Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención de Sistemas)

Aplicación de metodología Scrum para el desarrollo de una API web de gestión médica con MEAN Stack

Autor:
D. Óscar Fernández Aranda
Tutor:
D. César Vaca Rodríguez

RESUMEN

Este proyecto describe el desarrollo de una aplicación web de gestión de citas e historiales médicos, conocido como HCE (EMR o EHR en inglés), empleando la metodología de desarrollo ágil Scrum. El proyecto es llevado a cabo en varios 'Sprints' de los que obtenemos un prototipo. El sistema está implementado con el stack MEAN.

Palabras clave: HCE, CRUD, API, REST, JSON, HTTP, Middleware, MEAN, NPM, SCRUM, Sprints.

ABSTRACT

This project describes the development of a web application for managing appointments and medical records, known as EMR or EHR (HCE in spanish), developed using the Agile Scrum development methodology. The project is carried out in several 'Sprints' from which we obtain a prototype. The system is implemented with the MEAN stack.

Keywords: EMR/EHR, CRUD, API, REST, JSON, HTTP, Middleware, MEAN, NPM, SCRUM, Sprints.

AGRADECIMIENTOS

Me gustaría agradecer a la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Valladolid por todos los conocimientos que me han brindado durante esta etapa de formación universitaria.

Agradecer a mi familia, amigos y pareja su apoyo incondicional ya que sin su ayuda se habría hecho más difícil llegar hasta aquí.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

El proyecto consiste en el desarrollo de un sistema de gestión médica mediante la utilización de la metodología de desarrollo ágil Scrum. En este documento se describen las distintas fases del proyecto así como el contexto que lo rodea en cuanto a tecnologías y metodologías.

1.1. Contexto

Este proyecto se sitúa dentro del área de los sistemas de información si se tiene en cuenta la siguiente definición: "Un sistema de información es un conjunto de elementos que interactúan entre sí con el fin de apoyar las actividades de una empresa o negocio."

En este caso se trata de un HCL (en inglés EMR o EHR), aplicación que recopila en un sistema web la información de salud de pacientes y gestiona la comunicación entre médicos y pacientes.

Por otra parte, para el desarrollo de la aplicación se ha optado por seguir metodologías ágiles de desarrollo de software, especialmente Scrum. Actualmente las metodologías ágiles son una tendencia en auge frente a métodos más tradicionales.

Todas las metodologías que se consideran ágiles cumplen con el manifiesto ágil que no es más que una serie de principios que se agrupan en 4 valores:

1. Los individuos y su interacción, por encima de los procesos y las herramientas.
2. El software que funciona, frente a la documentación exhaustiva.
3. La colaboración con el cliente, por encima de la negociación contractual.
4. La respuesta al cambio, por encima del seguimiento de un plan.

Con estas premisas el objetivo es incrementar la eficiencia de las personas involucradas en el proyecto, minimizando el coste de éste. Especialmente se pueden encontrar beneficios en las metodologías ágiles frente a las clásicas en proyectos nuevos y desconocidos, es decir, proyectos en los cuales se carece de la experiencia de desarrollos similares o iguales, lo que implica innovar y adentrarse en lo desconocido. En este punto es donde la flexibilidad que aportan las metodologías ágiles, junto a la participación del cliente a la hora de decidir los cauces que toma el desarrollo, proporciona un marco muy ventajoso al proyecto, facilitando el éxito y moderando los costes.

1.2. Motivaciones

La motivación para la elección y desarrollo de este proyecto ha sido utilizar los conocimientos adquiridos durante la formación universitaria e investigar las tecnologías más usadas en el ámbito empresarial, con el objetivo de plantear una posible solución a un problema que me he encontrado en varias ocasiones tras diferentes cambios de localidad y que a todo el mundo le podría ocurrir: no tener el historial médico propio actualizado, centralizado y con posibilidad de acceder a él en todo momento.

Por otra parte, la elección de utilizar el uso de metodologías ágiles de desarrollo de software y de proyectos, Scrum en este caso, es debido a las ventajas que ofrecen frente a métodos clásicos.

1.3. Objetivo

El objetivo que persigue este proyecto es el desarrollo de una aplicación web orientada para profesionales médicos autónomos y pequeñas clínicas e implementada con el Stack MEAN, que se encargue de la gestión médica, ofreciendo la solicitud de citas y la consulta del historial médico para usuarios pacientes, y permita a usuarios médicos ver su agenda de citas, consultar la historia clínica de un paciente y generarle nuevos informes tras la consulta.

Se pone en práctica la metodología de desarrollo Scrum, basándose principalmente en este modelo para la planificación y desarrollo de la aplicación.

De esta manera se optimizarán los procesos clínicos permitiendo el acceso a toda la información de forma precisa, confiable y asegurando la integridad de los datos. Además posibilita la oportunidad de compartir la información entre los miembros de la aplicación de diferentes especialidades médicas.

Este objetivo se subdivide en una serie de subobjetivos que son:

- **Utilización de una metodología ágil de desarrollo:** se busca aprovechar las ventajas que ofrecen este tipo de metodologías en el desarrollo, aprovechando su flexibilidad.
- **Desarrollo de una aplicación utilizable por el usuario medio:** con este objetivo se busca que cualquier empleado de la empresa pueda hacer uso del sistema sin necesidad de ser un usuario experto.
- **Modularidad:** la aplicación en si será una sola, pero debe tener sus funciones moduladas para facilitar tanto su mantenimiento como cualquier posible evolución.
- **Persistencia de la información:** los datos introducidos en el sistema deben ser persistentes entre sesiones y usuarios.

1.4. Estructura

El documento se divide en una serie de capítulos que a su vez se subdividen en apartados:

1. **Introducción:** da una visión global del proyecto, exponiendo los objetivos a alcanzar, él porque de la realización de este proyecto y el contexto.

2. **Estado del arte:** expone el contexto tecnológico que rodea al proyecto, detallando diferentes opciones disponibles e introduciendo al lector a las principales ofertas que existen en el mercado en relación al proyecto.

3. **Análisis del sistema:** es un estudio previo al desarrollo en el cual se detalla las propiedades del sistema y los elementos que se deben desarrollar.

4. **Desarrollo del sistema:** en este capítulo se detalla todo lo relativo al propio desarrollo del proyecto.

5. **Planificación y presupuesto:** se expone la planificación previa basada en las estimaciones para la duración del proyecto y se realiza un pequeño análisis económico de éste para hallar el coste final de su realización.

6. **Conclusiones:** se plasman las conclusiones extraídas después de la finalización del proyecto, tanto personales como de resultados. También se proponen desarrollos futuros que enriquezcan al sistema.

2. ESTADO DEL ARTE

En este apartado se presentan las tecnologías y metodologías relacionadas con el proyecto de este documento con el fin de acercar al lector a estos conceptos y al contexto que rodea al proyecto en sí, de forma que pueda comprender con más facilidad el resto del documento.

El proyecto está basado en el desarrollo de un pequeño sistema de gestión médica con stack MEAN empleando la metodología de desarrollo ágil Srum, por ello, en este capítulo se hablará de lo que es un sistema de gestión médica y los diferentes productos que existen en el mercado actualmente, de los componentes del stack MEAN y algunos ejemplos de grandes compañias tecnológicas que lo utilizan y en que consiste la metodología Scrum.

2.1. Aplicaciones de gestión médica

El término software médico o Historia Clínica Electrónica (HCE) se utiliza para referirse a una plataforma de software que gestiona los registros de los actos médicos de los pacientes, en un centro médico, hospital o clínica. Es un registro electrónico de la información de salud del paciente, generada por una o más visitas médicas en urgencias, consultas, cirugías o pruebas diagnósticas.

Esta secuencia de registros médicos, ordenada cronológicamente o por patología, típicamente incluye los datos demográficos del paciente, problemas personales o familiares, notas de progreso, los medicamentos, los signos vitales, vacunas, datos de laboratorio e informes o imágenes de radiología.

El software médico automatiza y agiliza el flujo de trabajo del profesional de la salud. Ofrece la capacidad de generar un registro completo de un encuentro clínico del paciente, así como el apoyo a otras actividades relacionadas con el cuidado directa o indirectamente a través de la interfaz, gestión de la calidad y resultados de informes.

La mayor ventaja de un buen software médico es evitar la pérdida de la información clínica del paciente, evita errores médicos por falta de comprensión de la letra de otros doctores, emite de forma correcta las recetas de medicamentos, genera comunicación y satisfacción en los pacientes y aumenta la productividad de la actividad clínica.

Existen varios estándares para un sistema HCE [**2**], el cual tiene una estructura compleja y por ello los sistemas o servicios de HCE incorporan muchos elementos de información. En consecuencia, existen diferentes conjuntos de normas que se aplican a los diferentes componentes del sistema. Entre estos cabe destacar:

- Estándares de contenidos y estructura (arquitectura).

- Representación de datos clínicos (codificación).
- Estándares de comunicación (formatos de mensajes).
- Seguridad de datos, confidencialidad y autentificación.

Existen diferentes organizaciones reconocidas en el ámbito internacional que dedican parte de su actividad a tareas de normalización en informática médica y sistemas de Historia Clínica Electrónica [**2**]:

- ISO que es una organización de ámbito mundial.
- CEN (Comité Europeo de Normalización) en el que participan organismos nacionales como es el caso de AENOR en España. El estándar más importante en Europa en el ámbito de la informática médica es el CEN/ISO 13606.
- ANSI (American Nacional Standard Institute) es el organismo oficial de EEUU que gestiona las actividades nacionales de normalización en informática para la salud mediante el HISPP (Healthcare Informatics Standard Planning Panel).
- Comités internacionales como IT14 (Australia) o MEDIS-DC del MITI (Japón).
- Organización HL7 con su estándar HL7 y FHIR [**3**].

Para los médicos autónomos, hospitales o consultorios, la adopción de un software médico que utilicen o no estos estándares es una inversión de enormes proporciones en tiempo y dinero. Sin embargo, a fin de mantenerse al día con la tecnología y evitar problemas legales, la adopción de un buen software médico y utilización de estándares hoy en día es un avance necesario.

2.1.1. Productos existentes

Algunos de los programas de Historia Clínica Electrónica para los doctores o para las clínicas multi-especialidad son ERP-Izaro, DriCloud, Infomed, Salus, MNprogram, Ofimedic y Axon.

Indudablemente hay más en el mercado y cada día aumenta la oferta sobre este tipo de software ERP, pero estos incluyen un alto nivel de calidad, ya que se ajustan a la normativa legal Europea, que es la normativa legal de protección de datos más estricta del mundo y no todos los programas informáticos de gestión clínica pueden cumplir con esta legislación.

2.2. Implementación Ágil

La implementación ágil se realiza de una forma incremental o por fases. El principal compromiso de la implementación de la metodología ágil es evitar los obstáculos y problemas asociados a la implementación big bang[1]. La metodología ágil se basa en la simplicidad, en entregar las funcionalidades operativas del software lo más rápido posible, empezando por los componentes más importantes del negocio. La implementación ágil consiste en dos fases: análisis y realización de Sprint como se muestra en la Figura .

2.2.1. 1^a Fase: Análisis

Esta fase consiste en 4 pasos:

1. Preparación del proyecto durante el cual se definen los diferentes elementos, tales como labores,responsabilidades, estándares para la documentación y requisitos del hardware.

2. Proceso de visualización en el que se identifican cuidadosamente todos los procesos operativos y procesos que dependen de condiciones específicas, como seguridad, autorizaciones e interfaces. Estos resultados serán la base para la construcción de unos cimientos sólidos para el proyecto entero.

3. Funcionamiento de referencia del sistema, el cual se basará en el software de ERP.

4. Fase de evaluación. En esta fase, se determina la prioridad de los requisitos adicionales y las funcionalidades, en orden del valor del negocio. Después de esto, el equipo de implementación estima el esfuerzo que será necesario para realizar esto y determina la planificación de los sprints para que se suministren los componentes del sistema.

2.2.2. 2^a Fase: Realización del Sprint

Esta fase consiste en 5 pasos:

1. Realización de los requisitos, de pruebas y documentación.

2. Reuniones diarias del estado del proyecto. En esta fase se registra el estado del proyecto y se discuten los diversos obstáculos que el equipo ha podido encontrar.

3. Sesión de prueba del Sprint. Durante esta fase los usuarios y el equipo IT determinan si los procesos cumplen los requisitos.

4. Se llevará a cabo una revisión del Sprint para comprobar que se puede mejorar en los futuros sprints.

[1] Gran cantidad de planificación tiene que ser hecha antes de la aplicación.

Fig. 2.1. Ciclo de entrega en desarrollo agil.

2.3. Metodología de desarrollo

2.3.1. ¿Qué es Scrum?

Scrum es una metodología ágil para el desarrollo de software o la gestión de proyectos, más concretamente un marco de trabajo a través del cual las personas pueden abordar problemas complejos adaptativos, a la vez que se entregan productos de forma eficiente y creativa con el máximo valor [**4**].

Scrum esta compuesto de procesos que se utilizan para gestionar el trabajo de productos complejos desde principios de los años 90. Scrum no es un proceso, una técnica, o método definitivo, sino un marco de trabajo donde se pueden emplear un conjunto de diferentes procesos y técnicas. Scrum muestra la eficacia relativa de las técnicas de gestión de producto y de trabajo de modo que podemos continuamente mejorar el producto, el equipo y el entorno de trabajo. El marco de trabajo Scrum se compone por los Equipos Scrum, sus Roles, Eventos, Artefactos y Reglas asociadas. Cada componente dentro del marco de trabajo sirve a un propósito específico y es esencial para el éxito de Scrum y para su uso.

Scrum se ha utilizado para el desarrollo de software, hardware, software embebido, redes de funciones interactiva, vehículos autónomos, escuelas, gobiernos, márqueting, gestión operacional de las organizaciones y en casi todo lo que utilizamos en nuestra vida diaria, como individuos y sociedades.

La esencia de Scrum es pequeños equipos de personas. El equipo individualmente es muy flexible y adaptivo. Estas fortalezas continúan operando en equipos individuales, varios, muchos, y redes de equipos que desarrollan, lanzan, operan y mantienen el trabajo y el producto de trabajos de miles de personas. Ellos colaboran e inter-operan a través del desarrollo de sofisticadas arquitecturas y objetivos en entornos de desarrollo.

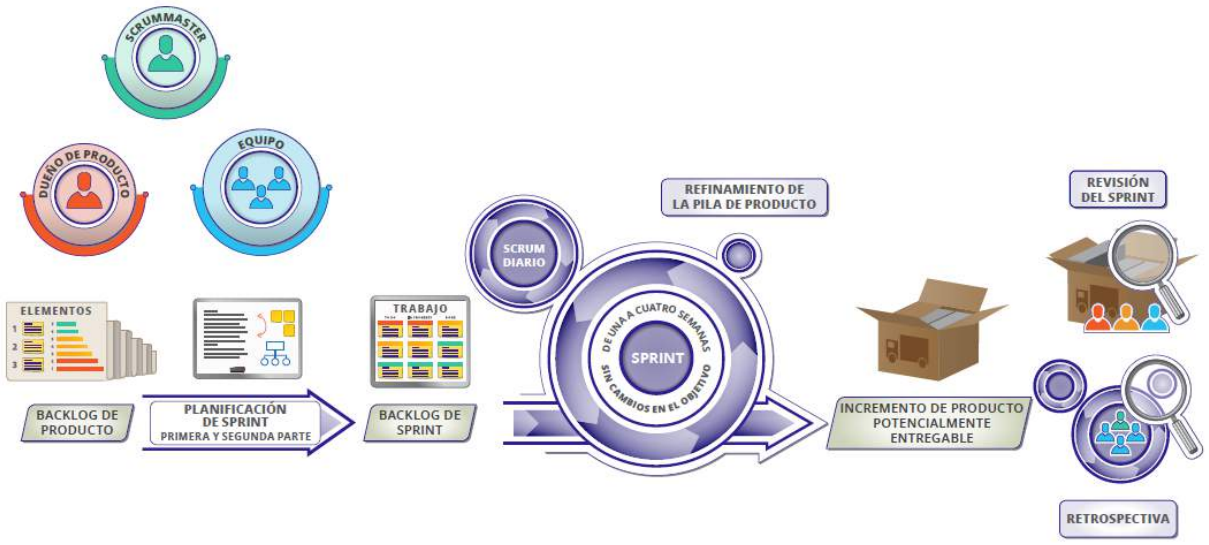

Fig. 2.2. Ciclo de entrega de producto en SCRUM.

2.3.2. ¿Qué caracteriza a Scrum?

De todas las metodologías ágiles, Scrum es única porque introduce la idea del control empírico de los procesos. Esto significa que Scrum utiliza el progreso real de un proyecto para planificar y concertar los lanzamientos. En Scrum, los proyectos se dividen en ritmos de trabajo breves, conocidos como sprints. Normalmente, tienen una, dos o tres semanas de duración. Al final de cada sprint, el cliente y los miembros del equipo se reúnen para evaluar el progreso del proyecto y planear los siguientes pasos a seguir. Esto permite que la dirección del proyecto se ajuste o se reoriente una vez finalizado el trabajo, sin especulaciones ni predicciones.

Filosóficamente, este énfasis continuo de evaluar las tareas finalizadas es el principal causante del éxito que tiene esta metodología entre los directores y desarrolladores. Pero lo que verdaderamente permite funcionar a la metodología Scrum es un conjunto de papeles, responsabilidades y reuniones.

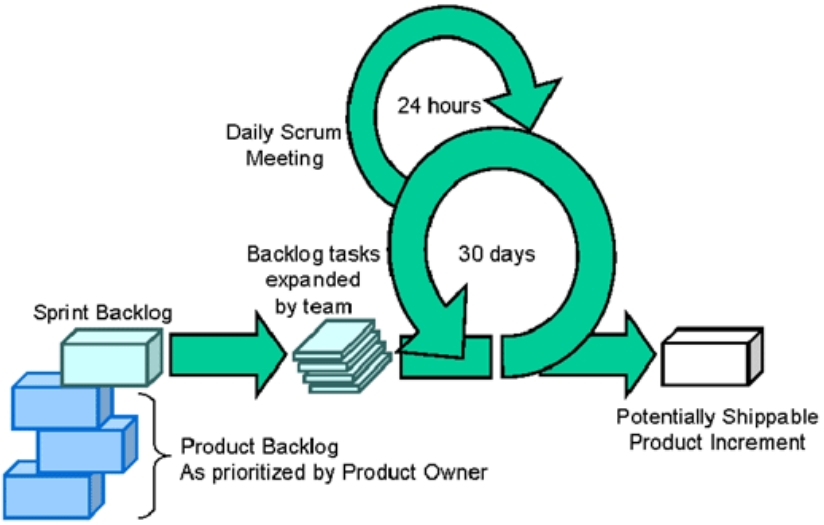

Fig. 2.3. Flujo de artefactos en SCRUM.

2.3.3. Los papeles de Scrum

Scrum tiene tres papeles fundamentales: Product Owner (propietario del producto), Scrum Master (especialista en Scrum) y Team Member (miembros del equipo) [10].

- Product Owner: En Scrum, el Product Owner se encarga de comunicar la visión del producto al equipo de desarrollo. Su papel también debe representar el interés del cliente por medio de los requisitos y la priorización. Es quien más autoridad tiene de los tres papeles en Scrum, por ende el que mayor responsabilidad recibe, en consecuencia el Product Owner es el individuo que tiene afrontar las consecuencias cuando un proyecto va mal.

La autoridad y responsabilidad del Product Owner le hace difícil conseguir el balance correcto de implicación. Como Scrum evalúa la auto-organización entre equipos, el Product Owner debe evitar supervisar hasta el último detalle. Al mismo tiempo, el Product Owner tiene que estar a la disposición de las preguntas del equipo.

- Scrum Master: El Scrum Master actúa como enlace entre el Product Owner y el equipo. El Scrum Master no dirige al equipo. Su papel se encarga de evitar cualquier barrera que impida al equipo lograr sus objetivos de sprint. Hace que el equipo sea creativo y productivo, a la vez que permite que los logros del equipo sean visibles ante el Product Owner. El Scrum Master también aconseja al Product Owner sobre cómo maximizar el ROI (Return Of Investment) para el equipo.

- Team Member: En la metodología Scrum, el equipo es el responsable de terminar el trabajo. Idealmente, los equipos están formados por siete miembros multifuncionales, más/menos dos personas. Para proyectos de software, un equipo habitual sería una mezcla de ingenieros de software, arquitectos, programadores, analistas, testers, y diseñadores de UIs. En cada

sprint, el equipo es responsable de determinar cómo va a lograr acabar el trabajo. Esto garantiza al equipo un grado de autonomía, pero, al igual que pasa con la situación del Product Owner, esta libertad viene acompañada por la responsabilidad de cumplir los objetivos del sprint.

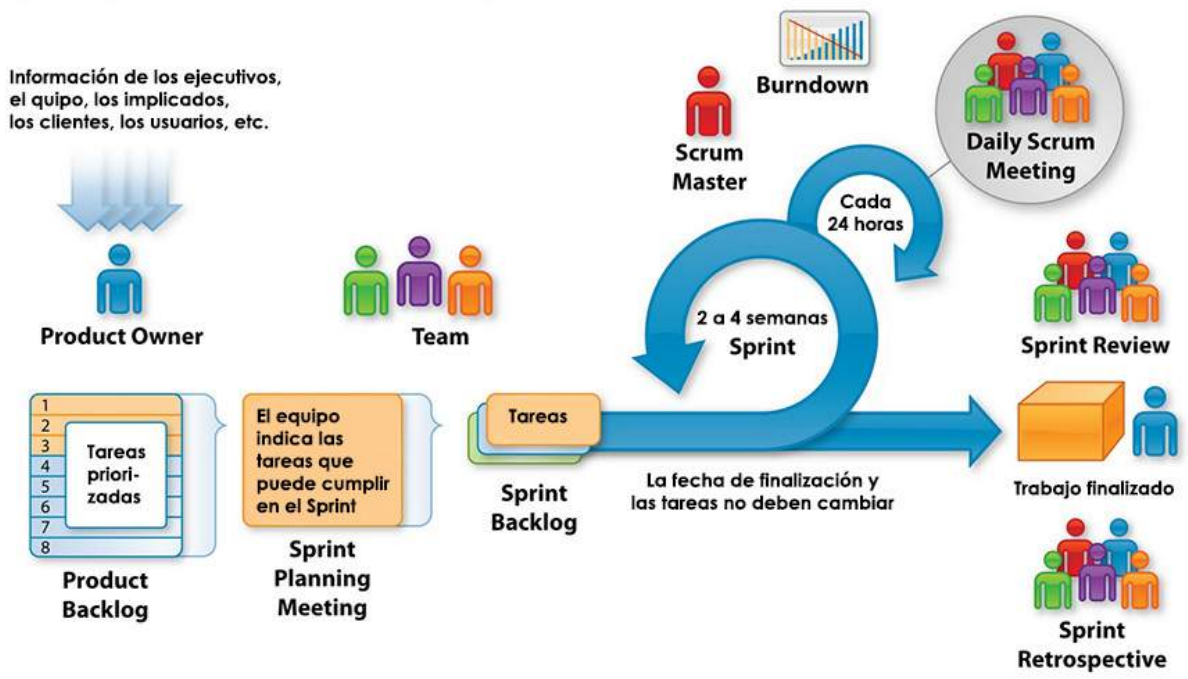

Fig. 2.4. Participación de los diferentes papeles en el desarrollo en SCRUM.

2.4. API Rest

El sistema REST es un protocolo de intercambio y manipulación de datos en los servicios de internet. Casi toda empresa o aplicación dispone de una API REST para creación de negocio.

En la actualidad no existe proyecto o aplicación que no disponga de una API REST para la creación de servicios profesionales a partir de ese software. Twitter, YouTube, los sistemas de identificación con Facebook... hay cientos de empresas que generan negocio gracias a REST y las APIs REST. Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de APIs para servicios de Internet.

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también mucha complejidad.

2.4.1. Características de REST:

- Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla, aunque algunas aplicaciones HTTP incorporan memoria caché.

Se configura con lo que se conoce como protocolo cliente-caché-servidor sin estado: existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un futuro la misma respuesta para peticiones idénticas.

- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).

- Los objetos en REST siempre se manipulan a partir de la URI. La URI es el elemento identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.

- Interfaz uniforme: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.

- Sistema de capas: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.

- Uso de hipermedios: el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Para cualquier API REST es obligatorio disponer del principio HATEOAS (Hypermedia As The Engine Of Application State - Hipermedia Como Motor del Estado de la Aplicación) para ser una verdadera API REST. Este principio es el que define que cada vez que se hace una petición al servidor y éste devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente.

A continuación vermos una petición a una API REST según el principio HATEOAS (enlaza a un tutorial explicativo del concepto de hipermedia en API REST con un ejemplo práctico de una petición a una base de datos de automóviles):

```
1  {
2    "id": 78,
3      "nombre": "Francisco",
4      "apellido": "Matas",
5      "coches": [
6          {
7              "coche": "http://miservidor/concesionario/api/v1/clientes/78/
                   coches/1033"
8          },{
9              "coche": "http://miservidor/concesionario/api/v1/clientes/78/
                   coches/3889"
10         }
```

```
11      ]
12  }
```

2.4.2. Ventajas que ofrece REST para el desarrollo

1. Separación entre el cliente y el servidor:

El protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

2. Visibilidad, fiabilidad y escalabilidad.

La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

3. La API REST siempre es independiente del tipo de plataformas o lenguajes:

La API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

2.5. Entorno, herramientas de desarrollo y elaboración del documento

A continuación vemos el entrono en el que se desarrolla este trabajo y que herramientas se han utilizado.

2.5.1. Manjaro Linux

Fig. 2.5. Logotipo Manjaro.

Es un sistema operativo de código abierto para computadoras basada en la distribución de Linux 'Arch Linux'.

Manjaro Linux se enfoca en la facilidad de uso y accesibilidad cuyo sistema en sí está diseñado para funcionar completamente con su variedad de software preinstalado, contando con un modelo de actualización de versión móvil y utilizando pacman como su administrador de paquetes. Los repositorios se administran con su propia herramienta llamada BoxIt, que está diseñada como git.

El modelo de lanzamiento continuo hace que el usuario no necesite reinstalar el sistema para mantenerlo actualizado. La administración de paquetes es manejada por pacman a través de la línea de comandos (terminal), y las herramientas frontend del administrador de paquetes GUI, llamadas Pamac (para su edición predeterminada de Xfce) y Octopi (para su edición de KDE).

El lanzamiento actual de Manjaro Linux, en el cual esta desarrollado este proyecto, es la versión 17.1, llamado "Hakoila", lanzado el 31 de diciembre de 2017.

Manjaro Linux se basa en Arch Linux, teniendo su propia colección de repositorios y compatible con los de Arch Repositories. Manjaro usa tres conjuntos de repositorios:

- Inestables que contienen la mayoría de los paquetes Arch, posiblemente uno o dos días con retraso.
- De prueba que contienen paquetes de repositorios inestables sincronizados cada semana, lo que proporciona un cribado inicial.
- Estables que solo contienen paquetes que el equipo de desarrollo considera estables.

2.5.2. Visual Studio

Fig. 2.6. Logotipo Visual Studio.

Microsoft Visual Studio es un entorno (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación tales como C++, C#, Visual Basic .NET, Java, Python, Ruby, PHP; al igual que entornos de desarrollo web como ASP.NET MVC, Django,

etc., a lo cual sumarle las nuevas capacidades online bajo Windows Azure en forma del editor Monaco.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos, consolas, etc.

Para el desarrollo de la aplicación en este proyecto se utilizará el lenguaje JavaScript.

2.5.3. MEAN

Fig. 2.7. Logotipo Stack MEAN.

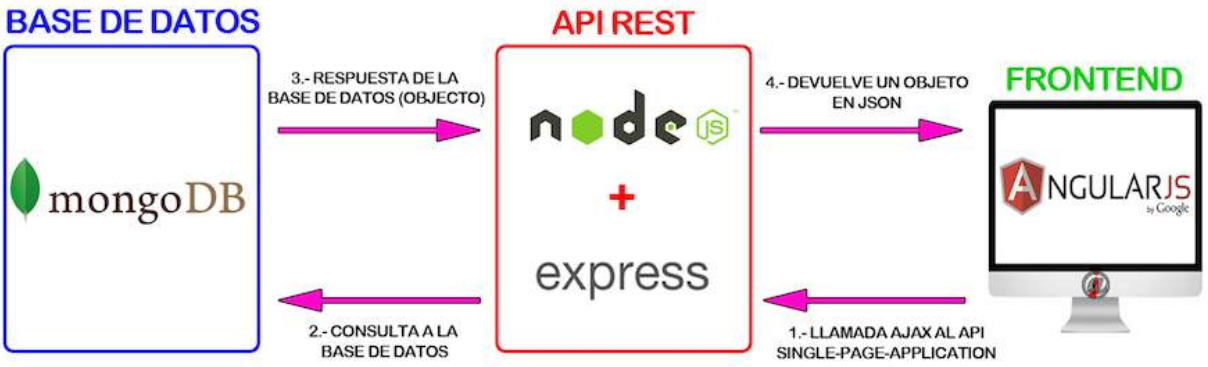

Fig. 2.8. Interacción entre frameworks Stack MEAN.

MEAN (acrónimo para: MongoDB, ExpressJS, AngularJS, NodeJS), es un framework o conjunto de subsistemas de software para el desarrollo de aplicaciones y páginas web dinámicas.

MEAN es una solución muy potente y full stack Javascript, formada por cuatro grandes bloques, MongoDB como la base de datos, Express como el framework backend, AngularJS como el framework frontend y Node.js como el servidor. Desde el cliente al servidor y finalmente a la base de datos, todos tienen un mismo lenguaje en común: Javascript. Juntos nos garantizán la flexibilidad necesaria y adecuada a los desarrollos de software actuales.

Cada subsistema del Mean stack es de código abierto y de uso gratuito:

- MongoDB:

Fig. 2.9. Logotipo MongoDB.

MongoDB es una base de datos NoSQL ágil y escalable. El nombre Mongo viene de la palabra “humongous” la cual enfatiza la escabilidad y desempeño que MongoDB provee.

Está basada en un modelo de documentos NoSQL, lo que significa que los datos son almacenados como objetos JSON en vez de las tradicionales filas y columnas de una base de datos relacional. MongoDB ofrece gran capacidad de almacenamiento de back-end para los sitios web de alto tráfico que necesitan almacenar datos tales como los comentarios de los usuarios , blogs , u otros artículos , ya que es rápidamente escalable y fácil de implementar [**23**].

Estas son algunas razones por las que MongoDB se ajusta perfectamente en el stack de NodeJS:

- Orientación a Documentos: Dado que MongoDB es una base de datos orientada a objetos, los datos son almacenados en un formato que es muy cercano al que se requiere para los scripts tanto del servidor como del cliente. Esto elimina la necesidad de transferir datos de columnas a objetos y viceversa.
- Alto desempeño: MongoDB es una de las principales bases de datos de alto desempeño disponibles al momento. Especialmente hoy, donde más y más gente interactúa con sitios web, es importante tener un backend que pueda soportar tráfico pesado.
- Alta disponibilidad: El modelo de replicación de datos de MongoDB hace muy sencilla la tarea de escalabilidad mientras mantiene el alto desempeño.
- Alta escalabilidad: La estructura de MongoDB hace fácil la escalabilidad horizontal a través dela distribución de datos a través de múltiples servidores[**23**].

Un ejemplo de un documento en JSON es:

```
1 {
2  id: "cd132450cafdef",
3  nombre: "foo",
4  alias: "bar",
```

```
 Direccion:
  {
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
  }
}
```

- Compass: es una aplicación que nos permite analizar los datos y la estructura de las colecciones de forma visual, así como realizar consultas y conectarnos con MongoDB Atlas[2].

- ExpressJS:

Fig. 2.10. Logotipo ExpressJS.

El módulo de Express actúa como el servidor web dentro del stack MEAN. Dado que corre sobre NodeJS, es sencillo de configurar, implementar y controlar.

El módulo de Express extiende NodeJS para proveer muchos componentes clave en el manejo de peticiones web. Por ejemplo, provee la habilidad para definir rutas de destino (URLs) para el consumo de recursos web.

También provee gran funcionalidad en términos de trabajo con peticiones HTTP y objetos de respuesta, incluyendo algunos como cookies y encabezados HTTP.

Además, permite implementar un servidor web corriendo en NodeJS en poco tiempo y con pocas líneas de código.

Algunas características disponibles en Express son:

- Manejo de rutas: Express hace muy sencilla la definición de rutas (URLs) que ligan directamente los scripts de NodeJS con la funcionalidad sobre el servidor.

[2]Base de datos como servicio que permite implementar, utilizar y escalar una base de datos de MongoDB

- Manejo de errores: Express provee manejo de errores para áquellos comunes como “documento no encontrado” y otros más.
- Integración sencilla: Un servidor de Express puede ser fácilmente implementado detrás de un sistema proxy existente, como Nginx or Varnish. Esta característica permite su fácil integración en entornos de seguridad pre existentes.
- Cookies: Express provee un manejo de cookies muy sencillo.
- Administración de sesiones y caché: Express también soporta manejo de sesiones y administración de caché.

- AngularJS:

Fig. 2.11. Logotipo AngularJS.

AngularJS es un framework para el cliente desarrollado por Google. Provee toda la funcionalidad necesaria para manejar las entradas del usuario en el navegador, manipular datos del lado del cliente, y controlar como los elementos son desplegados en la vista del navegador.

Está escrito en JavaScript, con una librería de JQuery reducida. La teoría detrás de AngularJS es proveer un framework que haga sencilla la implementación de aplicaciones web siguiendo el patrón MVC.

Algunos de los beneficios de utilizar AngularJS son los siguientes:

- Ligado de datos: AngularJS tiene un método muy claro para ligar datos a elementos de HTML.
- Extensibilidad: La arquitectura de AngularJS permite extender fácilmente casi cualquier aspecto del lenguaje para proveer implementaciones personalizadas.
- Limpieza: AngularJS impone una escritura de código clara.
- Código Reusable: La combinación de extensibilidad y limpieza hace muy sencillo escribir código reusable en AngularJS. De hecho, el lenguaje frecuentemente impone la reutilización cuando se crean servicios personalizados.
- Soporte: Google invierte una gran cantidad de recursos en éste proyecto, lo cual le da una ventaja significativa sobre sus competidores.

- Compatibilidad: AngularJS está basado en JavaScript y tiene una relación muy estrecha con JQuery.

Esto hace sencillo la integración de AngularJS en un ambiente de desarrollo y reutilizar piezas de código existente dentro de la estructura del framework de AngularJS [**23**].

Angular también tiene la capacidad de hacer el enrutamiento del sistema y claro está también incorpora la función de ajax (asincronico javascript y xml), para hacer peticiones al servidor sin tener que cargar toda la pagina.

AngularJS junto con otros frameworks MVC han revolucionado el desarrollo web a través de la transformación de lo que una vez fue código de frontend no mantenible en una base de código estructurado que puede soportar paradigmas de desarrollo más avanzados [**18**].

- NodeJS:

Fig. 2.12. Logotipo NodeJS.

NodeJS es una plataforma de desarrollo basada en el motor de JavaScript V8 de Google. Se puede escribir prácticamente todo el código que habrá de ejecutarse del lado del servidor en NodeJS, incluyendo el servidor web, los scripts de backend del servidor y cualquier funcionalidad de apoyo a la aplicación web.

El hecho de que el servidor web y los scripts de apoyo de la aplicación web estén corriendo juntos permite una mayor integración. Además, el servidor web puede correr directamente en la plataforma de NodeJS, lo que significa que a diferencia por ejemplo de Apache, es mucho más sencillo habilitar nuevos servicios o scripts a ejecutarse del lado del servidor[**23**].

Con la aparición de NodeJS, los desarrolladores de JavaScript pueden acceder al sistema de archivos, abrir sockets de red y crear procesos hijos [**21**].

Algunas características de NodeJS son las siguientes:

- Javascript end to end: Una de las principales ventajas de NodeJS es que permite escribir código tanto para el servidor como para el cliente usando el mismo lenguaje: JavaScript.

Siempre han existido dificultades en decidir si poner lógica del lado del cliente o del lado del servidor. Con NodeJS se puede tomar el JavaScript escrito para el cliente y fácilmente adaptarlo para el servidor y viceversa.

Un punto adicional es que los desarrolladores del cliente y del servidor manejan el mismo lenguaje. El poder escribir la aplicación con un solo lenguaje, ayuda en la reducción en los cambios de contexto que normalmente se realizan entre el cliente y el servidor, además permite que el código entre cliente y servidor pueda ser compartido, lo cual deriva en reutilización de código[**23**].

- Escalabilidad orientada a eventos: NodeJS aplica una lógica única para el manejo de peticiones web. En vez de tener múltiples hilos esperando para procesar las peticiones web, en NodeJS éstas son procesadas en el mismo hilo, usando un modelo basado en eventos. Esto permite a los servidores web escritos en NodeJS escalar en formas que los servidores tradicionales no pueden[**23**].
- Extensibilidad: NodeJS cuenta con una comunidad de desarrollo muy activa. Sus desarrolladores proveen nuevos módulos que extienden su funcionalidad todo el tiempo. Además, instalar e incluir nuevos módulos en NodeJS es una tarea sumamente sencilla y puede realizarse en poco tiempo[**23**].

En muy pocos años desde el momento de su creación en 2009, cerca de 650,000 módulos han sido publicados para npm, el manejador de paquetes de NodeJS. De acuerdo con Module Counts [**24**], un sitio web que monitorea el número de módulos en varios repositorios, el registro de npm crece a una tasa de 526 módulos nuevos por día. El manejador de paquetes más cercano en términos de crecimiento es Maven Central (Java) con 130 módulos al día [**21**]. Con esta cantidad de módulos disponibles, los desarrolladores pueden prácticamente encontrar al menos uno que resuelva casi cualquier problema que se les presente. (Obviamente, esos módulos se encuentran en varias etapas de desarrollo, y no todos ellos están listos para producción).

JavaScript es el lenguaje usado en muchas bases de datos NOSQL (como CouchDB y MongoDB), por lo que las interfaces entre NodeJS con dichos manejadores son naturales [**22**].

NPM

Fig. 2.13. Logotipo NPM.

NPM (Node Package Manager) está incluido como una característica recomendada en el instalador de Node.js. NPM consiste en un cliente de línea de comandos que interactúa con un registro

remoto, permitiendo a los usuarios consumir y distribuir los módulos de JavaScript que están disponibles en el registro. Los paquetes en el registro están en formato CommonJS e incluyen un archivo de metadatos en formato JSON.

NPM depende de los informes de los usuarios para eliminar paquetes si violan las políticas por ser de baja calidad, inseguros o maliciosos. npm expone estadísticas que incluyen la cantidad de descargas y la cantidad de paquetes dependientes para ayudar a los desarrolladores a juzgar la calidad de los paquetes.

NPM puede administrar paquetes que son dependencias locales de un proyecto en particular, así como también herramientas de JavaScript instaladas globalmente. Cuando se utiliza como administrador de dependencias para un proyecto local, npm puede instalar, en un comando, todas las dependencias de un proyecto a través del archivo package.json. En el archivo package.json, cada dependencia puede especificar un rango de versiones válidas utilizando el esquema de control de versiones semánticas, lo que permite a los desarrolladores actualizar automáticamente sus paquetes y, al mismo tiempo, evitar los cambios de interrupción no deseados. npm también proporciona herramientas de recuperación de versiones para que los desarrolladores etiqueten sus paquetes con una versión particular. npm también proporciona el archivo package-lock.json que tiene la entrada de la versión exacta utilizada por el proyecto después de evaluar las versiones semánticas en package.json.

A continuación vemos el fichero package.json que ha sido utilizado en este proyecto:

```
{
  "name": "medicalhistory",
  "version": "0.0.1",
  "dependencies": {
    "body-parser": "^1.18.2",
    "compression": "^1.7.1",
    "connect-flash": "^0.1.1",
    "ejs": "^2.5.7",
    "express": "^4.16.2",
    "express-session": "^1.15.6",
    "fs": "0.0.1-security",
    "method-override": "^2.3.10",
    "moment": "^2.22.2",
    "moment-timezone": "^0.5.17",
    "mongoose": "^5.0.2",
    "morgan": "^1.9.0",
    "npm": "^6.1.0",
    "passport": "^0.4.0",
    "passport-google-oauth": "^1.0.0",
    "passport-local": "^1.0.0"
  }
}
```

- Body-parser: es un módulo Node.js que proporciona middleware de conexión para analizar los cuerpos de solicitud de HTTP. Admite formatos JSON y urlencoded.
- Compression: módulo que facilita el manejo de las respuestas del backend.
- Connect-flash: módulo para intercambio de mensajes entre backend y frontend.
- Ejs: motor de plantillas ejs [**11**].
- Express-session: módulo que monitorizar el comportamiento de los usuarios. Este módulo funciona mediante el almcenamiento de una cookie que será identificada mediante un string secreto, es recomendable que en cada entorno (desarrollo o produción) este string sea diferente.
- Method-override: módulo para el soporte de verbos http.
- Moment: módulo para el manejo de datos tipo fechas, horas...
- Mongoose: módulo que enlaza node.js con la persistencia de datos en la BBDD de MongoDB [**12**].
- Morgan: módulo para el sistema de identificación (loggin).
- Passport: middleware de autentificacion de node.js para la identificacion de usuarios [**13**].
- Passport-google: estrategía de identificación con google.
- Passport-local: estrategía de identificación con local.

2.5.4. Bower

Fig. 2.14. Logotipo Bower.

Bower ofrece una solución para la administración de paquetes front-end, al tiempo que expone el modelo de dependencia del paquete a través de una API. No hay dependencias de todo el sistema,

no se comparten dependencias entre diferentes aplicaciones y el árbol de dependencias es plano [**14**].

Bower corre sobre Git, y es independiente del paquete. Un componente empaquetado puede estar compuesto por cualquier tipo de activo y usar cualquier tipo de transporte (por ejemplo, AMD, CommonJS, etc.).

A continuación vemos el fichero 'bower.json' que ha sido utilizado en este proyecto:

```
{
  "name": "medicalhistory",
  "version": "0.0.7",
  "dependencies": {
    "angular": "~1.2",
        "angular-route": "~1.2",
        "angular-resource": "~1.2"

  }
}
```

- Angular: módulo de AngularJS ya visto anteriormente.
- Angular-route: módulo que facilita el manejo de la navegación de una vista a la siguiente a medida que los usuarios realizan tareas de aplicación.
- Angular-resource: "fábrica"que crea un objeto de recursos que permite interactuar con fuentes de datos RESTful del lado del servidor. El objeto del recursos tiene métodos de acción que proporcionan comportamientos de alto nivel sin la necesidad de interactuar con el servicio http de bajo nivel.

2.5.5. Postman

Fig. 2.15. Logotipo Postman.

Postman es un compositor de peticiones HTTP (con versión online y local) que facilita la llamada a los servicios web, similar a Fiddler's Composer. También proporciona una alternativa para la documentación autogenerada de la API al soporte de Swagger de ServiceStack que hace que sea

más fácil llamar a los servicios existentes [**15**].

