Desarrollo de una aplicación web y base de datos para el seguimiento de pacientes con problemas de movilidad cervical

Borja González Díaz

Septiembre 2017

A toda mi familia y en especial a mi padre Peps.

**Resumen**

Con la llegada de las nuevas tecnologías al mercado la mayoría de ámbitos profesionales se están viendo forzados a adaptarse mediante la digitalización de todos sus sistemas, actualmente manuales y prácticamente obsoletos. Quedarse atascado en estos sistemas manuales significaría una gran desventaja a la hora de competir contra otros sistemas digitalizados, que son mucho más rápidos y eficientes, con una necesidad nula o mínima de mantenimiento. Esta disyuntiva también se aplica a los sistemas médicos, que se pueden beneficiar enormemente de la digitalización de la información relativa a los pacientes, permitiendo análisis más avanzados y una mayor agilidad para tratar a los pacientes.

Con un enfoque moderno a los métodos de visualización y recopilación de datos he construido una aplicación web con su correspondiente base de datos, que permiten al usuario acceder rápidamente a los datos de los pacientes permitiendo mostrar toda la información de manera sencilla y clara.

La aplicación web que he desarrollado se centra en el seguimiento de pacientes con problemas de movimiento cervical, específicamente en el rango de movimiento. El rango de movimiento cervical puede verse afectado o disminuido por varias causas como podrían ser los accidentes y más concretamente, el síndrome del latigazo vertical (SLC). Para poder medir el rango de movimiento del paciente, hay disponibles una serie de instrumentos que nos facilitan ésta tarea. Para este proyecto he utilizado un sensor inercial que proporciona una forma flexible y económica de medir la movilidad. Concretamente, el instrumento que vamos a utilizar se llama IMU (Inertial Mearument Unit).

Una parte fundamental de este trabajo ha sido la captura de los requisitos con el cliente, donde el cliente expresó las necesidades que requería para crear una herramienta para la medición del rango de movimiento cervical.

Para empezar el trabajo se han realizado pruebas en las que se han realizado varias mediciones del rango de movimiento de un grupo de 5 personas donde participaban tanto hombres como mujeres, ya que cada sexo tiene un rango de movimiento distinto.

Utilizando varios lenguajes de programación, plataformas y sistemas he conseguido satisfacer las necesidades dicha aplicación, cumpliendo así sus requisitos.

**Agradecimientos**

Tras estos años de carrera …

**Índice general**

1. Introducción 6

1.1. Funcionamiento de la aplicación web 6

2. Estado del arte 6

2.1. Diseño de web estático 7

2.1.1. HTML 7

2.1.2. CSS 7

2.2. NodeJS 7

2.2.1 Express.js 8

2.2.2. App.js 8

2.2.3. Javascript 9

2.2.4. Socket.io 9

2.2.5 Papa Parse 9

2.2.6. Chart.js 10

2.3. Bases de Datos 10

2.3.1. SQLite 11

2.3.2. ACID 11

2.4. Sensor Inercial - IMU 12

2.4.1. Werium Basic Pro 12

3. Diseño 13

3.1. Descripción del problema 13

3.2. Requisitos 13

3.2.1. Requisitos Funcionales 14

3.2.2. Requisitos no Funcionales 14

3.3. Casos de uso 15

3.4. Matriz de trazabilidad 19

3.5. Arquitectura del sistema 19

3.5.1. Aspecto de la aplicación web 20

3.5.2 Aspecto de la Base de datos 22

3.5.3 Estructura del archivo CSV 22

3.6. Especificación de la API 23

4. Implementación 23

4.1. Comunicación Cliente-Servidor 23

4.1.1. Servidor 23

4.1.2. Cliente 24

4.1.3 Ejecución del servidor 25

4.2. SQLite 25

4.2.1. Instalación de SQLite 25

4.2.2 Compatibilidad con el Servidor 26

4.3. Funciones 27

4.3.1. Obtener pacientes 27

4.3.2. Borrar Paciente 29

4.3.3. Añadir un Paciente 31

4.3.4. Obtener datos de movimiento de un paciente 34

4.3.5. Añadir datos de movimiento 37

4.3.6. Borrar un sesión de movimientos 40

4.3.7 Mostrar un grafico de un movimiento 42

4.3.8 Mostrar un grafico de evolución de un movimiento 45

5. Pruebas 45

5.1. Pruebas unitarias 45

5.1.2. Obtener paciente 46

5.1.3. Añadir paciente 46

5.1.4. Borrar un paciente 47

Resultados y conclusiones 48

# 1. Introducción

## 1.1. Funcionamiento de la aplicación web

El objetivo de esta aplicación web es facilitar el acceso y visualización de datos de movimientos cervicales de pacientes.

La aplicación permite el acceso a una base de datos donde se almacena un listado de pacientes y de movimientos. De dicha base de datos podemos consultar movimientos cervicales en todos los planos y observar la evolución con el tiempo de estos movimientos y comprobar si están dentro de unos parámetros de normalidad, dependiendo del sexo del paciente, ya que cada sexo tiene rangos de movilidad cervical distintos.

En la sección de pacientes se listarán todos los pacientes disponibles, pudiendo visualizar su nombre y apellidos, además de tener la opción de añadir o borrar pacientes también podremos acceder a sus datos.

En la sección de datos de cada paciente estará disponible un set de movimientos, que incluye los movimientos en los planos Transversal, Coronal y Sagital, con una fecha asociada que corresponde a la fecha de la medición. Se podrá consultar cada movimiento por separado en forma de gráfico. Además se podrá visualizar, en forma de gráfico, la evolución de cada movimiento a medida que pasa el tiempo. En el gráfico de evolución habrá disponible unos valores de normalidad para que sea posible observar si el paciente entra dentro de dichos valores. Se podrán añadir y borrar sets de movimientos, teniendo en cuenta que para añadir un set de movimientos habrá que seleccionar un archivo local y asociar una fecha de medición a este set de movimientos con la hora incluida por si hubiese más de una medición diaria.

# 

# 2. Estado del arte

## 2.1. Diseño de web estático

### 2.1.1. HTML

HTML, sigla en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al [lenguaje de marcado](https://es.wikipedia.org/wiki/Lenguaje_de_marcado) para la elaboración de [páginas web](https://es.wikipedia.org/wiki/P%C3%A1gina_web). Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del [World Wide Web Consortium](https://es.wikipedia.org/wiki/World_Wide_Web_Consortium" \o "World Wide Web Consortium) ([W3C](https://es.wikipedia.org/wiki/W3C)) o Consorcio WWW, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la [World Wide Web](https://es.wikipedia.org/wiki/World_Wide_Web" \o "World Wide Web)(WWW). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado.[1](https://es.wikipedia.org/wiki/HTML#cite_note-1)​

El lenguaje HTML basa su filosofía de desarrollo en la diferenciación. Para añadir un elemento externo a la página (imagen, vídeo, [script](https://es.wikipedia.org/wiki/Script), entre otros.), este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene solamente texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, HTML busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

### 2.1.2. CSS

**Hojas de Estilo en Cascada** (**C**ascading **S**tyle **S**heets) es el lenguaje utilizado para describir la presentación de documentos [HTML](https://developer.mozilla.org/es/docs/HTML) o [XML](https://developer.mozilla.org/es/docs/XML), esto incluye varios lenguajes basados en [XML](https://developer.mozilla.org/es/docs/XML) como son [XHTML](https://developer.mozilla.org/es/docs/XHTML) o [SVG](https://developer.mozilla.org/es/docs/SVG). CSS describe como debe ser renderizado el elemento estructurado en pantalla, en papel, hablado o en otros medios.

**CSS** es uno de los lenguajes base de la *Open Web* y posee una [especificación estandarizada](http://www.w3.org/Style/CSS/#specs) por parte del W3C. Desarrollado en niveles, CSS1 es ahora obsoleto, CSS2.1 es una recomendación y CSS3, ahora dividido en módulos más pequeños, está progresando en camino al estándar.

## 2.2. NodeJS

**Node.js** es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación [ECMAScript](https://es.wikipedia.org/wiki/ECMAScript" \o "ECMAScript), asíncrono, con [I/O](https://es.wikipedia.org/wiki/I/O) de datos en una [arquitectura orientada a eventos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_dirigida_por_eventos) y basado en el motor [V8](https://es.wikipedia.org/wiki/V8_(motor_JavaScript)) de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, [servidores web](https://es.wikipedia.org/wiki/Servidor_web).



### 2.2.1 Express.js

Express es una infraestructura de aplicaciones web Node.js mínima, flexible y rápida que proporciona un conjunto sólido de características para las aplicaciones web y móviles.

Una API con miles de métodos de programa de utilidad HTTP y middleware a su disposición, la creación de una API sólida es rápida y sencilla.

Express proporciona una delgada capa de características de aplicación web básicas, que no ocultan las características de Node.js, por lo que proporciona un rendimiento muy alto.

### 2.2.2. App.js

App.js es un librería UI (user interface) ligera que permite crear aplicaciones web móviles que se comportan como aplicaciones nativas, sin sacrificar el rendimiento de la aplicación.

* Funciona en las distintas plataformas (Android 2.2+, iOS 4.3+).
* Diseños de la interfaz de usuario específicos para cada plataforma.
* Transiciones nativas configurables.
* La pila de navegación se gestiona de forma automática.
* Widgets de uso para casos de uso generales.

El objetivo de App.js es aportar un punto de comienzo para las aplicaciones web móviles, manejar escenarios generales y mantener la compatibilidad con otras librerías JavaScript comunes.

### 2.2.3. Javascript

JavaScript es un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) [interpretado](https://es.wikipedia.org/wiki/Int%C3%A9rprete_(inform%C3%A1tica)), dialecto del estándar [ECMAScript](https://es.wikipedia.org/wiki/ECMAScript" \o "ECMAScript). Se define como [orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), [basado en prototipos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_basada_en_prototipos), [imperativo](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa), débilmente tipado y dinámico.

Se utiliza principalmente en su forma del [lado del cliente (*client-side*)](https://es.wikipedia.org/wiki/Lado_del_cliente), implementado como parte de un [navegador web](https://es.wikipedia.org/wiki/Navegador_web) permitiendo mejoras en la [interfaz de usuario](https://es.wikipedia.org/wiki/Interfaz_de_usuario) y [páginas web](https://es.wikipedia.org/wiki/P%C3%A1gina_web) dinámicas​ aunque existe una forma de JavaScript del [lado del servidor](https://es.wikipedia.org/wiki/Script_del_lado_del_servidor)(Server-side JavaScript o SSJS). Su uso en [aplicaciones](https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_inform%C3%A1tica) externas a la [web](https://es.wikipedia.org/wiki/World_Wide_Web), por ejemplo en documentos [PDF](https://es.wikipedia.org/wiki/PDF), aplicaciones de escritorio (mayoritariamente [widgets](https://es.wikipedia.org/wiki/Widget)) es también significativo.

### 2.2.4. Socket.io

Socket.io es una librería de JavaScript utilizada para aplicaciones web en tiempo real. Permite una comunicación bidireccional y en tiempo real entre cliente y servidor. Sockets.io está dividido en dos partes, una librería del cliente que se ejecuta en el navegador y una librería del servidor para Node.js. Ambas partes tienen prácticamente la misma API. Al igual que Node.js, Socket.io presenta una arquitectura orientada a eventos.

Generalmente Socket.io utiliza el protocolo WebSocket, y como opción alternativa utiliza el sondeo, pero siempre utiliza la misma interfaz.

Además de ofrecer la funcionalidad de WebSocket, Socket.io es capaz de emitir mensajes a varios sockets, almacenar datos asociados a cada cliente y E/S asíncrona.

### 2.2.5 Papa Parse

Papa parse es un analizador sintáctico que convierte archivos de texto delimitados, principalmente archivos CSV, en estructuras de datos y viceversa. Papa Parse es el analizador más rápido en navegadores para JavaScript. Según la RFC 4180, Papa parse es el analizador sintáctico más fiable y por lo tanto el más recomendable. Posee las siguientes características:

* Muy fácil de usar.
* Parsea ficheros CSV en un entorno local o a través de la red directamente.
* Transmite archivos de gran tamaño (incluso a través de HTTP).
* El análisis sintáctico con la conversión se realizan, además, de forma inversa(JSON a CSV).
* Detección automática de delimitadores.
* Pausa, reanudación y anulación del parseo.
* Papa parse no tiene dependencias.
* Uno de los únicos analizadores que maneja correctamente los saltos de línea y las comillas.
* Puede convertir números y booleanos a sus tipos.
* Soporte del encabezado de fila.

### 2.2.6. Chart.js

Chart.js es una plataforma de JavaScript que nos permite crear gráficos simples pero a la vez flexibles.

* A través del elemento <canvas> se consiguen crear gráficos simples.
* Con Chart.js podemos crear hasta 8 tipos de gráficos personalizables y con los que se puede interactuar.
* Posee un gran rendimiento en todos los navegadores actuales (IE9+).
* Redibuja los gráficos a la hora de ampliar o reducir sobre ellos para conseguir una escala perfecta de granularidad.
* Chart.js funciona muy bien a la hora de visualizar los gráficos en navegadores de otros dispositivos como tablets y móviles

## 2.3. Bases de Datos

Una **base de datos** o **banco de datos** es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente, y debido al desarrollo tecnológico de campos como la [informática](https://es.wikipedia.org/wiki/Inform%C3%A1tica) y la [electrónica](https://es.wikipedia.org/wiki/Electr%C3%B3nica), la mayoría de las bases de datos están en formato digital, siendo este un componente electrónico, por tanto se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos.

