GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GESTIÓN DE EXPEDIENTES MÉDICOS MEDIANTE IDENTIDADES DIGITALES CON BLOCKCHAIN

Autor: Javier Julio Estívariz López

Director: Atilano Fernández-Pacheco Sánchez-Migallón

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
Gestión de Expedientes Médicos mediante Identidades Digitales con Blockchain
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el
curso académico 2021/22 es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos.
El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido
tomada de otros documentos está debidamente referenciada.

Fdo.: Javier Julio Estívariz López Fecha: 07/ 07/ 2022

Autorizada la entrega del proyecto
EL DIRECTOR DEL PROYECTO

71216314B ATILANO FERNÁNDEZ-PACHECO

Firmado digitalmente por 71216314B ATILANO FERNÁNDEZ-PACHECO
Fecha: 2022.07.05 16:30:14 +02'00'

Fdo.: Atilano Fernández-Pacheco Sánchez-Migallón Fecha: 07/ 07/ 2022

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GESTIÓN DE EXPEDIENTES MÉDICOS MEDIANTE IDENTIDADES DIGITALES CON BLOCKCHAIN

Autor: Javier Julio Estívariz López

Director: Atilano Fernández-Pacheco Sánchez-Migallón

Madrid

Agradecimientos

A mi familia

GESTIÓN DE EXPEDIENTES MÉDICOS MEDIANTE IDENTIDADES DIGITALES CON BLOCKCHAIN

Autor: Estívariz López, Javier Julio.
Director: Fernández-Pacheco Sánchez-Migallón, Atilano.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

El siguiente proyecto estudia el uso de la tecnología Blockchain para su aplicación en el entorno de la sanidad. Utilizando esta tecnología y el concepto de identidad digital, se busca desarrollar una aplicación mediante la cual pacientes, médicos y hospitales sean capaces de compartir expedientes médicos entre ellos de manera descentralizada, asegurando confidencialidad y seguridad en el proceso.

Palabras clave: Blockchain, Expediente médico, SmartContract, DApp

1. Introducción

Hoy en día la Sanidad Española carece de un sistema por el cual sea posible transferir expedientes médicos de manera descentralizada y rápida entre los distintos hospitales, médicos y pacientes.

La tecnología Blockchain fue creada por Satoshi Nakamoto en el año 2008 para ser utilizada como un libro de cuentas de doble entrada de la criptomoneda BitCoin. Esta tecnología se ha ido desarrollando hasta la actualidad y con ella se pueden desarrollar aplicaciones mediante *SmartContracts.* Con ella se pueden asegurar la confidencialidad, seguridad e integridad de las transacciones realizadas.

2. Definición del proyecto

En este proyecto se pretende desarrollar una aplicación capaz de asegurar la confidencialidad, seguridad e integridad al compartir expedientes médicos entre médicos, hospitales y pacientes, haciendo uso de identidad digital con Blockchain.

3. Descripción de la solución

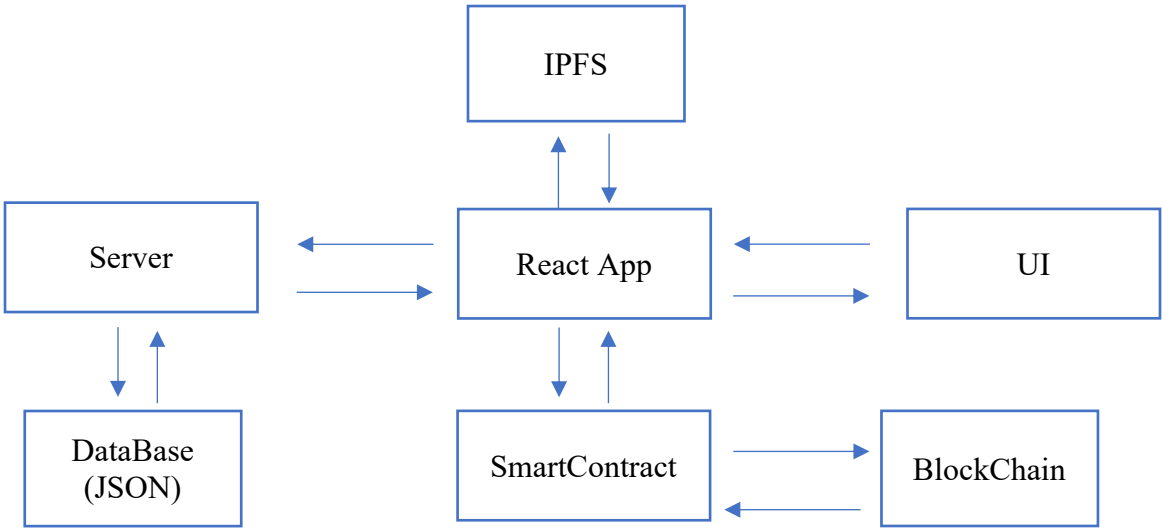
Se ha desarrollado una DApp en la que el usuario podrá logarse y, en función del rol (médico, paciente, hospital), tendrá acceso a distintas funcionalidades como subir un expediente, dar permisos a otro usuario para ver sus expedientes o consultar los expedientes de otro usuario.

Esta DApp se compone de un *SmartContract* encargado de almacenar un *hash* del expediente, que será subido a una red IPFS, y el identificador del paciente al que corresponde dicho expediente. Además, consta de un servidor encargado de llevar a cabo la identificación del usuario con una base de datos.

Para el desarrollo de esta DApp se ha utilizado la metodología *Agile*, con varias entregas periódicas con funcionalidad completa.

Finalmente, se ha llevado a cabo una memoria descriptiva de la solución expuesta.

4. Resultados



La DApp desarrollada permite a los usuarios subir expedientes a la red IPFS y los *hash* de los expedientes se almacenan en la Blockchain. El sistema de permisos e identificación se ha llevado a cabo mediante un fichero JSON que actúa a modo de base de datos.

La DApp no utiliza la identidad digital en Blockchain, ya que toda la identidad digital se lleva a cabo en el registro.

5. Conclusiones

La tecnología Blockchain es una tecnología relativamente joven pero que está teniendo grandes avances en los últimos años. Esta tecnología permite llevar a cabo transacciones de manera segura en internet de manera descentralizada y más sencilla y transparente.

En este proyecto se ha visto la viabilidad de la aplicación de esta tecnología al entorno de la sanidad, pero las posibles aplicaciones aún son desconocidas, pues Blockchain tiene un gran recorrido y evolución por delante.

6. Referencias

[1]. Batra, Gaurav. Olson, Remy. Pathak, Silphi. Santhanam, Nick. Soundararajan, Harish. "Blockchain 2.0: What's in store for the two ends—semiconductors (suppliers) and industrials (consumers)?". *McKinsey & Co.* Retrieved 25 July 2021.

[2]. *Castellanos, Sara.* "A Cryptocurrency Technology Finds New Use Tackling Coronavirus". *The Wall Street Journal*. Retrieved 21 October 2020.

MANAGEMENT OF MEDICAL RECORDS THROUGH DIGITAL IDENTITIES WITH BLOCKCHAIN

Author: Estívariz López, Javier Julio.
Supervisor: Fernández-Pacheco Sánchez-Migallón, Atilano.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

The following project studies the use of Blockchain technology for its application in the healthcare environment. Using this technology and the concept of digital identity, the aim is to develop an application through which patients, doctors and hospitals are able to share medical records among themselves in a decentralized manner, ensuring confidentiality and security in the process.

Keywords: Blockchain, Medical Record, SmartContract, DApp

1. Introduction

Today, Spanish Health does not have a system by which it is possible to transfer medical records in a decentralized and rapid manner between the different hospitals, doctors and patients.

Blockchain technology was created by Satoshi Nakamoto in 2008 to be used as a double entry ledger for the cryptocurrency BitCoin. This technology has been developed to date and with it applications can be developed using SmartContracts. With it, the confidentiality, security and integrity of the transactions carried out can be ensured.

2. Project definition

This project aims to develop an application capable of ensuring confidentiality, security and integrity when sharing medical records between doctors, hospitals and patients, making use of digital identity with Blockchain.

3. Description of the solution

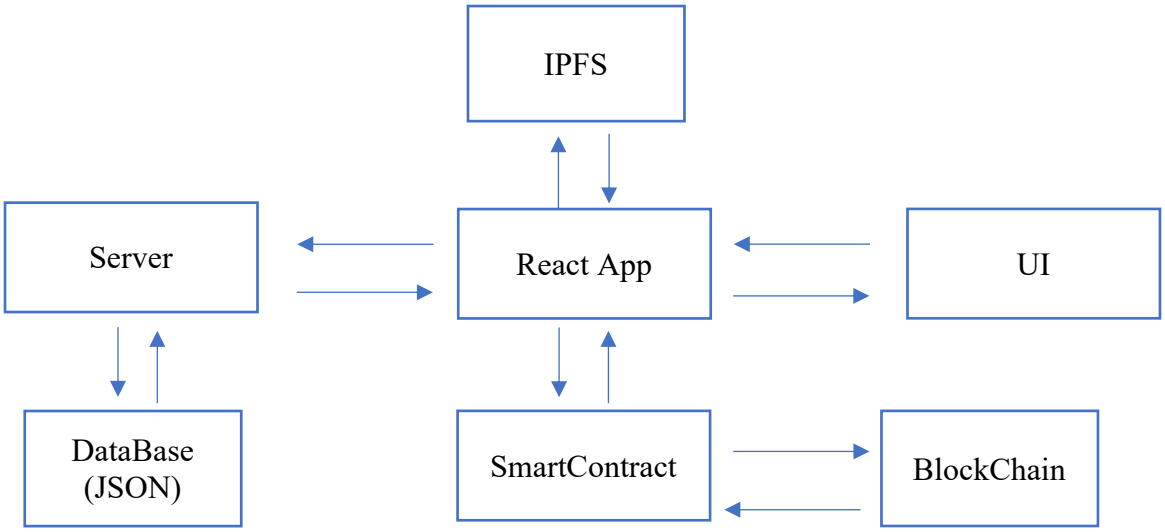
A DApp has been developed in which the user can log in and, depending on the role (doctor, patient, hospital), will have access to different functionalities such as uploading a file, giving permission to another user to view their files or consulting the files of another user.