2.5.6. GIT y GitHub

Fig. 2.16. Logotipo Git.

GIT es un software de control de versiones pensando para la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos [**16**].

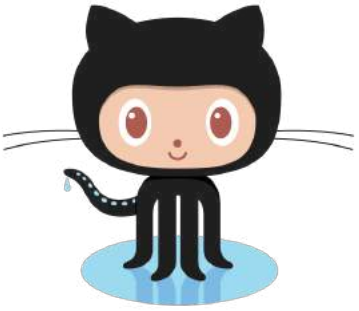

Fig. 2.17. Logotipo GitHub.

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de computadora. El código de los proyectos alojados en GitHub se almacena típicamente de forma pública, aunque utilizando una cuenta de pago permite hospedar repositorios privados.

2.5.7. Apache OpenOffice Calc

Fig. 2.18. Logotipo Apache OpenOffice Calc.

Apache OpenOffice Calc es en una hoja de cálculo libre y de código abierto compatible con Microsoft Excel. Forma parte de la suite ofimática Apache OpenOffice y como todos los componentes de la suite Apache OpenOffice, Calc puede usarse entre otroas plataformas con Mac OS X, Windows, GNU/Linux, FreeBSD y Solaris, estando disponible bajo licencia LGPL.

2.5.8. Texmaker (LaTeX)

Fig. 2.19. Logotipo LaTeX.

LaTeX (escrito LaTeX en texto plano) es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. Por sus características y posibilidades, es usado de forma especialmente intensa en la generación de artículos académicos, tesis y libros técnicos. LaTeX está formado por un gran conjunto de macros de TeX, con la intención de facilitar el uso de este lenguaje de composición tipográfica [**17**].

LaTeX presupone una filosofía de trabajo diferente a la de los procesadores de texto habituales (conocidos como WYSIWYG, es decir, «lo que ves es lo que obtienes») y se basa en instrucciones. El modo en que LaTeX interpreta la «forma» que debe tener el documento es mediante etiquetas.

Tradicionalmente, este aspecto se ha considerado una desventaja (probablemente la única), sin embargo, a diferencia de los procesadores de texto de tipo WYSIWYG, permite a quien escribe un documento centrarse exclusivamente en el contenido, sin tener que preocuparse de los detalles del formato. Además de sus capacidades gráficas para representar ecuaciones, fórmulas complicadas, notación científica e incluso musical, permite estructurar fácilmente el documento (con capítulos, secciones, notas, bibliografía, índices analíticos, etc.).

Con LaTeX, la elaboración del documento requiere normalmente de dos etapas:

1. Crear mediante cualquier editor de texto llano un archivo o fichero fuente (fichero con extensión .tex) que, con las órdenes y comandos adecuados, contenga el texto que queramos imprimir.

2. Procesar este archivo; el procesador de textos interpreta las órdenes escritas en él y compila el documento, dejándolo preparado para que pueda ser enviado a la salida correspondiente con independencia del dispositivo (impresora, pantalla, etc.) o sistema operativo (MS Windows, MacOS, Unix, distribuciones GNU/Linux, etc.); puede ser exportado a partir de una misma fuente a numerosos formatos tales como Postscript, PDF, SGML, HTML, RTF, etc.

Si se quiere añadir o cambiar algo en el documento, se deberán hacer los cambios en el archivo fuente y procesarlo de nuevo. Esta idea de compilación es análoga a las que se realiza con los lenguajes de programación de alto nivel (C, C++, etc.).

Existen varias herramientas o aplicaciones que ayudan a una persona a escribir estos documentos de una manera más visual (LyX, TeXmacs y otros). En el desarrollo de este documento ha sido utilizado TexMaker.

Una de las ventajas de LaTeX es que la salida que ofrece es siempre la misma, o el sistema operativo (MS Windows, MacOS, Unix, distribuciones GNU/Linux, etc.) y puede ser exportado a partir de una misma fuente a numerosos formatos tales como Postscript, PDF, SGML, HTML, RTF, etc.

3. ANÁLISIS DEL SISTEMA

En este apartado, la meta a conseguir es identificar los objetivos que han de ser alcanzados para el desarrollo del sistema final. Para lograrlo se tomará como referencia las historias de usuario que representan las necesidades que deben cubrir las funcionalidades de la aplicación, y de este modo satisfacer las exigencias del cliente.

Para la realización del proceso de extracción de información relativa a las necesidades a cubrir se llevará a cabo entre los miembros del equipo de desarrollo y el propio cliente. Este proceso, gracias a la metodología Scrum, no se limitará a la fase inicial del proyecto, como ocurriría generalmente en un desarrollo clásico, si no que se tratará de un proceso iterativo en constante evolución para poder amoldarse a los requerimientos del cliente de la forma más eficiente.

Las historias de usuario que se extraigan de las reuniones con el cliente, describirán una funcionalidad que debe incorporar un sistema de software y cuya implementación aporta valor al cliente.

La estructura de una historia de usuario está formada por:

- Nombre breve y descriptivo.
- Descripción de la funcionalidad en forma de diálogo o monólogo del usuario describiendo la funcionalidad que desea realizar.
- Criterio de validación y verificación que determinará para considerar terminado y aceptable por el cliente el desarrollo de la funcionalidad descrita.

Las historias de usuario se descomponen en tareas. Estas tareas son unidades de trabajo que tienen asignado un esfuerzo estimado y un estado. Por lo que es en las tareas en los que se basa la estimación de esfuerzos general del proyecto.

3.1. Definición

Para la definición del sistema, es decir, para la extracción de la información que ayude a modelar la aplicación según las necesidades del cliente, se realiza una reunión entre todos los miembros interesados en el proyecto. En dicha reunión el cliente expone su idea del sistema, que necesidades debe cubrir y que funciones quiere que le aporte. También los miembros del equipo técnico aportan su visión para enriquecer la definición final del sistema a desarrollar. Finalmente con toda la información extraída de la reunión el Product Owner, el actor que representa la voz del cliente, escribe las historias de usuario donde se representa formalmente la definición del sistema, además se extraen las tareas que componen las historias de usuario, se les asigna un coste (un esfuerzo

estimado) y las prioriza. Quedando formado el Product Backlog que será la base del proyecto a desarrollar.

Es importante aclarar que debido a la naturaleza ágil del proyecto, la definición inicial del sistema puede verse alterada en el transcurso del desarrollo del mismo. Ya que como más adelante se verá, el desarrollo se divide en diferentes etapas o Sprints, entre los cuales se repetirá la reunión entre los miembros del proyecto, pudiéndose incluir, modificar o eliminar historias de usuario y/o tareas. Por este motivo el análisis expuesto en este capítulo únicamente se trata del análisis inicial, y en próximos capítulos se describirán los análisis correspondientes a la reunión de cada Sprint.

3.2. Historias de usuario

Las historias de usuario se definirán utilizando el siguiente modelo:

Historia de Usuario	
ID	
Nombre	
Prioridad	
Riesgo	
Descripción	
Validación	

TABLA 3.1. MODELO DE HISTORIA DE USUARIO.

Donde cada campo tiene el siguiente significado:

- **ID:** Identificador único asignado a este elemento del proyecto, con formato *HUxx* (siendo *xx* numeración progresiva de 01 a 99).
- **Nombre:** Nombre corto utilizado para describir muy brevemente la historia de usuario.
- **Prioridad:** Preferencia de cara al desarrollo de la historia de usuario respecto a las demás. Toma valores: *Alta, media y baja*.
- **Riesgo:** Se trata de la importancia de la historia de usuario en relación al conjunto del proyecto. Cuantificando de este modo el daño provocado en caso de fallo.
  Toma valores: *Alto, medio y bajo*.
- **Descripción:** Breve explicación de las intenciones de la historia de usuario. Debe dejar clara la idea de la propia historia.
- **Validación:** Condiciones que deben cumplirse una vez la historia está completamente desarrollada para que se pueda dar por finalizada.

Historia de Usuario	
ID	HU01
Nombre	Registrar y administrar una cita.
Prioridad	Alta
Riesgo	Alto
Descripción	Como paciente quiero registrar citas nuevas en el sistema.
Validación	- Quiero poder introducir nuevas citas en el sistema, seleccionando al médico, Introducciendo una fecha y hora e indicando un motivo de la consulta. - Quiero que la operación quede registrada a nombre del paciente que realiza la cita, De la fecha en la que se realizará y del médico que la realizará. - Quiero que al crear la cita esta se añada a la agenda del médico que la atenderá. - Las citas pueden ser modificadas o canceladas posteriormente.

TABLA 3.2. HISTORIA DE USUARIO 01.

Historia de Usuario	
ID	HU02
Nombre	Consultar historial médico
Prioridad	Alta
Riesgo	Alto
Descripción	Como paciente quiero poder ver todo mi historial médico
Validación	- Quiero poder consultar todos los informes que han realizado los médicos que me han atendido.

TABLA 3.3. HISTORIA DE USUARIO 02.

Historia de usuario	
ID	HU03
Nombre	Consultar agenda
Prioridad	Alta
Riesgo	Alto
Descripción	Como médico quiero consultar las citas solicitadas
Validación	- Quiero poder consultar todas las citas que me han solicitado los pacientes.

TABLA 3.4. HISTORIA DE USUARIO 03.

Historia de usuario	
ID	HU04
Nombre	Realizar historia médica
Prioridad	Alta
Riesgo	Alto
Descripción	Como médico al atender una cita (realizar consulta) quiero crear una historia médica.
Validación	- Quiero que la historia médica quede registrada en el historial del paciente que realiza la cita. - Quiero que en la historia médica quede registrada la fecha de creación y el médico que lo realiza, además de los datos: - Antecedentes personales - Tratamiento actual - Antecedentes familiares - Anamnesis - Altura - Peso - Frecuencia respiratoria - Presión arterial - Frecuencia cardiaca - Exploración física - Diagnostico - Prescripción médica - Instruciones médicas

TABLA 3.5. HISTORIA DE USUARIO 04.

Historia de Usuario	
ID	HU05
Nombre	Consultar historial de paciente
Prioridad	Alta
Riesgo	Alto
Descripción	Como médico al atender una cita quiero poder consultar el historial del paciente
Validación	- Quiero poder consultar todos los informes que han realizado al paciente.

TABLA 3.6. HISTORIA DE USUARIO 05.

Historia de Usuario	
ID	HU06
Nombre	Establecer hora de consulta
Prioridad	Media
Riesgo	Medio
Descripción	Como médico quiero poder establecer mis horas de consulta
Validación	- Quiero que el médico pueda determinar las horas que esta disponible para pasar consulta.

TABLA 3.7. HISTORIA DE USUARIO 06.

Historia de Usuario	
ID	HU07
Nombre	Realizar artículos
Prioridad	Baja
Riesgo	Bajo
Descripción	Como médico quiero poder crear artículos
Validación	- Quiero que el médico pueda realizar artículos - Quiero que los médicos puedan consultar artículos del resto de médicos. - Quiero que el médico que realiza un articulo pueda editarlo o borrarlo.

TABLA 3.8. HISTORIA DE USUARIO 07.

Historia de Usuario	
ID	HU08
Nombre	Funcionamiento en Navegador
Prioridad	Media
Riesgo	Bajo
Descripción	Como usuario quiero que el sistema funcione en los navegadores más utilizados
Validación	- Quiero que la aplicación funcione en Chrome, Internet Explorer y Firefox.

TABLA 3.9. HISTORIA DE USUARIO 08.

Historia de Usuario	
ID	HU09
Nombre	Base de datos
Prioridad	Alta
Riesgo	Medio
Descripción	Como desarrollador quiero que los datos introducidos sean persistentes.
Validación	- Quiero el uso de una base de datos. - Quiero que la base de datos se la puedan introducir nuevos datos. - Quiero que los datos de la base de datos se puedan modificar. - Quiero que los datos de la base de datos se puedan eliminar.

TABLA 3.10. HISTORIA DE USUARIO 09.

Historia de Usuario	
ID	HU10
Nombre	Apariencia de la aplicación
Prioridad	Media
Riesgo	Bajo
Descripción	Como usuario quiero que la apliación web tenga un aspecto simple y sencillo de manejar.
Validación	- Quiero que no aparezca publicidad. - Quiero que las ventanas contengan solo la información imprescindible. - Quiero que cada tipo de usuario tenga su acceso y registro diferenciado.

TABLA 3.11. HISTORIA DE USUARIO 10.

Historia de Usuario	
ID	HU11
Nombre	Seguridad de la aplicación
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario quiero que la apliación web tenga una seguridad según el artículo 9 de la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter personal
Validación	- Quiero el uso de sesiones de usuario - Quiero la encriptación de información sensible

TABLA 3.12. HISTORIA DE USUARIO 11.

Historia de Usuario	
ID	HU12
Nombre	Tipo de aplicación web
Prioridad	Alta
Riesgo	Alto
Descripción	Como desarrollador quiero que la apliación web sea asíncrona y con páginas dinámicas
Validación	- Quiero un entorno de ejecución de E/S dirigida por eventos y por lo tanto asíncrona - Quiero páginas dinámicas

TABLA 3.13. HISTORIA DE USUARIO 12.

3.3. Tareas

Las tareas que componen las historia de usuario se definirán utilizando el siguiente modelo:

Tarea	
Hsitoria de Usuario	
Estado	
Descripción	

TABLA 3.14. MODELO DE TAREAS.

Donde cada campo tiene el siguiente significado:

- **Tarea:** Identificador único para este elemento del proyecto con formato *Txx*, (siendo *xx* numeración progresiva de 01 a 99).
- **Historia de Usuario:** Historia de usuario a la que pertence esta tarea.
- **Estado:** Fase en la que se encuentra el desarrollo de la tarea.
  Toma valores: *No iniciada, En progreso y Completada*.
- **Descripción:** Breve explicación de la finalidad de la tarea.

Tarea	T01
Hsitoria de Usuario	HU12
Estado	
Descripción	Crear y configurar aplicación básica express: • Definición de módulos en el archivo json. ◦ Módulo Express. • Implementación server.js.

TABLA 3.15. TAREA 01.

Tarea	T02
Hsitoria de Usuario	HU12
Estado	
Descripción	Configuración e implementación de estructura de la API Web (Backend-frontend).

TABLA 3.16. TAREA 02.

Tarea	T03
Hsitoria de Usuario	HU12
Estado	
Descripción	Implementación de la estructura del Backend: • App (MVC Backend) ◦ Modelos. ◦ Controladores. ◦ Rutas (enrutamiento de la aplicación). ◦ Vistas. • Config (archivos de configuración de la aplicación) • Public (MVC Frontend)

TABLA 3.17. TAREA 03.

Tarea	T04
Hsitoria de Usuario	HU12
Estado	
Descripción	Configuración del backend: • Implementación básica del controlador de index. • Implementación básica del enrutamiento de index. • Implementación básica de la vista index.

TABLA 3.18. TAREA 04.

Tarea	T05
Hsitoria de Usuario	HU12
Estado	
Descripción	Instalación y configuración de módulos en la aplicación express: • Configuración del módulo node-env. • Configuración del módulo morgan. • Configuración del módulo compression. • Configuración del módulo body-parser. • Configuración del módulo ethod-override.

TABLA 3.19. TAREA 05.

Tarea	T06
Hsitoria de Usuario	HU12
Estado	
Descripción	Configuración de renderización de las vistas: • Instalación y configuración del módulo ejs.

TABLA 3.20. TAREA 06.

Tarea	T07
Hsitoria de Usuario	HU12
Estado	
Descripción	Implementación y configuración para archivos estáticos.

TABLA 3.21. TAREA 07.

Tarea	T08
Hsitoria de Usuario	HU11
Estado	
Descripción	Configurar el uso de sesiones. • Instalación y configuración del módulo express-sessión.

TABLA 3.22. TAREA 08.

Tarea	T09
Hsitoria de Usuario	HU09
Estado	
Descripción	Configurar el uso de base de datos MongoDB. • Instalación y configuración del módulo mongoose.

TABLA 3.23. TAREA 09.

Tarea	T10
Hsitoria de Usuario	HU11
Estado	
Descripción	Implementación del módulo 'Usuarios' backend: • Implementación del modelo con Schema 'Users'. • Cargar el modelo 'Users' en el middleware mongoose. • Implementación del controlador 'Users'. • Implementación de las rutas de 'Users'. • Cargar las rutas de 'Users' en la aplicación express.

TABLA 3.24. TAREA 10.

Tarea	T11
Hsitoria de Usuario	HU11
Estado	
Descripción	Configuración e implementación de registro e identificación: • Estrategía local. • Estrategía google.

TABLA 3.25. TAREA 11.

Tarea	T12
Hsitoria de Usuario	HU12
Estado	
Descripción	Configuración de la estructura del Frontend

TABLA 3.26. TAREA 12.

Tarea	T13
Hsitoria de Usuario	HU11
Estado	
Descripción	Implementación del módulo 'Principal' frontend: • Crear el módulo 'Main'. • Crear el controlador 'Main'. • Crear las rutas de 'Main'. • Crear las vistas de 'Main'. ◦ Usuarios no autenticados. ◦ Usuarios autenticados. ► Medicos. ► Pacientes.

TABLA 3.27. TAREA 13.

Tarea	T14
Hsitoria de Usuario	HU07
Estado	
Descripción	Implementación del módulo 'Artículos' backend: • Implementación del modelo con Schema 'Artículos'. • Cargar el modelo 'Artículos' en el middleware mongoose. • Implementación del controlador 'Artículos'. • Implementación de las rutas de 'Artículos'. • Cargar las rutas de 'Artículo' en la aplicación express.

TABLA 3.28. TAREA 14.

Tarea	T15
Hsitoria de Usuario	HU07
Estado	
Descripción	Implementación del módulo 'Artículos' frontend: • Crear el módulo 'Articules'. • Crear el servicio 'Articules'. • Crear el controlador 'Articules'. ◦ Crear artículo. ◦ Buscar todos los artículos. ◦ Buscar un artículo. ◦ Actualizar un artículo. ◦ Borrar un artículo. • Crear las rutas de 'Artículo'. • Crear las vistas de 'Artículo'. ◦ Crear artículo. ◦ Editar artículo. ◦ Listar artículos. ◦ Visualizar un artículo.

TABLA 3.29. TAREA 15.

Tarea	T16
Hsitoria de Usuario	HU01
Estado	
Descripción	Implementación del módulo 'Citas' backend: • Implementación del modelo con Schema 'Citas'. • Cargar el modelo 'Citas' en el middleware mongoose. • Implementación del controlador 'Citas'. • Implementación de las rutas de 'Citas'. • Cargar las rutas de 'Citas' en la aplicación express.

TABLA 3.30. TAREA 16.

Tarea	T17
Hsitoria de Usuario	HU01
Estado	
Descripción	Implementación del módulo 'Citas' frontend: • Crear el módulo 'Citas'. • Crear el servicio 'Citas'. • Crear el controlador 'Citas'. ◦ Crear Citas. ◦ Buscar todos los citas. ◦ Buscar una citas. ◦ Modificar cita. ◦ Eliminar cita. • Crear las rutas de 'Cita'. • Crear las vistas de 'Cita'. ◦ Crear cita. ◦ Editar cita. ◦ Listar citas. ◦ Visualizar cita.

TABLA 3.31. TAREA 17.

Tarea	T18
Hsitoria de Usuario	HU09
Estado	
Descripción	Implementación del módulo 'Doctores' backend: • Implementación del modelo con Schema 'Doctores'. • Cargar el modelo 'Doctores' en el middleware mongoose. • Implementación del controlador 'Doctores'. • Implementación de las rutas de 'Doctores'. • Cargar las rutas de 'Doctores' en la aplicación express.

TABLA 3.32. TAREA 18.

Tarea	T19
Hsitoria de Usuario	HU09
Estado	
Descripción	Implementación del módulo 'Doctores' frontend: • Crear el módulo 'Doctores'. • Crear el servicio 'Doctores'. • Crear el controlador 'Doctores'. ◦ Crear Doctor. ◦ Buscar todos los Doctores. ◦ Buscar un doctor. ◦ Modificar doctor. ◦ Eliminar doctor. • Crear las rutas de 'Doctores'. • Crear las vistas de 'Doctor'. ◦ Crear perfil. ◦ Editar perfil. ◦ Listar doctores. ◦ Visualizar doctor.

TABLA 3.33. TAREA 19.

Tarea	T20
Hsitoria de Usuario	HU09
Estado	
Descripción	Implementación del módulo 'Pacientes' backend: • Implementación del modelo con Schema 'Pacientes'. • Cargar el modelo 'Pacientes' en el middleware mongoose. • Implementación del controlador 'Pacientes'. • Implementación de las rutas de 'Pacientes'. • Cargar las rutas de 'Pacientes' en la aplicación express.

TABLA 3.34. TAREA 20.

Tarea	T21
Hsitoria de Usuario	HU09
Estado	
Descripción	Implementación del módulo 'Pacientes' frontend: • Crear el módulo 'Pacientes'. • Crear el servicio 'Pacientes'. • Crear el controlador 'Pacientes'. ◦ Crear Paciente. ◦ Buscar todos los Pacientes. ◦ Buscar un paciente. ◦ Modificar paciente. ◦ Eliminar paciente. • Crear las rutas de 'Pacientes'. • Crear las vistas de 'Pacientes'. ◦ Crear perfil. ◦ Editar perfil. ◦ Listar pacientes. ◦ Visualizar paciente.

TABLA 3.35. TAREA 21.

Tarea	T22
Hsitoria de Usuario	HU04
Estado	
Descripción	Implementación del módulo 'Consulta' backend: • Implementación del modelo con Schema 'Consulta'. • Cargar el modelo 'Consulta' en el middleware mongoose. • Implementación del controlador 'Consulta'. • Implementación de las rutas de 'Consulta'. • Cargar las rutas de 'Consulta' en la aplicación express.

TABLA 3.36. TAREA 22.

Tarea	T23
Hsitoria de Usuario	HU04
Estado	
Descripción	Implementación del módulo 'Consulta' frontend: • Crear el módulo 'Consultas'. • Crear el servicio 'Consultas'. • Crear el controlador 'Consultas'. ◦ Crear consulta. ◦ Buscar todos las consultas. ◦ Buscar una consulta. ◦ Actualizar una consulta. ◦ Borrar una consulta. • Crear las rutas de 'Consulta'. • Crear las vistas de 'Consulta'. ◦ Crear consulta. ◦ Editar consulta. ◦ Listar consultas. ◦ Visualizar un consulta.

TABLA 3.37. TAREA 23.

Tarea	T24
Hsitoria de Usuario	HU07
Estado	
Descripción	Implementación de asignación de 'Articulos' → 'Doctores'.

TABLA 3.38. TAREA 24.

Tarea	T25
Hsitoria de Usuario	HU03
Estado	
Descripción	Implementación de asignación de 'Citas' → 'Doctores'.

TABLA 3.39. TAREA 25.

Tarea	T26
Hsitoria de Usuario	HU01
Estado	
Descripción	Implementación de asignación de 'Citas' → 'Pacientes'.

TABLA 3.40. TAREA 26.

Tarea	T27
Hsitoria de Usuario	HU04
Estado	
Descripción	Implementación de asignación de 'Consulta' → 'Citas'.

TABLA 3.41. TAREA 27.

Tarea	T28
Hsitoria de Usuario	HU10
Estado	
Descripción	Enriquecer la vista de 'Principal'.

TABLA 3.42. TAREA 28.

Tarea	T29
Hsitoria de Usuario	HU10
Estado	
Descripción	Enriquecer las vista de 'Artículos'.

TABLA 3.43. TAREA 29.

Tarea	T30
Hsitoria de Usuario	HU10
Estado	
Descripción	Enriquecer las vista de 'Citas'.

TABLA 3.44. TAREA 30.

Tarea	T31
Hsitoria de Usuario	HU10
Estado	
Descripción	Enriquecer las vista de 'Doctores'.

TABLA 3.45. TAREA 31.

Tarea	T32
Hsitoria de Usuario	HU10
Estado	
Descripción	Enriquecer las vistas de 'Pacientes'.

TABLA 3.46. TAREA 32.

Tarea	T33
Hsitoria de Usuario	HU10
Estado	
Descripción	Enriquecer las vistas de 'Consultas'.

TABLA 3.47. TAREA 33.

3.4. Estimación

La estimación es el proceso por el que se obtiene una previsión del esfuerzo necesario para desarrollar las diferentes partes del proyecto. Esta información nos orientará para organizar el desarrollo del producto.

La estimación de proyectos software es de vital importancia en toda la etapa de desarrollo del software. Cuanto más precisas sean las estimaciones mejor se podrá planificar logrando así cumplir los objetivos y mejorar la rentabilidad del proyecto. Para las metodologías ágiles existe una técnica bastante extendida para la estimación: el método **Planning Poker**. Esta técnica se puede utilizar conjuntamente con otros métodos como el juicio de un experto y las estadísticas de rendimiento del equipo en proyectos similares.

En caso de utilizar la combinación de varias técnicas habría que asignar un peso a cada una de ellas y utilizar una ponderación dando pesos entre los diferentes métodos utilizados para la estimación.

En este proyecto se utilizará únicamente Planning Poker debido a la falta de un experto que pueda aportar su opinión de la falta de experiencia del equipo en proyectos similares, por lo que no se tiene nada en lo que basarse.

3.4.1. Planning Poker

Este proceso tiene como objetivo el poder realizar un cálculo del esfuerzo necesario para llevar a cabo las distintas historias de usuario mediante un consenso entre los miembros encargados de realizar las distintas tareas que componen cada fase.

La técnica consiste en:

1. Cada miembro del equipo elige una carta de la baraja que representa el valor estimado del esfuerzo que él considera para la tarea sin mostrársela al resto de los miembros del equipo para no influenciar su decisión.

2. Una vez todos los miembros del equipo han elegido su carta, la muestran al resto para así elegir la estimación final según la mayoría. * En caso de discrepancias insalvables se puede realizar un pequeño debate para ver los puntos de discordia y llegar a un acuerdo.

3. Se suelen dar a los valores de la baraja la secuencia de Fibonacci, con alguna pequeña variación, por lo que no es una sucesión lineal, de esta forma se refleja la incertidumbre a la hora de realizar las estimaciones, a mayor grado de complejidad, mayor es esta incertidumbre.

Esta técnica sirve como se ha mencionado anteriormente para la estimación de historias de usuario, es decir el conjunto de todas las tareas de que la componen, en lugar de estimar cada tarea de forma individualizada.

En el caso de este proyecto se ha optado por la estimación de tareas, puesto que se ha considerado esta como la forma más óptima de realizar el proceso de estimación.

3.4.2. Estimación en el proyecto

En este apartado se explicará la forma que se estimará este proyecto después del análisis para la extracción de las historias de usuario y sus tareas. El método que se va a utilizar estará basado en el Planning Poker anteriormente explicado, pero adaptado a las condiciones especiales de este proyecto. Esta adaptación será lo que se detalle en este apartdado.

Para realizar la estimación de las historias de usuario se utilizará la técnica de Planning Poker al inicio de cada iteración (sprint), donde se aplicará a todas las historias de usuario restantes. De este modo se ordenarán por prioridad y se obtendrá como resultado la cantidad de trabajo que se llevará a cabo durante ese sprint.

Hay que tener en cuenta que el prototipo resultante de una iteración debe ser funcional en medida de sus posibilidades, por lo que a la hora de escoger las historias de usuario a realizar en el sprint siempre debe ser inferior la suma de tiempos estimados que el tiempo total de la iteración. Así se evitaría el quedase una historia a medio hacer afectando al funcionamiento del prototipo.

Para priorizar las historias de usuario, al principio del proyecto, y a intervalos regulares durante el proyecto, se va estimando el esfuerzo de cada una en puntos de usuario. Normalmente se utilizan los números 0, 1, 3, 5, 8, 13, 20, 40 y 100 como puntos de usuario, siendo en orden creciente el esfuerzo de las historias, pudiendo incluso no ser asumible en ese momoetno (100 puntos). El Product Owner expone cada una de las historias y el equipo intenta llegar a un consenso con el esfuerzo asignado. Se suelen utilizar otras historias para comparar. Un ejemplo de funcionamiento y de formato de las tablas de estimación utilizando Planning Poker es la siguiente:

Historia de Usuario	Miembro			Estimación Media	Prioridad
	A	B	C		
HU01	13	13	20	15,3333333	Alta
HU02	13	13	20	15,3333333	Media
HU03	5	8	8	7	Baja

TABLA 3.48. EJEMPLO PLANINNG POKER.

En la tabla del Planning Poker podemos observar las valoraciones que cada miembro del equipo ha supuesto para el desarrollo del proyecto. Se observa como han utilizado valores para las horas de la serie 0, 1, 3, 5, 8, 13, 20, 40 y 100. De este modo lo primero sería establecer la capacidad de trabajo del equipo durante una iteración.

Para este ejemplo se va a suponer una capacidad de trabajo de 20 horas. La solución sería "rellenar" ese espacio disponible de 20 horas con las historias de usuario por completar, teniendo en cuenta lo prioritario de cada una de estas, por lo que se deben coger primero las de prioridad alta, después prioridad media y por último prioridad baja.

Según el ejemplo anterior en este primer sprint se llevaría a cabo la historia de usuario 1, que tiene una estimación de 15,3 horas y prioridad Alta.

El tiempo que existe de diferencia entre el estimado y el total de la iteración puede utilizarse para solventar los normales errores de estimación, permitiendo recuperar posibles retrasos en el desarrollo debidos a cualquier causa. Lo importante es que al finalizar el sprint el resultado sea un prototipo funcional que pueda ser presentado ante el cliente.

En este proyecto se realizará el Planning Poker con tareas en vez de con historia de usuario, ya que se realizará una estimación más precisa que con las historias de usuario. Posteriormente en el inicio del primer sprint veremos el Product Backlog logrado trás el planning poker.

4. DESARROLLO DEL SISTEMA

En este capítulo se realizará el seguimiento del desarrollo del proyecto, desde el fin del análisis hasta completar todas las tareas de las historias de usuario detalladas en el análisis del sistema.

Primeramente se explicará una visión global del desarrollo, donde se explicarán las tecnologías utilizadas y el motivo de su elección. Luego se detallará cada fase del desarrollo, llamada Sprint. En total se contará con siete Sprints de dos semanas cada uno que posteriormente se describirán y explicarán con más detalle el funcionamiento y composición de cada Sprint.

4.1. Visión general

Para el desarrollo del sistema se ha seleccionado una combinación de tecnologías y metodologías que incluyen las explicadas en el capítulo de "Estado del arte".

En este capítulo se exponen las herramientas para el desarrollo del proyecto. La metodología a utilizar está ya predeterminada por la naturaleza del proyecto, siendo imprescindible el empleo de Scrum por este motivo. La combinación elegida para el desarrollo es:

- Plataforma Sistema operativo: Manjaron Linux.
- Plataforma servidor y Base de datos: MEAN.
- Navegador: Mozilla Firefox.
- Aplicación de desarrollo: Visual Studio.
- Control de versiones: GitHub.
- Pruebas CRUD sobre URL: Postman.
- Seguimiento SCRUM: OpenOffice Calc.

4.2. Arquitectura

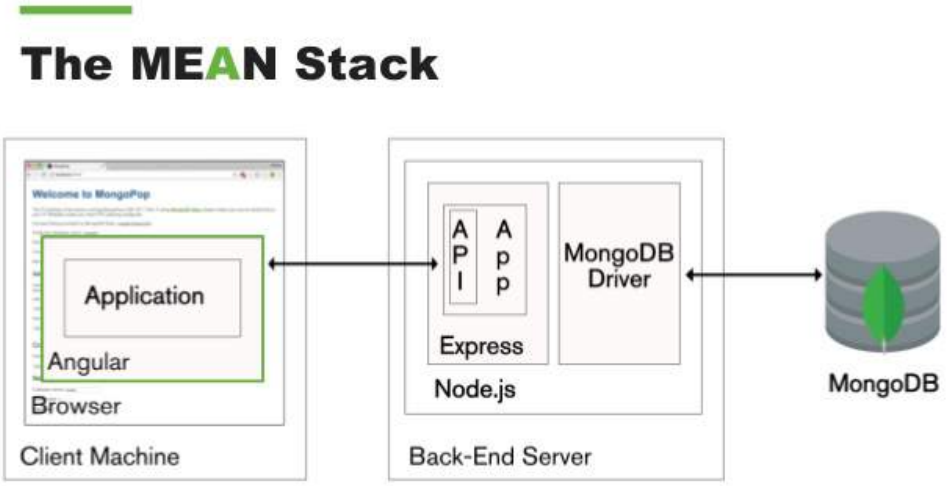

Fig. 4.1. Arquitectura MEAN Stack.

4.2.1. MVC Backend-frontend)

El patrón arquitectónico MVC es implementado usualmente con un software framework que define una familia de aplicaciones relacionadas, y contiene elementos que son comunes a esas aplicaciones.

Los frameworks proveen estructura a través del uso de ciertas convenciones de nombrado (carpetas, archivos, etc) y reglas para construir un sistema.

También proveen componentes que soportan la construcción de un sistema. Los desarrolladores de aplicaciones, extienden el framework para construir sus propias aplicaciones. Por lo tanto, un framework es básicamente un ambiente de desarrollo en el cual los programadores pueden desarrollar aplicaciones de todo tipo mucho más fácil y rápido, incluyendo aplicaciones web. Un framework puede consistir de librerías de código fuente, utilerías, plug ins, modelos de desarrollo, y una gran cantidad de herramientas, cuyo propósito es acelerar el desarrollo de aplicaciones.

Existe un conjunto de características recurrentes en la mayoría de los frameworks:

- Proveen funcionalidad dentro de un dominio bien definido. Por ejemplo videojuegos, telecomunicaciones, interfaces gráficas de usuario, etc.
- Definen los patrones de interacción entre un conjunto de componentes u objetos bien definidos.

  Para usar el framework, los desarrolladores requieren entender esos patrones de interacción y deben programar de acuerdo a ellos.
- Fomentan reuso de código y de diseño.

- Proveen un ambiente completo para el desarrollo de sitios web, interoperabilidad, seguridad y mantenimiento de tal forma que los desarrolladores no tienen que construir sistemas desde cero cada vez que lanzan un nuevo sitio [**19**].
- Ofrecen numerosas ventajas técnicas y organizacionales sobre los métodos de desarrollo clásico, tales como un desarrollo más rápido y una estructura de aplicación más limpia.
- Su uso reduce el tiempo invertido en mantenimiento de código y desarrollo futuro debido a que provee numerosas librerías, haciendo posible que los desarrolladores logren las funcionalidades deseadas con menos código, de tal manera que el software es más fácilmente mantenible.
- Brindan oportunidad de mejoras futuras tales como medidas de protección contra técnicas de hackeo y vulnerabilidades aún no determinadas.
- Proveen un modelo o arquitectura estándar que permite fácilmente, la visualización de cómo trabaja el sistema entero[**19**].
- Permiten la obtención de un código bien estructurado a través del uso de patrones arquitectónicos.

Existen muchos frameworks de MVC para diversos lenguajes y sistemas incluyendo Java, Python, PHP, Microsoft ASP y más [**20**]. Ahora bien, es importante señalar que también existen desventajas en el uso de frameworks tal como lo establecen Sanovic y Petrijevcanin en [**19**]: Entre las principales desventajas al usar frameworks se encuentran:

- Complejidad y sobrecarga en el código, lo que en algunas ocasiones visiblemente reduce el desempeño de la aplicación y crea una sobre carga de hardware.
- Vulnerabilidades de seguridad en el código de los frameworks afectan a las aplicaciones que los utilizan.
- En algunas ocasiones la curva de aprendizaje es alta.
- Las convenciones estrictas se contraponen a la creatividad de los desarrolladores.

Sin embargo, Sanovic y Petrijevcanin en [**19**] también establecen que las ventajas superan por mucho a las desventajas no sólo en aplicaciones grandes y complejas, también en sitios pequeños:

Las ventajas claramente pesan más que las desventajas cuando la aplicación que está siendo desarrollada es grande y compleja y desarrollado por un equipo.

Por otro lado, si la aplicación es pequeña y simple y desarrollada por un solo desarrollador o un equipo pequeño, la percepción común es que los beneficios que el framework provee no son suficientes para justificar su uso.

Finalmente los mencionados autores después de un análisis completo, concluyen favoreciendo su uso:

- Los frameworks para aplicaciones web ofrecen numerosas ventajas técnicas y organizacionales.
- También la programación es más cómoda para desarrolladores, dado que no tienen que lidiar con muchas tareas comunes de programación.
- A pesar de éstas ventajas, el desarrollo de aplicaciones pequeñas es usualmente hecho sin frameworks, debido a sus desventajas.
- Sin embargo, creemos que las ventajas mencionadas son igualmente aplicables en el desarrollo de aplicaciones pequeñas, especialmente por el hecho de que el desarrollo futuro de la aplicación es difícil de predecir.

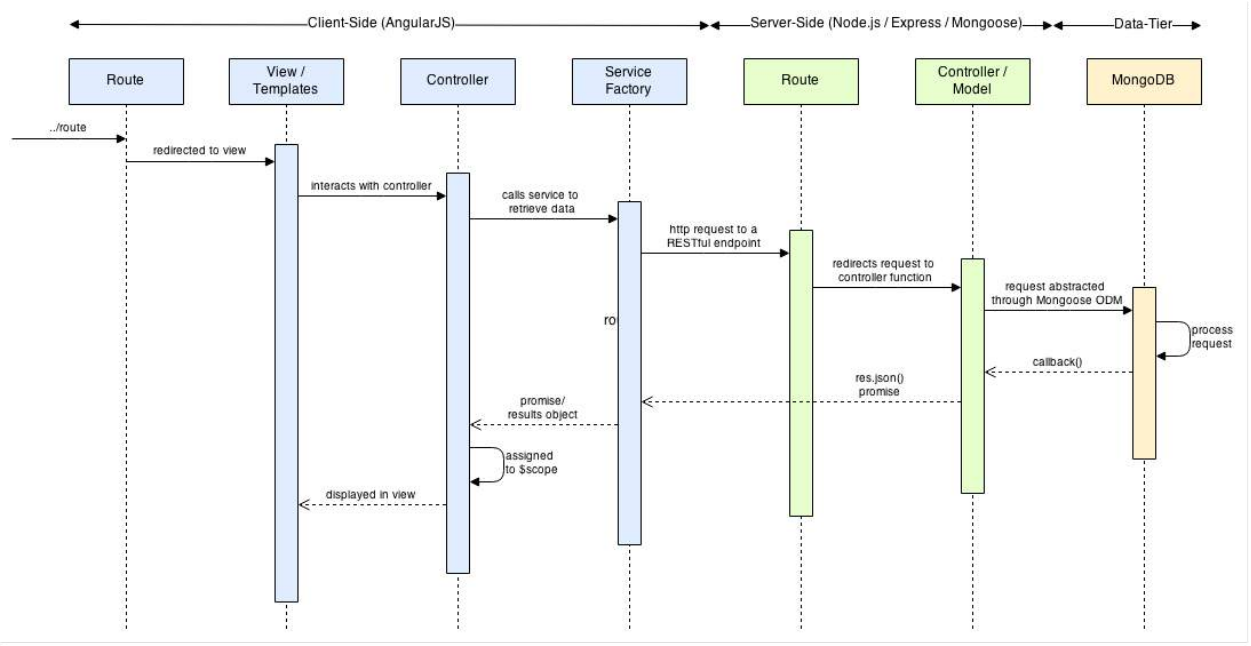

Fig. 4.2. Peticiones y respuestas en arquitectura MEAN Stack.

4.3. Sprints

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto que sea potencialmente entregable, de manera que cuando el cliente (Product Owner) lo solicite sólo sea necesario un esfuerzo mínimo para que el producto esté disponible para ser utilizado. Para ello, durante la iteración los integrantes del equipo colaboran estrechamente y se llevan a cabo las siguientes dinámicas:

- Cada día el equipo realiza una reunión de sincronización, donde cada miembro inspecciona el trabajo de los otros para poder hacer las adaptaciones necesarias, comunica cuales son los

impedimentos con que se encuentra, actualiza el estado de la lista de tareas de la iteración (Sprint Backlog) y los gráficos de trabajo pendiente (Burndown charts).

- El Facilitador (Scrum Master) se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad.

  - Elimina los obstáculos que el equipo no puede resolver por sí mismo.
  - Protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

También existen una serie de restricciones que deben ser tenidas en consideración:

- No se pueden cambiar los objetivos/requisitos de la iteración en curso:

  - En la reunión de planificación de la iteración el equipo adquirió el compromiso de completar en la iteración unos requisitos concretos, basó su plan y organización en ellos. Cambiar los objetivos/requisitos de la iteración dificulta la concentración del equipo, baja su moral y su compromiso.
  - El hecho de no poder cambiar los objetivos/requisitos de la iteración una vez iniciada facilita que el cliente cumpla con su responsabilidad de conocer qué es lo más prioritario a desarrollar, antes de iniciar la iteración.
  - Scrum minimiza esta necesidad ya que, por un lado, los objetivos/requisitos que se están desarrollando eran los más prioritarios justo antes de iniciar la iteración y, por otro lado, las iteraciones en Scrum son suficientemente cortas (2 o 4 semanas) como para que la probabilidad de cambios una vez iniciada la iteración sea mínima.

- Terminación anormal de la iteración:

  - Sólo en situaciones muy excepcionales el cliente o el equipo pueden solicitar una terminación anormal de la iteración. Esto puede suceder si, por ejemplo, el contexto del proyecto ha cambiado enormemente y no es posible esperar al final de la iteración para aplicar cambios, o si el equipo encuentra que es imposible cumplir con el compromiso adquirido. En ese caso, se dará por finalizada la iteración y se dará inicio a otra mediante una reunión de planificación de la iteración.

4.3.1. Sprint Planning

La planificación de las tareas a realizar en la iteración se divide en dos partes:

- Primera parte de la reunión. Se realiza en un timebox de cómo máximo 4 horas (aproximadamente):

- El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto, pone nombre a la meta de la iteración (de manera que ayude a tomar decisiones durante su ejecución) y propone los requisitos más prioritarios a desarrollar en ella.
- El equipo examina la lista, pregunta al cliente las dudas que le surgen, añade más condiciones de satisfacción y selecciona los objetivos/requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.

- Segunda parte de la reunión. Se realiza en un timebox de cómo máximo 4 horas. El equipo planifica la iteración, elabora la táctica que le permitirá conseguir el mejor resultado posible con el mínimo esfuerzo. Esta actividad la realiza el equipo dado que ha adquirido un compromiso. El equipo es responsable de organizar su trabajo y el que mejor conoce cómo realizarlo.

  - Define las tareas necesarias para poder completar cada objetivo/requisito, creando la lista de tareas de la iteración (Sprint Backlog) basándose en la definición de completado.
  - Realiza una estimación conjunta del esfuerzo necesario para realizar cada tarea.
  - Cada miembro del equipo se autoasigna a las tareas que puede realizar.

