se comprobará cada uno de sus enlaces establecidos. En el caso de que se cumpla la condición, el enlace se habilita y se mandará una señal de activación al actuador.

⁸SEO significa Search Engine Optimization (optimización de motores de búsqueda), que es la práctica de aumentar la cantidad y la calidad del tráfico a un sitio web a través de los resultados orgánicos de los motores de búsqueda [79].

⁹Es una herramienta elaborada por Google de código abierto para mejorar la calidad de las páginas web [46].

3 Sistema propuesto.

- **Usuario:** es el actor que puede usar las principales funcionalidades que ofrece la aplicación una vez que tenga un certificado de autoridad dentro de la red Blockchain. Podrá llevar una gestión de sus dispositivos y enlaces dentro de la aplicación y gracias a la tecnología Blockchain, podrá ver toda la trazabilidad de las transacciones realizadas y resistente a modificaciones.
- **Administrador:** es el actor encargado de la gestión de usuarios, se encargará de administrar todo lo relacionado con los certificados de autoridad de cada usuario del sistema.

3.2.2. Diagramas de casos de uso.

- Gestión de usuarios.

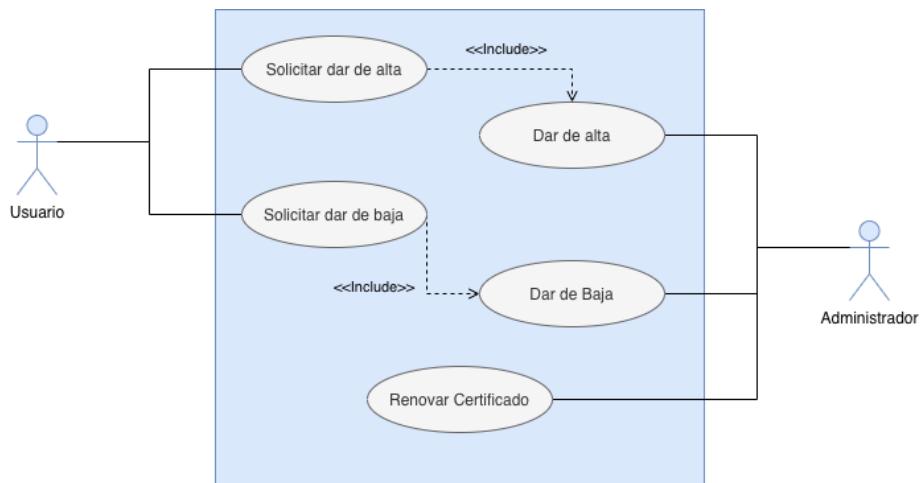


Figura 8: Diagrama de casos de uso de Gestión de usuarios.

3 Sistema propuesto.

- Gestión de dispositivos.

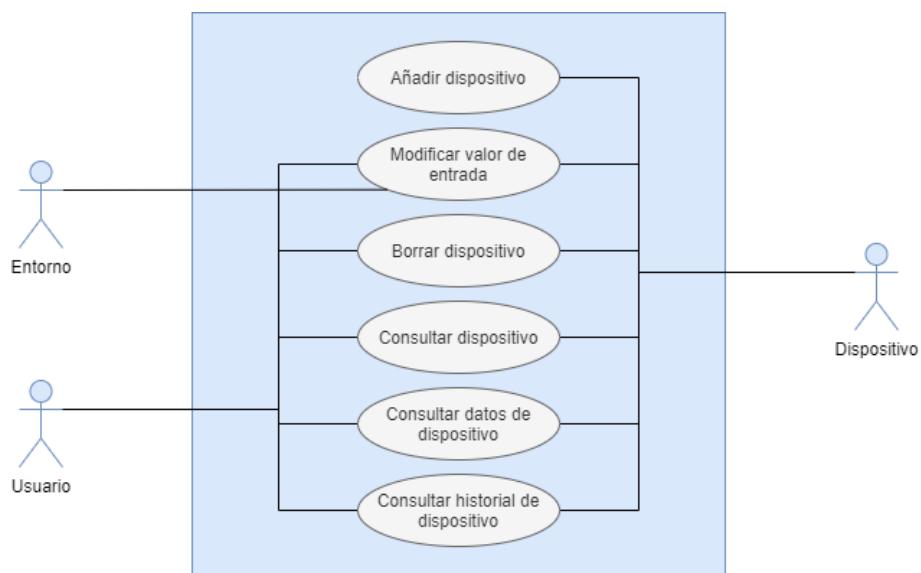


Figura 9: Diagrama de casos de uso de Gestión de dispositivos.

- Gestión de enlaces.

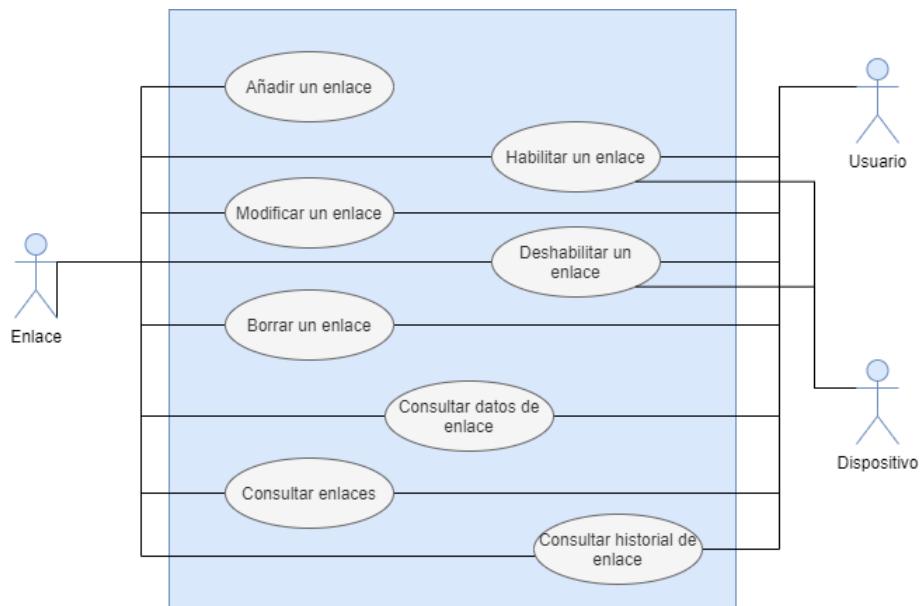


Figura 10: Diagrama de casos de uso de Gestión de enlaces.

3 Sistema propuesto.

3.2.3. Especificación de casos de uso.

- Caso de uso 1: Solicitar dar de alta a un usuario.

Casos de Uso	Solicitar dar de alta a un usuario (CU_01).
Actores	Usuario (Principal) y Administrador (Secundario).
Tipo	Primario y esencial.
Referencias	RF-1.1., RF-1.3.
Precondición	El usuario no debe de tener un certificado de autoridad asociada en la red.
Postcondición	El usuario tendrá un certificado de autoridad.

Propósito
El usuario obtendrá el certificado de autoridad para poder realizar operaciones en el sistema, firmando las transacciones.

Resumen
El usuario solicitará al administrador la petición de darse de alta del sistema. El administrador se encargará de crear el certificado de autoridad del usuario y lo guardará en la cartera, para que el usuario pueda realizar acciones en la aplicación.

Curso Normal			
1	Usuario: manda petición de darse de alta.		
2	Administrador: manda petición para crear el certificado de autoridad.		
		3	El sistema crea el certificado de autoridad.
		4	El sistema guarda el certificado en la cartera (wallet).

Otros datos			
Frecuencia esperada	Media	Rendimiento	
Importancia	Alta	Urgencia	Alta
Estado		Estabilidad	Alta

Cuadro 1: Caso de uso 1: Solicitar dar de alta a un usuario.

3 Sistema propuesto.

- Caso de uso 2: Solicitar dar de baja a un usuario.

Casos de Uso	Solicitar dar de baja a un usuario (CU_02).
Actores	Usuario (Principal) y Administrador (Secundario).
Tipo	Primario y esencial.
Referencias	RF-1.2., RF-1.4.
Precondición	El usuario debe de tener un certificado de autoridad asociada en la red.
Postcondición	El usuario no tendrá un certificado de autoridad.

Propósito
El usuario perderá el certificado de autoridad y por tanto no podrá realizar ninguna operación en la red.

Resumen
El usuario solicitará al administrador la petición de darse de baja del sistema. El administrador se encargará de revocar el certificado de autoridad del usuario. El certificado de autoridad se borrará de la cartera (wallet).

Curso Normal			
1	Usuario: manda petición de darse de baja.		
2	Administrador: manda petición para revocar el certificado de autoridad.		
		3	El sistema revoca el certificado de autoridad.
		4	El sistema borra el certificado de la cartera (wallet).

Otros datos			
Frecuencia esperada	Media	Rendimiento	
Importancia	Alta	Urgencia	Alta
Estado		Estabilidad	Alta

Cuadro 2: Caso de uso 2: Solicitar dar de baja a un usuario.

3 Sistema propuesto.

- Caso de uso 3: Dar de alta a un usuario.

Casos de Uso	Dar de alta a un usuario (CU_03).
Actores	Administrador (Principal) y Usuario (Secundario)
Tipo	Primario y esencial.
Referencias	RF-1.3., RF-1.1.
Precondición	El Administrador debe de recibir una solicitud de dar de alta.
Postcondición	El certificado de autoridad se guarda en el sistema.

Propósito
El administrador mandará la solicitud al sistema, para crear el certificado de autoridad.

Resumen
El administrador se encargará de crear el certificado de autoridad del usuario y lo guardará en la cartera, para que el usuario pueda realizar acciones en la aplicación.

Curso Normal			
1	Administrador: manda petición para crear el certificado de autoridad.		
		2	El sistema crea el certificado de autoridad.
		3	El sistema guarda el certificado en la cartera (wallet).

Otros datos			
Frecuencia esperada	Media	Rendimiento	
Importancia	Alta	Urgencia	Alta
Estado		Estabilidad	Alta

Cuadro 3: Caso de uso 3: Dar de alta a un usuario.

3 Sistema propuesto.

- Caso de uso 4: Dar de baja a un usuario.

Casos de Uso	Dar de baja a un usuario (CU_04).
Actores	Administrador (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-1.4., RF-1.2.
Precondición	Debe de existir un certificado de autoridad asociado al usuario.
Postcondición	Se revoca el certificado.

Propósito
El administrador mandará la solicitud al sistema, para revocar el certificado de autoridad del usuario.

Resumen
El administrador se encargará de revocar el certificado de autoridad del usuario. El certificado de autoridad se borrará de la cartera (wallet).

Curso Normal			
1	Administrador: manda petición para revocar el certificado de autoridad.		
		2	El sistema revoca el certificado de autoridad.
		3	El sistema borra el certificado de autoridad de la cartera (wallet).

Otros datos			
Frecuencia esperada	Media	Rendimiento	
Importancia	Alta	Urgencia	Alta
Estado		Estabilidad	Alta

Cuadro 4: Caso de uso 4: Dar de baja a un usuario.

3 Sistema propuesto.

- Caso de uso 5: Renovar el certificado de autoridad del usuario.

Casos de Uso	Renovar el certificado de autoridad del usuario (CU_05).
Actores	Administrador (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-1.5., RF-1.4.
Precondición	Debe de existir un certificado de autoridad asociado al usuario y tiene que haber expirado la fecha de finalización del certificado.
Postcondición	Se emite un nuevo certificado de autoridad para el usuario.

Propósito

El administrador mandará la solicitud al sistema, para renovar el certificado de autoridad del usuario.

Resumen

El administrado manda la petición de renovación, que consiste en la emisión de un nuevo certificado para que el sistema (CA) extienda la vida útil de la misma más allá de la fecha de finalización del certificado original.

Curso Normal

1	Administrador: comprueba la fecha de expiración del certificado.		
2	Administrador: manda la petición para renovar el certificado.		
		3	El sistema renueva el certificado de autoridad.
		4	El sistema guarda el certificado en la cartera (wallet).

Otros datos

Frecuencia esperada	Media	Rendimiento	Alta
Importancia	Alta	Urgencia	Media
Estado		Estabilidad	Alta

Cuadro 5: Caso de uso 5: Renovar el certificado de autoridad del usuario.

3 Sistema propuesto.

- Caso de uso 6: Añadir un dispositivo.

Casos de Uso	Añadir un dispositivo (CU_06).
Actores	Dispositivo (Principal).
Tipo	Primario y esencial.
Referencias	RF-2.1.
Precondición	No debe de existir un dispositivo con el mismo número serial o dirección IP en el sistema.
Postcondición	Se añade el nuevo dispositivo con valor -1.

Propósito
El usuario podrá añadir un dispositivo que no esté registrado anteriormente.

Resumen
El usuario introducirá los datos asociados con el dispositivo, que son el nombre, el número serial y la dirección IP. Se comprobará que no haya un dispositivo existente con el número serial y la dirección IP iguales, en ese caso se añadirá el dispositivo con el atributo valor -1 por defecto.

Curso Normal			
1	Usuario: introducirá los datos del dispositivo.		
		2	El sistema comprueba los datos introducidos.
		3	El sistema valida los datos y guarda el nuevo dispositivo en la red Blockchain.

Otros datos			
Frecuencia esperada	Alta	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 6: Caso de uso 6: Añadir un dispositivo.

3 Sistema propuesto.

- Caso de uso 7: Modificar el valor de entrada de un sensor.

Casos de Uso	Modificar el valor de entrada de un sensor (CU_07).
Actores	Dispositivo (Principal), Entorno (Secundario) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-2.2., RF-2.3.
Precondición	El dispositivo debe de estar registrado en el sistema y tiene que ser un sensor.
Postcondición	El valor del sensor se habrá modificado y se habrá comprobado las condiciones de sus enlaces.

Propósito

El entorno o el usuario podrá modificar los valores de entrada de un sensor para habilitar o deshabilitar enlaces en el cual se encuentre involucrado.

Resumen

Los datos que se obtiene del entorno o los datos que introduce un usuario a un sensor, el sensor guardará el dato obtenido y se comprobará los enlaces que esté involucrado el dispositivo, para habilitar o deshabilitar el enlace de acuerdo a la condición establecida en el enlace.

Curso Normal

1	Entorno o Usuario: Introduce el dato de entrada al sensor.		
2	Dispositivo (Sensor): guarda el valor de entrada.		
		3	El sistema guarda el dato.
		4	El sistema comprueba los enlaces.

Otros datos

Frecuencia esperada	Muy Alta	Rendimiento	Alta
Importancia	Muy Alta	Urgencia	Alta
Estado		Estabilidad	Alta

Cuadro 7: Caso de uso 7: Modificar el valor de entrada de un sensor.

3 Sistema propuesto.

- Caso de uso 8: Borrar un dispositivo.

Casos de Uso	Borrar un dispositivo (CU_08).
Actores	Dispositivo (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-2.4.
Precondición	El dispositivo de estar registrado en el sistema.
Postcondición	Se ha borrado el dispositivo.

Propósito
El usuario podrá borrar el dispositivo seleccionado.

Resumen
El sistema facilitará al usuario la opción de borrar un dispositivo.

Curso Normal			
1	Usuario: Selecciona el dispositivo a borrar.		
		2	El sistema borra el dispositivo.

Otros datos			
Frecuencia esperada	Baja	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 8: Caso de uso 8: Borrar un dispositivo.

3 Sistema propuesto.

- Caso de uso 9: Consultar dispositivos.

Casos de Uso	Consultar dispositivos (CU_09).
Actores	Dispositivo (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-2.5.
Precondición	
Postcondición	El usuario obtendrá una lista de dispositivos.

Propósito
Permitir al usuario poder consultar los dispositivos guardados.

Resumen
El usuario podrá consultar los dispositivos guardados y ver los datos de cada uno de ellos.

Curso Normal			
1	Usuario: solicita consultar los dispositivos.		
		2	El sistema devuelve una lista de los dispositivos guardados.

Otros datos			
Frecuencia esperada	Alta	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 9: Caso de uso 9: Consultar dispositivos.

3 Sistema propuesto.

- Caso de uso 10: Consultar los datos de un dispositivo.

Casos de Uso	Consultar los datos de un dispositivo (CU_10).
Actores	Dispositivo (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-2.6.
Precondición	El dispositivo debe de estar registrado en el sistema.
Postcondición	El usuario obtendrá los datos del dispositivo.

Propósito
Permitir al usuario poder consultar los datos de un dispositivo.

Resumen
El usuario seleccionará el dispositivo que quiere consultar y el sistema se lo proporcionará.

Curso Normal			
1	Usuario: solicita consultar los datos del dispositivo.		
		2	El sistema comprueba si existe el dispositivo.
		3	El sistema devuelve los datos del dispositivo guardado.

Otros datos			
Frecuencia esperada	Alta	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 10: Caso de uso 10: Consultar los datos de un dispositivo.

3 Sistema propuesto.

- Caso de uso 11: Consultar el historial de un dispositivo.

Casos de Uso	Consultar el historial de un dispositivo (CU_11).
Actores	Dispositivo (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-2.7.
Precondición	El dispositivo debe de estar registrado en el sistema.
Postcondición	El usuario obtendrá el historial de un dispositivo.

Propósito
Permitir al usuario consultar el historial de un dispositivo.

Resumen
Gracias al tecnología Blockchain, se puede obtener todas las transacciones realizadas y con ello poder ver los diferentes cambios que ha tenido un dispositivo, desde su momento de creación hasta ahora.

Curso Normal			
1	Usuario: solicita consultar el historial del dispositivo.		
		2	El sistema comprueba si existe el dispositivo.
		3	El sistema devuelve el historial del dispositivo guardado.

Otros datos			
Frecuencia esperada	Alta	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 11: Caso de uso 11: Consultar el historial de un dispositivo.

3 Sistema propuesto.

- Caso de uso 12: Añadir un enlace.