This DApp consists of a SmartContract in charge of storing a *hash* of the file, which will be uploaded to an IPFS network, and the identifier of the patient to whom said file corresponds. In addition, it consists of a server in charge of carrying out the identification of the user with a database.

For the development of this DApp, the Agile methodology has been used, with several periodic deliveries with full functionality.

Finally, a descriptive memory of the solution has been carried out.

4. Results



The DApp developed allows users to upload files to the IPFS network and the file *hashes* are stored in the Blockchain. The permission and identification system has been carried out by means of a JSON file that acts as a database.

The DApp does not use digital identity on the Blockchain, as all digital identity takes place in the registry.

5. Conclusions

Blockchain technology is a relatively young technology but is also having great advances in recent years. This technology allows to carry out transactions safely on the internet in a decentralized, simpler and more transparent way.

This project has seen the viability of applying this technology to the healthcare environment, but the possible applications are still unknown, as Blockchain has a long journey and evolution ahead.

6. References

[1]. *Batra, Gaurav. Olson, Remy. Pathak, Silphi. Santhanam, Nick. Soundararajan, Harish.* "Blockchain 2.0: What's in store for the two ends—semiconductors (suppliers) and industrials (consumers)?". *McKinsey & Co.* Retrieved 25 July 2021.

[2]. *Castellanos, Sara.* "A Cryptocurrency Technology Finds New Use Tackling Coronavirus". *The Wall Street Journal*. Retrieved 21 October 2020.

Índice

1 Introducción

La Sanidad Española carece de un sistema descentralizado mediante el cual un paciente pueda transferir su expediente de manera automática al médico que desee.

La carencia de este sistema hace que para la transferencia de expedientes sea necesario llevar a cabo varios trámites burocráticos que llevan tiempo (generalmente hay que ir en persona) para asegurar que el paciente es quien dice ser.

En este proyecto se presenta e implementa el diseño de una aplicación para llevar a cabo el seguimiento de expedientes médicos mediante identidad digital en *Blockchain*, que dará solución a este problema.

2 Descripción de las Tecnologías

2.1 Atom

Atom es un editor de texto pensado especialmente para el desarrollo de aplicaciones. Permite integración de HTML, CSS, JavaScript y Node.js Consta de paquetes desarrollados por la comunidad (open source) que permiten añadir funcionalidades como el resaltado de otros lenguajes no incluidos en la aplicación original.

Posee una interfaz muy limpia e intuitiva, lo que permite trabajar con comodidad y velocidad. Además, se trata de una aplicación gratuita.

Para el desarrollo de este proyecto se han utilizado paquetes para el lenguaje Solidity y para integrar GitHub en Atom.

2.2 Solidity

Lenguaje utilizado para desarrollar los SmartContracts en redes Ethereum (también se usa en otras redes como Binance Smart Chain, pero Ethereum es la más importante). Se trata de un lenguaje desarrollado por el equipo de Ethereum liderado por Christian Reitwiessner a partir de 2014, por lo que está pensado para ser ejecutado en la Ethereum Virtual Machine (EVM). Este lenguaje es orientado a objetos y de tipado estático.

2.3 Web3js

Librerías JavaScript utilizadas para conectarse a una red BlockChain Ethereum. Estas librearías permiten utilizar HTTP, IPC o WebSocket para la conexión con los nodos y poseen una amplia documentación, lo que las hace muy útiles para el desarrollo de aplicaciones BlockChain. Tanto el uso como la descarga de estas librerías es completamente gratuito.

2.4 React

Librerías JavaScript para desarrollo de aplicaciones web. Estas librerías están pensadas para facilitar el proceso de creación de la interfaz de usuario. Se basan en el uso de componentes, lo que ayuda a reciclar código y simplificar y agilizar el desarrollo.

Tanto el uso como la descarga de estas librerías es completamente gratuito.

2.5 Truffle

Truffle es un framework para el desarrollo de aplicaciones en Ethereum. Permite llevar un control sobre el ciclo de vida de los SmartContracts, así como crear un entorno de desarrollo con capacidad de test automatizado. Truffle posee también una consola interactiva muy potente, aunque en este proyecto no será necesario su uso. Este framework es muy sencillo de instalar y es completamente gratuito.

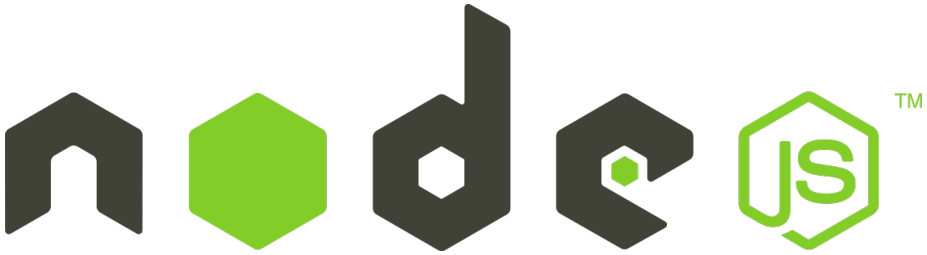
2.6 Ganache

Ganache simula de manera local una red BlockChain Ethereum. Ganache es una herramienta imprescindible para desarrollar aplicaciones BlockChain en lenguaje Solidity para la red Ethereum, pues permite realizar todo tipo de pruebas sobre la red BlockChain sin tener que pagar (en la red principal Ethereum cada transacción tiene un precio) puesto que todo consumo de GAS es simulado. Igualmente, los bloques se minan automáticamente de manera instantánea. Igual que Truffle (pertenece a los mismos desarrolladores) es gratuito.

2.7 Mocha

Framework de test de JavaScript ejecutado en Node.js. Se trata de un framework gratuito y de fácil manejo que permite realizar tests tanto síncronos como asíncronos para las aplicaciones con Smart Contracts. Esta herramienta es gratuita.

2.8 Node.js



Se trata de un script en JavaScript dirigido por eventos en tiempo de ejecución. En este proyecto será utilizado para llevar a cabo el despliegue de la aplicación.

Es gratuito y sencillo de utilizar.

2.9 NPM

Gestor de archivos de Node.js. Utilizado para llevar a cabo el despliegue de los archivos necesarios para la aplicación. Permite instalar librerías necesarias para el desarrollo, así como ejecutar la aplicación mediante simples comandos. Es también gratuito y fácil de utilizar.

2.10 MetaMask

Extensión del navegador que permite la conexión con la red Ethereum. También sirve para gestionar la cartera con las distintas cuentas y saldos. Es una extensión gratuita.

2.11 IPFS Desktop

Cliente para gestionar la conexión y el tráfico a la red IPFS (*InterPlanetary File System*). Se trata de una aplicación gratuita.

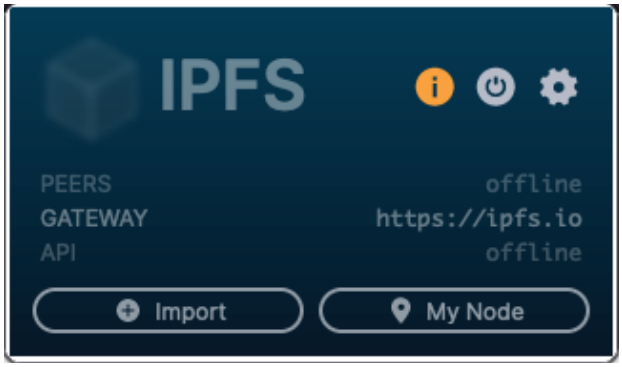
IPFS es un protocolo de hipermedia peer-to-peer que promueve una red resiliente y descentralizada, buscando mejorar la actual internet.

2.12 Go-IPFS

Implementación gratuita que simula de manera local un nodo IPFS.

2.13 IPFS Companion

Extensión *opensource* del navegador que permite la conexión con la red IPFS. Muy sencilla de usar y gratuita.



3 Motivación

Utilizando la tecnología *Blockchain* y el concepto de identidad digital se pretende asegurar la seguridad, confidencialidad e integridad a la hora de transferir expedientes entre médicos, hospitales y pacientes a través de internet, eliminando trámites burocráticos y agilizando el proceso.

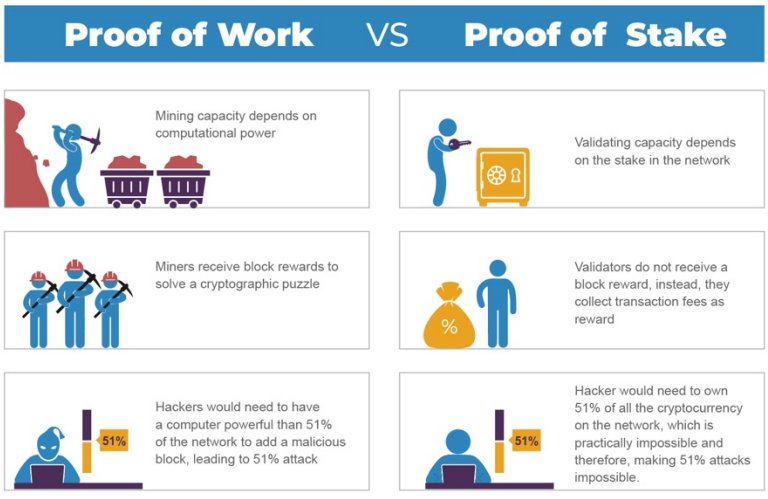
Esto ayudará a pacientes que quieran transferir sus expedientes a su médico privado o a otro médico u hospital en el que desee ser atendido, aumentando su autonomía a la hora de decidir dónde ser tratado o por quién. Además, ayudará a mantener un mayor control sobre los pacientes al poder acceder a los expedientes de manera más rápida y sencilla, asegurando la confidencialidad y la seguridad en el acceso a dichos expedientes.

