UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN

MENCIÓN EN TELEMÁTICA

Procesado de señales procedentes de smartphones para el análisis de temblores fisiológicos

Autor:

Javier Pérez Ácimas

Tutores:

Rodrigo De Luis García

Juan Pablo Casaseca de la Higuera

Valladolid, 6 de junio de 2018

Resumen

El temblor fisiológico es un movimiento involuntario de tipo oscilatorio que se produce en algunas articulaciones del cuerpo, incluso en las personas sanas. Con el desarrollo de una aplicación móvil diseñada para la realización de capturas del temblor de un usuario podemos estudiar la naturaleza de los temblores de éste, con el fin de llegar a comprender las diferencias entre el temblor normal y aquel asociado a una patología concreta.

El sistema ideado con este propósito permitirá no solo realizar mediciones de temblores, sino que también se encargará de almacenarlas para cada usuario del sistema en un servidor remoto y ofrecer la posibilidad de visualizarlas en cualquier momento. De este modo, se obtiene una herramienta que, de forma sencilla y no intrusiva, nos sirve como ayuda tanto para estudiar las características de los temblores humanos como para el seguimiento de pacientes con enfermedades en las que estén involucrados los temblores.

Palabras clave: *temblor, dispositivo móvil, aplicación, servidor, medición*.

Abstract

Physiological tremor is an involuntary and oscillatory movement that occurs in many articulations of the body, even in healthy people. With the development of a mobile application designed to capture the tremor of a user, we can study the nature of his tremors in order to be able to learn the differences between normal tremor and pathology associated tremor.

The system designed for this purpose will not only make measurements of tremors, but will also be able to store them for each user of the system on a remote server and offer the possibility of viewing them at any time. Thereby, we obtain a tool that, in a simple and non-intrusive way, serves as an aid both to study the characteristics of human tremors and to monitor patients with diseases in which tremors are involved.

Keywords: *tremor, mobile device, application, server, measurement*.

Agradecimientos

En primer lugar, y por encima de todo, a todas aquellas personas que han colaborado en el desarrollo de este proyecto, ya sea probando el sistema o realizando medidas como voluntarios, sin todos vosotros nada de esto habría sido posible.

Al Laboratorio de Procesado de Imagen, en especial a mis tutores, por todos los medios proporcionados, por su ayuda y sus consejos a lo largo de este Trabajo.

A mi familia, por todo su apoyo durante estos años y por no dejar nunca de animarme.

A mis amigos, a todos, por su paciencia, por aguantarme, por estar siempre dispuestos a echar una mano, literalmente, y por saber cuándo uno tiene que dejar el trabajo a un lado y despejarse.

Índice general

Índice de figuras

1. Introducción

En el presente capítulo se presenta y contextualiza el tema a tratar, así como el problema del que viene originado y la justificación de la herramienta que se plantea diseñar, de modo que, gracias a ella, se facilite la extracción de datos y el análisis de los mismos. Se describen los objetivos generales y específicos de este trabajo, y se explican en profundidad los medios que serán empleados, además de la estructura seguida en el resto del documento.

1.1. Motivación del problema

El temblor es un movimiento de agitación oscilatorio, rítmico e involuntario, que puede darse en cualquier parte del cuerpo, pero que es especialmente notorio en brazos y manos [1] [2]. La causa de los temblores tiene su origen en las partes del cerebro que controlan el movimiento y, aunque son más comunes en gente de edad avanzada [3], pueden presentarse en cualquier persona.

Todo el mundo tiene algún leve temblor en las manos, sin bien sus características puede depender de factores externos como el consumo de cafeína, alcohol, medicamentos, tabaco, etc…; también puede derivarse del estrés, cansancio o enfado o incluso puede estar afectado por elementos hereditarios [4].

Aunque puedan aparecer temblores a cualquier persona sin resultar síntoma de ninguna enfermedad, cuando éste no desaparece con el tiempo, es una buena práctica consultar con un especialista [5][6]. De todos los tipos de temblor (de los que hablaremos más adelante), el temblor esencial [7], caracterizado por movimientos pequeños y rápidos, es el más común.

Hay numerosas enfermedades neurológicas que incluyen temblores entre de sus síntomas, de ellas probablemente la más característica sea la enfermedad de Parkinson [8], pero también aparecen en otras enfermedades neurodegenerativas como el Alzheimer

o la enfermedad de Huntington, además de en otras como la esclerosis múltiple o los accidentes cerebrovasculares [9].

Para el diagnóstico del temblor es necesaria una evaluación clínica detallada y realizar un seguimiento al paciente [10]. La abundancia actual de dispositivos móviles y la disponibilidad dentro los mismos de sensores mecánicos de cada vez mejor calidad, los hace ideales para recoger información acerca del temblor de cualquier persona en cualquier momento y lugar, simplemente fijándonos en los datos recogidos a través del acelerómetro y el giroscopio.

Para poder diferenciar el temblor fisiológico normal del temblor patológico (propio de una enfermedad) es necesario estudiar ambos, de modo que conozcamos las características que estos presentan. Con el fin de facilitar este análisis se propone diseñar una aplicación, como cualquier otra que podamos encontrar en un dispositivo móvil, que permita recoger y almacenar los datos necesarios de una forma rápida, sencilla y que no resulte invasiva para sus usuarios.

Con esta idea en mente, lo que se intenta alcanzar es la construcción de un prototipo capaz de reunir todas las capacidades básicas necesarias para realizar las mediciones desde un teléfono móvil y almacenarlas de forma fiable en un sistema externo, de modo que se pueda realizar un análisis de todos los datos de los que se haga acopio en busca las particularidades de los temblores capturados.

Comenzar este análisis por el estudio de temblores fisiológicos facilita un abanico más amplio de muestras que obtener, sin embargo, el sistema buscado debería ser igualmente aplicable al trabajo con cualquier otra clase de temblor, de igual modo, el sistema deberá proporcionar las funcionalidades mínimas de una aplicación de seguimiento médico, de modo que, con los progresos futuros oportunos, pudiera llegar a desplegarse y tener un uso práctico en este contexto.

1.2. Objetivos

El objetivo general de este trabajo es el diseño de un sistema que obtenga los datos recogidos de los temblores de las manos de una persona a través de una aplicación móvil y su envío a un servidor remoto, que se encargará de almacenar dichos datos, para su posterior procesado y análisis.

A partir de este objetivo general, podemos distinguir varios objetivos específicos:

- Construir la estructura necesaria para la acumulación de datos de numerosos individuos, de forma que se le pueda aplicar el estudio que se desee a los mismos.
- La creación de una aplicación útil para el seguimiento de pacientes diagnosticados de enfermedades con el temblor entre uno de sus síntomas, de modo que resulte sencilla de manejar para cualquier médico y que le permita evaluar remotamente el avance de la enfermedad.
- Análisis exploratorio de las señales obtenidas mediante el sistema construido sobre un pequeño conjunto de voluntarios sanos.

Así pues, se debe pasar por el desarrollo de una aplicación móvil que será la encargada de la captura de las señales, y por el desarrollo de un sistema web que se encargará de recoger y almacenar dichas señales. Con esto ya logrado, podremos pasar a implementar los procedimientos que se consideren oportunos para el análisis de las señales.

Una vez recogida una cantidad significativa de muestras por parte de personas voluntarias, podremos probar los análisis diseñados, tanto en el dominio del tiempo como de la frecuencia, y estudiar la validez de los mismos.

Con todo esto, podremos extraer valiosas conclusiones sobre la viabilidad del sistema, así como las características y requisitos a mayores que debería poseer un sistema de esta clase para ser totalmente funcional.

1.3. Fases

Con el fin de cumplir con el objetivo marcado se plantean varias fases que permitirán alcanzar el resultado final deseado.

1. Revisión de trabajos previos, búsqueda de bibliografía y adquisición de conocimientos relacionados con el tema de estudio.

2. Desarrollo del sistema.
    a. Inicio: descripción y análisis del producto, identificación de riesgos, captura de requisitos para el funcionamiento interno de la aplicación y el servidor.
    b. Elaboración: definición de los casos de uso del sistema, diseño de la interfaz gráfica.
    c. Construcción: diseño de las funcionalidades del sistema.
    d. Transición: pruebas unitarias de la aplicación móvil y del servidor, depuración de errores y adaptación del software al entorno final.

3. Diseño e implementación del modelo de datos necesario para almacenar los usuarios, datos e información necesaria para el correcto funcionamiento del sistema.

4. Integración de la aplicación móvil con el servidor.

5. Pruebas de integración de la aplicación y el servidor en conjunto.

6. Recogida de datos de acelerómetros y giroscopios de individuos de distintas edades, sexos y patologías.

7. Realización de análisis de las señales obtenidas a través de Matlab.

8. Estudio de los resultados y extracción de conclusiones.

1.4. Medios empleados

Para la realización del sistema en su conjunto se requiere del uso de varios lenguajes de programación y, por ende, de varios entornos de desarrollo especializados que nos simplificarán la tarea de desarrollo. Para la construcción de la aplicación móvil en Android tendremos que desarrollar en Java y XML y para el caso del servidor desarrollaremos en Python y MySQL.

- **Android Studio 3.1.2**. [11] Este es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android (tras reemplazar a Eclipse) y se basa en IntelliJ IDEA[1]. La licencia de este programa es gratuita y está disponible para múltiples plataformas: Microsoft Windows, Mac OS y GNU/Linux.
  Entre las múltiples características que nos ofrece esta plataforma podemos destacar:
  - Un emulador rápido, de numerosos modelos de dispositivos móviles, con varias funciones para poder ejecutar y probar aplicaciones rápidamente y su integración en distintos dispositivos.
  - Un sistema de compilación basado en Gradle[2] flexible.
  - Integración con GitHub y otros sistemas de gestión de versiones que facilitan el mantenimiento de la aplicación a largo plazo y el control de los cambios realizados en la misma.
  - Herramientas para detección de problemas de rendimiento, usabilidad, compatibilidad…
  - Soporte incorporado para Google Cloud Platform.
  - Instant Run para la inyección de cambios en la aplicación durante su ejecución.
  - Funciones de firma de aplicaciones.
  - Plantillas para creación de diseños típicos de Android.

- **PyCharm 2018.1.3 (Community Edition).** [14] Es un entorno de desarrollo integrado (IDE) para el desarrollo en lenguaje Python desarrollado por JetBrains y basado, al igual que Android Studio, en IntelliJ IDEA. La licencia de la versión comunitaria (Community Edition), aunque no posee todas las funcionalidades disponibles en la versión profesional, es gratuita, está disponible para múltiples plataformas (Microsoft Windows, Mac OS y GNU/Linux) y nos ofrece las prestaciones suficientes para el desarrollo de nuestro servidor, entre las cuales se encuentran:

[1] IntelliJ IDEA [12]: es un entorno de desarrollo integrado (IDE) para el desarrollo de programas informáticos en el cual están basados varios entornos de desarrollo para diversos lenguajes: Pycharm, PhpStorm, AppCode, RubyMine… Está desarrollado por JetBrains y posee dos ediciones: una de código abierto (Community) y otra comercial (Professional).
[2] Gradle [13]: es un sistema de automatización de construcción de código abierto que fue diseñado para dar apoyo a la construcción de proyectos grandes de una forma eficiente.

  - Asistente inteligente que ofrece inspección de código, reconocimiento de errores de sintaxis, capacidades avanzadas de navegación, reconstrucción de código automática…
  - Compatibilidad con herramientas (paquetes) científicas para procesamiento matemático.
  - Frameworks para desarrollo web.
  - Capacidades de desarrollo remoto.
  - Herramientas gráficas de construcción: unidades de testeo, debugger…
  - Integración con múltiples sistemas de control de versiones.

- **MySQL 5.7.19**. [15] Es el sistema de gestión de bases de datos relacional de código abierto más popular del mundo. Gracias al lenguaje SQL podemos almacenar, manipular y recuperar los datos almacenados en dichas tablas. Este sistema está desarrollado bajo licencia dual: pública y comercial por Oracle; es por ese motivo que trabajaremos indistintamente con **MariaDB 10.2.8**, un sistema derivado de MySQL con licencia de código abierto y las mismas órdenes, instrucciones, APIs y bibliotecas que éste.

- **MATLAB R2017a**. [16] Es un software de cálculo numérico que posee tanto un entorno de desarrollo integrado (IDE) ideal para análisis y procesado de señal, como un lenguaje de programación propio (lenguaje M) especialmente diseñado para tareas de este ámbito y está disponible para varias plataformas: Microsoft Windows, Mac OS, GNU/Linux… Entre las características que ofrece Matlab podemos encontrar:
  - Manipulación de matrices
  - Implementación de algoritmos matemáticos.
  - Múltiples modos de representación de datos.
  - Posibilidad de comunicación e integración con otros lenguajes como Python, lo cual permite la automatización de tareas de mayor complejidad en el servidor.
  - Numerosos toolboxes para la realización de tareas muy variadas: procesamiento en paralelo, cálculo estadístico, deep learning, machine learning, optimización de algoritmos, conexión con bases de datos, procesamiento de imágenes…

- **Servidor accesible en red**. Para albergar el software de nuestro servidor utilizaremos un ordenador de sobremesa con distribución Linux y procesador Intel® Core™ i7-4790 @ 3.60GHz quad-core.

- **Señales de muestra recogidas**. Trabajaremos con las señales recogidas de tres ejes espaciales (x, y, z), perpendiculares entre sí, para el acelerómetro y el giroscopio del dispositivo móvil que ejecuta la aplicación. Las muestras se obtendrán de personas voluntarias siguiendo el proceso que previamente se haya establecido para realizar el proceso de medida del temblor.

1.5. Estructura del documento

En posteriores capítulos, se detalla el proceso que se ha llevado a cabo para alcanzar los objetivos planteados, siguiendo las fases anteriormente detalladas y valiéndonos de los medios de los que disponíamos, así como el resultado finalmente logrado con él.

En el segundo capítulo se presentan los distintos tipos de temblores, clasificados en función de la situación en la que aparecen y de su origen, y que podríamos llegar a estudiar valiéndonos del sistema a desarrollar.

En el tercer capítulo se desarrollan en profundidad las fases del proceso de desarrollo de software que se han seguido para el desarrollo del sistema del que trata el proyecto. Este proceso incluye las fases de análisis, diseño, construcción, despliegue y pruebas; y toda la documentación necesaria para cada una de ellas.

En el cuarto capítulo se incluye el tutorial de uso de la aplicación móvil desarrollada, que comprende la explicación de todas las funcionalidades de la misma. Con este tutorial, un usuario desconocedor de la aplicación estaría en disposición de usarla sin mayores dificultades.

En el quinto capítulo se introduce el análisis de señal realizado sobre señales recogidas de distintos sujetos y correspondientes a temblores de naturaleza fisiológica, así como, los resultados obtenidos a partir de éstos.

En el sexto y último capítulo se recogen las conclusiones extraídas de la realización del proyecto en su conjunto, así como las posibles mejoras y avances que podrían implementarse sobre él.

Por último, se proveen dos apéndices en los que figura la documentación relativa al código que implementa, por una parte, la solución adoptada para la implementación de la aplicación móvil, y, por otra, para la implementación del servidor.

2. Clasificación del temblor

Podemos clasificar los temblores en tres grandes grupos [4] [17] en función de la situación en la que aparecen:

- **Temblor en reposo o estático**. Es el temblor que aparece cuando el músculo está relajado, el ejemplo más típico es cuando las manos están apoyadas sobre el regazo. Este tipo de temblor es el más frecuente en las personas que padecen enfermedad de Parkinson.

- **Temblor en acción**. Es el temblor que aparece cuando se realizan movimientos con un músculo voluntariamente. Dentro de esta clase hay varios tipos de temblores como el temblor intencional (cuando se realiza un movimiento hacia un objeto), el temblor isométrico (cuando se realiza la contracción de un músculo contra un objeto rígido) o el temblor específico de una tarea (que aparece al realizar una tarea muy concreta).

- **Temblor postural**. Este temblor, que también se podría incluir dentro de los temblores en acción, es el que aparece cuando una persona mantiene una postura concreta, sin realizar movimiento, oponiéndose a la fuerza de la gravedad.

2.1. Tipos de temblor

Además de la clasificación anterior, los distintos tipos de temblor se pueden diferenciar analizando su origen y apariencia [6] [9]. Podemos destacar dos grupos: por un lado, el temblor fisiológico, presente en las personas sanas; y por otro lado los distintos temblores patológicos o no fisiológicos, que pueden aparecer por muchas causas. Lo vemos a continuación con más detalle:

- **Temblor normal o fisiológico**. Es el temblor que está presente en todas las personas y es un fenómeno totalmente normal, resultado de las propiedades físicas

del cuerpo humano. Se caracteriza por ser de leve amplitud y normalmente no es apreciable a simple vista. Su frecuencia de oscilación típica va de 8 a 12 Hz, pero varía con la edad, situándose entre los 6 y 7 Hz para los niños y la gente de la tercera edad. Es principalmente un temblor de acción y postural, y su amplitud se acentúa con la ansiedad, el estrés emocional, el ejercicio y la fatiga, así como con la cafeína y algunos tipos de medicamentos.

- **Temblor esencial**. [7] [18] Este tipo de temblor es uno de los trastornos del movimiento más común, es un temblor leve y que se mantiene durante los años. Aunque normalmente aparece en ambos lados del cuerpo, es más evidente en la mano dominante. Es principalmente un temblor de acción con predominancia en brazos y manos (más del 90% de los casos), pero también es frecuente que afecte a la cabeza. Su frecuencia de oscilación típica se sitúa entre los 4 y 12 Hz y su amplitud, al contrario que dicha frecuencia, se incrementa con la edad. Su origen, en la mayor parte de los casos, viene determinado por un factor hereditario y en muchas ocasiones es diagnosticado erróneamente como temblor parkinsoniano, sin embargo, estos trastornos tienen respuestas muy diferentes a las terapias disponibles, de ahí la necesidad de realizar un análisis correcto a la mayor brevedad posible.

- **Temblor parkinsoniano**. Este temblor es un síntoma común (si bien no siempre está presente) en la enfermedad de Parkinson y es, de hecho, la manifestación inicial de la enfermedad en alrededor de la mitad de los casos. Generalmente, se trata de un temblor en reposo en una o ambas manos, aunque también es frecuente la aparición de temblor postural, y en su inicio suele aparecer únicamente en una extremidad o un solo lado del cuerpo. La frecuencia de oscilación de este tipo de temblor varía entre los 3 y 7 Hz y su amplitud se incrementa con el estrés y disminuye con la realización de movimientos voluntarios.

- **Temblor psicogénico**[3]. Este tipo de temblor, con componentes en reposo, en acción y posturales, es el movimiento anormal con origen psicológico más común. Su amplitud es muy variable y aparece repentinamente en cualquier parte del cuerpo. Normalmente desaparece cuando la persona está distraída, su aparición se debe a situaciones de depresión, estrés postraumático o trastornos psiquiátricos.

- **Temblor ortostático**. [19] Es un trastorno raro caracterizado por contracciones muy rápidas de los músculos y más frecuentemente en las piernas. Su frecuencia de oscilación es cercana a los 16 Hz, y es casi imperceptible a simple vista.

- **Temblor distónico**. Este tipo de temblor es el que afecta a las personas que padecen de distonía, un trastorno del movimiento que causa contracciones involuntarias en los músculos, en algunos casos dolorosas, debido a un problema en la parte del cerebro encargada de esa función. Suele estar asociado a una

[3] Psicogénico: que es de origen psicológico.

postura concreta, su frecuencia de oscilación es menor a los 7 Hz y su amplitud es variable e irregular.

- **Temblor cerebeloso**. Este temblor, fácil de ver a simple vista, aparece típicamente en brazos y piernas en casos de lesiones en el cerebelo o en algunas enfermedades como la esclerosis múltiple. Se trata de un temblor de acción, concretamente un temblor de intención, y su frecuencia de oscilación es típicamente menor a 5 Hz [20]. Su amplitud aumenta al acercarse al objetivo (final del movimiento) y su dirección es perpendicular a la del mismo.

3. Desarrollo del sistema

Una vez conocida la problemática a la que responde el proyecto a tratar, podemos comenzar a desarrollar el sistema software sobre el que trabajaremos y que constituirá la base a partir de la cual podremos elaborar posteriormente los análisis que deseemos, con independencia de la solución adoptada para la construcción del sistema.

Al separar el sistema de captura y almacenamiento del análisis de las señales tenemos la posibilidad de utilizar el que queramos, con posibilidad de cambiarlo en cualquier momento de forma sencilla. En el futuro, una vez que se escoja un cierto análisis que se adapte a la situación que deseemos, podremos integrarlo en el sistema propiamente dicho, sin embargo, este sería un paso posterior a la primera fase de experimentación.

El proceso que seguiremos para el desarrollo del sistema está enmarcado dentro de lo que se conoce como Ingeniería de Software y que comprende el análisis, diseño, construcción, pruebas y mantenimiento del sistema software con el que trataremos [21]. A través del avance por estas etapas del desarrollo tendremos que concluir diversas cuestiones: la solución que queremos elaborar, qué características posee, cómo se va a construir, si funciona como se esperaba…

Gracias al seguimiento de estas fases logramos un avance más continuado y mejor enfocado, facilitando las tareas posteriores y aumentando la calidad del software generado, además de reducir notablemente el tiempo de desarrollo.

3.1. Fase de análisis

En esta primera fase del desarrollo de nuestro software realizaremos una descripción del sistema, la captura de los requisitos de la aplicación, una primera enunciación de los principales casos de uso y también estudiaremos los posibles riesgos y la viabilidad del sistema.

3.1.1. Descripción del sistema

El sistema por diseñar constará de dos partes: por un lado, consistirá en una aplicación móvil en la que un usuario podrá registrarse y a través de la cual podrá realizar una medición del temblor de su mano; por otro lado, y una vez hecho esto, la medida será enviada a un servidor remoto que se encargará de almacenarla. Las señales obtenidas de las mediciones de todos los usuarios del sistema serán posteriormente analizadas del modo que consideremos oportuno fuera de éste y de forma independiente a él.

El sistema también deberá soportar el acceso a través de perfiles distintos, con algunas diferencias entre ellos:

- **Perfil paciente**: el usuario se introduce con sus datos personales en la aplicación, estos datos se asociarán con las medidas que realice, tendrá acceso a las funcionalidades básicas del sistema.

- **Perfil experimentador**: se trata de un usuario encargado de recoger medidas de otras personas con el fin de realizar estudios sobre la naturaleza de los temblores, es decir, aunque el usuario se introduce en la aplicación con sus datos personales las mediciones que éste realice se presupone que no le pertenecerán a él, por lo tanto, se deberán solicitar unos datos mínimos sobre ella al realizar las medidas para los posteriores análisis.

- **Perfil médico**: el usuario se introduce en la aplicación con sus datos personales, pero no puede realizar mediciones asociadas a sí mismo. Este perfil está ideado para la monitorización de los usuarios con perfil de paciente y podrá tanto ver sus mediciones previas como realizarles otras nuevas.

3.1.2. Requisitos de la aplicación

Los requisitos son las capacidades que necesita el usuario para resolver una dificultad u objetivo concreto y que, por tanto, debe cumplir el sistema [22] [23]. Tenemos varios tipos de requisitos en función del servicio que proporcionan:

- **Requisitos funcionales**: indican una característica o funcionalidad concreta que expresa una capacidad que debe poseer el sistema. Según su importancia e impacto en el sistema global podemos separarlos en requisitos principales y secundarios.

Requisitos funcionales principales del sistema

- El sistema permitirá loguearse en la aplicación a cualquier usuario registrado, a través de su DNI y contraseña e independientemente de su perfil.
- El sistema permitirá realizar una medición del temblor de la mano del usuario.
- El sistema mostrará al usuario, gráficamente, la señal capturada cuando ésta termine de ser grabada.
- El sistema permitirá consultar un historial con las medidas grabadas que posee el usuario logueado.
- El sistema permitirá, a los usuarios con perfil *experimentador*, realizar mediciones del temblor que se correspondan con personas no registradas en el mismo.
- El sistema permitirá, a los usuarios de perfil *médico*, consultar los datos y medidas de los usuarios de perfil *paciente* con los que esté asociado.

Requisitos funcionales secundarios del sistema

- El sistema permitirá registrase a un nuevo usuario con perfil *paciente* directamente a través de la aplicación móvil.
- El sistema permitirá a los usuarios consultar sus datos personales (con los que se ha registrado en él).
- El sistema mostrará gráficamente al usuario los datos que se están recopilando durante el transcurso de la grabación del temblor.
- El sistema ofrecerá la posibilidad de no guardar una medición si ha ocurrido algún problema durante la grabación de la misma.
- El sistema ofrecerá, a los usuarios de perfil *médico*, un modo de asociarse a un usuario de perfil *paciente*.
- La aplicación permitirá a los usuarios desconectarse del sistema cuando éstos lo deseen.

- **Requisitos no funcionales**: indican una característica del sistema o los servicios ofrecidos que supone una restricción del mismo.
  - El sistema no permitirá más de un usuario registrado con el mismo DNI ni con la misma dirección de correo electrónico.
  - El sistema solicitará al usuario su consentimiento para tratar con sus datos personales y la información obtenida a partir del análisis de las señales que éste grabe.
  - Las comunicaciones entre la aplicación móvil y el servidor se realizarán a través de una conexión segura y encriptada.
  - El sistema no debe compartir la información almacenada de unos usuarios con otros.
  - Las mediciones se realizarán siguiendo uno o varios protocolos que serán indicados previamente al usuario. Estos protocolos indicaran al

usuario la postura o las condiciones en que deberá realizarse la medida.
- El sistema no debe permitir el almacenamiento de medidas sin ninguna información adicional, necesaria para la realización de análisis, asociada a las mismas.
- El sistema debe almacenar para los usuarios registrados información sobre ellos: número del DNI, nombre, apellidos, fecha de nacimiento, sexo, dirección de correo electrónico, fecha de alta como usuario y contraseña en el sistema.
- Las contraseñas de los usuarios se almacenarán encriptadas en todo momento.
- Los usuarios no podrán eliminar las mediciones registradas en el sistema.
- El sistema realizará una medida de calibración antes de realizar una nueva medición de temblor. La necesidad de esta calibración viene dada por el uso que se va a dar a la aplicación, ya que, si ésta es instalada en múltiples y distintos dispositivos, es posible que necesitemos dichos datos para el análisis posterior.
- El sistema debe ser intuitivo y fácil de usar para cualquier usuario desde la primera toma de contacto.
- El sistema deberá mostrar los textos en un tamaño adecuado para ser fácilmente legibles.
- El sistema deberá avisar al usuario ante cualquier fallo imprevisto y solucionarlo para evitar posibles bloqueos.
- El sistema deberá ser fácil de instalar para cualquier usuario.

3.1.3. Casos de uso

Los casos de uso son una herramienta que permite capturar las funcionalidades del sistema [21], describiendo el comportamiento del mismo como solución a un requisito funcional.

En los casos de uso se presentan las interacciones que tendrán lugar entre los actores[4] y el sistema, sin describir el funcionamiento en detalle del mismo. En esta primera fase enunciaremos los casos de uso que, presumiblemente, tendrá el sistema y, más adelante, se explicará de forma más detallada cómo se producen las interacciones entre el sistema y sus actores.

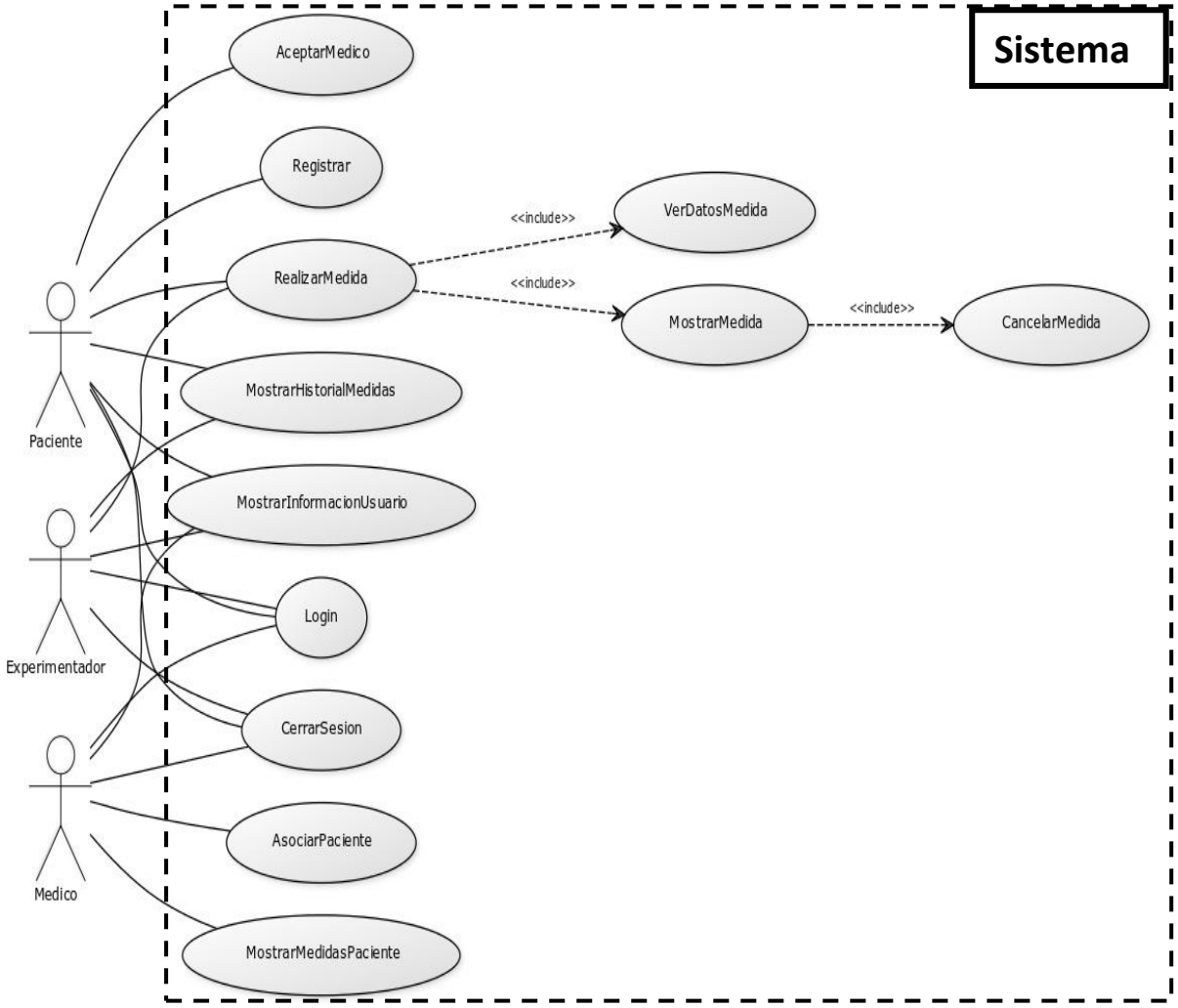
[4] Actor: entidad, entendiendo por tal tanto seres humanos como otros sistemas o eventos temporales, externa al sistema y que demanda una funcionalidad del mismo, desencadenando un caso de uso.

Caso de uso	Requisito	Actores implicados
Login	*El sistema permitirá loguearse en la aplicación a cualquier usuario registrado, a través de su DNI y contraseña.*	Paciente Experimentador Médico
RealizarMedida	*El sistema permitirá realizar una medición del temblor de la mano del usuario.*	Paciente Experimentador
MostrarMedida	*El sistema mostrará al usuario, gráficamente, la señal capturada cuando ésta termine de ser grabada.*	Paciente Experimentador
MostrarHistorialMedidas	*El sistema permitirá consultar un historial con las medidas grabadas que posee el usuario logueado.*	Paciente Experimentador
MostrarMedidasPaciente	*El sistema permitirá, a los usuarios de perfil médico, consultar los datos y medidas de los usuarios de perfil paciente con los que esté asociado.*	Médico
Registrar	*El sistema permitirá registrase a un nuevo usuario con perfil paciente directamente a través de la aplicación móvil.*	Paciente
MostrarInformacionUsuario	*El sistema permitirá a los usuarios consultar sus datos personales (con los que se ha registrado en él).*	Paciente Experimentador Médico
VerDatosMedida	*El sistema mostrará gráficamente al usuario los datos que se están recopilando durante el transcurso de la grabación del temblor.*	Paciente Experimentador
CancelarMedida	*El sistema ofrecerá la posibilidad de no guardar una medición si ha ocurrido algún*	Paciente Experimentador

	problema durante la grabación de la misma.	
AsociarPaciente	*El sistema ofrecerá, a los usuarios de perfil médico, un modo de asociarse a un usuario de perfil paciente.*	Médico
AceptarMedico		Paciente
CerrarSesion	*La aplicación permitirá a los usuarios desconectarse del sistema cuando éstos lo deseen.*	Paciente Experimentador Médico

3. 1. Enunciación de casos de uso.

Una vez enunciados los casos de uso y los actores que los desencadenan estamos en disposición de mostrar una visión general del sistema en la que plasmaremos las relaciones entre unos y otros.



3. 2. Gráfica de casos de uso.

3.1.4. Estudio de riesgos, viabilidad del sistema y elección de tecnologías

Una vez tenemos claro cuáles son las funciones básicas de las que tiene que disponer el sistema que vamos a diseñar y las situaciones ante las que se va a enfrentar, debemos analizar si el sistema es realmente factible y puede ser implementado, y en caso afirmativo qué tecnologías serán las que se utilicen para llevarlo a cabo, además de los problemas que pueden surgirnos durante las tareas de desarrollo.

Problemas

En primer lugar, debemos ser conscientes de que vamos a tratar con los datos personales de los usuarios de nuestro sistema, eso incluye tanto su nombre, apellidos, género, fecha de nacimiento… como los datos que recogeremos de la medición de sus temblores y los resultados de los análisis que se puedan obtener trabajando con ellos.

Es por ese motivo que se informará al usuario de la aplicación sobre las condiciones del uso de la misma cuando éste se registre en el sistema y se le solicitará que acepte dichas condiciones antes de que sus datos sean almacenados.

También queremos intentar cumplir con el proyecto utilizando un presupuesto lo más reducido posible, por lo tanto, debemos intentar adoptar soluciones lo más económicas que podamos ante las situaciones a las que nos enfrentemos.

El último de los problemas identificados consiste en la necesidad de disponer de un servidor que sea accesible, en cualquier momento y desde cualquier parte, para los dispositivos móviles que ejecutan nuestra aplicación.

La solución a este problema pasa por poseer una dirección (IP) pública en internet a través de la cual nuestra aplicación pueda comunicarse directamente con el servidor. Sin embargo, optaremos por una solución que supondrá un coste menor: utilizaremos un ordenador ubicado dentro de la red de la escuela para alojar nuestro programa servidor y nos valdremos de uno de los puertos visibles desde el exterior de la red para redirigir el tráfico desde éste hacia nuestra máquina.

Discusión de tecnologías

- **Aplicación móvil**. A la hora de realizar cualquier desarrollo móvil nos encontramos ante la disyuntiva de elegir entre cuatro tipos de aplicaciones especialmente orientadas a este uso [24].

- **Aplicaciones nativas**. Se trata de las aplicaciones desarrolladas en un lenguaje de programación clásico[5], con funciones de bajo nivel para cada entorno y que no necesitan de un navegador para ser ejecutadas. Están pensadas para ser usadas en dispositivos móviles y por ello permiten una mejor experiencia para el usuario, tanto en calidad de funciones gráficas, como en velocidad de ejecución y acceso a los recursos del dispositivo (sensores, GPS, cámara, memoria…).
- **Aplicaciones web**. Esta clase de aplicaciones se ejecutan habitualmente desde un navegador y se desarrollan en lenguajes de programación propios del desarrollo web: HTML, JavaScript, CSS… Se trata de la alternativa más sencilla de las cuatro y su presencia está en aumento actualmente. Entre las ventajas que presentan se puede destacar el acceso a las mismas desde cualquier dispositivo con un navegador sin necesidad de instalarlas, además de su sencillez de actualización.
- **Aplicaciones generadas**. Para el desarrollo de estas aplicaciones se utilizan lenguajes específicos de la herramienta utilizada, de entre las que podemos destacar Xamarin [25] o Genexus [26]. Una vez hecho esto se genera la aplicación en el lenguaje de la plataforma de destino y ésta se compila con las herramientas nativas. Aunque el resultado final pueda parecerse al de una aplicación nativa, ni el rendimiento obtenido es el mismo ni tenemos las mismas facilidades para el acceso a los recursos del dispositivo.
- **Aplicaciones híbridas**. Esta solución consiste en aplicaciones nativas que ejecutan determinados servicios a través de un WebView[6], de forma transparente al usuario (en gran parte de los casos). Al basarse en aplicaciones nativas, se desarrollan en lenguajes propios de cada entorno, y las páginas web accedidas a través del navegador se desarrollan en los lenguajes de programación web. Todo esto nos permite combinar una aplicación web con varias de las ventajas de las aplicaciones nativas.

Una vez conocidos los distintos tipos de aplicaciones debemos escoger cuál es la solución que mejor se adapta a nuestro sistema, para nuestro caso el factor determinante ha sido la necesidad de acceder al acelerómetro y al giroscopio del dispositivo móvil. Para este caso concreto, las **aplicaciones nativas** son las que mayores facilidades nos aportan, además presentan otras ventajas que nos hacen decantarnos por ellas, como son: un mejor rendimiento, almacenamiento local seguro, acceso a la aplicación incluso sin tener acceso a la red y acceso a todas las APIs del dispositivo.