Casos de Uso	Añadir un enlace (CU_12).
Actores	Enlace (Principal).
Tipo	Primario y esencial.
Referencias	RF-3.1.
Precondición	No debe de existir un enlace con el mismo sensor y actuador, además deben de existir ambos dispositivos.
Postcondición	Se añade el nuevo enlace y está habilitado.

Propósito
El usuario podrá añadir un nuevo enlace que no esté registrado anteriormente.

Resumen
El usuario introducirá el número serial del sensor y del actuador, una región donde se localice dicho enlace y por último, una condición. Dicha condición va a estar formado por una variable "value", acompañado de un operador de comparación y un valor numérico. Se generará un ID que es el hash del número serial del sensor y el actuador.

Curso Normal			
1	Usuario: introducirá los datos del enlace.		
		2	El sistema comprueba los datos introducidos.
		3	El sistema valida los datos y guarda el nuevo enlace en la red Blockchain.

Otros datos			
Frecuencia esperada	Alta	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 12: Caso de uso 12: Añadir un enlace.

3 Sistema propuesto.

- Caso de uso 13: Modificar un enlace.

Casos de Uso	Modificar un enlace (CU_13).
Actores	Enlace (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-3.2., RF-3.4., RF-3.5.
Precondición	El enlace debe de estar registrado en el sistema.
Postcondición	Los datos del enlace se habrán actualizado.

Propósito
El usuario podrá modificar la región, el estado o la condición del enlace.

Resumen
El usuario, verá los datos del enlace, modificará lo que desee y el sistema actualizará esos datos.

Curso Normal			
1	Usuario: introducirá los datos del enlace.		
		2	El sistema comprueba los datos introducidos.
		3	El sistema valida los datos y guarda el nuevo enlace en la red Blockchain.

Otros datos			
Frecuencia esperada	Muy Alta	Rendimiento	Alta
Importancia	Muy Alta	Urgencia	Alta
Estado		Estabilidad	Alta

Cuadro 13: Caso de uso 13: Modificar un enlace.

3 Sistema propuesto.

- Caso de uso 14: Borrar un enlace.

Casos de Uso	Borrar un enlace (CU_14).
Actores	Enlace (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-3.3.
Precondición	El enlace de estar registrado en el sistema.
Postcondición	Se ha borrado el enlace.

Propósito
El usuario podrá borrar el enlace seleccionado.

Resumen
El sistema facilitará al usuario la opción de borrar un enlace.

Curso Normal			
1	Usuario: Selecciona el enlace a borrar.		
		2	El sistema borra el enlace.

Otros datos			
Frecuencia esperada	Baja	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 14: Caso de uso 14: Borrar un enlace.

3 Sistema propuesto.

- Caso de uso 15: Habilitar un enlace.

Casos de Uso	Habilitar un enlace (CU_15).
Actores	Enlace (Principal), Dispositivo (Secundario) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-3.4.
Precondición	El enlace debe de estar registrado en el sistema.
Postcondición	El enlace se habilita.

Propósito
El usuario y el sensor, podrán habilitar el enlace.

Resumen
El enlace se podrá habilitar dependiendo de el valor de entrada recibido por el sensor, en el caso de que se cumpla la condición. También el usuario podrá habilitar el enlace. Cuando se habilite, se mandará una petición HTTP para activar el actuador.

Curso Normal			
1	Dispositivo o Usuario: solicita habilitar el enlace.		
		2	El sistema habilita el enlace.
		3	El sistema manda la petición HTTP.

Otros datos			
Frecuencia esperada	Muy Alta	Rendimiento	Alta
Importancia	Muy Alta	Urgencia	Alta
Estado		Estabilidad	Alta

Cuadro 15: Caso de uso 15: Habilitar un enlace.

3 Sistema propuesto.

- Caso de uso 16: Deshabilitar un enlace.

Casos de Uso	Deshabilitar un enlace (CU_16).
Actores	Enlace (Principal), Dispositivo (Secundario) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-3.5.
Precondición	El enlace debe de estar registrado en el sistema.
Postcondición	El enlace se deshabilita.

Propósito
El usuario y el sensor, podrán deshabilitar el enlace.

Resumen
El enlace se podrá deshabilitar dependiendo de el valor de entrada recibido por el sensor, en el caso de que no se cumpla la condición. También el usuario podrá deshabilitar el enlace. Cuando se deshabilite, se mandará una petición HTTP para desactivar el actuador.

Curso Normal			
1	Dispositivo o Usuario: solicita deshabilitar el enlace.		
		2	El sistema deshabilita el enlace.
		3	El sistema manda la petición HTTP.

Otros datos			
Frecuencia esperada	Muy Alta	Rendimiento	Alta
Importancia	Muy Alta	Urgencia	Alta
Estado		Estabilidad	Alta

Cuadro 16: Caso de uso 16: Deshabilitar un enlace.

3 Sistema propuesto.

- Caso de uso 17: Consultar enlaces.

Casos de Uso	Consultar enlaces (CU_17).
Actores	Enlace (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-3.6.
Precondición	
Postcondición	El usuario obtendrá una lista de enlaces.

Propósito
Permitir al usuario poder consultar los enlaces guardados.

Resumen
El usuario podrá consultar los enlaces guardados y ver los datos de cada uno de ellos

Curso Normal			
1	Usuario: solicita consultar los enlaces.		
		2	El sistema devuelve una lista de los enlaces guardados.

Otros datos			
Frecuencia esperada	Alta	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 17: Caso de uso 17: Consultar enlaces.

3 Sistema propuesto.

- Caso de uso 18: Consultar los datos de un enlace.

Casos de Uso	Consultar los datos de un enlace (CU_18).
Actores	Enlace (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-3.7.
Precondición	El enlace debe de estar registrado en el sistema.
Postcondición	El usuario obtendrá los datos del enlace.

Propósito
Permitir al usuario poder consultar los datos de un enlace.

Resumen
El usuario seleccionará el enlace que quiere consultar y el sistema se lo proporcionará.

Curso Normal			
1	Usuario: solicita consultar los datos del enlace.		
		2	El sistema comprueba si existe el enlace.
		3	El sistema devuelve los datos del enlace guardado.

Otros datos			
Frecuencia esperada	Alta	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 18: Caso de uso 18: Consultar los datos de un enlace.

3 Sistema propuesto.

- Caso de uso 19: Consultar el historial de un enlace.

Casos de Uso	Consultar el historial de un enlace (CU_19).
Actores	Enlace (Principal) y Usuario (Secundario).
Tipo	Primario y esencial.
Referencias	RF-3.8.
Precondición	El dispositivo debe de estar registrado en el sistema.
Postcondición	El usuario obtendrá el historial de un dispositivo.

Propósito
Permitir al usuario consultar el historial de un dispositivo.

Resumen
Gracias al tecnología Blockchain, se puede obtener todas las transacciones realizadas y con ello poder ver los diferentes cambios que ha tenido un enlace, desde su momento de creación hasta ahora.

Curso Normal			
1	Usuario: solicita consultar el historial del enlace.		
		2	El sistema comprueba si existe el enlace.
		3	El sistema devuelve el historial del enlace guardado.

Otros datos			
Frecuencia esperada	Alta	Rendimiento	Media
Importancia	Media	Urgencia	Baja
Estado		Estabilidad	Alta

Cuadro 19: Caso de uso 19: Consultar el historial de un enlace.

3 Sistema propuesto.

3.2.4. Diagramas de clase.

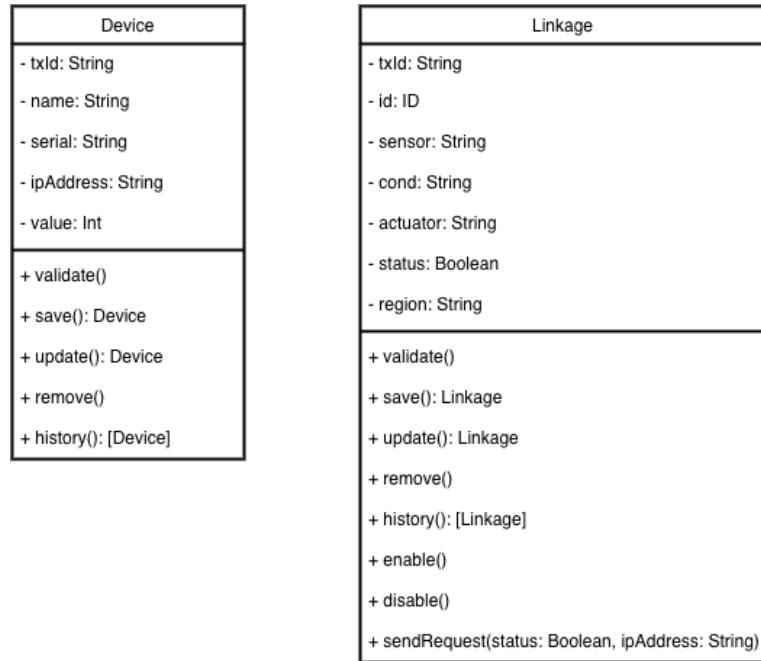


Figura 11: Diagrama de clases.

3.2.5. Diagramas de secuencia.

- Listar dispositivos

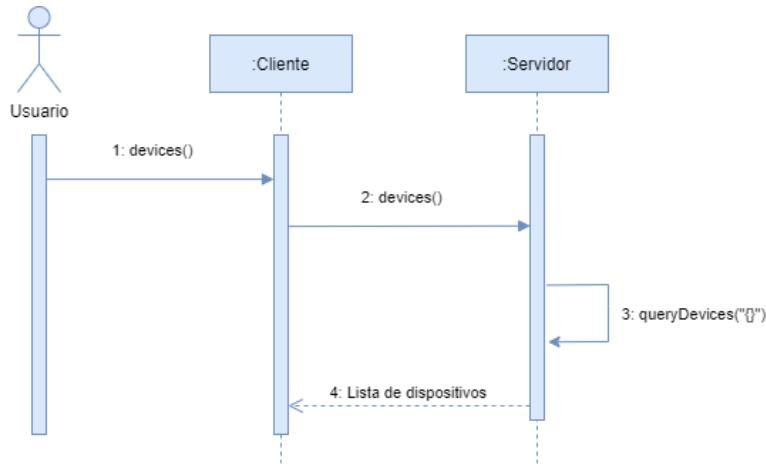


Figura 12: Diagrama de secuencia de Listar dispositivos.

3 Sistema propuesto.

- Consultar dispositivo

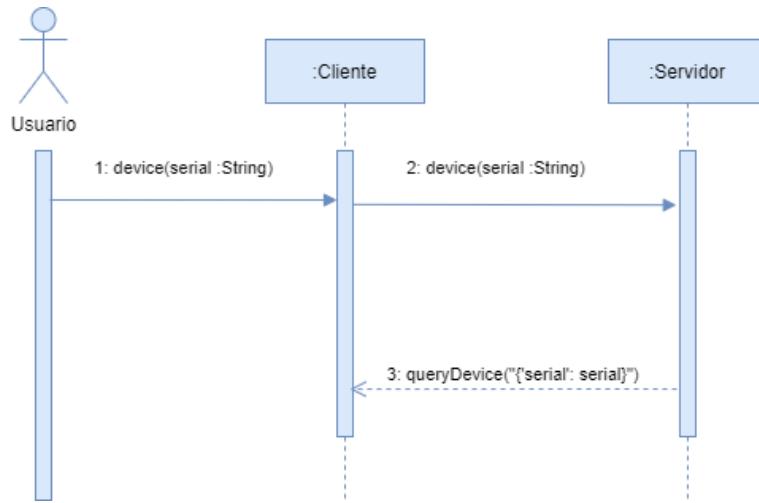


Figura 13: Diagrama de secuencia de Consultar dispositivo.

- Historial dispositivo

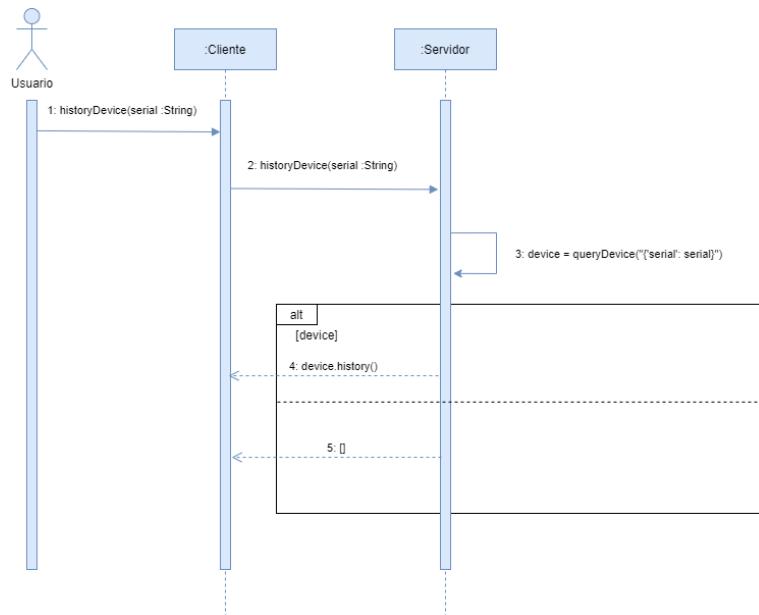


Figura 14: Diagrama de secuencia de Historial dispositivo.

3 Sistema propuesto.

- Añadir dispositivo

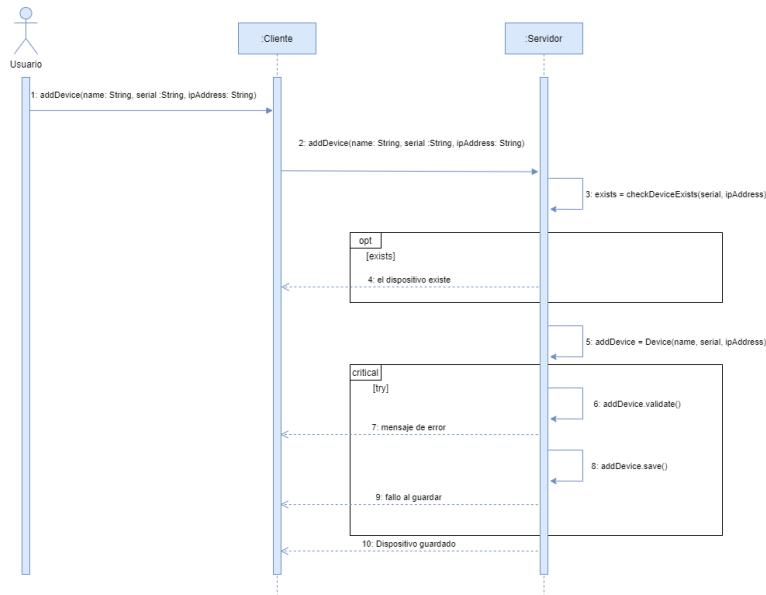


Figura 15: Diagrama de secuencia de Añadir dispositivo.

- Establecer valor a dispositivo

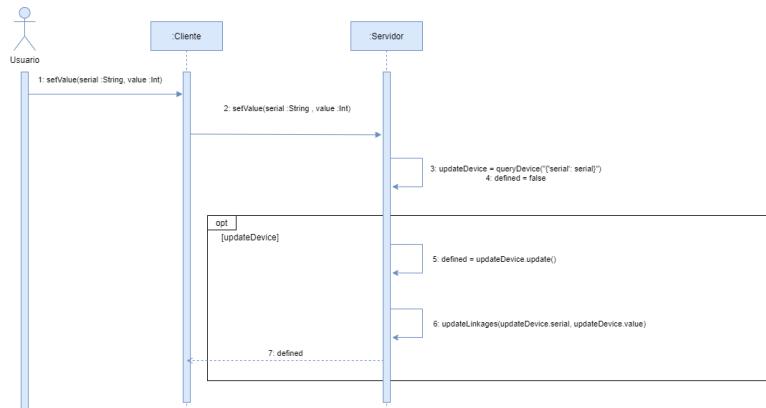


Figura 16: Diagrama de secuencia de Establecer valor a dispositivo.

3 Sistema propuesto.

- Borrar dispositivo

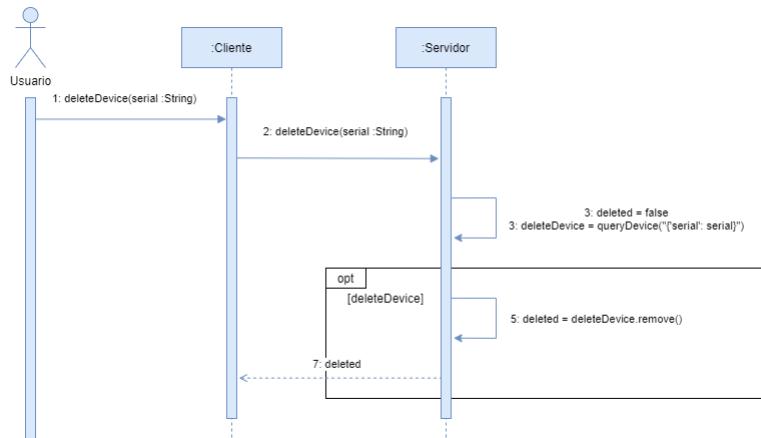


Figura 17: Diagrama de secuencia de Borrar dispositivo.

- Listar enlaces

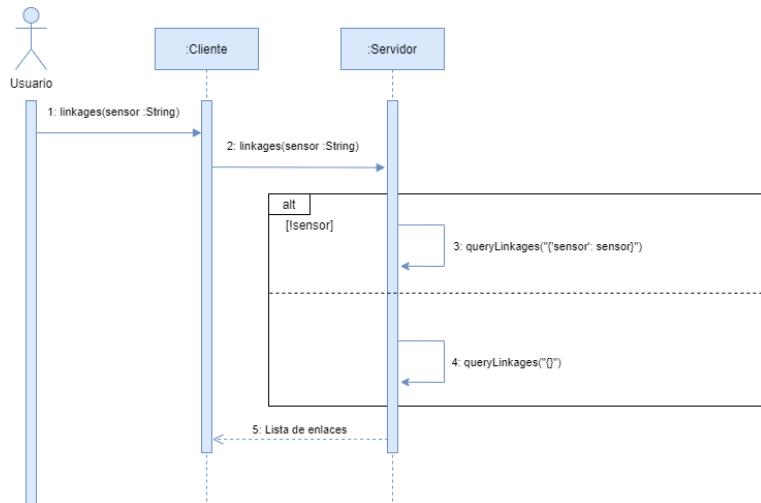


Figura 18: Diagrama de secuencia de Listar enlaces.