Existen [programas](https://es.wikipedia.org/wiki/Programa_inform%C3%A1tico) denominados [sistemas gestores de bases de datos](https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_bases_de_datos), abreviado SGBD (del inglés *database management system* o DBMS), que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Las propiedades de estos DBMS, así como su utilización y administración, se estudian dentro del ámbito de la informática.

Las aplicaciones más usuales son para la gestión de empresas e instituciones públicas; También son ampliamente utilizadas en entornos científicos con el objeto de almacenar la información experimental.



### 2.3.1. SQLite

SQLite es un sistema de gestión de [bases de datos relacional](https://es.wikipedia.org/wiki/Base_de_datos_relacional) (DBMS). Éste sistema es compatible con [ACID](https://es.wikipedia.org/wiki/ACID) y está contenido en una bibloteca en C que se podría considerar pequeña para ciertos casos (275kiB). El problema del tamaño se ha resuelto en la reciente versión · del sistemas que permite bases de datos de hasta 2 Terabytes. SQLite es un proyecto de [dominio público](https://es.wikipedia.org/wiki/Dominio_p%C3%BAblico" \o "Dominio público) creado por [D. Richard Hipp](https://es.wikipedia.org/w/index.php?title=D._Richard_Hipp&action=edit&redlink=1).

A diferencia de los [sistema de gestión de bases de datos](https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_bases_de_datos" \o "Sistema de gestión de bases de datos) cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

En su versión 3, como hemos mencionado antes, SQLite permite bases de datos de hasta 2 Terabytes de tamaño, y también permite la inclusión de campos tipo BLOB

### 2.3.2. ACID

En bases de datos se denomina **ACID** a las características de los parámetros que permiten clasificar las [transacciones](https://es.wikipedia.org/wiki/Transacci%C3%B3n_de_base_de_datos) de los [sistemas de gestión de bases de datos](https://es.wikipedia.org/wiki/SGBD). Cuando se dice que es *ACID compliant* se indica -en diversos grados- que éste permite realizar transacciones.

En concreto **ACID** es un acrónimo de **A**tomicity, **C**onsistency, **I**solation and **D**urability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

* [**Atomicidad**](https://es.wikipedia.org/wiki/Atomicidad)**:** Si cuando una operación consiste en una serie de pasos, bien todos ellos se ejecutan o bien ninguno, es decir, las transacciones son completas.
* [**Consistencia**](https://es.wikipedia.org/wiki/Consistencia_de_datos)**:** (*Integridad*). Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de *Integridad* de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido. "La Integridad de la Base de Datos nos permite asegurar que los datos son exactos y consistentes, es decir que estén siempre intactos, sean siempre los esperados y que de ninguna manera cambien ni se deformen. De esta manera podemos garantizar que la información que se presenta al usuario será siempre la misma."
* [**Aislamiento**](https://es.wikipedia.org/wiki/Aislamiento_(ACID))**:** Esta propiedad asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.  Esta propiedad define cómo y cuándo los cambios producidos por una operación se hacen visibles para las demás operaciones concurrentes. El aislamiento puede alcanzarse en distintos niveles, siendo el parámetro esencial a la hora de seleccionar SGBDs.
* [**Durabilidad**](https://es.wikipedia.org/wiki/Datos_persistentes)**:** (*Persistencia*). Esta propiedad asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema y que de esta forma los datos sobrevivan de alguna manera.

Cumpliendo estos 4 requisitos un sistema gestor de bases de datos puede ser considerado *ACID Compliant*.

## 2.4. Sensor Inercial - IMU

Mediante el uso combinado de acelerómetros y giroscopios, hay ciertos aparatos electrónicos que miden y dan información acerca de la velocidad, orientación y fuerzas gravitacionales de un objeto en movimiento.orie﷽﷽﷽﷽﷽﷽iñon locidaddratos electrometros rticular tera

### 2.4.1. Werium Basic Pro

Basic Pro es una herramienta que se ha diseñado para facilitar el trabajo de los terapeutas. Es un sistema muy preciso a la hora de medir el rango de movimiento articular cervical.  Captura los movimientos en los planos de flexo extensión, inclinaciones laterales y rotación.

Está compuesto por tres giroscopios y tres acelerómetros que a su vez integran un reloj que permite asociar valores temporales a las medidas. Éste tipo de herramienta implementa tres ejes ortogonales que referencian a los sensores y así se puede asignar a cada eje un acelerómetro y un giroscopio. La información que proporciona un IMU es la velocidad angular y la aceleración lineal en cada eje, asociada a los valores temporales.



Sensor inercial Werium

# 3. Diseño

En este capítulo se intenta analizar en profundidad el problema que se pretende resolver, intentando abarcar la mayor cantidad de información posible y estudiando los principales aspectos que habrían de cubrirse en el desarrollo del proyecto. En este punto deberían surgir las características principales del sistema a desarrollar, por lo que se establecen también los casos de uso que deberá poder efectuar el sistema resultado del proyecto para cumplir los requisitos que se establezcan.

## 3.1. Descripción del problema

El objetivo principal es construir una aplicación web con una base de datos asociada para el seguimiento de pacientes con problemas de movilidad cervical. Esta aplicación web permitirá el acceso de uno o más usuarios o clientes, y que estos puedan trabajar con la base de datos obteniendo información de los pacientes. Para describir el problema en cuestión voy a seguir un proceso que se divide en varios apartados. Empezaré con el diseño de la arquitectura del sistema para satisfacer las necesidades descritas previamente, seguiré con la implementación de dicho sistema, y terminaré describiendo el despliegue de las aplicaciones y pruebas.

## 3.2. Requisitos

La fase de análisis incluye el estudio y la definición de los requisitos del sistema. Los requisitos definen en detalle los servicios que el sistema debería poder ofrecer, identificando las restricciones y características de este.

### 3.2.1. Requisitos Funcionales

RF1 – Una vez dentro la aplicación web, el sistema debe mostrar una sección de inicio con una breve descripción de las distintas funcionalidades de la pagina.

RF2 – El sistema ofrecerá un listado de los clientes con su nombre, apellidos y un link para acceder a sus datos de movimiento.

RF3 – El usuario tendrá la opción de añadir pacientes, saltando un error en el caso de que el usuario no rellene algún dato solicitado.

RF4 – El usuario tendrá la opción de borrar pacientes, borrando así todos sus datos asociados.

RF5 – Para cada paciente, el sistema mostrará una tabla con los tres movimientos disponibles.

RF6 – El sistema deberá ser capaz de mostrar las distintas sesiones de movimientos.

RF7 – El sistema debe proporcionar a los clientes la opción de añadir datos de movimientos con una fecha asociada.

RF8 - El sistema debe proporcionar a los clientes la opción de borrar datos de movimientos.

RF9- El sistema ofrece la opción de mostrar la evolución de cada movimiento.

### 3.2.2. Requisitos no Funcionales

RNF1 - El sistema debe permitir el acceso a la aplicación web desde cualquier dispositivo. Una vez activo el servidor, cualquier usuario puede acceder a la página web desde cualquier dispositivo.

RNF2 - Solo se admitirán ficheros CSV. Esto quiere decir que a la hora de que el usuario seleccione un archivo local para añadir un set de datos, solo se le permitirá seleccionar un archivo del tipo CSV y no cualquier otro.

RNF3 - Dentro de los gráficos de evolución el sistema marcará unos límites de normalidad dependiendo del sexo del paciente.

RNF4 - El sistema deberá ser capaz de almacenar los datos de los pacientes entre sesiones.

RNF5 - Cuando se realice un cambio en un dispositivo (añadir paciente, borrar sesión de datos, etc…) se reflejará automáticamente en el resto de dispositivos, mostrando la tabla correspondiente con el dato añadido o sin el dato borrado.

RNF6 – Cada vez que el usuario muestre un gráfico, el sistema debe ser capaz de eliminar el gráfico anterior y poner en su lugar el nuevo gráfico elegido por el usuario.

## 3.3. Casos de uso

Los casos de uso representan las diferentes posibles interacciones que pueden existir entre los diferentes actores dentro de un sistema. Los actores de un sistema son entidades externas al sistema que pueden interactuar con el. Los actores pueden ser humanos u objetos (servidores, aplicaciones, servicios...). Los actores que interactúan con nuestra aplicación son los siguientes:

* Usuarios del sistema a través del navegador de cualquier dispositivo.

CU1: Acceso a la página web.

El usuario introduce la dirección del servidor en su navegador accediendo instantáneamente al servidor.

Actores implicados: Usuario/cliente.

Precondiciones: El sistema debe estar correctamente funcionando.

Secuencia de acciones:

1. El cliente accede a la aplicación web introduciendo la dirección IP del servidor en su navegador.

CU2: Acceso a la pestaña de pacientes/Listado de pacientes.

El usuario accede a un listado de pacientes.

Actores implicados: Usuario/cliente.

Precondiciones: Acceso a la página web.

Secuencia de acciones:

1. El cliente accede a la aplicación web introduciendo la dirección IP del servidor en su navegador.
2. El cliente accede a la sección de pacientes mediante la pestaña de pacientes.
3. El navegador solicita la lista de pacientes a la base de datos.
4. El cliente recibe la lista de pacientes y se la presenta al usuario en una tabla.

CU3: Añadir un paciente.

Actores implicados: Usuario/cliente.

Precondiciones: Acceso a la pestaña de pacientes/Listado de pacientes.

Secuencia de acciones:

1. El cliente accede a la aplicación web introduciendo la dirección IP del servidor en su navegador.
2. El cliente accede a la sección de pacientes mediante la pestaña de pacientes.
3. El usuario introduce el nombre, apellidos y sexo del paciente y presiona el botón “Añadir”.
4. El navegador manda la petición, para añadir el paciente, al servidor.
5. El servidor añade el nuevo paciente a la base de datos, persistiendo los cambios.

Post-condiciones: El nuevo paciente tiene que ser mostrado inmediatamente en el listado de pacientes, sin que el usuario tenga que refrescar la página.

CU4: Borrar un paciente.

Actores implicados: Usuario/cliente.

Precondiciones: Acceso a la pestaña de pacientes/Listado de pacientes.

Secuencia de acciones:

1. El cliente accede a la aplicación web introduciendo la dirección IP del servidor en su navegador.
2. El cliente accede a la sección de pacientes mediante la pestaña de pacientes.
3. El navegador solicita la lista de pacientes a la base de datos.
4. El cliente recibe la lista de pacientes y se la presenta al usuario en una tabla.
5. El usuario elige el paciente a borrar y presiona el botón de borrado correspondiente, ya que cada paciente tiene su botón de borrado.
6. El navegador pregunta al usuario si desea realizar esta operación, teniendo en cuenta que borraría todos los datos de movimientos asociados a ese paciente.
7. El usuario acepta la operación, teniendo la opción de cancelarla.
8. El navegador manda la petición para borrar el paciente al servidor.
9. El servidor borra al paciente y todos sus datos de movimientos asociados de la base de datos, persistiendo los cambios.

Post-condiciones: La lista de pacientes ha de actualizarse, mostrando todos los pacientes menos el borrado. Se actualiza de forma automática. Todos los movimientos asociados a este paciente tienen que desaparecer.

CU5: Listado de movimientos de un paciente.

El usuario accede a un listado de movimientos del paciente.

Actores implicados: Usuario/cliente.

Precondiciones: Acceso a la pestaña de pacientes/Listado de pacientes.

Secuencia de acciones:

1. El cliente accede a la aplicación web introduciendo la dirección IP del servidor en su navegador.
2. El cliente accede a la sección de pacientes mediante la pestaña de pacientes.
3. El navegador solicita la lista de pacientes a la base de datos.
4. El cliente recibe la lista de pacientes y se la presenta al usuario en una tabla.
5. El cliente elige el paciente del que quiere mostrar sus datos de movimiento presionando el botón adecuado, ya que cada paciente tiene su propio botón para mostrar sus datos.
6. El navegador se posiciona en la pestaña de Datos.
7. El navegador solicita la lista de movimientos del paciente a la base de datos.
8. El navegador recibe la lista de movimientos y se la presenta al usuario en una tabla.

CU6: Añadir datos de movimiento.

El usuario añade un set de movimientos al paciente.

Actores implicados: Usuario/cliente.

Precondiciones: Acceso al listado de movimientos de un paciente.

Secuencia de acciones:

1. El cliente accede a la aplicación web introduciendo la dirección IP del servidor en su navegador.
2. El cliente accede a la sección de pacientes mediante la pestaña de pacientes.
3. El navegador solicita la lista de pacientes a la base de datos.
4. El cliente recibe la lista de pacientes y se la presenta al usuario en una tabla.
5. El cliente elige el paciente del que quiere mostrar sus datos de movimiento presionando el botón adecuado, ya que cada paciente tiene su propio botón para mostrar sus datos.
6. El navegador se posiciona en la pestaña de Datos.
7. El usuario selecciona un archivo, que solo podrá ser del tipo CSV, y una fecha de medición del set movimientos y presiona el botón de “Añadir datos”.
8. El navegador manda la petición para añadir los datos de movimiento al servidor.
9. El servidor añade el set de datos de movimientos al la base de datos persistiendo los cambios.

