Trabajo de Fin de Grado

Grado en Ingeniería Informática

Blockchain, los Smart Contracts y un caso de uso.

Blockchain, the Smart Contracts and a use case.

Marcos Luis Delgado

La Laguna, 5 de septiembre de 2017

D. **Jesús Alberto González Martínez**, con N.I.F. 43.779.378-M profesor Colaborador de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"Blockchain, los Smart Contracts y un caso de uso."

ha sido realizada bajo su dirección por D. **Marcos Luis Delgado**, con N.I.F. 78.648.054-Z.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de septiembre de 2017.

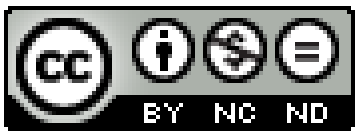
Agradecimientos

Agradezco a toda la gente que me rodea por ser capaces de soportar a alguien como yo.

Cabe la especial mención de mi madre y mi hermano.

También me gustaría mostrar mi gratitud hacia mi tutor Jesús Alberto González Martínez por su colaboración y tiempo.

Licencia



Resumen

El objetivo de este trabajo ha sido la investigación de la tecnología Blockchain y los Smart Contracts, hasta llegar al nivel de crear una aplicación que pueda aprovecharlos para un caso real de uso.

Tras el estudio de lo anterior y ciertas deliberaciones concluimos el caso de una empresa de seguros que quiere proporcionar beneficios a sus clientes de forma automatizada si cumplen ciertos requisitos previamente acordados. El enfoque ha sido un seguro de enfermedad para el que baja la cuota si el usuario se comporta de forma saludable.

Para ello decidimos crear una aplicación web que usa un Smart Contract previamente negociado entre ambas partes, y que, alimentándose de los datos recogidos por una pulsera Fitbit pueda automatizar este proceso.

Hemos utilizado bastantes tecnologías que se han visto en la carrera, aparte de nuestra capacidad de trabajo autónomo y resolución de problemas para crear una aplicación usable e interactiva que cumple con los objetivos planteados.

Palabras clave: Blockchain, Smart Contract, Fitbit, Automatización.

Abstract

The objective of this project has been the research about the Blockchain technology, and, more concretely, about the Smart Contracts. We wanted to reach a level where we would be able to develop an application that could be used in a real case.

After the previous study, we took advantage of the knowledge acquired to think of an application where the technology would fit. This leaded us to the idea of using it to provide the clients of an insurance company certain benefits (which would of course benefit the company too). We wanted an application that, using blockchain, would automatize the process of granting these benefits.

We decided creating an app that would use a Smart Contract previously negotiated between parts would grant discounts in the quote of health insurance to those users who acted healthy. To know if users acted healthy we used Fitbit.

We have taken advantage of the technologies that we used in the career, as of the skills of autonomous work and capacity to solve problems to create a usable and interactive application that fits the desired objectives.

Keywords: *Blockchain, Smart Contract, Fitbit, automatization.*

Índice General

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

Blockchain en la actualidad es considerado por los gurús como una de las tecnologías más innovadoras y disruptivas en el mundo de la informática [1][5]. A pesar de lo anterior, y de que, al contrario de lo que se puede pensar Blockchain ha sido un concepto desde la concepción del Bitcoin en el año 2009, la tecnología aún se encuentra en estado de inmadurez y en periodo de investigación y crecimiento [2].

Blockchain puede ser concebido como una tecnología equiparable a Internet. Esto se debe a que trata sobre los mismos protocolos que esta. Es un nuevo paradigma orientado al descubrimiento, evaluación y transferencia de todo tipo de datos, por lo que se puede considerar a Bitcoin como una criptomoneda que se apoya sobre la estructura Blockchain, pero esta puede soportar mucho más [3].

Esta tecnología cuenta con una gran cantidad de aplicaciones, pero en este trabajo nos centraremos en su uso para la creación de contratos inteligentes.

Actualmente, en el mundo de los seguros de enfermedad no existen bonificaciones para los clientes en relación con sus hábitos de vida. Esto probablemente se debe a que no hay forma de certificar los mismos por parte del usuario.

Este trabajo se enfoca a la creación de un Contrato Inteligente sobre una red de blockchain que se encargue automáticamente de otorgar estos beneficios al usuario partiendo de los datos proporcionados por una pulsera Fitbit.

Esto implicaría un cliente más contento y una aseguradora con la probabilidad a su favor.

1.1 Blockchain

Llegados a este punto, la primera pregunta que surge es qué es Blockchain. La respuesta a dicha pregunta se irá desarrollando a lo largo del estudio, ya que, como toda tecnología, no es algo que se pueda dar de manera rápida y concreta.

Como primera toma de contacto, se puede decir que Blockchain (cadena de bloques) es algo parecido a una base de datos distribuida que funciona como un libro de cuentas que va almacenando cualquier tipo de transacción realizada. Cabe aclarar que cuando hablamos de transacción no nos referimos a algo monetario, sino a un "intercambio de información". Cada una de estas transacciones se verifica a través de consenso entre la mayoría de los participantes de la red [4][5].

1.1.1 La red de Blockchain.

La red de Blockchain es una red distribuida y desintermediarizada [4], algo similar a las redes peer to peer utilizadas por eMule o Ares. La principal diferencia es que toda la información está almacenada por todos los usuarios o nodos de la misma.

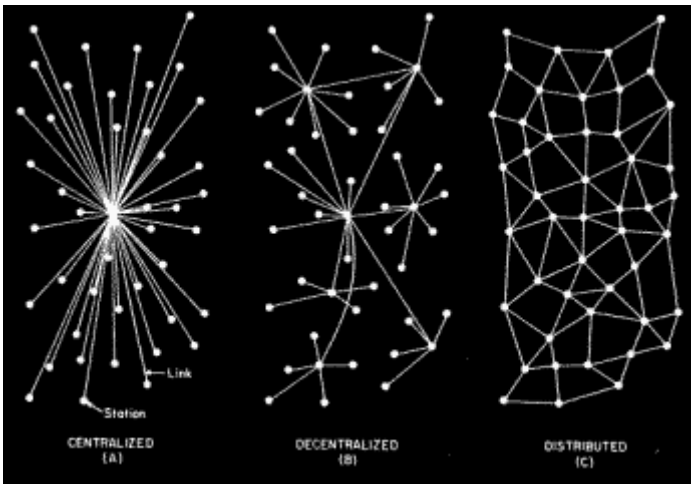


Figura 1. Tipos de redes.

Cuando un nodo entra en la red, lo primero que hace es descargar toda la información que la misma contiene para llegar al nivel de avance de los demás. Esto se debe, como se ha mencionado, a que la red actúa como un

libro de cuentas. Por lo tanto, para poder validar cualquier intercambio se debe conocer por donde ha pasado dicha "moneda" anteriormente. Es decir, no se lleva un registro de "monedas" existentes y sus propietarios, se lleva un registro del historial de las mismas. Para saber si un usuario tiene se deben comprobar todas las transacciones que le impliquen, siendo el resultado de esta operación su balance.

Cabe destacar que, si entra en la red un nodo que simplemente quiere realizar transacciones y no validarlas, no necesita descargar todos los bloques.

Cada vez que se añade un nuevo bloque a la red, los nodos deben validarlo resolviendo una especie de puzle informático. Puede haber varios tipos de consenso, pero el más común obliga para la aceptación del mismo que la mayoría (51%) de los nodos estén de acuerdo en el resultado.

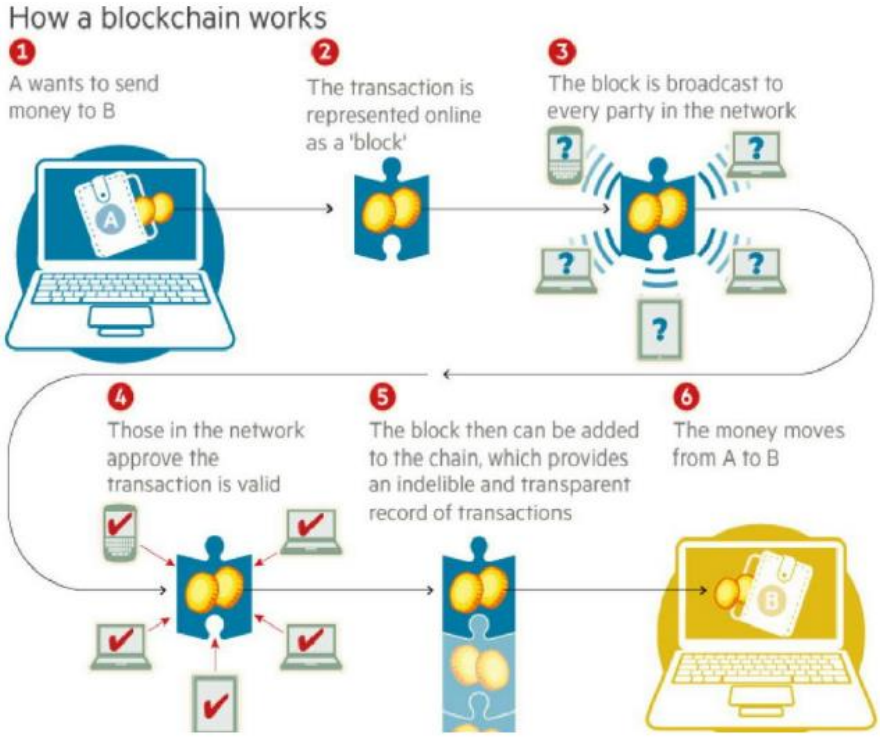


Figura 2. Funcionamiento de la red.

Estos algoritmos de consenso permiten que toda la red se mantenga en el mismo punto en todo momento, evitando las discrepancias entre interesados [4].

A parte de lo anterior, la red Blockchain está pensada para permitir la existencia tanto de datos privados como públicos. Los datos públicos pueden ser vistos por todos los nodos, mientras que los privados se hayan encriptados

para que, aunque todos los nodos los posean, no se les permita el acceso y desencriptación.

1.1.2 Los bloques.

Toda la información de la red Blockchain se almacena en bloques. Estos bloques de datos se almacenan de forma cronológica y pueden estar o no encriptados. Cada uno de ellos tiene un hash del bloque anterior, por lo tanto, desde que entra en la cadena no puede ser eliminado. Esto implica que una vez que un bloque lleva almacenado un determinado tiempo en la cadena es prácticamente imposible su alteración [1][4].

Estos bloques suelen llevar una cantidad fija o máxima de datos, o un tiempo de almacenaje, dependiendo del tipo de Blockchain.

El primer bloque de una cadena se llama bloque génesis, y contiene las primeras transacciones o la configuración de la cadena.

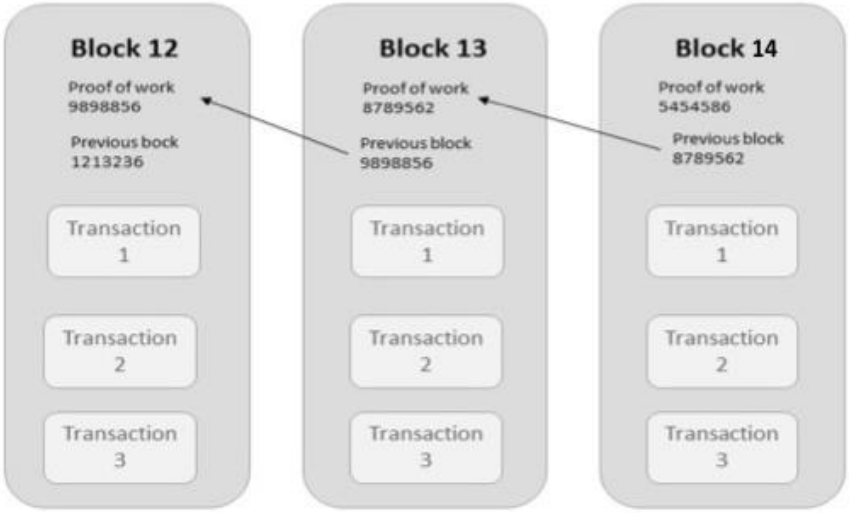


Figura 3. Los bloques.

