Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Gestión de tratamientos y técnicas de fisioterapia y osteopatía sobre pacientes

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Albert Torres Badia

Tutor: Manuela Albert Albiol, Victoria Torres Bosch

2019-2020

Resumen

El objetivo del presente TFG es dar soporte a profesionales de la fisioterapia y la osteopatía en su actividad diaria como de consulta a pacientes. Dicho soporte consiste en una plataforma web que les permita tanto introducir información y datos sobre los tratamientos y/o técnicas realizadas a sus pacientes como consultar los datos introducidos por otros profesionales en dicha plataforma. Con estos se consigue que los profesionales de la salud puedan mantenerse actualizados respecto a nuevos tratamientos de forma sencilla; y por lo tanto puedan ofrecer un servicio de mayor calidad a sus pacientes.

En cuanto a la implementación, se ha escogido el framework MVC Ruby on Rails que permite un fácil desarrollo y mantenimiento del software. Esto lo consigue gracias a una sintaxis legible y una documentación fácil de entender.

Palabras clave: Ruby on Rails, sitio web, patrón MVC, fisioterapeuta, paciente, tratamiento, técnica.

Abstract

The goal of this end-of-degree project is providing an assistance tool for physiotherapists and osteopaths in their daily activities regarding patient consultation. This is done through a website that allows them to input information about treatments and techniques applied to specific patients, as well as checking data introduced by other professionals. In this fashion, healthcare professionals can keep themselves up to date with the latest treatments, allowing them to offer a quality service for their patients.

In regards to the implementation, the framework of choice is MVC Ruby on Rails, which simplifies both development and maintenance thanks to its readable syntax and an understandable documentation.

Keywords: Ruby on Rails, website, MVC pattern, physiotherapist, patient, treatment, technique.

Resum

L'objectiu d'aquest TFG és donar suport als professionals de la fisioteràpia y l'osteopatia en la seua activitat diària de consulta de pacients. Aquest suport consisteix en un lloc web que els permet tant introduir informació sobre els tractaments i tècniques aplicats als seus pacients com consultar les dades introduïdes per altres professionals usuaris de la plataforma. D'aquesta manera, s'aconsegueix que els professionals de la salut puguen mantindre's actualitzats respecte als últims tractaments de forma senzilla; i per tant puguen oferir un servei de major qualitat als seus pacients.

En quant a la implementació, s'ha escollit el framework MVC Ruby on Rails, que permet un fàcil desenvolupament i manteniment del software. Açò ho aconsegueix gràcies a una sintaxi llegible i una documentació fàcil d'entendre.

Paraules clau: Ruby on Rails, lloc web, patró MVC, fisioterapeuta, pacient, tractament, tècnica.

Tabla de contenidos

1. Introducción

1.1 Motivación

Basarse en el trabajo realizado por otros profesionales, y tomar éste como inspiración es una forma bastante adecuada de mejorar en muchos sectores profesionales también en el de la fisioterapia que es en el que se centra este trabajo final de grado (TFG).

Aunque existen plataformas para compartir información sobre fisioterapia y osteopatía, estas no están orientadas a profesionales sino a un público más general. Esto, junto con el problema de cómo lidiar con datos personales de los pacientes dificulta que se comparta información entre profesionales de este sector.

En el presente TFG se propone desarrollar un sitio web que dé respuesta a esta necesidad de compartir información entre profesionales del sector de la fisioterapia y osteopatía. Es por ello por lo que esta aplicación deberá ser capaz de: recopilar información sobre los tratamientos y técnicas aplicados a pacientes por fisioterapeutas y osteópatas, guardar dicha información de una manera estructurada y mostrar datos extraídos sobre la misma, permitiendo filtrarla según el criterio del usuario. En cuanto al problema de la privacidad de los datos de los pacientes cabe destacar que, aunque se recopilan sus datos, no se mostrará información sensible de los mismos.

El objetivo de esta aplicación es que cualquier profesional del sector registrado en ella tenga acceso al conocimiento y experiencia de otros profesionales, basándose en su experiencia con pacientes reales. Con esto se pretende que puedan tratar y diagnosticar mejor a sus propios pacientes.

1.2 Objetivos

Específicos

- Construir una aplicación web para gestionar usuarios, crear pacientes, crear diagnósticos y crear tratamientos. Así como mostrar tratamientos y su efectividad.
- Crear una base de datos usando el patrón de diseño active record en la que se guarda la información de pacientes, osteópatas, visitas, diagnósticos y tratamientos.

De aprendizaje

- Demostrar la capacidad del alumno de aplicar lo aprendido durante el grado en un proyecto real.
- Aprender la arquitectura modelo-vista-controlador y como ésta se concreta en código ejecutable.
- Aprender el uso de un framework desde cero, concretamente Ruby on Rails.

1.3 Estructura de la memoria

La memoria se va a estructurar en nueve bloques principales:

1. Introducción:
   Se trata del apartado actual, donde se hace una primera aproximación al proyecto, explicando que problemas pretende solucionar y cuáles son los objetivos planteados.
2. Estado del arte:
   En esta sección se analiza los productos del mercado ofrecen un servicio similar al que queremos ofrecer, así como cuáles son sus puntos fuertes y débiles y en qué aspectos porta algo diferente el software desarrollado en el presente Trabajo de Fin de Grado (TFG).
3. Contexto tecnológico:
   En esta sección se explican las herramientas de software utilizados durante el desarrollo, haciendo hincapié en Ruby on Rails y Ruby así como otras tecnologías web. También se comenta brevemente otras herramientas software.
4. Metodología:
   En la siguiente sección se describe la metodología utilizada durante el proyecto, concretamente el modelo en cascada.
5. Análisis de requisitos:
   Esta sección contiene el diagrama de casos de uso del sitio web desarrollado en el presente TFG.
6. Diseño:
   En esta sección se muestra los bocetos del sitio web así como el diagrama de clases a partir del cual se ha creado la base de datos.
7. Implementación:
   En esta sección se describe cómo funciona cada una de las interfaces y como ha sido implementada.
8. Conclusiones:
   En esta sección analizamos en qué medida se han cumplido los objetivos planteados, así como posibles mejoras en el producto y el proceso de creación del mismo.

2. Estado del arte

En este punto se analizan algunas de las alternativas que existen para solucionar problemas parecidos al que queremos resolver y se exponen algunas de las limitaciones que se considera que presentan. Finalmente, se explica en qué aspectos se va a diferenciar y/o mejorar el sitio web tratado en el presente TFG. De esta forma podemos saber en qué aspectos podemos aportarle valor al usuario.

2.1 Redes sociales

Existen multitud de cuentas en las redes sociales como YouTube o Instagram que hablan sobre fisioterapia. No obstante, estos suelen estar enfocados a un público más general y suelen centrase más en ejercicios de rehabilitación, en lugar de en técnicas aplicables en una consulta. Además, no suelen tener la información concreta y consultable sobre la efectividad de estos tratamientos en pacientes reales.

Algunas de estas cuentas son los canales de YouTube como Fisioterapia a tu alcance y FisioOnline[1] entre otros. En cuanto a cuentas en otras redes sociales podríamos mencionar cuentas de Instagram como strengthcoachtherapy, dr.jacob.harden, msk_physio_london[2]...

2.2 Foros de fisioterapia

Existen multitud de foros de fisioterapia. En este caso hablaremos sobre el rincondefisioterapia[3].

Cuando entramos en la ventana principal (ver Página principal del foro Rincondefisioterapia) se puede ver que existen diversos enlaces que agrupan los diferentes temas relacionados con un mismo aspecto de la fisioterapia.

[1] Se pueden visitar estos dos canales de YouTube en las URLs https://www.youtube.com/user/Fisitoterapialcance y https://www.youtube.com/user/MrFisiotube respectivamente.

[2] Se pueden visitar estas cuetas de Instagram en las URLs https://www.instagram.com/strengthcoachtherapy/?hl=es, https://www.instagram.com/dr.jacob.harden/?hl=es y https://www.instagram.com/msk_physio_london/?hl=es respectivamente.

[3] Se puede visitar el rincondefisioterapia en la URL https://rincondefisioterapia.mforos.com/

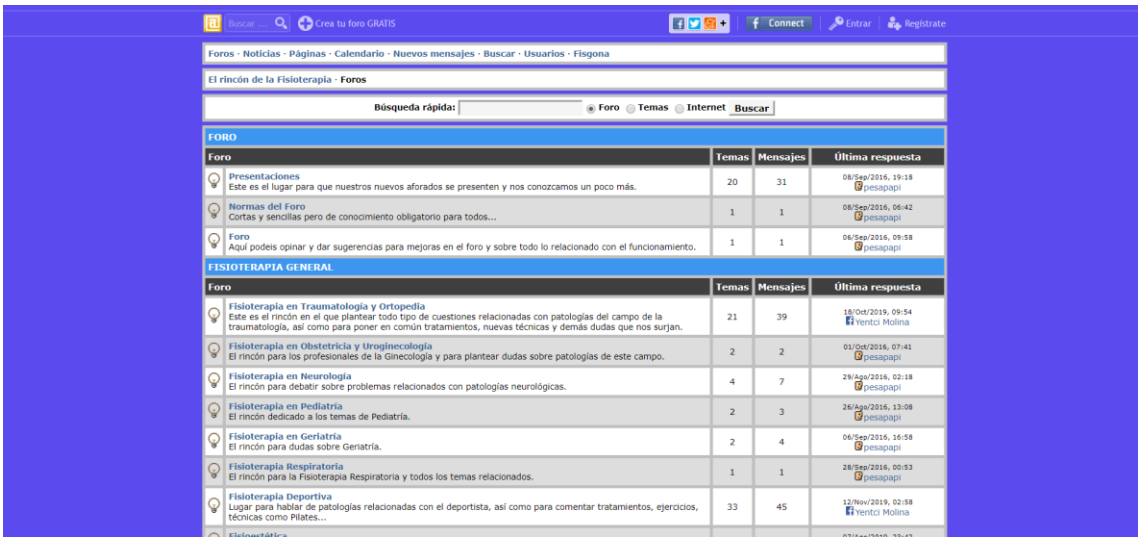

Figura 1 Página principal del foro Rincondefisioterapia

Figura 2 Hilo de conversación del foro Rincondefisioterapia

Si bien en este foro se habla sobre fisioterapia, suele estar enfocado más al paciente que suele abrir un tema planteando un problema de salud que le ha surgido y del cual cree que se puede beneficiar de un tratamiento de fisioterapia.

Además es bastante habitual encontrar temas con una interacción pobre por parte de los usuarios, tal y como se puede ver en la imagen:

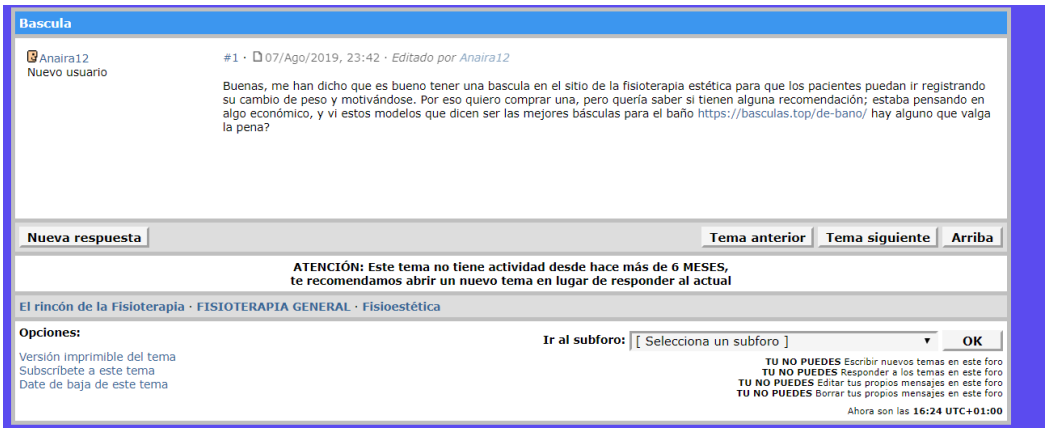

Figura 3 Tema de conversación del foro Rincondefisioterapia

Para finalizar cabe destacar que si bien la interfaz de usuario no resulta difícil de utilizar puede resultar poco atractiva visualmente.

2.3 Crítica al estado del arte y propuesta

Como hemos visto en las alternativas estudiadas anteriormente, estas están enfocadas más a la divulgación al público general. Es en este aspecto donde nuestra propuesta aporta valor diferencial ya que está enfocado a fisioterapeutas y profesionales de la salud.

Además nuestra aplicación aporta datos sobre el contexto y la efectividad de los tratamientos aplicados, guardándose estos de forma estructurada. Con ello se permite acceder a ellos de forma automática pudiéndose incluso inferir estadísticas o filtrar los tratamientos en función de su efectividad.

3. Contexto tecnológico

En esta sección describiremos el contexto tecnológico, esto es, la arquitectura y las herramientas software utilizadas.

3.1 Arquitectura modelo-vista-controlador (MVC)

Modelo-vista-controlador o **MVC** (Tutorials Point (I) Pvt. Ltd., 2015) es un patrón de diseño que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Tiene tres componentes distintos que se relatan a continuación:

- **Modelo:** es la representación de los datos del sistema.
- **Controlador:** invoca al modelo cuando éste es requerido y responde a eventos (por ejemplo por alguna acción del usuario). También puede interactuar su vista asociada. Por lo tanto se podría decir que ha de intermediario entre la vista y el modelo.
- **Vista:** muestra los datos al usuario.

Cabe destacar que, aunque MVC fue desarrollado originalmente para aplicaciones de escritorio, ha sido ampliamente adaptado para implementar aplicaciones web. Existen una gran variedad de frameworks que implementan este patrón; diferenciándose entre ellos en la interpretación de como las funciones MVC se dividen entre cliente y servidor.

3.2 Ruby on Rails

Para hacer el proyecto se ha usado principalmente el framework de Ruby on Rails así como algunas gemas:

Ruby on Rails (Sitio web oficial Ruby on Rails) es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, que sigue el patrón Modelo Vista Controlador. Su principal baza es que permite escribir un código más sencillo que otros frameworks sin dejar de ser apropiado para crear aplicaciones relativamente grandes.

El lenguaje de programación Ruby permite una sintaxis en el framework que muchos de sus usuarios encuentran muy legible. Rails hace uso de las RubyGems, que es el formato oficial de paquete para Ruby.

En el proyecto se han usado varias gemas, que es como se denominan a las librerías en Ruby. En concertó las siguientes:

- **Devise:** Se trata de una gema para Ruby on Rails que sirve para la autenticación de usuarios. Concretamente en el proyecto ha permitido crear la ventana para hacer log in y registrarse así como guardar la contraseña de los osteópatas encriptada en la base de datos.
- **Bootstrap**: hemos incluido de Bootstrap en el Gemfile (que es un fichero donde se guardan todas la gemas de las que depende Rails) ya que, aunque no es obligatorio, es bastante recomendable pues de esta manera las librerías están instaladas de forma local en el servidor. Se habla de este

framework en la sección Lenguajes de programación y tecnologías web utilizadas.

La información para estas dos gemas se ha obtenido de (Documentación oficial Devise) y (Documentación gema Bootstrap GitHub) respectivamente.

Al usar Ruby on Rails resulta obvio que se ha programado gran parte del código en **Ruby**. Este es un lenguaje de programación interpretado, reflexivo y orientado a objetos creado por Yukihiro Matsumoto. Su sintaxis está inspirada en Python y Perl con características de programación orientada a objetos similares a Smalltalk. Su implementación oficial es distribuida bajo una licencia de software libre.

Ruby es un lenguaje muy amigable, diseñado para la productividad y diversión del programador, según las palabras de su propio creador. Ruby sigue el *principio de la menor sorpresa* lo cual significa que el lenguaje debe comportarse de manera previsible.

3.3 Lenguajes de programación y tecnologías web utilizadas

Además de Ruby on Rails se han usado los siguientes lenguajes y tecnologías web:

- **HTML**, siglas en inglés de HyperText Markup Language (o 'lenguaje de marcas de hipertexto'), hace referencia al lenguaje de marcado para la elaborar páginas web. Es un estándar a cargo del World Wide Web Consortium (W3C) usado principalmente para la elaboración de páginas web, permitiendo introducir contenido de diversos formatos tales como texto, imágenes, videos, juegos, etc.

  En el caso de nuestro proyecto, como se ha hecho uso de Rails, se ha utilizado eRuby o Embedded Ruby (Documentación oficial Embedded Ruby). Este es un sistema similar ASP, JSP o PHP en el cual se mezcal código Ruby con HTML. Por ello, se ha usado principalmente en la vista de la aplicación. Los ficheros de eRuby se guardan con la extensión html.erb.

- **CSS** (siglas en inglés de Cascading Style Sheets) es un lenguaje para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Por lo tanto el lenguaje puede ser aplicado a cualquier documento XML. En el caso de nuestro proyecto se ha usado para definir el estilo de los documentos HTML.
- **JavaScript** es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

  Aunque JavaScript se puede usar del lado del servidor en el proyecto se ha utilizado en la parte del cliente. Ha sido necesario para que algunas interfaces de usuario resultaran más dinámicas.

- **Bootstrap** es una biblioteca multiplataforma de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas con diferentes elementos de diseño basado en HTML y CSS. También contiene extensiones de JavaScript. Bootstrap solo se ocupa del desarrollo front-end.

La información sobre HTML, CSS y JavaScript se ha obtenido de (Gauchat, 2012) la información sobre Bootstrap se ha obtenido de (Lett, 2018)

3.4 Entorno de desarrollo

Como entorno de desarrollo se ha usado **Visual Studio Code**. Éste es un editor de código desarrollado por Microsoft para Windows, Linux y macOS. Entre sus funcionalidades destacan: soporte para la depuración de código, control integrado con Git, resaltado de sintaxis, finalización inteligente de código... Es personalizable, permitiendo a los usuarios cambiar los atajos de teclado y las preferencias entre otros. Es gratuito y de código abierto.

Además fue necesario agregar el plugin Ruby Peng Lv a Visual Studio Code para que resaltara el código eRuby.

3.5 Otras herramientas

La herramienta de control de versiones empleada ha sido **Git** (Decripción caractericas e historia de Git), diseñado y distribuido por Linus Torvalds. Algunas de sus características son:

- Fuerte apoyo al desarrollo no lineal, destacando la rapidez en la gestión de ramas y mezclado de diferentes versiones; incluyendo herramientas para navegar y visualizar un historial de desarrollo no lineal. Git asume que un cambio será fusionado más veces de lo que es escrito.
- Los repositorios Subversion y svk se pueden usar directamente con git-svn.
- Gestión eficiente de proyectos grandes

Para alojar el proyecto Git hemos hecho uso de GitHub. Este es una forja (plataforma de desarrollo colaborativo) para alojar proyectos. A diferencia de otras forjas solo acepta el uso de Git como sistema de control de versiones. Anteriormente los repositorios privados eran de pago. Actualmente esto ya no sucede.

StarUML es una herramienta para la creación de diagramas UML. Gracias a ella se ha creado el diagrama de clases y el diagrama de casos de uso en el proyecto.