Post-condiciones: El nuevo set de datos tiene que ser mostrado inmediatamente en el listado de datos de movimientos, sin que el usuario tenga que refrescar la página.

CU7: Borrar datos de movimiento.

El usuario borra un set de movimientos del paciente.

Actores implicados: Usuario/cliente.

Precondiciones: Acceso al listado de movimientos de un paciente.

Secuencia de acciones:

1. El cliente accede a la aplicación web introduciendo la dirección IP del servidor en su navegador.
2. El cliente accede a la sección de pacientes mediante la pestaña de pacientes.
3. El navegador solicita la lista de pacientes a la base de datos.
4. El cliente recibe la lista de pacientes y se la presenta al usuario en una tabla.
5. El cliente elige el paciente del que quiere mostrar sus datos de movimiento presionando el botón adecuado, ya que cada paciente tiene su propio botón para mostrar sus datos.
6. El navegador se posiciona en la pestaña de Datos.
7. El navegador solicita la lista de movimientos del paciente a la base de datos.
8. El navegador recibe la lista de movimientos y se la presenta al usuario en una tabla.
9. El usuario elige los movimientos a borrar y presiona el botón de borrado correspondiente, ya que cada set de movimientos tiene su botón de borrado.
10. El navegador pregunta si desea realizar esta operación.
11. El usuario acepta la operación, teniendo la opción de cancelarla.
12. El navegador manda la petición para borrar los datos de movimiento al servidor.
13. El servidor borra el set de datos de movimiento del paciente de la base de datos y persiste los cambios.

Post-condiciones: La nueva lista de movimientos, sin el set de movimientos borrado, tiene que ser mostrada inmediatamente en el listado de movimientos, sin que el usuario tenga que refrescar la página.

CU8: Mostar datos del paciente.

El usuario muestra un movimientos del paciente.

Actores implicados: Usuario/cliente.

Precondiciones: Acceso al listado de movimientos de un paciente.

Secuencia de acciones:

1. El cliente accede a la aplicación web introduciendo la dirección IP del servidor en su navegador.
2. El cliente accede a la sección de pacientes mediante la pestaña de pacientes.
3. El navegador solicita la lista de pacientes a la base de datos.
4. El cliente recibe la lista de pacientes y se la presenta al usuario en una tabla.
5. El cliente elige el paciente del que quiere mostrar sus datos de movimiento presionando el botón adecuado, ya que cada paciente tiene su propio botón para mostrar sus datos.
6. El navegador se posiciona en la pestaña de Datos.
7. El navegador solicita la lista de movimientos del paciente a la base de datos.
8. El navegador recibe la lista de movimientos y se la presenta al usuario en una tabla.
9. El usuario elige el movimiento (Sagital, Coronal, Transversal) con su fecha asociada presionando en el botón adecuado.
10. El navegador muestra por pantalla un gráfico con los datos requeridos.

Post-condiciones: El navegador tiene que borrar el contenido del gráfico anterior (en el caso de que exista) y graficar los datos requeridos.

CU9: Mostar datos de evolución del paciente.

El usuario muestra la evolución de un movimiento del paciente.

Actores implicados: Usuario/cliente.

Precondiciones: Acceso al listado de movimientos de un paciente.

Secuencia de acciones:

1. El cliente accede a la aplicación web introduciendo la dirección IP del servidor en su navegador.
2. El cliente accede a la sección de pacientes mediante la pestaña de pacientes.
3. El navegador solicita la lista de pacientes a la base de datos.
4. El cliente recibe la lista de pacientes y se la presenta al usuario en una tabla.
5. El cliente elige el paciente del que quiere mostrar sus datos de movimiento presionando el botón adecuado, ya que cada paciente tiene su propio botón para mostrar sus datos.
6. El navegador se posiciona en la pestaña de Datos.
7. El navegador solicita la lista de movimientos del paciente a la base de datos.
8. El navegador recibe la lista de movimientos y se la presenta al usuario en una tabla.
9. El usuario elige la evolución del movimiento (Sagital, Coronal, Transversal) que quiere mostrar presionando el botón “Evolución del movimiento”.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Casos de Uso | RF1 | RF2 | RF3 | RF4 | RF5 | RF6 | RF7 | RF8 | RF9 |
| CU1 | ✓ |  |  |  |  |  |  |  |  |
| CU2 |  | ✓ |  |  |  |  |  |  |  |
| CU3 |  |  | ✓ |  |  |  |  |  |  |
| CU4 |  |  |  | ✓ |  |  |  |  |  |
| CU5 |  |  |  |  | ✓ |  |  |  |  |
| CU6 |  |  |  |  |  | ✓ |  |  |  |
| CU7 |  |  |  |  |  |  | ✓ |  |  |
| CU8 |  |  |  |  |  |  |  | ✓ |  |
| CU9 |  |  |  |  |  |  |  |  | ✓ |

1. El navegador muestra por pantalla un gráfico con los datos requeridos

Post-condiciones: El navegador tiene que borrar el contenido del gráfico anterior (en el caso de que exista) y graficar los datos requeridos.

## 3.4. Matriz de trazabilidad

## 

## 3.5. Arquitectura del sistema

### 3.5.1. Diseño visual (Storyboard) de la aplicación web

#### 3.5.1.1 Sección de Inicio



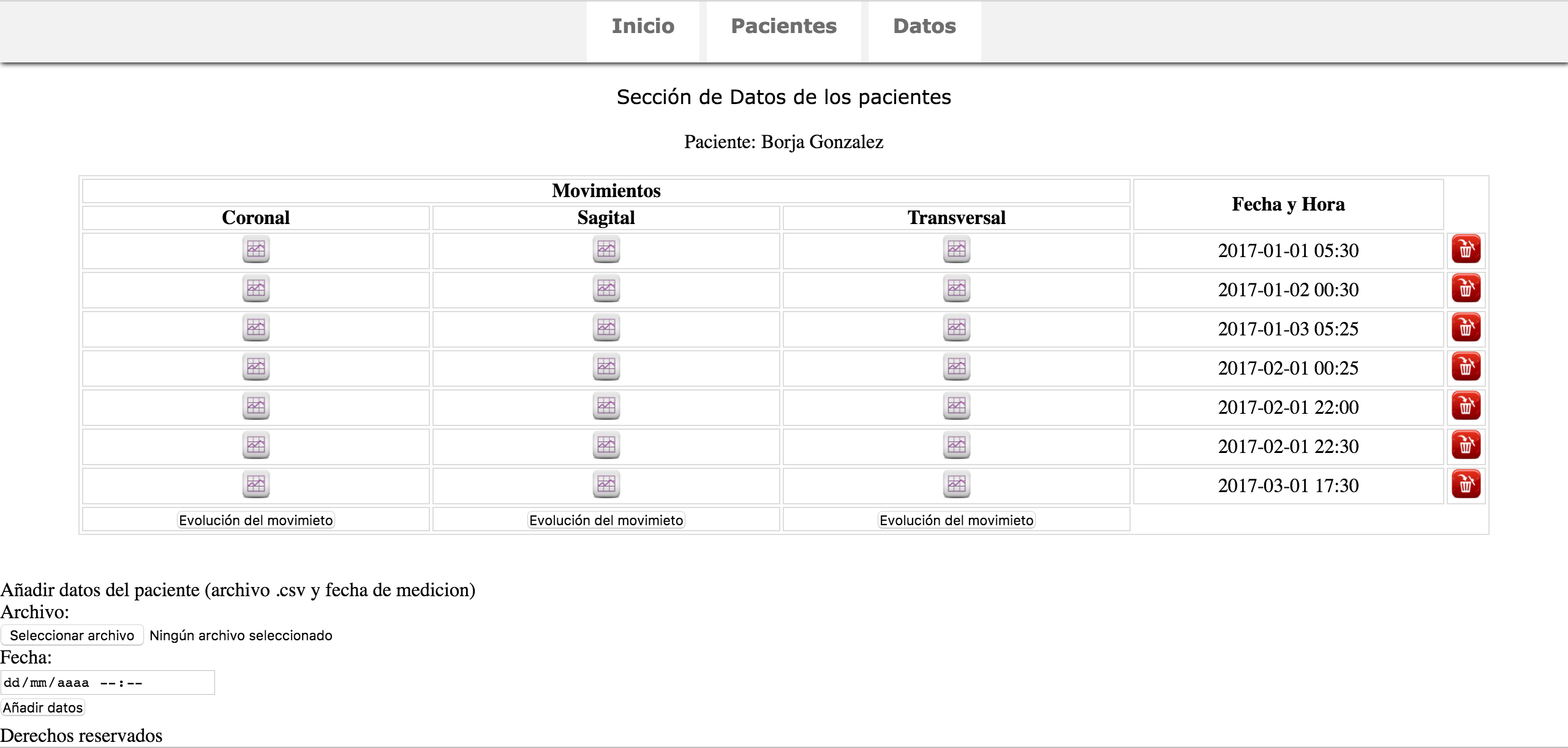
En la sección de inicio encontramos una breve descripción de cómo funciona la página web.

#### 3.5.1.2 Sección de Pacientes

## 

En la sección de pacientes encontramos un listado de pacientes con el nombre, apellidos, un botón para acceder a los datos del paciente y otro botón para borrar al paciente. También existe la posibilidad de añadir pacientes, introduciendo su nombre, apellidos y sexo.

#### 3.5.1.3 Sección de Datos

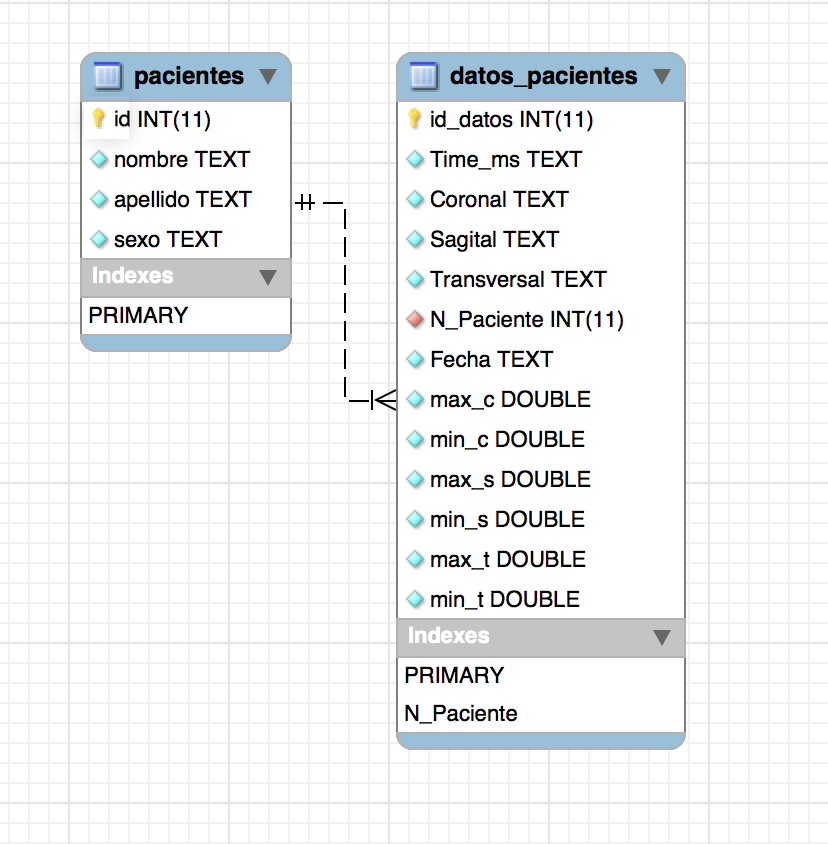


En la sección de datos tenemos un listado de conjuntos de movimientos (movimientos en los planos Coronal, Transversal y Sagital) con una fecha de medición asociada a cada conjunto. A demás es posible añadir un conjunto de datos con su fecha asociada y borrar un conjunto de datos.