3 Sistema propuesto.

- Consultar enlace

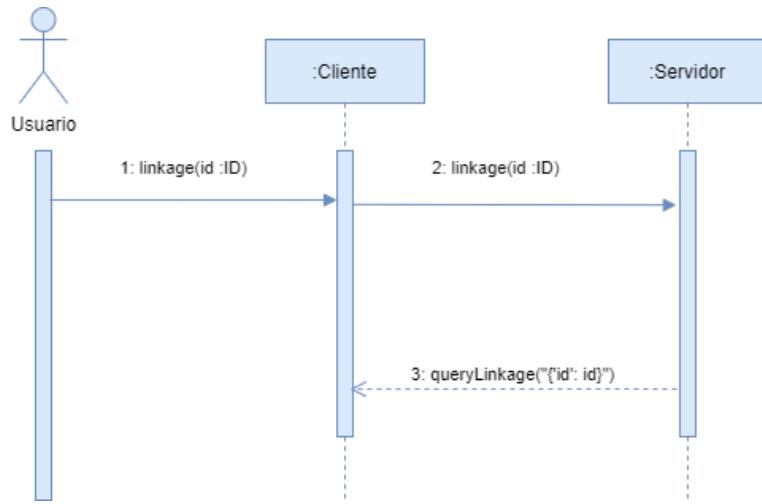


Figura 19: Diagrama de secuencia de Consultar enlace.

- Historial enlace

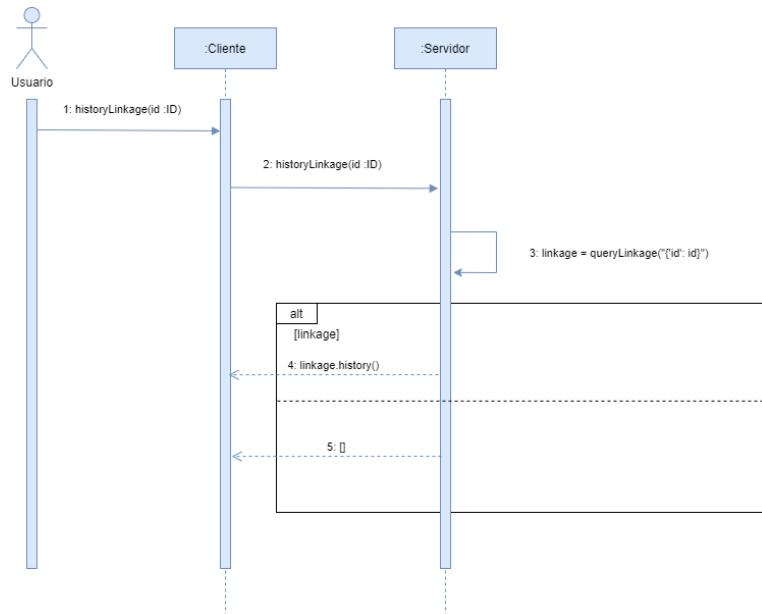


Figura 20: Diagrama de secuencia de Historial enlace.

3 Sistema propuesto.

■ Añadir enlace

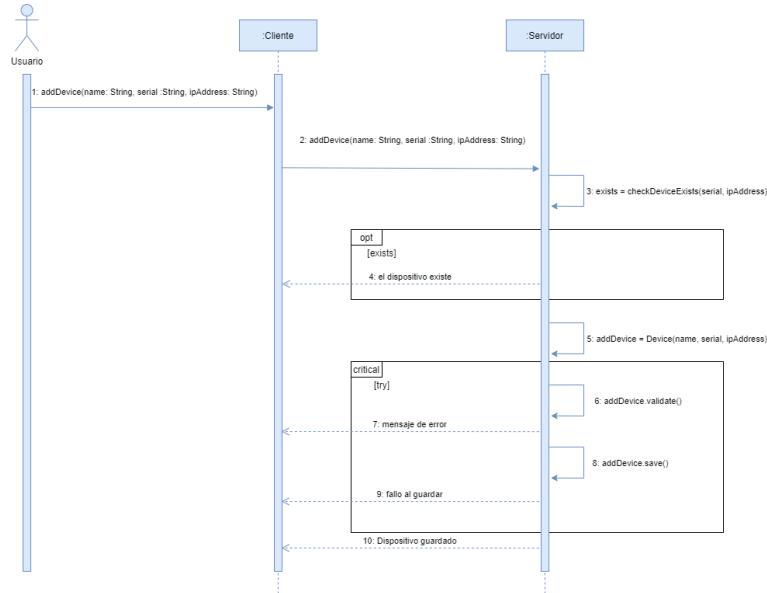


Figura 21: Diagrama de secuencia de Añadir enlace.

■ Actualizar enlace

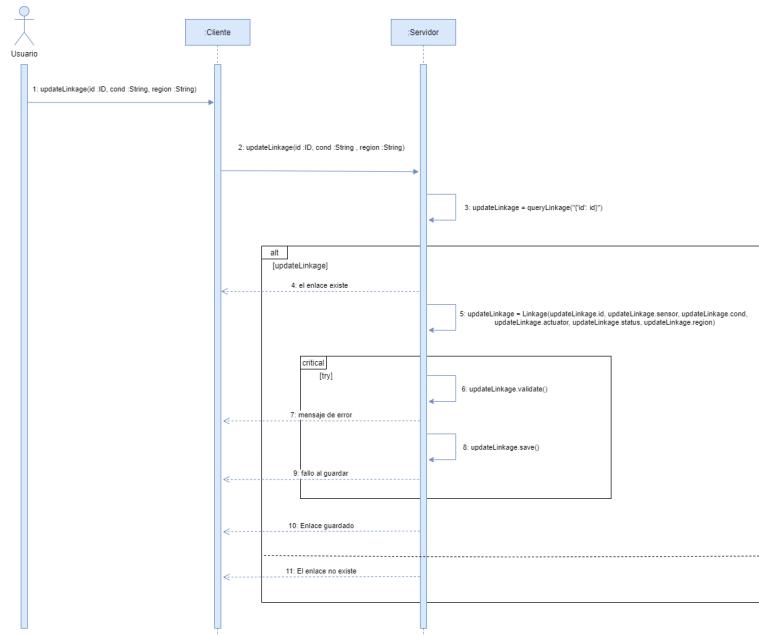


Figura 22: Diagrama de secuencia de Actualizar enlace.

3 Sistema propuesto.

- Borrar enlace

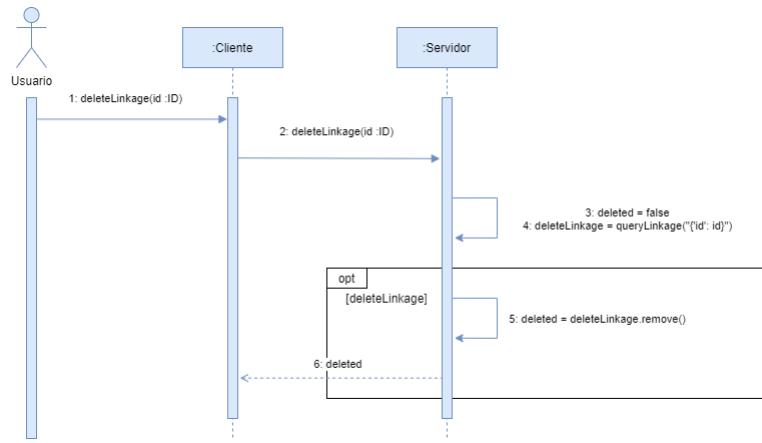


Figura 23: Diagrama de secuencia de Borrar enlace.

- Habilitar enlace

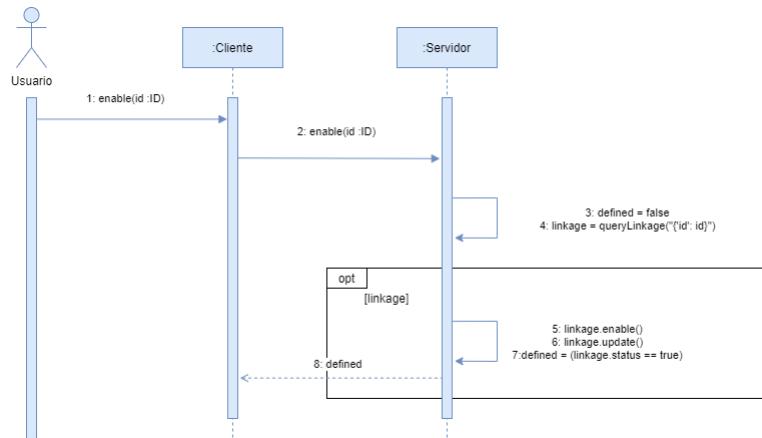


Figura 24: Diagrama de secuencia de Habilitar enlace.

3 Sistema propuesto.

- Deshabilitar enlace

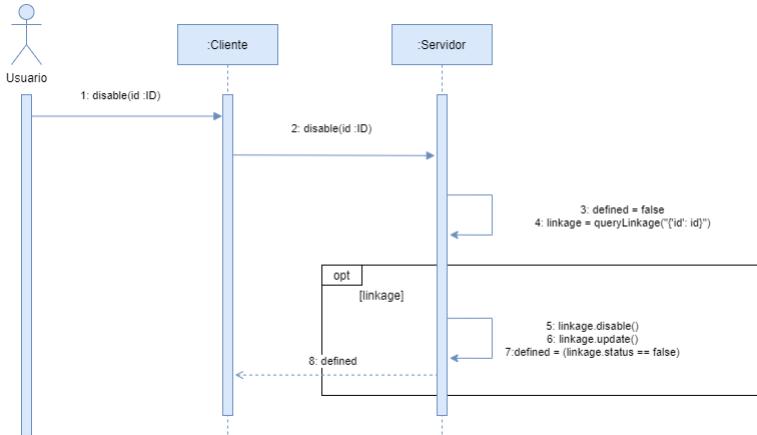


Figura 25: Diagrama de secuencia de Deshabilitar enlace.

3.3. Planificación y presupuesto del proyecto.

Planificación

Después de explicar sobre el análisis de requerimientos, vamos a tratar sobre la planificación llevada a cabo en este proyecto y el presupuesto con el coste de desarrollo del mismo.

La planificación del proyecto se puede dividir en cuatro partes principalmente, que se puede reflejar en el siguiente diagrama de Gantt (ver fig. 26). La planificación consta de un proceso de investigación y lecturas de trabajos anteriores, libros, comparativa de plataformas Blockchain, etc. Por otro lado, se dedicó un plazo de tiempo en el aprovisionamiento de dispositivos hardware para la elaboración de la prueba de concepto y su posterior configuración e instalación del sistema operativo.

Antes de pasar a la implementación de la red, era necesario realizar una labor de investigación para la creación de los binarios e imágenes Docker de Hyperledger Fabric, para las arquitecturas ARM. Después de realizar modificaciones en el propio código de base de Hyperledger Fabric [27], implementé la red Blockchain en las Raspberry PI, que se puede encontrar en la rama “config-network” de mi repositorio [45]. A esta parte se dedicó cerca de más de 3 meses por la complejidad que tiene Hyperledger Fabric, ya que era muy difícil encontrar soluciones a determinados problemas que tenía durante la elaboración, además de la poca información acerca de la implementación de la red en máquinas como las utilizadas. Por lo tanto, tuve que dedicar un

3 Sistema propuesto.

tiempo en implementar la red en mi máquina local y luego transladarlo a los nodos. Todo ello se puede ver en la rama “dev” del repositorio [45].

En la fase de análisis y diseño, se obtuvo los requisitos y los casos de uso del proyecto, para elaborar un boceto de la lógica de la aplicación. También se buscaron herramientas para elaborar las partes del Back end y el Front end, con sus principales ventajas y desventajas.

Para la fase de la implementación de la aplicación se dedicaron cerca de 2 meses y medio, para elaborar la API, creando los Resolvers, Schemas y Models para los enlaces y dispositivos, y su posterior conexión a la red Blockchain. Se creó un script para levantar el servicio en el nodo maestro y para poder hacerlo accesible desde fuera, se obtuvo un dominio mediante DDNS en la plataforma NO-IP [52], y se abrieron los puertos 80 y 443 del router personal.

Para la implementación de la interfaz de usuario, se llevó a cabo la implementación de PWA, que usa HTTPS, tiene que incluir un manifiesto de Aplicación Web y debe de implementar un trabajador de servicio. Se desarrollaron las funcionalidades de enlaces y dispositivos, se conectó a la API y se realizaron test unitarios. Por último se desplegó en el servicio de Gatsby Cloud, que permite hacer análisis de SEO con la herramienta LightHouse.

Se puede obtener una planificación más detallada, extendiendo cada una de las partes elaboradas en la siguiente página (ver fig. 26).

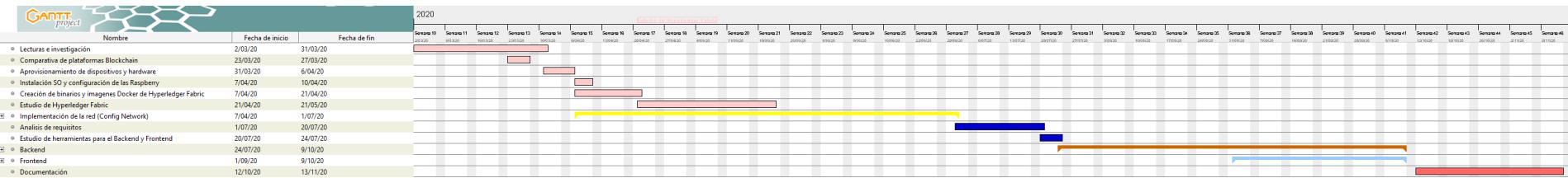


Figura 26: Planificación general.

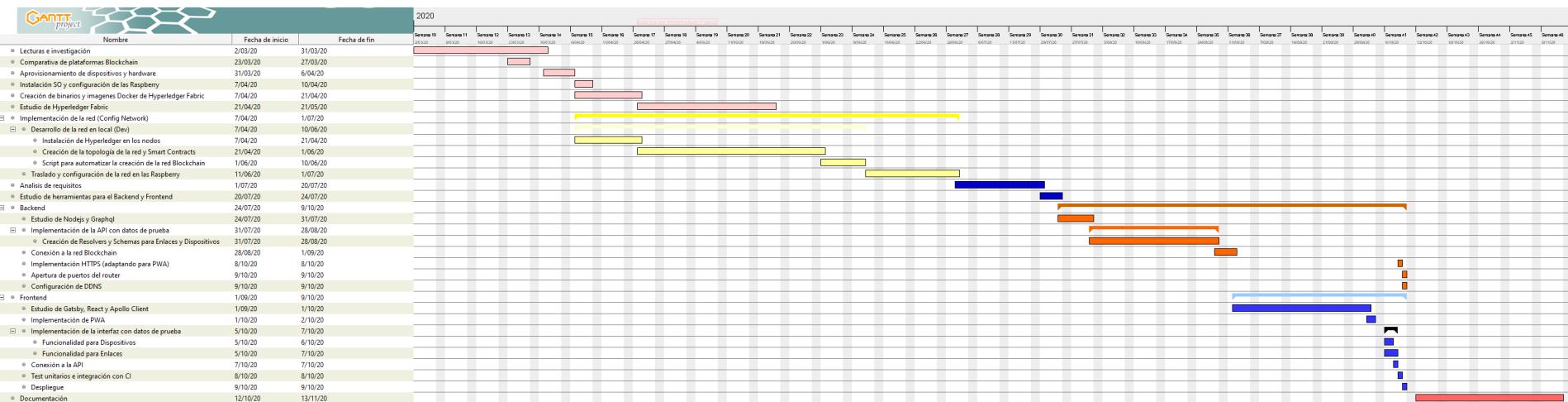


Figura 27: Planificación expandida.

Presupuesto

Seguidamente, pasaremos hablar sobre el presupuesto para la realización del proyecto, teniendo en cuenta que se van a invertir 9 meses en él, desde el diseño de la plataforma hasta el despliegue y mantenimiento del mismo.

El calculo del presupuesto se va a valorar en función de los siguientes recursos:

- Recurso humano.

Se va a necesitar:

- Jefe de proyecto, para que coordine las tareas del proyecto.
- Analista de sistemas, para que diseñe la plataforma, teniendo en cuenta los requerimientos del usuario.
- Full Stack Developer, para elaborar la implementación tanto del Back end y el Front end y su despliegue.
- Desarrollador Blockchain, encargado de la creación, gestión y mantenimiento de la red Blockchain.

Coste:

Perfil	Coste
Jefe de proyecto	40.000 / 12 meses x 9 meses = 30.000€
Analista de sistemas	36.000 / 12 meses x 3 meses = 9.000€
Full Stack Developer	30.000 / 12 meses x 6 meses = 15.000€
Desarrollador Blockchain	40.000 / 12 meses x 4 meses = 13.333€
Coste total:	30.000 + 9.000 + 15.000 + 13.333 = 67.333€

Cuadro 20: Coste de recursos humanos. Fuente: [41].

- Recurso material.