Para hacer los prototipos (ver sección Bocetos de las interfaces gráficas) se ha usado **MockFlow**. Ésta es una herramienta de prototipado en la nueve que también tiene una versión de pago. En el caso del presente TFG se ha usado la versión gratuita.

4. Metodología

4.1 Introducción

En el proyecto tratado en el presente TFG se ha adoptado una metodología de trabajo en cascada (Álvarez, 2015). Esta metodología divide el desarrollo del producto en las etapas del proceso para el desarrollo de software, siendo necesario acabar la etapa anterior para empezar con la siguiente.

4.2 Fases del modelo en cascada

Las fases en las que se suele dividir el trabajo en la metodología en cascada son, en orden temporal, las siguientes:

1. **Análisis de requisitos:** en esta fase se analizan las necesidades de los usuarios finales del software para determinar qué objetivos debe cubrir. El objetivo de esta etapa es obtener todos los requisitos del sistema. Cabe destacar que no se pueden pedir nuevos requisitos en fases posteriores del desarrollo.
2. **Diseño:** en esta etapa principalmente se decide cual va a ser la arquitectura a utilizar y se crean los prototipos.
3. **Implementación:** en esta etapa se escribe el código necesario para implementar los prototipos.
4. **Testing:** los elementos, ya programados, se ensamblan para componer el sistema y se comprueba que funciona correctamente. Se buscan errores en el programa y se corrigen antes de entregarlo al usuario final.
5. **Validación y Verificación:** Es la fase en donde el usuario final ejecuta el sistema, y se asegura que cubra sus necesidades.
6. **Mantenimiento**: en esta etapa el usuario/usuarios finales ya usan el sistema. Es bastante común que el software no cumpla con todas nuestras expectativas y sea necesario añadir nuevas funcionalidades.

4.3 Ventajas e inconvenientes y justificación de su uso

En esta sección describiremos cuales son las ventajas y desventajas de esta metodología, así como porque se ha usado esta metodología.

En cuanto a las ventajas de esta metodología destacan:

- Es un modelo sencillo, fácil de implementar y entender.
- Es un modelo conocido y utilizado con frecuencia.
- Promueve una metodología de trabajo efectiva: definir antes que diseñar, diseñar antes que codificar.

En cuanto sus inconvenientes son los siguientes:

- En la vida real, un proyecto rara vez sigue una secuencia lineal, esto va en contra de la propia esencia de esta metodología de trabajo.
- El proceso de creación del software tarda mucho tiempo ya que se realiza cada una de las fases de desarrollo para la totalidad del sistema y este no está finalizado hasta que se llega a la fase de testing.
- Cualquier error de diseño detectado en la etapa de testing conduce necesariamente al rediseño y a volver a programar del código afectado, aumentando los costos del desarrollo.

- Es obligatorio terminar una etapa del desarrollo para continuar con la siguiente.

A pesar de los inconvenientes enumerados, se ha decidido adoptar esta metodología de trabajo porque el software a implementar era un sistema bastante cerrado, en el que era poco probable que fuera necesario implementar nuevas funcionalidades en etapas posteriores a la de análisis de requisitos. A esto se le suma la ventaja de que es una metodología fácil de entender e implementar, por lo tanto, no es necesario invertir mucho tiempo en el aprendizaje de la misma.

5. Análisis de requisitos

En esta sección mostraremos y explicaremos el diagrama de casos de uso. Como fuente bibliográfica sobre cómo hacer un diagrama de casos de uso se escogió (Tutorial dirargamas UML Sparxsystems).

5.1 Diagrama de casos de uso

A continuación, se muestra las diferentes acciones que puede llevar a cabo un osteópata con la aplicación a desarrollar.

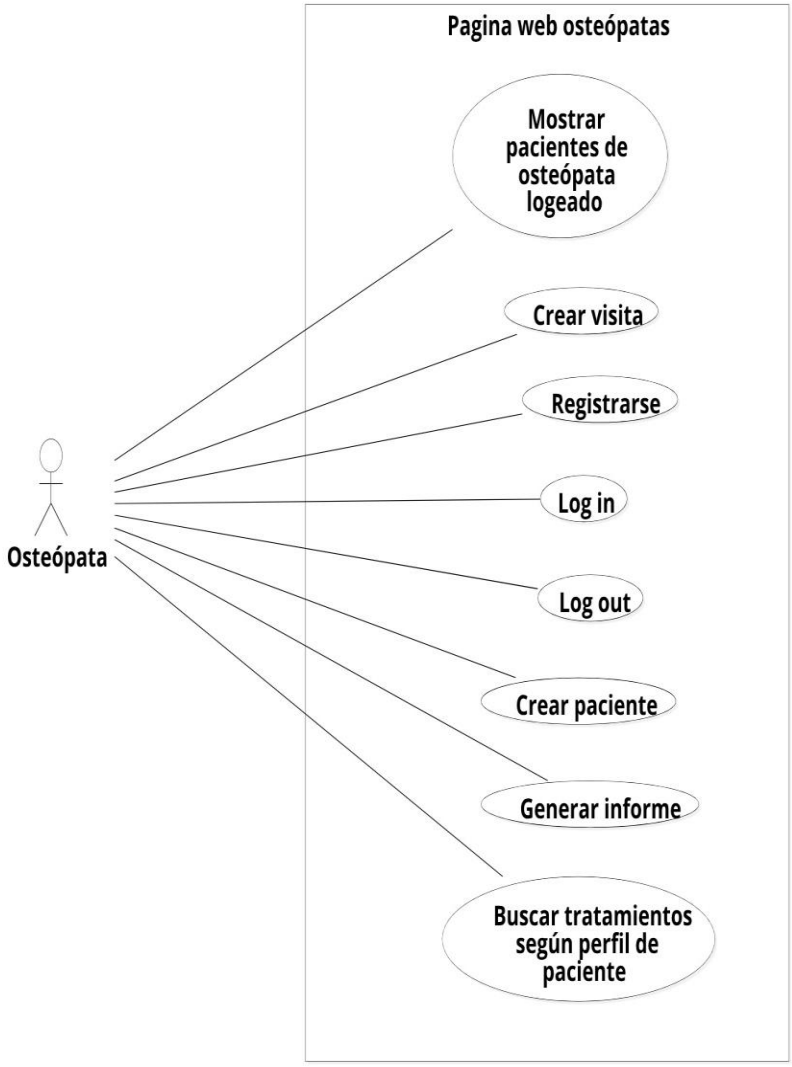

Figura 4 Diagrama de casos de uso general

Estos casos de uso dan lugar a la siguiente lista de requisitos funcionales:

ID RF	Descripción
RF1-Reg	Permite a los osteópatas crear un usuario en la web. Debe introducir su nombre, apellidos, un email y una contraseña. La contraseña, se guarda en la encriptada en la base de datos por motivos de seguridad. En cuanto al email, debe ser único entre todos los usuarios.

RF2-Log_in	Permite a los osteópatas acceder a su cuenta introduciendo su email y contraseña. Se deben haber registrado previamente.
RF3-Log out	Permite a los osteópatas salir de su cuenta de usuario. Deben haber hecho log in previamente.
RF4-Crear_visita	Crea una visita. Se debe introducir el paciente, el osteópata, la anamnesis y la escala de dolor (una valoración subjetiva del 0 al 10 de cuanto dolor presenta el paciente).
RF5-Mostrar_pacientes	Muestra todos los pacientes del osteópata que esta logeado.
RF6-Buscar_tratamientos	Busca en la base de datos los tratamientos aplicados a cada paciente para una patología concreta.
RF7-Crear_paciente	Crea un paciente en la base de datos.
RF8-Generar_informe	Se genera un informe con los datos recogidos en la plataforma. Estos pueden ser de diversa índole, como por ejemplo: técnicas realizadas con mayor frecuencia, técnicas más efectivas, técnicas menos efectivas, listado de técnicas posibles a aplicar dado paciente con un determinado cuadro clínico.

Tabla 1 Descripción casos de uso

- **Generar informe:** se genera un informe con los datos recogidos en la plataforma. Estos pueden ser de diversa índole, como por ejemplo: técnicas realizadas con mayor frecuencia, técnicas más efectivas, técnicas menos efectivas, listado de técnicas posibles a aplicar dado paciente con un determinado cuadro clínico.
- **Crear paciente:** crea un paciente en la base de datos.

Los casos de uso que hemos descrito incluyen y extienden algunos casos de uso. Estos se muestran a continuación junto con la descripción de los mismos.

En cuanto al caso de uso “Mostrar pacientes osteópata logeado”, se detalla en los dos siguientes casos de uso.

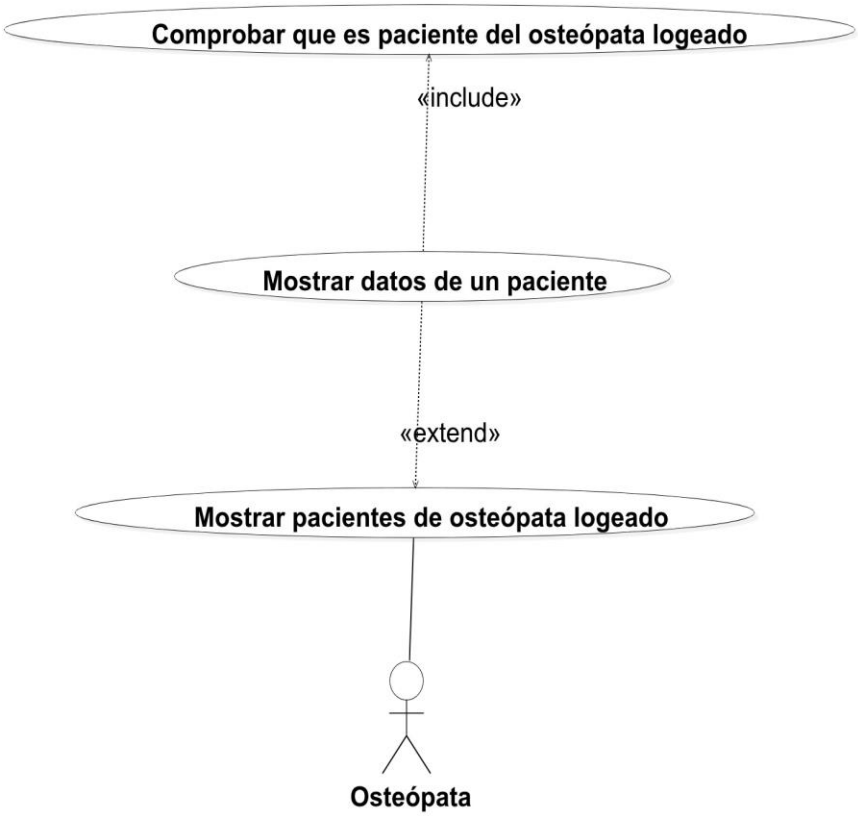

Figura 5 Casos de uso "Mostrar pacientes de osteópata logeado"

Estos casos de uso representan lo siguiente:

- **Mostrar datos de un paciente:** muestra el nombre, apellidos, sexo, edad y los antecedentes personales y familiares de un paciente. Si el paciente que se está mostrando no pertenece al osteópata que está logeado (cosa que se comprueba mediante el caso de uso "Comprobar que es paciente del osteópata logeado") no muestra el nombre y los apellidos pues se considera que estos datos pueden ser sensibles.
- **Comprobar que es paciente del osteópata logeado:** comprueba si un paciente pertenece al osteópata logeado o no.

En cuanto al caso de uso "Buscar tratamientos según perfil de paciente", se detalla en el siguiente caso de uso.

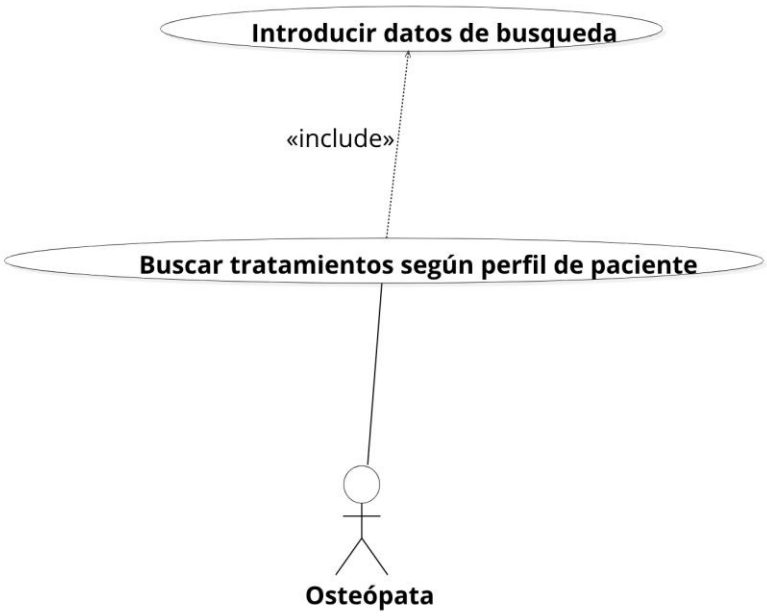

Figura 6 Casos de uso "Buscar tratamientos según perfil de paciente"

Este caso de uso representa lo siguiente:

- **Introducir datos de búsqueda:** permite al usuario introducir los datos por los que quiere filtrar los tratamientos en la base de datos.

En cuanto al caso de uso "Generara informe", se detalla en el siguiente caso de uso.

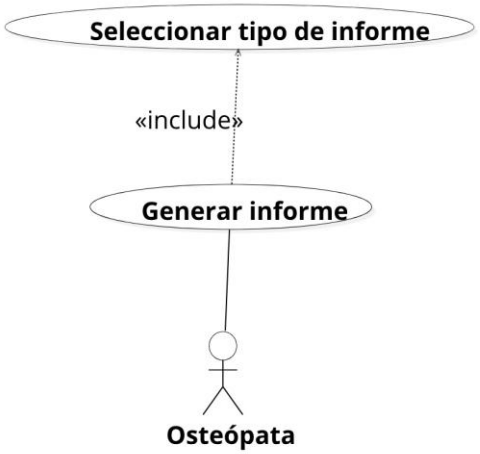

Figura 7 Casos de uso "Generar informe"

Este caso de uso representa lo siguiente:

- **Seleccionar tipo de informe:** permite al usuario seleccionar el tipo de informe de entre los existentes en la aplicación.

En cuanto al caso de uso "Crear paciente", se detalla en los dos siguientes casos de uso.

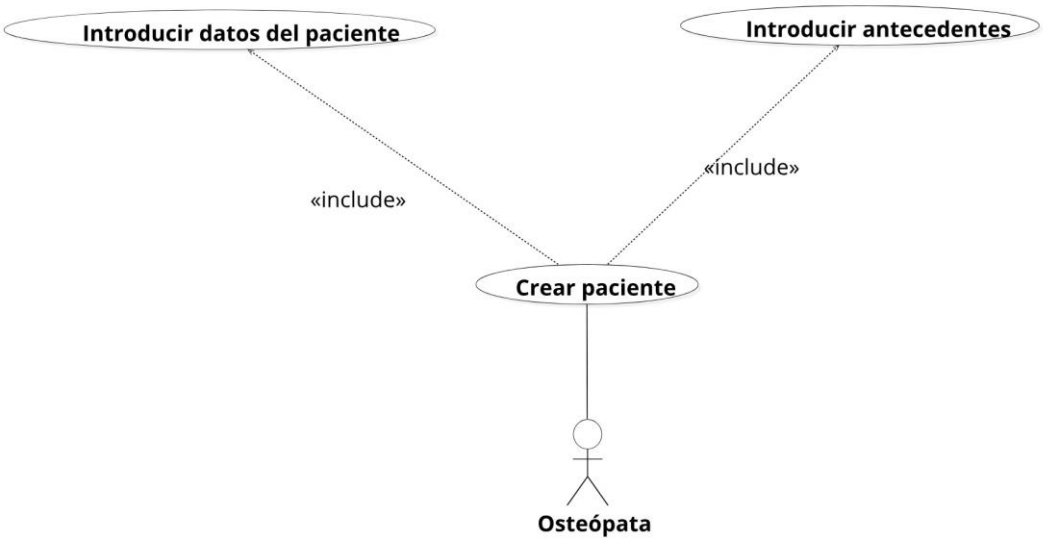

Figura 8 Casos de uso "Crear paciente"

Estos casos de uso representan lo siguiente:

- **Introducir datos del paciente**: permite introducir los datos del paciente. Estos son: el nombre, los apellidos, el año nacimiento y el sexo.
- **Introducir antecedentes:** permite introducir los antecedentes médicos del paciente. Se pueden introducir tanto antecedentes personales como familiares.

En cuanto al caso de uso "Crear visita", se detalla en los tres siguientes casos de uso.

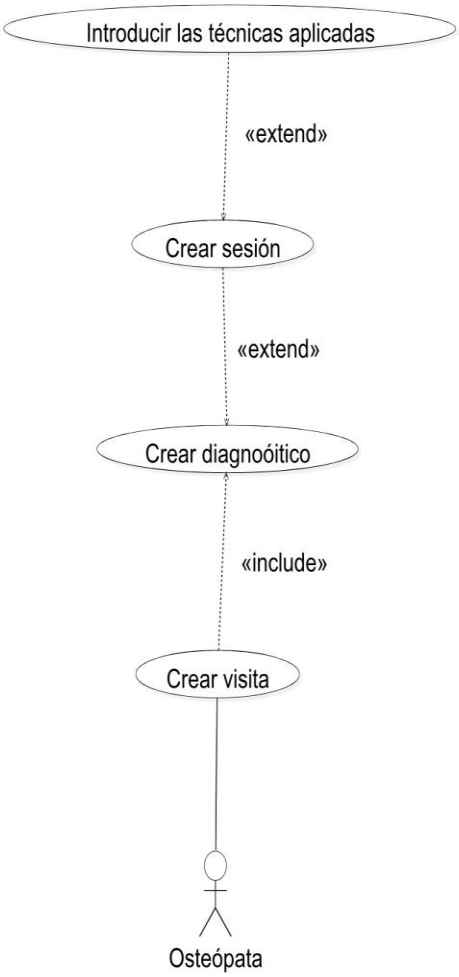

Figura 9 Casos de uso "Crear visita"

Estos casos de uso representan lo siguiente:

- **Crear diagnóstico:** crea un diagnostico; este se asocia a la vista creada mediante el casos de uso "Crear visita". Requiere introducir la enfermedad que presenta el paciente y la descripción de la misma, así como el número de sesiones de fisioterapia que se recomiendan.
- **Crear sesión:** crea una sesión de fisioterapia con la fecha, escala de dolor antes de la sesión y la escala de dolor después de la sesión. Estas dos últimas se miden con un número de 0 al 10 al igual que la escala de dolor mencionada en el caso de uso "Crear visita".
- **Introducir técnicas aplicadas:** permite añadir técnicas a la sesión introducida mediante el caso de uso "Crear sesión". Además permite modificar la fecha, escala de dolor antes de la sesión y la escala de dolor después de la sesión.

