UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería de la Salud

Desarrollo de un sistema de rehabilitación telemático de bajo coste

Development of a low-cost telematic rehabilitation system

Realizado por
Fernando Saldaña González

Tutorizado por
Francisco Javier Mata Contreras

Departamento
Ingeniería de Comunicaciones

MÁLAGA, junio de 2022

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA DE LA SALUD

Desarrollo de un sistema de rehabilitación telemático de bajo coste

Development of a low-cost telematic rehabilitation system

Realizado por
Fernando Saldaña González

Tutorizado por
Francisco Javier Mata Contreras

Departamento
Ingeniería de Comunicaciones

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2022

Fecha defensa: julio de 2022

Abstract

Telemedicine is becoming more and more relevant in people's lives. An example of this is the system developed carried out in this project, which exposes a system for measuring, managing and displaying the electromyogram of a rehabilitation patient with or without reduced mobility. This system has been designed in three modules that communicate with each other. The acquisition of the biological signal is carried out by an Arduino Uno architecture together with the e-Health Senson Platform V2.0 extension. The information obtained is sent to a mobile application through a Bluetooth system based on the HC-06 module compatible with Arduino and Android. The application has been implemented using Android Studio development environment. This application communicates with a server that contains a database implemented using PostgreSQL. The communication between both has been implemented using distinct HTTP methods. The database stores patient and professional information as well as the results of the exercises performed. The functions available to the end user are the measurement and visualisation of the electromyogram resulting from an exercise prescribed by a medical professional.

Keywords: Android, Arduino, Bluetooh, electromyogram (EMG), PostgreSQL

Resumen

La telemedicina se hace cada día más notable en la vida de las personas. Un ejemplo de ello es el desarrollo realizado en el presente Trabajo Fin de Grado, en el que se expone un sistema de toma, gestión y visionado del electromiograma de un paciente de rehabilitación con o sin movilidad reducida. Se ha diseñado un sistema en tres módulos que se comunican entre ellos. La adquisición de la señal biológica la realiza una arquitectura Arduino Uno junto a la extensión e-Health Senson Platform V2.0. La información obtenida es enviada a una aplicación móvil a través de un sistema Bluetooth basado en el módulo HC-06 compatible con Arduino y Android. La aplicación ha sido implementada usando el entorno de desarrollo Android Studio. Esta aplicación se comunica con un servidor que contiene una base de datos. La comunicación entre ambos se ha implementado haciendo uso de distintos métodos HTTP. La base de datos guarda información de pacientes y profesionales, así como los resultados de los ejercicios realizados. Las funciones disponibles al usuario final son la medición y el visionado del electromiograma resultante de realizar un ejercicio prescrito por un profesional médico.

Palabras clave: Android, Arduino, Bluetooh, electromiograma (EMG), PostgreSQL

Índice

1 Introducción

A continuación, se describen tanto la motivación y objetivos referentes al Trabajo Fin de Grado (TFG), como su organización y temporización. De igual forma, se muestra la estructura del presente documento y sus diferentes apartados.

También se incluyen diferentes subcapítulos donde se presentan las ideas básicas para la comprensión de las herramientas utilizadas en el desarrollo del TFG. Se describen, de manera muy breve, las características básicas de los sistemas basados en arquitectura Arduino, así como sus extensiones e-Health y HC 06 (Bluetooth) utilizadas para la captura de las señales biométricas y la comunicación inalámbrica. Por otro lado, se resume el entorno de desarrollo Android Studio, que permite la programación orientada a dispositivos móviles con Sistema Operativo Android, y se introduce al lenguaje de programación Clojure, usado en el lado del servidor. Finalmente, se mencionan de manera simplificada las características fundamentales de la bioseñal de interés para el TFG: El electromiograma (EMG).

1.1. Motivación y Objetivos

En la actualidad, y más concretamente en el ámbito de pandemia que estamos viviendo, se ha instaurado una situación de contacto casi nulo y de actividades principalmente telemáticas. Bajo este marco, el desarrollo de los trabajos de los profesionales a cargo de nuestra salud se ha complicado notablemente, además de suponer un nuevo riesgo para estos mismos. De igual manera, los pacientes no reciben la misma seguridad y calidad que se merecen debido a las restricciones que se han impuesto a lo largo de todo el territorio español.

Desde el punto de vista del sector de la fisioterapia y la rehabilitación, donde el contacto con pacientes de movilidad reducida resulta imprescindible para el desarrollo normal de las actividades, la pandemia y las restricciones de libertad de movimiento han resultado ser un golpe muy duro. Es aquí donde nace la idea de un sistema de rehabilitación telemática que, evitaría contactos y desplazamientos de pacientes además de dar cierta libertad a los profesionales del sector ahora que más la necesitan.

El sistema propuesto, por tanto, sería capaz de recolectar información precisa de los ejercicios realizados por los pacientes y del desempeño de los mismos durante el proceso. Así mismo, esta información se haría llegar al profesional al cargo de la rehabilitación con la idea de que pueda ver los resultados de la sesión y proporcionarle nuevas instrucciones acorde a los datos obtenidos.

Para llevar esta propuesta a cabo se hace uso de varias facetas pertenecientes a la Titulación de Grado de Ingeniería de la Salud, principalmente en su Mención de Ingeniería Biomédica. Haremos uso, por tanto, de los conceptos de bases de datos, el diseño y uso del hardware electrónico (Arduino), las bases de programación orientada a objetos, y la adquisición y tratamiento de la seña biológica de nuestro estudio (electromiograma), así como su origen y características.

1.2. Organización del TFG y estructura de la memoria

La realización del presente trabajo se ha basado en las fases que se comentaban en el anteproyecto del mismo y que listaremos de nuevo a continuación:

- Estudio del estado actual del tema del TFG y realización de una lista con fuentes de información útiles y fiables.

- Desarrollo del sistema de adquisición de señales de EMG.
- Desarrollo del sistema software que almacenará y procesará la información obtenida.
- Desarrollo de un método de conexión entre ambos sistemas.
- Integración del trabajo realizado, verificación de su funcionamiento y pruebas.
- Búsqueda de posibles mejoras (e implementación si procede).
- Redacción de la memoria.

Evidentemente, estas fases se han ido haciendo de manera simultanea y no necesariamente en el orden en que se han presentado, pero siguiendo cierta cronología lógica (no es posible buscar mejoras de algo que aun no ha sido implementado).

Respecto a la estructura del presente documento, en este se muestra un resumen del proceso de diseño. Comenzando primero por el capítulo introductorio en el que se desarrollan la motivación, los objetivos, la organización y la información general y principales características de las tecnologías que vamos a usar, así como una pequeña redacción sobre el electromiograma y su procesado.

En el segundo capítulo se presenta el diseño y montaje del sistema de recogida de medias EMG basado en Arduino y e-Health. Se detallan las conexiones y programación necesarios para la captura de datos.

En el tercer capítulo se desarrolla la estructura básica del sistema servidor que va a recibir las medidas realizadas por nuestro sistema y que será el encargado de persistir a la base de datos. Se describen los métodos más significativos del código y las funciones que cumplen.

El cuarto capítulo se centra en el desarrollo de una aplicación móvil en el entorno Android Studio. Se describe la estructura general de la aplicación junto con sus principales funcionalidades y los aspectos más importantes de su programación. Esta aplicación servirá como conexión entre los distintos bloques (Arduino y servidor). Tras esto, podríamos considerar que nuestro sistema está en su primera forma funcional.

Por último, en el quinto capítulo se comentan las principales conclusiones tras el proyecto y se adelantan algunas mejoras y líneas interesantes que explorar a partir del punto en que ha finalizado el trabajo.

1.3. Tecnologías usadas y conceptos básicos

1.3.1. Arduino

Arduino es una plataforma de desarrollo basado en hardware y software libre que consiste en un microcontrolador fácil de programar y reconfigurable en un instante. Nace en Italia en 2005 bajo la idea de ofrecer una manera sencilla y barata de crear dispositivos capaces de interactuar con el medio mediante el uso de sensores y actuadores.[1] El proyecto consta de un sistema hardware (placa de componentes) y un software de desarrollo (IDE) con el que programarlo. Aunque hay una gran variedad de hardware Arduino disponible, nosotros siempre nos referiremos al Arduino Genuino UNO y lo llamaremos meramente Arduino (se especificará cuando hablemos del proyecto Arduino).

Respecto al sistema hardware, tal y como se muestra en la figura 1, no es más que un microcontrolador con varias entradas y salidas tanto analógicas como digitales y un puerto de comunicación USB con el que se conectará al ordenador. Esto es esencial, pues de esta forma es como se le transmitirán las instrucciones a ejecutar al sistema, y este podrá mandarnos su respuesta. Además, es el principal método de alimentación del Arduino.

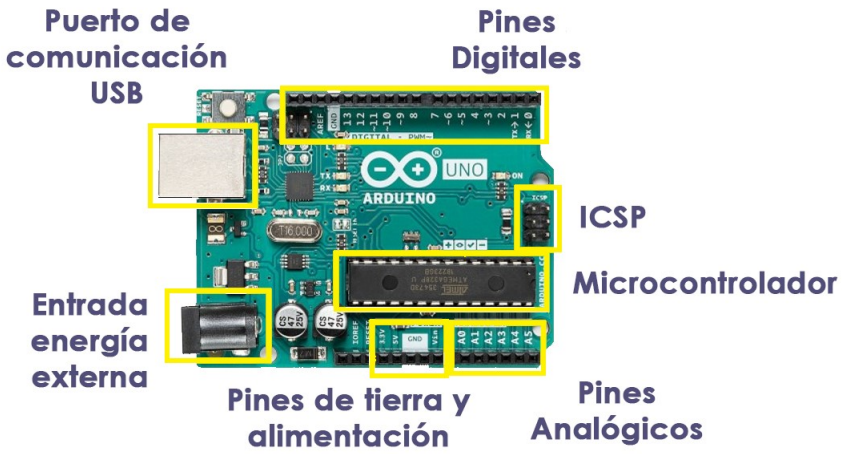

Figura 1: Componentes del Arduino UNO

Esto es lo básico para trabajar, sin embargo, existen placas con muchas más funcionalidades, así como módulos y "Shields" que permiten mejorar una placa básica a algo más complejo sin necesidad de tener que comprar una nueva. Estos Shields no son más que extensiones que se conectan a los pines de la placa básica dejando al descubierto un sistema más completo

donde realizar tareas más específicas. [2]

Como decíamos antes, vamos a trabajar con el Arduino UNO, el cual es el considerado el standard básico. Esta es la versión más usada y verificada y por ello hay una gran disponibilidad de información sobre la misma, así como una gran comunidad que comparte su experiencia. Entre sus características principales encontramos: [3]

- Microcontrolador ATmega328P de la familia Atmel ARM.
- 14 pines digitales que pueden ser usados tanto de entrada como de salida y de los cuales 6 poseen la capacidad de usar modulación de pulso (PWM).
- 6 pines analógicos que servirán sólo como entradas.
- Reloj 16 MHz.
- Puerto de comunicación USB.
- ISCP (In Circuit Serial Programing) para programar el BootLoader del microcontrolador.
- Botón de reset.
- Entrada de alimentación externa (hasta 12V).
- Pines de tierra (GND) y voltaje (de 3.3 y 5 voltios).

Respecto al software, el cual se muestra en la figura 2, el entorno de desarrollo puede encontrarse en la web de Arduino (www.arduino.cc) bien para su descarga e instalación o bien en formato online. Ambas opciones son de acceso gratuito y de uso intuitivo. Además, los drivers de cada placa se instalan automáticamente tras su primera conexión al ordenador.

Al abrir el IDE se nos muestra un editor de textos sencillo en el que es visible un sketch, pero antes de entrar en esto es necesario señalar al entorno qué tipo de placa estamos usando y en qué puerto está conectada (Herramientas->Placa y Herramientas->Puerto). Volviendo al sketch, podemos diferenciar varias zonas:

- Importación de librerías y declaración de variables y funciones. Las librerías a importar pueden ser todas las que haya instaladas, ya sean ofrecidas por el propio entorno de desarrollo o creadas por terceros que las distribuyen.

- Función setup(). Este bloque contiene todas las instrucciones que se van a ejecutar una única vez justo al cargar el programa en la placa (o al reiniciarla usando el botón).

- Función loop(). Aquí estará el grueso del programa, que consistirá de una serie de instrucciones ejecutadas secuencialmente en bucle de forma indefinida una vez finalizada la función setup().

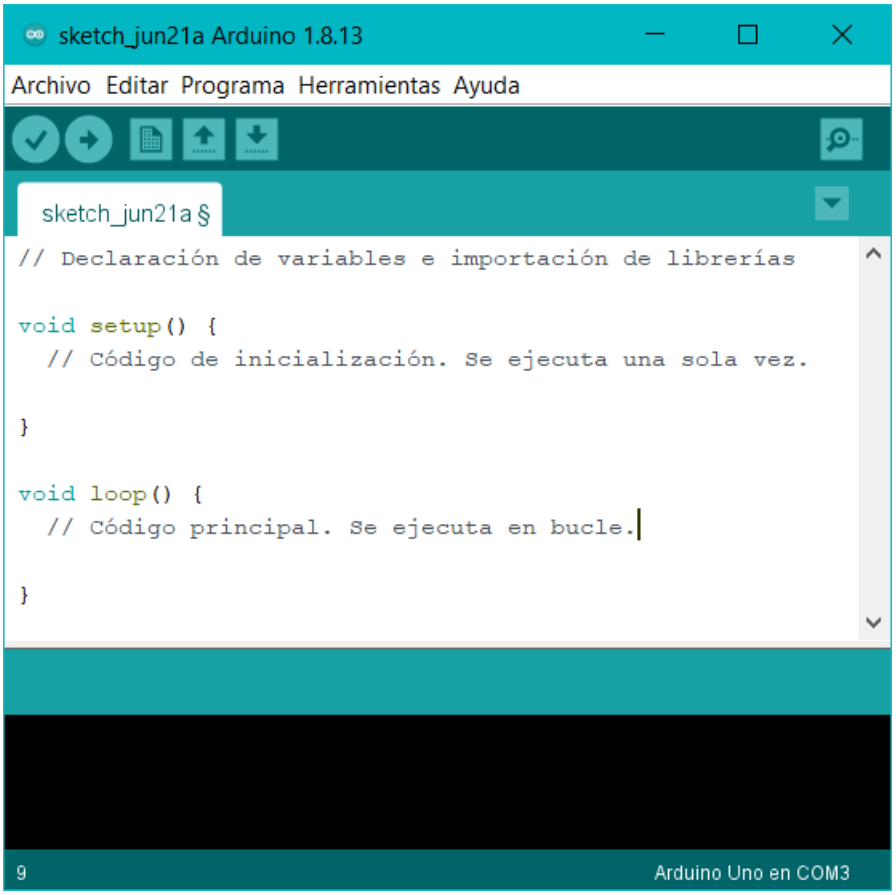

Figura 2: Estructura básica de un sketch Arduino

1.3.2. E-Health

A la hora de buscar el complemento perfecto para Arduino con vistas a tomar las medidas del EMG para el presente TFG surgieron numerosas extensiones que permitían ampliar las funcionalidades del mismo. Una de ellas y con la que hemos decido quedarnos es el Shield e-Health.

En concreto, se ha utilizado el e-Health Sensor Platform V2.0 de Cooking Hacks, la marca de Libelium que comercializa los e-Health. A día de hoy existe una versión mejorada llamada MySignals que no necesita ningún accesorio externo como Arduino o Raspberri Pi para proporcionar todas las funcionalidades de las que dispone, sin embargo, se descartó su uso por su elevado precio y el poco tiempo disponible para la elaboración del proyecto.