4.3.2. Sprint 0

Se conoce como Sprint 0 a la fase inicial del proyecto donde se dedica aproximadamente una semana, independientemente del formato, duración, etc, del resto de sprints. En este Sprint es donde se va a preparar el equipo tanto tecnológicamente como metodológicamente para que el desarrollo del proyecto tenga un buen comienzo, en especial en casos en los que se desconoce la metodología o no se tiene mucha práctica con ella.

El objetivo del Sprint 0 es preparar el conjunto del proyecto desde una perspectiva:

- Tecnológica.
- Metodológica.
- Organizativa para que el desarrollo del proyecto tenga un buen comienzo y mejor finalización.

Todo ello consiste básicamente en:

- **Definir con el cliente.** En primer lugar "the product owner", es decir, yo mismo en este caso me estoy ocupando de definir con el cliente las características y funcionalidades del proyecto con el mayor número de detalles posibles. Con toda esa información construyo un documento en forma de historias de usuarios (vistas anteriormente).

- **Construir el Product Backlog.** Las historias de usuario presentan unidades que pueden presentarse a los clientes como elementos acabados. Con ellas construimos el "Product Backlog".

- **Reuniones de equipo.** Se realiza una reunión con el equipo donde se presentan las historias de usuario y donde el equipo deberá: identificar lagunas de información, dificultades técnicas, dependencias entre historias, proponer creación/división de historias de usuario...

Los motivos para hacerlo de este modo, son que nos aporta las siguientes ventajas, siendo éstas de gran ayuda en el desarrollo del proyecto, mejorando la productividad del equipo.

- Obtenemos una definición contrastada con el cliente en forma de historias de usuarios.

- El equipo participa en la preparación del desarrollo identificando necesidades, dificultades, ventajas...

- La preparación de cada sprint será más fácil de realizar porque el equipo conoce con detalle el proyecto.

El Sprint 0 de este proyecto lo conforman la fase de análisis descrita en capítulos anteriores de este documento y la formación de las diferentes tecnologías que eran novedosas para mí, por lo que ha durado 6 semanas en vez de 2 como típicamente duraría.

Además de esto en el sprint se define la capacidad de trabajo del equipo por cada iteración. Para ello se han de definir el tamaño de la iteración y el rendimiento del equipo:

Tamaño del Sprint	2 semanas (10 días laborables)
Trabajo por día	4 horas
Horas por Sprint	40 horas

TABLA 4.1. CAPACIDAD DE TRABAJO.

En este caso la cantidad de horas es relativamente baja para una iteración puesto que, aunque se simule la existencia de otros miembros en el equipo para estimaciones, reuniones, etc., debido a que trabajo del proyecto es realizado realmente por una única persona. De esta forma para todo lo relacionado con el desarrollo en sí de la aplicación siempre se tendrá en consideración esta limitación.

4.3.3. Primer Sprint

Lo primero que se hace en la iteración es poner sobre la mesa el Product Backlog conseguido y utilizarlo para estimar. El siguiente es una versión resumida de las tareas sacadas de las historias de usuario presentadas en el capítulo de análisis.

Tareas	Historia de Usuario	Miembro			Estimación Media	Prioridad
		A	B	C		
T01	HU12	1	3	3	2,33333333	Alta
T02	HU12	1	1	1	1	Alta
T03	HU12	1	1	1	1	Alta
T04	HU12	3	3	5	3,66666667	Alta
T05	HU12	1	1	1	1	Alta
T06	HU12	3	3	5	3,66666667	Alta
T07	HU12	1	1	3	1,66666667	Alta
T08	HU11	1	1	3	1,66666667	Alta
T09	HU09	1	3	3	2,33333333	Alta
T10	HU11	8	8	13	9,66666667	Alta
T11	HU11	5	5	8	6	Alta
T12	HU12	1	1	1	1	Alta
T13	HU11	5	8	13	8,66666667	Alta
T14	HU07	8	8	13	9,66666667	Alta
T15	HU07	8	13	20	13,6666667	Baja
T16	HU01	5	8	13	8,66666667	Alta
T17	HU01	5	8	13	8,66666667	Alta
T18	HU09	8	8	13	9,66666667	Alta
T19	HU09	8	13	13	11,3333333	Alta
T20	HU09	8	8	13	9,66666667	Alta
T21	HU09	8	8	13	9,66666667	Alta
T22	HU04	8	8	13	9,66666667	Alta
T23	HU04	8	8	13	9,66666667	Alta
T24	HU07	5	8	8	7	Alta
T25	HU03	8	13	13	11,3333333	Alta
T26	HU01	8	13	13	11,3333333	Alta
T27	HU04	8	13	20	13,6666667	Alta
T28	HU10	8	13	8	9,66666667	Alta
T29	HU10	5	8	8	7	Alta
T30	HU10	8	13	13	11,3333333	Alta
T31	HU10	8	13	13	11,3333333	Alta
T32	HU10	8	13	13	11,3333333	Media
T33	HU10	8	13	13	11,3333333	Media

TABLA 4.2. PRODUCT BACKLOG.

Como se puede ver en la tabla todas la tareas escogidas tienen prioridad alta, por lo que al presentarse esta situación queda a juicio del equipo seleccionar las de más prioridad a la hora de realizaralas.

Horas máximas	40
Horas utilizadas	35
Horas restantes	5
Tareas	**Horas estimadas**
T01	2,3333333333333
T02	1
T03	1
T04	3,6666666666667
T05	1
T06	3,6666666666667
T07	1,6666666666667
T08	1,6666666666667
T09	2,3333333333333
T10	9,6666666666667
T11	6
T12	1

TABLA 4.3. TAREAS ESCOGIDAS PRIMER SPRINT.

Vamos a realizar los objetivos de la tarea 1.

- Crear y configurar aplicación básica express:

  Lo primero que vamos a hacer será instalar NodeJS y NPM mediante pacman en nuestro sistema y el módulo express. El comando utilizad es:

  pacman -S nodejs npm

  + Definición de módulos en el archivo json.

    Creamos el fichero package.json dentro del directorio que contendrá la aplicación donde vamos a definir los módulos, el nombre y la versión de la aplicación.

    - Módulo Express.
      Añadimos como dependencia 'express' en el fichero package.json y utilizamos npm para su instalación.

Fig. 4.3. Contenido de package.json con express.

+ Implementación server.js.

  Creamos el fichero server.js dentro del directorio que contendrá la aplicación donde creamos una estancia de express y la referenciamos a la aplicación, establecemos el puerto de escucha del servidor y creamos un mensaje que aparecerá en la terminal para ver que se inicia correctamente la aplicación.

```
// Autorización para el uso de express
var express = require('./config/express');

// Instancia del módulo express
var app = express();

// Puerto de escucha para el servidor
app.listen(3000);

// Nos devuelve el objeto 'app'
module.exports = app;

// Mensaje para aparecer en consola del correcto arranque del servidor
console.log('Servidor ejecutandose en http://localhost:3000/');
```

Fig. 4.4. Contenido de server.js.

Vamos a realizar los objetivos de la tarea 2.

- Configuración e implementación de estructura de la API Web (Backend-frontend).

  Me decido por una estructura horizontal, basada en la division de carpetas y ficheros por su papel funcional en lugar de por la característica que implementan, lo que significa que todos los ficheros de la aplicación están dentro de una carpeta principal que contiene una estructura de carpetas MVC. Esto también significa que hay una sola carpeta controllers que contiene todos los controladores de la aplicación, una sola carpeta modelos que contiene todos los modelos de la aplicación, etc.

Vamos a realizar los objetivos de la tarea 3.

- Implementación de la estructura del Backend:

  Creamos el siguiente árbol de directorios dentro de nuestro directorio de aplicación.

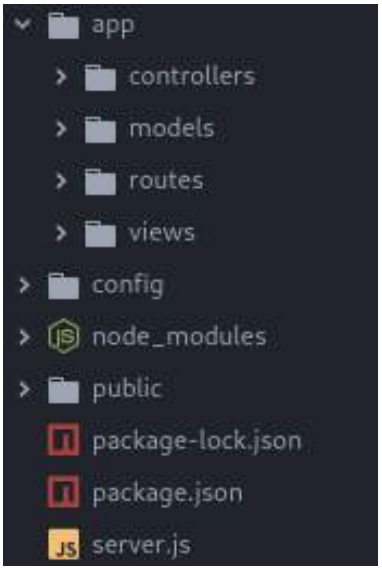

Fig. 4.5. Estructura de directorios Backend.

+ App (MVC Backend).
  - Modelos.
    Contendrá los Schemas de nuestra aplicación.
  - Controladores.
    Contendrá las operaciones CRUD sobre la base de datos y algunas funciones más dependiendo de los objetos que manejemos.
  - Rutas (enrutamiento de la aplicación).
    Contendrá las rutas de la apliación y los verbos HTTP que dispondrá cada una.
  - Vistas.
    Contendrá la página única de la aplicación donde se cargarán las diferentes vistas del frontend.
+ Config

  Mantendremos los archivos de configuración de todo el entorno de la aplicación.
+ Public (MVC Frontend)

  Contendrá los ficheros frontend de la aplicación.

Vamos a realizar los objetivos de la tarea 4.

- Configuración del backend:
  + Implementación básica del controlador de index.

    Dentro de la carpeta de controladores crearemos uno para el index de nuestra aplicación, donde responderemos con un mensaje para mostrar y ver que funciona correctamente.

```
module.exports = function(app) {
  var index = require('../controllers/index.server.controller');
  app.get('/', index.render);
};
```

Fig. 4.6. Contenido del controlador básico de index.

+ Implementación básica del enrutamiento de index.

  Dentro de la carpeta de rutas crearemos una para el index, que hará mostrar el mensaje del controlador cuando se acceda a la ruta '/' de la aplicación.

```
module.exports = function(app) {
  var index = require('../controllers/index.server.controller');
  app.get('/', index.render);
};
```

Fig. 4.7. Contenido de enrutador básico de index.

Vamos a realizar los objetivos de la tarea 5.

- Instalación y configuración de módulos en la aplicación express:

  Instalamos los siguientes módulos con npm mediante el fichero package.json. Los comandos utilizados son:

```
{
  "name": "medicalhistory",
  "version": "0.0.1",
  "dependencies": {
    "body-parser": "^1.18.2",
    "compression": "^1.7.1",
    "express": "^4.16.2",
    "method-override": "^2.3.10",
    "morgan": "^1.9.0",
  }
}
```

Fig. 4.8. Fichero package.json con módulos node-env, morgan, compression, body-parser y method-override.

Trás haber añadido los módulos en el fichero package.json, ejecutamos npm para su instalación:

npm install

+ Configuración del módulo node-env.

  Configuramos la variable NODE_ENV para cambiar automáticamente entre un entorno de desarrollo y uno de producción en el fichero express.js.

+ Configuración del módulo morgan.
  Middleware para sistema de login.
+ Configuración del módulo compression.
  Facilita las respuesta con el backend.
+ Configuración del módulo body-parser.
  Para el manejo de peticiones de datos.
+ Configuración del módulo method-override.
  Soporte para verbos HTTP.

Los configuramos en el fichero ./config/express.js

```
// Requerimiento de modulos
var config = require('./config'),
    express = require('express'),
    morgan = require('morgan'),
    compress = require('compression'),
    bodyParser = require('body-parser'),
    methodOverride = require('method-override'),

//Definir el método de configuración de Express
module.exports = function() {
  // Crear una nueva instancia de la apliacion que inicializa la aplicación Express
  var app = express();

  // Usar la variable 'NODE_ENV' para activar los middleware 'morgan' logger o 'compress'
  if (process.env.NODE_ENV === 'development') {
    app.use(morgan('dev'));
  } else if (process.env.NODE_ENV === 'production') {
    app.use(compress());
  }

  // Usar las funciones middleware 'body-parser' y 'method-override'
  app.use(bodyParser.urlencoded({extended: true}));
  app.use(bodyParser.json());
  app.use(methodOverride());

  // Cargar los archivos de enrutamiento
  require('../app/routes/index.server.routes.js')(app);

  // Devolver la instancia de la aplicación Express
  return app;
};
```

Fig. 4.9. Configuración de módulos node-env, morgan, compression, body-parser y method-override.

Para node-env tambien hemos de añadir en el fichero server.js:

Fig. 4.10. Configuración de módulo node-env.

Vamos a realizar los objetivos de la tarea 6.

- Configuración de renderización de las vistas:
  + Instalación y configuración del módulo ejs.
    Añadimos el módulo ejs en el fichero package.json, ejecutamos npm para su instalación:

    npm install

    En el fichero ./config/express.js añadimos:

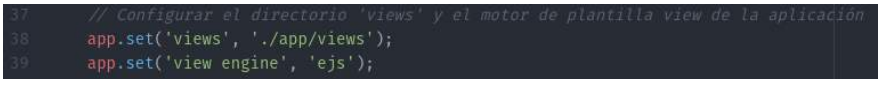

Fig. 4.11. Configuración de módulo ejs.

Vamos a realizar los objetivos de la tarea 7.

- Implementación y configuración para archivos estáticos.

  En el fichero ./config/express.js añadimos:

Fig. 4.12. Configuración de archivos estáticos.

Vamos a realizar los objetivos de la tarea 8.

- Configurar el uso de sesiones:
  + Instalación y configuración del módulo express-sessión.
    Añadimos el módulo express-session en el fichero package.json y ejecutamos npm para su instalación.
    Creamos el directorio ./config/env donde añadimos un fichero con el siguiente contenido:

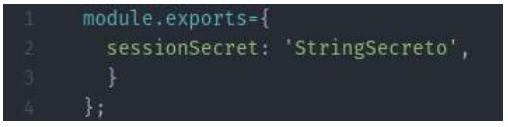

Fig. 4.13. Configuración string secreto de sesión.

y creamos en ./config el fichero config.js donde añadimos:

```
// Carga el archivo de configuración de entorno correcto
module.exports = require('./env/' + process.env.NODE_ENV + '.js');
```

Fig. 4.14. Configuración entorno de ejecución.

Añadimos el directorio ./config y el módulo express-session a express.js,

```
// Requerimiento de modulos
var config = require('./config'),
    session = require('express-session'),
```

Fig. 4.15. Configuración express-session en express.

e incluimos el uso de sesiones con el string secreto que acabamos de crear.

```
// Configuración del middleware 'session'
app.use(session({
  saveUninitialized: true,
  resave: true,
  secret: config.sessionSecret
}));
```

Fig. 4.16. Configuración de sesiones con el string secreto.

Vamos a realizar los objetivos de la tarea 9.
Instalamos mongoDB con pacman mediante el comando:

pacman -S mongodb

- Configurar el uso de base de datos MongoDB:

  Para el correcto funcionamiento de mongodb hemos de colocar el directorio /data/db en nuestra / del sistema. Antes de arrancar nuestra aplicación nodeJS hemos de ejecutar mongoDB a partir de ahora, ya que de no hacerlo, nodeJS nos mostrará un mensaje de error.

  + Instalación y configuración del módulo mongoose.

    Añadimos el módulo mongoose en el fichero package.json y ejecutamos npm para su instalación. A continuación, en ./config/env/development.js añadimos la ruta de nuestra base de datos mongodb.

```
// URI de conexion a la base de datos
db: 'mongodb://localhost/medicalhistory',
```

Fig. 4.17. Configuración de ruta de base de datos mongoDB.

Finalmente creamos el fichero ./config/mongoose.js que conecta express con mongoDB con el siguiente contenido:

```
// Cargar las dependencias del módulo
var config = require('./config'),
  mongoose = require('mongoose');

// Definir el método de configuración de Mongoose
module.exports = function() {
  // Usar Mongoose para conectar a MongoDB
  var db = mongoose.connect(config.db);

  // Devolver la instancia de conexión a Mongoose
  return db;
};
```

Fig. 4.18. Configuración de mongoose.

Vamos a realizar los objetivos de la tarea 10.

- Implementación del módulo ‘Usuarios’ backend:

  + Implementación del modelo con Schema ‘Users’.
    En el directorio ./app/models creamos el Schema usuarios.

```
// Invocar el modo JavaScript 'strict'
'use strict';

// Cargar las dependencias de módulos
var mongoose = require('mongoose'),
    crypto = require('crypto'),
    Schema = mongoose.Schema;

// Definición 'UserSchema'
var UserSchema = new Schema({
  firstName: String,
  lastName: String,
  email: {
  username: {
  password: {
  // Para poder hacer hash del password
  salt: {
  // Almacena la estrategia usada para registrar al usuario
  provider: {
  // Almacena el id del usuario para la estrategia de identificación
  providerId: String,
  // Almacenar el objeto 'user' al utilizar proveedores
  providerData: {},
  created: {
});

// Configurar la propiedad virtual 'fullname'
UserSchema.virtual('fullName').get(function() {

// Usar un middleware pre-save para hash la contraseña
UserSchema.pre('save', function(next) {

// Crear un método instancia para hashing una contraseña
UserSchema.methods.hashPassword = function(password) {

// Crear un método instancia para autentificar usuario
UserSchema.methods.authenticate = function(password) {

// Encontrar posibles username no usados
UserSchema.statics.findUniqueUsername = function(username, suffix, callback) {

// Configurar el 'UserSchema' para usar getters y virtuals cuando se transforme a JSON
UserSchema.set('toJSON',{

// Crear el modelo 'User' a partir del 'UserSchema'
mongoose.model('User', UserSchema);
```

Fig. 4.19. Schema de usuario.

+ Cargar el modelo ‘Users’ en el middleware mongoose.

  En el fichero mongoose añadimos el schema de usuario que acabamos de crear.

```
// Cargar el modelo 'User'
require('../app/models/user.server.model');
```

Fig. 4.20. Cargar Schema de usuario en mongoose.

+ Implementación del controlador ‘Users’.

  Creamos en ./app/controllers el controlador para operaciones crud, listar y buscar único sobre usuarios.

```
// Cargar el modelo Mongoose 'User'
var User = require('mongoose').model('User'),

// Crear un nuevo método controller 'create'
exports.create = function(req, res, next) {

// Crear un nuevo método controller 'list'
exports.list = function(req, res, next) {

// Crear un nuevo método controller
exports.read = function(req, res){

// Crear un nuevo método controller 'update'
exports.update = function(req, res, next) {

// Crear un nuevo método controller 'delete'
exports.delete = function(req, res, next) {

// Crear un nuevo método controller 'userByID'
exports.userByID = function(req, res, next, id) {
```

Fig. 4.21. Controlador de usuario.

+ Implementación de las rutas de ‘Users’.

  En ./app/routes creamos las rutas.

```
// Invocar el modo 'strict' de JavaScript
'use strict';

// Cargar las dependencias del módulo
var users = require('../../app/controllers/users.server.controller'),
    passport = require('passport');

//Definir el método del módulo routes
module.exports = function(app) {
  //Configurar las rutas 'signup'
  app.route('/signup')
  app.route('/signin')

  // Configurar las rutas Google OAuth
  app.get('/oauth/google', passport.authenticate('google', {
  app.get('/oauth/google/callback', passport.authenticate('google', {

  //Configurar la route 'signout'
  app.get('/signout', users.signout);

};
```

Fig. 4.22. Rutas de usuario.

+ Cargar las rutas de ‘Users’ en la aplicación express.

  Cargamos el fichero de rutas en express.js

```
46      // Cargar los archivos de enrutamiento
47      require('../app/routes/index.server.routes.js')(app);
48      require('../app/routes/users.server.routes.js')(app);
```

Fig. 4.23. Carga de rutas de usuario en express.

Vamos a realizar los objetivos de la tarea 11.

- Configuración e implementación de registro e identificación:

  Añadimos el módulo passport, passport-local y passport-google-oauth en el fichero package.json y ejecutamos npm para su instalación.

  Dentro de el directorio config creamos el directorio strategies, donde tendremos la configuración de las diferentes estrategias de autenticación para nuestra aplicación.

  + Estrategía local.

    Creamos el fichero ./strategies/local.js donde tendremos el siguiente código:

```
// Requerimos los objetos 'passport', 'passport-local', 'mongoose'
var passport = require('passport'),
    LocalStrategy = require('passport-local').Strategy,
    User = require('mongoose').model('User');

// Registramos la estrategia del método passport.use
module.exports = function() {
  passport.use(new LocalStrategy(function(username, password, done){
    User.findOne({
      username : username
    }, function(err, user){
      if(err){
        return done(err);
      }
      if (!user){
        return done(null, false, {
          message: 'Usuario Desconocido / Unknow user'
        });
      }
      if (!user.authenticate(password)){
        return done (null, false,{
          message: 'Contraseña Inválida / Invalid password'
        });
      }
      return done(null, user);
      });
  }));
};
```

Fig. 4.24. Estrategia local de autenticación.

  + Estrategía google.

    Creamos el fichero ./strategies/google.js donde tendremos el siguiente código:

```
// Requerimiento de modulos
var passport = require('passport'),
    url = require('url'),
    GoogleStrategy = require('passport-google-oauth').OAuth2Strategy,
    config = require('../config'),
    users = require('../../app/controllers/users.server.controller');

module.exports = function(){
  passport.use(new GoogleStrategy({
    clientID: config.google.clientID,
    clientSecret: config.google.clientSecret,
    callbackURL: config.google.callbackURL,
    passReqToCallback: true
  },
  function(req, accessToken, refreshToken, profile, done) {
    var providerData = profile._json;
    providerData.accessToken = accessToken;
    providerData.refreshToken = refreshToken;

    var providerUserProfile = {
      firstName: profile.name.givenName,
      lastName: profile.name.familyName,
      fullName: profile.displayName,
      email: profile.emails[0].value,
      username: profile.username,
      provider: 'google',
      providerId: profile.id,
      providerData: providerData
    };

    users.saveOAuthUserProfile(req, providerUserProfile, done);
  }));
};
```

Fig. 4.25. Estrategia google de autenticación.

Vamos a realizar los objetivos de la tarea 12.

- Configuración de la estructura del Frontend.

  Instalamos Bower con npm de manera global en nuestro sistema mediante el comando:

  npm install -g bower

  Creamos el fichero bower.js, que funciona de manera similar al de package.json, donde añadiremos las dependencias correspondientes a angularJS para su instalación.

```
{
  "name": "medicalhistory",
  "version": "0.0.1",
  "dependencies": {
    "angular": "^1.6.9"
  }
}
```

Fig. 4.26. Contenido de bower.json con angular.

Tambien hemos de crear en la raíz de nuestra aplicación el fichero .bowerrc, donde estableceremos la ubicación de los paquetes instalados con bower. Tendrá el siguiente contenido:

Fig. 4.27. Contenido de bowerrc.

Ejecutamos el siguiente comando para instalar angular:

bower install

Dentro del directorio ./app/views creamos el fichero index.ejs, que será nuestra página principal de la aplicación, signup y signin con el siguiente contenido:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Usar la propiedad 'title' para renderizar un titulo de página -->
    <title><%= title %></title>
  </head>
  <body>
    <!-- Si un usuario es autenticado, mostrar el nombre completo del usuario, si no mostrar los enlace de autenticación -->
    <% if ( user ) { %>
      <a href="/signout">Cerrar Sesión</a>
    <%} else { %>
      <a href="/signup">Crear cuenta</a>
      <a href="/signin">Identificarse</a>
    <% } %>

    <!-- <section ng-include="'example/views/example.client.view.html'"></section> -->
    <section ng-view></section>
    <script type="text/javascript">
      window.user = <%- user || 'null'%>
    </script>

    <script type="text/javascript" src="/lib/angular/angular.js"></script>
    <script type="text/javascript" src="/lib/angular-route/angular-route.js"></script>

    <script type="text/javascript" src="/users/users.client.module.js"></script>
    <script type="text/javascript" src="/users/services/authentication.client.service.js"></script>

    <script type="text/javascript" src="/application.js"></script>

  </body>
</html>
```

Fig. 4.28. Contenido de index.ejs.

Fig. 4.29. Contenido de signin.ejs.

Fig. 4.30. Contenido de signup.ejs.

Pruebas funcionales del sprint 1:

ID	P01_01
Resultado	Válido
Sistema	NodeJS
Descripción	Comprovación de arranque del servidor con la configuración establecida
Acciones	1. Iniciar el servidor en el puerto configurado.

TABLA 4.4. PRUEBA FUNCIONAL 1 DEL PRIMER SPRINT.

ID	P01_02
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas se muestran correctamente
Acciones	1. Acceder a las diferentes vistas creadas

TABLA 4.5. PRUEBA FUNCIONAL 2 DEL PRIMER SPRINT.

ID	P01_03
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Funcionamiento de MVC backend
Acciones	1. Registrar un usuario 2. Identificarse con un usuario 3. Cerrar sesión con un usuario

TABLA 4.6. PRUEBA FUNCIONAL 3 DEL PRIMER SPRINT.

Vemos un resumen del tiempo estimado y utilizado en el primer sprint.

Historia	Tiempo estimado	Tiempo final
T01	2,3333333333333	5
T02	1	2
T03	1	1
T04	3,6666666666667	5
T05	1	4
T06	3,6666666666667	2
T07	1,6666666666667	0,5
T08	1,6666666666667	2
T09	2,3333333333333	2
T10	9,6666666666667	10
T11	6	7
T12	1	0,2
Total	35	40,7

TABLA 4.7. RESUMEN DE TIEMPO PRIMER SPRINT.

Tarea	Hsitoria de Usuario	Estado Inicial	Estado Estimado	Estado Final
T01	HU12	No iniciada	Completada	Completada
T02	HU12	No iniciada	Completada	Completada
T03	HU12	No iniciada	Completada	Completada
T04	HU12	No iniciada	Completada	Completada
T05	HU12	No iniciada	Completada	Completada
T06	HU12	No iniciada	Completada	Completada
T07	HU12	No iniciada	Completada	Completada
T08	HU11	No iniciada	Completada	Completada
T09	HU09	No iniciada	Completada	Completada
T10	HU11	No iniciada	Completada	Completada
T11	HU11	No iniciada	Completada	Completada
T12	HU12	No iniciada	Completada	Completada
T13	HU11	No iniciada	No iniciada	No iniciada
T14	HU07	No iniciada	No iniciada	No iniciada
T15	HU07	No iniciada	No iniciada	No iniciada
T16	HU01	No iniciada	No iniciada	No iniciada
T17	HU01	No iniciada	No iniciada	No iniciada
T18	HU09	No iniciada	No iniciada	No iniciada
T19	HU09	No iniciada	No iniciada	No iniciada
T20	HU09	No iniciada	No iniciada	No iniciada
T21	HU09	No iniciada	No iniciada	No iniciada
T22	HU04	No iniciada	No iniciada	No iniciada
T23	HU04	No iniciada	No iniciada	No iniciada
T24	HU07	No iniciada	No iniciada	No iniciada
T25	HU03	No iniciada	No iniciada	No iniciada
T26	HU01	No iniciada	No iniciada	No iniciada
T27	HU04	No iniciada	No iniciada	No iniciada
T28	HU10	No iniciada	No iniciada	No iniciada
T29	HU10	No iniciada	No iniciada	No iniciada
T30	HU10	No iniciada	No iniciada	No iniciada
T31	HU10	No iniciada	No iniciada	No iniciada
T32	HU10	No iniciada	No iniciada	No iniciada
T33	HU10	No iniciada	No iniciada	No iniciada

TABLA 4.8. RESUMEN DEL PRIMER SPRINT.

4.3.4. Segundo Sprint

Horas máximas	40
Horas utilizadas	32
Horas restantes	8
Tareas	**Horas estimadas**
T13	8,6666666666667
T14	9,6666666666667
T15	13,666666666667

TABLA 4.9. TAREAS ESCOGIDAS SEGUNDO SPRINT.

Vamos a realizar los objetivos de la tarea 13.

- Implementación del módulo 'Principal' frontend:

  En nuestra carpeta public, crearemos el árbol de carpetas MVC que implementa la vista principal de la aplicación.

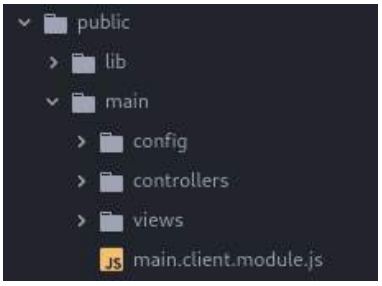

Fig. 4.31. MVC de la vista de la página principal.

+ Crear el módulo 'Main'.

  Creamos el fichero main.client.module.js con el nuevo módulo.

```
1   angular.module('main', []);
```

Fig. 4.32. Módulo main.

+ Crear el controlador 'Main'.

  En la carpeta controller creamos el fichero main.client.controller.js con el controlador de main.

```
angular.module('main').controller('MainController', ['$scope', 'Authentication','ShareDataService',
  function($scope, Authentication,ShareDataService) {
    $scope.authentication = Authentication;
    $scope.shareDataService = ShareDataService;
  }
]);
```

Fig. 4.33. Controlador del módulo main.

+ Crear las rutas de 'Main'.

  En la carpeta config creamos el fichero main.client.routes.js con las rutas de main.

```
angular.module('main').config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
    when('/', {
      templateUrl: 'main/views/main.client.view.html'
    }).
    otherwise({
      redirectTo: '/'
    });
  }
]);
```

Fig. 4.34. Rutas del módulo main.

+ Crear las vistas de 'Main'.

  En la carpeta views creamos el fichero main.client.view.js con las siguientes vistas de main.

  - Usuarios no autenticados.

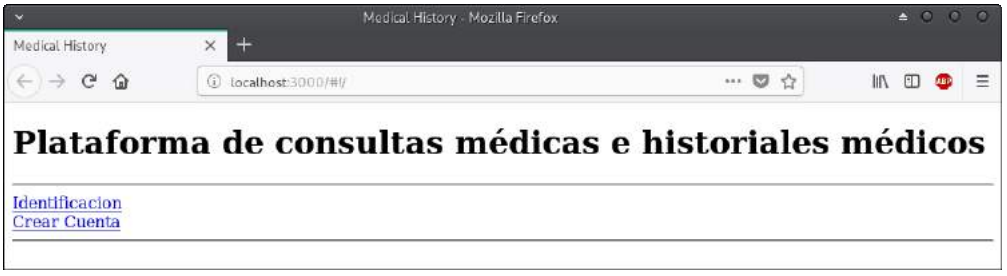

Fig. 4.35. Vista de la página principal de usuarios no autenticados.

  - Usuarios autenticados.

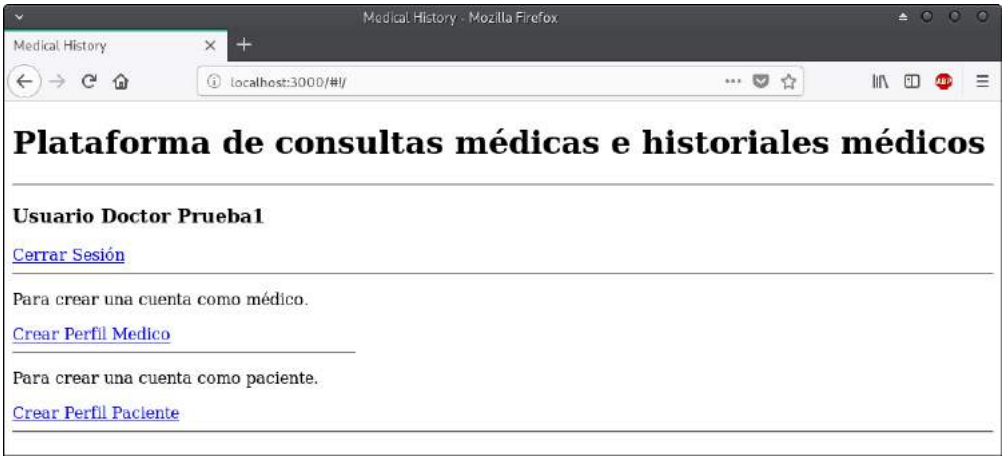

Fig. 4.36. Vista de la página principal de usuarios autenticados.

> Médicos.

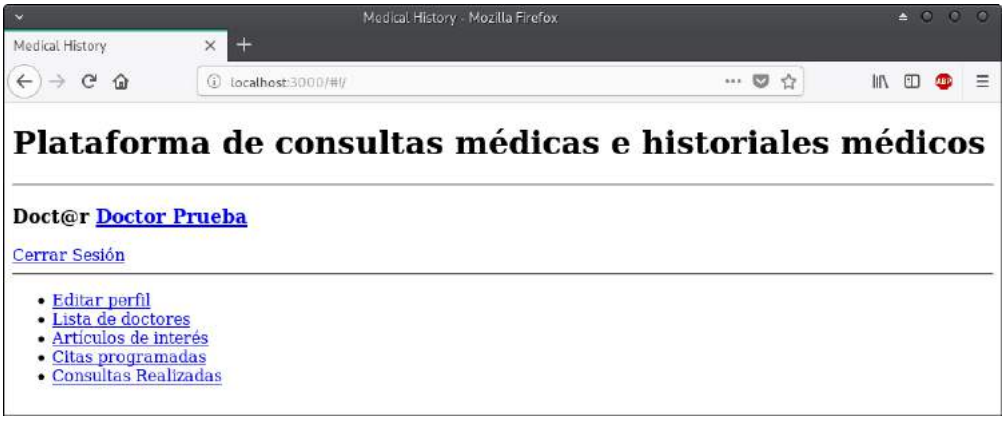

Fig. 4.37. Vista página principal usuarios médicos.

> Pacientes.

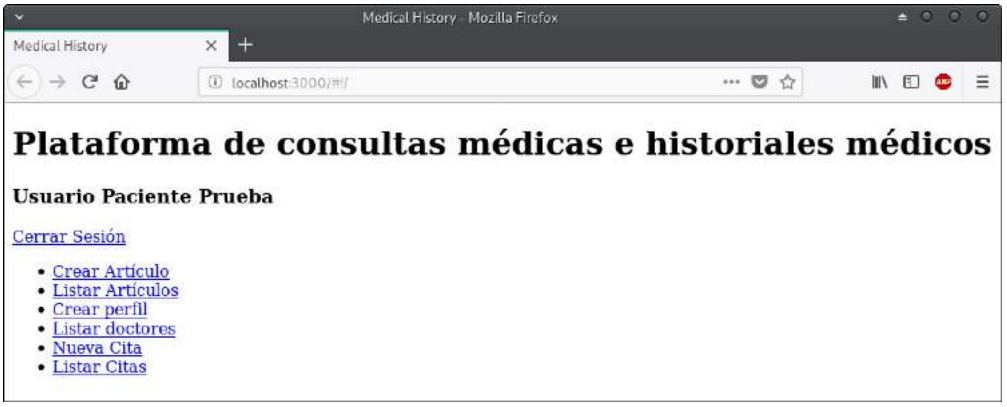

Fig. 4.38. Vista de la página principal de usuarios pacientes.

Vamos a realizar los objetivos de la tarea 14.

- Implementación del módulo ‘Artículos’ backend:

+ Implementación del modelo con Schema ‘Artículos’.

  En el directorio ./app/models creamos el Schema artículos.

```
var mongoose = require('mongoose'),
    Schema = mongoose.Schema,
    moment = require('moment');

var ArticleSchema = new Schema({
  // Fecha de creación
  creado: {
  // Titulo del articulo
  titulo: {
  // Contenido del articulo
  contenido: {
  // Url del articulo
  url: {
  // Creador (médico) del articulo
  creador: {
});

//Crear el model 'Article' a partir del 'ArticleSchema'
mongoose.model('Article', ArticleSchema);
```

Fig. 4.39. Schema de artículo.

+ Cargar el modelo ‘Artículos’ en el middleware mongoose.

  En el fichero mongoose añadimos el schema de artículos que acabamos de crear.

```
    // Cargar el modelo 'Article'
    require('../app/models/article.server.model');
```

Fig. 4.40. Cargar Schema de artículo en mongoose.

+ Implementación del controlador ‘Artículos’.

  Creamos en ./app/controllers el controlador para operaciones crud, listar y buscar único sobre usuarios.

Fig. 4.41. Controlador de artículo.

+ Implementación de las rutas de 'Artículos'.

  En ./app/routes creamos las rutas con los principios REST.

```
// Invocar modo JavaScript 'strict'
'use strict';

// Cargar las dependencias del módulo
var users = require('../../app/controllers/users.server.controller'),
  articles = require('../../app/controllers/articles.server.controller');

// Definir el método routes de module
module.exports = function(app) {
  // Configurar la rutas base a 'articles'
  app.route('/api/articles')
     .get(articles.list)
     .post(users.requiresLogin, articles.create);

  // Configurar las rutas 'articles' parametrizadas
  app.route('/api/articles/:articleId')
     .get(articles.read)
     .put(users.requiresLogin, articles.hasAuthorization, articles.update)
     .delete(users.requiresLogin, articles.hasAuthorization, articles.delete);

  // Configurar el parámetro middleware 'articleId'
  app.param('articleId', articles.articleByID);
};
```

Fig. 4.42. Rutas de artículo.

+ Cargar las rutas de 'Artículo' en la aplicación express.

Cargamos el fichero de rutas en express.js

```
require('../app/routes/articles.server.routes.js')(app);
```

Fig. 4.43. Carga de rutas de artículo en express.

Vamos a realizar los objetivos de la tarea 15.

- Implementación del módulo 'Artículos' frontend:

  Creamos el árbol de directorios similar al del módulo main que realizamos anteriormente donde crearemos los siguientes ficheros.

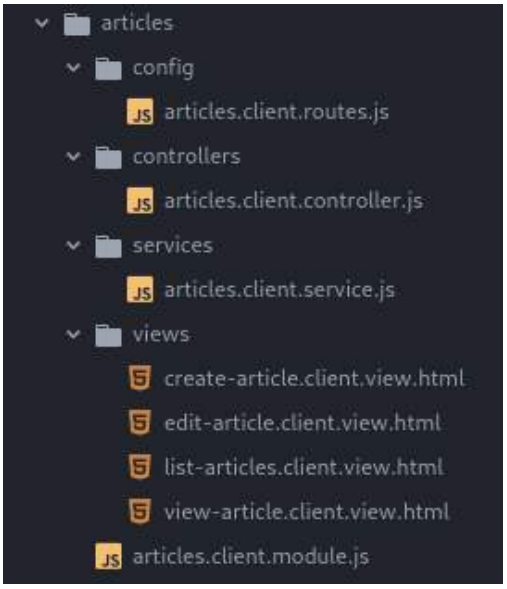

Fig. 4.44. Árbol de directorios MVC del módulo articles.

+ Crear el módulo 'Articles'.

  Creamos el módulo articles de igual manera que hicimos con el de main.

+ Crear el servicio 'Articles'.

  Creamos un servicio de que nos devolverá un recurso article uńico cuando queramos acceder o modificar un artículo determinado.

Fig. 4.45. Servicio del módulo articles.

+ Crear el controlador ‘Articles’.

  Dentro del controlador de articles, implementaremos las siguientes funciones.

  - Crear artículo.

Fig. 4.46. Función crear del módulo articles.

  - Buscar todos los artículos.

Fig. 4.47. Función buscar del módulo articles.

  - Buscar un artículo.

Fig. 4.48. Función buscar único del módulo articles.

  - Actualizar un artículo.

Fig. 4.49. Función actualizar del módulo articles.

  - Borrar un artículo.

```
// Crear un nuevo método controller para borrar un único artículo
        $scope.delete = function(article) {
            // Si un artículo fue enviado al método, borrarlo
            if (article) {
                // Usar el método '$remove' del artículo para borrar el artículo
                article.$remove(function() {
                    // Eliminar el artículo de la lista de artículos
                    for (var i in $scope.articles) {
                        if ($scope.articles[i] == article) {
                            $scope.articles.splice(i, 1);
                        }
                    }
                });
            } else {
                // En otro caso, usar el método '$remove' de article para borrar el article
                $scope.article.$remove(function() {
                    $location.path('articles');
                });
            }
        };

    }
]);
```

Fig. 4.50. Función eliminar del módulo articles.

+ Crear las rutas de 'Artículo'.

  En la carpeta config creamos el fichero articles.client.routes.js con las rutas de articles.

```
// Invocar modo JavaScript 'strict'
'use strict';

// Configurar el módulo routes de 'articles'
angular.module('articles').config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
    when('/articles', {
      templateUrl: 'articles/views/list-articles.client.view.html'
    }).
    when('/articles/create', {
      templateUrl: 'articles/views/create-article.client.view.html'
    }).
    when('/articles/:articleId', {
      templateUrl: 'articles/views/view-article.client.view.html'
    }).
    when('/articles/:articleId/edit', {
      templateUrl: 'articles/views/edit-article.client.view.html'
    });
  }
]);
```

Fig. 4.51. Rutas del módulo articles.

Finalmente agregamos los ficheros de las vistas a nuestra página única index.ejs.