4 Estado del Arte

Blockchain se trata de un sistema descentralizado compuesto por nodos independientes *peer to peer* (P2P) que gestiona un registro único. Es una tecnología basada en una lista de estructuras de datos – llamadas bloques – en la que cada bloque contiene un *hash* del bloque anterior, un *timestamp* y datos de transacción, generalmente como un árbol de Merkle.

Esta tecnología se desarrolló en 2008 por Satoshi Nakamoto (se desconoce si es una única persona o un pseudónimo utilizado por un grupo de personas para mantener la creación de *Blockchain* en el anonimato) para ser utilizada como libro de cuentas de la criptomoneda *bitcoin*. Con el uso de *Blockchain* se resolvía el problema del doble gasto, presente en las transacciones virtuales hasta la fecha, sin necesidad de utilizar una entidad acreditadora de confianza o un servidor central.

La tecnología Blockchain 2.0 es la evolución del desarrollo de Satoshi Nakamoto expandiendo las aplicaciones del ámbito de las criptomonedas a aplicaciones de negocio [1]. El mayor problema que apareció aquí fue la escalabilidad. Hoy en día, con la Blockchain 3.0, estos problemas de escalabilidad han sido resueltos, añadiendo además mejoras de velocidad y eficiencia con tecnologías como *Directed Acyclic Graph* (DAG), que posibilita que las transacciones se procesen casi en tiempo real.



El mecanismo de consenso utilizado en este momento es el llamado *Proof of Work (PoW)*, basado en que los verificadores de la *blockchain* dan por buena una transacción porque una de las partes ha "gastado" una cantidad específica de computación. Este mecanismo está en vías de ser reemplezado (e.g. finales de 2021 en *Ethereum*, aunque atrasado esta fecha se ha visto atrasada) por el *Proof of Stake (PoS)*, dado que *PoW* incentiva a consumir grandes cantidades de energía para verificar cada bloque. Sin embargo, *PoS* selecciona los validadores en función de la cantidad de participaciones que tengan. Dependiendo de la implementación la definición de esta participación puede variar desde simplemente la cantidad de criptomonedas hasta conceptos como la edad de la moneda, que tiene en cuenta cuánto tiempo se han poseído dichas criptomonedas. Otra práctica es la *Delegated PoS* (*DPoS*), según la cual los *stake-holders* pueden delegar el rol de validador en otros.

Actualmente, la tecnología *Blockchain* en el ámbito de la sanidad es utilizada para agilizar pagos entre aseguradoras, clientes y servicios médicos (China) y para llevar a cabo un seguimiento de las personas con anticuerpos o posibles inmunidades a la COVID-19 (USA) [2].

La identidad digital se define como el conjunto de datos y acciones que realiza un individuo en el ciberespacio. Así, mediante la identidad digital se pueden saber los comportamientos, intereses o reputación de una persona, al igual que se pueden conocer sus datos personales como nombre, edad, sexo o incluso dirección.

Para asegurar que un individuo es quien dice ser, generalmente es un organismo el que acredita esta identidad. El ejemplo más claro es el de un estado asegurando, mediante un pasaporte o algún otro documento oficial que una persona es, en realidad, quien dice ser. En el ámbito digital, esta "confianza" la proveen terceras partes que se presuponen de confianza. Un ejemplo de esta práctica son los certificados de clave pública, en los que un sujeto firma de manera digital con el certificado expedido por un proveedor – *Certificate Authority* (CA) – que acredita que ese sujeto es quien dice ser. La comprobación de esa identidad no reside en que el certificado en sí lo acredite, sino en la confianza depositada en el CA para asegurarlo.

En el artículo [3], se explica un modelo para la gestión descentralizada mediante *Blockchain* de la identidad digital. En este proyecto no se utilizará ese modelo, pero sí algunos conceptos

expuestos en él. La identidad digital se verificará mediante una tercera parte (e.g. el Estado), asegurando así que tanto el paciente que desea compartir su expediente, como el doctor o clínica correspondiente, son quienes dicen ser.

En la actualidad, en España la gestión de la Sanidad está transferida a las distintas comunidades autónomas, aunque bajo la coordinación del Estado. En el caso, por ejemplo, de la Comunidad de Madrid, existe una página web [4] en la que se informa sobre cómo acceder al expediente médico digital mediante un certificado digital o con el DNI electrónico. Pero este sistema tiene el problema de que solo da acceso al expediente almacenado en el sistema del Servicio Madrileño de Salud o en el Sistema Nacional de Salud. No es posible acceder ni a expedientes de otras comunidades o centros no madrileños (si no está el expediente en el Sistema Nacional de Salud) o a expedientes de clínicas del sector privado. Aparte de esta problemática de acceso, no todos los informes están disponibles actualmente, ya que aún están en proceso de digitalización (aunque la mayoría como los de atención primaria, urgencias, pruebas de laboratorio o imagen sí lo están).

Gracias a este sistema, es posible intercambiar información entre el sistema público y un usuario particular de manera segura (la identidad del usuario se verifica mediante el DNI electrónico o el certificado digital) pero el usuario no tiene manera de transferir estos documentos mediante una plataforma que verifique su identidad a una clínica o a un doctor privados. El usuario podría imprimir los documentos y llevarlos personalmente o podría enviarlos por correo.

Hoy en día, el único sistema unificado a nivel nacional en España en el ámbito de sanidad es la Organicación Nacional de Trasplantes (ONT). Este organismo se encarga de la coordinación a nivel nacional, autonómico y por hospital de las actividades de donación, extracción, preservación, distribución, intercambio y trasplante de órganos, tejidos y células en el Sistema Sanitario Español.

Existen varios *softwares* de gestión de expedientes en *cloud*, aunque no permiten la transferencia de éstos entre sistemas distintos. Es decir, actualmente en España no existe una plataforma que permita realizar la transferencia de documentos médicos entre entornos diferentes de manera transparente. Algunos de estos *softwares* son MedicosPro, DriCloud,

ClinicCloud. Estas plataformas están más centradas en el día a día de una consulta (recetas, citas, facturación etc.)

5 Objetivos

Para realizar la transferencia de expedientes médicos asegurando su integridad, confidencialidad y seguridad entre sistemas heterogéneos mediante *Blockchain,* se pretende alcanzar los siguientes objetivos:

- Uso de *Blockchain* para la gestión de expedientes médicos aplicando *Smart Contracts* y su acceso a través de identidades digitales
- Desarrollo de una aplicación web que implemente esta solución.
- Solucionar la problemática del uso compartido de expedientes entre distintas administraciones

6 Metodología de trabajo

La metodología de desarrollo Agile está basada en una serie de prácticas para asegurar la mejor calidad del software con una gran eficiencia. En Agile es importante que tanto desarrolladores como clientes estén en continuo contacto durante todo el proceso de desarrollo. De esta manera, las fases de planificación, desarrollo y entrega en Agile se llevan a cabo de manera casi simultánea: se hace una planificación global y se van realizando pequeñas entregas completamente funcionales de manera periódica para poder realizar mejoras sobre el desarrollo de manera flexible. Los principales valores de Agile son:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

Individuos e interacciones sobre procesos y herramientas
Software funcionando sobre documentación extensiva
Colaboración con el cliente sobre negociación contractual
Respuesta ante el cambio sobre seguir un plan

Aunque los elementos de la derecha sean importantes, lo son más lo elementos de la izquierda [5].

Existen varias técnicas derivadas de *Agile*, como son *scrum* o *kanban*.

La metodología a seguir en este proyecto será una metodología *Agile*, concretamente *scrum*, realizando pequeñas entregas periódicas, en cada *sprint*, que aporten valor hasta tener el producto final. Para el seguimiento del trabajo se utilizarán tablones *scrum* para las distintas tareas entregables y *kanban* para el seguimiento global del proyecto.

Se desarrollará una lógica utilizando *Blockchain* y una aplicación web que servirá como interfaz para que los usuarios puedan transferir los expedientes médicos. En definitiva, se desarrollará lo conocido como una DApp (*Decentralized Application* – Aplicación Descentralizada).

7 Cronograma

ACTIVIDADES	E	F	M	A	M	J
Búsqueda de Información						
Preparación entorno de desarrollo						
Diseño						
Desarrollo *SmartContracts*						
Desarrollo de la Aplicación						
Pruebas y Validación						
Evaluación de los Resultados						
Memoria del Proyecto						

8 Estimación económica

A continuación se muestra la estimación económica del desarrollo. Para la puesta en producción, existen servicios gratuitos o con un coste reducido a partir de una cantidad de tráfico en AWS, Azure o Google Cloud. No se ha incluido dicho coste de producción en esta estimación.

Costes directos				13845 €
Mano de obra	30 €/hora	450 horas		13500 €
Ordenador	2300 €	Uso: 9 meses	Amortización: 5 años	345 €
Costes indirectos	15 % directos			2076.75 €
Beneficio industrial	6 % directos + indirectos			955.31 €
Subtotal				16877.06 €
IVA	21 %			3544.19 €
Total				20421.25 €

Atendiendo a este presupuesto, el coste principal es el del tiempo y conocimiento invertido en el desarrollo de la solución, ya que no se ha requerido de ninguna inversión importante más que el ordenador en el que se ha desarrollado dicha solución.

A continuación, se muestran varios escenarios de puesta en producción, incluyendo beneficios esperados tanto de la aplicación como de soporte. Los costes de los posibles nuevos empleados que hubiera que contratar para dar el servicio de soporte no están incluidos. En el primer escenario el producto ofrecido es la aplicación en sí.