1.2 Los Smart Contracts.

Se puede definir un Smart Contract o Contrato Inteligente como un acuerdo entre partes que se resuelve por sí mismo. Esto significa que la propia red de Blockchain es la que se encarga de hacer cumplir y ejecutar los acuerdos entre las partes interesadas [1].

Blockchain facilita que este contrato viva en un entorno no controlado por ninguna de las partes (la red).

Los contratos inteligentes dentro de Blockchain no son más que pequeños programas en algún lenguaje que la red pueda interpretar. Estos pueden almacenar datos y realizar operaciones sobre ellos.

1.2.1 Redes que soportan Smart Contracts.

En la actualidad existen multitud y cada vez hay más redes Blockchain que soportan los Smart Contracts. Entre ellas, las más conocidas son Eris [7], Ethereum [8] y, la más reciente, Hyperledger [9].

Las 2 primeras soportan Contratos Inteligentes codificados en Solidity, un lenguaje creado para la realización de los mismos muy parecido a Javascript [8].

Hyperledger, sin embargo, opta por el uso de un lenguaje asentado, Go [9].

Capítulo 2.
Estado del arte.

Este capítulo trata de detallar los Blockchains más conocidos para la realización de Smart Contracts, así como las principales características de los mismo. También se analizan los diferentes algoritmos de consenso.

2.1 Introducción.

Actualmente, cada día, la tecnología Blockchain va madurando incrementalmente y con ella, las plataformas que la implementan.

Hoy en día, las redes Blockchain son clasificadas por generaciones. La clasificación se realiza de la siguiente manera:

- **Primera generación**: se basa en la idea de realizar un sistema de registro compartido o un "ledger" donde poder ver las transacciones.
- **Segunda generación**: se extiende la idea anterior donde las plataformas crean una red donde se pueden utilizar criptomonedas y donde se almacenan las relaciones del crédito y divisas definidas por los usuarios.
- **Tercera generación**: plataformas cuyo propósito principal es la creación de aplicaciones descentralizadas usando como tecnología subyacente las plataformas de segunda generación.

A lo largo de esta sección, haremos una comparación sobre tres plataformas Blockchain: Eris, Ethereum y Hyperledger, que, desde nuestro punto de vista, son las más potenciales y aconsejables a la hora de ser utilizadas para la implementación de aplicaciones basadas en esta nueva tecnología.

Antes de centrarnos sobre la comparativa de las características técnicas de estas plataformas, en el caso de que éstas no sean conocidas, haremos una breve descripción sobre cada una para familiarizarnos con ellas.

2.2 Ethereum

Ethereum es una plataforma de segunda generación descentralizada que fue publicada en el año 2014. Inicialmente fue implementada en los lenguajes C++ y Go. Actualmente cuenta con REST API y APIs para los lenguajes Javascript, Python y Go, entre otros, y está en desarrollo para Java [8].

Esta plataforma cuenta con su propia criptomoneda, el Ether. Este tipo de moneda, no es únicamente utilizada en el sistema con la misma función que el resto de criptomonedas existentes, transacciones de valor monetario, sino que la moneda también es utilizada a la hora de desplegar los contratos inteligentes [8].

2.3 Eris

Eris, creada por Eris Industries (recientemente han cambiado su nombre por Monax Industries), es una plataforma de tercera generación que está desarrollada por dicha compañía, que es la principal valedora de esta plataforma y su mantenimiento. La plataforma ERIS es una plataforma de Blockchain que no está anclada en el mundo bancario y que propone un modelo más de propósito general [7].

2.4 Hyperledger

Finalmente, Hyperledger es la plataforma más nueva y que lleva menos tiempo en el "mercado" de las plataformas Blockchain. El proyecto colaborativo, Hyperledger Project, se inició en diciembre de 2016 por parte de Linux Foundation y la plataforma sigue aún en desarrollo [9].

Al igual que Eris, esta plataforma es de tercera generación. Sin embargo, su estado en desarrollo no implica que algunas de sus funcionalidades no puedan ser utilizadas.

El ideal básico de esta plataforma es que el estándar abierto de Blockchain debe basarse en la modularidad.

2.5 Composición de la red

Ambas redes Blockchain están compuestas por nodos. Sin embargo, cada red tiene su propio tipo de nodo e incluso tienen una propia clasificación de nodos como iremos viendo.

Mientras que Ethereum tiene un único tipo de nodo que puede realizar todas las funcionalidades de la cadena, Eris y Hyperledger tienen una clasificación interna de nodos.

Por parte de Eris, la clasificación de los nodos que conforman la red están basados en un sistema de permisos. Cada nodo, tiene las propiedades del anterior. La clasificación es la siguiente:

- **Participante**: no tiene permisos de validación de la cadena, su principal función es distribuir las transacciones por la red.
- **Validador:** tras realizar el algoritmo de consenso, validan las transacciones.
- **Full:** posee todo tipo de privilegios en la red y es el que más carga de trabajo pude tener a la hora de realizar el consenso como se verá más adelante.

Por otro lado, la clasificación de nodos de Hyperledger está basado en la funcionalidad que desempeña cada nodo. Los tipos de nodos que posee son los siguientes:

- **Membership Services**: Emite y administra la identidad de los usuarios y organizaciones. Para ello, emite: los certificados de inscripción de los participantes finales, entre otros tipos de certificados.
- **No validador:** sus funciones son la gestión de los certificados generados por el Membership Service y la construcción y reenvío de transacciones hacia el nodo validador.
- **Validador:** crea y valida las transacciones.

La gran ventaja de poder tener diferentes tipos de nodos es evitar la sobrecarga de la red. Por otro lado, a la hora también de tener diferentes tipos de nodos, de cara al usuario, se podría ver como una medida de seguridad de modo que se podrían limitar las acciones que puedan realizar sobre la red Blockchain.

2.6 Algoritmos de consenso

Ahora, pasaremos a hacer una comparativa entre los diferentes algoritmos de consenso que posee cada plataforma. El algoritmo de consenso es utilizado en las redes Blockchain para realizar un acuerdo entre los nodos que conforman la red y poder así validar o no una transacción.

Los algortimos de consenso utilizados son: Proof of Stake (PoS) en el caso de Eris, Proof of Work (PoW) en el caso de Ethereum y Practical Byzantine Fault Tolerance (PBFT) en el caso de Hyperledger. En el caso de esta última plataforma, aparte de utilizar PBFT, ésta ha sido implementada para poder seleccionar qué algoritmo de consenso utilizar [6].

Ethreum tiene su propia variante de Proof of Work conocida como Ethash. Proof of Work es un algoritmo de consenso que básicamente consiste en proporcionar a los nodos una serie de operaciones computacionales para que sean realizadas.

Estas operaciones computacionales tienen como objetivo calcular el hash de un bloque para ser incluido en la cadena. El hash de este bloque está formado por el hash del bloque anterior, los hashes de las transacciones que van a ser incluidas en el mismo y un número. El cálculo de este número es donde se encuentra la mayor carga computacional y lo que determina si el hash del bloque es válido o no para ser incluido a la red.

Está previsto que, en la próxima versión de Ethereum, el algoritmo de consenso sea cambiado a Proof of Stake.

Proof of Stake, nació teniendo como base Proof of Work y, como hemos comentado anteriormente, es el algoritmo de consenso de Eris**.** Su principal diferencia con la anterior se basa en que, a la hora de llegar a un acuerdo en el consenso, el nodo que tenga más moneda es el que más repercute sobre la decisión de validar o no una transacción.

Este algoritmo, con vistas a un futuro lejano, podría dejar de cumplir el principio más importante de las redes Blockchain, la descentralización.

Por el contrario, Hyperledger utiliza el algoritmo PBFT para entornos de producción, el que el usuario desee (en un futuro) y NOOPS para entornos desarrollo. La idea principal de primer algoritmo es que, para poder validar

una transacción en n nodos, ha de ser validada por dos tercios de los nodos más uno para que esta sea correcta.

Actualmente, dado al estado de crecimiento de Hyperledger, su versión estable sólo incluye el modo para desarrolladores y sólo se puede utilizar el algoritmo de consenso NOOPS, en el que se toman todas las transacciones como válidas para incluirlas en la cadena [9].

Por lo tanto, en vista al breve análisis que hemos realizado, a la hora de realizar una aplicación basada en redes Blockchain hay que tener en cuenta cuál es el caso de uso sobre el que se va a basar la misma.

Si el caso de uso está relacionado con las transacciones financieras, es recomendable utilizar una red de segunda generación como Ethereum. Sin embargo, si se quiere implementar una aplicación descentralizada basada en un smart contract se debería utilizar una red de tercera generación como Eris.

Actualmente, Hyperledger es una gran candidata a ser una de las redes Blockchain de tercera generación más potentes. Sin embargo, debido a su grado de desarrollo, es más recomendable utilizarla para desarrollos de contratos o implementación de pruebas de concepto.

Capítulo 3.
Caso práctico.

En este capítulo se describen el caso práctico a realizar. Comenzamos con los objetivos, para luego pasar a la investigación y desarrollo y finalizar con una breve descripción de las herramientas utilizadas.

3.1 Objetivos.

Una vez estudiadas las redes y los Smart Contracts, comenzamos a buscar ideas.

Un posible uso de un Contrato Inteligente puede ser el hecho de mediar entre un cliente y su proveedor, asegurando que lo acordado se va a cumplir.

Tras ahondar más en esa idea comenzamos a pensar en casos más concretos. El problema principal es que los contratos a los que estamos acostumbrados en el día a día es que son muy complicados de transcribir a un programa informático con un lenguaje básico como lo es Solidity, por lo tanto, decidimos crear algún tipo de contrato nuevo y más "coloquial".

La primera idea sólida es crear una biblioteca pública peer to peer. Aquí se nos plantea el problema de cómo sería el contrato. Las cláusulas más básicas, como asegurar la devolución de los libros son complicadas, por no decir imposibles, de asegurar.

Pensamos entonces en modos de penalizar a los usuarios, pero vemos que este caso de uso es demasiado complicado.

Llegamos entonces a la segunda idea, que surge de una charla con un compañero acerca de la pulsera Fitbit. Esta pulsera monitoriza la actividad física de los usuarios, que es almacenada en una base de datos y accesible a través de una API.

Aquí surge la pregunta, ¿a quién le puede interesar esta información a parte del propio usuario?

Llegamos entonces a la conclusión de que a las compañías de seguros. Una compañía de seguros que conoce los hábitos de vida de su cliente cuenta con la probabilidad de su parte, y un cliente al que se le ofrecen beneficios por sus hábitos de vida es un cliente feliz.

Se pretende crear una aplicación que monitorice la actividad física del usuario (a través de una pulsera Fitbit) y sea capaz, por medio de un contrato inteligente de proporcionarle ciertos beneficios si alcanza unos objetivos previamente acordados.

Esto plantea 2 tipos de usuario de la misma:

- **Administrador**: Sería el que es capaz de dar de alta a nuevos usuarios y crear contratos para los mismos especificando una serie de objetivos. Por motivos de testeo también incluiremos la posibilidad de borrar usuarios/contratos.
- **Usuario**: Debe ser capaz de ver su progreso accediendo al portal. También sería interesante que reciba algún tipo de notificación en caso de que cumpla el contrato y vaya a obtener su notificación.

Dado que no tenemos la posibilidad física de otorgar un beneficio (como podría ser contar con integración con la aseguradora para realizar un descuento en la cuota) simplemente mostraremos este beneficio.

3.2 Investigación y desarrollo.

3.2.1 Blockchain.

Tras conocer el estado del arte se procede a la prueba de las redes. Se siguen los manuales de Eris y Ethereum, ya que Hyperledger ha sido descartado debido a la inmadurez del proyecto y el alto número de fallos.

Después de lograr hacer funcionar ambas redes comenzamos con los primeros ejemplos de Smart Contracts que se pueden encontrar online.