```
<!-- Enlaces a los ficheros de articles -->
<script type="text/javascript" src="/articles/articles.client.module.js"></script>
<script type="text/javascript" src="/articles/controllers/articles.client.controller.js"></script>
<script type="text/javascript" src="/articles/services/articles.client.service.js"></script>
<script type="text/javascript" src="/articles/config/articles.client.routes.js"></script>
```

Fig. 4.52. Agregación del módulo articles a index.ejs.

+ Crear las vistas de ‘Artículo’.

  En la carpeta views creamos los siguientes ficheros que contienen las vistas de articles.

  - Crear artículo.

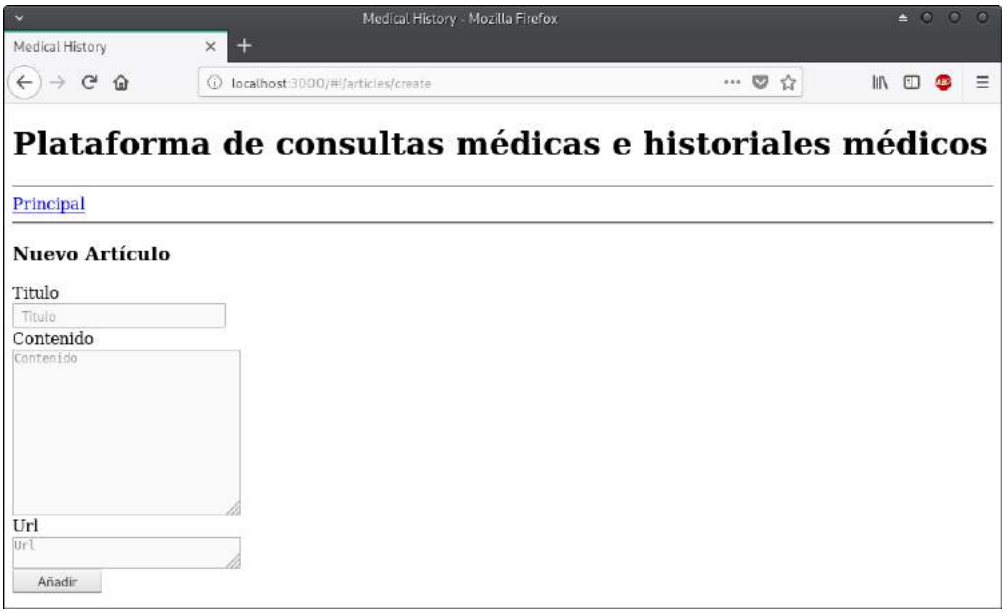

Fig. 4.53. Vista de la página crear artículo.

  - Editar artículo.

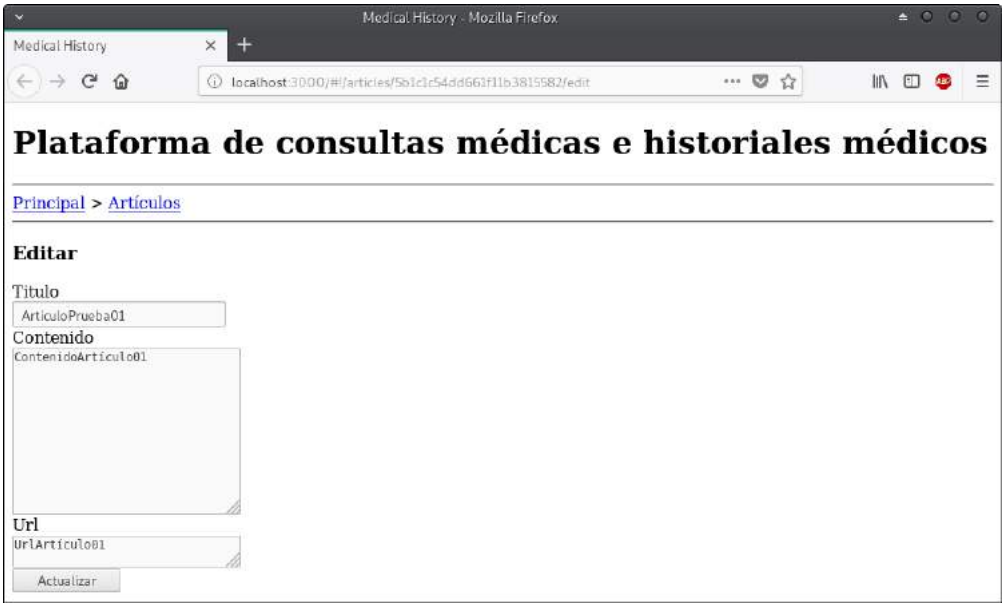

Fig. 4.54. Vista de la página editar artículo.

  - Listar artículos.

Fig. 4.55. Vista de la página de lista de artículos.

- Visualizar un artículo.

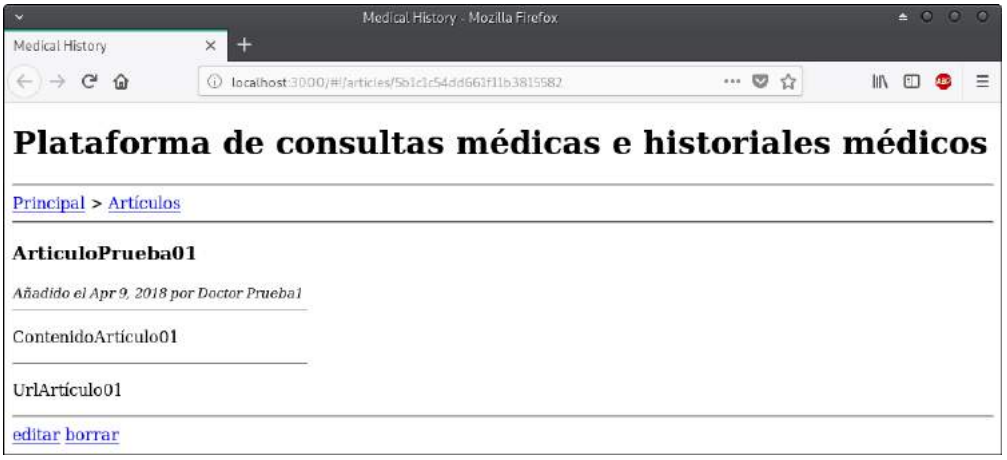

Fig. 4.56. Vista para el médico creador de un artículo de la página de un artículo.

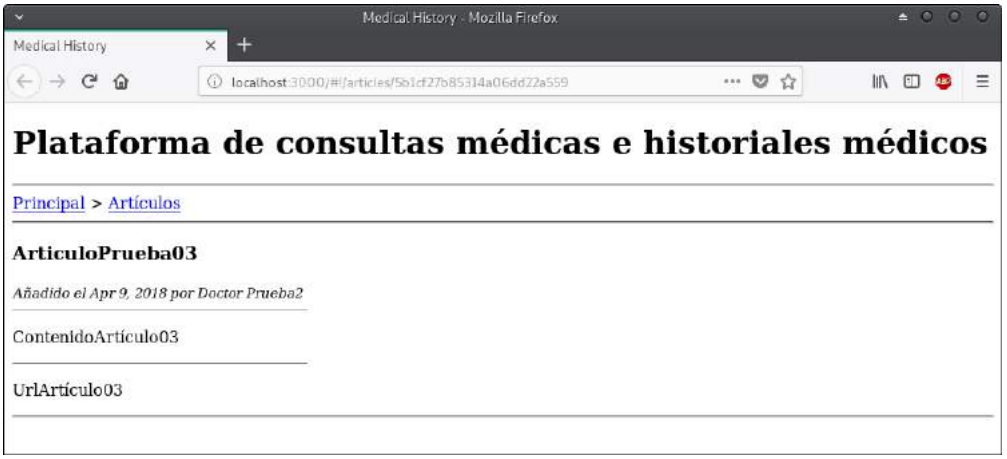

Fig. 4.57. Vista para un médico no creador de un artículo de la página de un artículo.

Pruebas funcionales del sprint 2:

ID	P02_01
Resultado	Válido
Sistema	Cliente / servidor
Descripción	La vista de la página principal se muestra y funciona correctamente
Acciones	1. Acceder y probar la vista principal de la aplicación para: ◦ Usuarios no autenticados ◦ Usuarios autenticados

TABLA 4.10. PRUEBA FUNCIONAL 1 DEL SEGUNDO SPRINT.

ID	P02_02
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de artículos se muestran y funcionan correctamente
Acciones	1. Acceder y probar las vistas de artículo ◦ Crear artículo. ◦ Buscar todos los artículos. ◦ Buscar un artículo. ◦ Actualizar un artículo. ◦ Borrar un artículo.

TABLA 4.11. PRUEBA FUNCIONAL 2 DEL SEGUNDO SPRINT.

Vemos un resumen del tiempo estimado y utilizado en el segundo sprint.

Historia	Tiempo estimado	Tiempo final
T13	8,6666666666667	6
T14	9,6666666666667	8
T15	13,666666666667	15
Total	32	29

TABLA 4.12. RESUMEN DE TIEMPO SEGUNDO SPRINT.

Tarea	Hsitoria de Usuario	Estado Inicial	Estado Estimado	Estado Final
T01	HU12	Completada	Completada	Completada
T02	HU12	Completada	Completada	Completada
T03	HU12	Completada	Completada	Completada
T04	HU12	Completada	Completada	Completada
T05	HU12	Completada	Completada	Completada
T06	HU12	Completada	Completada	Completada
T07	HU12	Completada	Completada	Completada
T08	HU11	Completada	Completada	Completada
T09	HU09	Completada	Completada	Completada
T10	HU11	Completada	Completada	Completada
T11	HU11	Completada	Completada	Completada
T12	HU12	Completada	Completada	Completada
T13	HU11	No iniciada	Completada	Completada
T14	HU07	No iniciada	Completada	Completada
T15	HU07	No iniciada	Completada	Completada
T16	HU01	No iniciada	No iniciada	No iniciada
T17	HU01	No iniciada	No iniciada	No iniciada
T18	HU09	No iniciada	No iniciada	No iniciada
T19	HU09	No iniciada	No iniciada	No iniciada
T20	HU09	No iniciada	No iniciada	No iniciada
T21	HU09	No iniciada	No iniciada	No iniciada
T22	HU04	No iniciada	No iniciada	No iniciada
T23	HU04	No iniciada	No iniciada	No iniciada
T24	HU07	No iniciada	No iniciada	No iniciada
T25	HU03	No iniciada	No iniciada	No iniciada
T26	HU01	No iniciada	No iniciada	No iniciada
T27	HU04	No iniciada	No iniciada	No iniciada
T28	HU10	No iniciada	No iniciada	No iniciada
T29	HU10	No iniciada	No iniciada	No iniciada
T30	HU10	No iniciada	No iniciada	No iniciada
T31	HU10	No iniciada	No iniciada	No iniciada
T32	HU10	No iniciada	No iniciada	No iniciada
T33	HU10	No iniciada	No iniciada	No iniciada

TABLA 4.13. RESUMEN DEL SEGUNDO SPRINT.

4.3.5. Tercer Sprint

Horas máximas	40
Horas utilizadas	38,3333333333333
Horas restantes	1,66666666666669
Tareas	Horas estimadas
T16	8,66666666666667
T17	8,66666666666667
T18	9,66666666666667
T19	11,3333333333333

TABLA 4.14. TAREAS ESCOGIDAS TERCER SPRINT.

Vamos a realizar los objetivos de la tarea 16.

- Implementación del módulo 'Citas' backend:

  Al igual que hicimos con los artículos, repetiremos la misma implementación tanto en el backend como en el frontend.

  + Implementación del modelo con Schema 'Citas'.

    Creamos el fichero appointment.server.model.js que contiene el schema de una cita.

  + Cargar el modelo 'Citas' en el middleware mongoose.

    Agregamos el schema al fichero mongoose.

  + Implementación del controlador 'Citas'.

    Creamos el fichero appointment.server.controller.js que contiene las operaciones CRUD de una cita.

  + Implementación de las rutas de 'Citas'.

    Creamos el fichero appointment.server.routes.js que contiene las rutas de una cita.

  + Cargar las rutas de 'Citas' en la aplicación express.

    Agregamos las rutas al fichero express.js.

Vamos a realizar los objetivos de la tarea 17.

- Implementación del módulo 'Citas' frontend:

  Creamos el árbol de directorios similar al de los módulos que realizamos anteriormente donde crearemos los siguientes ficheros.

  + Crear el módulo 'Citas'.

    Creamos el módulo appointment en el fichero appointment.client.module.js.

  + Crear el servicio 'Citas'.

    Creamos un servicio de que nos devolverá un recurso appointment uńico cuando queramos acceder o modificar una cita determinada.

+ Crear el controlador ‘Citas’.

  Creamos el fichero appointment.client.controller.js con las siguiente funciones.

  - Crear Citas.
  - Buscar todos los citas.
  - Buscar una citas.
  - Modificar cita.
  - Eliminar cita.

+ Crear las rutas de ‘Cita’.

  Creamos el fichero appointment.client.routes.js con las rutas de las vistas del módulo appointment.

+ Crear las vistas de ‘Cita’.

  Creamos las siguientes vistas:

  - Crear cita.

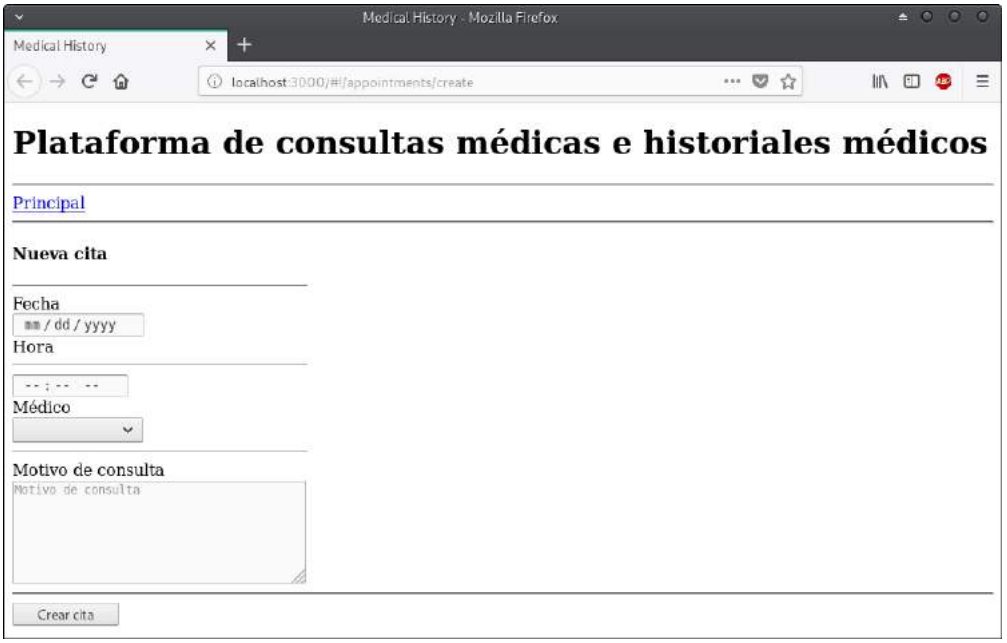

Fig. 4.58. Vista de la página crear cita.

  - Editar cita.

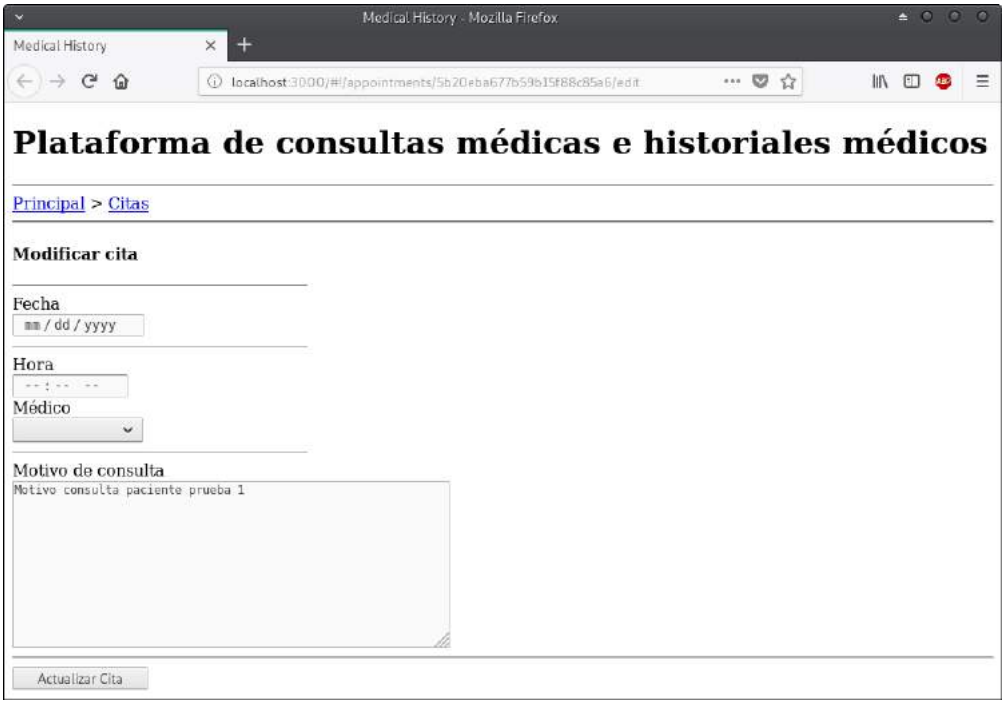

Fig. 4.59. Vista de la página editar cita.

- Listar citas.

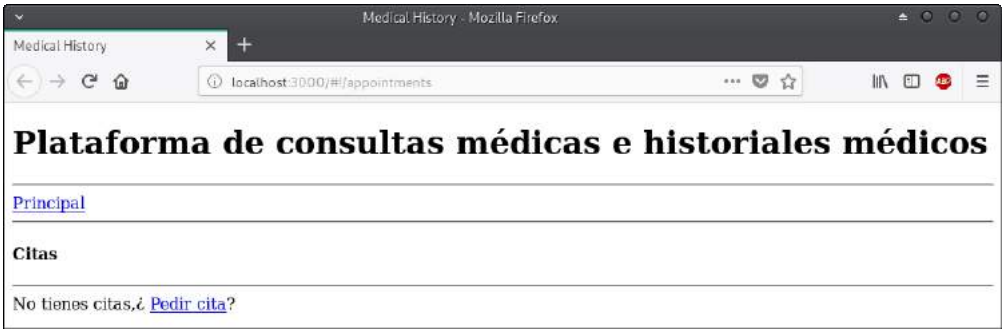

Fig. 4.60. Vista de la página de lista de citas vacía para un paciente.

Fig. 4.61. Vista de la página de lista de citas para un paciente.

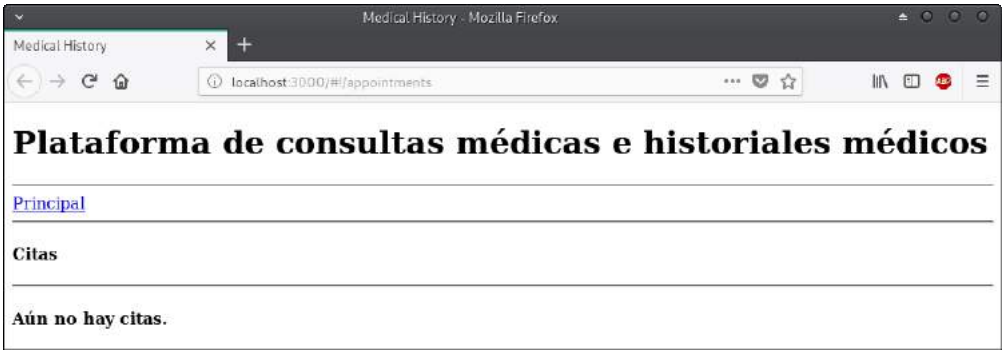

Fig. 4.62. Vista de la página de lista de citas vacía para un médico.

Fig. 4.63. Vista de la página de lista de citas para un médico.

- Visualizar cita.

Fig. 4.64. Vista de la página de una cita para un paciente.

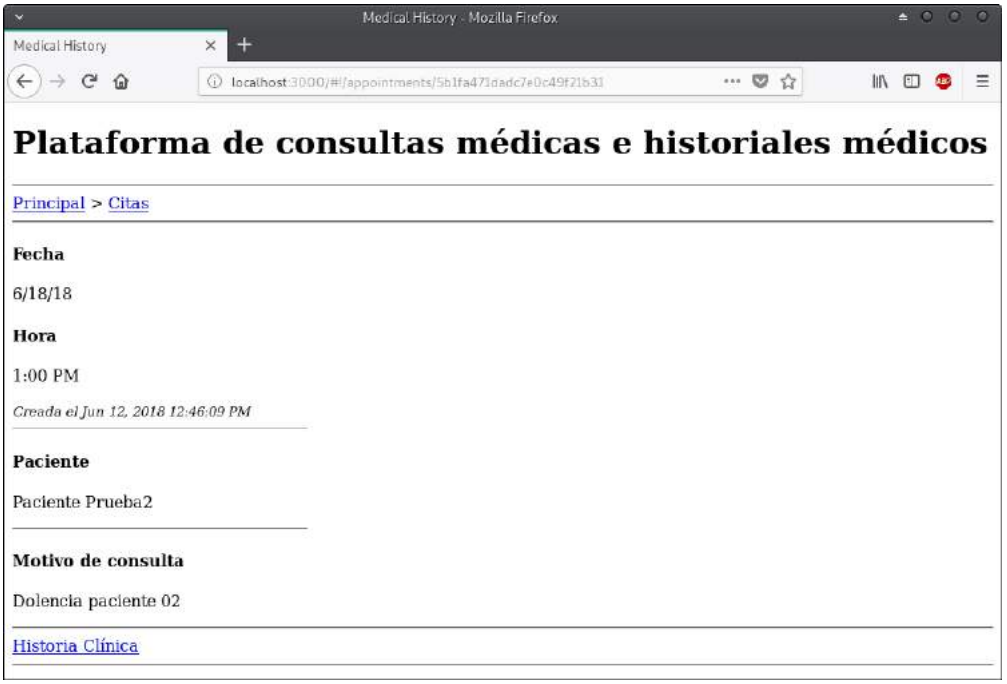

Fig. 4.65. Vista de la página de una cita para un médico.

Vamos a realizar los objetivos de la tarea 18.

- Implementación del módulo ‘Doctores’ backend:

Al igual que hicimos con los anteriores módulos, repetiremos la misma implementación tanto en el backend como en el frontend.

+ Implementación del modelo con Schema 'Doctores'.
  Creamos el fichero profiledoctor.server.model.js que contiene el schema de una cita.
+ Cargar el modelo 'Doctores' en el middleware mongoose.
  Agregamos el schema al fichero mongoose.
+ Implementación del controlador 'Doctores'.
  Creamos el fichero profiledoctor.server.controller.js que contiene las operaciones CRUD de una cita.
+ Implementación de las rutas de 'Doctores'.
  Creamos el fichero profilesdoctor.server.routes.js que contiene las rutas de una cita.
+ Cargar las rutas de 'Doctores' en la aplicación express.
  Agregamos las rutas al fichero express.js.

Vamos a realizar los objetivos de la tarea 19.

- Implementación del módulo 'Doctores' frontend:

  Creamos el árbol de directorios similar al de los módulos que realizamos anteriormente donde crearemos los siguientes ficheros.

  + Crear el módulo 'Doctores'.
    Creamos el módulo profiledoctor en el fichero profiledoctor.client.module.js.
  + Crear el servicio 'Doctores'.
    Creamos un servicio de que nos devolverá un recurso profiledoctor uńico cuando queramos acceder o modificar un perfil médico determinada.
  + Crear el controlador 'Doctores'.
    Creamos el fichero profiledoctor.client.controller.js con las siguiente funciones.
    - Crear Doctor.
    - Buscar todos los Doctores.
    - Buscar un doctor.
    - Modificar doctor.
    - Eliminar doctor.
  + Crear las rutas de 'Doctores'.
    Creamos el fichero profiledoctor.client.routes.js con las rutas de las vistas del módulo profiledoctor.
  + Crear las vistas de 'Doctor'.
    Creamos las siguientes vistas:

- Crear perfil.

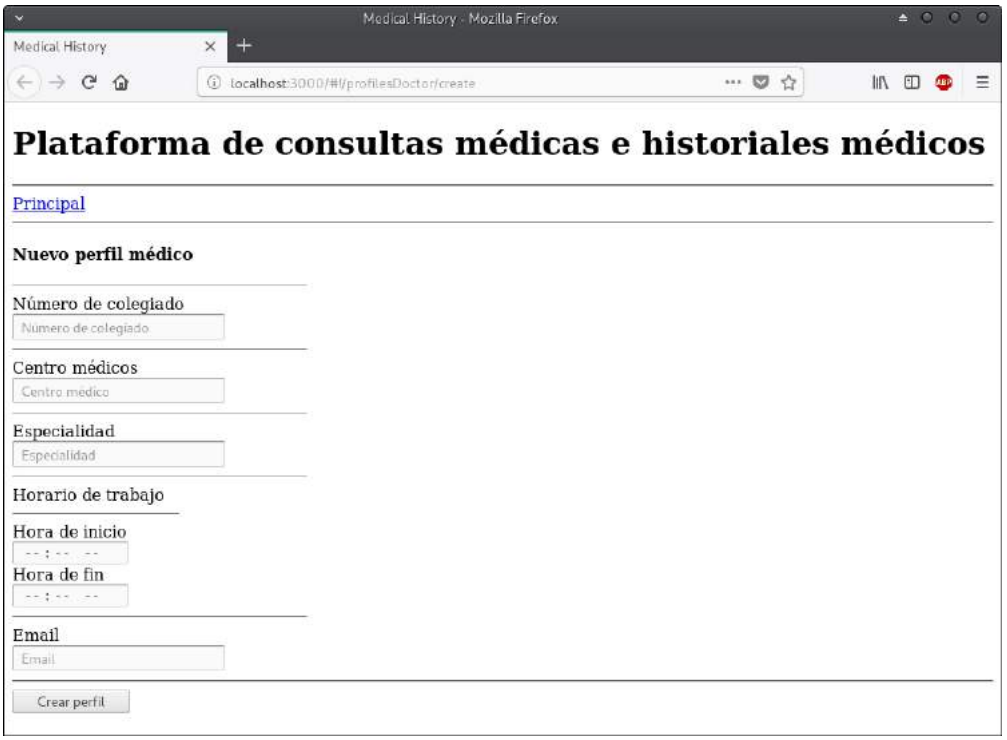

Fig. 4.66. Vista de la página de crear perfil médico.

- Editar perfil.

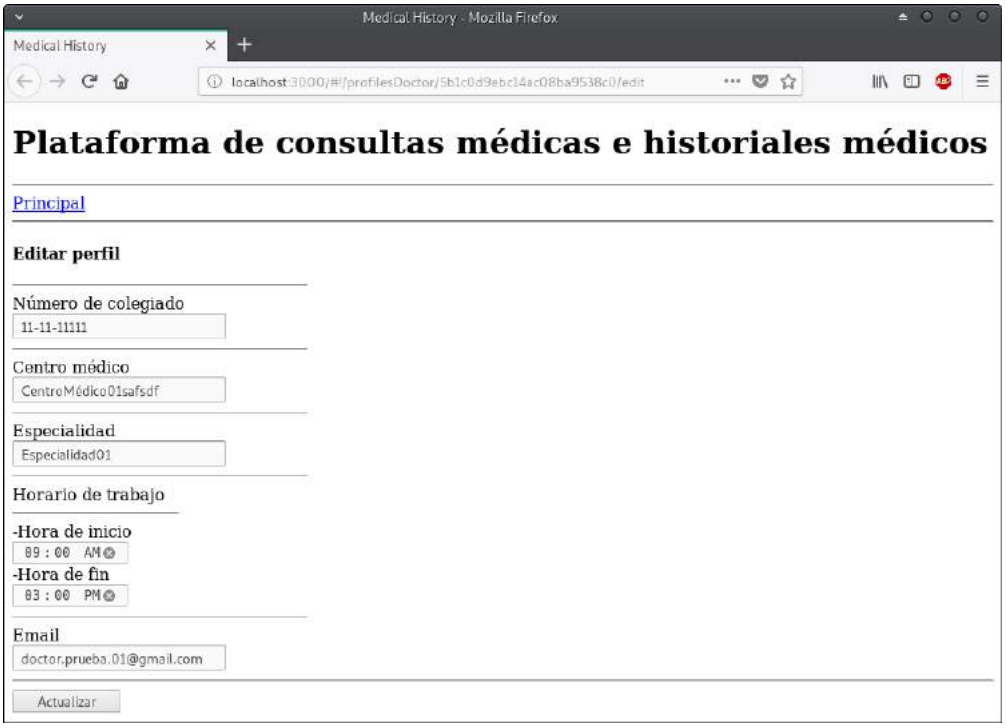

Fig. 4.67. Vista de la página de editar perfil médico.

- Listar doctores.

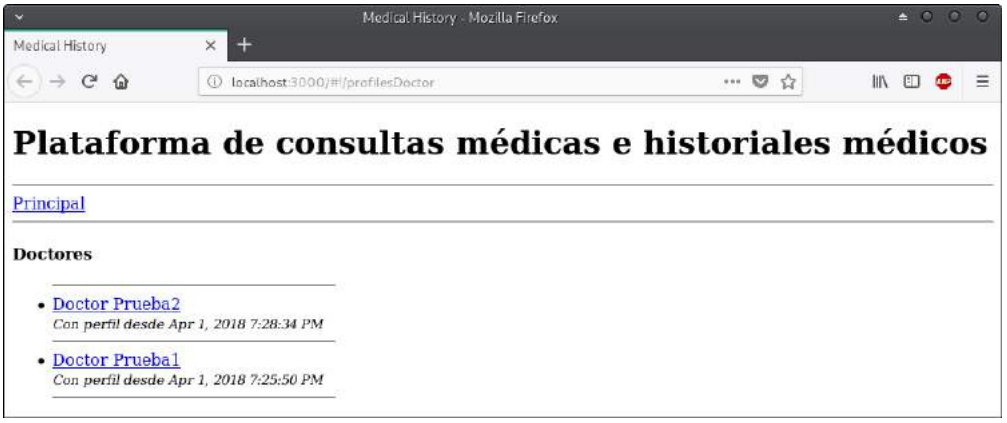

Fig. 4.68. Vista de la página de lista de perfiles médicos.

- Visualizar doctor.

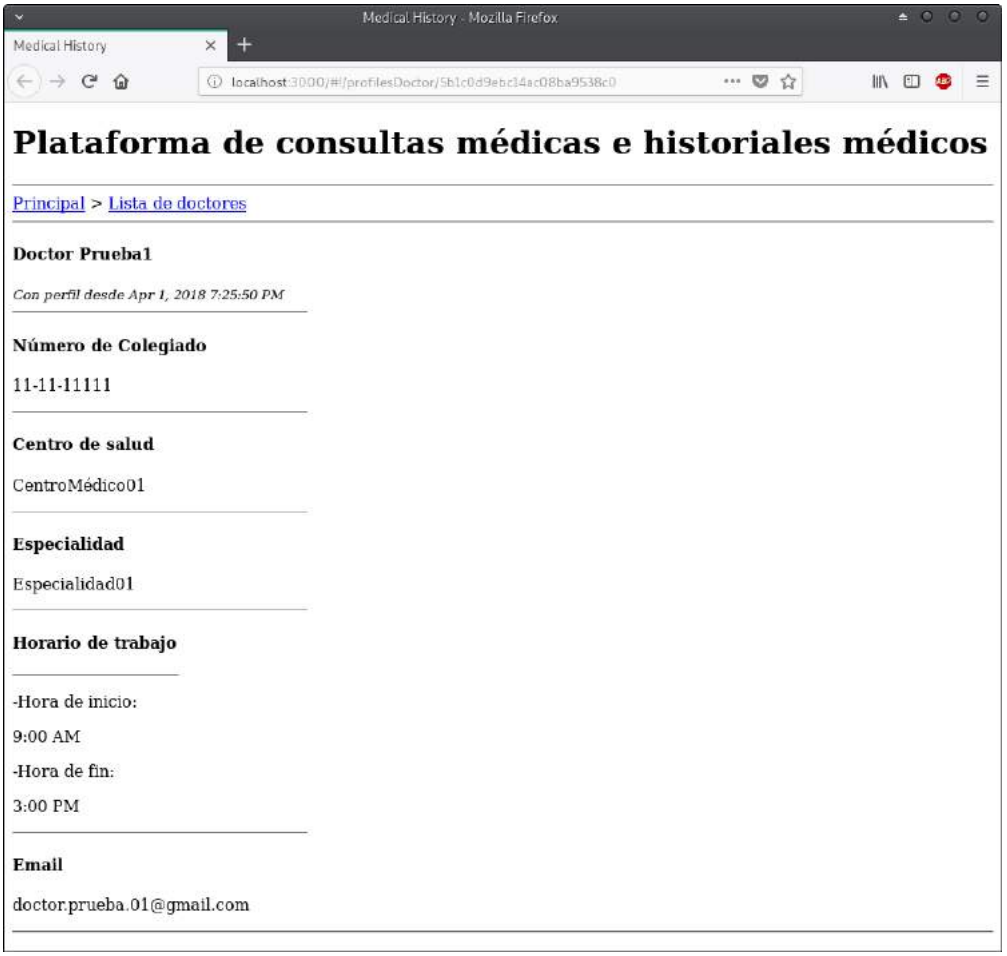

Fig. 4.69. Vista de la página de perfil de un médico.

Pruebas funcionales del sprint 3:

ID	P03_01
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de citas se muestran y funcionan correctamente
Acciones	1. Acceder y probar las vistas de citas ◦ Crear cita. ◦ Editar cita. ◦ Listar citas. ◦ Visualizar cita.

TABLA 4.15. PRUEBA FUNCIONAL 1 DEL TERCER SPRINT.

ID	P03_02
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de doctor se muestran y funcionan correctamente
Acciones	1. Acceder y probar las vistas de doctor ◦ Crear perfil. ◦ Editar perfil. ◦ Listar doctores. ◦ Visualizar doctor.

TABLA 4.16. PRUEBA FUNCIONAL 2 DEL TERCER SPRINT.

Vemos un resumen del tiempo estimado y utilizado en el tercer sprint.

Historia	Tiempo estimado	Tiempo final
T16	8,66666666666667	7
T17	8,66666666666667	8
T18	9,66666666666667	8
T19	11,3333333333333	13
Total	**38,3333333333333**	**36**

TABLA 4.17. RESUMEN DE TIEMPO TERCER SPRINT.

Tarea	Hsitoria de Usuario	Estado Inicial	Estado Estimado	Estado Final
T01	HU12	Completada	Completada	Completada
T02	HU12	Completada	Completada	Completada
T03	HU12	Completada	Completada	Completada
T04	HU12	Completada	Completada	Completada
T05	HU12	Completada	Completada	Completada
T06	HU12	Completada	Completada	Completada
T07	HU12	Completada	Completada	Completada
T08	HU11	Completada	Completada	Completada
T09	HU09	Completada	Completada	Completada
T10	HU11	Completada	Completada	Completada
T11	HU11	Completada	Completada	Completada
T12	HU12	Completada	Completada	Completada
T13	HU11	Completada	Completada	Completada
T14	HU07	Completada	Completada	Completada
T15	HU07	Completada	Completada	Completada
T16	HU01	No iniciada	Completada	Completada
T17	HU01	No iniciada	Completada	Completada
T18	HU09	No iniciada	Completada	Completada
T19	HU09	No iniciada	Completada	Completada
T20	HU09	No iniciada	No iniciada	No iniciada
T21	HU09	No iniciada	No iniciada	No iniciada
T22	HU04	No iniciada	No iniciada	No iniciada
T23	HU04	No iniciada	No iniciada	No iniciada
T24	HU07	No iniciada	No iniciada	No iniciada
T25	HU03	No iniciada	No iniciada	No iniciada
T26	HU01	No iniciada	No iniciada	No iniciada
T27	HU04	No iniciada	No iniciada	No iniciada
T28	HU10	No iniciada	No iniciada	No iniciada
T29	HU10	No iniciada	No iniciada	No iniciada
T30	HU10	No iniciada	No iniciada	No iniciada
T31	HU10	No iniciada	No iniciada	No iniciada
T32	HU10	No iniciada	No iniciada	No iniciada
T33	HU10	No iniciada	No iniciada	No iniciada

TABLA 4.18. RESUMEN DEL TERCER SPRINT.

4.3.6. Cuarto Sprint

Horas máximas	40
Horas utilizadas	38,6666666666667
Horas restantes	1,33333333333332
Tareas	Horas estimadas
T20	9,66666666666667
T21	9,66666666666667
T22	9,66666666666667
T23	9,66666666666667

TABLA 4.19. TAREAS ESCOGIDAS CUARTO SPRINT.

Vamos a realizar los objetivos de la tarea 20.

- Implementación del módulo 'Pacientes' backend:

  Al igual que hicimos con los anteriores módulos, repetiremos la misma implementación tanto en el backend como en el frontend.

  + Implementación del modelo con Schema 'Pacientes'.

    Creamos el fichero profilepatient.server.model.js que contiene el schema de una paciente.

  + Cargar el modelo 'Pacientes' en el middleware mongoose.

    Agregamos el schema al fichero mongoose.

  + Implementación del controlador 'Pacientes'.

    Creamos el fichero profilepatient.server.controller.js que contiene las operaciones CRUD de una paciente.

  + Implementación de las rutas de 'Pacientes'.

    Creamos el fichero profilepatient.server.routes.js que contiene las rutas de una paciente.

  + Cargar las rutas de 'Pacientes' en la aplicación express.

    Agregamos las rutas al fichero express.js.

Vamos a realizar los objetivos de la tarea 21.

- Implementación del módulo 'Pacientes' frontend:

  Creamos el árbol de directorios similar al de los módulos que realizamos anteriormente donde crearemos los siguientes ficheros.

  + Crear el módulo 'Pacientes'.

    Creamos el módulo profilepatient en el fichero profilepatient.client.module.js.

+ Crear el servicio ‘Pacientes’.

  Creamos un servicio de que nos devolverá un recurso profilepatient uńico cuando queramos acceder o modificar un paciente determinado.

+ Crear el controlador ‘Pacientes’.

  Creamos el fichero profilepatient.client.controller.js con las siguiente funciones.

  - Crear Paciente.
  - Buscar todos los Pacientes.
  - Buscar un paciente.
  - Modificar paciente.
  - Eliminar paciente.

+ Crear las rutas de ‘Pacientes’.

  Creamos el fichero profilepatient.client.routes.js con las rutas de las vistas del módulo profilepatient.

+ Crear las vistas de ‘Pacientes’.

  Creamos las siguientes vistas:

  - Crear perfil.

Fig. 4.70. Vista de la página de crear perfil de paciente.

- Editar perfil.

Fig. 4.71. Vista de la página de editar perfil de paciente.

- Visualizar paciente.

Fig. 4.72. Vista de la página de perfil de un paciente.

Vamos a realizar los objetivos de la tarea 22.

- Implementación del módulo ‘Consulta’ backend:

Al igual que hicimos con los anteriores módulos, repetiremos la misma implementación tanto en el backend como en el frontend.

+ Implementación del modelo con Schema 'Consulta'.

  Creamos el fichero consultation.server.model.js que contiene el schema de una consulta.

+ Cargar el modelo 'Consulta' en el middleware mongoose.

  Agregamos el schema al fichero mongoose.

+ Implementación del controlador 'Consulta'.

  Creamos el fichero consultations.server.controller.js que contiene las operaciones CRUD de una consulta.

+ Implementación de las rutas de 'Consulta'.

  Creamos el fichero consultations.server.routes.js que contiene las rutas de una consulta.

+ Cargar las rutas de 'Consulta' en la aplicación express.

  Agregamos las rutas al fichero express.js.

Vamos a realizar los objetivos de la tarea 23.

- Implementación del módulo 'Consulta' frontend:

  Creamos el árbol de directorios similar al de los módulos que realizamos anteriormente donde crearemos los siguientes ficheros.

  + Crear el módulo 'Consultas'.

    Creamos el módulo consultations en el fichero consultations.client.module.js.

  + Crear el servicio 'Consultas'.

    Creamos un servicio de que nos devolverá un recurso consultations uńico cuando queramos acceder o modificar una consulta determinada.

  + Crear el controlador 'Consultas'.

    Creamos el fichero consultations.client.controller.js con las siguiente funciones.

    - Crear consulta.
    - Buscar todas las consultas.
    - Buscar una consulta.
    - Actualizar una consulta.
    - Borrar una consulta.

  + Crear las rutas de 'Consulta'.

    Creamos el fichero consultations.client.routes.js con las rutas de las vistas del módulo consultations.

+ Crear las vistas de ‘Consulta’.

  Creamos las siguientes vistas:

  - Crear consulta.

Fig. 4.73. Vista de la página de crear consulta.

  - Editar consulta.

Fig. 4.74. Vista de la página de editar consulta.

- Listar consultas.

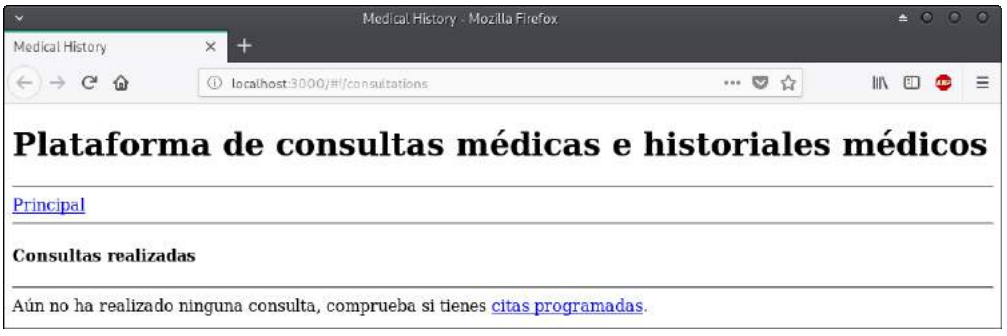

Fig. 4.75. Vista de la página de lista vacia de consultas para un médico.

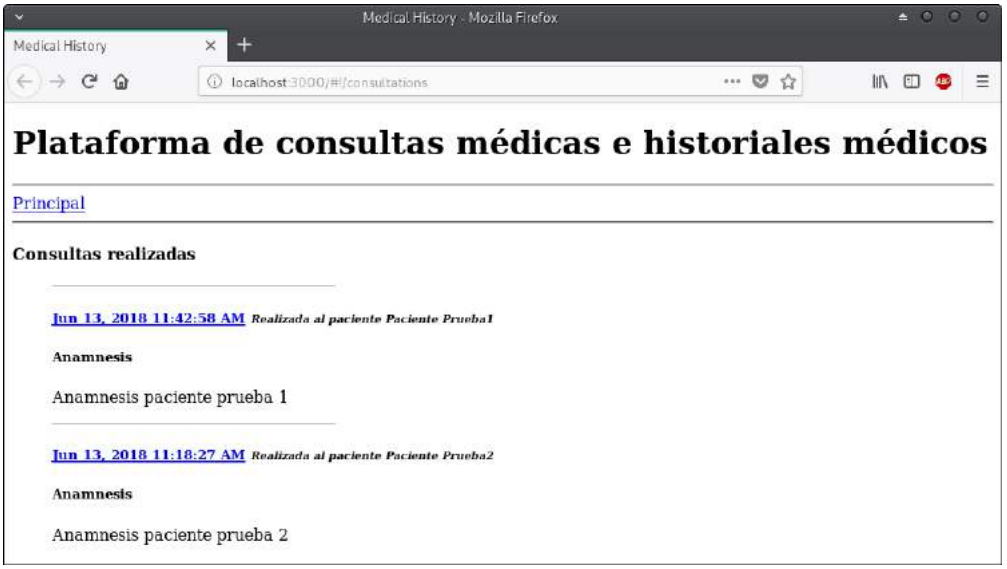

Fig. 4.76. Vista de la página de lista de consultas para un médico.

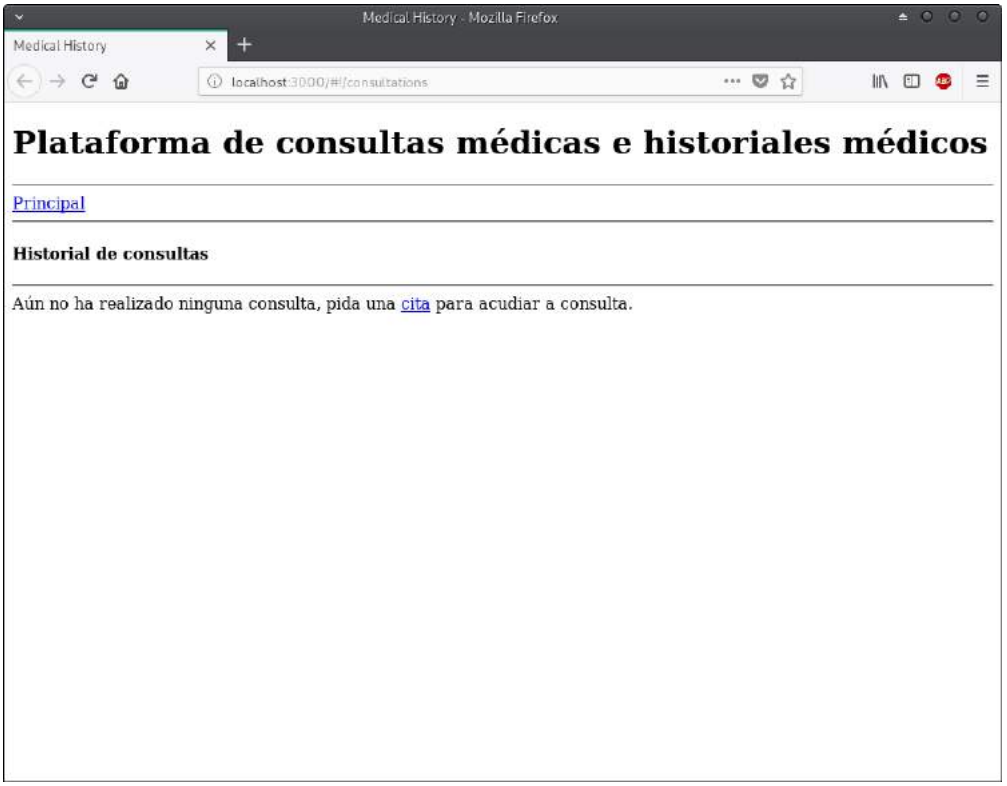

Fig. 4.77. Vista de la página de lista vacia de consultas para un paciente.

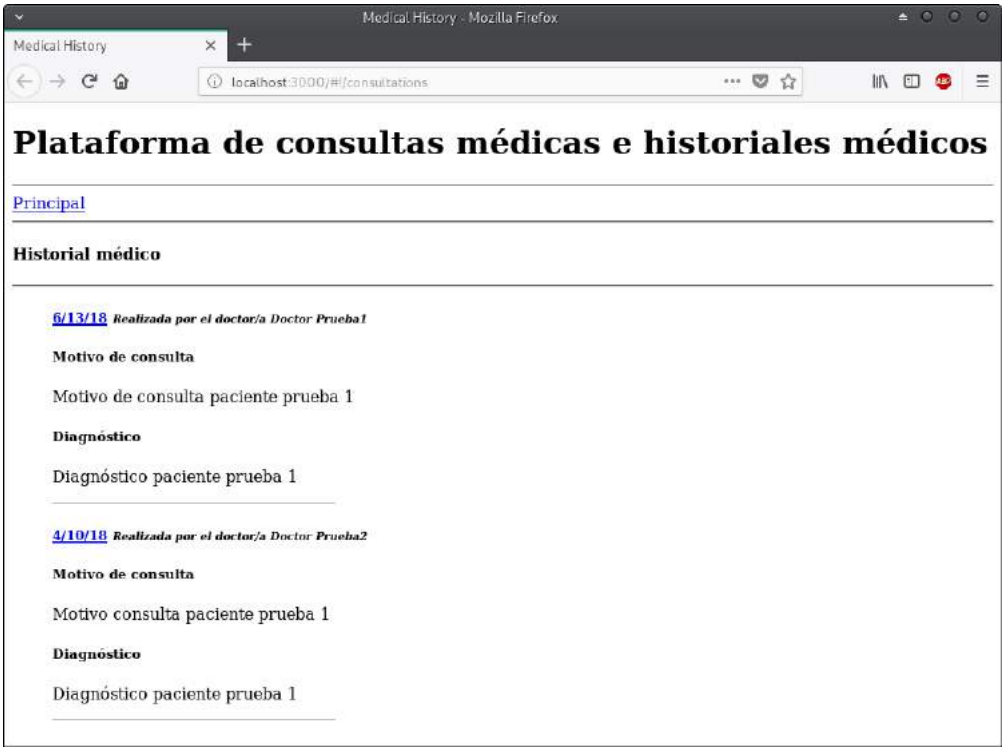

Fig. 4.78. Vista de la página de lista de consultas para un paciente.

- Visualizar un consulta.

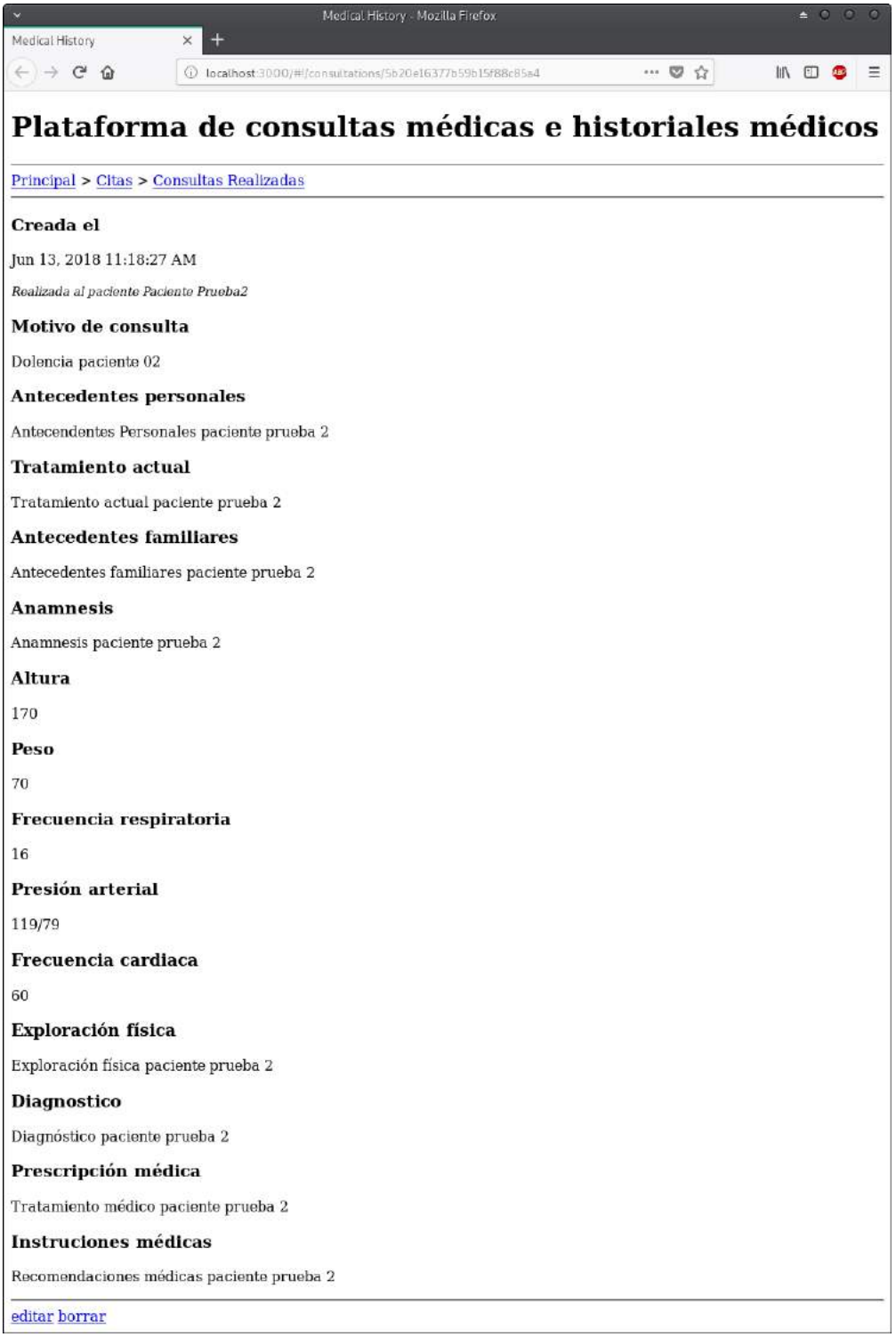

Fig. 4.79. Vista de la página de detalles de una consulta para un médico.

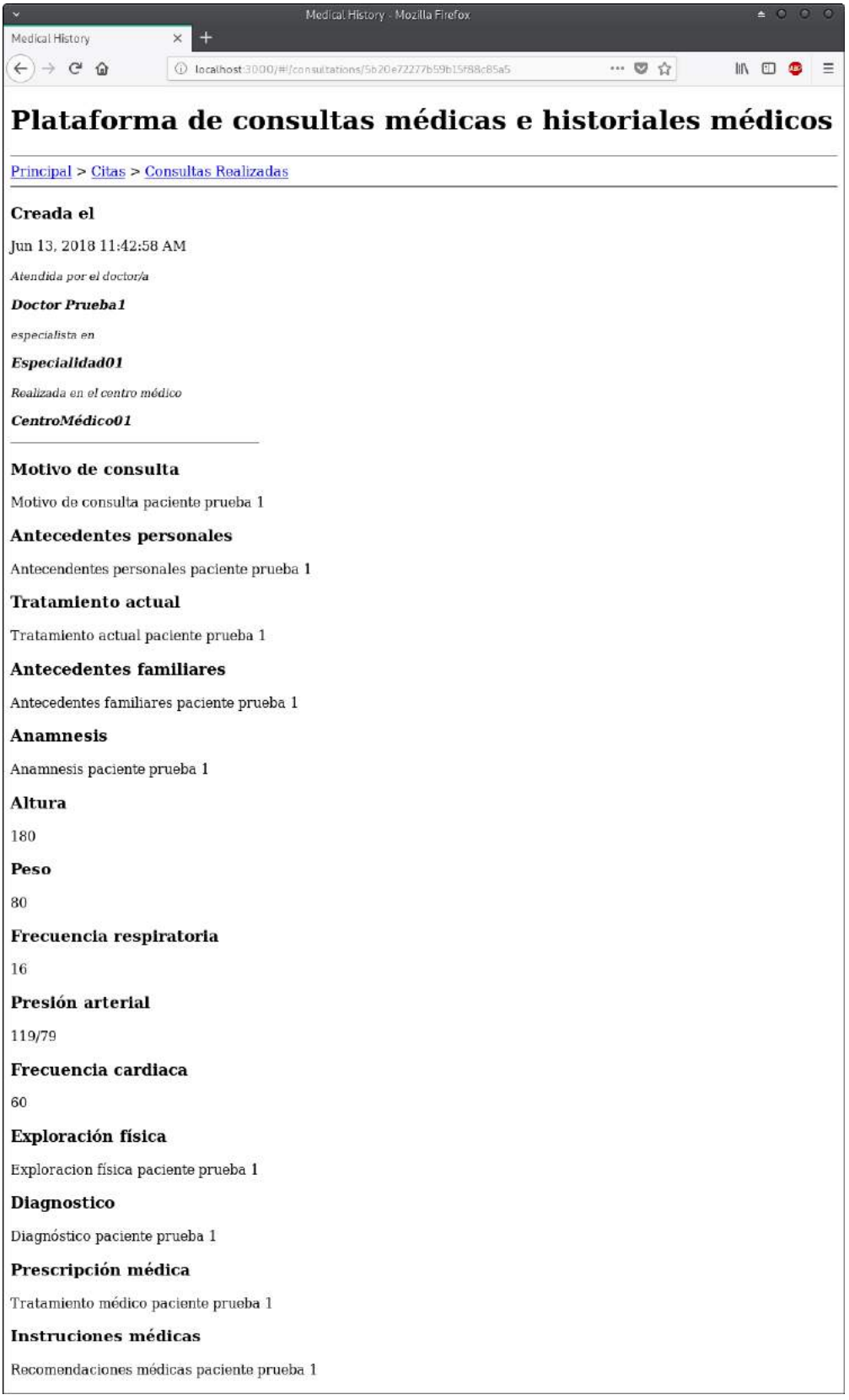

Fig. 4.80. Vista de la página de detalles de una consulta para un paciente.

Pruebas funcionales del sprint 4:

ID	P04_01
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de paciente se muestran y funcionan correctamente
Acciones	1. Acceder y probar las vistas de paciente por un paciente ◦ Crear perfil. ◦ Editar perfil. ◦ Visualizar paciente.

TABLA 4.20. PRUEBA FUNCIONAL 1 DEL CUARTO SPRINT.

ID	P04_02
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de consulta se muestran y funcionan correctamente
Acciones	1. Acceder y probar las vistas de consulta ◦ Crear consulta. ◦ Editar consulta. ◦ Listar consultas. ◦ Visualizar una consulta.

TABLA 4.21. PRUEBA FUNCIONAL 2 DEL CUARTO SPRINT.

Vemos un resumen del tiempo estimado y utilizado en el cuarto sprint.

Historia	Tiempo estimado	Tiempo final
T20	9,66666666666667	8
T21	9,66666666666667	13
T22	9,66666666666667	10
T23	9,66666666666667	13
Total	38,6666666666667	44

TABLA 4.22. RESUMEN DE TIEMPO CUARTO SPRINT.

Tarea	Hsitoria de Usuario	Estado Inicial	Estado Estimado	Estado Final
T01	HU12	Completada	Completada	Completada
T02	HU12	Completada	Completada	Completada
T03	HU12	Completada	Completada	Completada
T04	HU12	Completada	Completada	Completada
T05	HU12	Completada	Completada	Completada
T06	HU12	Completada	Completada	Completada
T07	HU12	Completada	Completada	Completada
T08	HU11	Completada	Completada	Completada
T09	HU09	Completada	Completada	Completada
T10	HU11	Completada	Completada	Completada
T11	HU11	Completada	Completada	Completada
T12	HU12	Completada	Completada	Completada
T13	HU11	Completada	Completada	Completada
T14	HU07	Completada	Completada	Completada
T15	HU07	Completada	Completada	Completada
T16	HU01	Completada	Completada	Completada
T17	HU01	Completada	Completada	Completada
T18	HU09	Completada	Completada	Completada
T19	HU09	Completada	Completada	Completada
T20	HU09	No iniciada	Completada	Completada
T21	HU09	No iniciada	Completada	Completada
T22	HU04	No iniciada	Completada	Completada
T23	HU04	No iniciada	Completada	Completada
T24	HU07	No iniciada	No iniciada	No iniciada
T25	HU03	No iniciada	No iniciada	No iniciada
T26	HU01	No iniciada	No iniciada	No iniciada
T27	HU04	No iniciada	No iniciada	No iniciada
T28	HU10	No iniciada	No iniciada	No iniciada
T29	HU10	No iniciada	No iniciada	No iniciada
T30	HU10	No iniciada	No iniciada	No iniciada
T31	HU10	No iniciada	No iniciada	No iniciada
T32	HU10	No iniciada	No iniciada	No iniciada
T33	HU10	No iniciada	No iniciada	No iniciada

TABLA 4.23. RESUMEN DEL CUARTO SPRINT.

4.3.7. Quinto Sprint

Horas máximas	40
Horas utilizadas	36,6666666666666
Horas restantes	3,3333333333334
Tareas	**Horas estimadas**
T24	7
T25	11,3333333333333
T26	11,3333333333333
T29	7

TABLA 4.24. TAREAS ESCOGIDAS QUINTO SPRINT.

Vamos a realizar los objetivos de la tarea 24.

- Implementación de asignación de 'Articulos' ->'Doctores'.

  1. Modificamos el modelo de artículos donde el campo creador referenciará a un usuario.