Se ha utilizado:

- Portatil: MacBook Pro 2018 con procesador 2,7 GHz Intel Core i7 de 4 núcleos de 16 GB de RAM y con capacidad de 256 GB. 1500€
- 5 Raspberry Pi: 5 Raspberries Pi (50€ cada una):
 - 3 Raspberries Pi 4 Modelo B+ 2GB, que servirán como nodos principales para la red.
 - Actuador (Raspberry Pi 2 Modelo B+) y sensor (Raspberry Pi 3 Modelo B+) para simular la red domótica.
- Un switch TP-Link TL-SG105 - Switch 5 Puertos 10/100/1000 MBps 17,99€

3 Sistema propuesto.

- Cables ethernet: 14 €
- Xiaomi Router 4A WiFi inalámbrico 2.4GHz 5GHz Banda Dual 1167Mbps WiFi Repeater 4 Antenas 34,99 €
- Otros gastos:
 - Internet y Luz: 132 € x 9 meses = 1.188€
 - Servicios de la nube: Utilizo la suscripción gratuita de Gatsby Cloud.
 - Software: 0€

Coste total material: 3.004,98€

Recurso	Coste
Humano	67.333€
Material	3.004,98€
Total	70.337,98€

Cuadro 21: Coste total de los recursos

4. Desarrollo del sistema.

En esta sección vamos a dar paso al desarrollo del proyecto, en primer lugar hablaremos sobre la implementación de la red blockchain en las Raspberry previamente configuradas, para pasar al desarrollo de la API basada en GraphQL con NodeJS y posteriormente a la interfaz de usuario con Gatsby y React.

4.1. Configuración y creación de la topología de la red.

Para el montaje de la red Blockchain se ha utilizado 3 Raspberry Pi 4 Modelo B+ 2GB, que servirán como nodos principales.

El sistema operativo utilizado es Ubuntu Server 18.04 [42] la versión de 64-bit, que se puede encontrar en la página oficial de Raspberry [57]. Para la instalación seguimos los pasos aconsejados en la misma página de instalación de Ubuntu. Necesitaremos:

- Tarjeta microSD.
- Imagen de Ubuntu Server.

Una vez descargada la imagen de Ubuntu Server vamos a crear un punto de arranque en la tarjeta microSD.

Desde una terminal ejecutamos los siguientes comandos:

```
1 $ diskutil unmountDisk <sdcard>
2 $ sudo sh -c 'gunzip -c <imagen> | sudo dd of=<sdcard> bs=32m'
```

Una vez instalado el sistema operativo, vamos a establecer la dirección estática a cada una de las Raspberry para evitar la asignación por DHCP, para ello modificamos el fichero “/etc/netplan/50-cloud-init.yaml” con la siguiente configuración [61]:

```
1 network:
2   renderer: networkd
3   ethernets:
4     eth0:
5       addresses: [<direccion>/<mascara>]
6       gateway4: <default gateway>
7       nameservers:
8         addresses: [<direccion_dns>]
9   version: 2
```

Y aplicamos los cambios con el siguiente comando:

```
1 $ sudo netplan apply
```

4 Desarrollo del sistema.

En la siguiente imagen se muestra la topología final de la red (ver fig.28):

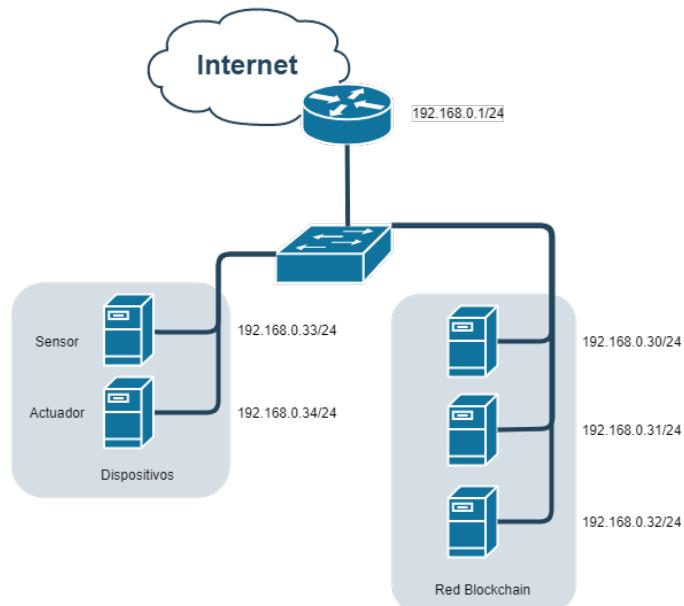


Figura 28: Topología de la red.



Figura 29: Captura de las Raspberry Pi.

Instalación de Hyperledger Fabric 1.4.4.

Para la instalación de Hyperledger Fabric (1.4.4) necesitaremos que cada nodo tengan los siguientes requisitos instalados [8]:

- cURL.
- Docker version 17.06.2-ce o mayor [48, 30].
- git.
- Docker Compose 1.14.0 o mayor [48, 30].
- Go version 1.12.x
- Node.js
- npm version 5.6.0
- Python 2.7

Una vez instalado los requisitos procedemos a la creación de las imágenes de Hyperledger Fabric a partir de los binarios. Actualmente no hay soporte para arquitecturas ARM ya que Hyperledger solamente soporta las siguientes arquitecturas [62]:

- amd64
- s390x
- ppc64le

Por tanto el principal objetivo y uno de los obstáculos importantes, era conseguir que Hyperledger Fabric se ejecutara en Raspberry Pi. Por ello, realice la construcción de mis propias imágenes de Docker que se pueden encontrar en mi DockerHub [20]. Para ello, he tenido que realizar una serie de cambios en el código fuente de Hyperledger Fabric que se encuentra distribuido en dos repositorios: **fabric-baseimage**¹⁰ y **fabric** [25, 24].

fabric-baseimage

Para poder llevar acabo la construcción de las imágenes Docker y los binarios, he forkeado el repositorio de Hyperledger/fabric-baseimage [25] desde la etiqueta v0.4.18 y he realizado los siguientes cambios para construir las imágenes con la última versión de arm64v8. Todos estos cambios se pueden encontrar en la rama “project” del repositorio [27].

¹⁰Actualmente, este proyecto se encuentra deprecated.

4 Desarrollo del sistema.

Clonamos el repositorio fork desde la rama project:

```
1 $ git clone -b project \
2   https://github.com/Thejokeri/fabric-baseimage.git
```

Con la siguiente orden del Makefile:

```
1 $ make docker couchdb kafka zookeeper
```

generamos las imágenes necesarias:

- fabric-baseos
- fabric-baseimage
- fabric-couchdb
- fabric-kafka
- fabric-zookeeper

fabric

Desde el repositorio oficial, clonamos desde la rama v1.4.4:

```
1 $ git clone -b v1.4.4 https://github.com/hyperledger/fabric.git
```

Y lanzamos la siguiente orden Makefile, para generar las imágenes restantes de Hyperledger:

```
1 $ make native license spelling linter docker
```

- fabric-tools
- fabric-buildenv
- fabric-ccenv
- orderer
- peer

4 Desarrollo del sistema.

El resultado final es el siguiente:



```
ubuntu@ubuntu:~/IoTBlockchainAPP/infrastructure/config-network/bin$ ls
chaintool configtxgen configtxlator cryptogen discover idemixgen orderer peer
```

Figura 30: Binarios de Hyperledger Fabric.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hyperledger/fabric-tools	arm64-1.4.4-snapshot-7917a40ff	c9704ea000a9	5 days ago	1.65GB
hyperledger/fabric-tools	arm64-latest	c9704ea000a9	5 days ago	1.65GB
hyperledger/fabric-tools	latest	c9704ea000a9	5 days ago	1.65GB
hyperledger/fabric-ccenv	arm64-1.4.4-snapshot-7917a40ff	2cb90301ea98	5 days ago	1.51GB
hyperledger/fabric-ccenv	arm64-latest	2cb90301ea98	5 days ago	1.51GB
hyperledger/fabric-ccenv	latest	2cb90301ea98	5 days ago	1.51GB
hyperledger/fabric-buildenv	arm64-1.4.4-snapshot-7917a40ff	92dd3b26051b	5 days ago	1.56GB
hyperledger/fabric-buildenv	arm64-latest	92dd3b26051b	5 days ago	1.56GB
hyperledger/fabric-buildenv	latest	92dd3b26051b	5 days ago	1.56GB
hyperledger/fabric-orderer	arm64-1.4.4-snapshot-7917a40ff	b221ec6cab07	5 days ago	114MB
hyperledger/fabric-orderer	arm64-latest	b221ec6cab07	5 days ago	114MB
hyperledger/fabric-orderer	latest	b221ec6cab07	5 days ago	114MB
hyperledger/fabric-peer	arm64-1.4.4-snapshot-7917a40ff	8a828c6b69c7	5 days ago	121MB
hyperledger/fabric-peer	arm64-latest	8a828c6b69c7	5 days ago	121MB
hyperledger/fabric-peer	latest	8a828c6b69c7	5 days ago	121MB
hyperledger/fabric-zookeeper	arm64-0.4.18	77c8822313d8	5 days ago	362MB
hyperledger/fabric-zookeeper	latest	77c8822313d8	5 days ago	362MB
hyperledger/fabric-kafka	arm64-0.4.18	d153e494662e	5 days ago	355MB
hyperledger/fabric-kafka	latest	d153e494662e	5 days ago	355MB
hyperledger/fabric-couchdb	arm64-0.4.18	ac0765e291e2	5 days ago	357MB
hyperledger/fabric-couchdb	latest	ac0765e291e2	5 days ago	357MB
hyperledger/fabric-baseimage	arm64-0.4.18	b669e04bdf51	5 days ago	1.46GB
hyperledger/fabric-baseimage	latest	b669e04bdf51	5 days ago	1.46GB
hyperledger/fabric-baseos	arm64-0.4.18	45157e979a2f	5 days ago	73.4MB
hyperledger/fabric-baseos	latest	45157e979a2f	5 days ago	73.4MB
arm64v8/ubuntu	bionic	428b2f74b0fb	3 weeks ago	57.7MB
arm64v8/ubuntu	xenial	11f8c8b83194	7 weeks ago	112MB
golang	1.13.8-alpine	b98704846f0c	8 weeks ago	353MB
adoptopenjdk/openjdk8	aarch64-ubuntu-jdk8u222-b10	4263f1002511	6 months ago	301MB

Figura 31: Imágenes Docker de Hyperledger Fabric.

Como se mencionó anteriormente, vamos a utilizar varias Raspberry como nodos principales de la red Blockchain, voy a utilizar Docker Swarm para la orquestación de multiples contenedores dentro de múltiples máquinas anfitrionas. Para llevarlo a cabo, nos conectamos al nodo que va a ser el maestro, en este caso es la máquina con IP **192.168.0.30**, e introducimos el siguiente comando:

```
1 $ docker swarm init
```

Este comando genera dos token al azar, una de trabajador y otra de gerente. Cuando unes un nuevo nodo al swarm, el nodo se une como un nodo trabajador o gerente basado en el token que usas para unirte al swarm.

Unimos los nodos **192.168.0.31** y **192.168.0.32** al swarm con el comando que se genera:

```
1 $ docker swarm join --token SWMTKN-1-XXXX 192.168.0.30:2377
```

4 Desarrollo del sistema.

En la siguiente figura podemos ver los nodos unidos y activos (ver fig. 32):

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
iqttnivtrwafiu1578m08w7if	ubuntu	Ready	Active	Leader	19.03.13
ooithsvpn7bc3pf44i40jt95g *	ubuntu	Ready	Active	Active	19.03.13
sn6fliieg4l1ai9oxm73s28b8g	ubuntu	Ready	Active	Active	19.03.13

Figura 32: Nodos de Docker Swarm.

Con Docker Swarm conseguimos que los contenedores que vayamos a crear para la red Blockchain, se lance de forma arbitraria en cada uno de los nodos y no nos tenemos que preocupar en ir levantando cada contenedor uno por uno en las máquinas. Solamente basta con lanzarlo en el nodo maestro y él se encargará de asignarle los contenedores a los nodos trabajadores activos, todo en una red privada [51].

Para facilitar el proceso de instalación de prerequisitos, creación de binarios, imágenes, Docker Swarm, etc. Se ha creado un script **setup_base.sh** que tiene las siguientes opciones:

```
1 ./setup_base.sh -h
2 Uso: setup_base.sh [opciones]
3
4 opciones:
5 -h : muestra esta ayuda
6 -p : instalar prerequisitos
7 -f : instalar binarios e imagenes de Hyperlegder Fabric
8 -s : iniciar Docker Swarm cluster
9 -d : pull imagenes Docker
10 -w : muestra las versiones de los prerequisitos
11
12 e.g. setup_base.sh -f
13 creara los binarios e imagenes de Hyperledger Fabric
```

4.2. Implementación de la red Blockchain.

Después de configurar las Raspberry Pi e instalar los requisitos, vamos a implementar la red Blockchain. Dentro del repositorio, la estructura de ficheros que rige la infraestructura de la red es la siguiente:



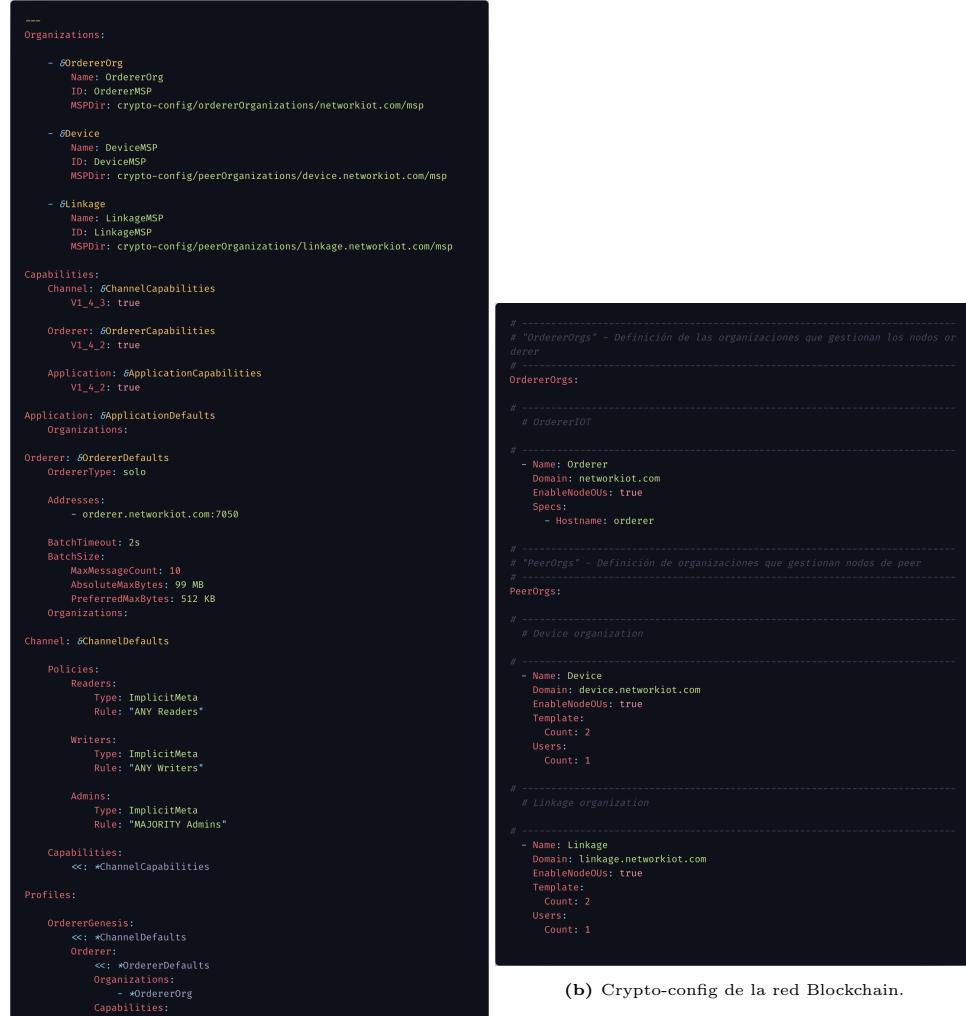
```
thejoker in IoTBlockchainAPP on ↵ documentacion [!?]
→ tree infrastructure
infrastructure
└── base
    ├── README.md
    └── setup_base.sh
config-network
└── bin
    ├── chaintool
    ├── configtxgen
    ├── configtxlator
    ├── cryptogen
    ├── discover
    ├── idemixgen
    ├── orderer
    └── peer
    ├── chaincode
    ├── device
    │   └── device.go
    └── linkage
        └── linkage.go
    └── channel-artifacts
        └── genesis.block
network
└── channel-artifacts
    ├── configtx.yaml
    ├── crypto-config.yaml
    ├── docker-compose.yaml
    └── networkiot.sh

10 directories, 16 files
thejoker in IoTBlockchainAPP on ↵ documentacion [!?]
→
```

Figura 33: Tree de la carpeta infraestructure.

La carpeta **config-network** contiene los binarios necesarios para las imágenes Docker de Hyperledger Fabric, los chaincode creados para realizar las operaciones a los dispositivos y enlaces [16], y todos los ficheros necesarios para la configuración de la red Blockchain, que son:

4 Desarrollo del sistema.



```

---  

Organizations:  

- &OrdererOrg  

  Name: OrdererOrg  

  ID: OrdererMSP  

  MSPDir: crypto-config/ordererOrganizations/networkiot.com/msp  

- &Device  

  Name: DeviceMSP  

  ID: DeviceMSP  

  MSPDir: crypto-config/peerOrganizations/device.networkiot.com/msp  

- &Linkage  

  Name: LinkageMSP  

  ID: LinkageMSP  

  MSPDir: crypto-config/peerOrganizations/linkage.networkiot.com/msp  

Capabilities:  

  Channel: &ChannelCapabilities  

    V1_4_3: true  

  Orderer: &OrdererCapabilities  

    V1_4_2: true  

  Application: &ApplicationCapabilities  

    V1_4_2: true  

Application: &ApplicationDefaults  

  Organizations:  

  Orderer: &OrdererDefaults  

    OrdererType: solo  

    Addresses:  

      - orderer.networkiot.com:7050  

    BatchTimeout: 2s  

    BatchSize:  

      MaxMessageCount: 10  

      AbsoluteMaxBytes: 99 MB  

      PreferredMaxBytes: 512 KB  

  Organizations:  

  Channel: &ChannelDefaults  

  Policies:  

    Readers:  

      Type: ImplicitMeta  

      Rule: "ANY Readers"  

    Writers:  

      Type: ImplicitMeta  

      Rule: "ANY Writers"  

    Admins:  

      Type: ImplicitMeta  

      Rule: "MAJORITY Admins"  

  Capabilities:  

    <<: *ChannelCapabilities  

  Profiles:  

  OrdererGenesis:  

    <<: *ChannelDefaults  

    Orderer:  

      <<: *OrdererDefaults  

        Organizations:  

          - &OrdererOrg  

        Capabilities:  

          <<: *OrdererCapabilities  

  Consortiums:  

    SampleConsortium:  

      Organizations:  

        - &Device  

        - &Linkage  

  ChannelAll:  

    Consortium: SampleConsortium  

    <<: *ChannelDefaults  

    Application:  

      <<: *ApplicationDefaults  

      Organizations:  

        - &Device  

        - &Linkage  

      Capabilities:  

        <<: *ApplicationCapabilities

```

(a) Configtx de la red Blockchain.