6. Diseño

En esta sección mostraremos y explicaremos los prototipos de las interfaces gráficas del sitio web (Comesaña) así como el diagrama de clases (Tutorial dirargamas UML Sparxsystems).

6.1 Arquitectura

En esta sección se explica cómo implementa Ruby on Rails la arquitectura MVC (consultar el Glosario y la sección Arquitectura modelo-vista-controlador (MVC) si no se está familiarizado con este patrón arquitectónico). Se hablará de las principales características de cada capa así como de la comunicación entre estas.

La información contenida en esta sección se ha obtenido de *El maldito libro de los descarrilados* (Herrera, 2010).

Modelo

Rails hace uso del mapeo objeto-relacional para la persistencia. Cada tabla de la base de datos tiene su propia clase Ruby, que hereda de la clase ApplicationRecord que a su vez hereda la clase ActiveRecord::Base (Documetación ofcial Ruby on Rails ActiveRecord:Base). En la clase ActiveRecord::Base están definidos unos métodos mediante los cuales se accede a la información de la base de datos.

Para crear una de las clases mencionadas en el párrafo anterior se pude hacer de diversas maneras, en el caso del proyecto tratado en el presente TFG se ha hecho siguiendo el protocolo que se muestra a continuación:

Primeramente se ejecutan un comando proporcionado por el propio framework (concretamente “rails generate model [nombre_de_la_clase]). Este comando crean unos ficheros de código Ruby (los cuales se pueden modificar manualmente de forma relativamente sencilla) en el directorio “/db/migrate”. Finalmente, solo tenemos que ejecutar el comando “rails db:migrate” para crear la clase mencionada.

Para, por ejemplo, crear una instancia de dicha clase en la base de datos debemos abrir la consola de Rails (mediante el comando “rails console”) y ejecutar la siguiente línea de código Ruby: “[nombre_de_la_clase].create()”

Controlador

El controlador hace de intermediario entre la vista y el modelo. Su función más típica es acceder a la base de datos y procesar la información, para que así pueda ser por la vista.

Existen diversas formas de crear el controlador, en el caso del sitio web creado en el presente TFG se ha hecho también mediante el comando “rails generate controller”. Este crea, entre muchos otros ficheros y directorios, el archivo del controlador con el nombre “nombre_controador_controller.rb” en el directorio “/app/controllers\patients_controller.rb”; además una carpeta con el nombre “nombre_controlador” en la ruta “app/views”.

El controlador es una clase Ruby que hereda de la clase ApplicationController. Ésta tiene definidos dentro unos métodos Ruby. Método HTTP concreto sobre una URL concreta tiene asociado un método del controlador y un archivo de la carpeta “app/views”.

En cuanto a la comunicación entre el controlador y el resto de capas se hace de la siguiente manera: el controlador se comunica con la vista a través de un atributo público en el controlador, el cual puede leer y escribir la vista. Para comunicarse con la base de datos se hace mediante los métodos ya mencionados en la sección anterior)

Vista

La capa de la vista está formada por ficheros eRuby (consultar la sección Lenguajes de programación y tecnologías web utilizadas si se tienen dudas sobre el mismo) en el directorio “app/views”. Esta contiene un directorio por cada controlador; dentro de este último directorio es donde se ubican los ficheros eRuby con extensión “.html.erb”.

6.2 Bocetos de las interfaces gráficas

En esta sección se exponen los prototipos de la vista que se hicieron al principio del proyecto tratado en el presente TFG.

Log in y registrarse

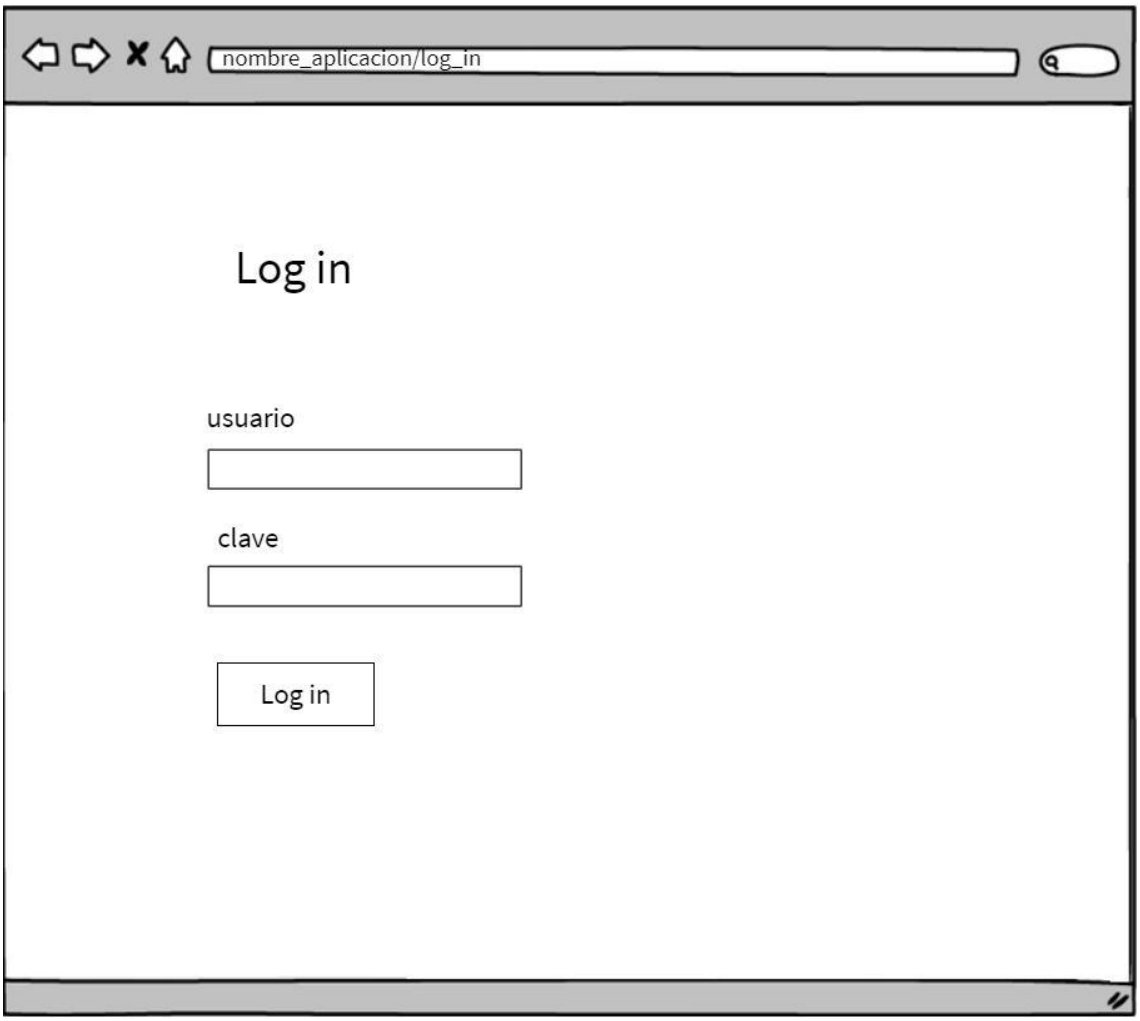

Figura 10 Boceto interfaz Log in

Figura 11 Boceto interfaz Sign up

Filtrar tratamientos

En este boceto se ha dividido en dos partes, una primera en la que el usuario introduce los datos por los que quiere filtrar y una segunda parte en la que se muestran los mismos. La primera parte abarca desde el botón de "Filtrar tratamientos" hacia arriba. En cuanto a la segunda parte, está oculta y solo se muestra una vez acciona el botón de "Filtrar tratamientos". Ésta contiene los tratamientos aplicados a los pacientes que cumplan las condiciones impuestas en los campos de la parte superior.

Figura 12 Boceto interfaz Filtrar tratamientos

Crear pacientes

En este boceto se permite crear pacientes. En la tabla se añaden los antecedentes asociados al paciente, mientras que en los campos en la parte superior se introduce el nombre, apellidos, año de nacimiento y sexo del paciente.

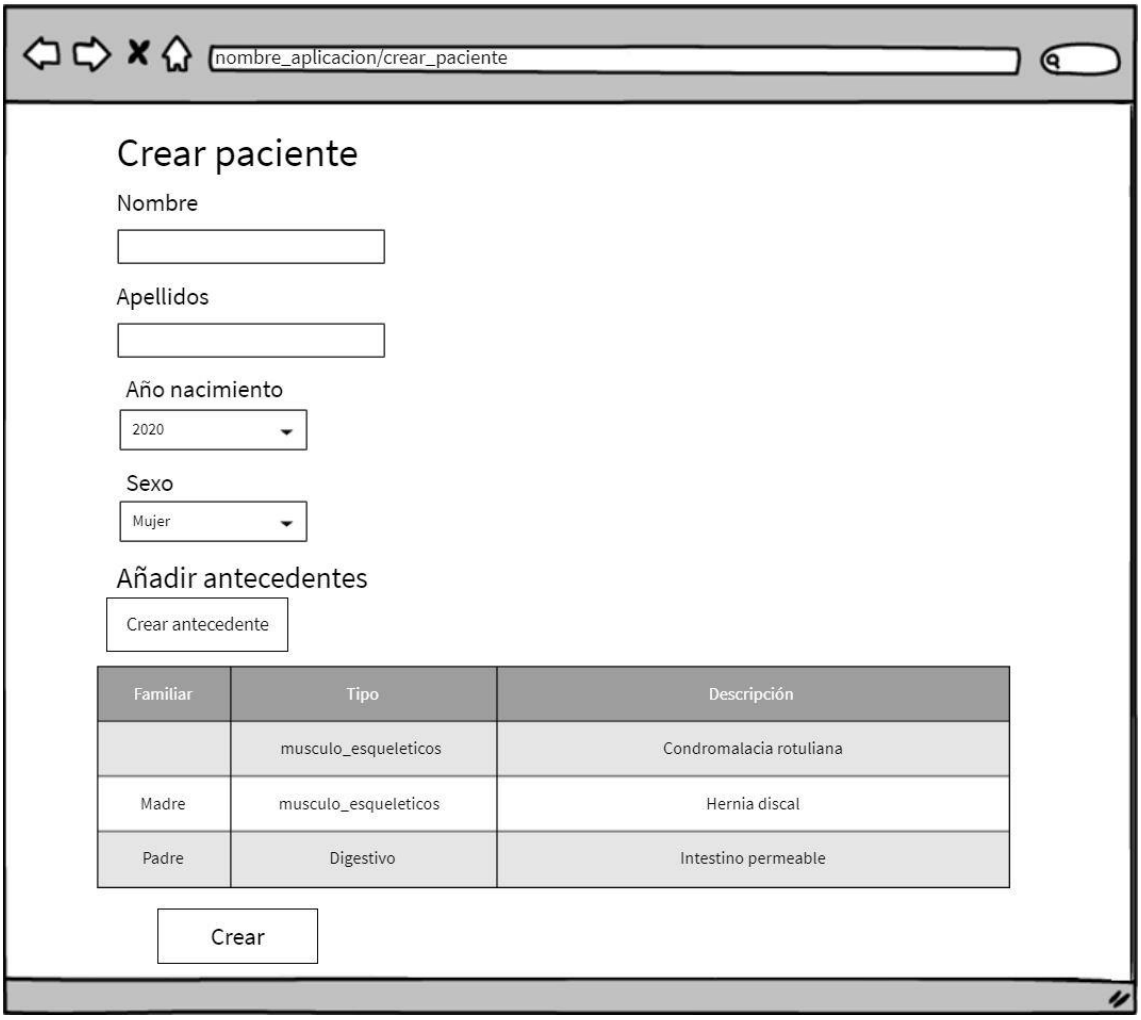

Figura 13 Boceto interfaz Crear paciente

Mostrar pacientes

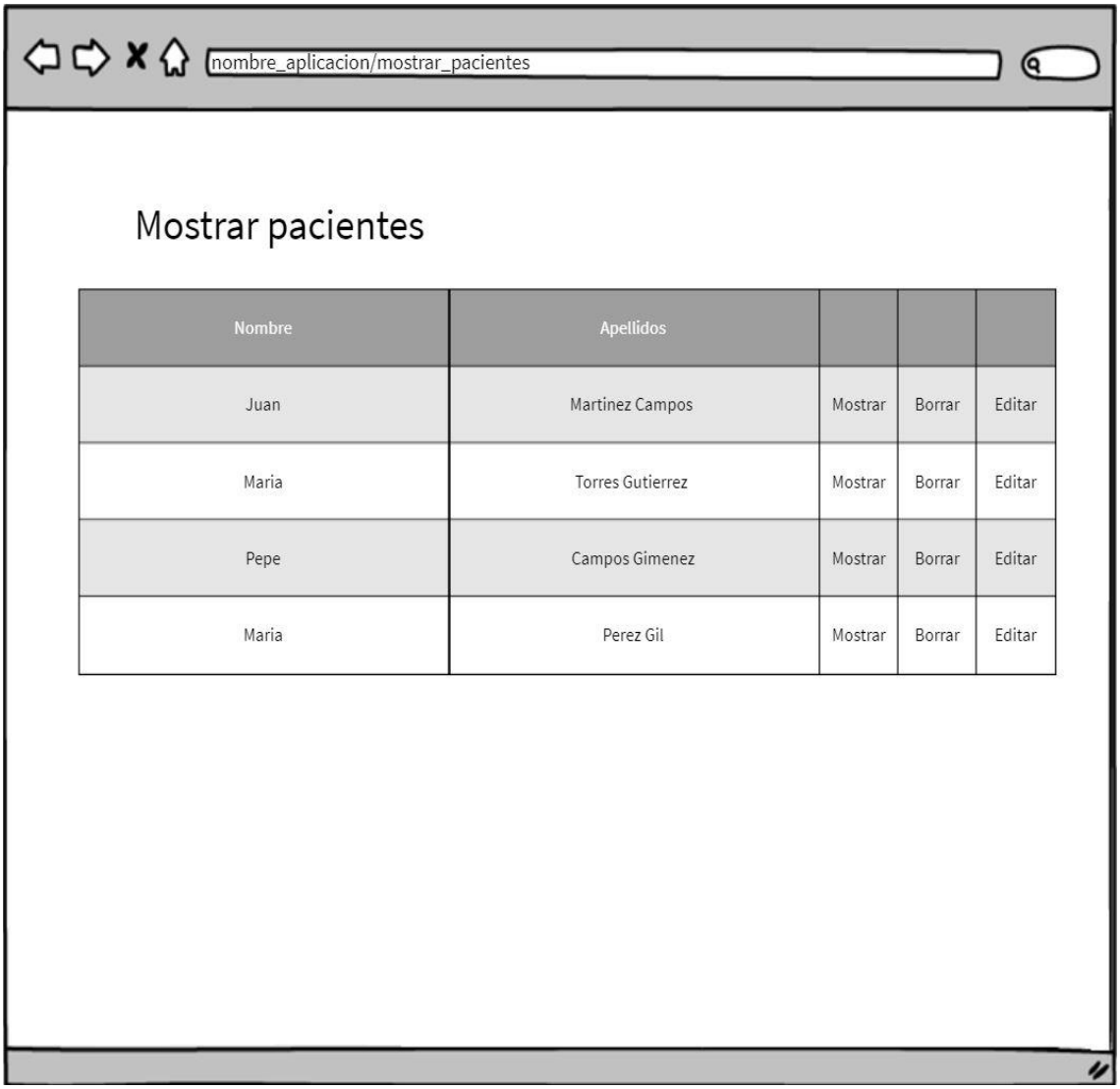

Figura 14 Boceto interfaz Mostrar pacientes

Crear visita

En este boceto además de visitas también se pueden crear diagnósticos, sesiones y los tratamientos asociados a dicha sesión. Se ha tomado esta decisión ya que no tiene sentido crear alguno de estos tres datos si no se indica la visita asociada a los mismos. Para la navegación entre ventanas se ha optado por botones de anterior y siguiente que navegan a la respectiva ventana. Las pestañas en la parte superior no son navegables, solamente indican la ventana en la que el usuario se encuentra actualmente.