### 3.5.2 Esquema del modelo de datos

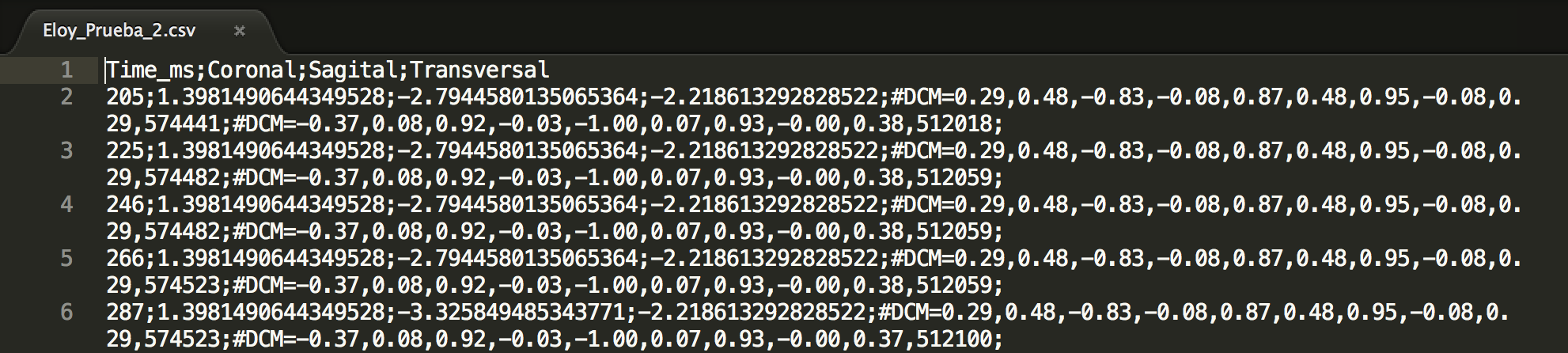
#### 3.5.2.1 EER – Enhanced Entity-relationship model

Un modelo entidad-relación o diagrama entidad-relación (a veces denominado por sus siglas en inglés, E-R "Entity relationship"; en español DER: "Diagrama de Entidad-Relación") es una herramienta para el [modelado de datos](https://es.wikipedia.org/wiki/Modelo_de_datos) que permite representar las entidades relevantes de un [sistema de información](https://es.wikipedia.org/wiki/Sistema_de_informaci%C3%B3n) así como sus interrelaciones y propiedades.



La base de datos está formada por dos tablas relacionadas (1…n) , lo que quiere decir que por cada paciente podrán existir varios sets de datos. En este tipo de relaciones siempre existe un Foreign Key, que es la clave que relaciona el set de datos con el paciente.

### 3.5.3 Estructura del archivo CSV



En el archivo CSV podemos observar un documento de texto delimitado por un punto y coma (semicolon) que divide los datos en tiempo, movimiento coronal, movimiento sagital, movimiento transversal y datos adicionales que ayudan a calcular los previamente dichos. Éstos últimos datos no serán utilizados ya que no son necesarios.

## 3.6. Especificación de la API

# 4. Implementación

En este capitulo se explica paso a paso el proceso de implementación que se ha llevado a cabo para el sistema de acuerdo a los requisitos funcionales y no funcionales establecidos. A continuación se va a poder visualizar algunos extractos de código y ficheros para que el lector pueda comprender en detalle los procesos y funcionalidades implementadas.

## 4.1. Comunicación Cliente-Servidor

Como ya sabemos he utilizado sockets.io para establecer una comunicación bidireccional entre el servidor y el navegador (cliente). A continuación podremos ver como he implementado estos sockets en mi aplicación web.

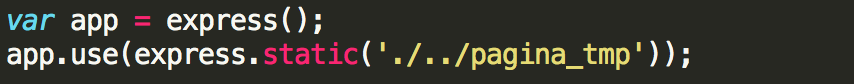
### 4.1.1. Servidor

Para establecer una conexión entre el cliente y el servidor, la parte del servidor realiza los siguientes pasos:

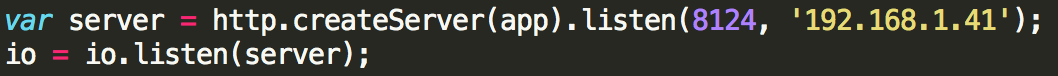
1. Solicitar las librerías correspondientes para que el servidor haga uso de sus funcionalidades (express, socket.io, http):



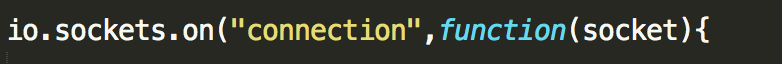
2. Utilizando el middleware estático, especificamos la carpeta pública del servidor para disponer de las páginasHTML. Éstas páginasserían la parte del navegador donde está toda la parte que visualiza el usuario en su navegador. En mi caso las códigos HTML (index.html, paciente.html y evolución.html) se encuentran en la carpeta pagina\_tmp.



3. Creamos una instancia “server” con un puerto asociado (8124) y una dirección IP que corresponde a la dirección que se asocia a mi terminal. La combinación de dirección IP y puerto será la que el usuario tendrá que introducir en su navegador (192.168.1.41:8124) para acceder a la aplicación web. Con la instancia creada se iniciara una comunicación websocket pasando la instancia como parámetro.



4. Asociamos una función callback que se ejecutará cuando el cliente visite la página y establezca un websocket. A partir de aquí el servidor estará escuchando permanentemente a peticiones del cliente y mandará respuestas según lo que se solicite.



### 4.1.2. Cliente

La parte del cliente es muy sencilla ya que solo tiene que inicializar una conexión que activará la función callback del servidor.



A partir de aquí el cliente utilizará la instancia creada “socket” para enviar datos al servidor o recibir datos del servidor.

### 4.1.3 Despliegue del servidor

Para que la comunicación entre el cliente y el servidor funcione es necesario ejecutar el código del servidor para que empiece a escuchar peticiones de clientes.

Es un paso muy sencillo que requiere posicionarse en el terminal de la máquina donde se encuentre el código del servidor y cambiar al directorio donde se encuentre el fichero JavaScript ejecutable.

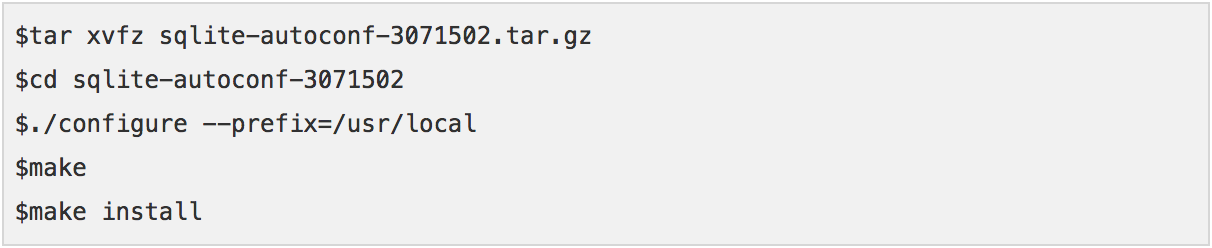


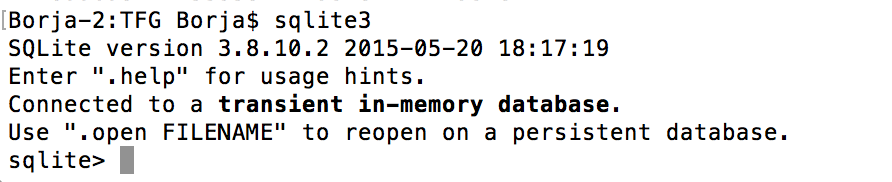
## 4.2. SQLite

### 4.2.1. Instalación de SQLite

En mi caso he realizado todo el desarrollo en MAC OS X. En la última versión de Mac os x el paquete de SQLite viene preinstalado pero en caso de no tener la última versión, a continuación se mostrará como instalar el paquete:

1. Ir a la página de descargas de SQLite (<https://www.sqlite.org/download.html>) y descargue *sqlite-autoconf - \*. tar.gz*.
2. Ejecute el siguiente comando en su terminal:



La secuencia de pasos anteriormente descrita terminará con la instalación de SQLite en la máquina Mac OS X. Se puede verificar la instalación mediante la emisión del siguiente comando:

### 4.2.2 Compatibilidad con el Servidor

Para que el servidor pueda hacer un uso adecuado de SQLite necesitará un script llamado sql.js, que es el que le permitirá manejar la base de datos de los pacientes.

#### 4.2.2.1. Sql.js

Sql.js es un puerto de SQLite a JavaScript. Utiliza una base de datos virtual que se almacena en memoria y por los tanto los cambios realizados a la base de datos no se persisten. Lo interesante y que es muy funcional para este desarrollo es que permite importar cualquier fichero SQLite y exportar la base de datos creada como un array escrito en JavaScript. Usaremos el array exportado para poder persistir los cambios realizados con la ayuda de node.js.

#### 4.2.2.2. Uso de sql.js con node.js

Todas las operaciones que se realizan para manejar la base de datos (consultar, borrar, ordenar o insertar) requieren de una lectura de la base de datos y/o de una operación para guardar los cambios realizados en la base de datos.

1. Lectura de una base de datos de memoria:



1. Escritura de una base de datos en memoria:

Hace falta convertir el resultado de db.export a un buffer



## 4.3. Funciones

A continuación expondré el funcionamiento de las distintas funciones que piden, añaden, borran u organizan la base de datos. Todas estas funciones se realizan gracias a la comunicación entre el navegador y el servidor, que utilizan sockets para establecer una comunicación bidireccional, así como node.js para pedir datos a la base de datos o para persistir cambios en la base de datos.

### 4.3.1. Obtener pacientes

Para que el navegador obtenga la lista de pacientes de la base de datos es necesario que se realizar una conexión bidireccional con el servidor y que desde el servidor se haga una consulta a la base de datos, enviando el resultado de la consulta al navegador mediante el websocket establecido. Todo esto ha sido explicado en los apartados anteriores. En este apartado veremos como se realizan todas las operaciones en conjunto, además de las consultas de SQL.

#### 4.3.1.1. Funcionalidad en el lado del cliente

Cuando el usuario entra en la sección de pacientes, se genera una tabla y se llama a una función que devuelve un array de pacientes. Utilizando un bucle for que recorre el array rellenamos la tabla.



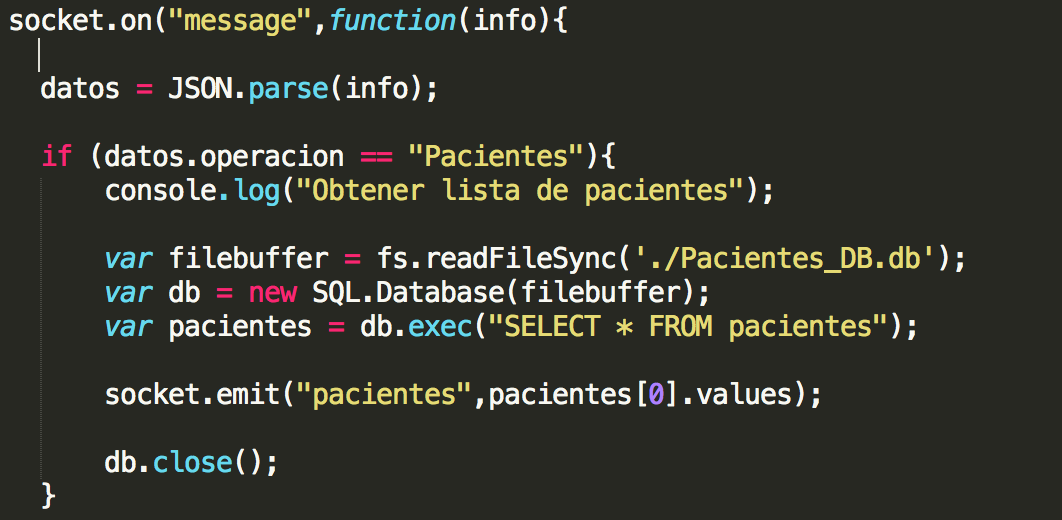
La función get\_paciente\_node() es la que se encarga de establecer la conexión con el servidor a través de un websocket.



Como se mencionó en la sección 4.1.2, el cliente utiliza la instancia socket para escuchar al servidor a través de socket.on y para enviar mensajes a través de socket.send. Todas las solicitudes al servidor se realizan enviando un JSON con una “cabecera” llamada operación que permitirá al servidor distinguir entre las distintas operaciones, y así devolver unos datos específicos. En este caso la cabecera contiene la operación Pacientes.

#### 4.3.1.2. Funcionalidad en el lado del servidor

Ahora podremos ver como el servidor maneja la solicitud para enviar el listado de pacientes al cliente.



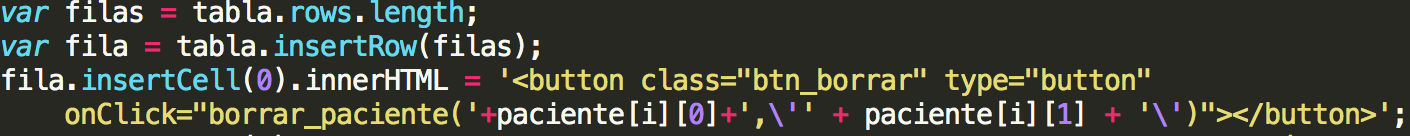
La variable datos es la que contiene el JSON con la cabecera que ha enviado el navegador. El servidor comprueba si esta cabecera es equivalente a pacientes y así reconoce la petición del cliente. Podemos observar la operación de lectura de la base de datos descrita en el apartado 4.2.2.2. Al tener acceso a la base de datos se realiza una consulta para obtener los pacientes mediantes db.exec(). El resultado de esta consulta (array de pacientes) se envía al cliente mediante el websocket con la operación socket.emit() y el cliente muestra los pacientes en la tabla.