(b) Crypto-config de la red Blockchain.

Figura 34: ficheros de configuración de la red Blockchain.

El fichero crypto-config se divide en dos secciones, las organizaciones orderer y las organizaciones peer. Para esta arquitectura hemos escogido una organización orderer con host **orderer.networkiot.com** y dos organizaciones peer con host **device.networkiot.com** y **linkage.networkiot.com**.

4 Desarrollo del sistema.

Por otra parte, el fichero configtx se estructura en varias secciones, tenemos organizaciones, orderers, aplicaciones, capacidades y perfiles. Las secciones que podemos destacar son las organizaciones que van a formar la red Blockchain, en este caso tenemos una organización orderer y dos organizaciones peer: **Device** y **Linkage**. Por otro lado, tenemos los perfiles donde indicamos cuáles organizaciones pertenecen a la organización orderer y canales, en este caso en el archivo indicamos que el único canal que vamos a crear, llamado **ChannelAll**, van a poder comunicarse los peers pertenecientes a la organización Device o Linkage.

Cabe destacar el documento **docker-compose.yaml**, en el se indican los contenedores que se va a lanzar para formar la red blockchain, que variables de entorno van a tener cada uno, en que nodos se van a lanzar y la red privada en la que se van a encontrar. Dentro del fichero vamos a encontrar un nodo orderer y un CLI para realizar consultas directamente a la red, luego dos parejas de nodos peer por cada organización que hay (Device y Linkage), cuatro bases de datos CouchDB para cada nodo peer y dos entidades certificadoras. Para ilustrar mejor la arquitectura de la red Blockchain (ver fig. 35). Los nodos orderer y CLI se van a lanzar en la máquina maestro y el resto se van a administrar en las otras dos máquinas trabajadoras.

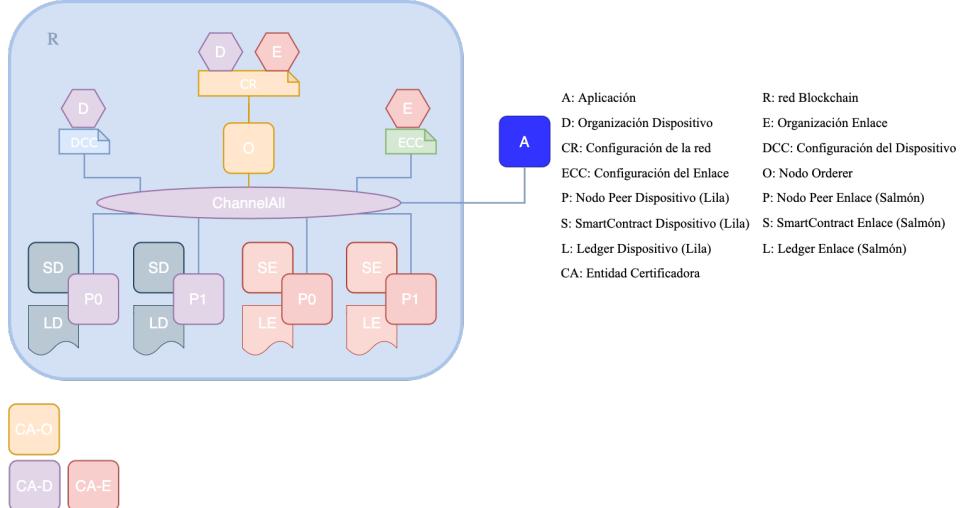


Figura 35: Arquitectura de la red Blockchain.

Se ha elaborado un script, **networkiot.sh**, para automatizar el proceso de generación de la red, la orquestación de los contenedores, la creación y copia de los certificados y ficheros necesarios a cada una de las máquinas y la instalación y el proceso de instanciar de los chaincode. Incluso se ha añadido el comando para la eliminación de la red. [26, 19]

4 Desarrollo del sistema.

```

1 ./networkiot.sh help
2 Uso: networkiot.sh [help, up, down, generate, restart, remove]
3
4 Comandos:
5     help      Mostrar esta ayuda
6     up       Activar red
7     generate  Generar red
8     remove   Eliminar red
9     all      Generar y Activar red

```

```

ssh
ubuntu@ubuntu:~/IoTBlockchainAPP/infrastructure/config-network/network$ ./networkiot.sh generate
Generando certificados con cryptogen...
device.networkiot.com
linkage.networkiot.com

Generando bloque genesis...
2020-11-11 11:49:08.662 UTC [common.tools.configtxgen] main → INFO 001 Loading configuration
2020-11-11 11:49:08.687 UTC [common.tools.configtxgen.localconfig] completeInitialization → INFO 002 orderer type: solo
2020-11-11 11:49:08.688 UTC [common.tools.configtxgen.localconfig] Load → INFO 003 Loaded configuration: /home/ubuntu/IoTBlockchainAPP/infrastructure/config-network/network/configtx.yaml
2020-11-11 11:49:08.709 UTC [common.tools.configtxgen.localconfig] completeInitialization → INFO 004 orderer type: solo
2020-11-11 11:49:08.710 UTC [common.tools.configtxgen.localconfig] LoadTopLevel → INFO 005 Loaded configuration: /home/ubuntu/IoTBlockchainAPP/infrastructure/config-network/network/configtx.yaml
2020-11-11 11:49:08.710 UTC [common.tools.configtxgen.encoder] NewOrdererGroup → WARN 006 Default policy emission is deprecated, please include policy specifications for the orderer group in configtx.yaml
2020-11-11 11:49:08.757 UTC [common.tools.configtxgen.encoder] NewOrdererOrgGroup → WARN 007 Default policy emission is deprecated, please include policy specifications for the orderer org group OrdererOrg in configtx.yaml
2020-11-11 11:49:08.761 UTC [common.tools.configtxgen.encoder] NewConsortiumOrgGroup → WARN 008 Default policy emission is deprecated, please include policy specifications for the orderer org group DeviceMSP in configtx.yaml
2020-11-11 11:49:08.763 UTC [common.tools.configtxgen.encoder] NewConsortiumOrgGroup → WARN 009 Default policy emission is deprecated, please include policy specifications for the orderer org group LinkageMSP in configtx.yaml
2020-11-11 11:49:08.763 UTC [common.tools.configtxgen] doOutputBlock → INFO 00a Generating genesis block
2020-11-11 11:49:08.765 UTC [common.tools.configtxgen] doOutputBlock → INFO 00b Writing genesis block

Generando channelall ...
2020-11-11 11:49:08.852 UTC [common.tools.configtxgen] main → INFO 001 Loading configuration
2020-11-11 11:49:08.867 UTC [common.tools.configtxgen.localconfig] Load → INFO 002 Loaded configuration: /home/ubuntu/IoTBlockchainAPP/infrastructure/config-network/network/configtx.yaml
2020-11-11 11:49:08.881 UTC [common.tools.configtxgen.localconfig] completeInitialization → INFO 003 orderer type: solo
2020-11-11 11:49:08.881 UTC [common.tools.configtxgen.localconfig] LoadTopLevel → INFO 004 Loaded configuration: /home/ubuntu/IoTBlockchainAPP/infrastructure/config-network/network/configtx.yaml
2020-11-11 11:49:08.881 UTC [common.tools.configtxgen] doOutputChannelCreateTx → INFO 005 Generating new channel config tx
2020-11-11 11:49:08.882 UTC [common.tools.configtxgen.encoder] NewApplicationGroup → WARN 006 Default policy emission is deprecated, please include policy specifications for the application group in configtx.yaml
2020-11-11 11:49:08.883 UTC [common.tools.configtxgen.encoder] NewApplicationOrgGroup → WARN 007 Default policy emission is deprecated, please include policy specifications for the application org group DeviceMSP in configtx.yaml
2020-11-11 11:49:08.885 UTC [common.tools.configtxgen.encoder] NewApplicationOrgGroup → WARN 008 Default policy emission is deprecated, please include policy specifications for the application org group LinkageMSP in configtx.yaml
2020-11-11 11:49:08.885 UTC [common.tools.configtxgen.encoder] NewApplicationGroup → WARN 009 Default policy emission is deprecated, please include policy specifications for the application group in configtx.yaml
2020-11-11 11:49:08.886 UTC [common.tools.configtxgen.encoder] NewApplicationOrgGroup → WARN 00a Default policy emission is deprecated, please include policy specifications for the application org group DeviceMSP in configtx.yaml
2020-11-11 11:49:08.888 UTC [common.tools.configtxgen.encoder] NewApplicationOrgGroup → WARN 00b Default policy emission is deprecated, please include policy specifications for the application org group LinkageMSP in configtx.yaml
2020-11-11 11:49:08.888 UTC [common.tools.configtxgen] doOutputChannelCreateTx → INFO 00c Writing new channel tx

Copiando archivos de configuracion a los nodos de trabajo...
Done
ubuntu@ubuntu:~/IoTBlockchainAPP/infrastructure/config-network/network$ 

```

Figura 36: Opción generate del script networkiot.

Al lanzar el comando **generate** en el nodo maestro, se creará la carpeta **channel-artifacts** que guardarán los archivos generados por el binario **cryptogen**, que es usado para la creación de los archivos de la red, pasandole como parámetro el archivo de configuración **crypto-config**. Los archivos que se generan son:

- **channel.tx:** archivo para la transacción de configuración del canal.

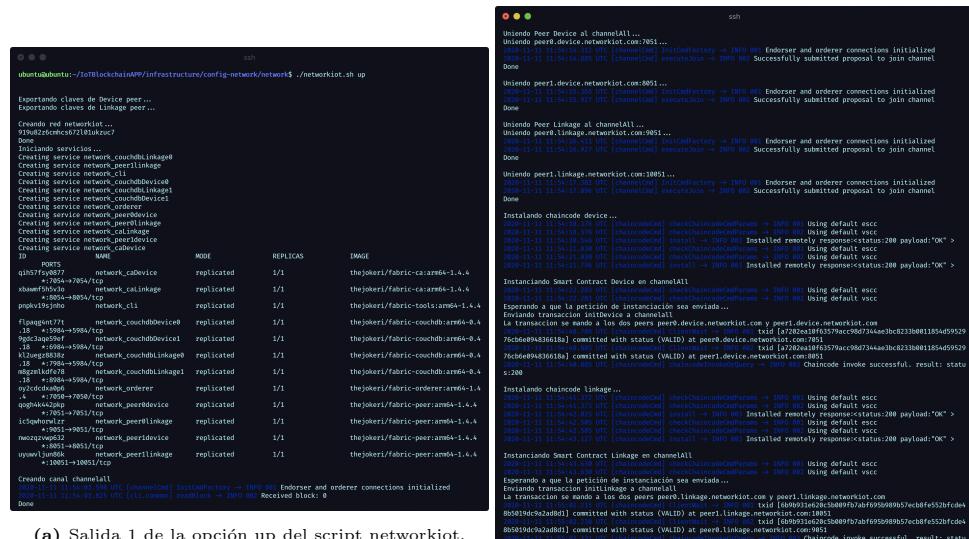
4 Desarrollo del sistema.

- genesis.block: es el primer bloque de la cadena, el bloque génesis, el cual inicia la cadena de bloques de nuestra red.

También se creará la carpeta crypto-config, que guardará las credenciales de nuestra red Hyperledger Fabric, para que se puedan conectar los diferentes usuarios y aplicaciones, tanto del ordenante como de todos los participantes. Las credenciales tienen que estar disponibles en todos los nodos y han de ser las mismas. Por tanto, se realiza una copia tanto de la carpeta channel-artifacts como crypto-config, a las otras máquinas que forman la red.

Ahora toca lanzar el comando **up**, que se encargará de:

- Crear los certificados de los CA y compartirlos con las otras máquinas.
- Crear la red privada **networkiot** para los contenedores.
- Levantar los contenedores y los servicios Docker.
- Crear el canal **ChannelAll** e unir los nodos peer al canal.
- Instalar e instanciar los chaincode en el canal.



The image shows two terminal sessions on a Mac OS X system. Both sessions are running the command `/opt/BlockchainAPP/infrastructure/config-network/network$./networkiot.sh up`.
 Terminal 1 (Left): Shows the initial steps of the 'up' command, including creating the network, starting services, and joining the 'ChannelAll' channel. It includes logs for peer devices, network links, and channel joins.
 Terminal 2 (Right): Shows the continuation of the 'up' command, including installing chaincode, instantiating smart contracts, and sending transactions between peers. Logs show responses like 'OK', transaction IDs, and chaincode invoke successful results.

Figura 37: Salidas del comando up.

Este proceso es el más importante para la creación de la red Blockchain. A continuación vamos a mostrar los servicios y los contenedores que se han creado. Vamos a poder comprobar como Docker Swarm se ha encargado de gestionar la administración de los contenedores en cada uno de las máquinas. El comando **remove** se encargará de eliminar toda la configuración de la red Blockchain y eliminará los contenedores y servicios creados.

4 Desarrollo del sistema.

```
ubuntu@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
539add235d9b      thejokeri/fabric-orderer:arm64-1.4.4   "orderer"          5 minutes ago     Up 5 minutes       7050/tcp           network_orderer.1.xmyfi4fp14ck6g9112kj7uxt
4437efbf6b9c3      thejokeri/fabric-tools:arm64-1.4.4    "/bin/bash"        5 minutes ago     Up 5 minutes       network_cli.1.eypwbl3eu0ts4x78mpv8h9z1
ubuntu@ubuntu:~$
```

Figura 38: Salida de contenedores en el nodo maestro.

```
ubuntu@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
539add235d9b      dev-peer0.linkage.networkiot.com-linkage-1.0-5177954a8ca079cd9e5e9bb89df6f19abe11284ffdb23ed0ff0b9dc54   "chaincode -peer.add..." About a minute ago Up About a minute
aecd8dbfc0a4      thejokeri/fabric-peer:arm64-1.4.4   "dev-peer1.linkage..._linkage-1.0"   2 minutes ago     Up 2 minutes
7054/tcp          network_calinkage.1.sjcs9fx7o2e8xt0j05shu62   "sh -c 'fabric-ca-se..." 2 minutes ago     Up 2 minutes
0b398e91b9      thejokeri/fabric-peer:arm64-1.4.4   "peer node start" 2 minutes ago     Up 2 minutes
a8c405d6f8e2      thejokeri/fabric-couchdb:arm64-0.4.18   "tini -- /docker-ent..." 2 minutes ago     Up 2 minutes
af0dce77af6      thejokeri/fabric-couchdb:arm64-0.4.18   "tini -- /docker-ent..." 2 minutes ago     Up 2 minutes
4369/tcp          5984/tcp                         "tini -- /fabric-pe..._tcp" 2 minutes ago     Up 2 minutes
4e3b040cc0      thejokeri/fabric-couchdb:arm64-0.4.18   "tini -- /docker-ent..." 2 minutes ago     Up 2 minutes
4369/tcp          5984/tcp                         "tini -- /fabric-pe..._tcp" 2 minutes ago     Up 2 minutes
4369/tcp          9100/tcp                        "network_couchdbDevice0.1.rcjkrtrumgjicxbkzvgilt" 2 minutes ago     Up 2 minutes
ubuntu@ubuntu:~$
```

Figura 39: Salida de contenedores en el nodo worker 1.

```
ubuntu@ubuntu:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
956674f0bb4c      dev-peer1.linkage.networkiot.com-linkage-1.0-29bb0d5085f84e45005273ec502368a84ae22d9a5ba75b80ca7e6c5e597ec3   "chaincode -peer.add..." About a minute ago Up About a minute
b431a2137192      dev-peer1.device.networkiot.com-device-1.0-59a2073420b5e6bca079887f9d3d573b64572b769deed86da4dee094b938bb  "chaincode -peer.add..." About a minute ago Up About a minute
1edd47802ad      dev-peer0.device.networkiot.com-device-1.0-189057ce73cf6b85ca8668928f76b099c628eff2a29d9696f415ef0b5a60fa2   "chaincode -peer.add..." 2 minutes ago     Up 2 minutes
94cd6122c43      thejokeri/fabric-peer:arm64-1.4.4   "peer node start" 2 minutes ago     Up 2 minutes
9f4c211b05      thejokeri/fabric-peer:arm64-1.4.4   "sh -c 'fabric-ca-se..." 3 minutes ago     Up 2 minutes
7054/tcp          7054/tcp                         "tini -- /docker-ent..." 3 minutes ago     Up 3 minutes
bd82f42a56f      thejokeri/fabric-couchdb:arm64-0.4.18   "tini -- /network_couchdbDevice1.1.cts5j7ncl1dhw33nc1p4q" 3 minutes ago     Up 3 minutes
cd3a895d32e      thejokeri/fabric-peer:arm64-1.4.4   "peer node start" 3 minutes ago     Up 3 minutes
9589b992d769      thejokeri/fabric-peer:arm64-1.4.4   "network_peerlinkage.1.b3mzkssql5zda0t7w1y3v4g5a" 3 minutes ago     Up 3 minutes
ubuntu@ubuntu:~$
```