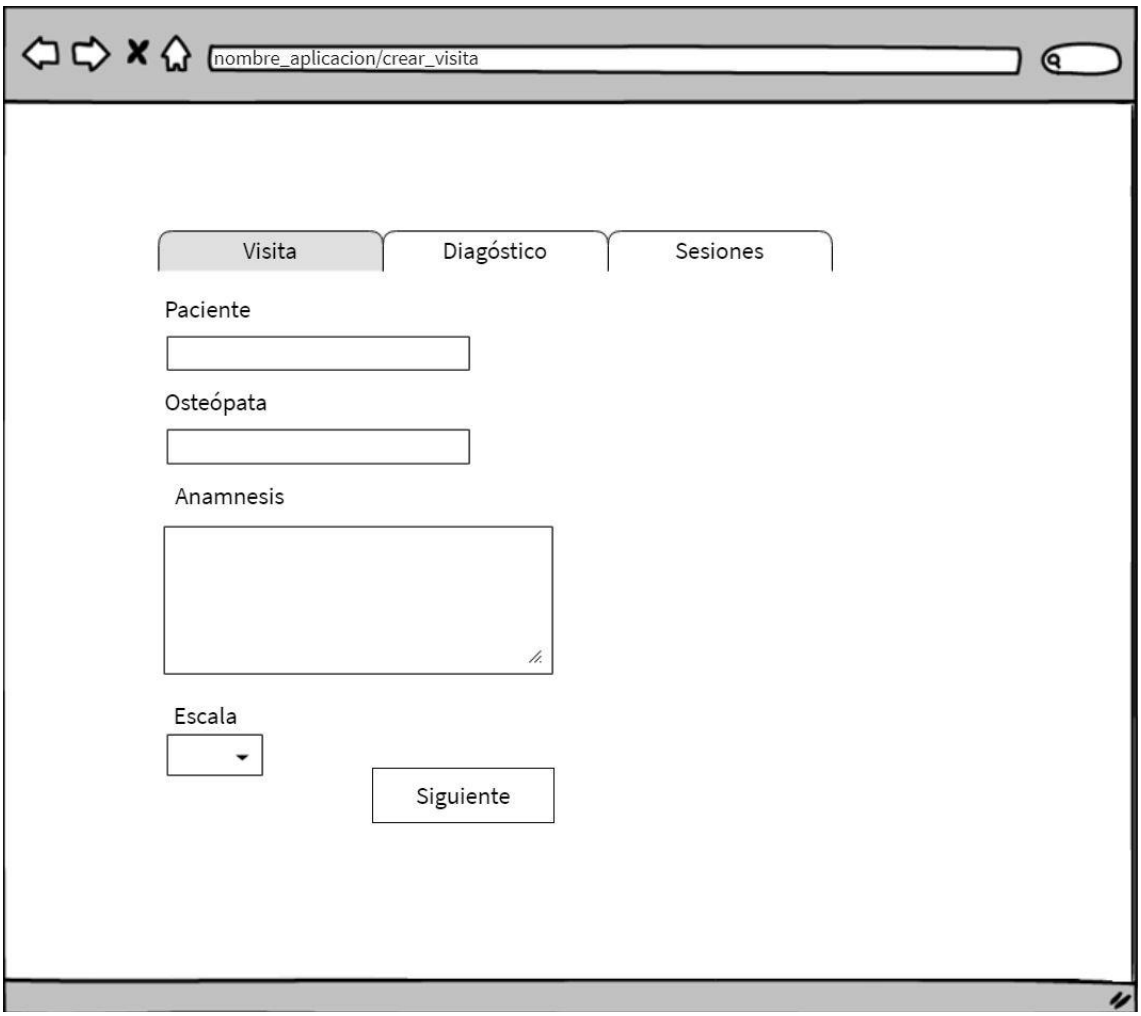

Figura 15 Boceto interfaz Crear visita pestaña visita

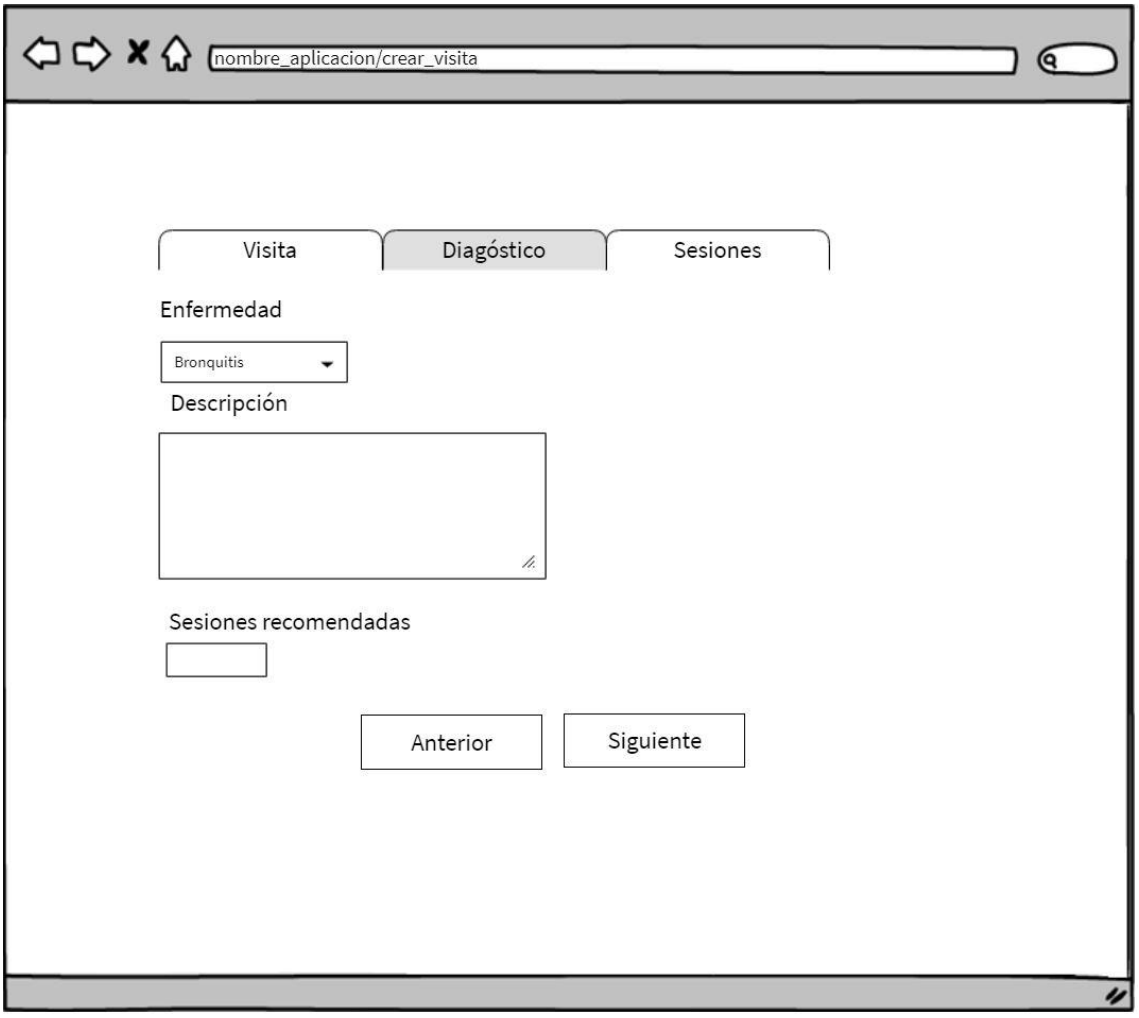

Figura 16 Boceto interfaz Crear visita pestaña diagnóstico

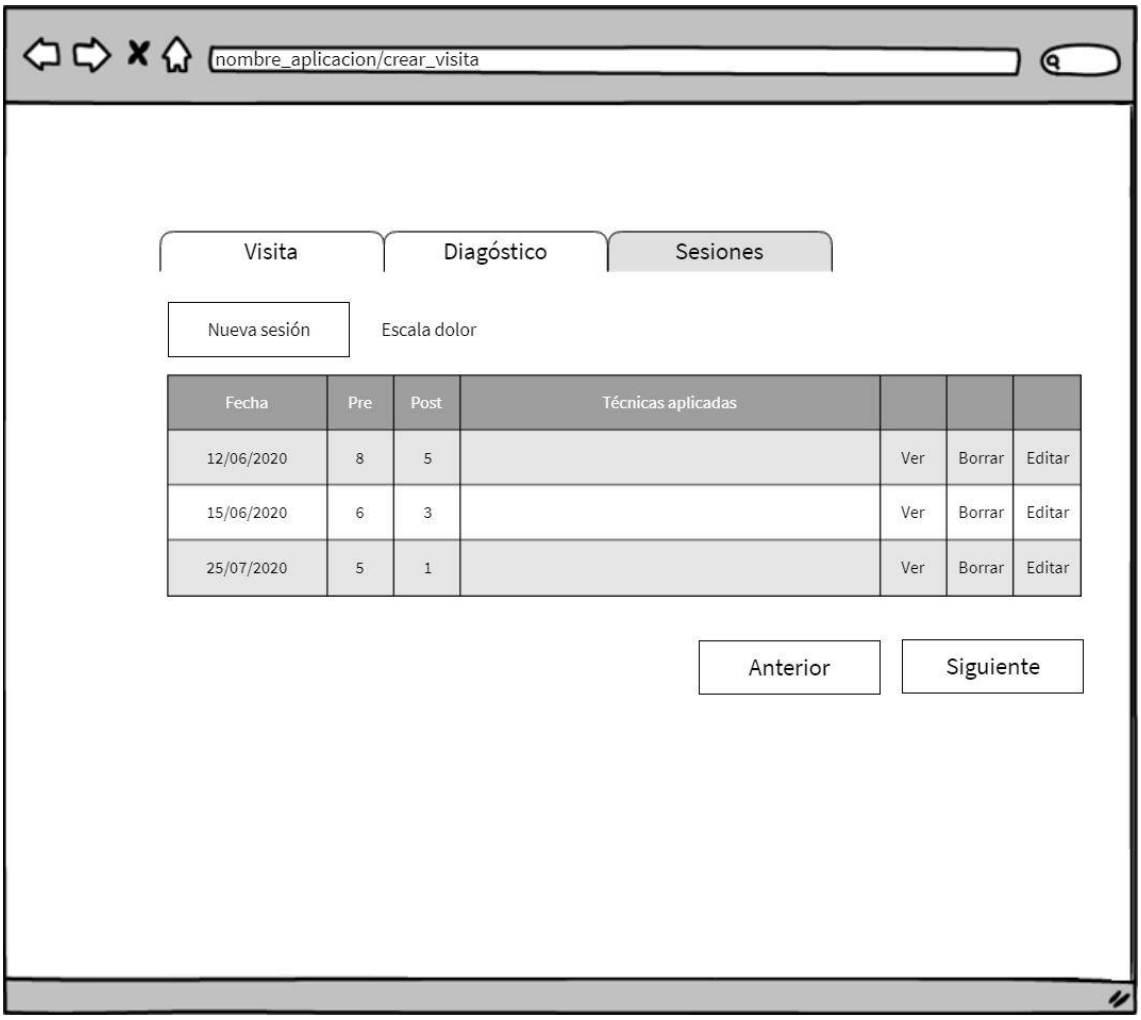

Figura 17 Boceto interfaz Crear visita pestaña sesión

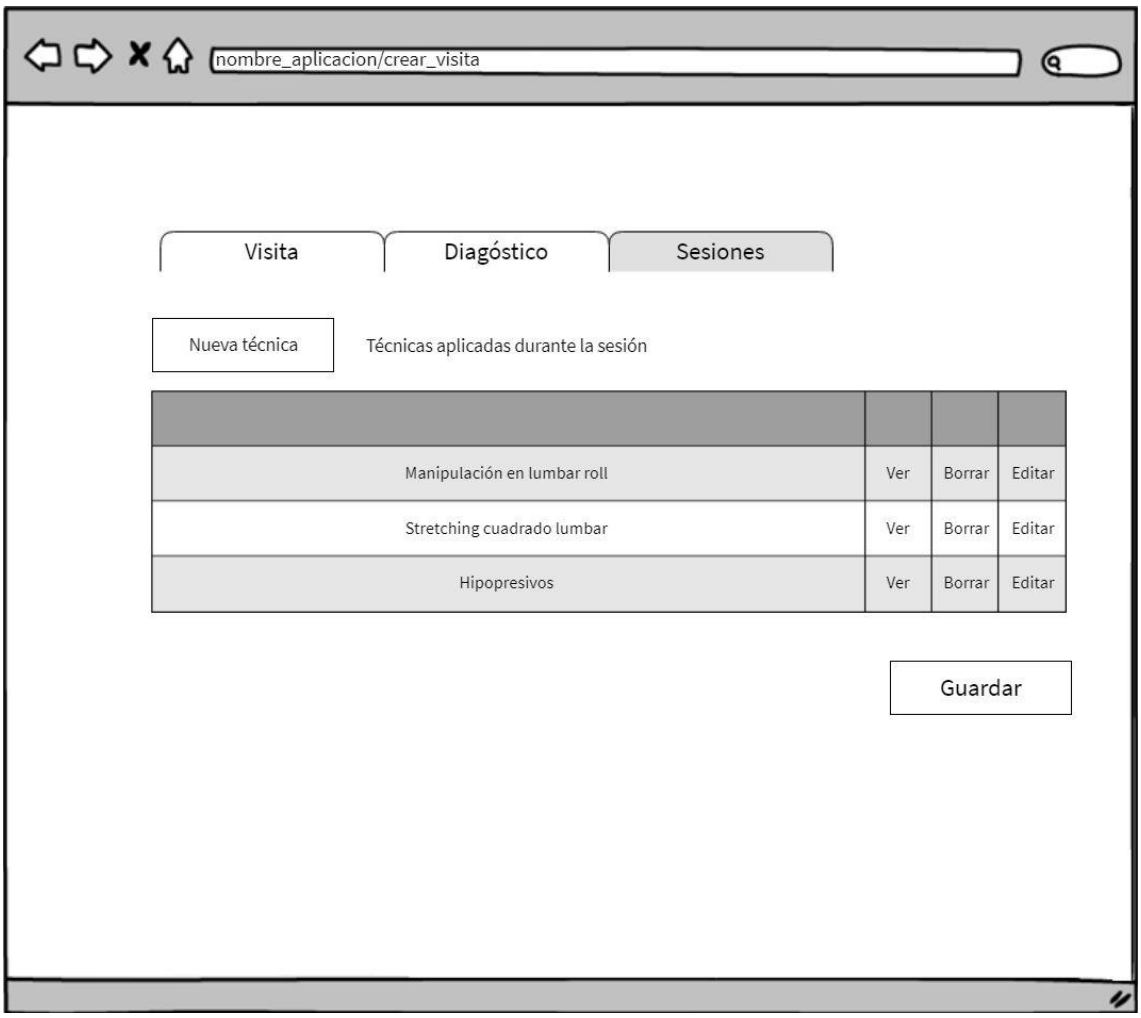

Figura 18 Boceto interfaz Crear visita pestaña técnicas

6.3 Diagrama de clases

A partir de la información identificada en el sistema y que se desea manejar a través de la aplicación web, a continuación se presenta el diagrama de clases que se ha diseñado para tal fin. Cabe destacar que en Ruby on Rails se hace el mapeo objeto relacional. Por lo tanto para implementar la base de datos lo único necesario fue pasar este diagrama de clases a código Ruby.

Los nombres de las clases están en ingles porque Ruby on Rails hace uso de los plurales en el código. Esto se explica más detalladamente en el punto Ruby on Rails.

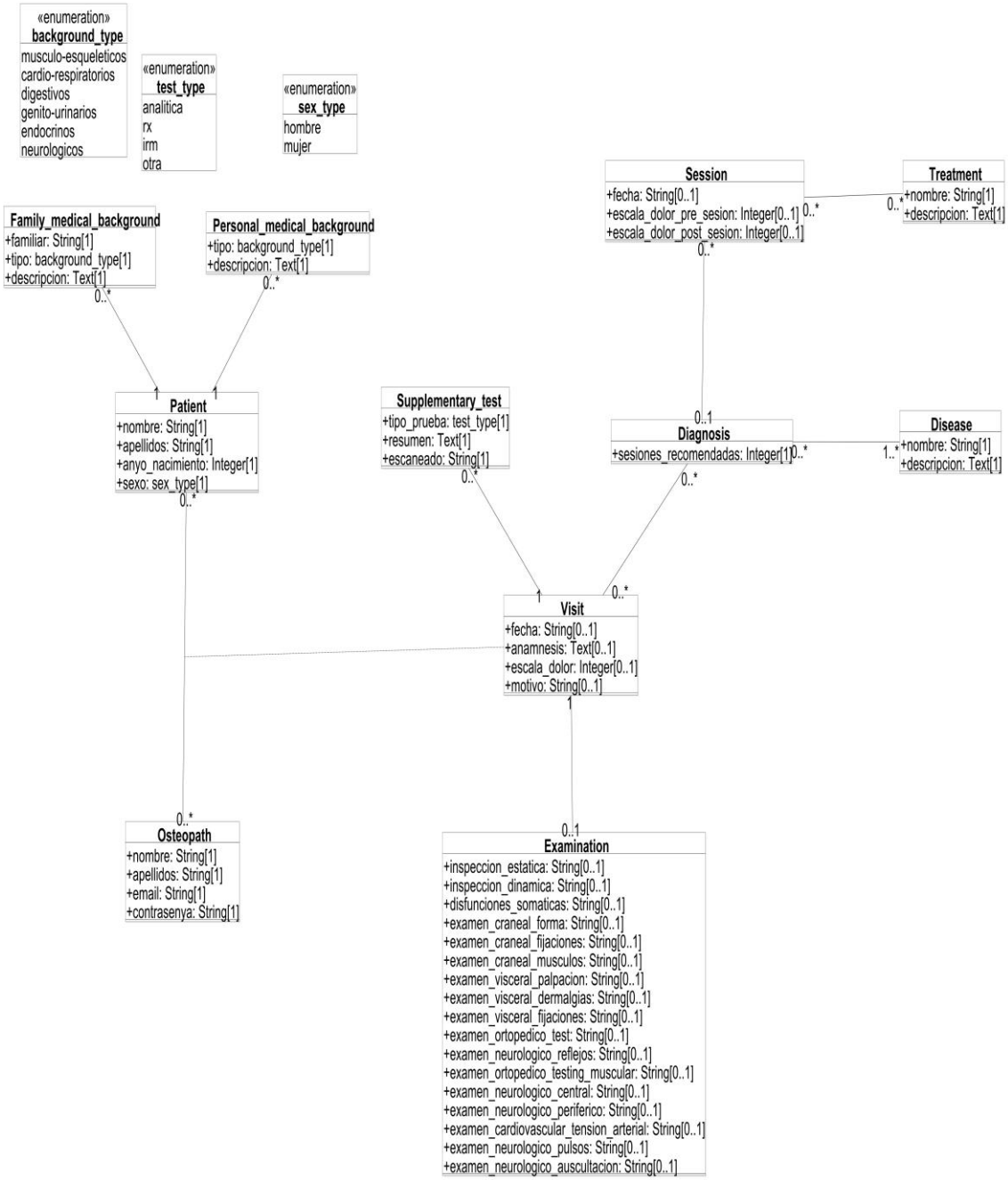

Figura 19 Diagrama de clases

A groso modo podemos decir que en el diagrama de clases se representa información sobre pacientes, osteópatas, visitas, diagnósticos y sesiones. Podríamos decir que estas clases son las principales del diagrama. El resto de clases actuarían como "atributos" de las mismas.

En total hay once tablas:

- **Osteopath:** representa a los osteópatas que se registran en la aplicación. El atributo "contrasenya" se guardará encriptado en la base de datos. Para esto se hará uso de Devise tal y como se explica en el punto "Tecnología Utilizada".
- **Patient:** representa a los pacientes a los que atiende un osteópata.

- **Family_medical_background:** representa los antecedentes médicos de un familiar del paciente.
- **Personal_medical_background:** representa los antecedentes médicos de un paciente.
- **Visit:** representa una visita que le hace un osteópata a un paciente. En cuanto a la asociación ternaria que vemos representa que la visita pertenece a la relación entre paciente y osteópata.
- **Examination:** representa las pruebas que hace el osteópata para examinar al paciente cuando acude a la consulta.
- **Supplementary_test:** representa cualquier prueba adicional que aporte el paciente (por ej. unas placas, una analítica, etc.).
- **Diagnosis:** representa el diagnostico que le hace un osteópata a uno de sus pacientes.
- **Disease:** representa una enfermedad concreta que tiene el paciente.
- **Treatment:** representa el tratamiento que le ofrece un osteópata a uno de sus pacientes. Este tratamiento se lo ofrece parara una enfermedad concreta.
- **Session:** esto representa una sesión de fisioterapia que le ofrece un osteópata a un paciente.

Además de estas clases hay tres enumeraciones:

- **sex_type:** que representa los dos posibles sexos que se contemplan en nuestro software: hombre y mujer. Esta enumeración la usa la clase Patient en su atributo sexo.
- **test_type:** representa el tipo de pruebas que se pueden realizar. Esta enumeración la usa la clase Supplementary_test en su atributo tipo_prueba.
- **background_type:** representa el tipo de antecedente médico. Esta enumeración la usan las clases Family_medical_background y Personal_medical_background en su atributo tipo.

Por último mencionar la relación entre Patient y Osteopath que tiene la clase Visit que pertenece a la relación. Este tipo de relación se suele representar de la siguiente forma en la base de datos:

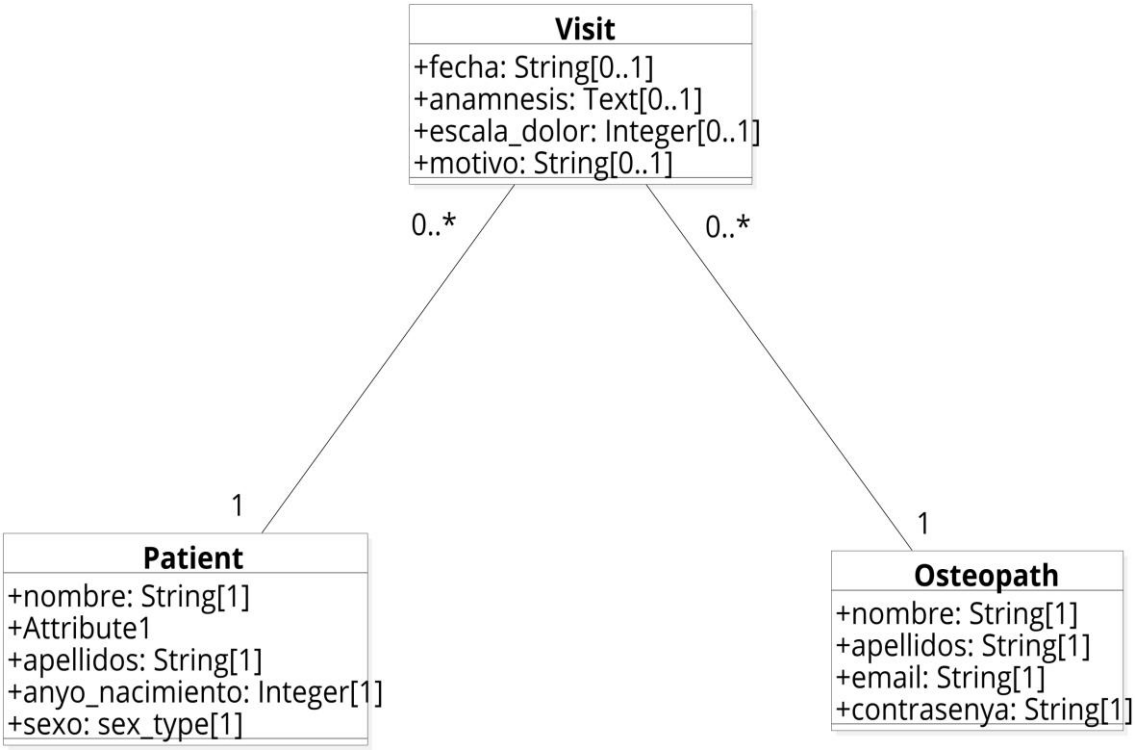

Figura 20 Relación Patient-Osteopath-Visit desglosada

Esta es una relación equivalente a la que se ve en el diagrama de clases representado en la Figura 19 Diagrama de clases que sí que podemos implementar fácilmente en la base de datos.

7. Implementación

En este apartado, se describirán las diferentes vistas que conforman el sitio web y se describirá brevemente como han sido implementadas.

7.1 Vista "/patients/new"

Esta vista implementa el caso de uso RF7-Crear_paciente (ver Tabla 1 Descripción casos de uso); por lo tanto, permite que un osteópata cree un paciente. Los datos que se deben introducir son: el nombre, los apellidos, el año de nacimiento y el sexo del paciente, así como los antecedentes médicos personales y familiares, siendo estos dos últimos opcionales y el resto obligatorios.

Esta ventana esta implementada principalmente con Ruby y JavaScript. Para los campos nombre, apellidos, año nacimiento y sexo simplemente se hace un POST a la base de datos cuando se acciona el botón "Crear". En este POST se transmiten los datos que hay en los campos directamente al servidor.

La tabla de antecedentes está hecha de tal forma que cuando se acciona añade o se borra un antecedente este se actualiza en la vista mediante JavaScript (pues con este lenguaje se puede actualizar el DOM). Además también se actualiza una variable JavaScript. En el momento que se hace POST mediante el botón "Crear" se transmiten al servidor los datos almacenados en la variable mencionada anteriormente.

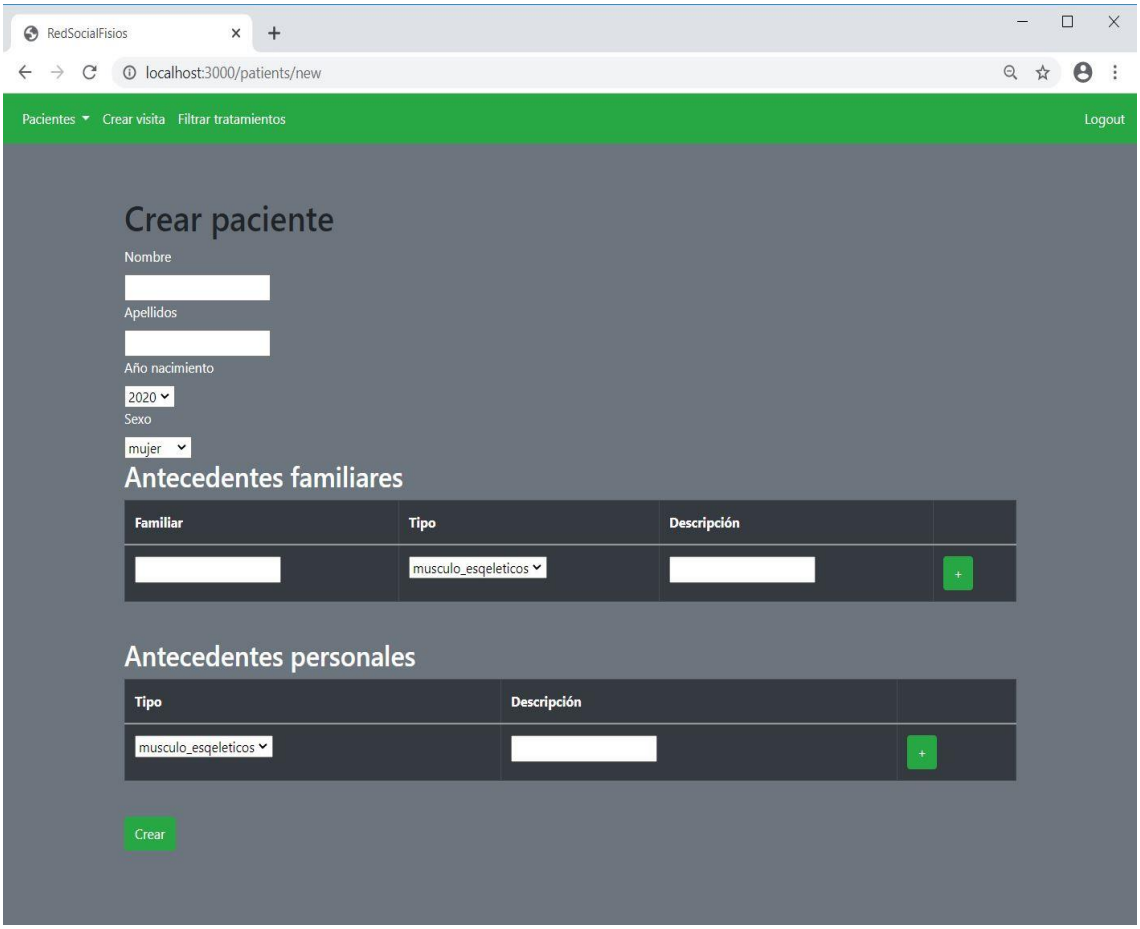

Figura 21 Captura de pantalla vista "/patients/new"

7.2 Vista "/crear_visita"

Esta vista implementa el caso de uso RF4-Crear_visita (ver Tabla 1 Descripción casos de uso); por lo tanto, permite crear visitas, diagnóstico y sesiones. En cuanto a la creación de vistas se hace mediante la interfaz que se puede ver en la Figura 22 Captura de pantalla vista "/crear_visita" creación sesiones. Tiene los campos paciente, osteópata, anamnesis y escala de dolor. En cuanto al osteópata y al paciente busca en la base de datos las entidades de estos (que deben haberse creado previamente) y los asocia a la vista creada. Además el campo osteópata tiene como valor por defecto el nombre del osteópata logeado.

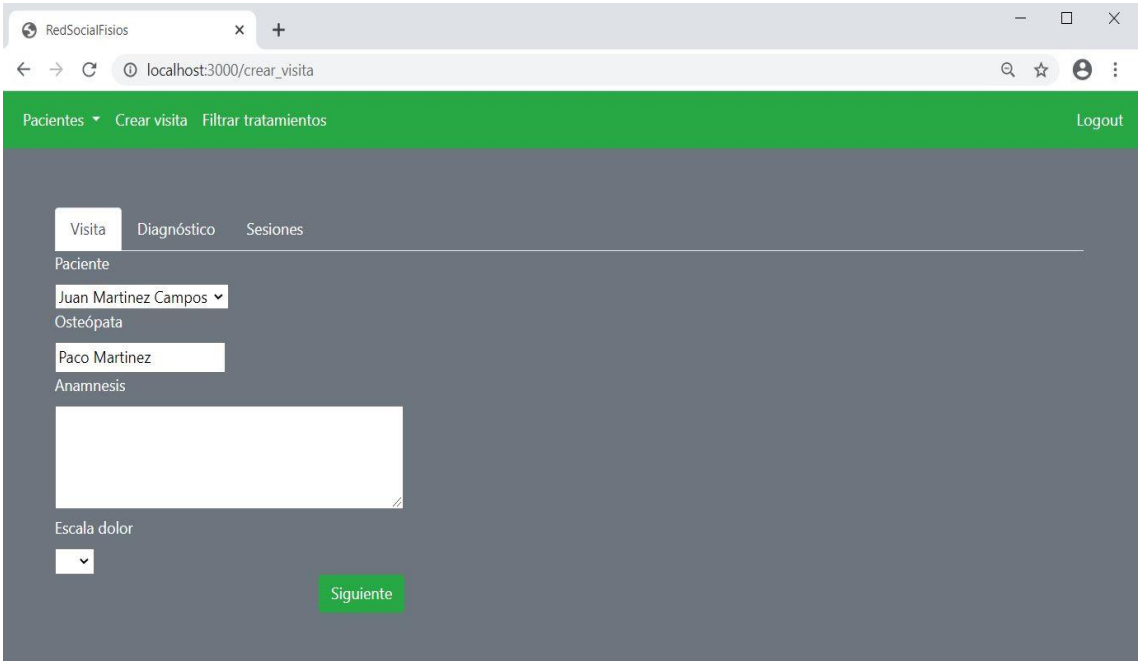

Figura 22 Captura de pantalla vista "/crear_visita" creación sesiones

En cuanto a la creación de diagnósticos (ver Figura 23 Captura de pantalla vista "/crear_visita" creación de diagnósticos). Se debe introducir los campos enfermedad, descripción y sesiones recomendadas. Con los dos primeros se crea una nueva enfermedad mientras que con el último se crea un diagnóstico. La enfermedad y diagnóstico creados se asocian.

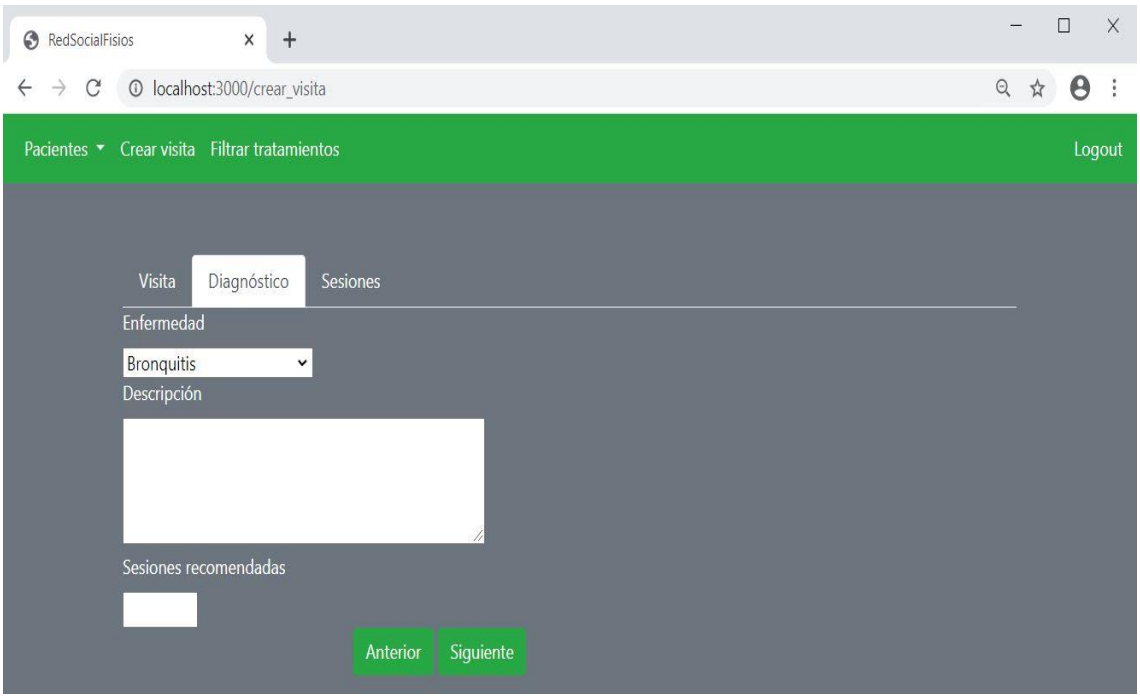

Figura 23 Captura de pantalla vista "/crear_visita" creación de diagnósticos

En cuanto a la creación de sesiones de fisioterapia se hace en las ventanas que se ven a continuación (ver en la Figura 22 Captura de pantalla vista "/crear_visita" creación sesiones y en la Figura 27 Captura de pantalla vista "/crear_visita" creación de técnicas modal introducción de datos). Se pueden crear varias sesiones pulsando el botón "Nueva sesión"; cuando sucede esto se abre la ventana modal (ver figura Figura 27 Captura de pantalla vista "/crear_visita" creación de técnicas modal introducción de datos). A continuación se llenan los campos de ésta y se pulsa el botón "Crear sesión". Entonces añade la visita a la tabla. Además se permiten borrar sesiones mediante el texto "Borrar".

Esta tabla se ha hecho con JavaScript y HTML. Cuando se pulsa el botón crear sesión se añade un nuevo campo a la tabla, recogiendo los datos de los campos fecha, escala de dolor presesión y escala de dolor postsesión. Estas dos acciones se hacen accediendo al DOM. Por último, mencionar que el modal se ha hecho gracias a Bootstrap.

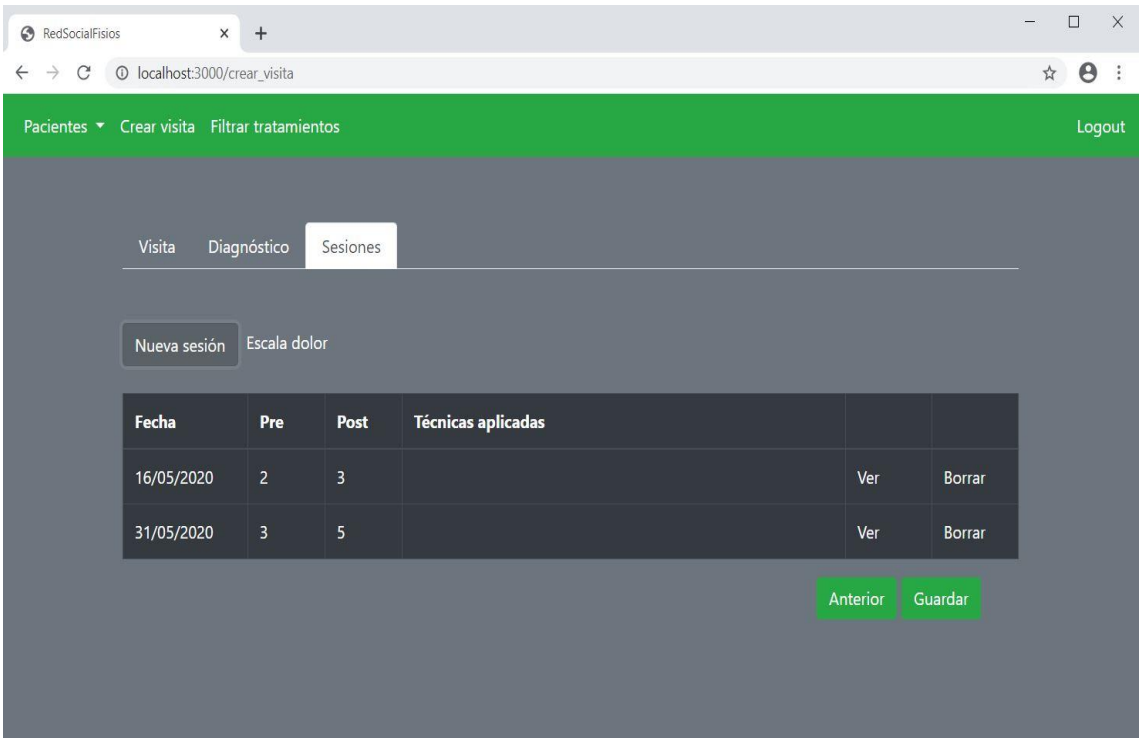

Figura 24 Captura de pantalla vista "/crear_visita" creación de sesiones

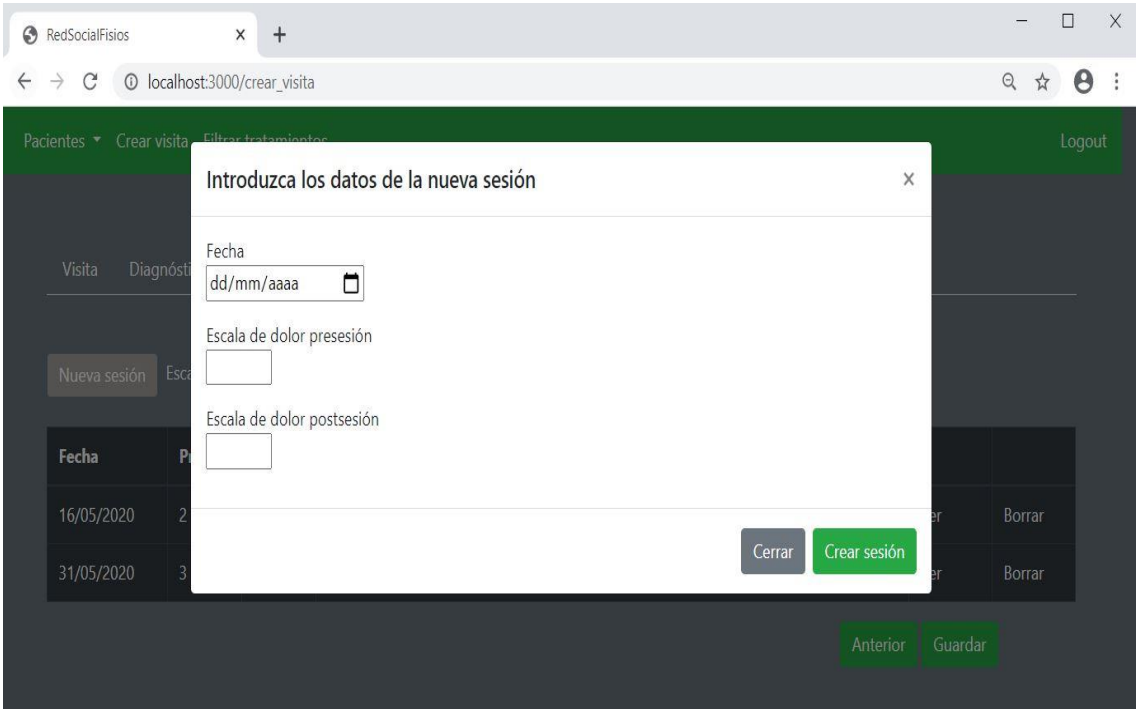

Figura 25 Captura de pantalla vista "/crear_visita" creación de sesiones modal introducción de datos

Para crear tratamientos se hace con las dos ventanas que se ven a continuación (ver la Figura 26 Captura de pantalla vista "/crear_visita" creación de técnicas y la Figura 27 Captura de pantalla vista "/crear_visita" creación de técnicas modal introducción de datos). Se accede a la ventana representada en la figura Figura 26 Captura de pantalla vista "/crear_visita" creación de técnicas pulsando el texto

"Ver" de una de las sesiones de la tabla con las sesiones mencionada anteriormente.