### 4.3.2. Borrar Paciente

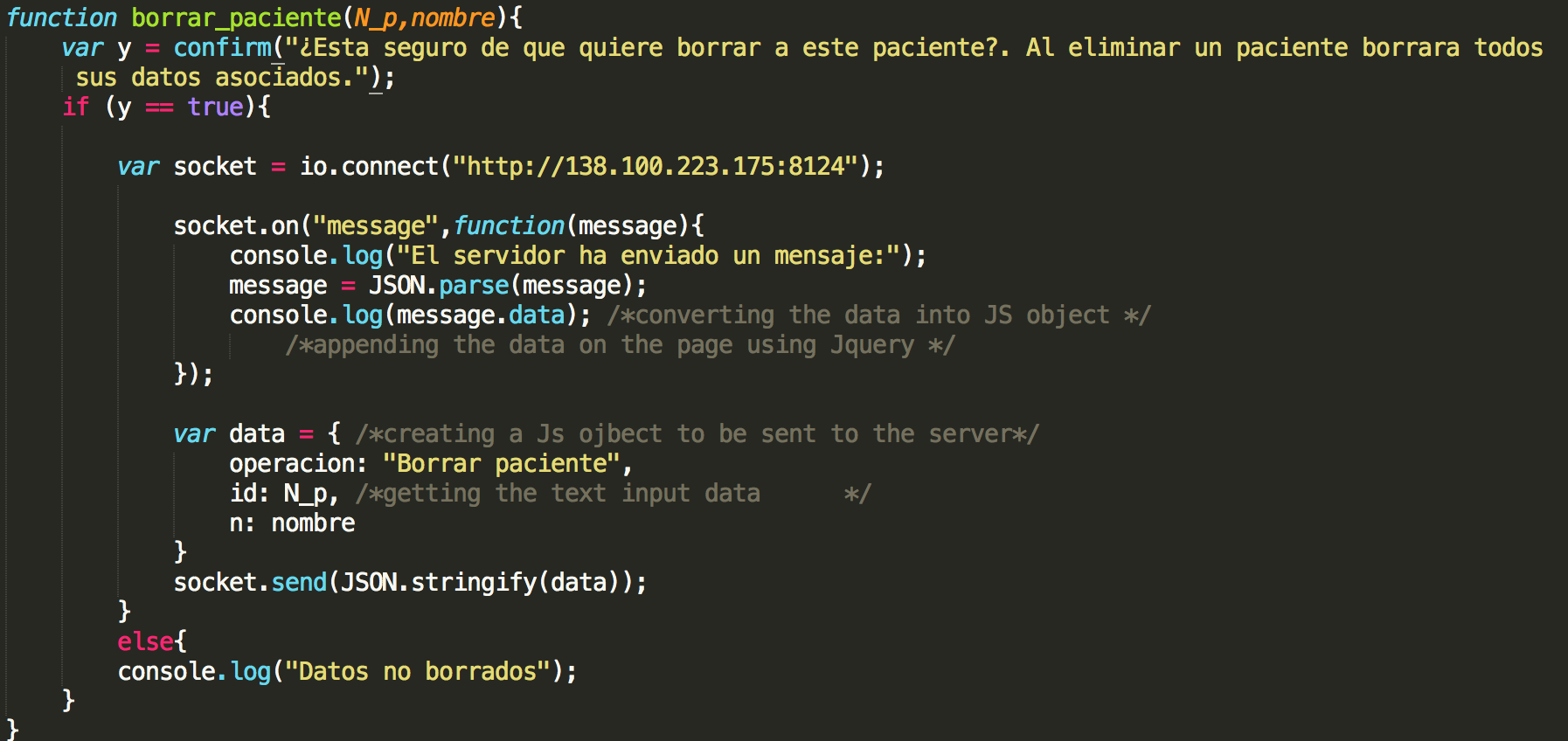
Para borrar un paciente seguiremos una dinámica similar a la de añadir un paciente. Cambiará la cabecera de la operación y el comando SQL que en vez de extraer datos de la base de datos borrará datos de la tabla de pacientes y de la tabla de datos de paciente, ya que si un paciente tiene datos de movimientos y lo borramos, todos sus datos de movimientos asociados serán eliminados.

#### 4.3.2.1. Funcionalidad en el lado del cliente

Cuando el usuario entra en la sección de pacientes, se genera una tabla y se llama a una función que devuelve un array de pacientes. Utilizando un bucle for que recorre el array rellenamos la tabla. Dentro de cada fila se rellenan varias elementos, incluyendo nombre, apellidos, botón de acceso a los datos y botón de borrado. El usuario presionara el botón de borrado del paciente que quiera borrar.



En este trozo de código vemos que una de las filas insertadas es un botón y cuando se presiona sobre este botón se llama a una función llamada borrar\_paciente() a la que le pasamos el id del paciente y el nombre.



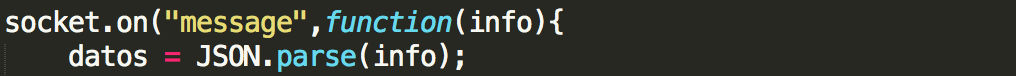
En este caso le salta un mensaje de confirmación al usuario para asegurar que quiera borrar al paciente y sus datos de movimientos asociados. Si confirma la operación se establecerá un websocket con el servidor donde el navegador escuchará al servidor para recibir la confirmación de conexión. Después el navegador enviará un objeto JSON con la cabecera “Borrar paciente”, el identificador único del paciente y su nombre.

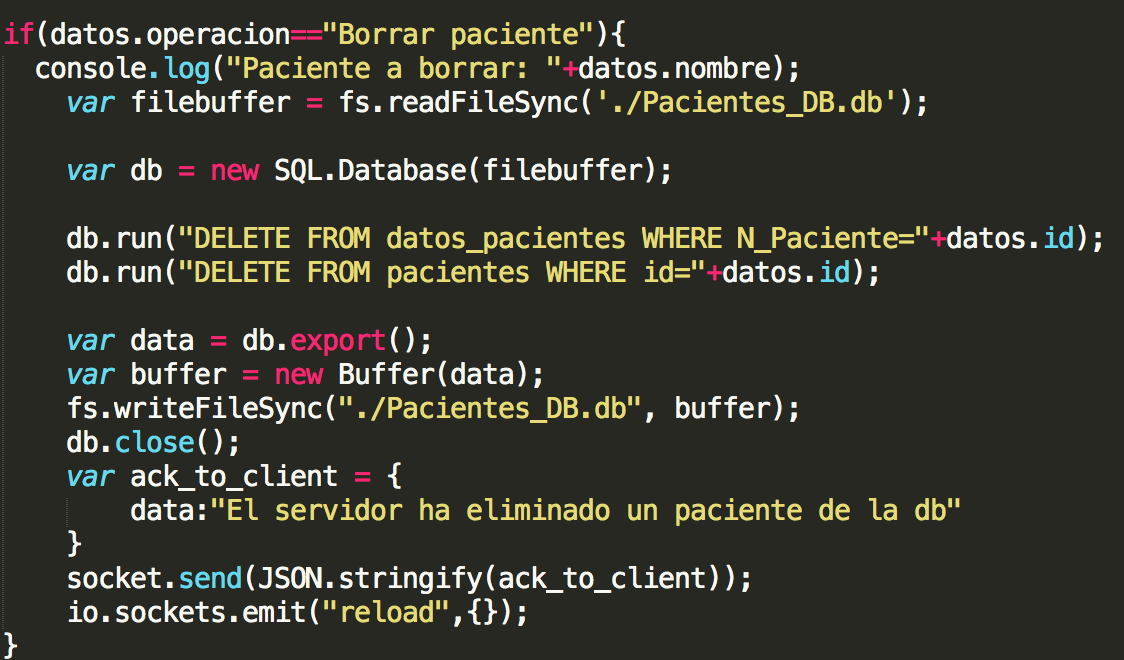
Para este caso es necesario que cuando borremos el paciente se actualice la tabla con los pacientes y así no se muestre el paciente borrado. Para realizar esta función el servidor manda un mensaje que el cliente recibe y le sirve para saber cuando tiene que actualizar el contenido de la tabla.



Éste código se encuentra en el navegador y espera a que el servidor confirme que se han borrado los datos. Una vez recibida la confirmación, el navegador realiza un location.reload() que básicamente actualiza la página web, lo que obliga a que se vuelva a llamar a la función que lista los pacientes. En esta ocasión se mostrará la lista de pacientes actualizada.

#### 4.3.2.2. Funcionalidad en el lado del servidor





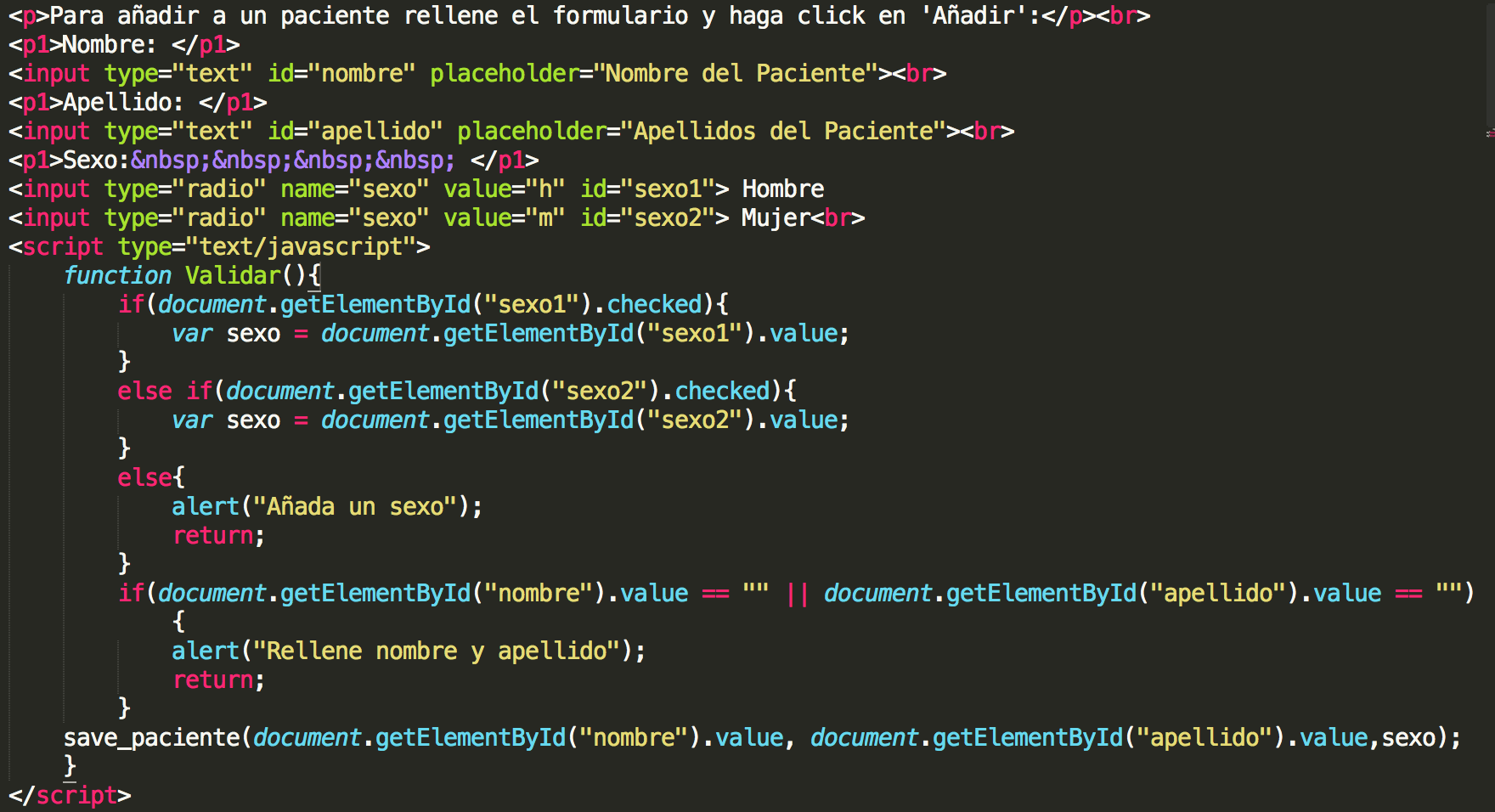
Como vimos para el caso para añadir un paciente el servidor escucha mediante socket.on(). Cuando recibe un mensaje comprueba su cabecera y al reconocer la operación “Borrar Paciente” realiza la conexión con la base de datos y elimina al paciente y cualquier set de movimientos que tenga. A continuación guarda los cambios y cierra la base de datos. Como hemos dicho en la parte del cliente, cuando se realizan los cambios el servidor ejecuta la función io.sockets.emit(). Básicamente fuerza a cualquier cliente conectado a actualizar su lista de clientes, por lo que si hay varios dispositivos conectados, en todos estos se actualizarán los cambios.

### 4.3.3. Añadir un Paciente

Para este caso, existirá un procesa similar al de borrar un paciente. Cambiará la cabecera u operación enviada al servidor y el comando SQL que añadirá datos a la base de datos.