Ethereum, al ser una red de segunda generación comienza a dar problemas de falta de tokens, ya que se basa en el uso de los mismos para desplegar contratos. Como para el caso de uso planteado no se necesitan tokens la descartamos.

Finalmente, nos quedamos con Eris.

Tras ello, comenzamos a dockerizar todo el entorno para ser capaces de desplegarlo con la mínima cantidad posible de comandos y ponerlo en funcionamiento.

Creamos un contenedor para cada nodo y así simulamos una red real entre máquinas.

3.2.2 Smart Contract.

Comenzamos con el desarrollo del Smart Contract. El primer problema que surge es al darnos cuenta del aislamiento que provee la red Blockchain. Esta no es capaz de realizar una comunicación hacia fuera, por lo tanto no puede conectarse directamente a la API de Fitbit para recuperar los datos. Por consiguiente, el servidor será el encargado de hacer llegar estos datos al contrato.

Descubrimos que Solidity (el lenguaje de programación de los Smart Contracts en Eris) es capaz de lanzar eventos. Esto nos valdrá a la hora de notificar al usuario en caso de cumplimiento de los objetivos.

Con lo anterior en mente se empieza a codificar. Creamos un contrato capaz de guardar una serie de objetivos que nos parecen relevantes y que sabemos que Fitbit guarda y están disponibles a través de su API.

El contrato debe guardar los objetivos finales, los acumuladores de los mismos y la última fecha de actualización.

También añadimos una función que comprueba los objetivos y devuelve si el contrato ha sido completado. También se lanza un evento por cada uno si este ha sido alcanzado.

3.2.3 Frontal.

Para el frontal decidimos utilizar tecnologías que ya conocemos. Estas son AngularJS como framework de javascript y Bootstrap para los estilos.

Empezamos creando la página de inicio que permitirá a los usuarios y a los administradores acceder a sus respectivos paneles de control.

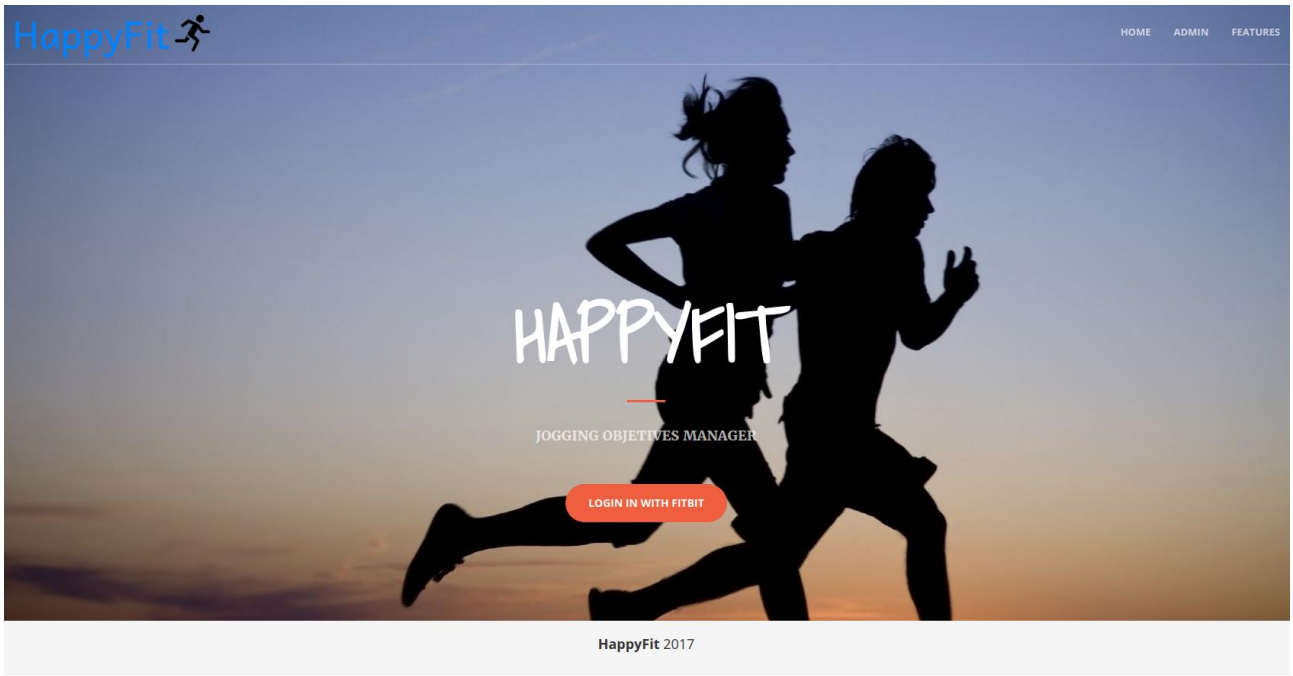


Figura 4. Página de inicio.

Luego creamos la página del login de administrador, así como su panel de control, que consta de dos tablas. En la primera se pueden crear nuevos contratos, asociados a un token de acceso que será utilizado por el usuario para registrarse. En la segunda se verán los usuarios activos asociados a su contrato, especificando los parámetros del mismo. La idea es que el servidor actualice los contratos con los datos de Fitbit diariamente, por lo tanto, para poder realizar pruebas también se ha decidido añadir un botón que permita actualizar los datos en el momento para motivos de testeo.

El administrador debe poder crear nuevos contratos. Para este menester se crea un pop up donde puede introducir todos los datos necesarios.

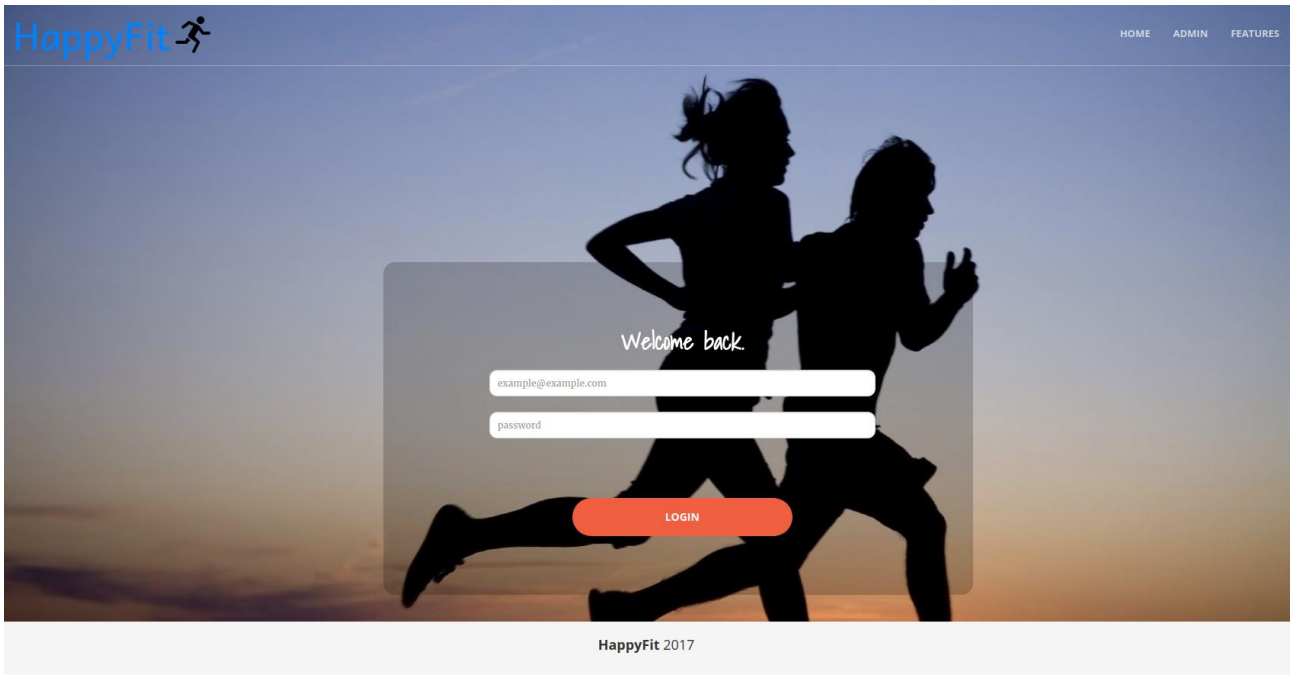


Figura 5. Página de login de administrador.

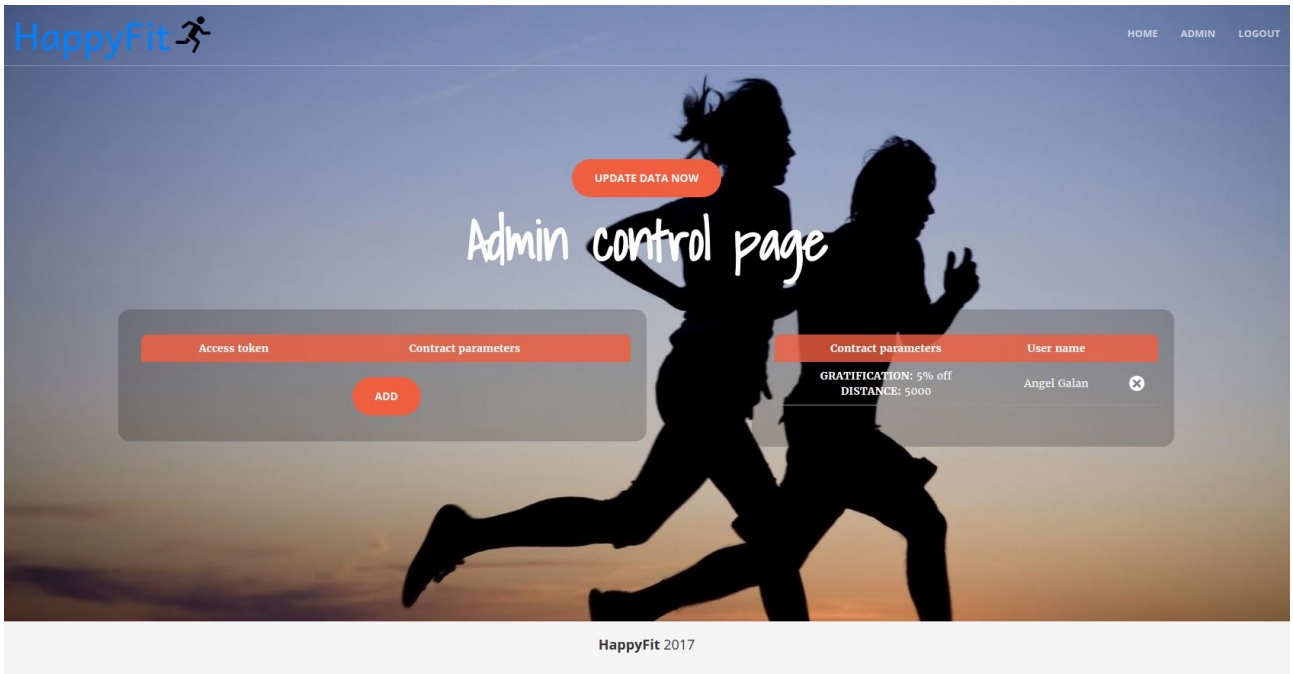


Figura 6. Página de administrador.

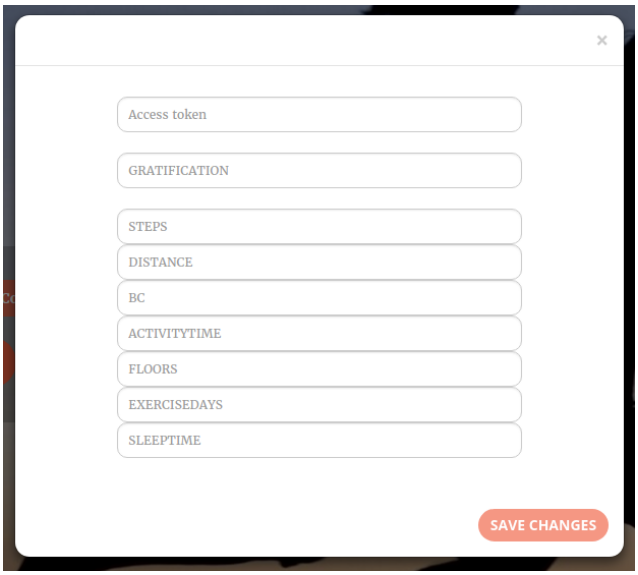


Figura 7. Pop up de creación de contrato.