Sin embargo, esta elección nos sitúa ante una nueva decisión, puesto que para que la aplicación pueda ser utilizada en la mayor parte de los dispositivos

[5] Entre los lenguajes de programación clásicos para el desarrollo móvil encontramos: las aplicaciones Java o Kotlin para Android, Objective-C o Swift para iOS y Visual Basic .Net o C# para el caso de Windows X Mobile.
[6] WebView: navegador web alojado en la aplicación.

supone desarrollarla para Android e iOS por separado, ya que el código para una plataforma solo funciona sobre esa plataforma, lo cual implica duplicar el trabajo.

Finalmente, se tomó la decisión de desarrollar la aplicación únicamente para el sistema operativo **Android**, en base a la experiencia previa en este entorno y a que, actualmente, su presencia en el mercado a nivel mundial, y en España en particular, es bastante mayor que la de su competidor.

- **Servidor**. Llegado el momento de escoger la tecnología que se utilizará para el desarrollo del servidor nos encontramos con un abanico de opciones muy extenso de entre los cuales tan solo nos centraremos en dos: PHP y Python; aunque se podría discutir sobre otros muchos: ASP.NET[7], Node.js[8], Perl, Ruby…

    - **PHP** (PHP Hypertext Preprocesor). Es, con diferencia, el lenguaje de programación más utilizado actualmente en el lado del servidor por los sitios web [28], aunque actualmente su uso está en ligero descenso. Este lenguaje interpretado de código abierto es ejecutado en el servidor y permite la creación de documentos HTML dinámicos y el acceso a múltiples sistemas gestores de bases de datos. PHP se encuentra disponible para muchas plataformas (Windows, Linux…) y posee una sintaxis muy similar a otros lenguajes como C, JavaScript, Perl…
    - **Python**. Es un lenguaje de programación interpretado y multiplataforma, tiene una sintaxis simple, sus programas son claros y fáciles de leer y está disponible gratuitamente [29]. El uso de Python ha crecido notablemente en los últimos años gracias a su aplicación en el campo del procesamiento de datos y la inteligencia artificial, pero puede ser igualmente utilizado en otras áreas gracias a su librería estándar, que le permite ejecutar tareas y funciones complejas con gran facilidad.

  Ante el dilema de elegir entre PHP y Python para desarrollar nuestro servidor la elección que podría parecer más atractiva a primera vista es PHP debido a su antigüedad, facilidad de instalación y gran cantidad de documentación, sin embargo, la tecnología finalmente elegida fue **Python**. Esta decisión se basó en las posibilidades que el lenguaje ofrece para la integración futura del análisis de las señales recogidas por la aplicación móvil, un análisis que gracias Python podría realizarse íntegramente en el servidor sin la necesidad de trasladarlo a Matlab, además, al no tratarse de un sitio web, PHP podría llegar a limitarnos mientras que Python es mucho más rápido y flexible.

[7] ASP.NET: entorno para el desarrollo de aplicaciones web de Microsoft sucesor de ASP (Active Server Pages)
[8] Node.js: plataforma para el desarrollo de aplicaciones JavaScript en el lado del servidor ejecutable sobre Windows, Linux o Mac OS.

3.2. Fase de elaboración o diseño

Durante esta fase nos encargaremos de desarrollar el funcionamiento de los casos de uso identificados durante el análisis, además de una caracterización preliminar de la interfaz de usuario que necesitaremos para responder a los mismos y de realizar el diseño de la base de datos que crearemos para el almacenamiento de la información utilizada por el sistema.

Al finalizar esta fase ya tendremos una idea bastante clara de en qué consistirá el proyecto al que nos vamos a enfrentar y nos encontraremos en disposición de comenzar con la construcción del mismo.

3.2.1. Desarrollo de los casos de uso

A continuación, se desarrollarán en profundidad los casos de uso que satisfacen los requisitos de nuestro sistema explicando de forma detallada la interacción que tiene lugar entre los actores y el sistema, tanto durante el flujo normal del programa como en los flujos alternativos que puedan surgir [30]. También se introducen algunos detalles adicionales para cada caso de uso como son: las condiciones previas que deben cumplirse para su desarrollo, el resultado que se obtendrá de su realización, la importancia que tiene para el sistema en su conjunto, el número de ejecuciones esperadas…

Identificador	**Login**	
Descripción	Un usuario accede a las funcionalidades de la aplicación móvil introduciendo su DNI y contraseña.	
Actores	Paciente, Experimentador, Médico	
Precondición	El usuario está registrado, con uno de los perfiles existentes, en el sistema.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1	El usuario lanza la aplicación en su dispositivo móvil.
	2	La aplicación solicita al usuario su DNI y contraseña.
	3	El usuario introduce su DNI y contraseña.

	4	El sistema redirige al usuario a la pantalla principal de la aplicación.
Postcondición	El usuario está conectado al sistema y tiene acceso a todas las funcionales que le ofrece su perfil.	
Flujo alternativo	**Paso**	**Acción**
	4.1	Si la contraseña es incorrecta se informa al usuario y se le vuelven a solicitar su DNI y contraseña.
	4.2	Si el usuario no dispone de conexión con la red se informa al usuario y se aborta la transmisión de datos
Frecuencia esperada	Muy alta.	
Importancia	Requisito principal	

3. 3. Desarrollo del caso de uso Login.

Identificador	**RealizarMedida**	
Descripción	Un usuario realiza una medición del temblor de su mano	
Actores	Paciente, Experimentador.	
Precondición	El usuario está logueado en el sistema con un perfil paciente o experimentador. El dispositivo móvil del usuario posee un acelerómetro y un giroscopio integrados.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1	El usuario selecciona la opción “Realizar Nueva Medida”.
	2	La aplicación solicita realizar una medida de calibración.
	3	El usuario acepta la realización de la medida de calibración.

	4	El sistema guarda la medida de calibración y solicita al usuario que escoja un protocolo para realizar la medición.
	5	El usuario escoge un protocolo para realizar la medición.
	6	El sistema guarda la elección del usuario y le muestra información sobre la postura que debe adoptar.
	7	<<include[9]>> VerDatosMedida
	8	<<include>> MostrarMedida
	9	El usuario acepta el envío de la medida.
	10.a	Si el usuario tiene perfil paciente: la aplicación envía la medida al servidor.
	10.b.1	Si el usuario tiene perfil experimentador: la aplicación solicita al usuario la fecha de nacimiento y el sexo de la persona que ha realizado la medida.
	10.b.2	El usuario completa la información solicitada por la aplicación.
	10.b.3	La aplicación envía la medida al servidor
Postcondición	La medida realizada queda guardada en el sistema asociada el usuario que ha realizado la misma y sus datos.	
Flujo alternativo	**Paso**	**Acción**
	4.1	Si durante la calibración se observan picos inesperados se informa al usuario de que la calibración ha fallado y se vuelve al paso 2.
	1-12	Si el usuario pulsa el botón Atrás, se retorna a la pantalla principal de la aplicación y el caso de uso finaliza sin ninguna postcondición.

[9] Include: hace referencia a un caso de uso distinto (desarrollado más adelante), que será utilizado dentro de otro.

	10.a.1, 10.b.3.1	Si el usuario no dispone de conexión con la red se informa al usuario y se aborta la transmisión de datos
Frecuencia esperada	Muy alta.	
Importancia	Requisito principal	

3. 4. Desarrollo del caso de uso RealizarMedida.

Identificador	**MostrarMedida**	
Descripción	Se muestra al usuario, gráficamente, la señal capturada cuando ésta termine de ser grabada.	
Actores	Paciente, Experimentador	
Precondición	El usuario ha realizado una calibración correcta de los sensores de la aplicación.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1	El usuario graba una medición del temblor de su mano.
	2	La aplicación captura los valores numéricos obtenidos durante la medición y los pinta en una gráfica contra el tiempo que ha durado dicha medición.
	3a	El usuario da por válida la medición.
	3b	<<include>> CancelarMedida
Postcondición	Ninguna.	
Flujo alternativo	**Paso**	**Acción**
	1-3	Si el usuario pulsa el botón Atrás, se retorna a la pantalla principal de la aplicación.
Frecuencia esperada	Muy alta.	
Importancia	Requisito principal	

3. 5. Desarrollo del caso de uso MostrarMedida.

Identificador	**MostrarHistorialMedidas**	
Descripción	El usuario accede a los datos de todas sus mediciones guardadas en el sistema.	
Actores	Paciente, Experimentador	
Precondición	El usuario está logueado en el sistema con un perfil paciente o experimentador. El usuario tiene alguna medida grabada en el sistema.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1	El usuario selecciona la opción "Ver Historial de Medidas"
	2	El sistema recupera la fecha y hora de todas las mediciones que tiene guardadas un usuario y lo muestra.
	3	El usuario selecciona una de las medidas.
	4	La aplicación muestra, gráficamente, la medida seleccionada por el usuario.
Postcondición	Ninguna.	
Flujo alternativo	**Paso**	**Acción**
	2.1	Si el usuario selecciona la opción "Atrás", retorna al menú principal.
	2.2.1	El usuario selecciona la opción de visualización en modo calendario.
	2.2.2	El sistema muestra un calendario.
	2.2.3	El usuario selecciona un día del calendario.
	2.2.4	La aplicación muestra, debajo del calendario, un historial con las medidas realizadas ese día.
	2.3, 4.1	Si el usuario no dispone de conexión con la red se informa al usuario y se aborta la transmisión de datos
	4.2	Si el usuario selecciona la opción "Atrás", retorna al historial de medidas.

Frecuencia esperada	Media.
Importancia	Requisito principal

3. 6. Desarrollo del caso de uso MostrarHistorialMedidas.

Identificador	**MostrarMedidasPaciente**	
Descripción	Un usuario con perfil médico consulta los datos y medidas de un usuario perfil paciente con el que esté asociado.	
Actores	Médico	
Precondición	El usuario está registrado con un perfil de médico. El médico tiene asocial algún usuario con perfil paciente.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1	El usuario selecciona la opción “Ver Lista de Pacientes”
	2	El sistema muestra una lista con todos los pacientes asociados a ese usuario.
	3	El usuario selecciona uno de los pacientes.
	4	El sistema muestra al usuario la pantalla principal del paciente seleccionado.
Postcondición	El usuario tiene acceso a las opciones del menú del paciente elegido.	
Flujo alternativo	**Paso**	**Acción**
	1-4	Si el usuario selecciona la opción “Atrás”, retorna a la pantalla anterior.
	2.1, 4.1	Si el usuario no dispone de conexión con la red se informa al usuario y se aborta la transmisión de datos
Frecuencia esperada	Media.	
Importancia	Requisito principal	

3. 7. Desarrollo del caso de uso MostrarMedidasPaciente.

Identificador	**Registrar**	
Descripción	Un usuario se registra en el sistema con perfil de paciente.	
Actores	Paciente.	
Precondición	Ninguna.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1	El usuario selecciona la opción "Registrarse" de la pantalla de inicio.
	2	La aplicación solicita al usuario: DNI, nombre, apellidos, fecha de nacimiento, contraseña, sexo y dirección de correo electrónico; y le muestra las condiciones y los términos y uso de la aplicación.
	3	El usuario rellena los datos solicitados por la aplicación y envía los datos.
	4	El sistema redirige al usuario a la pantalla principal de la aplicación
Postcondición	El usuario y sus datos personales quedan registrados en el sistema. El usuario está conectado al sistema y tiene acceso a todas las funcionales del perfil de paciente.	
Flujo alternativo	**Paso**	**Acción**
	4.1	Si algún dato queda sin rellenar o se rellena de forma inadecuada se informa al usuario y se vuelve al paso 2.
	4.2	Si el usuario no dispone de conexión con la red se informa al usuario y se aborta la transmisión de datos
Frecuencia esperada	Media.	
Importancia	Requisito secundario.	

3. 8. Desarrollo del caso de uso Registrar.

Identificador	**MostrarInformacionUsuario**	
Descripción	Un usuario visualiza sus datos personales.	
Actores	Paciente, Experimentador, Médico.	
Precondición	El usuario está logueado en el sistema.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1	El usuario selecciona la opción “Información de usuario” de la pantalla principal.
	2	El sistema muestra al usuario: su DNI, nombre, apellidos, fecha de nacimiento, sexo y dirección de correo electrónico.
Postcondición	Ninguna.	
Flujo alternativo	**Paso**	**Acción**
	2.1	Si el usuario selecciona la opción “Atrás”, retorna a la pantalla anterior.
	2.2	Si el usuario no dispone de conexión con la red se informa al usuario y se aborta la transmisión de datos
Frecuencia esperada	Baja.	
Importancia	Requisito secundario.	

3. 9. Desarrollo del caso de uso MostrarInformacionUsuario.

Identificador	**VerDatosMedida**
Descripción	Se mostrará al usuario, gráficamente, los datos que se están recopilando durante el transcurso de la grabación del temblor.
Actores	Paciente, Experimentador.
Precondición	El usuario está logueado en el sistema con un perfil paciente o experimentador. El dispositivo móvil del usuario posee un acelerómetro y un giroscopio integrados.

	El usuario ha seleccionado un protocolo para realizar la medición.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1	El usuario selecciona la opción “Comenzar Medición”.
	2	El sistema recopila los datos de los tres ejes del acelerómetro y giroscopio del dispositivo móvil y pinta en una gráfica, en tiempo real, los valores que se capturan.
Postcondición	Los valores obtenidos quedan guardados de forma local hasta decidir si serán o no guardados en el servidor.	
Flujo alternativo	**Paso**	**Acción**
	1-2	Si el usuario selecciona la opción “Atrás”, se retorna al menú principal.
	1-2	Si la pantalla se bloquea, se aborta la grabación y se retorna al paso anterior.
Frecuencia esperada	Alta.	
Importancia	Requisito secundario.	

3. 10. Desarrollo del caso de uso VerDatosMedida.

Identificador	**CancelarMedida**	
Descripción	Un usuario decide no guardar en el servidor una medida previamente grabada.	
Actores	Paciente, Experimentador.	
Precondición	El usuario está logueado en el sistema con un perfil paciente o experimentador. El usuario ha grabado una medida.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1.a.1	El usuario observa alguna anomalía en la medida grabada y pulsa la opción “Cancelar”

	1.a.2	El sistema redirige al usuario a la pantalla principal.
	1.b.1	El usuario observa alguna anomalía en la medida grabada y pulsa la opción “Repetir”
	1.b.2	El sistema redirige al usuario a la pantalla de selección de protocolo de medida.
Postcondición	La medida que había sido grabada no se guarda en el servidor.	
Flujo alternativo	**Paso**	**Acción**
	1.1	Si el usuario selecciona la opción “Atrás”, retorna a la pantalla principal.
Frecuencia esperada	Baja.	
Importancia	Requisito secundario.	

3. 11. Desarrollo del caso de uso CancelarMedida.

Identificador	**AsociarPaciente**	
Descripción	Un médico solicita a un paciente que se asocie con él.	
Actores	Médico.	
Precondición	El usuario está logueado en el sistema con un perfil médico. Hay algún usuario con perfil paciente registrado en el sistema.	
Escenario principal de éxito (flujo normal)	**Paso**	**Acción**
	1	El usuario selecciona la opción “Asociar Paciente” del menú principal.
	2	El sistema solicita al usuario el DNI del paciente con el que quiere asociarse.
	3	El usuario introduce el DNI del paciente con el que quiere asociarse.
	4	El sistema envía la solicitud al paciente elegido.

Postcondición	El paciente seleccionado por el médico recibe una solicitud de asociación.	
Flujo alternativo	**Paso**	**Acción**
	3.1.1	El usuario elige la opción del menú de asociación mediante correo electrónico.
	3.1.2	El sistema solicita al usuario el correo electrónico del paciente con el que quiere asociarse
	3.1.3	El usuario introduce el correo electrónico del paciente con el que quiere asociarse.
	4.1	Si el paciente buscado por el médico no existe en el sistema, se informa al usuario y se vuelven a solicitar los datos.
	4.2	Si el paciente seleccionado ya tiene una petición pendiente por parte de otro médico, se informa al usuario y no se envía la solicitud.
	4.3	Si el usuario no dispone de conexión con la red se informa al usuario y se aborta la transmisión de datos
	1-3	Si el usuario selecciona la opción “Atrás”, se vuelve a la pantalla principal sin ninguna postcondición.
Frecuencia esperada	Baja.	
Importancia	Requisito secundario.	

3. 12. Desarrollo del caso de uso AsociarPaciente.

Identificador	**AceptarMedico**	
Descripción	Un paciente acepta la solicitud de asociación enviada por un médico.	
Actores	Paciente.	
Precondición	Algún usuario con perfil médico ha enviado al paciente una solicitud de asociación.	
	Paso	**Acción**

Escenario principal de éxito (flujo normal)	1	El usuario se loguea en la aplicación.
	2	El sistema muestra una notificación al usuario informando del nombre del médico que quiere asociarse con él.
	3	El usuario acepta la solicitud del médico.
	4	La asociación queda registrada en el sistema.
Postcondición	El paciente y el médico involucrados en la acción quedan ligados hasta que el paciente se asocie con otro médico.	
Flujo alternativo	**Paso**	**Acción**
	3.1	Si el usuario rechaza la solicitud del médico, se elimina la notificación y el caso de uso finaliza sin postcondiciones.
	3.2.1	El usuario no contesta a la solicitud.
	3.2.2	El sistema vuelve a mostrar la notificación la próxima vez que el usuario se introduzca en la aplicación.
	3.4	Si el usuario no dispone de conexión con la red se informa al usuario y se aborta la transmisión de datos
Frecuencia esperada	Baja.	
Importancia	Requisito secundario.	

3. 13. Desarrollo del caso de uso AceptarMedico.

Identificador	**CerrarSesion**	
Descripción	Un médico solicita a un paciente que se asocie con él.	
Actores	Paciente, Experimentador, Médico.	
Precondición	El usuario está logueado en el sistema.	
	Paso	**Acción**

Escenario principal de éxito (flujo normal)	1	El usuario selecciona del menú de la pantalla principal la opción “Cerrar Sesión”.
	2	El sistema redirige al usuario a la pantalla de inicio.
Postcondición	El usuario se desconecta del sistema.	
Flujo alternativo	**Paso**	**Acción**
Frecuencia esperada	Media.	
Importancia	Requisito secundario.	

3. 14. Desarrollo del caso de uso CerrarSesion.

3.2.2. Diseño de la interfaz de gráfica

Una interfaz gráfica de usuario (GUI, Graphical User Interface) consiste en un entorno visual que facilita el uso de un sistema utilizando un conjunto de imágenes y objetos para representar la información y operaciones disponibles.

Una interfaz gráfica bien formada debe cumplir varias características para convertirse en una interfaz que mejore la experiencia del usuario y sea realmente amigable para él. Entre estas características podemos destacar: la claridad, referida al nivel de comprensión que alcanza el usuario al usar la interfaz; la brevedad, de modo que no se sature al usuario con demasiada información; la intuitividad, que implica que el diseño generado es familiar para el usuario y la curva de aprendizaje en el entorno es rápida; la fluidez, con la que conseguimos un avance rápido y cómodo entre las distintas pantallas; y la atracción, una característica complicada de medir y que incluye los colores, figuras, letras, estilos… que se son utilizados en el diseño de la interfaz.

De las características del sistema localizadas a lo largo de la fase de diseño podemos deducir algunas de las condiciones que tiene que cumplir la interfaz que generaremos y las pantallas que debemos crear, pues ya conocemos la mayor parte de los escenarios a los que nos vamos a enfrentar.

Entre las interfaces que desarrollaremos se encuentran:

- **Pantalla principal para común para usuarios con perfil paciente y experimentador**. Ofrecerá las funcionalidades relacionadas con los mismos, esto es: consultar sus datos de usuario, realizar una medida, ver el historial de medidas.

- **Pantalla principal para usuarios con perfil médico**. Ofrecerá las funcionalidades relacionadas con los mismos, esto es: consultar datos de usuario, asociarse con un paciente, ver el listado de pacientes.

- **Pantalla de login para introducirse en el sistema**. En esta pantalla tendremos que introducir DNI y contraseña para luego enviarlos al servidor.

- **Pantalla de registro para pacientes**. En esta pantalla se tendrán que poder introducir todos los datos relativos al nuevo usuario.

- **Información de los datos del usuario**. En ella se mostrarán todos los datos introducidos en el registro del mismo, tendremos que introducir algunos cambios en función del tipo de usuario del que se trate.

- **Historial de medidas**. En esta pantalla aparecerán todas las medidas que ha realizado un usuario ordenadas por fecha de realización, también añadiremos la posibilidad de mostrarlas marcadas sobre un calendario.

- **Lista de pacientes de un médico**. En esta pantalla aparecerán todos los pacientes relacionados con un médico, ordenados alfabéticamente.

- **Pantalla para la realización de medidas**. La información que debemos ofrecer en este caso es bastante variada: gráfica con los datos recogidos, información sobre cómo se debe realizar la medida, información sobre la realización de calibraciones, opciones para repetir, cancelar y enviar las medidas… En el caso de los perfiles experimentadores se debe disponer de una vista adicional en la que introducir los datos de la persona que ha realizado la medida.

3.2.3. Diseño de la base de datos

Con lo que ya conocemos del proyecto podemos deducir cuáles son los datos que necesitamos para el correcto funcionamiento del sistema. Para simplificar el estudio de estos datos, los dividiremos en cuatro bloques, en función de la información con la que tratan cada uno de ellos:

- **Datos relacionados con las medidas**. En lo que respecta a cada una de las medidas que se realizan en el sistema debemos guardar, por supuesto, los valores recogidos de los tres ejes (x, y, z) del acelerómetro y del giroscopio del dispositivo móvil, además, tendremos que almacenar la duración de la medida y la frecuencia a la que se muestrea para ambos sensores (no tiene por qué ser la misma).
  Uno de los requisitos identificados durante la fase de análisis consistía en que cada medición del temblor debía ir asociada a una medida de calibración, por

lo tanto, debemos incluir también esos datos capturados (al igual que con la medición, para los tres ejes de los dos sensores).
También guardaremos la fecha en que se realiza la medida (nos servirá más adelante para listarlas ordenadamente) y toda la información que podamos recopilar sobre el acelerómetro y el giroscopio que utiliza el dispositivo.
Por último, hay que destacar que tendremos que implementar una relación entre cada una de las medidas y el paciente que la ha realizado.

- **Datos relacionados con los usuarios del sistema**. En primer lugar, tenemos que almacenar toda la información imprescindible para el registro de los usuarios, esto es: DNI, email, contraseña[10], sexo, fecha de nacimiento, nombre y apellidos; pero también podemos almacenar algunos datos más como son: dirección, teléfono, patología (para los pacientes) y centro médico asociado (para los médicos).
  Debido a las condiciones impuestas por el sistema igualmente será necesario indicar de alguna forma cuales son los privilegios de los que dispondrá el usuario, si tiene o no alguna notificación pendiente y, para el caso de los pacientes, si están asociados a algún médico.
  Con objeto de simplificar las consultas a realizar sobre los usuarios repartiremos todos estos datos en tres tablas: una general con los datos únicos de los usuarios, otra para el almacenamiento de pacientes/experimentadores (por las similitudes entre sus perfiles) y una última para el almacenamiento de los médicos.

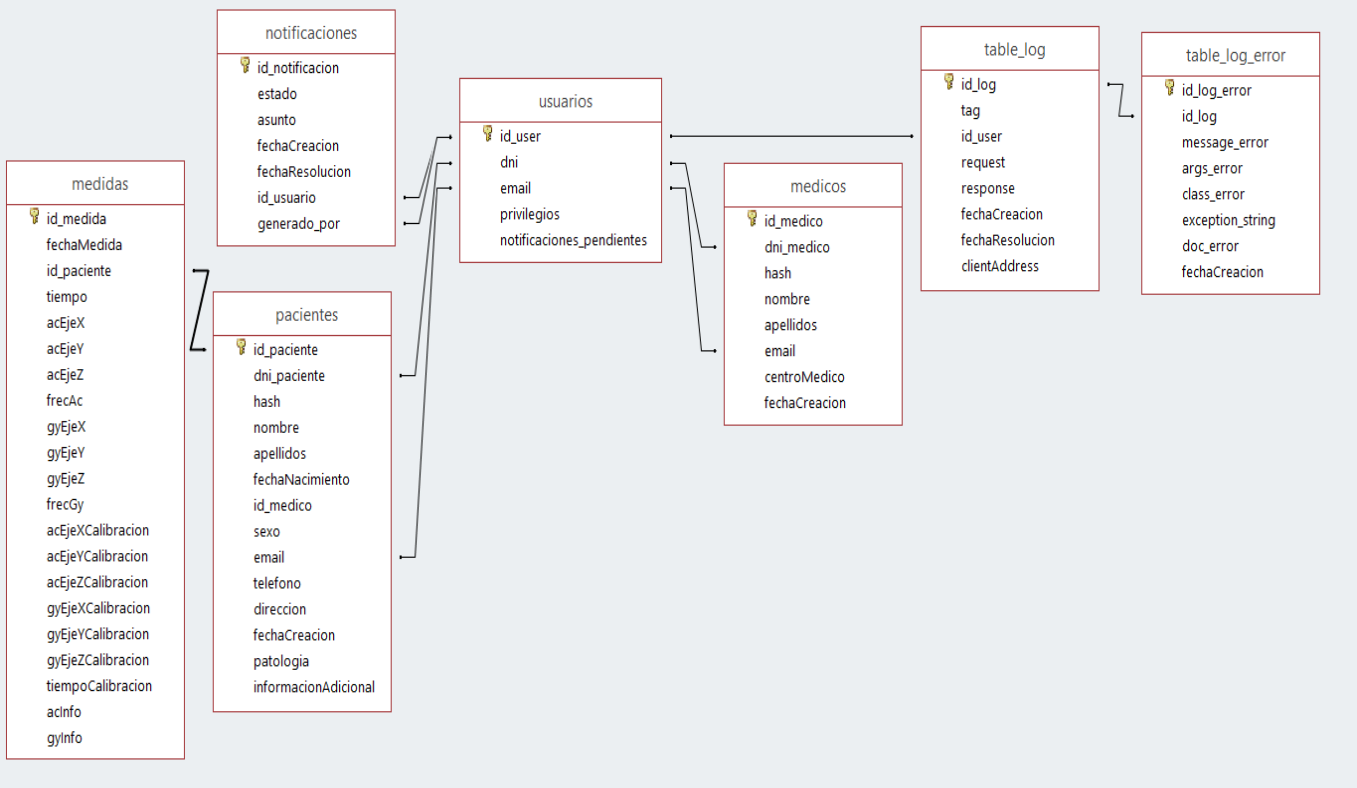
- **Datos relacionados con las notificaciones**. En lo relativo a las notificaciones intentaremos almacenar los datos de la manera más genérica posible, de forma que el sistema sea más fácilmente escalable al poder utilizar este almacén para nuevos tipos de notificación. En este contexto, necesitaremos almacenar el asunto de la notificación, así como su estado, quién ha desencadenado la creación de la notificación y hacia quién va dirigida.
  Otro dato práctico que podemos almacenar son las fechas en que se crean y resuelven cada una de las notificaciones.

- **Datos relacionados con las interacciones con el sistema**. Aunque estos datos no responden a ninguno de los casos de uso ni requisitos identificados para el proyecto, sí que pueden ser significativos y prácticos a nivel de gestión del sistema ya que nos permitirán controlar las interacciones que tienen lugar con nuestro servidor e identificar los errores que pudieran generarse de dicha interacción.
  Entre los datos que se pueden almacenar para esta situación se encuentran la dirección IP de la que llega la petición, el asunto de la petición, la propia petición y la respuesta generada por el servidor, de igual modo tenemos que relacionar cada registro de una interacción con el usuario que la genera.

[10] Respecto a la contraseña, y por seguridad, lo que almacenaremos será un código hash obtenido a partir de la misma de modo que la contraseña original no sea visible en ningún momento.

Para los casos en los que se desencadene algún error en el sistema almacenaremos una entrada propia con toda la información que podamos recoger del mismo y relacionaremos esta con la interacción que la genera.

Con toda esta información podemos generar una imagen de lo que será la base de datos al combinar todas las tablas necesarias para el almacenamiento de la información del sistema y que posteriormente implementaremos.



3. 15. Diseño de la base de datos.

3.3. Fase de construcción o implementación

La fase de construcción comprenderá la mayor carga, tanto en lo que a tiempo como a trabajo se refiere, del ciclo de vida del proyecto. Basándonos en los resultados de la fase de diseño tendremos que convertir la idea planteada inicialmente en algo tangible que, finalmente, constituya nuestro sistema.

Para llevar a cabo el desarrollo del proyecto de forma que se puedan ir obteniendo resultados progresivos, se adoptará un desarrollo basado en iteraciones. De esta forma, al final de cada iteración obtendremos una versión del sistema que ofrecerá una

funcionalidad de entre las que se plantearon al realizar el análisis y sobre la que podremos realizar pruebas unitarias[11] con el fin de comprobar si el resultado obtenido es el esperado.

3.3.1. Implementación de la aplicación móvil

Como decidimos previamente, la aplicación móvil se desarrollará para el sistema operativo Android, lo que nos permite realizar la construcción en varios lenguajes de programación. De entre esos lenguajes hemos elegido Java, que es el lenguaje oficial y más utilizado actualmente para este entorno.

El desarrollo de aplicaciones móviles implica una programación orientada a eventos, lo que significa que el flujo del programa viene determinado por las acciones del usuario (eventos), éstas pueden ir desde pulsaciones de teclas hasta gestos en la pantalla táctil.

El programa estará formado por un bucle principal con dos secciones: la detección de eventos y la gestión del evento, de las cuales solo tendremos que preocuparnos de la segunda, pues la detección de eventos vendrá proporcionada por el entorno de programación.

Dentro de nuestro sistema intentaremos mantener una arquitectura de tres capas [31] lo más clara posible, que nos permita simplificar tanto el desarrollo como el posterior mantenimiento:

- **Capa de presentación** (vista): se encargará de la presentación de los resultados de la aplicación y comprenderá las interfaces de la aplicación móvil desarrolladas en lenguaje XML y parte de los ficheros Java que interactúan con ellas.

- **Capa de lógica de negocio** (controlador): comprenderá las tareas que realizará la aplicación sobre los datos y los servicios ofrecidos por el sistema. Se trata de la parte de desarrollo más densa y compleja de nuestra aplicación y estará repartida entre la lógica implementada en la aplicación (Java) y la del servidor (Python).

- **Capa de datos** (modelo): consiste, básicamente, en el conjunto de los datos sobre los que operará la aplicación, aunque para nuestra aplicación la mayor parte de ellos se localizarán en la base de datos MySQL alojada en el servidor, también necesitaremos de algunos ficheros con información básica y útil de acceso local para la aplicación móvil y el propio servidor.

[11] Pruebas unitarias: pruebas realizadas sobre la aplicación móvil con independencia del servidor, y viceversa. Aunque estas pruebas y las de integración deberían constituir planes propios, tienen múltiples casos en común y es por eso por lo que se han combinado en un plan de pruebas conjunto que se puede consultar en la sección 3.4.2.

A continuación, se enumeran las clases Java generadas para la resolución del sistema propuesto junto con su descripción y el fichero XML que contiene la declaración de la interfaz gráfica asociada, en el caso de que la tuvieran. En el Apéndice I se encuentra la información detallada de las clases junto a la descripción de sus atributos y sus métodos.

Clase	Descripción	XML asociado
AddPatientActivity	Actividad encargada de solicitar el DNI/email de un paciente y enviar a dicho paciente una petición para relacionarlo con un médico.	*activity_add_patient*
DetectionActivity	Actividad encargada de grabar los cambios en los sensores cuando el usuario inicia una medición.	*activity_detection*
DetectionActivityInfo	Actividad encargada de mostrar la gráfica con los datos previamente grabados al medir y ofrecer la posibilidad de enviar los mismos al servidor para su almacenamiento.	*activity_detection_info*
DoctorInformationActivity	Actividad encargada de mostrar los datos de un usuario con rol de médico.	*activity_doctor_information*
Graph	Clase encargada de crear y mostrar las gráficas de visualización de datos a través de las funcionalidades ofrecidas por la librería GraphView[12].	-
HistoryActivity	Actividad encargada de mostrar el historial de medidas que están asociadas a un paciente.	*activity_history* *activity_history_calendar*

[12] GraphView [32]: librería, de código abierto, para la creación de diagramas en Android. Se trata de una librería fácil de integrar en nuestro proyecto y que nos permite la creación de gráficas de múltiples tipos y fáciles de personalizar. En nuestro sistema será la base de la que partiremos para la representación de las señales capturadas del acelerómetro del dispositivo móvil en una gráfica de líneas y la usaremos tanto en las representaciones en tiempo real como en las de recuperación de medidas almacenadas en el servidor.

HttpClient	Clase encargada de las comunicaciones https entre la aplicación y el servidor.	-
LoginActivity	Actividad que ofrece una pantalla de login a través de DNI y contraseña.	*activity_login*
MainActivity	Actividad principal de los pacientes y experimentadores, que muestra todas las funcionalidades ofrecidas al mismo.	*activity_main*
MainDoctorActivity	Actividad principal del médico, que muestra todas las funcionalidades ofrecidas al mismo.	*activity_main_doctor*
MeasureActivityInfo	Actividad que muestra la información relativa a una medida concreta.	*activity_measure_info*
PatientInformationActivity	Actividad encargada de mostrar los datos personales de un paciente.	*activity_patient_information*
PatientListActivity	Actividad que muestra la lista de los pacientes que tiene ligados un médico y desde la cual se puede acceder a las funcionalidades de cualquiera de ellos.	*activity_patient_list*
RegisterActivity	Actividad que ofrece un formulario de registro para poder interactuar con la aplicación.	*activity_register*
ResponseException	Excepción que se genera cuando la respuesta recibida por el servidor indica que no ha habido éxito en alguna petición o cuando se genera algún error propio en la aplicación.	-

SecondaryTask	Clase encargada de crear la tarea asíncrona que interactuará con el servidor en segundo plano para enviar y recibir información.	-
SettingsActivity	Actividad encargada de manejar los distintos ajustes de la aplicación, en principio, duración y frecuencia de muestreo de las medidas.	*settings*
Utils	Clase que contiene los literales para identificar recursos tanto en las comunicaciones cliente-servidor cómo dentro del propio cliente y algunas funciones útiles para la aplicación.	-

3. 16. Clases Java de la aplicación móvil.

3.3.2. Implementación del servidor

Nuestro servidor consistirá en un ordenador en el que se estará ejecutando el programa Python (que debemos implementar) de forma ininterrumpida. Cómo el desarrollo de la aplicación móvil ya ha finalizado, esta implementación consistirá en gestionar y responder las peticiones, que ya son conocidas, y que nos llegarán desde cualquiera de nuestros clientes.

Entre las tareas de nuestro servidor estará la de actuar de intermediario entre la aplicación móvil y la base de datos que contiene toda la información que ésta requiere para su correcto funcionamiento.

Otra de las tareas, de las que ya hemos hablado anteriormente, consistirá en almacenar para cada petición recibida una entrada, o log, que podamos consultar con el asunto de la interacción y lo que ha surgido a partir de ella (ya sea una respuesta correcta o un error imprevisto).

A continuación, se enumeran los módulos Python desarrollados junto con una breve descripción de los mismos, que constituyen la solución adoptada para la implementación del servidor. En el Apéndice II se encuentra la documentación detallada asociada a estos módulos y sus respectivas funciones.

Módulo	Descripción
database	Módulo encargado de conectar con la base de datos y ejecutar las consultas necesarias para el tratamiento y recuperación de los datos.
gestionarMedidas	Módulo encargado de los servicios relacionados con las mediciones: grabación, recuperación, historiales…
Log	Clase encargada del registro de las interacciones entre clientes y servidor.
notificaciones	Módulo encargado de la generación y recuperación de notificaciones y de la gestión de los casos de uso que tratan con ellas.
server	Módulo principal a la escucha de peticiones generadas por los clientes y que las redirige al módulo apropiado en función del servicio solicitado.
usuarios	Módulo encargado de los servicios relacionados con los usuarios del sistema: login, registro, datos de usuario…
Utils	Clase que contiene los literales para identificar recursos tanto en las comunicaciones cliente-servidor cómo dentro del propio servidor y algunas funciones útiles para el sistema.

3. 17. Módulos Python del servidor

.

Aunque el sistema se plantea enfocado hacia al flujo normal de los casos de uso, igualmente tenemos que estar preparados para casos en los que la situación no discurre del modo ideal. En estas situaciones, o casos de no éxito, el servidor tendrá que informar a la aplicación de que nos hemos desviado del escenario de éxito y que debemos proceder de otra manera.

Con objeto de comunicar estas situaciones se han desarrollado una serie de códigos de error internos, conocidos tanto por la aplicación como por el servidor, que serán utilizados para notificar la situación ante la que nos encontramos dentro del sistema y que podemos consultar en la siguiente tabla.