```
30      // Creador (médico) del artículo
31      creador: {
32        type: Schema.ObjectId,
33        ref: 'User'
34      }
35    });
```

Fig. 4.81. Campo creador modificado del schema artículo.

  2. Comprobamos que en las rutas de artículos del backend ya dejamos implementado la función de tener autorización en las diferentes rutas.
  3. Mostramos una vista diferente para médicos y pacientes en todas las vistas del frontend (crear, editar, listar y ver), adecuamos las funciones de crear, editar, listar y ver artículo solo para médicos.

     A continuación vemos la vista de creación de artículo similar al resto:

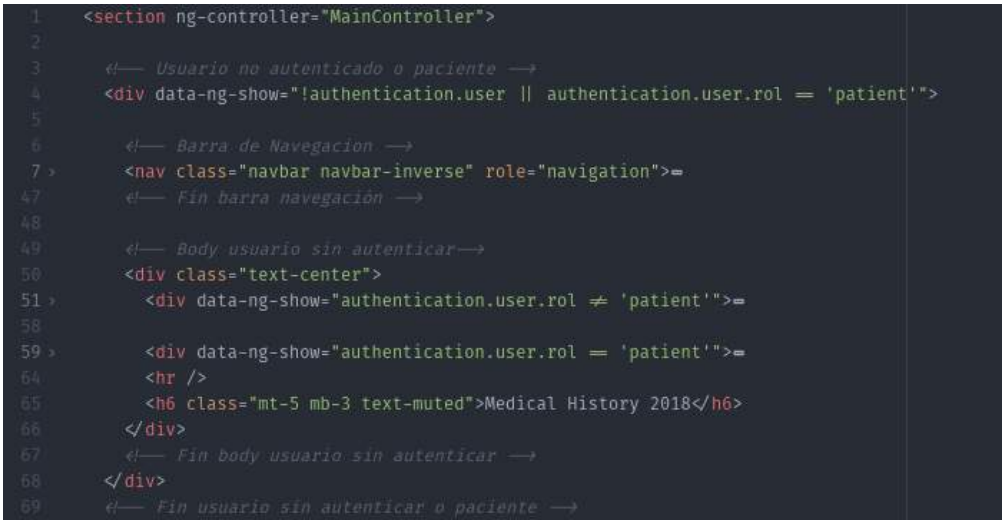

Fig. 4.82. Vista crear artículo: usuario no autenticado o paciente.

Fig. 4.83. Vista crear artículo: usuario autenticado médico.

Vamos a realizar los objetivos de la tarea 25.

- Implementación de asignación de ‘Citas’ ->‘Doctores’.

  1. Modificamos el modelo de citas donde el campo doctor referenciará a un perfil de doctor.

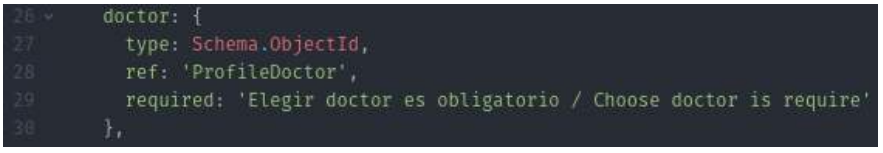

Fig. 4.84. Campo doctor modificado del schema cita.

  2. Comprobamos que en las rutas de citas del backend ya dejamos implementado la función de tener autorización en las diferentes rutas.

3. Mostramos una vista diferente para médicos y pacientes en todas las vistas del frontend (crear, editar, listar y ver), las funciones de crear y editar no están disponibles para médicos. Adecuamos las funciones de listar y ver para médicos.

Vamos a realizar los objetivos de la tarea 26.

- Implementación de asignación de 'Citas' ->'Pacientes'.

1. Modificamos el modelo de citas donde el campo creador referenciará a un perfil de doctor y el creador a un usuario.

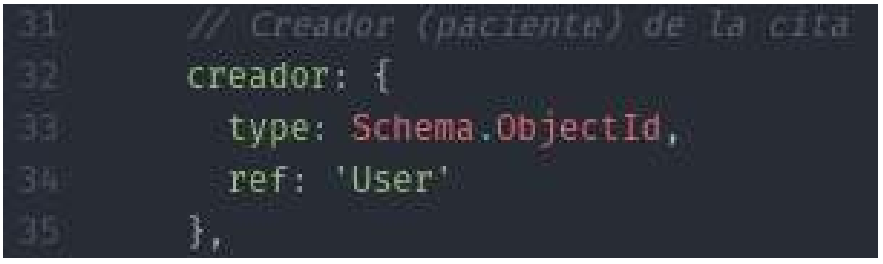

Fig. 4.85. Campo creador modificado del schema cita.

2. Comprobamos que en las rutas de citas del backend ya dejamos implementado la función de tener autorización en las diferentes rutas.
3. Mostramos una vista diferente para médicos y pacientes en todas las vistas del frontend (crear, editar, listar y ver), adecuamos todas las funciones para pacientes.

Vamos a realizar los objetivos de la tarea 29.

- Enriquecer las vista de 'Artículos'.

  Añadimos bootstrap al proyecto y modificamos los estilos de todas las vistas de artículos, creando mensajes de información, de advertencia y de error según convenga en cada caso.

  También creamos contenedores para que las vistas se adapten a la pantalla de los distintos dispositivos según su tamaño.

Pruebas funcionales del sprint 5:

ID	P05_01
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de artículo solo se muestran y funcionan para un usuario médico
Acciones	1. Acceder y probar las vistas de artículo por un paciente ◦ Crear artículo. ◦ Buscar todos los artículos. ◦ Buscar un artículo. ◦ Actualizar un artículo. ◦ Borrar un artículo.

TABLA 4.25. PRUEBA FUNCIONAL 1 DEL QUINTO SPRINT.

ID	P05_02
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de citas se muestran y funcionan para cada usuario según su rol
Acciones	1. Acceder y probar las vistas de citas por un médico ◦ Crear cita. ◦ Editar cita. ◦ Listar citas. ◦ Visualizar cita. 2. Acceder y probar las vistas de citas por un paciente ◦ Listar citas. ◦ Visualizar cita.

TABLA 4.26. PRUEBA FUNCIONAL 2 DEL QUINTO SPRINT.

ID	P05_03
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de artículos se muestran y funcionan correctamente
Acciones	1. Acceder y probar las vistas de artículo por un médico ◦ Crear artículo. ◦ Buscar todos los artículos. ◦ Buscar un artículo. ◦ Actualizar un artículo. ◦ Borrar un artículo.

TABLA 4.27. PRUEBA FUNCIONAL 3 DEL QUINTO SPRINT.

ID	P05_04
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de artículos se adaptan al tamaño de pantalla
Acciones	1. Acceder y probar las vistas de artículo ◦ Pantalla monitor ◦ Pantalla tablet ◦ Pantalla móvil

TABLA 4.28. PRUEBA FUNCIONAL 4 DEL QUINTO SPRINT.

Vemos un resumen del tiempo estimado y utilizado en el quinto sprint.

Historia	Tiempo estimado	Tiempo final
T24	7	8
T25	11,3333333333333	13
T26	11,3333333333333	10
T29	7	5
Total	**36,6666666666666**	**36**

TABLA 4.29. RESUMEN DE TIEMPO QUINTO SPRINT.

Tarea	Hsitoria de Usuario	Estado Inicial	Estado Estimado	Estado Final
T01	HU12	Completada	Completada	Completada
T02	HU12	Completada	Completada	Completada
T03	HU12	Completada	Completada	Completada
T04	HU12	Completada	Completada	Completada
T05	HU12	Completada	Completada	Completada
T06	HU12	Completada	Completada	Completada
T07	HU12	Completada	Completada	Completada
T08	HU11	Completada	Completada	Completada
T09	HU09	Completada	Completada	Completada
T10	HU11	Completada	Completada	Completada
T11	HU11	Completada	Completada	Completada
T12	HU12	Completada	Completada	Completada
T13	HU11	Completada	Completada	Completada
T14	HU07	Completada	Completada	Completada
T15	HU07	Completada	Completada	Completada
T16	HU01	Completada	Completada	Completada
T17	HU01	Completada	Completada	Completada
T18	HU09	Completada	Completada	Completada
T19	HU09	Completada	Completada	Completada
T20	HU09	Completada	Completada	Completada
T21	HU09	Completada	Completada	Completada
T22	HU04	Completada	Completada	Completada
T23	HU04	Completada	Completada	Completada
T24	HU07	No iniciada	Completada	Completada
T25	HU03	No iniciada	Completada	Completada
T26	HU01	No iniciada	Completada	Completada
T27	HU04	No iniciada	No iniciada	No iniciada
T28	HU10	No iniciada	No iniciada	No iniciada
T29	HU10	No iniciada	Completada	Completada
T30	HU10	No iniciada	No iniciada	No iniciada
T31	HU10	No iniciada	No iniciada	No iniciada
T32	HU10	No iniciada	No iniciada	No iniciada
T33	HU10	No iniciada	No iniciada	No iniciada

TABLA 4.30. RESUMEN DEL QUINTO SPRINT.

4.3.8. Sexto Sprint

Horas máximas	40
Horas utilizadas	34,6666666666667
Horas restantes	5,33333333333333
Tareas	**Horas estimadas**
T27	13,6666666666667
T28	9,66666666666667
T30	11,3333333333333

TABLA 4.31. TAREAS ESCOGIDAS SEXTO SPRINT.

Vamos a realizar los objetivos de la tarea 27.

- Implementación de asignación de 'Consulta' ->'Citas'.

  1. Modificamos el modelo de consulta donde el campo cita referenciará a una cita desde la que un doctor la crea al esta ser atendida.