Clínica

Concepto	**Precio**
DApp	15,000 €
Soporte	25,000 €/año

Hospital

Concepto	**Precio**
DApp	15,000 €
Soporte	50,000 €/año

Red de hospitales

Concepto	**Precio**
DApp	15,000 €
Soporte	45,000 €/hospital, año

En el segundo escenario el producto ofrecido es la aplicación en sí, más la infraestructura necesaria en un CPD propio. De esta manera, en el coste de la aplicación se incluyen los gastos de servidor.

Clínica (máx. 20 médicos y 1500 pacientes)

Concepto	**Precio**
DApp	20,000 €
Soporte	25,000 €/año
Mantenimiento	10,000 €

Hospital

Concepto	**Precio**
DApp	30,000 €
Soporte	50,000 €/año
Mantenimiento	17,000 €

Red de hospitales

Concepto	**Precio**
DApp	20,000 € + 5,000 €/hospital
Soporte	45,000 €/hospital, año
Mantenimiento	15,000 €

En el tercer escenario el producto ofrecido es la aplicación en sí, más la implantación de la aplicación en un servidor *cloud* como AWS, Azure o Google Cloud.

Clínica (máx. 20 médicos y 1500 pacientes)

Concepto	**Precio**
DApp	20,000 €
Soporte	25,000 €/año
Servidor	1,000 €/año

Hospital

Concepto	**Precio**
DApp	15,000 €
Soporte	50,000 €/año
Mantenimiento	1,500 €/año

Red de hospitales

Concepto	**Precio**
DApp	15,000 €
Soporte	45,000 €/hospital, año
Mantenimiento	1,500 €/hospital, año

9 Desarrollo de la solución

9.1 Aplicación web

La propuesta implementada en este proyecto ha sido una DApp que permite a los distintos usuarios (hospitales, médicos y pacientes) identificarse y consultar o compartir uno o varios expedientes. De esta manera, en la aplicación habrá tres roles diferenciados con distintos permisos:

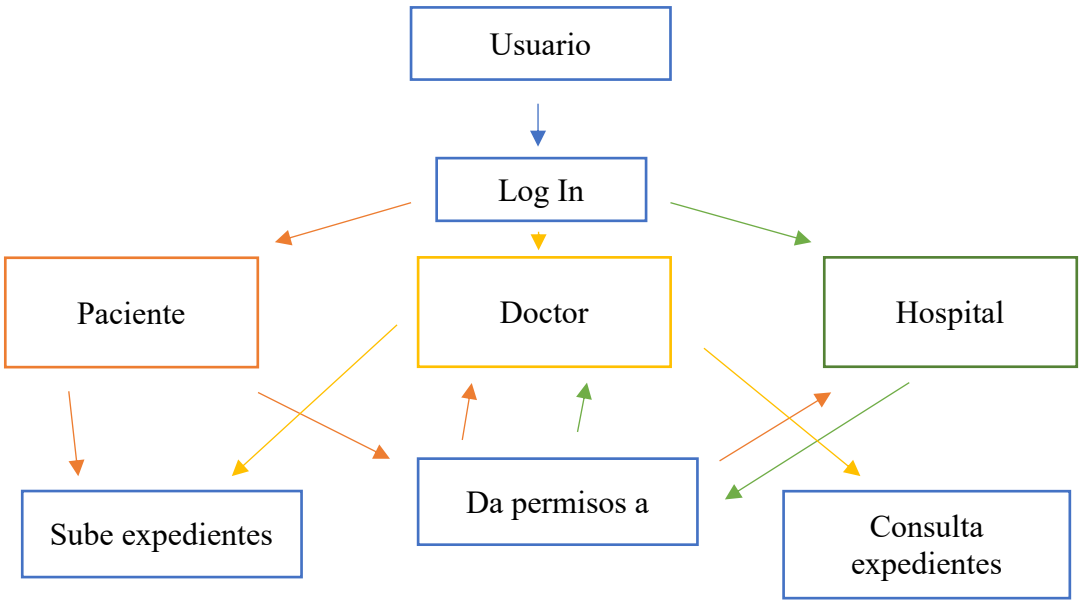
Paciente: puede subir su expediente y compartirlo con los otros dos roles (dando permiso).

Médico: capaz de ver los expedientes compartidos con él y de subir expedientes (actualizar uno o crear uno nuevo a un paciente) de pacientes que le hayan autorizado.

Hospitales: tienen permiso para ver los expedientes compartidos con ellos y pueden compartirlos con un médico (el expediente se comparte con el hospital y es el hospital el que le asigna un médico al paciente).

Todo usuario de la aplicación deberá identificarse (login) para poder tener acceso a las funcionalidades permitidas para su rol.

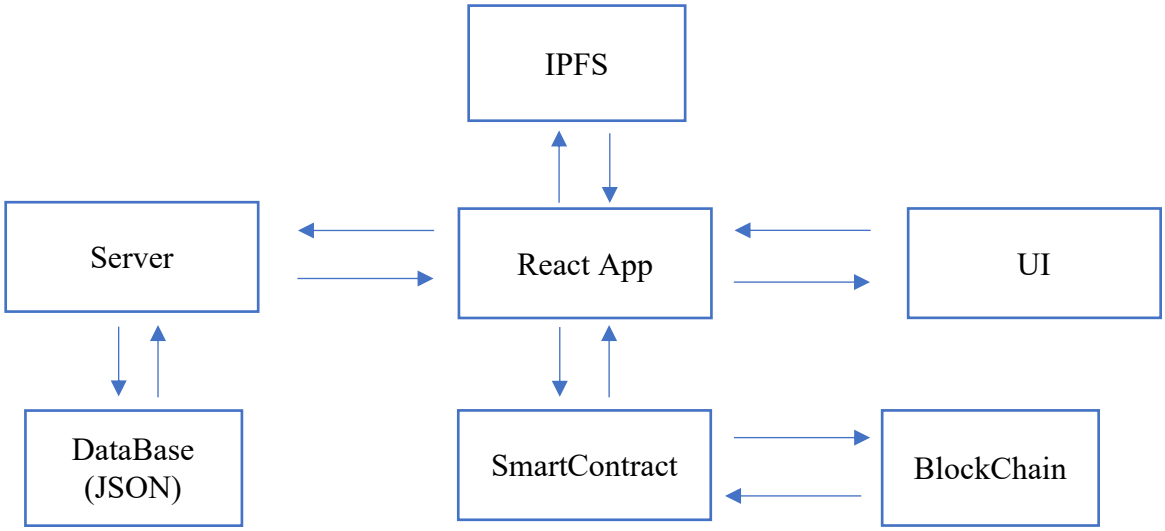
9.2 Diagramas de flujo



9.3 Smart contracts

Para esta DApp se ha desarrollado únicamente el *smart contract* UploadFile.sol, en el que se implementa la funcionalidad de guardar el *hash* del expediente subido y el DNI del paciente a la *blockchain*. El expediente en sí será almacenado en la red IPFS y el identificador del fichero en esta red es el *hash* que se guardará en la *blockchain*. El DNI será el del dueño del expediente y la gestión de los permisos de subida y vista de los expedientes se gestionarán desde la aplicación.

9.4 Arquitectura



El acceso se lleva a cabo mediante una API Rest que actúa a modo de servidor. Al realizar el login, el servidor devuelve toda la información del usuario mediante un *fetch* y esta información se guarda como una *cookie* de sesión y como un estado de la aplicación React. Este estado es el que se actualizará al dar permisos y subir expedientes.

El servidor consulta la lista de usuarios de un JSON que hace la función de base de datos. Esta implementación no es óptima, pero se ha realizado de esta manera por motivos prácticos.

El servidor tiene también la función de actualizar esta base de datos cuando se haya subido un expediente o se hayan cambiado permisos de visualización de un expediente ya subido.

Si el usuario lo que desea es visualizar un expediente, podrá consultarlo directamente sin descargarlo, pues el *hash* de dicho expediente permite realizar un link a la *url* del expediente subido a la red IPFS. La red IPFS almacena de manera pública un fichero. Para asegurar la confidencialidad de los ficheros subidos, los ficheros deberían encriptarse al subirse, y se debería facilitar al usuario que lo tenga que consultar una clave. Otra solución sería utilizar una red privada IPFS, además de la encriptación para aumentar la seguridad. En esta solución no se ha llevado a cabo.

La transferencia de expedientes se realiza dando permiso al médico o al hospital para poder ver dicho expediente. El expediente en sí no es transferido, pues está guardado en la red IPFS.

La tecnología IPFS funciona dividiendo el fichero subido a la red IPFS en pequeños trozos, con un *hash*, a los que se les asigna un CID *(Content IDentifier)*. Este CID es un registro del fichero en el momento de su subida a la red. Cuando un nodo de la red busca el fichero, pregunta a los demás nodos quién tiene el fichero al que referencia ese CID. Al descargar o ver el fichero, ese nodo guarda en la caché una copia y se convierte en un nuevo proveedor de ese fichero, hasta que se borre ese fichero de la caché.

Un nodo puede marcar uno de los contenidos que provee para tenerlo siempre guardado y ser así su proveedor, o puede eliminarlo para dejar de serlo y liberar espacio. Las nuevas versiones de los ficheros subidos, al tener distinto *hash*, reciben también un nuevo CID. De esta manera, los ficheros subidos se mantienen inmutables. Los trozos en los que se dividen los ficheros al subirlos, si son compartidos entre versiones, pueden ser reutilizados al subir uno nuevo, ahorrando en términos de almacenamiento. Para evitar que el control de versiones necesite de una lista extensa de *hashes*, IPFS utiliza IPNS *(InterPlanetary Name System)*, que se trata de un sistema de nombrado descentralizado para que nuevas versiones del mismo fichero puedan ser consultadas desde un mismo enlace. También se puede utilizar DNSLink para mapear CIDs con nombres DNS.