Código de error	Descripción
1xx	Errores asociados a los usuarios de la aplicación
100	Usuario no registrado (tabla de pacientes)
101	Contraseña incorrecta (tabla de pacientes)

102	Usuario no registrado (tabla de médicos)
103	Contraseña incorrecta (tabla de médicos)
104	Usuario no registrado (tabla usuarios)
105	Privilegios no reconocidos
150	No hay pacientes asociados a ese médico
2xx	Errores asociados al registro de usuarios
200	DNI ya registrado
201	Fallo de recuperación de datos servidor
202	DNI no válido
203	Password no válido
204	Nombre no válido
205	Apellidos no válidos
206	Fecha de nacimiento no válida
207	Email no válido
208	Género no válido
209	Email ya registrado
3xx	Errores asociados a las nuevas mediciones
300	Faltan datos para grabar la medida
4xx	Errores asociados al tratamiento y recuperación de las medidas registradas en el sistema
400	No hay mediciones
401	El id_medida no corresponde con ninguna medida
402	Más de una medida eliminada
5xx	Errores asociados a las notificaciones
500	Se buscan notificaciones, pero no se encuentran
501	Paciente ya solicitado por un médico

9xx	Errores propios de red/servidor y desconocidos (generados en la aplicación)
999	Error genérico -> sin respuesta del servidor
998	Tiempo de espera agotado
997	URL de servidor mal formada
996	Dispositivo sin acceso a red

3. 18. Códigos de error del sistema.

3.3.3. Comunicaciones cliente – servidor

Una cuestión importante de implementación que debemos tener en cuenta es que para realizar la transferencia de datos entre la aplicación móvil y el servidor de forma segura necesitaremos un canal cifrado en las comunicaciones de nuestro sistema. Para ello nos valdremos del protocolo HTTPS (Hypertext Transport Protocol Secure), que utiliza un cifrado basado en SSL/TLS sobre el protocolo HTTP.

Para que nuestro servidor acepte conexiones HTTPS debemos crear un certificado de clave pública, lo cual haremos con ayuda de la herramienta OpenSSL[13]. En primer lugar, crearemos un par de claves RSA, pública y privada; dónde el texto encriptado con una será desencriptado únicamente con la otra, y viceversa.

Empleando el par de claves generadas anteriormente podemos crear el certificado, que contendrá la clave pública que nuestros clientes tendrán que utilizar tanto para encriptar los datos que serán enviados al servidor (y que el servidor desencriptará con su clave privada), como para desencriptar los datos que serán recibidos del servidor (y que el servidor habrá encriptado previamente con su clave privada).

El último paso consistiría en que una autoridad certificadora de confianza firme el certificado para que este pueda ser posteriormente instalado en nuestro servidor y utilizado en todas las comunicaciones.

[13] OpenSSL: paquete de herramientas criptográficas que permite la creación y uso de claves y la generación de certificados y firmas digitales, entre otros servicios.

3.4. Fase de pruebas y despliegue

Ésta será la última fase del ciclo de vida de nuestro proyecto y concluirá con el lanzamiento del sistema a los usuarios finales. Durante ella se llevará el sistema a una versión final que cumplirá con todas las especificaciones requeridas utilizando para eso el feedback obtenido de las pruebas, principalmente de integración, entre los dos sistemas construidos.

Entre las tareas que corresponden a esta fase nos encontramos con la necesidad de adaptar el software servidor a la máquina que tendrá que albergarlo, la resolución de las incidencias que surjan durante las pruebas y la creación de manuales de usuario.

3.4.1. Despliegue del sistema

En lo que a nuestra aplicación móvil respecta, lo primero que debemos hacer es generar el fichero instalable que requerirán los usuarios. Este fichero APK (Android Application Package) será en el que se empaquetará nuestra aplicación (todos los ficheros que sean necesarios) y tendrá una extensión .apk.

El fichero APK que generaremos estará firmado con un certificado digital, de ese modo, garantizaremos su origen y, si fuese necesario, podríamos subirlo al servicio de aplicaciones de Google Play e ir actualizándolo con sucesivas versiones.

Con el fichero ejecutable generado cualquier usuario en disposición de un dispositivo Android podrá instalarse nuestra a aplicación[14]. Ésta, además, ha sido diseñada de modo que sea adaptable para múltiples dimensiones de pantalla de modo que las interfaces generadas no sufran cambios notables entre la mayor parte de pantallas existentes.

En cuanto al software del servidor, tendremos que crear un entorno virtual (*virtual enviroment*) dentro de nuestra máquina en el que lanzarlo [33]. Un entorno virtual consiste en un directorio en el que se incluyen la instalación de la versión de Python a utilizar y los paquetes adicionales (que no formen parte de la librería estándar) que usará nuestro proyecto.

Otra de las tareas que debemos llevar a cabo, será la conexión del servidor con nuestra base de datos, de modo que las interacciones entre ambos se realicen correctamente y se pueda leer y escribir en ella durante el uso del sistema.

[14] La aplicación se ha desarrollado de tal forma que se dé soporte a versiones del sistema operativo Android desde la API 15 (*release* Android 4.0.3, familia: Ice Cream Sandwich), lo que engloba casi el 100% de los dispositivos Android disponibles actualmente.

Tras esto, y una vez que activemos el entorno, ya podemos lanzar nuestro programa servidor, que estará escuchando en uno de los puertos de la máquina, a la espera de las peticiones que se reciban por parte de nuestros clientes.

A partir de este momento, todas las partes de nuestro sistema se hallarán conectadas entre sí y podremos usarlo de la forma en que se estableció en las especificaciones. Nuestra misión ahora consistirá en encontrar los posibles fallos que puedan aparecer en él (o *bugs*) y solucionarlos con el desarrollo de nuevas versiones, en las que incluso se podrían llegar a añadir nuevas funcionalidades.

3.4.2. Plan de pruebas

El plan de pruebas del proyecto se diseña con el fin de validar y comprobar que se cumplen los requisitos planteados para el sistema. Con la ejecución de este plan podemos recoger información sobre los defectos presentes en la solución generada, con el fin de aplicar las correcciones necesarias para alcanzar un producto de calidad y que funcione de manera adecuada.

A continuación, podemos encontrar las pruebas desarrolladas para la comprobación de las principales funcionalidades del sistema. La mayoría de estas pruebas deberían haberse realizado de manera independiente en la aplicación o el servidor al terminar la construcción de la misma, pero ahora, deben ser repetidas para la comprobación del funcionamiento del sistema en su conjunto.

- Pruebas de acceso al sistema (login).

Prueba	**Resultado esperado**
El usuario no introduce el DNI ni la contraseña.	La aplicación muestra un aviso informando de que ambos campos deben rellenarse. Ningún dato es enviado al servidor.
El usuario introduce su DNI, pero no la contraseña o viceversa.	La aplicación muestra un aviso informando de que debe rellenarse el campo incompleto. Ningún dato se envía al servidor.
El usuario introduce un DNI que no está registrado en el sistema.	Se envía la información al servidor, que responde con el error correspondiente. La aplicación informa de que el usuario no está registrado.
El usuario introduce un DNI que está registrado en el sistema, pero la contraseña no es la correcta.	Se envía la información al servidor, que responde con el error correspondiente. La aplicación informa de que la contraseña no es correcta.

El usuario intenta acceder sin tener acceso a la red.	La aplicación muestra un aviso informando de que el usuario debe conectarse a la red. Ningún dato se envía al servidor.
El usuario intenta acceder y el servidor está desconectado.	Se intentan transmitir los datos al servidor, al no recibir respuesta se muestra un aviso al usuario informando de lo sucedido.
El usuario intenta acceder con un DNI registrado en el sistema y la contraseña correcta.	Se envía la información al servidor, que informa que los datos son correctos. La aplicación pasa a la pantalla principal del usuario en función del perfil del mismo.

3. 19. Plan de pruebas para el login.

- Pruebas de registro en el sistema.

Prueba	**Resultado esperado**
El usuario no rellena los datos considerados obligatorios, esto es: DNI, nombre, apellidos, fecha de nacimiento, contraseña, confirmación de la contraseña, género, correo electrónico.	La aplicación muestra un aviso informando de que todos los campos deben rellenarse. Ningún dato es enviado al servidor.
El usuario introduce un DNI o un correo electrónico que no cumple con los patrones propios de un DNI o de un correo electrónico.	La aplicación muestra un aviso informando de que valores introducidos no son válidos. Ningún dato es enviado al servidor.
El usuario introduce una fecha de nacimiento que carece de sentido (mes superior a 12, día superior a 31…).	La aplicación muestra un aviso informando de que la fecha introducida no es válida. Ningún dato es enviado al servidor.
El usuario no lee, y por lo tanto no puede aceptar, los términos y condiciones de uso de la aplicación.	La aplicación muestra un aviso informando de que deben aceptarse los términos de uso. Ningún dato es enviado al servidor.
El usuario introduce una contraseña y una confirmación de la misma que no coinciden.	La aplicación muestra un aviso informando de que las contraseñas no coinciden. Ningún dato es enviado al servidor.
El usuario introduce un DNI o un correo electrónico que ya está registrado en el sistema.	Se envía la información al servidor, que responde con el error correspondiente. La aplicación informa de que el DNI o el correo electrónico ya están registrados.

El usuario intenta registrarse sin tener acceso a la red.	La aplicación muestra un aviso informando de que el usuario debe conectarse a la red. Ningún dato se envía al servidor.
El usuario intenta registrarse y el servidor está desconectado.	Se intentan transmitir los datos al servidor, al no recibir respuesta se muestra un aviso al usuario informando de lo sucedido.
El usuario rellena todos los datos obligatorios correctamente y acepta los términos y condiciones de uso.	Se envía la información al servidor, que informa que los datos son correctos. El usuario queda registrado en el sistema con perfil de paciente y la aplicación pasa a la pantalla principal del usuario.

3. 20. Plan de pruebas para el registro.

- Pruebas de realización de una medida por parte de un paciente y un experimentador.

Prueba	**Resultado esperado**
El paciente intentar realizar una medición sin que su dispositivo disponga de acelerómetro o giroscopio.	La aplicación muestra un aviso informando de que es necesario disponer de acelerómetro y giroscopio. Ningún dato se envía al servidor.
El paciente comienza la calibración de los sensores y durante ella aplica un movimiento sobre el dispositivo móvil que hace que se superen los umbrales máximos establecidos para dar por válida la calibración.	La aplicación muestra un aviso informando de que la calibración ha fallado y se ofrece al usuario la posibilidad de comenzar de nuevo.
El paciente realiza la calibración correctamente, pero decide utilizar la opción de repetir calibración.	La aplicación ofrece al usuario la posibilidad de volver a realizar la calibración.
El paciente, durante la realización de una medida (ya sea de calibración o de temblor) bloquea la pantalla el dispositivo o pulsa el botón “Atrás”.	Se detiene la grabación de la medida y se devuelve al usuario a la pantalla anterior: comienzo de calibración o comienzo de medición de temblor.
El paciente graba una medición de temblor y decide enviarla sin tener acceso a la red.	La aplicación muestra un aviso informando de que el usuario debe conectarse a la red. Ningún dato se envía al servidor.

El paciente graba una medición y decide enviarla estando el servidor desconectado.	Se intentan transmitir los datos al servidor, al no recibir respuesta se muestra un aviso al usuario informando de lo sucedido.
El paciente graba una medición de temblor y la envía al servidor correctamente.	Se envía la información al servidor, que informa de que la medición se ha grabado con éxito. Los datos relacionados con la medición realizada, y su calibración correspondiente, quedan registrados en el sistema. El usuario vuelve a la pantalla principal de la aplicación.
El paciente graba una medición de temblor y la envía al servidor correctamente mediante la opción de "Enviar y realizar nueva medida"	Se envía la información al servidor, que informa de que la medición se ha grabado con éxito. Los datos relacionados con la medición realizada, y su calibración correspondiente, quedan registrados en el sistema. El usuario es enviado a la pantalla de comienzo de nueva medición de temblor
El paciente graba una medición de temblor, pero decide repetirla.	El usuario es enviado a la pantalla de comienzo de nueva medición de temblor. Ninguna información es enviada al servidor.
El paciente graba una medición de temblor, pero decide no guardarla.	El usuario es enviado a la pantalla principal de la aplicación. Ninguna información es enviada al servidor.

3. 21. Plan de pruebas para la realización de medidas.

A continuación, repetimos las pruebas para el caso del perfil experimentador, si recordamos, la diferencia reside en que al realizar el envío desde un perfil experimentador tendremos que introducir ciertos datos sobre quien realiza la medida: fecha de nacimiento, género, lo cual supone añadir una nueva prueba para este caso.

Prueba	**Resultado esperado**
El usuario no rellena la fecha de nacimiento o introduce una que carece de sentido (mes superior a 12, día superior a 31…).	La aplicación muestra un aviso informando de que la fecha introducida no es válida. Ningún dato es enviado al servidor.

3. 22. Pruebas adicionales para la realización de medidas con perfil experimentador.

- Acceso a las funcionalidades de un paciente por parte de un médico.

Prueba	**Resultado esperado**
El médico accede a la lista de pacientes y no está asociado con ninguno.	Se envía la petición al servidor, que contesta con el error correspondiente. Se informa al usuario de que la lista está vacía.
El médico intenta acceder a la lista de pacientes sin tener acceso a la red.	La aplicación muestra un aviso informando de que el usuario debe conectarse a la red. Ningún dato se envía al servidor.
El médico intenta acceder a lista de pacientes y el servidor está desconectado.	Se intentan transmitir los datos al servidor, al no recibir respuesta se muestra un aviso al usuario informando de lo sucedido.
El médico escoge a uno de sus pacientes de la lista y accede a sus funcionalidades correctamente.	Se envía la información al servidor, que contesta con el identificador del paciente seleccionado y redirige al médico a la pantalla principal de éste.

3. 23. Plan de pruebas para la gestión de pacientes

.

- Visualización de los datos de un usuario.

Este caso tiene una complejidad menor y tan solo comprobamos que los datos se reciben y muestran correctamente al seleccionar la opción correspondiente del menú principal (además del caso en el que no se dispone acceso a la red o el servidor está desconectado).

- Asociación de un médico y un paciente.

Prueba	**Resultado esperado**
El médico accede a la opción “Asociar Paciente” y no introduce un DNI o correo electrónico.	La aplicación muestra un aviso informando de que debe rellenarse el campo. Ningún dato es enviado al servidor.
El médico accede a la opción “Asociar Paciente” e introduce un DNI o un correo electrónico que no cumple con los patrones correspondientes.	La aplicación muestra un aviso informando de que el formato no es correcto. Ningún dato es enviado al servidor.
El médico accede a la opción “Asociar Paciente” e introduce un DNI o un correo electrónico que no está registrado en el sistema.	Se envía la información al servidor, que responde con el error correspondiente. La aplicación informa de que el usuario no está registrado.

El médico accede a la opción "Asociar Paciente" e introduce un DNI o un correo electrónico sin tener acceso a la red.	La aplicación muestra un aviso informando de que el usuario debe conectarse a la red. Ningún dato se envía al servidor.
El médico intenta accede a la opción "Asociar Paciente" e introduce un DNI o un correo electrónico y el servidor está desconectado.	Se intentan transmitir los datos al servidor, al no recibir respuesta se muestra un aviso al usuario informando de lo sucedido.
El médico accede a la opción "Asociar Paciente" e introduce un DNI o un correo electrónico que corresponde con un paciente que ya está asociado a un médico.	Se envía la información al servidor, que registra la petición e informa a la aplicación de que el usuario seleccionado será notificado.
El médico accede a la opción "Asociar Paciente" e introduce un DNI o un correo electrónico que corresponde con un paciente que ya tiene una petición de asociación a la espera de respuesta.	Se envía la información al servidor, que responde con el error correspondiente. La petición no es registrada. La aplicación informa de que el paciente está pendiente de otra asociación.
El médico accede a la opción "Asociar Paciente" e introduce un DNI o un correo electrónico que corresponde con un paciente que no está asociado con ningún médico ni pendiente de la respuesta a ninguna notificación.	Se envía la información al servidor, que registra la petición e informa a la aplicación de que el usuario seleccionado será notificado.
El paciente se loguea en la aplicación cuando ha recibido una solicitud por parte de un médico y rechaza la asociación.	Se envía la información al servidor, que registra la respuesta a la petición y no formaliza la asociación entre médico y paciente.
El paciente se loguea en la aplicación cuando ha recibido una solicitud por parte de un médico y selecciona la opción de "No contestar aún".	La petición queda pendiente de respuesta y aparece de nuevo en el siguiente logueo del paciente. Ningún dato se envía al servidor.
El paciente se loguea en la aplicación cuando ha recibido una solicitud por parte de un médico y acepta la asociación.	Se envía la información al servidor, que registra la respuesta a la petición y formaliza la asociación entre médico y paciente. A partir de ese momento el paciente aparece en la lista de pacientes del médico.

3. 24. Plan de pruebas para la asociación entre médicos y pacientes.

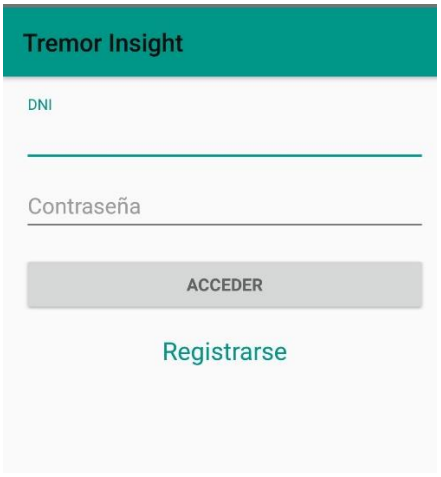
4. Funcionalidad del sistema

A lo largo de este capítulo se explicarán las funcionalidades de la aplicación desarrollada, bautizada con el nombre de ***Tremor Insight***, y el modo de manejo de las mismas, de manera que, esencialmente, podría llegar a aplicarse como un manual de usuario al uso. La aplicación ha sido desarrollada de igual forma para ser manejada en castellano o en inglés, en función del idioma configurado en el dispositivo móvil que se esté utilizando.

4.1. Pantalla inicial

Ésta será la pantalla que aparecerá al lanzar la aplicación y en ella podemos encontrar dos campos de texto para que el usuario introduzca su DNI y contraseña. A través de ella podremos autenticarnos en el sistema y acceder al resto de funcionalidades.

Para usuarios nuevos en el sistema disponemos de un enlace que nos enviará a la pantalla de registro.



4. 1. Pantalla inicial de la aplicación.

4.2. Registro de pacientes

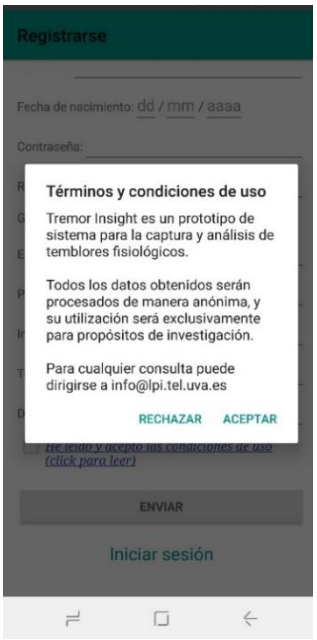
En esta pantalla los usuarios podrán darse de alta con perfil de paciente en el sistema. En ella podemos ver numerosos campos para la introducción de texto de los cuales, aquellos marcados con un asterisco son de carácter opcional.

Los parámetros de carácter obligatorio serán: el DNI del usuario, su nombre, sus apellidos, su fecha de nacimiento, la contraseña que se establecerá para la cuenta (y la confirmación de ésta), su sexo y su correo electrónico.

Los parámetros de carácter opcional serán: patología del usuario si la tuviese, información adicional (este es un campo de texto libre donde se pueden introducir los datos que se consideren oportunos), número de teléfono y dirección.

4. 2. Pantalla de registro de la aplicación.

Por último, debemos leer y aceptar las condiciones de uso de la aplicación, dónde se explica al usuario el tratamiento que se realizará sobre su información personal y se pide su consentimiento para el mismo.



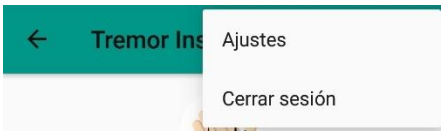
4. 3. Términos y condiciones de uso de la aplicación.

4.3. Pantalla principal

Una vez autenticados en el sistema accederemos a la pantalla principal de la aplicación, donde encontraremos las opciones de las que dispone el usuario.

Según el perfil del usuario distinguimos dos pantallas con opciones distintas: una pantalla principal común para los usuarios con perfil paciente y experimentador y otra pantalla principal para los usuarios con perfil médico. Ambas pantallas tendrán en común:

- Un menú superior desde el que podremos acceder a la pantalla de ajustes y finalizar la sesión del usuario.

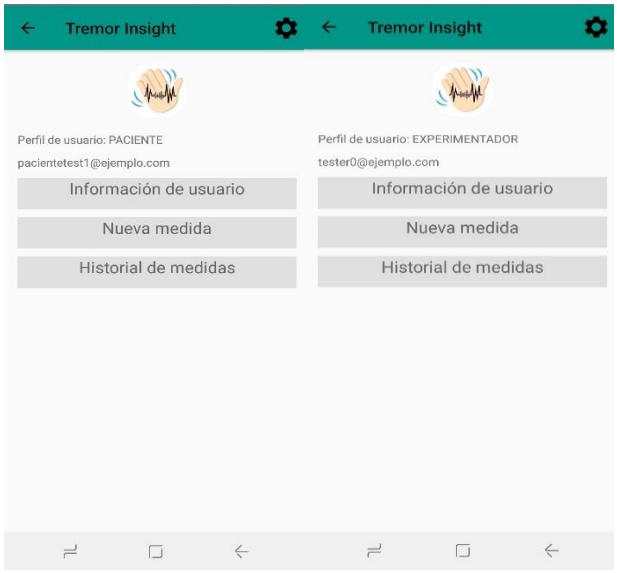


4. 4. Menú superior de la pantalla principal.

- La opción de visualizar la información del usuario.
- Un texto informando acerca del perfil del usuario y su correo electrónico.

4.3.1. Pantalla principal: perfil *paciente* y perfil *experimentador*

Entre las opciones de las que disponen exclusivamente los perfiles paciente y experimentador encontramos la funcionalidad de realizar una nueva medición y la de consultar el historial de medidas del usuario. Podemos acceder a cualquiera de ellas pulsando sobre la opción deseada.



4. 5. Pantalla principal de usuarios con perfil paciente y experimentador.

4.3.2. Pantalla principal: perfil *médico*

Entre las opciones de las que dispone exclusivamente el perfil médico encontramos la funcionalidad de consultar la lista de pacientes asociados al médico en cuestión y la de relacionar pacientes con él. Podemos acceder a cualquiera de ellas pulsando sobre la opción deseada.

4. 6. Pantalla principal de usuarios con perfil médico.

4.4. Consultar información de usuario

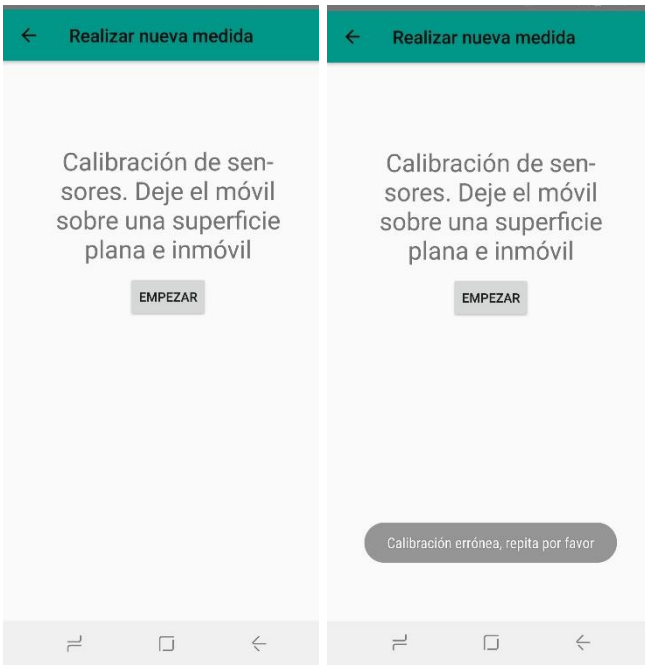
Si seleccionamos la opción de consultar la información de usuario desde la pantalla principal, la aplicación nos enviará a una nueva pantalla, en la que podremos consultar nuestros datos personales. Para volver a la pantalla principal podemos usar indistintamente el botón "Atrás" del dispositivo o el que aparece en la barra superior de la aplicación.

4. 7. Pantalla de consulta de información de usuario.

4.5. Realización de medidas

Si seleccionamos la opción de realizar una nueva medida desde la pantalla principal (con una cuenta de usuario con perfil de *paciente* o de *experimentador*) nos aparecerá una pantalla con las instrucciones para realizar la calibración previa a la medición.

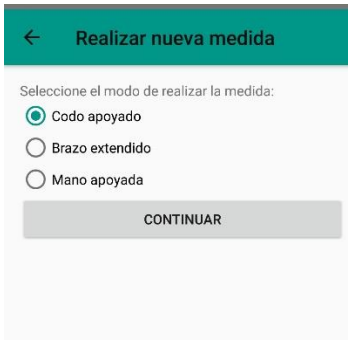
En ese momento, debemos dejar el dispositivo móvil sobre una superficie fija y empezar la calibración pulsando el botón correspondiente. Al hacer esto, aparecerá una cuenta atrás de tres segundos a partir de los cuales se empezarán a recoger los valores medidos por el acelerómetro y el giroscopio del dispositivo, si estos valores superan un umbral establecido se mostrará un aviso informando de que la calibración ha fallado y tendremos que repetirla.



4. 8. Pantalla de comienzo de calibración.

Una vez que se realice la calibración con éxito tenemos que elegir el protocolo que vamos a seguir para realizar la medida. Entre las opciones disponibles tenemos:

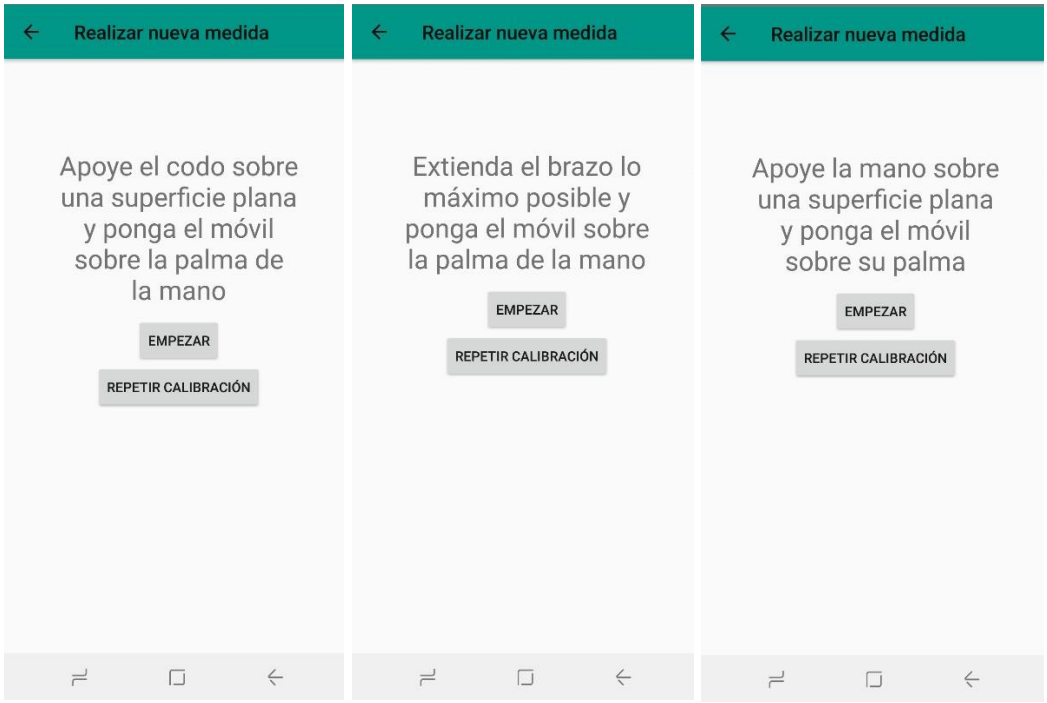
- Grabación del temblor de la mano con el **codo apoyado** sobre alguna superficie.
- Grabación del temblor de la mano con el **brazo extendido**.
- Grabación del temblor de la mano con la **mano apoyada** sobre alguna superficie.



4. 9. Pantalla de selección de protocolo de medición.

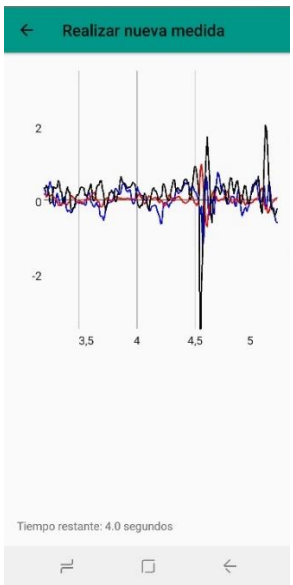
Una vez elijamos el protocolo que consideremos más oportuno pulsamos el botón "Continuar" y podremos ver las indicaciones pertinentes en función del escogido. En este

punto, podremos repetir la calibración que realizamos previamente si así lo deseamos o comenzar con la medición del temblor una vez adoptemos la postura indicada.



4. 10. Pantallas de instrucciones para los distintos protocolos: codo apoyado, brazo extendido, mano apoyada.

Una vez pulsemos el botón para comenzar con la medición aparecerá una cuenta atrás de tres segundos, tras la cual, se empezarán a grabar los valores capturados por el acelerómetro y el giroscopio del dispositivo móvil, al mismo tiempo, podremos ver en la pantalla los datos que se recogen de los ejes x, y, z del acelerómetro y el tiempo restante para completar la medición.

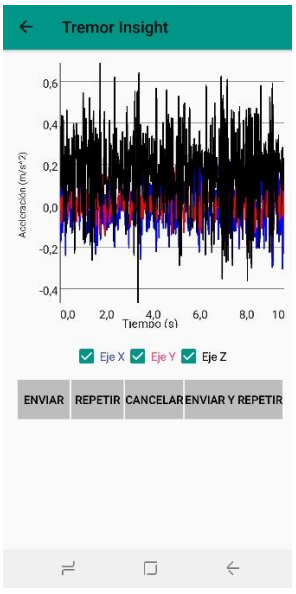


4. 11. Pantalla de muestra de medida mientras se realiza la medición.

Cuando la medida finalice, podremos ver por pantalla la medición completa que hemos grabado en una gráfica en la que se representa el valor obtenido del acelerómetro (en m/s^2) y el instante del proceso de medida en el cual se ha medido (en segundos). En esta gráfica también podremos ocultar o mostrar cualquiera de las tres señales obtenidas e incluso ampliarlas para observar con mayor claridad los datos registrados.

Una vez hayamos observado la medición realizada tendremos varias opciones en una barra de botones situada en el inferior de la pantalla. Las opciones serán:

- **Enviar la medida al servidor**. En este caso, la medida será almacenada en el servidor y volveremos a la pantalla principal de la aplicación.
- **Repetir la medida**. Si algo imprevisto ha sucedido durante el proceso de medición, volveremos al paso posterior a la realización de la calibración de los sensores para realizar la medida correctamente.
- **Cancelar la medida**. Con esta acción la medida grabada no será almacenada en el servidor y volveremos a la pantalla principal de la aplicación.
- **Enviar la medida y repetir**. En este caso, la medida realizada será almacenada en el servidor, pero, en lugar de volver a la pantalla principal de la aplicación, volveremos al paso posterior a la realización de la calibración de los sensores para realizar una nueva medida.



4. 12. Pantalla de visualización de medida.

En el caso de que la medida se realice desde una cuenta con perfil experimentador, al utilizar las opciones “Enviar” o “Enviar y repetir” pasaremos a una pantalla adicional previa al envío de los datos al servidor. En esta pantalla introduciremos, obligatoriamente, la fecha de nacimiento y el sexo de la persona que ha realizado la medida; y, opcionalmente, información sobre si ésta posee alguna patología o algún dato relevante que queramos almacenar sobre ella.

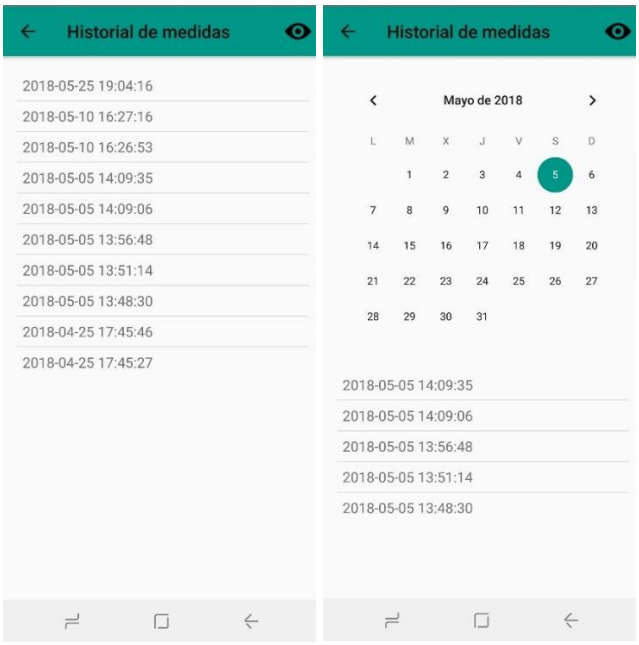
4. 13. Pantalla de introducción de datos adicionales en perfil experimentador.

En cualquier punto del proceso, podemos utilizar el botón “Atrás” del dispositivo móvil o el que tenemos disponible en la barra superior de la aplicación para retroceder a la fase anterior.

4.6. Consultar el historial de medidas

Si seleccionamos la opción de ver el historial de medidas del usuario obtendremos un listado, ordenado por fecha y hora de realización, con todas las medidas que ha registrado.

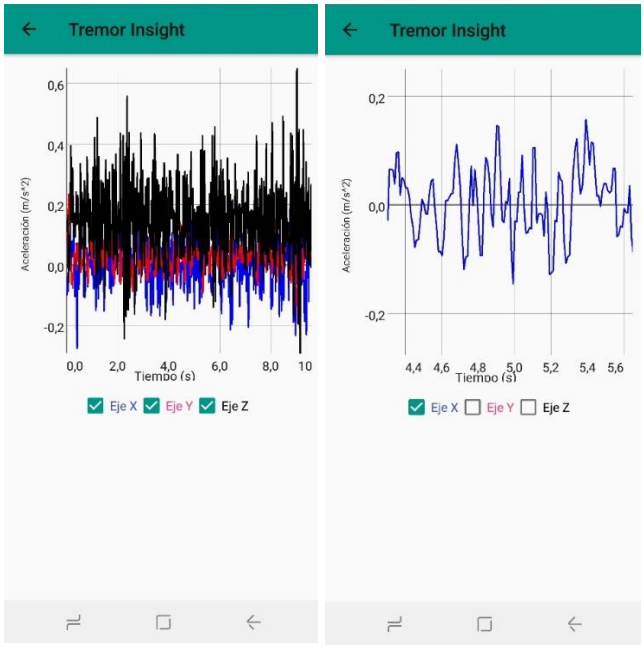
En el menú de la barra superior de la aplicación tenemos la opción de cambiar esta vista a un calendario en el que, seleccionando un día de cualquier mes, podremos ver la lista de medidas filtrada por ese día concreto.



4. 14. Pantallas de historial de medidas: modo lista y modo calendario.

A continuación, y para ambos modos de visualización del historial, podemos seleccionar cualquiera de las medidas para ver la gráfica con los datos grabados previamente para los tres ejes del acelerómetro (en m/s^2) a lo largo de la duración del proceso de medición.

Cuando estemos viendo una medida concreta, tenemos la posibilidad de ampliar o reducir el tamaño de la misma, así como ocultar y mostrar cualquiera de las señales obtenidas de los tres ejes del sensor.

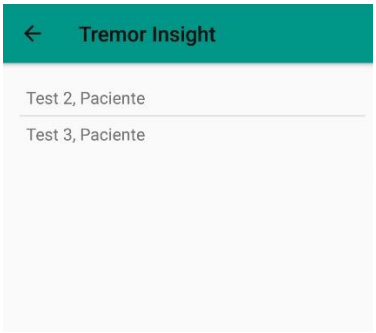


4. 15. Pantalla de visualización de medidas del historial.

Desde la pantalla de visualización de una medida podemos usar el botón “Atrás” propio del dispositivo o el que aparece en la barra superior de la aplicación para volver de nuevo a la lista de medidas. Si realizamos la misma acción desde la pantalla con la lista de medidas volveremos a la pantalla principal de la aplicación.

4.7. Consultar el listado de pacientes

Si accedemos a la pantalla principal de la aplicación con un perfil *médico* y seleccionamos la opción de ver la lista de pacientes, obtendremos el nombre y apellidos de todos los pacientes asociados con esa cuenta (si los hubiera), ordenados alfabéticamente en función de su apellido.



4. 16. Pantalla de lista de pacientes.