Figura 40: Salida de contenedores en el nodo worker 2.

```
ubuntu@ubuntu:~/IoTBlockchainAPP/infrastructure/config-network/network$ ./networkiot.sh remove

Exportando claves de Device peer...
Exportando claves de Linkage peer...
Parando servicios...
Removing service network_caDevice
Removing service network_caLinkage
Removing service network_cli
Removing service network_couchdbDevice0
Removing service network_couchdbDevice1
Removing service network_couchdbLinkage0
Removing service network_couchdbLinkage1
Removing service network_orderer
Removing service network_peer0Device
Removing service network_peerLinkage
Removing service network_peer1Device
Removing service network_peerLinkage
Done

Eliminando volúmenes
Removing Stack Volumes from network:
--- Removing volume network_orderer.networkiot.com
Waiting for dangling volume network_orderer.networkiot.com
Volume network_orderer.networkiot.com is dangling, proceeding to its deletion
network_orderer.networkiot.com
Done

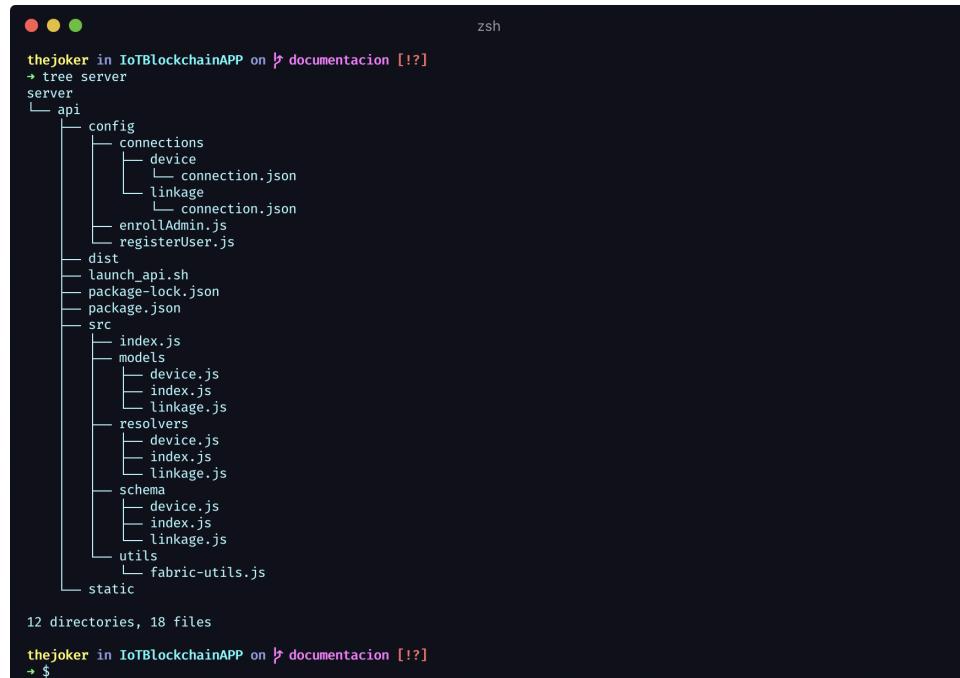
Eliminando red networkiot...
networkiot
Done

Eliminando configuración previa...
Done
ubuntu@ubuntu:~/IoTBlockchainAPP/infrastructure/config-network/network$
```

Figura 41: Opción remove del script networkiot.

4.3. Desarrollo del Back end.

En este capítulo, se mencionará la arquitectura del lado del servidor utilizando GraphQL, Apollo Server y Express, implementado con Node.js. Se utilizará el SDK de Hyperledger Fabric para realizar consultas y mantener un registro en la red Blockchain [21, 23, 17]. La API consta de tres partes esenciales: Models, Resolvers y Schemas. Además de un apartado de config para la creación y gestión de usuarios y una carpeta utils para realizar la conexión directa a la red Blockchain [73] (ver fig. 42).



```
thejoker in IoTBlockchainAPP on ↵ documentacion [!?]
→ tree server
server
└── api
    ├── config
    │   ├── connections
    │   │   ├── device
    │   │   │   └── connection.json
    │   │   └── linkage
    │   │       └── connection.json
    │   ├── enrollAdmin.js
    │   └── registerUser.js
    ├── dist
    ├── launch_api.sh
    ├── package-lock.json
    └── package.json
└── src
    ├── index.js
    ├── models
    │   ├── device.js
    │   │   ├── index.js
    │   │   └── linkage.js
    ├── resolvers
    │   ├── device.js
    │   │   ├── index.js
    │   │   └── linkage.js
    ├── schema
    │   ├── device.js
    │   │   ├── index.js
    │   │   └── linkage.js
    └── utils
        └── fabric-utils.js
static

12 directories, 18 files
thejoker in IoTBlockchainAPP on ↵ documentacion [!?]
→ $
```

Figura 42: Tree de la carpeta server.

Los Schemas en GraphQL se define por sus tipos, las relaciones entre los tipos y su estructura. Por lo tanto, GraphQL utiliza un lenguaje de definición de esquemas (SDL). Sin embargo, el esquema no define de dónde provienen los datos. Esta responsabilidad es manejada por los Resolvers que se encargan de obtener los datos desde la red Blockchain. Son ellos, los que se encargan de mandar las peticiones para crear, consultar, actualizar y/o borrar [66]. Y por último, los Models son los encargados de manejar esas queries y formatear los datos recibidos en forma de objetos, aplicando el paradigma de orientado a objetos [81, 68].

Con la herramienta de GraphQL Playground, podemos realizar consultas directamente a la API y nos permite mostrar la documentación y los Schemas de la misma. Permite una ayuda durante el desarrollo y además elaborar

4 Desarrollo del sistema.

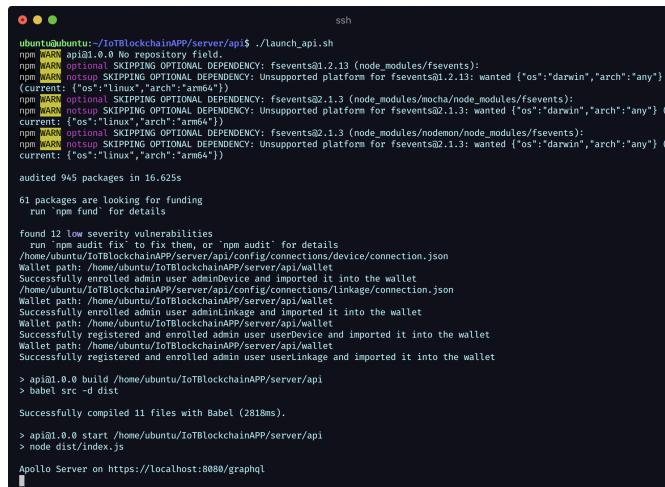
documentación sobre la API. (ver fig. 46, 47, 48a, 48b).

Para permitir conexiones por HTTPS en el lado del servidor, se ha utilizado Let's Encrypt y Certbot, una herramienta de EFF, conocido como Electronic Frontier Foundation, la principal organización sin ánimo de lucro centrada en la privacidad, la libertad de expresión y las libertades civiles en general en el mundo digital. Con ella podemos generar certificados SSL con la herramienta Certbot. Y Con Express indicamos la lectura de ficheros estáticos para permitir que se ejecute bajo HTTPS [15].

Para llevarlo acabo, es necesario establecer un dominio, por ello he utilizado la herramienta NO-IP, para crear dominios bajo DDNS. En el router se establece el DDNS indicando el nombre del dominio y se abre los puertos 80 y 443, para permitir conexiones HTTP y HTTPS. Todas las conexiones se redireccionan al servidor maestro (192.168.0.30).

Una vez creado el dominio, se creará tres ficheros: key.pem, cert.pem y chain.pem. Hacemos que Express lea los ficheros y cree el servidor bajo el puerto 443 y cualquier otra conexión por el puerto 80, sea redireccionada. Con ello hemos conseguido que la API sea accesible desde fuera y que tenga conexiones seguras.

Para levantar el servidor en el nodo maestro, he creado un script **launch_api.sh** que se va a ocupar de copiar los archivos de crypto-config, instalar las dependencias del package.json, inscribir al usuario admin, userDevice y userLinkage y comenzar el servicio escuchando por el puerto 443.



```
ubuntu@ubuntu:~/IoTBlockchainAPP/server/api$ ./launch_api.sh
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules/fsevents):
npm WARN   peer optional peer fsevents@>0.2.0 node_modules/electron-squirrel-startup/node_modules/fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"arm64"})
npm WARN   optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.3 (node_modules/mocha/node_modules/fsevents):
npm WARN     notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"arm64"})
npm WARN   optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.3 (node_modules/nodenode/node_modules/fsevents):
npm WARN     notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"arm64"})
audited 945 packages in 16.625s
61 packages are looking for funding
  run `npm fund` for details
found 12 low severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details
/home/ubuntu/IoTBlockchainAPP/server/api/config/connections/device/connection.json
Wallet path: /home/ubuntu/IoTBlockchainAPP/server/api/wallet
Successfully enrolled admin user admin and imported it into the wallet
/home/ubuntu/IoTBlockchainAPP/server/api/config/connections/linkage/connection.json
Wallet path: /home/ubuntu/IoTBlockchainAPP/server/api/wallet
Successfully enrolled admin user adminLinkage and imported it into the wallet
Wallet path: /home/ubuntu/IoTBlockchainAPP/server/api/wallet
Successfully registered and enrolled admin user userDevice and imported it into the wallet
Wallet path: /home/ubuntu/IoTBlockchainAPP/server/api/wallet
Successfully registered and enrolled admin user userLinkage and imported it into the wallet
> apim@0.0 build /home/ubuntu/IoTBlockchainAPP/server/api
> babel src -d dist
Successfully compiled 11 files with Babel (2818ms).
> apim@0.0 start /home/ubuntu/IoTBlockchainAPP/server/api
> node dist/index.js
Apollo Server on https://localhost:8089/graphql
```

Figura 43: Ejecución del script launch_api.

4.4. Desarrollo del Front end y despliegue.

Para elaborar la interfaz del usuario, he utilizado Gatsby.js que es un generador de sitios web estáticos para aplicaciones React, y soportado por Apollo Client. Gatsby producirá archivos HTML estáticos precargados de antemano, lo cual permite renderizar la interfaz muy rápidamente. Una de las ventajas que tiene Gatsby, es que utiliza GraphQL para realizar las consultas y por tanto, es un candidato perfecto para la API elaborada [59, 43].

También tiene un rico ecosistema de plugins, que para este proyecto he utilizado el plugin de PWA, en el que hay que indicar un manifiesto, añadir el plugin de offline y trabajar bajo HTTPS [56, 32, 33].

Para que Gatsby pueda trabajar de forma dinámica, se ha configurado la interfaz para realizar consultas a la API con Apollo Client [63]. Además de crear una utilidad que permite renderizar tablas dependiendo de los datos que reciba [67].

```
thejoker in IoTBlockchainAPP on ↵ documentacion [!?]
→ tree web
web
└── mocks
    ├── file-mock.js
    └── gatsby.js
└── gatsby-browser.js
└── gatsby-config.js
└── gatsby-ssr.js
└── jest-preprocess.js
└── jest.config.js
└── loadershim.js
└── package-lock.json
└── package.json
└── src
    ├── components
    │   ├── _tests
    │   │   ├── snapshots
    │   │   │   └── header.js.snap
    │   │   └── header.js
    │   ├── form-device.js
    │   ├── form-linkage.js
    │   ├── header.js
    │   ├── layout.js
    │   ├── seo.js
    │   ├── sidebar.js
    │   └── table.js
    ├── images
    │   └── icon.png
    └── pages
        ├── 404.js
        ├── devices.js
        ├── edit-device.js
        ├── edit-linkage.js
        ├── index.js
        ├── linkages.js
        ├── new-device.js
        └── new-linkage.js
    └── styles
        └── global.css
└── static
    └── favicon.ico
9 directories, 30 files
thejoker in IoTBlockchainAPP on ↵ documentacion [!?]
→ █
```

Figura 44: Tree de la carpeta web.

4 Desarrollo del sistema.

Cabe destacar, que se ha llevado a cabo test unitarios, para comprobar que se renderiza bien la interfaz de usuario, además de vincular el repositorio con Continious Integration (CI), para que en cada actualización que se realice en el código, se lancen los test antes de desplegarlo en Gatsby Cloud [70, 13].

Finalmente, la arquitectura de la aplicación y de la infraestructura se puede ver en la siguiente figura y las URL son:

- <https://iotblockchainapp.gtsb.io/> (Front end)
- <https://iotblockchainapi.ddns.net/> (Back end)

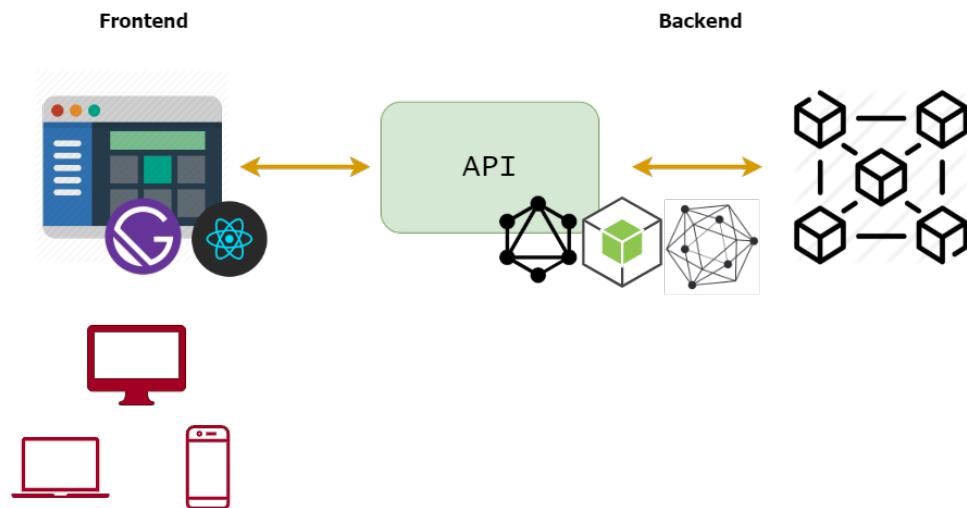


Figura 45: Arquitectura de la aplicación.

5. Conclusiones.

Actualmente, la tecnología IoT ha tenido un gran impacto dentro de la sociedad, y aumentará considerablemente en el futuro por las redes 5G. Son inseguros e incapaces de defenderse, principalmente por la limitación de recursos que tienen. Por ello es requerido realizar esfuerzos para implementar mecanismos hardware o software, para palidiar dicha carencia. En este documento hemos realizado un estudio y revisión de los principales problemas de seguridad de IoT y como la tecnología Blockchain puede afrontarlo. Se ha elaborado una aplicación para llevar el registro distribuido para IoT utilizando Hyperledger Fabric. Se ha creado una red Blockchain local, para llevar la gestión y administración de dispositivos domóticos, mediante contratos inteligentes, además de implementar una aplicación web para facilitar al usuario, accesible desde cualquier parte, segura y ligera. Aunque, la prueba de concepto se ha empezado con un pequeño número de dispositivos, el objetivo de este proyecto es poder llevarlo a gran escala y que sea posible llevarlo a nivel empresarial con múltiples dispositivos.

Pese a las dificultades que he tenido con Hyperledger Fabric para implementar la red Blockchain en arquitecturas ARM, ha sido muy gratificante conseguir llevarlo a cabo y poder implementar una aplicación web, aplicando todas las etapas de desarrollo de la ingeniería software, además de utilizar tecnologías innovadoras y usadas a nivel empresarial.

Personalmente, aunque considero que la tecnología Blockchain permite resolver, en cierta medida, los problemas de los dispositivos IoT. Aún así, es necesario realizar ciertas mejoras para decrementar el tiempo de validación de las transacciones entre dispositivos y que sea escalable por la gran cantidad que hay. Pero por otro lado, veo factible el uso de la tecnología Blockchain en otros ámbitos, como sustituto de las bases de datos, para permitir la trazabilidad, fiabilidad e inmutabilidad. En casos como el de los historiales clínicos de pacientes, registros de propiedad privada y propiedad intelectual, uso en la banca o la educación, entre otros.

6. Trabajos futuros.

Si bien se cumplieron los objetivos planteados inicialmente, existen posibles mejoras y se dejan como propuestas abiertas para futuras investigaciones:

- Una mejor administración de usuarios, utilizando role, mejor administración de certificados de autoridad, además de permitir que varios usuarios puedan llevar un registro de sus propios dispositivos IoT.

6 Trabajos futuros.

- Utilizar una base de datos noSQL como apoyo para la red Blockchain.
- Implementar acceso de seguridad con el framework Oauth2.
- Permitir que la aplicación pueda realizar un escaneo a todos los dispositivos conectados en la red privada mediante ARP, evitando que el usuario tenga que introducir a mano uno por uno cada dispositivo que posea.
- Comprobar que funcione realmente la activación y desactivación de los actuadores, cuando reciba una señal desde su sensor.
- Utilizar sistema de manejo de colas de mensaje para multiples dispositivos como la tecnología SQS de AWS o RabbitMQ.
- Mejorar las llamadas de los datos asíncronos, las devoluciones de llamada y los programas basados en eventos del Front end, utilizando RxJS.
- Mejoras en la validaciones en los formularios del Front end.
- Realizar más test tanto en el Front end como Back end y utilizar lint, para implementar buenas prácticas a nivel de producción.
- Permitir que la infraestructura de la red Blockchain sea más escalable, para que permita la incorporación de nuevos participantes, nodos, canales, chaincode y hasta nuevas máquinas anfitrionas.
- Realizar cambios en el código de la infraestructura para sea adaptable a diferentes máquinas similares a las Raspberry Pi, como Arduino, Banana Pi, etc.