El e-Health Sensor Platform V2.0 admite 9 sensores, tanto digitales como analógicos, que incluyen: temperatura corporal, flujo de aire al respirar, respuesta galvánica de la piel, posición del paciente, medición del pulso y el oxígeno en sangre (SPO2), medidor de glucosa, presión arterial, electrocardiograma (ECG) y electromiograma (EMG) [4]. Las conexiones de estos sensores a la placa se pueden ver en la figura 3.

El equipo se compone por la placa o shield, la cual contiene el acondicionamiento eléctrico para la señal, y los distintos sensores. Para alimentar el circuito se usa la señal eléctrica enviada por el Arduino (o Raspberri Pi) al que esté conectado. Si la alimentación de este es el puerto USB de un ordenador puede no ser suficiente para el correcto funcionamiento teniendo que recurrir a una fuente de alimentación externa (señal DC de 12V/2A). En el caso del sensor EMG y el ordenador usado no ha sido necesario añadir una alimentación externa.

Una gran ventaja de esta extensión es que te permite seguir usando muchos de los pines y conectores de entrada y salida del Arduino, lo que habilita e incita al uso de los mismo con la idea de añadir nuevas funcionalidades a nuestro sistema. Además, el componente e-Health Sensor Platform V2.0 da la opción de conectar una pantalla LCD con la que mostrar los resultados, sin embargo, no se usará este añadido en el desarrollo de nuestra actividad.

En el presente TFG nos centraremos en el sensor EMG, que tal y como se demuestra en estudios ya realizados, presenta una buena precisión y nos da resultados en los que se puede confiar [5]. La extensión que estamos presentando sólo tiene una entrada para el sensor EMG, por lo que sólo se podrá realizar una medida a la vez. Además, posee un puente en el que se debe indicar si la medida entrante es de ECG o de EMG, por lo que no se podrán usar ambos simultáneamente (en nuestro caso no hay problema, pues sólo usamos el electromiogama).

El resto de información y detalles específicos sobre el e-Health Sensor Platform V2.0 necesarios para el desarrollo se proporcionarán más adelante en el presente documento.

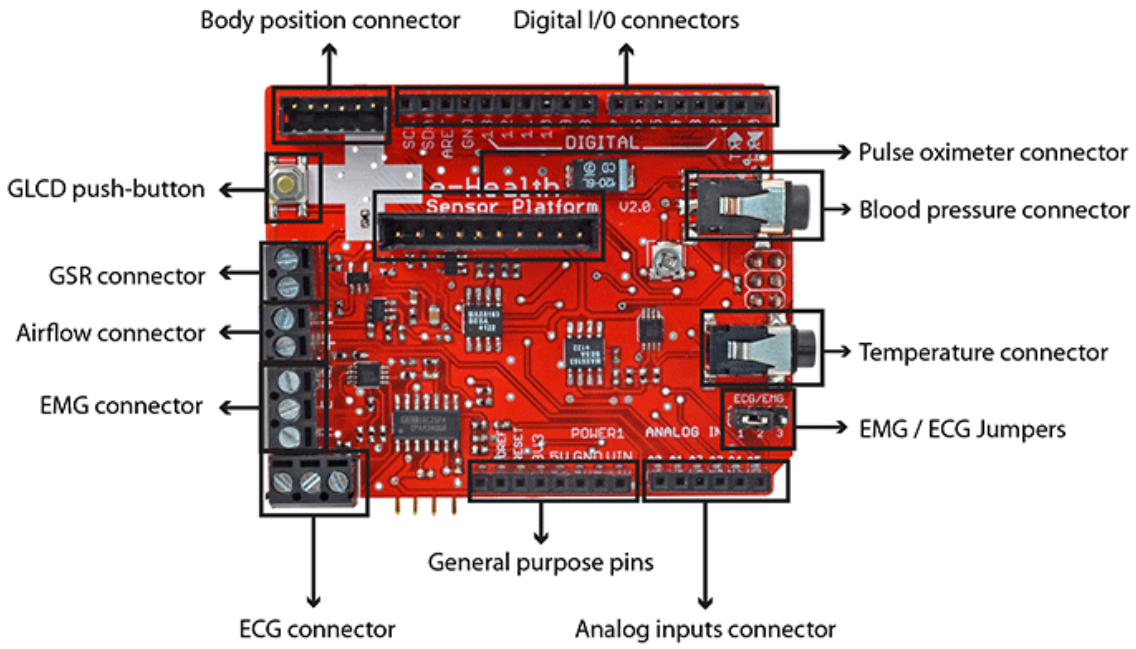

Figura 3: Conexiones disponibles del módulo e-health

1.3.3. HC-06 (Bluetooth)

El módulo Bluetooth usado en el proyecto es el HC-06 de DSD TECH, el cual se muestra en la figura 4. Es un módulo clásico, solamente compatible con Android, cuyo fin será el de funcionar como el puerto serie del Arduino, pero de manera remota. La información que normalmente se escribe en el puerto serie llegará al módulo Bluetooth y este lo transmitirá a los dispositivos que estén conectados al mismo.

Por defecto, el HC-06 se ejecuta en modo esclavo, y será el dispositivo interesado en comunicarse el que inicie la conexión. Por otro lado, el baudrate de la configuración predeterminada es de 9600 y la clave de conexión 1234. Todos estos valores pueden cambiarse usando los comandos AT. Para ello, DSD TECH proporciona un programa sencillo con el que realizar estos cambios. Sin embargo, es posible hacerlo mediante un sketch (este es el nombre que se le da a los programas desarrollados en Arduino).

Por último, mencionar que funciona con una alimentación de 3.6 a 6 voltios, por lo que la alimentación que proporciona el propio Arduino será más que suficiente para hacerlo funcionar.

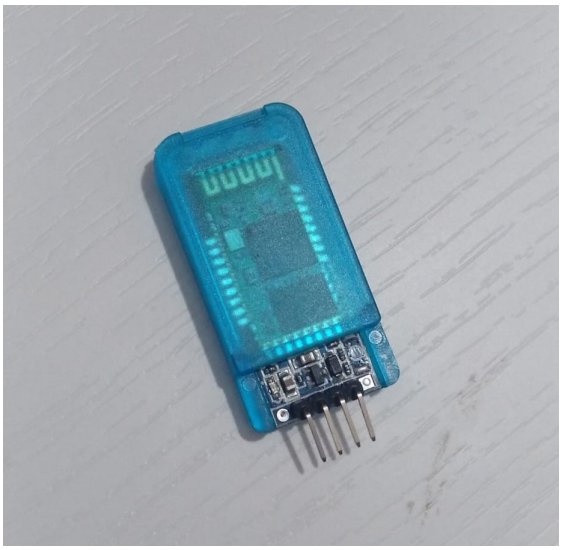

Figura 4: Módulo Bluetooth HC-06 de DSD TECH

1.3.4. Clojure, IntelliJ IDEA y Cursive

Clojure pertenece a una familia de lenguajes de programación Lisp, del inglés *List Processor* (procesamiento de listas), lo que lo hace tener un núcleo de lenguaje minúsculo, casi sin sintaxis, y una poderosa facilidad de macros. Sin embargo, Clojure representa un nuevo enfoque de Lisp manteniendo las ideas esenciales y mejorando la sintaxis para que sea más amigable para los programadores [6].

Se trata de un lenguaje compilado por la máquina virtual de Java (JVM), lo que proporciona un fácil acceso a sus marcos de trabajo, con sugerencias de tipo opcionales e inferencia de tipo.

Este lenguaje es predominantemente funcional, y cuenta con un rico conjunto de estructuras de datos inmutables y persistentes. Sin embargo, cuando se necesita un estado mutable, Clojure ofrece un sistema de memoria transaccional de software y un sistema de agente reactivo que permiten su tratamiento [7].

La estructura básica del código de Clojure viene a ser *(operador operandos)*. Las expresiones se evalúan dentro de paréntesis y siempre siguiendo el esquema dado, donde primero va la llamada a la función y detrás los parámetros. Algunos ejemplos simples se muestran en el cuadro **Ejemplos básicos de código Clojure**.

Ejemplos básicos de código Clojure

```
;; 1 + 2 = 3
(+ 1 2)

;; Definimos algunas variables:
(def entero  1 ) ; Variable 'entero' con valor 1
(def float  1.5 ) ; Variable 'float' con valor 1.5
(def string  "string" ) ; Variable 'string' con valor "string"
(def vector [:a 150 "hola"]) ; Un vector con varios valores
(def mapa {:a 150 :b "hola"}) ; Un mapa con varios valores

;; Definimos una función llamada 'hola'. Detrás del nombre va la
;; documentación, después los parámetros, y por último la implementación
(defn hola
  "Recibe cualquier tipo de parámetro y lo concatena para imprimirlo"
  [x]
  (println "Hola " x))
```

El entorno de desarrollo usado en este proyecto ha sido IntelliJ Idea (versión comunidad) junto con el complemento Cursive, el cual proporciona una integración completa con Clojure, permitiendo aprovechar hasta el último de sus recursos. Aunque Cursive necesita, por regla general, una licencia de pago, tiene una opción gratuita para proyectos no comerciales como pueden ser proyectos de código libre o trabajos de estudiantes.

Aunque Emacs (un editor de textos capaz de compilar código, entre otras funciones) es una de las opciones más populares al usar Clojure, el entorno elegido para el presente TFG proporciona todas las funcionalidades que vamos a necesitar y resulta mucho más familiar. En la figura 5 se muestra una captura en la que se puede distinguir la estructura básica:

- **Archivos del proyecto y ventana de cambios.** Se encuentra en la zona de la izquierda y tenemos tres pestañas en las que acceder rápidamente a los archivos del proyecto o bien a los cambios de nuestro código no registrados en el repositorio.

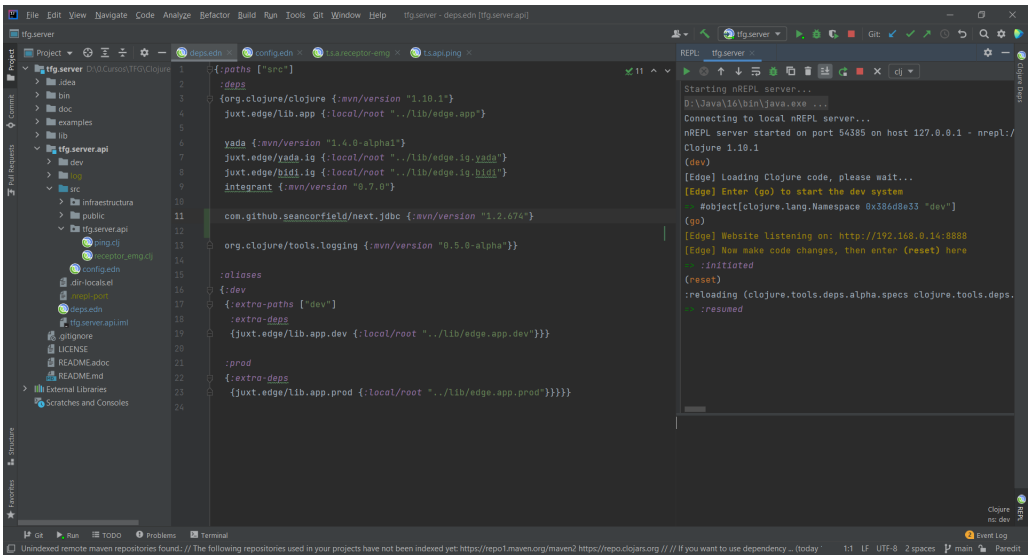

Figura 5: Estructura general del entorno de desarrollo

- **Editor de archivos.** Se encuentra en la parte central y en él podemos ver y editar cualquier archivo del proyecto, ya sea código o documentación.

- **REPL.** Del inglés *Read Eval Print Loop* (Leer Evaluar Imprimir Repetir). Se encuentra a la derecha y es la zona de ejecución. En el REPL podemos evaluar cualquier expresión en el mismo momento en que es escrita sin necesidad de recargar el proyecto por completo. Esto supone una gran comodidad a la hora de realizar el desarrollo.

Estos son los requisitos básicos para comprender la actividad desarrollada en el resto del TFG. Cualquier otro aspecto que sea importante para el entendimiento del mismo será explicada más adelante.

1.3.5. Android Studio

Para la conexión entre el sistema de captura de medidas y el servidor se ha decidido crear una aplicación móvil usando Android (dada la limitación del módulo bluetooth). Para ello, se ha empleado el entorno de desarrollo Android Studio, el cual está basado en IntelliJ Idea [8]. Este IDE posee un sistema de compilación flexible basado en Gradle. Esto nos permite usar un dispositivo Android para ejecutar y depurar directamente sobre él, vía conector USB, y también realizar desarrollos para todos los dispositivos Android en un entorno unificado.

Los programas desarrollados con Android Studio están compuestos por módulos que contienen tres bloques principales: Manifiestos, que consiste en el archivo AndroidManifest.xml; Carpeta Java, que contiene todos los archivos de fuente en java; y Carpeta Res, que contiene los recursos (diseños XML, imágenes, estilos, etc).

El AndroidManifest.xml contiene las principales características de la aplicación (versión del DSK de Java, actividades que contiene, librerías, configuraciones, permisos, etc). Se muestra un ejemplo en el cuadro **Muestra de AndroidManifest.xml**.

Muestra de AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:android="http://schemas.android.com/tools"
  package="nombre_del_paquete">

  <!--Declaración de permisos-->
  <uses-permission android:name="nombre del permiso" />

  <application
    <!--Características básicas (temas, icono, nombre...)-->
    android:nombreCaracterística="valor"
    <!--Declarar actividades-->
    <activity android:name=".NombreActividad" />
    </activity>
  </application>
</manifest>
```

En la carpeta Java se encuentra la implementación de todas las actividades que componen la aplicación. Cada una de ellas va a contener un número variable de funcionalidades, entre las cuales está la de mostrar otra actividad. En esta carpeta se declaran las clases de java que van a contener todo el código fuente de la aplicación. Es visible un ejemplo sencillo en el cuadro **Muestra de Clase Java**,

Muestra de Clase Java

```
package nombre_del_paquete;

// Importamos los paquetes y clases que se van a usar
import ...