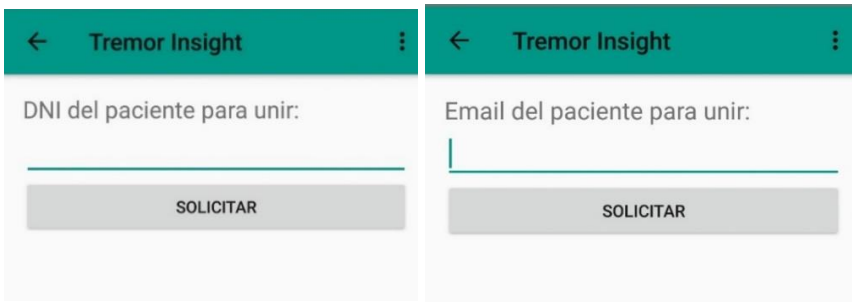
A continuación, y simplemente con pulsar sobre cualquiera de los nombres disponibles, podremos acceder a la pantalla principal de dicho paciente y tener acceso a las funcionalidades disponibles para el mismo.

Del mismo modo que para el resto de casos, podemos retroceder en cualquier punto del proceso mediante el botón “Atrás” del propio dispositivo móvil o el que tenemos disponible en la barra superior de la aplicación.

4.8. Proceso de asociación médico - paciente

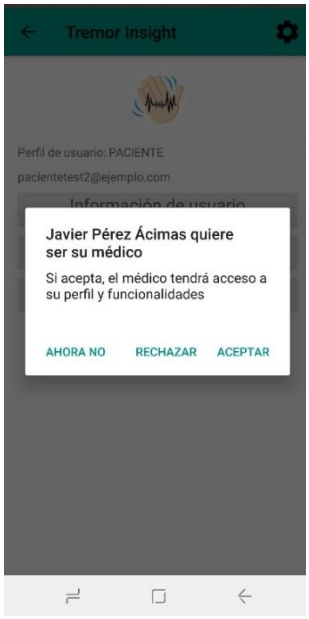
Con el proceso de asociar un paciente, un médico podrá disponer del paciente asociado en su lista de pacientes y visualizar tanto sus medidas como sus datos.

Si elegimos la opción de asociar a un nuevo paciente (desde la pantalla principal de un usuario con perfil médico), accederemos a una nueva pantalla en la que tendremos que introducir el DNI del paciente con el que queremos asociarnos. Si así lo deseamos podemos introducir su correo electrónico, si previamente seleccionamos este modo de asociación en el menú superior de la misma pantalla.



4. 17. Pantallas de asociación de paciente: modo asociación por DNI y modo asociación por email.

Una vez que la solicitud haya sido enviada y el paciente seleccionado se introduzca en la aplicación desde su cuenta, recibirá un mensaje informándole de la solicitud recibida y podrá elegir si aceptar al médico que la ha enviado (de modo que ambos queden ligados) o rechazarle (de modo que ambos permanecerán en la misma situación en que se encontraban previamente).



4. 18. Notificación de asociación con un médico cuando se loguea el paciente

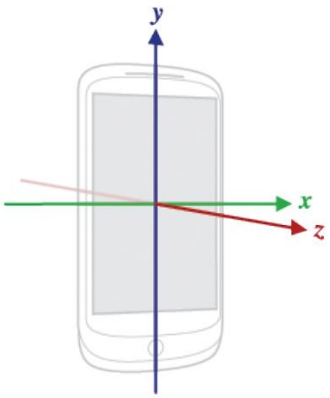
5. Análisis de señal

A lo largo de este capítulo comenzaremos a trabajar con las señales que ya podemos obtener gracias al sistema desarrollado anteriormente. Antes de nada, realizaremos la limpieza que sea necesaria para comenzar con el tratamiento de los datos y una vez hecho podremos llevar nuestras señales a Matlab, donde las procesaremos de la forma que consideremos oportuna.

5.1. Pasos previos al análisis

En primer lugar, nuestra misión consistirá en recopilar las señales, con las que vamos a trabajar, valiéndonos del sistema diseñado previamente, para después comenzar a estudiarlas en tiempo y frecuencia con el objetivo de encontrar similitudes, diferencias, patrones, anomalías…

Como ya hemos comentado anteriormente, trabajaremos con dos tipos de señales: las recogidas a través del acelerómetro y las recogidas a través del giroscopio del dispositivo móvil. Estos dos sensores nos aportarán información tridimensional (ejes X, Y, Z), cuya distribución respecto al dispositivo podemos ver en la figura 5.1.



5. 1. Distribución de los ejes X, Y, Z de los sensores respecto al dispositivo móvil.

Los acelerómetros son capaces de calcular la aceleración[15] (a), medida en m/s^2, del dispositivo móvil que los contiene. Es importante tener en cuenta que cualquier objeto, independientemente de su masa, se ve acelerado hacia la tierra por el valor de la gravedad (alrededor de 9.8 m/s^2), esta aceleración afectará al eje Z de nuestras mediciones y, por tanto, será una buena idea eliminar dicha perturbación antes de nuestros análisis.

Los giroscopios sirven para calcular la velocidad angular[16] (ω), medida en rad/s, del dispositivo móvil, alrededor de los tres mismos ejes vistos en la figura anterior.

Recordemos que también disponemos de una medida de calibración de ambos sensores previa a cada medición. De esta calibración obtendremos un valor medio que restaremos a la medida real, de modo que se eliminen posibles offset inducidos por ruido o imperfecciones en el sensor.

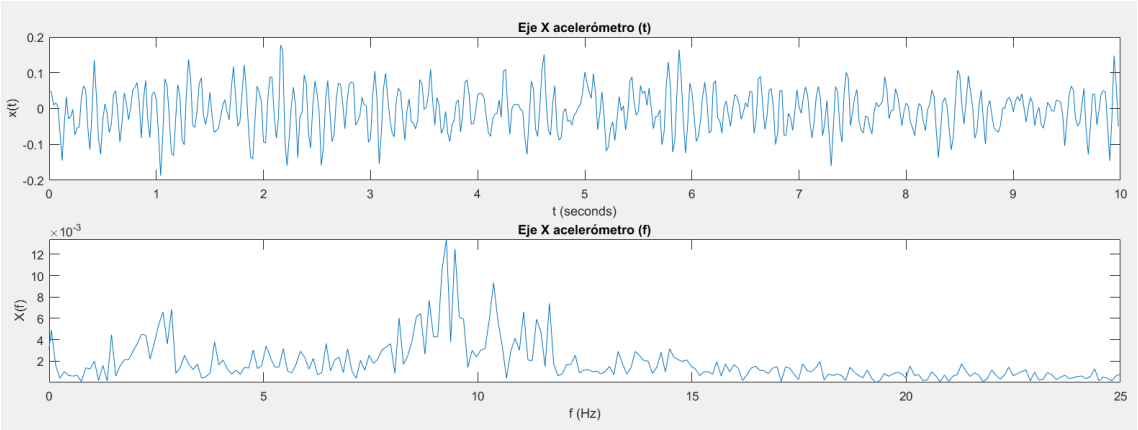
Para poder estudiar las señales correctamente necesitamos conocer dos características de la señal: su duración y la cantidad de muestras por segundo de ésta, ambos son aspectos que también almacenamos y de los que, por tanto, disponemos. Respecto a las muestras por segundo (frecuencia de muestreo) se utilizará la máxima que sea capaz de alcanzar cada uno de los sensores del dispositivo.

Antes de comenzar el análisis de las señales observamos el aspecto de éstas, de modo que sepamos con que estamos trabajando y en busca de alguna característica que pueda resultar clara a simple vista.

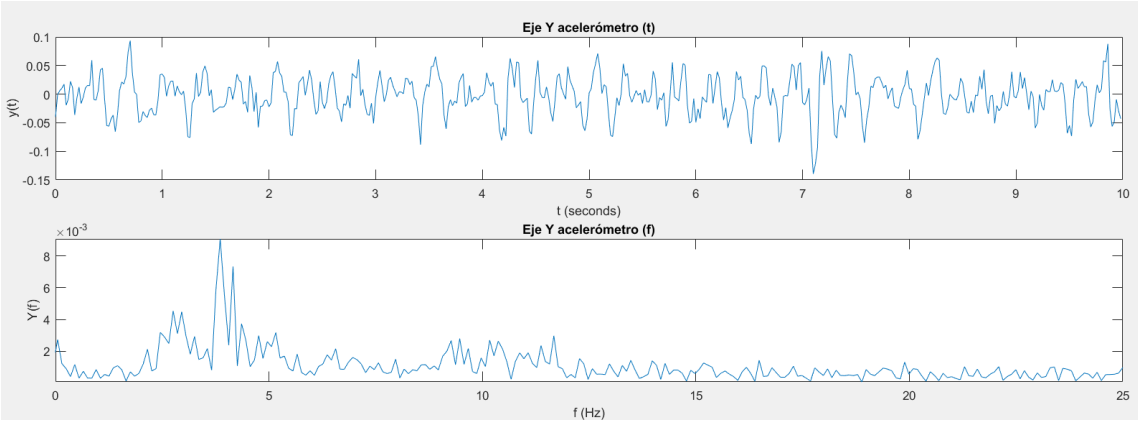
Echando un vistazo a varias de ellas podemos ver que se trata de señales oscilatorias, con una amplitud variable pero no muy elevada y apenas componentes espectrales más allá de los 20 Hz.

En las figuras de la 5.2 a 5.7, tenemos un ejemplo de señales del acelerómetro y del giroscopio para cada uno de los tres ejes espaciales, así como la representación de su espectro. Más concretamente, la medida corresponde al temblor fisiológico de un sujeto con el codo apoyado sobre una superficie plana, y de ella se ha eliminado el efecto de la gravedad y restado la media de la señal de calibración.

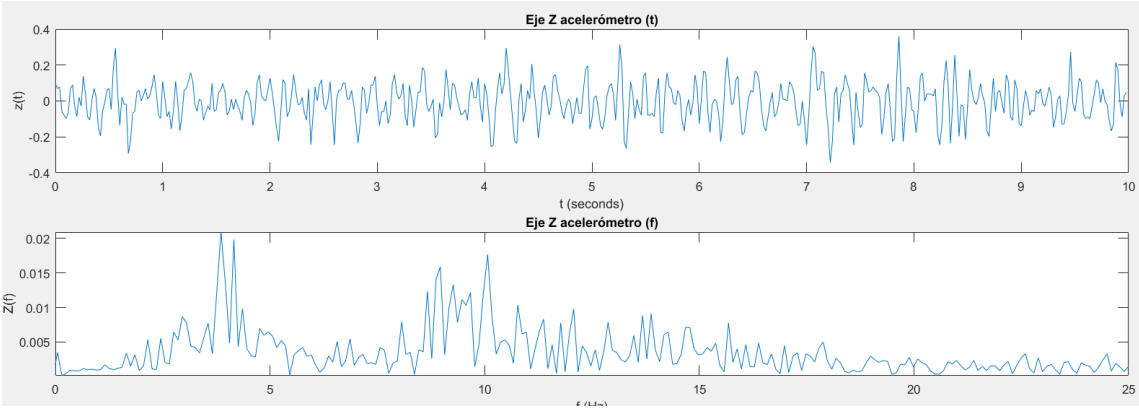
[15] Aceleración: magnitud que mide la variación que sufre la velocidad por unidad de tiempo. Se obtiene como la derivada de la velocidad con respecto al tiempo.
[16] Velocidad angular: magnitud que mide la velocidad de rotación de un objeto. Se obtiene como el ángulo girado por unidad de tiempo.



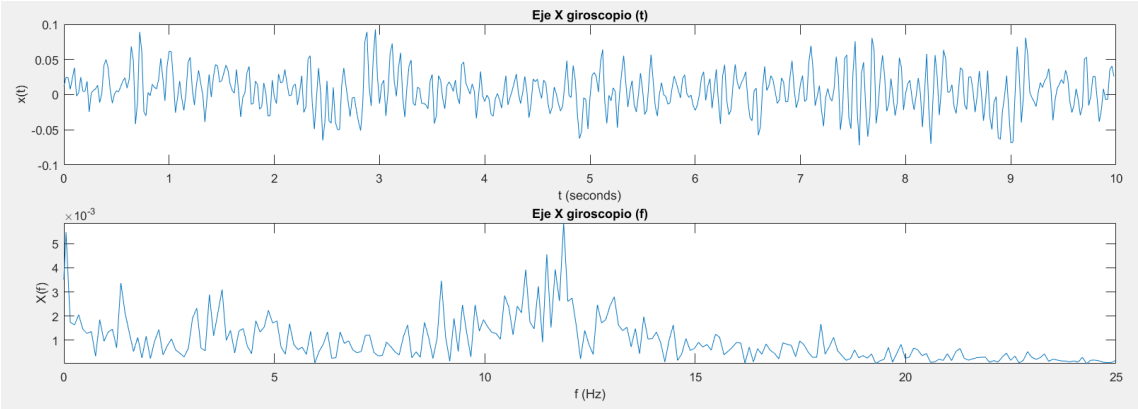
5. 2. Ejemplo de señal recogida del eje X del acelerómetro en el dominio temporal y frecuencial.



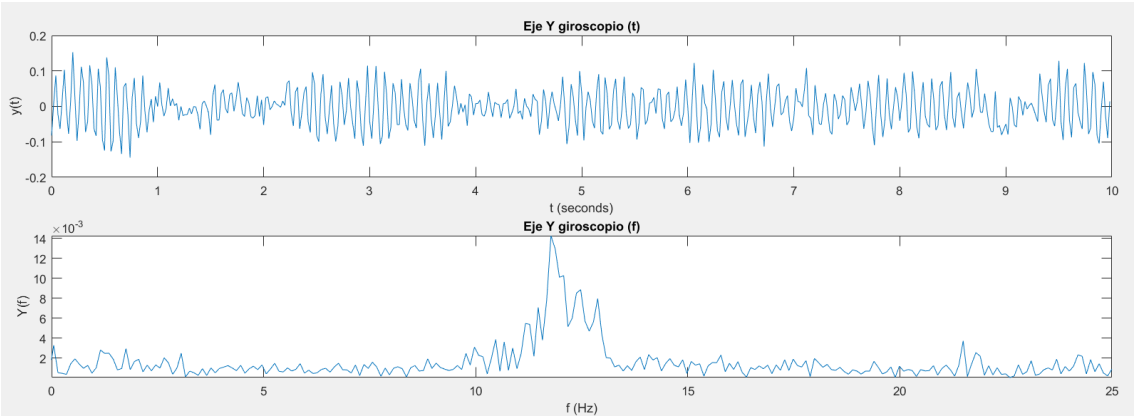
5. 3. Ejemplo de señal recogida del eje Y del acelerómetro en el dominio temporal y frecuencial.



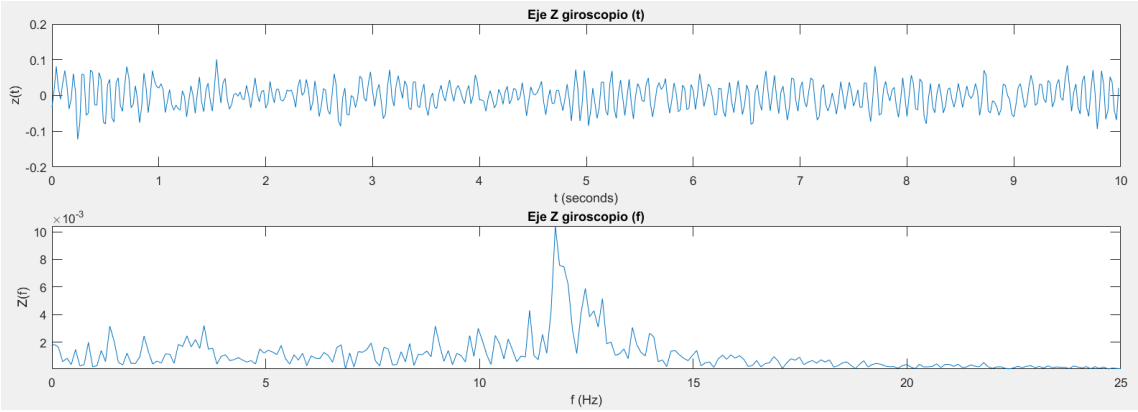
5. 4. Ejemplo de señal recogida del eje Z del acelerómetro en el dominio temporal y frecuencial.



5. 5. Ejemplo de señal recogida del eje X del giroscopio en el dominio temporal y frecuencial.



5. 6. Ejemplo de señal recogida del eje Y del giroscopio en el dominio temporal y frecuencial.



5. 7. Ejemplo de señal recogida del eje Z del giroscopio en el dominio temporal y frecuencial.

5.2. Elección de análisis

De entre los múltiples análisis que podríamos realizar sobre el conjunto de las señales recopiladas nos encargaremos de obtener los siguientes parámetros, que se pueden calcular fácilmente y nos ofrecerán una idea preliminar sobre la naturaleza de las señales.

- **Media**: con este valor obtenemos una aproximación sobre el valor esperado de la señal. Calcularemos este parámetro según la fórmula siguiente:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

dónde N es el número de muestras de la señal y x_i el valor de cada una de esas muestras.

También, cuando tratamos con señales que tienen tanto valores positivos como negativos, podemos hacernos una idea de la amplitud de la señal calculando la media sobre el valor absoluto de sus muestras, del siguiente modo:

$$\bar{x}_+ = \frac{1}{N}\sum_{i=1}^{N} |x_i|$$

- **Varianza**: con el cálculo de la varianza obtenemos una medida de la dispersión de la señal definida como el cuadrado de la desviación con respecto a la media. A partir de la varianza también podemos obtener la desviación típica, calculada como la raíz cuadrada de ésta. Calcularemos la varianza según la fórmula siguiente:

$$\sigma_N^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

dónde N es el número de muestras de la señal, $\bar{x}$ es su media y x_i el valor de cada una de esas muestras.

- **Potencia media**: a partir de las muestras obtenidas podremos calcular la potencia de la señal con la siguiente fórmula:

$$P = \lim_{N\to\infty} \frac{1}{N}\sum_{n=0}^{N} |x_i|^2$$

- **Frecuencia dominante**: al trasladarnos al dominio de la frecuencia podemos hallar nuevas características de la señal. Para esta tarea, nos valdremos de la transformada de Fourier, con la que podremos representar nuestras señales como una suma de senos y cosenos, gracias a la cual podremos ver a qué frecuencia se obtiene el valor máximo, y que tomaremos como frecuencia dominante. Aunque la señal original del temblor fuese una señal analógica, las señales con las que trabajaremos habrán sido muestreadas por los sensores de los dispositivos móviles y por tanto serán señales discretas. Podemos obtener la transformada de Fourier de una señal discreta con la siguiente fórmula:

$$X(\Omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n}$$

dónde Ω es la frecuencia discreta y n el índice de cada una de las muestras de la señal.

Sin embargo, en realidad trabajaremos con la DFT (Transformada Discreta de Fourier) de la señal, que consiste en una discretización del periodo fundamental de su transformada de Fourier.
También comprobaremos si el espectro de las señales varía con el tiempo valiéndonos de la transformada de Fourier de tiempo corto (STFT), gracias a esta herramienta dividiremos la señal en segmentos más cortos y obtendremos la transformada de Fourier de éstos. El resultado será un espectrograma en el que se relacionan el dominio temporal y espectral.

5.3. Resultados

A continuación, calcularemos los parámetros que nos hemos propuesto en busca de algo que diferencie o relacione las señales a nuestra disposición y los compararemos en función de la edad de los sujetos que han realizado la medida.

5.3.1. Conjunto de datos

Para realizar los estudios se realizaron 200 medidas de temblores pertenecientes a cerca de 50 personas voluntarias, sin ninguna patología diagnosticada, de entre 18 y 55 años. De estas medidas la mayor parte se realizó con el codo del sujeto apoyado.

A la hora de interpretar los resultados hay que tener en cuenta que el número de las muestras disponibles es limitado y, por tanto, ninguno de los resultados es totalmente

concluyente, además el número de muestras tampoco es del mismo tamaño para los distintos grupos de edad, siendo este superior en la franja de entre los veinte y treinta años (figura 5.8).

	Cualquier edad	**Edades entre 18 y 25 años**	**Edades entre 26 y 35 años**	**Edades entre 36 y 45 años**	**Edades entre 46 y 55 años**
Número total de medidas	212	136	20	44	12
Medidas realizas con el codo apoyado	142	72	14	44	12
Medidas realizadas con el brazo extendido	35	32	3	-	-
Medidas realizadas con la mano apoyada	35	32	3	-	-

5. 8. Número de medidas realizadas para cada protocolo y edades de las personas que las realizaron.

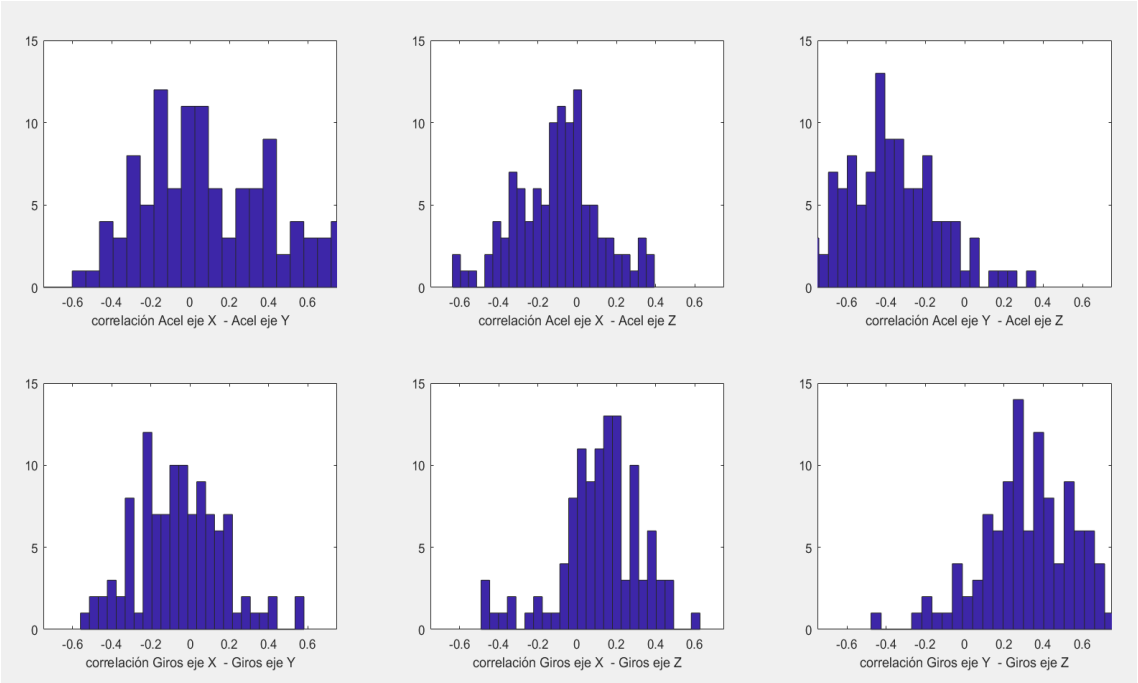
Estudiaremos por separado las señales obtenidas siguiendo distintos protocolos de los disponibles en la aplicación. Recordemos que éstos consistían en: realizar la medición del temblor de la mano con el codo apoyado sobre una superficie inmóvil, con la mano apoyada sobre una superficie inmóvil o con el brazo extendido en el aire. De todos ellos nos centraremos más especialmente en las realizadas con el codo del sujeto apoyado, puesto que es del que hemos logrado reunir un mayor número de muestras.

5.3.2. Estudio de la correlación entre señales

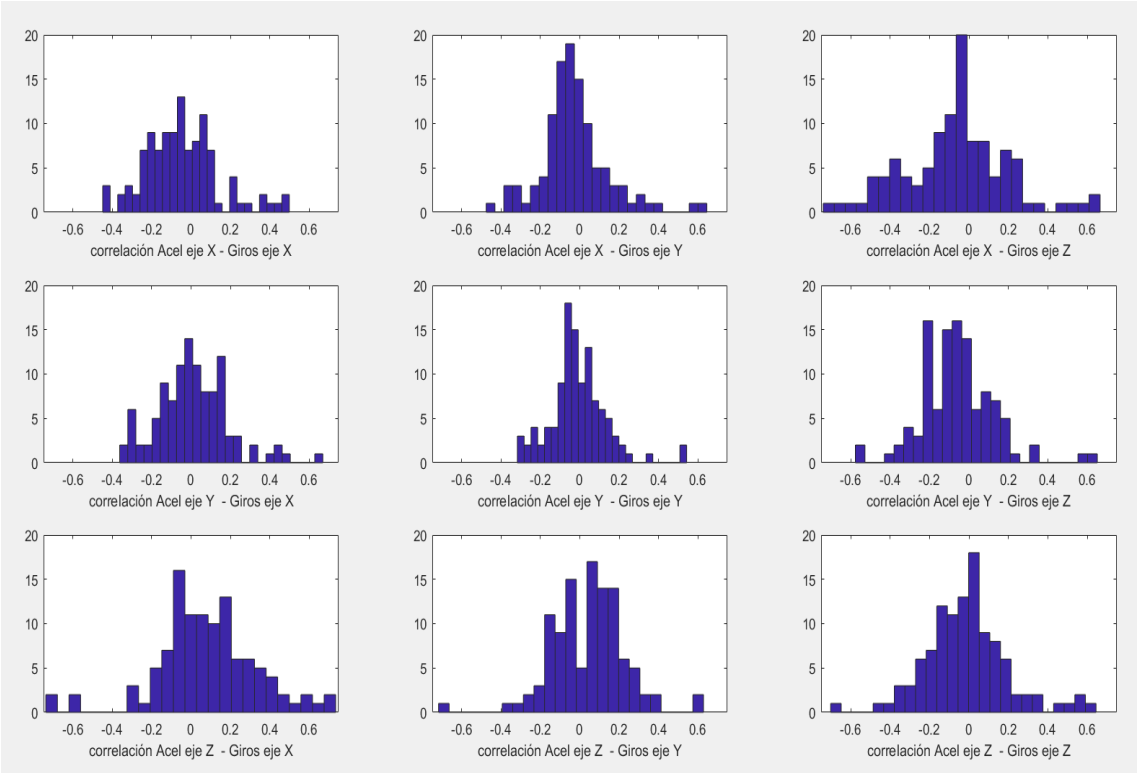
Disponemos de medidas de dos sensores con tres ejes en cada uno, por lo tanto, tenemos un total de seis señales, para cada medición, que podemos estudiar. Al provenir todas estas señales de un la mano de un mismo individuo y ser muestreadas al mismo tiempo es posible que estén muy correladas entre sí, de modo que no tendría mucho sentido estudiarlas por separado.

Una vez evaluadas las correlaciones entre cada par de señales, tomamos la decisión de estudiar cada una de ellas de forma independiente al resto. Esta decisión se basa en que, aun tratándose de señales de características similares y obtenidas de la misma fuente, al estudiar las correlaciones entre todas ellas obtenemos que en la mayor parte de los casos tenemos un factor de correlación inferior al 0.4 para la relación entre señales de un mismo sensor (figura 5.9). Este factor se reduce aún más, inferior al 0.2 para la mayor parte de los casos, cuando cruzamos las señales de los distintos sensores entre sí (figura

5.10). Ante estos resultados, con unas correlaciones de estas magnitudes, sería demasiado arriesgado prescindir de alguna de las señales, o incluso llegar a considerar el uso de una medida agregada.



5. 9. Correlaciones obtenidas entre las señales de los tres ejes de un mismo sensor: acelerómetro (primera fila) y giroscopio (segunda fila).

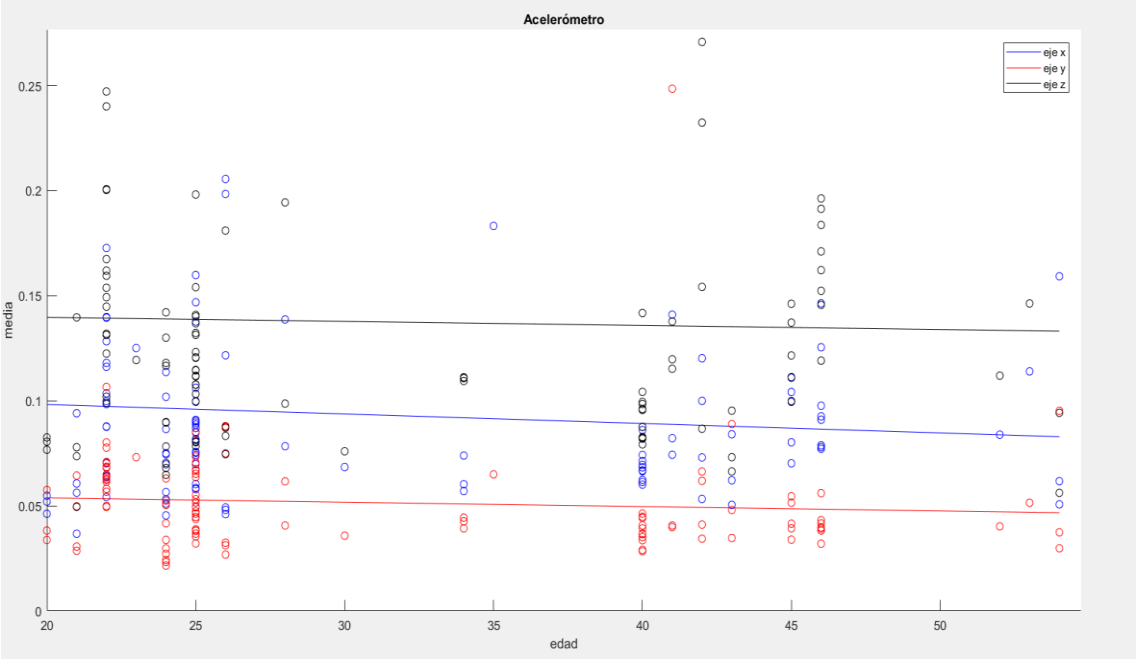


5. 10. Correlaciones obtenidas entre las señales de los ejes de sensores distintos.

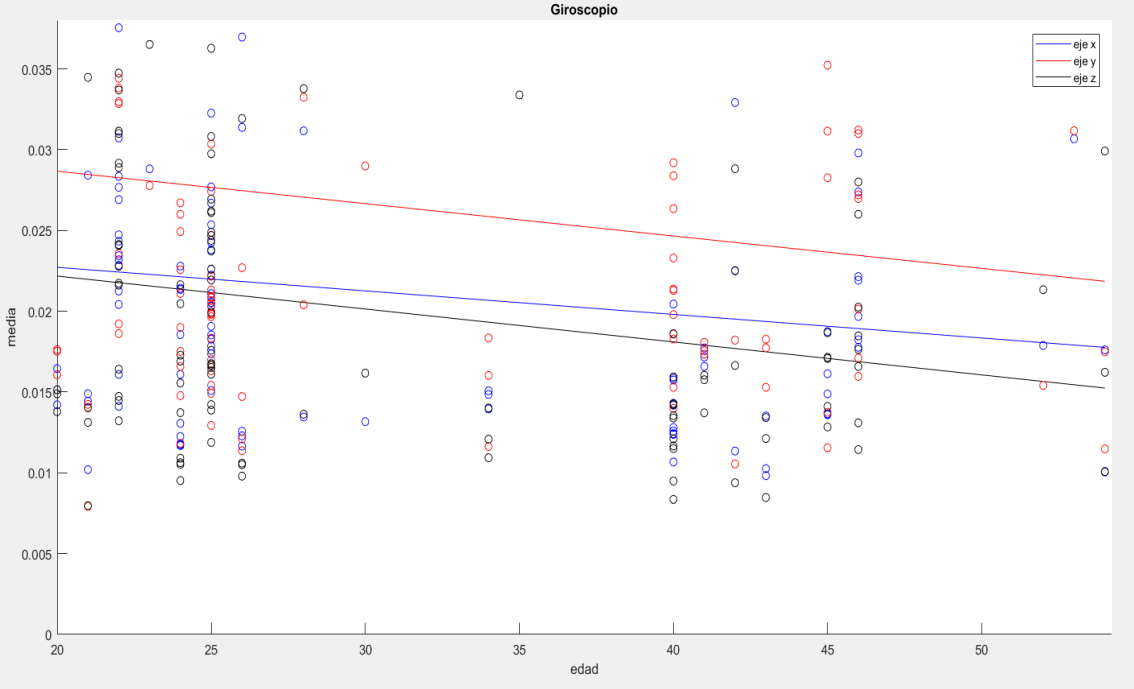
5.3.3. Estudio de medidas realizadas con el codo apoyado

En primer lugar, y en lo que respecta a las medias de las señales, podemos considerarla muy cercana a cero, ya que se trata de señales, con valores positivos y negativos, que oscilan alrededor de este valor. Esto será así siempre y cuando eliminemos correctamente el efecto de la gravedad sobre la señal, así como el posible error introducido por los sensores, de lo cual nos encargaremos con la calibración previa a cada medición.

Podemos, sin embargo, calcular la media del valor absoluto de la señal, de esta manera obtendremos un valor positivo que se corresponderá con, aproximadamente, el que esperamos que valga la señal en función de la edad de los sujetos de pruebas, tanto para las señales recogidas de los acelerómetros (figura 5.11) como de los giroscopios (figura 5.12).



5. 11. Para medidas realizas con el codo apoyado, medias calculadas para el valor absoluto de las señales obtenidas del acelerómetro, y la recta que aproxima dichos valores, respecto a la edad de los sujetos.

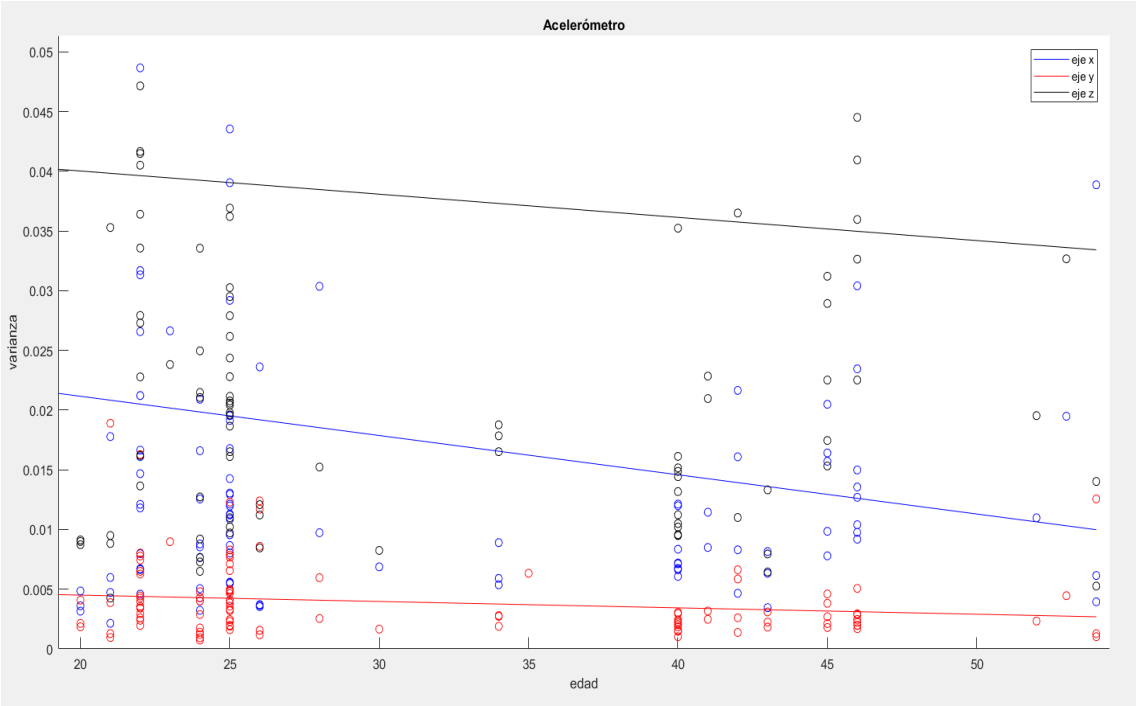


5. 12. Para medidas realizas con el codo apoyado, medias calculadas para el valor absoluto de las señales obtenidas del giroscopio, y la recta que aproxima dichos valores, respecto a la edad de los sujetos.

Obtenemos, independientemente de la edad, una aceleración media aproximada de 0.15 m/s^2 para el caso del eje Z, de 0.1 m/s^2 para el eje X y de 0.05 m/s^2 para el eje Y; los cuales se corresponden con desplazamientos verticales, laterales y frontales, respectivamente.

Para el caso del giroscopio obtenemos una velocidad angular que desciende con la edad y que vale entre 0.029 y 0.024 rad/s en el eje Y, entre 0.023 y 0.02 rad/s en el eje X y entre 0.022 y 0.018 rad/s en el eje Z; los cuales se corresponde con giros hacia los laterales, frontales y en el plano horizontal, respectivamente.

Ahora, calcularemos la varianza de todas las señales recogidas, con el fin de observar entre que valores oscilan las mismas, estos datos, junto con los obtenidos de las medias anteriores, nos servirán para obtener una idea de por qué, por norma general, se obtienen valores más altos para algunos de los ejes en concreto.