Para la parte de usuario, tras un primer estudio de la API de Fitbit descubrimos que el login se realiza mediante OAuth. Por lo tanto, no necesitamos una página de login en sí, ya que esta la proporcionará Fitbit.

Sí que necesitamos, sin embargo, un formulario donde el usuario introduzca su correo y el token de acceso proporcionado por el administrador cuando haga su primer login. De esta forma se realizará su registro y el despliegue del Contrato Inteligente acordado.

También se crea la página donde el usuario puede ver su progreso mensual. Utilizamos barras de progreso y porcentajes para ofrecer una vista intuitiva.

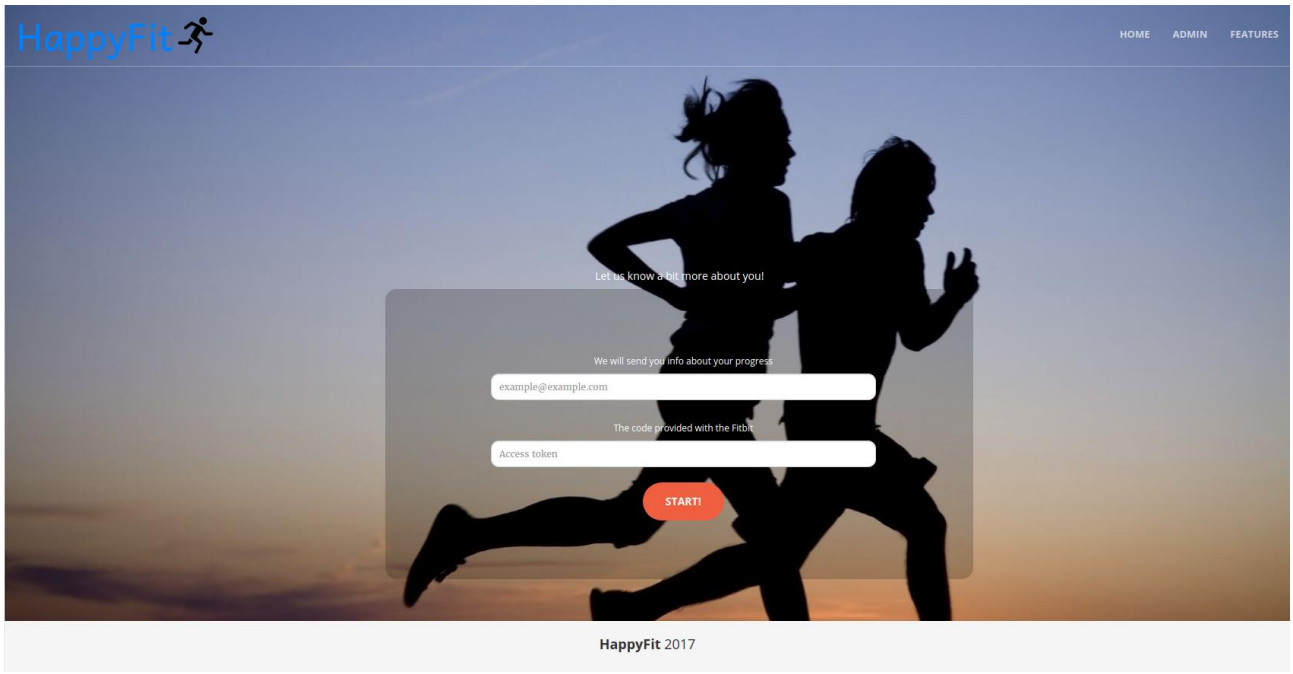


Figura 8. Página de registro de usuario.

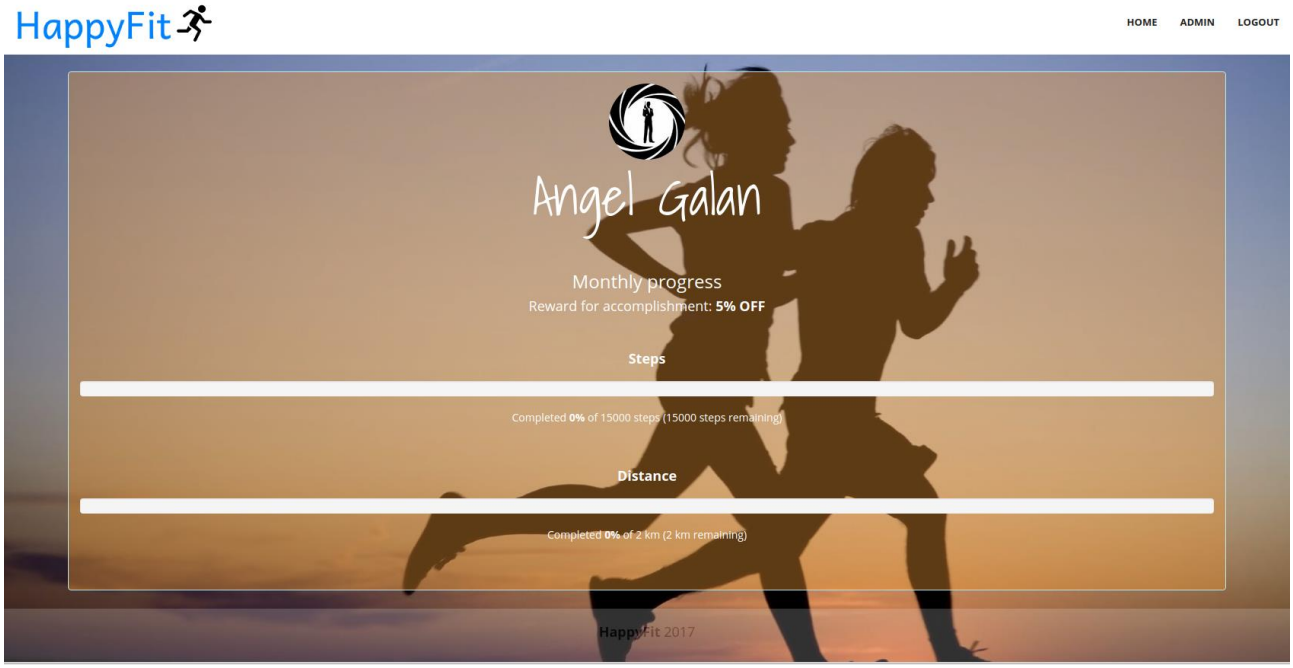


Figura 9. Página con información para el usuario.

3.2.4 API Fitbit.

Lo primero que debemos hacer es crear una nueva aplicación en el panel de control de desarrolladores de Fitbit.

Debido al uso del método OAuth debemos configurar también las URLS a las que se permitirá la redirección después del login por parte del usuario.

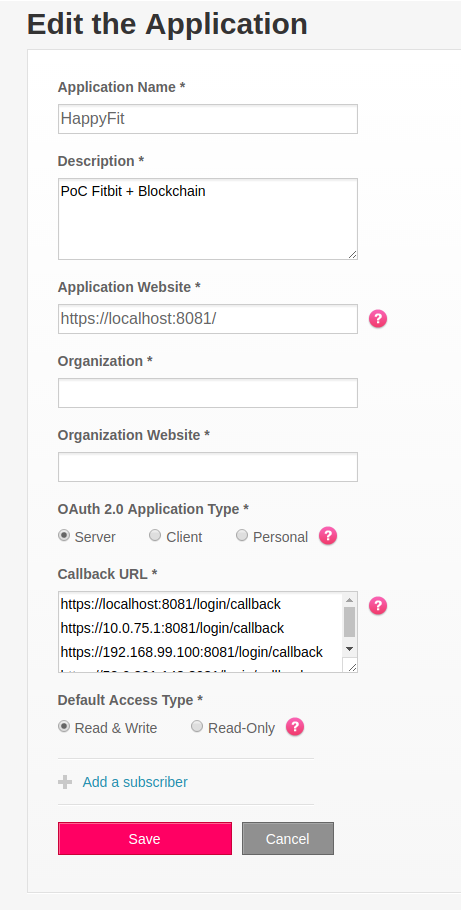


Figura 10. Configuración de la aplicación Fitbit.

También configuramos los permisos que tendrá la aplicación y empezamos con el estudio de la documentación de la API.

Encontramos los 3 métodos HTTP a usar: el método de autenticación, el método para recibir el perfil de usuario y el método para recibir los datos de actividad realizada.

- **Perfil de usuario**: Necesario para poder guardar datos como el nombre o el avatar que luego serán mostrados en el frontal.
- **Datos de actividad**: Los datos de actividad pueden ser recuperados entre determinadas fechas y horas. Esto nos será útil a la hora de actualizar el contrato añadiendo solo los datos necesarios.

3.2.5 MySQL Server.

Tras el estudio de la problemática y lo que será necesario guardar fuera de Blockchain se llega a la conclusión del siguiente modelo de datos.

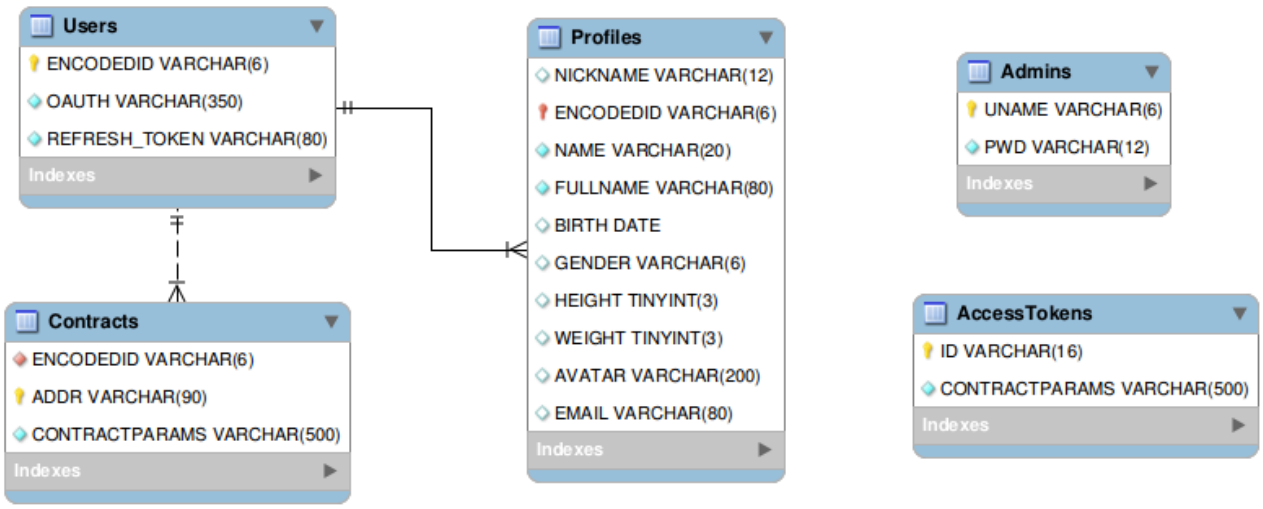


Figura 11. Modelo de datos MySQL.

- **Users**: Será la tabla encargada de guardar los usuarios. El campo ENCODEDID contendrá el id que le proporciona Fitbit a cada usuario, ya que será la forma de hacer login. Guardaremos también el token de OAuth y el token de refresh del propio OAuth. Esto se debe a que el servidor deberá ser capaz de recuperar datos de los usuarios aun cuando estos no están logueados físicamente para actualizar el Contrato Inteligente.
- **Contracts**: Esta tabla guarda los datos del contrato asociado al usuario. También se guardarán los parámetros con los que se ha inicializado el mismo para no tener que acceder a Blockchain cada vez que se quieran visualizar y minimizar así la carga.

