Universidad
de La Laguna

Escuela Superior de
Ingeniería y Tecnología
Sección de Ingeniería Informática

Trabajo de Fin de Grado

Desarrollo e Implementación de Aplicaciones Seguras en Medicina

Development and Implementation of Secure Applications in Medicine

José Saúl Giffuni Becerra

La Laguna, 4 de julio de 2016

D. **Pino Caballero Gil**, con D.N.I. 45534310-Z profesora Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **Alexandra Rivero García**, con D.N.I. 78646309-V investigadora de FPI adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

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

Que la presente memoria titulada:

"Desarrollo e Implementación de Aplicaciones Seguras en Medicina"

ha sido realizada bajo su dirección por D. **José Saúl Giffuni Becerra**,

con N.I.E. X-3834360-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de julio de 2016

Agradecimientos

En primer lugar al Señor, que me ha permitido llegar hasta aquí y que me ha provisto de una familia maravillosa y de todo aquello que pudiera necesitar.

A mi familia, en especial a mis padres por su ejemplo y dedicación y también a mi hermana por enseñarme que la perseverancia logra lo que la dicha no alcanza.

A Betsi, por su amor y paciencia en cada etapa.

A Luis Gabriel, mi profesor de tecnología en La Gomera y amigo, a quien respeto y estimo.

A Pino y Alexandra, mi tutora y cotutora respectivamente, por su esfuerzo en que este trabajo saliera adelante.

A todos lo profesores que he tenido el privilegio de conocer y aprender de ellos.

Licencia

Resumen

En los últimos años la tecnología informática se ha expandido a multitud de sectores dispares. Uno de los ámbitos que ha adoptado este tipo de tecnología de forma más natural es el concerniente a la medicina. Muchos de los avances tecnológicos tienen lugar en esta rama, razón por la cual se puede considerar que está en la cresta de innovación del sector informático. Esto se debe a que es un campo vital para la sociedad, y por tanto cualquier avance tiene una alta repercusión. Por ello se hace necesario que las tecnologías que se implementen en este sector sean totalmente seguras y confiables, debido a que afecta de forma directa al propio ser humano.

Teniendo en cuenta el panorama descrito, en este proyecto se ha desarrollado una aplicación móvil segura que permite interactuar a un médico con su paciente a través de preguntas aleatorias de distintos tipos, y que hace uso de un smartwatch para obtener diferentes datos médicos como son: la actividad diaria y el ritmo cardiaco. A su vez, se ha planificado y se ha llevado a cabo, el desarrollo de un servidor que permite almacenar los datos y provee de un entorno web para el acceso de los médicos a la información almacenada.

Palabras clave: Android, eHealth, app, smartwatch, servidor, AES, ECDSA.

Abstract

In recent years, computer technology has expanded into many diverse sectors. One of the areas that has adopted such technology is medicine. Many of the technological advances, that can be considered are on the crest of innovation in the computer industry. This is because it is a field vital to society, and therefore any progress has a high impact. Therefore it is necessary that the technologies to be implemented in this sector must be completely safe and reliable, because it directly affects the life of the human being.

Given the described scenario in this project a secure mobile application that allows a doctor to interact with her patient through random questions of different types, uses a smartwatch for obtain different medical data, such as: the daily activity and heart rate. In turn, this application was planned and carried out with the development of a server that allows storing data and providing a web environment for physicians to access stored information.

Keywords: Android, eHealth, app, smartwatch, server, AES, ECDSA.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción al trabajo

1.1 Motivación

La tecnología evoluciona rápidamente cada día, los aparatos voluminosos del pasado que recibían el nombre de “Teléfonos móviles”, han dado paso a herramientas tecnológicas de apenas unos milímetros de espesor y con capacidades casi ilimitadas. Los relojes han evolucionado desde los relojes de sol, creados en el antiguo Egipto, pasando por el primer reloj mecánico de Richard de Wallingford, en el siglo XIV, hasta llegar a los relojes inteligentes, también denominados, smartwatches. Estos dispositivos en la actualidad no solo son capaces de dar la hora, sino que traen consigo un amplio abanico de utilidades. Ya no son aparatos que solo sirven para llamar o para conocer la hora. Teniendo esto en mente, ¿por qué no utilizar esas nuevas características que nos ofrecen estas tecnologías y aplicarlas en una de las relaciones más importantes de la vida? la relación médico – paciente.

Las posibilidades y los beneficios de usar las nuevas herramientas tecnológicas en la medicina aportan grandes posibilidades en áreas de investigación y desarrollo. Muestra de ello es el gran número de aplicaciones móviles que existen y que tienen como fin, desde ayudarnos a perder peso hasta llevar un control de nuestra actividad física, entre otros. Sin embargo, a pesar de la cantidad de aplicaciones existentes, son pocas las aplicaciones que permiten a un médico acceder a toda esa gran cantidad de datos e información que los dispositivos móviles y wearables son capaces de obtener, y de permitir que sea un especialista, el que analice los datos y que le sirva para diagnosticar hábitos no saludables, enfermedades cognitivas e incluso cardiovasculares. Es en este apartado, donde el presente trabajo se centra.

Tras la finalización del tercer curso del grado en Ingeniería Informática,

este trabajo ha supuesto una oportunidad de aplicar todo el conocimiento transmitido por parte del profesorado poco a poco y con bastante esfuerzo, en algo que fuera útil para la sociedad y a su vez me interesante. La oportunidad de hacerlo con el apoyo del grupo CryptULL, y que se tratase de una herramienta que permita ayudar a prevenir hábitos no saludables y en la detección de diferentes trastornos y enfermedades, fueron alicientes más que suficientes para intentarlo y probar que era posible hacerlo.

1.2 Objetivos

El presente proyecto parte de un objetivo principal: disminuir la distancia existente entre el médico y el paciente, para ayudar al diagnóstico precoz, y al seguimiento de enfermos con enfermedades cognitivas y/o afecciones cardiacas. Para poder cumplir este propósito principal, se ha tenido que establecer una serie de objetivos.

A priori, se estableció el uso de dispositivos móviles junto con dispositivos wearables, así como el uso de un servidor que hiciese las veces de controlador del sistema, de almacenaje de toda la información y de plataforma web. Cada uno de estos componentes tiene sus objetivos específicos y un objetivo común. Este objetivo comú no es otro que el de la seguridad, siendo este un requisito indispensable dado el tipo de información con la que se trabaja, información personal y de la salud. Es por ello que la comunicación de este tipo de información debe ser cifrada en todo momento.

Los objetivos específicos los podemos dividir entre aquellos que van destinados a las herramientas que serán utilizadas por los pacientes (el dispositivo móvil y el reloj) y a la herramienta que será utilizada por el médico (la plataforma web).

Objetivos de la aplicación móvil y el reloj

El principal objetivo específico de este apartado es que se trate de una herramienta personal y sencilla de utilizar, no pudiendo ser su interfaz compleja sino cómoda e intuitiva, para que pueda ser usada por la mayor parte de la población que utiliza teléfonos inteligentes.

Otro objetivo importante es poder realizar en la aplicación los test de memoria asignados por el médico y que los resultados puedan ser enviados de vuelta para su posterior análisis, es decir, poder tener un feedback de todos esos datos obtenidos.

Por último, otro objetivo específico es la recogida de datos de interés

médico, tales como el ritmo cardiaco del paciente en un determinado momento del día y un registro de la actividad física diaria.

Objetivos de la plataforma web

El objetivo principal de la plataforma web es permitir el acceso de los médicos a los datos y tests enviados desde la aplicación móvil de sus respectivos pacientes. Es necesario para cumplir este propósito:

- Acceder al perfil de cada paciente, permitiendo mostrar al médico, los datos personales y médicos más relevantes, además, de los datos que se han ido recopilando, junto con los resultados de los test realizados.
- Gestionar el acceso y almacenamiento de los test resueltos, así como el historial médico de los pacientes.
- Visualizar gráficos en donde se pueda observar la evolución de los datos cardiacos diarios del paciente.
- Asignar y visualizar los test de cada paciente.
- Visualizar de una manera cómoda los demás datos que se pudieran obtener.

1.3 Fases del desarrollo

El presente proyecto ha sido desarrollado en 6 fases debido a la gran cantidad de tecnologías utilizadas, al aprendizaje de las mismas y a mejoras que se han ido proponiendo e intentando implantar a medida que iba avanzando el desarrollo.

En la primera fase del proyecto se ha llevado a cabo una investigación sobre los antecedentes y el estado actual del tema de las aplicaciones médicas, en especial aquellas que estuviesen relacionadas con los test de memoria para personas con Alzheimer o trastornos cognitivos. Además, se ha buscado información sobre la enfermedad y los test que se utilizan para ayudar a su diagnóstico. A partir de la información obtenida, se modelaron las preguntas y las posibles tipos de respuesta. Se definió por un lado, un formato usando ficheros JSON para poder importar los cuestionarios, los cuales son listas de preguntas que el médico envía al paciente, y otro formato para exportar las respuestas.

En segundo lugar, se desarrolló e implementó en Java la lógica funcional de los test para los dispositivos móviles Android, sistema elegido debido a su

gran expansión en el mercado. En este paso se desarrolló además una interfaz única (fragments) para cada tipo de pregunta. También se diseñó y se programó la transformación de los formularios JSON en objetos definidos mediante clases para su posterior utilización.

Una vez lograda la fase anterior, el tercer paso fue empezar a trabajar con el servidor, la base de datos y el entorno visual de la aplicación web. En este punto, se decidió apostar por Node-JS, express, y el generador de la estructura Modelo Vista Controlador (MVC), Yeoman para montar el servidor.

En el lado de la base de datos se escogió Mongo-DB como motor de la base de datos. Esta decisión supuso al comienzo un pequeño retraso debido a que se trata de un modelo de base de datos no relacional y que trabaja con colecciones y documentos en lugar de tablas y filas, una tecnología relativamente moderna. No obstante, el que la base de datos trabajase con documentos JSON supuso una enorme ventaja tanto a la hora de almacenar los test, como a la de guardar los test resueltos y los datos del reloj de los usuarios. También, ayudó a ganar tiempo en la comunicación entre el servidor y el móvil, ya que uno de los formatos estándar para la transmisión de datos en Internet es, el mencionado anteriormente formato JSON, por lo que no hacía falta una previa conversión de los resultados de las consultas ni del almacenamiento.

Posteriormente, la cuarta fase consistió en lograr un correcto funcionamiento de los test, tanto a nivel de servidor como de móvil. Es decir, se hizo posible la comunicación de la aplicación con el servidor para poder descargar los test que el médico le hubiese asignado al paciente, y enviar la respuesta con el test resuelto, una vez el usuario lo haya finalizado. En el lado del servidor, consistió en definir la forma en la que el médico asigna los tests, almacena las respuestas y visualiza los datos de los tests resueltos por el paciente.

En la penúltima fase, se procedió a la obtención de diferentes datos del reloj inteligente. Por un lado, se obtiene la frecuencia cardiaca en un periodo determinado de tiempo. Por otro lado,se almacena la actividad física que el usuario realiza durante el día. Una vez obtenidos los datos, se implementó el método de envío de los mismos al servidor y su visualización en el entorno web.

Por último, se procedió a la incorporación de seguridad en la app para cifrar los datos clínicos obtenidos del smartwatch (pulsaciones y actividad física) así como los test resueltos. Para ello, se utilizó un cifrado simétrico AES de 256 bits en modo CBC y la firma digital basada en curvas elípticas.

1.4 Estructura de la memoria

La presente memoria se encuentra estructurada en los siguientes 10 capítulos:

- Capítulo 1: Introducción del proyecto, en donde se exponen los motivos que impulsaron elaborar el proyecto, los objetivos y las distintas fases del desarrollo hasta llegar a la finalización y presentación del mismo.
- Capítulo 2: Introducción a la herramienta, explicando el funcionamiento de cada componente y se introducen conceptos de la misma.
- Capítulo 3: En este capítulo se explican las tecnologías utilizadas. Se realiza un análisis de cada una de ellas y se justifica la utilización de las mismas en el proyecto. Así mismo, se explican sus características especiales y se dedica un apartado a profundizar en la seguridad.
- Capítulo 4: Este capítulo contiene una explicación detallada del servidor, rutas, modelos, controladores, middlewares utilizados y conexión con la base de datos. Se profundiza en los documentos y las relaciones entre las colecciones, además de enseñar los formatos JSON de los test y de los datos.
- Capítulo 5: App móvil y el Smart-Watch, definición detallada de la aplicación móvil, así como una breve explicación del sistema lógico de los tests. Profundización en la recolección de los datos provenientes del reloj, así como el acceso a los sensores del dispositivo que suministran la información.
- Capítulo 6: Se encarga de mostrar el funcionamiento conjunto de todos los componentes: el servidor, el móvil y el reloj.
- Capítulo 7: Abarca desde la importancia de la implementación de la seguridad en este tipo de proyectos, así como la implantación en el proyecto.
- Capítulo 8: Se analiza el presupuesto total del proyecto.
- Capítulos 9 y 10: Se recogen las conclusiones, ideas, y posibles incorporaciones futuras a la herramienta. El capítulo 9 se encuentra en español y el 10 en inglés.

Capítulo 2
Introducción a la herramienta

2.1 Definición

El presente proyecto trata de cubrir el vacío existente entre tres distintos tipo de aplicaciones: las aplicaciones de seguimiento personal, las aplicaciones de juegos de memoria y las aplicaciones médicas. Para conseguirlo, se ha extraído lo mejor de cada tipo y se ha entregado en una sola, consiguiendo así, no solo una aplicación más completa, sino una herramienta completa para el médico.

El proyecto consta de tres componentes principales: una plataforma web (que realiza las funciones de servidor de la base de datos y servidor web), un teléfono móvil y un reloj inteligente. Su funcionamiento es sencillo y se basa en que los médicos disponen de un sitio web donde poder llevar un seguimiento de los pacientes, permitiéndoles no solo ver el perfil de cada uno, sino asignarles diferentes test de memoria y tener a la mano (de una manera gráfica) diferentes datos de interés como son: la frecuencia cardiaca y la actividad diaria física. Una vez que el médico haya dado de alta a un paciente (en la plataforma), podrá asignarle cualquiera de los test disponibles. Una lista con los tests pendientes, aparecerá en la app móvil del paciente. Este podrá seleccionarlos uno por uno para completarlos. La aplicación descargará el test seleccionado y procederá a mostrarlo en pantalla para su realización. Una vez finalizado el mismo, se creará un formulario con las preguntas y las respuestas dadas por el usuario, así como el número de aciertos y errores. Este formulario se enviará inmediatamente al servidor para estar disponible en el historial del paciente. Además de esta función, la app móvil estará en constante comunicación con el smartwatch del paciente, el cual le proveerá de dos datos de interés médico: la frecuencia cardiaca y la actividad física diaria. Para que se pueda llevar a cabo este paso, merece la pena aclarar, que no todos los smartwatch que

existen actualmente en el mercado traen consigo estos tipos de sensores. Es por esto que hay que investigar y leer detenidamente la descripción del producto a utilizar. Para este proyecto, el reloj escogido fue un Motorola Moto 360 v2 Sport que trae incorporado un pulsómetro óptico PPG (Fotopletismógrafo, por sus siglas en Inglés). Estos datos son enviados por el reloj al teléfono móvil mediante la tecnología Bluetooth. El teléfono se encarga de procesarlos y enviarlos en formato JSON al servidor, el cual los almacena en la base de datos y los pone a disposición del personal médico, tal y como se representa en la Figura 2.1.

Figura 2.1: Esquema de la herramienta.

2.2 Conceptualización

Para poder llegar a un correcto entendimiento del proyecto y de la presente memoria, es necesario definir y conceptualizar diferentes términos que aparecerán a lo largo de todo el documento. En este apartado se

comenzará con la definición de las partes que componen este sistema propuesto.

En primer lugar, haremos uso de un servidor, el cual jugará dos diferentes dentro del proyecto: plataforma web y base de datos. La plataforma web consiste en una herramienta de acceso único y exclusivo para el personal médico. En ella se lleva un control de los pacientes, los cuales están asignados por médico. Esta plataforma permite llevar un control centralizado de toda la herramienta. Desde la plataforma web se darán de alta a los médicos y también permitirá dar de alta a los pacientes. El médico que dé de alta a un paciente, en la plataforma, pasará a ser considerado el médico personal del paciente. Mediante el uso de la plataforma, podrá, gestionar el perfil de cada uno de sus pacientes, añadir tests, asignar los tests disponibles en la base de datos, obtener y visualizar los parámetros recogidos por el reloj de cada individuo y estudiar las respuestas de los test realizados. La segunda función del servidor será la de alojar el sistema gestor de la base de datos, que permitirá almacenar todos los datos identificativos de los médicos y pacientes de la aplicación, además de todos los datos recabados y los test generados.

En segundo lugar, contaremos con una aplicación móvil y una aplicación en el smartwatch. Estas serán instaladas en los dispositivo oportunos de cada paciente. En la aplicación móvil se podrán realizar los tests mostrando por pantalla una lista de los mismos y se gestionará todo el proceso que conlleva desde la descarga, hasta el envío de los resultados. Por otro lado, la aplicación para el reloj inteligente se encargará de acceder a los sensores disponibles del mismo y de obtener los parámetros ya anteriormente descritos.

A continuación, después de haber definido las partes de la herramienta, se explicarán los dos tipos de usuarios que aparecen en el proyecto, y otros conceptos clave:

- Médico: Se trata de un perfil para los doctores que usarán la herramienta web. Ellos serán los únicos usuarios capaces de visualizar la información de los pacientes y de asignar los test que se encuentran en la base de datos.
- Paciente: Es el usuario para el que se destina la aplicación móvil. Son el objeto de estudio y la parte fundamental de todo el ecosistema. Utilizarán la aplicación para realizar los test y para recoger la información médica que luego será enviada al servidor. No tienen acceso al sitio web.

- Quiz o Test: Son los cuestionarios que se encuentran disponibles en la aplicación y que permiten ser asignados a los pacientes. Están compuestos por una serie de preguntas con sus respectivas posibles respuestas, entre ellas la correcta.
- Pregunta: Se trata de un enunciado junto con sus posibles respuestas. Existen 4 tipos de preguntas en la herramienta: preguntas de selección simple (sólo una es la respuesta correcta), de selección múltiple (una o más son correctas), preguntas de tipo numérico (el usuario introduce un valor numérico) y las preguntas visuales (aparecen imágenes y se debe elegir aquella que es correcta).

2.3 Aplicaciones similares

Son pocas las aplicaciones similares a la de este proyecto. No obstante, existen ejemplos que tienen objetivos similares.

- Emotion ECG, desarrollado por Mega Electronics, es una aplicación disponible para IOS y Windows Phone que permite monitorear el ritmo cardiaco por medio de sensores especializados (Figura 2.2). El objetivo es detectar cualquier problema cardiaco y avisar a emergencias en el caso de que esto suceda. Su uso ha sido aprobado por la Oficina de Drogas y Alimentos de Estados Unidos (FDA).

Figura 2.2: Emotion ECG.

- Einstein Math Academy. Se trata de una herramienta fácil de usar que por medio de rápidos cálculos mentales ayuda a los pacientes con Alzheimer.

Capítulo 3
Tecnologías y componentes

3.1 Tecnologías

En el presente apartado se describen las tecnologías utilizadas divididas en dos bloques: las del servidor y las de la aplicación móvil.

3.1.1 Tecnologías usadas en el servidor

Node.js

Node es un intérprete Javascript del lado del servidor que utiliza el motor V8 JavaScript [1] que Google usa en su navegador Google Chrome. Este interprete está escrito en C++ lo que lo convierte en rápido y permite incorporarlo a cualquier aplicación que se desee. La principal característica de este programa servidor es la forma en que maneja las conexión. En otros programas servidor, como por ejemplo Apache, cada conexión con el servidor se traduce en la creación de un nuevo hilo. Este proceso consume memoria, según un artículo publicado en IBM developersWorks [2]. Cada conexión viene acompañado de 2MB lo que se traduce en un máximo teórico de 4000 usuarios para un servidor de 8GB. En lugar de trabajar con hilos por cada conexión, Node trabaja con eventos, evitando así un gran uso de la memoria permitiendo que la escalabilidad de la herramienta no sea un problema.

Otra característica que le otorga un gran poder para expandir funcionalidades es la inclusión de módulos. Los módulos son paquetes que se integran en el proyecto y que traen consigo características ya implementadas, como puede ser: express, que proporciona un conjunto sólido de recursos para aplicaciones web, de la cual hablaremos más adelante. Para poder instalar y administrar cualquier módulo, existe el Node Package Module (npm) que es el controlador de paquetes de Node por

defecto. Con él se pueden instalar los paquetes sin preocuparse de las dependencias. Es tan fácil como:

```
$ npm install nombre_del_módulo
```

Express

Express **[3]** es el módulo principal que usaremos para la infraestructura de la aplicación web. Proporciona una fina capa de características de aplicaciones web básicas, además de varios métodos, utilidades HTTP, y middlewares que permiten crear una API sólida y perfectamente funcional de forma rápida. Entre sus características ofrece la generación de las rutas y peticiones HTTP mediante URL (GET, POST, PUT…). Ayuda en la integración de distintos motores de plantilla (Jade, Ejs…) y es compatible con la mayoría de middlewares.

Yeoman

Yeoman **[4]** es un encapsulador de herramientas para Node.js que facilitan el desarrollo de las aplicaciones. Su principal ventaja es que es gratuita y con tan solo configurar una serie de parámetros nos genera el esqueleto de la aplicación. Permite añadir distintos generadores y se pueden usar para crear el esqueleto que más nos interese (no estando limitada a una sola tendencia). Para este proyecto, se utilizó el Generator express [5], que permite estructurar el proyecto como un Modelo Vista Controlador (MVC). Por último, automatiza la construcción y puesta en marcha del servidor, gracias a Gulp [6] o Grunt [7], y permite gestionar las dependencias del back-end con npm, y las del front end con Bower[8].

MongoDB

MongoDB **[9]** es una base de datos no relacional y NoSQL, que como bien su propio nombre indica, no usan el lenguaje SQL para sus consultas. También se diferencian de las bases de datos convencionales en tres principales aspectos: escalabilidad horizontal, alta velocidad en las búsquedas y no tienen un esquema (lo que no significa que no se pueda simular). Por otra parte, Mongo-DB almacena los datos en documentos y los agrupa por colecciones, no siendo necesario que dos documentos almacenados en una misma colección sigan una misma estructura (ver en Figura 3.1).

```
db.familia.insert(
      [
        {id_:2, name:"Claudia", parentesco:"Madre", nacionalidad:"Colombiana"},
        {id_:3, name:"Joyce", apellido:"Giffuni", parentesco:"Hermana"}
      ])
```

Figura 3.1: Inserción de documentos en la colección de familia

3.1.2 Tecnologías usadas en el dispositivo móvil

Android

En la parte de la aplicación móvil, se optó por realizar una aplicación nativa para el sistema operativo Android. Los motivos por los cuales se escogió este sistema operativo en lugar de cualquier otro, fueron varios, destacando principalmente de que se trata de un proyecto con un núcleo libre, gratuito y multiplataforma, que consta de una amplia comunidad de desarrolladores y que está presente en un 93,9% (Figura 3.2) de dispositivos del mercado español, según la agencia Kantar [10]. De esta forma, nuestra aplicación va a llegar a la mayor parte de la población que utiliza un teléfono inteligente en España.

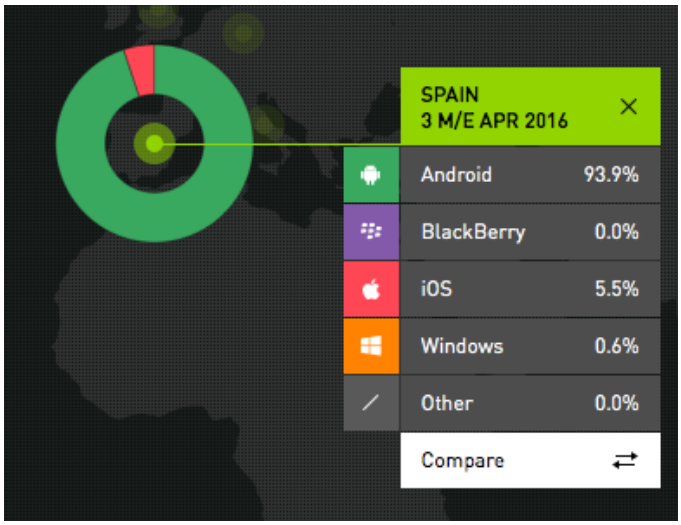

Figura 3.2: Cuota de mercado de los SO móviles.

Otros motivos que nos llevaron a decantarnos por este SO, fueron: la madurez de su sistema operativo para relojes y que la curva de aprendizaje de Android es la misma para ambos tipos de dispositivos (móvil y reloj). Hay que aclarar que el lenguaje de programación del sistema operativo Android es Java.

Por último, la versión a partir de la cual se puede usar la aplicación para los dispositivos móviles es la 4.4 Kit-Kat, (siendo la misma para el reloj, en su versión Wear), alcanzando una cuota del 73,9 % dentro del mercado Android en dispositivos móviles (Figura 3.3) y un 100% del mercado de relojes Android.

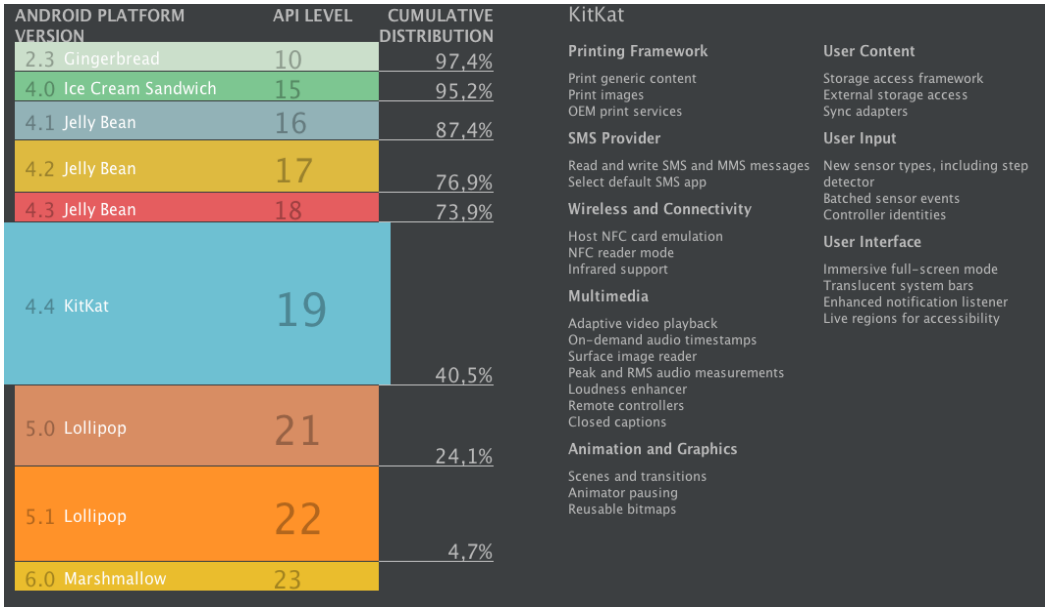

Figura 3.3: Alcance dentro del mundo Android

3.2 Componentes

Los 3 componentes hardware que se detallan a continuación mediante tablas fueron los que se utilizaron para llevar a cabo el proyecto.

Servidor

Marca	Apple
Modelo	MacBook Pro Early 2011.
Procesador	I7-2720QM
Memoria Ram	4 GB
OS	OS X 10.11.5

Tabla 3.1: Detalles del servidor

Móvil

Marca	Samsung
Modelo	I9105P
Procesador	Dual-core 1.2 GHz
Memoria Ram	1 GB
OS	Android Lollipop 5.1

Tabla 3.2: Detalles del móvil

Smartwatch

Marca	Motorola
Modelo	Moto 360 Sport
Procesador	Snapdragon 400 (4x 1.2 GHz)
Memoria Ram	512 Mb
OS	Android Wear

Tabla 3.3: Detalles del smartwatch

3.3 Seguridad

Al ser una herramienta que trata con datos médicos y personales de los pacientes, se hace necesaria la confidencialidad de los datos obtenidos y para ello se han implementado diferentes medidas de seguridad para transmitir a los usuarios finales una mayor confianza en el uso de la aplicación y del entorno web.

Los principales riesgos de seguridad cubiertos en este proyecto han sido:

I. Captura de los datos: los datos se transmiten a través de un medio no

seguro como es Internet así que pueden ser capturados fácilmente con herramientas especializadas y que permiten la visualización de los paquetes, obteniendo todo tipo de información que se comparta en la comunicación: nombres de usuario, contraseñas, mensajes, test resueltos, información cardiaca, actividad diaria… Incluso acceso a las propias cuentas.

II. La suplantación de identidad: tener una conexión cifrada no garantiza que el emisor y/o receptor, sean los agentes que afirman ser. Un atacante puede actuar como intermediario sin que el emisor y el receptor se percaten de ello, siendo para el emisor el receptor, y para el receptor el emisor, evitando de esta manera cualquier sospecha. Mediante esta técnica se puede obtener acceso a la totalidad de la información que se comparta, con el riesgo que esto conlleva tanto para el emisor como para el receptor.

III. Ataque por fuerza bruta: Intentan obtener por medio de todas las combinaciones posibles las combinaciones de nombre de usuario y contraseñas en una página web. Las combinaciones más afectadas son aquellas que son cortas y que utilizan siempre el mismo tipo de carácter. Mediante una política de contraseñas que obligue a utilizar cierta combinación de caracteres y establezca una longitud mínima este riesgo puede verse reducido considerablemente.

Las soluciones implementadas para hacer frente a esta diversidad de ataques han sido:

I. Cifrar las conexiones de extremo a extremo con un cifrado simétrico AES de 256 bits en modo CBC utilizando para ello un servidor HTTPS.

II. Usar curvas elípticas para firma digital, en este caso ECDSA.

III. Política de contraseñas.

En el capítulo 7 se explica en mayor profundidad la implantación de las dos primeras medidas debido a su dificultad y se exponen ejemplos del antes y del después de la implantación de ambas soluciones.

Capítulo 4
Servidor y la base de datos

A continuación, se detalla la implementación del servidor y de la base de datos, su conexión y su funcionamiento. No entra en el objetivo de esta memoria enseñar la instalación detallada y el uso de las herramientas por encontrar en Internet multitud de documentación disponible para ello.

El capítulo se encuentra dividido en dos apartados dedicados a los agentes implicados y los problemas encontrados. En el primero se especificarán los componentes del servidor que se han desarrollado, así como explicaciones sobre parte de código. En el segundo apartado, se expondrán algunos problemas durante la fase de desarrollo y las soluciones llevadas a cabo.

4.1 Agentes implicados

Los agentes implicados en el lado del servidor son dos: la base de datos y el entorno web.

4.1.1 Servidor web

Para el servidor web se llevó a cabo el diseño: Modelo-Vista-Controlador (MVC), que separa el código en tres capas independientes (Figura 4.1):

- Modelo: Capa que trabaja con los datos y que contiene los mecanismos para acceder y actualizar la información almacenada en la base de datos.
- Vista: Capa que se encarga de la visualización de las interfaces. Necesita datos para mostrar en la interfaz pero no puede obtenerlos directamente, sino que se los proporciona la capa anterior.
- Controlador: Se podría definir como la capa de conexión de las dos anteriores, actuando de intermediaria, y proveyendo de los

mecanismos necesarios para ello.

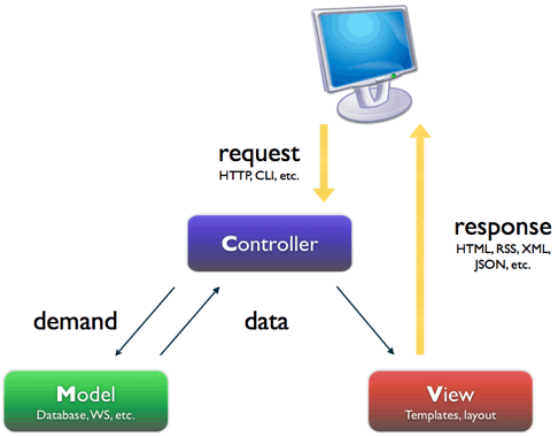

Figura 4.1: Modelo-Vista-Controlador. Libros Web

Tal y como se aprecia en la Figura 4.2, en nuestro proyecto existen varios controladores y modelos, así como distintas vistas. La aparición de la carpeta routers es para facilitar la visualización de la asignación: ruta-controlador; no modificando el diseño.

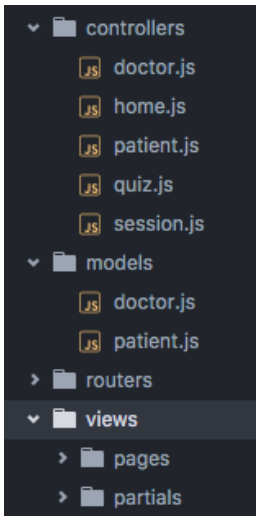

Figura 4.2: Modelo-Vista-Controlador, del proyecto.

A continuación se dan detalles de los siguientes aspectos del servidor: modelos, vistas, controladores y routers.

Modelos

Tal y como aparece en la imagen anterior, en nuestro proyecto se han definido dos modelos o esquemas. El lector se puede preguntar el porqué

utilizar esquemas en una base no relacional. La respuesta es sencilla, porque facilita la organización de los datos y la exportación de los mismos a base de datos relacionales. No obstante, en nuestro caso, solo se han creado dos modelos de las seis colecciones que almacena nuestra base de datos. Debido a que no era necesario crear un esquema para cada colección (En el apartado 4.4.2 se defiende esta decisión).

Los modelos definidos y que aparecen en el proyecto, se pueden visualizar en el Apéndice A: MODELO DOCTOR y MODELO PACIENTE.

Vistas

Las vistas son las interfaces de la aplicación web. Cada vista web debe llamar a un template, esta puede utilizarse multitud de veces en otras páginas. Para poder llevar a cabo esta característica de una forma eficaz, se ha hecho uso de los partials. Los partials son líneas de código que en vez de representar una página completa, representan pequeñas porciones de ellas, que se pueden llamar desde la vista o incluso, desde otro partial. Esto permite, por ejemplo, tener nuestra nav-bar en un partial y llamarlo en la plantilla, no teniendo que repetir el mismo código de la nav-bar por cada template, sino añadir el partial de la nav-bar. Otro ejemplo es el de utilizar una plantilla general para todo el sitio web (Figura 4.3) y añadirle los partials (Figura 4.4) necesarios dependiendo de lo que se quiera mostrar. En nuestro caso, para poder usar esta característica descrita, se hizo uso del middleware: express-ejs-layouts [11], que permite usar plantillas y añadirles los partials mediante una simple notación.

```
<div id="page-wrapper">
        <div class="row">
            <div class="col-lg-12">
                <h1 class="page-header"> <%= title %> </h1>
            </div>
            <!-- /.col-lg-12 -->
        </div>
        <!-- /.row -->
        <% include ../../partials/patient/patientInfo %>
        <% include ../../partials/patient/quizesPanel %>

</div>
```

Figura 4.3: Uso de la una plantilla con un partial.

errorValidate.ejs

```
<% if(errors.length || Object.keys(errors).length) { %>
  <span id="ErrorMsgs">
    Corregir errores: <br>
    <% for(var i in errors){ %>
      <div class="alert alert-warning" role="alert"> <%= errors[i].message %> </div>
    <% } %>
  </span>
<% } %>
```

Figura 4.4: Ejemplo de un partial.

Los templates muestran información, y en nuestro caso, esa información debe ser extraída de la base de datos. Las vistas no pueden comunicarse con los modelos ni con las colecciones de nuestro proyecto. Esos datos se consiguen a través de los controladores.

Controladores

Los controladores forman la capa de interconexión entre los modelos y las vistas. Se encargan de solicitar los datos que las vistas necesitan a la base de datos (Modelos), y luego renderizan las vistas junto con la información obtenida.

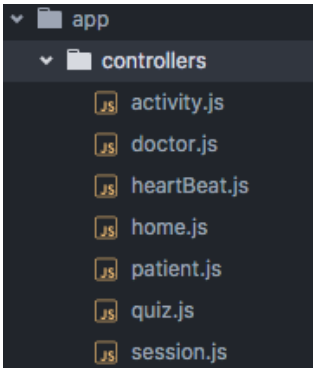

Figura 4.5: Controladores del sitio web.

En la Figura 4.5 se muestran los controladores de nuestro sitio web. Se encuentran agrupados por funcionalidades en distintos ficheros. Se puede apreciar que algunos nombres con los que se describen los controladores coinciden con el de las colecciones y esto no es casual. Se ha tratado que cada controlador, consulte una sola colección, estructurando mejor la aplicación. En el caso de que se necesitasen datos de dos colecciones diferentes para una misma vista un controlador llama a otro.

Según hemos explicado hasta el momento, el controlador es el

encargado de consultar a los modelos y enviar la información a las vistas, pero, ¿quién llama al controlador?

Routers

Son manejadores de rutas. Un Router es un sistema de middleware y direccionamiento completo que se utiliza para que cuando se realiza una petición al servidor con una URL específica se llame a un determinado controlador. Nuestro manejador se encuentra en el fichero express.js donde se han agregado las siguientes rutas que aparecen en la Figura 4.6.

```
// routes
var router = require(config.root + '/app/routers/index');
var doctorRouter = require(config.root + '/app/routers/doctor');
var quizRouter = require(config.root + '/app/routers/quiz');
var patientRouter = require(config.root + '/app/routers/patient');
```

Figura 4.6: Declaración de los routers

También se han asignado a una ruta específica (Figura 4.7).

```
//Asignación de las rutas
app.use('/', router);
app.use('/doctor/', doctorRouter);
app.use('/quiz/', quizRouter);
app.use('/patient/', patientRouter);
```

Figura 4.7: Asignación explícita de las rutas.

Esto significa que por ejemplo la ruta '/doctor/new' será manejada por doctorRouter y que la ruta 'quiz/show' será manejada por quizRouter, y así sucesivamente.

La declaración dentro de los ficheros de los routers es la siguiente:

```
var express = require('express');
var patientRouter = express.Router();
var patientController = require('../controllers/patient');
var sessionController = require('../controllers/session');
var quizController = require('../controllers/quiz');

// Autoloads
patientRouter.param('patientId', patientController.load);
patientRouter.param('quizId', quizController.load);
patientRouter.param('solvedId', patientController.solvedQuizLoad);

//  GET '/new'
patientRouter.get('/new', sessionController.loginRequired, patientController.new);
patientRouter.post('/new', sessionController.loginRequired, patientController.create);
patientRouter.get('/', sessionController.loginRequired, patientController.index);
```

Figura 4.8: Ejemplo de parte del fichero routers/patient.js

Como se puede observar, sigue la estructura:

- router.RequestMethod(URL,controller)

Donde RequestMethod puede ser: GET, POST, PUT, …

También es importante señalar que se ha tratado de seguir el concepto de una API REST [12] , manteniendo las URL una jerarquía lógica, en la medida de lo posible.

4.1.2 La base de datos: MongoDB

El sistema escogido como base de datos fue Mongo-DB, debido a su simplicidad, su buena documentación, su fácil escalabilidad, velocidad horizontal y a que ha sido diseñada para la web. Las consultas no se realizan mediante SQL, sino mediante objetos JSON y las respuestas se devuelven en ese mismo formato, porque Mongo almacena la información en documentos BSON, que en la práctica facilita la representación de la información que se devuelve a la web.

Lo primero que se hizo, fue instalar un servidor Mongo-DB (Mac):

```
$ brew install mongodb
```

Luego se creó un directorio donde almacenar los datos de nuestra base de datos y pasárselo como parámetro a la hora de arrancar el servicio:

```
$ mongod –dbpath /ruta_al_dir_data
```

Figura 4.9: Colecciones necesarias para nuestra proyecto

A continuación, se creó la base de datos para la aplicación (En este caso, dapsserver-development) y se diseñó el número de colecciones necesarias para el funcionamiento correcto de nuestra aplicación:

- Activity: almacena la distancia recorrida por los pacientes.
- Doctors: almacena los médicos de la aplicación.

- Heartbeats: almacena la información cardiaca de los pacientes.
- Patients: almacena los pacientes.
- Quizes: almacena los test de memoria.
- Solvedquizes: almacena los test resueltos por los pacientes.

Dentro de cada una de las colecciones se almacenan documentos, que no tienen que seguir una estructura o un esquema. Esto representa una gran ventaja respecto a las bases de datos relacionales, y en este proyecto sacamos ventaja de esto. Un ejemplo claro en nuestra app es el almacenamiento de los tests Con Mongo podemos almacenar dos tests distintos (distinto número de preguntas, distinto número de posibles respuestas, etc) en una misma colección. En una base de datos convencional esto no se puede hacer, no puedes almacenar dos filas con diferente estructura, en una misma tabla, resulta imposible. Para conseguirlo, debes diseñar una tabla para las preguntas, otra para las respuestas y otra para los test, siendo mucho más complicado e ineficiente, además que tener que crear un objeto JSON, con la información resultante, cosa que con la opción elegida no es necesario (Figura 4.10).


```
{
  "questionId": 3,
  "questionType": "Memoria",
  "phrase": "¿Cuántas patas tiene una araña?",
  "answerType": "number",
  "nAnswer": 1,
  "correctAnswersId": [0],
  "answers": [
    {
      "answerId": 0,
      "type": "number",
      "body": "8"
    }
  ]
},
{
  "questionId": 4,
  "questionType": "Memoria",
  "phrase": "¿Cuál de las siguientes imágenes es un perro?",
  "answerType": "image",
  "nAnswer": 4,
  "correctAnswersId": [2],
  "answers": [
    {
      "answerId": 0,
      "type": "image",
      "body": "http://i.imgur.com/tYve4JU.jpg"
    },
    {
      "answerId": 1,
      "type": "image",
      "body": "http://i.imgur.com/3nBkoeb.jpg"
    },
    {
      "answerId": 2,
      "type": "image",
      "body": "http://i.imgur.com/BqYRtb2.jpg"
    },
    {
      "answerId": 3,
      "type": "image",
      "body": "http://i.imgur.com/etAhhpV.jpg"
    }
  ]
},
```


Figura 4.10: Ejemplo de dos preguntas con diferente esquema

No obstante, y como se mencionó anteriormente, sí se llevó a cabo un esquema para dos colecciones: Doctors y Patients. Esto se realizó para poder hacer las consultas de una forma más sencilla. El middleware utilizado para poder realizar consultas de los esquemas definidos e insertar datos en ellos fue mongoose [13]. Debido a distintas limitaciones de este middleware fue necesario utilizar también otro middleware MongoJS [14]. Con este último, no es necesario declarar esquemas para realizar las operaciones, algo sumamente importante a la hora de manipular el resto de colecciones.

4.2 Problemas encontrados

4.2.1 Introducir los tests

Una de las ideas originales era introducir los test directamente mediante la consola, en la base de datos. A pesar de que funcionaba, no era atractivo para la expansión de la herramienta y mucho menos se aprovechaba la potencia de la base de datos. La solución fue introducir los tests mediante documentos JSON. Así, los test se pueden importar en documentos con extensión JSON, a la base de datos. Para ello, se habilitó una página dentro del sitio web, en la que se puede subir un test, y el controlador se encarga de volcarlo en la base de datos.

4.2.2 Métodos asíncronos

El tiempo que se tarda en obtener una consulta en la base de datos no es constante. Puede influir el número de conexiones, la velocidad de la red, etc. El problema no es grave en el caso de que la vista necesite datos de una sola colección puesto las funciones para realizar la consulta permiten un callback, ejemplo:

```
db.mycollection.find(function (err, docs) {
    // docs is an array of all the documents in mycollection
})
```

En este caso, dentro de la función se puede renderizar la vista.

```
db.mycollection1.find(function (err, docs) {
    // docs is an array of all the documents in mycollection
     db.mycollection2.find(function (err, docs) {
          // docs is an array of all the documents in mycollection
          db.mycollection2.find(function (err, docs) {
               // docs is an array of all the documents in mycollection
               ...
```

```
        })
    })
})
```

El código se va complicando cada vez más. La solución consiste en usar las promesas de JavaScript [15] y el método Promise.all(), que ejecuta el código, si y solo si se han cumplido las promesas que se han introducido como parámetro.

```
Promise.all([getAllQuizesP, getAllUnSolvedQuizesP, getAllSolvedQuizesP]).then(function(){
  res.render('pages/patient/show', { title: 'Datos del paciente', patient: req.patient,
  solvedQuizes: solvedQuizes, unSolvedQuizes: unSolvedQuizes, quizes: quizes});
}, function(){res.send('Se ha producido un error');});
```

Figura 4.11: Ejemplo de uso del método Promise.all() en el proyecto

L a Figura 4.11 muestra un ejemplo de uso en el que se hace necesario esperar por tresoperaciones: getAllQuizesP, getAllUnsolvedQuizesP y getAllSolvedQuizesP; para poder renderizar un template almacenado en: *Pages/patient/show.*

4.2.3 Relaciones en una base de datos no relacional

La pregunta de cómo relacionar dos esquemas en una base de datos no relacional apareció en el momento de añadir la función de agregar pacientes. En esta herramienta, un paciente solo puede tener un médico y un médico puede tener varios pacientes. Para resolver este problema se usa el método populate().

Primero, se añadió el campo doctor al esquema del paciente:

```
doctor: { type: Schema.ObjectId, ref: "Doctor"}
```

A continuación, en los datos obtenidos de una consulta, se ‘popula’ el campo doctor con la siguiente línea:

```
Doctor.populate(patient, {path: 'doctor'},function(err, patient)
```

Esta implementación se puede ver en el método autoload, disponible en el REPOSITORIO DEL SERVIDOR.

Capítulo 5
Aplicación móvil y el smartwatch

A continuación, se detalla la implementación de la aplicación del servidor y de la aplicación para el reloj. Se explican diferentes funcionalidades que se han introducido, así como la comunicación entre los dos dispositivos y el servidor.

El capítulo se encuentra dividido en tres partes: la aplicación móvil, la aplicación del smartwatch y la comunicación entre el móvil y el reloj. En los dos primeros apartados se profundizará en las aplicaciones. En el tercero, se explicará brevemente la comunicación entre ambas aplicaciones.

5.1 Aplicación móvil

Debido al propósito de este proyecto, se ha llevado a cabo una aplicación para dispositivos móviles Android que permite realizar los test, recoger los datos provenientes de la aplicación del smartwatch y enviarlos para su posterior análisis al servidor. A continuación, se detallan diferentes características y su implementación.

Para poder realizar los test de memoria que el médico ha asignado al paciente (usuario de la aplicación móvil) se hace necesario en primer lugar, descargarlos. En Android, se pueden realizar peticiones HTTP usando distintas librerías. Unas de las más conocidas y de las que hacemos uso son: Volley [16] y HttpURLConnection [17]

Volley, es usada para la descarga de imágenes y de la lista de los tests (Figura 5.1), pero no para la descarga de los test en sí. El motivo de que esto sea así es debido a la implementación llevada a caba para la construcción del objeto quiz, que más adelante se explicará en profundidad, necesita de

una variable InputStream, en lugar de un array JSON, que es lo que proporciona en la variable response este método.

```
String url = URL_BASE + ID_USER + UNSOLVED_QUIZES_URL;
System.out.println(url);

JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(url, new Response.Listener<JSONArray>() {
    @Override
    public void onResponse(JSONArray response) {
        System.out.println("Se han descargado los tests");
        try {
            unsolvedQuizList_ = new SimpleQuizObject(response);

            if (Comprobar.comprobarFirma2(unsolvedQuizList_.getMensaje_(),unsolvedQuizList_.getSignature_()) == false){
                Toast.makeText(getApplicationContext(),"La firma es falsa",Toast.LENGTH_SHORT).show();
            }else {
                Toast.makeText(getApplicationContext(),"La firma es verdadera",Toast.LENGTH_SHORT).show();
                loadQuizesListFragment();
            }
```

Figura 5.1: Descarga de la lista de los test asignados al paciente

Para el envío de los cuestionarios resueltos y de los datos recogidos del smartwatch al servidor (Figura 5.2) sí se hace uso de la librería Volley gracias a que permite enviar objetos JSON sin la mayor dificultad mediante una petición PUT, objetos que después son agregados directamente (después de haber comprobado su firma digital) en la base de datos.

```
private void uploadQuiz(JSONObject jsonObject) throws JSONException {
    /*String url = "http://192.168.1.39:3000/patient/5759e87fb78c9ddd2917b35c" + "/quiz/solvedQuizes/add";
    String url = URL_BASE + ID_USER + "/quiz/solvedQuizes/add";

    JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.PUT, url,jsonObject,
            new Response.Listener<JSONObject>(){
                @Override
                public void onResponse(JSONObject response) {
                    Toast.makeText(getApplicationContext(),response.toString(),Toast.LENGTH_LONG).show();
                }
            }, new Response.ErrorListener(){
        @Override
        public void onErrorResponse(VolleyError error) {
            Toast.makeText(getApplicationContext(),error.toString(),Toast.LENGTH_LONG).show();
        }
    }){...};

    RequestQueue queue = Volley.newRequestQueue(this);
    queue.add(jsonObjectRequest);
}
```

Figura 5.2: Envío del test resuelto al servidor

Por otra parte, HttpURLConnection ha sido empleada en la descarga del cuestionario escogido por el usuario (Figura 5.3) de entre la lista de tests asignados por el médico desde el sitio web. Esta librería devuelve un objeto InputStream, que podemos ir leyendo y almacenando en un JsonReader, para la posterior llamada al constructor del objeto Quiz (objeto que representa un cuestionario).

```
URL url = new URL(params[0]);

connection = (HttpURLConnection) url.openConnection();

connection.setRequestProperty("Content-Type", "application/json");
connection.setRequestMethod("GET");

statusCode = connection.getResponseCode();
System.out.println("Async call code: " + connection.getResponseCode());

if (statusCode ==  200) {
    System.out.println("Server responded with code: " + statusCode);

    InputStream inputStream = new BufferedInputStream(connection.getInputStream());
    JsonReader reader = new JsonReader(new InputStreamReader(inputStream, "UTF-8"));
    quiz_ = new Quiz(reader);
    result = true;
lose Stream */
    if(null!=inputStream){
        inputStream.close();
    }
}
else{
    System.out.println("Error, no se pudo conectar con el servidor");
}
```

Figura 5.3: Descarga del test seleccionado

En el constructor de Quiz, se realizan llamadas a distintas funciones que se encargan de ir parseándolo y construyendo el nuevo objeto Quiz (Figura 5.4), compuesto de unos parámetros que describen el test y de las preguntas y respuestas.

```
reader.beginObject();
while(reader.hasNext()){
    String name = reader.nextName();
    System.out.println(name);
    switch (name){
        case "_id":
            id_ = reader.nextString();
            break;
        case "quizName":
            quizName = reader.nextString();
            break;
        case "doctor":
            author = reader.nextString();
            break;
        case "nQuestions":
            nQuestions_ = reader.nextInt();
            break;
        case "questions":
           readArrayQuestions(reader);
            break;
    }
```

Figura 5.4: Función readQuizDetails

Finalizada la construcción del objeto quiz, se procede a generar su representación gráfica para el usuario por medio de Fragments [18]. El uso de los mismos en lugar de Activities [19] es para evitar un uso excesivo de la memoria RAM del teléfono, pero trae consigo un problema. Los Fragments, a diferencia de las Activities, no se pueden comunicar entre ellos, haciendo

necesario el uso de una Activity base que sea intermediaria. El número de fragments distintos creados en la aplicación, para realizar los test, es el mismo que los diferentes tipos de preguntas que existen en la herramienta, es decir, cuatro: preguntas de selección simple, de selección múltiple, de selección de imágenes y de introducir un valor numérico. Por cada fragment, definido, hay que definir también sus adaptadores y sus vistas respectivas, creciendo la cantidad de ficheros a generar. Un ejemplo del número de ficheros necesarios para representar las preguntas de selección de imágenes se muestra en la Figura 5.5.

Figura 5.5: Archivos que componen el ImageFragment en su conjunto

Por último, se ha creado una función que controla la asignación correcta fragment-pregunta y que permite al paciente realizar el test. Durante la realización del test, los enunciados de las preguntas y las respuestas dadas por el paciente se van almacenando en una nueva clase (CurrentUserAnswer) y que en el momento en que se finaliza el test se transforma a un objeto JSON. Este objeto es firmado digitalmente y se encapsula dentro de otro objeto JSON acompañado de la firma que es lo que finalmente se envía al servidor.

5.2 Aplicación para el smartwatch

La aplicación del smartwatch ha sido diseñada para la obtención de ciertos parámetros de interés médico, en el paciente. Para ello, se hizo necesario acceder a los sensores disponibles en el reloj. En nuestro dos: el sensor del ritmo cardiaco (Heart-Rate Sensor, Figura 5.6) y el contador de pasos (Step-Counter Sensor).

```
mSensorManager_ = (SensorManager) getSystemService(SENSOR_SERVICE);
Sensor mHeartRateSensor = mSensorManager_.getDefaultSensor(Sensor.TYPE_HEART_RATE);
```

Figura 5.6: Acceso al sensor del ritmo cardiaco

El siguiente paso era registrar cualquier cambio en el sensor (Figura 5.7) y registrarlo para su posterior envío al servicio correspondiente de la aplicación móvil.

```
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_HEART_RATE && event.values.length > 0){
        int newValue = Math.round(event.values[0]);
        heartBeatDataQueue_.add(newValue);
        if(currentValue_ != newValue){
            Log.d(LOG_HEART_TAG, "Heart-beat: " + newValue);
            // Actualizo
            currentValue_ = newValue;
        }
```

Figura 5.7: Función que detecta cambios en el Sensor Heart-Rate

5.3 Comunicación entre móvil y smartwatch.

En el apartado dedicado a la aplicación móvil se ha explicado la comunicación entre el servidor y la aplicación móvil. A continuación se expone la transferencia de los datos capturados en el smartwatch hacia el dispositivo móvil.

En el móvil se ha definido un servicio que se mantiene activo escuchando cualquier envío de parte del reloj. Este servicio hereda de la clase WearableListenerService (Figura 5.8) y permite que la aplicación no esté necesariamente abierta para recibir los datos.

```
public class DataLayerListenerService extends WearableListenerService
```

Figura 5.8: Declaración de la clase DataLayerListenerService

En el reloj, se comprueba antes de enviar los datos que se puede conectar al servicio, enviando la información por Bluetooth y quedando a la espera de un nuevo inicio programado. Ejemplo en la Figura 5.9.

```
AlarmManager alarmMgr = (AlarmManager) context_.getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(context_, AlarmHeartBeat.class);
PendingIntent alarmIntent = PendingIntent.getBroadcast(context_,0,intent,0);

// Set the alarm to start at approximately 18:00
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 18);

alarmMgr.setInexactRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), AlarmManager.INTERVAL_DAY, alarmIntent);
```

Figura 5.9: Alarma programada a las 18 horas

Capítulo 6
Combinación de sistemas

En el presente capítulo se detalla el funcionamiento de la herramienta, combinando los sistemas que la componen. Este capítulo se divide en dos apartados. En el primero se propondrá un ejemplo ilustrativo de lo que se pretende y en el segundo se llevará a cabo el funcionamiento del mismo en la aplicación.

6.1 Ejemplo ilustrativo

Se desea agregar un médico al sitio web, con el nombre de usuario, “Ejemplo” y la contraseña, “password”. A continuación, se procederá a visualizar los test disponibles y a añadir uno nuevo. Más tarde, el médico añadirá un paciente nuevo y se le asignará el test previamente cargado. El paciente realizará el test desde la aplicación móvil y se enviarán también los parámetros de la frecuencia cardiaca y de la actividad física y se almacenarán en la base de datos del Servidor. Por último, el médico podrá visualizarlos desde el perfil del paciente.

6.2 Funcionamiento

Agregar un médico a la plataforma web.

Para ello se hace clic en “¿Nuevo?”, en la página principal del Login (Figura 6.1).

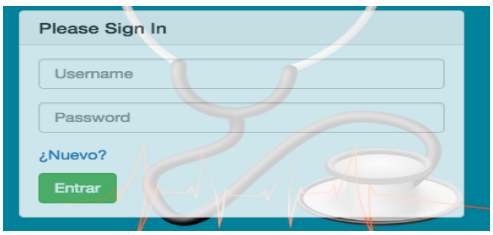

Figura 6.1: Login del sitio web

A continuación, hay que rellenar un formulario, en el cual, si se produce algún error en los campos, la aplicación avisa por medio de un recuadro de advertencia (Figura 6.2).

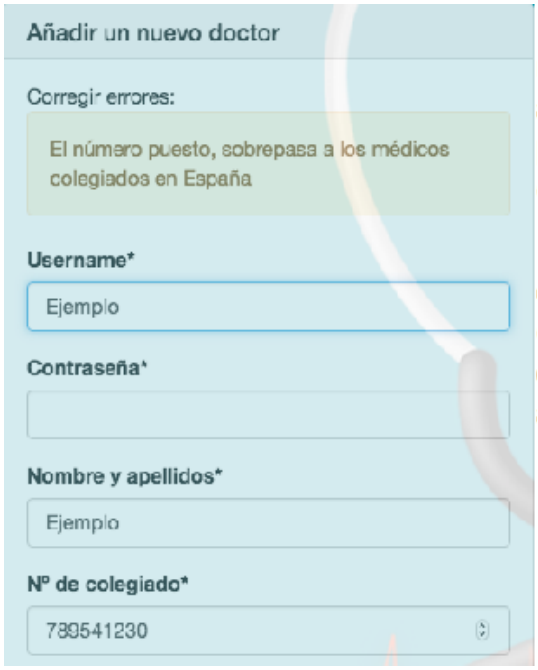

Figura 6.2: Error durante el registro

Una vez completado el registro con éxito, redirigirá al login principal (Figura 6.1). Introduciremos nuestros datos de acceso y damos click en aceptar. A continuación se nos mostrará el Dashboard del médico (Figura 6.3), desde el cual podemos acceder a todas las funcionalidades disponibles.

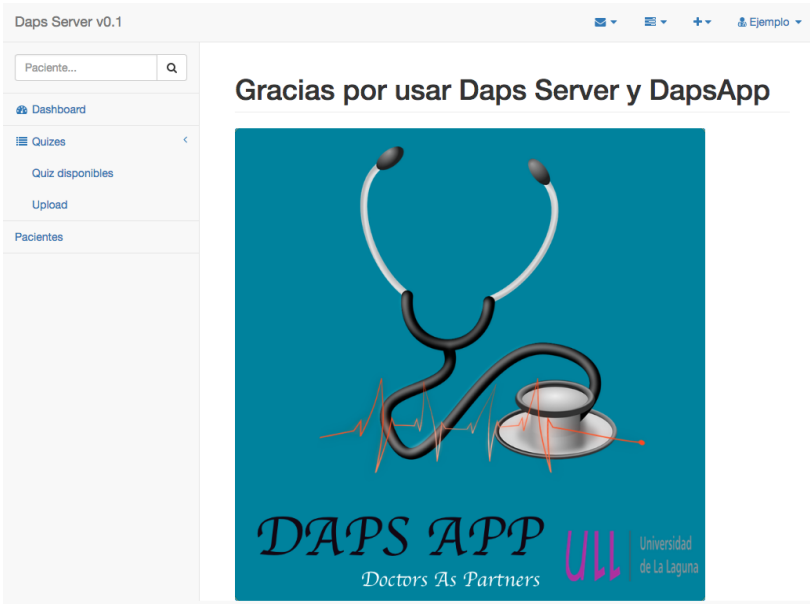

Figura 6.3: Dashboard

Añadir y visualizar un test a la herramienta.

Para añadir un test, nos dirigimos a la pestaña Quizes y hacemos clic en Upload. Seleccionamos el archivo con extensión JSON y pulsamos en, Subir test. El sistema nos avisará del éxito de la operación (Figura 6.4).

Figura 6.4: Test subido

Para visualizarnos, nos dirigiremos haciendo uso del menú, a la lista de los test disponibles(Figura 6.5).

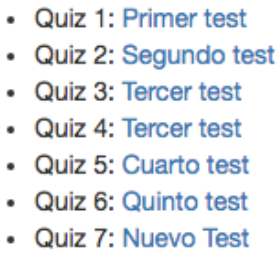

Figura 6.5: Lista de test

Desde ahí, podemos seleccionar el test para visualizarlo (Figura 6.6).

Figura 6.6: Visualización del test subido

Creación de un paciente y asignación de un test al mismo

Para crear un paciente dentro de la aplicación, hay que dirigirse al menú superior derecho (Figura 6.7) y escoger la opción correspondiente. Acto seguido, rellenar el formulario y hacer clic en, Crear.

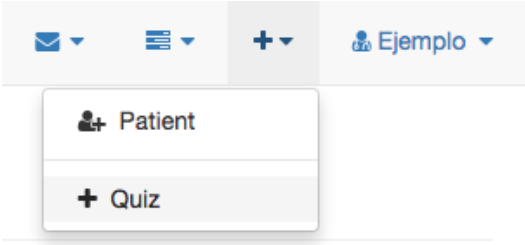

Figura 6.7: Pestaña para añadir un paciente

En el perfil del paciente, tenemos acceso a los datos personales, de contacto y clínicos (Figura 6.8) del mismo, además de los test de memoria resueltos, aún sin resolver, y los test disponibles para asignar.

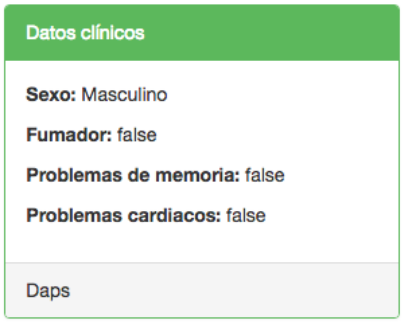

Figura 6.8: Datos clínicos

En esta última pestaña, seleccionamos el test que previamente hemos subido y pulsamos en el botón añadir (Figura 6.9).

Figura 6.9: Añadiendo un test al usuario

Realización del test en el dispositivo móvil del paciente

Abrimos la aplicación del paciente (Figura 6.10) y seleccionamos en el menú deslizable (Figura 6.10), el apartado Test. Acto seguido, se obtendrá una lista

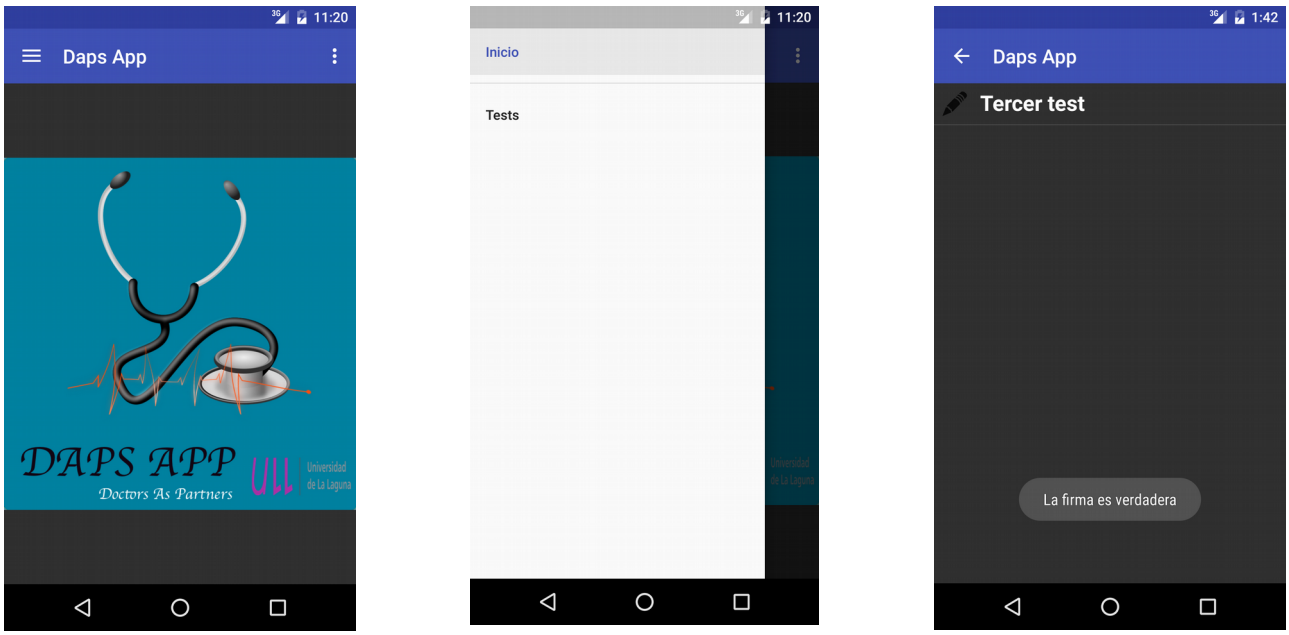

Figura 6.10: Daps App

de los test asignados y se comprobará la firma del servidor. A continuación se seleccionará el test y se procederá a la realización del mismo. Una vez finalizado, se mostrarán el número de aciertos y se enviará el test, junto con la firma (Figura 6.11), al servidor. Automáticamente, el reloj, también recogerá los datos de los sensores y se enviarán al servidor.

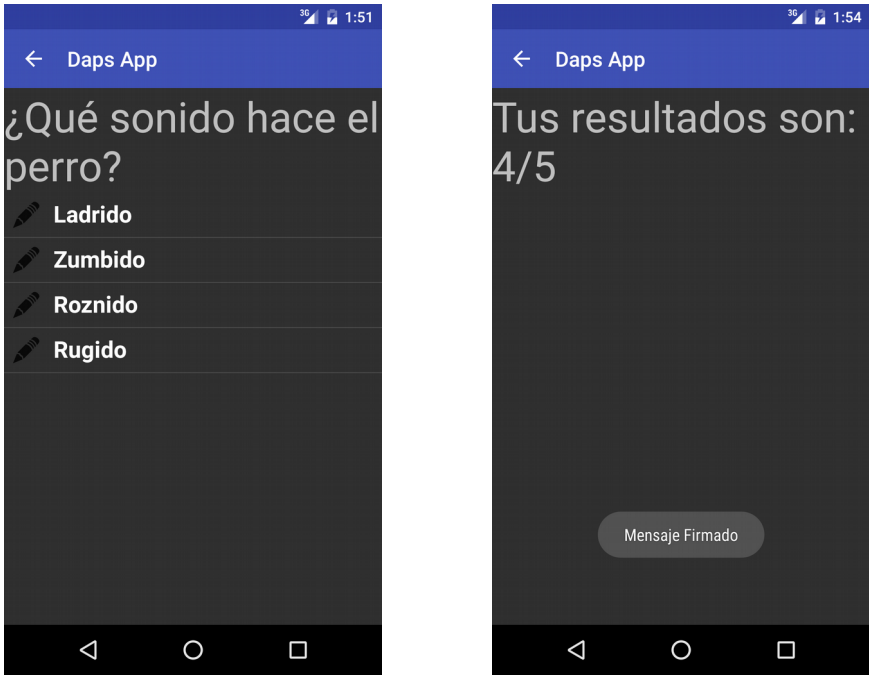

Figura 6.11: Realización y firma del test

El la Figura 6.12, se muestra en funcionamiento el pulsómetro del smartwatch.

Figura 6.12: Sensor del reloj

Visualización del test y de los parámetros obtenidos.

El test aparecerá automáticamente en la tabla de test resueltos, y se puede acceder a él, haciendo clic en ver (Figura 6.13).

Figura 6.13: Lista de los test resueltos

La visualización es idéntica a la de los test originales (Figura 6.6).

Por último, la visualización de los parámetros del smartwatch: El ritmo cardiaco y la actividad física diaria (Figura 6.14) se representan mediante gráficas para facilitar el análisis y la comparación de las mismas.

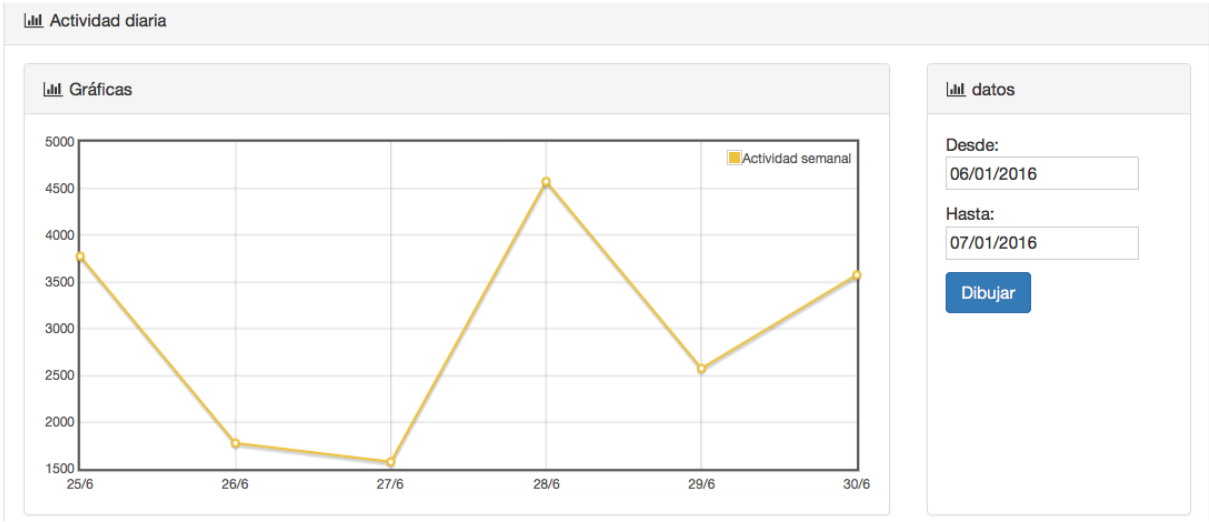

Figura 6.14: Representación de la actividad física.

Capítulo 7
Seguridad

En este capítulo se profundiza en la implantación de la seguridad en la herramienta. Se destacan las mejoras introducidas en este apartado y se muestran capturas con las soluciones ya implementadas.

7.1 Problemática

Para el uso, transporte y análisis de datos médicos y personales de los pacientes, se hace indispensable proteger la seguridad. En nuestro caso, se han cubierto dos problemas principales:

IV. Cifrado de los datos: los datos se van a transmitir a través de un medio no seguro como es Internet. Por ello, se hace necesario que toda la información que se transmita por la red vaya cifrada.

V. Evitar la suplantación de identidad: tener una conexión cifrada no garantiza que el emisor y/o receptor, sean los agentes que afirman ser. Por lo tanto, se hace necesario implementar la firma digital.

7.2 HTTPS

En la actualidad, existen varias maneras de encriptar la información por Internet. Uno de los estándares más extendidos y que soporta gran multitud de algoritmos de cifrado es el Protocolo Seguro de Transferencia de Hypertexto (HTTPS).

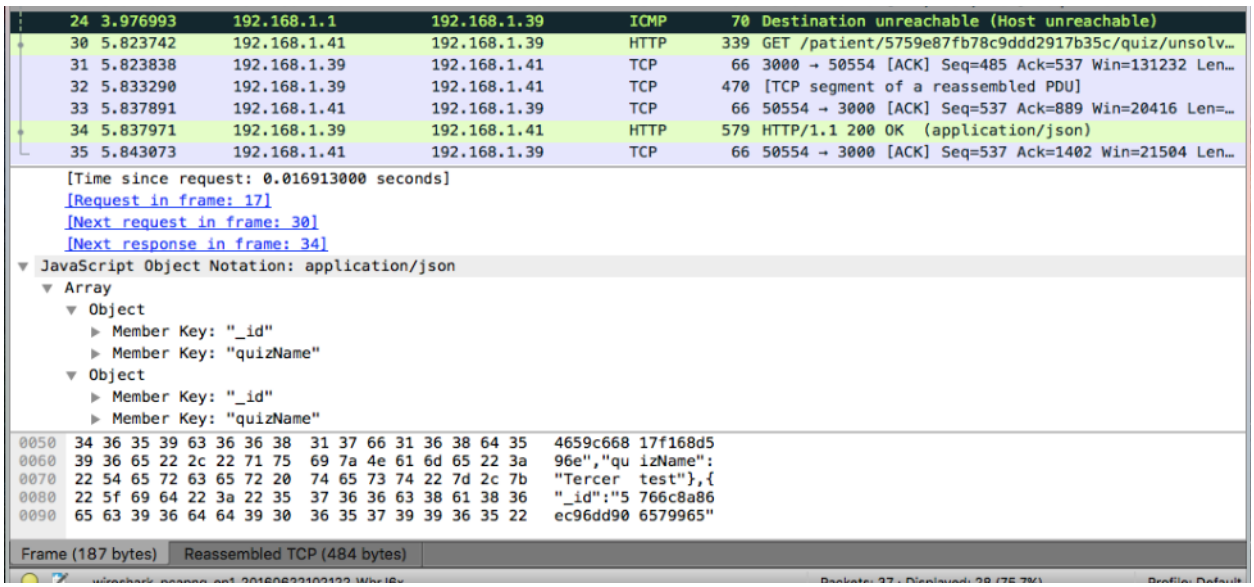

Figura 7.1: Captura de un paquete sin cifrar

Los mensajes que se transmiten por Internet, entre cualquier dispositivo que pueda acceder a la red y los servidores HTTP, van en texto plano. Esto significa que si un atacante decidiera capturar los paquetes que se transmiten durante la comunicación, podría acceder a la información sin ningún esfuerzo. La Figura 7.1 muestra la captura de un paquete durante la comunicación entre la aplicación móvil y el servidor web funcionando con el protocolo HTTP y donde se aprecia el texto plano transmitido. En el protocolo HTTPS, los mensajes van cifrados de extremo a extremo. En el momento de establecer la comunicación se comprueba el certificado del servidor, el cual debe estar avalado por alguna Autoridad Certificadora (CA) y se utiliza la clave pública proporcionada en el mismo, para cifrar los datos desde el emisor al servidor y descifrar los datos que llegan desde el servidor

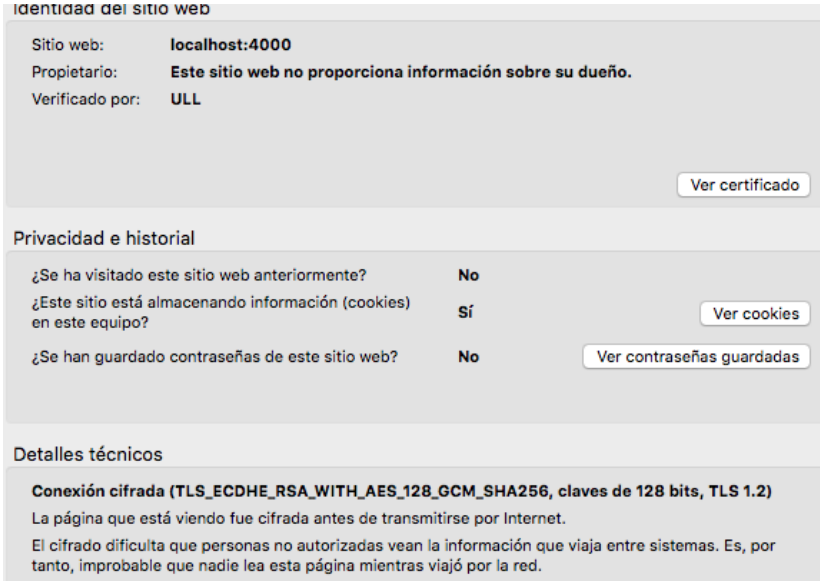

Figura 7.2: Certificado del servidor

Para la implantación en nuestro servidor, se escogió el cifrado AES, también conocido como Rijndael. La clave utilizada para las pruebas fue de 128 bits, pudiendo ser implementada con tamaños de 128, 192 o 256 bits. Los certificados del servidor se generaron con la herramienta OpenSSL y fueron firmados por nosotros mismos debido a que nos encontramos en un entorno de desarrollo. En la Figura 7.2 se aprecian más características.

Por último, se procedió a volver a capturar los paquetes para comprobar su funcionamiento, tal y como se aprecia en la Figura 7.3.

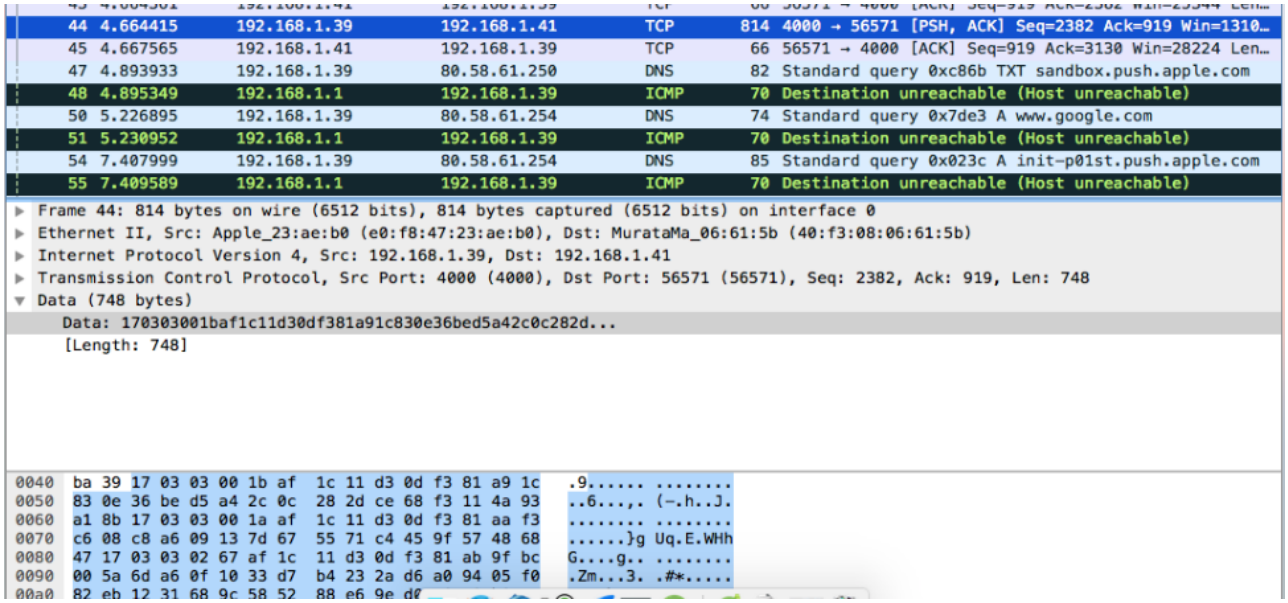

Figura 7.3: Captura de un paquete cifrado

7.3 Firma Digital

Los mensajes van cifrados de extremo a extremo pero y conocemos la identidad del servidor por medio del certificado pero para estar seguros de que el dispositivo que envía los datos del paciente es realmente el del mismo usamos la firma digital basada en curvas elípticas, que consiste tal y como se representa en la Figura 7.4 en la firma del mensaje a transmitir al receptor, por parte del emisor, y en la verificación del receptor de que el mensaje recibido corresponda con la firma electrónica, usando para ello la clave pública del emisor.

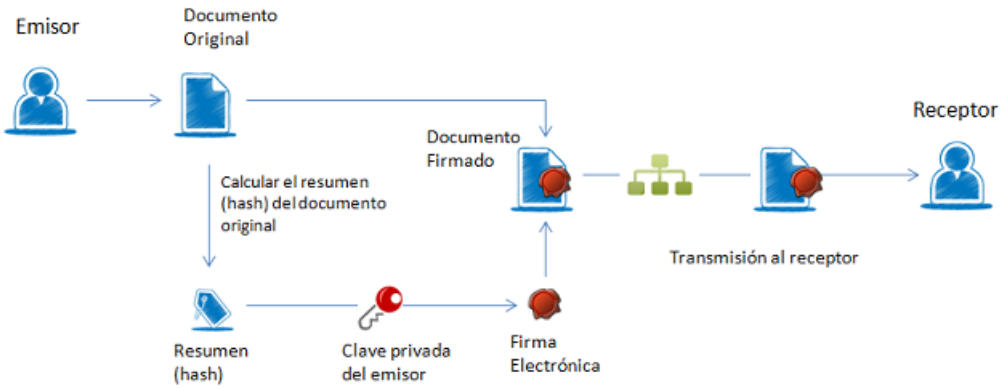

Figura 7.4: Firma digital

En nuestro proyecto, la firma implementada fue la Elliptic Curve Digital Signature Algorithm (ECDSA) que utiliza el algoritmo SHA1 para la función hash. La Figura 7.5 representa la implementación llevada a cabo en el servidor para la comprobación de un mensaje enviado desde el móvil de paciente.

```
// Choose algorithm to the hash function
var algorithm = 'sha1';
var message = JSON.stringify(respuesta);
console.log(message);
// Comprobar la firma y el mensaje
var verifier = crypto.createVerify(algorithm);
verifier.update(message);
var ver = verifier.verify(clavePublicaMovil, mensajefirmado,'base64');
```

Figura 7.5: Verificación de la firma en el servidor

En la parte de la aplicación móvil, la Figura 7.6 muestra el uso de la clase Firmar, desarrollada para firmar cualquier mensaje dirigido al servidor. El código de la clase se disponible en el Apéndice B.

```
signature = Firmar.firmar(mensaje);
respuestaCompleta.put(SIGNATURE,signature);
respuestaCompleta.put(MENSAJE,solvedQuiz);
```

Figura 7.6: Firma de un test resuelto en la aplicación móvil

Capítulo 8
Presupuesto

A continuación se presenta el desglose del presupuesto necesario para la puesta en producción, mantenimiento y desarrollo de la herramienta que se ha expuesto en el presente documento. Se han incluido horas de programación de un ingeniero informático especializado para la realización de algunos cambios y mejoras. Asimismo se ha procedido a incluir el coste de un certificado firmado por CA o Android.

Por otra parte, hay que puntualizar que existen pulseras con pulsómetro con un precio bastante reducido y que hacen más plausible e interesante la implantación de la herramienta. Sin embargo las API de la mayoría de ellas no son abiertas, impidiendo el desarrollo. No obstante, si esto no fuera así y fuera posible la utilización de dichas pulseras, habría que realizar un mínimo cambio en la configuración de la toma de datos en la aplicación móvil, no siendo necesario realizar ningún cambio en el servidor. Esto conllevaría un pequeño costo en el presupuesto del personal, debido a la necesidad de añadir horas de desarrollo.

8.1 Personal

Para la implantación, mantenimiento y desarrollo de la herramienta se necesita personal. Para facilitar el cálculo mostrado en la tabla 8.1 en este apartado, el presupuesto se ha realizado teniendo en cuenta el número de horas necesarias para cada actividad y que cada hora suponga un gasto de 15€.

Actividad	Horas	Coste
Implantación	120	1800 €
Instalación del certificado	10	150 €
Mejoras de la interfaz móvil	95	1425 €
Mejoras en la interfaz web	75	1125 €
Total	300	4500 €

Tabla 8.1

8.2 Componentes

A continuación en la tabla 8.2 se detallan el precio de los componentes y se hace un presupuesto comprando cada uno de ellos. Dependiendo de la política de los hospitales o clínicas, este costo podría variar.

Componente	Modelo	Cantidad	Coste
Servidor	Dell, PowerEdge T130	1	629 €
Móvil	Motorola Moto G (3ª Generación)	1	132€
Smart-Watch	Motorola Moto 360 v2 Sport	1	199 €
Total			960 €

Tabla 8.2

8.3 Coste total

Uniendo los costes totales de los presupuestos anteriores y añadiendo el coste de un certificado firmado y el de un dominio, el coste total es:

Presupuesto	Coste
Personal	4500 €
Componentes	960 €
Certificado firmado por Android	100 €
Dominio / año	12 €
Total	5572,50 €

Tabla 8.3: Coste total

Capítulo 9
Conclusiones y líneas futuras

Una aplicación móvil segura que permite interactuar a un médico con su paciente, que hace uso de distintos sensores para obtener diferentes datos médicos como son: la actividad diaria y el ritmo cardiaco. Unida a un sitio web que permite al médico correspondiente, visualizar los datos de los pacientes, ha sido un reto que llevado a cabo con el propósito aprovechar las nuevas tecnologías al servicio de la sociedad.

La idea de diseñar y construir una herramienta que permita ayudar a prevenir y a diagnosticar enfermedades entre los pacientes que la utilizan es esperanzadora y ha sido sumamente influyente a la hora de cuidar el desarrollo de la aplicación. Las posibilidades son infinitas y todo granito de arena que se aporte a la utilización de las nuevas tecnologías en la medicina es importante. Nunca se sabe, tal vez lo que se desarrolle en este momento sea muy simple, pero la idea siempre se puede mejorar y servir como base para un proyecto que puede usarse en la vida real. Asimismo, la tecnología avanza muy rápido y cada vez baja más de precio y puede estar disponible a un mayor público. El precio, como en la gran mayoría de los casos, es el limitante, pero van apareciendo aparatos como la pulsera mencionada en el capítulo anterior o marcas como fit-bit, que están entrando en el mercado y bajando los precios. Es por ello, y previendo la aparición de más dispositivos wearables, que se ha intentado diseñar, en la medida de lo posible, que la herramienta sea modulable. El servidor no ha sido desarrollado exclusivamente para conectarse con un teléfono móvil con sistema operativo Android, ni tampoco la aplicación necesita usar una única marca de smartwatch para capturar los datos, pudiendose usar también otros dispositivos inteligentes. Estas dos últimas características no serían posible sin la importancia de los estándares, por ejemplo en el envío de información en Internet, JSON; o en las conexiones entre los dispositivos: la tecnología Bluetooth. Esta es tal vez la muestra de que trabajando todos en una misma dirección avanzamos más rápido y la sociedad puede sacar más partido de

las investigaciones e inversiones.

Durante el desarrollo de la aplicación móvil y del sitio web fueron apareciendo ideas que, por falta de tiempo, no se han podido llevar a cabo pero que la inclusión en versiones futuras no son una utopía. Entre esas ideas, destacan:

- Creación de un médico, en la plataforma web, por medio del número de colegiado, sin necesidad de introducir otros parámetros.
- Prescripción de recetas digitales: uno de los principales motivos para pedir una cita con los médicos, es la renovación de las recetas. La implementación de esta característica permitiría desahogar un poco el número de citas en los centros de salud.
- Calendario de medicamentos: Las personas, a medidas que envejecen van necesitando una mayor cantidad de medicamentos, y a veces se les hace difícil recordar en qué momento y a qué hora, deben tomarlos. Un calendario administrado por el médico y que se sincronizase con la app móvil, evitaría muchas equivocaciones.
- Recordatorios de citas: En varias ocasiones, las citas médicas con especialistas, no se conceden de un día para otro, y también pueden sufrir cambios. Esta característica podría ayudar en este sentido.
- Obtención de otros parámetros médicos, como la saturación de oxígeno y la temperatura por medio de otro tipo de sensores.
- Desarrollo de la app móvil para otros sistemas operativos.
- Habilitar la ejecución de los test de memoria en la propia plataforma web, facilitando la realización de los mismos para las personas mayores.
- Aplicación móvil de los médicos para la gestión de los pacientes.
- Más estadísticas.
- Estudio automático de los resultados.
- Comparación del comportamiento de los usuarios frente a un mismo test para poder detectar puntos en común y analizarlos.

Capítulo 10
Summary and Conclusions

A secure mobile application that allows a doctor interact with his patient , which makes use of different sensors to get different medical data such as: daily activity and heart rate. Attached to a website that allows, view patient data , it has been a challenge carried out with the purpose harness new technologies at the service of society.

The idea of designing and building a tool to help to prevent and diagnose diseases among patients who use it, is hopeful and has been extremely influential in taking care of the development of the application. The possibilities are endless and every grain of sand to be provided to the use of new technologies in medicine is important. You never know, maybe what develops at this point is very simple, but the idea can always improve and serve as the basis for a project closer to entering the real life. Also, the technology is advancing very fast and becoming cheaper and may be available to a wider audience. The price, as in most cases, is the limit, but now we have several bracelet apparatus as mentioned in the previous chapter or as fit-bit marks, which are entering the market and lowering prices. That is why, and foreseeing the emergence of more wearable devices, that we have tried to design, to the extent possible, that the tool is adjustable. The server has not been developed exclusively for connecting to a mobile phone with Android operating system, nor does the app need to use a unique brand of smartwatch to capture the data, being able to also use other smart devices. These last two features would not be possible without the importance of standards, for example in sending information on the Internet, JSON; or connections between devices: Bluetooth technology. This is perhaps the sign that all working in the same direction move faster and society can get more from the research and investment.

During the development of the mobile application and website ideas were appearing that, for lack of time, have not been carried out but their inclusion in future versions is not a utopia. Those ideas include:

- Creating a physician, on the web platform, through the membership number without introducing other parameters.
- Prescription of digital recipes: one of the main reasons for an appointment with doctors, is the renewal of recipes. The implementation of this feature would relieve a little the number of citations in health centers.
- Calendar of medications: People, aging measures will need a greater amount of drugs, and find it difficult sometimes to remember at what time , should they take their medication. A calendar administered by the doctor and that sincronizase with the mobile app, avoid many mistakes.
- Appointment Reminders: On several occasions, medical appointments with specialists, not granted from one day to another, and may also undergo changes. This feature could help in this regard.
- Obtaining other medical parameters such as oxygen saturation and temperature by means of other sensors.
- Mobile app development for other operating systems.
- Enable running memory tests on the web platform itself, facilitating the realization of the same for the elderly.
- Mobile App of the medical management of patients.
- More statistics.
- Automatic study of the results.
- Comparison of the behavior of users from a single test to detect and analyze commonalities

Bibliografía

[1] Chrome V8, Google's high perfomance, open source Javascript engine [online]. Disponible en: https://developers.google.com/v8/

[2] IBM developerWorks, ¿Simplemente qué es Node.js? [online]. Disponible en: https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/

[3] Express, Infraestructura de web rápida, minimalista y flexible para Node.js [online]. Disponible en: http://expressjs.com/es/

[4] Yeoman, the web's scaffolding toor for modern webapps [online]. Disponible en: http://yeoman.io/

[5] Generator express, an express generator for Yeoman [online]. Disponible en: https://github.com/petecoop/generator-express

[6] Gupl, página oficial [online]. Disponible en: http://gulpjs.com/

[7] Grunt, The JavaScript Task Runner [online]. Disponible en: http://gruntjs.com/

[8] Bower, A package manager for the web [online]. Disponible en: https://bower.io/

[9] MongoDB, página oficial en español [online]. Disponible en: https://www.mongodb.com/es

[10] Kantar, Smartphone OS sales market share [online]. Disponible en: http://www.kantarworldpanel.com/global/smartphone-os-market-share/

[11] Express-ejs-layout [online]. Disponible en: https://www.npmjs.com/package/express-ejs-layouts

[12] Asiermarquez.com, Conceptos sobre API REST [online]. Disponible en:

http://asiermarques.com/2013/conceptos-sobre-apis-rest/

[13] Mongoose, Página oficial [online]. Disponible en: http://mongoosejs.com/

[14] MongoJS, Repositorio en GitHub [online]. Disponible en: https://github.com/mafintosh/mongojs

[15] Mozilla Developer Network, Promise [online]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[16] Volley, Android: Transmitting Network Data Using Volley [online]. Disponible en: https://developer.android.com/training/volley/index.html

[17] HttpURLConnection, Android [online]. Disponible en: https://developer.android.com/reference/java/net/HttpURLConnection.html

[18] Fragments, Developers Android [online]. Disponible en: https://developer.android.com/guide/components/fragments.html

[19] Activity, Developers Android [online]. Disponible en: https://developer.android.com/reference/android/app/Activity.html

Apéndice A. Modelos

MODELO DOCTOR

```
/
**************************************************************************
*****
*
* doctor.js
*
**************************************************************************
******
*
* JOSÉ SAÚL GIFFUNI
*
*
* MAYO, 2016
*
*
* ESQUEMA DE LOS MÉDICOS
*
*
**************************************************************************
******/

var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var validators = require('mongoose-validators');
var uniqueValidator = require('mongoose-unique-validator');

var DoctorSchema = new Schema({
  username: {
    type: String,
    required: [true, 'El nombre de usuario es obligatorio'],
    unique: true
  },
  password: {
    type: String,
    minlength: [6, 'La contraseña debe tener al menos 6 dígitos'],
    required: [true, 'La contraseña es obligatoria']
  },
  name_and_surname: {
    type: String,
    maxlength: [200, 'No existen nombres tan largos'],
```

```
    required: [true, 'El campo, nombre y apellidos, es obligatorio']
  },
  colegiado: {
    type: Number,
    max: [10, 'El número puesto, sobrepasa a los médicos colegiados en
España'],
    required: [true, 'El número de colegiado es obligatorio'],
    unique: true
  },
  especialidad: {
    type: String,
    maxlength: [100, 'El campo es demasiado grande']
  },
  email: {
    type: String,
    required: [true, 'Falta rellenar el campo: email'],
    validate: validators.isEmail(),
    unique: true
  },
  mobile: {
    type: String,
    validate: {
      validator: function(phone){
        return /\d{3}-\d{2}-\d{2}-\d{2}/.test(phone);
      },
      message:'{VALUE} No es un número válido, debe seguir la estructura:
XXX-XX-XX-XX'
    }
  }
});

DoctorSchema.plugin(uniqueValidator, { message: 'El {PATH} ya se
encuentra en uso'});

mongoose.model('Doctor', DoctorSchema);
```

MODELO PACIENTE

```
/
**************************************************************************
*****
*
* doctor.js
*
**************************************************************************
```

```
******
*
* JOSÉ SAÚL GIFFUNI
*
*
* MAYO, 2016
*
*
* ESQUEMA DE LOS MÉDICOS
*
*
**************************************************************************
******/

var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var validators = require('mongoose-validators');
var uniqueValidator = require('mongoose-unique-validator');
var Doctor = mongoose.model('Doctor');


var PatientSchema = new Schema({
  name: {
    type: String,
    required: [true, 'El {PATH} es obligatorio']
  },
  lastName: {
    type: String,
    required: [true, 'El {PATH} es obligatorio']
  },
  dni: {
    type: String,
    required: [true, 'El {PATH} es obligatorio'],
    unique: true,
    validate: {
      validator: function(dni){
        return /([a-z,A-Z]{1}\d{7}[a-z,A-Z]{1})|(\d{8}[a-z,A-Z]
{1})/.test(dni);
      },
      message:'{VALUE} No es un DNI o NIE válido.'
    }
  },
  password: {
    type: String,
    require: [true, 'El {PATH} es obligatorio']
```

```
  },
  sex: {
    type: String,
    required: [true, 'El {PATH} es obligatorio']
  },
  birthDate: {
    type: Date,
    required: [true, 'El {PATH} es obligatorio']
  },
  mobileNumber: {
    type: String,
    validate: {
      validator: function(phone){
        return /\d{3}-\d{2}-\d{2}-\d{2}/.test(phone);
      },
      message:'{VALUE} No es un número válido, debe seguir la estructura:
XXX-XX-XX-XX'
    }
  },
  contactPerson: {
    type: String
  },
  personTlf: {
    type: String,
    validate: {
      validator: function(phone){
        return /\d{3}-\d{2}-\d{2}-\d{2}/.test(phone);
      },
      message:'{VALUE} No es un número válido, debe seguir la estructura:
XXX-XX-XX-XX'
    }
  },
  smoker: {
    type: Boolean,
    required: [true, 'El {PATH} es obligatorio']
  },
  memoryProblems: {
    type: Boolean,
    required: [true, 'El {PATH} es obligatorio']
  },
  heartCondition: {
    type: Boolean,
    required: [true, 'El {PATH} es obligatorio']
  },
  unSolvedQuizes: {
```

```
    type: [Schema.ObjectId]
  },
  solvedQuizes: {
    type: [Schema.ObjectId]
  },
  heartBeatDataQueue: {
    type: [Schema.ObjectId]
  },
  activityDataQueue: {
    type: [Schema.ObjectId]
  },
  doctor: { type: Schema.ObjectId, ref: "Doctor"}
});

PatientSchema.plugin(uniqueValidator, {message: 'El {PATH} ya se
encuentra en uso'});
mongoose.model('Patient', PatientSchema);
```

Apéndice B. Código destacado.

FUNCIÓN POSTUPLOAD

```
/
**************************************************************************
*****
*
* controllers/quiz.js
*
**************************************************************************
******
*
* JOSÉ SAÚL GIFFUNI
*
*
* MAYO, 2016
*
*
* Función controlador postUpload. Volca en la base de datos, los test.
*
*
**************************************************************************
******/
exports.postUpload = function (req, res, next){
  upload(req, res, function(err){
    if (err) {
      next(new Error(err));
    } else {
      if(req.file){
        console.log('Multen: Archivo subido sin errors');
        console.log('Ruta completa al archivo: ' + req.file.path);
        jsonfile.readFile(req.file.path, function(err, obj) {
          db.quizes.insert(obj, function (err) {
            if(err){
              console.log('Error: No se ha podido guardar el documento en
la bd');
              res.render('pages/quiz/index', {title: 'Subir un nuevo
test', partial: '../../partials/quiz/quizUpload', message: null, errors:
err.errors});
            } else {
              console.log('Correct: El documento se ha almacenado
correctamente en la bd');
              res.render('pages/quiz/index', {title: 'Subir un nuevo
test', partial: '../../partials/quiz/quizUpload', message: 'Documento
```

```
almacenado y volcado en la BBDD',errors: []});

            }
          });
        });
      } else {
        res.render('pages/quiz/index', {title: 'Subir un nuevo test',
partial: '../../partials/quiz/quizUpload', message: 'Debes elegir un
archivo.json',errors: []});
      }
    }
  });
};
```

FUNCIÓN PATIENT/SHOW

```
/
**************************************************************************
*****
*
* controllers/patient.js
*
**************************************************************************
******
*
* JOSÉ SAÚL GIFFUNI
*
*
* MAYO, 2016
*
*
* Función controlador show. En la que se hace un uso intensivo de las
* promesas.
*
**************************************************************************
******/
exports.show = function (req, res){
  var quizes;
  var usqCont = 0;
  var sqCont = 0;
  var unSolvedQuizes = [];
  var solvedQuizes = [];

  getOneUnSolvedQuiz = function(callback,callback2){
    get = new Promise (function(resolve, reject){
      db2.quizes.findOne({ _id:
mongojs.ObjectId(req.patient.unSolvedQuizes[usqCont])}, function(err,
```

```
doc) {
        if(err){
          reject(err);
        } else if(doc !== null){
          unSolvedQuizes.push(doc);
          resolve();
        }else {
          resolve();
        }
      });
    }).then(function(doc){
      usqCont++;
      console.log("Se ha cumplido la promesa, getOneUnsolvedQuizP; El
contador = " + usqCont);
      if(usqCont < req.patient.unSolvedQuizes.length){
        getOneUnSolvedQuiz(callback,callback2);
      }else {
        callback();
      }
    },function(err){console.log('Se ha producido un error en la promesa
getOneUnsolvedQuizP:' + err); callback2();});
  };

  var getAllUnSolvedQuizesP = new Promise(function(resolve,reject){
    getOneUnSolvedQuiz(resolve,reject);
  });

  getAllUnSolvedQuizesP.then(function (){
    console.log('getAllUnsolvedQuizesP se ha cumplido. Armacenando los
docs en quizes');
  }, function (err) {
    console.log('getAllUnsolvedQuizesP no se ha cumplido');
  });

  getOneSolvedQuiz = function(callback,callback2){
    get = new Promise (function(resolve, reject){
      dbSolvedQuises.solvedquizes.findOne({ _id:
mongojs.ObjectId(req.patient.solvedQuizes[sqCont])}, function(err, doc) {
        if(err){
          reject(err);
        } else if(doc !== null) {
          solvedQuizes.push(doc);
          resolve();
        } else {
          resolve();
        }
```

```
      });
    }).then(function(doc){
      sqCont++;
      console.log("Se ha cumplido la promesa, getOneSolvedQuizP; El
contador = " + sqCont);
      if(sqCont < req.patient.solvedQuizes.length){
        getOneSolvedQuiz(callback,callback2);
      }else {
        callback();
      }
    },function(err){console.log('Se ha producido un error en la promesa
getOneSolvedQuizP:' + err); callback2();});
  };

  var getAllSolvedQuizesP = new Promise(function(resolve,reject){
    getOneSolvedQuiz(resolve,reject);
  });

  getAllSolvedQuizesP.then(function (){
    console.log('getAllSolvedQuizesP se ha cumplido');
  }, function (err) {
    console.log('getAllSolvedQuizesP no se ha cumplido');
  });

  var getAllQuizesP = new Promise(function(resolve,reject){
    db2.quizes.find(function (err, docs) {
      if (err) {
        reject(err);
      } else {
        resolve(docs);
      }
    });
  });

  getAllQuizesP.then(function (docs){
    console.log('allQuizesP se ha cumplido. Armacenando los docs en
quizes');
    quizes = docs;
  }, function (err) {
    console.log('allQuizesP no se ha cumplido. Error: ' + err);
  });

  Promise.all([getAllQuizesP, getAllUnSolvedQuizesP,
getAllSolvedQuizesP]).then(function(){
    res.render('pages/patient/show', { title: 'Datos del paciente',
patient: req.patient, solvedQuizes: solvedQuizes, unSolvedQuizes:
```

```
unSolvedQuizes, quizes: quizes});
  }, function(){res.send('Se ha producido un error');});

};
```

CLASE FIRMAR

```
/
**************************************************************************
*****
*
* firmaDigital/Firmar.java
*
**************************************************************************
******
*
* JOSÉ SAÚL GIFFUNI
*
*
* JUNIO, 2016
*
*
* Clase Firmar, uso de curvas elipticas.
*
**************************************************************************
******/
  public class Firmar {

    private static final String clavePrivada_ =
"MEECAQAwEwYHKoZIzj0CAQYIKoZIzj0DAQcEJzAlAgEBBCA36tW9G38S3nsda0xu82FI4eiq
fcmarCeZDKWP0u3ljw==";
    private static final String clavePublica_ =
"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEbANPZ/m6DDJKt3QFYMIzHOeGzoJ0avpVCdDv
2JY3VOMoavbqxVk0aS/jOI5lUmt5k9sasYtFgQ9bqHYVTilmRQ==";


    public static String firmar(String mensaje) throws
UnsupportedEncodingException, NoSuchAlgorithmException,
InvalidKeySpecException, InvalidKeyException, SignatureException{
        PrivateKey privateKey = obtenerPrivateKey();
        Signature firma = crearFirma(privateKey);
        String msgFirmado = mensajeFirmado(mensaje, firma);
        System.out.println("El mensaje es: \n" + mensaje);
        System.out.println("Mensaje firmado: \n" + msgFirmado);
        return msgFirmado;
```

```
    }

    private static PrivateKey obtenerPrivateKey() throws
UnsupportedEncodingException, NoSuchAlgorithmException,
InvalidKeySpecException{
        byte[] clear = Base64.decode(clavePrivada_.getBytes("UTF-
8"),Base64.DEFAULT);
        PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(clear);
        KeyFactory fact = KeyFactory.getInstance("EC");
        PrivateKey privateKey = fact.generatePrivate(keySpec);
        return privateKey;
    }

    private static PublicKey obtenerPublicKey() throws
UnsupportedEncodingException, NoSuchAlgorithmException,
InvalidKeySpecException {
        byte[] encKey = Base64.decode(clavePublica_.getBytes("UTF-
8"),Base64.DEFAULT);
        X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(encKey);
        KeyFactory keyFactory = KeyFactory.getInstance("EC");
        PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);
        return pubKey;
    }

    private static Signature crearFirma(PrivateKey privateKey) throws
NoSuchAlgorithmException, InvalidKeyException{
        Signature firma = Signature.getInstance("SHA1withECDSA");
        firma.initSign(privateKey);
        return firma;
    }

    private static String mensajeFirmado(String mensaje, Signature firma)
throws UnsupportedEncodingException, SignatureException{
        byte[] strByte = mensaje.getBytes("UTF-8");
        firma.update(strByte);
        byte[] realSig = firma.sign();
        String sign = Base64.encodeToString(realSig,Base64.DEFAULT);
        return sign;
    }

}
```

Apéndice C. Repositorios de código.

REPOSITORIO DEL SERVIDOR

- **GitHub**

REPOSITORIO DE LA APP MÓVIL

- **GitHub**