La versatilidad que aporta IPFS viene dada también por la sencilla incorporación a la DApp, ya que con descargar el paquete con NPM de `ipfs-api` se instala una API a la que se le pasa el fichero y devuelve el CID, tras haberlo subido a la red.

La aplicación se ha desarrollado utilizando *React*, unas librerías que permiten desarrollar interfaces de usuario de manera sencilla y pudiendo reutilizar gran parte del código.

En la aplicación React se implementan las distintas funcionalidades y casos de uso de la solución:

Registro/acceso

El usuario al acceder a la aplicación tendrá la posibilidad de crear un nuevo usuario o acceder utilizando uno ya creado. Los campos requeridos para crear un usuario son el nombre, el identificador – en el caso del paciente será el DNI, en el de los médicos será su número de colegiado y en el del hospital será el nombre del hospital – y la contraseña.

Subir expedientes

El usuario podrá subir expedientes médicos a la red IPFS a través de la aplicación React. El paciente podrá subir sus propios expedientes y tanto médicos como hospitales que tengan permisos dados por los pacientes podrán subir expedientes de dichos pacientes. Para esto se utiliza la API de IPFS. Tras subir el expediente a la red IPFS se guarda el *hash* devuelto y el DNI del paciente al que corresponde en la *blockchain,* utilizando el *Smart Contract*.

Dar permisos

Los usuarios darán permiso a médicos y hospitales introduciendo el identificador de dicho médico, o el identificador del hospital. La aplicación se encarga de guardar ese ID en la base de datos (JSON) dentro del propio usuario que da el permiso, en el campo *permits*. Esto se lleva a cabo mandado los identificadores introducidos por el usuario al servidor, que se encarga de procesar los datos y actualizar la base de datos. Los hospitales tendrán guardados aquí los médicos que tengan en plantilla, y los pacientes tendrán guardados tanto los médicos como los hospitales a los que deseen dar permisos. Los médicos deberían tener este campo vacío *(null)*.

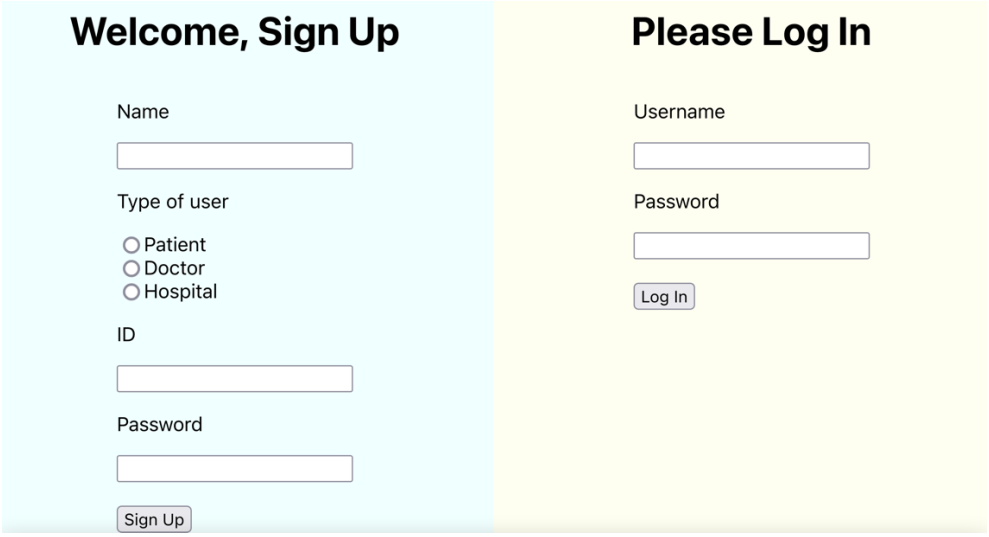
Consultar expedientes

Los usuarios podrán consultar los expedientes que hayan subido, en el caso de los pacientes, así como los expedientes de los pacientes de los que tengan permisos, en el caso de los médicos. Para consultar los expedientes el médico introduce el DNI del paciente y la aplicación, tras comprobar que el campo no está vacío, manda el DNI al servidor. El servidor comprueba si el que realiza la consulta tiene los permisos pertinentes y, en caso afirmativo, devuelve a la aplicación el campo *files* del paciente en concreto.

9.5 Interfaz de usuario

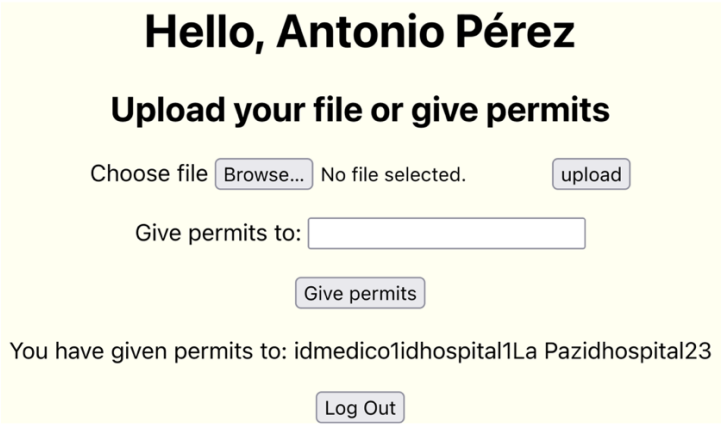
La interfaz elegida ha sido una interfaz simple y concisa, que integra todas las funcionalidades de cada usuario.

Registro/acceso



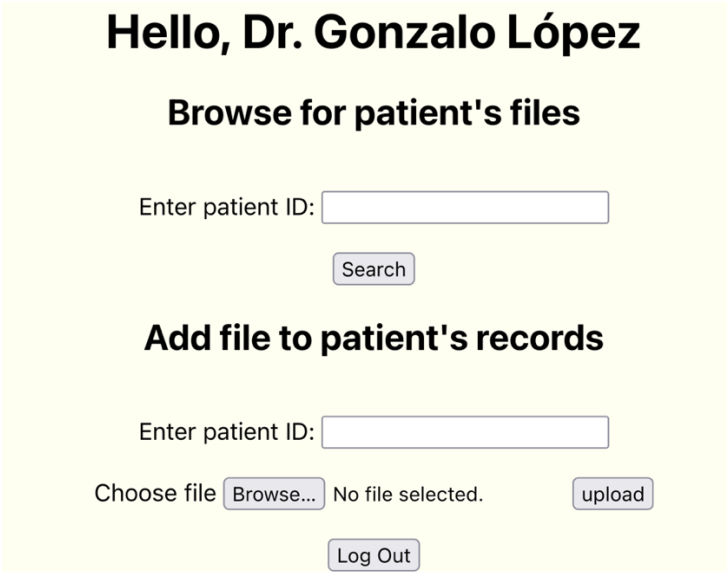
En esta pantalla se encuentran tanto el registro como el acceso de un usuario ya registrado. El registro se realiza seleccionando el propio usuario su rol.

Paciente



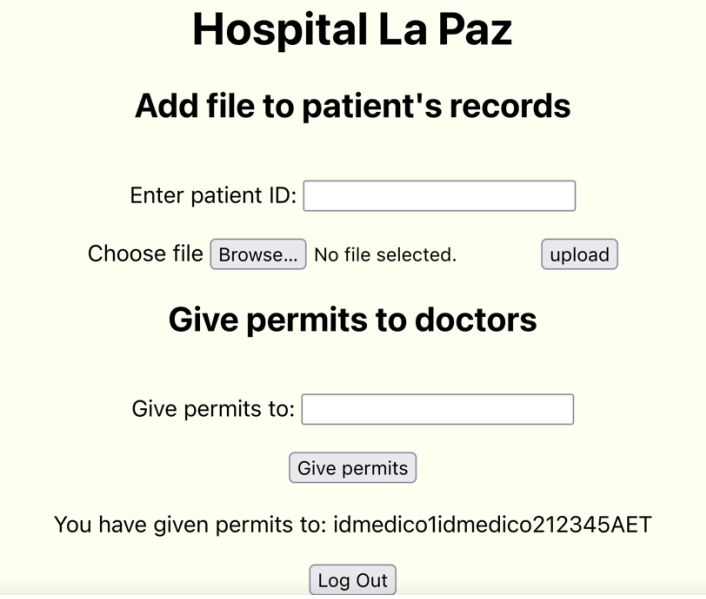
En la vista del paciente, el usuario puede subir un expediente a la plataforma, así como dar permisos a los médicos u hospitales que el paciente desee.

Médico



Desde la vista del médico se puede buscar un paciente para ver su expediente, siempre y cuando se tengan los permisos necesarios. Además, si el médico tiene permiso para ver, también puede subir un expediente a la plataforma en nombre del paciente.

Hospital



Finalmente, la vista del hospital permite subir expedientes en nombre de un paciente que haya dado permisos al hospital o dar permisos a doctores para que puedan acceder a los expedientes a los que tiene acceso el propio hospital.

10 Conclusiones

La tecnología Blockchain tiene un amplio futuro en el ámbito sanitario. La DApp desarrollada en este proyecto es una pequeña parte de lo que puede significar esta tecnología, descentralizando y agilizando trámites burocráticos, así como asegurando la confidencialidad, seguridad, integridad y trazabilidad de los mismos.

La DApp desarrollada es un prototipo funcional que permite la transferencia de expedientes médicos, pero tiene aún muchas mejoras posibles, desde la interfaz hasta la propia lógica de la aplicación, haciéndola más segura y robusta.

En la DApp entregada no se aplica la identidad digital con Blockchain, toda la identidad digital se lleva a cabo con el fichero JSON (base de datos), pero la capacidad de Blockchain para asegurar la identidad digital es mucho más potente debido a la inmutabilidad de la Blockchain. Una posible solución sería la creación de una base de datos en la que se relacionen direcciones de *wallet* con documentos como el DNI. Esta base de datos debería ser gestionada y acreditada por una entidad como el Gobierno de España y tendría una funcionalidad similar al actual DNI electrónico, pero con identificación de Blockchain.

Desde el punto de vista de la confidencialidad, el uso de la tecnología conlleva un problema, debido precisamente a una de sus virtudes, en lo referido a integridad: la inmutabilidad. La confidencialidad se puede conseguir cifrando cualquier tipo de información que se guarde en la Blockchain, pero este cifrado puede no ser seguro con el paso del tiempo; los algoritmos de cifrado se van quedando obsoletos y se vuelven descifrables según la capacidad de computación aumenta, así como con el desarrollo de nuevas técnicas para revertir dichos algoritmos. Una solución a este problema podría ser manejar Blockchain privadas, gestionando su acceso mediante algoritmos y aplicaciones que puedan ser actualizadas en sus mecanismos de cifrado. Otra opción es no almacenar nada de información sensible en la Blockchain, sino referencias a la información en sí; por ejemplo, en el caso de la DApp desarrollada, el expediente no se almacena en la Blockchain, sino que se envía a IPFS y lo que se almacena es el *hash* del expediente en cuestión, que es lo que utiliza también IPFS para luego mostrar el fichero almacenado. De esta manera, la información no es inmutable pero sí la referencia de

dónde se encuentra dicha información. Esta idea de almacenamiento es la usada también por la tecnología *Non Fungible Token (NFT),* en la que la Blockchain almacena las referencias a los activos. Así, mientras todo lo almacenado en la Blockchain es visible a todos los usuarios, al ser solo referencias esa información solo es accesible y descifrable para quien tenga el permiso, aunque los algoritmos cambien y el cifrado original de la información almacenada se quede obsoleto. Solventado este problema, la manera de comprometer la información reside en conseguir acceso al lugar donde se almacena la información y cambiarla, algo que es un problema que ya existe con las tecnologías de hoy en día.

11 Mejoras y trabajos futuros

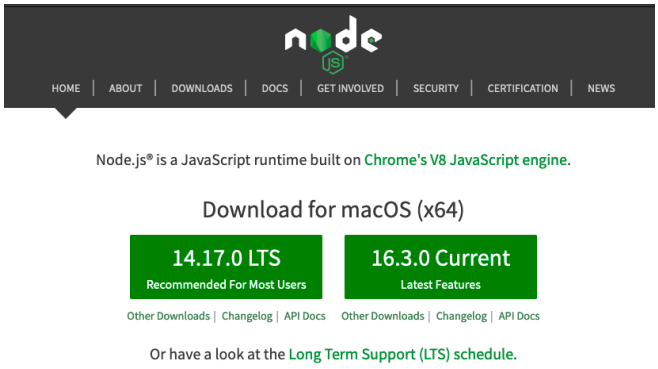
Como posibles mejoras y trabajos futuros los principales serían los siguientes:

- Integrar una base de datos: actualmente se ha utilizado un fichero JSON para simular una base de datos de usuarios. Sería conveniente utilizar una base de datos como MySQL.
- Integrar la identidad digital con Blockchain: en lugar de usar la base datos para dar permisos y para llevar un registro de los expedientes subidos, utilizar la Blockchain para ello.
- Aumentar la seguridad: en la DApp no se utilizan métodos de encriptado para los datos de los usuarios. Tampoco se utilizan filtros para evitar ataques similares a SQL Injection. Toda la información del usuario que ha iniciado sesión se guarda como *cookie* de sesión, lo cual puede suponer una falla de seguridad. Por último, IPFS no encripta los ficheros subidos, por lo que el expediente subido es público. Para asegurar la confidencialidad del expediente se puede encriptar el expediente o utilizar una red IPFS privada.
- Uso de Blockchain privada: utilizar otras Blockchains como Hyperledger Fabric, para aumentar la confidencialidad al gestionar quién tiene acceso a la Blockchain.

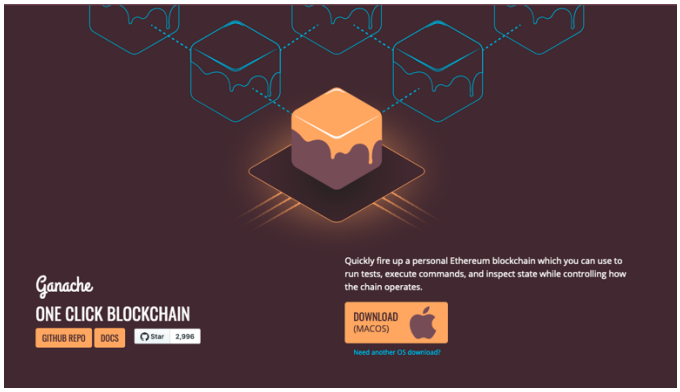
12 Anexo: Instalación del entorno de desarrollo

Para la instalación del software utilizado hay que seguir los siguientes pasos:

- Instalación de Node.js: se descarga desde la página web de node.

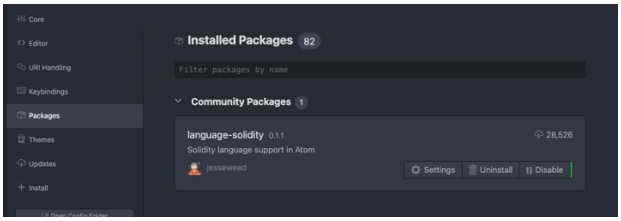


- Instalación NPM: mediante línea de comandos, se introduce `node install npm -g`.
- Instalación Truffle: mediante línea de comandos, se introduce `npm install -g truffle`.
- Instalación Ganache: se descarga desde la página web de Ganache.

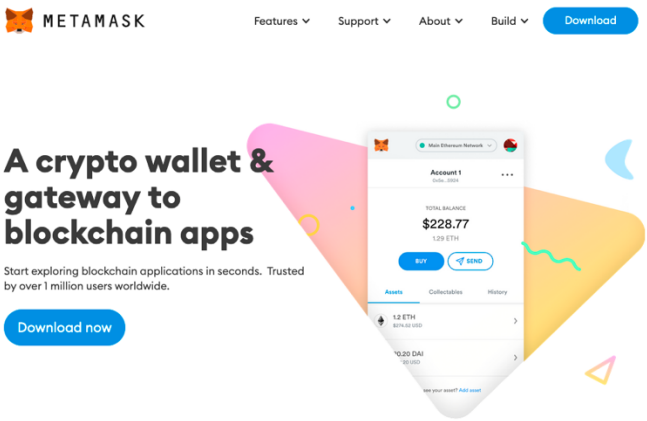


- Instalación Mocha: mediante línea de comandos, se introduce `node install mocha -g`.
- Instalación React: mediante línea de comandos, se introduce `node install react -g` y `node install react react-home -g`.
- Instalación Atom: se descarga desde la página web de Atom.

Una vez descargado, se pueden utilizar Packages para que la tarea de desarrollo del código sea más sencilla, como resaltadores del texto para el lenguaje solidity o incluso un compilador del lenguaje. En este proyecto se utilizó language-solidity.

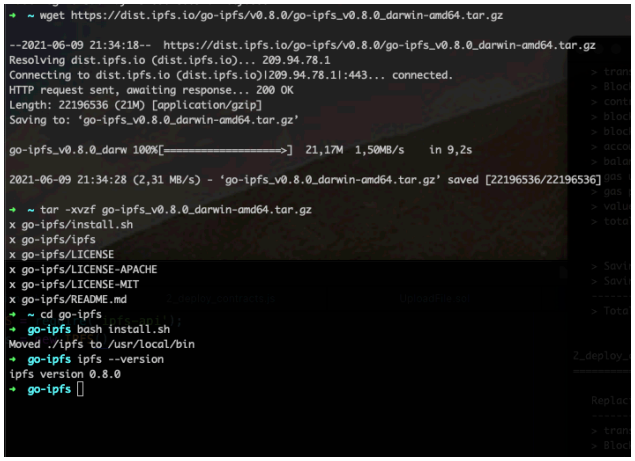


- Una vez instalado todo lo necesario, mediante el comando `truffle unbox react` se descarga un proyecto que servirá de base para el desarrollo de la aplicación.
- Metamask: se descarga desde la página web de Metamask la extensión.

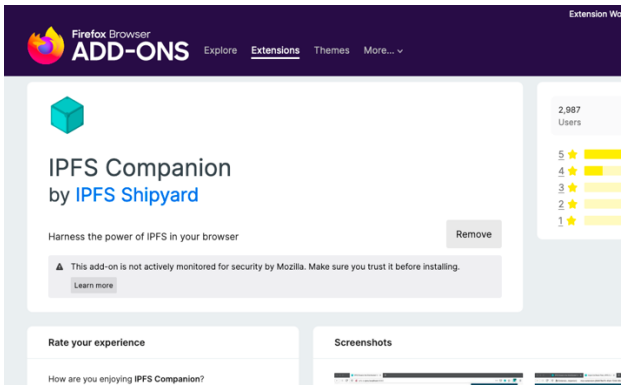


- Go-IPFS: siguiendo la siguiente secuencia de comandos

```
wget https://dist.ipfs.io/go-ipfs/v0.8.0/go-ipfs_v0.8.0_darwin-amd64.tar.gz
tar -xvzf go-ipfs_v0.8.0_darwin-amd64.tar.gz
cd go-ipfs
bash install.sh
```

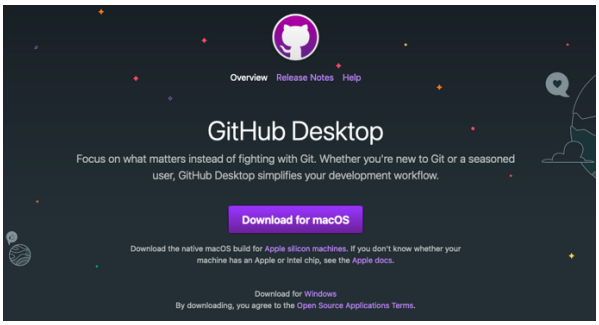


- IPFS Companion: se descarga desde los add-ons del navegador.



- IPFS Desktop: se descarga desde el companion.

- Para este proyecto se ha utilizado GitHub como repositorio de código, aunque no es estrictamente necesario para el desarrollo. Para instalar GitHub Desktop se descarga desde la página web de GitHub Desktop.



13 Anexo: Objetivos de Desarrollo Sostenible

Este proyecto está enfocado a facilitar una manera descentralizada y segura de compartir expedientes médicos, lo que proporciona un mejor y más ágil acceso a la sanidad para todos los individuos. Es por esto que este proyecto se alinea con el ODS 3: Salud y Bienestar. Concretamente, ayudará al cumplimiento de los puntos 3.d y 3.8, que se refieren a la cobertura sanitaria universal y a la alerta temprana y reducción de riesgos y gestión, respectivamente.

Igualmente, al contribuir al acceso a la sanidad de manera universal, también se alinea con el ODS 10: Reducción de las Desigualdades.

El uso de la tecnología Blockchain en su variante *PoW* no favorece al cambio climático ni al gasto sostenible de energía, por lo que en ese sentido este proyecto no fomenta algunos de los ODS, pero cuando esta tecnología se convierta a *PoS* como se espera en un futuro, será una tecnología sostenible que promoverá la innovación y el uso sostenible de la energía, estando de esta manera alineado con los ODS 9: Industria, Innovación e Infraestructura, 12: Producción y Consumo Responsables y 13: Acción por el Clima. En los ODS 12 y 13, concretamente se alinean con lo referido a energías.

14 Anexo: Código

En este anexo se incluye el código de la DApp con breves explicaciones de cada módulo.

Upload.sol

El SmartContract encargado de guardar el *hash* del expediente en la Blockchain.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.21;

contract UploadFile {
  string ipfsHash;
  string id;

  function set(string memory x, string memory y) public {
    ipfsHash = x;
    id = y;
  }

  function get() public view returns (string memory, string memory) {
    return (ipfsHash,id);
  }
}
```

App.js

Esta es la aplicación web, utilizando React. Contiene las funciones necesarias para llevar a cabo las funcionalidades de dar permisos, subir un expediente o consultarlo. En la función `render()` se encuentra el código HTML de la página web.

```
import React, { Component } from "react";
import UploadFileContract from "./contracts/UploadFile.json";
import getWeb3 from "./getWeb3";
import ipfs from "./ipfs";
import Login from './Login';
import SignUp from './SignUp';


import "./App.css";

class App extends Component {
  state = { storedHash: null, web3: null, accounts: null, contract:
null, buffer: null};

  constructor(props) {
    super(props);

    this.captureFile = this.captureFile.bind(this);
    this.killSession = this.killSession.bind(this);
    this.capturePermits = this.capturePermits.bind(this);
    this.lookFiles = this.lookFiles.bind(this);
    this.onSubmit = this.onSubmit.bind(this);
    this.setToken = this.setToken.bind(this);
  }

  componentDidMount = async () => {
    try {
      // Get network provider and web3 instance.
      const web3 = await getWeb3();

      // Use web3 to get the user's accounts.
      const accounts = await web3.eth.getAccounts();

      // Get the contract instance.
      const networkId = await web3.eth.net.getId();
      const deployedNetwork = UploadFileContract.networks[networkId];
      const instance = new web3.eth.Contract(
        UploadFileContract.abi,
        deployedNetwork && deployedNetwork.address,
      );

      // Set web3, accounts, and contract to the state, and then
proceed with an
```

```
      // example of interacting with the contract's methods.
      this.setState({ web3, accounts, contract: instance });
    } catch (error) {
      // Catch any errors for any of the above operations.
      alert(
        'Failed to load web3, accounts, or contract. Check console
for details.',
      );
      console.error(error);
    }
  };

  captureFile(e){
    //console.log('file captured');
    e.preventDefault();
    const file = e.target.files[0];
    const reader = new window.FileReader();
    reader.readAsArrayBuffer(file);
    reader.onloadend = () => {
      this.setState({buffer: Buffer(reader.result)});
      //console.log('buffer',this.state.buffer);
    }
  }

  async capturePermits(e){
    e.preventDefault();
    const userToken = this.getToken()
    //userToken
    //console.log('prepermit',userToken.user.permits)
    if(e.target.permits.value!=='' &&
!userToken.user.permits.includes(e.target.permits.value)){
      userToken.user.permits.push(e.target.permits.value)
      //console.log('postpermit',userToken.user.permits)
      this.setToken(userToken)
      this.fetchPermits(userToken)

      const token = await this.fetchPermits(userToken)
      //console.log(token)
      this.setToken(token);

//this.setState({permits:this.state.permits+e.target.permits.value})
      //console.log('tokenupdated',this.state.token.user)
      //console.log("permits",this.state.token.user.permits)

        //this.setToken()
        //this.setToken(data)
    }
  }

  async fetchPermits(userToken){
    return fetch('http://localhost:8082/permits', {
      method: 'POST',
```

```
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(userToken)
    })
      .then(data => data.json())
  }

  onSubmit(e){
    e.preventDefault();
    e.persist()
    console.log('submitted');
    ipfs.files.add(this.state.buffer, (error,result) => {
      if(error){
        console.error(error)
        return
      }
      const userToken = this.getToken()
      var patientId = userToken.user.id
      if(userToken.user.type!=='Patient'){
        patientId = e.target.patientId.value
      }

      this.setState({ipfsHash:result[0].hash})
      console.log('Hash: ',this.state.ipfsHash)
      this.saveHash(patientId,userToken)
      const                          newWindow                          =
window.open('http://localhost:8080/ipfs/'+result[0].hash,   '_blank',
'noopener,noreferrer')
      if (newWindow) newWindow.opener = null
    })
  }

  async saveFiles(patientId,userToken,hash){
    return fetch('http://localhost:8082/savefiles', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body:
JSON.stringify({patientId:patientId,ipfsHash:hash,userToken})
    })
      .then(data => data.json())
  }

  async lookFiles(e){
    e.preventDefault();
    const userToken = this.getToken()
    const patientId = e.target.patientId.value
    console.log(patientId,userToken)
    if(patientId!==''){
      const files = await this.fetchFiles(patientId,userToken)
```

```
      this.setPatientFiles(files)
      if(files!==null){
        const                          newWindow                          =
window.open('http://localhost:8080/ipfs/'+files.files,      '_blank',
'noopener,noreferrer')
        if (newWindow) newWindow.opener = null
      }
    }
  }

  async fetchFiles(patientId,userToken){
    return fetch('http://localhost:8082/lookfiles', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({patientId:patientId,userToken})
    })
      .then(data => data.json())
  }

  async saveHash(patientId,userToken){
    const { accounts, contract } = this.state;
    await   contract.methods.set(this.state.ipfsHash,).send({   from:
accounts[0] });
    const response = await contract.methods.get().call();
    this.setState({ storedHash: response });
    console.log('storedHash: ',this.state.storedHash)
    const token = await this.saveFiles(patientId,userToken,response)
    this.setToken(token)
  }

  async getHash(){
    const { accounts, contract } = this.state;
    const response = await contract.methods.get().call();
    this.setState({ storedHash: response });
    console.log('Hash: ',this.state.storedHash)
  }

  setToken(userToken) {
    sessionStorage.setItem('token', JSON.stringify(userToken));
    this.setState({token:JSON.stringify(userToken)});
  }

  killSession() {
    sessionStorage.clear();
    this.setState({token:null});
    console.log('logged out')
  }

  setPatientFiles(files) {
    sessionStorage.setItem('Patient Files', JSON.stringify(files));
```

```
    this.setState({patientFiles:JSON.stringify(files)});
  }

  getToken() {
    var tokenString = sessionStorage.getItem('token');
    var userToken = JSON.parse(tokenString);
    if(!this.state.token && userToken!=null)
this.setState({token:userToken})
    return userToken;
  }

  render() {
    const token = this.getToken();
    //console.log('token:',token)
    if (!this.state.web3) {
      return <div>Loading Web3, accounts, and contract...</div>;
    }
    //console.log('prelogin',this.state.token)
    if(!token) {
      //console.log('setin token')
      return (
        <div className='SignUpLogIn'>
          <div  className='SignUp'><SignUp  setToken={this.setToken}
/></div>
          <div   className='LogIn'><Login   setToken={this.setToken}
/></div>
        </div>
      )
    }

    //console.log('postlogin',this.state.token)
    //console.log('settoken:',token)
    //console.log('usertype',this.state.token.user.type)
    if(token.user.type==='Patient'){
      return (
            <div className="App">
              <h1>Hello, {token.user.name}</h1>
              <h2>Upload your file or give permits</h2>
              <form      className=""      onSubmit={this.onSubmit}
method="post">
                <label htmlFor="fileinput">Choose  file </label>
                <input  type="file"  id="fileinput"  name="fileinput"
onChange={this.captureFile} accept=".pdf"/>
                <input type="submit" name="" value="upload"/>
              </form>
              <form   className=""   onSubmit={this.capturePermits}
method="post">
                <div><br/>Give  permits  to:  <input  type="text"
id="permits" name="permits" /></div>
                <br/><input   type="submit"   name=""   value="Give
permits"/>
              </form>
```