7 Anexo: Vista de la aplicación web e instalación PWA.

7. Anexo: Vista de la aplicación web e instalación PWA.

7.1. Vista de la API.

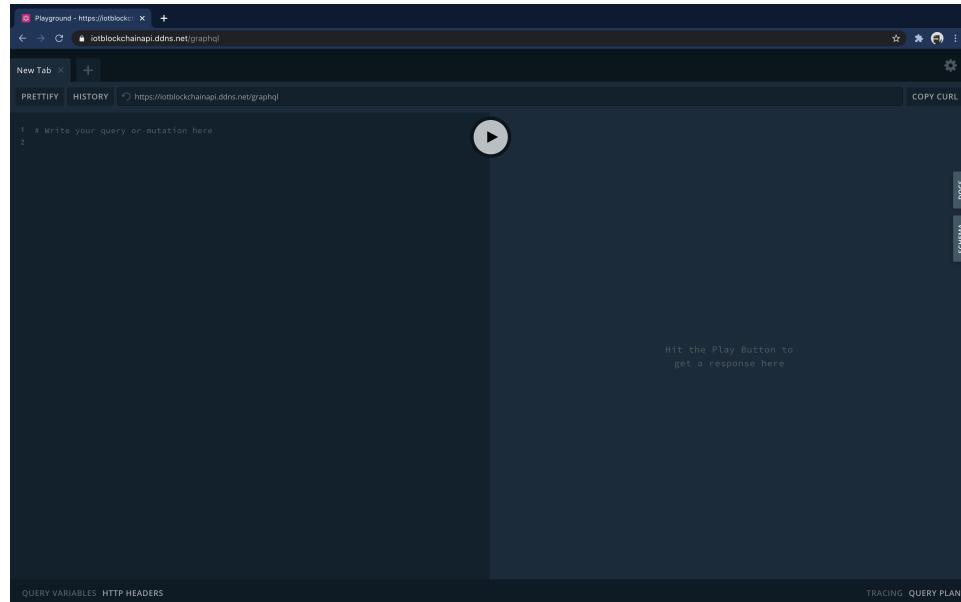


Figura 46: GraphQL API endpoint.

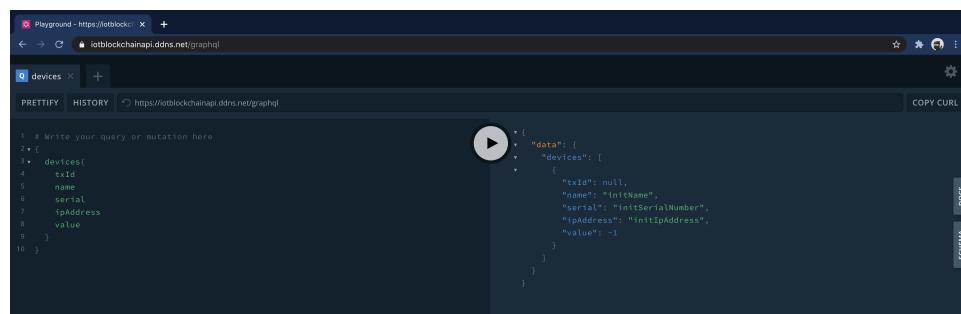


Figura 47: Consulta a GraphQL API endpoint.

7 Anexo: Vista de la aplicación web e instalación PWA.

(a) Docs GraphQL API.

QUERIES

- _ Boolean
- devices: [Device!]
- device(...): Device
- historyDevice(...): [Device]
- linkages(...): [Linkage]
- linkage(...): Linkage
- historyLinkage(...): [Linkage]

MUTATIONS

- _ Boolean
- addDevice(...): Device!
- setValue(...): Boolean!
- deleteDevice(...): Boolean!
- addLinkage(...): Linkage!
- updateLinkage(...): Linkage!
- deleteLinkage(...): Boolean!
- enable(...): Boolean!
- disable(...): Boolean!

SCHEMA

```

directive @cacheControl(
  maxAge: Int!
  scope: CacheControlScope
) on FIELD_DEFINITION | OBJECT | INTERFACE
enum CacheControlScope {
  PUBLIC
  PRIVATE
}

type Device {
  txId: String!
  name: String!
  serial: String!
  ipAddress: String!
  value: Int!
}

type Linkage {
  txId: String!
  id: ID!
  sensor: String!
  cond: String!
  actuator: String!
  status: Boolean!
  region: String!
}

type Mutation {
  _: Boolean!
  addDevice(name: String!, serial: String!, ipAddress: String!): Device!
  setValue(serial: String!, value: Int!): Boolean!
  deleteDevice(serial: String!): Boolean!
  addLinkage(
    sensor: String!
    cond: String!
    actuator: String!
    region: String!
  ): Linkage!
  updateLinkage(id: ID!, cond: String!, region: String!): Linkage!
  deleteLinkage(id: ID!): Boolean!
  enable(id: ID!): Boolean!
  disable(id: ID!): Boolean!
}

type Query {
  _: Boolean!
  devices: [Device!]!
  device(serial: String!): Device
  historyDevice(serial: String!): [Device]
  linkages(sensor: String!): [Linkage]
  linkage(id: ID!): Linkage
  historyLinkage(id: ID!): [Linkage]
}

```

(b) Schema GraphQL API.

Figura 48: Información de GraphQL API.

7 Anexo: Vista de la aplicación web e instalación PWA.

7.2. Vista del Front end.

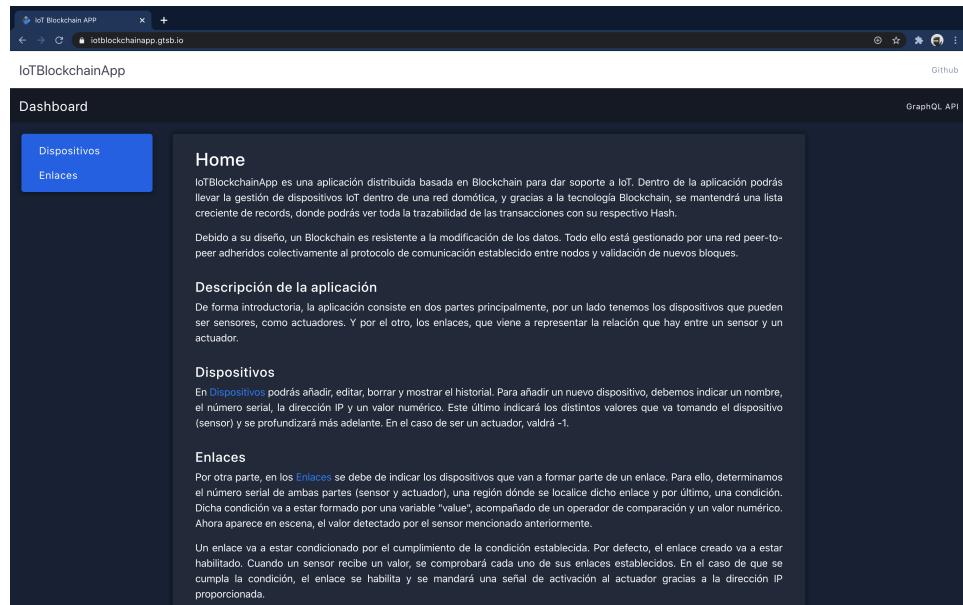


Figura 49: Página inicial.

Vista dispositivo.

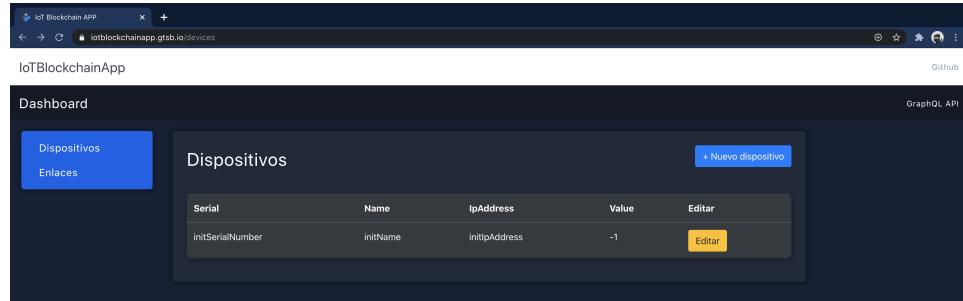


Figura 50: Vista dispositivos.

7 Anexo: Vista de la aplicación web e instalación PWA.



Figura 51: Añadir dispositivo.

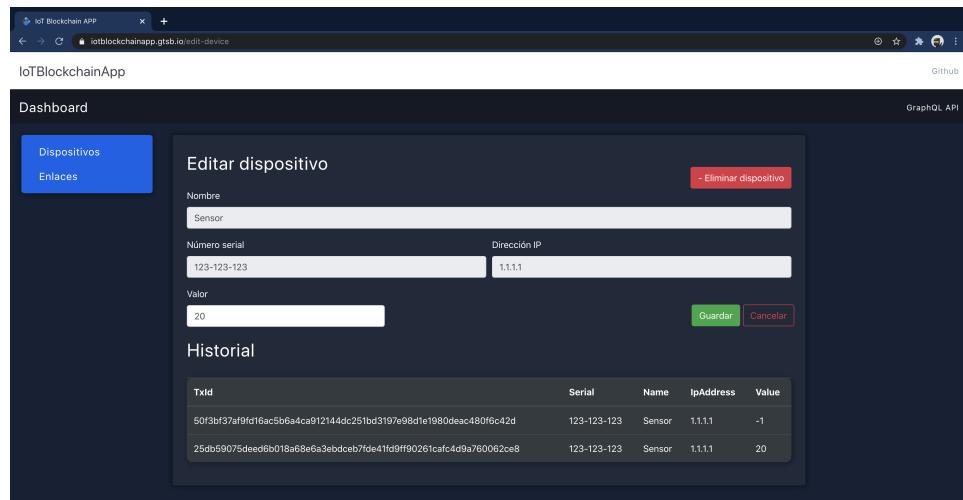


Figura 52: Editar dispositivo.

Vista enlace.

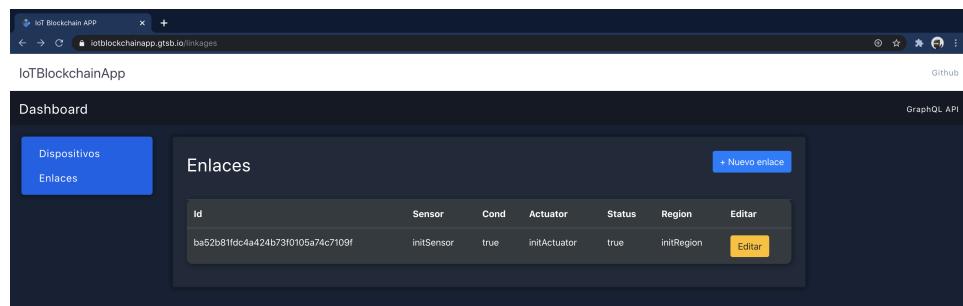


Figura 53: Vista enlaces.

7 Anexo: Vista de la aplicación web e instalación PWA.

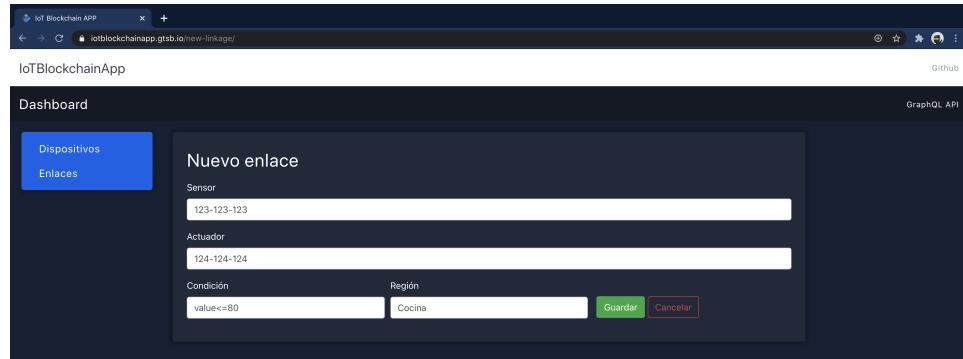


Figura 54: Añadir enlace.

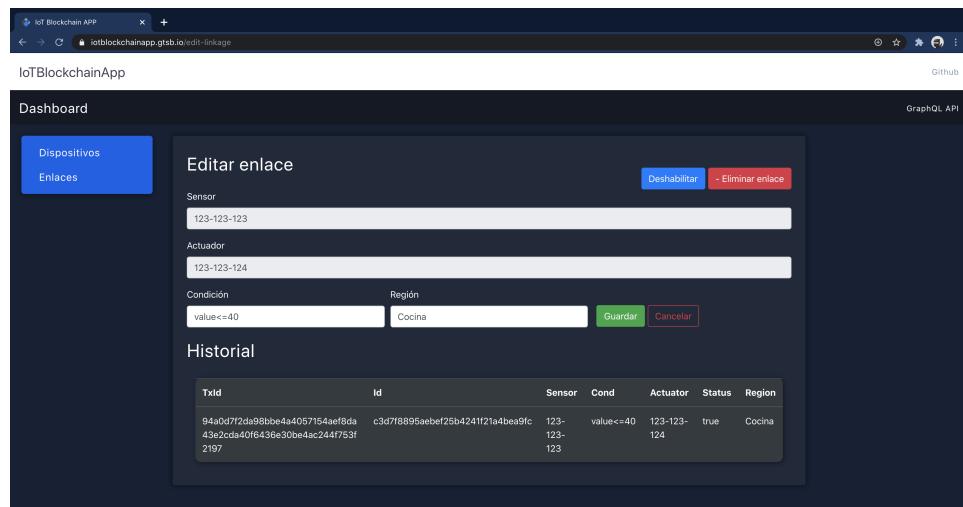


Figura 55: Editar enlace.

7.3. Instalación de PWA.

Portatil.



Figura 56: Instalación PWA en portatil.

7 Anexo: Vista de la aplicación web e instalación PWA.

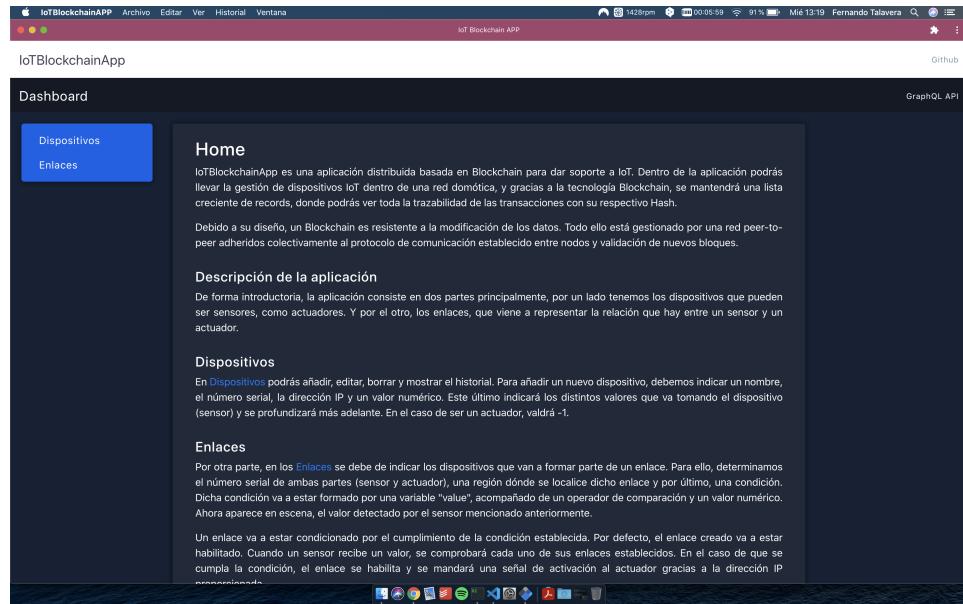
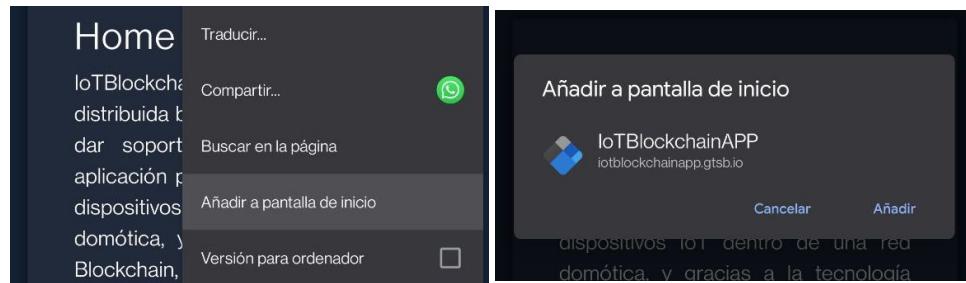


Figura 57: App PWA en portatil.