// Definimos la clase, que puede ser pública o privada, y heredar de otra
public class NombreDeLaClase extends ClaseDeLaQueHereda {

    // Definimos variables y/o constantes públicas y/o privadas
    // No tienen por qué estar inicializadas
    private static final tipo nombre = valor;

    // Definimos los métodos públicos, privados y/o protegidos
    protected tipoQueDevuelve nombreFunción
    (tipoParametro nombreParametro){
        // Implementación
    }
}
```

Finalmente, en la carpeta Res se encuentran todos los recursos (imágenes, sonidos, estilos, textos...). Uno de los recursos más importantes de esta carpeta son los "Layouts", que codifican en XML el interfaz de usuario de cada una de las actividades. Android Studio también ofrece una interfaz gráfica para poder definir estos archivos, como se puede apreciar en el imagen 6.

Una vez creado el código, existen dos vías para probar su efectividad: usar el emulador que proporciona el propio entrono de desarrollo o activar la depuración USB de un dispositivo físico. A pesar de cumplir todos los requisitos para usar el emulador, para las pruebas se ha usado un dispositivo real, dado que el emulador no permite el uso de la tecnología Bluetooth.

Para configurar la depuración por USB del dispositivo se ha seguido la propia guía implementada en el entorno de desarrollo que proporciona los pasos a seguir y algunas comprobaciones de que se está conectando el teléfono de forma adecuada. Además, una vez registrado,

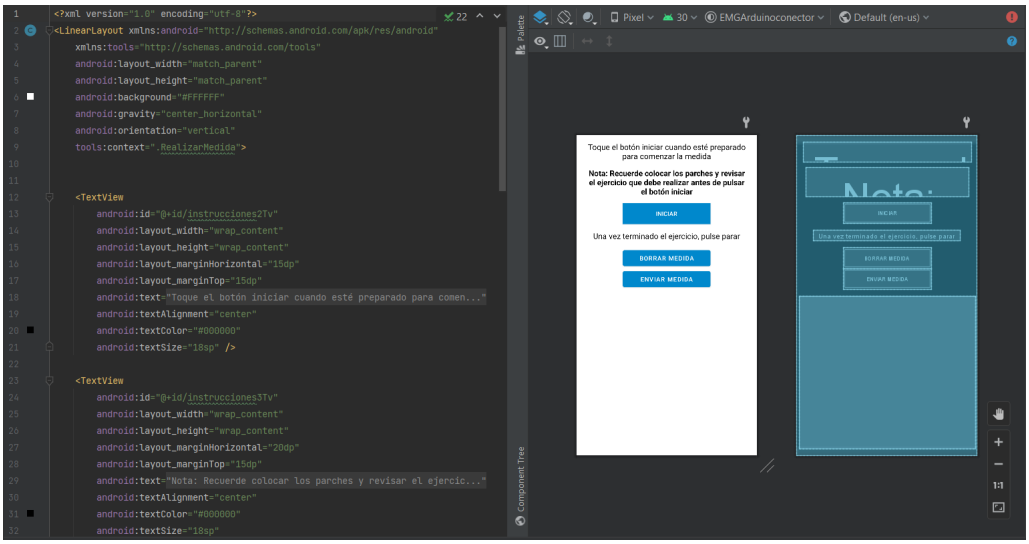

Figura 6: Muestra del interfaz de Android Studio para ficheros layout

no hará falta volver a pasar por este proceso.

1.3.6. Bioseñales. El electromiograma (EMG)

Para concluir este capítulo introductorio vamos a dar una breve descripción del concepto de bioseñal y a aclarar los conceptos básicos del electromiograma (EMG).

Según el diccionario de la Real Academia Española (RAE), una señal es una variación de una corriente eléctrica u otra magnitud que se utiliza para transmitir información. Las señales biomédicas, por tanto, son señales utilizadas en los campos de la biología y la medicina, principalmente con el fin de extraer información de un sistema biológico.

En el caso del electromiograma, mide una variación eléctrica producida al realizar una contracción o relajación muscular. Esta variación eléctrica se conoce como potencial de acción y se genera por las membranas celulares al conducir un impulso eléctrico. Cuando el impulso llega al músculo, se produce una contracción de las fibras, siendo esta mayor según el número de fibras que se activan o del número de potenciales que llegan a las mismas.

A la hora de realizar la medición se utilizan electrodos que pueden ser de aguja (método invasivo) o de superficie (método no invasivo). Estos electrodos funcionarán como transductores de señal, ya que el potencial eléctrico medido será provocado por acción iónica y la señal transmitida al sistema será por medio de electrones. En la figura 7 se muestra la representación gráfica de las medidas de ambos sensores, siendo la resultante del electro de superficie la

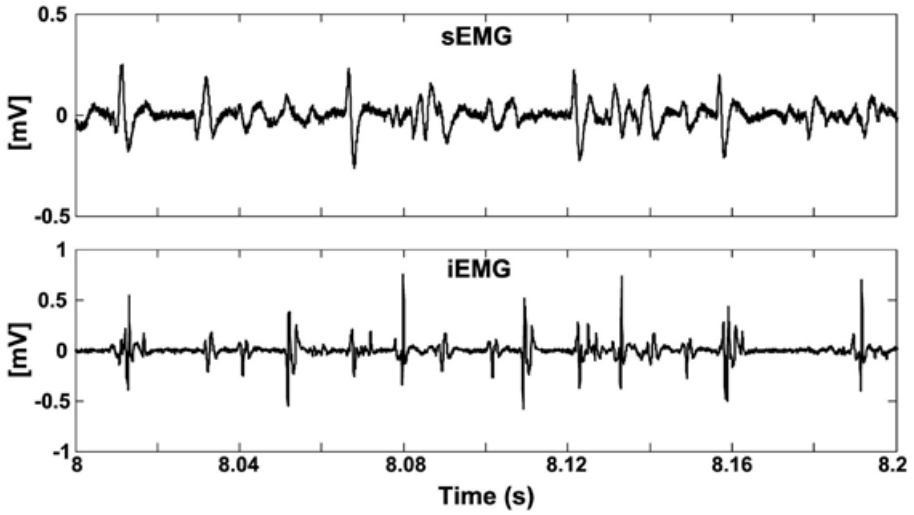

Figura 7: Representación de EMG superficial (sEMG) contra EMG invasivo (iEMG)[10]

etiquetada como sEMG y la resultante del electrodo de aguja la etiquetada como iEMG.

Para realizar una correcta medida, colocaremos un electrodo en una zona con poca musculatura que servirá como señal de referencia. Además, serán necesarios al menos dos electrodos más que harán la función de sensores. Por lo general, colocaremos el electrodo de referencia en la articulación más cercana al músculo a medir, y los electrodos sensores en la parte superior e inferior del mismo. La mayoría de las veces, en las aplicaciones biomédicas (como en muchas otras), la adquisición de la señal no es suficiente y resulta necesario procesar la señal adquirida para obtener la información relevante "enterrada" en ella [9]. A este proceso de "limpieza" de la señal se le conoce como acondicionamiento, y en el caso del electromiograma no es más que un filtrado de la señal que se recibe para evitar el ruido generado por la alimentación o por movimientos indeseados del electrodo.

Tras el acondicionamiento de la señal medida, se puede apreciar que los electrodos de aguja resultan ser muy precisos a la hora de medir las diferentes activaciones de las células motoras, siendo capaz de distinguir entre reclutamiento espacial (se activan varias fibras con un único impulso) y reclutamiento temporal (se activan más fibras con la llegada de más impulsos). Los electrodos de superficie, por el contrario, miden el campo generado al activar las fibras musculares, lo que les hace perder precisión frente a los de aguja. Sin embargo, resultan muy útiles a la hora de conocer cuándo se ha producido una contracción muscular y la intensidad de la misma.

2

Sistema de captura de señales electromiográficas

En este capítulo se recoge la información necesaria y los procesos a seguir para preparar el sistema de obtención de la señal biomédica de nuestro estudio (EMG). Se detallan las conexiones realizadas en los sistemas Arduino y e-Health y el código para la captura de la señal, así como una simple prueba para la comprobación de su correcto funcionamiento.

2.1. Estructura básica

Como bien se ha explicado ya, el sistema está principalmente compuesto por una placa Arduino y la extensión e-Health, la cual realizará la adquisición y acondicionamiento de la señal EMG.

El objetivo que vamos buscando en este primer hito será obtener las señales a través de la conexión serie que implementa Arduino al conectarlo por USB a un ordenador. Una vez recibida la escribiremos en un fichero con el fin de tener estos datos disponibles para el resto del proyecto.

2.2. Estructura del Hardware

El circuito de acondicionamiento que implementa el propio eHealth consta de cuatro etapas: amplificación diferencial, rectificado, filtrado activo y amplificación regulable.

La primera etapa amplifica la señal diferencial proveniente de los electrodos haciendo uso del amplificador AD8221 comercializado por Analog Device. La ganancia puede regularse mediante una resistencia.

La siguiente etapa es de rectificación de la señal. Se compone de un filtrado paso alto seguido del rectificador propiamente dicho. En función del signo de la señal de entrada, conducirá un diodo u otro invirtiendo o no dicha señal, de forma que a la salida tendremos la señal rectificada.

La tercera etapa es un inversor, con comportamiento paso bajo y ganancia unidad. Su función será eliminar todas las componentes de alta frecuencia resultantes del rectificador. El resultado del tratamiento de la señal hasta ahora se traduce en la envolvente, que agrupa la información de todas las fibras participantes en el movimiento muscular.

Finalmente se tiene un amplificador inversor de ganancia regulable que dependerá del valor de una resistencia.

A la hora de tomar la medida, la posición de los electrodos dependerá del músculo cuya actividad se va a medir. En el presente proyecto, se ha analizado el bíceps del brazo, por lo que los electrodos se han colocado como se muestra en 8.

Una vez colocados los electrodos, y teniendo conectada la extensión al Arduino y este al ordenador, todo lo que resta es comprobar que el puente está en la posición del EMG y lanzar

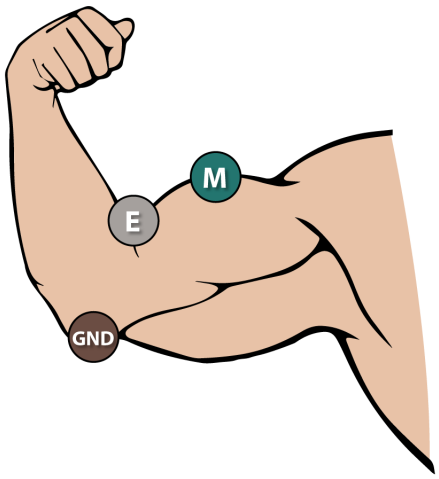

Figura 8: Emplazamiento de los electrodos [4]

el sketch que tomará la medida. En la figura 9 se muestran las conexiones a realizar.

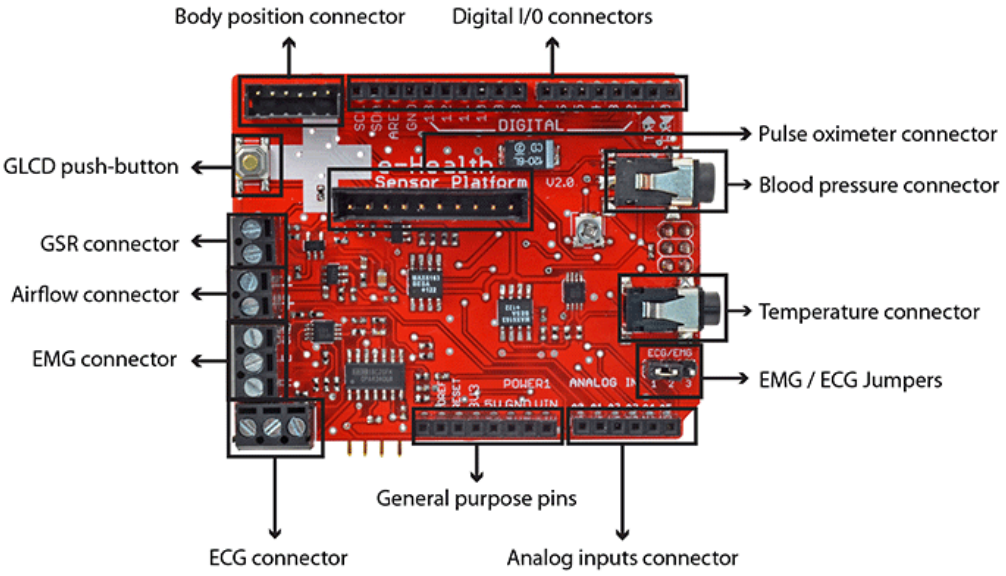

Figura 9: Conexiones del sistema Hardware básico

2.3. Estructura del Software

La programación por parte del Aduino es bastante sencilla al hacer uso de la librería propia que nos proporcionan en la web de la extensión e-Health. A pesar de su utilidad, esta librería sólo se puede utilizar en el Software Arduino 1.0.1, versión que a día de hoy resulta anticuada,

pero que es capaz de proporcionar todas las herramientas necesarias para el correcto funcionamiento del sistema.

El Sketch usado se muestra en el cuadro **Sketch Arduino para toma de medidas EMG**. En primer lugar se incluye la librería e-Health y se inicializa una variable contador a 0, la cual se irá incrementando conforme se reciban las medidas. El fin de este contador será fijar un valor máximo e igual para todas las medidas que hagamos.

En la función setup() iniciamos el puerto serie a 115200 baudios.

Por último, en la función loop() capturamos la señal del EMG de forma continuada hasta que el contador llegue al valor máximo elegido. Los datos son enviados por el puerto serie y se separan con un fin de línea y retorno de carro. Antes de iniciar el loop() de nuevo hay una espera de 10 ms.

Sketch Arduino para toma de medidas EMG

```
#include < eHealth.h >
int contador = 0;

void setup()  {
    Serial.begin(115200);
}

void loop() {
    if (contador < 1000){
        float EMG = eHealth.getEMG();
        Serial.println(EMG,2); // 2 decimales
        Serial.println("");
        contador++;
    }

    delay(10);
}
```

Por tanto, el Sketch descrito se está ejecutando en el sistema Arduino y, como resultado,

está enviando los datos al puerto serie (interfaz de comunicación del Arduino). En el modelo de Arduino usado, los pines receptor (Rx) y emisor (Tx) del puerto serie son los pines 0 y 1 respectivamente, y ambos están conectados por defecto con el cable USB. Es por ellos que al enviar las medidas al puerto serie, estas están siendo enviadas al ordenador por la conexión USB.

Estos datos son recibidos por un pequeño script (programa corto y sencillo creado a partir de pocas instrucciones) de python, el cual se está ejecutando en el ordenador al que está conectado el Arduino. Las medidas recibidas son procesadas y escritas en un fichero de texto.

De esta forma, se ha generado una colección de medidas con las que poder trabajar y probar los sistemas que se detallarán más adelante sin la necesidad de realizar el montaje a diario y ahorrar materiales al reducir el uso de los electrodos.

Script Python