```
12        // Cita a la que es referida la consulta
13      appointment:{
14        type: Schema.ObjectId,
15        ref: 'Appointment'
16      },
```

Fig. 4.86. Campo cita modificado del schema consulta.

  2. Comprobamos que en las rutas de consulta del backend ya dejamos implementado la función de tener autorización en las diferentes rutas.
  3. Mostramos una vista diferente para médicos y pacientes en todas las vistas del frontend (crear, editar, listar y ver), adecuamos todas las funciones para pacientes.

Vamos a realizar los objetivos de la tarea 28.

- Enriquecer la vista de 'Principal'.

  Modificamos los estilos de la vista principal, creando mensajes de información, de advertencia y de error según convenga en cada caso.

  También creamos contenedores para que las vistas se adapten a la pantalla de los distintos dispositivos según su tamaño.

Vamos a realizar los objetivos de la tarea 30.

- Enriquecer las vista de 'Citas'.

Modificamos los estilos de todas las vistas de citas, creando mensajes de información, de advertencia y de error según convenga en cada caso.

También creamos contenedores para que las vistas se adapten a la pantalla de los distintos dispositivos según su tamaño.

Pruebas funcionales del sprint 6:

ID	P06_01
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de consultas se muestran y funcionan para cada usuario según su rol
Acciones	1. Acceder y probar las vistas de consultas por un médico ◦ Crear consulta. ◦ Editar consulta. ◦ Listar consulta. ◦ Visualizar consulta. 2. Acceder y probar las vistas de consulta por un paciente ◦ Listar consulta. ◦ Visualizar consulta.

TABLA 4.32. PRUEBA FUNCIONAL 1 DEL SEXTO SPRINT.

ID	P06_02
Resultado	Válido
Sistema	Cliente / servidor
Descripción	La vista de la página principal se muestra y funciona correctamente
Acciones	1. Acceder y probar la vista principal de la aplicación para: ◦ Usuarios no autenticados ◦ Usuarios autenticados ◦ Usuarios médicos ◦ Usuarios pacientes

TABLA 4.33. PRUEBA FUNCIONAL 2 DEL SEXTO SPRINT.

ID	P06_03
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de citas se muestran y funcionan para cada usuario según su rol
Acciones	1. Acceder y probar las vistas de citas por un médico ◦ Crear cita. ◦ Editar cita. ◦ Listar citas. ◦ Visualizar cita. 2. Acceder y probar las vistas de citas por un paciente ◦ Listar citas. ◦ Visualizar cita.

TABLA 4.34. PRUEBA FUNCIONAL 3 DEL SEXTO SPRINT.

ID	P06_04
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de citas se adaptan al tamaño de pantalla
Acciones	1. Acceder y probar las vistas de cita ◦ Pantalla monitor ◦ Pantalla tablet ◦ Pantalla móvil

TABLA 4.35. PRUEBA FUNCIONAL 4 DEL SEXTO SPRINT.

ID	P06_05
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vista de principal se adapta al tamaño de pantalla
Acciones	1. Acceder y probar la vista principal ◦ Pantalla monitor ◦ Pantalla tablet ◦ Pantalla móvil

TABLA 4.36. PRUEBA FUNCIONAL 5 DEL SEXTO SPRINT.

Vemos un resumen del tiempo estimado y utilizado en el sexto sprint.

Historia	Tiempo estimado	Tiempo final
T27	13,6666666666667	15
T28	9,66666666666667	7
T30	11,3333333333333	13
Total	**34,6666666666667**	**35**

TABLA 4.37. RESUMEN DE TIEMPO SEXTO SPRINT.

Tarea	Hsitoria de Usuario	Estado Inicial	Estado Estimado	Estado Final
T01	HU12	Completada	Completada	Completada
T02	HU12	Completada	Completada	Completada
T03	HU12	Completada	Completada	Completada
T04	HU12	Completada	Completada	Completada
T05	HU12	Completada	Completada	Completada
T06	HU12	Completada	Completada	Completada
T07	HU12	Completada	Completada	Completada
T08	HU11	Completada	Completada	Completada
T09	HU09	Completada	Completada	Completada
T10	HU11	Completada	Completada	Completada
T11	HU11	Completada	Completada	Completada
T12	HU12	Completada	Completada	Completada
T13	HU11	Completada	Completada	Completada
T14	HU07	Completada	Completada	Completada
T15	HU07	Completada	Completada	Completada
T16	HU01	Completada	Completada	Completada
T17	HU01	Completada	Completada	Completada
T18	HU09	Completada	Completada	Completada
T19	HU09	Completada	Completada	Completada
T20	HU09	Completada	Completada	Completada
T21	HU09	Completada	Completada	Completada
T22	HU04	Completada	Completada	Completada
T23	HU04	Completada	Completada	Completada
T24	HU07	Completada	Completada	Completada
T25	HU03	Completada	Completada	Completada
T26	HU01	Completada	Completada	Completada
T27	HU04	No iniciada	Completada	Completada
T28	HU10	No iniciada	Completada	Completada
T29	HU10	Completada	Completada	Completada
T30	HU10	No iniciada	Completada	Completada
T31	HU10	No iniciada	No iniciada	No iniciada
T32	HU10	No iniciada	No iniciada	No iniciada
T33	HU10	No iniciada	No iniciada	No iniciada

TABLA 4.38. RESUMEN DEL SEXTO SPRINT.

4.3.9. Septimo Sprint

Horas máximas	40
Horas utilizadas	33,9999999999999
Horas restantes	6,0000000000001
Tareas	Horas estimadas
T31	11,3333333333333
T32	11,3333333333333
T33	11,3333333333333

TABLA 4.39. TAREAS ESCOGIDAS SÉPTIMO SPRINT.

Vamos a realizar los objetivos de la tarea 31.

- Enriquecer las vista de ‘Doctores’.

  Modificamos los estilos de todas las vistas de doctor, creando mensajes de información, de advertencia y de error según convenga en cada caso.

  También creamos contenedores para que las vistas se adapten a la pantalla de los distintos dispositivos según su tamaño.

Vamos a realizar los objetivos de la tarea 32.

- Enriquecer las vistas de ‘Pacientes’.

  Modificamos los estilos de todas las vistas de paciente, creando mensajes de información, de advertencia y de error según convenga en cada caso.

  También creamos contenedores para que las vistas se adapten a la pantalla de los distintos dispositivos según su tamaño.

Vamos a realizar los objetivos de la tarea 33.

- Enriquecer las vistas de ‘Consultas’.

  Modificamos los estilos de todas las vistas de consulta, creando mensajes de información, de advertencia y de error según convenga en cada caso.

  También creamos contenedores para que las vistas se adapten a la pantalla de los distintos dispositivos según su tamaño.

Pruebas funcionales del sprint 7:

ID	P07_01
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de doctor se muestran y funcionan para cada usuario según su rol
Acciones	1. Acceder y probar las vistas de doctor por un médico ◦ Crear perfil. ◦ Editar perfil. ◦ Listar doctores. ◦ Visualizar doctor. 2. Acceder y probar las vistas de doctor por un paciente ◦ Listar doctores. ◦ Visualizar doctor.

TABLA 4.40. PRUEBA FUNCIONAL 1 DEL SÉPTIMO SPRINT.

ID	P07_02
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de paciente se muestran y funcionan para cada usuario según su rol
Acciones	1. Acceder y probar las vistas de paciente por un paciente ◦ Crear perfil. ◦ Editar perfil. ◦ Visualizar paciente. 2. Acceder y probar las vistas de paciente por un médico ◦ Visualizar paciente.

TABLA 4.41. PRUEBA FUNCIONAL 2 DEL SÉPTIMO SPRINT.

ID	P07_03
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de consultas se muestran y funcionan para cada usuario según su rol
Acciones	1. Acceder y probar las vistas de consultas por un médico ◦ Crear consulta. ◦ Editar consulta. ◦ Listar consulta. ◦ Visualizar consulta. 2. Acceder y probar las vistas de consulta por un paciente ◦ Listar consulta. ◦ Visualizar consulta.

TABLA 4.42. PRUEBA FUNCIONAL 3 DEL SÉPTIMO SPRINT.

ID	P07_04
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de consultas se adaptan al tamaño de pantalla
Acciones	1. Acceder y probar las vistas de consulta ◦ Pantalla monitor ◦ Pantalla tablet ◦ Pantalla móvil

TABLA 4.43. PRUEBA FUNCIONAL 4 DEL SÉPTIMO SPRINT.

ID	P07_05
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de pacientes se adaptan al tamaño de pantalla
Acciones	1. Acceder y probar las vistas de paciente ◦ Pantalla monitor ◦ Pantalla tablet ◦ Pantalla móvil

TABLA 4.44. PRUEBA FUNCIONAL 5 DEL SÉPTIMO SPRINT.

ID	P07_06
Resultado	Válido
Sistema	Cliente / servidor
Descripción	Las vistas de doctor se adaptan al tamaño de pantalla
Acciones	1. Acceder y probar las vistas de doctor ◦ Pantalla monitor ◦ Pantalla tablet ◦ Pantalla móvil

TABLA 4.45. PRUEBA FUNCIONAL 6 DEL SÉPTIMO SPRINT.

Vemos un resumen del tiempo estimado y utilizado en el séptimo y último sprint.

Historia	Tiempo estimado	Tiempo final
T31	11,3333333333333	15
T32	11,3333333333333	7
T33	11,3333333333333	13
Total	**33,9999999999999**	**35**

TABLA 4.46. RESUMEN DE TIEMPO SÉPTIMO SPRINT.

Tarea	Hsitoria de Usuario	Estado Inicial	Estado Estimado	Estado Final
T01	HU12	Completada	Completada	Completada
T02	HU12	Completada	Completada	Completada
T03	HU12	Completada	Completada	Completada
T04	HU12	Completada	Completada	Completada
T05	HU12	Completada	Completada	Completada
T06	HU12	Completada	Completada	Completada
T07	HU12	Completada	Completada	Completada
T08	HU11	Completada	Completada	Completada
T09	HU09	Completada	Completada	Completada
T10	HU11	Completada	Completada	Completada
T11	HU11	Completada	Completada	Completada
T12	HU12	Completada	Completada	Completada
T13	HU11	Completada	Completada	Completada
T14	HU07	Completada	Completada	Completada
T15	HU07	Completada	Completada	Completada
T16	HU01	Completada	Completada	Completada
T17	HU01	Completada	Completada	Completada
T18	HU09	Completada	Completada	Completada
T19	HU09	Completada	Completada	Completada
T20	HU09	Completada	Completada	Completada
T21	HU09	Completada	Completada	Completada
T22	HU04	Completada	Completada	Completada
T23	HU04	Completada	Completada	Completada
T24	HU07	Completada	Completada	Completada
T25	HU03	Completada	Completada	Completada
T26	HU01	Completada	Completada	Completada
T27	HU04	Completada	Completada	Completada
T28	HU10	Completada	Completada	Completada
T29	HU10	Completada	Completada	Completada
T30	HU10	Completada	Completada	Completada
T31	HU10	No iniciada	Completada	Completada
T32	HU10	No iniciada	Completada	Completada
T33	HU10	No iniciada	Completada	Completada

TABLA 4.47. RESUMEN DEL SÉPTIMO SPRINT.

4.4. Planificación y presupuesto

4.4.1. Planificación

En este capítulo se detallarán lo relativo a la planificación y los costes económicos de la realización de este proyecto.

La planificación del desarrollo de los hitos del proyecto se muestra en el siguiente diagrama de Gantt.

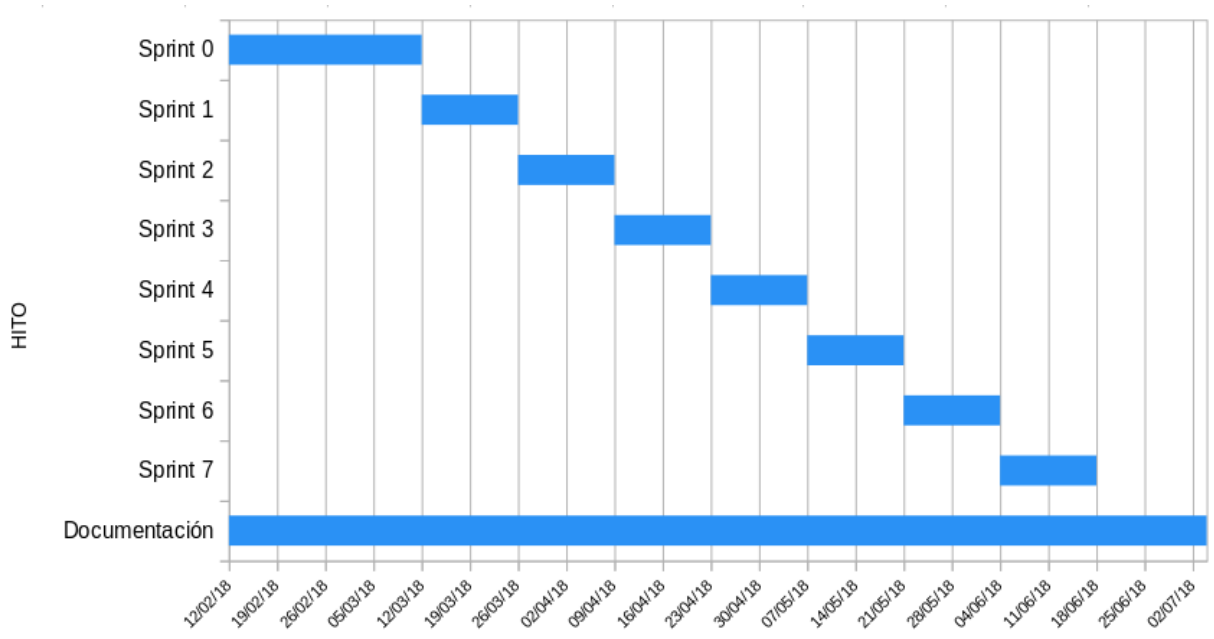

Fig. 4.87. Diagrama de Gantt.

4.4.2. Presupuesto

En este apartado se detalla el presupuesto que se estima para llevar a cabo el proyecto.
Se desglosarán los costes en función de su naturaleza.

Resumen de tiempo dedicado

Según las estimaciones de esfuerzo mostradas a lo largo de este documento se pueden extraer el número de horas dedicadas a cada parte del proyecto. Se utilizarán las tareas de usuario como principal referencia para dividir el proyecto en partes del desarrollo, además se le añade una fase adicional correspondiente a la formación y análisis previo y otra fase final correspondiente a la documentación.

Tarea	Tiempo estimado	Tiempo invertido
Formación y análisis	-	35
T01	2,3	5
T02	1,0	2
T03	1,0	1
T04	3,7	5
T05	1,0	4
T06	3,7	2
T07	1,7	0,5
T08	1,7	2
T09	2,3	2
T10	9,7	10
T11	6,0	7
T12	1,0	0,2
T13	8,7	6
T14	9,7	8
T15	13,7	15
T16	8,7	7
T17	8,7	8
T18	9,7	8
T19	11,3	13
T20	9,7	8
T21	9,7	13
T22	9,7	10
T23	9,7	13
T24	7,0	8
T25	11,3	13
T26	11,3	10
T27	13,7	15
T28	9,7	7
T29	7,0	5
T30	11,3	13
T31	11,3	15
T32	11,3	7
T33	11,3	13
Documentación	-	25
Total tareas	**249,3**	**255,7**
	Total	**315,7**

Fig. 4.88. Resumen tiempo dedicado.

Coste de personal

Para calcular los costes relativos al personal se deben tener en cuenta los roles mencionados anteriormente que formarían el equipo y calculamos las horas utilizadas por cada uno.

Empleado	Rol	Horas dedicadas	Coste (empleado/hora)	Coste
Empleado Nº1	Product Owner	43	75	3225
Empleado Nº2	Scrum Master	86	50	4300
Empleado Nº3	Team Member	172	50	8600
			Total	16125

Fig. 4.89. Resumen coste de personal.

Coste de Hardware

Para el desarrollo del sistema a sido necesario contar con un ordenador para el equipo, por lo que se ha adquirido uno.

Elemento	Coste	% Uso dedicado al proyecto	Dedicación (meses)	Periodo de deprecación (meses)	Coste imputable
Xiaomi Mi Notebook Air Pro 15.6 pulgadas Laptop Intel Core i7 CPU NVIDIA GeForce 16GB RAM 256GB SSD	1.474,38 €	100 %	4	36	163,82 €

Fig. 4.90. Resumen coste de hardware.

Para el cálculo de coste imputable se ha dividido el número de meses que se empleará el equipo entre el número de meses de vida útil que tiene el equipo multiplicado por el uso dedicado al proyecto y el coste de adquisición del equipo.

Coste de Software

En este apartado se detalla el coste del software utilizado para el desarrollo del proyecto.

Elemento	Coste
Manjaro Linux	0
MEAN Stack	0
Apache Open Office Calc	0
TexMaker	0
Postman	0
Visual Studio Professional 2017	641
Total	641,00 €

Fig. 4.91. Resumen coste de software.

Resumen de costes

Finalmente tras analizar los costes por partes del proyecto, sumaremos todos ellos para ver el coste total del proyecto. Después se ha de añadir una previsión de fondos como costes indirectos para cubrir los posibles riesgos del proyecto y gastos imprevistos no indicados en el presupuesto. Estos costes se establecen en un 10 % del total.

Concepto	Coste
Personal	16.125,00 €
Hardware	163,82 €
Software	641,00 €
Costes indirectos(10%)	1.692,98 €
Total	18.622,80 €

Fig. 4.92. Resumen coste de proyecto.

5. CONCLUSIONES

En este apartado se va a analizar los resultados obtenidos en el desarrollo del proyecto, así como lo experimentado durante el transcurso de este en relación a la metodología empleada.

Respecto a los objetivos iniciales del proyecto se ha cumplido lo que se buscaba en esencia de este trabajo:

- Utilización de una metodología ágil de desarrollo: se ha empleado con éxito la metodología Scrum, por lo que se cumple con este punto.

- Desarrollo de una aplicación utilizable por el usuario medio: en el diseño de las interfaces se ha buscado que fuese todo lo más simple posible, facilitando al usuario sin experiencia poder utilizar todas las funcionalidades que el sistema le ofrece.

- Modularidad: tal como se detallará en el apartado dedicado a trabajos futuros, la aplicación está ideada para recibir actualizaciones con nuevos módulos que implementes nuevas funcionalidades.