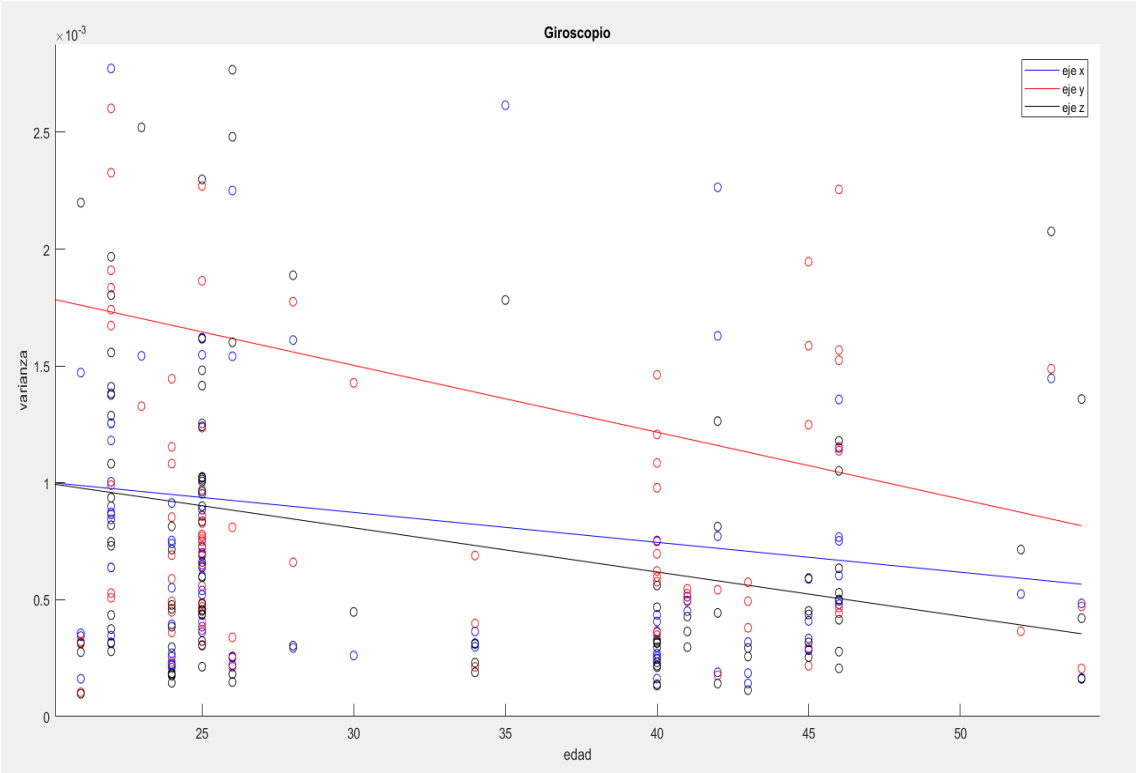


5. 13. Para medidas realizadas con el codo apoyado, varianzas calculadas para cada una de las señales obtenidas del acelerómetro y la recta que aproxima dichos valores respecto a la edad de los sujetos.

Para el caso del acelerómetro (figura 5.13) obtenemos, de media, varianzas cercanas a 0.04 (lo que implica una desviación típica de ± 0.2 m/s^2), para su eje Z; cercanas a 0.015 (desviación típica de ± 0.1225 m/s^2), para su eje X; y cercanas a 0.0045 (desviación típica de ± 0.067 m/s^2), para su eje Y. Estos datos tienen sentido respecto a los resultados que obteníamos para el cálculo de la media del valor absoluto de la señal, pues ésta se encuentra dentro de los rangos cubiertos por las desviaciones típicas.

Se halla por tanto una alta superioridad en la varianza obtenida para el eje Z (alrededor del doble que para el siguiente) puesto que la amplitud del temblor en él es dominante para la mayor parte de los casos. Este hecho se explica por la propia naturaleza del experimento: al tener la mano plana la mayor amplitud del temblor de la misma se corresponde con el movimiento de oscilación hacia arriba y abajo (eje Z), por encima de los movimientos horizontales, que suponen además realizar movimiento con el brazo, y que, en este caso, se encuentra apoyado.

Por otra parte, encontramos muy pocas diferencias en la varianza calculada en función de la edad de los sujetos, apareciendo un ligero descenso para los ejes Z y X, cuando ésta aumenta (aunque también es en las edades más elevadas donde disponemos de menor cantidad de muestras).



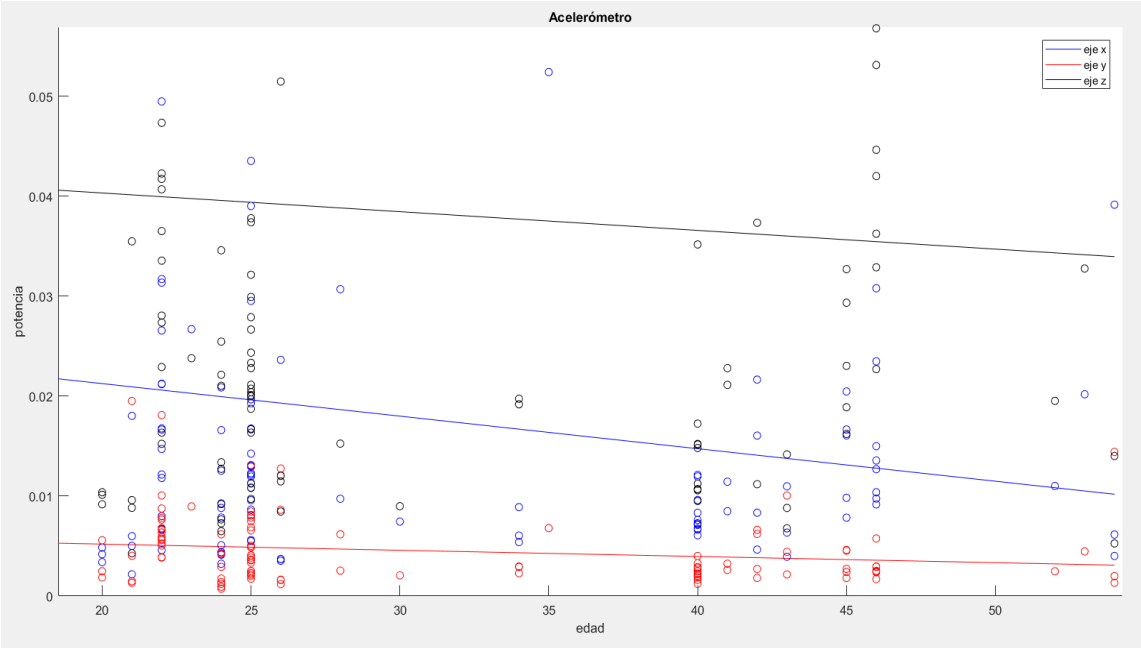
5. 14. Para medidas realizadas con el codo apoyado, varianzas calculadas para cada una de las señales obtenidas del giroscopio y la recta que aproxima dichos valores respecto a la edad de los sujetos.

Para el caso del giroscopio (figura 5.14) obtenemos, de media, varianzas cercanas a 0.0015 (lo que implica una desviación típica de ± 0.0387 rad/s), para su eje Y; cercanas a 0.00085 (desviación típica de ± 0.0292 rad/s), para su eje X; y cercanas a 0.00065 (desviación típica de ± 0.0255 rad/s), para su eje Z. De nuevo, los valores de la desviación típica engloban las medias de la señal cuando tomamos su valor absoluto y que calculábamos al principio de la sección.

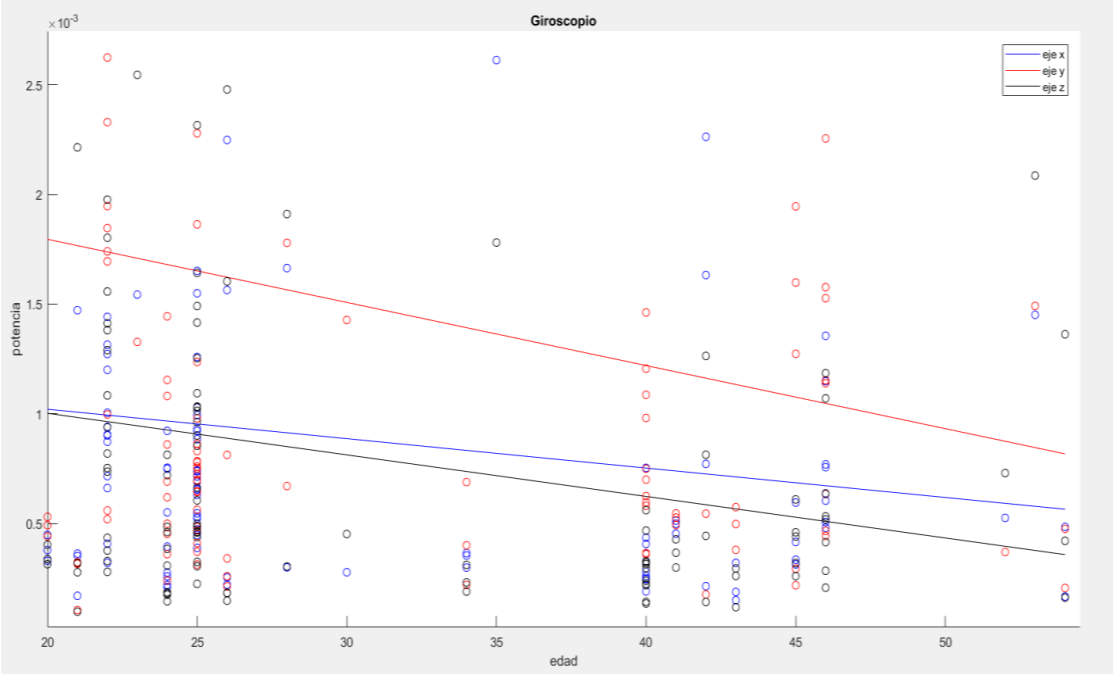
Se halla en este caso una alta superioridad en la varianza obtenida para el eje Y (alrededor del doble que los otros dos ejes) puesto que la amplitud del temblor en él es dominante para la mayor parte de los casos. La explicación para esta situación viene de la mano de lo expuesto para el caso del acelerómetro, aquí el movimiento de giro más sencillo corresponde con el realizado hacia ambos lados de la mano (eje Y), ya que no supone apenas movimiento del brazo (que se encuentra fijo).

De nuevo, la evolución de la varianza con respecto a la edad de los sujetos, muestra un descenso, en este caso para los tres ejes, en función del ascenso de ésta.

A continuación, calcularemos la potencia media de las señales de las que disponemos. Al tratarse de señales con un media similar y cercana a cero, y debido la forma de calcular la potencia descrita en la sección 5.2, los valores que obtendremos serán prácticamente los mismos que la varianza para todas ellas.



5. 15. Para medidas realizadas con el codo apoyado, potencias calculadas para cada una de las señales obtenidas del acelerómetro y la recta que aproxima dichos valores respecto a la edad de los sujetos.

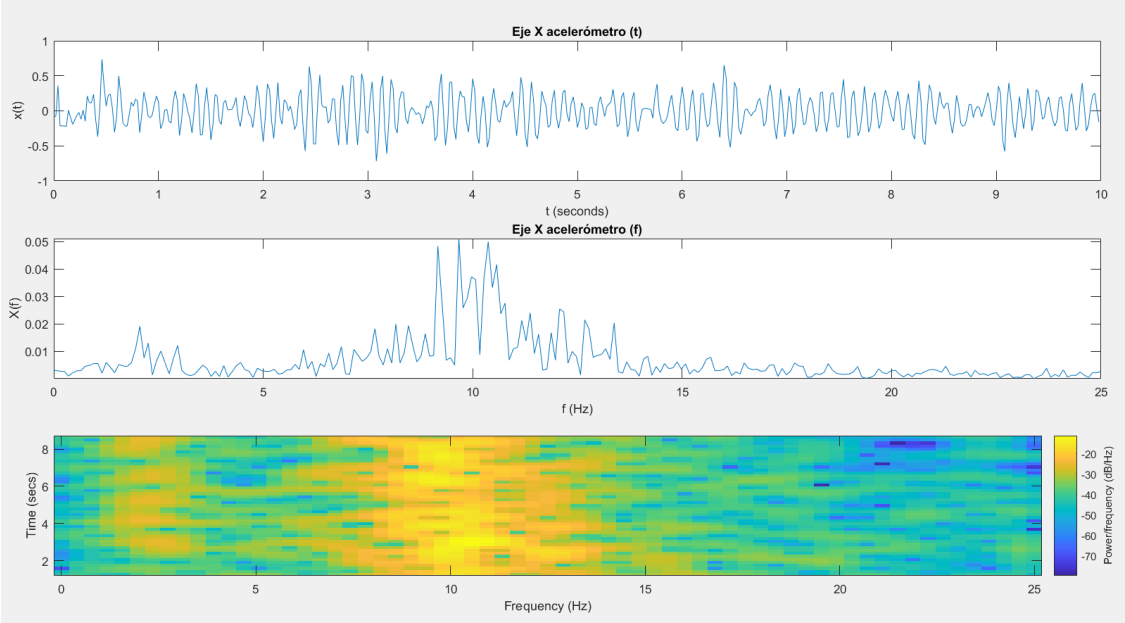


5. 16. Para medidas realizadas con el codo apoyado, potencias calculadas para cada una de las señales obtenidas del giroscopio y la recta que aproxima dichos valores respecto a la edad de los sujetos.

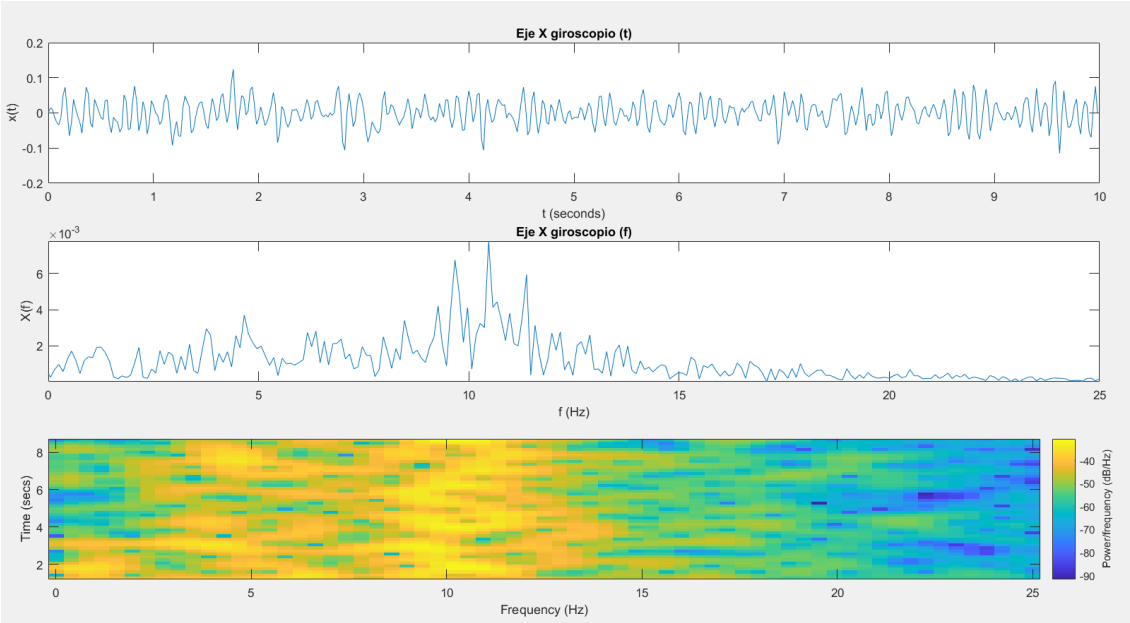
Como esperábamos, tanto para el caso del acelerómetro (figura 5.15) como para el del giroscopio (figura 5.16), obtenemos una potencia que se aproxima mucho a la varianza calculada anteriormente, con una superioridad de los mismos ejes que en el caso anterior.

Por último, pasamos a estudiar la señal en el dominio de la frecuencia. En primer lugar, obtenemos los espectrogramas de las señales para comprobar si se trata de señales

estacionarias, es decir, si mantienen un espectro similar a lo largo de la duración de la medición.



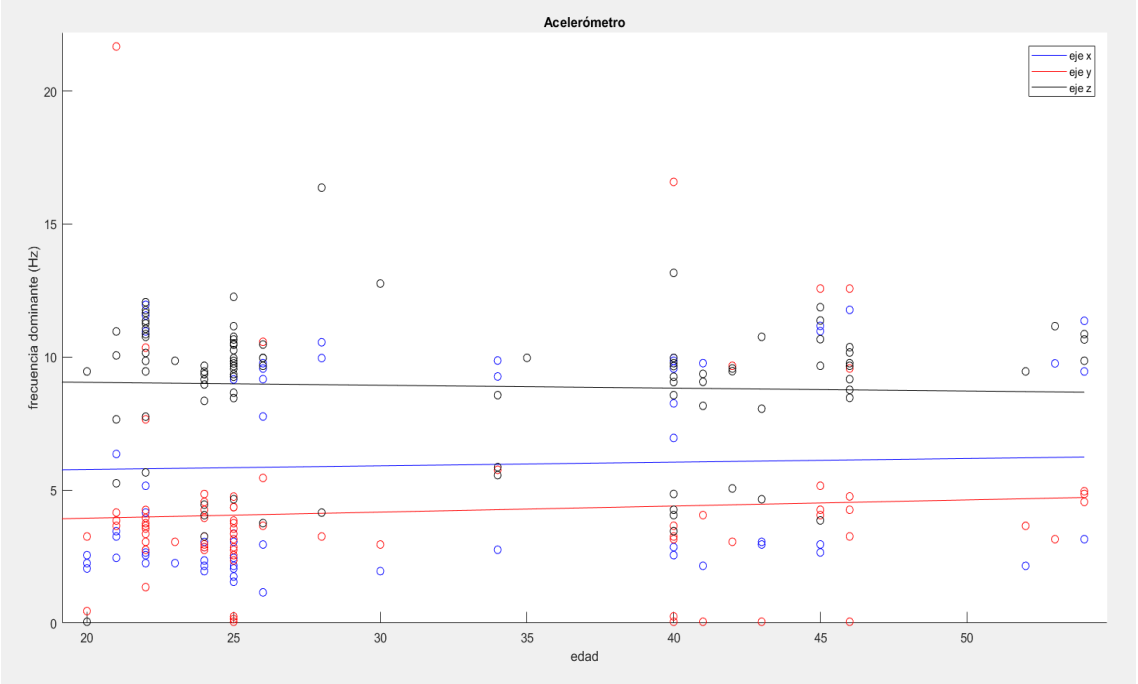
5. 17. Ejemplo de señal obtenida del eje X del acelerómetro en el dominio del tiempo y de la frecuencia y su espectrograma, para una medida realizada con el codo apoyado.



5. 18. Ejemplo de señal obtenida del eje X del giroscopio en el dominio del tiempo y de la frecuencia y su espectrograma, para una medida realizada con el codo apoyado.

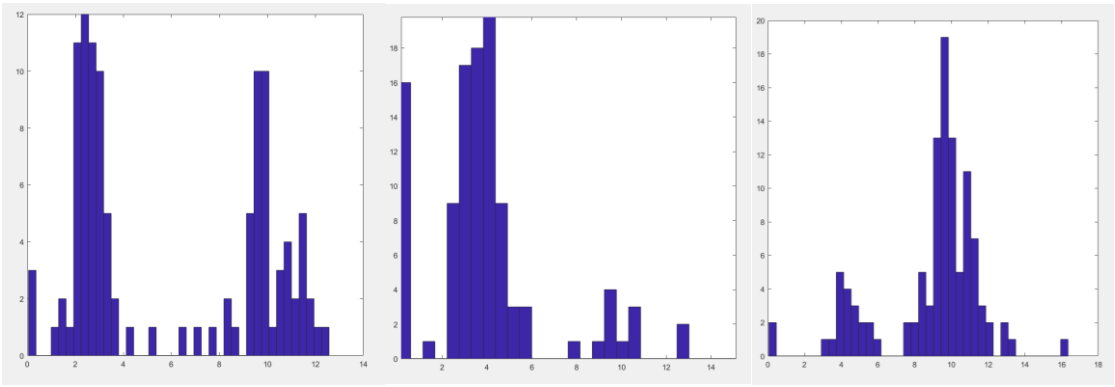
A la vista de los espectrogramas de unas cuantas muestras (ejemplos en las figuras 5.17 y 5.18), observamos que, aunque las frecuencias con mayores potencias puedan cambiar de unas a otras, éstas se mantienen bastante constantes a lo largo de la duración de la señal.

A continuación, buscaremos la frecuencia dominante para el temblor en cualquiera de los sensores y ejes a partir de las transformadas de Fourier calculadas.



5. 19. Para medidas realizadas con el codo apoyado, frecuencias dominantes calculadas para cada una de las señales obtenidas del acelerómetro y la recta que aproxima dichos valores respecto a la edad de los sujetos.

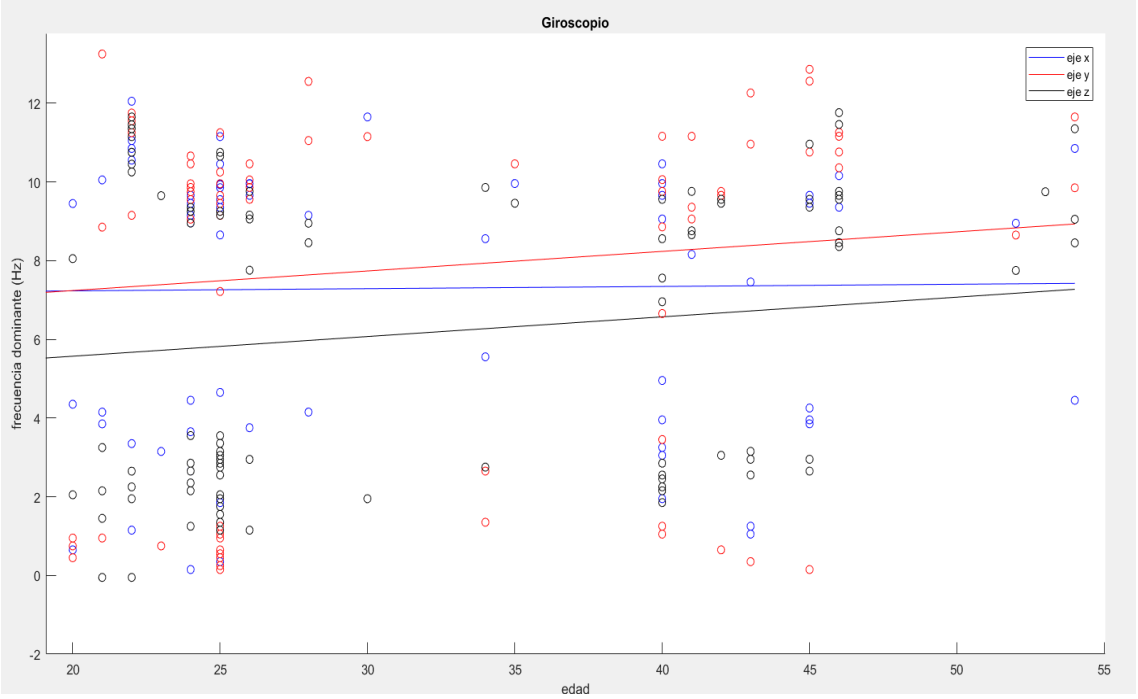
Para el caso del acelerómetro (figura 5.19) nos encontramos con frecuencias dominantes que se encuentran entre dos zonas: unas en el rango de los 5.5 – 2 Hz y otras entre los 13 – 7.5 Hz. Si calculamos una aproximación por ejes, obtenemos una frecuencia dominante superior para el caso del eje Z (unos 9 Hz) a las de los ejes X (unos 6 Hz) e Y (unos 4 Hz), independientemente de la edad de los sujetos.



5. 20. Para medidas realizadas con el codo apoyado, de izquierda a derecha, histograma del número de cada una de las frecuencias dominantes calculadas para las señales obtenidas de los ejes X, Y, Z, respectivamente, del acelerómetro.

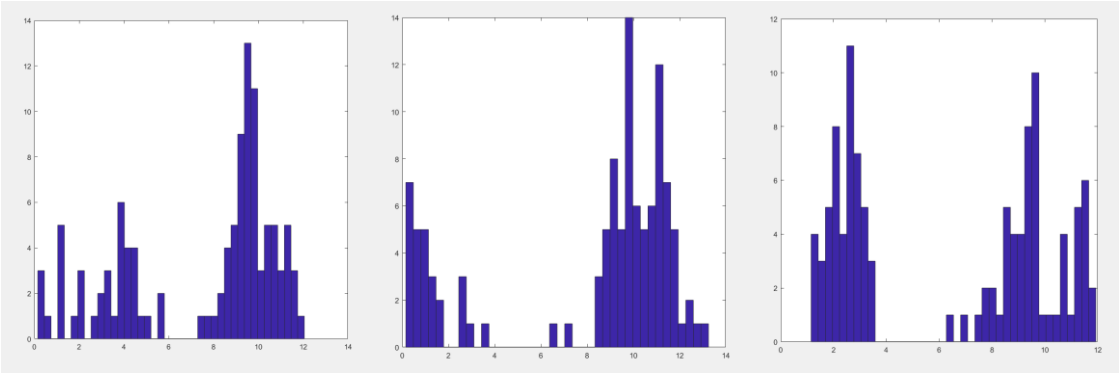
Si profundizamos un poco más en la distribución de las frecuencias y calculamos los histogramas para el total de muestras disponible (figura 5.20) obtenemos que para el eje X el número de muestras en ambas zonas es muy similar y predominante en 2.5 Hz para

la franja inferior y en 10 Hz para la superior. Para el eje Y encontramos un número superior de muestras en la franja inferior, cerca de los 4 Hz; al contrario que para el eje Z, donde encontramos un número superior de muestras en la franja superior, cerca de los 10 Hz.



5. 21. Para medidas realizadas con el codo apoyado, frecuencias dominantes calculadas para cada una de las señales obtenidas del giroscopio y la recta que aproxima dichos valores respecto a la edad de los sujetos.

Para el caso del giroscopio (figura 5.21), de nuevo nos encontramos con frecuencias dominantes que se encuentran en dos zonas: unas en el rango de los 12 – 8 Hz y otras entre los 5 – 1 Hz. Si calculamos una aproximación por ejes, obtenemos una frecuencia dominante superior para el caso del eje Y (unos 8 Hz) a las de los ejes X (unos 7 Hz) y Z (unos 6 Hz), con un ligero aumento en función de la edad de los sujetos.



5. 22. Para medidas realizadas con el codo apoyado, de izquierda a derecha, histograma del número de cada una de las frecuencias dominantes calculadas para las señales obtenidas de los ejes X, Y, Z, respectivamente, del giroscopio.

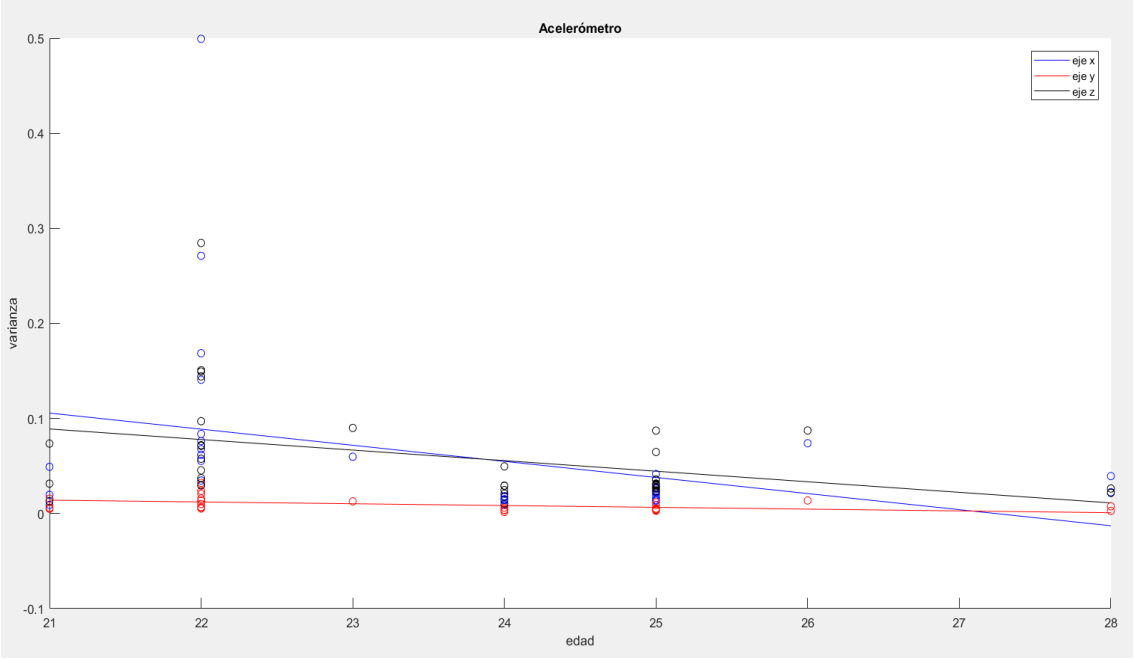
Al mostrar ahora los histogramas con el número de frecuencias obtenidas en cada zona (figura 5.22) podemos ver como para los ejes X e Y obtenemos predominancias en la franja superior, cercanas a los 9 y 10 Hz, respectivamente. Por otra parte, para el eje Z tenemos un número similar de muestras en ambas zonas y centradas en los 3 y 9 Hz, para cada una de ellas.

A la vista de estos resultados en ambos sensores, se observa que, para esta situación, los ejes que en el dominio temporal presentaban una mayor amplitud, son también los que, de media, poseen frecuencias dominantes más altas (más ciclos por segundo).

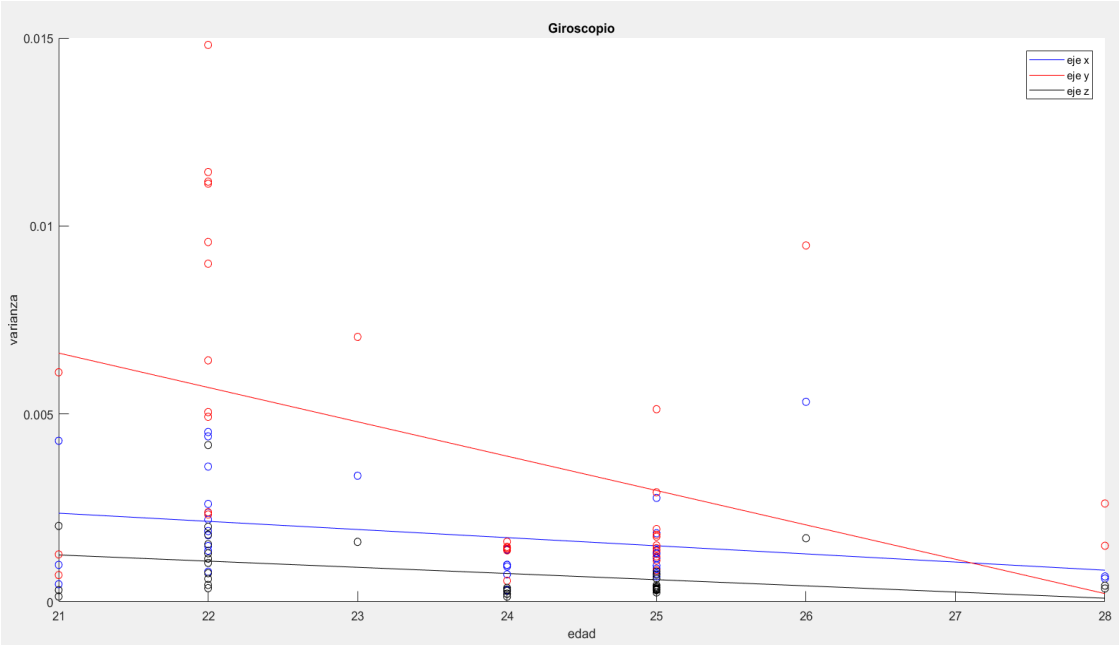
5.3.4. Comparativa con medidas realizadas con otros protocolos

En este apartado, estudiaremos las medidas realizadas con el brazo extendido y con la mano apoyada sobre una superficie plana. Para ambos casos, el número de muestras recogidas es menor que del que disponíamos para el caso de las medidas realizadas con el codo apoyado, pero nos servirán para hacernos una idea aproximada.

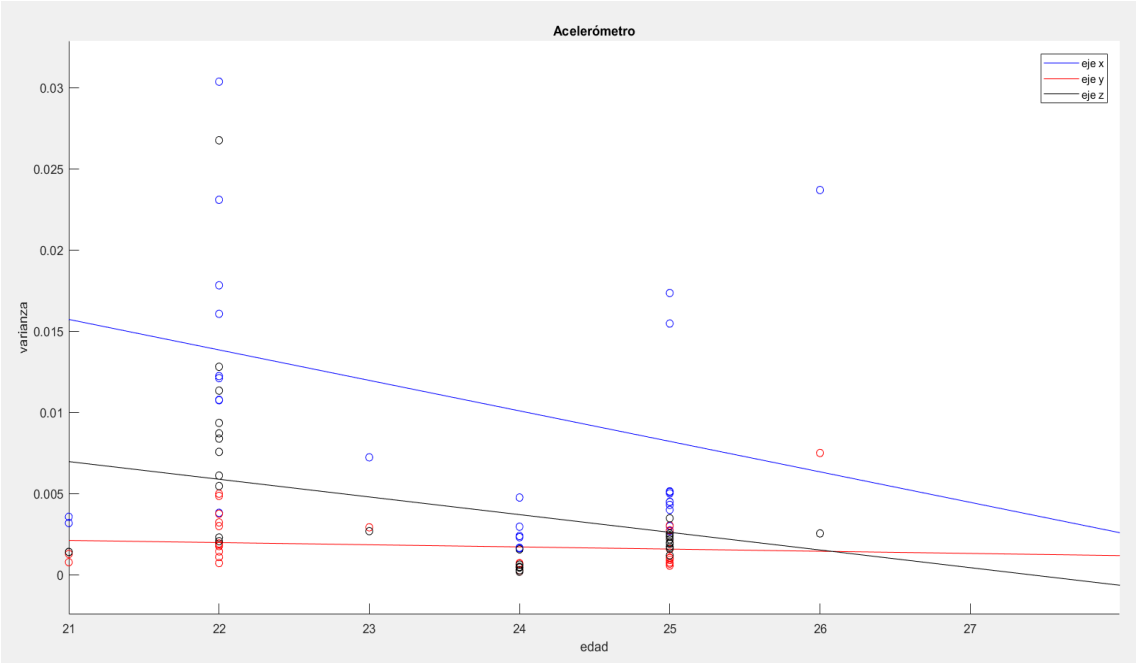
De nuevo empezaremos calculando la varianza de las señales para ambos sensores, en primer lugar, para medidas realizadas con el brazo extendido y, a continuación, para medidas realizadas con la mano apoyada.



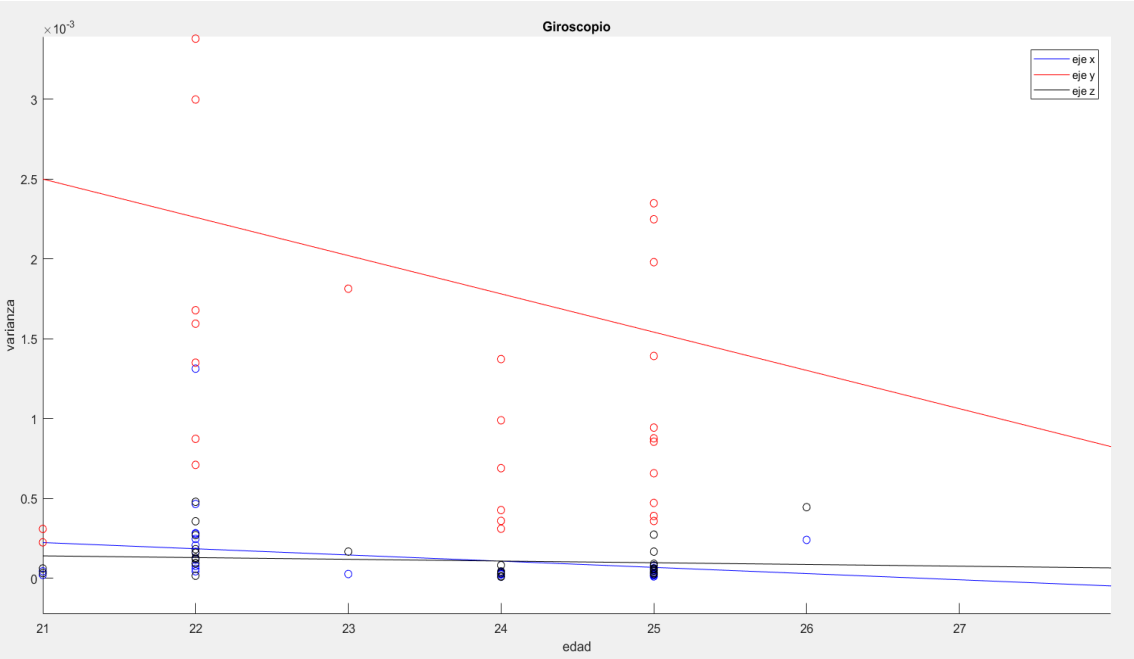
5. 23. Para medidas realizadas con el brazo extendido, varianzas calculadas para cada una de las señales obtenidas del acelerómetro y la recta que aproxima dichos valores respecto a la edad de los sujetos.



5. 24. Para medidas realizadas con el brazo extendido, varianzas calculadas para cada una de las señales obtenidas del giroscopio y la recta que aproxima dichos valores respecto a la edad de los sujetos.



5. 25. Para medidas realizadas con la mano apoyada, varianzas calculadas para cada una de las señales obtenidas del acelerómetro y la recta que aproxima dichos valores respecto a la edad de los sujetos.



5. 26. Para medidas realizadas con la mano apoyada, varianzas calculadas para cada una de las señales obtenidas del giroscopio y la recta que aproxima dichos valores respecto a la edad de los sujetos.

En el caso del acelerómetro, y para medidas realizadas con el brazo extendido (figura 5.23), observamos dos cosas: primero, una varianza en las medidas superior a la que obteníamos para el primero de los protocolos, lo cual tiene sentido puesto que ahora el brazo se encuentra sin ningún apoyo y la amplitud de la oscilación será mayor; y segundo, que la varianza de los ejes X y Z es muy similar, de nuevo esto se debe a que ahora el brazo está libre y la vibración se produce de igual modo en sentido horizontal y vertical.

Cuando tratamos con medidas realizadas con la mano apoyada (figura 5.25), en cambio, lo que obtenemos es un descenso en la varianza para el caso del eje Z, ya que la posibilidad de realizar movimiento ascendente/descendente se ve muy limitada, llegando a verse superada por la varianza del eje X (movimiento hacia los lados).

En el caso del giroscopio, y para medidas realizadas con el brazo extendido (figura 5.24), podemos ver como de nuevo la varianza es superior a la del caso en el que el codo se encontraba apoyado, pero en este caso el eje con varianza superior continúa siendo el Y, ya que el movimiento de giro sigue siendo más sencillo de realizar hacia los laterales de la mano.

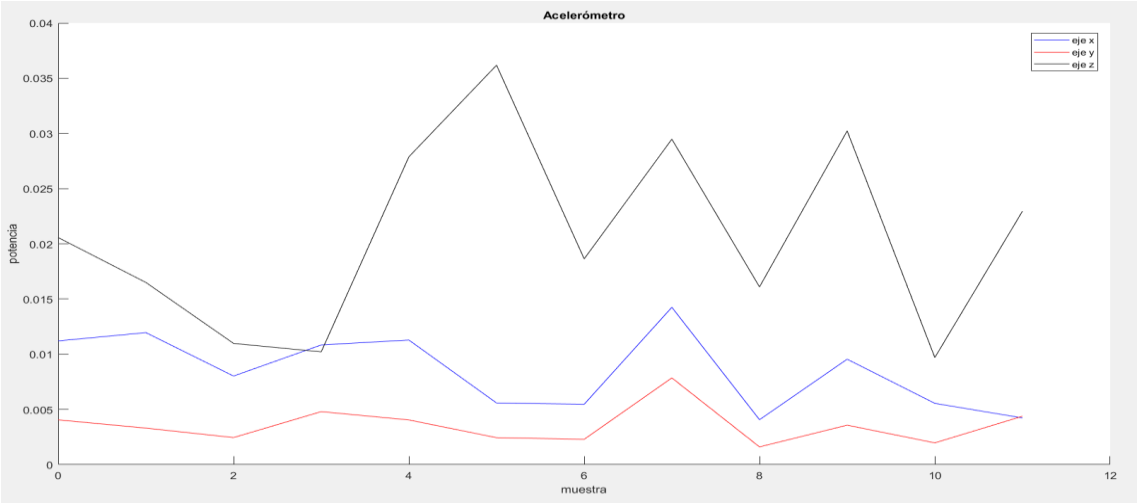
En cambio, si nos fijamos ahora en las medidas realizadas cuando la mano está en reposo (figura 5.26), nos encontramos con un ligero parecido a lo que obteníamos en el caso de las medidas realizadas con el codo apoyado, ya que en ambos casos la mano se encuentra rígida, pero de nuevo en este caso la amplitud obtenida es menor especialmente para los ejes X e Y cuyas posibilidades de giro en este caso se encuentran altamente limitadas.

5.3.5. Comparativa entre la potencia intrasujeto e intersujetos

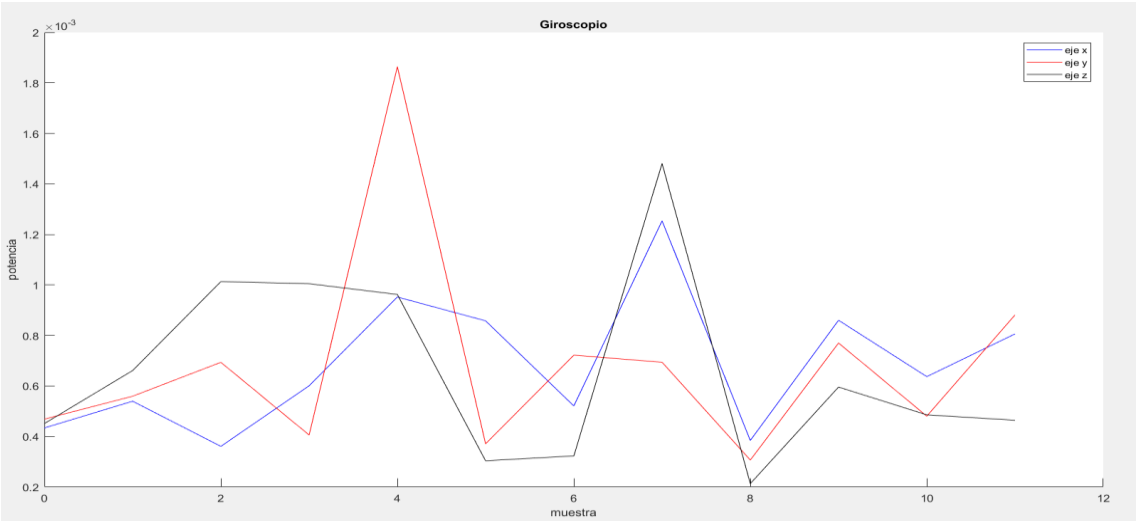
En esta ocasión, tomaremos las distintas medidas realizadas por una misma persona (registrada en la aplicación) con el codo apoyado a lo largo de tres meses y calcularemos la potencia de éstas. A partir de ahí compararemos la varianza que se obtiene para dichas potencias, con la obtenida al comparar con el conjunto total de las muestras del resto de personas que han realizado medidas en la aplicación.

Para el sujeto elegido tenemos once medidas de las cuales obtenemos su potencia (figuras 5.27 y 5.28). Si calculamos la varianza entre dichas potencias obtenemos, para el acelerómetro, una varianza de 1.1921×10^{-5} en el eje X, de 2.8747×10^{-6} en el eje Y, y de 7.5674×10^{-5} en el eje Z.

Para el caso del giroscopio se obtiene una varianza de 7.1455×10^{-8} en las potencias en el eje X, de 1.6914×10^{-7} en las potencias en el eje Y, y de 1.423×10^{-7} en las potencias en el eje Z.



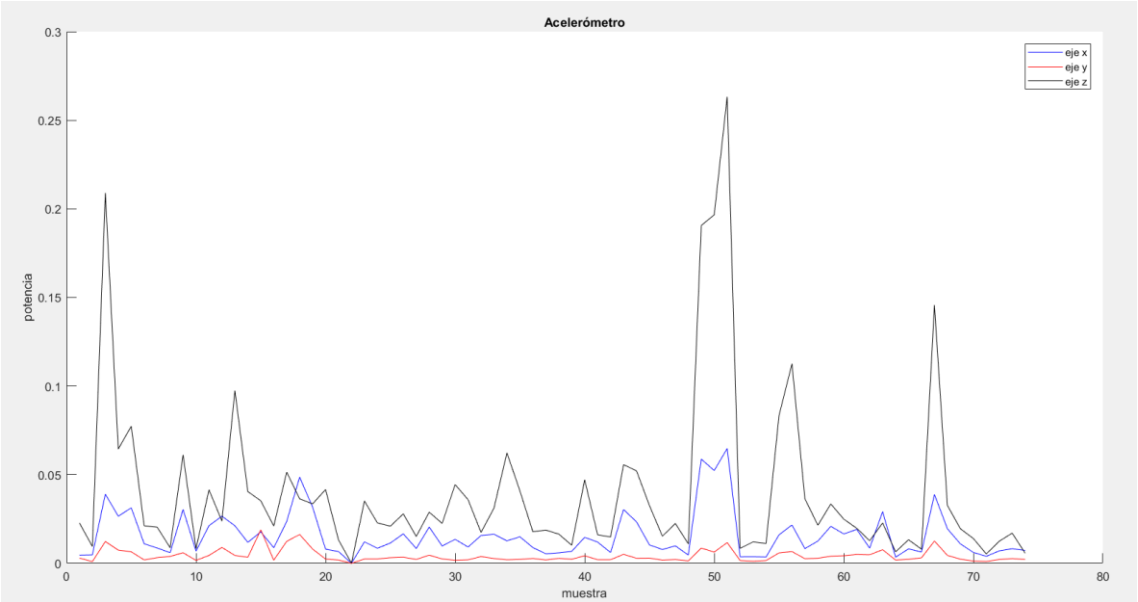
5. 27. Potencias calculadas para las distintas señales del acelerómetro de una misma persona.



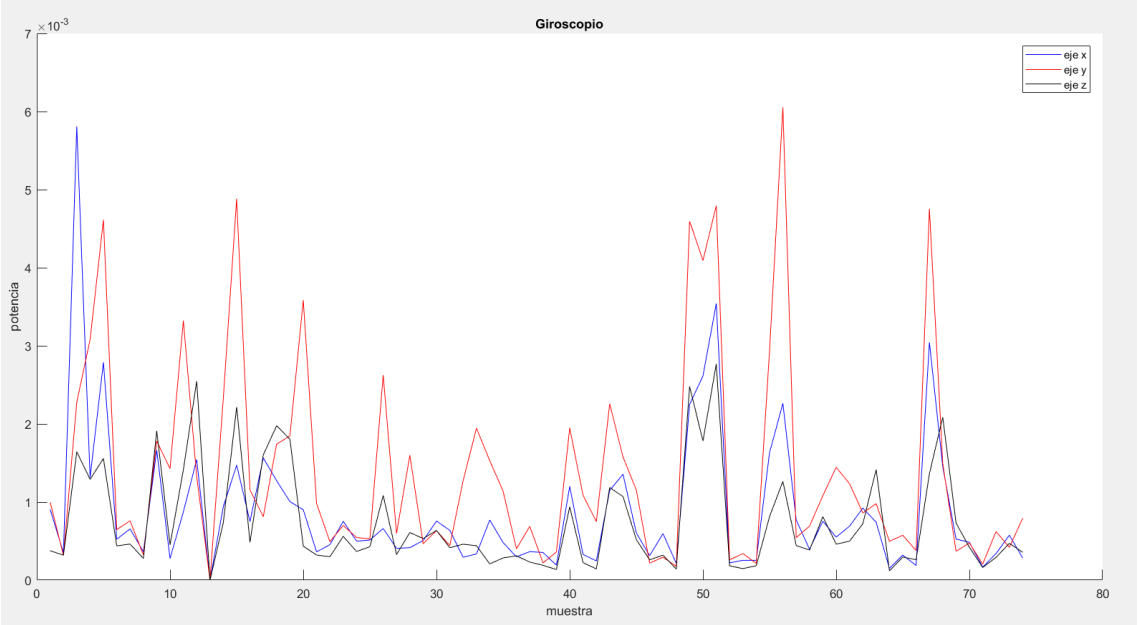
5. 28. Potencias calculadas para las distintas señales del giroscopio de una misma persona.

Repitiendo el mismo experimento para otras 75 muestras realizadas de la misma forma por personas distintas (figuras 5.29 y 5.30) se obtiene, para el caso del acelerómetro, una varianza de 1.7578×10^{-4} en el eje X, de 1×10^{-3} en el eje Y, y de 2.4×10^{-3} en el eje Z.

Para el caso del giroscopio se obtiene una varianza de 8.9013×10^{-7} en las potencias en el eje X, 3.3403×10^{-6} en las potencias en el eje Y, y de 4.459×10^{-7} en las potencias en el eje Z.



5. 29. Para medidas realizadas con el codo apoyado, potencias calculadas para las señales del acelerómetro de personas distintas.



5. 30. Para medidas realizadas con el codo apoyado, potencias calculadas para las señales del giroscopio de personas distintas.

De este modo podemos ver como al estudiar a un mismo sujeto, la varianza que se obtiene en la potencia de sus medidas es mucho menor tanto para el caso del acelerómetro (más de 15 veces) como para el del giroscopio (entre 4 y 20 veces), por lo que podemos comprobar que esta potencia tiene cierta coherencia para dicho individuo y puede llegar a caracterizarlo, al compararlo con el resto de la población (figura 5.31).

	Varianza de la potencia intrasujeto	**Varianza de la potencia intersujeto**
Acelerómetro, eje X	1.1921×10^{-5}	1.7578×10^{-4}
Acelerómetro, eje Y	2.8747×10^{-6}	1×10^{-3}
Acelerómetro, eje Z	7.5674×10^{-5}	2.4×10^{-3}
Giroscopio, eje X	7.1455×10^{-8}	8.9013×10^{-7}
Giroscopio, eje Y	1.6914×10^{-7}	3.3403×10^{-6}
Giroscopio, eje Z	1.423×10^{-7}	4.459×10^{-7}

5. 31. Varianzas en la potencia intrasujeto e intersujetos para los distintos ejes y sensores.

5.4. Discusión

Finalmente, a partir de los resultados anteriores, es posible destacar varios aspectos que pueden resultar de interés, relativos a los temblores fisiológicos presentes en cualquier persona.

- Los temblores producidos en las tres direcciones espaciales poseen correlaciones relativamente pequeñas entre sí, incluso para señales obtenidas a partir del mismo sensor.

- Para mediciones de temblor realizadas tanto con acelerómetros como con giroscopios, podemos encontrar ejes que dominan sobre los demás, cual sea este eje, depende del modo en que se realiza la medición del mismo y es independiente de la edad de la persona. Cuando se realiza la medición, el eje que menor oposición ofrece a la hora de realizar movimientos es el que tiene una mayor amplitud.

- El temblor es una señal oscilatoria con valores positivos y negativos y, debido a que su media es prácticamente cero, su varianza y su potencia media coinciden.

- Los temblores tienen características espectrales similares, con frecuencias inferiores a los 20 Hz y que se mantienen a lo largo del tiempo. Estas frecuencias se encuentran, típicamente dentro de dos franjas: una más baja entre 2 – 5 Hz y otra más alta entre los 8 – 12 Hz, aproximadamente.

- Los temblores, cuando nos fijamos en una única persona, poseen características relativamente estables si los comparamos con la población al completo. Esto significa que tiene sentido caracterizar los temblores de un cierto individuo y describirlos en comparación con los de una población de referencia.

6. Conclusiones y líneas futuras

Este capítulo recoge las conclusiones generales que se han extraído durante el desarrollo del sistema en su conjunto, así como con el posterior análisis preliminar de las señales estudiadas. Por último, se mencionan también posibles modificaciones y ampliaciones que podrían introducirse, más allá del trabajo desarrollado, con el fin de mejorar las utilidades ya disponibles o de añadir funcionalidades nuevas.

6.1. Conclusiones

El propósito principal del proyecto consistía en la creación de una herramienta para ayudar a recoger muestras del temblor de cualquier persona, de forma sencilla, a través de una aplicación móvil, de modo que éstas pudieran ser almacenadas para la realización de estudios sobre las características del mismo. Además, el sistema en mente debía conformar un primer prototipo aplicable al seguimiento, e incluso diagnóstico, de pacientes con patologías que poseen el temblor entre uno de sus síntomas.

Con el sistema desarrollado tenemos ante nosotros una herramienta con dos partes claramente diferenciadas: por un lado, la aplicación móvil que utiliza el usuario para acceder a todas las funcionalidades del sistema y de entre las que debemos destacar la captura de medidas del temblor; y, por otro lado, el servidor, que interactúa con la aplicación y se encarga de procesar y almacenar tanto las señales obtenidas como los datos de los usuarios.

Gracias a esto podemos reducir las interacciones personales en dos situaciones: entre investigador y sujeto de pruebas y entre médico y paciente. De esta forma logramos, para el primer caso, simplificar la metodología a la hora de recoger muestras para realizar un estudio acerca del tema, suponiendo una menor implicación por parte del investigador, ya que no tiene necesidad de estar presente en la realización de las medidas, eso sí, a coste de no poder vigilar en persona la realización de éstas.

Para el segundo caso, logramos un seguimiento más continuado del paciente, sin necesidad de desplazamiento ni por parte de éste ni de su médico, lo cual supone la clara ventaja de una mayor comodidad para ambos. Si este hecho pudiera combinarse con un buen conocimiento de la naturaleza de los temblores, supondría poder evaluar día a día cómo evoluciona la patología del paciente o incluso llegar a detectar de manera temprana la aparición de enfermedades con este síntoma en personas aparentemente sanas.

Respecto a los resultados obtenidos a partir del análisis de un número reducido de muestras de temblores de carácter fisiológico, se ha podido observar que esta clase de temblor posee una frecuencia de oscilación baja y acotada en dos regiones, así como que su amplitud es también bastante baja, de ahí que resulte prácticamente imperceptible al ojo humano.

Además, se ha comprobado que el modo en que se realice la medición del temblor afecta a los resultados que se obtenga del estudio de los mismos, de modo que es importante recoger estas señales siguiendo un protocolo claro y de una forma adecuada para no introducir errores en el análisis.

Por desgracia, el número de muestras recogidas y su calidad ha limitado las posibilidades de estudio de la naturaleza de los temblores y por lo tanto es recomendable seguir investigando al respecto, en busca de resultados más claros y definitivos con un número de personas mayor y más variado.

6.2. Líneas futuras

El sistema obtenido, aun cumpliendo con los objetivos planteados en inicio, puede ser ampliamente mejorado con nuevas funcionalidades. Además, la granularidad en el trabajo desarrollado permite que la inclusión de muchas de ellas se pueda realizar sin afectar al código actual, de modo que simplemente consista en la generación de nuevas versiones que sean distribuidas a los usuarios de forma casi transparente a éstos.

Entre los progresos y modificaciones ideados para el sistema se plantean:

- **Desarrollo de la aplicación para otros sistemas operativos**. Actualmente, los dispositivos que no dispongan de sistema operativo Android no pueden utilizar la aplicación en sus móviles. Se plantea, por tanto, el desarrollo del sistema para dispositivos iOS (con lo que se cubriría el mercado actual casi en su totalidad). Este trabajo no supondría partir de cero, puesto que en este punto conocemos perfectamente las características del sistema y el resultado que queremos obtener y, con ello, reduciríamos nuestras tareas a adecuar la construcción adoptada a un nuevo lenguaje de programación, muy probablemente Swift.

- **Nuevos modos de visualización de las señales**. Puede ser interesante mostrar, igual que se hace ahora en el dominio temporal, las señales en el dominio frecuencial en la propia aplicación móvil e incluso realizar lo propio para las señales capturadas del giroscopio.

- **Implementar un almacenamiento local temporal**. Con una modificación de este tipo podrían guardarse, durante un tiempo, las mediciones en el propio dispositivo móvil, de modo que pudieran realizarse medidas incluso sin estar conectados a la red, y que éstas fueran transmitidas al servidor cuando el usuario volviera a conectarse.

- **Establecimiento remoto del tiempo de medición y frecuencia de muestreo**. Si la duración de las medidas y el tiempo entre muestras fuese establecida en el servidor e informado por éste a la aplicación móvil, podríamos mantener mejor la coherencia entre los datos recogidos de forma automática y sin intervención del usuario.

- **Incorporar imágenes o vídeos sobre los protocolos**. Con el fin de facilitar la experiencia del usuario y evitar errores a la hora de realizar las medidas, sería práctico incluir, cuando un usuario selecciona un protocolo para realizar la medida, una imagen descriptiva o incluso un vídeo corto que explicase la postura que éste debe adoptar para realizar la medida correctamente.

- **Creación de un sistema de mensajería entre médicos y pacientes**. Con esta funcionalidad podríamos proporcionar un método para que, al ver los resultados que se recogen del paciente, el médico pudiera enviarle comentarios, recomendaciones o establecer una fecha de consulta con él.

- **Adaptación del sistema a la nueva política de privacidad de datos**. A partir de finales de mayo de 2018, entra en vigor en España el Reglamento General de Protección de Datos (RGPD) de la Unión Europea relativo al tratamiento y circulación de los datos personales de las personas físicas. Para adaptarnos a esta normativa será necesario asegurar la implementación de las medidas aplicadas por la ley y solicitar el consentimiento explícito del usuario para el tratamiento de sus datos, así como informarle de la necesidad y aplicación del procesamiento de éstos.

- **Recogida de un número mayor de muestras**. En este primer análisis realizado, el restringido número de muestras ha limitado los estudios a elaborar, de manera que las conclusiones y resultados no pueden asumirse como totalmente concluyentes. Para ello, deben recogerse más muestras de temblores fisiológicos, con lo que podríamos descartar posibles medidas erróneas y profundizar en mayor detalle.

- **Captura de medidas de temblores pertenecientes a pacientes con patologías**. El estudio de medidas de temblores procedentes de personas con

enfermedades asociadas a los temblores puede ofrecernos nuevos e interesantes resultados, distintos a los que podemos descubrir con el estudio de los temblores fisiológicos. Es por este motivo, que la obtención de esta clase de señales es especialmente importante para el uso del sistema en el campo de la medicina. Por otra parte, tratar con nuevos tipos de temblores supone la necesidad de validar los protocolos utilizados actualmente para la adquisición de medidas en el sistema y su aplicación en los diferentes casos, así como, la posibilidad de incluir otros nuevos que se adapten mejor a cada patología concreta.

- **Estudio más avanzado de las señales**. Para llegar a una conclusión totalmente definitiva y convincente en lo que respecta a la naturaleza de los temblores es necesario estudiar más en profundidad todos los datos que puedan ser recogidos y aplicar sobre ellos análisis más complejos. Podría ser interesante, especialmente si el número de muestras es lo suficientemente grande, la aplicación de estudios relacionados con el deep y el machine learning, que tantos avances han sufrido en los últimos años, y que se podrían implementar en nuestro servidor Python.

- **Desarrollo de una plataforma web de visualización**. Con esta herramienta podríamos observar tanto las medidas realizadas con los dispositivos móviles como los resultados obtenidos, en una página web, de una forma más atractiva para médicos e investigadores.

- **Automatización del análisis de señal**. Una vez conocidas las características de los distintos tipos de temblores podría sistematizarse el análisis directamente en el servidor. Esta es una de las principales ventajas que nos ofrece haber desarrollado la lógica del sistema en Python, pues consta de numerosas librerías especializadas en el tratamiento y procesamiento de datos.

Referencias

[1] Fasano A, Deuschl G. *Therapeutic advances in tremor*. Mov Disord, septiembre 2015.

[2] Ruiz-Martínez J., Arratible-Echarren I., Gorostidi-Pagola A., Bergareche A., Martí-Massó J.F. *El temblor: actualidad y controversias*. Enero, 2009.

[3] Marshall J. *The effect of aging upon physiological tremor*. J Neurol Neurosurg Psychiat 2002.

[4] National Institute of Neurological Disorders and Stroke, Office of Communications and Public Liaison, National Institute of Health. *Tremor*. U.S. Department of health and human services, agosto 2017.

[5] Hernández Mandado P., Idania Vela T. *Diagnóstico diferencial del temblor*. Agosto, 2009.

[6] Ojeda López M.C., Rodríguez Weber F., Amaya Sánchez L.E. *Diagnóstico diferencial del temblor*. Mayo, 2009.

[7] Venegas P., Gómez R., Sinning M. *Temblor esencial: una revision crítica*. Marzo, 2010.

[8] Jankovic J, Lang AE. *Diagnosis and assessment of Parkinson disease and other movement disorders*. In: Daroff RB, Jankovic J, Mazziotta JC, Pomeroy SL, eds. Bradley's Neurology in Clinical Practice. 7th ed. Philadelphia, PA: Elsevier, 2016.

[9] Héctor A. Gonzalez-Usigli, Alberto Espay. *Temblor – Trastornos neurológicos*. In: MSD (Merck and Co., Inc).

[10] De Jong RN. *The Neurological Examination*. New York, New York. Hoeber, 3. ED, 1967.

[11] *Android Developers*. Página web oficial: https://developer.android.com

[12] *IntelliJ IDEA: The Java IDE for Proffessional Developers by JetBrains*. Página web oficial: https://www.jetbrains.com/idea

[13] *Gradle Build Tool*. Página web oficial: https://gradle.org

[14] *PyCharm: Python IDE for Professional Developers by JetBrains*. Página web oficial: https://www.jetbrains.com/pycharm

[15] *MySQL*. Página web oficial: https://www.mysql.com

[16] *MATLAB – El lenguaje del cálculo técnico*. Página web oficial: https://es.mathworks.com/products/matlab.html

[17] Hallett M. *Overview of human tremor physiology. Movement Disorders*, 1998.

[18] Marshall J. *Observation on essential tremor*. J Neurol Neurosurg Psichiat 2004.

[19] Gates PC. *Orthostatic tremor of the shaky legs sindrome*. Clin Exp Neurol 2005.

[20] Deuschl G, Krack P, Lauk M, Timmer J. *Clinical neurophysiology of tremor*. J Clin Neurophysiol, 1996.

[21] Jacobson, I., Booch, G., & Rumbaugh, J. *El proceso unificado de desarrollo de software/The unified software development process*. Pearson Educación. 2000.

[22] Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology*. Diciembre, 1990.

[23] Toro, A. D., & Jiménez, B. B. *Metodología para la Elicitación de Requisitos de Sistemas Software*. Octubre, 2000.

[24] Delía, L. N., Galdamez, N., Thomas, P. J., Corbalán, L. C., & Pesado, P. M. *Análisis experimental de desarrollo de aplicaciones móviles*

multiplataforma. In XX Congreso Argentino de Ciencias de la Computación Buenos Aires, 2014.

[25] *Desarrollo de aplicaciones de Xamarin con Visual Studio*. Página web oficial: https://visualstudio.microsoft.com/es/xamarin

[26] *Desarrollo de aplicaciones multiplataforma – GeneXus*. Página web oficial: https://www.genexus.com/

[27] Tomás J. *El gran libro de Android*. Marcombo, 2013

[28] Minera, F. *Curso de programación PHP*. 2008

[29] *Python*. Página web oficial: https://www.python.org

[30] Lund, M. I., Ferrarini Oliver, C., Aballay, L. N., Romagnano, M. G., & Meni, E. *Plantilla para documentar casos de uso*. In V Congreso de Tecnología en Educación y Educación en Tecnología, 2010.

[31] González, Y. D., & Romero, Y. F. *Patrón Modelo-Vista-Controlador*. Revista Telem@tica, 2012.

[32] *Android Graph View plotting library*. Página web oficial: http://www.android-graphview.org

[33] Python Software Foundation. *Entornos virtuales y paquetes*. In: Python documentation. Disponible en http://docs.python.org.ar/tutorial/3/venv.html

A. Apéndice I – Documentación de la aplicación Android

A lo largo de este apéndice se detalla toda la información relevante relativa a los atributos y métodos de las clases Java desarrolladas para implementar la aplicación Android.

A.1. Clase AddPatientActivity

```
public class AddPatientActivity
extends android.support.v7.app.AppCompatActivity
```

Actividad encargada de solicitar el dni/email de un paciente y enviar a dicho paciente una petición para relacionarlo con un médico

Author:

Javier Pérez Ácimas

Field Detail

- *modoBusqueda*

```
private int modoBusqueda
```

Variable que distingue el modo en que se va a buscar al paciente: dni o email

- *modoBusquedaTextView*

```
private android.widget.TextView modoBusquedaTextView
```

Textview que indica el modo de búsqueda que se ha seleccionado

- *modoBusquedaEditText*

```
private android.widget.EditText modoBusquedaEditText
```

Edittext en el que se introduce el dni o email del paciente que se busca

Constructor Detail

- *AddPatientActivity*

```
public AddPatientActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de vida de la actividad que se invoca cuando ésta se crea y que obtiene las referencias a los elementos de la interfaz

Overrides:

```
onCreate in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
savedInstanceState - estado guardado de la actividad
```

- *onCreateOptionsMenu*

```
public boolean onCreateOptionsMenu(android.view.Menu menu)
```

Método encargado de la creación de un menú de opciones que es llamado al crear la actividad

Overrides:

```
onCreateOptionsMenu in class android.app.Activity
```

Parameters:

```
menu - menu
```

Returns:

```
true si se muestra el menú
```

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

```
onOptionsItemSelected in class android.app.Activity
```

Parameters:

```
item - elemento seleccionado
```

Returns:

```
true cuando se gestiona correctamente la opción seleccionada / false en
caso contrario
```

- *anadirPaciente*

```
public void anadirPaciente(android.view.View view)
```

Método invocado al pulsar el botón de enviar para realizar la petición a un paciente de añadirse al médico

Parameters:

```
view - vista del botón pulsado
```

- *modoDni*

```
public void modoDni()
```

Método que realiza los cambios necesarios en la actividad cuando se selecciona el modo de búsqueda mediante dni

- *modoEmail*

```
public void modoEmail()
```

Método que realiza los cambios necesarios en la actividad cuando se selecciona el modo de búsqueda mediante email

- *onSaveInstanceState*

```
public void onSaveInstanceState(android.os.Bundle outState)
```

Método que guarda el estado de ciertos elementos de la aplicación para su posible recuperación posterior

Overrides:

```
onSaveInstanceState in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
outState - estado de la aplicación
```

- *onRestoreInstanceState*

```
public void onRestoreInstanceState(android.os.Bundle savedInstanceState)
```

Método que recupera el estado de los elementos de la aplicación que han sido guardados previamente

Overrides:

```
onRestoreInstanceState in class android.app.Activity
```

Parameters:

```
savedInstanceState - estado de la aplicación que se recuperará
```

A.2. Clase DetectionActivity

```
public class DetectionActivity
extends android.support.v7.app.AppCompatActivity
implements android.hardware.SensorEventListener
```

Actividad encargada de grabar los cambios en los sensores cuando el usuario inicie una medición

Author:

Javier Pérez Ácimas

Field Detail

- *mSensor*

```
private android.hardware.SensorManager mSensor
```

Atributo SensorManager necesario para acceder a los sensores del dispositivo

- *acEjeX*

```
private java.util.ArrayList<java.lang.Float> acEjeX
```

Lista de valores almacenados para el eje X del acelerómetro (en m/s^2)

- *acEjeY*

```
private java.util.ArrayList<java.lang.Float> acEjeY
```

Lista de valores almacenados para el eje Y del acelerómetro (en m/s^2)

- *acEjeZ*

```
private java.util.ArrayList<java.lang.Float> acEjeZ
```

Lista de valores almacenados para el eje Z del acelerómetro (en m/s^2)

- *gyEjeX*

```
private java.util.ArrayList<java.lang.Float> gyEjeX
```

Lista de valores almacenados para el eje X del giroscopio (en rad/s)

- *gyEjeY*

```
private java.util.ArrayList<java.lang.Float> gyEjeY
```

Lista de valores almacenados para el eje Y del giroscopio (en rad/s)

- *gyEjeZ*

```
private java.util.ArrayList<java.lang.Float> gyEjeZ
```

Lista de valores almacenados para el eje Z del giroscopio (en rad/s)

- *frecAc*

```
private float frecAc
```

Frecuencia de muestreo para el acelerómetro (en Hz)

- *frecGy*

```
private float frecGy
```

Frecuencia de muestreo para el giroscopio (en Hz)

- *msTime*

```
private int msTime
```

Tiempo que durará la medición (en milisegundos)

- *lastTime*

```
private double lastTime
```

Contador del instante de tiempo de la medición

- *graph*

```
private GraphView graph
```

Gráfica en la que se muestran los datos recogidos del acelerómetro

- *acEjeXLine*

```
private <any> acEjeXLine
```

Línea de la gráfica que muestra los valores recogidos para el eje X del acelerómetro

- *acEjeYLine*

```
private <any> acEjeYLine
```

Línea de la gráfica que muestra los valores recogidos para el eje Y del acelerómetro

- *acEjeZLine*

```
private <any> acEjeZLine
```

Línea de la gráfica que muestra los valores recogidos para el eje Z del acelerómetro

- *grabar*

```
private boolean grabar
```

Booleano que decide cuando se almacenan (true) y cuando no (false) los datos del sensor

- *cuentaAtrasTimer*

```
private android.os.CountDownTimer cuentaAtrasTimer
```

Temporizador para empezar a almacenar los datos del sensor

- *medicionTimer*

```
private android.os.CountDownTimer medicionTimer
```

Temporizador para la duración de la medida

- *acInfo*

```
private java.lang.String acInfo
```

Información del acelerómetro del dispositivo

- *gyInfo*

```
private java.lang.String gyInfo
```

Información del giroscopio del dispositivo

- *acEjeXCalibracion*

```
private java.util.ArrayList<java.lang.Float> acEjeXCalibracion
```

Lista de valores de calibración almacenados para el eje X del acelerómetro (en m/s^2)

- *acEjeYCalibracion*

```
private java.util.ArrayList<java.lang.Float> acEjeYCalibracion
```

Lista de valores de calibración almacenados para el eje Y del acelerómetro (en m/s^2)

- *acEjeZCalibracion*

```
private java.util.ArrayList<java.lang.Float> acEjeZCalibracion
```

Lista de valores de calibración almacenados para el eje Z del acelerómetro (en m/s^2)

- *gyEjeXCalibracion*

```
private java.util.ArrayList<java.lang.Float> gyEjeXCalibracion
```

Lista de valores de calibración almacenados para el eje X del giroscopio (en rad/s)

- *gyEjeYCalibracion*

```
private java.util.ArrayList<java.lang.Float> gyEjeYCalibracion
```

Lista de valores de calibración almacenados para el eje Y del giroscopio (en rad/s)

- *gyEjeZCalibracion*

```
private java.util.ArrayList<java.lang.Float> gyEjeZCalibracion
```

Lista de valores de calibración almacenados para el eje Z del giroscopio (en rad/s)

- *msTimeCalibration*

```
private int msTimeCalibration
```

Duración de la medida de calibración (en ms)

- *sensoresCalibrados*

```
private boolean sensoresCalibrados
```

Indicador de si se ha realizado o no la medida de calibración

- *instrucciones*

```
private android.widget.TextView instrucciones
```

TextView indicador con las instrucciones que tiene que ir siguiendo el usuario

- *modoMedida*

```
private java.lang.String modoMedida
```

Cadena que identifica la manera (protocolo) de la que se va a realizar la medida el usuario

- *modoMedidaSeleccionado*

```
private boolean modoMedidaSeleccionado
```

Indicador de si se ha seleccionado la elección del protocolo de medida

Constructor Detail

- *DetectionActivity*

```
public DetectionActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método llamado al crearse la actividad que obtiene las referencias necesarias para la misma y define el funcionamiento de los temporizadores

Overrides:

```
onCreate in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
savedInstanceState - estado guardado de la actividad
```

- *resetearTemporizadores*

```
public void resetearTemporizadores()
```

Método encargado de crear los temporizadores, estableciendo el tiempo adecuado para la siguiente medida a realizar

- *detectarTemblores*

```
public void detectarTemblores(android.view.View view)
```

Método que se invoca cuando el usuario pulsa el botón de empezar a grabar datos, encargado de inicializar las listas para almacenar los datos y activar el primer temporizador

Parameters:

```
view - vista del botón pulsado
```

- *onSensorChanged*

```
public void onSensorChanged(android.hardware.SensorEvent event)
```

Método invocado cuando se detecta un cambio en el valor medido por un sensor

Specified by:

```
onSensorChanged in interface android.hardware.SensorEventListener
```

Parameters:

```
event - cambio en un sensor
```

- *refreshValuesAc*

```
private void refreshValuesAc(android.hardware.SensorEvent event)
```

Método que se invoca cuando se detecta un cambio en el acelerómetro que almacena los datos si está activada la grabación y los pinta en la gráfica

Parameters:

```
event - cambio en el acelerómetro
```

- *refreshValuesGy*

```
private void refreshValuesGy(android.hardware.SensorEvent event)
```

Método que se invoca cuando se detecta un cambio en el giroscopio que almacena los datos si está activada la grabación

Parameters:

```
event - cambio en el giroscopio
```

- *onPause*

```
protected void onPause()
```

Método del ciclo que de vida que se ejecuta cuando se pausa la actividad y que detiene la medición en curso, apaga los sensores y detiene los temporizadores

Overrides:

```
onPause in class android.support.v4.app.FragmentActivity
```

- *onResume*

```
protected void onResume()
```

Método del ciclo de vida que se ejecuta cuando se activa la actividad

Overrides:

```
onResume in class android.support.v4.app.FragmentActivity
```

- *iniciarSensores*

```
protected boolean iniciarSensores()
```

Método que se encarga de obtener y activar los sensores (acelerómetro y giroscopio)

Returns:

```
true si se obtienen y activan los sensores correctamente / false en caso
contrario
```

- *confirmarMedida*

```
public void confirmarMedida()
```

Método que se ejecuta cuando se finaliza una medición y que se encarga de lanzar una nueva actividad para la confirmación del envío de los datos al servidor

- *onAccuracyChanged*

- ```
  public void onAccuracyChanged(android.hardware.Sensor sensor,
                                int accuracy)
  ```

Método que se ejecuta cuando la precisión del sensor registrado cambia

Specified by:

```
onAccuracyChanged in interface android.hardware.SensorEventListener
```

Parameters:

```
sensor - sensor implicado

accuracy - la nueva precisió del sensor
```

- *parsearResultados*

```
public float[] parsearResultados(java.util.ArrayList<java.lang.Float> listaRes
ultados)
```

Método que pasa los resultados grabados de un ArrayList de Floats a un array float[] para poder enviarlos entre actividades

Parameters:

```
listaResultados - ArrayList con los datos almacenados
```

Returns:

```
el array[] con los mismos datos
```
```

- *lanzarGrafica*

```
private void lanzarGrafica()
```

Método que incializa la gráfica en la que se mostrarán los valores que se van capturando medidos del acelerómetro

- *addEntry*

```
private void addEntry(float valorAcEjex,
                      float valorAcEjey,
                      float valorAcEjez)
```

Método que añade un nuevo punto a cada línea de la gráfica

Parameters:

`valorAcEjex - punto a añadir medido para el eje X del acelerómetro`

`valorAcEjey - punto a añadir medido para el eje Y del acelerómetro`

`valorAcEjez - punto a añadir medido para el eje Z del acelerómetro`

- *onBackPressed*

```
public void onBackPressed()
```

Método que se ejecuta cuando el usuario pulsa el botón de retorno del dispositivo móvil

Overrides:

`onBackPressed in class android.support.v4.app.FragmentActivity`

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

`onOptionsItemSelected in class android.app.Activity`

Parameters:

`item - elemento seleccionado`

Returns:

`true cuando se gestiona correctamente la opción seleccionada / false en caso contrario`

- *finalizarCalibracion*

```
public void finalizarCalibracion()
```

Método invocado cuando la medida de calibración de los sensores ha finalizado y ya se puede realizar la medida de los temblores

- *calibracionCorrecta*

```
public boolean calibracionCorrecta()
```

Método que comprueba si la calibración se ha realizado correctamente

Returns:

```
true si la calibración ha sido correcta / false en caso contrario
```

- *onSaveInstanceState*

```
public void onSaveInstanceState(android.os.Bundle outState)
```

Método que guarda el estado de ciertos elementos de la aplicación para su posible recuperación posterior

Overrides:

```
onSaveInstanceState in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
outState - estado de la aplicación
```

- *onRestoreInstanceState*

```
public void onRestoreInstanceState(android.os.Bundle savedInstanceState)
```

Método que recupera el estado de los elementos de la aplicación que han sido guardados previamente

Overrides:

```
onRestoreInstanceState in class android.app.Activity
```

Parameters:

```
savedInstanceState - estado de la aplicación que se recuperará
```

- *floatArrayToList*

```
public java.util.ArrayList<java.lang.Float> floatArrayToList(float[] array)
```

Método que convierte un vector de números en una lista con los mismos

Parameters:

```
array - array a convertir
```

Returns:

```
la lista con los elementos del array
```

- *repetirCalibracion*

```
public void repetirCalibracion(android.view.View view)
```

Método invocado cuando el usuario pulsa el botón de repetir calibración que resetea los temporizadores e inicia la calibración

Parameters:

```
view - vista del botón pulsado
```

- *onRadioButtonClicked*

```
public void onRadioButtonClicked(android.view.View view)
```

Método que se invoca al pulsar cualquiera de los radio buttons de la interfaz para seleccionar el modo de realizar la medición

Parameters:

```
view - vista del RadioButton pulsado
```

- *protocoloSeleccionadoButton*

```
public void protocoloSeleccionadoButton(android.view.View view)
```

Método invocado cuando el usuario selecciona un protocolo para realizar la medida y pulsa el botón continuar

Parameters:

```
view - vista del botón pulsado
```

- *establecerInstrucciones*

```
public void establecerInstrucciones()
```

Método que actualiza el valor de la variable "modoMedida" en función del modo de medida que haya seleccionado el usuario

A.3. Clase DetectionActivityInfo

```
public class DetectionActivityInfo
extends android.support.v7.app.AppCompatActivity
```

Actividad encargada de mostrar la gráfica con los datos previamente grabados al medir y ofrecer la posibilidad de enviar los mismos al servidor para su almacenamiento

Author:

Javier Pérez Ácimas

Field Detail

- *acEjeX*

```
private float[] acEjeX
```

Datos almacenados del eje X del acelerómetro (en m/s^2)

- *acEjeY*

```
private float[] acEjeY
```

Datos almacenados del eje Y del acelerómetro (en m/s^2)

- *acEjeZ*

```
private float[] acEjeZ
```

Datos almacenados del eje Z del acelerómetro (en m/s^2)

- *gyEjeX*

```
private float[] gyEjeX
```

Datos almacenados del eje X del giroscopio (en rad/s)

- *gyEjeY*

```
private float[] gyEjeY
```

Datos almacenados del eje Y del giroscopio (en rad/s)

- *gyEjeZ*

```
private float[] gyEjeZ
```

Datos almacenados del eje Z del giroscopio (en rad/s)

- *tiempoMedidaMs*

```
private int tiempoMedidaMs
```

Tiempo que duró la medición (en ms)

- *frecuenciaAc*

```
private float frecuenciaAc
```

Frecuencia de muestreo del acelerómetro (en Hz)

- *frecuenciaGy*

```
private float frecuenciaGy
```

Frecuencia de muestreo del giroscopio (en Hz)

- *graph*

```
private Graph graph
```

Gráfica sobre la que se muestran los datos obtenidos del acelerómetro

- *acInfo*

```
private java.lang.String acInfo
```

Información del acelerómetro del dispositivo

- *gyInfo*

```
private java.lang.String gyInfo
```

Información del giroscopio del dispositivo

- *acEjeXCalibracion*

```
private float[] acEjeXCalibracion
```

Datos de calibración almacenados del eje X del acelerómetro (en m/s^2)

- *acEjeYCalibracion*

```
private float[] acEjeYCalibracion
```

Datos de calibración almacenados del eje Y del acelerómetro (en m/s^2)

- *acEjeZCalibracion*

```
private float[] acEjeZCalibracion
```

Datos de calibración almacenados del eje Z del acelerómetro (en m/s^2)

- *gyEjeXCalibracion*

```
private float[] gyEjeXCalibracion
```

Datos de calibración almacenados del eje X del giroscopio (en rad/s)

- *gyEjeYCalibracion*

```
private float[] gyEjeYCalibracion
```

Datos de calibración almacenados del eje Y del giroscopio (en rad/s)

- *gyEjeZCalibracion*

```
private float[] gyEjeZCalibracion
```

Datos de calibración almacenados del eje Z del giroscopio (en rad/s)

- *msTimeCalibration*

```
private int msTimeCalibration
```

Duración de la medida de calibración (en ms)

- *sexo*

```
private java.lang.String sexo
```

Cadena con el sexo del usuario para medidas a través de un perfil experimentador

- *input_diaNacimiento*

```
private android.widget.EditText input_diaNacimiento
```

EditText en que el usuario inserta su dia de nacimiento para medidas a través de un perfil experimentador

- *input_mesNacimiento*

```
private android.widget.EditText input_mesNacimiento
```

EditText en que el usuario inserta su mes de nacimiento para medidas a través de un perfil experimentador

- *input_anioNacimiento*

```
private android.widget.EditText input_anioNacimiento
```

EditText en que el usuario inserta su año de nacimiento para medidas a través de un perfil experimentador

- *input_patologia*

```
private android.widget.EditText input_patologia
```

EditText en el que el usuario inserta su patología para medidas a través de un perfil experimentador

- *input_comentarios*

```
private android.widget.EditText input_comentarios
```

EditText en el que el usuario inserta comentarios adicionales para medidas a través de un perfil experimentador

- *masMediciones*

```
private boolean masMediciones
```

Booleano que indica si se va a enviar la medición y realizar una nueva o simplemente se envía la actual

- *modoMedida*

```
private java.lang.String modoMedida
```

Cadena que identifica la manera de la cual va a realizar la medida el usuario

Constructor Detail

- *DetectionActivityInfo*

```
public DetectionActivityInfo()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de vida que se invoca al crear la actividad que recupera los datos obtenidos de los sensores y muestra los del acelerómetro en una gráfica

Overrides:

```
onCreate in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
savedInstanceState - estado guardado de la actividad
```

- *repetirMedicion*

```
public void repetirMedicion(android.view.View view)
```

Método que se invoca cuando el usuario pulsa el botón de repetir la medición y que devuelve a la actividad de medición

Parameters:

```
view - vista del botón pulsado
```

- *cancelar*

```
public void cancelar(android.view.View view)
```

Método que se incoa cuando el usuario pulsa el botón de cancelar la medición y que devuelve a la actividad principal

Parameters:

```
view - vista del botón pulsado
```

- *onResume*

```
public void onResume()
```

Mëtodo del ciclo de vida de la actividad que se invoca cuando esta se activa y que se encarga de que las líneas de la gráfica mantengan el estado de acuerdo a los checkboxes de visibilidad

Overrides:

```
onResume in class android.support.v4.app.FragmentActivity
```

- *confirmarMedicion*

```
public void confirmarMedicion(android.view.View view)
```

Método que se invoca cuando el usuario pulsa el botón de confirmar la medición y se encarga de enviar los datos al servidor

Parameters:

```
view - vista del botón pulsado
```

- *recuperarResultados*

```
public void recuperarResultados()
```

Método que se encarga de recuperar los datos que han sido enviados desde la actividad invocadora de esta

- *enviarMasNueva*

```
public void enviarMasNueva(android.view.View view)
```

Método invocado al pulsar el botón "send an new" que envía la medición realizada y vuelve a la actividad de medida para realizar una nueva

Parameters:

```
view - vista del botón pulsado
```

- *arrayToString*

```
public java.lang.String arrayToString(float[] array)
```

Método que convierte los valores contenidos en un array en un String para ser enviados al servidor

Parameters:

```
array - array con los datos
```

Returns:

```
la cadena de texto con los datos
```

- *mostrarResultadosGraficos*

```
public void mostrarResultadosGraficos()
```

Método encargado de mostrar los datos de los tres ejes del acelerómetro en la gráfica

- *enviarDatos*

```
public void enviarDatos(java.util.Map<java.lang.String,java.lang.String> data)
```

Método encargado de completar un mapa con los datos que serán enviados al servidor

Parameters:

```
data - mapa con parte de las parejas clave-dato que se enviará al
servidor
```

- *confirmarUsuarioExperimental*

```
public void confirmarUsuarioExperimental(android.view.View view)
```

Método que se invoca cuando un usuario experimentador pulsa el botón de finalizar al completar la medición con los datos del usuario que ha realizado la medición

Parameters:

```
view - vista del botón pulsado
```

- *onCheckBoxClicked*

```
public void onCheckBoxClicked(android.view.View view)
```

Método invocado cuando se pulsa cualquiera de los tres checkboxes que controlan la visibilidad de las líneas en la gráfica

Parameters:

```
view - checkbox pulsado
```

- *onRadioButtonClicked*

```
public void onRadioButtonClicked(android.view.View view)
```

Método que se invoca al pulsar cualquiera de los radio buttons de la interfaz para seleccionar el sexo del usuario

Parameters:

```
view - vista del RadioButton pulsado
```

- *onSaveInstanceState*

```
public void onSaveInstanceState(android.os.Bundle outState)
```

Método que guarda el estado de ciertos elementos de la aplicación para su posible recuperación posterior

Overrides:

```
onSaveInstanceState in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
outState - estado de la aplicación
```

- *onRestoreInstanceState*

```
public void onRestoreInstanceState(android.os.Bundle savedInstanceState)
```

Método que recupera el estado de los elementos de la aplicación que han sido guardados previamente

Overrides:

```
onRestoreInstanceState in class android.app.Activity
```

Parameters:

```
savedInstanceState - estado de la aplicación que se recuperará
```

- *onBackPressed*

```
public void onBackPressed()
```

Método que se ejecuta cuando el usuario pulsa el botón de retorno del dispositivo móvil

Overrides:

```
onBackPressed in class android.support.v4.app.FragmentActivity
```

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

```
onOptionsItemSelected in class android.app.Activity
```

Parameters:

```
item - elemento seleccionado
```

Returns:

```
true cuando se gestiona correctamente la opción seleccionada / false en
caso contrario
```

A.4. Clase DoctorInformationActivity

```
public class DoctorInformationActivity
extends android.support.v7.app.AppCompatActivity
```

Actividad encargada de mostrar los datos de un usuario con rol de médico

Author:

Javier Pérez Ácimas

Field Detail

- *input_dni_paciente*

```
private android.widget.EditText input_dni_paciente
```

Edit text con el dni del médico

- *input_nombre*

```
private android.widget.EditText input_nombre
```

Edit text con el nombre del médico

- *input_apellidos*

```
private android.widget.EditText input_apellidos
```

Edit text con los apellidos del médico

- *input_email*

```
private android.widget.EditText input_email
```

Edit text con el email del médico

- *input_centroMedico*

```
private android.widget.EditText input_centroMedico
```

Edit text con el centro médico del doctor

Constructor Detail

- *DoctorInformationActivity*

```
public DoctorInformationActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de vida de la actividad que se invoca cuando ésta es creada y que obtiene la referencia a todos los elementos de la interfaz

Overrides:

```
onCreate in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
savedInstanceState - estado guardado de la actividad
```

- *onResume*

```
public void onResume()
```

Método del ciclo de vida de la actividad que recupera los datos del médico

Overrides:

```
onResume in class android.support.v4.app.FragmentActivity
```

- *getData*

```
public void getData()
```

Método que recupera los datos de un médico y los muestra por pantalla

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

```
onOptionsItemSelected in class android.app.Activity
```

Parameters:

```
item - elemento seleccionado
```

Returns:

```
true cuando se gestiona correctamente la opción seleccionada / false en
caso contrario
```

- *onBackPressed*

```
public void onBackPressed()
```

Método que se ejecuta cuando el usuario pulsa el botón de retorno del dispositivo móvil

Overrides:

```
onBackPressed in class android.support.v4.app.FragmentActivity
```

A.5. Clase Graph

```
public class Graph
extends java.lang.Object
```

Clase encargada de crear y mostrar las gráficas de visualización de datos a través de las funcionalidades ofrecidas por la librería GraphView

Author:

```
Javier Pérez Ácimas
```

Field Detail

- *graph*

```
private GraphView graph
```

Gráfica sobre la que se muestran los datos

- *activity*

```
private android.app.Activity activity
```

Actividad que genera la gráfica

- *acEjeXLine*

```
private <any> acEjeXLine
```

Línea de la gráfica que muestra los datos del eje X del acelerómetro (en m/s^2)

- *acEjeYLine*

```
private <any> acEjeYLine
```

Línea de la gráfica que muestra los datos del eje Y del acelerómetro (en m/s^2)

- *acEjeZLine*

```
private <any> acEjeZLine
```

Línea de la gráfica que muestra los datos del eje Z del acelerómetro (en m/s^2)

- *tMuestreo*

```
private float tMuestreo
```

Tiempo entre muestras (inverso de la frecuencia de muestreo) en ms

Constructor Detail

- *Graph*

```
public Graph(android.app.Activity activity)
```

Constructor de la gráfica que obtiene la referencia al elemento Graphview de la interfaz

Parameters:

```
activity - actividad que quiere generar la gráfica
```

Method Detail

- *mostrarResultadosGraficos*

```
public void mostrarResultadosGraficos(float[] acEjeX,
                                      float[] acEjeY,
                                      float[] acEjeZ,
                                      int tiempoMedidaMs,
                                      float frec)
```

Método que genera la gráfica a partir de los datos recogidos de los tres ejes del acelerómetro

Parameters:

```
acEjeX - datos recogidos del eje x del acelerómetro

acEjeY - datos recogidos del eje y del acelerómetro

acEjeZ - datos recogidos del eje z del acelerómetro

tiempoMedidaMs - duración de la medida (en ms)

frec - frecuencia de muestreo del acelerómetro (en Hz)
```

- *generarSerie*

```
public <any> generarSerie(float[] datos)
```

Método que genera una línea de puntos que se pueda pintar en la gráfica a partir de un array con los datos obtenidos

Parameters:

```
datos - datos para generar la línea de puntos
```

Returns:

```
línea de puntos que se puede mostrar en la gráfica
```

- *borrarLinea*

```
public void borrarLinea(java.lang.String nombreLinea)
```

Método que muestra una de las líneas en la gráfica

Parameters:

```
nombreLinea - nombre de la línea a mostrar
```

- *mostrarLinea*

```
public void mostrarLinea(java.lang.String nombreLinea)
```

Método que elimina una de las líneas de la gráfica

Parameters:

```
nombreLinea - nombre de la línea a eliminar
```

A.6. Clase HistoryActivity

```
public class HistoryActivity
extends android.support.v7.app.AppCompatActivity
```

Actividad encargada de mostrar el historial de medidas que están asociadas a un paciente

Author:

Javier Pérez Ácimas

Field Detail

- *listaMedidas*

```
private java.util.ArrayList<java.lang.String> listaMedidas
```

Lista que contiene los identificadores de las medidas pertenecientes a un paciente

- *listaFechasMedidas*

```
private java.util.ArrayList<java.lang.String> listaFechasMedidas
```

Lista que contiene las fechas de realización de las medidas pertenecientes a un paciente

- *listaMedidasParcial*

```
private java.util.ArrayList<java.lang.String> listaMedidasParcial
```

Lista que contiene los identificadores de las medidas de un día concreto

- *modoCalendario*

```
private boolean modoCalendario
```

Variable que indica si el se visualiza el historial en modo Calendario o no

Constructor Detail

- *HistoryActivity*

```
public HistoryActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de vida de la actividad que se invoca cuando se crea

Overrides:

```
onCreate in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
savedInstanceState - estado guardado de la actividad
```

- *onResume*

```
public void onResume()
```

Método del ciclo de vida de la actividad que se invoca cuando esta se activa y se encarga de obtener la lista con las medidas de un usuario (si no se tienen ya) y mostrarla

Overrides:

```
onResume in class android.support.v4.app.FragmentActivity
```

- *onCreateOptionsMenu*

```
public boolean onCreateOptionsMenu(android.view.Menu menu)
```

Método encargado de la creación de un menú de opciones que es llamado al crear la actividad

Overrides:

```
onCreateOptionsMenu in class android.app.Activity
```

Parameters:

```
menu - menu
```

Returns:

```
true si se muestra el menú
```

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

```
onOptionsItemSelected in class android.app.Activity
```

Parameters:

```
item - elemento seleccionado
```

Returns:

```
true cuando se gestiona correctamente la opción seleccionada / false en
caso contrario
```

- *verLista*

```
public void verLista()
```

Método que muestra la lista con las mediciones que tiene asociadas un paciente

- *verCalendario*

```
public void verCalendario()
```

Método que muestra un calendario en el cual se pueden ver las mediciones que posee el paciente cada día concreto

- *recuperarHistorial*

```
public void recuperarHistorial()
```

Método que recupera del servidor la lista con las mediciones ha grabado un usuario

- *onSaveInstanceState*

```
public void onSaveInstanceState(android.os.Bundle outState)
```

Método que guarda el estado de ciertos elementos de la aplicación para su posible recuperación posterior

Overrides:

```
onSaveInstanceState in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
outState - estado de la aplicación
```

- *onRestoreInstanceState*

```
public void onRestoreInstanceState(android.os.Bundle savedInstanceState)
```

Método que recupera el estado de los elementos de la aplicación que han sido guardados previamente

Overrides:

```
onRestoreInstanceState in class android.app.Activity
```

Parameters:

```
savedInstanceState - estado de la aplicación que se recuperará
```

- *onBackPressed*

```
public void onBackPressed()
```

Método que se ejecuta cuando el usuario pulsa el botón de retorno del dispositivo móvil

Overrides:

```
onBackPressed in class android.support.v4.app.FragmentActivity
```

A.7. Clase HttpClient

```
public class HttpClient
extends java.lang.Object
```

Clase encargada de las comunicaciones https entre la aplicación y el servidor

Author:

Javier Pérez Ácimas

Field Detail

- *servidor*

```
private static java.lang.String servidor
```

Cadena que identifica la dirección del servidor para obtener la URL del mismo

- *urlConnection*

```
private javax.net.ssl.HttpsURLConnection urlConnection
```

Clase utiliza para realizar la conexión entre aplicación y servidor

- *response*

```
private org.json.JSONObject response
```

Objeto JSON en el que se guarda la respuesta

- *context*

```
private android.content.Context context
```

Contexto de la actividad invocadora

Constructor Detail

- *HttpClient*

```
public HttpClient(android.content.Context context)
```

Constructor de la clase que incializa el JSON en el que se guarda la respuesta

Method Detail

- *postData*

```
public org.json.JSONObject postData(org.json.JSONObject jsonMsg)
```

Método que envía un JSON al servidor a través del método POST y captura la respuesta del msimo

Parameters:

```
jsonMsg - JSON que se va a enviar
```

Returns:

```
el JSON que se recibe del servidor
```

- *readStream*

- ```
  private java.lang.String readStream(java.io.InputStream in)
                               throws java.io.IOException
  ```

Método encargado de leer la respuesta del servidor del buffer y convertirlo en cadena

Parameters:

```
in - flujo de entrda
```

Returns:

```
la cadena de texto leída del flujo de entrda
```

Throws:

```
java.io.IOException - si hay problemas al leer o cerrar el buffer
```
```

A.8. Clase LoginActivity

```
public class LoginActivity
extends android.support.v7.app.AppCompatActivity
```

Actividad que ofrece una pantalla de login a través de dni y password

Author:

Javier Pérez Ácimas

Field Detail

- *mdniView*

```
private android.widget.AutoCompleteTextView mdniView
```

Campo de introducción del dni en la pantalla

- *mPasswordView*

```
private android.widget.EditText mPasswordView
```

Campo de introducción del password en la pantalla

Constructor Detail

- *LoginActivity*

```
public LoginActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de vida de la actividad que se invoca cuando ésta es creada y que obtiene las referencias a las vistas del layout

Overrides:

onCreate in class android.support.v7.app.AppCompatActivity

Parameters:

savedInstanceState - estado guardado de la actividad

- *attemptLogin*

```
public void attemptLogin(android.view.View view)
```

Método que se ejecuta cuando el usuario pulsa el botón de inicar sesión y que se encarga de comprobar los datos que se han introducido por pantalla y enviarlos al servidor

Parameters:

```
view - vista del boton pulsado
```

- *registrarse*

```
public void registrarse(android.view.View view)
```

Método que se invoca cuando el usuario selecciona la opción de registrarse y que redirige a la actividad de registro

Parameters:

```
view - vista del elemento seleccionado
```

- *recuperarNotificaciones*

```
public org.json.JSONArray recuperarNotificaciones(java.lang.String idUser)
```

Método encargado de recuperar la lista con las notificaciones pendientes que tiene un usuario cuando este se ha logeado correctamente

Parameters:

```
idUser - identificador del usuario que se logea en la aplicación
```

Returns:

```
la lista con las notificaciones del usuario
```

- *onBackPressed*

```
public void onBackPressed()
```

Método invocado al pulsar el botón atrás del terminal móvil y que finaliza la aplicación

Overrides:

```
onBackPressed in class android.support.v4.app.FragmentActivity
```

A.9. Clase MainActivity

```
public class MainActivity
extends android.support.v7.app.AppCompatActivity
```

Actividad principal de los pacientes y que muestra todas las funcionalidades ofrecidas al mismo

Author:

Javier Pérez Ácimas

Constructor Detail

- *MainActivity*

```
public MainActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de vida de la actividad que es invocado cuando ésta es creada

Overrides:

onCreate in class android.support.v7.app.AppCompatActivity

Parameters:

savedInstanceState - estado guardado de la actividad

- *onCreateOptionsMenu*

```
public boolean onCreateOptionsMenu(android.view.Menu menu)
```

Método encargado de la creación de un menú de opciones que es llamado al crear la actividad

Overrides:

onCreateOptionsMenu in class android.app.Activity

Parameters:

menu - menu

Returns:

true si se muestra el menú

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

onOptionsItemSelected in class android.app.Activity

Parameters:

```
item - elemento seleccionado
```

Returns:

```
true cuando se gestiona correctamente la opción seleccionada / false en
caso contrario
```

- *realizarMedicion*

```
public void realizarMedicion(android.view.View view)
```

Método invocado cuando el paciente selecciona la opción de realizar una nueva medida

Parameters:

```
view - vista del elemento seleccionado
```

- *verHistorial*

```
public void verHistorial(android.view.View view)
```

Método invocado cuando el paciente selecciona la opción de ver su historial de medidas realizadas

Parameters:

```
view - vista del elemento seleccionado
```

- *verDatosUsuario*

```
public void verDatosUsuario(android.view.View view)
```

Método invocado cuando el paciente selecciona la opción de ver sus datos personales

Parameters:

```
view - vista del elemento seleccionado
```

- *onBackPressed*

```
public void onBackPressed()
```

Método invocado al pulsar el botón atrás del terminal móvil y que finaliza la aplicación

Overrides:

```
onBackPressed in class android.support.v4.app.FragmentActivity
```

- *salir*

```
public void salir()
```

Método invocado al querer salir de la app. Si el perfil es de médico devuelve a la lista de pacientes del mismo

- *gestionarNotificaciones*

```
public boolean gestionarNotificaciones(java.util.ArrayList<java.lang.String> l
istIdNotificaciones,

java.util.ArrayList<java.lang.String> listAsuntoNotificaciones,

java.util.ArrayList<java.lang.String> listNombresGeneradores,
                                      android.app.Activity activity)
```

Método encargado de tratar las notificaciones del usuario si las hubiera

Parameters:

`listIdNotificaciones` - identificador de las notificaciones pendientes

`listAsuntoNotificaciones` - asunto de las notificaciones pendientes

`listNombresGeneradores` - nombres de los usuarios que generaron cada notificación

`activity` - actividad invocadora

Returns:

`true` si la operación se realiza con éxito / `false` en caso contrario

- *solicitudPaciente*

```
public void solicitudPaciente(java.lang.String idNotificacion,
                              java.lang.String nombreMedico,
                              android.app.Activity activity)
```

Método que muestra un diálogo con el nombre del médico que quiere añadirse al paciente y ofrece la posiblidad de aceptar o rechazar al mismo

Parameters:

`idNotificacion` - identificador de la solicitud

`nombreMedico` - nombre del médico que solicita al paciente

`activity` - actividad generadora

A.10. Clase MainDoctorActivity

```
public class MainDoctorActivity
extends android.support.v7.app.AppCompatActivity
```

Actividad principal del médico y que muestra todas las opciones ofrecidas al mismo

Author:

Javier Pérez Ácimas

Constructor Detail

- *MainDoctorActivity*

```
public MainDoctorActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de la vida de la actividad que se invoca cuando ésta es creada

Overrides:

onCreate in class android.support.v7.app.AppCompatActivity

Parameters:

savedInstanceState - estado guardado de la actividad

- *verDatosUsuario*

```
public void verDatosUsuario(android.view.View view)
```

Método invocado cuando el médico selecciona la opción de ver sus datos personales

Parameters:

view - vista del elemento seleccionado

- *verListaPacientes*

```
public void verListaPacientes(android.view.View view)
```

Método invocado cuando el médico selecciona la opción de ver la lista de pacientes que posee

Parameters:

view - vista del elemento seleccionado

- *anadirPaciente*

```
public void anadirPaciente(android.view.View view)
```

Método invocado cuando el médico selecciona la opción de añadirse un nuevo paciente

Parameters:

```
view - vista del elemento seleccionado
```

- *onBackPressed*

```
public void onBackPressed()
```

Método invocado al pulsar el botón atrás del terminal móvil y que finaliza la aplicación

Overrides:

```
onBackPressed in class android.support.v4.app.FragmentActivity
```

- *salir*

```
public void salir()
```

Método invocado al querer salir de la app.

- *onCreateOptionsMenu*

```
public boolean onCreateOptionsMenu(android.view.Menu menu)
```

Método encargado de la creación de un menú de opciones que es llamado al crear la actividad

Overrides:

```
onCreateOptionsMenu in class android.app.Activity
```

Parameters:

```
menu - menu
```

Returns:

```
true si se muestra el menú
```

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

```
onOptionsItemSelected in class android.app.Activity
```

Parameters:

```
item - elemento seleccionado
```

Returns:

```
true cuando se gestiona correctamente la opción seleccionada / false en
caso contrario
```

A.11. Clase MeasureActivityInfo

```
public class MeasureActivityInfo
extends android.support.v7.app.AppCompatActivity
```

Actividad que muestra la información relativa a una medida concreta

Author:

Javier Pérez Ácimas

Field Detail

- *graph*

```
private Graph graph
```

Gráfica sobre la que se muestran los datos grabados de los ejes del acelerómetro

- *id_medida*

```
private java.lang.String id_medida
```

Indentificador en la base de datos de la medida a mostrar

- *acEjeX*

```
private float[] acEjeX
```

Array con los datos grabados para el eje X del acelerómetro (en m/s^2)

- *acEjeY*

```
private float[] acEjeY
```

Array con los datos grabados para el eje Y del acelerómetro (en m/s^2)

- *acEjeZ*

```
private float[] acEjeZ
```

Array con los datos grabados para el eje Z del acelerómetro (en m/s^2)

- *tiempoMedidaMs*

```
private int tiempoMedidaMs
```

Duración de la medida (en ms)

- *frecAc*

```
private float frecAc
```

Frecuencia de muestreo de la medición (en Hz)

Constructor Detail

- *MeasureActivityInfo*

```
public MeasureActivityInfo()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de vida de la actividad que se invoca cuando esta es creada

Overrides:

```
onCreate in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
savedInstanceState - estado guardado de la actividad
```

- *onResume*

```
protected void onResume()
```

Método del ciclo de la actividad que se invoca cuando ésta es activada y que se encarga de solicitar los datos de la medida a visualizar y de mostrarlos en un gráfica

Overrides:

```
onResume in class android.support.v4.app.FragmentActivity
```

- *getFloatArray*

```
public float[] getFloatArray(java.lang.String stringData)
```

Método que convierte el String (recibido del Json enviado por el servidor) de los datos capturados del sensor a un array para poder mostrarlo en la gráfica

Parameters:

```
stringData - string con los floats en la forma: [float1, float2, ...]
```

Returns:

```
el array con los mismos datos
```

- *onCheckBoxClicked*

```
public void onCheckBoxClicked(android.view.View view)
```

Método invocado cuando se selecciona cualquiera de los checkboxes que controlan la visibilidad de los ejes en la gráfica

Parameters:

```
view - vista del checkbox seleccionado
```

- *onSaveInstanceState*

```
public void onSaveInstanceState(android.os.Bundle outState)
```

Método que guarda el estado de ciertos elementos de la aplicación para su posible recuperación posterior

Overrides:

onSaveInstanceState in class android.support.v7.app.AppCompatActivity

Parameters:

outState - estado de la aplicación

- *onRestoreInstanceState*

```
public void onRestoreInstanceState(android.os.Bundle savedInstanceState)
```

Método que recupera el estado de los elementos de la aplicación que han sido guardados previamente

Overrides:

onRestoreInstanceState in class android.app.Activity

Parameters:

savedInstanceState - estado de la aplicación que se recuperará

- *mostrarGrafica*

```
public void mostrarGrafica()
```

Método que se encarga de pintar los tres ejes del acelerómetro en la gráfica y borrar aquellos que ha desseleccionado el usuario

- *onStop*

```
public void onStop()
```

Método del ciclo de vida de la actividad que se ejecuta cuando está va a ser detenida y que se encarga de limpiar la gráfica (será pintada de nuevo cuando se reinicie)

Overrides:

onStop in class android.support.v7.app.AppCompatActivity

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

onOptionsItemSelected in class android.app.Activity

Parameters:

item - elemento seleccionado

Returns:

true cuando se gestiona correctamente la opción seleccionada / false en caso contrario

- *onBackPressed*

```
public void onBackPressed()
```

Método que se ejecuta cuando el usuario pulsa el botón de retorno del dispositivo móvil

Overrides:

onBackPressed in class android.support.v4.app.FragmentActivity

A.12. Clase PatientInformationActivity

```
public class PatientInformationActivity
extends android.support.v7.app.AppCompatActivity
```

Actividad encargada de mostrar los datos personales de un paciente

Author:

Javier Pérez Ácimas

Field Detail

- *input_dni_paciente*

```
private android.widget.EditText input_dni_paciente
```

Edit text con el dni del paciente

- *input_nombre*

```
private android.widget.EditText input_nombre
```

Edit text con el nombre del paciente

- *input_apellidos*

```
private android.widget.EditText input_apellidos
```

Edit text con los apellidos del paciente

- *input_fechaNacimiento*

```
private android.widget.EditText input_fechaNacimiento
```

Edit text con la fecha de nacimiento del paciente

- *input_email*

```
private android.widget.EditText input_email
```

Edit text con el email del paciente

- *input_telefono*

```
private android.widget.EditText input_telefono
```

Edit text con el telefono del paciente

- *input_direccion*

```
private android.widget.EditText input_direccion
```

Edit text con la direccion del paciente

- *input_sexo*

```
private android.widget.EditText input_sexo
```

Edit text con el género del paciente

- *input_patologia*

```
private android.widget.EditText input_patologia
```

EditText con la patología del paciente

- *input_comentarios*

```
private android.widget.EditText input_comentarios
```

EditText con la información adicional del paciente

- *input_nombreMedico*

```
private android.widget.EditText input_nombreMedico
```

Edit text con el nombre del médico del paciente

- *input_contactoMedico*

```
private android.widget.EditText input_contactoMedico
```

Edit text con el email de contacto del médico del paciente

Constructor Detail

- *PatientInformationActivity*

```
public PatientInformationActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de vida de la actividad que se invoca cuando ésta es creada y que obtiene la referencia a todos los elementos de la interfaz

Overrides:

```
onCreate in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
savedInstanceState - estado guardado de la actividad
```

- *onResume*

```
public void onResume()
```

Método del ciclo de vida de la actividad que recupera los datos del paciente

Overrides:

```
onResume in class android.support.v4.app.FragmentActivity
```

- *getData*

```
public void getData()
```

Método que recupera los datos de un paciente y los muestra por pantalla

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

```
onOptionsItemSelected in class android.app.Activity
```

Parameters:

```
item - elemento seleccionado
```

Returns:

```
true cuando se gestiona correctamente la opción seleccionada / false en
caso contrario
```

- *onBackPressed*

```
public void onBackPressed()
```

Método que se ejecuta cuando el usuario pulsa el botón de retorno del dispositivo móvil

Overrides:

```
onBackPressed in class android.support.v4.app.FragmentActivity
```

A.13. Clase PatientListActivity

```
public class PatientListActivity
extends android.support.v7.app.AppCompatActivity
```

Actividad que muestra la lista de los pacientes que tiene ligados un médico y desde la cual se puede acceder a las funcionalidades de cualquiera de ellos

Author:

Javier Pérez Ácimas

Field Detail

- *listaPacientes*

```
private java.util.ArrayList<java.lang.String> listaPacientes
```

Lista con los id_paciente de la base de datos de los pacientes del médico

- *listaNombresPacientes*

```
private java.util.ArrayList<java.lang.String> listaNombresPacientes
```

Lista con los [apellidos, nombre] de los pacientes del médico

Constructor Detail

- *PatientListActivity*

```
public PatientListActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Método del ciclo de vida de la actividad que se invoca cuando ésta es creada

Overrides:

onCreate in class android.support.v7.app.AppCompatActivity

Parameters:

savedInstanceState - estado guardado de la actividad

- *onResume*

```
public void onResume()
```

Método del ciclo de vida de la actividad que se invoca cuando ésta es creada y que se encarga de mostrar la lista de pacientes tras recuperarla del servidor (si no está ya guardada)

Overrides:

```
onResume in class android.support.v4.app.FragmentActivity
```

- *recuperarListaPacientes*

```
public void recuperarListaPacientes()
```

Método que recupera la lista de los pacientes asociados a un médico

- *verLista*

```
public void verLista()
```

Método que muestra por pantalla la lista de los pacientes asociados al médico

- *onSaveInstanceState*

```
public void onSaveInstanceState(android.os.Bundle outState)
```

Método que guarda el estado de ciertos elementos de la aplicación para su posible recuperación posterior

Overrides:

```
onSaveInstanceState in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
outState - estado de la aplicación
```

- *onRestoreInstanceState*

```
public void onRestoreInstanceState(android.os.Bundle savedInstanceState)
```

Método que recupera el estado de los elementos de la aplicación que han sido guardados previamente

Overrides:

```
onRestoreInstanceState in class android.app.Activity
```

Parameters:

```
savedInstanceState - estado de la aplicación que se recuperará
```

- *onOptionsItemSelected*

```
public boolean onOptionsItemSelected(android.view.MenuItem item)
```

Método encargado de la gestión de eventos de pulsación de las opciones ofrecidas por el menú

Overrides:

```
onOptionsItemSelected in class android.app.Activity
```

Parameters:

```
item - elemento seleccionado
```

Returns:

```
true cuando se gestiona correctamente la opción seleccionada / false en
caso contrario
```

- *onBackPressed*

```
public void onBackPressed()
```

Método que se ejecuta cuando el usuario pulsa el botón de retorno del dispositivo móvil

Overrides:

```
onBackPressed in class android.support.v4.app.FragmentActivity
```

A.14. Clase RegisterActivity