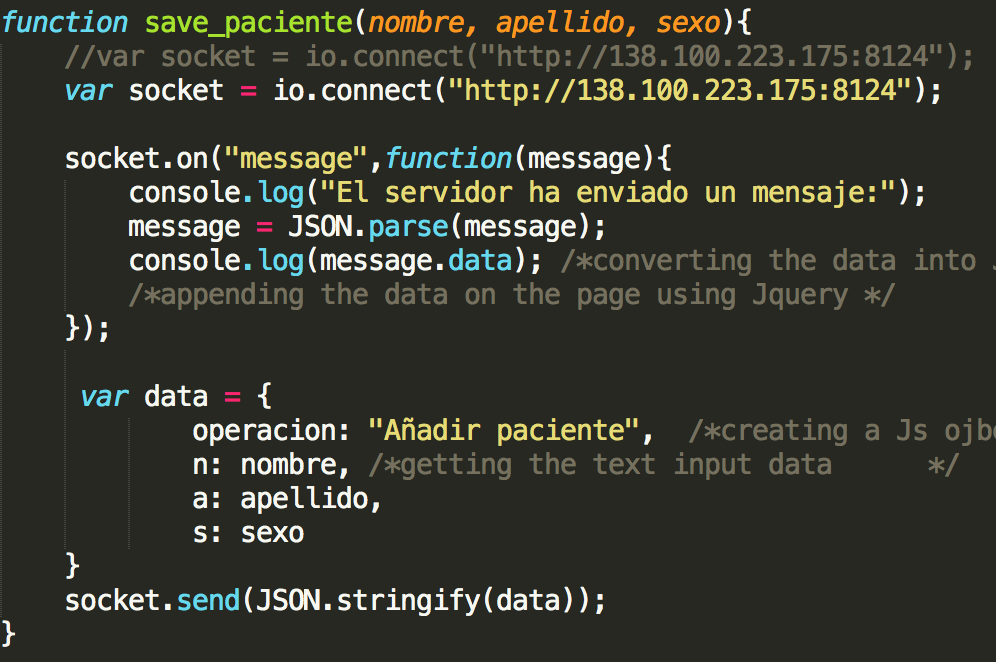
#### 4.3.3.1. Funcionalidad en el lado del cliente

Para poder añadir un paciente, el usuario tendrá que posicionarse en la pestaña de pacientes y rellenar el formulario que se encuentra justo después del listado de pacientes.





El formulario solicitará un nombre, apellidos y el sexo del paciente. Cuando se presione el botón añadir se llamará a una función “Validar()” que comprobará si hay algún campo vacío. En ese caso saltará un mensaje solicitando al usuario que rellene el campo que no ha rellenado. Cuando todos los campos estén adecuadamente rellenos y el usuario pulse “Añadir” se llamará a una función llamada “save\_paciente()” a la que se le pasarán los datos requeridos.



Aquí vemos un código muy similar al de los apartados anteriores, donde se establece un websocket con el servidor y se escucha un mensaje entrante de este. Después se envía al servidor un objeto JSON que contiene la cabecera “Añadir paciente” y los datos del paciente (nombre, apellidos y sexo). Como sabemos la cabecera es útil para que el servidor pueda distinguir las distintas operaciones que el navegador le pide.

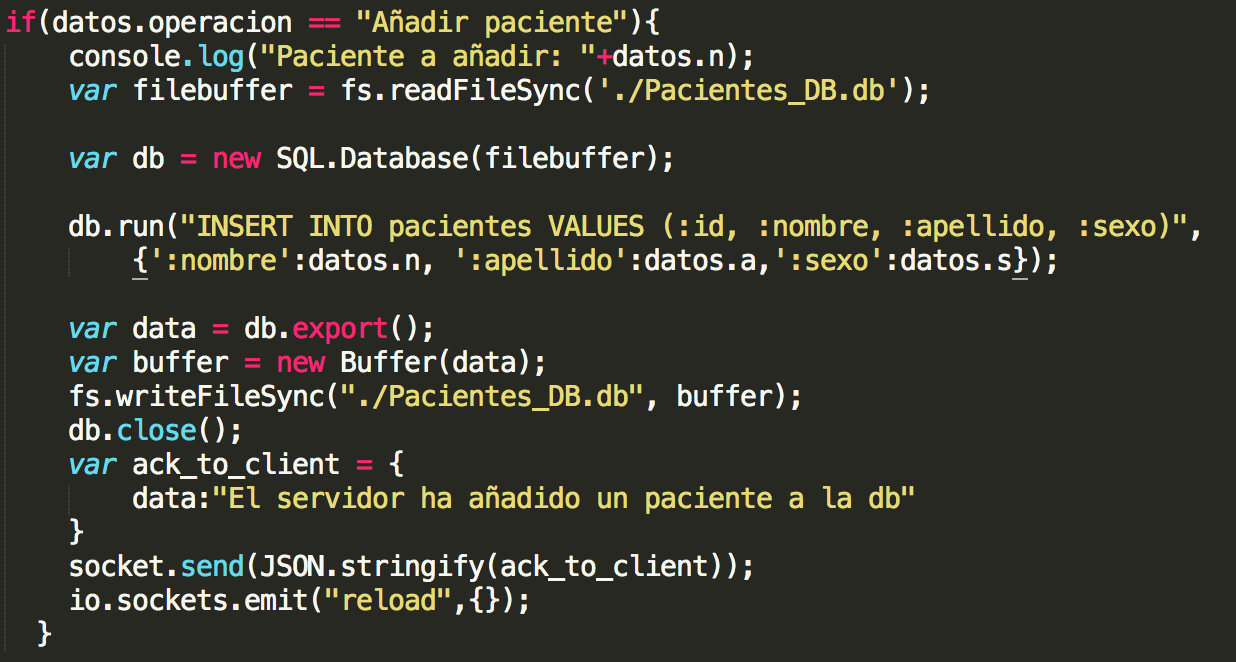
Para este caso es necesario que cuando añadamos al paciente se actualice la tabla de pacientes y así se mostrará la tabla de paciente con el nuevo paciente añadido. Para realizar esta función el servidor manda un mensaje que el cliente recibe y le sirve para saber cuando tiene que actualizar el contenido de la tabla.



Éste código se encuentra en el navegador y espera a que el servidor confirme que se han borrado los datos. Una vez recibida la confirmación, el navegador realiza un location.reload() que básicamente actualiza la página web, lo que obliga a que se vuelva a llamar a la función que lista los pacientes. En esta ocasión se mostrará la lista de pacientes actualizada.

#### 4.3.3.2. Funcionalidad en el lado del servidor





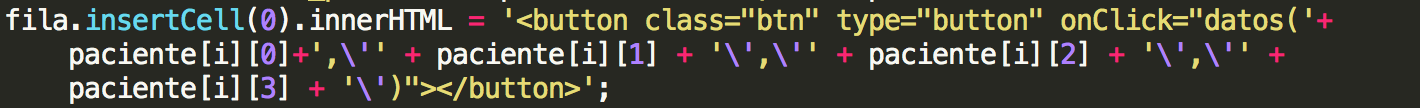
Como vimos para el caso para borrar un paciente el servidor escucha mediante socket.on(). Cuando recibe un mensaje comprueba su cabecera y al reconocer la operación “Añadir paciente” realiza la conexión con la base de datos y realiza un INSERT pasando los datos del paciente a añadir como parámetros. A continuación guarda los cambios y cierra la base de datos. Como hemos dicho en la parte del cliente, cuando se realizan los cambios el servidor ejecuta la función io.sockets.emit(). Básicamente fuerza a cualquier cliente conectado a actualizar su lista de clientes, por lo que si hay varios dispositivos conectados, en todos estos se actualizarán los cambios.

### 4.3.4. Obtener datos de movimiento de un paciente

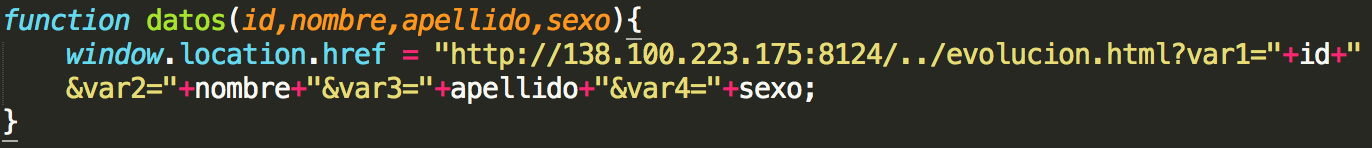
Para obtener estos datos seguiremos con la dinámica que se ha visto en las funciones anteriores (Conexión cliente-servidor/intercambio de datos entre cliente-servidor/apertura y cierre de la base de datos).

#### 4.3.4.1. Funcionalidad en el lado del cliente

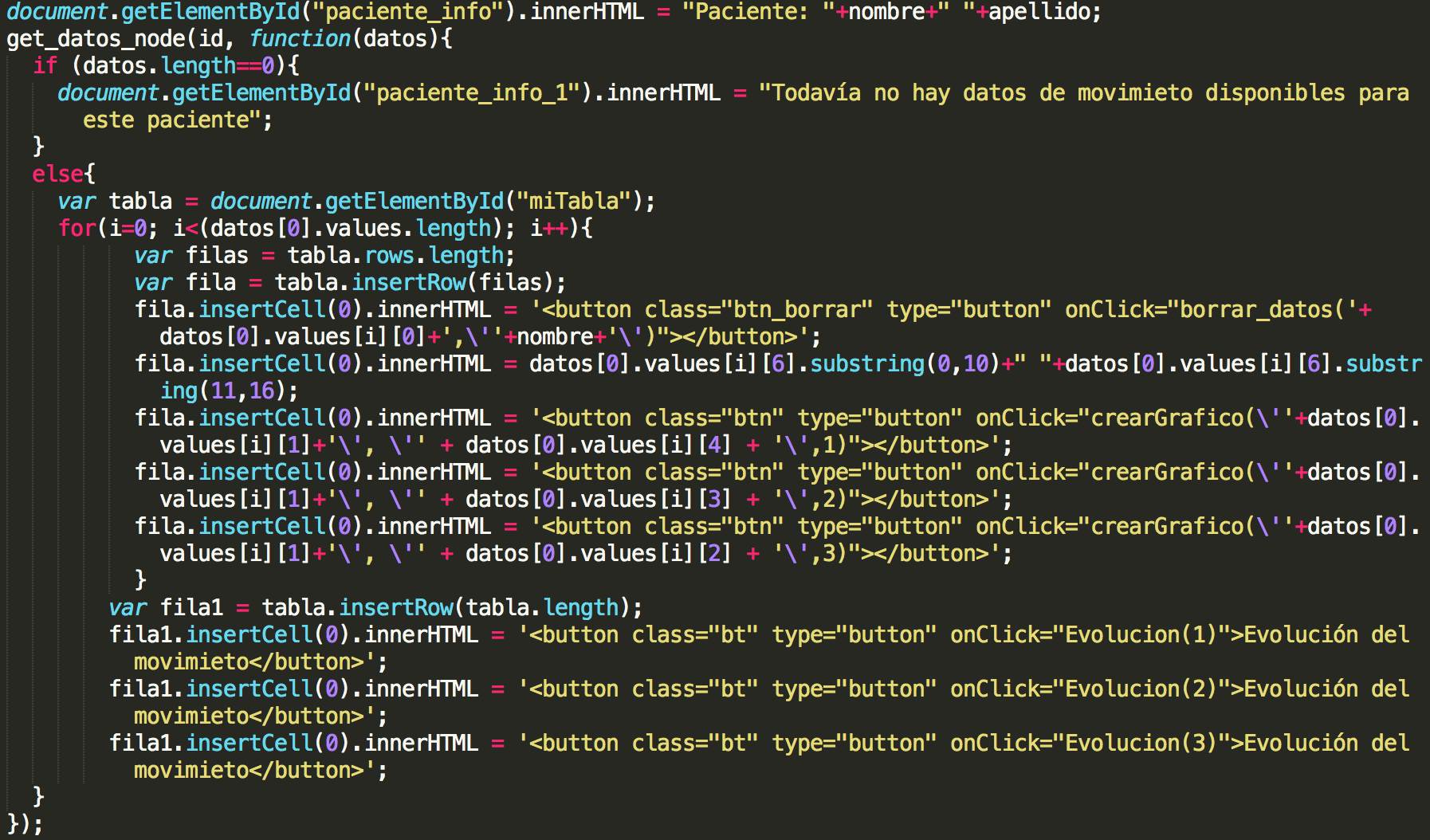
Para empezar, el usuario deberá posicionarse en la pestaña de pacientes y elegir el paciente del que quiere mostrar los datos de movimientos.



Cuando se genera la tabla de pacientes con el bucle for, una de las filas insertadas es un botón que permite el acceso a los datos de movimientos. Cuando el usuario presiona este botón se llama a una función “datos()” a la que se le pasan el id, nombre y apellidos del paciente.



La función datos posiciona al usuario en la pestaña de datos y pasa los datos del paciente a través de la URL. Así desde la pestaña de datos el navegador tendrá acceso a los datos del paciente del que tiene que mostrar los datos de movimientos.



Una vez posicionados en la pestaña de datos, se mostrará el nombre y los apellidos del paciente y se generará una tabla con distintos botones que al ser presionados permitirán al usuario mostrar los datos de movimientos para distintas fechas además de poder mostrar la evolución de estos datos con el tiempo. En el caso de que el paciente no tenga datos de movimientos se mostrará un mensaje comunicándolo.