```python
import serial

serial_port = 'COM3'
baud_rate = 115200
write_to_file_path = "output.txt"

output_file = open(write_to_file_path, "w+")
ser = serial.Serial(serial_port, baud_rate)
while True:
    line = ser.readline()
    line = line.decode("utf-8")
    print(line)
    output_file.write(line)
```

En el cuadro **Script Python** se muestra el código usado para guardar las medidas en ficheros. Para ello, se hace uso de la librería serial, la cual estará "monitoreando" la información recibida a través de la conexión USB para leer los datos que envía Arduino. En este script, se definen el puerto serie, el baudrate o baudaje, que es la tasa a la que se transmite la información, (igual que el del sketch Arduino) y el fichero donde vamos a escribir. Después se inicia

un bucle infinito que escribirá en el fichero de texto e imprimirá por pantalla cada valor que capture.

2.4. Pruebas y verificación

Las pruebas se llevaron a cabo mientras se ensamblaba el sistema final explicado a lo largo del presente capítulo. Estas se resumen en:

- Conexión del sistema Arduino al ordenador y comprobación del correcto funcionamiento del puerto serie con uno de los sketch de ejemplo del software.
- Repetición de la prueba anterior pero ahora con el shield e-health conectado. De esta forma comprobamos que el sistema no está bloqueando el puerto serie y que funciona correctamente.
- Toma de medidas EMG con un script Arduino básico que imprimía los resultados por pantalla. Con esta prueba se corroboró el correcto funcionamiento de la extensión e-Health.
- Por último, toma de medidas con el sistema completo. De esta manera se comprobó que tanto el código Arduino como el de Python funcionaban correctamente.

Los resultados recibidos de las pruebas fueron satisfactorios y, en general, demostraron ser de gran ayuda para asegurar que todos los componentes de nuestro sistema cumplían su cometido sin errores.

3

Desarrollo del servidor y la base de datos

En este tercer capítulo se recoge la descripción tanto de la base de datos (desarrollada con PostgreSQL) como del servidor (desarrollado con Clojure). La misión del servidor será la de recibir las medidas del EMG y guardarlas en la base de datos, donde el profesional podrá acceder a toda la información del paciente y a sus rutinas de ejercicios.

A continuación se detallarán la estructura del Software y sus principales funciones e implementaciones.

3.1. Base de datos

La base de datos será la encargada de almacenar la información más relevante relacionada con los usuarios del sistema. En ella guardaremos, por tanto, información de los profesionales médicos y de los pacientes a su cargo, así como los tratamientos (ejercicios) que el profesional considere oportunos y los resultados generados por el paciente.

3.1.1. Estructura de la base de datos

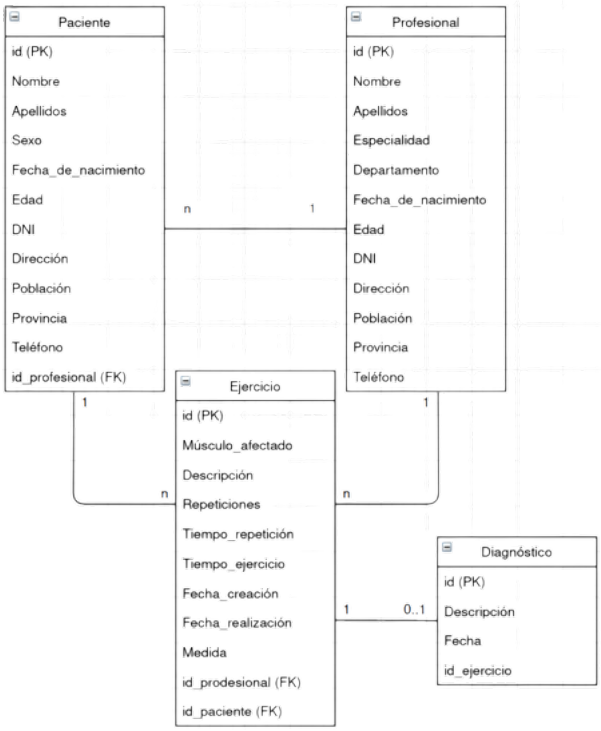

Figura 10: Estructura de la base de datos

Tal y como se muestra en la figura 10, se ha utilizado un modelo relacional en el que se encuentran 5 entidades. En primer lugar, se muestran las entidades del profesional y del paciente. En estas tablas se guardarán los datos de identificación básicos de cada persona, siendo comunes tanto para paciente como para profesional los campos nombre, apellidos, fecha de nacimiento, edad, DNI, dirección y teléfono.

Aunque ambas son muy parecidas, existe algún detalle específico que implementa cada una por su parte, como puede ser el sexo del paciente (Hombre/Mujer) o la especialidad del profesional (Enfermería, Fisioterapia...). La relación que mantienen es de 1 a n habiendo varios

pacientes por cada profesional pero un único profesional desde el punto de vista del paciente.

Estas dos entidades se relacionan con la entidad ejercicio, en la que guardamos el músculo a ejercitar, la descripción del ejercicio a realizar, el número de repeticiones, el tiempo que debe mantenerse el músculo tensionado cada repetición y el tiempo total aproximado que debe durar el ejercicio. También se incluyen la fecha en que el profesional creó el ejercicio y la fecha en que el paciente lo completó. la relación que mantiene con los pacientes y los profesionales es la misma (1 a n), de tal forma que cada profesional podrá crear varios ejercicios y cada paciente podrá realizar varios ejercicios.

Cada ejercicio, por su parte, debe guardar las medidas tomadas durante su realización. Para ello está el campo medida, en el que se guardan como un vector los resultados obtenidos durante la actividad física, evitando así la necesidad de crear una nueva tabla para guardarlas indexadas por el ejercicio al que pertenecen y el orden en el que fueron tomadas.

Por último, tenemos la tabla diagnóstico. En ella se guarda una breve descripción de la interpretación que realiza el profesional médico de los resultados del ejercicio junto con la fecha de publicación del diagnóstico. Destacar que cada ejercicio tendrá, como máximo, un único diagnóstico, así como no todos los ejercicios tienen por qué mantener un diagnóstico asociado.

De forma general, en cada tabla se ha creado una variable identificador que nos servirá como clave primaria de las distintas entidades, por lo que será único y se incrementará de forma automática. Aunque es posible evitar la creación de este id, dados los potentes sistemas informáticos que se comercializan a día de hoy, y el poco espacio que supone añadir un número extra a cada instancia, no es mala idea añadirlo y así tener más de una forma de identificar cada entidad haciendo uso de campos únicos (DNI o teléfono) o de la combinación de varios que resulten ser únicos (nombre y dirección).

El modelo está pensado para ser acoplado a la posible base de datos que hubiera en la clínica u hospital de destino, reduciendo así el número de entidades nuevas a las tres últimas: ejercicio, medida y diagnóstico, ya que las entidades de paciente y médico estarían creadas en los sistemas del destinatario.

3.1.2. Implementación de la base de datos

Para la creación y administración de la base de datos se ha hecho uso de PostgreSQL y DBeaver. PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos bastante potente. Es capaz de proporcionar una gran cantidad de herramientas con las que trabajar a la hora de montar nuestra base de datos. A diferencia de otros servicios parecidos, con este podemos incluso guardar variables con gran cantidad de información en una tabla (un JSON o un array, por ejemplo) sin estar limitados a los tipos clásicos (enteros, float, text...).

Para la creación de la base de datos se ha recurrido a un símbolo del sistema, el cual nos servirá de interfaz para usar nuestro sistema de gestión. En primer lugar, se debe usar el comando *psql -U postgres* con el que, tras introducir la contraseña especificada durante la instalación del software, iniciaremos el superusuario postgres. La razón de usar este usuario es que tiene todos los permisos. Sin embargo, se podría crear un nuevo usuario con los permisos necesarios para la creación y edición de bases de datos.

Una vez se ha iniciada la sesión, se utiliza el comando *CREATE DATABASE tfg;* para crear una base de datos llamada tfg, y a continuación podemos hacer una comprobación con *\l* para listar las bases de datos del sistema. En este listado debería poderse ver que la base de datos se ha creado correctamente, tal y como se puede apreciar en la figura 11.

Figura 11: Listado de las base de datos del sistema

Con la base de datos ya creada, se ha decidido configurar haciendo uso de BDeaver, que es una aplicación de software cliente de SQL y una herramienta de administración de bases de datos. Haciendo uso de este administrador, se crearán las tablas y se comprobará que el esquema resultante es como el definido. Para realizar la conexión con la base de datos, existe

la opción ”nueva conexión” donde se detallará la configuración pertinente (nombre, usuario, puerto...) y se realizará de forma automática.

A continuación, se añaden dos archivos a la carpeta scripts, la cual se puede apreciar en la esquina inferior izquierda de la ventana. Recordar que se debe de vincular estos archivos a la base de datos de interés, en el caso que nos atañe será la llamada tfg. Una vez creados y vinculados, se desarrollará en ellos el código para crear y borrar las tablas de nuestro modelo de datos.

Creación de la tabla ejercicio

```
create table ejercicio(
    id                serial primary key,
    musculo_afectado  text,
    descripcion       text,
    repeticiones      smallint,
    tiempo_repeticion smallint,
    tiempo_ejercicio  smallint,
    fecha_creacion    date,
    fecha_realizacion date,
    medida            float[],
    id_profesional    smallint,
    id_paciente       smallint,
    foreign key (id_profesional) references profesional (id),
    foreign key (id_paciente) references paciente (id));
```

En el código mostrado en la tabla **Creación de la tabla ejercicio** se puede apreciar un resumen de todo lo usado en el script de creación de tablas. Empezando por las claves primarias (id), las cuales son del tipo *serial*. Este tipo de dato representa un número entero que se irá incrementando en saltos de 1 de forma automática al añadir una fila nueva a la tabla. Gracias a esta definición, es posible obviar el insertar un valor a la hora de añadir nuevos datos.

El numero de repeticiones es de tipo *smallint*. Este tipo de dato es un número entero, pero está limitado a 2 bytes en lugar de a 4. Dado que es altamente improbable que un ejercicio requiera varios miles de repeticiones, este tipo es más que suficiente para cumplir con las

necesidades y además se consigue aprovechar un poco de espacio en el disco.

Por otro lado, el tipo *text* se trata de, tal y como indica su nombre, un texto (cadena de caracteres). El tipo *timestamp* representa una fecha con hora. Aunque no es realmente necesario conocer la hora exacta en que ocurre algo en nuestro sistema, resulta interesante saber las horas a las que el paciente realiza sus ejercicios con vista a la evaluación del médico, ya que los resultados no serán los mismos después de un día de trabajo que justo al despertar.

Quizá lo más llamativo del script es el tipo *float[]* usado para la medida. Este tipo es una array (vector) de valores reales. Esta estructura evita, como ya ha sido mencionado en el apartado anterior, crear una nueva tabla en la que guardar cada medida recibida por el sistema y relacionarla con el ejercicio.

Por último, aclarar que para crear claves ajenas es necesario primero crear la variable y luego hacer referencia a la variable de la tabla a la que pertenece. Esto se puede ver en el código al definir primero *idprofesional* y luego hacer referencia al id de la tabla profesional. Mientras tanto, las claves primarias se pueden definir escribiendo *primary key* justo detrás de la definición de la variable.

Una vez explicada en detalle la creación de la tabla ejercicio, no es necesario conocer nada más para comprender el resto del código del script de creación de tablas, el cuál se puede ver en el CD anexo.

3.2. Servidor

Para este proyecto, el servidor será la interfaz de la base de datos, por lo que funcionará como punto de acceso a la misma. Para llevar a cabo esta tarea, se ha decidido que implementará una serie de métodos http que que permitirán tanto consultar como grabar información.

Como ya se ha adelantado en la introducción, el servidor se ha implementado haciendo uso del lenguaje de programación Clojure. Más concretamente, se ha utilizado un proyecto ya creado conocido como *juxt/edge*, el cual representa la unión de distintos proyectos más pequeños de diversas índoles. Al usar este proyecto como base, podemos crear de forma muy fácil y rápida una aplicación servidor.

3.2.1. Estructura del servidor

La estructura de nuestro proyecto, la cual se puede ver en la figura 12, dependerá en cierto modo de la estructura del proyecto usado como base y del entorno de desarrollo. De esta manera, es posible diferenciar la estructura común a cualquier proyecto que haga uso de *juxt/edge*, las carpetas creadas por el entorno y la estructura propia del proyecto desarrollado en el presente TFG.

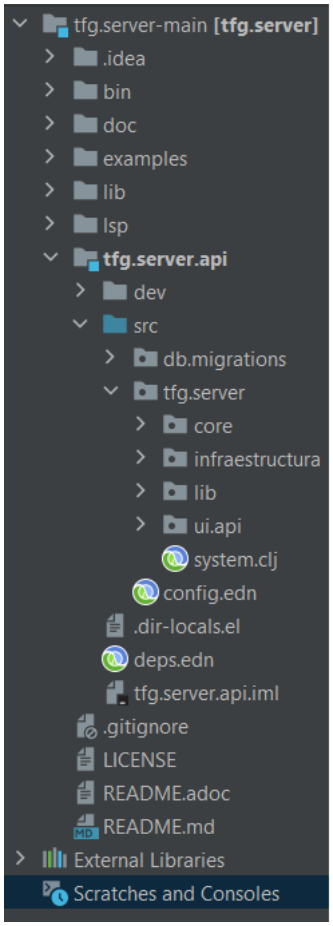

Figura 12: Estructura del servidor

La estructura común a otros proyectos basados en *juxt/edge* es la parte más externa del proyecto, formada por las primeras carpetas y archivos que se pueden ver al abrir la carpeta principal. Estamos hablando de las carpetas *bin*, *doc*, *examples*, *lib*...

Por otro lado, el entorno de desarrollo también ejerce su influencia sobre la estructura creando sus propias carpetas para así ejecutar el código correctamente. Estas carpetas son *.idea* y *lsp*.

Por último, tenemos la estructura creada para el desarrollo e implementación del proyecto. Esta se puede ver más de cerca en la figura 13. Lo que aparece en esta parte del proyecto ha sido creado de forma premeditada y contiene el código desarrollado durante la implementación del mismo.

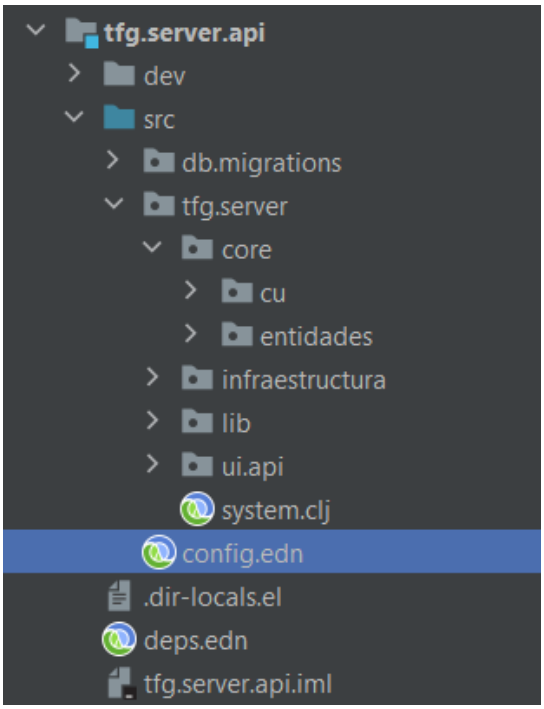

Figura 13: Estructura propia del proyecto

Uno de los archivos más importantes de esta sección es el llamado *deps.edn*, el cual podemos ver en el cuadro **Muestra de deps.edn**. En este se guarda la configuración del proyecto y se definen las rutas de las carpetas de desarrollo y las distintas librerías externas que se van a usar para el proyecto.

Para importar estas dependencias, de ahí el nombre *deps*, se hace uso de un repositorio conocido como *maven*. La forma de importar las dependencias es proporcionando el nombre y la versión que queremos usar, tal y como se aprecia en el cuadro Muestra de deps.edn.

Al igual que se pueden agregar librerías externas, también es posible hacer uso de librerías propias o que están descargadas en nuestro equipo. El proceso sería similar al ya explicado pero ahora indicaremos la ruta.

Muestra de deps.edn

```
;;; Definición de rutas
{:paths ["src"]
 ;;; Definición de dependencias
 :deps
 ;; Librería del repositorio
 {org.clojure/clojure {:mvn/version  "1.10.1"}
  ;; Librería en nuestro sistema de archivos
  juxt.edge/lib.app {:local/root "../lib/edge.app"}}
```

Por otro lado, está la carpeta *src*, que es una abreviación de la palabra ”source”. En esta carpeta se encuentra todo el código fuente, el cual representa casi por completo el trabajo desarrollado durante la realización del presente TFG.

En esta carpeta encontramos tres componentes principales:

- *db.migrations*. Contiene los archivos para crear y borrar las distintas tablas de la base de datos. Por cada tabla hay dos archivos, uno para crearla y otro para borrarla. Además, los archivos irán numerados por el orden en el que se deben ejecutar para crear las tablas.
- *config.edn*. Este es el archivo de configuración del sistema. Es importante diferenciarlo del archivo *deps.edn*, pues en este se van a definir los componentes que se guardarán en la memoria del sistema y se definirán las rutas del interfaz http. Un ejemplo claro de lo que se define es la configuración para entrar a la base de datos.
- *tfg.server*. Esta carpeta contiene varios módulos como son el core o núcleo, la infraestructura, las librerías, la interfaz de usuario y el fichero *system.clj*.
    - *system.clj*. Este fichero implementa la inicialización de los componentes del sistema, es decir, las definidas en el archivo *config.edn*.
    - *Core o núcleo*. Este módulo contiene las entidades del sistema, que en este caso será una por cada tabla de la base de datos, y los casos de uso.
    - *Infraestructura*. La infraestructura define todo lo necesario para la comunicación con los sistemas externos al servidor.

- *Lib*. En este módulo se implementan pequeñas librerías específicas para nuestro sistema.

- *UI*. Del inglés, User Interface, que significa interfaz de usuario. Este modulo implementará los distintos métodos de la API.

Por último tenemos la carpeta *dev*, en la cual se recogen distintos cuadernos de desarrollo. Estos cuadernos van a contener las llamadas de algunas funciones específicas del sistema para tener un pequeño diario de pruebas. Gracias a esto tendremos un pequeño entorno controlado de las funciones más representativas. También contiene un fichero de configuración (dev-config.edn) en el que se definen los parámetros específicos del sistema, como puede ser la configuración de la base de datos.

3.2.2. Implementación del servidor

En esta sección se desarrollarán, de forma resumida, las funcionalidades esenciales del sistema con la idea de generar una base suficiente como para entender el resto del código, el cual se puede consultar en el CD anexo.

El proyecto usado de base tiene como fin el facilitar, en este caso, la implementación de un sistema de gestión de peticiones web, el cual usaremos para recibir tanto las medidas que realicen los pacientes, así como para enviar a los mismos los ejercicios que tienen pendientes de cumplir cuando estos lo soliciten. Al simplificar el desarrollo de la API del servidor, la mayor carga de trabajo se ha centrado en la comunicación con la base de datos y el tratamiento de la información tanto entrante como saliente.

Entidades

Se inicia de esta manera con las entidades del sistema. Estas se encuentran en el *core* y son ficheros de código que implementan las definiciones de las propias entidades y las funciones específicas de las mismas. Las entidades definidas en nuestro sistema son: ejercicio, diagnóstico, paciente y profesional.

Entidad ejercicio

```
(ns tfg.server.core.entidades.ejercicio
  (:require
    [clojure.spec.alpha :as s]
    [tfg.server.core.entidades.paciente :as paciente]
    [tfg.server.core.entidades.profesional :as profesional]
    [tfg.server.lib.specs :as specs]))

;;; Definiciones
(s/def ::id int?)
(s/def ::musculo string?)
(s/def ::descripcion string?)
(s/def ::repeticiones int?)
(s/def ::t-repeticion int?)
(s/def ::t-ejercicio int?)
(s/def ::medida (s/or #{::pendiente} (s/coll-of float? :kind vector?)))