```
public class RegisterActivity
extends android.support.v7.app.AppCompatActivity
```

Actividad que ofrece un formulario de registro para poder interactuar con la aplicación

Author:

Javier Pérez Ácimas

Field Detail

- *input_dni_paciente*

```
private android.widget.EditText input_dni_paciente
```

EditText en que el usuario inserta su dni

- *input_pass*

```
private android.widget.EditText input_pass
```

EditText en que el usuario inserta su password

- *input_nombre*

```
private android.widget.EditText input_nombre
```

EditText en que el usuario inserta su nombre

- *input_apellidos*

```
private android.widget.EditText input_apellidos
```

EditText en que el usuario inserta sus apellidos

- *input_diaNacimiento*

```
private android.widget.EditText input_diaNacimiento
```

EditText en que el usuario inserta su dia de nacimiento

- *input_mesNacimiento*

```
private android.widget.EditText input_mesNacimiento
```

EditText en que el usuario inserta su mes de nacimiento

- *input_anioNacimiento*

```
private android.widget.EditText input_anioNacimiento
```

EditText en que el usuario inserta su año de nacimiento

- *sexo*

```
private java.lang.String sexo
```

String en el que se guarda el sexo que selecciona el usuario

- *input_email*

```
private android.widget.EditText input_email
```

EditText en que el usuario inserta su email

- *input_telefono*

```
private android.widget.EditText input_telefono
```

EditText en que el usuario inserta su teléfono

- *input_direccion*

```
private android.widget.EditText input_direccion
```

EditText en que el usuario inserta su dirección

- *input_repeat_pass*

```
private android.widget.EditText input_repeat_pass
```

EditText en que el usuario inserta la confirmación del password

- *input_patologia*

```
private android.widget.EditText input_patologia
```

EditText en el que el usuario inserta su patología

- *input_comentarios*

```
private android.widget.EditText input_comentarios
```

EditText en el que el usuario inserta comentarios adicionales

Constructor Detail

- *RegisterActivity*

```
public RegisterActivity()
```

Method Detail

- *onCreate*

```
protected void onCreate(android.os.Bundle savedInstanceState)
```

Mëtodo del ciclo de vida de la actividad que se invoca cuando ésta es creada y obtiene las referencias a todos los elementos de la interfaz

Overrides:

```
onCreate in class android.support.v7.app.AppCompatActivity
```

Parameters:

```
savedInstanceState - estado guardado de la actividad
```

- *registrar*

```
public void registrar(android.view.View view)
```

Método que se invoca cuando el usuario selecciona el botón para formalizar el registro y que comprueba todos los elementos introducidos para su posterior envío al servidor

Parameters:

```
view - vista del botón seleccionado
```

- *onRadioButtonClicked*

```
public void onRadioButtonClicked(android.view.View view)
```

Método que se invoca al pulsar cualquiera de los radio buttons de la interfaz para seleccionar el sexo del usuario

Parameters:

```
view - vista del RadioButton pulsado
```

- *logearse*

```
public void logearse(android.view.View view)
```

Método invocado cuando el usuario selecciona el enlace que redirecciona a la pantalla de inicio de sesión

Parameters:

```
view - vista del elemento seleccionado
```

- *mostrarCondicionesUso*

```
public void mostrarCondicionesUso(android.view.View view)
```

Método invocado al seleccionar el enlace para leer las condiciones y términos de uso de la aplicación y permite aceptarlas o no

Parameters:

```
view - vista del elemento pulsado
```

A.15. Clase ResponseException

```
public class ResponseException
extends java.lang.RuntimeException
```

Excepción que se genera cuando la respuesta recibida por el servidor indica que no ha habido éxito en alguna petición o cuando se genera algún error propio en la aplicación

Author:

Javier Pérez Ácimas

See Also:

Serialized Form

Field Detail

- *message*

```
private java.lang.String message
```

Información del error sucecido generada a partir del código de error

- *errorCode*

```
private int errorCode
```

Código del error sucedido

- *serverErrorInfo*

```
private java.lang.String serverErrorInfo
```

Información enviada por el servidor del error sucedido

- *activity*

```
private android.app.Activity activity
```

Actividad que genera la excepción

Constructor Detail

- *ResponseException*

```
public ResponseException(int errorCode,
                         java.lang.String serverErrorInfo,
                         android.app.Activity activity)
```

Constructor de la excepción cuando el error se recibe notificación de error por parte del servidor

Parameters:

`errorCode - código del error`

`serverErrorInfo - información del error enviada por el servidor`

`activity - actividad en la que se detecta el error`

- *ResponseException*

```
public ResponseException(int errorCode,
                         android.app.Activity activity)
```

Constructor de la excepción cuando el error se genera directamente en la aplicación móvil

Parameters:

`errorCode - código del error`

`activity - actividad que genera el error`

Method Detail

- *getMessage*

```
public java.lang.String getMessage()
```

Método que devuelve el mensaje asociado a la excepción

Overrides:

`getMessage in class java.lang.Throwable`

Returns:

`el mensaje asociado a la excepción`

- *getErrorCode*

```
public int getErrorCode()
```

Método que devuelve el código de error asociado a la excepción

Returns:

`el código de error asociado a la excepción`

- *getServerErrorInfo*

```
public java.lang.String getServerErrorInfo()
```

Método que devuelve la información del error que ha enviado el servidor

Returns:

```
la información del error que ha enviado el servidor
```

- *getActivity*

```
public android.app.Activity getActivity()
```

Método que devuelve la actividad en que se ha generado la excepción

Returns:

```
la actividad en que se ha generado la excepción
```

- *makeErrorMessage*

```
private void makeErrorMessage(int errorCode)
```

Método que genera el mensaje con la información del error a partir del código de error

Parameters:

```
errorCode - código del error
```

A.16. Clase SecondaryTask

```
public class SecondaryTask
extends
android.os.AsyncTask<java.lang.Void,java.lang.Integer,org.json.JSONObject>
```

Clase encargada de crear la tarea asíncrona que interactuará con el servidor en segundo plano para enviar y recibir información

Author:

Javier Pérez Ácimas

Field Detail

- *data*

```
private java.util.Map<java.lang.String,java.lang.String> data
```

Conjunto de los datos a partir de los cuales se crea el JSON que será enviado al servidor

- *activity*

```
private android.app.Activity activity
```

Actividad invocadora de la clase

Constructor Detail

- *SecondaryTask*

```
SecondaryTask(java.util.Map<java.lang.String,java.lang.String> data,
              android.app.Activity activity)
```

Parameters:

data – mapa que contiene los datos que se quieren transmitir al servidor

activity – actividad invocadora

Method Detail

- *onPreExecute*

```
protected void onPreExecute()
```

Invocado en el hilo de UI antes de que se ejecute la tarea. Este paso se usa normalmente para configurar la tarea, por ejemplo, al mostrar una barra de progreso en la interfaz de usuario.**Overrides:**

```
onPreExecute in
class android.os.AsyncTask<java.lang.Void,java.lang.Integer,org.json.JSO
NObject>
```

- *doInBackground*

```
protected org.json.JSONObject doInBackground(java.lang.Void... params)
```

Invocado en el hilo en segundo plano inmediatamente después de que onPreExecute () termine de ejecutarse. Este paso se usa para realizar un cálculo de fondo que puede llevar mucho tiempo. Los parámetros de la tarea asíncrona se pasan a este método. El resultado del cálculo debe ser devuelto por este método al último paso. También se puede usar publishProgress (Progress ...) para publicar una o más unidades de progreso. Estos valores se publican en el subproceso UI, en el método onProgressUpdate (Progreso ...).

En nuestro caso, se encarga de crear el objeto JSON que se envió al servidor y recoger la respuesta.

Specified by:

```
doInBackground in
class android.os.AsyncTask<java.lang.Void,java.lang.Integer,org.json.JSO
NObject>
```

Parameters:

```
params - parametros pasados a la tarea asíncrona
```

Returns:

```
el resultado de la tarea asíncrona
```

- *onPostExecute*

```
protected void onPostExecute(org.json.JSONObject response)
```

Invocado en el hilo de la interfaz de usuario una vez que finaliza el cálculo de segundo plano. El resultado del cálculo de segundo plano se pasa a este paso como un parámetro.**Overrides:**

```
onPostExecute in
class android.os.AsyncTask<java.lang.Void,java.lang.Integer,org.json.JSO
NObject>
```

Parameters:

```
response - resultado del método doInBackground()
```

- *onCancelled*

```
protected void onCancelled()
```

Método invocado cuando una tarea es cancelada manualmente con el método cancel()

Overrides:

```
onCancelled in
class android.os.AsyncTask<java.lang.Void,java.lang.Integer,org.json.JSO
NObject>
```

A.17. Clase SettingsActivity

```
public class SettingsActivity
extends android.preference.PreferenceActivity
implements android.content.SharedPreferences.OnSharedPreferenceChangeListener
```

Actividad encargada de manejar los distintos ajustes de la aplicación, en principio, duración y frecuencia de muestreo de las medidas

Author:

Javier Pérez Ácimas

Field Detail

- *KEY_PREF_TIME*

```
public static final java.lang.String KEY_PREF_TIME
```

Palabra clave encargada de identificar la preferencia de selección de la duración de las medidas

See Also:

Constant Field Values

- *KEY_PREF_FREC*

```
public static final java.lang.String KEY_PREF_FREC
```

Palabra clave encargada de identificar la preferencia de selección de la precisión de la velocidad calculada

See Also:

Constant Field Values

Constructor Detail

- *SettingsActivity*

```
public SettingsActivity()
```

Method Detail

- *onCreate*

```
public void onCreate(android.os.Bundle savedInstanceState)
```

Método encargado de crear la interfaz de la actividad de preferencias tras obtener los valores de las mismas

Overrides:

onCreate in class android.preference.PreferenceActivity

Parameters:

savedInstanceState - estado de la aplicación

- *onSharedPreferenceChanged*

- `public void onSharedPreferenceChanged(android.content.SharedPreferences sharedPreferences,`
  `java.lang.String key)`

  Método encargado de la detección del evento de modificación de alguna de las preferencias de la aplicación por parte del usuario

  Specified by:

  `onSharedPreferenceChanged in interface android.content.SharedPreferences.OnSharedPreferenceChangeListener`

  Parameters:

  `sharedPreferences - valores de las preferencias`

  `key - identificador de la preferencia modificada`

- *changeTime*

  `public void changeTime(android.content.SharedPreferences sharedPreferences)`

  Método encargado de modificar el resumen de la preferencia de tiempo de muestreo cuando ésta es modificada por el usuario

  Parameters:

  `sharedPreferences - valor de las preferencias de la aplicación`

- *changePrecision*

  `public void changePrecision(android.content.SharedPreferences sharedPreferences)`

  Método encargado de modificar el resumen de la preferencia de precisión en el muestreo cuando ésta es modificada por el usuario

  Parameters:

  `sharedPreferences - valor de las preferencias de la aplicación`

A.18. Clase Utils

```
public class Utils
extends java.lang.Object
```

Clase que contiene los literales para identificar recursos tanto en las comunicaciones cliente-servidor cómo dentro del propio cliente y algunas funciones útiles para la app

Author:

Javier Pérez Ácimas

Field Detail

- *TAG_DNI*

```
public static java.lang.String TAG_DNI
```

Cadena de texto asociada a la identificación de los DNI

- *TAG_PASS*

```
public static java.lang.String TAG_PASS
```

Cadena de texto asociada a la identificación de las contraseñas

- *TAG_EMAIL*

```
public static java.lang.String TAG_EMAIL
```

Cadena de texto asociada a la identificación de los emails

- *TAG_PHONE*

```
public static java.lang.String TAG_PHONE
```

Cadena de texto asociada a la identificación de los números de teléfono

- *TAG_GENDER*

```
public static java.lang.String TAG_GENDER
```

Cadena de texto asociada a la identificación de los géneros

- *TAG_ADDRESS*

```
public static java.lang.String TAG_ADDRESS
```

Cadena de texto asociada a la identificación de las direcciones

- *TAG_NAME*

```
public static java.lang.String TAG_NAME
```

Cadena de texto asociada a la identificación de los nombres

- *TAG_SURNAME*

```
public static java.lang.String TAG_SURNAME
```

Cadena de texto asociada a la identificación de los apellidso

- *TAG_BIRTH*

```
public static java.lang.String TAG_BIRTH
```

Cadena de texto asociada a la identificación de las fechas de nacimiento

- *TAG_REGISTER*

```
public static java.lang.String TAG_REGISTER
```

Cadena de texto asociada a la identificación de los registros

- *TAG_LOGIN*

```
public static java.lang.String TAG_LOGIN
```

Cadena de texto asociada a la identificación de los logeos

- *TAG*

```
public static java.lang.String TAG
```

Cadena de texto asociada a la identificación de otras etiquetas

- *TAG_ERROR*

```
public static java.lang.String TAG_ERROR
```

Cadena de texto asociada a la identificación de los errores

- *TAG_MEASURE*

```
public static java.lang.String TAG_MEASURE
```

Cadena de texto asociada a la identificación de las medidas

- *AC_EJE_X*

```
public static java.lang.String AC_EJE_X
```

Cadena de texto asociada a la identificación del eje X del acelerómetro

- *AC_EJE_Y*

```
public static java.lang.String AC_EJE_Y
```

Cadena de texto asociada a la identificación del eje Y del acelerómetro

- *AC_EJE_Z*

```
public static java.lang.String AC_EJE_Z
```

Cadena de texto asociada a la identificación del eje Z del acelerómetro

- *GY_EJE_X*

```
public static java.lang.String GY_EJE_X
```

Cadena de texto asociada a la identificación del eje X del giroscopio

- *GY_EJE_Y*

```
public static java.lang.String GY_EJE_Y
```

Cadena de texto asociada a la identificación del eje Y del giroscopio

- *GY_EJE_Z*

```
public static java.lang.String GY_EJE_Z
```

Cadena de texto asociada a la identificación del eje Z del giroscopio

- *AC_EJE_X_CALIBRACION*

```
public static java.lang.String AC_EJE_X_CALIBRACION
```

Cadena de texto asociada a la identificación del eje X del acelerómetro en las calibraciones

- *AC_EJE_Y_CALIBRACION*

```
public static java.lang.String AC_EJE_Y_CALIBRACION
```

Cadena de texto asociada a la identificación del eje Y del acelerómetro en las calibraciones

- *AC_EJE_Z_CALIBRACION*

```
public static java.lang.String AC_EJE_Z_CALIBRACION
```

Cadena de texto asociada a la identificación del eje Z del acelerómetro en las calibraciones

- *GY_EJE_X_CALIBRACION*

```
public static java.lang.String GY_EJE_X_CALIBRACION
```

Cadena de texto asociada a la identificación del eje X del giroscopio en las calibraciones

- *GY_EJE_Y_CALIBRACION*

```
public static java.lang.String GY_EJE_Y_CALIBRACION
```

Cadena de texto asociada a la identificación del eje Y del giroscopio en las calibraciones

- *GY_EJE_Z_CALIBRACION*

```
public static java.lang.String GY_EJE_Z_CALIBRACION
```

Cadena de texto asociada a la identificación del eje Z del giroscopio en las calibraciones

- *TAG_CALIBRATION_TIME*

```
public static java.lang.String TAG_CALIBRATION_TIME
```

Cadena de texto asociada a la identificación de la duración de la calibración

- *TAG_ID_USER*

```
public static java.lang.String TAG_ID_USER
```

Cadena de texto asociada a la identificación de los id's de usuario

- *TAG_MEASURE_TIME*

```
public static java.lang.String TAG_MEASURE_TIME
```

Cadena de texto asociada a la identificación de la duración de la medida

- *TAG_HISTORY*

```
public static java.lang.String TAG_HISTORY
```

Cadena de texto asociada a la identificación de las recuperaciones de historial

- *TAG_MEASURE_FREC_AC*

```
public static java.lang.String TAG_MEASURE_FREC_AC
```

Cadena de texto asociada a la identificación de las frecuencias del acelerómetro

- *TAG_MEASURE_FREC_GY*

```
public static java.lang.String TAG_MEASURE_FREC_GY
```

Cadena de texto asociada a la identificación de las frecuencias del giroscopio

- *TAG_MEASURE_DATE*

```
public static java.lang.String TAG_MEASURE_DATE
```

Cadena de texto asociada a la identificación de las fechas de realización de las medidas

- *TAG_SUCCESS*

```
public static java.lang.String TAG_SUCCESS
```

Cadena de texto asociada a la identificación de las respuestas del servidor

- *TAG_ID_MEDIDA*

```
public static java.lang.String TAG_ID_MEDIDA
```

Cadena de texto asociada a la identificación de los id's de las medidas

- *TAG_GET_MEASURE*

```
public static java.lang.String TAG_GET_MEASURE
```

Cadena de texto asociada a la identificación de la recuperación de medidas

- *TAG_LIST_MEASURES*

```
public static java.lang.String TAG_LIST_MEASURES
```

Cadena de texto asociada a la identificación de las listas de medidas

- *TAG_ERROR_CODE*

```
public static java.lang.String TAG_ERROR_CODE
```

Cadena de texto asociada a la identificación de los códigos de error

- *TAG_DELETE_MEASURE*

```
public static java.lang.String TAG_DELETE_MEASURE
```

Cadena de texto asociada a la identificación de un las eliminaciones de medidas

- *TAG_ID_MEDICO*

```
public static java.lang.String TAG_ID_MEDICO
```

Cadena de texto asociada a la identificación de los id's de los médicos

- *TAG_PATIENT_LIST*

```
public static java.lang.String TAG_PATIENT_LIST
```

Cadena de texto asociada a la identificación de la recuperación de listas de pacientes

- *TAG_PRIVILEGES*

```
public static java.lang.String TAG_PRIVILEGES
```

Cadena de texto asociada a la identificación de los privilegios de los usuarios

- *TAG_ID_PATIENT*

```
public static java.lang.String TAG_ID_PATIENT
```

Cadena de texto asociada a la identificación de los id's de los pacientes

- *TAG_USER_DATA*

```
public static java.lang.String TAG_USER_DATA
```

Cadena de texto asociada a la identificación de la recuperación de datos de los usuarios

- *TAG_GET_NOTIFICATIONS*

```
public static java.lang.String TAG_GET_NOTIFICATIONS
```

Cadena de texto asociada a la identificación de la recuperación de notificaciones

- *TAG_NOTIFICACIONES_PENDIENTES*

```
public static java.lang.String TAG_NOTIFICACIONES_PENDIENTES
```

Cadena de texto asociada a la identificación de notificaciones pendientes

- *TAG_NOTIFICATIONS_LIST*

```
public static java.lang.String TAG_NOTIFICATIONS_LIST
```

Cadena de texto asociada a la identificación de las listas de notificaciones

- *TAG_LINK_DOCTOR*

```
public static java.lang.String TAG_LINK_DOCTOR
```

Cadena de texto asociada a la identificación aosciaciones paciente-médico

- *TAG_AC_INFO*

```
public static java.lang.String TAG_AC_INFO
```

Cadena de texto asociada a la identificación de información del acelerómetro

- *TAG_GY_INFO*

```
public static java.lang.String TAG_GY_INFO
```

Cadena de texto asociada a la identificación de información del giroscopio

- *TAG_PATHOLOGY*

```
public static java.lang.String TAG_PATHOLOGY
```

Cadena de texto asociada a la identificación de patologias

- *TAG_INFO_EXTRA_PATIENT*

```
public static java.lang.String TAG_INFO_EXTRA_PATIENT
```

Cadena de texto asociada a la identificación de informacion extra de los usuarios

- *TAG_LINK_PATIENT*

```
public static java.lang.String TAG_LINK_PATIENT
```

Cadena de texto asociada a la identificación de asociaciones médico-paciente

- *TAG_EXPERIMENTAL_MEASURE*

```
public static java.lang.String TAG_EXPERIMENTAL_MEASURE
```

Cadena de texto asociada a la identificación de medidas realizadas por un experimentador

- *TAG_REJECT_DOCTOR*

```
public static java.lang.String TAG_REJECT_DOCTOR
```

Cadena de texto asociada a la identificación de rechazos paciente-médico

- *TAG_MEASURE_MODE*

```
public static java.lang.String TAG_MEASURE_MODE
```

Cadena de texto asociada a la identificación de protocolos de medida

- *TAG_NOMBRE_MEDICO*

```
public static java.lang.String TAG_NOMBRE_MEDICO
```

Cadena de texto asociada a la identificación de nombres de médicos

- *TAG_CONTACTO_MEDICO*

```
public static java.lang.String TAG_CONTACTO_MEDICO
```

Cadena de texto asociada a la identificación de los emails de los médicos

- *TAG_MEDICAL_CENTER*

```
public static java.lang.String TAG_MEDICAL_CENTER
```

Cadena de texto asociada a la identificación de los centros de los médicos

Constructor Detail

- *Utils*

```
public Utils()
```

Method Detail

- *isDniValid*

```
public static boolean isDniValid(java.lang.String dni)
```

Método que comprueba si una cadena de texto cumple las condiciones necesarias para ser un dni

Parameters:

```
dni - string con un posible dni
```

Returns:

```
true si el dni es correcto / false en caso contrario
```

- *isEmailValid*

```
public static boolean isEmailValid(java.lang.String email)
```

Método que comprueba si una cadena de texto cumple las condiciones necesarias para ser un correo electrónico

Parameters:

```
email - string con un posible email
```

Returns:

```
true si el email es correcto / false en caso contrario
```

- *isDateValid*

```
public static boolean isDateValid(java.lang.String date)
```

Método que comprueba si una cadena de texto cumple las condiciones necesarias para ser una fecha en formato MySQL [yyyy-mm-dd]

Parameters:

```
date - string con una posible fecha en formato MySQL
```

Returns:

```
true si la fecha es correcta / false en caso contrario
```

- *codificarPassword*

```
public static java.lang.String codificarPassword(java.lang.String password)
```

Método que obtiene un hash codificado de la contraseña a partir de un algoritmo elegido

Parameters:

```
password - password a codificar
```

Returns:

```
el hash generado
```

- *isConnected*

```
public static boolean isConnected(android.content.Context context)
```

Método que comprueba si el dispositivo móvil está conectado a la red

Parameters:

```
context - contexto de la actividad invocadora
```

Returns:

```
true si el dispositivo está conectado / false en caso contrario
```

- *quitaEspeciales*

```
public static java.lang.String quitaEspeciales(java.lang.String input)
```

Función que elimina acentos y caracteres especiales de una cadena de texto.

Parameters:

```
input - cadena de entrada
```

Returns:

```
cadena de texto limpia de acentos y caracteres especiales.
```

B. Apéndice II – Documentación del servidor Python

A lo largo de este apéndice se detalla toda la información relevante relativa a las variables y funciones de los módulos Python desarrollados para implementar el servidor.

B.1. Módulo database

```
# -*- coding: utf-8 -*-
# Fichero: database.py
# Autor: Javier Pérez Ácimas
# Descripción: Módulo encargado de conectar con la base de datos y
ejecutar las consultas necesarias para el tratamiento y recuperación
de los datos.
```

Modules

mysql.connector.errorcode mysql

Functions

generar_json_response(data, columns)

```
Función invocada cuando una consulta de tipo SELECT devuelv
e más de un registro y que se encarga de parsear
una tupla a string en formato json array {...},{...},{...}
con los nombres de las columnas de la tabla como claves

:param data: registros devueltos por la consulta
:param columns: columnas que componen el registro
:return: un string con varios json separados por comas (,)
```

run_query(query='')

```
Función que ejecuta un consulta sobre la base de datos

:param query: consulta a ejecutar
:return: para consultas SELECT el resultado de la consulta,
 para INSERT el número de la línea, para DELETE y
UPDATE el número de filas afectadas
```

Author

Javier Pérez Ácimas

B.2. Módulo gestionarMedidas

```
# -*- coding: utf-8 -*-
# Fichero: gestionarMedidas.py
# Autor: Javier Pérez Ácimas
# Descripción: Módulo encargado de los servicios relacionados con las
mediciones: grabación, recuperación, historiales…
```

Modules

Utils mysql.connector.errorcode mysql time

Functions

add_measure(json_data_measure)

Función invocada encargada de la grabación de una nueva medida en la BD.

Se invoca cuando llega al servidor una petición de inserción de nueva medida
:param json_data_measure: json recibido con la medida a insertar y sus datos asociados
:return: mensaje de éxito si se inserta la medida correctamente o de error si ocurre algún imprevisto

delete_measure(json_request_delete)

Función invocada cuando se solicita eliminar una medición

:param json_request_delete: json con la petición de borrado y el identificador de la medida
:return: mensaje de éxito si se elimina la medida correctamente o de error si ocurre algún imprevisto

get_measure(json_request_measure)

Función encargada de recuperar una medición concreta

Se invoca cuando el usuario elige una medida del historial y quiere verla gráficamente
:param json_request_measure: json con la petición del usuario que oontiene el identificador de la medida
:return: la información de la medida a recuperar

list_measures(json_request_history)

Función encargada de recuperar el historial de medidas de un usuario

Se invoca al recibir una petición de recuperación de historial
:param json_request_history: json recibido en el servidor con la información del usuario invocador
:return: la lista con el las fechas-horas en que el usuario realizó medidas

Author

Javier Pérez Ácimas

B.3. Módulo Log

```
# -*- coding: utf-8 -*-
# Fichero: Log.py
# Autor: Javier Pérez Ácimas
# Descripción: Clase encargada del registro de las interacciones entre
 clientes y servidor.
```

Modules

Utils json time

Classes

class Log

```
Clase que se encarga de almacenar logs en la base de datos con
 lo que recibe y envía el servidor
```

Methods defined here:

insertar_error(self, exception)

```
Función invocada cuando ocurre un error inesperado en
el servidor y que almacena dicho error en la tabla de
logs de error

:param exception: excepción recogida en el servidor
:return: None
```

insertar_log(self, request, client_address)

```
Función invocada cuando llega una petición al servidor
que almacena dicha petición y de quien proviene en
la tabla de logs

:param request: petición recibida
:param client_address: IP:puerto que envía la petición
:return: None
```

terminar_log(self, response)

```
Función invocada cuando el servidor contesta a una
petición que actualiza una entrada en la tabla de logs

:param response: respuesta enviada por el servidor
:return: None
```

Data and other attributes defined here:

log_id = 0

Author

Javier Pérez Ácimas

B.4. Módulo notificaciones

```
# -*- coding: utf-8 -*-
# Fichero: notificaciones.py
# Autor: Javier Pérez Ácimas
# Descripción: Módulo encargado de la generación y recuperación de not
ificaciones y de la gestión de los casos de uso que tratan con ellas.
```

Modules

Utils json time
mysql.connector.errorcode mysql

Functions

aceptar_medico(json_data)

```
Función invocada por un paciente para aceptar una
invitación de un médico de ser tutorado por él

:param json_data: json con los ids del paciente y con el id
del médico
:return: mensaje de éxito cuando se realizan las consultas
en la base de datos
```

obtener_notificaciones(json_data)

```
Función invocada para obtener las notificaciones pendientes
correspondientes a un usuario en concreto

Devuelve los id, asunto y usuario generadores de las
notificaciones de un usuario
:param json_data: json con el id del usuario del que se
buscan las notificaciones
:return: el json con los id, asunto y usuario generadores d
e las notificaciones de un usuario
```

rechazar_medico(json_data)

```
Función invocada por un paciente para rechazar una
invitación de un médico de ser tutorado por él

:param json_data: json con el id de la notificación de
petición que se va a rechazar
:return: mensaje de éxito si se realizan las consultas en
la base de datos
```

solicitar_anadir_paciente(json_data)

```
Función invocada por un médico para solicitar que un
paciente esté tutorado por el mismo

:param json_data: json con el id del médico que solicita y
el dni o el email del paciente solicitado
:return: mensaje de éxito si se realiza la solicitud o de
error si el paciente ya estaba solicitado
```

Author

Javier Pérez Ácimas

B.5. Módulo server

```
# -*- coding: utf-8 -*-
# Fichero: server.py
# Autor: Javier Pérez Ácimas
# Descripción: Módulo principal a la escucha de peticiones generadas
por los clientes y que las redirige al módulo apropiado en función del
servicio solicitado.
```

Modules

Utils	ssl	traceback
json	sys	urlparse

Classes

class GetHandler(BaseHTTPServer.BaseHTTPRequestHandler)

Method resolution order:

GetHandler
BaseHTTPServer.BaseHTTPRequestHandler
SocketServer.StreamRequestHandler
SocketServer.BaseRequestHandler

Methods defined here:

do_GET(self)

```
Función encargada del tratamiento de las peticiones GE
T al servidor

Se encarga de responder con un error 405
:return: null
```

do_POST(self)

```
Función encargada del tratamiento de las peticiones PO
ST al servidor

Se encarga de responder con un JSON al cliente
:return: null
```

Methods inherited from BaseHTTPServer.BaseHTTPRequestHandler:

address_string(self)

```
Return the client address formatted for logging.

This version looks up the full hostname using gethostb
yaddr(),
and tries to find a name that contains at least one do
t.
```

date_time_string(self, timestamp=None)

```
Return the current date and time formatted for a messa
ge header.
```

end_headers(self)

```
Send the blank line ending the MIME headers.
```

handle(self)

```
Handle multiple requests if necessary.
```

handle_one_request(self)
Handle a single HTTP request.

You normally don't need to override this method; see t
he class
__doc__ string for information on how to handle specif
ic HTTP
commands such as GET and POST.

log_date_time_string(self)
Return the current time formatted for logging.

log_error(self, format, *args)
Log an error.

This is called when a request cannot be fulfilled. By
default it passes the message on to log_message().

Arguments are the same as for log_message().

XXX This should go to the separate error log.

log_message(self, format, *args)
Log an arbitrary message.

This is used by all other logging functions. Override
it if you have specific logging wishes.

The first argument, FORMAT, is a format string for the
message to be logged. If the format string contains
any % escapes requiring parameters, they should be
specified as subsequent arguments (it's just like
printf!).

The client ip address and current date/time are prefix
ed to every
message.

log_request(self, code='-', size='-')
Log an accepted request.

This is called by send_response().

parse_request(self)
Parse a request (internal).

The request should be stored in self.**raw_requestline**;
the results
are in self.**command**, self.**path**, self.**request_version** a
nd
self.**headers**.

Return True for success, False for failure; on failure
, an
error is sent back.

send_error(self, code, message=None)
Send and log an error reply.

Arguments are the error code, and a detailed message.
The detailed message defaults to the short entry match
ing the
response code.

This sends an error response (so it must be called bef

```
        ore any
        output has been generated), logs the error, and finall
        y sends
        a piece of HTML explaining the error to the user.
```

send_header(self, keyword, value)

```
        Send a MIME header.
```

send_response(self, code, message=None)

```
        Send the response header and log the response code.

        Also send two standard headers with the server softwar
        e
        version and the current date.
```

version_string(self)

```
        Return the server software version string.
```

Methods inherited from SocketServer.BaseRequestHandler:

__init__(self, request, client_address, server)

Author

Javier Pérez Ácimas

B.6. Módulo usuarios

```
# -*- coding: utf-8 -*-
# Fichero: usuarios.py
# Autor: Javier Pérez Ácimas
# Descripción: Módulo encargado de los servicios relacionados con los
usuarios del sistema: login, registro, datos de usuario…
```

Modules

Utils	json	re
mysql.connector.errorcode	mysql	time

Functions

crear_paciente_experimental(jsonDataUser)

Función encarga de crear un paciente que se asocia a una única medida

La función se invoca cuando una medida se realiza desde un perfil de experimentador y se encarga
de registrar los datos del paciente en la tabla correspondiente para poder asociarlo
con su medida
:param jsonDataUser: json con la petición de inserción de medida desde el usuario experimentador
:return: el identificador del paciente creado

get_patients(json_data_user)

Función invocada ante una petición de recuperación de lista de pacientes.

Se encarga de recopilar la lista de los usuarios ligados a un médico y devolvérsela
a la aplicación
:param json_data_user: json con la petición de lista de pacientes
:return: la lista con los pacientes asociados al médico que la solicita

login(jsonDataUser)

Función invocada cuando un usuario intenta logearse en el sistema

Función encargada de comprobar si el usuario está registrado en el sistema, cuales son sus privilegios
y si la contraseña es correcta y devolver a éste sus identificadores
:param jsonDataUser: json con la petición de login del usuario y sus credenciales
:return: mensaje de éxito y sus identificadores si el login se realiza correctamente o mensaje de error
si algún dato es incorrecto

obtener_datos_medico(id_medico)

Función encargada de recuperar los datos de un médico (tabla de médicos) a partir de su identificador

```
        :param id_medico: identificador del médico a buscar
        :return: una cadena con los datos del médico
```
obtener_datos_medico_dni(dni)
```
        Función encargada de recuperar los datos de un médico
        (tabla de médicos) a partir de su DNI

        :param dni: dni del médico a buscar
        :return: un json con los datos del médico
```
obtener_datos_paciente(id_paciente)
```
        Función encargada de recuperar los datos de un paciente
        (tabla de pacientes) a partir de su identificador

        :param id_paciente: identificador del paciente a buscar
        :return: una cadena con los datos del paciente
```
obtener_datos_paciente_dni(dni)
```
        Función encargada de recuperar los datos de un paciente
        (tabla de pacientes) a partir de su DNI

        :param dni: dni del paciente a buscar
        :return: un json con los datos del paciente
```
obtener_datos_paciente_email(email)
```
        Función encargada de recuperar los datos de un paciente
        (tabla de pacientes) a partir de su email

        :param email: email del paciente a buscar
        :return: un json con los datos del paciente
```
obtener_datos_usuario(jsonDataUser)
```
        Función invocada para recuperar la información de un sistema

        Función que se llama cuando un usuario quiere recuperar su
        información personal para
        visualizarla en la apllicación
        :param jsonDataUser: json con el identificador del usuario
        :return: la información del usuario correspondiente
```
obtener_datos_usuario_dni(dni)
```
        Función encargada de recuperar los datos de un usuario
        (tabla de usuarios) a partir de su DNI

        :param dni: dni del usuario a buscar
        :return: un json con los datos del usuario
```
obtener_datos_usuario_email(email)
```
        Función encargada de recuperar los datos de un usuario
        (tabla de usuarios) a partir de su email

        :param email: email del usuario a buscar
        :return: un json con los datos del usuario
```
password_correcto(password1, password2)
```
        Función encargada de comprobar si dos contraseñas coinciden

        :param password1: constraseña recibida
        :param password2: contraseña de comparación
        :return: True si las contraseñas coinciden / False en caso c
        ontrario
```
register(jsonDataUser)
```
        Función invocada cuando un usuario intenta registrarse en el
        sistema.

        Se encarga de comprobar que los datos cumplen con el formato
        y las condiciones adecuadas y, en caso
```

```
        afirmativo, de almacenarlos en la BD
        :param jsonDataUser: json con los datos del usuario a
        registrar
        :return: mensaje de éxito y los identificadores del usuario
        si el registro es correcto o mensaje de error si algún
        dato es incorrecto
```

Author

Javier Pérez Ácimas

B.7. Módulo Utils

```
# -*- coding: utf-8 -*-
# Fichero: Utils.py
# Autor: Javier Pérez Ácimas
# Descripción: Clase que contiene los literales para identificar
recursos tanto en las comunicaciones cliente-servidor cómo dentro del
propio servidor y algunas funciones útiles para el sistema.
```

Functions

addInt(json_string, key, int_value)

```
Función encargada de añadir un elemento de tipo integer a u
n cada de texto con formato JSON

:param json_string: cadena con formato json a la cual se va
 a añadir el nuevo elemento
:param key: identificador del integer que se va a añadir al
 json
:param int_value: valor tipo integer que se va a añadir al
json
:return: la cadena de texto con formato json formada al aña
dir el integer a la cadena inicial
```

addString(json_string, key, string_value)

```
Función encargada de añadir un elemento de tipo string a un
 cada de texto con formato JSON

:param json_string: cadena con formato json a la cual se va
 a añadir el nuevo elemento
:param key: identificador del string que se va a añadir al
json
:param string_value: valor tipo string que se va a añadir a
l json
:return: la cadena de texto con formato json formada al aña
dir el string a la cadena inicial
```

createJsonWithInt(key, int_value)

```
Función que crea un cadena de texto con formato JSON a part
ir de una clave y un integer

:param key: clave que identifica el integer contenido en el
 json
:param int_value: integer contenido en el json
:return: la cadena de texto con formato json creada
```

createJsonWithString(key, string_value)

```
Función que crea un cadena de texto con formato JSON a part
ir de una clave y una cadena de texto

:param key: clave que identifica la cadena contenida en el
json
:param string_value: cadena de texto contenida en el json
:return: la cadena de texto con formato json creada
```

deunicodify_hook(pairs)

```
Función que convierte los datos cargados de un json de form
ato unicode a formato utf-8 para que no
se generer errores al formar la cadena

:param pairs: clave valor del json generado
```

```
        :return: los mismos datos recibidos pero como str en vez de
         u'
```

response_error(mensaje_error, codigo_error)

```
        Función que genera un JSON con una respuesta de "éxito no a
        lcanzado" a la aplicación móvil

        Se incluye un 0 en la etiqueta de éxito, se añaden un códig
        o y un mensaje identificando al error
        :param mensaje_error: mensaje informativo acerca del error
        sucedido
        :param codigo_error: código que identifica el error sucedid
        o
        :return: la respuesta errónea en formato json generada
```

Author

Javier Pérez Ácimas