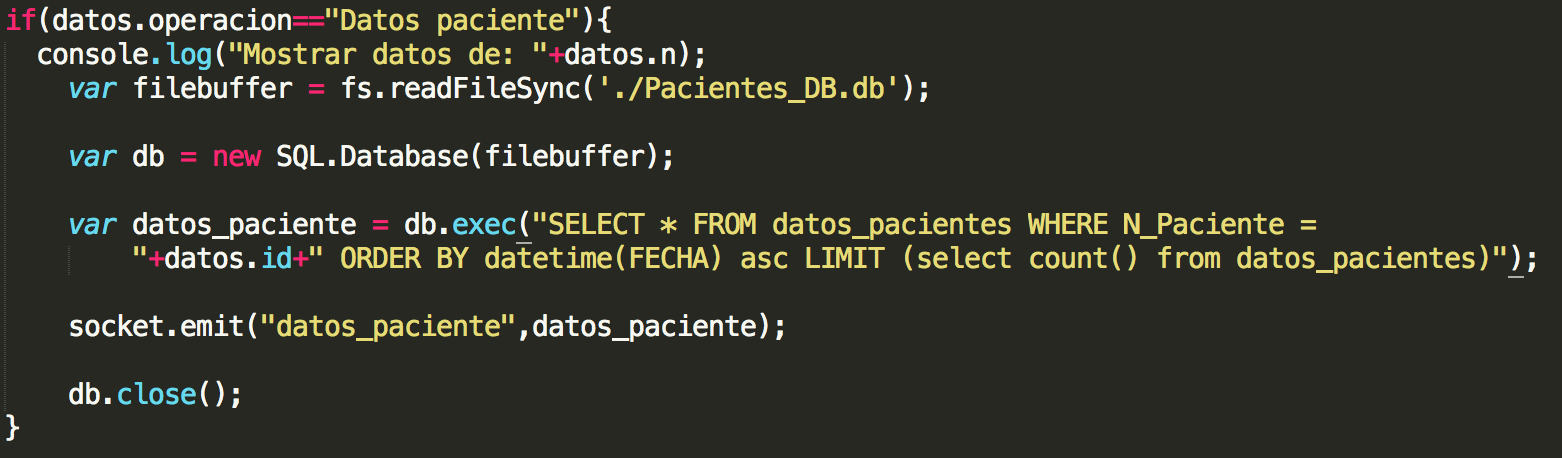
Los únicos datos que necesitaremos para la tabla de datos de movimientos serán las fechas para cada sesión de movimientos. Para obtener estos datos llamamos a la función “get\_datos\_node()” pasándole el id del paciente, que hemos obtenido del URL.



Observamos que se repite el proceso en el que se establece un websocket con el servidor, escuchando un mensaje de este y enviándole un objeto JSON con la cabecera “Datos paciente”. Se vuelve a escuchar al servidor mediante socket.on(), y se reciben los datos requeridos.

#### 4.3.4.2. Funcionalidad en el lado del servidor





Volvemos a observar el mismo comportamiento que en los aparados anteriores donde el servidor distingue la cabecera enviada por el cliente y opera en consecuencia. En este caso el comando SQL pide los datos de la base de datos ordenados por fecha ya que a la hora de mostrar los datos de movimiento es conveniente mostrarlos ordenados por fecha, pero sobretodo a la hora de mostrar la evolución de los movimientos es esencial que los datos se muestren en progresión con el tiempo, ya que no tendría sentido un gráfico de evolución temporal sin orden cronológico.

### 4.3.5. Añadir datos de movimiento

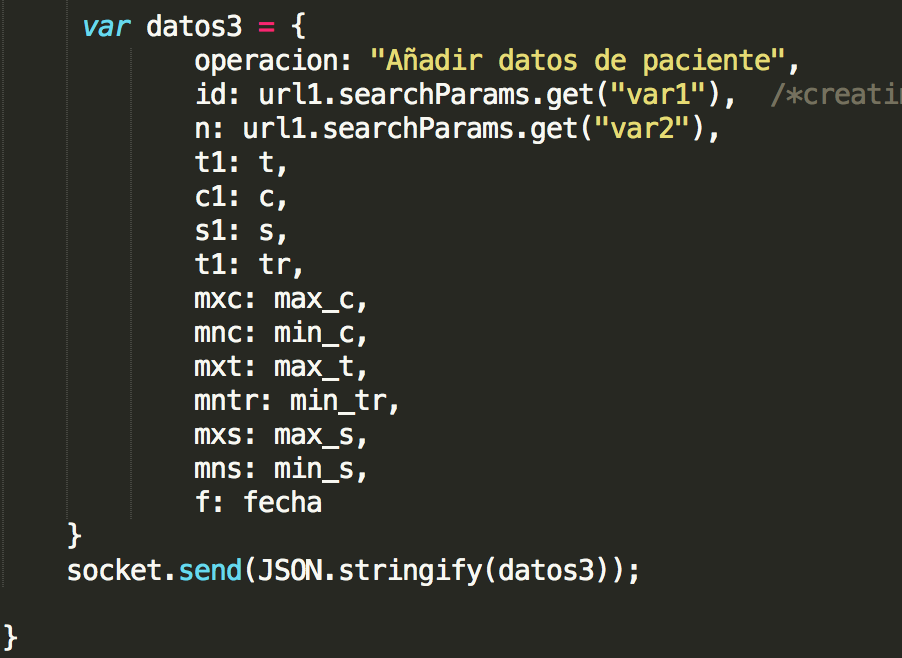
Para esta función tendremos una funcionalidad similar a la de añadir un paciente. Cambiará la forma de introducir los datos, que en vez de introducirlos en el navegador se seleccionará un archivo local.

#### 4.3.5.1. Funcionalidad en el lado del cliente



Como ocurría en el apartado de añadir pacientes tenemos una sección debajo de la tabla de datos de movimientos donde se requiere un archivo y una fecha de medición de la sesión de movimientos. A la hora de añadir un archivo se le pide al usuario que seleccione un archivo local. Este archivo solo podrá ser del formato CSV ya que este archivo es el que genera el aparato utilizado para medir los movimientos (Werium Basic Pro). Cuando el usuario ha seleccionado el archivo adecuado utilizamos la tecnología paparse para transformar el archivo CSV en un JSON ya que JavaScript no maneja archivos de texto delimitados mientras que si puede manejar objetos JSON. Cuando el usuario presione el botón para añadir los datos se comprobará que hay un archivo seleccionado y una fecha introducida. Si es correcto se llamará a la función “add\_datos()”, pasándole como parámetros el archivo CSV en formato JSON y la fecha.





En este caso hay que realizar varias operaciones para obtener la información que nos interesa del JSON. El JSON está compuesto por varios arrays. El bucle for se dedica a recorrer estos arrays para sacar los datos de tiempo y de los movimientos en los planos coronal, sagital y transversal. Cuando se extraen los datos para los movimientos se redondean los datos a dos decimales ya que el archivo CSV, generado por el aparato, genera números con 16 decimales, y guardar tantos números con tantos números decimales sería una carga muy pesada para la base de datos. Una vez extraídos estos datos sacamos los máximos y los mínimos de los tres movimientos, que nos servirán para mostrar la evolución de cada movimiento.

Una vez obtenidos todos los datos de interés crearemos un websocket para conectarnos al servidor, escucharemos el mensaje de confirmación de este y enviaremos los datos extraídos del JSON con la cabecera adecuada para que el servidor reconozca la acción que debe realizar “Añadir datos de paciente”. Podemos observar que el objeto JSON enviado al servidor tiene bastantes elementos que son los que hemos descrito anteriormente.

#### 4.3.5.2. Funcionalidad en el lado del servidor



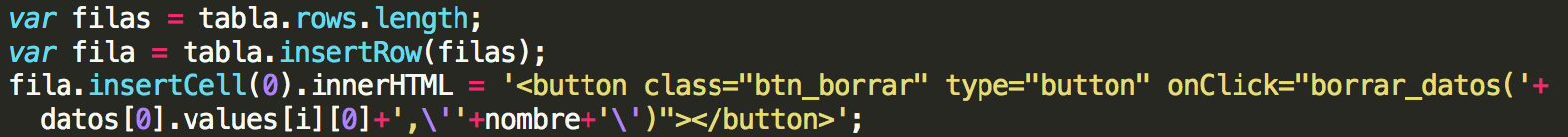


Aquí vemos el comportamiento que ha tenido el servidor para el resto de funciones. Se reconoce la cabecera contenida en el objeto JSON y se actúa en consecuencia. Después se vuelve a acceder a la base de datos y se insertan todos los datos que se han extraído del objeto JSON. Se guardan los datos mediante un db.export() y el uso de un buffer como ya se ha explicado anteriormente y se emite un mensaje para que todos los clientes conectados actualicen la tabla de movimientos.

### 4.3.6. Borrar un sesión de movimientos

Para borra un set de movimientos de un paciente seguiremos una dinámica similar a la de borrar un paciente.

#### 4.3.6.1. Funcionalidad en el lado del cliente



Como vimos en el aparatado de obtención de datos de un paciente, cuando se genera la tabla con movimientos se van insertando filas con los distintos movimientos, la fecha y por último un fila que incluye un botón para borrar esa sesión de movimientos. Como vemos en el código cuando el usuario pulsa el botón se llama a una función borrar\_datos() a la que se le pasan el identificador único de la sesión de movimientos y el nombre del paciente al que pertenece dicha sesión.



Como en todas las operaciones en las que se eliminan datos, se pregunta al usuario si quiere borrar los datos. Cuando el usuario confirma la operación se crea una conexión con el servidor y se sigue el procedimiento visto en funciones anteriores (escuchar mensaje del servidor y envío de datos). Como ya hemos visto se manda un objeto JSON con la cabecera “Borrar datos de paciente” para que el servidor proceda con la operación correspondiente.

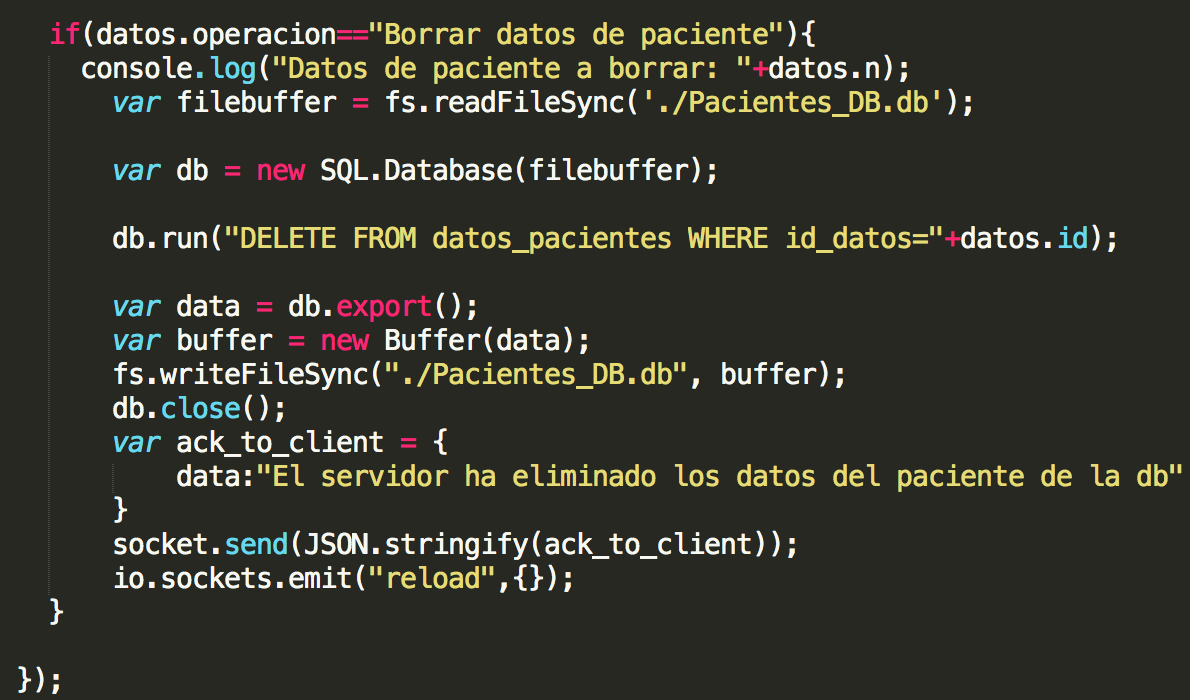
Para este caso es necesario que cuando borremos la sesión de datos se actualice la tabla con los movimientos y así no se muestre la sesión borrada. Para realizar esta función el servidor manda un mensaje que el cliente recibe y le sirve para saber cuando tiene que actualizar el contenido de la tabla.



Éste código se encuentra en el navegador y espera a que el servidor confirme que se han borrado los datos. Una vez recibida la confirmación, el navegador realiza un location.reload() que básicamente actualiza la página web, lo que obliga a que se vuelva a llamar a la función que obtiene las sesiones de movimientos. En esta ocasión se mostrará la lista de sesiones de movimientos actualizada.