- Persistencia de la información: se ha incluido una base de datos donde la información se almacena entre sesiones y dispositivos, así todos los registros almacenados en el sistema son persistentes.

Finalmente se puede concluir que los objetivos marcados inicialmente para el proyecto se han cumplido, además de los requisitos hablados con los clientes recogidos en las historias de usuario.

A continuación se detalla los resultados del esfuerzo de dedicado real en comparación con el esfuerzo que se ha ido estimando durante el proyecto:

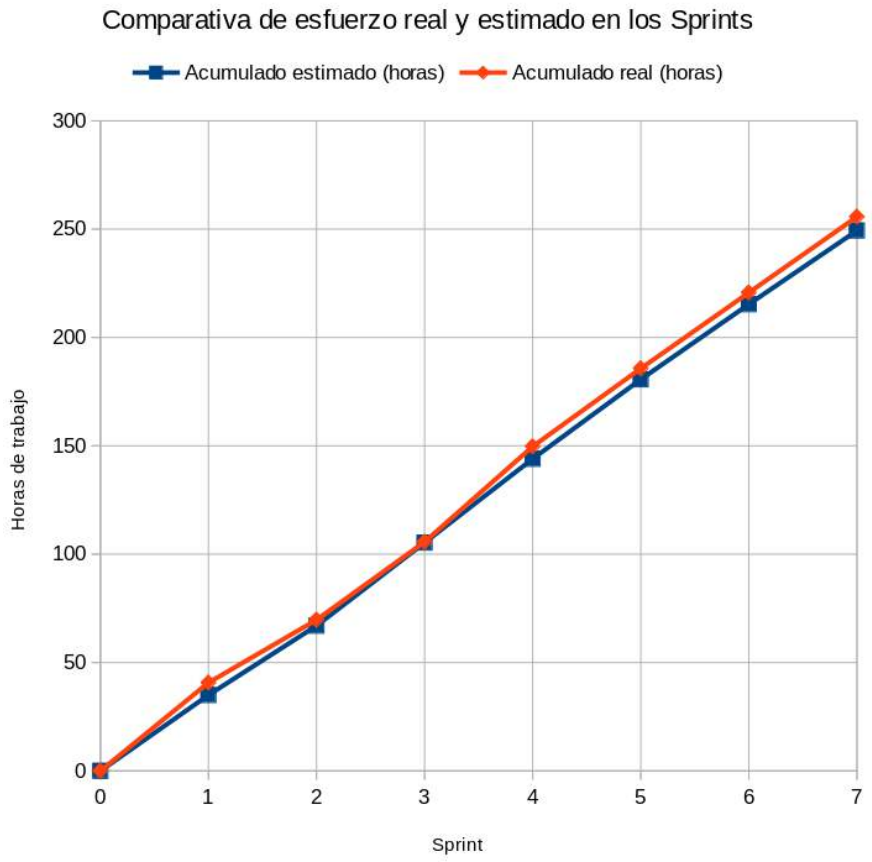

Fig. 5.1. Gráfica comparativa del esfuerzo real y estimada en los sprints.

En el gráfico anterior se puede observar como el rendimiento real ha sido superior al estimado, aunque en general ha sido bastante aproximado. Esto puede deberse a la propiedad adaptativa de la metodología, ya que al permitir reestimaciones en cada iteración, nos da la posibilidad de utilizar la experiencia actual en el mismo proyecto para afinar la estimación más que si hubiese que quedarse con los valores previos al inicio del proyecto. Especialmente beneficioso en casos en los que la experiencia del equipo en proyectos similares sea baja y por tanto la estimación tenga mayor margen de error.

Se ha cumplido con lo acordado con el cliente en las historias de usuario teniendo al final de cada sprint una versión de producto funcional.

5.1. Desarrollos futuros

Existe un amplio abanico de posibilidades de desarrollo futuro para este proyecto por la gran cantidad de funcionalidades a complementar que ofrece por el ámbito social en el que se desarrolla.

- Módulo administración.
- Módulo de gestión.

- Módulo orientado a la hospitalización.
- Módulo orientado a pruebas médicas.
- Módulo orientado a la historia biológica de pacientes (Árboles genealógicos).
- Módulo orientado a la investigación.
- Módulo telemático de comunicación (video-consulta).
- Módulo de urgencias (Aplicación movil).
- Módulo de especialización médica.
- Seguimiento terapéutico de pacientes.
- Módulo colaborativo de profesionales (salas de video-conferencia).

5.2. Conclusiones personales

Como experiencia personal, la realización de este Trabajo Fin de Grado ha sido una puesta en escena de como un proyecto se desarrollaría en un equipo para conseguir un producto utilizando una metodología de desarrollo ágil (SCRUM).

Por otra parte, también quería realizar este proyecto en una tecnología no utilizada previamente (MEAN Stack) en el transcurso de las asignaturas, pero utilizando las herramientas, conocimientos y formas de trabajo que he ido aprendiendo en la realización de estas.

En definitiva, la experiencia de este TFG ha sido muy positiva, me ha permitido acercarme de una forma más global a un proyecto en todas sus fases.

El desarrollo de este trabajo lo considero una experiencia útil y de ayuda al crecimiento personal y profesional.

BIBLIOGRAFÍA

[1] [Último acceso: 18/06/18] Carlos Hernández Salvador. (2003)
Estándares para la historia clínica electrónica, Capítulo VII.
http://www.conganat.org/seis/informes/2003/PDF/CAPITULO7.pdf

[2] [Último acceso: 18/06/18] Fernando Sánchez Durán. (2016) *Estándares actuales de la HCE*.
https://www.ramonramon.org/blog/2016/01/14/estandares-actuales-de-la-hce-historia-clinica-electronica/

[3] [Último acceso: 18/06/18] HL7.org. (2011) *Estándar FHIR*.
https://www.hl7.org/fhir/index.html

[4] [Último acceso: 18/06/18] Ken Schwaber y Jeff Sutherland (2017) *Guía SCRUM*.
https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Spanish-European.pdf

[5] [Último acceso: 18/06/18] MongoDB, Inc. (2009) *Sitio web oficial MongoDB*.
https://www.mongodb.com/

[6] [Último acceso: 18/06/18] Node.js Foundation (2010) *Sitio web oficial ExpressJS*.
http://expressjs.com/

[7] [Último acceso: 18/06/18] MIT (2010) *Sitio web oficial AngularJS*. https://angularjs.org/

[8] [Último acceso: 18/06/18] Comunidad de desarrolladores AngularJS (2014) *Curso AngularJS*.
https://cursoangularjs.es/

[9] [Último acceso: 18/06/18] Node.js Foundation (2009) *Sitio web oficial NodeJS*.
https://nodejs.org/

[10] [Último acceso: 18/06/18] Isaac Z. Schlueter, Laurie Voss, CJ Silverio. (2014) *Sitio web oficial NPM*.
https://www.npmjs.com/

[11] [Último acceso: 18/06/18] Matthew Eernisse (2012) *Sitio web EJS*. http://ejs.co/

[12] [Último acceso: 18/06/18] MIT (2011) *Sitio web oficial MongooseEJS*.
http://mongoosejs.com/

[13] [Último acceso: 18/06/18] Auth0 Inc. (2013) *Sitio web Passport*. http://www.passportjs.org/

[14] [Último acceso: 18/06/18] Twitter's open source effort (2012) *Sitio web Bower*.
https://bower.io/

[15] [Último acceso: 18/06/18] Postdot Technologies, Inc (2012) *Sitio web oficial Postman*. https://www.getpostman.com/

[16] [Último acceso: 18/06/18] Comunnity GitHub (2007) *Sitio web oficial GitHub*. https://github.com/

[17] [Último acceso: 29/06/18] Comunnity LaTeX (2009) *Sitio web oficial LaTeX*. https://www.latex-project.org/

[18] Amos Q. Haviv, *MEAN Web Development*, Packt Publishing Ltd., p.456, 2014.

[19] Vuk sanovic, Irena Petrijevcanin, and Bojan Sudarevic., *Use of web application frameworks in the development of small applications.*, MIPRO, 2011 Proceedings of the 34th International Convention. IEEE, 2011.

[20] Chao, Joseph, Kevin Parker, and Bill Davey. *Navigating the framework jungle for teaching web application development.* Issues in Informing Science and Information Technology, 10, p.95- 109,2013.

[21] Ihrig, Colin J., and Adam Bretz. *Full Stack Javascript Development with MEAN*. SitePoint, p.297, 2014.

[22] Cantelon, Mike, et al. *Node. js in Action*, Manning, p.396, 2014.

[23] Dayley, Brad. *Node.js, MongoDB, and AngularJS Web Development.* Addison-Wesley Professional, p.609, 2014.

[24] [Último acceso: 23/06/18] Erik DeBill (2011) *Module Counts*. http://modulecounts.com/

6. ANEXO A: MANUAL DE USUARIO

En este manual de usuario se explicará el manejo de la aplicación desarrollada, para que un usuario inexperto pueda desenvolverse sin problemas en las diferentes interfaces de la aplicación.

1. Registro de usuario

La primera página que vemos al entrar en Medical History es la de identificación o registro.

Fig. 6.1. Página principal de Medical History.

En ella vemos dos botones, para registrarnos pulsamos sobre el botón 'Registrarse'.

1.1. Registro con proveedor local

Si queremos realizar un registro con el proveedor local de Medical History rellenamos los campos que vemos en la siguiente pantalla.

Fig. 6.2. Página registro de usuario proveedor local.

- Nombre: Nombre de usuario.
- Apellido: Apellido/s de usuario.
- Email: Dirección de correo electrónico de contacto.
- Usuario: Nombre de usuario con el que se accederá a Medical History.
- Contraseña: Contraseña con el que se accederá a Medical History.

Para finalizar el registro pulsamos sobre el botón 'Enviar'.

1.2. Registro con proveedor Google

Si queremos realizar un registro con el proveedor Google, vemos en la pantalla anterior (Fig.6.2) el botón 'Registrarse con Google', donde pulsaremos y se nos mostrarán las siguientes pantallas.

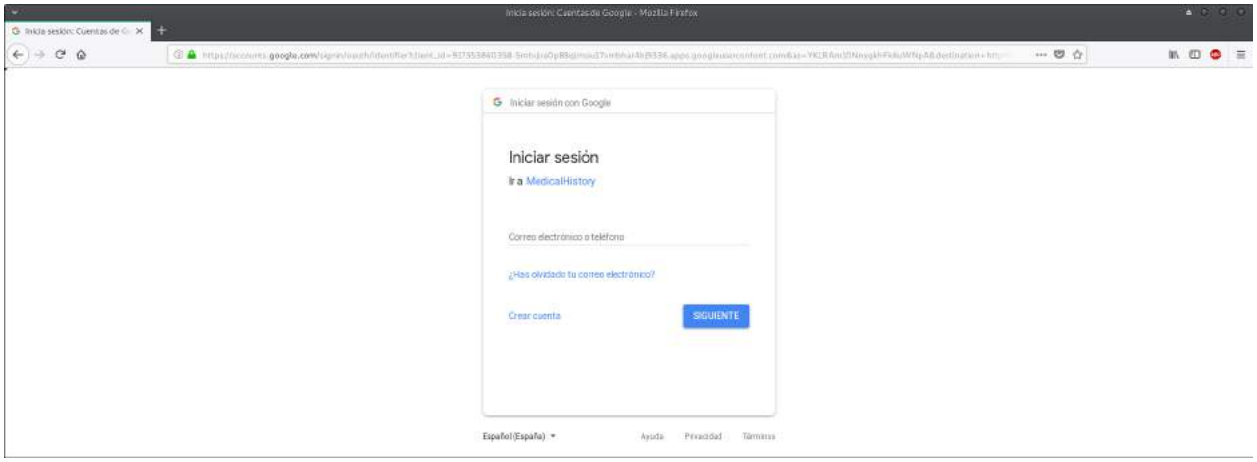

Fig. 6.3. Página registro de usuario proveedor Google: Usuario.

Introduciremos nuestro correo electrónico de la cuenta gmail con la que queremos registrarnos (si no tenemos cuenta puedes crearla pulsando sobre 'Crear cuenta') y pulsaremos sobre el botón 'SIGUIENTE'.

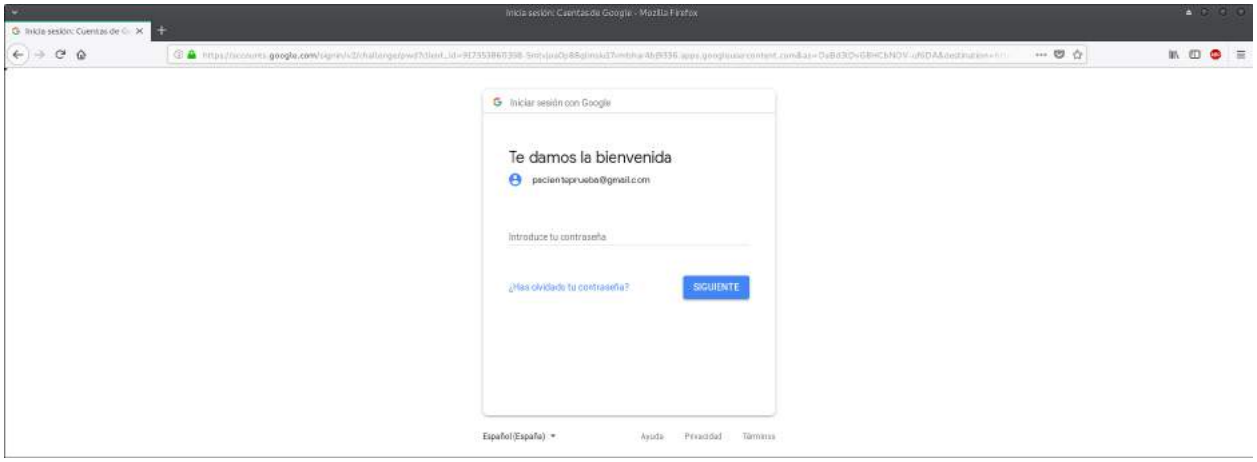

Fig. 6.4. Página registro de usuario proveedor Google: Contraseña.

Introduciremos la contraseña de la cuenta gmail con la que estas registrandote y pulsaremos sobre el botón 'SIGUIENTE'.

Tras habernos registrado con alguno de los proveedores de registro se nos redireccionará automaticamente a la página de elección de perfil ya como usuario registrado e identificado, mostrándonos la pantalla que vemos en el apartado 'Usuario identificado' (Fig.6.8).

2. Identificación de usuario

Al acceder a Medical History nos mostrará de nuevo la página vista en Fig.6.1. y una vez que ya estamos registrados, pulsaremos el botón 'Iniciar sesión' para acceder a Medical History, mostrandonos la siguiente pantalla.

Fig. 6.5. Página identificación de usuario.

* A la hora de identificarnos, lo deberemos de hacer con el mismo proveedor con el que realizamos el registro.

1.1. Identificación con proveedor local

Para identificarnos tan solo debemos rellenar los campos de 'usuario' y 'contraseña' con los que nos registramos que nos muestra la pantalla anterior (Fig.6.5).

1.2. Identificación con proveedor Google

Para identificarnos con Google, en la pantalla anterior (Fig.6.5), pulsamos sobre el botón 'Iniciar sesión con Google', donde se nos mostrará las siguientes pantallas:

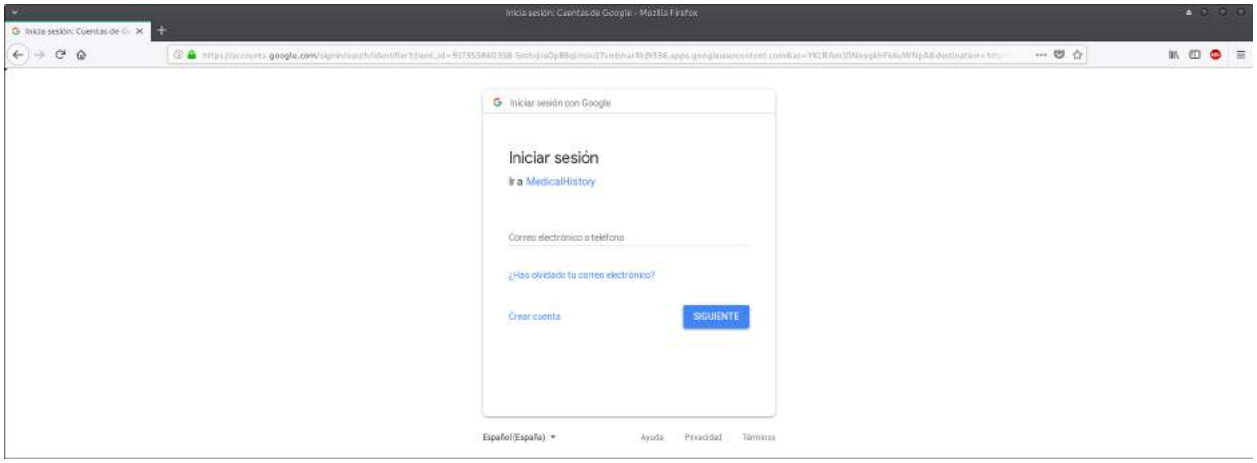

Fig. 6.6. Página identificación de usuario con proveedor Google: Usuario.

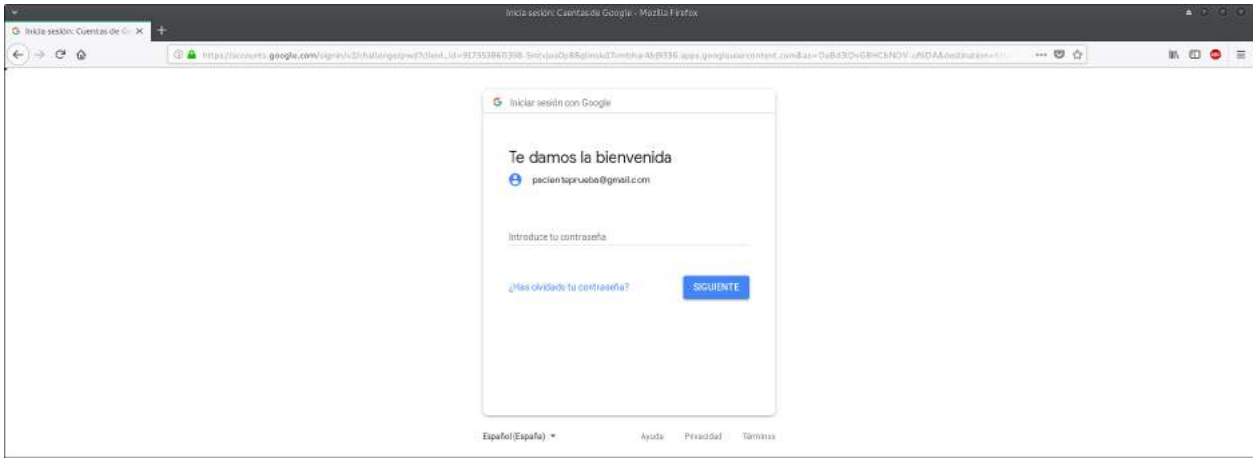

Fig. 6.7. Página identificación de usuario con proveedor Google: Contraseña.

Si aún no hemos elegido perfil, se nos mostrará la pantalla de elección de perfil (Fig.6.8), si ya tenemos nuestro perfil de médico o paciente, se nos mostrará la pantalla principal correspondiente a nuestro perfil, como veremos en sendos apartados 4.1 y 5.1.

3. Usuario identificado

Una vez identificados, si aún no tenemos perfil, se nos mostrará la siguiente pantalla, donde elegiremos el perfil que vamos a crear dentro de Medical History, de paciente o médico. Si ya tenemos perfil nos mostrará la pantalla principal correspondiente a nuestro perfil,como veremos en el apartado 4.1 y 5.1 respectivamente.

Fig. 6.8. Página elección de perfil.

Si aún no vamos a crear ningún perfil podemos salir de Medical History Pulsando sobre nuestro nombre en la barra de navegación superior, donde se nos mostrará el botón 'Logout'.

Fig. 6.9. Logout medical history sin perfil.

3.1. Crear perfil médico

Para crear un perfil médico (previamente identificado) pulsaremos sobre el botón 'Crear Perfil Médico'.

Fig. 6.10. Página elección de perfil: médico.

A continuación se nos mostrará la siguiente pantalla:

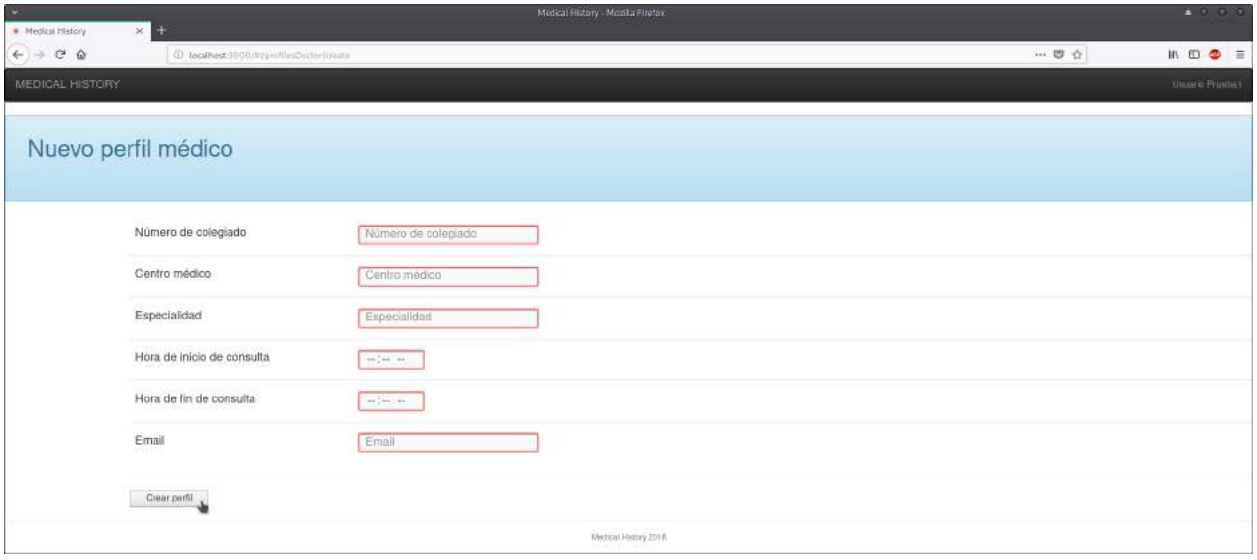

Fig. 6.11. Página crear perfil médico.

- Número de colegiado: número de colegiado que se te asigno al colegiarte como médico, con el formato XX-YY-ZZZZZ.
- Centro médico: lugar donde realizás las consultas.
- Especialidad: especialidad médica que ejerces.
- Hora de inicio de consulta: hora en la que estas disponible para atender consultas en el centro médico.
- Hora de fin de consulta: hora en la que dejas de estar disponible para atender consultas en el centro médico.
- Email: correo electrónico de contacto.

* Los campos recuadrados en color deben ser rellenados para crear el perfil.

Una vez cumplementado todos los datos, pulsamos en el botón 'Crear perfil' para finalizar y crear el perfil médico en Medical History.

3.2. Crear perfil de paciente

Para crear un perfil de paciente (previamente identificado) pulsaremos sobre el botón 'Crear Perfil Paciente'.

Fig. 6.12. Página elección de perfil: paciente.

A continuación se nos mostrará la siguiente pantalla:

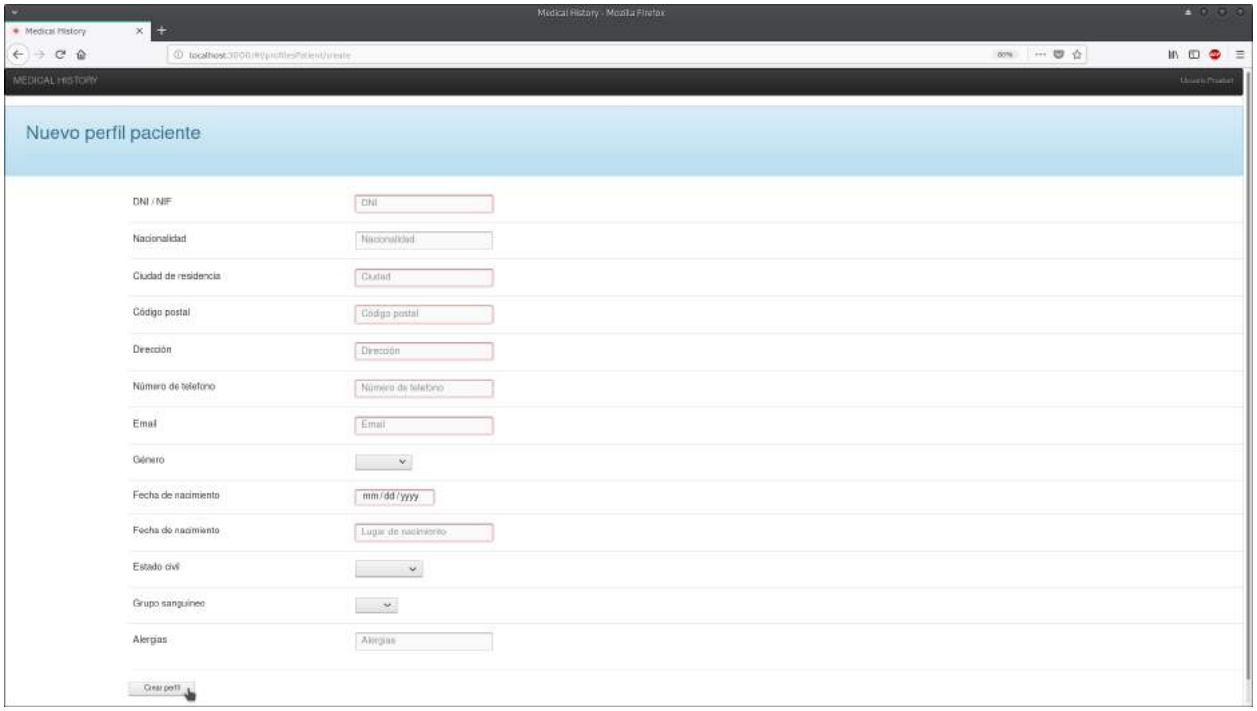

Fig. 6.13. Página crear perfil de paciente.

- DNI: número nacional de identidad con el formato 12345678A.
- Nacionalidad: país de nacimiento.
- Ciudad de residencia.
- Código postal: código postal del domicilio.
- Dirección: dirección del domicilio.
- Número de teléfono: número de telefono de contacto.
- Email: correo electrónico de contacto.
- Género: sexo masculino o femenino.
- Fecha de nacimiento.
- Estado civil: estado civil actual.
- Grupo sanguíneo.
- Alergias: alergias conocidas.

* Los campos recuadrados en color deben ser rellenados para crear el perfil.

Una vez cumplementado todos los datos, pulsamos en el botón 'Crear perfil' para finalizar y crear el perfil de paciente en Medical History.

* Una vez que hemos creado algún perfil, no se podrá crear otro perfil desde la misma cuenta de usuario.

4. Médico identificado

4.1. HOME

Una vez tengamos nuestro perfil médico creado, trás identificarnos, veremos la pantalla principal que vemos a continuación.

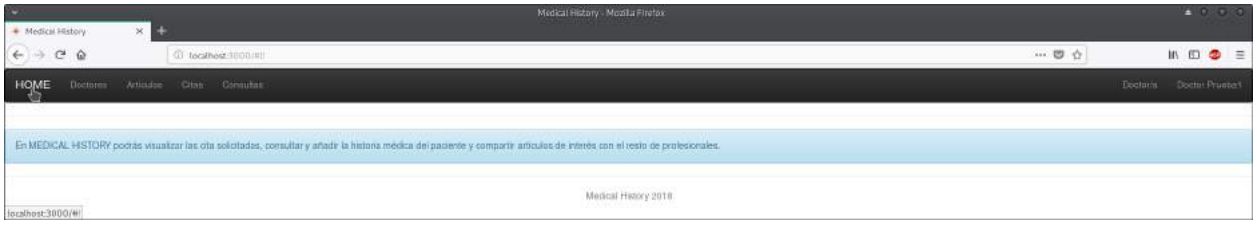

Fig. 6.14. Página principal usuario médico.

Siempre podemos volver a la pantalla principal pulsando sobre ’HOME’ en la barra de navegación.

4.2. Perfil de médico

Si queremos ver nuestro perfil médico, pulsamos sobre nuestro nombre situado en la barra superior de navegación,

Fig. 6.15. Barra navegación usuario médico.

y a continuación en el botón ’Perfil’,

Fig. 6.16. Barra navegación usuario médico: acceso a perfil.

donde ya podremos ver nuestro perfil de médico.

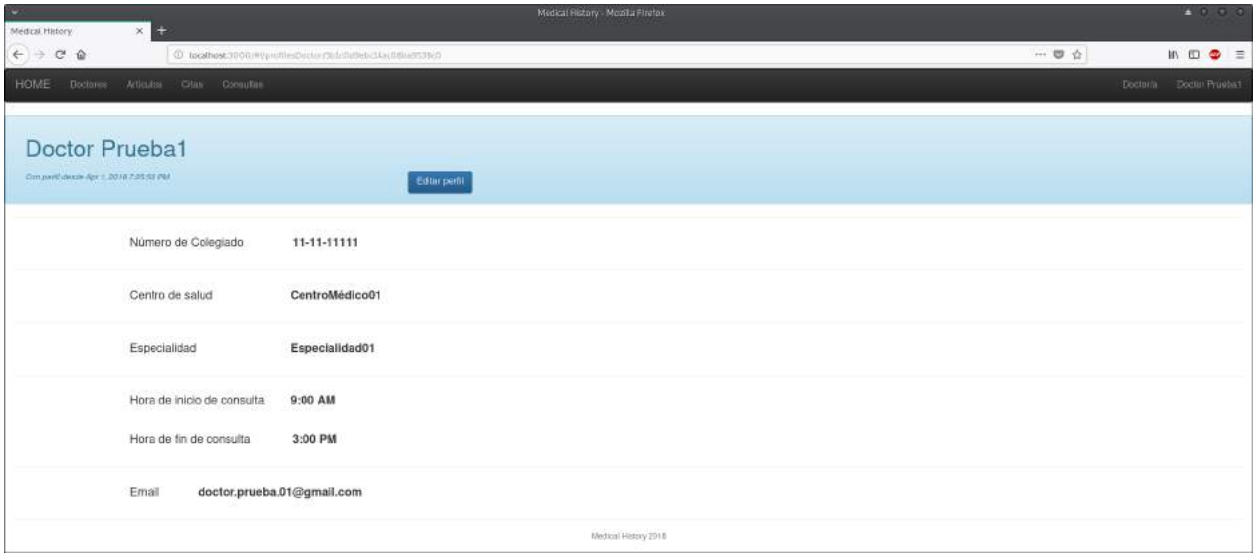

Fig. 6.17. Página de detalle de perfil médico.

4.2.1. Modificar perfil médico

Para modificar nuestro perfil médico, hemos de seguir los pasos vistos anteriormente en el apartado 4.2. y a continuación hacer click sobre el botón 'Editar perfil', o también lo podemos modificar accediendo al perfil mediante la lista de doctores(Fig.6.21).

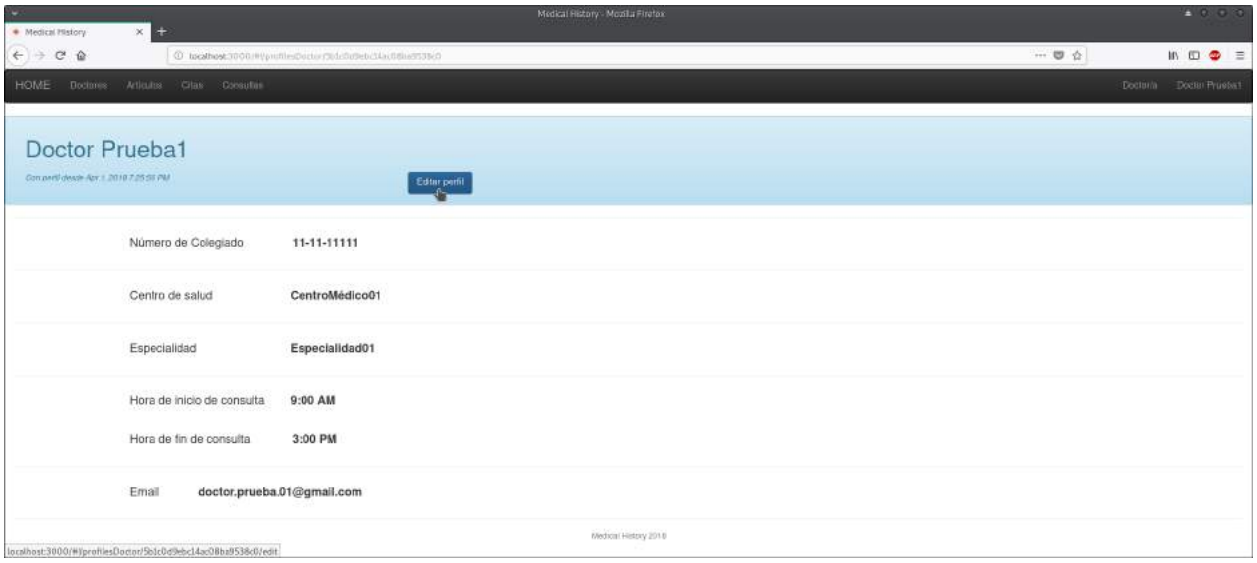

Fig. 6.18. Página de perfil: modificación de perfil.

, donde nos dirigirá a la siguiente pantalla.

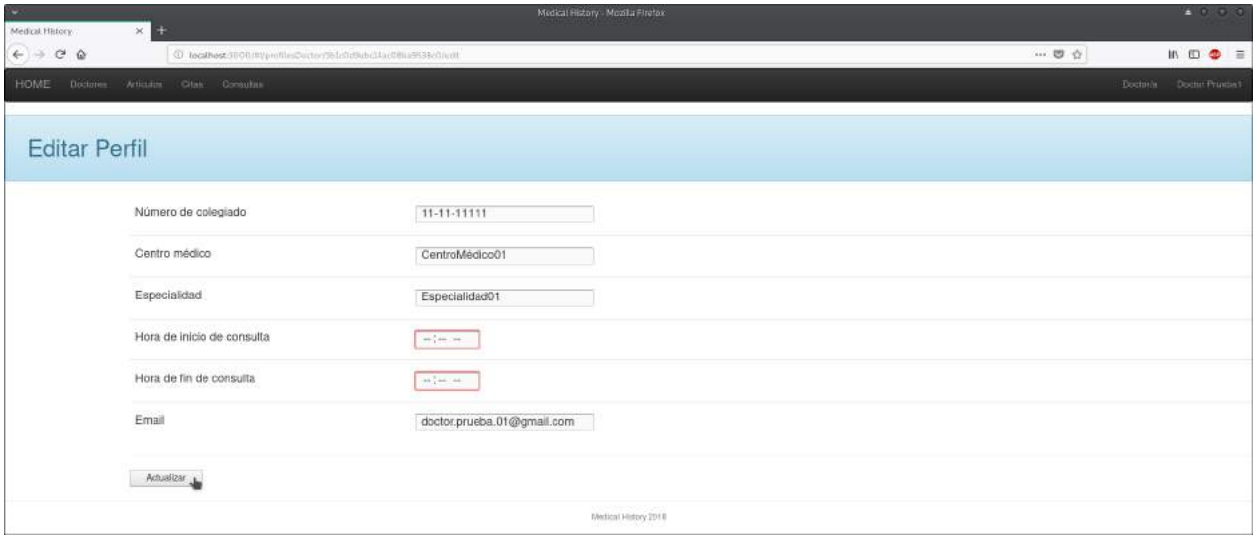

Fig. 6.19. Pantalla de actualización de perfil médico.

Una vez modificados los datos pulsamos sobre el botón 'Actualizar'.

4.3. LogOut

Si queremos salir de Medical History pulsamos sobre nuestro nombre situado en la barra superior de navegación (Fig. 6.15.) y posteriormente sobre el botón 'Logout'.

Fig. 6.20. Barra navegación usuario médico: logout.

4.4. Doctores/as

Para ver una lista de los doctores de Medical History pulsamos sobre 'Doctores' en la barra de navegación.

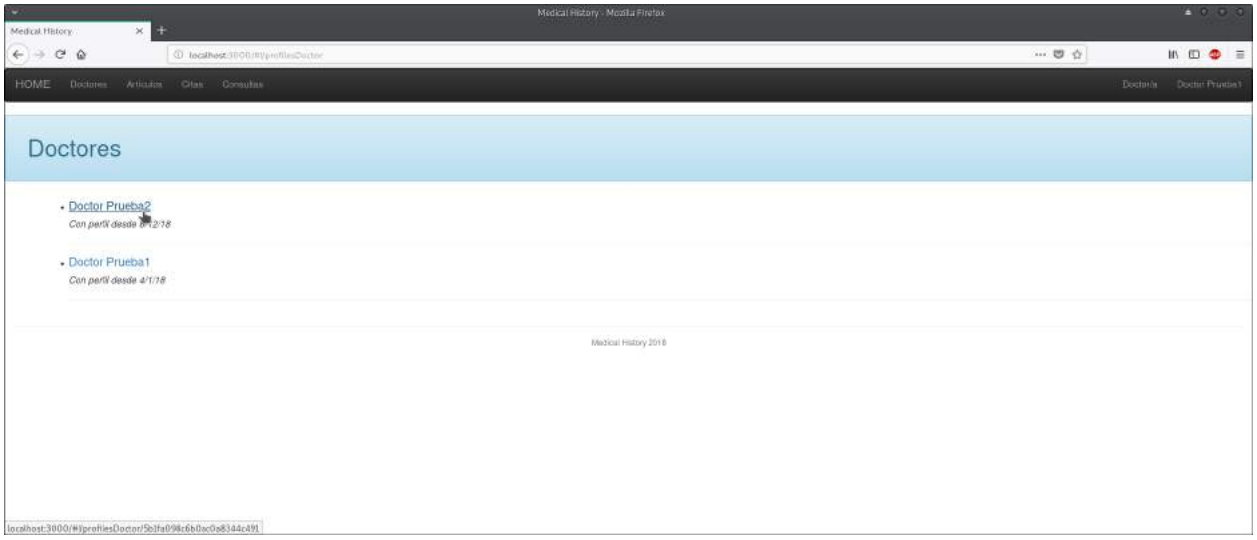

Fig. 6.21. Página de lista de doctores.

Si pulsamos sobre el nombre de algún médico veremos su perfil.

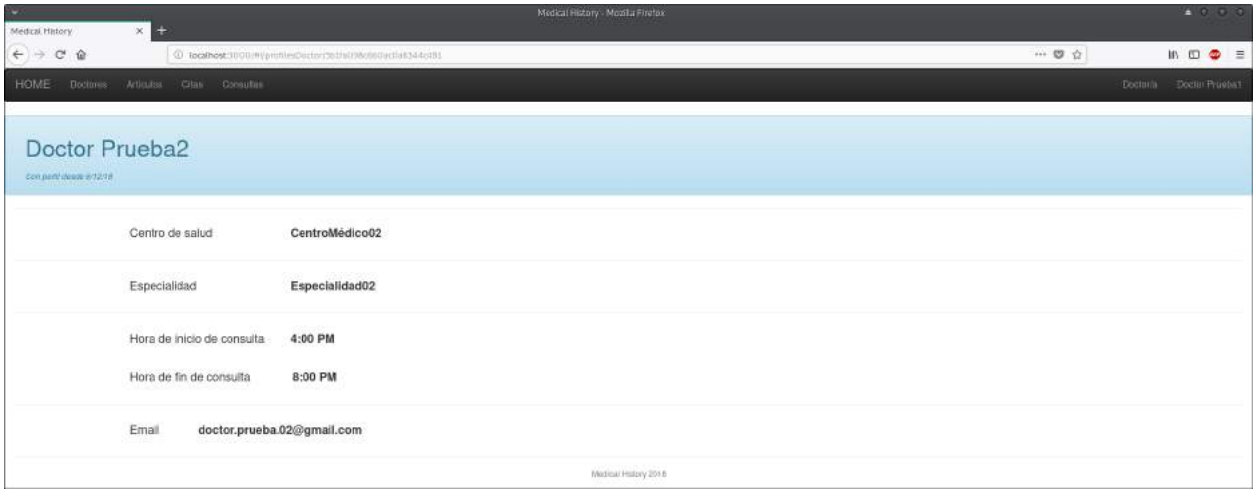

Fig. 6.22. Página de perfil de doctor.

4.5. Artículos

Para ver una lista de los artículos de interés que han sido creados en Medical History pulsamos sobre 'Artículos' en la barra de navegación.

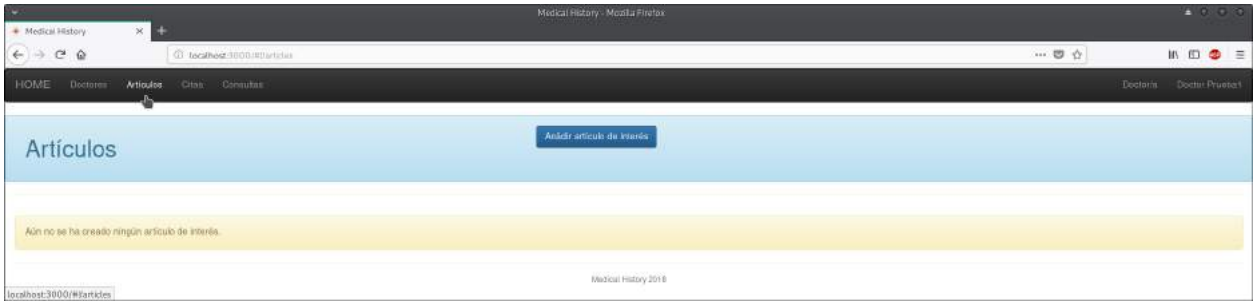

Fig. 6.23. Página de lista de artículos de interés.

4.5.1. Crear artículo de interés

Para crear un nuevo artículo de interés accederemos a la lista de artículos (pulsando sobre 'Artículos' en la barra de navegación Fig.6.23) y haremos click sobre el botón 'Añadir artículo de interés'.

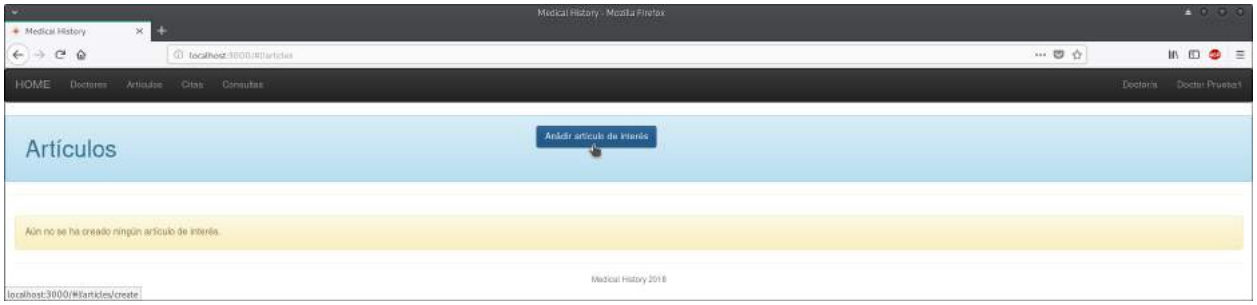

Fig. 6.24. Página de lista de artículos de interés: añadir artículo.

Seguidamente completaremos los campos del artículo que vemos en la siguiente pantalla.

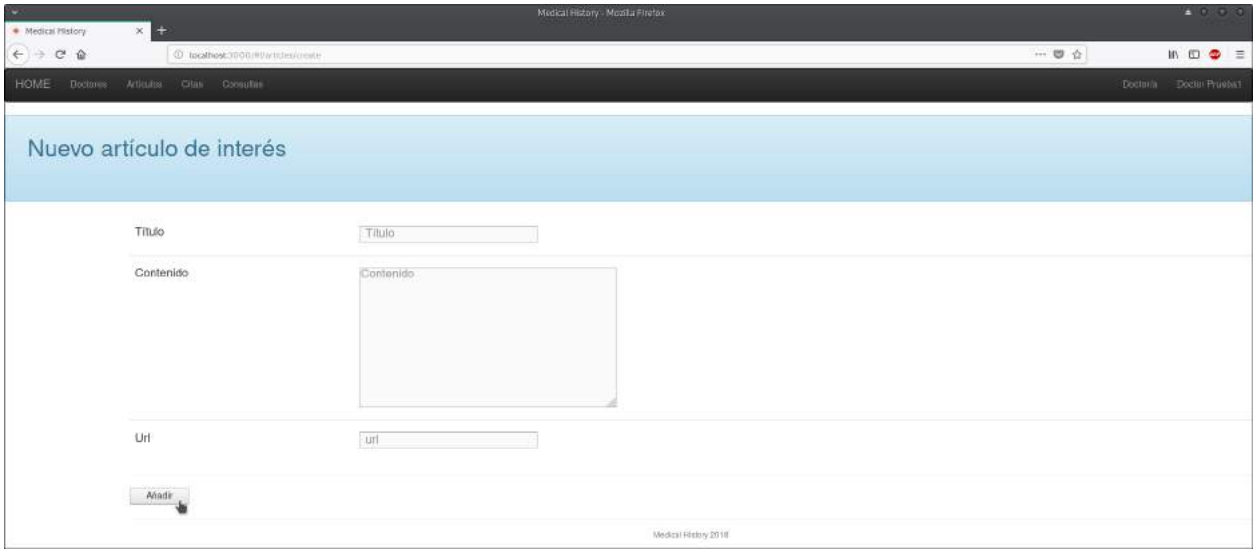

Fig. 6.25. Página de creación de artículo de interés.

- Título: título del artículo.
- Contenido: tema o resumen del artículo.
- URL: enlace al artículo.

  * El título del artículo es obligatorio.

Según creemos el artículo se nos mostrarán los detalles de este, donde podremos modificarlo o eliminarlo.

4.5.2. Modificar artículo de interés

Para modificar un artículo de interés (hemos de ser el creador del artículo) accederemos a la lista de artículos creados (pulsando sobre 'Artículos' en la barra de navegación Fig.6.23) y pode-

mos editarlo de dos formas: pulsando el botón editar situado a la derecha del nombre o entrando en los detalles del artículo pulsando sobre su título, donde aparecerá también el botón de editar.

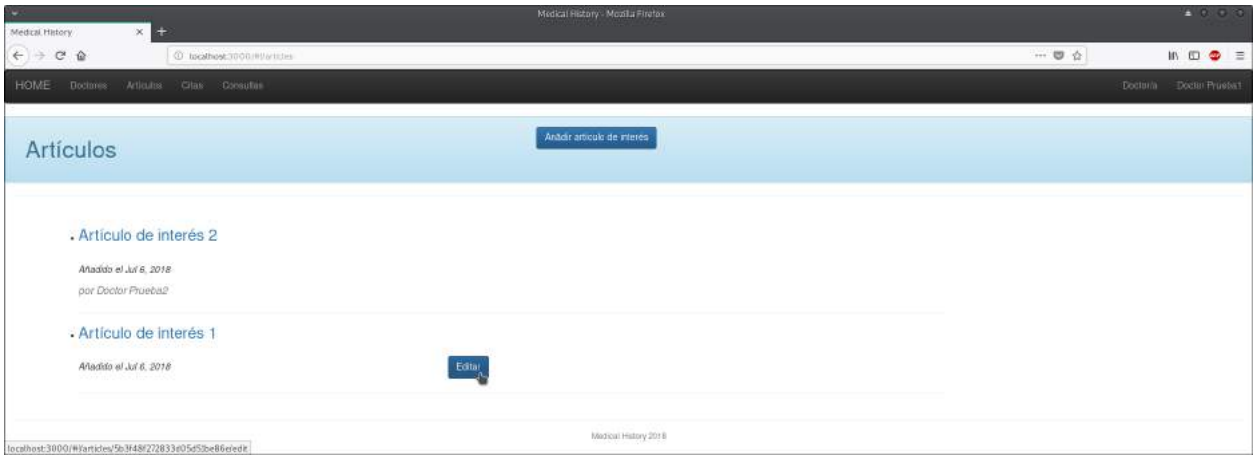

Fig. 6.26. Página lista de artículos: modificación de artículo de interés.

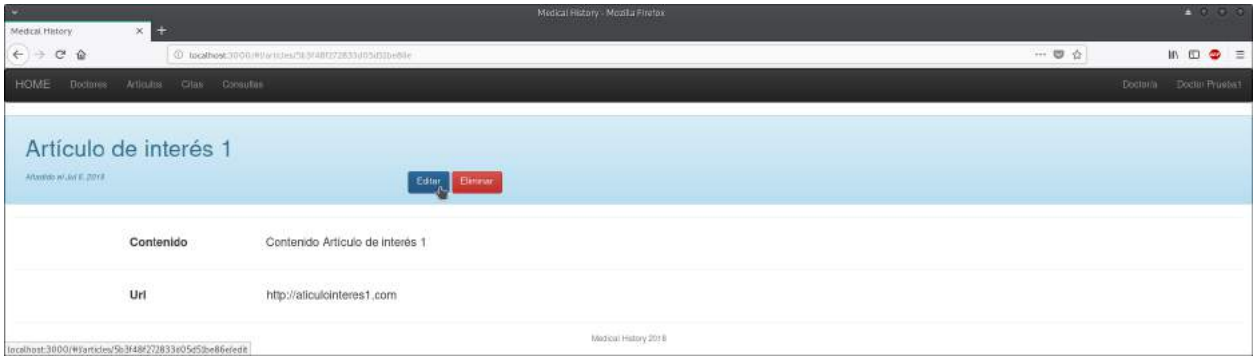

Fig. 6.27. Página detalle de artículo: modificación de artículo de interés.

En ambos casos no mostrará la siguiente pantalla, donde introduciremos los datos que queremos modificar.

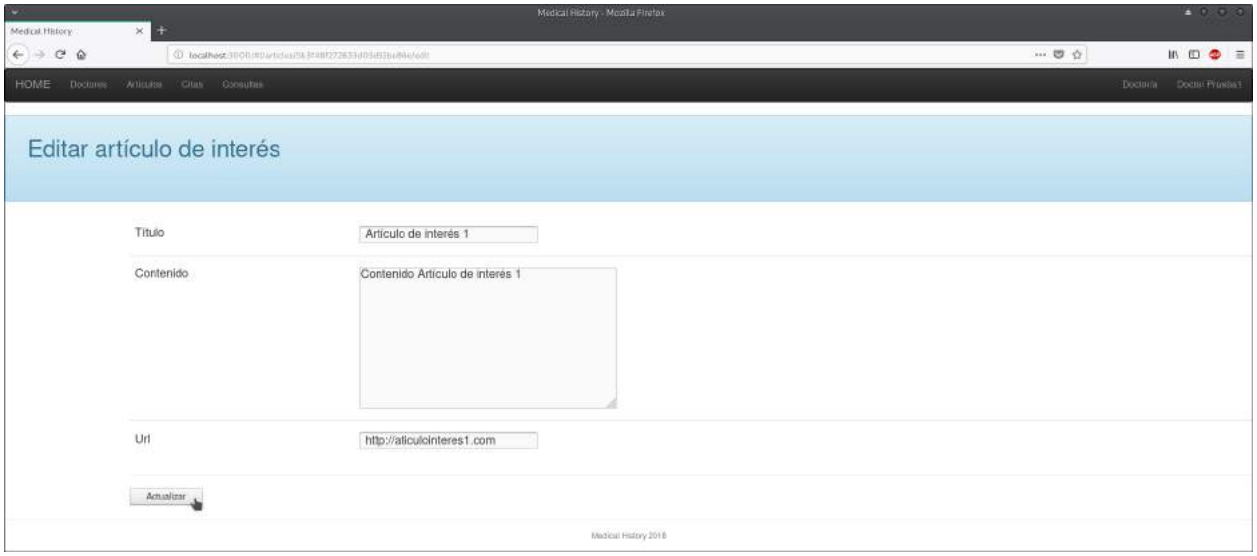

Fig. 6.28. Página de modificación de artículo de interés.

Finalmente hacemos click sobre el botón 'Actualizar'.

4.5.3. Eliminar artículo de interés

Para eliminar un artículo de interés (hemos de ser el creador del artículo) accederemos a la lista de artículos creados (pulsando sobre 'Artículos' en la barra de navegación Fig.6.23) y haciendo click sobre el título del artículo (Fig.6.26) aparecerá, junto al botón de 'Editar', el botón 'Eliminar'.

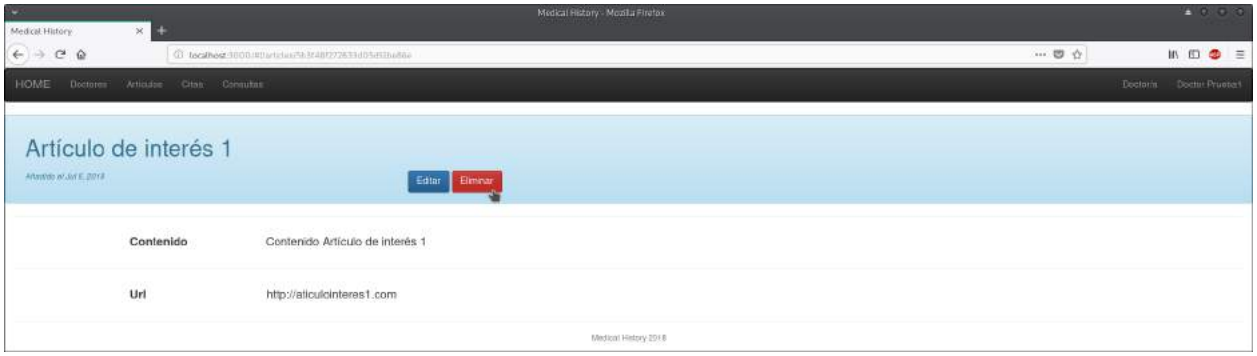

Fig. 6.29. Página de detalle de artículo de interés: eliminar.

4.6. Atender citas

Para ver la lista de las citas solicitadas pulsamos sobre 'Citas' en la barra de navegación.

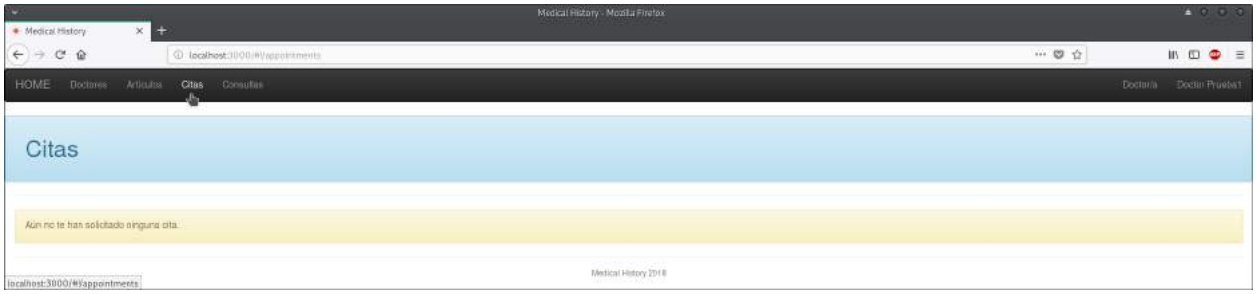

Fig. 6.30. Página de citas solicitadas vacia.

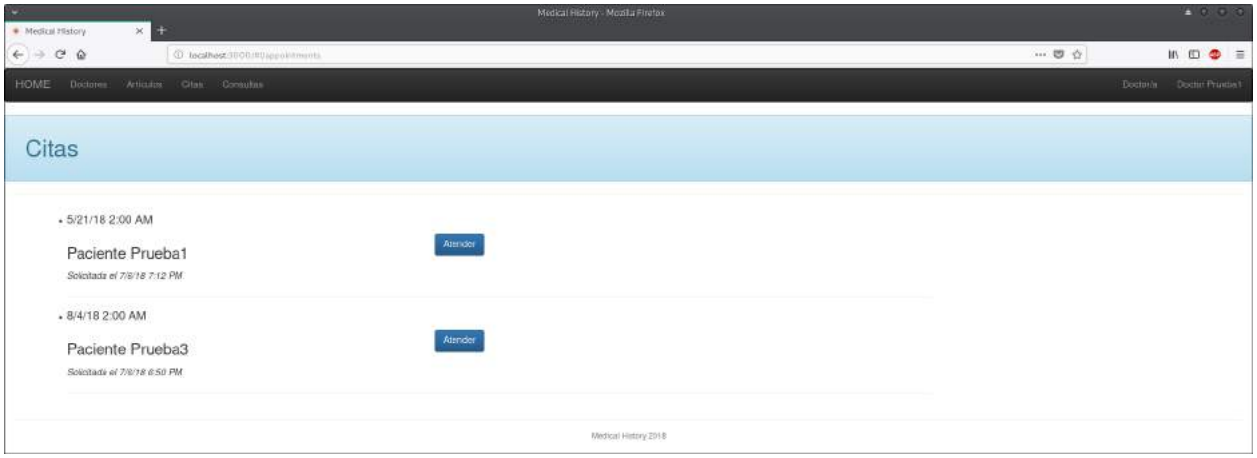

Fig. 6.31. Página de citas solicitadas.

Para atender una cita pulsamos sobre el botón 'Atender' situado junto a su fecha mostrandonos la pantalla que vemos a continuación.

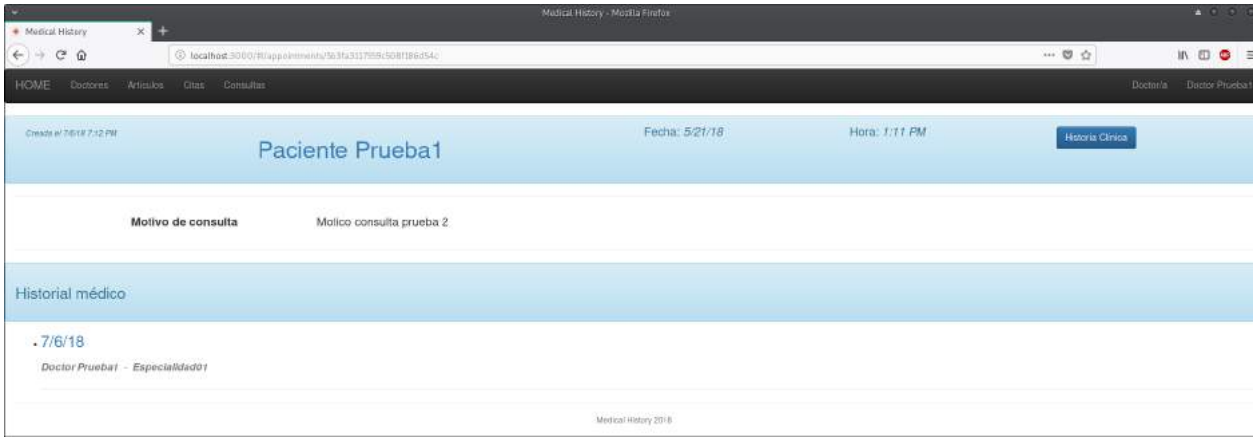

Fig. 6.32. Página de atención de cita.

4.6.1. Consultar perfil de paciente citado

Una vez estamos atendiendo una cita, si pulsamos sobre el nombre del paciente citado accedemos a su perfil.

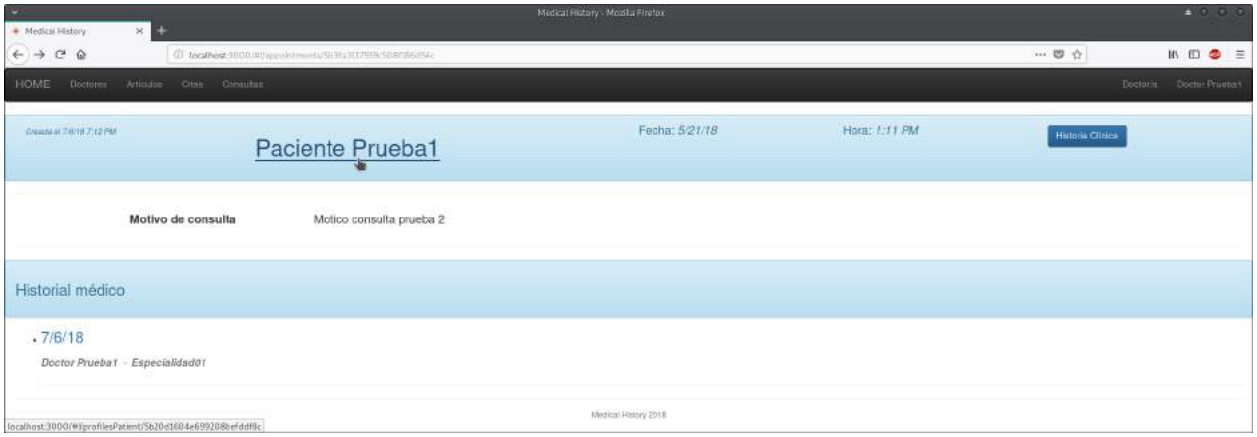

Fig. 6.33. Página de atención de cita: perfil de paciente.

Donde mostrará el perfil de paciente citado.

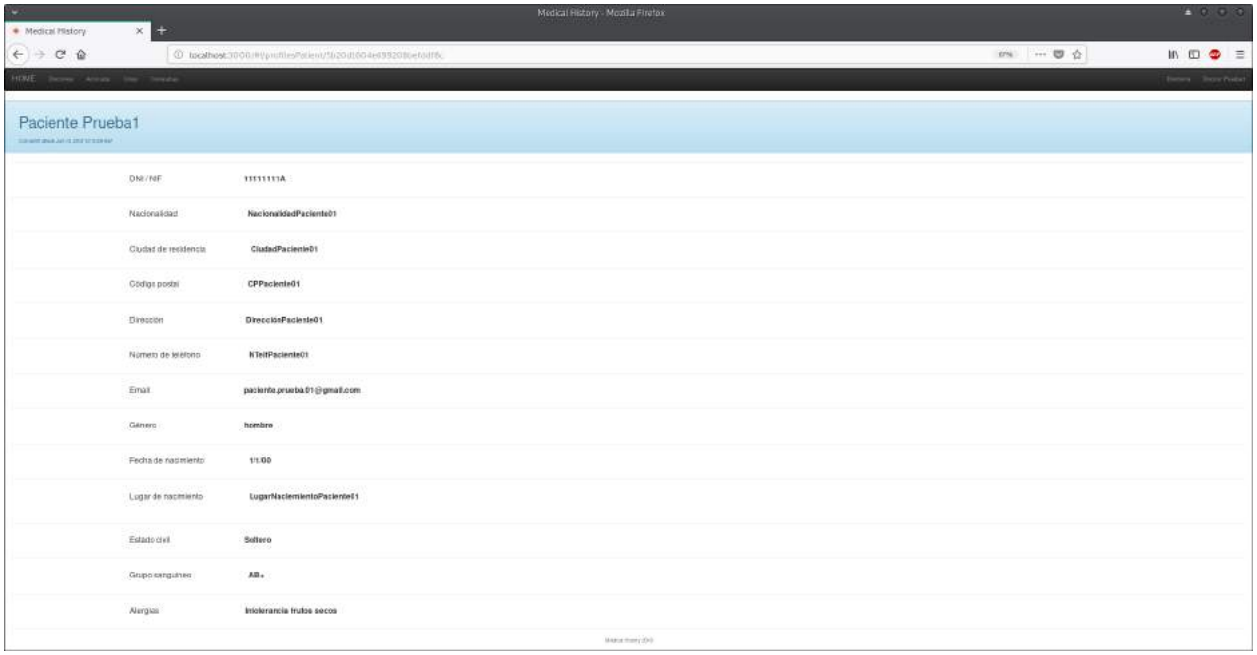

Fig. 6.34. Página de atención de cita: perfil de paciente.

4.6.2. Consultar historial médico de paciente citado

Una vez estamos atendiendo una cita, si pulsamos sobre la fecha de una historia médica,

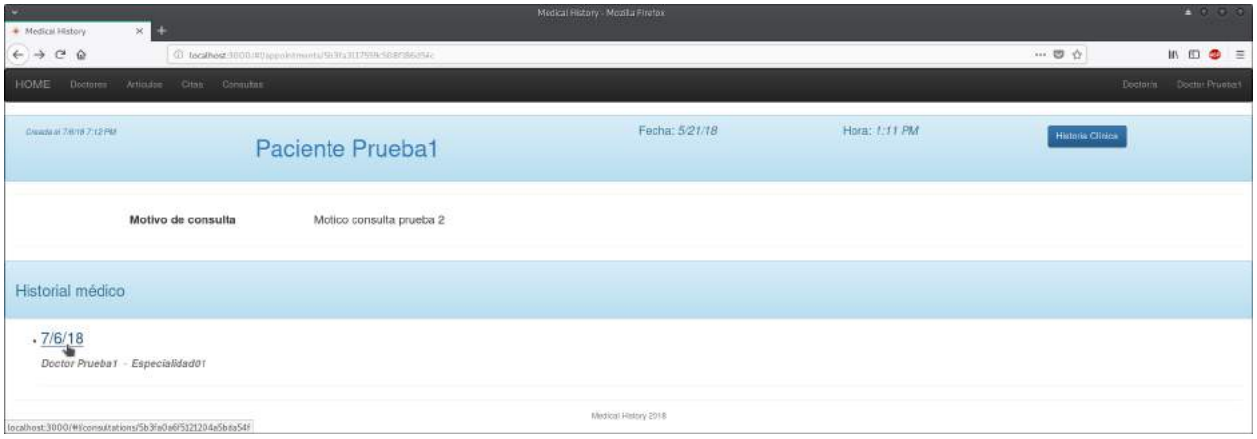

Fig. 6.35. Página de atención de cita: historia médica de paciente.

nos mostrará sus detalles:

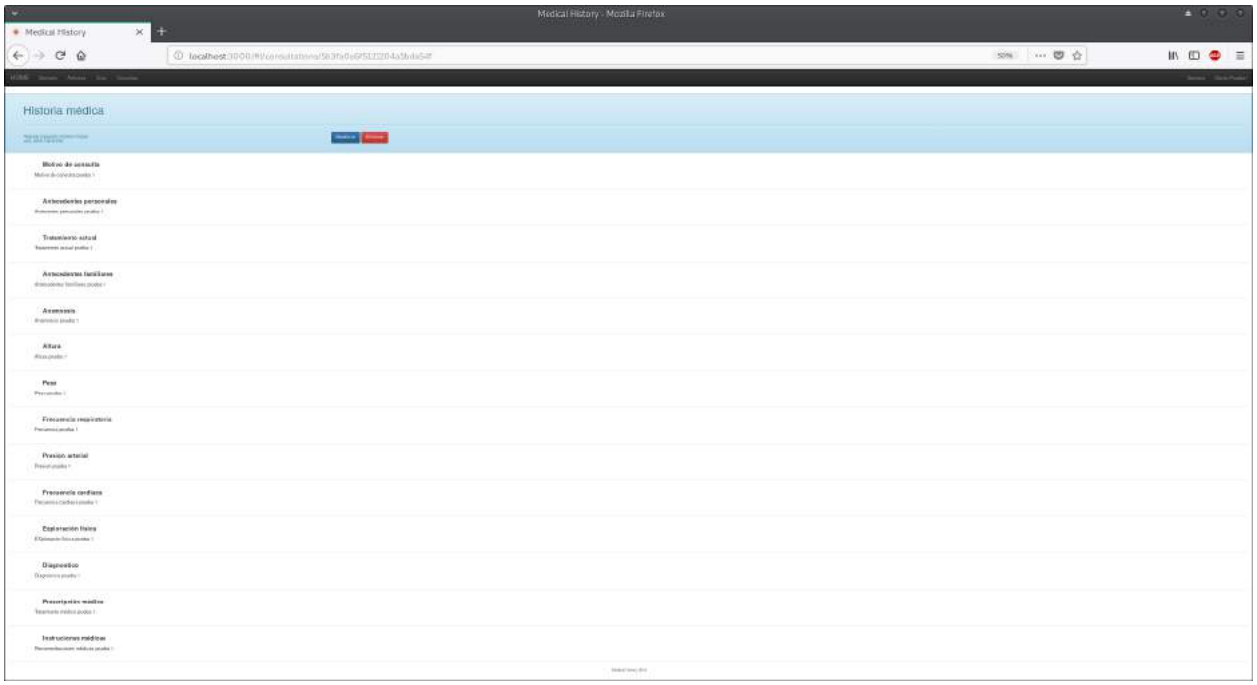

Fig. 6.36. Página de detalle de episodio médico.

4.6.3. Crear historia médica

Una vez estamos atendiendo una cita, para crear una historia médica hacemos click sobre el botón 'Historia Clínica',

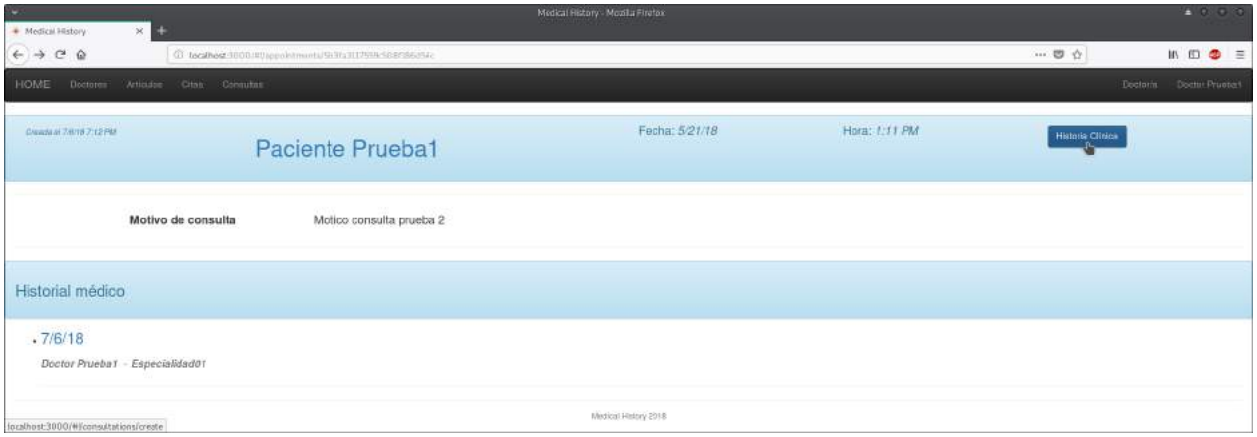

Fig. 6.37. Página de atención de cita: crear historia clínica.

donde aparecerá la siguiente pantalla:

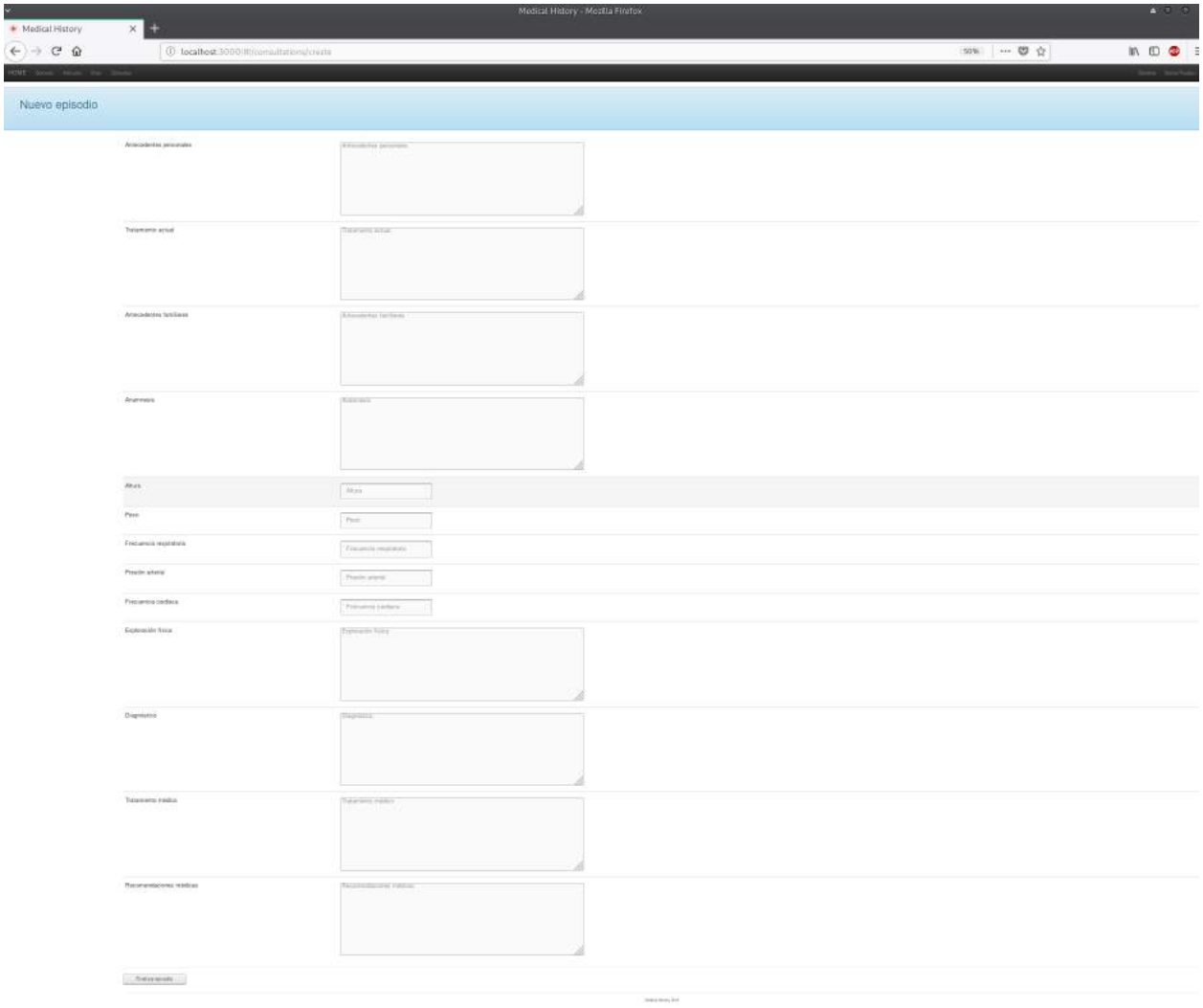

Fig. 6.38. Página de creación de historia médica.

- Motivo de consulta: principal síntoma/queja del paciente por el que acude a cosnulta.
- Antecedentes personales: enfermedades médicas y/o quirúrgicas que el paciente haya padecido y/o presente en la actualidad.
- Tratamiento actual: medicación actual del paciente.
- Antecedentes familiares: enfermedades de interés en la familia del paciente.
- Anamnesis: descripción detallada de la enfermedad por la que consulta el paciente, atendiendo a posibles factores asociados y de temporalidad.
- Altura: altura del paciente.
- Peso: peso del paciente.
- Frecuencia respiratoria: frecuencia respiratoria del paciente.
- Presión arterial: presión arterial del paciente.
- Frecuencia cardiaca: frecuencia cardiaca del paciente.
- Exploración física: evaluación del estado físico del paciente.
- Diagnostico: juicio clínico a emitir tras la evaluación del paciente.
- Prescripción médica: tratamiento médico.
- Instruciones médicas: recomendaciones terapéuticas a seguir.

Una vez hemos rellenado todos los campos necesarios, pulsamos en 'Finalizar episodio' para finalizar la consulta, mostrandonos la historia que acabamos de crear, pudiendo modificarla si procedieses haciendo click sobre el botón 'Modificar' (Fig.6.42).

4.7. Consultas

Para ver la lista de las consultas realizadas pulsamos sobre 'Consultas' en la barra de navegación.

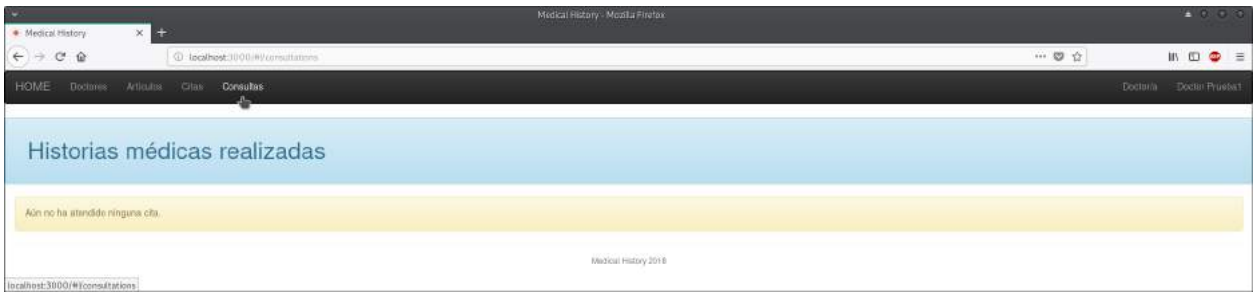

Fig. 6.39. Página de consultas realizadas vacía.

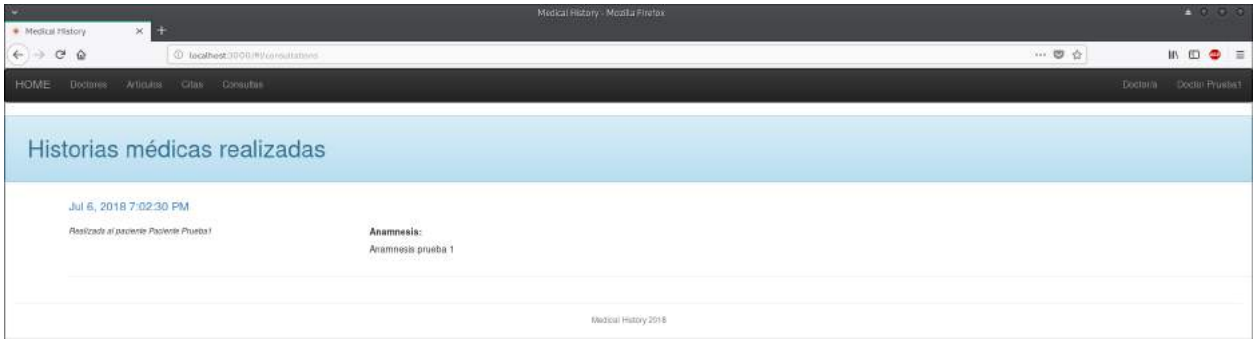

Fig. 6.40. Página de consultas realizadas.

Para acceder a los detalles de un episodio hacemos click sobre su fecha.

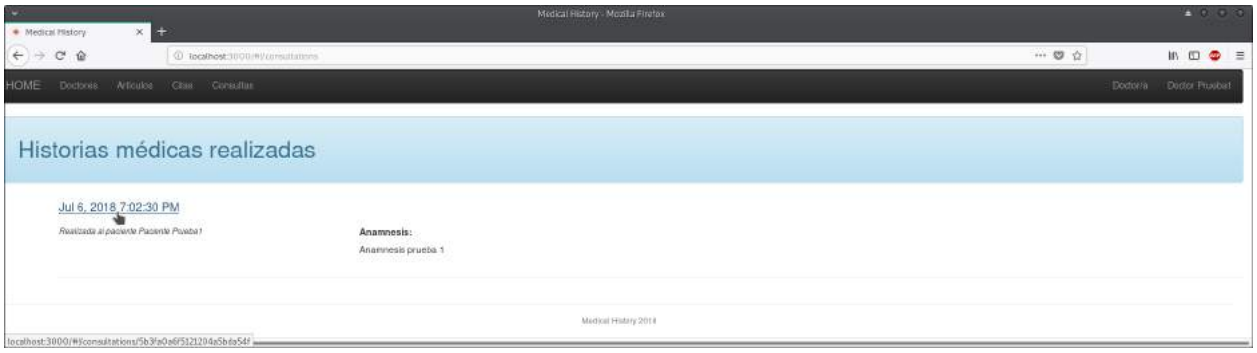

Fig. 6.41. Página de consultas realizadas: detalle.

Donde aparecerá la siguiente pantalla con los detalles del episodio.

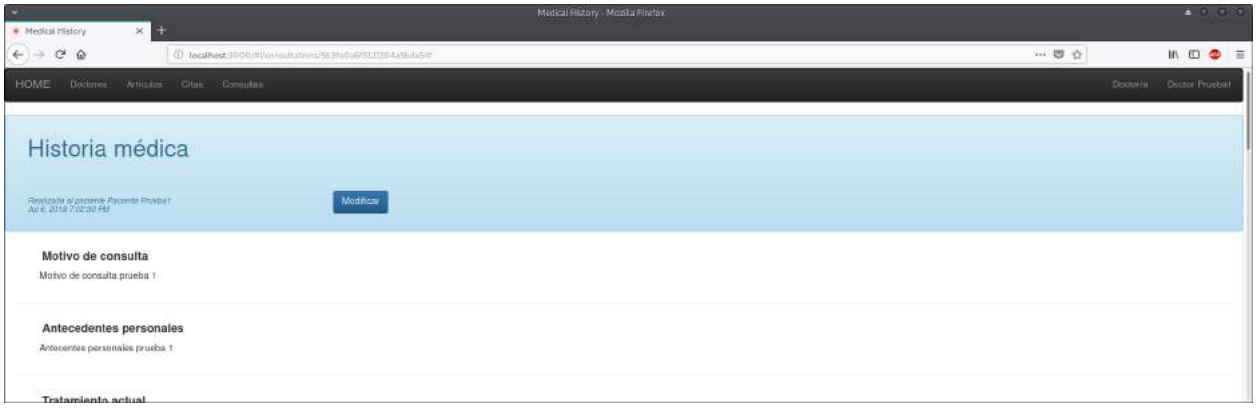

Fig. 6.42. Página de detalle de episodio médico.

4.7.1. Modificar historia médica

Para modificar un episodio (hemos de ser el creador de la historia médica) accederemos a la lista de historias médicas creadas (pulsando sobre 'Consultas' en la barra de navegación Fig.6.38) y entrar a sus detalles (Fig.6.40), donde se nos mostrará la Fig.6.41.

Hacemos click en el botón modificar para realizar cambios sobre el episodio.

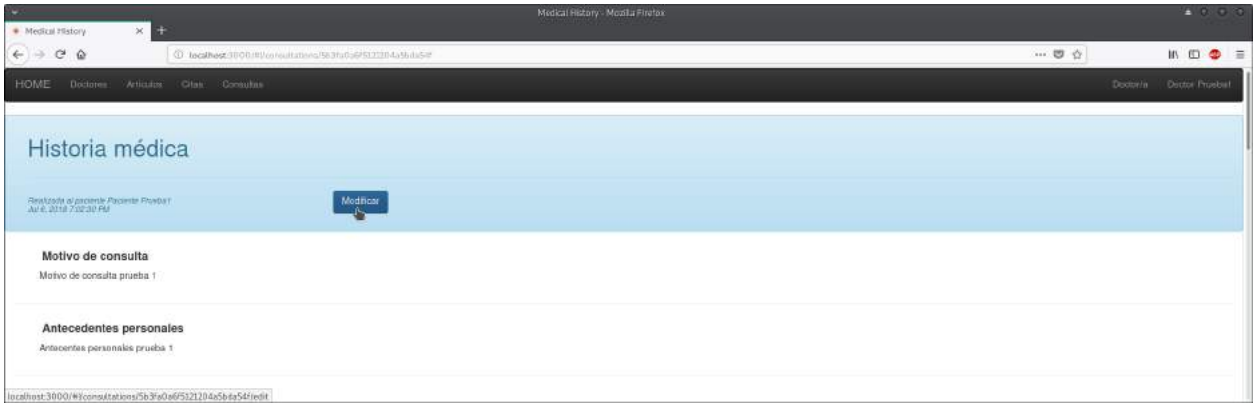

Fig. 6.43. Página de detalle de consulta realizada: modificar.

donde aparecerá la siguiente pantalla,

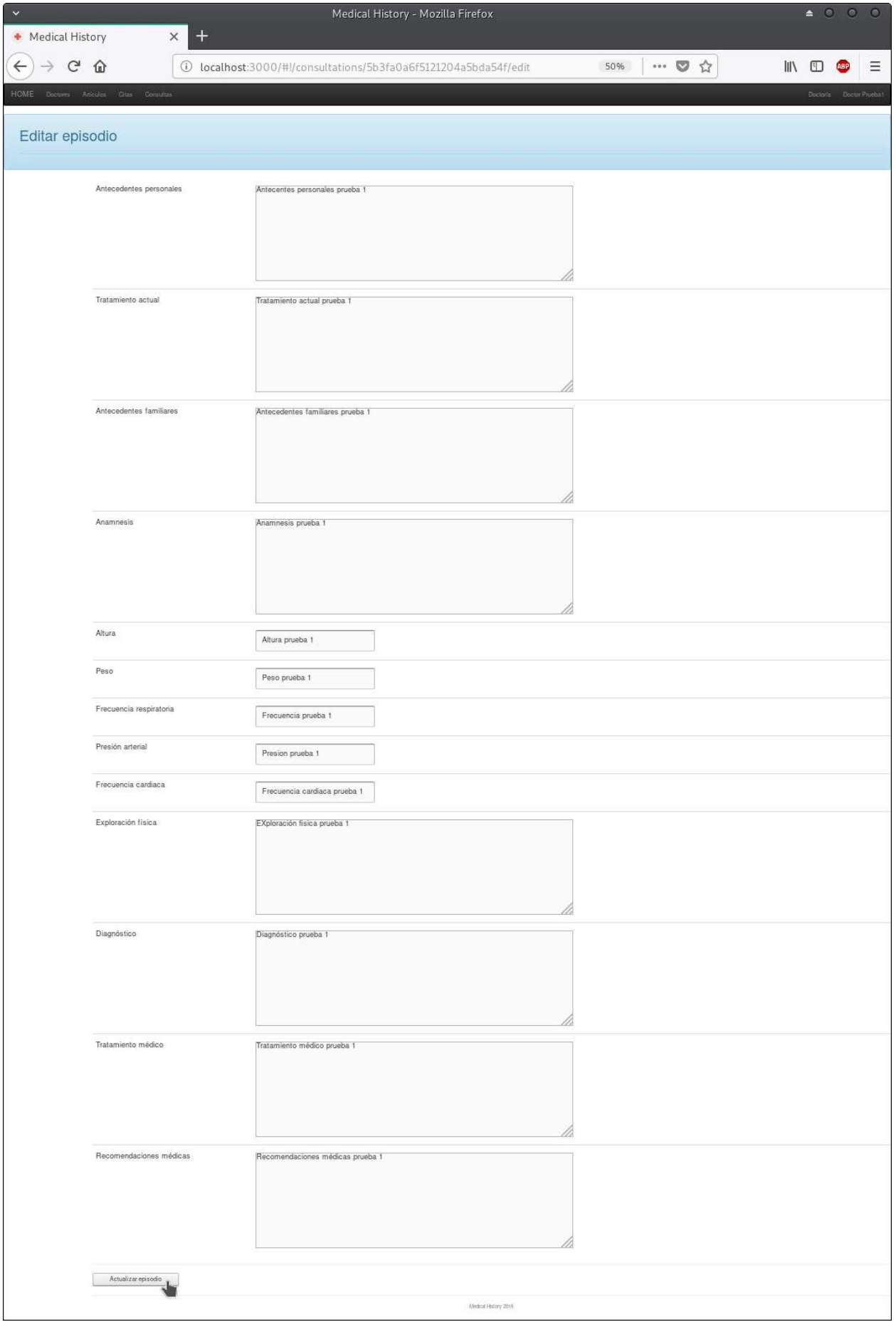

Fig. 6.44. Página de edición de consulta realizada.

Para finalizar y guardar los cambios hacemos click sobre el botón 'Actualizar episodio'.

5. Paciente identificado

5.1. HOME

Una vez tengamos nuestro perfil de paciente creado, trás identificarnos, veremos la pantalla principal que vemos a continuación.

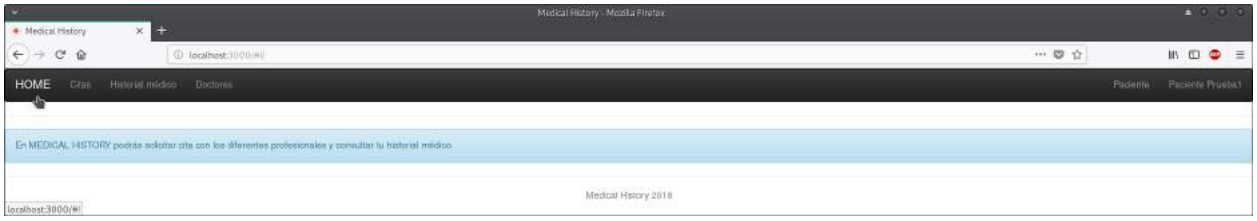

Fig. 6.45. Página principal usuario paciente.

Siempre podemos volver a la pantalla principal pulsando sobre 'HOME' en la barra de navegación.

5.2. Perfil de paciente

Si queremos ver nuestro perfil de paciente, pulsamos sobre nuestro nombre situado en la barra superior de navegación,

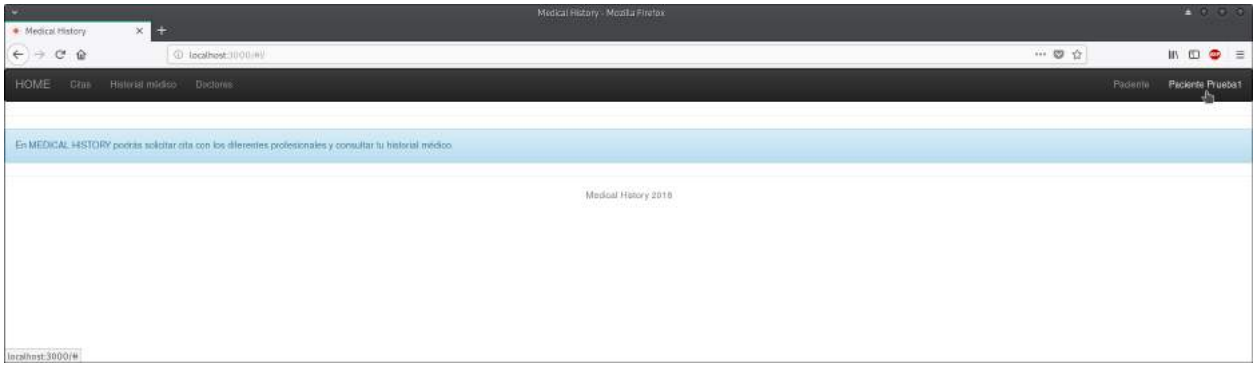

Fig. 6.46. Barra navegación usuario paciente.

y a continuación en el botón 'Perfil',

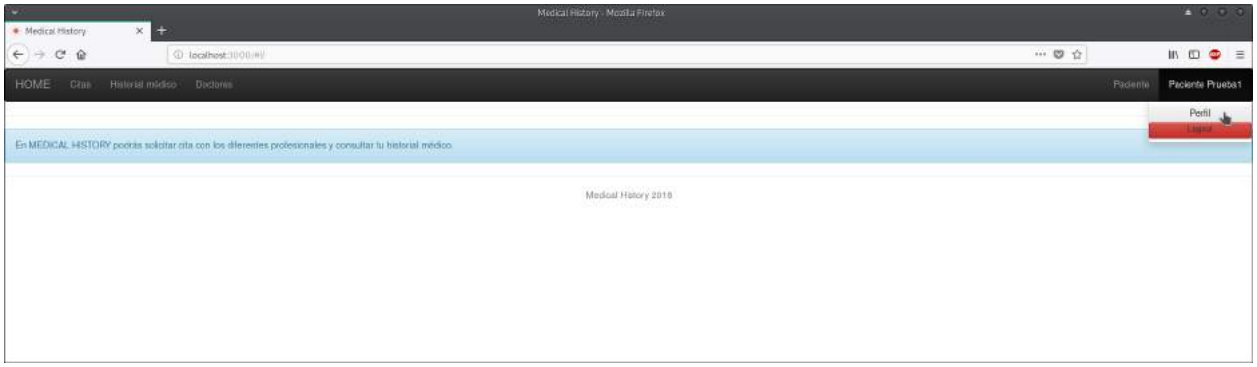

Fig. 6.47. Barra navegación usuario paciente: acceso a perfil.

donde ya podremos ver nuestro perfil de paciente.

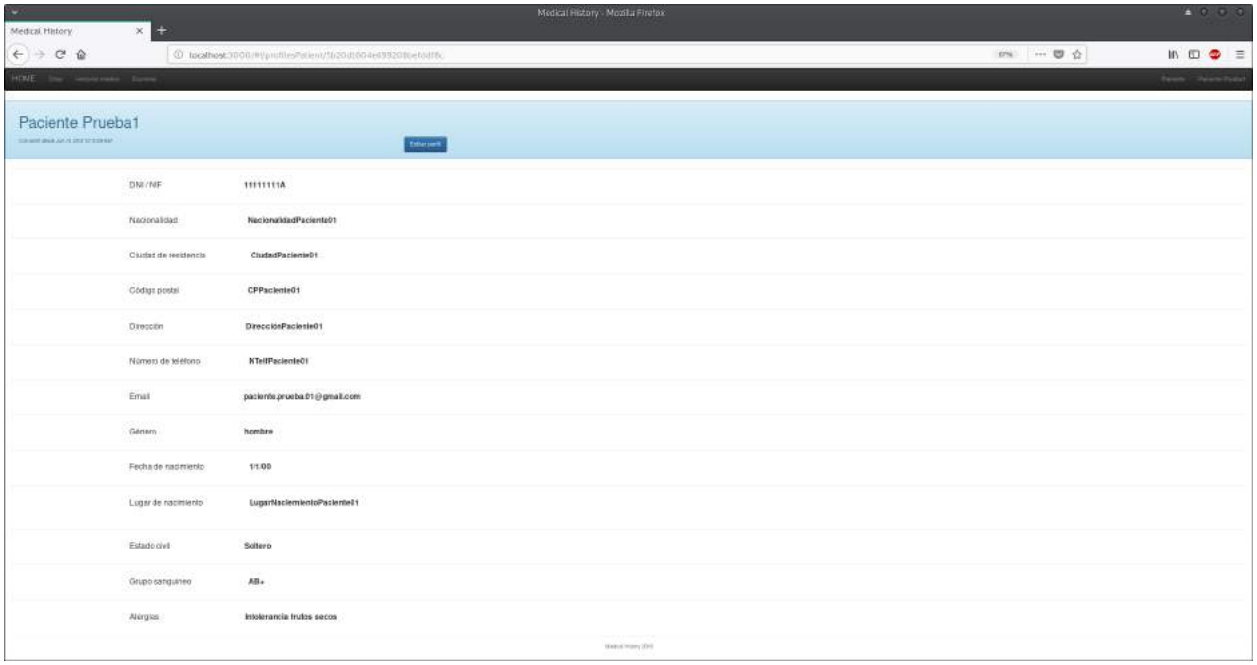

Fig. 6.48. Página de detalle de perfil de paciente.

5.2.1. Modificar perfil de paciente

Para modificar nuestro perfil de paciente, hemos de seguir los pasos vistos anteriormente en el apartado 5.2. y a continuación hacer click sobre el botón 'Editar perfil'

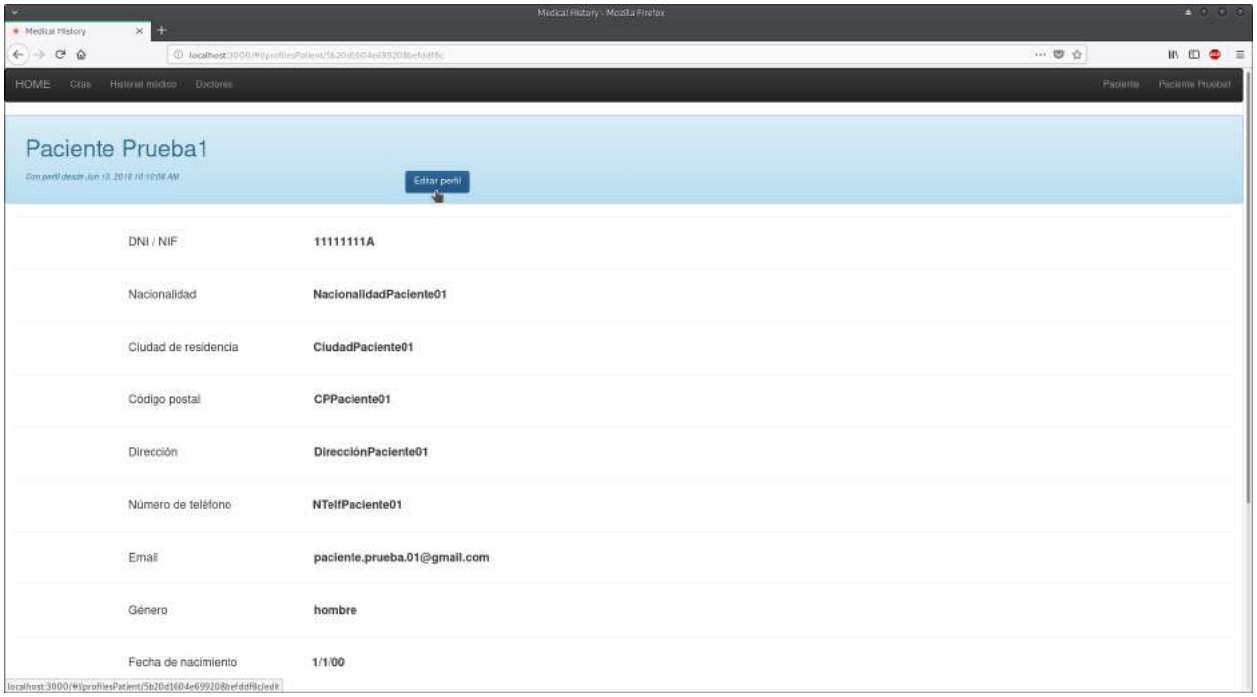

Fig. 6.49. Página de perfil: modificación de perfil.

, donde nos dirigirá a la siguiente pantalla.

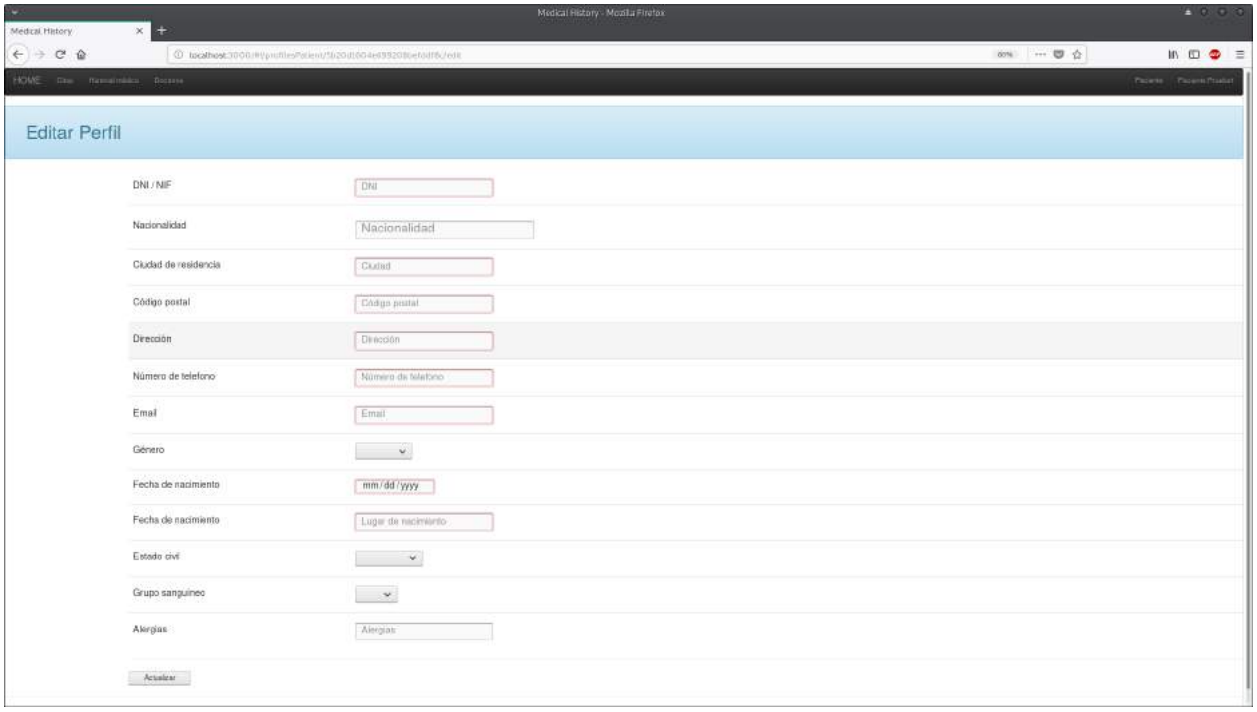

Fig. 6.50. Pantalla de actualización de perfil paciente.

Una vez modificados los datos pulsamos sobre el botón 'Actualizar'.

5.3. LogOut

Si queremos salir de Medical History pulsamos sobre nuestro nombre situado en la barra superior de navegación (Fig. 6.33.) y posteriormente sobre el botón 'Logout'.

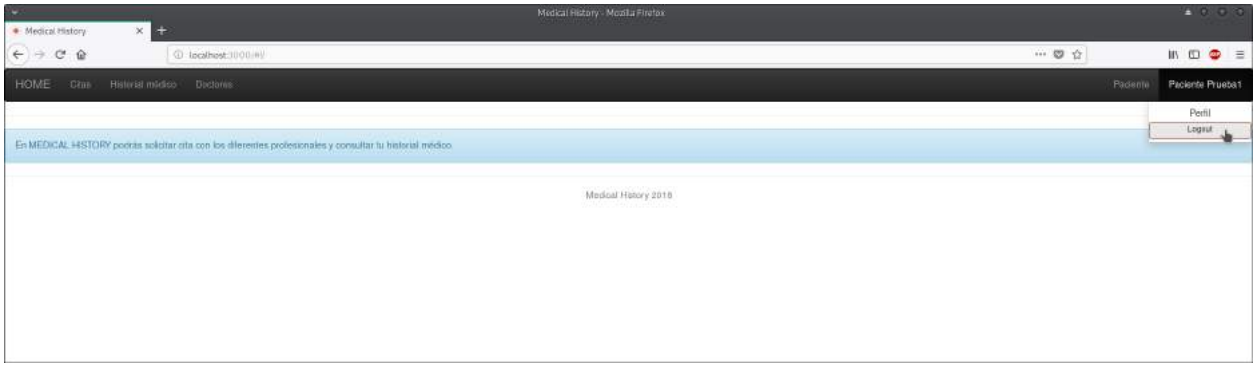

Fig. 6.51. Barra navegación usuario paciente: logout.

5.4. Doctores/as

Para ver una lista de los doctores de Medical History pulsamos sobre 'Doctores' en la barra de navegación.

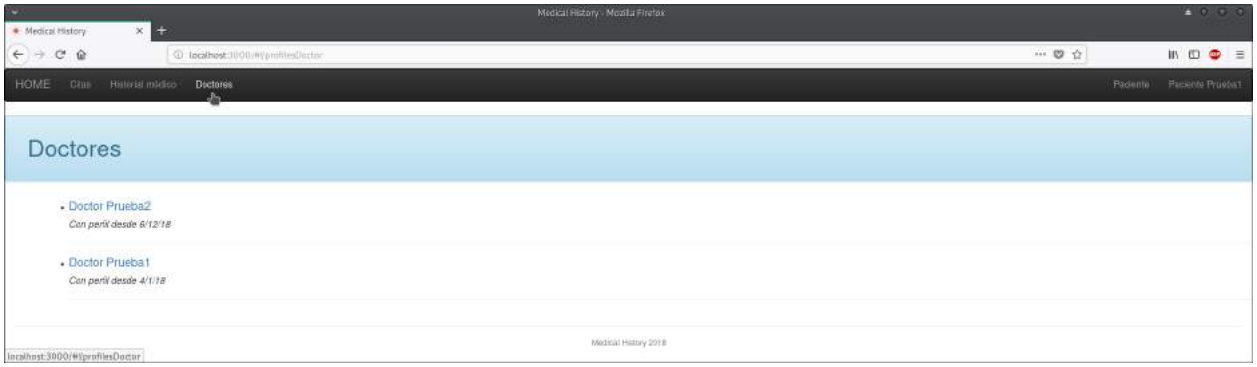

Fig. 6.52. Página de lista de doctores.

Si pulsamos sobre el nombre de algún médico veremos su perfil.

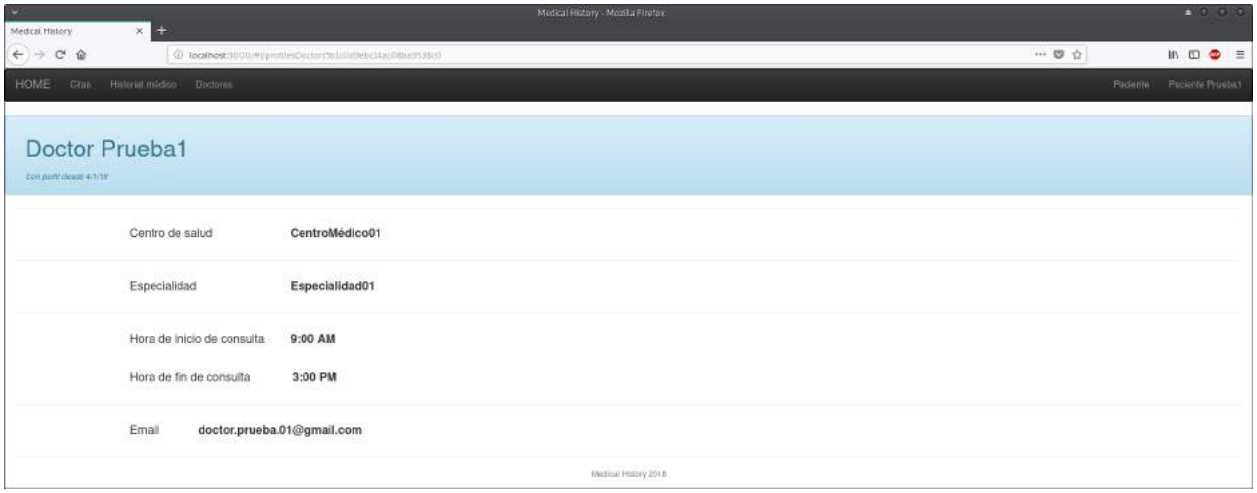

Fig. 6.53. Página de perfil de doctor.

5.5. Citas

Para ver las citas que tenemos solicitadas pulsamos sobre 'Citas' en la barra de navegación.

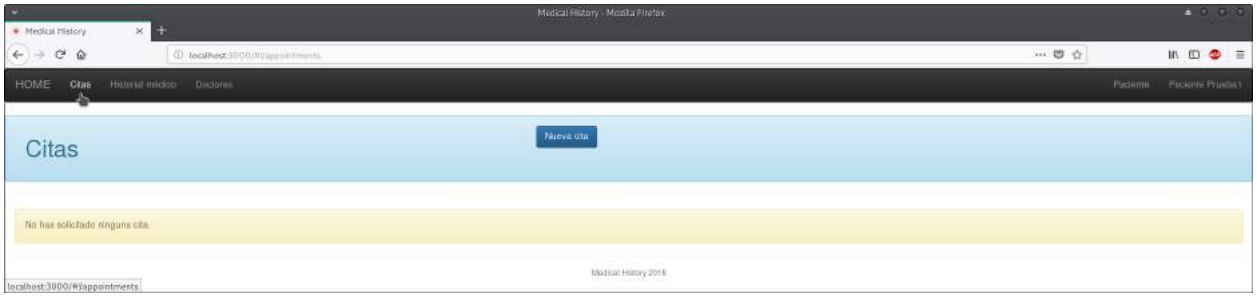

Fig. 6.54. Página de citas solicitadas.

5.5.1. Crear citas

Para solicitar una nueva cita accederemos a la lista de citas (pulsando sobre 'Citas' en la barra de navegación Fig.6.53) y haremos click sobre el botón 'Nueva cita'.

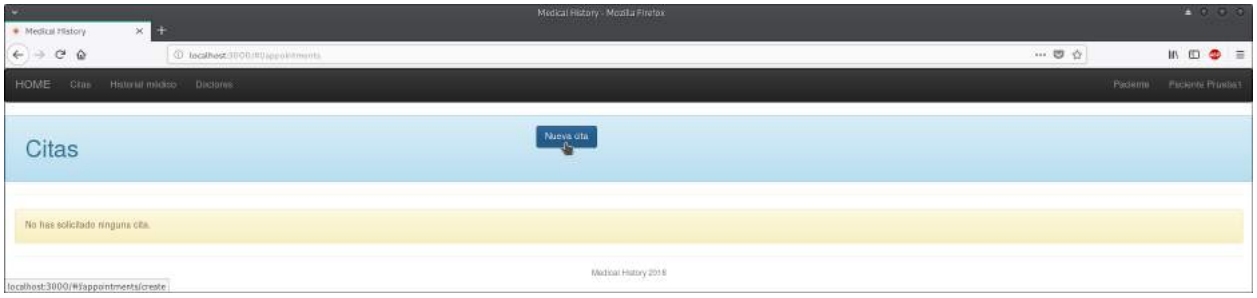

Fig. 6.55. Página de citas: añadir cita.

Seguidamente completaremos los campos de la cita que vemos en la siguiente pantalla.

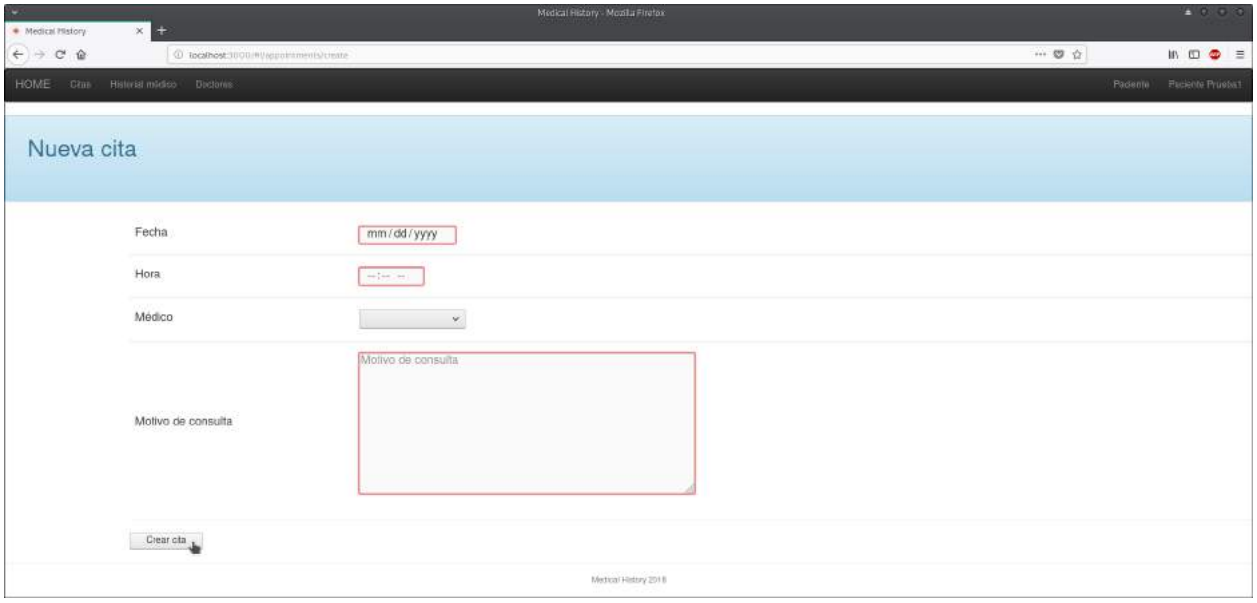

Fig. 6.56. Página de creación de cita.

- Fecha: fecha que se quiere concertar una cita.
- Hora: hora a la que se solicita la cita.
- Médico: selección de un médico de la lista de medical history.
- Motivo de consulta: motivo por el que se solicita la cita (dolencias, síntomas, cambio de medicación, etc)

  * Todos los campos son obligatorios para crear la cita.

Según creemos la cita se nos mostrarán los detalles de esta (Fig. 6.57), donde podremos modificarla o cancelarala.

5.5.2. Modificar cita

Para modificar una cita accederemos a la lista de citas solicitadas (pulsando sobre 'Citas' en la barra de navegación Fig.6.53) viendo la siguiente pantalla:

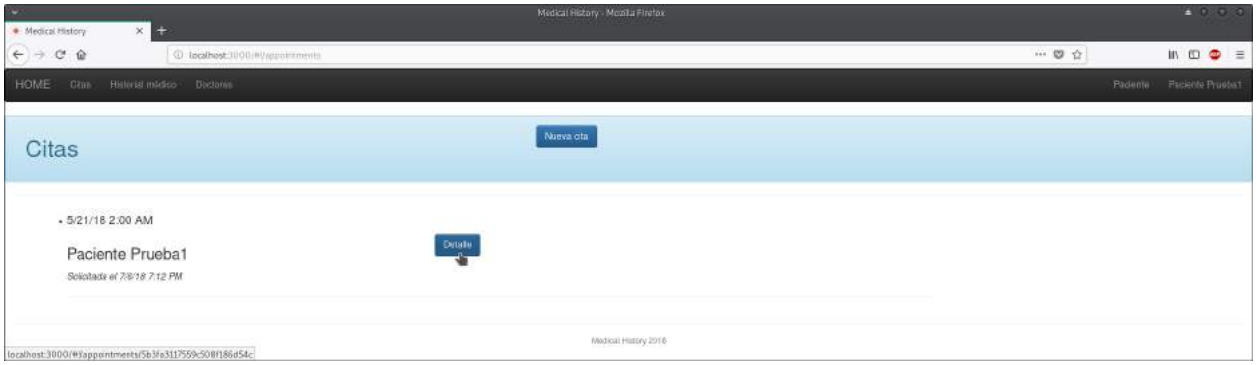

Fig. 6.57. Página de citas solicitadas con cita creada.

y haciendo click sobre el botón 'Detalle' accederemos a sus detalles.

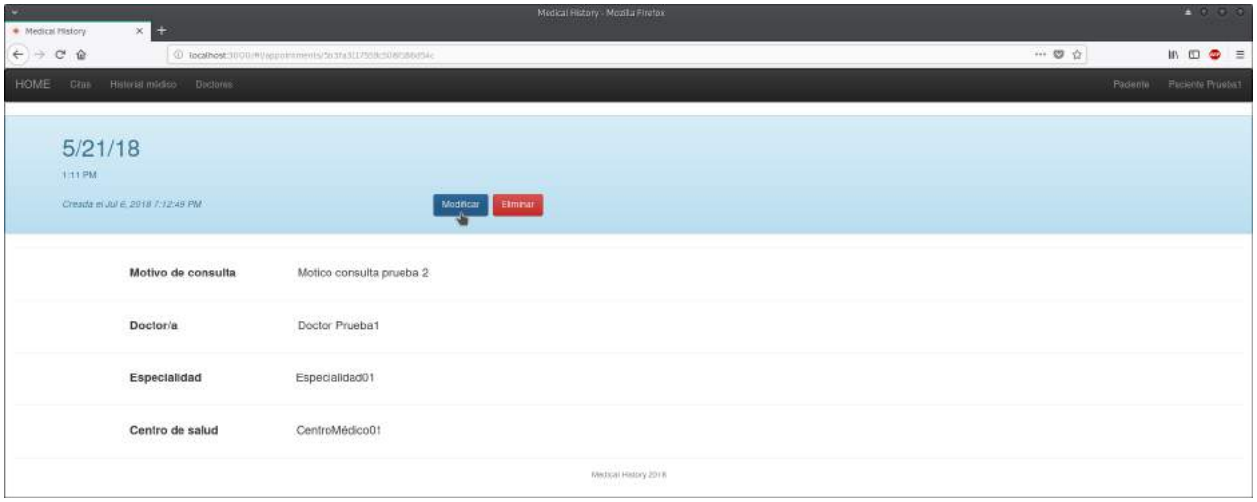

Fig. 6.58. Página de detalle de cita: modificar.

Tras pulsar el botón modificar parecerá la siguiente pantalla, donde introduciremos los datos que queremos modificar.

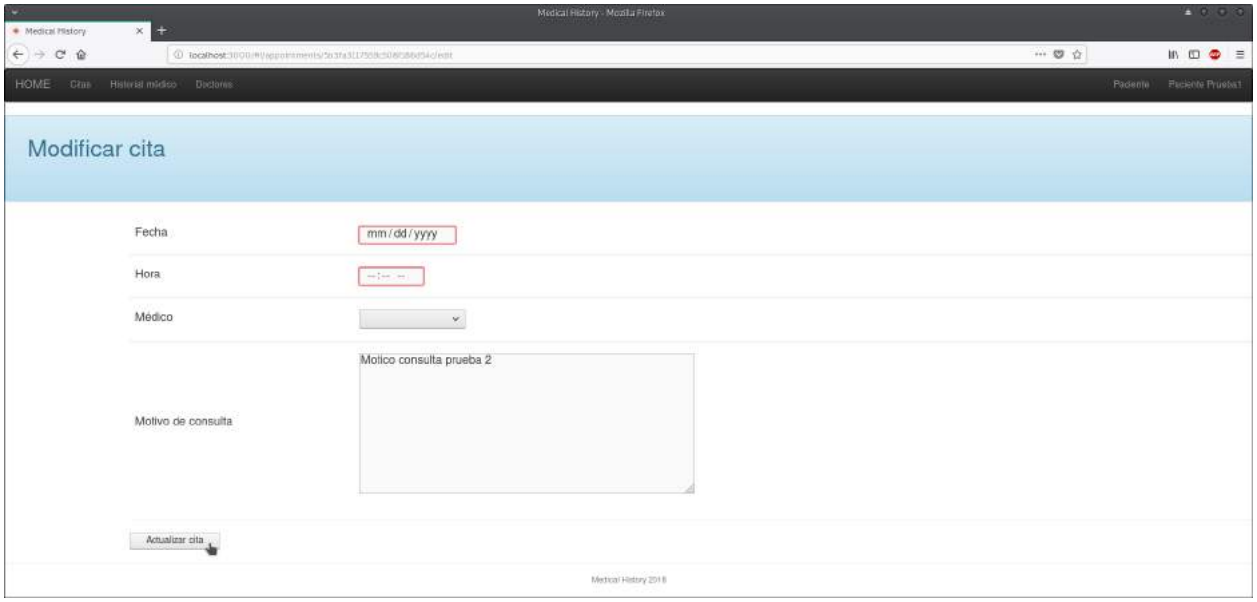

Fig. 6.59. Página de modificación de cita.

Finalmente hacemos click sobre el botón 'Actualizar cita'.

5.5.3. Cancelar cita

Para cancelar una cita accederemos a la lista de citas solicitadas (pulsando sobre 'Citas' en la barra de navegación Fig.6.53) y haciendo click sobre el botón 'Detalle'(Fig.6.56) aparecerá la pantalla de detalle de cita(Fig.6.57), que junto a la fecha de la cita y al botón de 'Modificar', encontramos el botón 'Cancelar'.

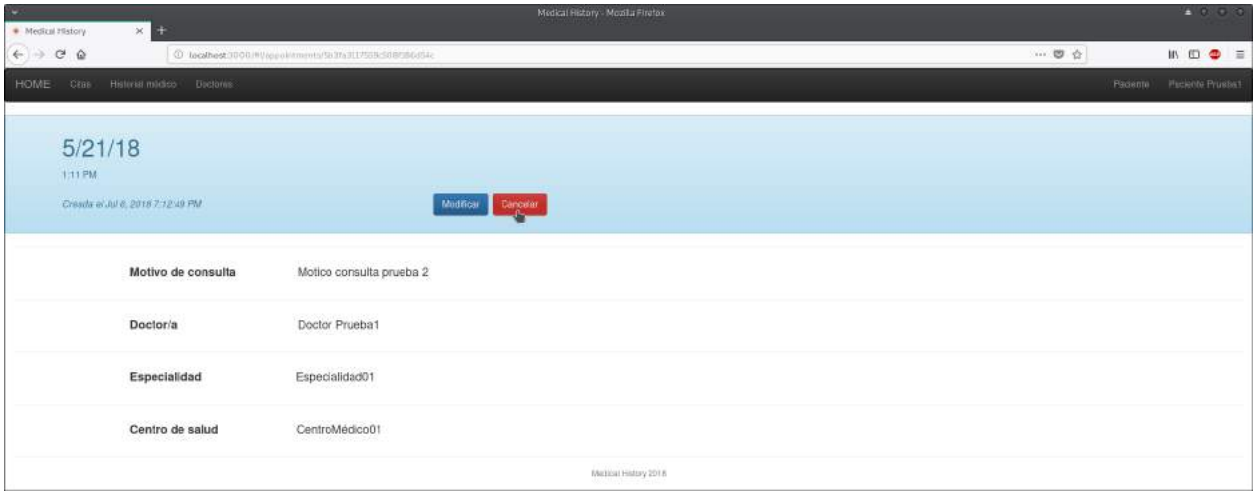

Fig. 6.60. Página de detalle de cita: cancelar.

5.6. Historial médico

Para ver la lista de historiales médicos que te han realizado trás cada cita pulsamos sobre ’Historial Médico’ en la barra de navegación.

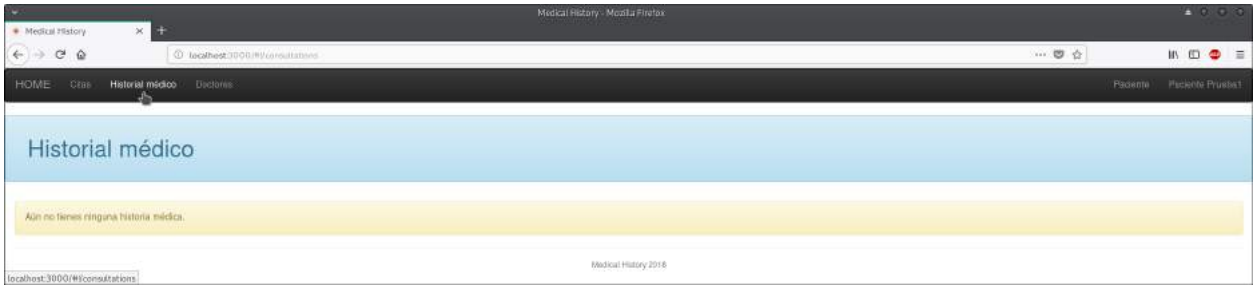

Fig. 6.61. Página de historiales médicos vacía.

Cuando ya hayas asistido a alguna consulta aparecerá la siguiente pantalla:

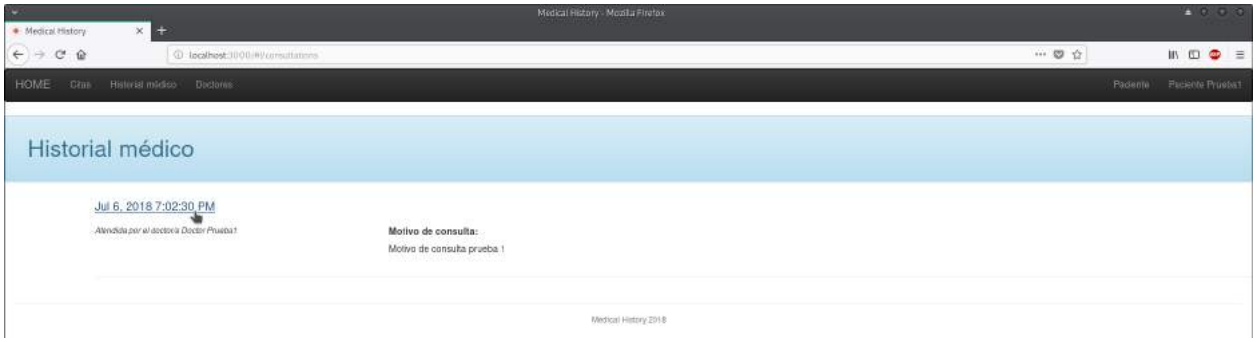

Fig. 6.62. Página de historiales médicos.

Si hacemos click sobre la fecha en que fue creada la historia médica nos mostrará sus detalles.

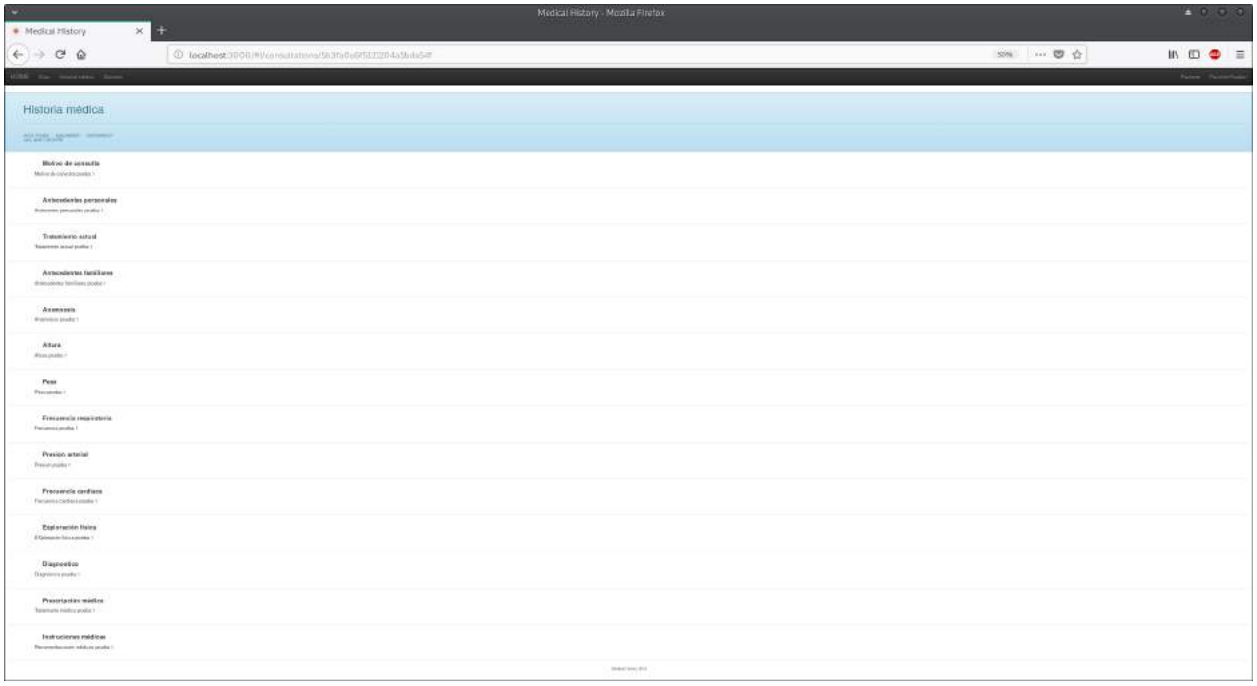

Fig. 6.63. Página de detalle de historia médica.

7. PALABRAS CLAVE (KEYWORD)

Palabras clave:

- HCE: siglas de Historia Clínica Electrónica. Recopilación sistematizada de información de salud almacenada electrónicamente de pacientes y población en un formato digital.
- CRUD: siglas de Create, Read, Update y Delete. Funciones básicas en bases de datos o la capa de persistencia.
- API: conjunto de funciones y procedimientos que permiten la creación de aplicaciones que acceden a las características o datos de un sistema operativo, aplicación u otro servicio.
- REST: interfaz entre sistemas que utiliza directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos.
- JSON: formato que permite la clasificación e intercambio de datos.
- HTTP: protocolo de comunicación que permite las transferencias de información en la World Wide Web.
- Middleware: software que conecta componentes de software o aplicaciones para que puedan intercambiar datos entre éstas. Es utilizado a menudo para soportar aplicaciones distribuidas. Esto incluye servidores web, servidores de aplicaciones, sistemas de gestión de contenido y herramientas similares.
- MEAN: siglas de MongoDB, Express, Angular y NodeJS. Se define com un framework o conjunto de subsistemas de software para el desarrollo de aplicaciones y páginas web dinámicas basadas cada una de estas en JavaScript.
- NPM: manejador de paquetes por defecto para NodeJS.
- SCRUM: metodología ágil para el desarrollo de software o la gestión de proyectos.
- Sprints: ritmos de trabajo breves en los que se dividen los proyectos bajo la metodología Scrum.

Keywords:

- EMR/EHR: acronym of Electronic Medical Record and Electronic Health Record. This is the systematized collection of patient and population electronically-stored health information in a digital format.
- CRUD: acronym of Create, Read, Update y Delete. Basic functions in databases or the persistence layer.
- API: a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.
- REST: interface between systems that directly use HTTP to obtain data or indicate the execution of operations on the data.
- JSON: format that allows the classification and exchange of data.
- HTTP: communication protocol that allows information transfers on the World Wide Web.
- Middleware: software that connects software components or applications so that they can exchange data between them. It is often used to support distributed applications. This includes web servers, application servers, content management systems and similar tools.
- MEAN: acronym of MongoDB, Express, Angular y NodeJS. It is defined as a framework or set of software subsystems for the development of applications and dynamic web pages based on each of these in JavaScript.
- NPM: default packet handler for NodeJS.
- SCRUM: agile methodology for software development or project management.
- Sprints: brief work rhythms in which projects are divided under the Scrum methodology.