;; Fechas: ZonedDateTime en la zona del sistema
(s/def ::fecha-creacion ::specs/tipo-fecha)
(s/def ::fecha-realizacion ::specs/tipo-fecha)

(s/def ::ejercicio
  (s/keys :req [::id ::musculo ::descripcion ::repeticiones
                ::t-repeticion ::t-ejercicio ::fecha-creacion ::medida
                ::fecha-realizacion ::paciente/id ::profesional/id]))

;;; Funciones
(defn pendiente?
"Devuelve true si el ejercicio NO ha sido finalizado."
[{m ::medida}]
  (= ::pendiente m))
```

Para presentar la estructura básica de las entidades y para entender el código desarrollado, se va a detallar la entidad ejercicio, la cual es visible en el cuadro titulado **Entidad ejercicio**. Esta es una de las más completas y no supone un gran reto el comprender su implementación.

En el cuadro anterior se muestra cómo se declara el espacio de nombres, las librerías, la estructura de la entidad y un ejemplo de función.

Iniciamos el fichero con la función *ns*. Este diminutivo viene del inglés *namespace*, que en español significa espacio de nombres. Con ella definimos el nombre del espacio de trabajo (ese fichero) y, al usar la clave *:require*definimos, entre corchetes, cada librería o espacio de nombres que vamos a necesitar.

El uso de la llave *:as* en la importación de librerías es para proporcionar una alias a las mismas.

A continuación se definen los distintos componentes de la entidad ejercicio. Al usar la función *s/def* estamos indicando que a la llave con el nombre especificado le corresponde un valor que satisfaga o bien una función que devuelva un booleano o bien otro *spec* ya definido.

De esta forma, sería posible definir una variable id con cualquier valor. Sin embargo, solamente un valor entero va a satisfacer el spec de *::id*.

Algunos casos especiales son los *spec* definidos para la medida y las fechas:

- En el caso de la medida se indica que puede ser o bien la llave *::pendiente* o bien un vector de números reales.

- En el caso de las fechas, se hace uso de la librería propia del sistema que hemos importado como *specs*, en la cual se define un tipo de dato *::specs/tipo-fecha*.

Por último, se declara que una entidad de tipo ejercicio es un mapa que contiene las llaves definidas con anterioridad (además de los id de paciente y profesional) y que deben cumplir las condiciones definidas en sus *spec*.

El fichero finaliza con la definición de las funciones que van a interactuar con nuestra entidad de forma directa. En este caso se muestra una función muy básica con el fin de facilitar la comprensión del proceso de definición de funciones.

Para definir una función se hace uso de la macro *defn*, la cual recibe el nombre de la función, una cadena de caracteres (opcional) que describe su utilidad, un vector donde se definen los parámetros de entrada y el cuerpo.

La función mostrada espera recibir un mapa con la llave *::medida* y asigna el nombre m al valor asociado.

El cuerpo de la función comprueba que el valor de la clave *::medida* es igual a *::pendiente*. De esta forma, devolverá *true* cuando no haya una medida y *false* en el caso contrario.

Infraestructura

La carpeta de infraestructura contiene el código que se relaciona con los componentes externos al servidor. Su principal función será la de adaptar la información recibida y enviada al formato adecuado.

Un ejemplo claro es la comunicación con la base de datos. En el cuadro titulado **Infraestructura BD** se muestra el caso de la entidad diagnóstico.

Infraestructura BD

```
(defn tupla-a-diagnostico
[{:diagnostico/keys [id descripcion fecha id_ejercicio]}]
  #::diagnostico{:id           id
                 :descripcion  descripcion
                 :fecha        fecha
                 ::ejercicio/id id_ejercicio})

(defn select-diagnosticos [c]
  (->> (jdbc/execute! c ["select * from diagnostico order by id"])
       (mapv tupla-a-diagnostico)))

(defn insert-diagnostico!
  [c {descripcion ::diagnostico/descripcion
    fecha ::diagnostico/fecha id-ejercicio ::ejercicio/id}]
  (jdbc/execute! c
    [(str "insert into diagnostico (descripcion, fecha, id_ejercicio) "
          " values (?,?,?)")
     descripcion fecha id-ejercicio]))
```

Las funciones *tupla-a-diagnóstico* y *selec-diagnosticos* van emparejadas. La primera recibe un mapa resultado de consultar un diagnóstico en la base de datos y asigna los valores a las llaves definidas en nuestro sistema.

Por otro lado, la segunda recibe una conexión a la base de datos y la usa para ejecutar un comando *sql* para solicitar todas las entradas de la tabla diagnóstico. La consulta se hace como primer parámetro de la macro -», la cual pasará el resultado (una lista de tuplas leídas de la base de datos) como último parámetro a la función *mapv*, que aplicará la función de transformación explicada antes a cada diagnóstico leído. El resultado será un vector de entidades de tipo diagnóstico tal y como se han definido en nuestro sistema.

Por último, se muestra la función *insert-diagostico!*. En este caso, el nombre termina en exclamación porque es una función que va a producir un cambio, que será añadir un diagnóstico a la base de datos.

La entrada de la función será una conexión y el objeto que queremos insertar. De manera muy similar a *select-diagnostico*, va a ejecutar un comando *sql* para insertar los datos. Dado que el identificador se genera automáticamente, no se inserta junto con el resto de información.

En la infraestructura, también tenemos el migrador, que se encargará de aplicar los parches a la base de datos. Su funcionamiento básico consiste en consultar los ficheros de la carpeta *db-migrations* y comprobar que la base de datos se corresponde con el formato ahí especificado.

Casos de uso

Los casos de uso agrupan las funcionalidades disponibles en las entidades y la infraestructura para generar funciones de más alto nivel capaces de resolver los problemas o peticiones más comunes del servidor. En el cuadro **Caso de uso obtener-paciente** se puede apreciar la estructura común de estos. Para el presente TFG se han definido 3 casos:

- Actualizar el esquema de la base de datos. Implementa las funciones del migrador para mantener una versión actualizada de la base de datos.

- Administrar ejercicio. Permite tanto consultar ejercicios completados como pendientes. Además, se proporciona un método para para añadir una medida.

- Obtener paciente. Permite obtener la información de un paciente registrado en la base de datos

Caso de uso obtener-paciente

```
;;; Interfaces del caso de uso
(defprotocol Input
  (obtener-paciente-por-nombre-dni [o-p nombre-completo dni]
    "Devuelve todos los datos del paciente dados un nombre y un dni"))

;;; Interfaces con la infraestructura
(defprotocol BD
  (obtener-paciente-por-dni [bd dni]
    "Devuelve el paciente con el DNI dado"))



;;; Implementación del caso de uso
(defrecord ObtenerPacienteImpl [bd]
  Input
  (obtener-paciente-por-nombre-dni [_ nombre-completo dni]
    ;; Implementación y lógica del cu.
    ))

(defn make-obtener-paciente [bd]
  (->ObtenerPacienteImpl bd))
```

Los casos de uso, al ser un nivel más externo del proyecto, se encuentran encapsulados en sí mismos para no mostrar los detalles de su implementación. Para ello, hacen uso de protocolos para comunicarse con la infraestructura y con el exterior. Estos se definen con un nombre y una serie de funciones de las que sólo se muestra el nombre, la entrada y una descripción de la misma.

Para implementar estos protocolos, se hace uso de *records*. Estos implementan sus propios tipos de datos (*o-p* en el caso del protocolo *Input* y *bd* en el caso del protocolo *BD*), los cuales se definen como mapas.

En nuestro sistema, estos *records* son elementos definidos en la memoria que se crean al

iniciarse. Para ello, deben estar definidos en el fichero de configuración *config.edn*, donde se definen todos los componentes del sistema y su inicialización en *system.clj*.

Por último, en el cuadro se puede apreciar la función *make-obtener-paciente*, cuya función será la de generar una nueva instancia del *record* llamado *ObtenerPacienteImpl*.

API

Por último, debemos otorgar a los sistemas externos una forma de comunicarse con el nuestro. Es por eso que se ha decidido implementar una interfaz http cuyo cometido será el de activar los casos de uso.

Para definir los métodos http nos hemos ayudado de dos librerías proporcionadas por el proyecto base: *bidi* para el enrutamiento y *yada* para la gestión de los métodos.

Los puntos de acceso y el direccionamiento de los mismos a su implementación junto con la inyección de los elementos de memoria que pueda necesitar se definen en el fichero *config.edn*.

Definición de rutas web

```
:edge.yada.ig/listener
:handler #ig/ref :edge.bidi.ig/vhost :port 8080

:edge.bidi.ig/vhost
[["http://IP:8080"
  [""
   [["/api"
     [["/ping" #ig/ref :tfg.server.ui.api.ping/recurso]]]]]]]

:tfg.server.ui.api.ping/recurso
{}
```

En el cuadro **Definición de rutas web** se define la dirección web *http://IP:8080/api/ping* y se le especifica que las peticiones entrantes se dirigirán a la función que responde a la llave *recurso* dentro del fichero *ping*. En este caso, se le inyecta un mapa vacío ya que no hará uso de dependencias adicionales.

En el fichero de destino, se ha definido un recurso de la librería *yada* que simplemente va a devolver texto plano. Este método se creo para realizar pruebas de conexión.

Para explicar la creación de recursos con *yada*, recurriremos al servicio definido para el caso de uso de obtención de pacientes, el cual se puede consultar en el cuadro **Implementación de métodos HTTP**. En este, utilizamos el verbo http *get* y definimos un par de *query-params* para proporcionar los parámetros de entrada del caso de uso.

En la primera función, se comprueban que los datos son correctos antes de iniciar el caso de uso. El resultado del mismo será un mapa que se transformará automáticamente a *JSON*. En caso de que falle algún dato de entrada, se devolverá un error 400 indicando que no se ha encontrado el recurso con esa información.

Además, podemos ver que en este caso se inyecta el objeto *obtener-paceinte*. Este es un mapa que complace el *record* definido anteriormente en este mismo capítulo.

Implementación de métodos HTTP

```
(defn obtener-paciente-por-nombre-dni
  [{{{:keys [nombre-y-apellidos dni]} :query} :parameters
  response :response} obtener-paciente]
  (if (and (string? nombre-y-apellidos) (valid? ::ls/tipo-dni dni))
    (o-p/obtener-paciente-por-nombre-dni
      obtener-paciente nombre-y-apellidos dni)
    ((assoc response :status 400)))