#### 4.3.6.2. Funcionalidad en el lado del servidor



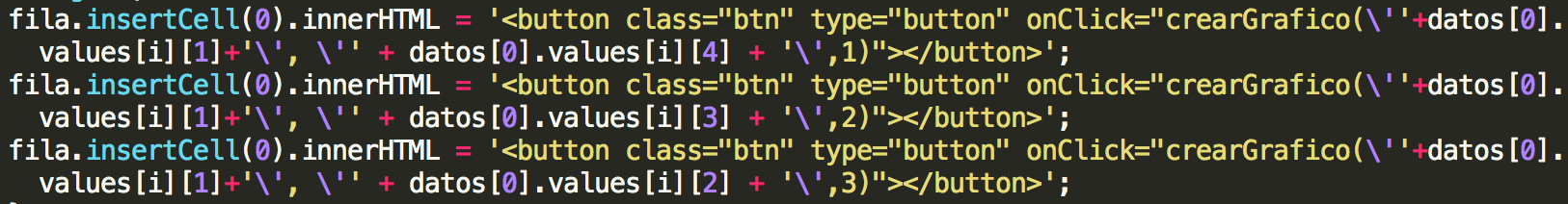


Como vimos para el caso para añadir una sesión de movimientos ,el servidor escucha mediante socket.on(). Cuando recibe un mensaje comprueba su cabecera y al reconocer la operación “Borrar datos de paciente” realiza la conexión con la base de datos y elimina la sesión de movimientos con el identificador que ha pasado el cliente. A continuación guarda los cambios y cierra la base de datos. Como hemos dicho en la parte del cliente, cuando se realizan los cambios el servidor ejecuta la función io.sockets.emit(). Básicamente fuerza a cualquier cliente conectado a actualizar su lista de sesiones de movimientos, por lo que si hay varios dispositivos conectados, en todos estos se actualizarán los cambios.

### 4.3.7 Mostrar un grafico de un movimiento

Para este caso no seguiremos el mismo procedimiento que para el resto de funciones, ya que no será necesaria una conexión con el servidor y además se hará uso de la herramienta chart.js para convertir los arrays de datos en gráficos.

#### 4.3.7.1. Funcionalidad en el lado del cliente



Este fragmento de código corresponde a la tabla de sesiones de movimientos. Tres de las filas insertadas para cada columna corresponden a un botón que permite crear un gráfico. Cada botón corresponde a un movimiento en un plano (Coronal, Sagital y Transversal). Dependiendo del botón que el usuario presione se mostrará uno de los tres movimientos. Los tres botones llaman a la misma función cuando son presionados “crearGrafico()”, y se le pasan los arrays de tiempo y del movimiento y un número que identifica el movimiento seleccionado.





Dentro de esta función reorganizamos los dos arrays para que la herramienta chart.js pueda utilizarlos. Con Chart.js creamos un nuevo tipo de gráfico que en este caso será lineal y le pondremos un título. También le pasaremos los dos arrays y ajustaremos las características del gráfico (color de la línea, grosor de la línea, interactividad con el gráfico, etc).

A continuación mostraremos un ejemplo de un gráfico generado por la aplicación web.



### 4.3.8 Mostrar un grafico de evolución de un movimiento

# 5. Pruebas

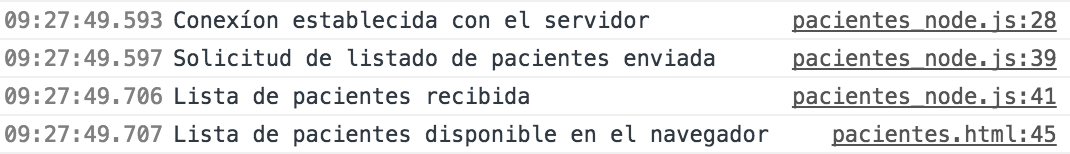
El objetivo del plan de pruebas es comprobar que el sistema funciona según lo especificado y responde a los requisitos de usuario. Para realizar pruebas en un entorno Java existen varias opciones como pueden ser la escritura de valores por pantalla, el uso de herramientas de log y la utilización de depuradores.

## 5.1. Pruebas de sistema

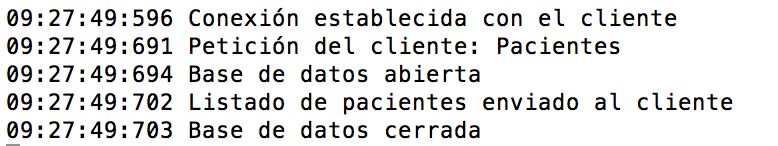
En este proyecto se utilizará la escritura de valores por pantalla, ya que es una estrategia simple y efectiva. Por lo tanto se observarán valores en la interfaz web y en el servidor que irán mostrando como cada función se va ejecutando. Para el caso del cliente/navegador se mostrarán valores en la consola del inspector de Google Chrome y para el caso del servidor se mostrarán valores en un terminal donde se ejecuta el código del servidor. Para poder observar que se realiza todo en el orden apropiado, se imprimirá un tiempo asociado a cada valor por pantalla.

### 5.1.2. Obtener paciente

Consola del navegador:



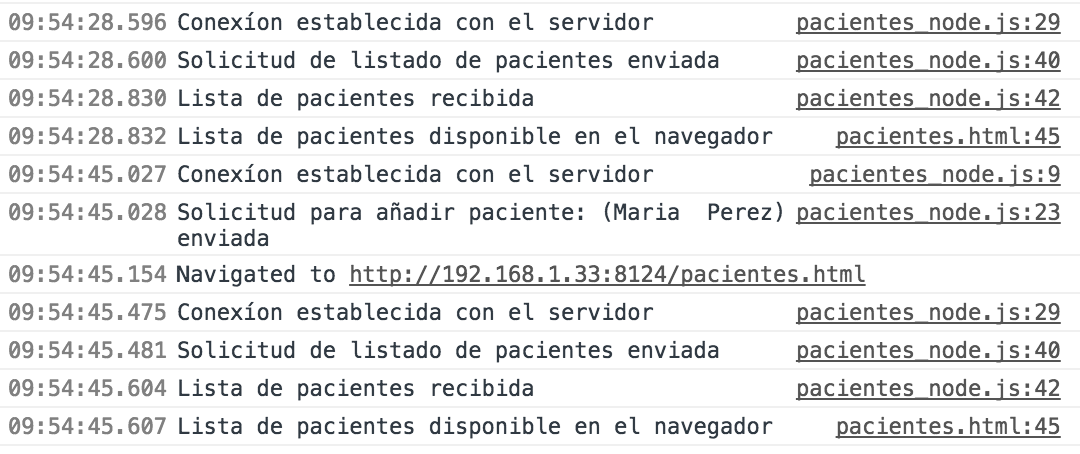
Terminal (Servidor):



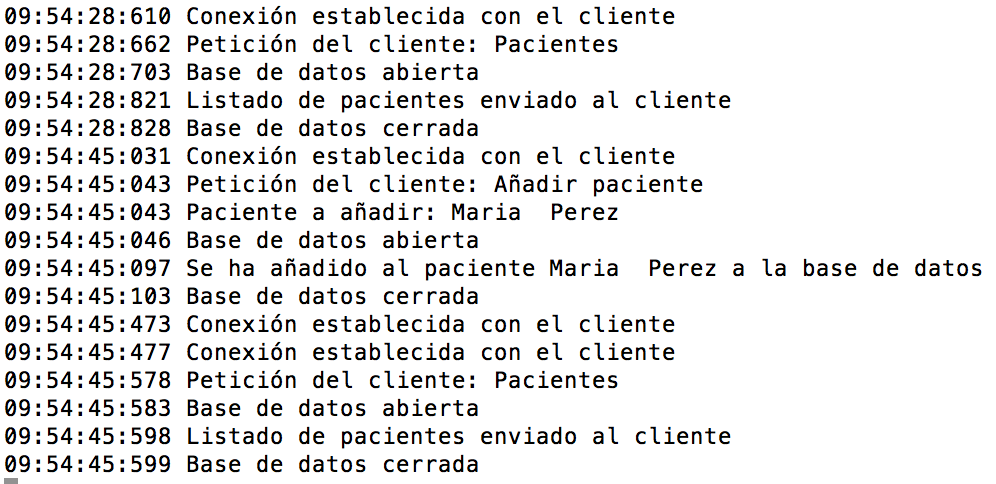
Como se puede observar todos los eventos ocurren en orden cronológico. Para empezar se establece la conexión mediante un websocket y después el cliente manda la solicitud para obtener el listado de pacientes. A continuación el servidor reconoce la petición, abre la base de datos, envía el listado al cliente y cierra la base de datos. Finalmente observamos como el cliente recibe la lista y la muestra en el navegador.

### 5.1.3. Añadir paciente

Consola del navegador:



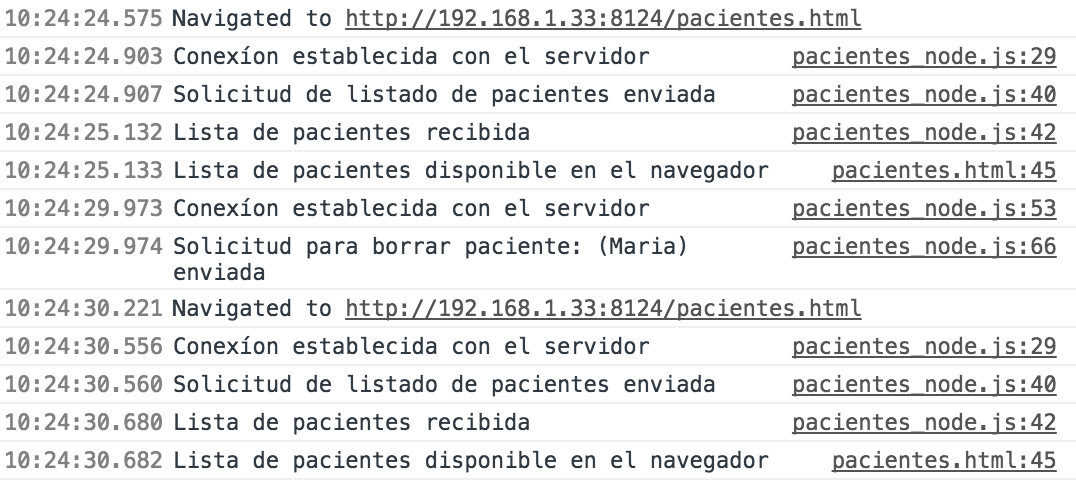
Terminal (Servidor):



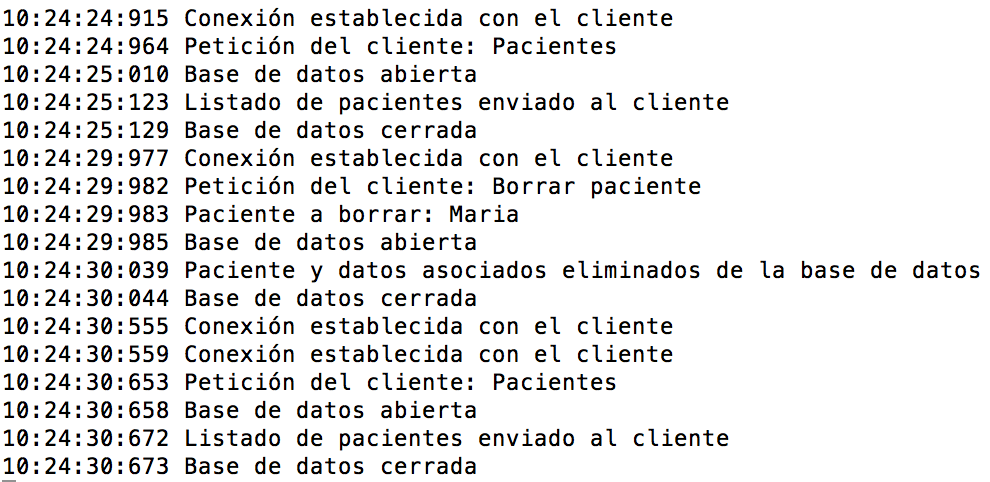
Cuando añadimos un paciente a la base de datos, primero se listan los pacientes, por lo que observaremos los mismos valores por pantalla que en el apartado anterior. Después se establecerá otra conexión con el servidor para añadir al paciente. Finalmente volveremos a observar, por segunda vez, los valores por pantalla para obtener el listado de pacientes, ya que como se explicó en la sección de implementación, cuando se añade un paciente, el servidor obliga al navegador a refrescarse, listando de nuevo los pacientes y por lo tanto mostrando el paciente añadido.

### 5.1.4. Borrar un paciente

Consola del navegador:



Terminal (Servidor):



Para este caso volvemos a observar el mismo comportamiento que se dio para añadir un paciente, ya que se obtiene la lista de pacientes en dos ocasiones (Listado inicial y listado de pacientes actualizado).

# Resultados y conclusiones

Puntos a poner:

* Enlace a Github. Tendrás que dejarlo bonito, poner licencia, etc. Por cierto, quita de tu repositorio de GitHub el TFG DE OTRO ALUMNO. No tienes permiso para subirlo a Internet.
* Resultados.
* Trabajos futuros.
* Qué has aprendido de la experiencia.

**Bibliografía**

<https://en.wikipedia.org/wiki/Socket.IO>

<https://es.wikipedia.org/wiki/JavaScript>

<https://es.wikipedia.org/wiki/Node.js>

<https://es.wikipedia.org/wiki/HTML>

<https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada>

<https://es.wikipedia.org/wiki/SQLite>

<https://es.wikipedia.org/wiki/Base_de_datos>

<https://es.wikipedia.org/wiki/ACID>

<https://github.com/kikinteractive/app>