  Por otra parte, para referenciar un contrato en Eris nos hace falta su dirección, que es un hash único que sirve para recuperarlo de la cadena.
- **Profiles**: Aquí persistiremos los datos del perfil de usuario que se recupera de Fitbit, a parte de su dirección de correo. Así podremos servirlos al frontal sin necesidad de recurrir a la API de Fitbit, que sería costoso.
- **Admins**: Guarda los administradores registrados.
- **AccessTokens**: Almacena los contratos sin asignar que crean los administradores. Para ello es necesario guardar el token que crea el administrador y los parámetros con los que se inicializará el contrato en caso de que un usuario lo reclame.

3.2.6 NodeJS Server.

Esta será realmente la pieza clave y más grande del trabajo. Más adelante explicaremos la estructura del código, pero ahora nos centraremos en su desarrollo.

Comenzamos con un sencillo servidor en NodeJS que simplemente sirve el frontal que ya hemos diseñado. Para ello utilizamos el paquete de NPM Express y las platillas EJS. Una vez que tenemos esto funcionando procedemos a la parte de la conexión con la base de datos.

Utilizamos el paquete sequelize, que ofrece una interfaz para conectar a MySQL y ser capaz de recuperar e insertar datos. Creamos los modelos de las tablas definidas en la base de datos.

A continuación, lo primero que debemos resolver es la conexión con Fitbit. Para ello utilizamos el paquete Passport, que cuenta con un plugin para conectarse a Fitbit. Lo configuramos con los datos de nuestra aplicación.

También debemos configurar la autenticación por parte de los administradores, que no depende de Fitbit. Utilizamos el mismo paquete con el tipo de autenticación local.

Ahora pasamos al ACL. Es necesario que se restrinja el acceso a los endpoints dependiendo del usuario que se halla actualmente logueado. Utilizamos el paquete acl, que puede funcionar también sobre la base de datos MySQL y se encarga de crear sus propias tablas e índices.

A parte de los roles anteriormente mencionados vemos necesaria la creación de dos más: el invitado y el usuario que no ha introducido el token del contrato que le corresponde.

- **Invitado**: Se le permitirá acceder a la página principal y la de registro por Fitbit.
- **Usuario sin confirmar**: A parte de los permisos de invitado contará con el permiso para acceder a la pantalla de introducción de token y correo electrónico, pero no a la de ver su progreso, ya que no tiene un contrato asociado.

Lo siguiente es crear las rutas de la API a las que podrá llamar el frontal para recuperar datos.

Creamos rutas para hacer login y logout. Luego realizamos métodos para la obtención de datos de usuario y su progreso. También creamos, para la parte de admin, métodos para obtener los datos que se mostrarán en las tablas, así como para crear nuevos contratos y borrarlos.

Para la parte de testeo, como mencionamos anteriormente, también necesitaremos una forma de actualizar los datos de un contrato instantáneamente.

Todas estas funciones de la API HTTP simplemente se encargan de realizar llamadas a los controladores, que explicaremos en el siguiente punto, siendo estos los que realizarán las operaciones y devolverán los resultados.

a) **Controladores**:

En primer lugar, para el acceso a la red Eris, realizamos un recubrimiento para la API Javascript que se nos ofrece. Esto se hace para simplificar las llamadas y enfocarlas a nuestras necesidades.

Sobre esta, construimos un nuevo módulo que es capaz de desplegar un Contrato Inteligente, así como de recuperar su referencia de la red Blockchain y llamar a sus funciones devolviendo sus resultados si procede.

Luego codificamos el controlador de la base de datos, que ofrece todos los métodos necesarios para la interacción con la misma.

A continuación, creamos un módulo que recorre los usuarios registrados y registra receptores para los eventos que escupe la red Blockchain, encargándose de mandar los correos a los usuarios cuando completan un objetivo apoyándose en un módulo previamente desarrollado.

Por último, creamos el módulo más complicado, que es el que actualiza los datos del contrato basándose en la fecha actual y la última fecha de actualización que persiste el contrato.

Esta complicación se basa en que asumimos que el servidor puede fallar y no enviar los datos todos los días como está previsto. Por lo tanto, se deben hacer una serie de comprobaciones que pueden incluir desde la necesidad de introducción de varios días hasta incluso un cambio de mes, caso en el que se debe actualizar los días restantes del mes pasado, resetear el contrato y luego cargar los días restantes del mes actual.

Finalmente, a través del paquete cron, configuramos la ejecución de la actualización de datos a todos los días.

3.3 Herramientas utilizadas.

3.3.1 Docker.

Para la creación de cada pieza del proyecto se ha utilizado Docker. Esta herramienta permite la creación de contenedores (algo parecido a máquinas virtuales) que pueden ser ejecutados en cualquier máquina que tenga Docker instalado.

Esto aporta portabilidad y sencillez para la ejecución de cada pieza.

3.3.2 Eris.

Se ha elegido como Eris como Blockchain a usar para la implementación del Smart Contract. Se trata de una red para la que se programa en Solidity.

Ofrece una API de Javascript que utilizaremos para las llamadas desde NodeJS.

3.3.3 NodeJS.

En la parte del servidor se ha utilizado NodeJS, que permite desarrollar el backend en Javascript.

3.3.4 NPM.

NPM Es el gestor de paquetes de NodeJS. Para la implementación del proyecto ha sido necesario el uso de varios paquetes que facilitaban el trabajo.

3.3.5 AngularJS.

En la parte del cliente se ha utilizado AngujarJS, un conocido framework que facilita el desarrollo.

3.3.6 MySQL.

Para el almacenamiento de datos se ha decidido utilizar una base de datos SQL que es fácilmente integrable con NodeJS.

3.3.7 Solidity.

La implementación del Smart Contract se ha realizado en Solidity, un lenguaje de programación creado específicamente para la creación de contratos inteligentes y que es el usado por la plataforma Eris.

3.3.8 API Fitbit.

Para recuperar los datos de la pulsera Fitbit se ha usado la API para desarrolladores (HTTP) que provee la propia empresa Fitbit.

Capítulo 4.
Aplicación desarrollada.

En este capítulo se detalla la arquitectura utilizada, la estructura por carpetas del proyecto y un caso de uso.

4.1 Arquitectura.

En esta sección se describe la arquitectura de la aplicación, haciendo un inciso en la de la propia red Blockchain utilizada.

4.1.1 Red de Blockchain.

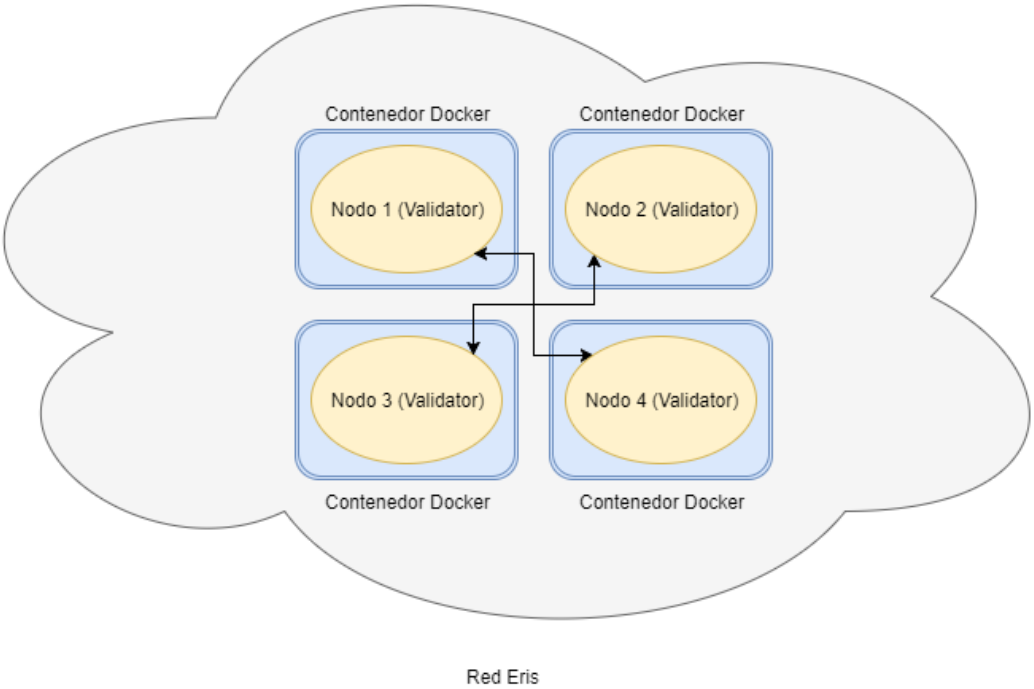


Figura 12. Arquitectura red Eris.

Para la arquitectura de la red Blockchain se han decidido crear 4 nodos validadores, que son el mínimo necesario para que la red funcione y haga consenso.

Para simular la existencia de los nodos en diferentes máquinas y facilitar su despliegue se han dockerizado. Como se puede ver en la Figura 1.4 están todos interconectados.

4.1.2 Aplicación.

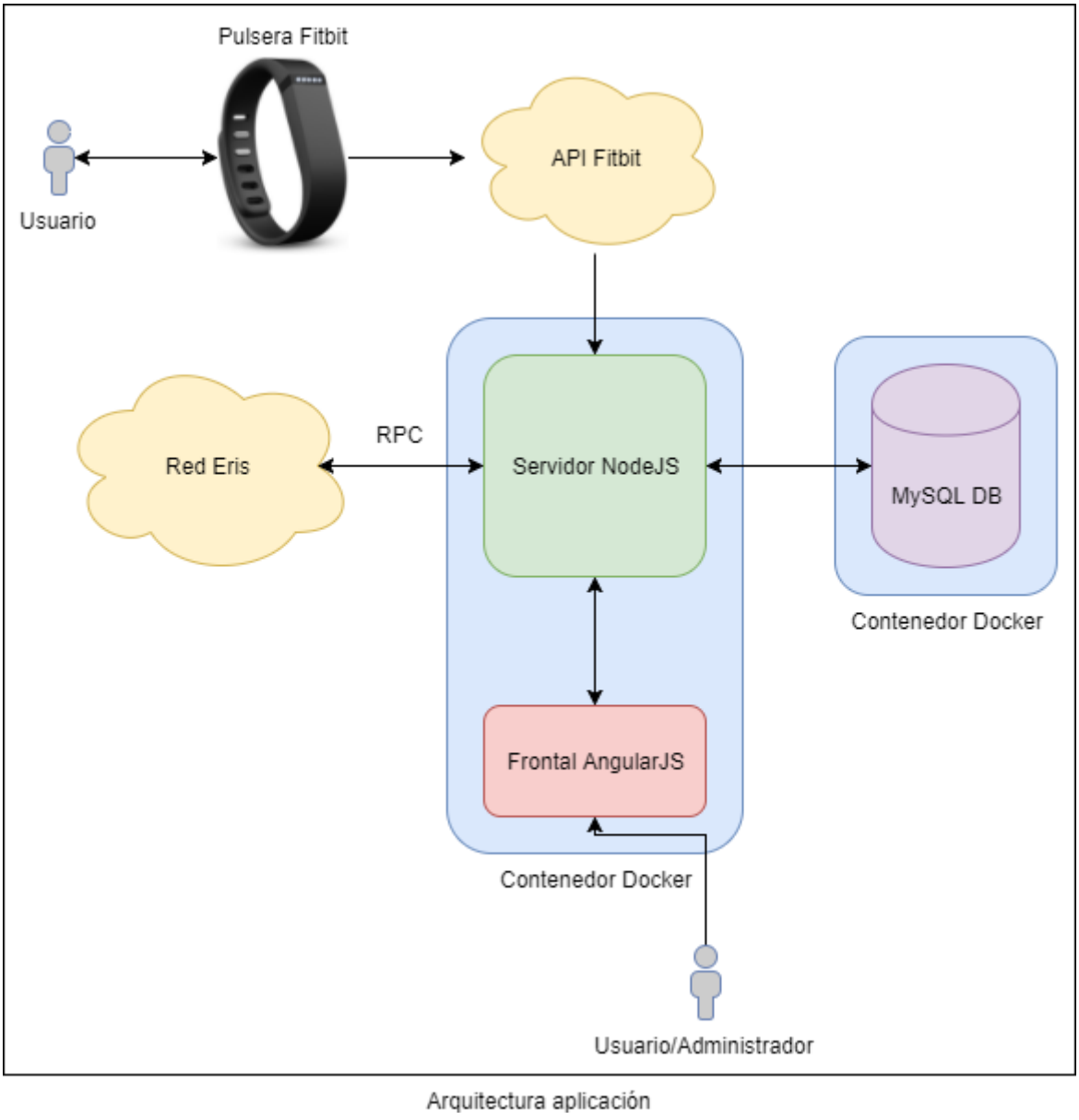


Figura 13. Arquitectura de aplicación.