Una vez hecho esto llena los campos fecha, pre y post (que podemos ver en la primera imagen) con los mismos valores de la tabla para crear sesiones. Estos pueden ser modificados y ello se reflejará en la tabla de sesiones. Por último el mecanismo para crear y borrar técnicas es el mismo que para crear sesiones; añadiéndose éstas a la tabla y pudiéndose borrar mediante el texto "Borrar".

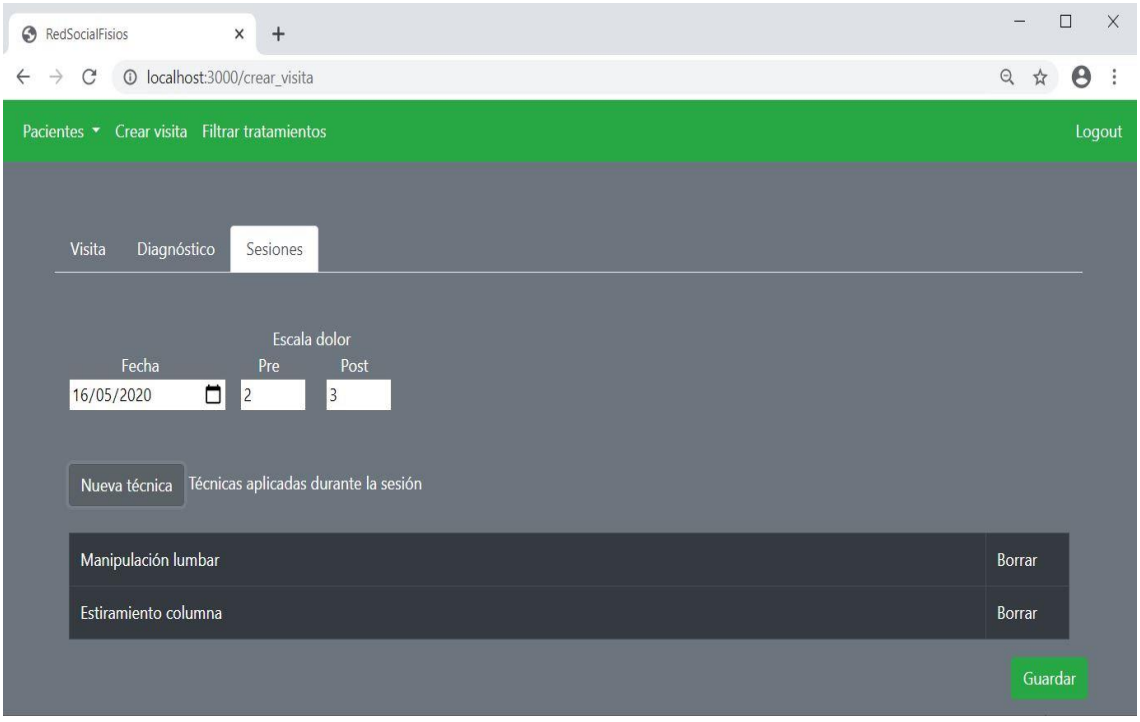

Figura 26 Captura de pantalla vista "/crear_visita" creación de técnicas

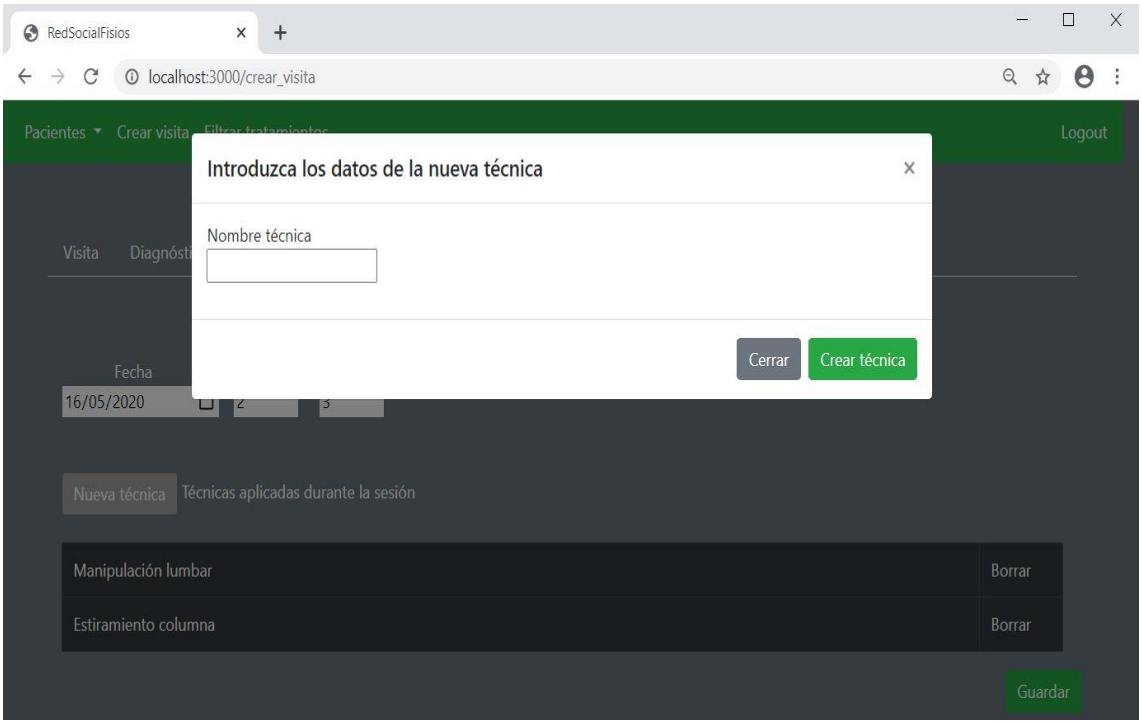

Figura 27 Captura de pantalla vista "/crear_visita" creación de técnicas modal introducción de datos

A todas estas ventanas se accede mediante la URL "/crear_visita". Se navega entre ventanas principalmente mediante los botones "Siguiente" y "Anterior". Esto se hace de la siguiente manera: cuando se carga la URL "/crear_visita" se cargan todas las ventanas. Luego usando JavaScript y CSS se ocultan o muestran las ventanas que sean oportunas.

7.3 Vista "/filtrar_tratamientos"

Esta vista implementa el caso de uso RF6-Buscar_tratamientos (ver Tabla 1 Descripción casos de uso); por lo tanto, permite filtrar los tratamientos aplicados para cada diagnóstico. Cuando se accede a la interfaz por primera vez el usuario ve los campos para introducir los valores por los que desea filtrar que vienen vacíos por defecto (ver Figura 28 Captura de pantalla vista "/filtrar_tratamientos" campos de datos por defecto).

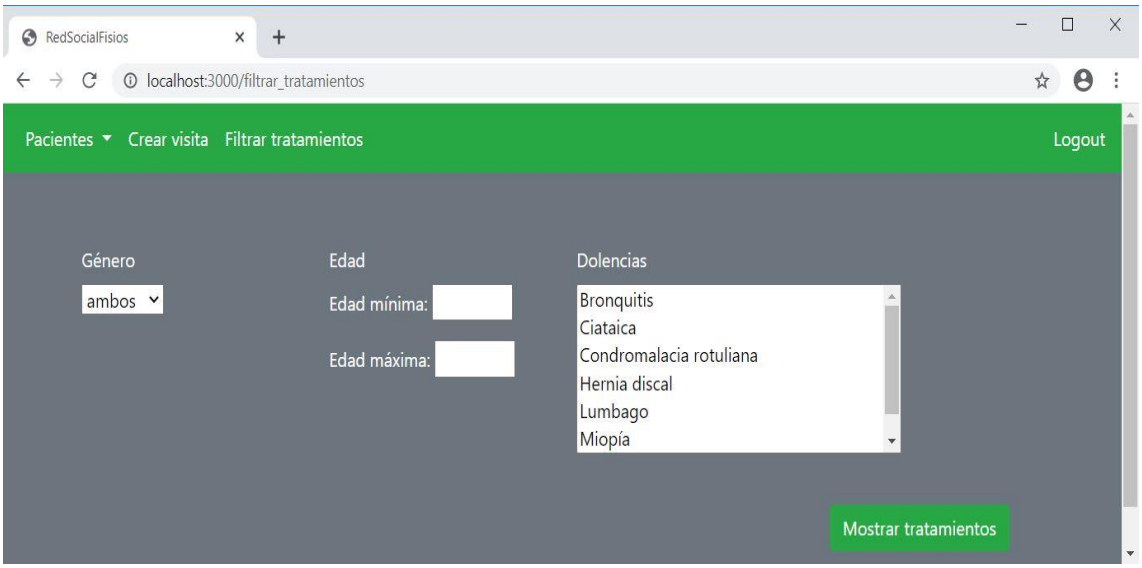

Figura 28 Captura de pantalla vista "/filtrar_tratamientos" campos de datos por defecto

Se permite filtrar por el género y la edad del paciente al que se le ha aplicado el diagnóstico, así como las dolencias para las que se ha aplicado el tratamiento.

En el campo género podemos escoger entre tres posibles valores: hombre, mujer y ambos. Si escogemos alguno de los dos primeros valores muestra los tratamientos aplicados a los pacientes hombre o mujer respectivamente; si elegimos ambos no tiene en cuenta el género del paciente a la hora de filtrar.

En el campo edad podemos elegir el rango de edad en el que están los pacientes a los que se ha aplicado el tratamiento. Si alguno de estos dos campos se deja vacío no tiene en cuenta la edad mínima y/o máxima a la hora de filtrar tratamientos. El valor introducido en estos campos es incluyente, es decir si se introduce una edad minina de 40 muestra todos los tratamientos aplicados a pacientes con una edad mayor o igual a 40 años.

Por último deja filtrar por las enfermedades para las que se ha aplicado el tratamiento, pudiéndose seleccionar varias a la vez. Por ejemplo, si se selecciona hernia discal y ciática muestra todos los tratamientos aplicados para solucionar hernias discales y todos los tratamientos aplicados para solucionar problemas de ciática. Si no se selecciona ninguna enfermedad no tiene en cuenta este criterio a la hora de filtrar tratamientos.

Cuando se pulsa el botón "Mostrar tratamientos" muestra los diagnósticos de la base de datos. Con los valores en los campos que se ven a continuación es la forma menos restrictiva; por lo tanto muestra todos los diagnósticos. Los diagnósticos extraídos se muestran en una tabla como la siguiente:

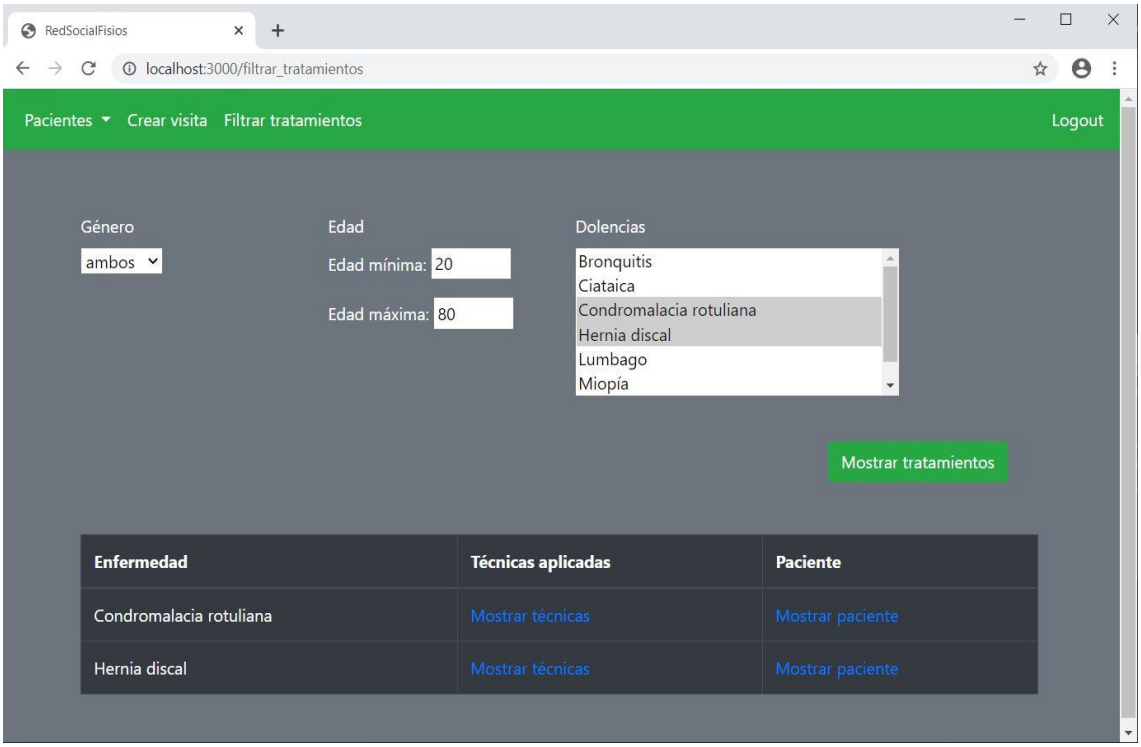

Figura 29 Captura de pantalla vista "/filtrar_tratamientos" tabla tratamientos

Para buscar en la base de datos por los tres valores introducidos se ha hecho de la siguiente manera: se han creado cuatro métodos; tres de ellos para buscar los tratamientos aplicados a pacientes según su sexo, su edad mínima y su edad máxima tratamientos. El cuarto método busca los tratamientos aplicados a las enfermedades seleccionadas en el campo dolencias.

En todos los métodos mencionados en el párrafo anterior, si reciben el argumento por el cual el usuario no desea considerar ese campo para filtrar los tratamientos (por ejemplo ambos en el campo sexo) se devuelven todos los tratamientos de la base de datos.

Finalmente, hay un cuarto método que recibe como argumento el género, la edad mínima y máxima y las dolencias. Este llama a los cuatro métodos ya mencionados y devuelve la intersección de estas tres listas. Con esto conseguimos que el código se bastante mantenible y fácil de entender, pudiéndose añadir más campos por los que filtrar en un futuro si fuera necesario.

Como podemos ver en la Figura 29 Captura de pantalla vista "/filtrar_tratamientos" tabla tratamientos hay dos enlaces que nos muestran información: "Mostrar técnicas" y "Mostrar paciente". El primero nos lleva a la siguiente ventana:

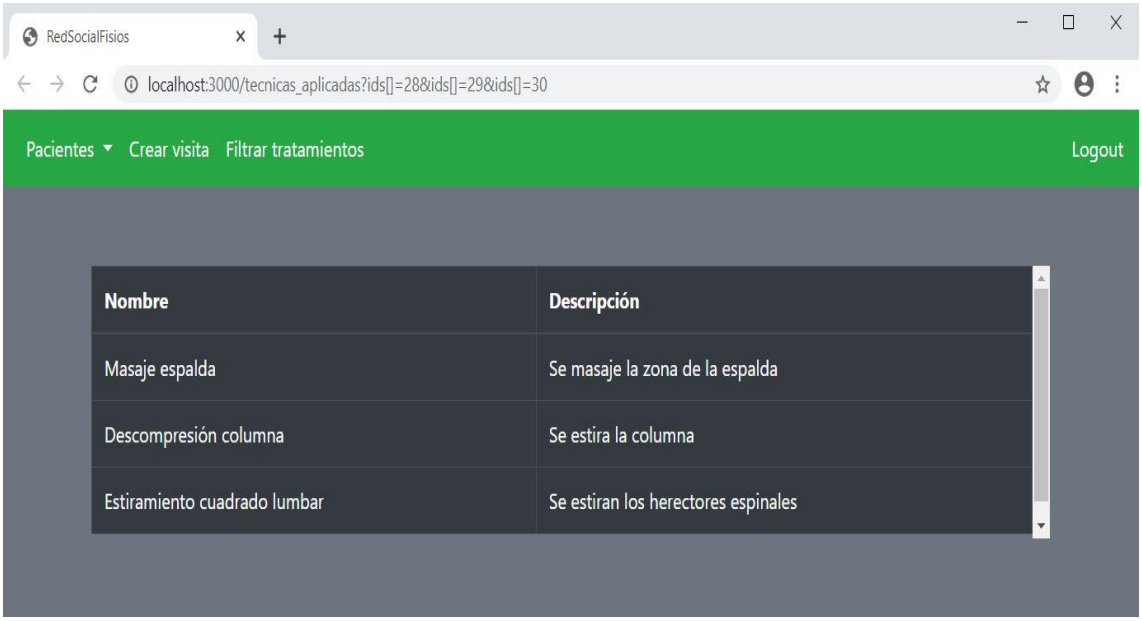

Figura 30 Captura de pantalla vista "/tecnicas_aplicadas"

Aquí se muestran todas las técnicas aplicadas para un cierto diagnóstico. Se omite la información de en qué sesiones se ha aplicado cada técnica. En cuanto al enlace "Mostrar paciente" lleva la vista que se muestra a continuación:

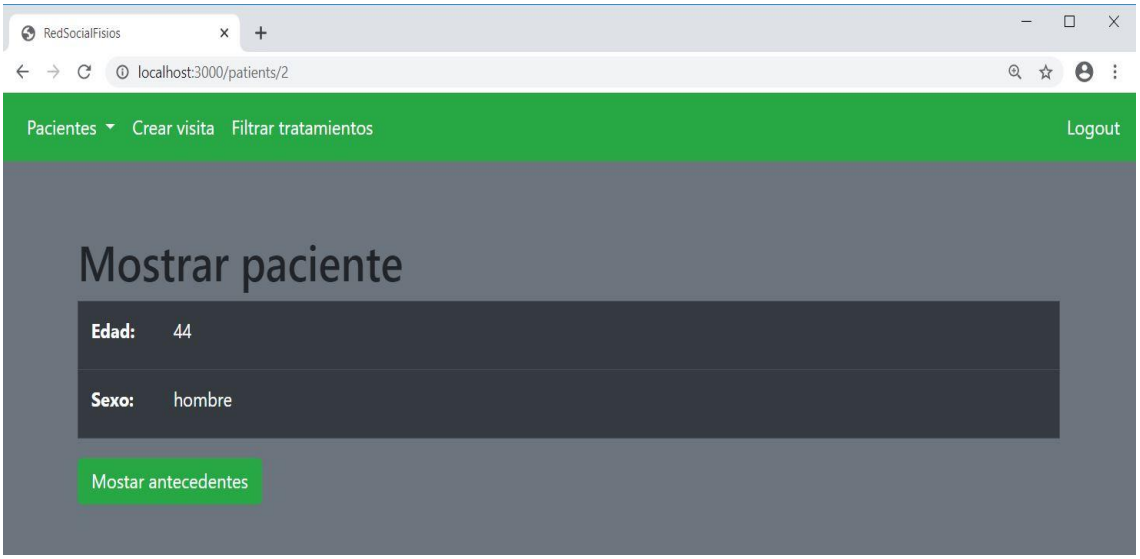

Figura 31 Captura de pantalla vista "/patients/id_paciete" sin nombre ni apellidos

Se accede a esta vista con el enlace "/patients/id_paciete". En esta, si ningún osteópata ha hecho log in o el paciente que se va a mostrar no pertenece al osteópata que esta logeado actualmente no se muestra el nombre y apellidos del paciente, pues se considera que esta puede ser información sensible. A continuación se muestra una captura de pantalla de la vista

"/patients/id_paciete" cuando el paciente que se va a mostrar pertenece al osteópata logeado:

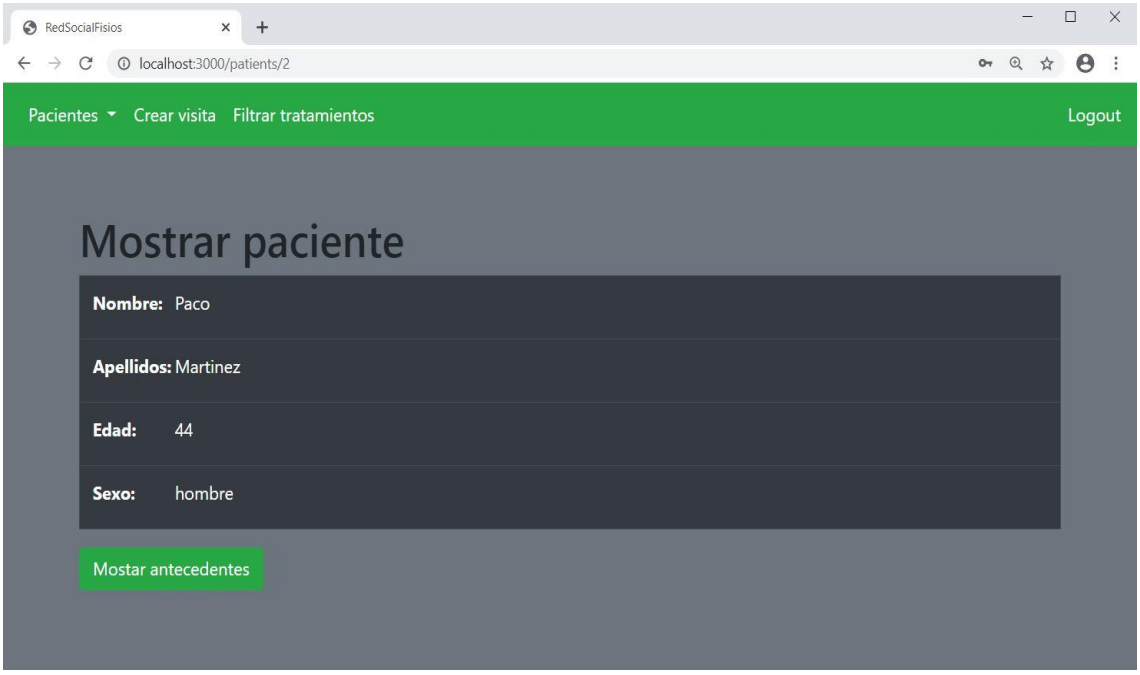

Figura 32 Captura de pantalla vista "/patients/id_paciete" con nombre ni apellidos

7.4 Vistas "/osteopaths/sign_in" y "/osteopaths/sign_up" y log out

Esta vista implementa los casos de uso RF1-Reg, RF2-Log_in, RF3-Log_out respectivamente (ver Tabla 1 Descripción casos de uso); por lo tanto, permite registrarse, hacer log in y log out. Se ha decidido juntar estas tres funcionalidades en un solo punto pues están relacionadas entre sí, además se programaron al mismo tiempo y de forma similar. Las interfaces gráficas para estas tres funcionalidades son las siguientes:

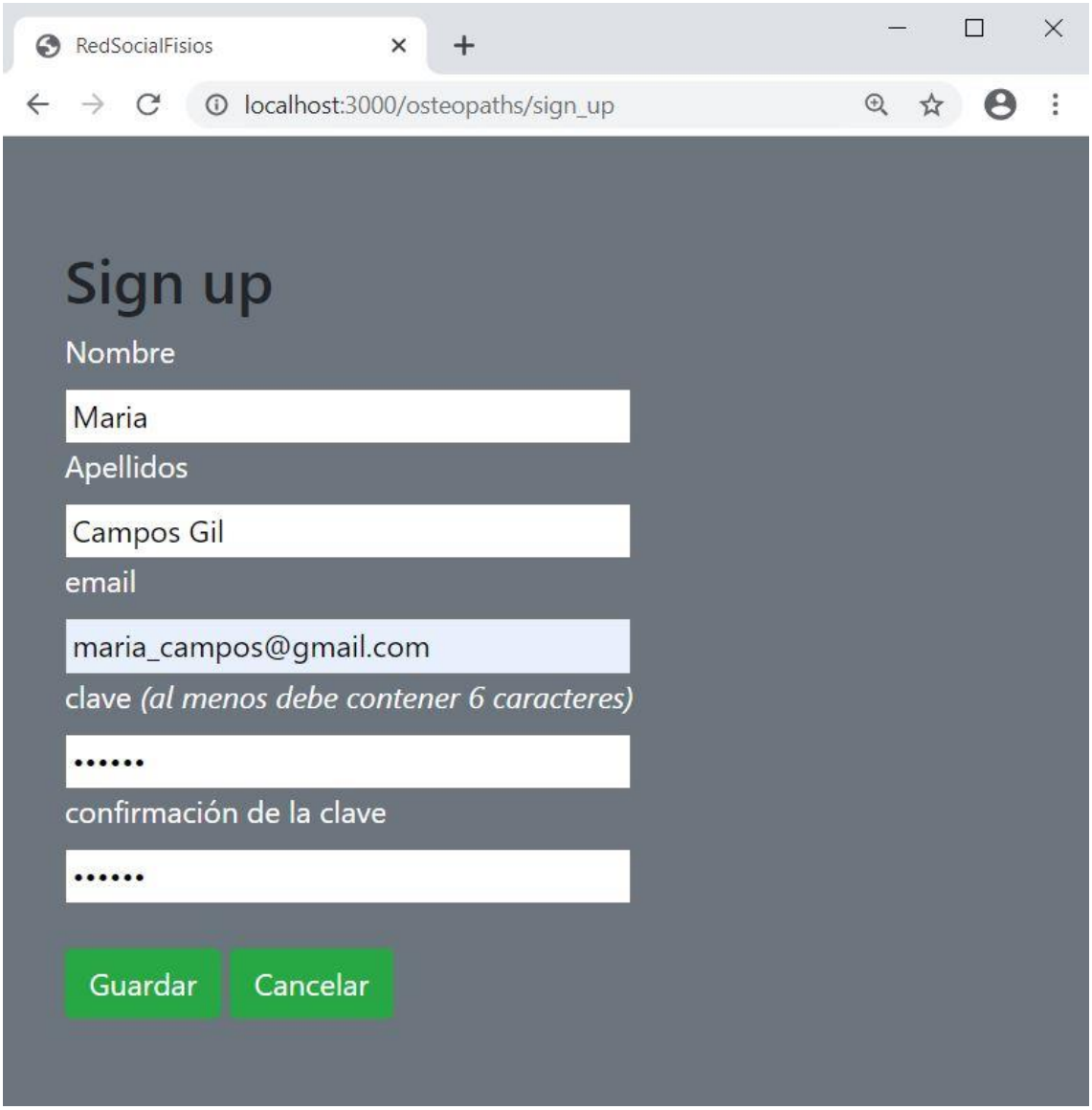

Figura 33 Captura de pantalla vista "/osteopaths/sign_up"

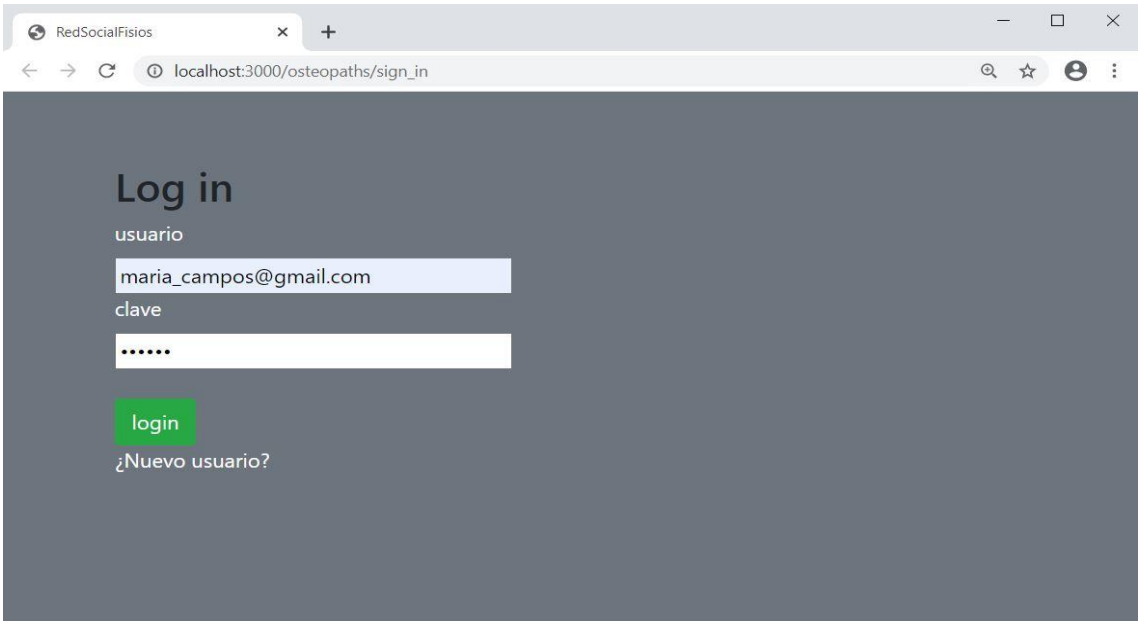

Figura 34 Captura de pantalla vista login

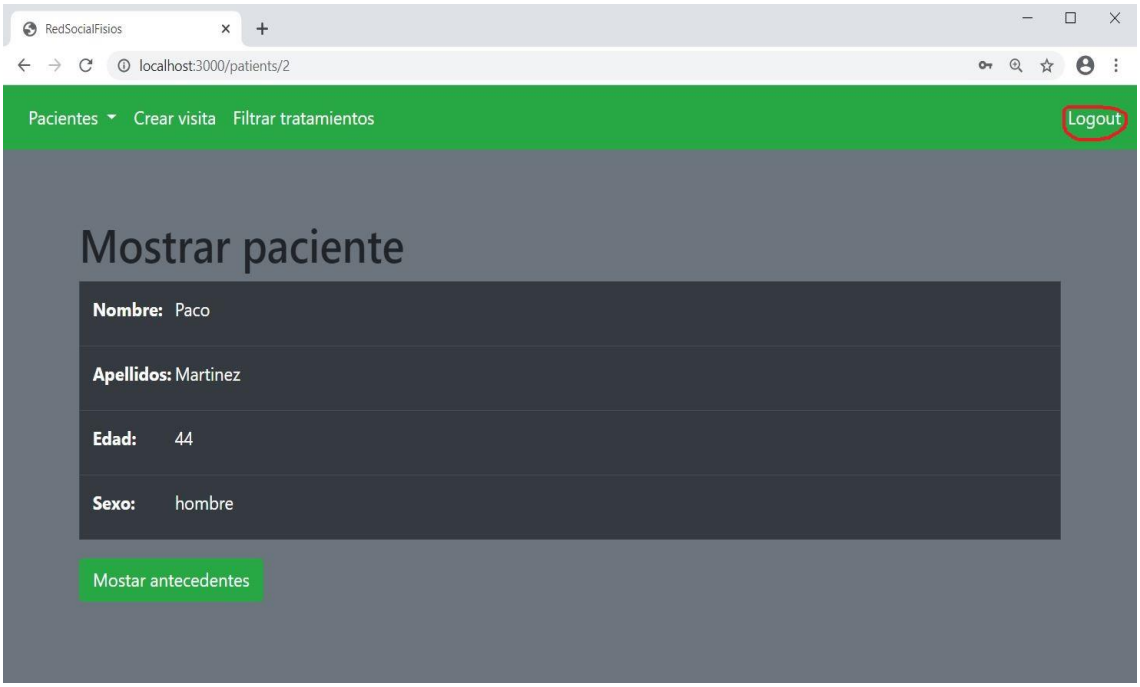

Figura 35 Captura de pantalla vista logout

En cuanto a la ventana para registrarse (ver Figura 33 Captura de pantalla vista "/osteopaths/sign_up") nos pide el nombre, apellidos, email y password del osteópata. Todos estos campos son obligatorios y además el password se debe repetir y debe tener al menos seis caracteres. En lo referente al email no debe haber ningún otro osteópata registrado con el mismo email y debe estar bien formado. Al pulsar el botón para registrarse hace log in con el usuario creado de forma automática.

Para hacer log in solo es necesario introducir el email y el password. Por último decir que la ventana de log out es un enlace que permite que aparece en todas la ventanas de la aplicación si el osteópata ha hecho log in previamente.

Para crear estas ventanas se ha usado Devise. Este es una gema para Ruby on Rails para la autenticación de usuarios. Esta gema tiene diversos usos pero en el caso de este proyecto ha servido para crear las ventanas de registrarse, hacer log in y log out y encriptar la contraseña en la base de datos.

En cuanto a la creación de las interfaces no fue necesario programarlas, ya que con Devise se pueden generar el código de estas mediante comandos. Cabe destacar que el código generado es bastante legible y que por supuesto tenemos acceso a él, con lo cual se puede modificar si se estima oportuno.

En cuanto a la encriptación de la contraseña es necesario por razones de seguridad. No obstante puede resultar bastante complejo de programar manualmente, además se corre el riesgo de no hacerlo de la mejor manera exponiendo por tanto datos muy sensibles. Con Devise solucionamos estos dos problemas, despreocupándonos de la complejidad del algoritmo de encriptación y corroborando de que se hace de manera segura.

7.5 Vista “/patients”

Esta vista implementa el caso de uso RF5-Mostrar_pacientes (ver Tabla 1 Descripción casos de uso); por lo tanto, permite mostrar todos los pacientes del osteópata que está logeado. Básicamente obtiene cuál es el osteópata logeado actualmente y muestra los pacientes de éste.

Podemos saber cuál es el osteópata logeado gracias a Devise; en concreto se ha hecho mediante la línea de código “current_osteopath”. En cuanto a mostrar los pacientes de este osteópata en primer lugar, se consulta a la base de datos cuales son los pacientes del osteópata logeado desde el controlador y se guarda en una variable. Después se accede a esta variable desde la vista y se crea el HTML con el listado.

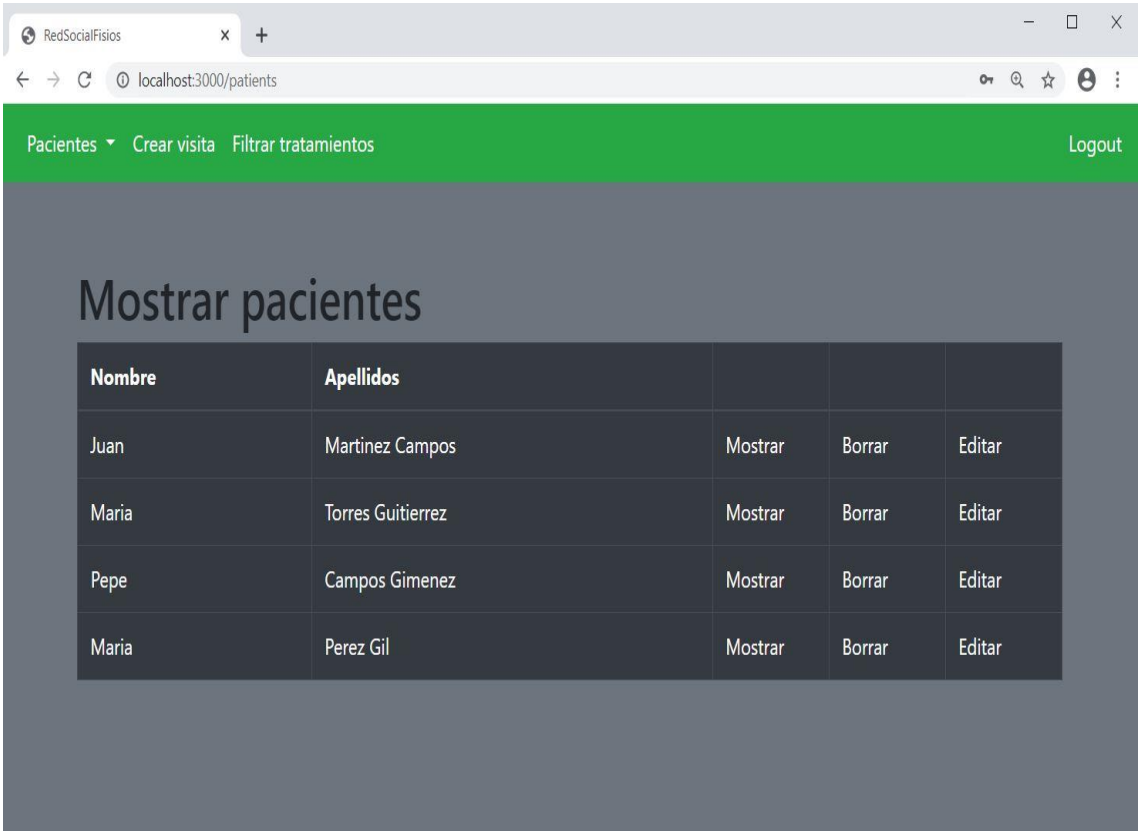

Figura 36 Captura de pantalla vista “/patients”

7.6 Vista “/patients/:id_paciente”

Esta vista implementa el caso de uso RF5-Mostrar_pacientes (ver Tabla 1 Descripción casos de uso); por lo tanto, permite mostrar los datos de un paciente en concreto. Se accede a ella mediante la URL “/patients/id_del_paciente”. id_del_paciente debe valer un número entero mayor que cero, éste representa la id del paciente en la base de datos. El usuario puede no saber el valor de la id del paciente que desea mostrar pero esto no es problema ya que se puede acceder a esta vista desde el enlace “mostrar” de la vista “/patients” (ver Figura 36 Captura de pantalla vista “/patients”). La ventana es la que se muestra a continuación:

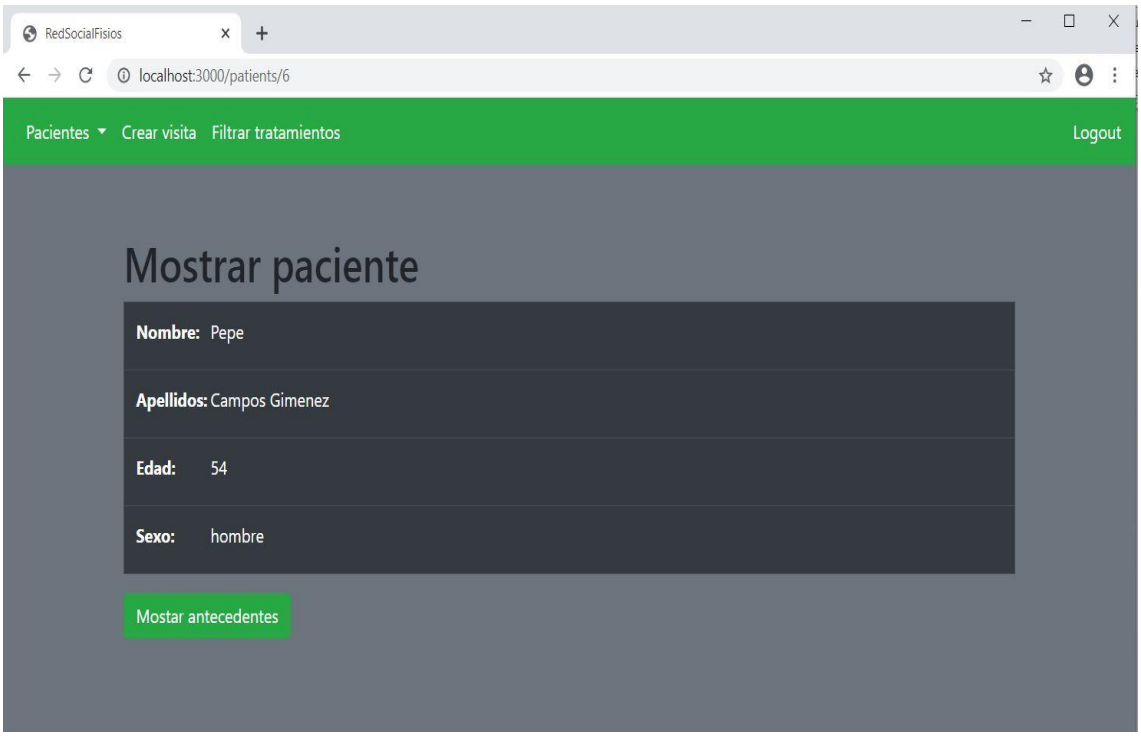

Figura 37 Captura de pantalla vista "/patients/id_del_paciente" con nombre y apellidos

Además en esta vista si no hay ningún osteópata logeado o el usuario que se desea mostrar no pertenece al osteópata logeado actualmente no muestra el nombre y apellidos del paciente, como ya hemos mencionado en el punto 'Vista "/filtrar_tratamientos"'. Por lo tanto la vista resultante sería la siguiente:

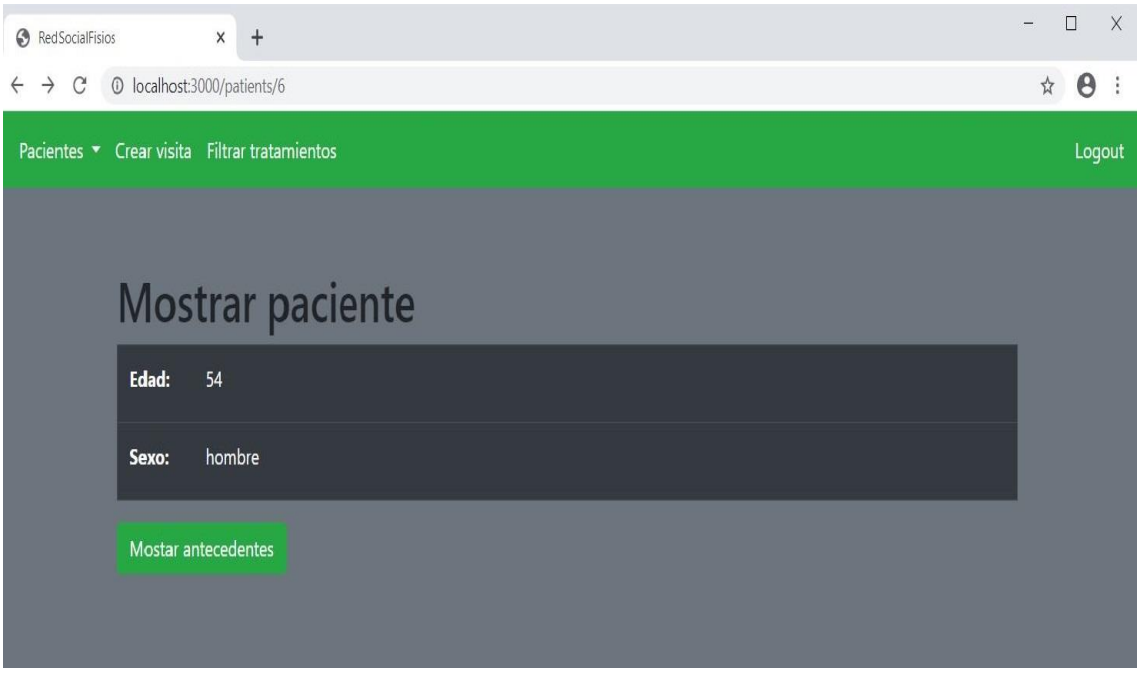

Figura 38 Captura de pantalla vista "/patients/id_del_paciente" sin nombre y apellidos

Para crear esta interfaz, primero se accede al parámetro id desde el controlador y se busca el paciente de la base de datos que tiene este id. Después se accede a dicho paciente desde la vista y se muestran los datos del mismo. Por último, en cuanto a mostrar o no el nombre y los apellidos, lo hace mediante un condicional en la vista.

Por último mencionar que cuando hacemos clic sobre el enlace "Antecedentes" nos lleva a la siguiente vista:

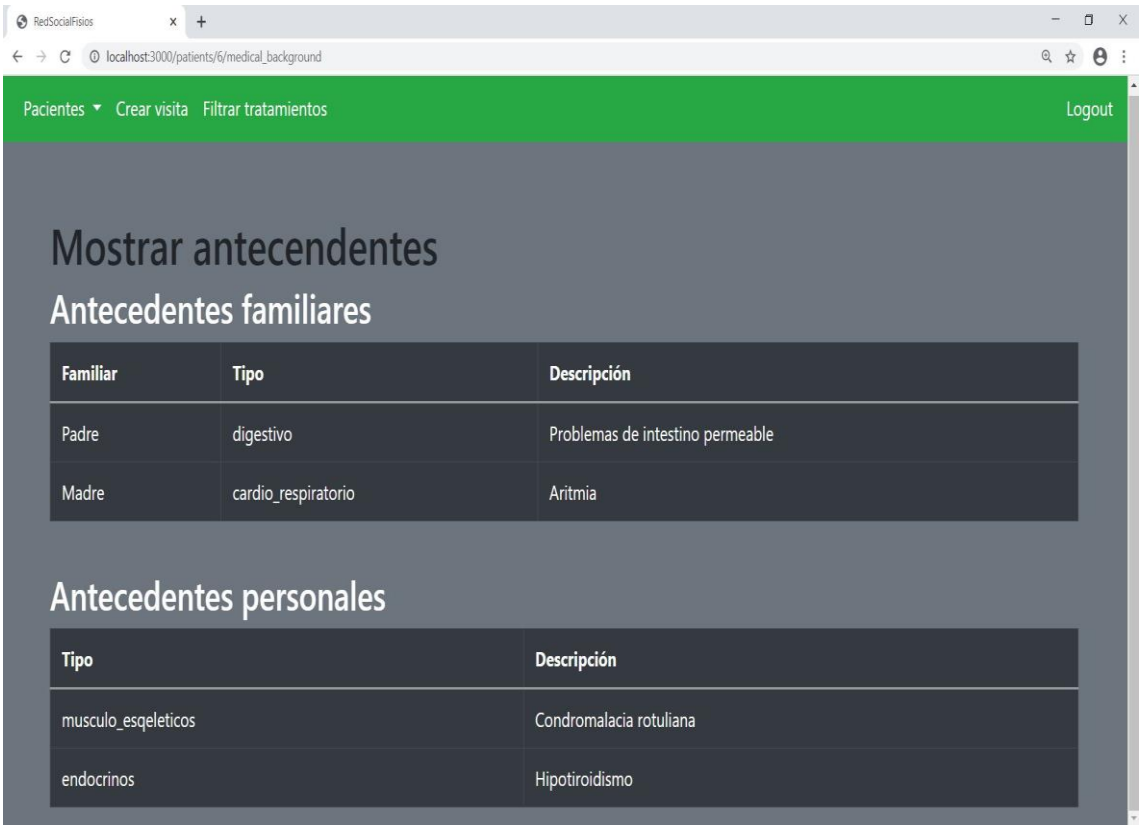

Figura 39 Captura de pantalla vista "/patients/4/medical_background"

En esta se muestran los antecedentes familiares y personales del paciente. Se accede a ella mediante la URL "/patients/id_del_paciente/medical_background".

Esta vista se ha hecho también buscando el paciente que tenga la id que se pasa en los parámetros. Posteriormente en la vista se crea un el código HTML que se pueden observar en la vista anterior.

8. Conclusiones

Iniciamos el proyecto con la idea de construir una aplicación web que permitiera compartir información entre profesionales de la fisioterapia. Podemos decir que lo hemos conseguido pues el software resultante puede gestionar y consultar los tratamientos y las técnicas aplicadas por osteópatas y fisioterapeutas a sus pacientes. Adicionalmente, también permite gestionar los pacientes de un osteópata. Por lo tanto, podemos decir que se han cumplido en gran parte los objetivos planteados:

- Durante el TFG se han utilizado técnicas para la planificación y gestión de proyectos como es la metodología en cascada. El conocimiento de las metodologías de trabajo para el desarrollo de software fue adquirido durante el grado. Por supuesto durante el proyecto se ha sido necesario ampliar el mismo, tanto de forma teórica como práctica.
- Durante el proyecto fue necesario aprender la arquitectura modelo-vista-controlador y como esta se concreta en código ejecutable. Para ello, en primer lugar se buscó información general a nivel teórico de este patrón de diseño. Este es un patrón muy usado y existen multitud de frameworks que lo implementan. En nuestro caso elegimos Ruby on Rails, lo cual nos lleva al siguiente punto.
- En cuanto al aprendizaje del uso de un nuevo framework, se empezó buscando información general de Ruby on Rails y su estructura de código.
  A continuación, esto se buscó información de pequeños proyectos hechos en Rails. Fue un proceso gradual de aprendizaje de las diferentes partes este framework (la base de datos, los modelos, las vistas, información sobre las gemas...). Una vez finalizado ya se consideró que se tenía el conocimiento suficiente para empezar el código del proyecto.
- En cuanto a ofrecer la capacidad de gestionar usuarios se ha hecho una ventana para gestionar el registro, el log in y el log out de osteópatas (que es el único tipo de usuario que existe en el proyecto) en la aplicación.
  En lo referente a crear pacientes deja crear pacientes en la aplicación pudiendo introducir tanto sus datos personales como sus antecedentes médicos.
  Respecto a crear diagnósticos y crear tratamientos estas funcionalidades están integradas en la vista para crear visitas, el diagnóstico y los tratamientos creados se asocian de forma automática con la vista creada.
  Por último, en relación con mostrar tratamientos y su efectividad, se ha creado una ventana capaz de mostrar los tratamientos que se han aplicado a los pacientes y filtrar éstos según sus datos.
- En cuanto a crear una base de datos usando el patrón de diseño active record en la que se guarda la información de pacientes, osteópatas, visitas, diagnósticos y tratamientos, en primer lugar se buscó información general del patrón de arquitectura active record. Después se buscó información sobre su funcionamiento en Rails.
- En segundo lugar, se propuso un diagrama de clases. Ya teniendo este diagrama y la información sobre el funcionamiento del active record en Rails solo hubo que añadir dichas clases al proyecto y dejar que el framework creara las tablas del modelo relacional.

No obstante, existen muchas posibles mejoras, que por limitaciones de tiempo de desarrollo no han sido implementadas:

El uso de un framework front-end, el cual no ha sido incluido en proyecto por las siguientes razones:

- En primer lugar, Rails sí que deja enviar ficheros JavaScript al cliente, por lo cual no es estrictamente necesario el uso de un framework. Además, a pesar de que sí que ha sido necesario programar en JavaScript, su uso no ha sido muy extenso pues para la mayoría de funcionalidades se podían implementar prescindiendo del mismo. Por lo tanto, se ha considerado que la inversión de tiempo en el aprendizaje e inclusión en el proyecto no se rentabiliza con los beneficios que pueda aportar el framework.
- El proyecto se ha abordado solo hasta la fase de implementación, han quedado pendientes por lo tanto las fases de testing (solo se ha realizado un testing informal de la interfaz gráfica) y validación con el usuario final. Será necesario por lo tanto realizar un testing formal en un futuro, con diferentes tipos de prueba (unitarias, de integración, de regresión…).
  Con el testing no solo se consigue un producto de mayor calidad sino también que esté más claro qué implementar.
  Incluso hubiera podido ser interesante el testing automático. Existen multitud de herramientas para este fin, por ejemplo Selenium (Burns, 2012). Ésta se enfoca en crear test de integración para la web. Además, Rails posee herramientas para la ceración de test unitarios integrados en el propio framework. Hay que recordar que el testing automático es una gran ayuda a la hora de refactorizar el código, pues permite modificar el software y luego simplemente ejecutar los test automáticos para comprobar que no se ha creado ningún bug en el proceso.
  Por último, mencionar que se consideró no crear hacer un testing más exhaustivo y planificado (y por lo tanto tampoco usar herramientas de testing automático) por falta de tiempo. Se estimó que la inversión de tiempo no se rentabilizaba, por lo menos a corto plazo. Sin embargo a largo plazo sí se rentabilizaría muy probablemente.

Bibliografía

Diapositivas de la asignatura de Ingeniería del Software. (2018). Valencia: Universidad Politécnica de Valencia.

Portal oficial Ruby on Rails para desarrolladores. (06 de 2020). Recuperado el 29 de 06 de 2020, de https://guides.rubyonrails.org/

Álvarez, C. G. (15 de 09 de 2015). Adaptación de las Metodologías Tradicionales Cascada y Espiral. *Adaptación de las Metodologías Tradicionales Cascada y Espiral*. México.

Burns, D. (2012). *Selenium 2 Testing Tools: Beginner's Guide*. Packt.

Comesaña, J. L. (s.f.). *Diseño de interfaces web*.

Decripción caractericas e historia de Git. (s.f.). Recuperado el 06 de 2020, de Decripción caractericas e historia de Git: https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Una-breve-historia-de-Git

Documentación gema Bootstrap GitHub. (s.f.). Recuperado el 30 de 06 de 2020, de Documentación gema Bootstrap GitHub: https://github.com/twbs/bootstrap-rubygem#bootstrap-ruby-gem--

Documentación oficial Devise. (s.f.). Recuperado el 30 de 06 de 2020, de Documentación oficial Devise : http://devise.plataformatec.com.br/

Documentación oficial Embedded Ruby. (s.f.). Recuperado el 06 de 30 de 2020, de Documentación oficial Embedded Ruby: https://ruby-doc.org/stdlib-1.9.3/libdoc/erb/rdoc/ERB.html

Documetación ofcial Ruby on Rails ActiveRecord:Base. (s.f.). Recuperado el 06 de 2020, de Documetación ofcial Ruby on Rails ActiveRecord:Base: https://api.rubyonrails.org/classes/ActiveRecord/Base.html

Gauchat, J. D. (2012). *El gran libro de HTML5, CSS3 y Javascript*. Ediciones Marcombo.

Herrera, R. (2010). *El maldito libro de los descarrilados*.

Lett, J. (2018). *Bottstrap 4 Quick Start*. Bootstrap Creative.

Sitio web oficial Ruby on Rails. (s.f.). Recuperado el 2020 de 06 de 30, de Sitio web oficial Ruby on Rails: https://rubyonrails.org/

Tutorial Bootstrap W3Schools. (s.f.). Recuperado el 06 de 30 de 2020, de Tutorial Bootstrap W3Schools: https://www.w3schools.com/bootstrap4/default.asp

Tutorial CSS W3Schools. (s.f.). Recuperado el 06 de 30 de 2020, de Tutorial CSS W3Schools: https://www.w3schools.com/css/default.asp

Tutorial dirargamas UML Sparxsystems. (s.f.). Recuperado el 06 de 2020, de Tutorial dirargamas UML Sparxsystems: https://sparxsystems.com/resources/tutorials/uml/use-case-model.html

Tutorial HTML W3Schools. (s.f.). Recuperado el 06 de 30 de 2020, de Tutorial HTML W3Schools: https://www.w3schools.com/html/

Tutorial JavaScript W3Schools. (s.f.). Recuperado el 06 de 30 de 2020, de Tutorial JavaScript W3Schools: https://www.w3schools.com/js/

Tutorials Point (I) Pvt. Ltd. (2015). MVC. En T. P. Ltd., *Java Design Patterns* (pág. 121). Tutorials Point (I) Pvt. Ltd.

Glosario

- **Patrón de diseño:** técnica para resolver un problema común en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.
- **Patrones de arquitectura**: ofrece una solución a problemas de arquitectura de software en ingeniería de software.
- **Active record:** patrón de arquitectura encontrado en aplicaciones que almacenan sus datos en bases de datos relacionales. La interfaz de un cierto objeto debe incluir funciones como por ejemplo insertar (INSERT), actualizar (UPDATE), eliminar (DELETE) y propiedades que correspondan de cierta manera directamente a las columnas de la base de datos asociada.
  Este patrón es un enfoque para acceso de datos en una base de datos. Una tabla de la base de datos o vista (view) está envuelta en una clase. Por lo tanto el programador manipula objetos. Luego, mediante métodos de los mismos, se accede a la base de datos, cambiando por lo tanto el estado de la misma.
- **modelo-vista-controlador**: patrón de arquitectura de software que separa en tres capas la representación de los datos (el modelo) de la forma de mostrar los mismos al usuario (la vista), existiendo una capa intermedia entre ambas (el controlador).
- **anamnesis**: información aportada por el paciente y por otros testimonios para confeccionar su historial médico.