(defmethod ig/init-key ::obtener-paciente-por-nombre-dni
  [k obtener-paciente]
  (yada/resource
    {:logger log-request
     :methods
     {:get
      {:parameters {:query {:nombre-y-apellidos String :dni String}}
      :produces "application/json"
      :response (fn [ctx]
      (obtener-paciente-por-nombre-dni ctx obtener-paciente))}}}))
```

3.3. Pruebas y verificación

Las pruebas se han llevado a cabo mientras se ensamblaba el sistema final explicado a lo largo del presente capítulo. Estas se pueden separar en dos secciones:

- Pruebas de la base de datos. Se realizaron a través de un cuaderno de desarrollo llamado *dev-bd*. En este se comprueba que para cada tipo de entidad definida, somos capaces de escribirla y leerla de la base de datos.
- Pruebas del interfaz web. Se resumen en la llamada a cada método http desarrollado y la verificación de que se producen los efectos deseados en el sistema. Con esto, se comprobó el funcionamiento de los casos de uso.

Las pruebas se repitieron hasta obtener resultados satisfactorios, lo que desencadenó una serie de correcciones y refactorizaciones del código desarrollado hasta alcanzar un punto óptimo.

4

Conexión entre el sistema de captura y el servidor (aplicación Android)

En este cuarto capítulo se recoge la estructura del sistema de conexión entre los bloques de captura de medidas y el servicio de base de datos. Se describirán los cambios necesarios tanto en el lado del servidor como en el sistema de captura de medidas para realizar la comunicación, así como también se desarrollará la estructura básica y el código más importante de la aplicación Android que sirve como nexo de unión. La finalidad de la aplicación será, por tanto, recibir las medidas por Bluetooth, procesarlar, y enviarlas al servidor.

En el estado actual, se ha presentado un sistema de captura de medidas cuya única forma de comunicación es el cableado a un ordenador y una base de datos cuyo punto de acceso es un servicio web. Aunque es posible hacer peticiones desde un ordenador al servidor, desde el punto de vista del paciente no es nada cómodo y no todos tienen por qué disponer de uno.

Además, esto limita al usuario a la hora de realizar los ejercicios, pues va a necesitar estar siempre cerca de la computadora al tener que conectar el sistema de medida de forma cableada.

Es por ello que nace la nace la necesidad de una conexión entre la toma de medidas y el servidor que otorgue una mejor experiencia de usuario y se adapte mejor a la situación. Como respuesta a esta necesidad, se propone una aplicación móvil que funcione como intermediaria. De esta forma, recibiría las medidas por *Bluetooth* y las enviaría al servidor vía internet.

Esta elección aparece tras valorar distintas ventajas. Entre ellas encontramos:

- Es más común que una persona disponga de un dispositivo móvil a que disponga de un ordenador personal.
- En caso de que el paciente no disponga de un teléfono móvil, es una opción más barata que puede administrar el propio centro sanitario.
- Ofrece más libertad, pues la conexión sería completamente inalámbrica. Esto también reduce el posible desgaste del sistema si hubiera que conectarlo siempre.
- Reduce la vulnerabilidad del sistema, pues es más complicado añadir un virus informático al sistema de medidas.

4.1. Estructura básica

La aplicación se compone de diversas actividades (Activity), que serían las ventanas o vistas que se muestran al usuario una vez comienza la interacción. Cada una tiene un objetivo definido:

- Inicio. Como su nombre indica, es la actividad de inicio de la aplicación. En ella se muestran dos botones que inician las dos funciones distintas disponibles y otro que ofrece información al usuario.
- Seleccionar paciente. Permite elegir al paciente al que van a pertenecer los ejercicios con los que se va a trabajar. Esta actividad se ejecuta siempre en segundo lugar.

- Seleccionar ejercicio. Muestra los ejercicios disponibles del paciente para que elija uno con el que continuar. Esta actividad siempre se ejecuta en tercer lugar.

- Aviso. Se dan algunas nociones básicas sobre cómo conectar el dispositivo móvil al sistema de medidas.

- Elegir dispositivo BT. Contiene los botones para encender y apagar el *Bluetooth* así como para mostrar los dispositivos vinculados al teléfono.

- Realizar medida. Permite iniciar la comunicación con el sistema de medida y recibir los datos del mismo. Además, posee un botón con el que se envían al servidor y la información relativa al ejercicio que se va a realizar.

- Mostrar resultados. Muestra algo de información del ejercicio que se está consultando y contiene un botón que inicia la actividad mostrar gráfico.

- Mostrar gráfico. Presenta de forma gráfica la medida tomada durante la realización del ejercicio. Esta gráfica permite variar su tamaño desplazarse por ella para obtener una mejor visión de la misma.

A continuación, en la figura 14, se muestra un esquema simple de relaciones entre las distintas actividades de la aplicación.

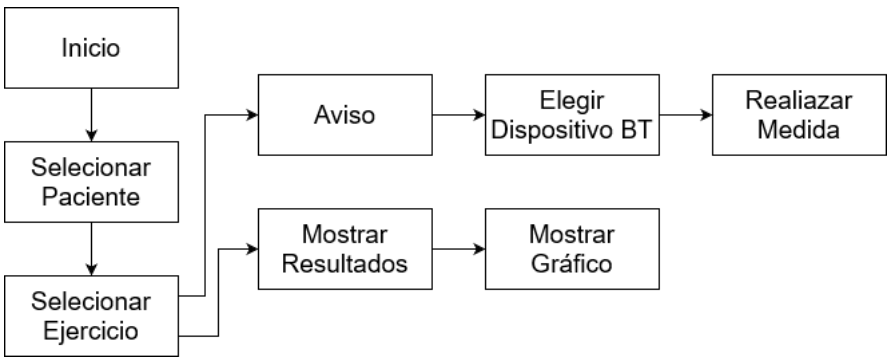

Figura 14: Diagrama de actividades de la aplicación

En este, podemos ver que hay dos flujos de actividad en la aplicación. Sin embargo, antes se ha mencionado que se elige el flujo por el que se va a ir en la actividad Inicio, pero la bifurcación no ocurre hasta la actividad Seleccionar Ejercicio. Esto es debido a que en ambos

flujos es necesario pasar por las actividades anteriores. La diferencia aparece en los ejercicios mostrados, pues en un caso estarán ya completados y en el otro estarán pendientes.

4.2. Flujo inicial

Se refiere a la parte común, la que siempre se va a ejecutar independientemente del flujo que se elija ejecutar en la aplicación.

4.2.1. Actividad Inicio

Figura 15: Actividad Inicio

En la figura 15 se muestra la actividad inicial de la aplicación. En ella se nos presentan 3 botones, que desencadenarán 3 acciones distintas.

El botón de ayuda abre un cuadro de diálogo en el que se muestra información básica sobre el uso de la aplicación y las funcionalidades disponibles. Este contiene un único botón cuya función es cerrar el cuadro.

El funcionamiento de los otros dos botones es prácticamente el mismo. Ambos inician la siguiente actividad y pasan un parámetro adicional a la misma. La diferencia reside en este parámetro, el cuál indica el botón que se ha presionado, pues es la forma de indicar qué camino tomar a la hora de la bifurcación del flujo.

En el cuadro titulado **Implementacióndel botón TOMAR MEDIDA** se puede ver la implementación. La función crea un *Intent* para cambiar a la actividad Seleccionar Paciente y le pasa el parámetro *EXTRA_TIPO_PETICION*, con el valor de la constante *EJERCICIOS_PENDIENTES*.

Implementacióndel botón TOMAR MEDIDA

```
tomarMedidaBtn.setOnClickListener(
  v-> {
    Intent intent = new Intent(Inicio.this, SeleccionarPaciente.class);
    intent.putExtra(Inicio.EXTRA_TIPO_PETICION, EJERCICIOS_PENDIENTES);
    startActivity(intent);
});
```

4.2.2. Actividad Seleccionar Paciente

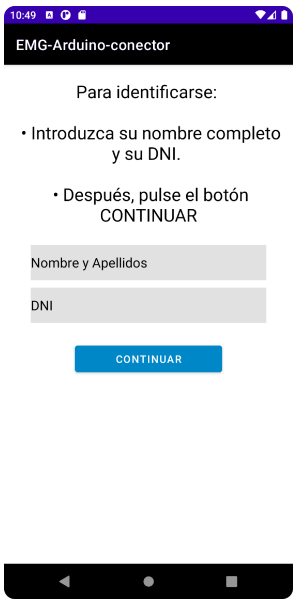

Figura 16: Actividad Seleccionar Paciente

En la figura 16 se muestra la segunda actividad de la aplicación. Esta actividad no cambia nunca, ya que no depende del flujo de actividades elegido al inicio. En ella se muestran dos entradas de texto y un botón.

Las entradas de texto son para recoger la información del paciente, con la que se realizará una petición http al servidor para obtener los datos necesarios para las siguientes.

Al pulsar el botón, se lanza dicha petición y se muestra el resultado en un cuadro de diálogo. El caso de recibir un error como respuesta se muestra en el cuadro **Resultado erróneo de la petición**, en el que se muestra información del error y un botón que cierra el diálogo.

Resultado erróneo de la petición

```
error -> {Log.e(TAG, error.toString());
 int tituloError;
 int mensajeError;
 switch (error.networkResponse.statusCode){
   case 400:
     tituloError = R.string.titulo_datos_incorrectos;
     mensajeError = R.string.datos_incorrectos;
     break;
   case 404:
     tituloError = R.string.titulo_error_conexion;
     mensajeError = R.string.error_conexion;
     break;
   default:
     tituloError = R.string.titulo_error_inesperado;
     mensajeError = R.string.error_inesperado;
     break;
 }
 AlertDialog.Builder builder = new AlertDialog.Builder(
          SeleccionarPaciente.this,
          R.style.Base_Theme_AppCompat_Light_Dialog_Alert);
 builder.setMessage(mensajeError)
        .setTitle(tituloError)
        .setNegativeButton(R.string.aceptar, dialog -> dialog.cancel());
 AlertDialog errorConexion = builder.create();
 errorConexion.show();
}
```

Por otro lado, tal y como representa el cuadro **Resultado existoso de la petición**, si es un acierto se mostrará la información obtenida de la base de datos sobre el paciente y dos botones:

- Botón Cancelar. Cierra el cuadro de diálogo y te devuelve a la actividad. Se presenta en caso de que la información obtenida no sea la correcta.
- Botón Aceptar. Inicia una nueva actividad, la cual recibe el parámetro extra de la actividad anterior junto con la información del paciente.

Resultado existoso de la petición

```
paciente -> {
 Log.i(TAG,paciente.toString());
 Paciente p = new Paciente(paciente);
 AlertDialog.Builder builder = new AlertDialog.Builder(
         SeleccionarPaciente.this,
         R.style.Base_Theme_AppCompat_Light_Dialog_Alert);
 builder.setMessage("Nombre: "+p.getNombre()+"Apellidos: "
    +p.getApellidos()+"Sexo: "+p.getSexo()+"Fecha de nacimiento: "
    +p.getFechaNacimiento()+"Edad: "+p.getEdad())
 .setTitle(R.string.confirmar_informacion)
 .setPositiveButton(R.string.aceptar, d -> {
    Intent intent = new Intent(SeleccionarPaciente.this,
            SeleccionarEjercicio.class);
    intent.putExtra(Inicio.EXTRA_TIPO_PETICION, tipoPeticion);
    intent.putExtra(EXTRA_ID, String.valueOf(p.getId()));
    startActivity(intent);
    })
 .setNegativeButton(R.string.cancelar, dialog -> dialog.cancel());
 AlertDialog respuestaPeticion = builder.create();
 respuestaPeticion.show();
}
```

Tanto en esta actividad como en la siguiente, es necesario añadir los permisos de uso de internet en el fichero *Manifest*.Esto se muestra en el cuadro **Resultado existoso de la petición**.

Importanción de permisos de red

```
<uses-permission Android:name="Android.permission.INTERNET" />
<uses-permission Android:name="Android.permission.NETWORK_STATE" />
```

4.2.3. Actividad Seleccionar Ejercicios

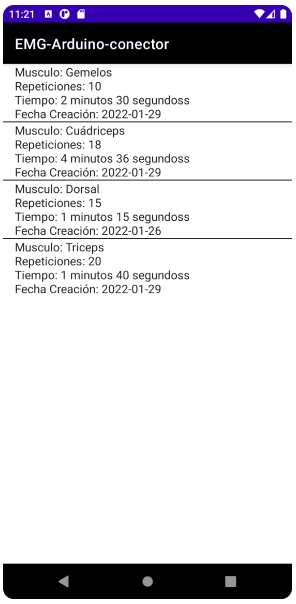

Figura 17: Actividad Seleccionar Ejercicio

En la figura 17 de puede apreciar la tercera actividad. Está compuesta por una lista de objetos que contienen información sobre los ejercicios del paciente. Dependiendo del botón que se pulsara en la actividad inicial, estos ejercicios serán pendientes de medida o completados.

Aunque no se aprecia en la imagen, la lista es desplazable. Esto significa que, en el caso de que haya más ejercicios de los que se pueden mostrar en pantalla, se podrá navegar hacia abajo para ver el resto.

Al tocar en un ejercicio, se inicia la siguiente actividad, la cual depende de si se va a ver un resultado o a completar un ejercicio. A esta nueva actividad se le envían únicamente los datos relativos al ejercicio, pues ya no será necesario saber qué botón presionó el usuario al inicio

de la aplicación.

Para obtener los ejercicios, se realiza una petición http cuya dirección de destino variará según el tipo de ejercicios que se desea visualizar. En el cuadro titulado **Definición de la URL** se muestra cómo se decide dicha dirección web.

Definición de la URL

```
tipoPeticion = intent.getStringExtra(Inicio.EXTRA_TIPO_PETICION);
String url;
switch (tipoPeticion){
    case Inicio.EJERCICIOS_PENDIENTES:
        url = URL_Ejercicios_Pendientes;
        break;
    case Inicio.EJERCICIOS_COMPLETADOS:
        url = URL_Ejercicios_Completados;
        break;
    default:
        url = null;
        break;
}
if (!(url == null)){
    obtenerEjercicios(idPaciente, url);
}else{
    AlertDialog.Builder builder = new AlertDialog.Builder(
        SeleccionarEjercicio.this,
        R.style.Base_Theme_AppCompat_Light_Dialog_Alert);
    builder.setMessage(R.string.error_inesperado)
            .setTitle(R.string.titulo_error_inesperado)
            .setNegativeButton(R.string.aceptar, d -> d.cancel());
    AlertDialog errorConexion = builder.create();
    errorConexion.show();
}
```

Primero se obtiene el tipo de petición (elegido en la actividad inicial) desde el *intent* que ha iniciado esta actividad. A partir de ahí, dependiendo del valor, se elige una dirección web válida o se le asigna un valor nulo. En caso de ser nulo, se presenta un cuadro de diálogo informando del error al usuario.

Una implementación parecida se ha utilizado para elegir la actividad que se va a abrir a continuación. Usando un *switch* sobre la variable *tipoPeticion* se realiza un *intent* a la actividad Elegir Dispositivo BT o bien a Mostrar resultados.

4.2.4. Pruebas y verificación

Para esta parte inicial de la aplicación, las pruebas se han centrado principalmente en el visionado correcto de los cuadros de diálogo y en el funcionamiento adecuado de las peticiones http. Todas se han llevado a cabo usando un dispositivo real en modo depuración, de forma que los mensajes de desarrollo se registraran en el entorno de programación.

El listado completo de pruebas sería:

- Prueba de compilación de la actividad en un dispositivo Android.
- Prueba del funcionamiento de todos los controles de la actividad.
- Prueba del funcionamiento de las peticiones http al servidor. Esto incluye no sólo que se haga una petición correcta, sino que también se reciben todos los posibles mensajes de error que genera la petición y que se procesan adecuadamente.
- Prueba de salto entre actividades. Esto incluye que se envían los datos necesarios entre las actividades afectadas.
- Prueba del flujo completo. Se verifica que todo funciona y no se producen errores desde la actividad de inicio hasta la actividad Seleccionar Ejercicio.

El resultado general de las pruebas ha sido correcto. Durante las mismas, se han corregido los fallos encontrados y se ha asegurado, al menos, una vía funcional.

Cabe mencionar que, en algunas circunstancias, si la petición http tarda demasiado, puede provocar que el paciente toque de nuevo el botón generando una nueva petición y colapsando (no en todos los casos) la aplicación. Una posible solución sería bloquear el botón hasta obtener

una respuesta del servidor, pero esto podría generar igualmente problemas en el caso de que el botón se bloquee y nunca se llegue a enviar la petición.

4.3. Toma de medidas

En este apartado, además de la implementación propia de la aplicación, se presentarán también los cambios realizados al sistema de medidas para poder realizar la comunicación de manera inalámbrica.

4.3.1. Cambios en el sistema Arduino

Como ya se ha mencionado con anterioridad, la conexión establecida entre ambos sistemas (Arduino y aplicación) será a través de Bluetooth. Concretamente, se ha decidido usar el módulo HC-06. Este módulo se ejecuta por defecto en modo esclavo (esperando recibir una solicitud de conexión) y es compatible con Arduino y Android, por lo que su elección se ve justificada.

Para poder usar el módulo en nuestro proyecto, lo primero es programarlo según los requisitos del mismo. Con este fin, lo primero es encontrar la frecuencia en la que se encontraba en ese momento, por lo que se recurrió al comando llamado *AT*.

Usando un programa Arduino, se iba variando la freciencia (baud rate) progresivamente y, en cada una, se enviaba el texto *AT* al módulo. Al pasar por la frecuencia en que estaba programada el HC-06, este respondía con un *OK* que era impreso por pantalla.

Para entablar la conexión con el módulo y mantener en funcionamiento el monitor serial, se ha hecho uso de la librería *SoftwareSerial*. Con ella, se pueden definir dos pines cualesquiera (diferentes al 0 y 1) como receptor y emisor de datos. Así, sólo restaría conectar el transmisor y receptor del HC-06 a estos pines y estaríamos comunicándonos via puerto serie con Arduino, y este con el módulo Bluetooth a través de sus propios pines.

Originalmente, el módulo estaba configurado a 9600 baudios, frecuencia que aumentamos a 115200 baudios para un mejor desempeño. Para ello, recurrimos al código arduino definido en el cuadro llamado **Comandos AT**.

El código ahí descrito se basa en el concepto ya explicado. De nuevo, se ha hecho uso de la librería *SoftwareSerial* para definir los pines 2 y 3 como transmisor y receptor respectivamente.

Los comando AT usados y sus efectos son:

- Comando AT. Para comprobar que la frecuencia era correcta.
- Comando AT+BAUD8. Su función es cambiar la frecuencia de señal a 115200 baudios.
- Comando AT+NAMEHC-06. Cambia el nombre con el que se identifica el módulo a HC-06.

Comandos AT