```
              <div><br/>You have given permits to:
{token.user.permits}</div>
              <div><br/>Records: {token.user.files}</div>
              <br/><input type="button" name="logout" value="Log Out"
onClick={this.killSession}/>
            </div>
      );
    }
    if(token.user.type==='Doctor'){
      return (
        <div className="App">
          <h1>Hello, Dr. {token.user.name}</h1>
          <h2>Browse for patient's files</h2>
          <form className="" onSubmit={this.lookFiles} method="post">
            <div><br/>Enter patient ID: <input type="text"
id="patientId" name="patientId" /></div>
            <br/><input type="submit" name="" value="Search"/>
          </form>
          <h2>Add file to patient's records</h2>
          <form className="" onSubmit={this.onSubmit} method="post">
            <div><br/>Enter patient ID: <input type="text"
id="patientId" name="patientId" /></div>
            <br/><label htmlFor="fileinput">Choose file </label>
            <input type="file" id="fileinput" name="fileinput"
onChange={this.captureFile} accept=".pdf"/>
            <input type="submit" name="" value="upload"/>
          </form>
          <br/><input type="button" name="logout" value="Log Out"
onClick={this.killSession}/>
        </div>
      );
    }
    if(token.user.type==='Hospital'){
      return (
        <div className="App">
          <h1>{token.user.name}</h1>
          <h2>Add file to patient's records</h2>
          <form className="" onSubmit={this.onSubmit} method="post">
            <div><br/>Enter patient ID: <input type="text"
id="patientId" name="patientId" /></div>
            <br/><label htmlFor="fileinput">Choose file </label>
            <input type="file" id="fileinput" name="fileinput"
onChange={this.captureFile} accept=".pdf"/>
            <input type="submit" name="" value="upload"/>
          </form>
          <h2>Give permits to doctors</h2>
          <form className="" onSubmit={this.capturePermits}
method="post">
            <div><br/>Give permits to: <input type="text"
id="permits" name="permits" /></div>
            <br/><input type="submit" name="" value="Give permits"/>
          </form>
```

```
          <div><br/>You have given permits to:
{token.user.permits}</div>
          <br/><input type="button" name="logout" value="Log Out"
onClick={this.killSession}/>
        </div>
      );
    }
    return <h1>Look for file</h1>
  }
}

export default App;
```

Login.js

A continuación se muestra el código de la página de inicio de sesión.

```
import React, { useState } from 'react';
import PropTypes from 'prop-types';
import './Login.css';

async function loginUser(credentials) {
 return fetch('http://localhost:8082/login', {
   method: 'POST',
   headers: {
     'Content-Type': 'application/json'
   },
   body: JSON.stringify(credentials)
 })
   .then(data => data.json())
}

export default function Login({ setToken }) {
  const [id, setID] = useState();
  const [password, setPassword] = useState();
  const handleSubmit = async e => {
    e.preventDefault();
    const token = await loginUser({
      id,
      password
    });
    setToken(token);
  }

  return(
    <div className="login-wrapper">
      <h1>Please Log In</h1>
      <form onSubmit={handleSubmit}>
        <label>
          <p>Username</p>
          <input     type="text"     name="id"     onChange={e     =>
setID(e.target.value)} />
        </label>
        <label>
          <p>Password</p>
          <input   type="password"   name="password"   onChange={e   =>
setPassword(e.target.value)} />
        </label>
        <div>
        <br/>
          <button type="submit">Log In</button>
        </div>
      </form>
    </div>
```

```
  )
}

Login.propTypes = {
  setToken: PropTypes.func.isRequired
};
```

SignUp.js

Muy parecido a Login.js.

```
import React, { useState } from 'react';
import PropTypes from 'prop-types';
import './Login.css';

async function signUser(credentials) {
 return fetch('http://localhost:8082/signup', {
   method: 'POST',
   headers: {
     'Content-Type': 'application/json'
   },
   body: JSON.stringify(credentials)
 })
   .then(data => data.json())
}

export default function SignUp({ setToken }) {
  const [name, setName] = useState();
  const [password, setPassword] = useState();
  const [type, setType] = useState();
  const [id, setID] = useState();

  const handleSubmit = async e => {
    e.preventDefault();
    const token = await signUser({
      name,
      type,
      id,
      password
    });
    setToken(token);
  }

  return(
    <div className="login-wrapper">
      <h1>Welcome, Sign Up</h1>
      <form onSubmit={handleSubmit}>
        <label>
          <p>Name</p>
          <input    type="text"    name="name"    onChange={e    =>
setName(e.target.value)} />
        </label>
        <label>
          <p>Type of user</p>
          <input      type="radio"      name="type"      id="patient"
value='Patient' onChange={e => setType(e.target.value)} />
          <label htmlFor="patient">Patient</label><br/>
```

```
          <input type="radio" name="type" id="doctor" value='Doctor'
onChange={e => setType(e.target.value)} />
          <label htmlFor="doctor">Doctor</label><br/>
          <input      type="radio"      name="type"      id="hospital"
value='Hospital' onChange={e => setType(e.target.value)} />
          <label htmlFor="hospital">Hospital</label>
        </label>
        <label>
          <p>ID</p>
          <input     type="text"     name="id"     onChange={e     =>
setID(e.target.value)} />
        </label>
        <label>
          <p>Password</p>
          <input   type="password"   name="password"   onChange={e   =>
setPassword(e.target.value)} />
        </label>
        <div>
        <br/>
          <button type="submit">Sign Up</button>
        </div>
      </form>
    </div>
  )
}

SignUp.propTypes = {
  setToken: PropTypes.func.isRequired
};
```

Server.js

Por último, se muestra el código del servidor, con todos los métodos para llevar a cabo identificación o guardado de información.

```
const express = require('express');
const cors = require('cors')
const app = express();
const bodyParser = require('body-parser');
const multer = require('multer');
const upload = multer();
const session = require('express-session');
const cookieParser = require('cookie-parser');

var fs=require('fs');
var data=fs.readFileSync('Users.json', 'utf8');
var DB=JSON.parse(data.toString());

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(upload.array());
app.use(cookieParser());
app.use(cors());

app.use('/signup', function(req, res){
  if(!req.body.id || !req.body.password){
    res.status("400");
    res.send("Invalid details!");
  } else {
    DB.Users.filter(function(user){
      if(user.id === req.body.id){
          console.log('User already existst')
          res.sendStatus(500)
        }
      });
      var newUser = {name: req.body.name, type: req.body.type, id:
req.body.id, password: req.body.password, files: null, permits:
null};
      DB.Users.push(newUser);
      fs.writeFileSync('Users.json', JSON.stringify(DB,null,4))
      res.send({user:newUser});
    }
});

app.use('/login', function(req, res){
  if(!req.body.id || !req.body.password){
    console.log("Please enter both id and password")
  }else{
    DB.Users.filter(function(user){
```

```
      if(user.id    ===    req.body.id    &&    user.password    ===
req.body.password){
        console.log("logged: ",user)
        res.send({user:user});
      }
    });
  }
});

app.use('/permits', function(req, res){
  DB.Users.filter(function(user){
    if(user.id === req.body.user.id){
      user.permits=req.body.user.permits
      fs.writeFileSync('Users.json', JSON.stringify(DB,null,4))
      res.send({user:user})
    }
  })
});

app.use('/lookfiles', function(req, res){
  if(req.body.userToken.type === 'Patient'){
    DB.Users.filter(function(user){
      if(user.id === req.body.patientId){
          res.send({files:user.files})
      }
    })
  }else{
    DB.Users.filter(function(user){
      if(user.id === req.body.patientId){
        DB.Users.filter(function(nestedUser){
          if((user.permits.includes(req.body.userToken.user.id)    &&
nestedUser.id       !==       user.id       &&       nestedUser.id
===req.body.userToken.user.id) || (nestedUser.type === 'Hospital' &&
nestedUser.permits.includes(req.body.userToken.user.id)           &&
user.permits.includes(nestedUser.id))){
            console.log('autorizado')
            res.send({files:user.files})
          }else{
            console.log('no autorizado')
          }
        })
      }
    })
  }
});

app.use('/savefiles', function(req, res){
  if(req.body.userToken.type === 'Patient'){
    DB.Users.filter(function(user){
      if(user.id === req.body.patientId){
          user.files = req.body.ipfsHash
          fs.writeFileSync('Users.json', JSON.stringify(DB,null,4))
```

```
            res.send({user:user})
          }
        })
      }else{
        DB.Users.filter(function(user){
          if(user.id === req.body.patientId){
            DB.Users.filter(function(nestedUser){
              if((user.permits.includes(req.body.userToken.user.id)    &&
nestedUser.id       !==       user.id       &&       nestedUser.id
===req.body.userToken.user.id) || (nestedUser.type === 'Hospital' &&
nestedUser.permits.includes(req.body.userToken.user.id)            &&
user.permits.includes(nestedUser.id))){
                user.files = req.body.ipfsHash
                fs.writeFileSync('Users.json',
JSON.stringify(DB,null,4))
                res.send({user:nestedUser})
              }else{
                console.log('no autorizado')
              }
            })
          }
        })
      }
});


app.listen(8082,    ()    =>    console.log('API    is    running    on
http://localhost:8082'));
```

15 Bibliografía

[1]. Batra, Gaurav. Olson, Remy. Pathak, Silphi. Santhanam, Nick. Soundararajan, Harish. *"Blockchain 2.0: What's in store for the two ends—semiconductors (suppliers) and industrials (consumers)?"*. McKinsey & Co. Retrieved 25 July 2021.

[2]. Castellanos, Sara. "A Cryptocurrency Technology Finds New Use Tackling Coronavirus". The Wall Street Journal. Retrieved 21 October 2020.

[3]. Grüner, A. Mühle, T. Gayvoronskaya and C. Meinel, "A Quantifiable Trust Model for Blockchain-Based Identity Management", *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Halifax, NS, Canada, 2018, pp. 1475-1482, doi: 10.1109/Cybermatics_2018.2018.00250.

[4]. Comunidad de Madrid. Consejería de Sanidad. *"Historia Clínica Digital del Sistema Nacional de Salud"*. Consultada el 28 de enero de 2021.

[5]. *Beck, Kent. ...[et al]*. "Manifesto for Agile Software Development". Consultada el 20 de junio de 2021.