En la figura 1.5 se puede observar la arquitectura de la aplicación.

Vemos como la conexión entre el servidor y la Red se realiza por medio de una conexión RPC a través de un paquete de NPM que provee la red de Eris.

4.2 Estructura del proyecto.

En esta sección se detalla la estructura por carpetas del proyecto, tanto del Docker que contiene Blockchain como del que contiene al resto de la aplicación.

4.2.1 Blockchain.

En la Figura 5.1 podemos observar la estructura del proyecto.

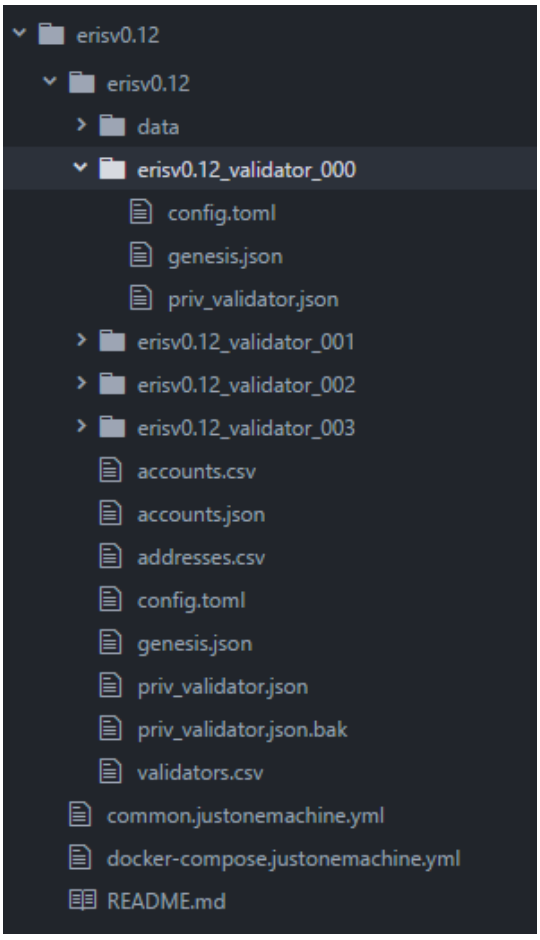


Figura 14. Estructura Blockchain.

Esta estructura que contiene la carpeta erisv0.12 del nivel 2 se genera con el wizard de Eris.

- **Data**.
  Contiene archivos internos de la red Eris, ya que esta usa por debajo otro blockchain más básico llamado tendermint.

- **Erisv0.12_validator_00X.**
  Contiene los archivos de configuración básicos de cada nodo. En ellos se especifica su configuración, el bloque génesis, que contiene las cuentas (nodos) con sus respectivas direcciones y claves públicas y el priv_validator, que mantiene las claves públicas y privadas del nodo o cuenta en cuestión junto con los datos del último bloque comprobado.
- **Resto**.
  Son archivos generados automáticamente por Eris que sirven de ejemplo para la configuración manual de otros nodos.
- **Common.justonemachine.**
  Este fichero contiene la configuración de docker común a todos los nodos.

```
node:
  image: quay.io/eris/db:0.12.0
  user: root
  command: eris-db serve --work-dir /validator/ --chain-id erisv0.12
  log_driver: "json-file"
  log_opt:
    max-file: "3"
    max-size: "300m"
  restart: unless-stopped
```

Figura 15. Fichero common.

Configuramos los logs para que roten, evitando llenar el disco con basura y activamos la opción que reinicia el nodo en caso de fallo.

- **Docker-compose.justonemachine.**
  Este fichero especifica la configuración específica de cada nodo. Como cabe de esperar se cambia el nombre de cada contenedor, se monta la carpeta de configuración de cada nodo y se cambian los puestos externos hacia los que se exponen los puertos de eris en la máquina local.

```
node0:
  container_name: validator_v0.12_0
  extends:
    file: common.justonemachine.yml
    service: node
  volumes:
    - ./erisv0.12/erisv0.12_validator_000/:/validator/
  ports:
    - "1337:1337"
    - "46656:46656"
    - "46657:46657"
```

Figura 16. Fragmento del fichero docker-compose.

4.2.2 Aplicación web.

En la siguiente figura tenemos la estructura de la aplicación web.

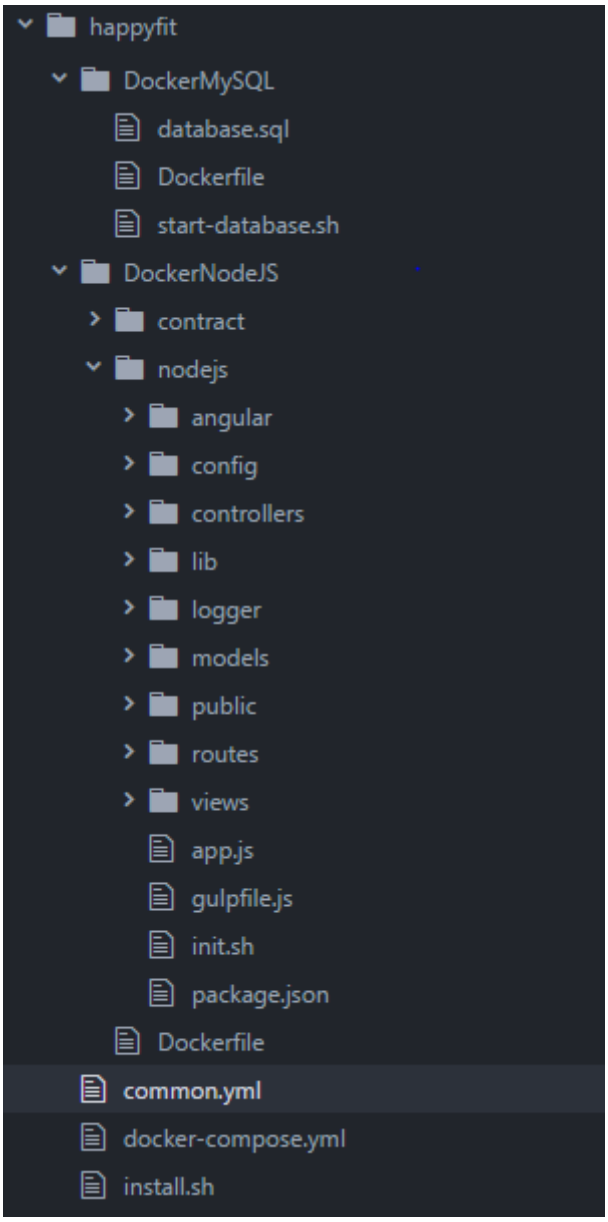


Figura 17. Estructura de la aplicación web.

- **DockerMySQL.**
  Esta carpeta contiene los ficheros de configuración necesarios para la imagen de Docker de MySQL. Database.sql inicializa la base de datos. El Dockerfile es el encargado de construir una imagen de docker con Ubuntu que instala mysql-server. Luego lanza el script start-database.sh que configura el contenedor para aceptar conexiones externas y crea las credenciales de acceso.
- **DockerNodeJS**.
  A continuación, se realiza una breve explicación de cada carpeta relevante. En el apartado de apéndices se incluirán los códigos más significativos.

a. **Contract:** Contrato desarrollado en solidity.
b. **Config:** Ficheros de configuración con constantes como direcciones IP, configuración de la ACL, autenticación y conexión a la base de datos.
c. **Controllers:** Aquí se encuentran los módulos que exponen funciones que son llamadas a través de la API HTTP que expone el servidor.
d. **Lib:** Aquí está la configuración del OAuth para que sea capaz de conectar con Fitbit.
e. **Models:** Modelos de la base de datos.
f. **Public:** Archivos que expone el servidor, tanto CSS, como imágenes, fuentes, iconos, etc.
g. **Routes:** Archivos para la configuración de la API HTTP consumida por el frontal.
h. **Views:** Vistas en EJS.
i. **App.js:** Fichero principal que levanta el servidor y pone en funcionamiento el trabajo cron que actualiza los contratos.
j. **Ficheros yml:** Contienen la configuración de Docker para levantar el contenedor con NodeJS.

4.3 Caso de uso.

En esta sección se realiza una breve explicación de cómo funcionaría la aplicación en un caso de uso real. Para ello se seguirá paso por paso lo que harían el usuario y el administrador.

El cliente recibe una llamada telefónica de la aseguradora, y esta le comunica que cuentan con un nuevo servicio que puede que sea de su agrado. Este servicio le ofrece un descuento de un 5 por ciento en su cuota de seguro de enfermedad si cumple una serie de objetivos. Se asume que el cliente tiene una pulsera Fitbit.

En este momento el administrador se loguearía en la página de inicio y accedería al panel de control, donde procederá a abrir el diálogo de creación de contrato.

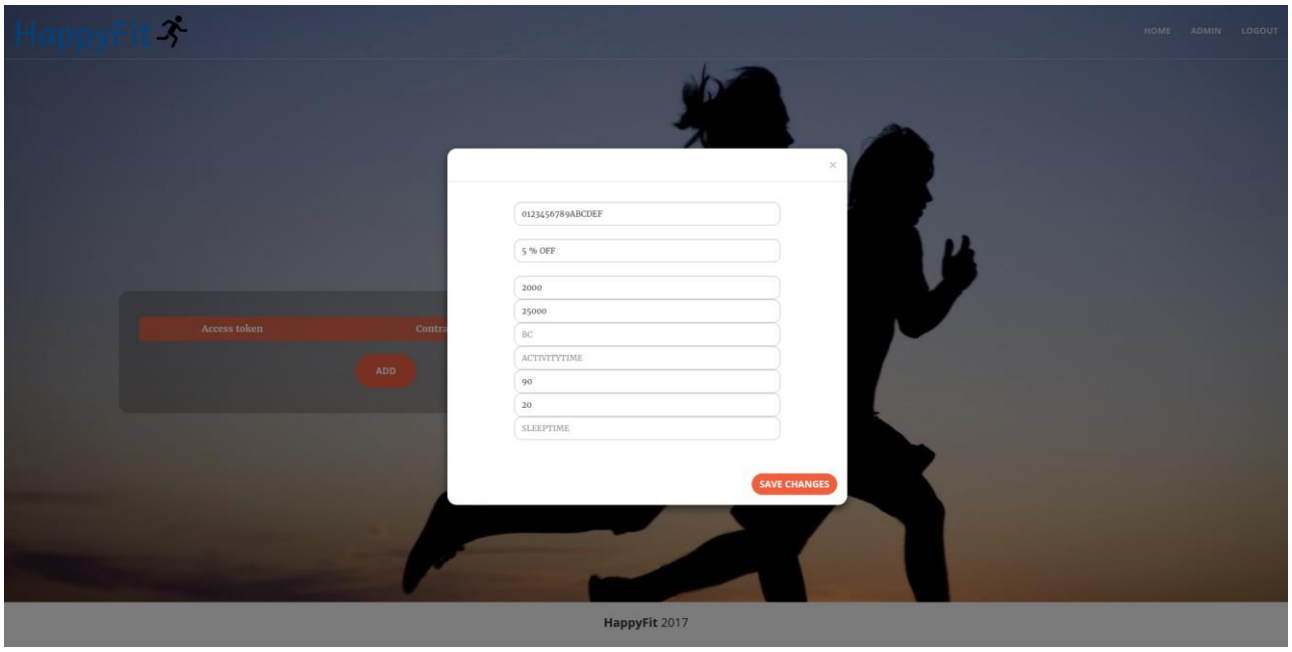


Figura 18. Creación del contrato.

Tras la negociación con el cliente, el administrador guardaría el contrato y le proporcionaría el token de acceso (0123456789ABCDEF), que este deberá usar para registrarse más adelante.

Ahora el usuario procedería a registrarse. Para ello entrará en la parte de login y será redireccionado a la página de fitbit, donde introducirá sus credenciales.

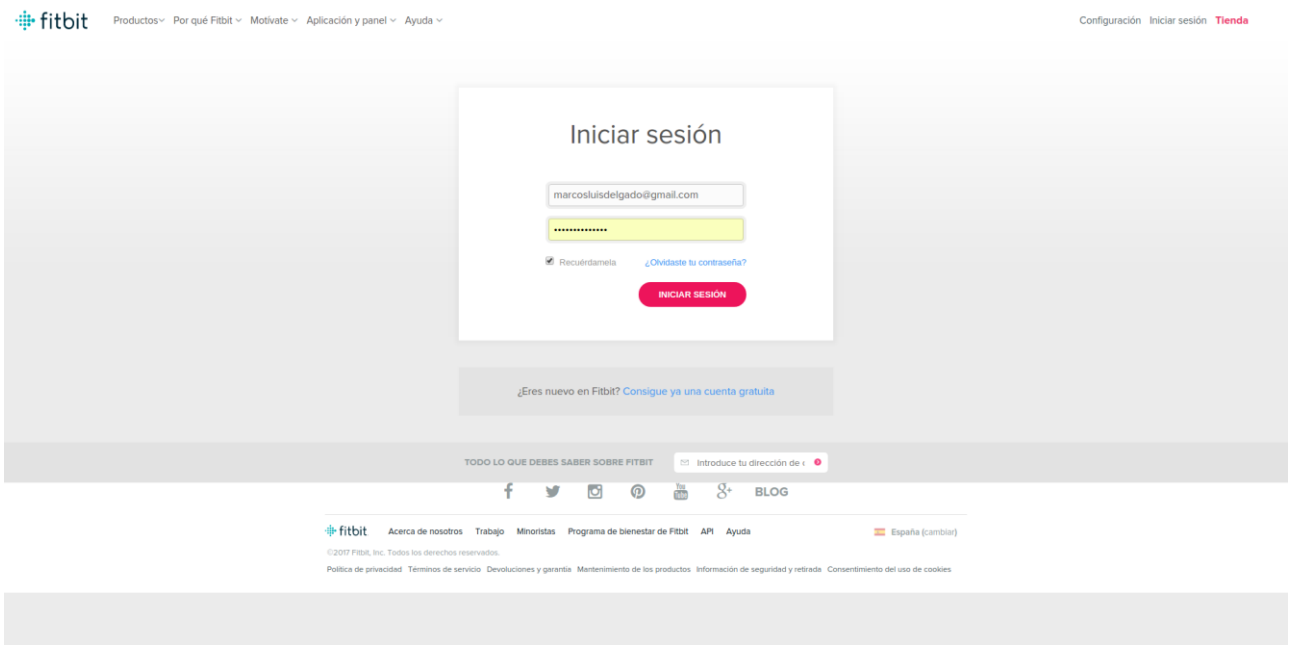


Figura 19. Introducción de credenciales en Fitbit.

Si las credenciales son correctas, el usuario será nuevamente redirigido a nuestra web, donde debe proporcionarnos el token de acceso que le facilitó el administrador y su correo para recibir las notificaciones de los objetivos alcanzados.

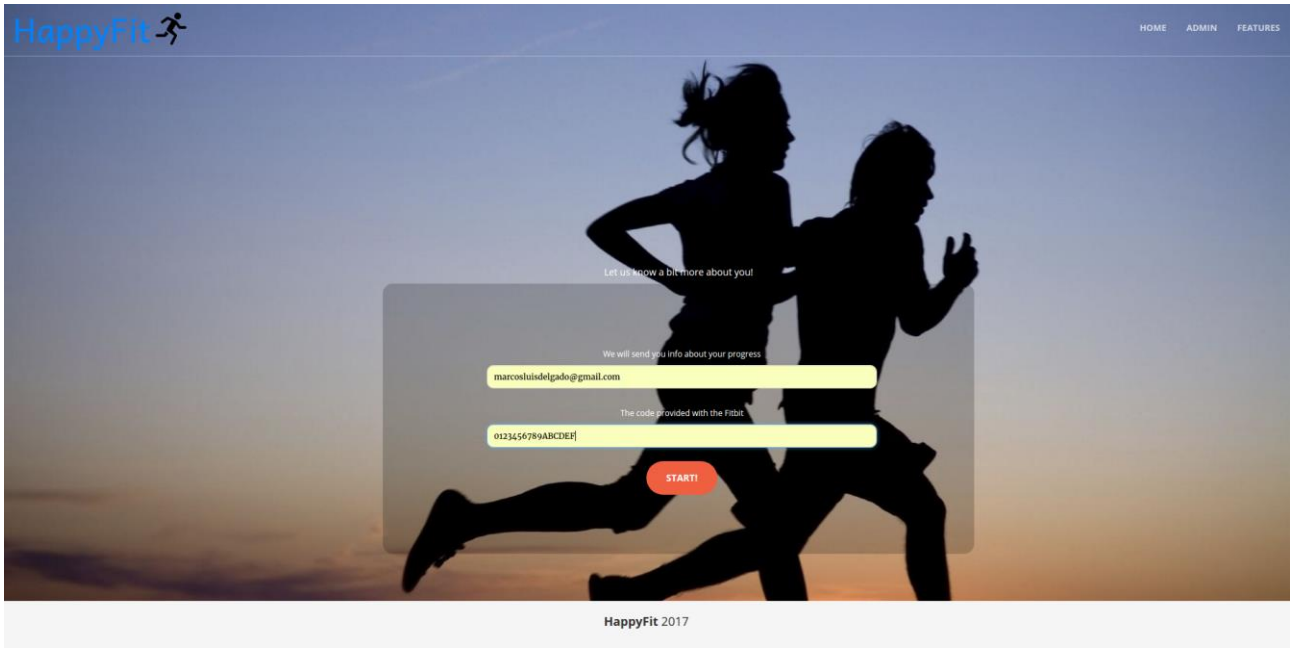


Figura 20. Introducción de token o email.

Tras ello, el usuario llegaría a la página principal, quedando registrado en la app y pudiendo acceder solamente con su cuenta de Fitbit en el futuro. Como se aprecia en la siguiente imagen, los datos están sin actualizar, ya que el servidor está programado para actualizarlos cada noche.

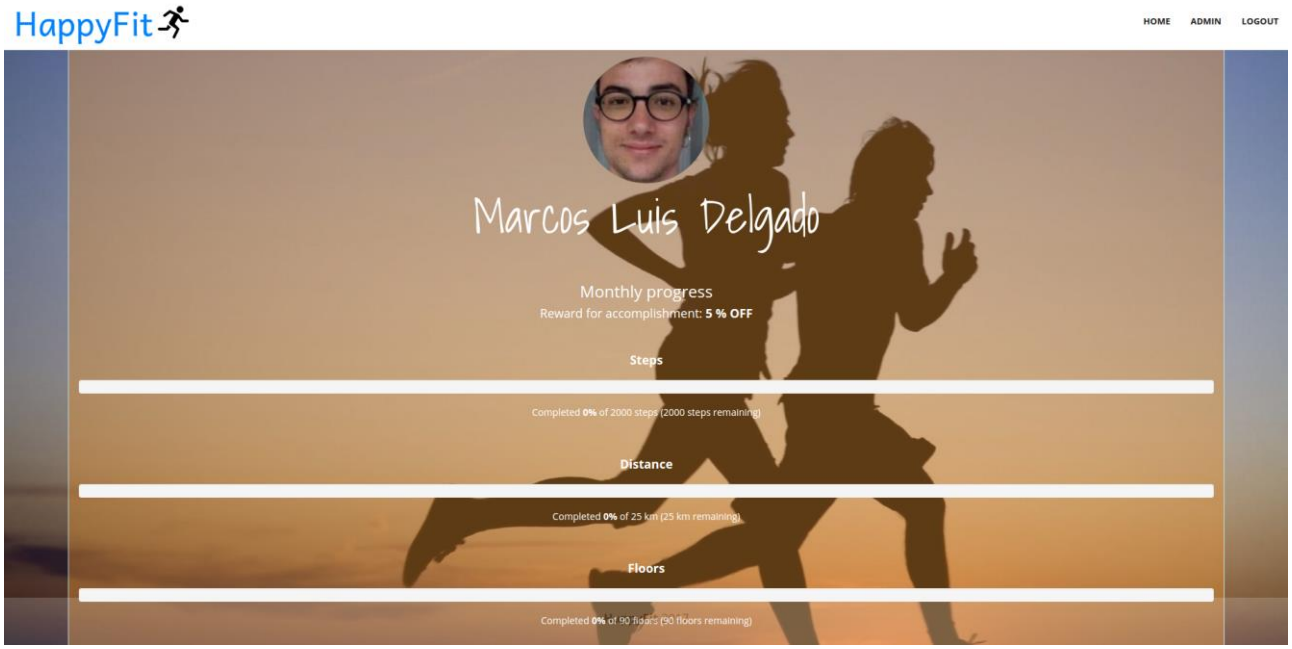


Figura 21. Página principal de usuario.

Llegados a este punto, solo quedaría esperar a que el usuario vaya progresando, como indica la figura 6.5.

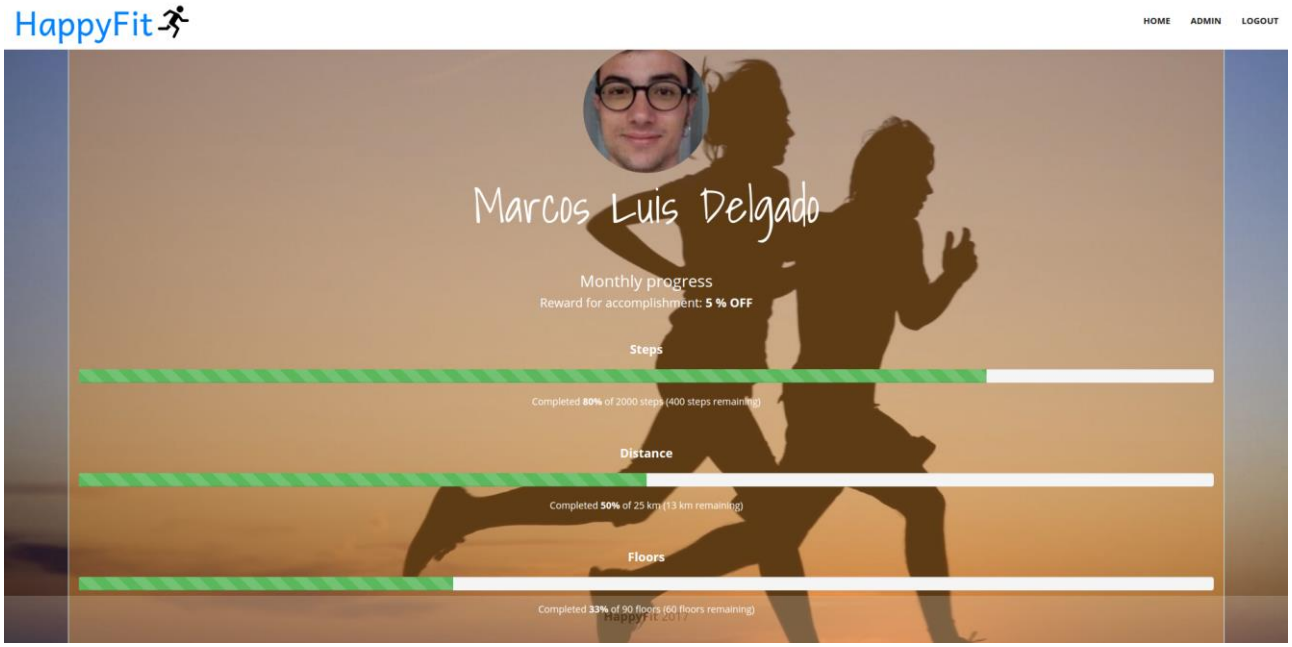


Figura 22. Página de usuario tras actualizaciones.

En el momento en el que el usuario fuera completando objetivos le irían llegando correos que le notifican de qué objetivo ha completado.

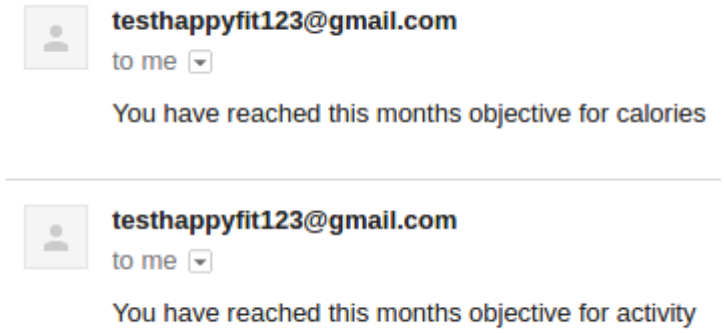


Figura 23. Correos de notificación.

Volviendo a la parte de administración, en el punto en el que el usuario asocia su cuenta de Fitbit al token el administrador puede ver en su página principal sus datos, como se aprecia en la ilustración 6.7.

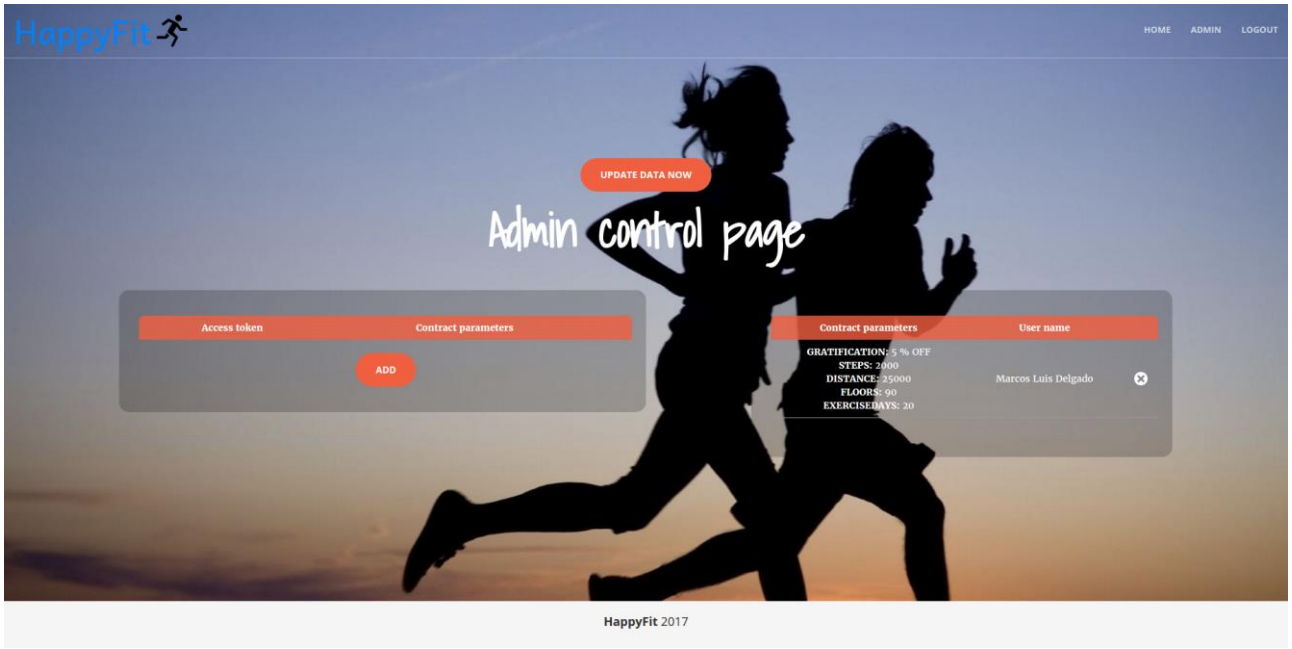


Figura 24. Datos del usuario.

Capítulo 5. Conclusiones y líneas futuras

En los siguientes apartados se describen las conclusiones y posibles líneas de trabajo futuras.

5.1 Conclusiones

- El estudio de la tecnología Blockchain ha resultado fructífera permitiendo identificar una tecnología con un potencial disruptivo enorme que seguro que dará que hablar a lo largo de los próximos años. Se ha logrado llegar a entender la misma a un nivel intermedio, así como se ha logrado desplegar varias redes en entornos privados Dockerizados.
- Meterse de lleno en una de las redes más conocidas (Eris) ha servido para comprender el principal lenguaje de programación de Smart Contracts, solidity, siendo capaces de desarrollar un contrato desde cero.
- La mayoría del trabajo se ha realizado siguiendo la propia documentación de las herramientas, una competencia crucial a la hora de trabajar y ser autónomo.
- Se han logrado resolver todos los problemas surgidos durante el desarrollo, ya sea modificando las arquitecturas que se plantearon inicialmente o solicitando ayuda a través de canales a los propios desarrolladores de las arquitecturas o a otros interesados en el tema.
- Al final se ha conseguido una aplicación funcional que cumple los objetivos planteados y que puede servir de base para futuros proyectos orientados al uso por un público de verdad, como comentaremos en las líneas de trabajo futuras.
- Como defecto cabe destacar que, a pesar de toda la seguridad y trazabilidad que nos ofrece Blockchain, su entorno está aislado, teniendo que confiar en factores externos que pueden no estar securizados. Esto será uno de los problemas que esta tecnología tendrá que resolver durante sus próximos años de vida.

5.2 Líneas futuras.

- La principal línea de trabajo futura y que no se ha realizado por falta de medios sería integrar esta solución con la propia aseguradora, que permitiría automatizar el proceso de garantizar el premio al usuario.
- Esta solución podría ser extensible a otro tipo de mecanismos de monitorización de actividad del usuario.
- Otra integración interesante sería con el móvil del usuario, pudiendo este realizar consultas y recibir notificaciones push de los objetivos que va completando.
- Sería ideal, con la próxima salida de Hyperledger versión 1.0, implementar esta red de Blockchain, ya que ofrece mayor seguridad y privacidad a los datos del cliente. Además, será capaz de lanzar los eventos a un Kafka.
- Se puede añadir la capacidad para crear otros usuarios administrador, ya que de momento este se encuentra hardcodeado.
- Una buena mejora de usabilidad sería cambiar el uso de tokens por el uso de códigos QR, que el usuario podría escanear con su móvil con facilidad.

Capítulo 6.
Summary and Conclusions

The following sections contain the conclusions and some possible future work lines that we consider interesting.

6.1 Conclusions.

- The study of the Blockchain technology has been successful, allowing us to understand one of the probably most disruptive technologies of the next years. This study has allowed us to learn to develop and deploy some of the most known networks in private and isolated dockerized environments.
- We have coded a contract from scratch using Solidity. Solidity is the most used language when developing Smart Contracts. We believe this will be useful knowledge for our future.
- Most part of the work has been done by following the tool's documentation. This is an important and desired skill that should be learned and improved through our career as software engineers.
- All the problems that appeared during the development of the project were solved in one or another way. As this is a growing technology, a big part of those problems were undocumented or unknown, so we have had to contact the platform developers and other interested users to be able to solve them.
- In the end, we managed to build a functional application that fits the objectives we wanted. On the other hand, this project can be the base for bigger and more user oriented projects in the future.
- As a negative point, we must mention that, through work and knowledge of this technology, we have found that, besides the blockchain network offers us security and traceability, the systems that work through them and it has to trust do not, so this is a problem that should be matter of interest during the following years.

6.2 Future work lines.

- The main work line that we find, and that we could not develop because of the lack of contacts, is integrating this solution with the insurance company systems. This would allow the automatization of the process of granting the gratification to the user.
- This project could be adapted to other kinds of activity monitoring tools that are cheaper and available to more users.
- Another interesting point would be making a mobile app or extension for an existent app. Nowadays most users prefer having their data accessible from anywhere with their smartphones.
- We have also thought of changing the underlying blockchain system with the release of the new version of Hyperledger, 1.0. This version provides data stored on the network more security and privacy. Furthermore, it will be allowed to send data to Apache Kafka.
- We should also implement the feature of creating, editing or deleting new administrator users, as the only administrator user is hardcoded at the moment.
- A good usability change would be using QR Codes instead of access tokens. This way the user can simply scan the code and register on our app.

Capítulo 7. Presupuesto

7.1 Presupuesto por horas de trabajo.

Actividad	Tiempo empleado (horas)	Precio por hora (€)	Coste (€)
Investigación sobre las redes Blockchain	60	8	480
Investigación sobre los Smart Contracts	20	8	160
Investigación sobre las plataformas	30	8	240
Investigación sobre la arquitectura	20	8	160
Desarrollo y despliegue de la red	60	15	900
Desarrollo del Smart Contract	20	15	300
Desarrollo y despliegue de la aplicación web	150	15	2250
Integración de los componentes	25	15	375
Total	385	-	4865

Tabla 1. Presupuesto horas de trabajo.

7.2 Presupuesto hardware.

Dispositivo	Descripción	Coste (€)
Fitbit flex 2	Dispositivo Fitbit utilizado para la monitorización del usuario.	80

Tabla 2. Presupuesto dispositivos hardware.

Apéndice A. Enlaces a códigos interesantes.

A.1. Smart Contract implementado.

A.2. Algoritmo para actualizar datos del contrato.

A.3. Módulo para crear y llamar contratos.

A.4. Módulo para escuchar los eventos que produce Eris.

Bibliografía.

[1] Lin William Con and Zhiguo He. Blockchain disruption and Smart Contracts.

https://poseidon01.ssrn.com/delivery.php?ID=049017110001011018011006066028028127017040064087064044065084030074104074122118015077026023006005053121011073117031087028101014060021056026068027025086014106090019088084051021116122021123113081107081125112003099081106108125113004066085065108126001119095&EXT=pdf

[2] Wikipedia. Blockchain.

https://en.wikipedia.org/wiki/Blockchain

[3] Wikipedia. Bitcoin.

https://en.wikipedia.org/wiki/Bitcoin

[4] What is Blockchain technology.

https://blockgeeks.com/guides/what-is-blockchain-technology/

[5] Breve historia de Blockchain y el largo futuro que nos espera juntos.

https://www.hbr.es/tecnolog/497/breve-historia-de-blockchain-y-del-largo-futuro-que-nos-espera-juntos

[6] A (short) guide for Blockchain consensus protocols.

https://www.coindesk.com/short-guide-blockchain-consensus-protocols/?lipi=urn%3Ali%3Apage%3Ad_flagship3_profile_view_base_recent_activity_details_shares%3BrlFuHB3fT5W1%2FjX4G0eTAA%3D%3D

[7] Monax Doc.

https://monax.io/docs/documentation/

[8] Ethereum Doc.

http://www.ethdocs.org/en/latest/

[9] Hyperledger Doc.

https://hyperledger-fabric.readthedocs.io/en/latest/