```
#include < SerialSoftware.h >
SoftwareSerial BT(2,3);

void setup()  {
    Serial.begin(9600);
    BT.begin(9600);
}

void loop() {
    if (BT.available()){
        Serial.write(BT.read());
    }
    if (Serial.available()){
        BT.write(Serial.read());
    }
}
```

Finalmente, el código de obtención de medidas presentado en el capítulo 2 también ha sufrido cambios como consecuencia de la adición de la comunicación inalámbrica. Estos cambios son visibles en el cuadro **Script final para el sistema de medidas Arduino**.

Para poder comunicar las medidas al HC-06, se ha añadido la librería ya mencionada *SoftwareSerial*. Sin embargo, esta es incompatible con la librería propia del *eHealth*, por lo que hubo que modificarla borrando la carpeta PinChangeInt para permitir el uso simultaneo de las

dos. Esta carpeta estaba pensada para las funcionalidades relacionadas con el pulsioximetro, por lo que no va a generar conflictos con nuestro sistema.

A continuación, se ha borrado la limitación de medidas introducida en un principio y en lugar de enviar las medidas al puerto serie, estas se envían al módulo Bluetooth.

Además, se han aumentado los decimales a 3 y se ha introducido un retardo de 100 milisegundos entre cada medida, por lo que la sensibilidad será de 10 medidas por segundo.

Script final para el sistema de medidas Arduino

```
#include < eHealth.h >
#include < SoftwareSerial.h >
SoftwareSerial BT(2,3);

void setup()  {
    BT.begin(115200);
    Serial.begin(115200);
}

void loop() {
    delay(100);
    float EMG = eHealth.getEMG();
    BT.print(EMG,3);
    BT.println("");
    Serial.println(EMG,3);
    Serial.println("");
}
```

4.3.2. Actividad Elegir Dispositivo BT

Esta es la primera actividad en la que se hace uso de las funciones Bluetooth del dispositivo móvil, y para ello, primero se deben declarar los permisos en el fichero *Maniffest* tal y como se muestra en el cuadro **Importanción de permisos Bluetooth**.

Importanción de permisos Bluetooth

```
<uses-permission Android:name="Android.permission.BLUETOOTH" />
<uses-permission Android:name="Android.permission.BLUETOOTH_ADMIN" />
```

Estos permisos nos dan acceso a realizar conexiones, transferir datos y manipular las preferencias del sistema relativas al servicio Bluetooth. Aunque se incluyen más funciones con los permisos importados, la aplicación usará principalmente lo mencionado.

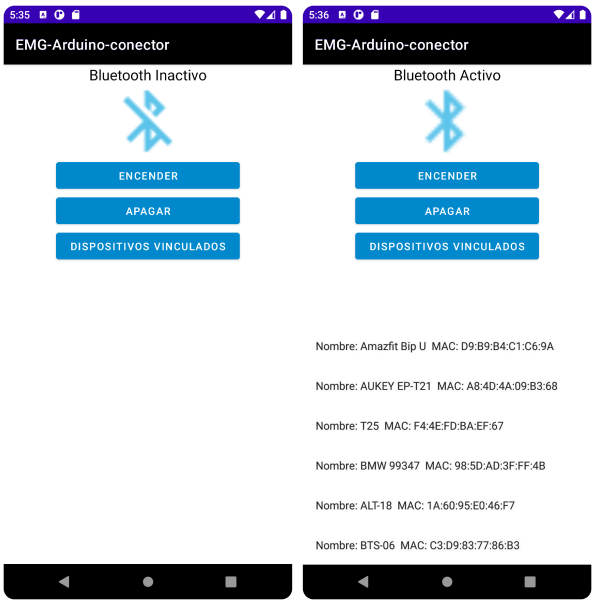

Figura 18: Actividad Elegir Dispositivo BT

En la figura 18 se muestran dos versiones distintas de la misma actividad. En la primera, el Bluetooth del dispositivo está desactivado. En la segunda, se ha activado el servicio Bluetooth y se ha pulsado el botón dispositivos vinculados.

Tal y como sus nombres indican, los botones ENCENDER y APAGAR desempeñan las acciones necesarias para activar y desactivar el Bluetooth del dispositivo Android. Para ello, es necesario primero definir un adaptador que nos de acceso al controlador de este servicio. Este procedimiento se muestra en el cuadro **Botones ENCENDER y APAGAR**.

Una vez tenemos el adaptador, podemos solicitar permiso al dispositivo para encender el controlador BLuetooth. Al hacer esto, se mostrará un cuadro de diálogo al usuario que debe aceptar para poder continuar. Además, si se consigue la activación, el texto y la imagen mostrados en la parte superior cambiarán para mostrarlo.

Por otro lado, el botón de apagado deshabilita el adaptador (lo que genera el fin del servicio) y a continuación cambia el texto y la imagen para reflejarlo.

Botones ENCENDER y APAGAR

```
BluetoothAdapter bAdapter = BluetoothAdapter.getDefaultAdapter();
// Botón para encender BT
onBtn.setOnClickListener(
    v -> {
        if(!bAdapter.isEnabled()){
           Intent inentEncenderBT =
                new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            ActivityResultLauncher.launch(inentEncenderBT);
    }
});

// Botón para apagar BT
offBtn.setOnClickListener(
    v -> {
        bAdapter.disable();
        estadoBtTv.setText("Bluetooth Inactivo");
        logoBtIv.setImageResource(R.drawable.ic_action_off);
        Toast.makeText(getApplicationContext(),"Bluetooth Desactivado",
        Toast.LENGTH_SHORT).show();
});
```

El objeto *ActivityResultLauncher* mostrado en el código anterior es el encargado de comprobar si el intento de iniciar el Bluetooth ha sido exitoso. En tal caso, cambia los valores del texto y la imagen de forma similar a como se hace en el botón de apagado.

Por otro lado, el botón DISPOSITIVOS VINCULADOS, cuya implementación se muestra en el cuadro **Botón DISPOSITIVOS VINCULADOS**, es el encargado de recolectar la información de los dispositivos recordados por el móvil y mostrarla en la lista de más abajo. Para ello, obtenemos la lista de dispositivos recordados desde el adaptador y generamos una nueva lista

en la que mostramos el nombre y la MAC de cada uno. Esta nueva lista es la que se muestra al usuario.

Botón DISPOSITIVOS VINCULADOS

```
Set<BluetoothDevice> pairedDevices = bAdapter.getBondedDevices();
ArrayList<String> list = new ArrayList<>();
if(pairedDevices.size()>0){
    for(BluetoothDevice device: pairedDevices){
        String devicename = device.getName();
        String macAddress = device.getAddress();
        list.add("Nombre: "+devicename+"  MAC: "+macAddress);
    }
    if(bAdapter.isEnabled()){
        ArrayAdapter<String> aAdapter = new ArrayAdapter<>(
            getApplicationContext(),
            android.R.layout.simple_list_item_1,
            list);
        vincularLv.setAdapter(aAdapter);
    }
```

Tal y como se muestra en el cuadro **Implementación de la lista**, al pulsar sobre uno de los objetos de la lista se obtiene su MAC y se intenta iniciar la actividad Realizar Medida, A esta se le proporciona la MAC y la información del ejercicio obtenida en la actividad anterior.

Implementación de la lista

```
vincularLv.setOnItemClickListener((parent, view, position, id) -> {
    String itemValue = (String) vincularLv.getItemAtPosition(position);
    String MAC = itemValue.substring(itemValue.length() - 17);
    Intent intent = new Intent(ElegirDispositivoBT.this,
            RealizarMedida.class);
    intent.putExtra(EXTRA_DIRECCION_MAC, MAC);
    intent.putExtra(SeleccionarEjercicio.EXTRA_EJERCICIO, ejercicio);
    startActivity(intent);}
```

4.3.3. Actividad Realizar Medida

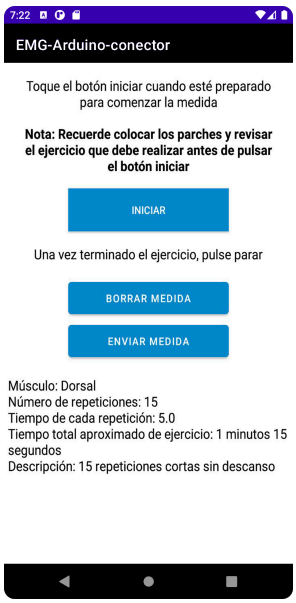

Figura 19: Actividad Realizar Medida

En la figura 19 se muestra la actividad encargada de la toma de medidas. Sin embargo, antes de iniciarse se realiza la conexión Bluetooth con el dispositivo seleccionado en la actividad anterior. Para ello, justo como se muestra en el cuadro **Creación de la conexión Bluetooth**, se crea un *socket* haciendo uso de un identificador UUID generado aleatoriamente para esta aplicación.

Creación de la conexión Bluetooth

```
BluetoothSocket socket;
try {
    socket = bAdapter.getRemoteDevice(MAC)
    .createRfcommSocketToServiceRecord(miUUID);
} catch (IOException e) {}
try {
    socket.connect();
} catch (IOException connectException) {}
hebraBT = new ConnectThread(socket);
hebraBT.start();
```

Una vez creado el *socket*, este se activa para iniciar la conexión y se utiliza para generar un objeto de tipo *ConnectThread*, el cual manejará la conexión Bluetooth.

Para hacer el código del más visible, se ha eliminado la implementación relativa al caso en que no es posible iniciar la conexión. Por ello, se muestran los bloques *catch* vacíos.

La clase *ConnectThread* ha sido creada para esta aplicación de forma independiente. Esta clase invoca un nuevo hilo que será el encargado de recibir los datos de la comunicación con el sistema de medida. Para enviar estos datos al hilo principal, se hace uso de un manejador (*handler*).

El hilo encargado de la conexión Bluetooth se crea en un objeto de tipo *ConnectedThread*. Esta clase implementa, junto a su constructor, un método run(), cuya implementación se puede consultar en el cuador **Método run() de la clase ConnectedThread**. Su principal cometido será leer los datos recibidos por el *socket* y guardarlos en un *InputStream* para después ser enviado al hilo principal a través del manejador.

Para evitar congestiones en el sistema, se realiza la lectura del *socket* haciendo uso de un *BufferedReader*. Gracias a esto, sólo tendremos que asegurarnos de que nunca leemos más lento de lo que recibimos para evitar llenar el buffer.

Método run() de la clase ConnectedThread

```
public void run() {
    InputStreamReader lectorDatos = new InputStreamReader(streamEntrada);
    BufferedReader br = new BufferedReader(lectorDatos);
    // Se mantiene escuchando hasta que se produce una excepción
     while (true) {
        try {
            // Leemos el stream de entrada
            String linea = br.readLine();
            // Mandamos los datos al manejador (handler)
            handler.obtainMessage(1,linea).sendToTarget();
        } catch (IOException e) {}
    }
}
```

Por último, para entender por completo el funcionamiento de la conexión Bluetooth, es necesario comprender la tarea del manejador o *handler*. Aunque este se crea en el hilo principal de la aplicación, será el hilo secundario el que lo use para enviar los datos. La implementación del mismo puede verse en el cuadro **Manejador o *handler***.

Para enviar los datos leídos al manejador usamos el método *obtainMessage().sendToTarget()*. Esto provoca que la función *handleMessage()* reciba el dato y realice las operaciones oportunas con este. En nuestro caso, lo añade a la variable *medidaEMG*.

Manejador o *handler*

```
private final Handler handler = new Handler(Looper.getMainLooper()) {
    @Override
    public void handleMessage(Message msg) {
        String mensaje;
        // Leemos el dato como texto y lo guardamos como número
        mensaje = (String) msg.obj;
        medidaEMG.add(Float.parseFloat(mensaje));
    }
};
```

Una vez conocido el funcionamiento de la conexión inalámbrica implementada, sólo restaría presentar la implementación de los controles de la actividad. En la figura del inicio del apartado (figura 19) se muestra que la actividad contiene, junto a una serie de instrucciones y la descripción del ejercicio seleccionado, tres botones distintos. En este caso, uno de ellos es u botón de activación (toggle button), mientras que los otros dos son simples botones normales.

El botón de activación, cuya implementación es visible en el cuadro titulado **Botón de inicio de la comunicación Bluetooth**, permite al usuario el encendido (INICIAR) y apagado de la comunicación Bluetooth. Esto le proporciona control sobre el inicio y el fin del periodo de toma de medidas. Para ello, llama a los métodos *comunicacion.start()* y *comunicacion.cancel()*, donde comunicacion es un objeto de tipo *ConnectedThread*.

Además de lo ya comentado, tanto el inicio como el fin de la comunicación producen otros efectos. Al iniciar la toma de medidas se borra el contenido del vector que las va a almacenar para asegurar que está vacío, mientras que al finalizarla esto se notifica al usuario con un

pequeño mensaje en la parte inferior de la pantalla del dispositivo.

Botón de inicio de la comunicación Bluetooth

```
iniciarBtn.setOnCheckedChangeListener((buttonView, isChecked) -> {
        if (isChecked) {
            medidaEMG.clear();
            comunicacion.start();
        } else {
            comunicacion.cancel();
            Toast.makeText(getApplicationContext(),
                "La toma de medidas ha finalizado",
                Toast.LENGTH_LONG).show();
        }
    });
```

Por otro lado, el botón para borrar la medida reinicia la variable *medidaEMG* dejándola vacía. Esta función, sin embargo, sólo está disponible cuando la comunicación Bluetooth está detenida. En caso de estar activa, al pulsar el botón se mostrará un mensaje al usuario informando de que primero debe finalizar la toma de medidas.

En la implementación del botón mencionado se han usado los métodos *medidaEMG.clear()* y *Toast.makeText()*, cuyo uso ya se ha mostrado anteriormente. Es por ello que no se incluirá dicha implementación en el presente documento.

Finalmente, se presenta un botón con la función de enviar la medida tomada al servidor. Sin embargo, esta no es su única utilidad, pues también cierra la conexión Bluetooth o, en el caso de estar la toma de medidas iniciada, se lanza un aviso al usuario.

Por otro lado, en el cuadro **Botón de envío de medidas** es visible la implementación del botón mencionado. Tal y como se muestra, antes de realizar la petición http al servidor, es necesario encapsular los datos en un objeto JSON. Para ello, se genera un JSONArray con las medidas y un objeto que va a contener estas y el identificador del ejercicio rescatado desde la actividad anterior.

Una vez generado el objeto que se enviará al servidor se llama al método *postMedida*, el cual iniciará la petición y, en caso de recibir una respuesta negativa, se lo notificará al usuario a

través de un cuadro de diálogo (tal y como hemos visto en otras implementaciones anteriores). En el caso de enviarlas exitosamente no se realizan acciones adicionales, pues este es el final de la actividad.

Botón de envío de medidas