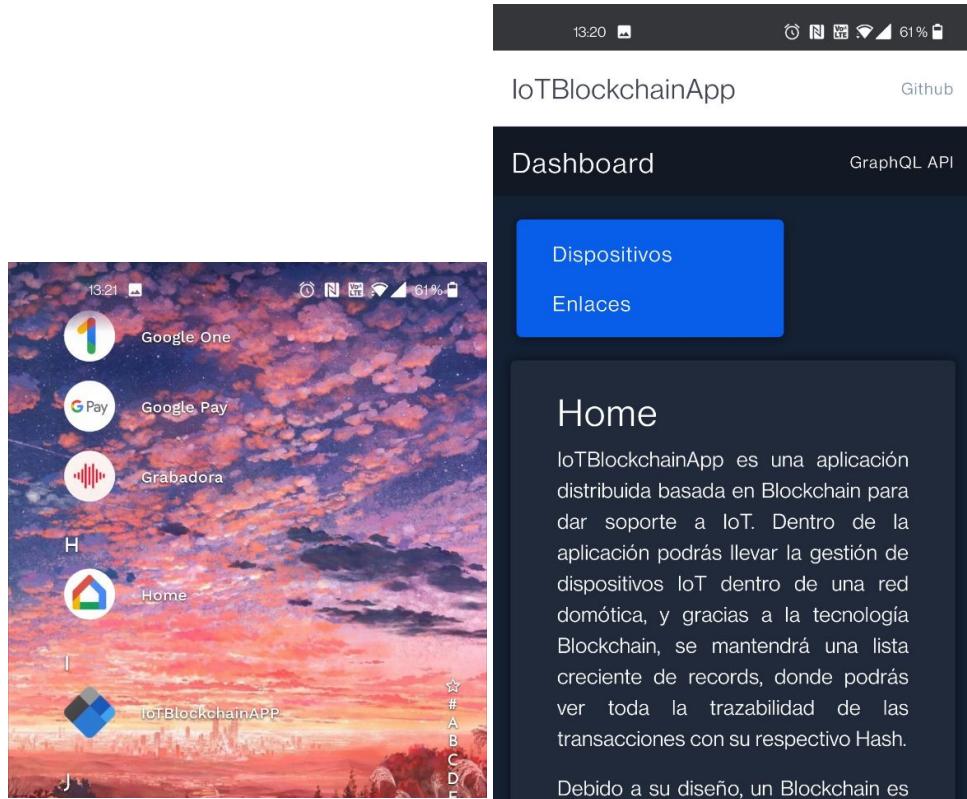
Móvil.



(a) Instalación PWA en móvil parte 1.

(b) Instalación PWA en móvil parte 2.

Figura 58: Instalación PWA en móvil.



(a) Icono de la aplicación en el móvil.

The image shows a screenshot of the IoTBlockchainApp mobile application. At the top, it says "IoTBlockchainApp" and "Github". Below that is a "Dashboard" section with "GraphQL API" and "Dispositivos" and "Enlaces" buttons. The main area is titled "Home" and contains descriptive text about the app's purpose and how blockchain technology ensures data integrity and traceability through hashes. There is also a "Descripción de la aplicación" section at the bottom.

(b) App PWA en móvil.

Figura 59: Aplicación móvil.

8. Bibliografía y recursos web

Referencias

- [1] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014.
- [2] Ali Dorri y col. «Blockchain for IoT security and privacy: The case study of a smart home». En: *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*. IEEE. 2017, págs. 618-623.
- [3] Seyoung Huh, Sangrae Cho y Soohyung Kim. «Managing IoT devices using blockchain platform». En: *2017 19th international conference on advanced communication technology (ICACT)*. IEEE. 2017, págs. 464-467.
- [4] Nir Kshetri. «Can blockchain strengthen the internet of things?» En: *IT professional* 19.4 (2017), págs. 68-72.
- [5] Minhaj Ahmad Khan y Khaled Salah. «IoT security: Review, blockchain solutions, and open challenges». En: *Future Generation Computer Systems* 82 (2018), págs. 395-411.
- [6] Oscar Novo. «Blockchain meets IoT: An architecture for scalable access management in IoT». En: *IEEE Internet of Things Journal* 5.2 (2018), págs. 1184-1195.
- [7] Ana Reyna y col. «On blockchain and its integration with IoT. Challenges and opportunities». En: *Future generation computer systems* 88 (2018), págs. 173-190.
- [8] *A Blockchain Platform for the Enterprise*. URL: <https://hyperledger-fabric.readthedocs.io/en/latest/index.html>. [Último acceso: 17 Octubre 2020].
- [9] *API - Wikipedia*. URL: <https://en.wikipedia.org/wiki/API>. [Último acceso: 27 Octubre 2020].
- [10] *B-money - Bitcoin Wiki*. URL: <https://en.bitcoin.it/wiki/B-money>. [Último acceso: 20 Octubre 2020].
- [11] Grant Bartel. *What is a Dapp? A Guide to Ethereum Dapps*. URL: <https://www.freecodecamp.org/news/what-is-a-dapp-a-guide-to-ethereum-dapps/>. [Último acceso: 25 Octubre 2020].
- [12] *Bootstrap (front-end framework) - Wikipedia*. URL: [https://en.wikipedia.org/wiki/Bootstrap_\(front-end_framework\)](https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)). [Último acceso: 28 Octubre 2020].

Referencias

- [13] *Building a JavaScript and Node.js project - Travis CI.* URL: <https://docs.travis-ci.com/user/languages/javascript-with-nodejs/>. [Último acceso: 16 Octubre 2020].
- [14] Jen Clark. *What is the internet of Things, and how does it work?* URL: <https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/>. [Último acceso: 20 Octubre 2020].
- [15] Flavio Copes. *Setup Let's Encrypt for Express.* URL: <https://flaviocopes.com/express-letsencrypt-ssl/>. [Último acceso: 16 Octubre 2020].
- [16] Salman Dabbakuti. *Start Developing Hyperledger Fabric Chaincode in Node.js.* URL: <https://medium.com/coinmonks/start-developing-hyperledger-fabric-chaincode-in-node-js-e63b655d98db>. [Último acceso: 16 Octubre 2020].
- [17] Salman Dabbakuti. *Walkthrough of Hyperledger Fabric Node SDK and Client Application.* URL: <https://medium.com/datadriveninvestor/walkthrough-of-hyperledger-fabric-client-application-aae5222bdf3>. [Último acceso: 16 Octubre 2020].
- [18] Shaumik Daityari. *Angular vs React vs Vue: Which Framework to Choose in 2020.* URL: <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>. [Último acceso: 28 Octubre 2020].
- [19] *Deploy a stack to a swarm — Docker Documentation.* URL: <https://docs.docker.com/engine/swarm/stack-deploy/>. [Último acceso: 17 Octubre 2020].
- [20] DockerHub. URL: <https://hub.docker.com/search?q=thejoker&type=image&architecture=arm64>. [Último acceso: 12 Noviembre 2020].
- [21] *Documentation Home - Apollo Basics - Apollo GraphQL Docs.* URL: <https://www.apollographql.com/docs/>. [Último acceso: 16 Octubre 2020].
- [22] *Ethereum Mainnet Statistics.* URL: <https://ethernodes.org/>. [Último acceso: 21 Octubre 2020].
- [23] *Express - Node.js web application framework.* URL: <https://expressjs.com/>. [Último acceso: 16 Octubre 2020].
- [24] *fabric.* URL: <https://github.com/hyperledger/fabric>. [Último acceso: 12 Noviembre 2020].
- [25] *fabric-baseimage.* URL: <https://github.com/hyperledger/fabric-baseimage>. [Último acceso: 12 Noviembre 2020].
- [26] *fabric-samples/byfn.sh at release-1.4 · hyperledger/fabric-samples.* URL: <https://github.com/hyperledger/fabric-samples/blob/release-1.4/first-network/byfn.sh>. [Último acceso: 17 Octubre 2020].

Referencias

- [27] *Fork fabric-baseimage*. URL: <https://github.com/Thejokeri/fabric-baseimage>. [Último acceso: 9 Noviembre 2020].
- [28] Jake Frankenfield. *Blockchain-as-a-Service (BaaS) Definition*. URL: <https://www.investopedia.com/terms/b/blockchainasaservice-baas.asp>. [Último acceso: 25 Octubre 2020].
- [29] Jake Frankenfield. *Proof of Stake (PoS)*. URL: <https://www.investopedia.com/terms/p/proof-stake-pos.asp>. [Último acceso: 25 Octubre 2020].
- [30] Paulo Frazao. *Happy Pi Day with Docker and Raspberry Pi - Docker Blog*. URL: <https://www.docker.com/blog/happy-pi-day-docker-raspberry-pi/>. [Último acceso: 17 Octubre 2020].
- [31] *Gatsby*. URL: <https://www.gatsbyjs.com/>. [Último acceso: 26 Octubre 2020].
- [32] *gatsby-plugin-manifest — Gatsby*. URL: <https://www.gatsbyjs.com/plugins/gatsby-plugin-manifest/?=>. [Último acceso: 16 Octubre 2020].
- [33] *gatsby-plugin-offline — Gatsby*. URL: <https://www.gatsbyjs.com/plugins/gatsby-plugin-offline/?=>. [Último acceso: 16 Octubre 2020].
- [34] *GraphQL - Wikipedia*. URL: <https://en.wikipedia.org/wiki/GraphQL>. [Último acceso: 27 Octubre 2020].
- [35] *GraphQL vs REST - A comparison*. URL: <https://www.howtographql.com/basics/1-graphql-is-the-better-rest/>. [Último acceso: 27 Octubre 2020].
- [36] *Hashcash - Wikipedia*. URL: <https://en.wikipedia.org/wiki/Hashcash>. [Último acceso: 20 Octubre 2020].
- [37] Grayson hicks. *What is Gatsby.js — Mediacurrent*. URL: <https://www.mediacurrent.com/blog/what-is-gatsbyjs/>. [Último acceso: 28 Octubre 2020].
- [38] Chris Hoffman. *What Are Progressive Web Apps?* URL: <https://www.howtogeek.com/342121/what-are-progressive-web-apps/>. [Último acceso: 26 Octubre 2020].
- [39] Parikshit Hooda. *Proof of Work (PoW) Consensus - GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/proof-of-work-pow-consensus/>. [Último acceso: 20 Octubre 2020].
- [40] *Hyperledger Fabric SDK for node.js Index*. URL: <https://hyperledger.github.io/fabric-sdk-node/release-1.4/index.html>. [Último acceso: 16 Octubre 2020].
- [41] *Info Jobs*. URL: <https://www.infojobs.net/>. [Último acceso: 9 Noviembre 2020].

Referencias

- [42] *Install Ubuntu on Raspberry PI.* URL: <https://ubuntu.com/download/raspberry-pi/thank-you?version=18.04.4&architecture=arm64+raspi3>. [Último acceso: 12 Noviembre 2020].
- [43] *Introduction to Apollo Client - Client (React) - Apollo GraphQL Docs.* URL: <https://www.apollographql.com/docs/react/>. [Último acceso: 28 Octubre 2020].
- [44] *Introduction to Apollo Server - Apollo Server - Apollo Graphql Docs.* URL: <https://www.apollographql.com/docs/apollo-server/>. [Último acceso: 28 Octubre 2020].
- [45] *IoTBlockchainAPP.* URL: <https://github.com/Thejokeri/IoTBlockchainAPP>. [Último acceso: 9 Noviembre 2020].
- [46] *Lighthouse — Tools for Web Developers.* URL: https://developers.google.com/web/tools/lighthouse/?utm_source=devtools. [Último acceso: 2 Noviembre 2020].
- [47] Demiro Massessi. *Public Vs Private Blockchain In a Nutshell.* URL: <https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f>. [Último acceso: 22 Octubre 2020].
- [48] Jonathan Meier. *Install Docker and Docker-Compose on your Raspberry Pi.* URL: <https://jonathanmeier.io/install-docker-and-docker-compose-raspberry-pi/>. [Último acceso: 17 Octubre 2020].
- [49] Fernando Talavera Mendoza. *Proyecto de SWAP (UGR).* URL: <https://github.com/Thejokeri/SWAP-2017-2018>. [Último acceso: 26 Octubre 2020].
- [50] Sarah Mischinger. *3 reasons why you should consider Gatsby.js for your next project.* URL: <https://www.storyblok.com/tp/3-reasons-why-you-should-consider-gatsby-js-for-your-next-project>. [Último acceso: 28 Octubre 2020].
- [51] Joe Motacek. *Hyperledger Fabric v1.0 on a Raspberry Pi Docker Swarm — Joe Motacek.* URL: <https://www.joemotacek.com/2017/08/04/hyperledger-fabric-v1-0-on-a-raspberry-pi-docker-swarm-part-1/>. [Último acceso: 17 Octubre 2020].
- [52] *NO-IP.* URL: <https://www.noip.com/>. [Último acceso: 9 Noviembre 2020].
- [53] *node.js - What is Express.js? - Stack Overflow.* URL: <https://stackoverflow.com/questions/12616153/what-is-express-js>. [Último acceso: 28 Octubre 2020].
- [54] Priyesh Patel. *What exactly is Node.js?* URL: <https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/>. [Último acceso: 28 Octubre 2020].

Referencias

- [55] *Peer-to-peer - Wikipedia*. URL: <https://en.wikipedia.org/wiki/Peer-to-peer>. [Último acceso: 20 Octubre 2020].
- [56] *Progressive Web Apps (PWAs) — Gatsby*. URL: <https://www.gatsbyjs.com/docs/progressive-web-app/>. [Último acceso: 16 Octubre 2020].
- [57] *Raspberry PI OS*. URL: <https://www.raspberrypi.org/downloads/>. [Último acceso: 12 Noviembre 2020].
- [58] *Raspberry Pi 4 Model B specifications - Raspberry Pi*. URL: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>. [Último acceso: 20 Octubre 2020].
- [59] *React - A JavaScript library for building user interfaces*. URL: <https://reactjs.org/>. [Último acceso: 16 Octubre 2020].
- [60] *Reactstrap - React Bootstrap 4 components*. URL: <https://reactstrap.github.io/>. [Último acceso: 16 Octubre 2020].
- [61] Lubos Rendek. *How to configure static IP address on Ubuntu 18.04 Bionic Beaver Linux - LinuxConfig.org*. URL: <https://linuxconfig.org/how-to-configure-static-ip-address-on-ubuntu-18-04-bionic-beaver-linux>. [Último acceso: 17 Octubre 2020].
- [62] Bhaskar S. *Building Docker Images for Hyperledger Fabric (ARM Edition)*. URL: <https://www.polarsparc.com/xhtml/Hyperledger-ARM-Build.html>. [Último acceso: 17 Octubre 2020].
- [63] *Setting Up Apollo Client 3 in a GatsbyJS APP - Youtube*. URL: <https://youtu.be/X5fE7iyBQ-0>. [Último acceso: 16 Octubre 2020].
- [64] Toshendra Kumar Sharma. *Top 2020 Blockchain Platforms for Building Blockchain-Based Applications*. URL: <https://www.blockchain-council.org/blockchain/top-2020-blockchain-platforms-for-building-blockchain-based-applications/>. [Último acceso: 25 Octubre 2020].
- [65] Nitish Singh. *Top Blockchain Platforms and Enterprise Solutions to Choose From*. URL: <https://101blockchains.com/blockchain-platforms/>. [Último acceso: 25 Octubre 2020].
- [66] Jonas Snellinckx. *Hyperledger Fabric & couchdb, fantastic queries and where to find them*. URL: <https://medium.com/wearetheledger/hyperledger-fabric-couchdb-fantastic-queries-and-where-to-find-them-f8a3aecef767>. [Último acceso: 16 Octubre 2020].
- [67] Tarak. *Create Dynamic Table from json in react js*. URL: <https://medium.com/@subalerts/create-dynamic-table-from-json-in-react-js-1a4a7b1146ef>. [Último acceso: 16 Octubre 2020].
- [68] *Thinking in Graphs — GraphQL*. URL: <https://graphql.github.io/learn/thinking-in-graphs/>. [Último acceso: 16 Octubre 2020].

Referencias

- [69] *Top Blockchain Platforms of 2020 for Blockchain Application.* URL: <https://www.leewayhertz.com/blockchain-platforms-for-top-blockchain-companies/>. [Último acceso: 25 Octubre 2020].
- [70] *Unit Testing — Gatsby.* URL: <https://www.gatsbyjs.com/docs/unit-testing/>. [Último acceso: 16 Octubre 2020].
- [71] *Using CouchDB - hyperledger-fabricdocs master documentation.* URL: https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html?highlight=LevelDB#why-couchdb. [Último acceso: 27 Octubre 2020].
- [72] Lionel Valdellon. *What is an SDK? Everything You Need to Know — CleverTap.* URL: <https://clevertap.com/blog/what-is-an-sdk/>. [Último acceso: 27 Octubre 2020].
- [73] Mike Vercoelen. *A scalable, modular structure for your GraphQL Node.js API - Codersmind.* URL: <https://codersmind.com/scalable-modular-structure-graphql-node-api/#setting-up-boilerplate>. [Último acceso: 16 Octubre 2020].
- [74] Shermin Voshmgir. *What is a Smart Contract? Auto enforceable Code - Blockchain.* URL: <https://blockchainhub.net/smart-contracts/>. [Último acceso: 22 Octubre 2020].
- [75] *Web-Based Application: What It Is, and Why You Should Use It.* URL: <https://lvivity.com/web-based-applications>. [Último acceso: 26 Octubre 2020].
- [76] *What is a Raspberry Pi.* URL: <https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>. [Último acceso: 20 Octubre 2020].
- [77] *What is a Raspberry Pi? — Opensource.com.* URL: <https://opensource.com/resources/raspberry-pi>. [Último acceso: 20 Octubre 2020].
- [78] *What is Docker Swarm? — Sumo Logic.* URL: <https://www.sumologic.com/glossary/docker-swarm/>. [Último acceso: 27 Octubre 2020].
- [79] *What is SEO?* URL: <https://moz.com/learn/seo/what-is-seo>. [Último acceso: 2 Noviembre 2020].
- [80] *Why Hyperledger Fabric? - hyperledger-fabricdocs master documentation.* URL: <https://hyperledger-fabric.readthedocs.io/en/release/whyfabric.html>. [Último acceso: 26 Octubre 2020].
- [81] Robin Wieruch. *GraphQL Server Tutorial with Apollo Server and Express - RWieruch.* URL: <https://www.robinwieruch.de/graphql-apollo-server-tutorial>. [Último acceso: 16 Octubre 2020].
- [82] Jairo Daniel Bautista Castro. *Desarrollo De Una Red Blockchain Para La Gestión De Registros Académicos.* Trabajo Fin de Máster de Ingeniería Informática, Septiembre de 2019.

Referencias

- [83] Matilde Cabrera González. *Aplicación Distribuida Usando una Red Blockchain para la gestión y control de suministros de medicamentos.* Trabajo Fin de Grado de Ingeniería Informática, Junio de 2018.