```
enviarMedidaBtn.setOnClickListener(v -> {
    hebraBT.cancel();
    if (!medidaEMG.isEmpty() && !iniciarBtn.isChecked()) {
        try {
            JSONArray arrayMedidas = new JSONArray(medidaEMG);
            JSONObject jsonObject = new JSONObject();
            jsonObject.put("id-ejercicio", ejercicio.getId());
            jsonObject.put("medida", arrayMedidas);
            postMedida(jsonObject);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    } else {
        Toast.makeText(getApplicationContext(),
            "Primero debe finalizar la medida.",
            Toast.LENGTH_SHORT).show();
    }
});
```

4.3.4. Pruebas y verificación

Durante la implementación de la conexión se han llevado a cabo varias pruebas tanto del lado del sistema Arduino como del lado de la aplicación. En la primera de estas pruebas, con el fin de comprobar el correcto funcionamiento del módulo usado, se utilizaron los comandos AT descritos al inicio de la sección.

A continuación, se vinculó el dispositivo móvil al módulo y con la ayuda de la aplicación *Arduino Bluetooth Controller*[13] se programó un botón que, al pulsarlo, se enviaba la palabra

ON al módulo y el sistema Arduino reaccionaba encendiendo un LED. Con esta prueba se comprobó que la integración del módulo con Arduino era correcta.

Una vez comprobado que el comportamiento del módulo era el esperado, se pasó a las pruebas de integración con la aplicación desarrollada en el presente capítulo. Para ello, se uso el sistema de registros (logs) del entorno de desarrollo para imprimir los datos recibidos desde el módulo. A su vez, los datos enviados a este se mostraban en el monitor serie.

Por último, cuando se corroboró que la conexión a través de la aplicación era funcional, se realizaron varios ejercicios ficticios y se comprobó que las medidas tomadas eran enviadas al servidor. De esta forma, se pudo verificar que el sistema realizaba lo esperado al completo.

Aunque los resultados obtenidos fueron mayoritariamente satisfactorios, cabe destacar que, en ciertas ocasiones, la aplicación puede ralentizarse al intentar establecer la conexión (llegando a tardar varios segundos). Otro problema encontrado con frecuencia es que, al cambiar el código del Arduino, es necesario desconectar el módulo de la placa durante la carga (de lo contrario sería imposible enviar información por la conexión).

4.4. Consulta de ejercicios completados

A continuación se muestra el último flujo de actividades disponible. La función de estas será la de presentar al usuario la información de un ejercicio que ya ha sido completado.

4.4.1. Actividad Mostrar Resultado

Tal y como se aprecia en la figura 20, esta es una actividad muy sencilla centrada en mostrar información. Contiene un cuadro de texto en el que se muestra la información relativa al ejercicio obtenida en la actividad anterior (Seleccionar Ejercicio) y un botón con el texto MOSTRAR GRAFICO cuya única función del botón mencionado será la de iniciar la siguiente actividad, a la cual pasará el vector de medidas para ser representado gráficamente.

Dado que en esta actividad no se implementa nada nuevo respecto al resto, no se va a mostrar la programación de la misma para evitar que el presente documente alcance una longitud excesiva.

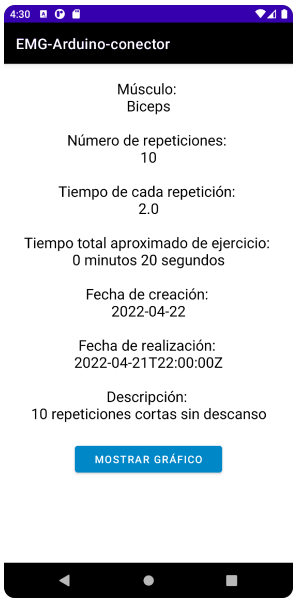

Figura 20: Actividad Mostrar Resultados

4.4.2. Actividad Mostrar Gráfico

Se muestra en la figura 21 la última actividad de la aplicación. En esta se presenta un gráfico simple de dos dimensiones en el que se puede ver la representación de la medida respecto al tiempo, es decir, voltios por segundo (V/s).

Para realizar dicha implementación se ha recurrido a la dependencia *AndroidPlot* [14], la cual posibilita de una manera relativamente sencilla la representación de gráficos tanto estáticos como actualizables.

En el cuadro titulado **Implementación de la actividad Mostrar Gráfico** se detalla el código desarrollado para esta actividad. En primer lugar, se ha de convertir el vector de medidas al formato usado por la librería, por lo que se hace una conversión a la clase *XYSeries* desde un array numérico recuperado de la actividad anterior. Además, se le indica que esta secuencia de datos va a tener la etiqueta EMG en la leyenda del gráfico.

A continuación se define el formato que va a tener el gráfico. Para ello, se utilizó el definido en el fichero *line_formater.xml*, el cual se muestra en el cuadro **Fichero line_formater.xml**. Su contenido define una línea de grosor 0.5dp y color azúl sobre un fondo blanco.

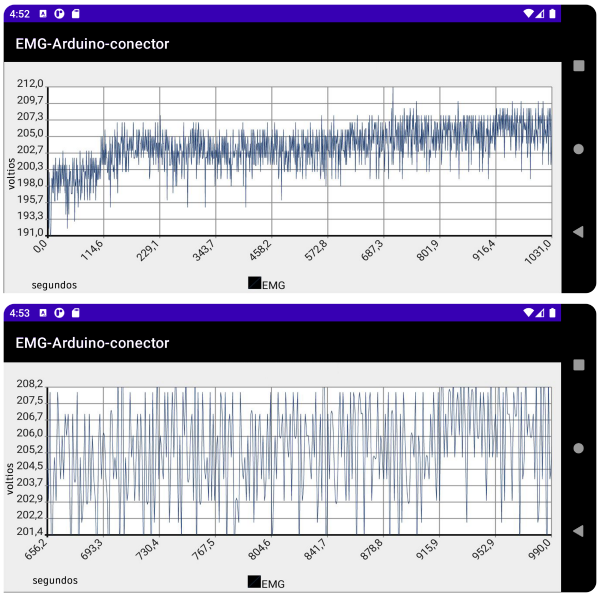

Figura 21: Actividad Mostrar Gráfico mostrando la representación completa y aumentada

Fichero line_formater.xml

```
<config  linePaint.strokeWidth="0.5dp"
         linePaint.color="#1B3A6B"
         fillPaint.color="#00000000" />
```

Para completar el formato del gráfico, se usa el método *setInterpolationParams()* para redondear la línea cerca de los bordes. Aunque este paso no es realmente necesario, se ha introducido para mejorar la visibilidad de la línea al modificar el tamaña el gráfico.

Por último, se añaden los datos y el formato al gráfico con el método *addSeries()* al que se le pasan ambos como parámetros. Con la programación realizada hasta este punto, ya sería posible generar la representación de los datos, sin embargo quedan algunos aspectos por personalizar.

Es por ello que se hace uso de las funciones *setDomainLabel()*, *setRangeLabel* y *setTitle()*, con las que se pueden sobrescribir los valores por defecto para los ejes y el título del gráfico (domain, range y A Simple XYPlot respectivamente). En este caso concreto, se ha decidido no asignar título y usar las magnitudes que representa cada eje.

Con el formato del gráfico definido, sólo resta generarlo. Se recurre pues al método *PanZoom.attach()* al que se le pasa como parámetro la configuración del gráfico. Al usar un objeto

de tipo *PanZomm*, se está definiendo un gráfico ampliable, por lo que se podrá navegar por la representación facilitando su comprensión.

Implementación de la actividad Mostrar Gráfico

```
XYPlot plot = findViewById(R.id.plot);

Intent intent = getIntent();
Number[] medidas = (Number[]) intent.getSerializableExtra(
    MostrarResultados.EXTRA_MEDIDAS);

XYSeries series1 = new SimpleXYSeries(Arrays.asList(medidas),
    SimpleXYSeries.ArrayFormat.Y_VALS_ONLY , "EMG");
LineAndPointFormatter series1Format = new LineAndPointFormatter(this,
    R.xml.line_formater);
series1Format.setInterpolationParams(new CatmullRomInterpolator.Params(
    3, CatmullRomInterpolator.Type.Centripetal));

plot.addSeries(series1, series1Format);
plot.setDomainLabel("segundos");
plot.setRangeLabel("Voltios");
PanZoom.attach(plot);
```

4.4.3. Pruebas y verificación

Dada la simplicidad del desarrollo realizado en esta última parte de la aplicación, las pruebas se realizaron directamente con el emulador incluido en el entorno de desarrollo Android Strudio.

Inicialmente, la generación del gráfico se probó usando los ficheros que se grabaron al inicio del proyecto (durante la fase de pruebas del sistema Arduino) y se fue variando el formato de la representación hasta obtener una visibilidad adecuada.

Una vez comprobado el funcionamiento del gráfico, se integró como una actividad más de la aplicación y se hizo el recorrido completo. En este, también se verificó que la información

del ejercicio se obtenía correctamente del servidor y se presentaba de una manera legible al usuario.

En general, el proceso de pruebas y verificación fue corto y con resultados satisfactorios.

5

Conclusiones y Líneas Futuras

En este último capítulo se recogen las conclusiones surgidas durante el desarrollo del proyecto, así como una breve lista de mejoras y nuevas líneas de trabajo en caso de continuar con el mismo.

5.1. Conclusiones

En el presente TFG se ha descrito un sistema capaz de capturar la señal biológica EMG mediante un sistema basado en Arduino y eHealth con el fin de ser enviada a través de una conexión Bluetooth a una aplicación móvil que la procesará y reenviará a la base de datos final. Una vez almacenada la información, la aplicación móvil se convierte en la interfaz del usuario para consultarla y actualizarla.

A lo largo del desarrollo de este proyecto, se ha aprendido a configurar y cablear tanto el sistema Arduino como sus módulos eHelath (para la medida del EMG) y HC-06 (para la comunicación Bluetooth). A su vez, se han trabajado los fundamentos del desarrollo de scripts con el lenguaje propio del Arduino y el uso de librerías externas. Por último, se ha investigado sobre la comunicación Bluetooth y la configuración de los dispositivos que la soportan mediante comandos AT.

Por otro lado, se ha adquirido el conocimiento necesario para controlar y configurar el entorno de desarrollo Android Studio con el fin de diseñar e implementar aplicaciones para diversos dispositivos y se ha aprendido a depurar el código mediante la facilidad de depuración USB y un dispositivo real.

También se ha estudiado el concepto de actividad y como integrar varias en una misma aplicación. Con este fin se han diseñado distintos *layouts* para cada una de ellas haciendo uso de diferentes tipos de controles, incluido el gráfico (para la representación gráfica del electromiograma) proporcionado por la dependencia *XYPlot*. Debido a esto último, ha resultado necesario comprender el funcionamiento de dependencias externas al proyecto y la manera de incorporarlas al mismo.

Finalmente, se ha aprendido sobre los threads o hilos dentro de una actividad y los distintos modos de enviar información ellos (uso de manejadores).

Respecto a la base de datos, se ha estudiado *postgresql* y sus ventajas en el proyecto (como el uso de vectores como tipo de dato). Se ha aprendido a usar DBeaver para configurar conexiones con bases de datos, crear scripts y consultar la información de las tablas.

Por último, se han investigado un lenguaje de programación funcional (Clojure) y los protocolos http para la implementación del servidor. Se ha aprendido sobre la arquitectura limpia y los sistemas de inyección de dependencias para usar de base el proyecto juxt/edge.

Además, se ha estudiado lo básico para la programación de una API usando las librerías *Yada* y *Bidi*, las que han provocado que se recabara información sobre los distintos verbos http, los códigos de respuesta y los fundamentos de las rutas web. Mencionar también la necesidad de estudiar los comandos básicos del lenguaje SQL y el uso de la librería *next.jdbc* para Clojure con el fin de conectar el servidor a la base de datos.

5.2. Líneas Futuras

Dado que uno de los principales objetivos era iniciar el estudio de un gran número de tecnologías, se ha realizado un trabajo ”horizontal” dejando diversos aspectos que pueden ser mejorados o desarrollados más en profundidad. Estas líneas futuras se detallan a continuación:

I. Quizá la mejora más llamativa sea ”encapsular” el sistema en un receptáculo más cómodo para el usuario. Para ello, sería también necesario añadir una batería que alimentase el sistema, lo que lo convertiría en un dispositivo completamente inalámbrico (salvo por los sensores).

II. Otra línea de investigación que merece la pena seguir es la ampliación del sistema de medida, pues la placa eHealth dispone de muchos más sensores que no se están usando en este proyecto. Por desgracia, el sensor encargado del electrocardiograma (ECG) no se encuentra disponible mientras el EMG está activo. Sin embargo, existen otras variantes en el mercado que pueden ser integradas o incluso sustituir al módulo actual.

III. Desde el punto de vista de la base de datos, la tabla de ejercicios se podría ampliar para añadir más tipos de medidas tomadas durante el mismo.

IV. De igual forma, se podrían desvincular las tablas paciente y profesional, de forma que el proyecto pudiese usarse por particulares para medir su rendimiento.

V. Todo el proyecto está centrado en el punto de vista del paciente, por lo que una de las principales líneas de desarrollo sería añadir soporte para los profesionales. Esto incluiría crear nuevos casos de uso en el servidor y métodos a la API para que profesionales registrados en la base de datos puedan ver los resultados de sus pacientes y generar diagnósticos.

VI. Respecto a la comunicación http, se debería añadir un sistema de seguridad o una encriptación, pues al tratarse datos personales y médicos de los usuarios podrían verse

muy expuestos.

vii. Ya sea en la aplicación, en el servidor, o en el propio *hardware* se podría implementar una mejora en el procesado de la señal, generando una representación más estable y clara de la misma.

viii. En el caso de la comunicación Bluetooth, esta podría hacerse de forma bidireccional, por lo que el sistema Arduino permanecería a la espera hasta recibir una petición de inicio de la actividad.

ix. Otro aspecto a mejorar en la aplicación sería el diseño y estructura del código, pues hay diversos tipos de datos y métodos que podrían definirse en clases separadas a las propias de la actividad. Esto además conduciría a una aplicación más fluida y de menor tamaño. Al igual que ocurre con las clases, muchos controles podrían definirse usando recursos XML para reducir la duplicidad de código.

x. Por último, la aplicación todavía puede ser mucho más adaptable. Se podrían implementar mejoras en la interfaz para ajustarla a todos los tipos de dispositivos, así como para respetar el tema del dispositivo que la ejecuta.

6
Referencias

[1] “The Making of Arduino”, David Kushner (26 Oct 2011). IEEE Spectrum.

[2] Web oficial Proyecto Arduino. (Creada, Enero 2006; Última consulta, Octubre 2021) https://www.arduino.cc/

[3] Verma, Manisha. International journal of engineering sciences & research technology working, operation and types of Arduino microcontrollers.

[4] Documentación e-Health de Cooking Hacks (Creada, Abril 2020; Última consulta, Marzo 2022) https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical.html#tutorial

[5] Petrellis, Nikos; Birbas, Michael K.; Gioulekas, Fotios. Evaluation of Sensors’ Precision in a Low Cost e-Health Monitoring System. En HAICTA. 2017. p. 377-382.

[6] Miller, Alex, Stuart Halloway, and Aaron Bedra. Programming Clojure. Sebastopol: Pragmatic Programmers, LLC, The, 2018. Print.

[7] Web oficial de Clojure. https://clojure.org/index

[8] Web de apoyo al desarrollo en Android, (Creada, Febrero 2009; Última consulta, Abril 2022), https://developer.android.com

[9] Bronzino, Joseph D. The Biomedical Engineering Handbook. 3rd ed. Boca Raton (Florida): CRC/Taylor & Francis, 2006, vol. 2 (Medical Devices and Systems), sección I.

[10] Carlo J. De Luca et al. “Decomposition of Surface EMG Signals.” Journal of Neurophysiology 96.3 (2006): 1646–1657. Web.

[11] Proyecto edge. (Última consulta, Febrero 2022) https://github.com/juxt/edge

[12] Manual de la librería juxt/yada. (Creada, Enero 2019; Última consulta, Febrero 2022) https://www.juxt.land/yada/manual/index.html

[13] Descarga Arduino Bluetooth Controller (Creada, Julio 2016; Última consulta, Abril 2022) https://play.google.com/store/apps/details?id=com.giumig.apps.bluetoothserialmonitor

[15] Página oficial XYPlot, (Última consulta, Abril 2022) http://androidplot.com/

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA