



Facultade de Informática

UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO  
GRAO EN ENXEÑARÍA INFORMÁTICA  
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

# **Aplicación de gestión para entrenamiento personal en instalaciones deportivas**

**Estudiante:** Pablo Collazo Collarte

**Dirección:** Mariano Javier Cabrero Canosa

A Coruña, xuño de 2020.



*A los "Luises"*



### **Agradecimientos**

En primer lugar, agradecerle a mi madre el confiar en mí día a día, apoyarme y aconsejarme sobre cada decisión a lo largo de estos años. A mi padre, por meterme de lleno en el mundo de la ingeniería. A mis hermanos por hacerme más amenas las horas en casa durante el trabajo. A los grandes amigos que he conocido en mis estancia de 4 años en A Coruña, especialmente, a Rubén y a Pedro. Y por supuesto, a mis amigos de toda la vida, a Iago, a Pedro, a Jenn y al resto de "Los Luises", por hacerme grande cada día. Gracias de corazón.



## Resumen

EL objetivo principal de este trabajo fin de grado es la realización de una aplicación de gestión para entrenamiento personal en instalaciones deportivas, orientado a entrenadores personales. Se basará en una aplicación en local (sin disponer de un servidor) que se programará principalmente en Python en su versión 3 y utilizando la biblioteca de GTK+ para la gestión de la interfaz gráfica de usuario para las distintas ventanas, ventanas emergentes (para mensajes de alerta) u otros elementos necesarios para la reproducción de las imágenes que utilizará el usuario por pantalla. Además se utilizará SQLite para almacenar todos los datos utilizados por la aplicación.

## Abstract

The main objective of this final degree project is to carry out a management application for personal training in sports facilities, aimed at personal trainers. It will be based on a local application (without having a server) that will be programmed mainly in Python in its version 3 and using the GTK + library in order to manage the graphical user interface for the different windows, pop-ups (for alert messages) or other necessary elements for the reproduction of the images that the user will use on screen. In addition, SQLite will be used to store all the data used by the application.

### Palabras clave:

- Interfaz Gráfica de Usuario (GUI)
- Wireframe
- Scrum
- Modelo-Vista-Controlador (MVC)
- Framework

### Keywords:

- Graphical User Interface (GUI)
- Wireframe
- Scrum
- Model-View-Controller (MVC)
- Framework





# Índice general

---

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1	Motivación . . . . .	1
1.2	Objetivos . . . . .	2
1.3	Estructura de la memoria . . . . .	2
<b>2</b>	<b>Estado del arte</b>	<b>5</b>
2.1	Trainingym . . . . .	5
2.2	Virtuagym . . . . .	6
2.3	Train2go . . . . .	6
<b>3</b>	<b>Fundamentos tecnológicos</b>	<b>7</b>
3.1	Herramienta para diseño de wireframes . . . . .	7
3.2	Lenguaje de programación . . . . .	8
3.3	Interfaz gráfica . . . . .	8
3.4	Base de Datos . . . . .	12
<b>4</b>	<b>Análisis de requisitos</b>	<b>15</b>
4.1	Idea principal sobre la aplicación y sus requisitos funcionales . . . . .	15
4.2	Diagrama de casos de uso . . . . .	16
4.2.1	Casos de uso para la gestión de usuarios . . . . .	17
4.2.2	Casos de uso para la gestión de grupos . . . . .	21
4.2.3	Casos de uso para la gestión de sesiones . . . . .	23
4.2.4	Casos de uso para la gestión de ejercicios . . . . .	24
4.2.5	Casos de uso para la gestión de programación de sesiones . . . . .	26
<b>5</b>	<b>Diseño</b>	<b>29</b>
5.1	Wireframes . . . . .	29
5.1.1	Wireframes para la gestión de usuarios . . . . .	30

5.1.2	Wireframes para la gestión de grupos . . . . .	33
5.1.3	Wireframes para la gestión de sesiones . . . . .	34
5.1.4	Wireframes para la gestión de ejercicios . . . . .	35
5.1.5	Wireframes para la gestión de programación de sesiones . . . . .	35
5.2	Modelado de la base de datos . . . . .	38
5.2.1	Diagrama Entidad-Relación . . . . .	38
5.2.2	Modelo relacional . . . . .	41
5.3	El patrón modelo-vista-controlador (MVC) . . . . .	41
<b>6</b>	<b>Planificación</b>	<b>45</b>
6.1	La metodología SCRUM . . . . .	45
6.2	Planificación inicial . . . . .	46
<b>7</b>	<b>Implementación</b>	<b>51</b>
7.1	Base de datos . . . . .	51
7.2	Interacción de la vista con el controlador . . . . .	55
7.2.1	Tablas resumen . . . . .	57
7.2.2	Formularios . . . . .	58
7.2.3	Ventanas de error o advertencia . . . . .	61
7.2.4	Ventanas emergentes . . . . .	62
7.2.5	Gráficas . . . . .	63
<b>8</b>	<b>Pruebas de funcionamiento y resultados</b>	<b>65</b>
8.1	Sección de usuarios . . . . .	65
8.2	Sección de ejercicios . . . . .	66
8.3	Sección de grupos . . . . .	68
8.4	Sección de sesiones . . . . .	68
8.5	Sección de programación . . . . .	68
<b>9</b>	<b>Conclusiones, lecciones aprendidas y líneas futuras de la aplicación</b>	<b>71</b>
9.1	Conclusiones . . . . .	71
9.2	Lecciones aprendidas . . . . .	71
9.3	Líneas futuras . . . . .	72
<b>A</b>	<b>Capturas completas de la aplicación</b>	<b>75</b>
<b>B</b>	<b>Contenido del directorio de la aplicación, enlace de gitHub y dependencias (requisitos no funcionales)</b>	<b>107</b>
B.1	Enlace de gitHub y contenido del directorio . . . . .	107

## ÍNDICE GENERAL

---

B.2	Ejecución . . . . .	107
B.3	Requisitos no funcionales . . . . .	108
	<b>Lista de acrónimos</b>	<b>109</b>
	<b>Glosario</b>	<b>111</b>
	<b>Bibliografía</b>	<b>113</b>



# Índice de figuras

---

3.1	Distintos tipos de botones que dispone GTK. . . . .	10
4.1	Casos de uso de la aplicación, de forma general. . . . .	16
4.2	Casos de uso específicos para la gestión de usuarios. . . . .	17
4.3	Casos de uso específicos para la gestión de usuarios. . . . .	21
4.4	Casos de uso específicos para la gestión de sesiones. . . . .	23
4.5	Casos de uso específicos para la gestión de ejercicios. . . . .	24
4.6	Casos de uso específicos para la gestión de programación de sesiones. . . . .	26
5.1	Ventana de inicio de la aplicación. . . . .	29
5.2	Ventana de gestión de usuarios. . . . .	30
5.3	Pop-up de confirmación de borrado de usuario. . . . .	31
5.4	Formulario de usuario. . . . .	31
5.5	Ejemplo de formato incorrecto. . . . .	32
5.6	Catálogo de lesiones. . . . .	32
5.7	Advertencia de introducción de datos repetidos. . . . .	33
5.8	Historial deportivo y estadísticas. . . . .	34
5.9	Detalles de sesión realizada. . . . .	34
5.10	Formulario de grupos. . . . .	35
5.11	Formulario de sesiones. . . . .	36
5.12	Formulario de ejercicios. . . . .	37
5.13	Pantalla principal de programación de sesiones. . . . .	37
5.14	Valoraciones post-sesión. . . . .	38
5.15	Formulario de creación de sesión programada. . . . .	38
5.16	Ventana emergente de horario incompatible. . . . .	39
5.17	Ventana emergente lesiones contraindicadas. . . . .	39
5.18	Diagrama ER. . . . .	40
5.19	Modelo relacional. . . . .	42

5.20	El modelo-vista-controlador [1]	43
6.1	Metodología SCRUM.	47
6.2	Diagrama de Gantt con las tareas distribuidas.	50
7.1	Ventana de inicio de la aplicación.	56
7.2	Entrada de texto para añadir y eliminar entradas de la lista de patologías de un usuario o ejercicio.	59
8.1	Ejemplo de mal formato de fecha	65
8.2	Advertencia por dato ya existente en base de datos.	66
8.3	Tras la advertencia, el sistema propone entradas similares.	66
8.4	Historial deportivo de un usuario.	67
8.5	Valoraciones post-sesión, variaciones de peso y lesiones adquiridas sobre un usuario.	67
8.6	Advertencia sobre el cambio de nombre de un ejercicio.	67
8.7	Ventana de error que indica que solo se puede introducir una foto y/o vídeo.	68
8.8	Gráfica con la media de las cargas en una sesión concreta.	68
8.9	Advertencia de horario no habitual para algún usuario o grupo.	69
8.10	Advertencia de lesión o patología incompatible con la realización de algún ejercicio.	69
8.11	Valoración post-sesión.	70
A.1	Pantalla nada más iniciarlo.	75
A.2	Pantalla de bienvenida.	76
A.3	Sección de usuarios con filtrado.	77
A.4	Advertencia de eliminación de usuario.	78
A.5	Formulario de modificación de un usuario.	79
A.6	Ejemplo de formato incorrecto de DNI.	80
A.7	Aviso de formato incorrecto.	81
A.8	Aviso de lesión introducida con palabras similares a alguna ya disponible.	82
A.9	Recomendación de lesiones disponibles.	83
A.10	Lesión ya en lista.	84
A.11	Catálogo de lesiones.	85
A.12	Historial deportivo de un usuario con su gráfica de evolución de peso y sesiones realizadas.	86
A.13	Valoración personal de una lesión y un usuario en concreto con posibilidad de introducir nuevo peso o lesiones adquiridas tras la sesión.	87
A.14	Sección de grupos.	88

A.15	Formulario de modificación de datos de grupos. . . . .	89
A.16	Aviso de algún dato vacío (en este caso, asignación de horario). . . . .	90
A.17	Sección de ejercicios. . . . .	91
A.18	Formulario de modificación de ejercicios. . . . .	92
A.19	Aviso al intentar introducir más de un elemento multimedia. . . . .	93
A.20	Solo se permite un vídeo y/o una foto, uno de cada. . . . .	94
A.21	La suma de las cargas no debe exceder el 100%. . . . .	95
A.22	Por consistencia, para modificar el nombre de un ejercicio se debe borrar y crearlo de nuevo. . . . .	96
A.23	Sección de sesiones. . . . .	97
A.24	Formulario de modificación de sesiones. . . . .	98
A.25	Aviso de sesión sin ejercicios. . . . .	99
A.26	Sección de programación de sesiones. . . . .	100
A.27	Formulario de creación de sesiones programadas. . . . .	101
A.28	Ejemplo de mal formato en los tiempos. . . . .	102
A.29	Aviso de día que no corresponde al grupo asignado. . . . .	103
A.30	Advertencia de usuario con lesión incluido en la programación con algún ejer- cicio contraproducente. . . . .	104
A.31	Valoración post-sesión. . . . .	105





# Índice de tablas

---

4.1	Caso de uso CU-1.1 CRUD de usuario . . . . .	17
4.2	CU-1.2 Añadir Lesión/Patología nueva a BD . . . . .	18
4.3	CU-1.3 Activar/desactivar. . . . .	19
4.4	CU-1.4 Mostrar historial deportivo y estadísticas . . . . .	19
4.5	CU-1.5 Añadir valoración post-sesión . . . . .	20
4.6	CU-1.6 Añadir nuevo peso. . . . .	20
4.7	CU-1.7 Añadir lesión tras entreno. . . . .	21
4.8	CU-2.1 CRUD de grupos . . . . .	22
4.9	CU-2.2 Buscar Usuarios . . . . .	22
4.10	CU-3.1 CRUD de sesiones . . . . .	23
4.11	CU-4.1 CRUD de ejercicios . . . . .	24
4.12	CU-4.2 Consultar catálogo de lesiones/patologías. . . . .	25
4.13	CU-5.1 CRUD sesión programada. . . . .	26
4.14	CU-5.2 Consultar sesiones. . . . .	27



# Introducción

---

## 1.1 Motivación

LA salud y el cuidado personal han sido en los últimos años dos aspectos muy influenciados por las modas y vertientes que han evolucionado mucho de lo tradicional, debido a la gran influencia de las redes sociales, los medios de comunicación y la gran globalización que vivimos hoy en día. Términos tales como "running", "HIIT" o "GAP" son solo ejemplos de palabras que se han inculcado muy naturalmente en nuestro lenguaje (a pesar de existir desde hace décadas), ya que el hecho de utilizar su terminología en inglés provoca que sea más llamativo e innovador hoy en día. Además, la figura del entrenador personal se ha vuelto cada vez más común como profesional capaz de diseñar rutinas de entrenamiento adaptadas a las características físicas y a las necesidades u objetivos de cada persona. De esta forma la eficacia de la práctica deportiva es mayor y los resultados son mejores. Debido a la situación que se ha vivido en los últimos meses por culpa del COVID-19, la población se ha percatado de la importancia que tiene el deporte y la vitalidad que puede aportarle a una persona en situaciones difíciles.

Todo ello no sería posible llevarlo a cabo de una forma correcta y segura sin profesionales del deporte especializados. Estos profesionales en estos meses de confinamiento en los que se ha llevado a cabo este TFG, han permitido a una gran parte de la población mantener una regularidad con una serie de rutinas de ejercicios para realizar desde casa de una manera virtual. De todas formas, una vez finalizado este confinamiento, la actividad en los gimnasios volverá a la normalidad, con la consiguiente necesidad de gestionar dichos entrenamientos, directamente al público con una atención personalizada en muchos casos. De aquí surge precisamente la necesidad de disponer de un sistema que proporcione a los monitores de gimnasios una gestión de las actividades impartidas en dichas instalaciones, distribuida en grupos de entrenamiento, para poder así crear sesiones personalizadas y disponer de una base de datos en la cual poder almacenar dicha información.

## 1.2 Objetivos

El objetivo principal es desarrollar una aplicación destinada a los monitores y entrenadores que permita suplir la gestión manual que se suele llevar en los gimnasios, en tanto a la creación de sesiones de entrenamiento por grupos. De esta forma se creará de una manera sencilla e intuitiva sesiones programadas en base a criterios que el propio monitor elija. Así, podremos almacenar a los usuarios por grupos que realicen por ejemplo, una sesión que busque hipertrofia muscular, otra sesión que busque bajar de peso, otra ganar fuerza etc. De esta forma vamos a disponer de una gestión centralizada de usuarios y sesiones de entrenamiento sustituyendo al papel.

## 1.3 Estructura de la memoria

A continuación se van a describir los distintos capítulos que van a componer la memoria:

- **Capítulo 1 - Introducción:** En este capítulo de la memoria se hace una introducción acerca de la temática del trabajo, los objetivos a conseguir y la estructura del presente documento.
- **Capítulo 2 - Estado del arte:** En este capítulo se exponen herramientas de entrenamiento personal que se encuentran actualmente en el mercado, describiéndolas brevemente.
- **Capítulo 3 - Fundamentos tecnológicos:** En este capítulo se exponen las herramientas utilizadas para la programación de la aplicación y su diseño.
- **Capítulo 4 - Análisis de requisitos:** En este capítulo se analizan los requisitos previos al desarrollo de la aplicación y sus casos de uso.
- **Capítulo 5 - Diseño:** En este capítulo se muestran patrones de diseño utilizados y los wireframes que van a servir de guía en la implementación de la aplicación a lo largo del ciclo de desarrollo.
- **Capítulo 6 - Planificación:** En este capítulo se divide el trabajo en distintos apartados para planificarlo a lo largo de los meses de duración del mismo, haciendo uso de la metodología Scrum.
- **Capítulo 7 - Implementación:** En este capítulo se explicarán los aspectos más relevantes acerca de la implementación de la aplicación, tanto de la lógica como de su interfaz gráfica.

- **Capítulo 8 - Pruebas de funcionamiento y resultados:** En este capítulo se expone el resultado final de la aplicación en distintas figuras, comprobando su correcto funcionamiento.
- **Capítulo 9 - Conclusiones, lecciones aprendidas y líneas futuras de la aplicación:** Por último, en este capítulo se concluirá el trabajo con unas conclusiones, valoraciones personales sobre su desarrollo y posibles mejoras en tanto a las líneas futuras de la aplicación.



# Estado del arte

---

En los últimos años la utilización de aplicaciones personales con las cuales gestionar todo tipo de eventos o actividades, ya sean deportivas, culturales o de mero ocio, se han popularizado en gran medida ya que disponen de una gran facilidad de utilización por parte de los usuarios, ya sea por tener una interfaz intuitiva o un almacenamiento necesario no muy elevado (muchas de ellas, en la nube). En este apartado se van a exponer tres populares aplicaciones que cubren el objetivo definido en el anterior apartado.

## 2.1 Trainingym

Trainingym es una aplicación que está especialmente dirigida a la gestión de un centro deportivo tales como clubes o gimnasios que permite automatizar las tareas y gestionar de una forma centralizada a sus clientes. Además, proporciona una interfaz dedicada que permite mostrar las actividades, salas, horarios u otro tipo de información, desde la perspectiva del cliente del gimnasio. Como indica su página oficial, hace hincapié en el feedback entre cliente y monitor lo que permite una mejora continua de las clases[2]. Trainingym permite el almacenamiento de información variada acerca de un usuario para así poder monitorizar su estado físico en base a una multitud de datos basados en los resultados que va obteniendo a medida que va realizando las rutinas.

Además esta aplicación permite crear dietas personalizadas para los clientes, dando una capacidad extra funcional más. Trainingym es una aplicación de almacenamiento en la nube por lo tanto se necesita de una conexión a internet para realizar cualquier operación que necesitemos.

## 2.2 Virtuagym

Otro ejemplo de software dedicado a la gestión de instalaciones deportivas es Virtuagym. Esta aplicación fue lanzada en 2008 en un primer momento dirigida a los usuarios para llevar un seguimiento de entrenamientos individuales. Posteriormente en 2012, dio el saltó al ámbito profesional de gestión de empresas y clubes deportivos. Esta aplicación también basada en almacenamiento en la nube, nos permite no solo gestionar nuestros clientes, sus rutinas y dietas personalizadas, sino que incluye también la gestión de pagos o facturaciones de estos. Este añadido, lo convierte en no solo un gestor deportivo, sino que también permite llevar las cuentas de una manera centralizada, todo ello en una aplicación [3].

En tanto al entrenamiento personal, permite organizar las sesiones de entrenamiento y gestionar los ejercicios a realizar, horarios, salas y material utilizado, además de poder monitorizar de una forma centralizada los resultados de los clientes para poder observar sus progresos. Cabe destacar que esta plataforma ha implantado un apartado dedicado al entrenamiento en casa, permitiendo subir vídeos y explicaciones de las distintas sesiones de entrenamiento y poder llevar a cabo estos de una forma remota.

## 2.3 Train2go

Train2Go es una start up gallega que fue fundada hace 7 años. Ha implementado un gestor de entrenamientos online orientado a entrenadores, que permite centralizar toda la información acerca de los entrenamientos en la nube. Por supuesto, al tratarse de una página web, es multiplataforma, permitiendo la gestión de los entrenamientos de los clientes en cualquier parte con conexión a internet. No dispone de una interfaz tan intuitiva como los otros dos anteriores, de todas formas, dispone de muchas funcionalidades que van a facilitar la organización de los entrenadores, sustituyendo los canales de comunicación con los clientes, y las formas engorrosas de almacenar entrenamientos, en una sola aplicación web. [4]

Para la gestión de entrenamiento personal, esta aplicación permite organizar sesiones de entrenamiento pudiendo planificar dichos entrenamientos con una periodicidad personalizada para los distintos grupos de entrenamiento. Además, proporciona un sistema de alarmas para alertar al entrenador de que alguno de sus alumnos lleva mucho tiempo sin ser atendido, para así mantener al entrenador actualizado y evitar dejar a algún cliente atrás.



# Fundamentos tecnológicos

---

Disponemos de metodologías o métodos de trabajo bastante arraigados en el mundo de la programación de interfaces ya sean web o locales, que proporcionan una base de facto (una especie de "plantilla") para la mayor parte de las aplicaciones para interacción directa con el usuario de a pie. En este trabajo fin de grado, el término que mejor explica el objetivo de este, es la creación de una **interfaz gráfica de usuario** (GUI), tema central del trabajo. Según wikipedia la definición más general para un GUI sería "un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz" [5]. Es lo que el usuario observa y la herramienta con la cual se comunica con la máquina, permaneciendo todo detalle a bajo nivel oculto. En este apartado se expondrán las herramientas empleadas para el desarrollo de dicha interfaz gráfica para la aplicación a desarrollar.

### 3.1 Herramienta para diseño de wireframes

Antes de comenzar la programación de la aplicación, es necesario realizar un diseño previo para establecer una guía a seguir durante el proceso de programación. La herramienta empleada para realizarlos será Balsamiq [6], una herramienta de pago que nos permite una prueba de 30 días de uso tanto en local como en su herramienta web. Es una herramienta muy completa que permite diseñar ventanas con muchos de los elementos que se van a utilizar en la aplicación tales como todo tipo de botones, ventanas emergentes, barras de navegación o tablas de datos en un formato sencillo y claro que permiten representar fielmente las estructuras a realizar.

## 3.2 Lenguaje de programación

Tal y como se indica en la introducción se ha elegido como lenguaje Python en su versión 3. Este lenguaje de programación dinámica es uno de los más populares actualmente. Es utilizado por grandes compañías tales como: Instagram, el cual emplea un framework web llamado Django (basado en Python completamente); Google, utilizado en su backend junto a otros como C, C++ o Java; o Spotify como su propia página de desarrollo explica: "En Spotify, los dos lugares principales en los que utilizamos Python son los servicios de backend y el análisis de datos.[...]La velocidad es un gran foco para Spotify. Python encaja bien en esta mentalidad, ya que nos da grandes victorias en velocidad de desarrollo." [7].

Python es un lenguaje interpretado, dinámicamente tipado (una variable puede tomar valores de distinto tipo) y multiparadigma ya que tiene soporte para orientación a objetos, para la programación imperativa e incluso para la programación funcional. Su facilidad de extensión incluye la inclusión de módulos nuevos escritos en otros lenguajes como C o C++.

Python incluye una librería estándar muy completa además de poder incluir librerías escritas en C, por lo tanto lo hace un lenguaje con mucha soporte y con capacidad para tareas de muchos tipos. Además de diversas librerías, dispone de importantes frameworks web muy extendidos, tales como Django, el cual utiliza el patrón MVC. Este framework facilita la creación de sitios web complejos, poniendo énfasis en el re-uso, la conectividad y extensibilidad. [8]

La elección de este lenguaje vino determinada por el gran soporte que proporciona GTK para éste y debido a la facilidad de uso de las librerías necesarias para la representación de gráficos en la aplicación (matplotlib [9]). Además, el hecho de ser dinámicamente tipado, beneficia en gran medida ya que en una aplicación con la que se interacciona con muchos widgets por pantalla, se emplean muchas variables que se repiten en varias pantallas, por lo que el re-uso es una característica que facilita el trabajo y ahorra memoria.

## 3.3 Interfaz gráfica

Además de emplear Python para todo el código del programa, es necesario incluir algún tipo de framework o biblioteca que permita representar las ventanas de dicho programa. Este desarrollo será llevado a cabo gracias a GTK+ (The GIMP Toolkit) [10]. Esta herramienta dispone de implementaciones para una gran variedad de lenguajes tales como C++, C#, Java, Python etc. En referencia a la implementación usada, disponemos de una documentación en línea bastante extensa de sus componentes principales. La implementación para Python de GTK se encuentra en el paquete llamado **PyGObject** [11], el cual proporciona enlaces para bibliotecas basadas en GObject tales como GTK, GStreamer, WebKitGTK, GLib, GIO etc.

Además de GTK, usaremos bibliotecas como GIO o GdkPixbuf.

La herramienta GTK está compuesta por muchos elementos gráficos pero a su vez, estos elementos necesitan de un control para poder sacarle partido a estos. Este control se mantiene con un modelo de programación **dirigido por eventos**. Esto consiste en que toda acción, ya sea presionando un botón, o haciendo click en un enlace, es percibida como un evento el cual provocará una acción en el sistema. Esta señal es percibida ya que GTK+, permanece en un bucle de "escucha" constante para detectar estos eventos.

```
1 import gi
2
3 gi.require_version("Gtk", "3.0")
4 from gi.repository import Gtk
5
6 win = Gtk.Window()
7 win.connect("destroy", Gtk.main_quit)
8 win.show_all()
9 Gtk.main()
```

El código anterior es un ejemplo muy simple de utilización del Gtk, que simplemente representa una ventana vacía. A parte del importado de la biblioteca gi y su componente Gtk en las líneas 1 y 4, se requiere indicar la versión mínima que se debe utilizar de Gtk, en la línea 3. A continuación se inicializa el elemento "Window", que constituye la ventana en sí, la conexión de la señal emitida por el elemento con una función interna de Gtk (permite que al presionar el botón de cerrar la ventana, finalice el bucle de Gtk) en la línea 7 que se explicará en el siguiente párrafo; y las dos últimas líneas, la primera hace que se muestre por pantalla la ventana y la siguiente constituye el bucle anteriormente mencionado.

Los elementos con los que el usuario interactúa son los llamados **widgets**. Estos, reciben los eventos por parte del usuario y emiten una o más señales que tienen como consecuencia la invocación de funciones que hayamos asociado a dicha señal previamente. Un claro ejemplo de ello nos lo muestra el soporte de Gtk para Python [12] en el siguiente código:

```
1 handler_id = widget.connect("event", callback, data)
```

Como podemos ver en el código anterior, vemos que el objeto widget (que se inicializa previamente) dispone de la función "connect". Esta función es la que permite asociar un evento a la ejecución de una función en particular. Cada tipo de Widget va a disponer de un evento concreto, ya que por ejemplo un botón simple, su evento es simplemente un "click" sobre él, un botón de tipo Switch sería la modificación de su estado (activado -> desactivado) o en

una lista de elementos marcar alguno de ellos. En caso de envío de estas señales, la función "callback" (en este caso) se ejecutará cuando la señal sea enviada. El campo "data" de la figura constituyen los argumentos de entrada de la función, por lo tanto es un campo opcional. Dicha función "connect" devolverá un id que representa la señal en particular y la función de respuesta, para que podamos utilizar en alguna situación esta asociación, por ejemplo en el caso de que queramos bloquear la señal por alguna razón.

La galería de widgets de los que dispone GTK es bastante completa, desde botones para interaccionar (3.1), etiquetas, imágenes, entradas de texto, contenedores a distintos tipos de ventanas emergentes.

### Buttons

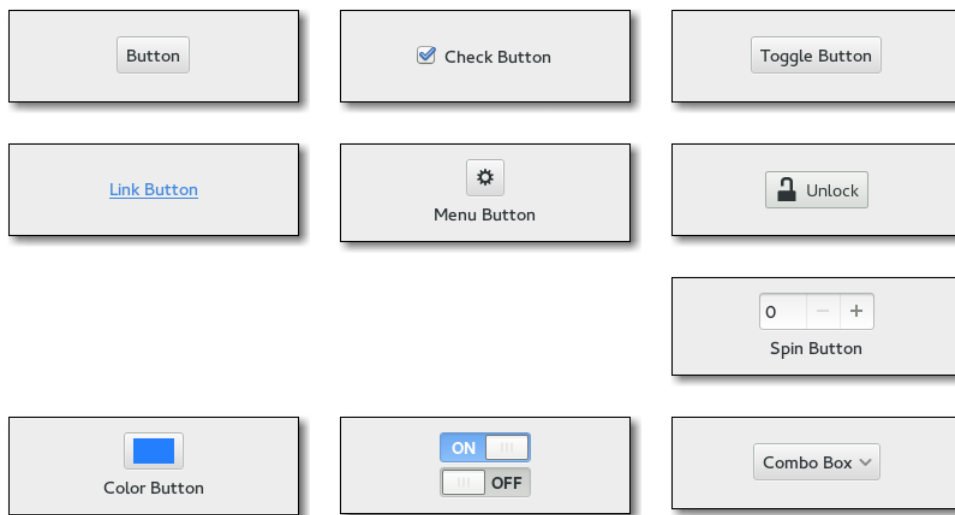


Figura 3.1: Distintos tipos de botones que dispone GTK.

Dichos widgets se pueden alinear fácilmente en la interfaz de la aplicación, organizándolos en columnas, filas o cuadrículas, dejando que GTK automáticamente ajuste estos según su tamaño y márgenes adecuados. Esta distribución se lleva a cabo gracias a la ayuda de otro elemento de GTK los contenedores o **Layout Containers**. Existen varios tipos de contenedores pero los más comunes son los tipo **Box** (tanto horizontal, como vertical) y los tipo **Grid**.

Los primeros se utilizan para empaquetar distintos widgets formando una fila (pudiendo ser horizontal o vertical). Para determinar un margen o espaciado entre los widgets particular, en el momento de inicializar el `Gtk.Box` podemos declarar una serie de propiedades, que se aplicarán a todo el `Box`:

```
1 Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=10,
margin=12)
```

En el código anterior se le aplican las propiedades de la orientación que queremos que tome el box, el espaciado entre widgets y el margen que mantendrá el box con los otros elementos de la pantalla (ambos en píxeles). Para introducir los widgets simplemente utilizamos la función `pack_start` o `pack_end`, según queramos que comience a añadir por la izquierda o por la derecha. Los widgets se añadirían de la forma:

```
1 links = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=10,
2     margin=10)
3 links.pack_start(self.Ini, False, False, 0)
4 links.pack_start(self.Programacion, False, False, 0)
5 links.pack_start(self.Grupos, False, False, 0)
```

Dicha función colocará los distintos widgets en el orden escrito en el código. Los parámetros de entrada serían el widget que queremos añadir, un booleano que indica si queremos que tenga la capacidad de poder disponer de espacio extra (en el caso de que lo haya), otro que indica si este espacio dado se asigne al widget, en lugar de solo rellenarlo, y por último, el número de píxeles extra a colocar entre el propio widget y sus vecinos. Con estas opciones adicionales permite refinar de mejor forma la precisión de la posición del widget en la pantalla.

En tanto al tipo **Grid**, se trata de un contenedor el cual distribuye los widgets en filas y columnas formando una cuadrícula como su propio nombre indica. Como en el tipo Box, también le podemos añadir una serie de propiedades, tales como margen, espaciado entre columnas o espaciado entre filas:

```
1 inicio = Gtk.Grid(margin=0, column_spacing=10, row_spacing=10)
```

Las funciones utilizadas para añadir los widgets es similar a la anterior con unos parámetros similares:

```
1 grid.attach(button2, 1, 0, 2, 1)
2 grid.attach_next_to(button3, button1, Gtk.PositionType.BOTTOM, 1, 2)
3 grid.attach_next_to(button4, button3, Gtk.PositionType.RIGHT, 2, 1)
4 grid.attach(button5, 1, 2, 1, 1)
5 grid.attach_next_to(button6, button5, Gtk.PositionType.RIGHT, 1, 1)
```

Como podemos observar en este extracto de código, vemos que tenemos dos tipos de funciones, `attach` y `attach_next_to`. La primera se utiliza para colocar un widget en el primer hueco en la cuadrícula libre. Los parámetros de entrada son el propio widget, el nº de columna donde colocar el lado izquierdo del widget, el nº de fila donde colocar la parte superior del

widget, el nº de columnas que utilizará el widget y por último el nº de filas que ocupará el widget (estos dos últimos definen el tamaño de dicho widget respecto al resto, dentro del Grid). En tanto a la segunda función, se utiliza para colocar un widget al lado de otro dado, según los parámetros introducidos, que son el widget a colocar, el widget que debe tomar como referencia, el lado de la cuadrícula vecina (el anterior parámetro) donde colocar el widget, el nº de columnas que empleará y el número de filas.

Existen gran variedad de widgets para colocar en los contenedores descritos anteriormente, que se explicarán según se vayan mostrando en siguientes apartados, con ejemplos reales de la propia aplicación.

### 3.4 Base de Datos

Para el almacenamiento de usuarios, de ejercicios, de sesiones etc. es necesario disponer de una base de datos para su mantenimiento, además de una forma sencilla de actualizar, consultar y borrar dichos datos. La base de datos elegida es SQLite, un sistema de gestión de base de datos relacional que está contenida en una biblioteca escrita en C. A diferencia de otras bases de datos tales como PostgreSQL o MySQL que utilizan un sistema de gestión cliente-servidor, SQLite utiliza su biblioteca pasando a ser directamente parte del propio programa, ya que SQLite "lee y escribe directamente en ficheros de disco ordinarios. Proporciona una base de datos SQL completa con múltiples tablas, índices, disparadores y vistas que están contenidos en un solo archivo de disco." [13]. Por lo tanto, esta característica "serverless" se traduce en que la lectura y escritura de pequeñas imágenes (con un tipo de dato llamado blob) es un 35% más rápido que con lecturas y escrituras ejecutadas a ficheros individuales en disco, según expone el soporte disponible en la web de SQLite. Ya que la aplicación va a hacer uso de imágenes tanto en el apartado de usuarios como en el de ejercicios, es un punto a favor bastante importante a tener en cuenta, en tanto a rendimiento. SQLite dispone de una instalación simple y su uso es trivial, además al tratarse de una aplicación de escritorio, el retardo en las consultas sería mínimo al disponer de las tablas directamente en memoria.

Para integrar el sistema de gestión de bases de datos de SQLite en la aplicación, Python dispone de un módulo para este, el `sqlite3`. Dicho módulo se importaría al código y con tres simples funciones se adquiere la capacidad de crear tablas y hacer todo tipo de operaciones utilizando queries de sentencias SQL. Para usar dicho módulo, se debe crear un objeto *Connection* que representa la base de datos, almacenándola en un fichero de texto de cualquier directorio del sistema.

```
1 import sqlite3
2 conn = sqlite3.connect('example.db')
3 c = conn.cursor()
```

```
4 # Create table
5 c.execute('''CREATE TABLE stocks
6             (date text, trans text, symbol text, qty real, price
7              real)''')
8 # Insert a row of data
9 c.execute("INSERT INTO stocks VALUES
10          ('2006-01-05', 'BUY', 'RHAT', 100, 35.14)")
11 # Save (commit) the changes
12 conn.commit()
13
14 # We can also close the connection if we are done with it.
15 # Just be sure any changes have been committed or they will be lost.
16 conn.close()
```

En el código anterior vemos un ejemplo proporcionado por el soporte oficial de Python relacionado con el módulo SQLite [14]. Como se ha nombrado anteriormente, es necesario crear un objeto *Connection* referenciando un fichero que en donde se almacenará la base de datos. Posteriormente, se crea un objeto *Cursor* a partir de la conexión previa el cual es una área de trabajo temporal en la memoria del sistema para almacenar y extraer de este, por ejemplo, el resultado de una sentencia "select". Con este podemos ejecutar las sentencias sql necesarias ya sean creación de tablas, insertados, actualizaciones o borrado de filas de una tabla. Por último, se confirman los cambios realizados y se cierra la conexión con la base de datos. En el capítulo de implementación 7, se detallará concretamente cómo se ha implementado en la aplicación.





# Análisis de requisitos

---

Antes de comenzar a codificar las distintas ventanas que conformarán la aplicación, es necesario hacer un análisis y un diseño previo de lo que va a constituir la aplicación. Para ello, se plantea un primer análisis de toda la aplicación, realizando un diagrama de casos de uso con el cual exponer las funcionalidades de las que va a disponer el programa y conocer el flujo general de funcionamiento. Una vez hecho esto se traduce estas funcionalidades en Wireframes que muestran de primera mano la aproximación del resultado final de este proyecto.

## 4.1 Idea principal sobre la aplicación y sus requisitos funcionales

Tal y como se indicaba en el anteproyecto del TFG, se basa en una aplicación de gestión para entrenamiento personal en instalaciones deportivas. En ella, se puede dar de alta a usuarios asociándoles una serie de datos de contacto que se van a almacenar en la base de datos. Se podrán añadir al usuario una foto de perfil y datos acerca de lesiones o patologías que pueda sufrir el usuario, disponibles en la base de datos o dándolas de alta en el sistema. Estos usuarios se podrán agrupar en grupos de entrenamiento y para los cuales es posible planificar una serie de sesiones de entrenamiento distribuidas a lo largo de distintos horarios y días de la semana. Estas sesiones están compuestas por ejercicios registrados previamente en la aplicación, asociándole una serie de información como una explicación de cómo se realiza el ejercicio, porcentaje de carga de trabajo (aeróbico, coordinación, equilibrio y fuerza), material utilizado o recomendaciones sobre si es adecuado o no para ciertos tipos de patologías o lesiones que pudiera tener alguna persona. De esta manera, a la hora de planificar una sesión de entrenamiento, la aplicación puede lanzar una advertencia automáticamente informando de que el usuario asociado X dispone de esa misma lesión. Una vez realizadas estas sesiones, el administrador de esta aplicación podrá realizar comentarios post-sesión tanto por sesión como por usuario, que puede almacenar en la aplicación y estar disponible para consultas

posteriores. Muchos de los datos introducidos en la aplicación pasan a través de una serie de controles para verificar, ya sea, que estén sintácticamente correctos, que sean coherentes, o simplemente advertencias de que el dato puede no tener sentido o recomendación de una acción mejor. Un ejemplo de esto es, en el caso de introducir datos incorrectos a la hora de dar de alta a un usuario, que el formato del correo electrónico sea sintácticamente incorrecto, por lo cual, la aplicación nos lanzaría un mensaje de error. Otro ejemplo de advertencia es que intentemos introducir en el catálogo de lesiones una lesión que puede que ya tengamos almacenada en la base de datos para así evitar introducir datos repetidos. En la descripción de los propios casos de uso veremos estas situaciones en detalle.

## 4.2 Diagrama de casos de uso

Una vez planteados los requisitos de la aplicación, el siguientes paso es analizar como serán los casos de uso de la aplicación una vez nos encontramos dentro de esta. El caso de uso general lo encontramos en la figura 4.1. De una forma general, se observa que la aplicación se basa en una interacción directa con la base de datos, ya sea para insertar, modificar o consultar los datos guardados, tanto de usuarios, de grupos, de sesiones, de ejercicios y programación de sesiones.

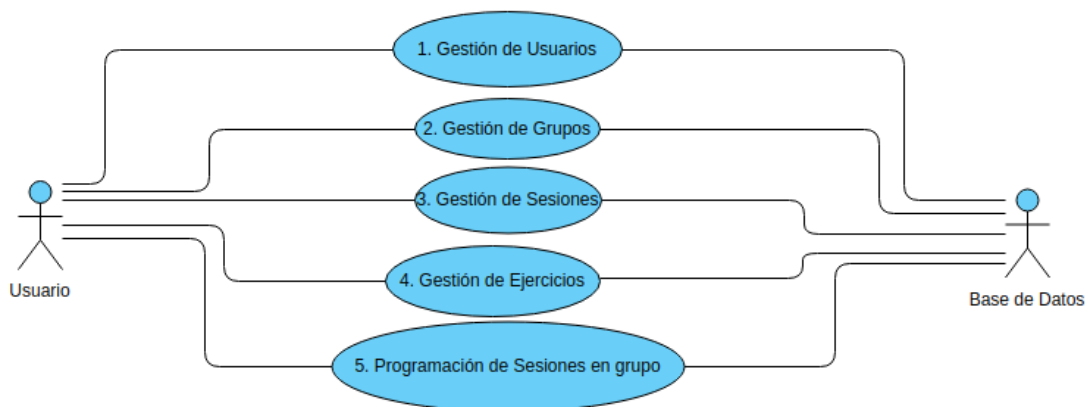


Figura 4.1: Casos de uso de la aplicación, de forma general.

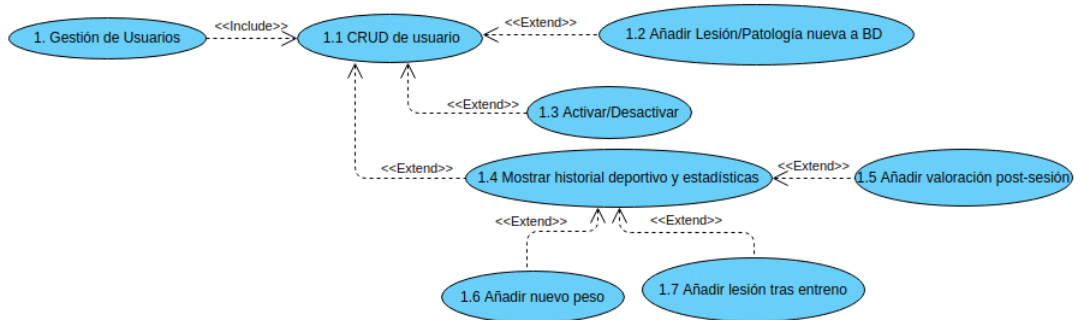


Figura 4.2: Casos de uso específicos para la gestión de usuarios.

### 4.2.1 Casos de uso para la gestión de usuarios

<b>CU-1.1</b>	<b>CRUD de usuario</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	- 1. Gestión de Usuarios	
<b>Precondición</b>	El usuario ha iniciado la aplicación.	
<b>Descripción</b>	El usuario podrá crear, consultar, actualizar o borrar un usuario de la base de datos.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario entra en la ventana de gestión de usuarios a través del botón "Usuarios" de la ventana.
	2	El sistema de gestión de bases de datos devuelve los usuarios disponibles en caso de que haya.
	3	El usuario crea, actualiza o elimina un usuario.
	4	Si el usuario confirma los datos cambiados, 4.1 El SGBD actualiza/inserta/borra los datos dados.
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
	3	Si el usuario introduce algún campo en un formato erróneo, E.1 El sistema avisa con una ventana emergente del error introducido
<b>Comentarios</b>		

Tabla 4.1: Caso de uso CU-1.1 CRUD de usuario

<b>CU-1.2</b>	<b>Añadir Lesión/Patología nueva a BD</b>			
<b>Versión</b>	1.0			
<b>Dependencias</b>	- 1. Gestión de Usuarios - 1.1 CRUD de usuario			
<b>Precondición</b>	El usuario ha iniciado la aplicación y se encuentra en el formulario de usuario (ya sea por creación de un usuario como por actualización).			
<b>Descripción</b>	El usuario añade o elimina una lesión de un usuario.			
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>		
	1	El usuario entra en el formulario de usuario.		
	2	Si el usuario ya esta creado y dispone de lesiones/patologías,		
		2.1	El SGBD devuelve las lesiones/patologías asociadas al usuario.	
		2.2	Si el usuario borra alguna patología el SGBD refleja esta actualización en la BD.	
	3	El usuario introduce por teclado lesiones/patologías en el campo correspondiente o las selecciona del catálogo.		
	4	Si la BD encuentra una lesión/patología similar en el catálogo,		
		4.1	El sistema las muestra para que el usuario las elija, para evitar introducir repetidos.	
		4.2	Si el usuario elige las recomendadas por el sistema,	
			4.2.1	El sistema actualiza la tabla del usuario pero no, la introduce en el catálogo general. En caso contrario, las introduce en ambas.
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.			
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>		
	3	Si el usuario introduce alguna lesión o patología que ya tiene el usuario,		
E.1		El sistema avisa con una ventana emergente de advertencia, indicando que ya se encuentra en la BD.		
<b>Comentarios</b>	El sistema detecta lesiones/patologías similares en base a las palabras introducidas, analizándolas individualmente.			

Tabla 4.2: CU-1.2 Añadir Lesión/Patología nueva a BD

<b>CU-1.3</b>	<b>Activar/Desactivar</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	- 1. Gestión de Usuarios - 1.1 CRUD de usuario	
<b>Precondición</b>	El usuario ha iniciado la aplicación y ha seleccionado un usuario en concreto.	
<b>Descripción</b>	Utilizado para activar o desactivar un usuario. De esta forma, si se encuentra desactivado no aparecerá a la hora de planificar sesiones.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario selecciona un usuario disponible.
	2	El usuario presionando un botón activa o desactiva al usuario, según el estado en que se encuentre.
	3	El sistema almacena el cambio en la BD.
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos. El usuario cambiado no aparecerá seleccionable en la planificación de sesiones.	

Tabla 4.3: CU-1.3 Activar/desactivar.

<b>CU-1.4</b>	<b>Mostrar historial deportivo y estadísticas</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	- 1. Gestión de Usuarios - 1.1 CRUD de usuario	
<b>Precondición</b>	El usuario ha iniciado la aplicación y ha seleccionado un usuario en concreto.	
<b>Descripción</b>	En este caso de uso se muestra una gráfica de evolución del peso y el historial de sesiones realizadas.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario selecciona un usuario disponible y accede al historial.
	2	El sistema muestra una gráfica que muestra la evolución del peso de la persona y una tabla con las sesiones hechas por el usuario.

Tabla 4.4: CU-1.4 Mostrar historial deportivo y estadísticas

<b>CU-1.5</b>	<b>Añadir valoración post-sesión</b>		
<b>Versión</b>	1.0		
<b>Dependencias</b>	<ul style="list-style-type: none"> <li>- 1. Gestión de Usuarios</li> <li>- 1.1 CRUD de usuario</li> <li>- 1.4 Mostrar historial deportivo y estadísticas</li> </ul>		
<b>Precondición</b>	El usuario ha iniciado la aplicación y ha seleccionado un usuario en concreto, además de acceder a su historial de sesiones.		
<b>Descripción</b>	En este caso de uso se añade una valoración tras la realización de la sesión.		
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>	
	1	El usuario selecciona una sesión concreta del historial.	
	2	En caso de que el sistema ya disponga de una valoración,	
		2.1	El sistema devuelve la valoración guardada.
	3	El usuario añade/actualiza la valoración post-sesión.	
4	El SGBD guarda los datos introducidos.		
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.		

Tabla 4.5: CU-1.5 Añadir valoración post-sesión

<b>CU-1.6</b>	<b>Añadir nuevo peso</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	<ul style="list-style-type: none"> <li>- 1. Gestión de Usuarios</li> <li>- 1.1 CRUD de usuario</li> <li>- 1.4 Mostrar historial deportivo y estadísticas</li> </ul>	
<b>Precondición</b>	El usuario ha iniciado la aplicación y ha seleccionado un usuario en concreto, además de acceder a su historial de sesiones.	
<b>Descripción</b>	En este caso de uso se añade un peso nuevo adquirido tras la realización de la sesión.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario selecciona una sesión concreta del historial.
	2	El usuario añade un peso nuevo adquirido tras la sesión.
	3	El SGBD guarda los datos introducidos.
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.	

Tabla 4.6: CU-1.6 Añadir nuevo peso.

<b>CU-1.7</b>	<b>Añadir lesión tras entreno</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	<ul style="list-style-type: none"> <li>- 1. Gestión de Usuarios</li> <li>- 1.1 CRUD de usuario</li> <li>- 1.4 Mostrar historial deportivo y estadísticas</li> </ul>	
<b>Precondición</b>	El usuario ha iniciado la aplicación y ha seleccionado un usuario en concreto, además de acceder a su historial de sesiones.	
<b>Descripción</b>	En este caso de uso se añade una lesión nueva adquirida tras la realización de la sesión de entrenamiento.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario selecciona una sesión concreta del historial.
	2	El usuario introduce la lesión adquirida o la selecciona del catálogo de lesiones.
	3	El SGBD guarda los datos introducidos.
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.	

Tabla 4.7: CU-1.7 Añadir lesión tras entreno.

#### 4.2.2 Casos de uso para la gestión de grupos

Una vez analizados los casos de uso específicos para la gestión de usuarios, vemos el siguiente grupo de casos de uso, los relacionados con la gestión de grupos de usuarios, como refleja la figura 4.3. Posteriormente veremos sus especificaciones en las tablas siguientes.



Figura 4.3: Casos de uso específicos para la gestión de usuarios.

<b>CU-2.1</b>	<b>CRUD de grupos</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	- 2. Gestión de Grupos	
<b>Precondición</b>	El usuario ha iniciado la aplicación. Se deben tener almacenados, usuarios para la formación de grupos.	
<b>Descripción</b>	El usuario podrá crear, consultar, actualizar o borrar grupos de la base de datos.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario entra en la ventana de gestión de grupos a través del botón "Grupos".
	2	El SGBD devuelve los grupos existentes y se muestran por pantalla.
	3	Si el usuario crea o modifica un grupo,
	3.1	Debe llevar a cabo el caso de uso 2.2 Buscar Usuarios
4	Si usuario confirma los cambios,	
4.1	El SGBD actualiza/inserta/borra los datos dados.	
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.	

Tabla 4.8: CU-2.1 CRUD de grupos

<b>CU-2.2</b>	<b>Buscar Usuarios</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	- 2. Gestión de Grupos - 2.1 CRUD de Grupos	
<b>Precondición</b>	El usuario ha iniciado la aplicación. Se deben tener almacenados, usuarios para la formación de grupos.	
<b>Descripción</b>	Se realiza una búsqueda de usuario por nombre/apellidos	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario, en la ventana de formación de grupos, introduce por teclado el nombre o apellidos de un usuario.
	2	Si el sistema devuelve algún usuario,
	2.1	El usuario podrá seleccionar los usuarios deseados y asignarles un horario.
	3	En caso de que se guardan los cambios,
3.1	El SGBD actualiza/inserta/borra los datos dados.	
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.	

Tabla 4.9: CU-2.2 Buscar Usuarios



### 4.2.3 Casos de uso para la gestión de sesiones

Posteriormente en la figura 4.4 se muestran los casos de uso específicos para la gestión de sesiones.

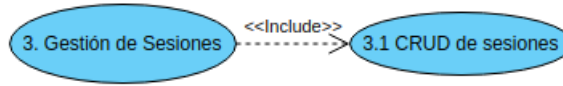


Figura 4.4: Casos de uso específicos para la gestión de sesiones.

<b>CU-3.1</b>	<b>CRUD de sesiones</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	- 3. Gestión de sesiones	
<b>Precondición</b>	El usuario ha iniciado la aplicación. Se deben tener almacenados ejercicios para la creación de Sesiones.	
<b>Descripción</b>	Caso de uso utilizado para creación, consulta, borrado, actualización y borrado de sesiones de entrenamiento.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario entra en la ventana de gestión de sesiones a través del botón "Sesiones".
	2	El SGBD devuelve las sesiones disponibles en caso de que haya y se muestran por pantalla.
	3	El usuario crea, actualiza o elimina una sesión.
	4	Si el usuario confirma los datos cambiados, 4.1 El SGBD actualiza/inserta/borra los datos dados.
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.	

Tabla 4.10: CU-3.1 CRUD de sesiones

#### 4.2.4 Casos de uso para la gestión de ejercicios

Para la gestión de ejercicios disponemos de los casos de uso disponibles en la figura 4.5, y posteriormente con las tablas que explican su especificación.

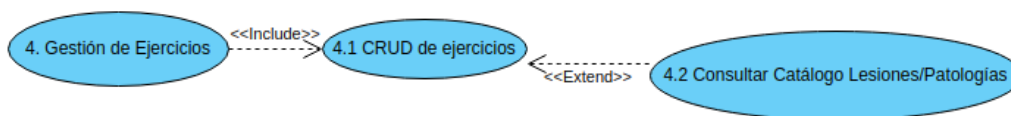


Figura 4.5: Casos de uso específicos para la gestión de ejercicios.

<b>CU-4.1</b>	<b>CRUD de ejercicios</b>		
<b>Versión</b>	1.0		
<b>Dependencias</b>	- 4. Gestión de ejercicios		
<b>Precondición</b>	El usuario ha iniciado la aplicación.		
<b>Descripción</b>	Caso de uso utilizado para creación, consulta, borrado, actualización y borrado de ejercicios.		
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>	
	1	El usuario entra en la ventana de gestión de sesiones a través del botón "Ejercicios".	
	2	El SGBD devuelve los ejercicios disponibles en caso de que haya y se muestran por pantalla.	
	3	El usuario crea, actualiza o elimina un ejercicio.	
	4	Si el usuario desea incluir lesiones/patologías por las cuales no se recomienda realizar el ejercicio,	
		4.1	Se realiza el caso de uso 4.2 Consultar Catálogo de lesiones/patologías
5	Si el usuario confirma los datos cambiados,		
	5.1	El SGBD actualiza/inserta/borra los datos dados.	
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.		

Tabla 4.11: CU-4.1 CRUD de ejercicios

<b>CU-4.2</b>	<b>Consultar catálogo de lesiones/patologías</b>		
<b>Versión</b>	1.0		
<b>Dependencias</b>	- 4. Gestión de ejercicios -4.1 CRUD de ejercicios		
<b>Precondición</b>	El usuario ha iniciado la aplicación.		
<b>Descripción</b>	Caso de uso utilizado para asociar lesiones o patologías a un ejercicio en concreto.		
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>	
	1	El usuario se encuentra en el formulario de creación de un ejercicio y consulta el catálogo de lesiones/patologías presionando el botón de consulta.	
	2	El sistema muestra lesiones/patologías disponibles, seleccionables por el usuario.	
	3	Si el usuario desea introducir algún/a que no se encuentre en el catálogo,	
		3.1	La introduce en el campo correspondiente y el sistema lo almacena en la BD.
	4	Si el usuario confirma los datos cambiados,	
4.1		El SGBD actualiza/inserta/borra los datos dados.	
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.		

Tabla 4.12: CU-4.2 Consultar catálogo de lesiones/patologías.

### 4.2.5 Casos de uso para la gestión de programación de sesiones

Por último se muestran los casos de uso específicos relacionados con la función principal que tiene la aplicación, gestionar la planificación de sesiones en grupo de entrenamiento, indicados en la figura 4.6.

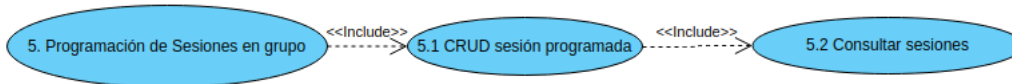


Figura 4.6: Casos de uso específicos para la gestión de programación de sesiones.

<b>CU-5.1</b>	<b>CRUD sesión programada</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	- 5. Programación de Sesiones en grupo.	
<b>Precondición</b>	El usuario ha iniciado la aplicación. Tener grupos de entrenamiento creados. Tener datos de alta sesiones.	
<b>Descripción</b>	Este caso de uso permite crear sesiones programadas para grupos de entrenamiento.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario entra en la ventana de gestión de la programación presionando el botón "Programación".
	2	El sistema muestra sesiones programadas guardadas en la BD.
	3	El usuario entra en el formulario de programación de una sesión y selecciona un grupo (y usuario individual si lo desea) y fecha del desplegable.
	4	Para seleccionar una sesión, se realiza el caso de uso 5.2 consultar sesiones, que interacciona directamente con la BD.
	5	Si el usuario confirma los datos cambiados,
	5.1	El SGBD actualiza/inserta/borra los datos dados.
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
	3	En caso de que se asigne un horario para la sesión que no sea el que le corresponde al grupo/usuario;
	E.1	El sistema manda una advertencia
	5	En caso de que algún usuario disponga de una lesión/patología no compatible,
E.1	El sistema hace una advertencia al usuario	

Tabla 4.13: CU-5.1 CRUD sesión programada.

<b>CU-5.2</b>	<b>Consultar sesiones</b>	
<b>Versión</b>	1.0	
<b>Dependencias</b>	- 5. Programación de Sesiones en grupo. -5.1 CRUD sesión programada.	
<b>Precondición</b>	El usuario ha iniciado la aplicación. Tener datos de alta sesiones.	
<b>Descripción</b>	Este caso de uso permite buscar sesiones guardadas en la BD en base a unos filtros.	
<b>Secuencia Normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El usuario utiliza como filtros la carga de trabajo y material para encontrar sesiones que encajen con lo deseado.
	2	El sistema realiza la búsqueda en base a los filtros introducidos en la BD y devuelve una lista de sesiones, con datos de objetivos, materiales o ejercicios incluidos.
	3	Si el usuario confirma los datos cambiados,
	3.1	El SGBD actualiza/inserta/borra los datos dados.
<b>Postcondición</b>	La base de datos queda actualizada en base a los datos introducidos.	

Tabla 4.14: CU-5.2 Consultar sesiones.



## 5.1 Wireframes

Una vez están terminados los casos de uso, es bueno reflejar estos en una representación visual la cual va a permitir exponer de una forma directa la idea principal que se tiene en mente. Esto se va a llevar a cabo por medio de wireframes. En esta sección se tratará de exponer todas las pantallas que estarán disponibles en la aplicación, agrupando el contenido por las diferentes secciones, tal y como se hizo en el apartado de casos de uso. Se ha utilizado la herramienta Balsamiq, mencionada en el apartado 3.1.

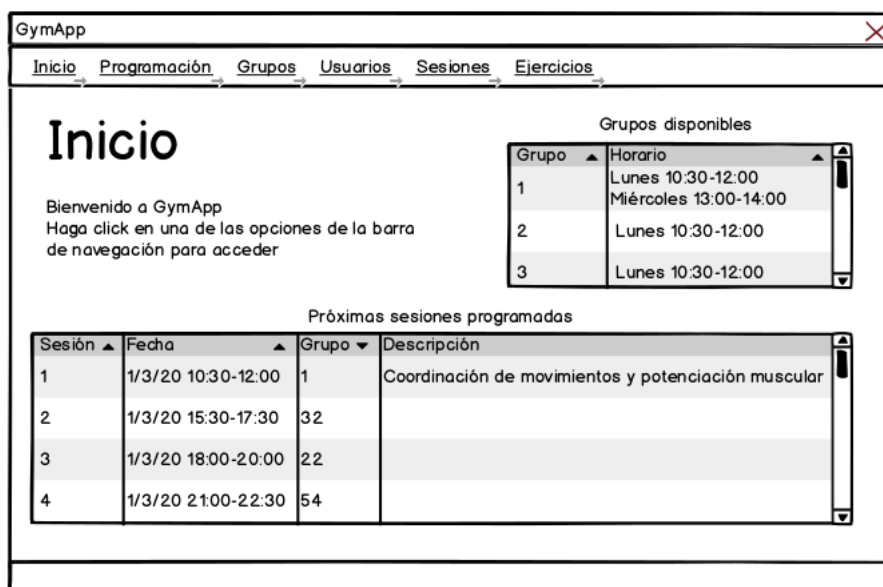


Figura 5.1: Ventana de inicio de la aplicación.

Para comenzar, mostramos una pantalla principal de inicio con la información más relevante, como carta de presentación de la aplicación. En esta pantalla (figura 5.1), en la parte

superior, se observan una serie de enlaces a las distintas secciones de la aplicación. Estos enlaces se van a mantener en el resto de secciones en el mismo sitio para ofrecer rapidez a la hora de desplazarse por el resto de secciones. Se observa además que se muestran dos tablas que son las que más se van a utilizar en el uso real de la aplicación ya que muestran los grupos con sus horarios disponibles y las próximas sesiones programadas, de esta forma, obtendremos la información más útil con solo iniciar la aplicación, sin necesidad de desplazarnos a dichas secciones.

### 5.1.1 Wireframes para la gestión de usuarios

En la figura 5.2, se muestra la pantalla principal de la sección de usuarios. Se trata de un tabla de datos la cual nos muestra los usuarios almacenados en la aplicación, permitiendo filtrar los resultados introduciendo nombre o correo electrónico. Además, vamos a disponer de una serie de botones que nos permitirán crear nuevos usuarios, ver o modificar los datos de usuarios existentes, acceder a su historial deportivo, eliminarlo y desactivarlo de planes de entrenamiento.

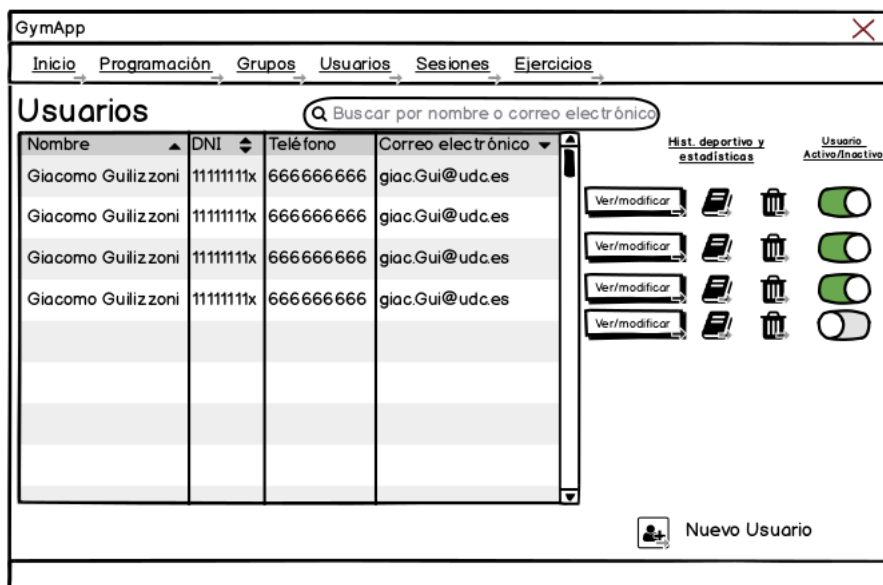


Figura 5.2: Ventana de gestión de usuarios.

A lo largo de todas las secciones se va a disponer de una opción de borrado tanto de usuarios como de sesiones, ejercicios, grupos etc. Debido a esto, cada borrado va a disponer de una ventana de confirmación para evitar que uno de estos elementos se borre sin querer. La ventana sería de la forma que muestra la figura 5.3.

Para crear o modificar un usuario, la aplicación mostrará un formulario el cual nos permitirá rellenar todos los datos relativos al usuario (figura 5.4). La aplicación controlará que esta



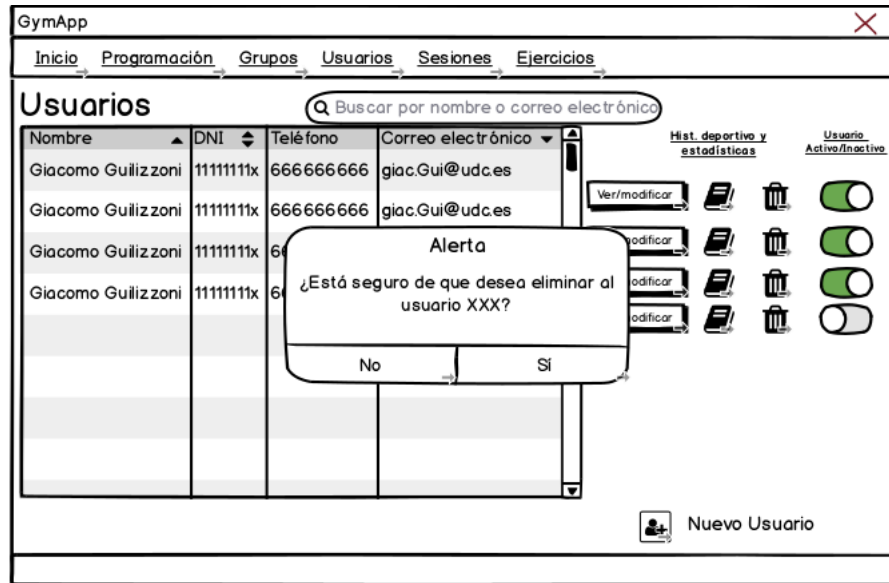


Figura 5.3: Pop-up de confirmación de borrado de usuario.

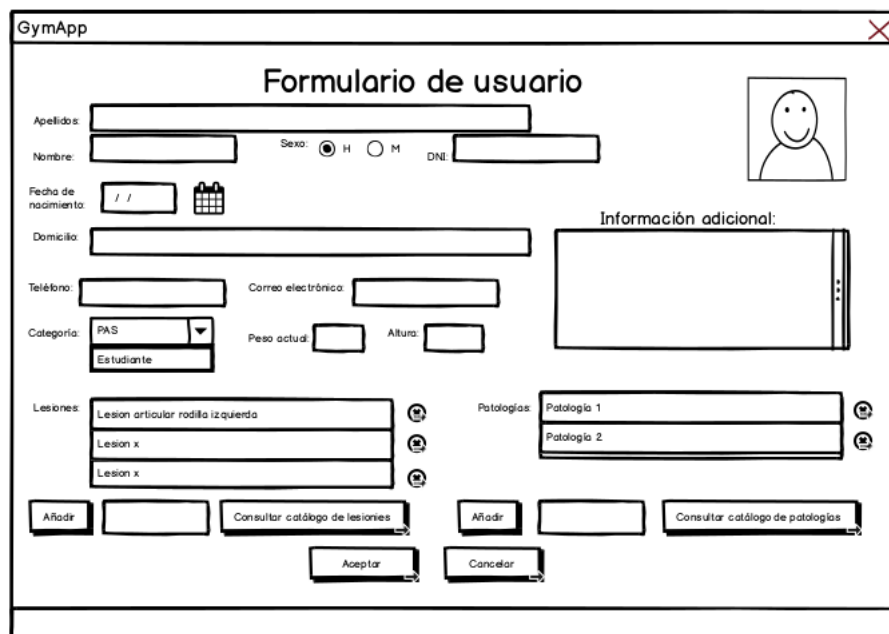


Figura 5.4: Formulario de usuario.

información es coherente controlando que por ejemplo, el formato de la fecha es correcto o que ningún campo se encuentre vacío, como muestra la figura 5.5. En esta misma ventana, se permite asociar lesiones o patologías al usuario. Esto se puede introducir por teclado, o consultando sus respectivos catálogos para así poder seleccionar una de ellas. Se podrán seleccionar varias en la misma ventanas, utilizando botones de tipo checkbox, como muestra la

The wireframe shows a window titled "GymApp" with a sub-header "Formulario de usuario". The form contains the following fields and elements:

- Apellidos: Colazo Collarte
- Nombre: Pablo
- Sexo: H (selected), M
- DNI: 11111111a
- Fecha de nacimiento: 30/08/ (with a calendar icon)
- Domicilio: Calle 113U 2ª C Vigo Pontevedra
- Teléfono: 666666666
- Categoría: PAS (dropdown menu)
- Peso actual: Estudiante (dropdown menu)
- Lesiones: Lesion articular rodilla izquierda, Lesion x, Lesion x (with plus icons)
- Patologías: Patología 1, Patología 2 (with plus icons)
- Buttons: Añadir, Consultar catálogo de lesiones, Añadir, Consultar catálogo de patologías, Aceptar, Cancelar

An "Alerta" dialog box is displayed in the center, containing the text "Formato introducido incorrecto" and an "Aceptar" button.

Figura 5.5: Ejemplo de formato incorrecto.

figura 5.6.

The wireframe shows a window titled "GymApp" with a sub-dialog titled "Catálogo de Lesiones". The dialog contains a list of lesions and a table of existing entries:

- Lesiones list:
  - Lesion en el biceps femoral  Añadir
  - Rotura de fibras en cuádriceps pierna izquierda  Añadir
  - Operado tendón rotuliano rodilla derecha  Añadir
- Table of existing entries:
 

Lesiones	Patologías
Lesion ar	
Lesion x	
Lesion x	
- Buttons: Añadir, Aceptar, Cancelar

Figura 5.6: Catálogo de lesiones.

Como explicaban los casos de uso de gestión de usuarios, el sistema detectará si una de las lesiones/patologías que se intenta introducir se encuentra en el catálogo, así se evita introducir lesiones o patologías que puedan repetirse en el catálogo (figura 5.7), y una vez advertido al

usuario se le mostrará una ventana (similar a la que muestra el catálogo), con las lesiones que coinciden con la introducida. El usuario podrá crear una nueva lesión/patología si no encuentra una adecuada en los resultados mostrados.

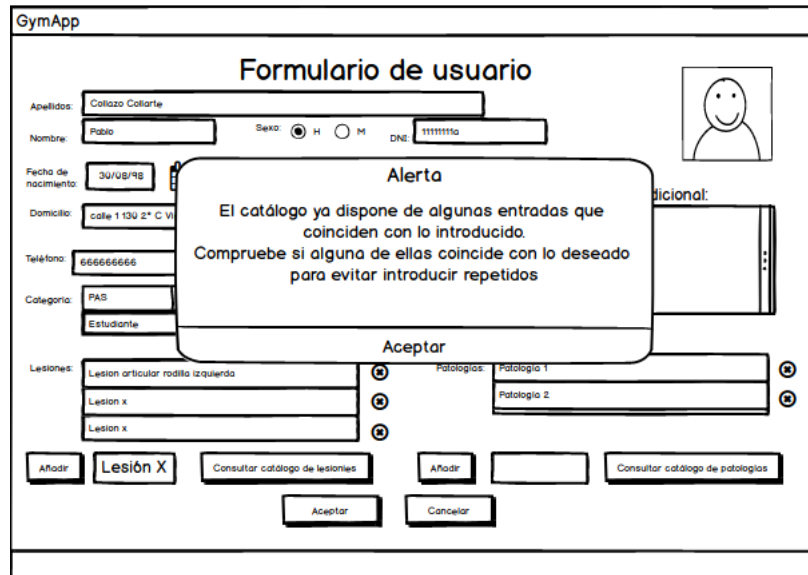


Figura 5.7: Advertencia de introducción de datos repetidos.

Otra opción disponible que vemos en la figura 5.2, es la de mostrar el historial deportivo y estadísticas de un usuario. Al acceder a esta opción se muestra la ventana de la figura 5.8. Esta ventana mostrará una gráfica con la evolución del peso del usuario además de un historial de las sesiones realizadas. Si accedemos a una sesión en concreto, observaremos una ventana emergente (figura 5.9) en la cual podemos introducir una valoración de la sesión personalizada para el usuario, introducir un nuevo peso tras la sesión y añadir lesiones que se hayan provocado en la ejecución de la sesión.

### 5.1.2 Wireframes para la gestión de grupos

Una vez gestionados los usuarios, podremos comenzar a agruparlos en distintos grupos con horarios personalizados. La ventana principal de grupos será de la forma similar en que se muestran los usuarios, una tabla a modo de resumen como en la figura 5.2.

En tanto al formulario de creación/actualización de grupos mostrado en la figura 5.10, se observa una entrada de texto en la cual podremos buscar usuarios por nombre y apellido e ir seleccionando y agrupándolos. Además, se le asocia un horario semanal que quedará almacenado para el grupo.

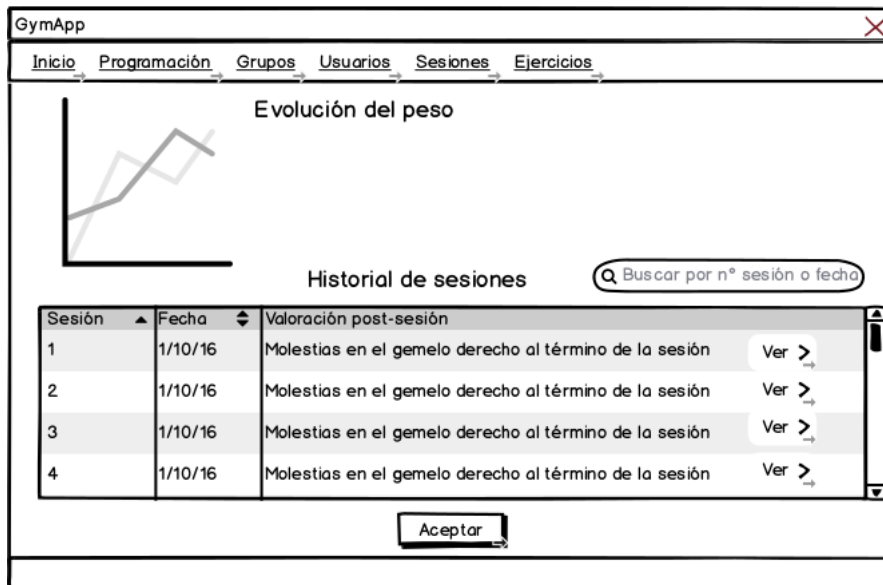


Figura 5.8: Historial deportivo y estadísticas.

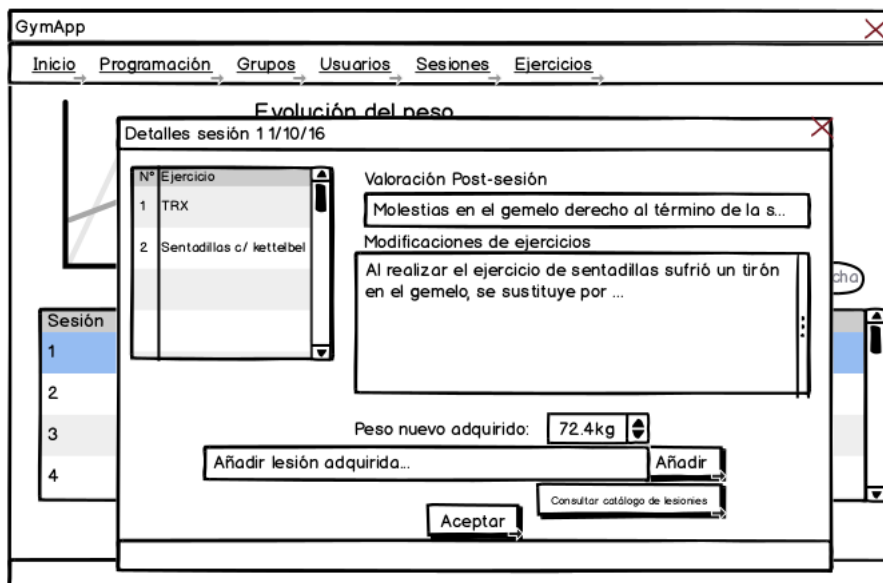


Figura 5.9: Detalles de sesión realizada.

### 5.1.3 Wireframes para la gestión de sesiones

En tanto a las sesiones, una vez más, disponemos de una tabla que muestra las sesiones similar a la de usuarios de la figura 5.2. Para su creación se dispondrá de un formulario como el de la figura 5.11. En esta pantalla, se podrá introducir una descripción de la sesión, un menú desplegable con los ejercicios a elegir y nos mostrará los materiales necesarios para su

The wireframe shows a window titled "GymApp" with a close button in the top right corner. The main title is "Formulario de grupos de entrenamiento".

Fields and controls include:

- Id:** A text input field containing "1234".
- Usuarios:** A search input field with a magnifying glass icon and the text "Nombre/Apellido", followed by a "Buscar" button.
- Usuarios seleccionados:** A list box containing "Jorge Pérez" and "Luis Pérez".
- Horario:** Two time selection fields labeled "Inicio" (9:15) and "Fin" (11:00).
- Días:** A list box containing "Lunes" and "Martes".
- Buttons:** "Añadir" and "Eliminar" buttons are placed between the user and day lists. "Aceptar" and "Cancelar" buttons are at the bottom.
- Table:** A table with two columns: "Día" and "Horario". It contains two rows: "Lunes" with "9:15-10:30" and "Miércoles" with "10:30-12:00".

Figura 5.10: Formulario de grupos.

realización. Por último también nos mostrará la distribución de las cargas de trabajo para la sesión e incompatibilidades relacionadas con las lesiones que no aconseja la realización de algunos de los ejercicios.

#### 5.1.4 Wireframes para la gestión de ejercicios

Para la gestión de ejercicios, además de una tabla similar a la de usuarios (figura 5.2), también se dispondrá de un formulario para éstos, como el de la figura 5.12. En ella se observan una serie de campos de diferentes tipos. Primeramente, se encuentran entradas de texto (nombre de ejercicio, descripción, materiales necesarios), también dispondremos de la opción de subir una foto, vídeo o enlace del ejercicio a modo explicativo, configuración de suma de cargas e introducción de lesiones/patologías no recomendadas para el ejercicio en concreto.

#### 5.1.5 Wireframes para la gestión de programación de sesiones

Por último se analizarán las pantallas relacionadas con la programación de sesiones para grupos de usuarios. En una primera pantalla (figura 5.13), se observan como elementos a destacar, el filtrado de sesiones mostradas, si fueron hechas o se harán próximamente. También disponemos de un enlace, en el caso de que fueran hechas, hacia una ventana emergente como la de la figura 5.14. En ella se muestran los usuarios que realizaron la sesión, los ejercicios realizados y la capacidad de introducir una valoración de la sesión. Además se incluye un enlace para poder acceder a la valoración personal de un usuario en concreto.

The wireframe shows a window titled "GymApp" with a close button in the top right corner. The main title is "Formulario de Sesión".

Fields and controls include:

- Id:** A text input field containing "1234".
- Objetivos:** A large text area containing "Potenciación muscular, coordinación de movimientos".
- Ejercicio:** A dropdown menu with "Elegir..." and a list showing "Bíceps" and "Triceps".
- Botones:** "Añadir" and "Eliminar" buttons.
- Materiales:** A list box containing "TRX", "Kettelbel", and "Escalera".
- Suma de cargas:** A pie chart showing a single slice.
- Incompatibilidades:** A text area containing "Lesión de hombro".
- Botones de acción:** "Aceptar" and "Cancelar" buttons at the bottom.

Figura 5.11: Formulario de sesiones.

En tanto al formulario de creación de una sesión programada dada en la figura 5.15, observamos una primera parte de introducción de grupos de usuarios (en donde también se pueden introducir usuarios pertenecientes a otros grupos) y una entrada para la fecha concreta de realización de esta sesión. Siguiendo con la pantalla, se propone una búsqueda de sesiones introduciendo como filtros la carga de trabajo o los materiales utilizados y esto provocará la aparición de sesiones relacionadas con estos datos. Esta ventana dispondrá de un par de controles útiles. El primero de ellos es la advertencia de asignación de un horario que no concuerda con el grupo o usuario introducido, como vemos en la figura 5.16, el segundo sería la advertencia de ejercicios incluidos en la sesión, contraindicados para un usuario determinado (figura 5.17).

**Formulario de Ejercicio**

Nombre:

Descripción:

Materiales:

Realización:

[http://www.youtube.com/watch\\_](http://www.youtube.com/watch_)  
<http://vimeo.com/>

Evitar lesiones:

Evitar patologías:

Carga: -aeróbico

-coordinación

-equilibrio

-fuerza

Figura 5.12: Formulario de ejercicios.

**Programación de Sesiones**

Mostrar todas  
 Mostrar hechas  
 Mostrar próximas

Id sesión:  Id Grupo:

Sesión	Fecha	Grupo	Descripción
1	1/4/20 10:30-12:00	1	Coordinación de movimientos y potenciación
2	1/3/20 15:30-17:30	32	Comentarios post-sesión >
3	1/3/20	22	
4	1/3/20	54	

Figura 5.13: Pantalla principal de programación de sesiones.

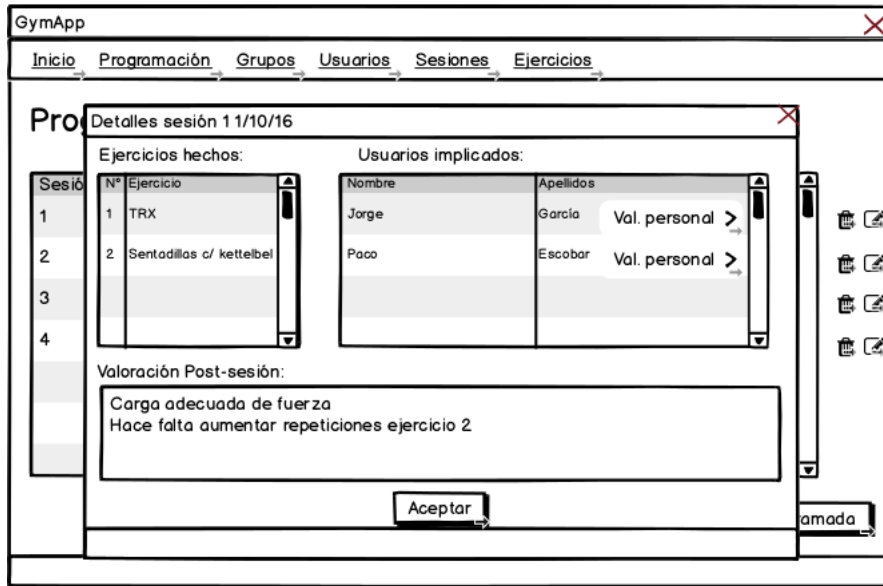


Figura 5.14: Valoraciones post-sesión.

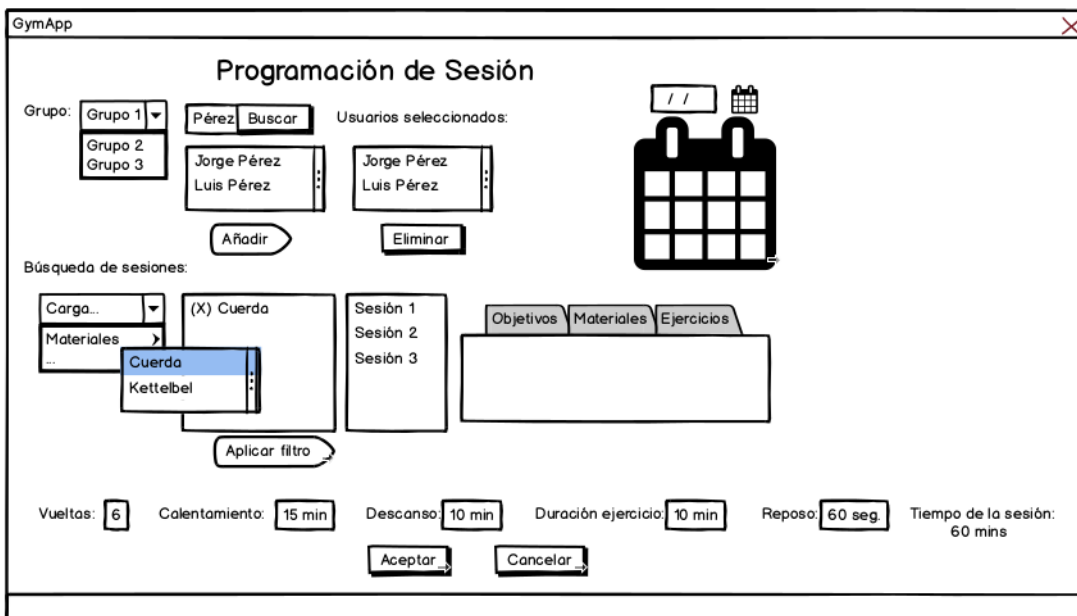


Figura 5.15: Formulario de creación de sesión programada.

## 5.2 Modelado de la base de datos

### 5.2.1 Diagrama Entidad-Relación



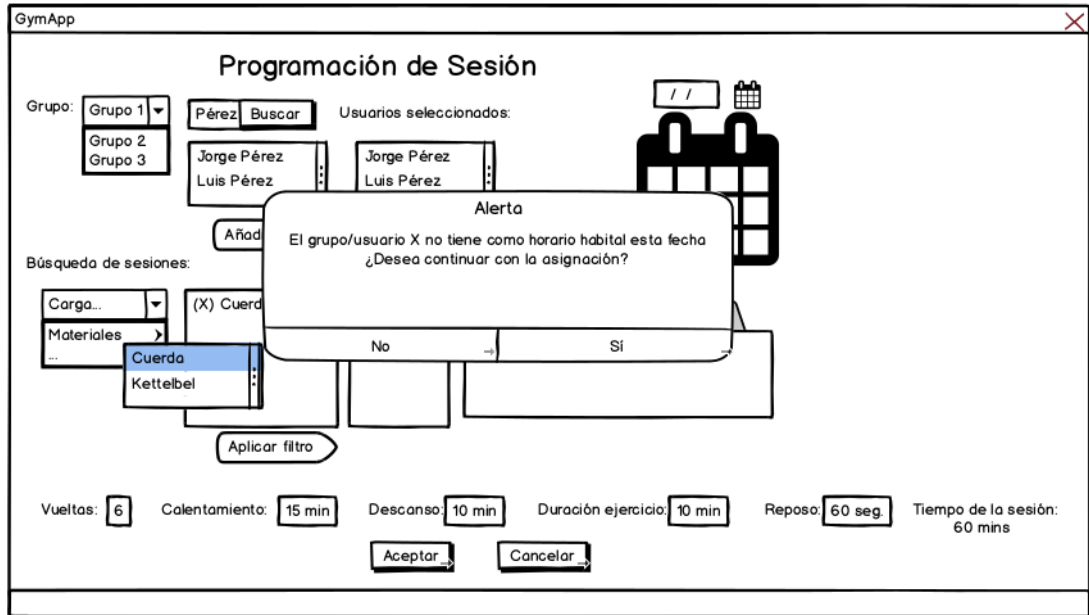


Figura 5.16: Ventana emergente de horario incompatible.

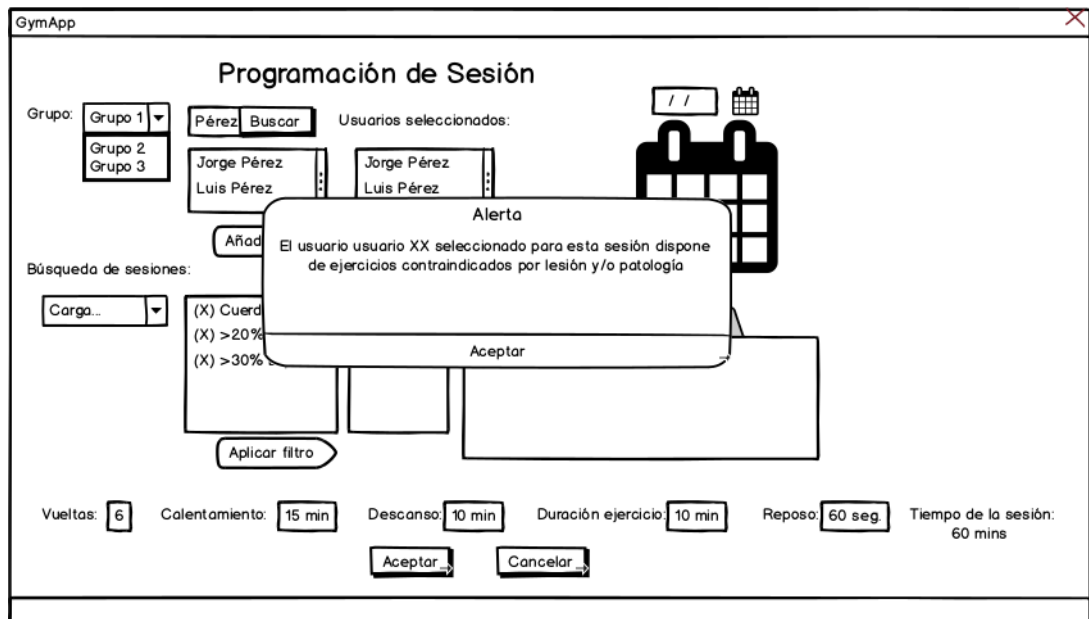


Figura 5.17: Ventana emergente lesiones contraindicadas.

Una vez realizado el diseño de la parte gráfica y expuestos los casos de uso que van a definir la aplicación, es necesario el diseño de una base de datos para almacenar todo tipo de datos que va a manejar el usuario al utilizar la aplicación. Se comienza con el diseño de un diagrama Entidad-Relación para reflejar la estructura del mundo real y posteriormente

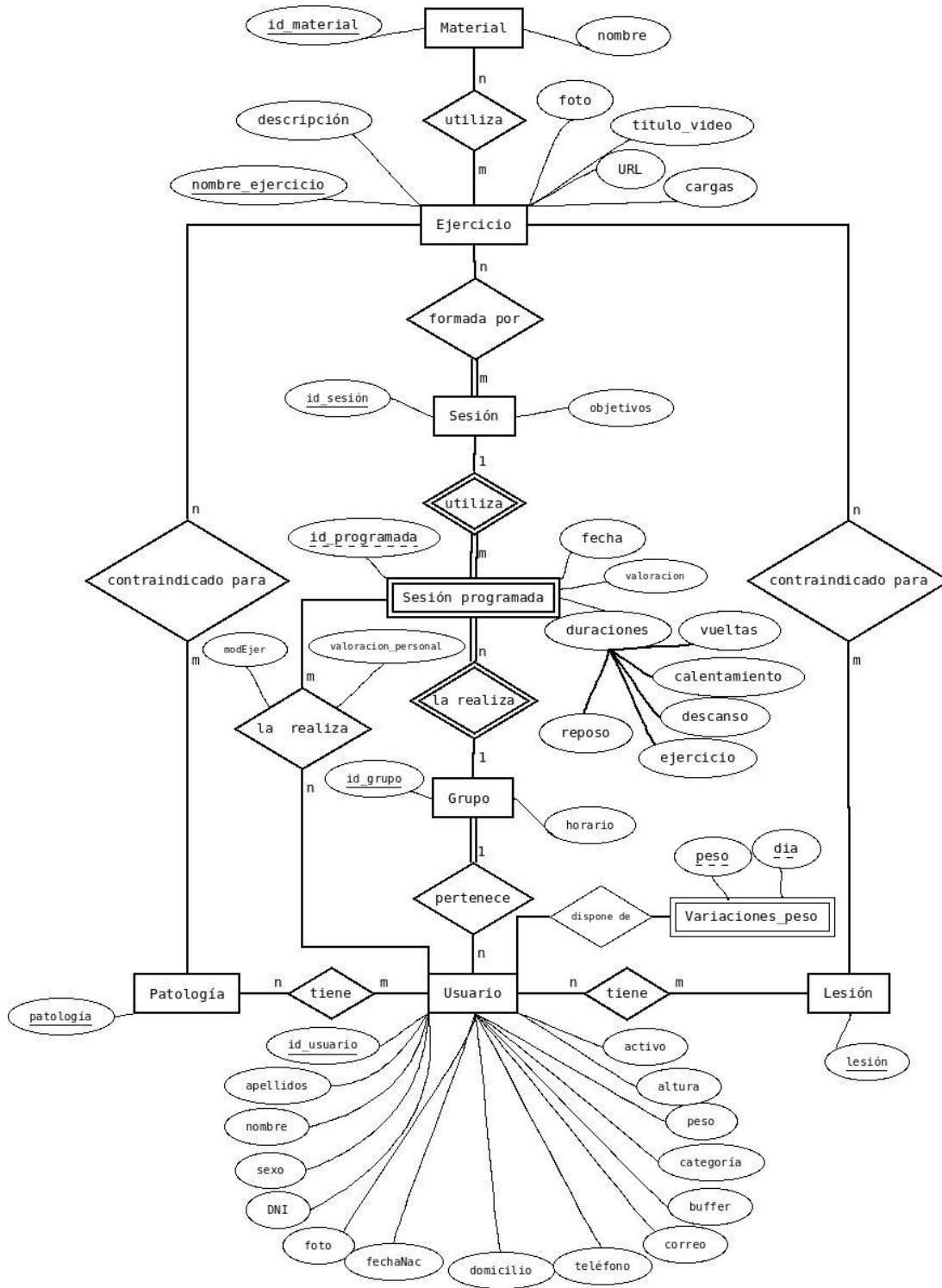


Figura 5.18: Diagrama ER.

aplicarla a las distintas relaciones utilizadas.

En el diagrama entidad-relación de la figura 5.18 podemos observar la relación entre la entidad Usuario y la entidad Grupo, con esta última con participación total, ya que todo grupo debe tener al menos un usuario. Esta entidad Grupo, mantiene una relación con la entidad débil Sesión Programada. Esta entidad es débil debido a que su existencia depende directamente de la entidad Sesión y del grupo al que se le haya asociado esa sesión. Si alguna de estas dos entidades no existiera, no tendría sentido su existencia. Además, está también relacionada con la entidad Usuario, debido a que podemos incluir usuarios individuales a sesiones que no pertenezcan al grupo establecido. La entidad Usuario mantiene una relación con la entidad débil Variaciones\_peso, ya que esta entidad representa los distintos pesos que va adquiriendo un usuario en las sesiones que realiza. De esta forma, podremos realizar la gráfica de variación de peso en función de su peso y el día en que se ha tomado esta medición. Se observa también la entidad Ejercicio relacionada con las entidades Lesión y Patología para que de esta forma, la aplicación advierta al usuario de ejercicios contraindicados. Es importante también la relación de Ejercicio con la entidad Sesión, con esta última con una participación total ya que no tiene sentido crear una sesión que no disponga de ejercicios para realizar.

### 5.2.2 Modelo relacional

Una vez realizado el diagrama entidad-relación, se hace la transformación a modelo relacional y obtenemos las relaciones que habría que realizar en la base de datos (figura 5.19). Cada relación n:m provoca la creación de una tabla que representa la relación entre dos entidades, como ocurre con las relaciones Usuario\_patología, Usuario\_lesión, Ejercicio\_Lesión, Ejercicio\_Patología, Sesión\_Ejercicio, Material\_Ejercicio y Sesión\_programada\_usuario. Estas relaciones y la utilización de la restricción de clave foránea, permiten mantener la integridad en los datos utilizados en la aplicación, además de que se van a emplear acciones referenciales como el borrado en cascada, para así por ejemplo en caso de que se borre un usuario, se borren las entradas de las lesiones y patologías de éste, en las tablas Usuario\_lesión y Usuario\_patología.

## 5.3 El patrón modelo-vista-controlador (MVC)

Dado que el proyecto resultante constará de una cantidad más o menos considerable de líneas de código, es importante seguir alguna metodología o patrón de arquitectura del software para así distribuir de una forma óptima las distintas partes de las que está compuesto el proyecto. Dado que es un proyecto en el que la mayor parte de trabajo se ve reflejado en la creación de la interfaz de usuario, el patrón modelo-vista-controlador es una buena opción para disponer de un proyecto ordenado, más legible y fácil de entender.

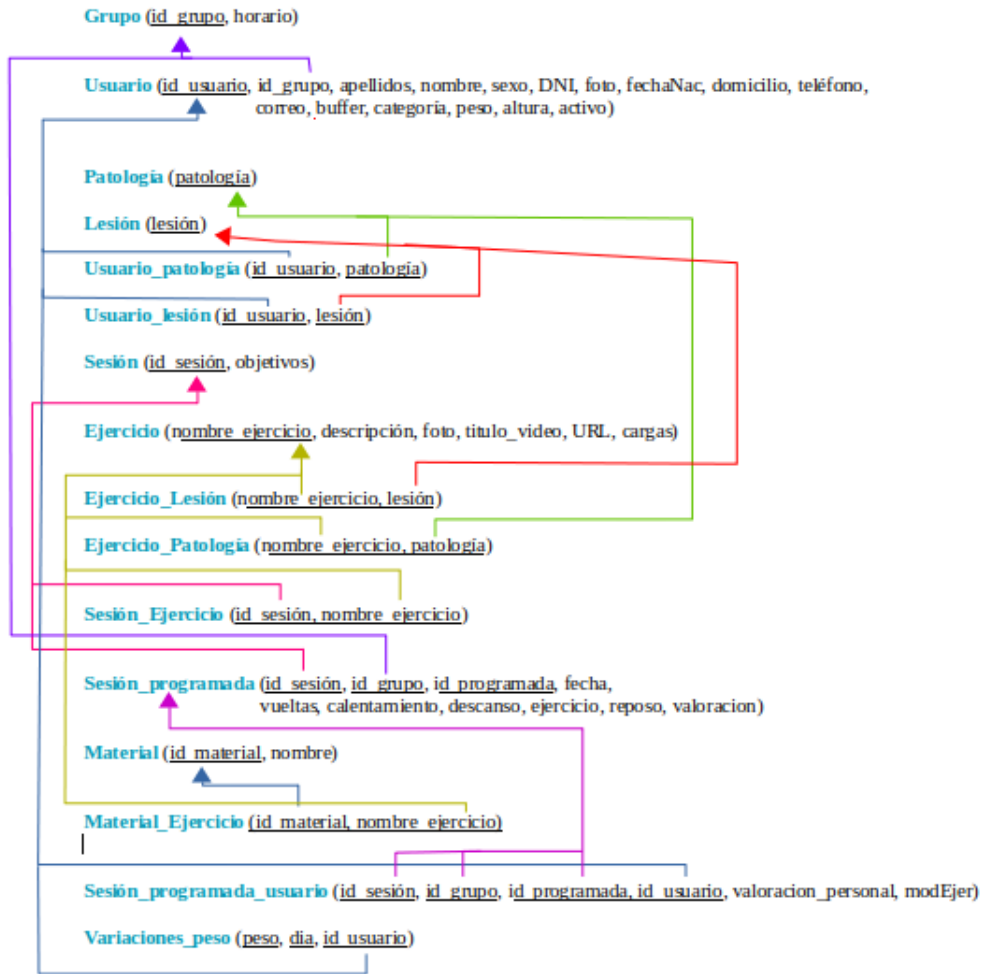


Figura 5.19: Modelo relacional.

El MVC 5.20 fue una de las primeras arquitecturas relativas a las interfaces gráficas de usuario. Fue creado por Trygve Reenskaug en 1979 [15]. Consiste en distribuir el código según tres partes, como su nombre indica independientes unas de otros, comunicándose solo entre ellas por medio de mensajes cuando es necesario. Dicho patrón se ha aplicado al proyecto siguiendo la distribución clásica:

- **El modelo:** Este está compuesto por la parte del programa en donde se almacenan los datos, donde está la lógica de negocio y representa el estado en el que se encuentra. En caso de que tenga que ser notificado un cambio en el modelo, se enviará un mensaje a modo de notificación al controlador, el cual puede considerar o no dar aviso al usuario a través de la vista, como vemos en la figura 5.20. Por lo tanto, aplicado al proyecto, contiene la lógica dedicada al insertado, actualización y eliminación de los datos utilizados en la aplicación.

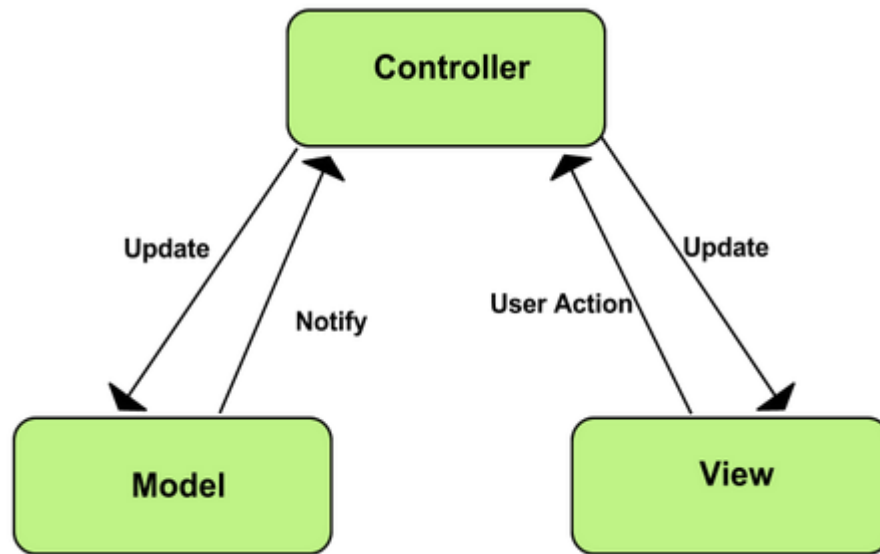


Figura 5.20: El modelo-vista-controlador [1]

- **El controlador:** Esta parte es donde se aloja la toma de decisiones y el enlace entre el usuario y el sistema. Básicamente se ocupa de dos tareas fundamentales en la aplicación: actualizar la vista cuando el modelo cambia, o actualizar el modelo cuando el usuario interactúa con la vista. También puede haber una interacción directa entre controlador y vista (excluyendo al modelo) en caso de manipular elementos que no afecten directamente a los datos o a la lógica de negocio, por ejemplo, al hacer scroll en una ventana o al marcar una opción de un checkbox (en el momento que se haga persistente el cambio, se avisaría al modelo). En estas situaciones la vista envía esta interacción al controlador, y el controlador al detectar estos cambios, actúa en consecuencia, actualizando elementos en la vista. La separación con la vista es importante, ya que la vista solo debe hacer lo que el controlador le indica y viceversa, el controlador no se debe ocupar de nada relacionado con lo que se muestra al usuario, de esta forma tendremos la información debidamente encapsulada, organizando cada función en su lugar correspondiente [16].
- **La vista:** Se compone de todo lo que se le muestra al usuario y por consiguiente, con todo con lo que interactúa. Aquí es donde se utilizaría todo tipo de frameworks, plantillas o bibliotecas de componentes gráficos que dan soporte a esta tarea, por lo tanto, es donde se establecen los widgets de GTK para la interfaz de la aplicación. Representa de una forma "user-friendly" los datos almacenados en el modelo y las acciones posibles para manipularlos (mandadas al controlador). Los mensajes utilizados para comunicar-

se con el controlador son en forma de interacciones de usuario (un click, arrastrar un objeto etc.) que se traducen en señales enviadas al controlador.

# Planificación

---

Para la realización de este trabajo, previamente se va a necesitar una planificación del trabajo y distribuirlo a lo largo de los meses que va a ocupar el TFG. Al ser un trabajo puramente de ingeniería, se dividirá básicamente en los distintos apartados de los que dispone la aplicación haciendo uso de una metodología de trabajo para una mejor organización y ejecución, la metodología Scrum. Estos apartados son, la pantalla principal (que dispondrá de información resumen), la gestión de usuario (almacenamiento y gestión de datos, pantallas, gestión de lesiones y patologías), la gestión de grupos de entrenamiento, la de ejercicios, de sesiones y por último la de programación de dichas sesiones.

## 6.1 La metodología SCRUM

A la hora de realizar un proyecto de desarrollo software es importante disponer de una organización adecuada con unos roles de trabajos bien definidos. Al tratarse de un proyecto individual, estos roles de desarrollo se condensan en tutor-alumno, por lo que se intentará seguir la esencia de la metodología, aunque no sea estrictamente.

El término SCRUM da nombre a una metodología acuñada por Ikujiro Nonaka y Hirotaka Takeuchi. Es un método creado en los años 80, resultado de analizar el trabajo de equipos de empresas tecnológicas, tales como Canon, Honda, Epson, Xerox..., las cuales adquirirían unos grandes valores de eficiencia y valor en sus productos. Scrum es un método, según la asesoría y empresa certificadora Scrum Manager, "llevado a cabo por equipos reducidos, multidisciplinares que trabajan con comunicación directa y empleando ingeniería concurrente" [17], sustituyendo a los clásicos ciclos o fases secuenciales de las metodologías en cascada. Es una metodología para mejora continua que adopta una estrategia de desarrollo incremental, solapando las distintas fases de un proyecto disponiendo de una serie de resultados preliminares que ayudan a determinar si el proyecto se está llevando a cabo adecuadamente. Iteración tras iteración se analiza lo conseguido y se verifica si es realmente lo esperado, de lo contrario se

corrigen las desviaciones. Esto conlleva correcciones tempranas que evitan que el proyecto arrastre errores, que ya podían haber sido corregidos en fases anteriores. Es importante marcarse unos plazos para distribuir el trabajo uniformemente y disponer de una estimación de la duración del proyecto total, aunque esta pueda no ser definitiva.

El proceso Scrum consiste en tres fases fundamentales: la planificación previa, el **Sprint** y el cierre del proyecto (figura 6.1).

La planificación es una fase muy importante del proceso, ya que es en donde se determinan las fechas de entrega, se distribuye el trabajo específico de los equipos formados y se conforma un diseño general en base a los requisitos dados por parte del cliente. Además es donde se estiman los costes y los riesgos que pueden generarse en todo el proceso de desarrollo.

Una vez hecha la planificación previa, se pasa directamente a realizar el Sprint. Este consiste en el trabajo de desarrollo que constituye el proyecto en sí. Como se ha mencionado anteriormente, es un proceso iterativo e incremental, en la que se mantienen reuniones diarias breves con el resto del equipo para situar en que punto se encuentran. Estas reuniones se llaman Daily Scrum: "El Daily Scrum es un evento de 15 minutos para que el Equipo de Desarrollo sincronice las actividades y cree un plan para las próximas 24 horas" [18]. El tiempo de duración de un Sprint viene determinado por la duración total del proyecto, aunque suele abarcar menos de un mes de duración. El resultado de este será una versión potencialmente usable, aunque evidentemente no completa. Tras la iteración, se procede a realizar una revisión del sprint en la cual se presenta el trabajo completado, inspeccionando el incremento realizado. Además, se realiza una retrospectiva del Sprint realizado, valorando posibles desviaciones e impresiones personales de los miembros del grupo que permiten exponer mejoras que pueden surgir, frente al producto realizado.

Tras iterar sobre una serie de Sprints dados, llega el punto del cierre del producto, el cual se determina cuando los requisitos y la calidad necesaria se consideran alcanzados y por lo tanto, se procede a cerrar una versión de la aplicación.

## 6.2 Planificación inicial

Para la planificación de este proyecto se va a hacer uso de la metodología descrita en el apartado 6.1 adaptada al proyecto. Los roles a desempeñar se resumen en el alumno como único grupo de trabajo, y director del trabajo, desempeñando el rol del cliente. Como alumno, para la planificación hago uso de sprints para definir metas intermedias e ir aumentando funcionalidades de la aplicación. Cuando uno de estos sprints se dé por terminado, se revisa el trabajo realizado y se somete a una serie de pruebas para comprobar que es satisfactorio, de lo contrario, se realizan las correcciones necesarias y no se continúa al siguiente sprint hasta que estos errores sean solucionados. De esta forma se evita arrastrar errores que pueden ser



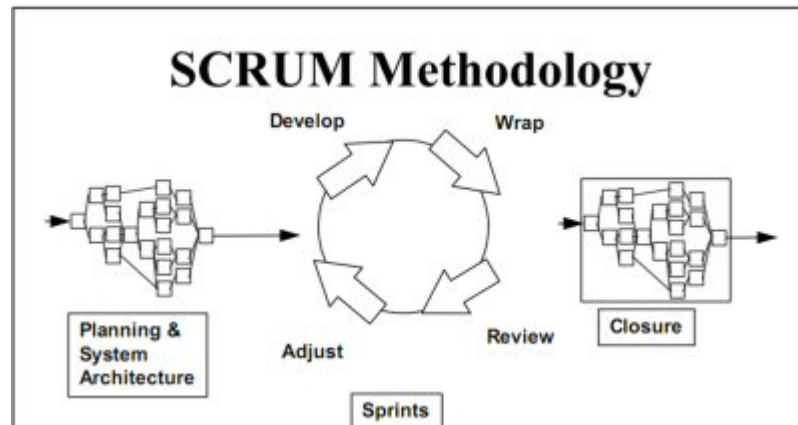


Figura 6.1: Metodología SCRUM.

corregidos al momento. De todas formas, es probable que surja la necesidad de cambiar elementos que se consideraban correctos, por el mero hecho de que al aumentar funcionalidades puede ser necesario adaptar el trabajo anterior al nuevo, de aquí surge la necesidad de emplear una metodología iterativa e incremental.

Por lo tanto, el trabajo se divide en distintos sprints que lo que van a hacer es aumentar funcionalidades y comprobar que éstas se adapten a las realizadas en sprints anteriores. Los sprints serían los siguientes.:

### Sprints:

- **Sprint 1) Familiarizarse con las herramientas:** Antes de empezar la programación de la aplicación es importante comenzar por conocer bien el toolkit GTK, la creación de bases de datos con SQLite y conocer como son de base estos elementos. De todas formas conforme se va programando el diseño, se irá investigando sobre como funcionan las componentes concretas necesarias ya que, al tratarse de algo muy visual, se conocerá con certeza que se necesita cuando lo vemos delante.
- **Sprint 2) Análisis de requisitos y diseño:** En este sprint se comenzará realizando el análisis de requisitos necesario y la creación de los distintos casos de uso para llevar a cabo la aplicación correctamente. Además se comenzarán a diseñar serie de wireframes para disponer de una guía visual de lo que queremos para hacer la programación más directa.
- **Sprint 3) Creación de plantilla utilizando el patrón MVC (Modelo-Vista-Controlador) y pantalla de inicio:** Tal y como se expuso en el apartado 5.3, se hará uso de este patrón. En esta primera fase de programación se comenzará a realizar la pantalla de inicio de la aplicación, tratando de ir explicando sus detalles en la redacción del TFG, a su vez.

- **Sprint 4) Creación de ventana de gestión de usuarios y formulario de creación/actualización:** En esta parte se comenzará con la programación de la ventana de gestión de usuarios, además de la creación de sus tablas en la base de datos correspondiente. Es una de las partes que llevará algo más de tiempo ya que los usuarios disponen de bastantes campos los cuales hay que comprobar que su formato es adecuado. Se llevará también a cabo la gestión de los aspectos relacionados con la base de datos (creación, modificaciones y consultas).
- **Sprint 5) Gestión de los catálogos de lesiones y patologías de los usuarios:** Se crearán bases de datos tanto para los catálogos generales como para cada usuario en concreto además de las distintas ventanas emergentes necesarias en la vista para poder gestionar estos.
- **Sprint 6) Creación de ventana de gestión de ejercicios y formulario de creación/actualización:** Además de la propia creación de la vista y el controlador de esta parte, también se crea una tabla en la base de datos para la manipulación de estos ejercicios.
- **Sprint 7): Creación y gestión de catálogos de materiales, lesiones y patologías de los ejercicios:** Los ejercicios también llevan asociados las lesiones y patologías con las cuales es contraproducente realizar un ejercicio concreto por lo tanto éstas, también tienen que estar asociadas a los ejercicios, para poder realizar esa interacción con los usuarios.
- **Sprint 8): Creación y gestión de grupos de entrenamiento:** Una vez más, en este apartado se programará tomando como guía el diseño previamente realizado. También se realizará sus tablas en la base de datos.
- **Sprint 9): Creación y gestión de sesiones de entrenamiento:** Similar a la tabla de usuarios y ejercicios, se hará una ventana principal con la que consultar sesiones guardadas, además de un formulario para su creación/actualización. Aparte de la tablas de datos propia de las sesiones, tenemos que asociar las tablas de los ejercicios para añadirlos a la sesión.
- **Sprint 10): Creación y gestión de la programación de sesiones de entrenamiento:** En este apartado se asociarán las sesiones anteriormente creadas a grupos de entrenamiento también previamente creados. Se comenzará realizando esta vista con grupos, añadiendo fechas de los entrenamientos y buscando una sesión adecuada al grupo utilizando distintos filtros (ya sea por material o carga de trabajo). Aquí entrará en juego tanto las contraindicaciones de los ejercicios por patología o lesión para los usuarios introducidos, como la advertencia de incompatibilidad horaria de los grupos de entrenamiento.

- **Sprint 11): Ventana de comentarios post-sesión en el apartado de la planificación:** Una vez finalizada la sesión, en la lista que muestra la planificación de las sesiones, si la sesión ya ha sido realizada, nos permitirá anotar una serie de valoraciones post-sesión tanto para la sesión en general como para usuarios concretos, a través de un enlace que se realizará en el siguiente punto.
- **Sprint 12): Historial deportivo individual y estadísticas del usuario:** En el apartado de usuario se añadirá un botón que nos permitirá observar estos datos. Si se realiza al final es porque se supone que todo lo anterior ya ha sido creado y por lo tanto ya se pueden realizar sesiones, ejercicios y planificarlos, por lo tanto disponemos de sesiones a aplicar en el historial de un usuario. Esta ventana dispondrá de un gráfico con la evolución del peso del usuario y un historial de las sesiones realizadas. En este historial se podrá realizar comentarios post-sesión para un usuario en concreto, además del nuevo peso del que dispone al realizar esa sesión, de esta forma se va generando una gráfica de evolución del peso.

En la figura 6.2 se muestra un **diagrama de Gantt** con una planificación estimada de las tareas anteriormente mencionadas, con sus dependencias directas y con una duración aproximada en días naturales, manteniendo una regularidad en horas diarias dedicadas al proyecto. Cabe destacar que no se espera que sean días de trabajo continuo reales sino que es una manera de contar los días pasados. Básicamente es una manera de establecer hitos para tener una referencia de como está siendo el proceso y poder conocer si hay algún retraso notable. Paralelamente, se fue realizando la memoria.

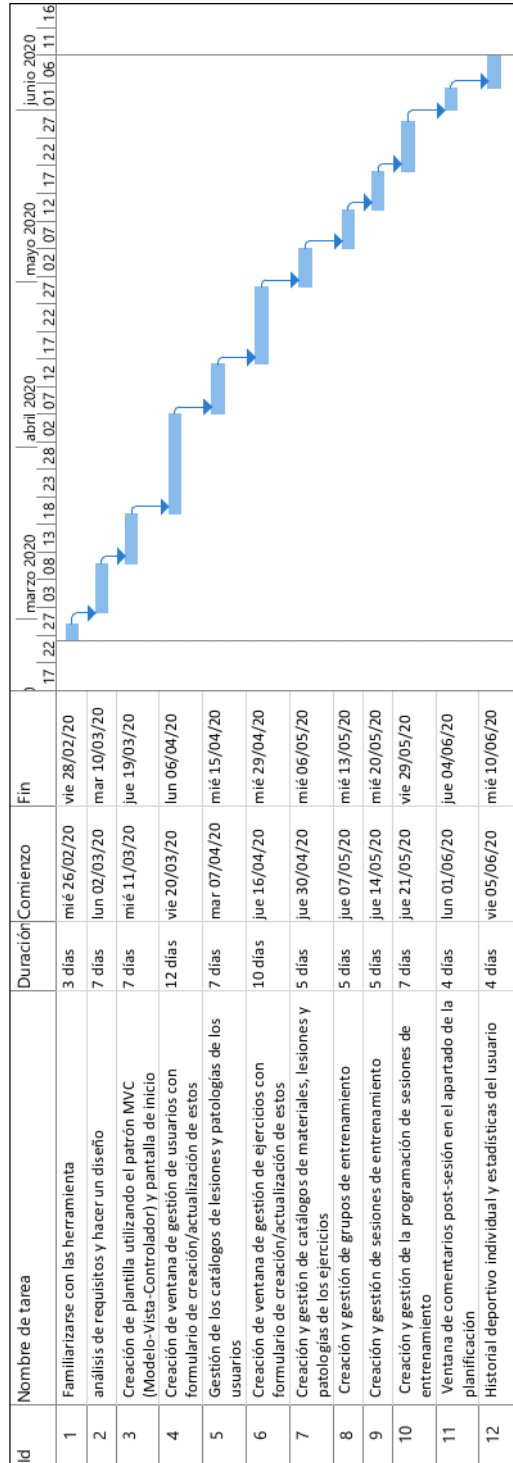


Figura 6.2: Diagrama de Gantt con las tareas distribuidas.

# Implementación

---

Una vez realizado el diseño y haber considerado unos objetivos bien definidos del trabajo a realizar, se pasa directamente a la parte de implementación. El proyecto se divide fundamentalmente en tres archivos que tienen como motivo la utilización del patrón explicado en el apartado 5.3, el modelo-vista-controlador. Estos archivos por lo tanto son el *model.py*, *view.py* y *controller.py*.

Además, se ha incluido un archivo *main.py*, utilizado para inicializar el controlador, el modelo y la vista, y para mantener el bucle de Gtk (con **Gtk.main()**) el cual permite mantener al sistema "escuchando" las señales que recibe del usuario, como se ha mencionado en anteriores apartados. Por lo tanto es el controlador el que comienza la ejecución del sistema:

```
1 BASEDEDATOS = 'database.db'
2
3 if __name__ == "__main__":
4
5     model = Model(BASEDEDATOS)
6     Controller(View(), model)
7
8     Gtk.main()
```

Como muestra el código anterior, se crea una variable "BASEDEDATOS" que hace referencia a un archivo en el directorio de la aplicación llamado "database.db". Este archivo en un inicio se encuentra vacío, y es el que utilizará el paquete `sqlite3` de python como soporte para almacenar la estructura de la base de datos. Esto se observará con detalle en el apartado referido a la capa modelo.

## 7.1 Base de datos

La capa modelo de la aplicación es la que va a alojar las operaciones que hacen referencia a la manipulación de los datos almacenados en el archivo anteriormente mencionado, data-

base.db. El modelo constituye una clase cuya inicialización se realiza por medio del método reservado `__init__()`, el cual ejerce de constructor de la clase modelo, que incluye la creación de las tablas necesarias en la base de datos.

Previamente a cualquier consulta SQL, se crea una conexión con el archivo que ejerce de base de datos, empleando la función `connect` disponible en el módulo `sqlite3` de python.

```

1 class Model:
2     def __init__(self, database):
3         self.conn = self.create_connection(database)
4         self.create_db_users()
5         self.create_db_lesiones()
6         self.create_db_patologias()
7         [...]
8
9     def create_connection(self, database):
10        conn = None
11        try:
12            conn = sqlite3.connect(database)
13            query = 'PRAGMA foreign_keys = ON' #para activar el uso de
14            claves foraneas
15
16            if conn is not None:
17                c = conn.cursor()
18                c.execute(query)
19                conn.commit()
20        except Exception as e:
21            print(e)
22        return conn

```

Como se observa en el fragmento de código anterior, se observa que la función `create_connection` tiene como parámetro de entrada el archivo `.db` (`database`) y que por medio de la función `connect` de `sqlite3`, se obtiene la conexión. Además `sqlite3`, por defecto, no tiene activadas las claves foráneas. Ejecutando la sentencia sql de la línea 13, activamos esta función para mantener la integridad en la base de datos.

```

1 def create_db_users(self):
2     query = 'CREATE TABLE IF NOT EXISTS user' + \
3           '( id INTEGER PRIMARY KEY AUTOINCREMENT, '+' \
4           ' id_grupo INTEGER, ' + \
5           ' Apellidos TEXT NOT NULL, ' + \
6           ' Nombre TEXT NOT NULL, ' + \
7           ' Sexo TEXT NOT NULL, ' + \

```

```

8         ' DNI TEXT NOT NULL, ' + \
9         ' Foto BLOB, ' + \
10        ' FechaNac TIMESTAMP NOT NULL, ' + \
11        ' domicilio TEXT NOT NULL, ' + \
12        ' telefono TEXT NOT NULL, ' + \
13        ' correo TEXT NOT NULL, ' + \
14        ' buffer1 TEXT, ' + \
15        ' Categoría TEXT NOT NULL, ' + \
16        ' Peso TEXT NOT NULL, ' + \
17        ' altura TEXT NOT NULL, ' + \
18        ' activo TEXT NOT NULL, ' + \
19        ' FOREIGN KEY (id_grupo) REFERENCES grupos(id_grupo) ON
DELETE SET NULL)'
20    if self.conn is not None:
21        try:
22            c = self.conn.cursor()
23            c.execute(query)
24            self.conn.commit()
25        except Exception as e:
26            print(e)

```

Una vez se dispone de la conexión, en caso de que las tablas no fueran creadas anteriormente, se ejecuta la sentencia SQL correspondiente a su creación. Se muestra un ejemplo para la tabla de usuarios en el fragmento de código anterior. Se utiliza la conexión previamente creada, esta vez utilizando el objeto **cursor** para la ejecución de la sentencia con la función **execute()**, tal y como se ha expuesto en el apartado 3.4, relativo a la base de datos utilizada.

Las operaciones a realizar contra la base de datos (insertado, consulta, modificación o borrado) dependerán de funciones concretas que en todas las tablas mantendrán una estructura general. Se expondrán ejemplos de estas operaciones para entender esta estructura.

```

1 def get_usuarios(self):
2     query = 'SELECT id, Apellidos, Nombre, Sexo, DNI, Foto,
3           FechaNac, domicilio, telefono, correo, buffer1, Categoría, Peso,
4           altura, activo ' + \
5           'FROM user'
6     work_time = None
7     if self.conn is not None:
8         try:
9             c = self.conn.cursor()
10            c.execute(query)
11            rows = c.fetchall()
12            work_time = []
13            for row in rows:

```

```

12         work_time.append(dict(id=row[0], Apellidos=row[1],
Nombre=row[2], Sexo=row[3], DNI=row[4], Foto=row[5],
FechaNac=row[6], domicilio=row[7], telefono=row[8],
13             correo=row[9], buffer1=row[10],
Categoría=row[11], Peso=row[12], altura=row[13], activo=row[14]))
14     except Exception as e:
15         print(e)
16     return work_time

```

Comenzando con las consultas, las habrá con algún tipo de condición (cláusula where de lenguaje SQL) ya sea para mostrar múltiples filas (en el caso de filtrar sesiones por ejemplo por material) o para devolver un solo elemento, por ejemplo el caso de la búsqueda de un usuario por id. Como se puede observar, el fragmento de código anterior muestra la función *get\_usuarios()* la cual devuelve todos los usuarios disponibles en la base de datos. Esta función se llama cuando accedemos a la sección de la gestión de usuarios, por lo tanto necesitamos todas las filas disponibles. Para ello, se observa en la línea 9 que la función utilizada para devolver dicha petición es la función **fetchall()**. Dicha función nos devuelve todas las filas, cada fila siendo una lista de todos los atributos de cada usuario. Para mayor comodidad, este resultado, como muestra la línea 12, se almacena en una lista de diccionarios para acceder fácilmente a los campos de cada usuario y enviarlos a la vista a través del controlador.

En el caso de consultas por id, se dispone de un ejemplo también para los usuarios en el siguiente código:

```

1 def get_user(self, id):
2     query = 'SELECT id, Apellidos, Nombre, Sexo, DNI, Foto,
FechaNac, domicilio, telefono, correo, buffer1, Categoría, Peso,
altura, id_grupo ' + \
3         'FROM user ' + \
4         'WHERE id = ?'
5     values = (id,)
6     work_time = None
7     if self.conn is not None:
8         try:
9             c = self.conn.cursor()
10            c.execute(query, values)
11            row = c.fetchone()
12            if row is not None:
13                work_time = dict(id=row[0], Apellidos=row[1],
Nombre=row[2], Sexo=row[3], DNI=row[4], Foto=row[5],
FechaNac=row[6], domicilio=row[7], telefono=row[8],
14                correo=row[9], buffer1=row[10], Categoría=row[11],
Peso=row[12], altura=row[13], id_grupo=row[14])

```



```

15     except Exception as e:
16         print(e)
17     return work_time

```

A diferencia del anterior, esta vez se añade una cláusula `WHERE` que limitará el resultado a la fila la cual tenga como atributo `id` el valor introducido como parámetro de entrada. Además, la función `execute` esta vez tiene como parámetro tanto la query SQL como los valores con los que se sustituye el carácter `"?"`, notación propia del lenguaje SQL. También se emplea esta vez la función `fetchone()`, que devolverá la única fila que cumpla la condición de la cláusula `WHERE`.

Para el resto de operaciones (insertado, borrado o modificación) que no requieren devolver datos tal y como lo hacían las consultas, la estructura es de forma similar, utilizando la secuencia sql correcta para cada caso:

```

1 def delete_user(self, id): #extracto para el caso del borrado
2     query = 'DELETE FROM user WHERE id=?'
3     values = (id,)
4     ...
5
6 #####
7 def update_active(self, activo, id)      #para el caso de
8     query = 'UPDATE user ' + \         #actualización de estado
9         'SET activo = ? ' + \         #activado/desactivado
10        'WHERE id =?'
11     values = (activo, id)
12     ...
13
14 #####
15 def insert_lesion(self, lesion):
16     query = 'INSERT INTO lesiones' + \ #para el caso de
17         '(lesion)' + \                #insertado
18         ' VALUES(?)'
19     values = [lesion]

```

## 7.2 Interacción de la vista con el controlador

En este apartado se expondrán los aspectos más relevantes de las interacciones entre interfaz gráfica y controlador, ya que cada acción en la interfaz, se traduce en un resultado en el controlador. La vista es la que interacciona directamente con funciones de la librería GTK.

En el anexo A se incluyen capturas de la aplicación completas, referenciándolas para poder entender de mejor forma las explicaciones. De todas formas se incluyen algunas destacables en la redacción de este apartado, para mayor comodidad.

Cada pantalla expuesta en la sección 5.1, va a disponer de al menos un bloque de código en la vista que estará formado por lo general por dos partes: la primera, la visualización en sí, lo que muestra el programa y con lo que puede interactuar el usuario; y la segunda, funciones que emiten señales y son enviadas al controlador, producto de acciones realizadas por el usuario en la propia pantalla. En tanto al controlador, se encargará de realizar las operaciones oportunas y decidirá que datos interactúan con la capa modelo, y que datos interactúan con la vista.

Para comenzar con la codificación, la forma de trabajo sería ir incluyendo los widgets de arriba-abajo para seguir un orden claro, ya que después se procederá a empaquetarlos en las estructuras mencionadas en la sección 3.3, en los Gtk.Box y en los Gtk.Grid.

Una vez iniciada la aplicación, se muestra la pantalla de inicio de la aplicación (figura 7.1) que como se ha expuesto en el apartado de diseño, constará de botones para acceder a las distintas secciones y dos tablas que proporcionarán información de próximas sesiones almacenadas en la aplicación y de grupos de usuario formados.

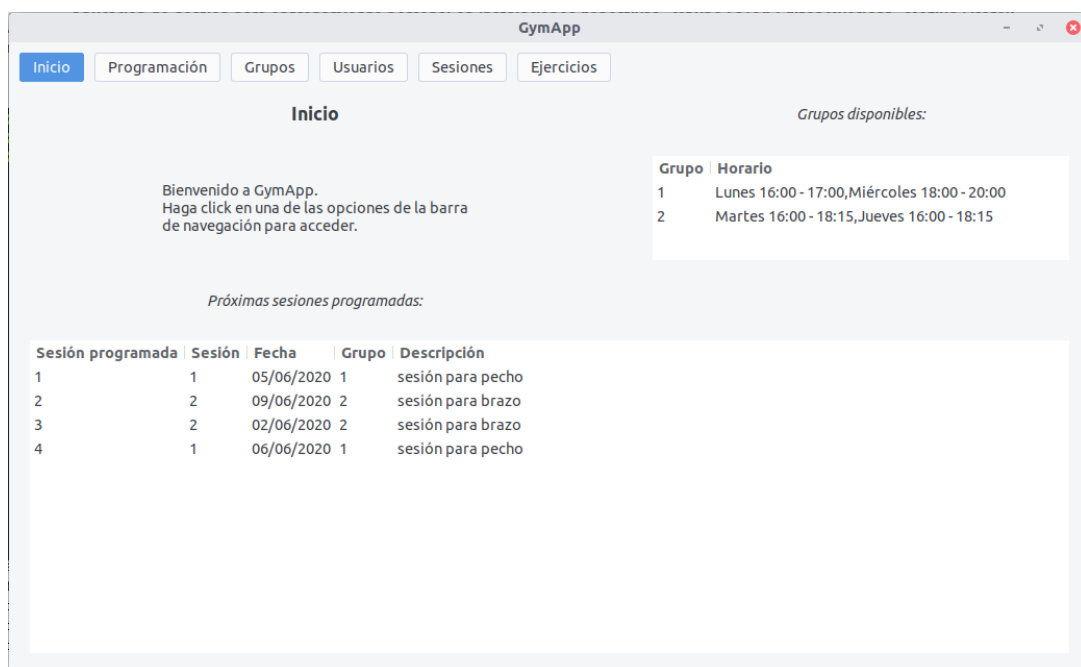


Figura 7.1: Ventana de inicio de la aplicación.

El método de trabajo para cada ventana consta de una serie de partes:

- 1) Declarar los widgets deseados con sus parámetros de entrada o funciones de configuración.
- 2) Empaquetarlos en un contenedor específico: En caso de que queramos más de un widget sobre esa fila/columna alineados entre sí, se sitúan en el mismo contenedor.
- 3) Empaquetar los contenedores específicos en el Gtk.Grid general que representa toda la ventana e introducirlo en la ventana
- 4) Introducir las funciones de respuesta a las interacciones con los widgets: Esta es la forma por la cual se interconecta la vista con el controlador.

La conexión de las acciones con el controlador para capturar las señales emitidas se hace por medio de la función *connect*, que se encuentra en prácticamente todos los widgets de Gtk:

```
1 def clickEnlaces(self, controller):
2     self.Ini.connect("toggled", controller.on_iniciar)
3     self.Programacion.connect("toggled", controller.on_programacion)
4     self.Grupos.connect("toggled", controller.on_grupos)
5     self.Usuarios.connect("toggled", controller.on_usuarios)
6     self.Sesiones.connect("toggled", controller.on_sesiones)
7     self.Ejercicios.connect("toggled", controller.on_ejercicios)
```

De esta forma si por ejemplo, se acciona el botón "Ejercicios", se ejecutará la función "on\_ejercicios" del controlador de la aplicación (línea 7 del código anterior).

### 7.2.1 Tablas resumen

Para la implementación de las tablas resumen de cada sección, se utilizan una serie de elementos de Gtk, tanto para mostrar la propia tabla, como para implementar un filtro (si procede) con el cual filtrar los datos que queremos mostrar de la tabla correspondiente:

```
1 liststoreUsuarios = Gtk.ListStore(str, str, str, str, str, str,
2     str) #modelo de la tabla a representar, cada str es una columna
3 self.liststoreUsuarios = liststoreUsuarios
4 filtroUsuarios = liststoreUsuarios.filter_new() #Hace un filtrado
5     de liststore. Crea otro liststore para aplicar el filtro
6 filtroUsuarios.set_visible_func(self.aplicar_filtro_usuarios)
7     #Filtra el liststore nuevo según lo que indique la función
8 self.filtroUsuarios = filtroUsuarios
9 self.filtroUsuarios_inicio = ""
```

```

8  busqueda = Gtk.SearchEntry(width_chars=8)
9  self.busqueda = busqueda
10 Etiqueta_busqueda = Gtk.Label(label = ("Filtrar por
    Apellidos/Nombre/Email:")) #entrada para filtrar
11 users = Gtk.TreeView(filtroUsuarios, headers_visible=True) #Lista
    grupos

```

El código anterior es un ejemplo de declaración de la tabla de usuarios. El **ListStore**, representaría el modelo de datos que se van a mostrar en la tabla (en este caso, 7 columnas de tipo string), la función *filter\_new()* representa la inicialización del filtro de la tabla y la función *set\_visible\_func()* sería la función que controlaría qué filas cumplen con la condición del filtro, el cual se encuentra en su función de entrada, *self.aplicar\_filtro\_usuarios*, en este caso. El elemento `Gtk.SearchEntry` sería la entrada de texto que corresponde a una barra de búsqueda (que emite una señal cuando ésta se cambie, al controlador) y el **Gtk.TreeView** de la última línea representa la visualización de la tabla en base a los datos mostrados por el filtro anterior. Cada tabla dispondrá de una serie de opciones para manipular las filas, tales como opción de eliminación, modificación, acceso a funcionalidades específicas (e.g.: historial deportivo de un usuario en concreto, activación de un usuario, etc).

### 7.2.2 Formularios

Para implementar los formularios de cada sección se hace uso de elementos de entrada de texto para los datos simples tales como descripciones o nombres, que en Gtk corresponden a los elementos **Gtk.Entry**. Para el caso de datos más complejos como son fotos, se hace uso de un elemento que interacciona con el equipo, para poder elegir dicha foto y almacenarla en la base de datos. Además, el programa controla su tamaño y lo ajustará a un tamaño estándar (introducido en el propio código), para así tener un tamaño regular en todas las fotos y evitar ocupar demasiada memoria en la base de datos.

```

1  def seleccionar_foto(self, widget):
2      Seleccionarfoto = Gtk.FileChooserDialog("Seleccione una foto",
3      self.win, Gtk.FileChooserAction.OPEN,
4      (Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL,
5      Gtk.STOCK_OPEN, Gtk.ResponseType.OK))
6
7      self.add_filters(Seleccionarfoto)
8      response = Seleccionarfoto.run()
9      if response == Gtk.ResponseType.OK:
10         FotoSeleccionada = Seleccionarfoto.get_filename()
11         self.FotoSeleccionada = FotoSeleccionada
12         fotoAnterior = self.CreacusuariosGrid.get_child_at(10,1)

```

```

12     pixbuf =
      GdkPixbuf.Pixbuf.new_from_file_at_scale(FotoSeleccionada, 150,
      175, True)
13
14     fotoAnterior.set_from_pixbuf(pixbuf)           #cambio la foto
      por la que se selecciona en archivos
15     Seleccionarfoto.destroy()

```

Como se observa en el código superior, en la línea 2, se hace uso del elemento **FileChooserDialog**, el cual corresponde a una ventana emergente de selección de archivo situado en el sistema operativo, que permite seleccionar una foto (si la extensión no es la de una foto, mostraría un error). En la línea 12 se observa la función *new\_from\_file\_at\_scale()*, la cual a partir de un buffer de píxeles (que representan la foto), ajusta dicha foto a una escala pasada por parámetro.

Para introducir lesiones y patologías para usuarios y ejercicios, se ha decidido implementarlo de forma que se actualice la base de datos al final de los cambios, así se evitan realizar demasiadas peticiones a la base de datos. La comprobación y consulta de las bases de datos de lesiones y patologías se realizan siempre en el controlador. La implementación gráfica de esto constituye una entrada de texto y también se podrán introducir lesiones seleccionándolas en el catálogo. Se irán añadiendo a una lista de elementos las lesiones/patologías que vayamos añadiendo, como vemos en la figura 7.2.

Figura 7.2: Entrada de texto para añadir y eliminar entradas de la lista de patologías de un usuario o ejercicio.

Además para este aspecto, se ha implementado el código en el controlador de forma que se evita que se introduzcan lesiones o patologías repetidas. En caso de que una de estas tenga

palabras similares a una lesión/patología ya disponible en la base de datos, el sistema recomendará lesiones ya disponibles que se pueden seleccionar, como comprobaremos en la sección de pruebas.

Continuando con elementos a destacar implementados en los formularios, en ejercicios se da la opción de insertar un archivo en formato vídeo y otro en formato imagen como explicación. Como se ha mencionado anteriormente, la imagen se ajusta a una escala dada y se almacena en la base de datos de **SQLite3**, con un formato tipo *BLOB*.

En tanto al vídeo, `sqlite3` no soporta este tipo de datos para ser almacenado, por lo que ha decidido guardar la **ruta del archivo de vídeo** como alternativa. Dicho vídeo, se ha podido incluir en la vista de la aplicación gracias a librería **Vlc**[19]. Dicha librería proporciona un framework multimedia que permite dichas capacidades a aplicaciones python integradas en este caso.

```

1 def configVideo(self, widget, video):
2     self.view.vlcInstance = vlc.Instance("--no-xlib")
3     self.view.player = self.view.vlcInstance.media_player_new()
4     win_id = widget.get_window().get_xid()
5     self.view.player.set_xwindow(win_id)
6     self.view.player.set_mrl(video)
7     self.view.video = video
8     self.view.player.play()
9     self.view.is_player_active = True
10    self.view.vlcInstance.release()

```

Como vemos en fragmento de código anterior, se crea una instancia de `Vlc` y se inicializa un reproductor donde el cual reproducir nuestro video a partir de una ruta introducida, por medio de la función `self.view.player.set_mrl(video)`.

Cabe destacar que para mantener la integridad de los datos, se ha decidido que no se pueda cambiar el nombre de un ejercicio guardado, para así evitar incongruencias por si este ejercicio ya se encuentra asociado a una sesión.

Todo elemento que represente una sucesión de líneas de texto, ya sea una lista o tabla que requiere más del espacio que permite la pantalla, se ha introducido dentro de un **Gtk.ScrolledWindow**, también llamado ventana deslizante, lo que nos permite deslizarnos vertical u horizontalmente a través de la pantalla. El elemento que requiera el deslizamiento, se introduce dentro de un `Gtk.ScrolledWindow`, como nos muestra el código siguiente:

```

1 ventana_scroll = Gtk.ScrolledWindow(expand=True, margin=20)
2     ventana_scroll.set_size_request(400, 100)
3     ventana_scroll.add(sesionesusers)

```

### 7.2.3 Ventanas de error o advertencia

Muchas de las acciones realizadas en cualquier sección deben pasar una comprobación para verificar que los datos introducidos tienen sentido o que se encuentran en un formato adecuado. De esta forma, evitaremos inconsistencias en la base de datos y mantendremos la aplicación en un estado estable. La aplicación tiene una función con una serie de mensajes de error a la cual se accede para mostrarlos por pantalla en caso de que se den las condiciones para que dichas excepciones se muestren:

```
1 def error(self, error):
2     if (error == 1):
3         Mensaje= Gtk.MessageDialog(self.win, 0,
4             Gtk.MessageType.WARNING,Gtk.ButtonsType.OK, ("\nFormato
5             introducido incorrecto"))
6     elif (error == 3):
7         Mensaje= Gtk.MessageDialog(self.win, 0,
8             Gtk.MessageType.WARNING,Gtk.ButtonsType.OK, ("\nEntrada no
9             encontrada"))
10    elif (error == 4):
11        Mensaje= Gtk.MessageDialog(self.win, 0,
12            Gtk.MessageType.WARNING,Gtk.ButtonsType.OK, ("\nImposible añadir
13            cambio, algún campo está vacío"))
14    elif (error == 5):
15        Mensaje= Gtk.MessageDialog(self.win, 0,
16            Gtk.MessageType.WARNING,Gtk.ButtonsType.OK, ("\nLa suma de las
17            cargas debe dar 100%"))
18    elif (error == 6):
19        Mensaje= Gtk.MessageDialog(self.win, 0,
20            Gtk.MessageType.WARNING,Gtk.ButtonsType.OK, ("\nEl máximo de
21            elementos multimedia a añadir son dos"))
22    elif (error == 7):
23        Mensaje= Gtk.MessageDialog(self.win, 0,
24            Gtk.MessageType.WARNING,Gtk.ButtonsType.OK, ("\nYa hay un
25            elemento de este tipo"))
26    elif (error == 8):
27        Mensaje = Gtk.MessageDialog(self.win, 0,
28            Gtk.MessageType.WARNING, Gtk.ButtonsType.OK,("\nModificación no
29            permitida; debe borrar y crear de nuevo"))
30    elif (error == 9):
31        Mensaje= Gtk.MessageDialog(self.win, 0,
32            Gtk.MessageType.WARNING,Gtk.ButtonsType.OK, ("\nSeleccione antes
33            una sesión"))
34    elif (error == 10):
35        Mensaje= Gtk.MessageDialog(self.win, 0,
36            Gtk.MessageType.WARNING,Gtk.ButtonsType.OK, ("\nEl vídeo a
```

```

mostrar se ha movido de directorio y no se encuentra en el path
almacenado. Introduzcalo de nuevo.))
20 else:
21     Mensaje= Gtk.MessageDialog(self.win, 0,
Gtk.MessageType.WARNING, Gtk.ButtonsType.OK, ("\nLesión ya
disponible en el usuario"))
22     Mensaje.run()
23     Mensaje.destroy()

```

En el fragmento de código anterior se muestran las excepciones que puede mostrar el sistema en caso de que se den dichas circunstancias. De esta forma, por ejemplo, en los casos en que demos de alta a un usuario con un DNI en un formato erróneo, falten campos importantes por rellenar o se intente incluir más apartados multimedia de los permitidos, el sistema mostrará dicha excepción por medio de una ventana emergente de error, informándonos de lo sucedido. Como ventanas a destacar, se han implementado advertencias que indican si algún usuario asociado a una sesión programada dispone de patologías o lesiones que le puedan impedir la realización de los ejercicios o también un aviso por el cual, se está programando una sesión en un día que no le corresponde al horario de grupo asignado. Todas ellas se capturan en el controlador y son enviadas a la vista para mostrar el mensaje correspondiente. Las situaciones más destacables, se expondrán en el capítulo 8 de pruebas de la aplicación.

#### 7.2.4 Ventanas emergentes

Además de las ventanas en sí que constituyen toda la aplicación, se han añadido ventanas de tipo emergente, que en Gtk son las tipo **Gtk.Dialog()**. Estas ventanas tienen la característica de que dependen directamente de una ventana principal y son ideales para rellenar campos o realizar acciones que dependan directamente de otra pantalla. De esta forma, cuando se termine de realizar la acción por la que se han creado, vuelve a mostrarse la pantalla de la cual heredan. Esto se da en el caso de incluir una valoración post-sesión a una sesión programada ya pasada:

```

1 def postsesionProg(self, sesionprog, sesion, ejerciciosnecesarios,
nombrenecesarios, apellidosnecesarios, controller, valoracion,
sesionid, usersid):
2     dialogo = Gtk.Dialog("Detalle sesión " + sesion, self.win,
3     Gtk.DialogFlags.MODAL|Gtk.DialogFlags.DESTROY_WITH_PARENT,
4     (Gtk.STOCK_OK, Gtk.ResponseType.OK))
5     dialogo.set_default_size(50,5)
6     contenido = dialogo.get_content_area()

```



Aquí se observa que la declaración de un `Gtk.Dialog` dispone de varios parámetros de entrada que representan datos como, la ventana de la que hereda, y el botón y respuesta que queremos que incluya el diálogo, en este caso, una respuesta que continúe la ejecución `ResponseType.OK`. Otras respuestas usadas podrían ser la de tipo `Gtk.ReponseType.CANCEL`, para cancelar los cambios realizados.

### 7.2.5 Gráficas

En la aplicación se disponen de dos gráficas para representar datos de una forma más intuitiva, como se ha mostrado en el capítulo 5. La primera gráfica representa la distribución de las cargas a la hora de diseñar una sesión de entrenamiento. Para ello se hace uso de la librería **matplotlib** utilizando su versión para python, `matplotlib.pyplot`. [9], además del backend `Gtk3Agg` el cual proporciona soporte a `Gtk` para la representación. El código para representar la gráfica sería el siguiente:

```
1 figureObject, axesObject = plotter.subplots()
2 axesObject.pie(porcentajes, labels=cargas,
3               autopct='%1.2f', startangle=90)
4 axesObject.axis('equal')
5 self.axesObject = axesObject
6 canvas = FigureCanvas(figureObject)
7 plotter.figure(figsize=[100, 100])
8 self.cargas = str(aero) + "% /" + str(coord) + "% /" + str(equi) +
9               "% /" + str(frz) + "%"
```

Aquí se observa que se utiliza la función de `matplotlib.pyplot` (redefinida con el nombre "plotter"), `subplot()`, para configurar los ejes por los que se va a representar la figura, el objeto `axesObject`, y la figura en sí, el objeto `figureObject`. Al `axesObject` le definimos la gráfica como una figura de tipo **pie** (gráfico de sectores), y le introducimos las unidades que va a representar, que son los porcentajes de cargas de los ejercicios (que representan la media de las cargas de todos los ejercicios incluidos en la sesión) y el nombre de dichas porciones, el parámetro `label=cargas`. De esta forma, obtenemos una representación visual del porcentaje de cargas.

La otra gráfica disponible en la aplicación es la que representa la evolución del peso de un usuario a lo largo de las sesiones de entrenamiento que va realizando. Las variaciones en el peso de un usuario son observaciones que se introducen como valoración del entrenador en la sesión correspondiente. Nuevamente, hacemos uso de la librería `matplotlib.pyplot` para dicha representación:

```
1 def MostrarGraficaPeso(self, pesos, fechas):
2     figureObject, axesObject = plotter.subplots()
3     if len(pesos) > 7:
4         while len(pesos) > 7:
5             pesos.pop(0)
6             fechas.pop(0)
7
8     axesObject.plot(fechas, pesos)
9     self.axesObject = axesObject
10    plotter.figure(figsize=(100,100))
11    canvas = FigureCanvas(figureObject)
12    self.boxgrafica.pack_start(canvas, True, True, 0)
13    self.win.show_all()
```

Nuevamente se utiliza la función `subplots()` para la configuración de sus ejes, como vemos en el fragmento de código anterior. Un ajuste realizado en la función es el de limitar el número de datos a representar para evitar introducir demasiados (en el código, está limitado a 7 datos de peso). Los datos introducidos en los ejes son simplemente las fechas con sus pesos correspondientes empleando en el eje x las fechas ordenadas cronológicamente.

# Pruebas de funcionamiento y resultados

En este apartado se procederá a mostrar las pantallas más importantes de esta aplicación para entender su implementación y el funcionamiento de estas. En esta sección se mostrarán capturas destacables de las secciones pero además, todas las capturas de la aplicación están situadas en el anexo A, dedicado a ellas.

## 8.1 Sección de usuarios

Para probar el funcionamiento de las distintas partes que tiene la sección de usuarios, se han propuestos diferentes escenarios para comprobar que todos los controles implementados hacen su función correctamente, además de que los datos se guardan debidamente en la base de datos.

Como se ha mencionado anteriormente, en todas las secciones habrá un control del formato en los distintos datos de entrada, tanto para la fecha, los datos numéricos, el correo electrónico o el DNI (figura 8.1):

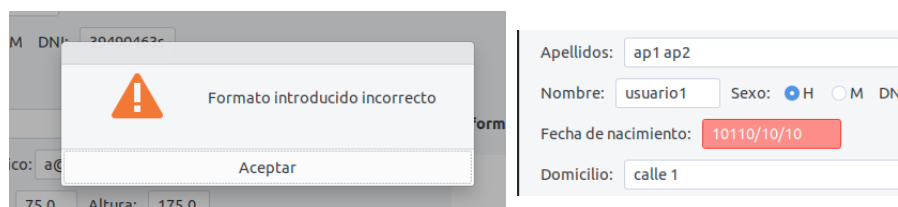


Figura 8.1: Ejemplo de mal formato de fecha

En caso de introducir una lesión o patología cuyas palabras se parezcan a alguna de las ya introducidas, el sistema aconsejará por medio de una ventana emergente alguna de ellas,

sino, el usuario puede introducirla finalmente (figuras 8.2 y 8.3):

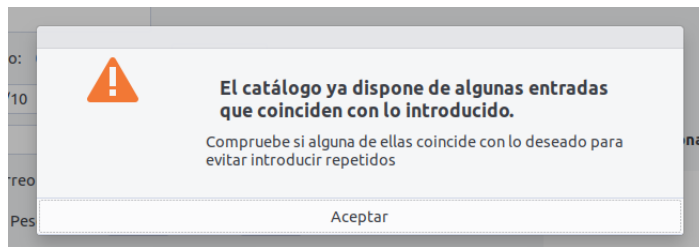


Figura 8.2: Advertencia por dato ya existente en base de datos.



Figura 8.3: Tras la advertencia, el sistema propone entradas similares.

Una vez un usuario realiza una sesión, se le pueden añadir valoraciones personales sobre esa sesión. Esto se realiza accediendo a su historial deportivo, el cual además de ver sus sesiones realizadas, muestra una gráfica con la variación del peso, a lo largo de las sesiones (figura 8.4). Además, como se ha mencionado en anteriores apartados, en su valoración se puede incluir variaciones de peso y/o lesiones adquiridas (figura 8.5).

## 8.2 Sección de ejercicios

Para dar de alta un ejercicio es necesario acompañar el nombre con porcentaje de cargas sobre 100% (aeróbica, fuerza, coordinación y equilibrio), un vídeo o foto explicativo, materiales a utilizar y lesiones y/o patologías por los cuales sea contraproducente realizarlo. Además, se han introducido otras advertencias como las de las figuras 8.6 y 8.7:

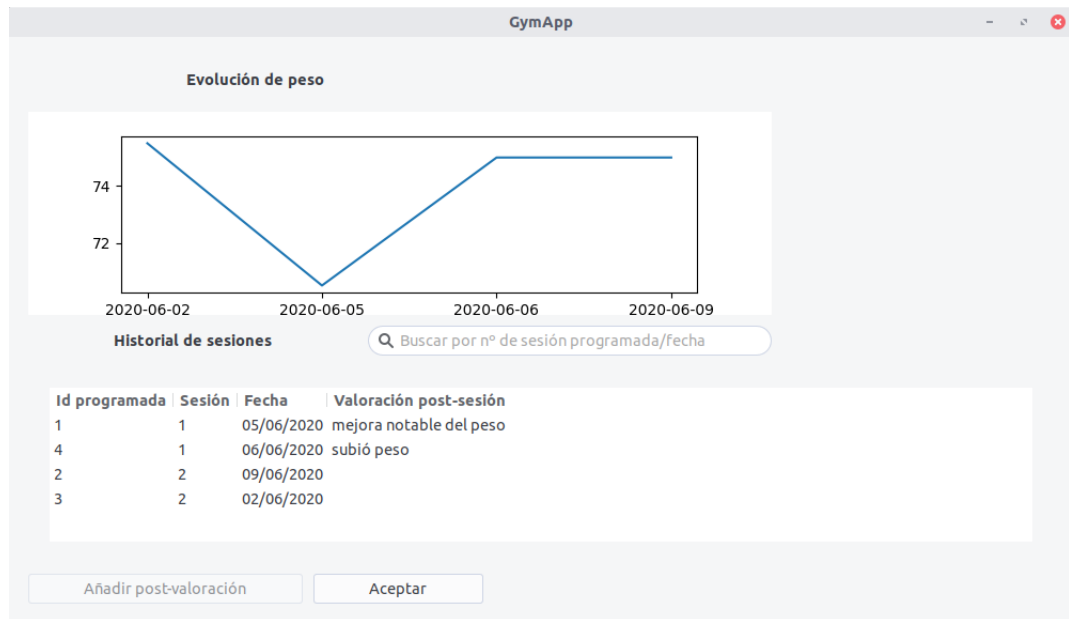


Figura 8.4: Historial deportivo de un usuario.

The screenshot shows the 'Detalle sesión 1' form. It contains the following elements:

- A table with columns 'Nº' and 'Ejercicio':
 

0	flexiones
1	press banca
- A 'valoración post-sesión' field containing the text 'mejora notable del peso'.
- A 'Modificaciones de ejercicios:' field, which is currently empty.
- A 'Peso nuevo adquirido:' field with the value '75,0' and minus/plus buttons.
- An 'Añadir lesión adquirida...' button and an 'Añadir' button.
- A 'Consultar catálogo de lesiones' button.
- A 'Guardar' button.
- An 'Aceptar' button at the bottom right.

Figura 8.5: Valoraciones post-sesión, variaciones de peso y lesiones adquiridas sobre un usuario.

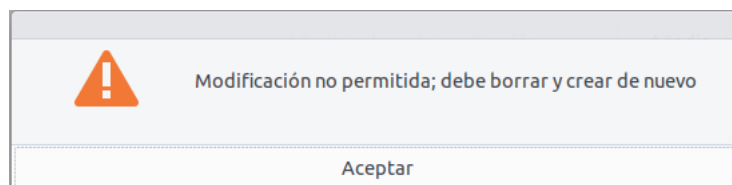


Figura 8.6: Advertencia sobre el cambio de nombre de un ejercicio.

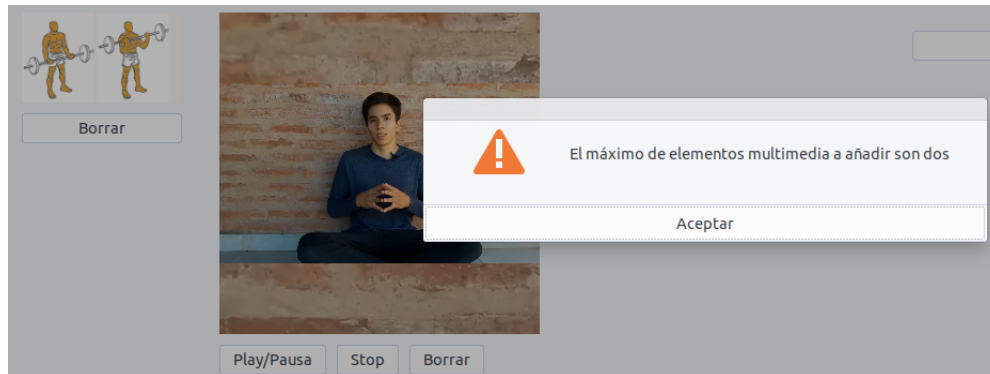


Figura 8.7: Ventana de error que indica que solo se puede introducir una foto y/o vídeo.

### 8.3 Sección de grupos

En tanto a los grupos, se ha comprobado que se almacenan adecuadamente en las tablas de la base de datos con su horario correspondiente, necesario para verificar que al programar una sesión, el día de la semana asignado es uno de los correspondientes a su horario, además de introducirle una serie de usuarios.

### 8.4 Sección de sesiones

Para las sesiones, se introduce una descripción y los ejercicios que la componen. De esta forma, nos mostrará las lesiones y patologías asociadas a los ejercicios y como se ha mencionado en el capítulo 7, se ha introducido una gráfica para representar la media de las cargas calculada en base a los ejercicios introducidos en la sesión (figura 8.8):



Figura 8.8: Gráfica con la media de las cargas en una sesión concreta.

### 8.5 Sección de programación

Por último para la sección de la programación de una sesión, se comprueba que se le asigna un grupo, una sesión y que las duraciones de descanso, reposo, calentamiento y duración siguen las unidades marcadas (s. y min.). Como se ha mencionado en el apartado anterior,

también se han implementado advertencias para indicar que un usuario asociado a esta sesión programada no dispone de ese día como horario habitual (figura 8.9), o que uno de los usuarios integrantes, dispone de una lesión o patología contraproducente (figura 8.10):

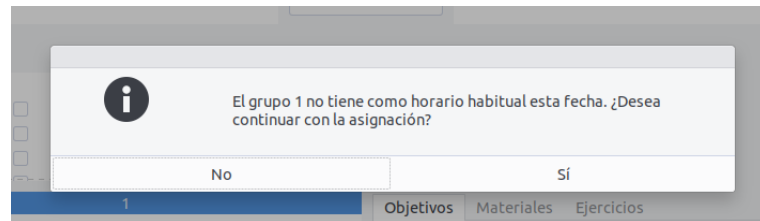


Figura 8.9: Advertencia de horario no habitual para algún usuario o grupo.

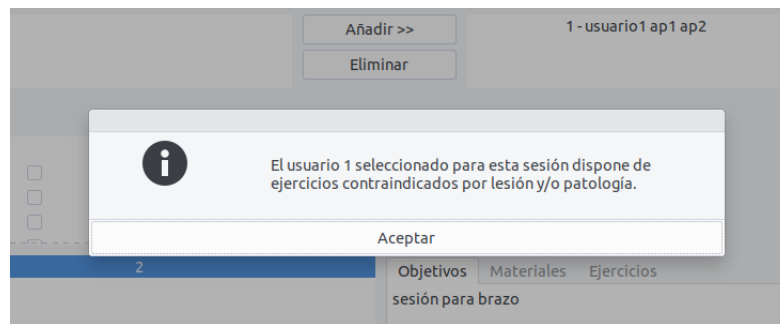


Figura 8.10: Advertencia de lesión o patología incompatible con la realización de algún ejercicio.

El sistema, una vez realizada una sesión, nos permitirá realizar comentarios sobre ella y seleccionar a un usuario para realizarle una valoración personal. Ambas opciones podremos consultarlas posteriormente (figura 8.11):

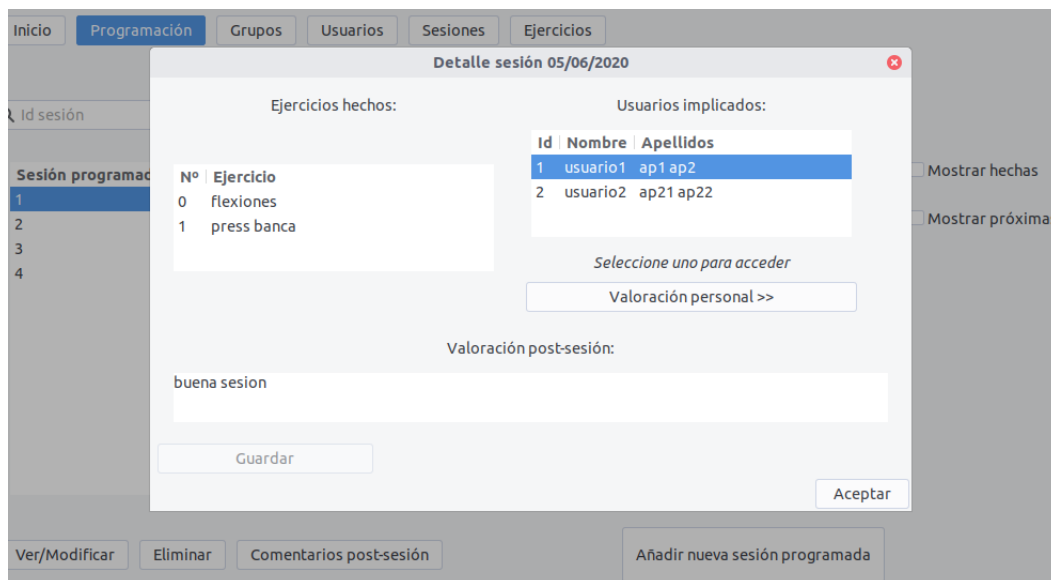


Figura 8.11: Valoración post-sesión.



# Conclusiones, lecciones aprendidas y líneas futuras de la aplicación

---

## 9.1 Conclusiones

El proyecto finalizado ha alcanzado los objetivos que se propusieron en un principio: gestión de usuarios, creación de ejercicios personalizados, su utilización en sesiones programadas y avisos útiles para dotar a la aplicación de una lógica adecuada. Por ello, el objetivo marcado en el anteproyecto y en el análisis de requisitos, se ha realizado satisfactoriamente. De todas formas, se han encontrado algunas dificultades que se han podido enmendar pero que han retrasado un poco el proyecto. Las dificultades encontradas fueron principalmente la distribución de la interfaz gráfica, es decir, la utilización de los contenedores, que resultó algo engorrosa y se esperaba un resultado más óptimo; la programación de un control de lesiones/-patologías repetidas; e integrar vídeos y fotos en la base de datos que llevó algo más de tiempo de lo esperado y muchas pruebas para que el formato fuera el adecuado. Además, no solo se han conseguido los objetivos, sino que se han añadido funcionalidades no pedidas, como la recomendación de lesiones/patologías ya disponibles en la base de datos, adaptar el formato de la foto introducida y la introducción de un historial deportivo con la capacidad de mostrar una gráfica con la evolución del peso de un usuario.

## 9.2 Lecciones aprendidas

La realización de este trabajo de fin de grado ha sido un desarrollo continuado de tres meses de duración que acarrió la necesidad de aportar bastante más a la organización de un proyecto software que en cualquiera de las prácticas realizadas a lo largo del grado. Esto es debido en gran medida a la independencia y autonomía a la hora de realizarlo. Esta autonomía trae consigo la necesidad de organizarse también a nivel personal, al tratarse de un trabajo

100% original y antes nunca realizado con estas características.

Por lo tanto, una de las lecciones aprendidas, es la necesidad de disponer de una capacidad de **organización** y de intentar mejorarla día a día para obtener mejores resultados, por lo menos, en el ámbito de desarrollo software. Otro aspecto a valorar, es la importancia que tienen los **recursos y documentaciones en línea**. Sin una documentación tan específica acerca de las herramientas utilizadas, como el caso de las librerías específicas de Gtk y Python, no se habría podido llevar a cabo correctamente este proyecto. Y por último cabe destacar como otra lección aprendida, la necesidad de **conocer adecuadamente un lenguaje de programación** para poder llevar a cabo en un tiempo adecuado un proyecto de estas características. Este aspecto debe destacarse ya que muchas veces se cree que por tener algunas nociones de lo que es un lenguaje, ya se puede replicar esto en aplicaciones más amplias, y al no tener un conocimiento un poco holgado, el tiempo de realización de un proyecto de tamaño mediano aumentaría de forma casi exponencial.

En resumen, a lo largo de este grado se han obtenido unas nociones y unas capacidades que considero adecuadas para la salida al mercado laboral en este sector. Además, a pesar de no haber manejado muchos lenguajes y herramientas para realizar interfaces gráficas (entre otras cosas, lo achaco a ser alumno de la rama de TI y no de desarrollo software), considero que me he manejado adecuadamente y de forma esperada con una herramienta como es GTK.

De todas formas sí que vería adecuado ampliar más allá de esta herramienta, alguna otra algo más actualizada (GTK+ es la única cursada para realizar interfaces gráficas en local, sin disponer de un servidor, en tercer curso en la asignatura interfaces persona-máquina, concretamente).

### 9.3 Líneas futuras

La aplicación desarrollada tenía como objetivo principal la sencillez y eficacia, y centrándose plenamente en el objetivo. A pesar de ello, se proponen una serie de cambios y/o funcionalidades que podrían añadirse:

- Capacidad de imprimir resúmenes de las sesiones.
- Capacidades en línea, tales como utilizar un servidor web donde guardar los datos y poder utilizarlos en otros dispositivos.
- Realizar una interfaz orientada a los clientes, no solo para los entrenadores.
- Incluir información más detallada de los usuarios, tales como edad, información de salud o condición física general e implantarlo a la hora de realizar sesiones más personalizadas.

# Apéndices



# Capturas completas de la aplicación

---



Figura A.1: Pantalla nada más iniciarlo.

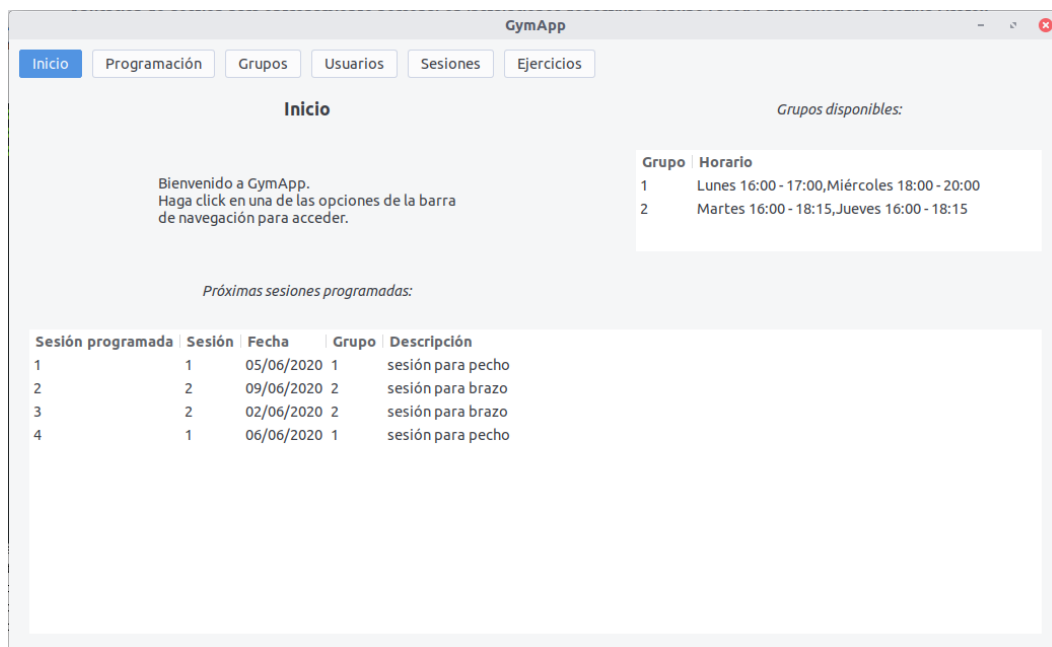


Figura A.2: Pantalla de bienvenida.

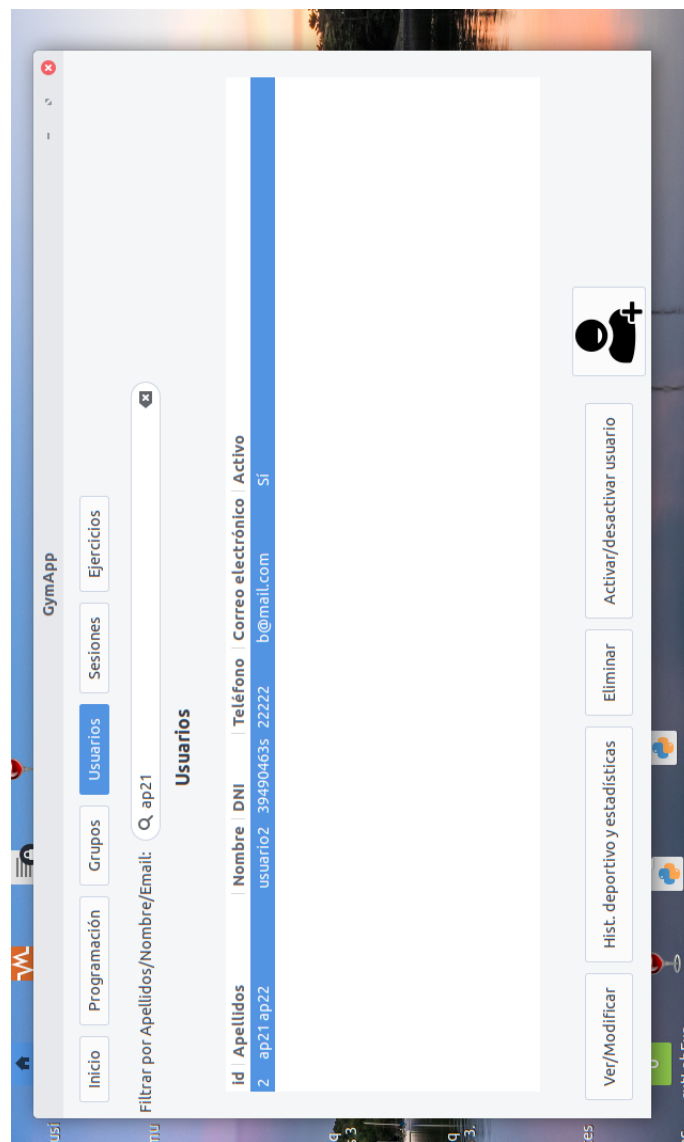


Figura A.3: Sección de usuarios con filtrado.

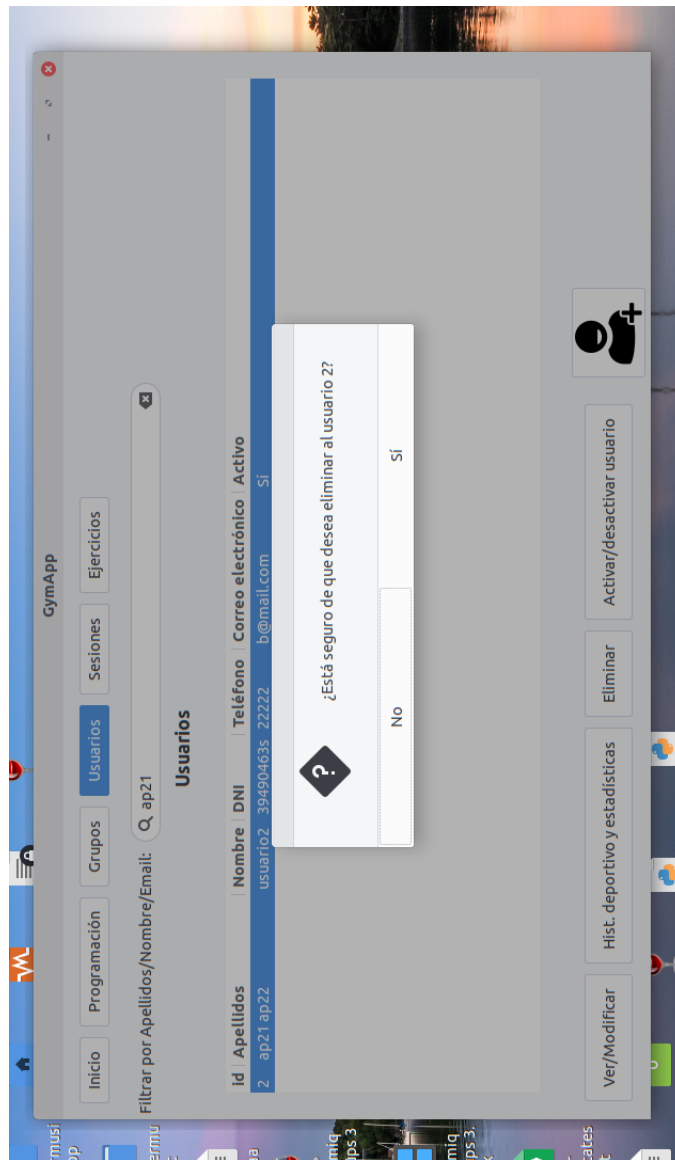


Figura A.4: Advertencia de eliminación de usuario.



The screenshot shows a web browser window titled 'GymApp' displaying a 'Formulario de usuario' (User Form). The form is organized into several sections:

- Personal Information:** Includes fields for 'Apellidos' (Last Name) with the value 'ap21 ap22', 'Nombre' (Name) with 'usuario2', 'Sexo' (Gender) with 'M' selected, and 'DNI' (National ID) with '39490463s'. There is a 'Cambiar foto' (Change photo) button and a note: 'Haga "Click" para añadir una foto' (Click to add a photo).
- Contact Information:** Includes 'Fecha de nacimiento' (Date of Birth) '12/12/12', 'Domicilio' (Address) 'calle 2', 'Teléfono' (Phone) '22222', and 'Correo electrónico' (Email) 'b@mail.com'.
- Physical Attributes:** Includes 'Peso actual' (Current Weight) '75.0' and 'Altura' (Height) '180.0'.
- Medical History:** A section titled 'Lesiones:' (Injuries) with the text 'microrotura isquios hombre' (ischio tendon tear in man). Below it are 'Añadir' (Add) and 'Borrar' (Delete) buttons, and a link to 'Consultar catálogo de lesiones' (View injury catalog).
- Additional Information:** A section titled 'Patologías:' (Pathologies) with a large empty text area. Below it are 'Añadir' (Add) and 'Borrar' (Delete) buttons, and a link to 'Consultar catálogo de patologías' (View pathology catalog).
- Actions:** At the bottom right, there are 'Guardar' (Save) and 'Cancelar' (Cancel) buttons.

Figura A.5: Formulario de modificación de un usuario.

GymApp

### Formulario de usuario

Haga "Click" para añadir una foto

Apellidos:

Nombre:  Sexo:  H  M DNI:

Fecha de nacimiento:

Domicilio:

Teléfono:  Correo electrónico:

Categoría:  Peso actual:  Altura:

Lesiones:

Patologías:

Figura A.6: Ejemplo de formato incorrecto de DNI.

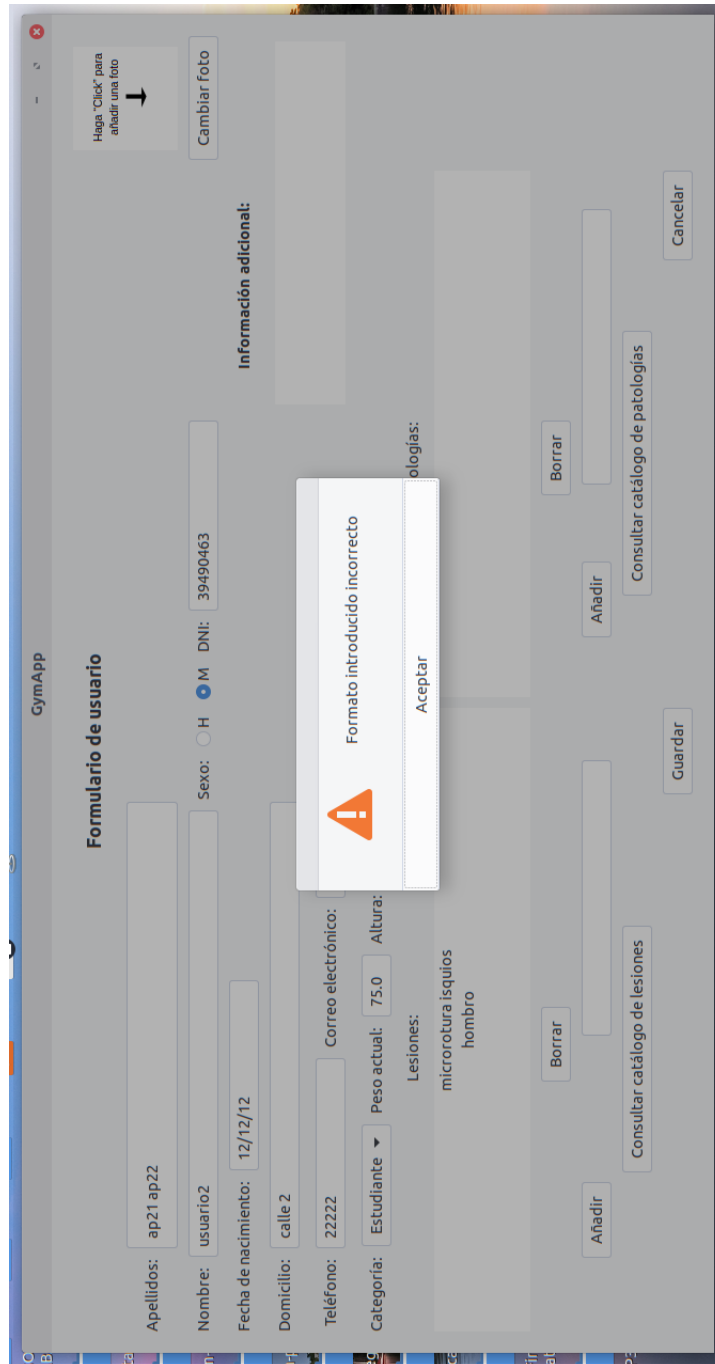


Figura A.7: Aviso de formato incorrecto.

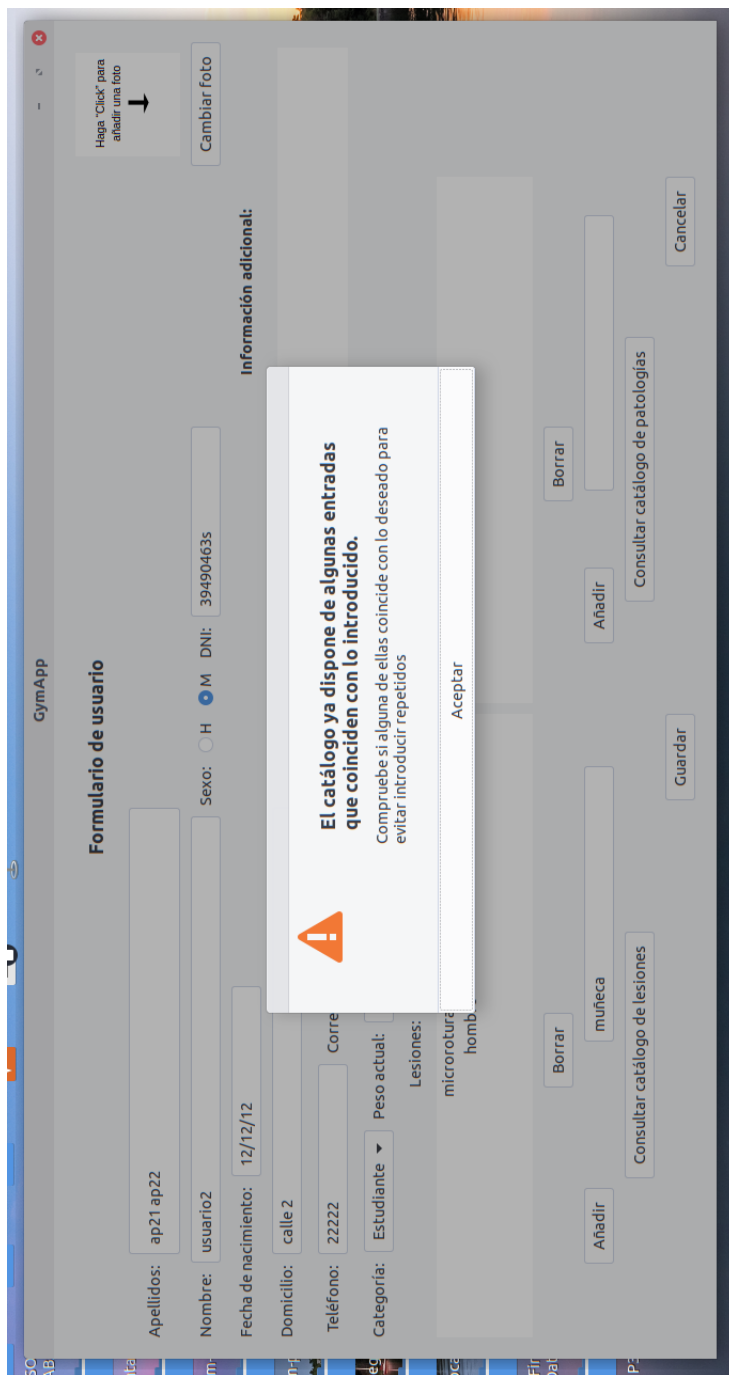


Figura A.8: Aviso de lesión introducida con palabras similares a alguna ya disponible.

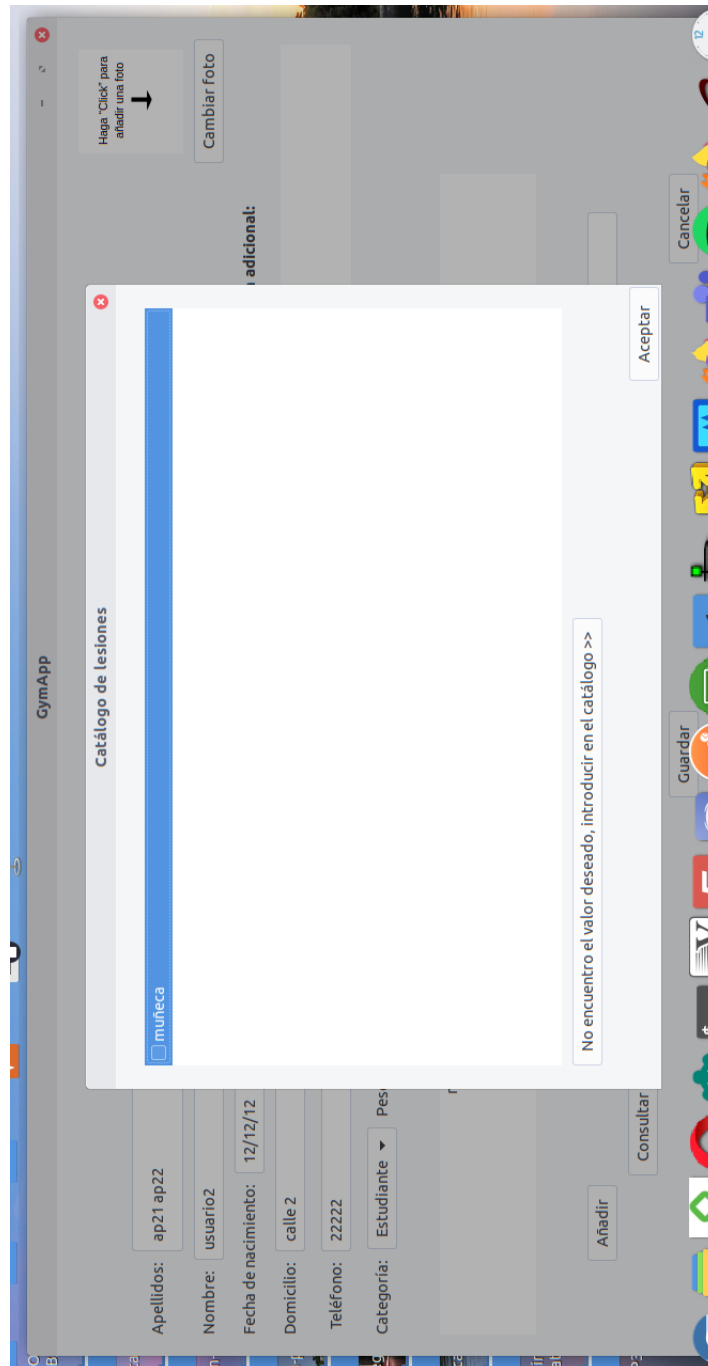


Figura A.9: Recomendación de lesiones disponibles.

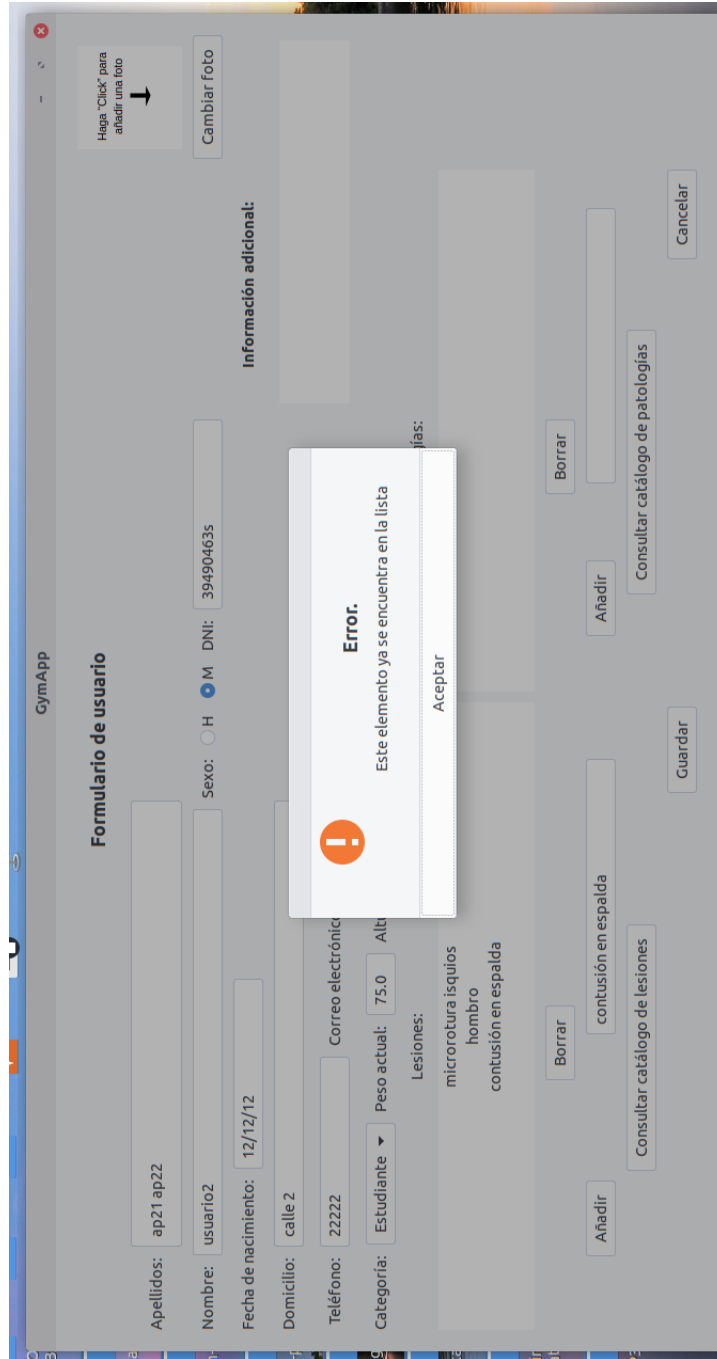


Figura A.10: Lesión ya en lista.

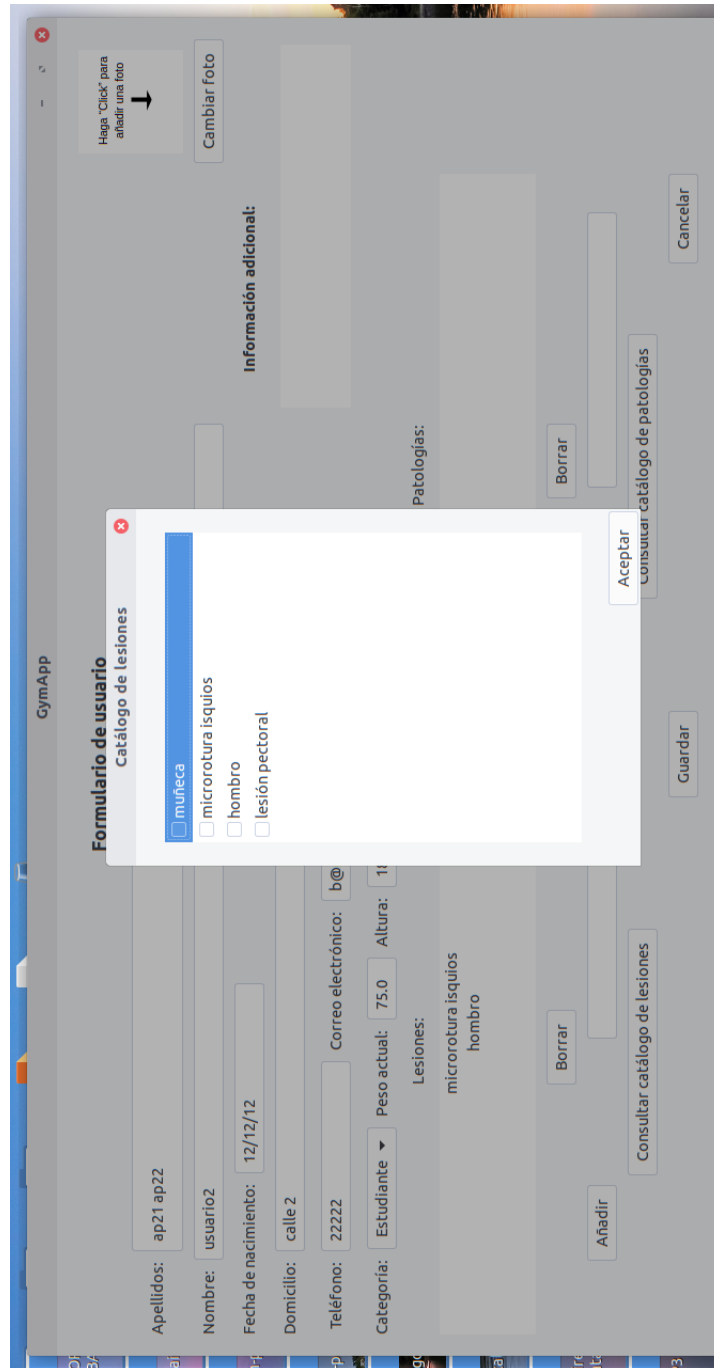


Figura A.11: Catálogo de lesiones.



Figura A.12: Historial deportivo de un usuario con su gráfica de evolución de peso y sesiones realizadas.



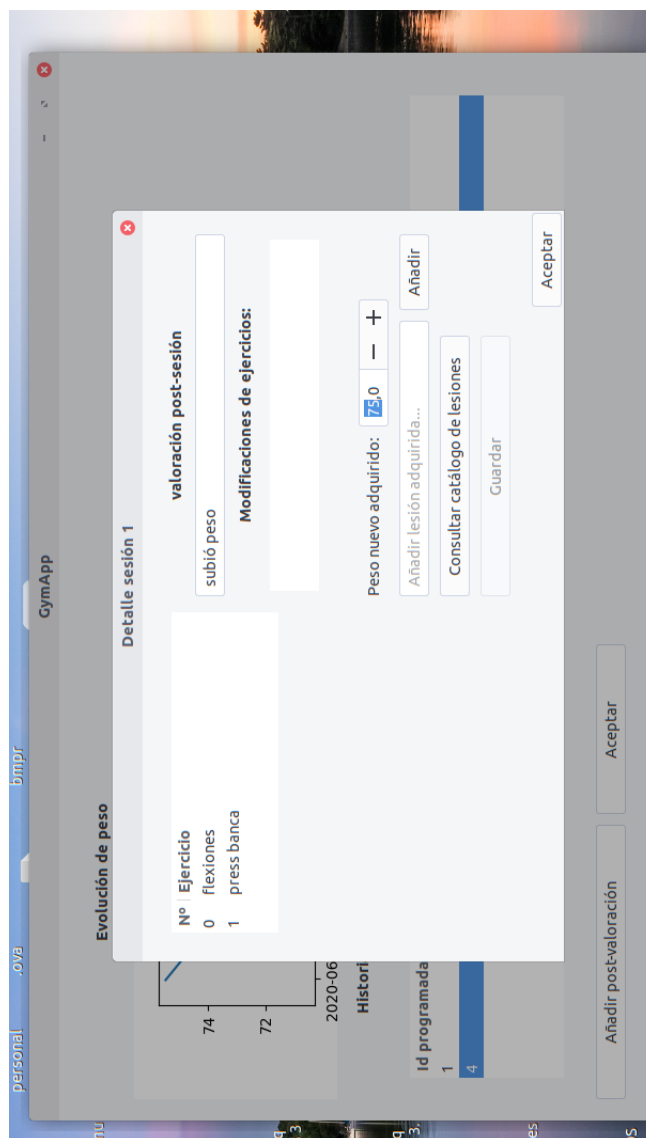


Figura A.13: Valoración personal de una lesión y un usuario en concreto con posibilidad de introducir nuevo peso o lesiones adquiridas tras la sesión.

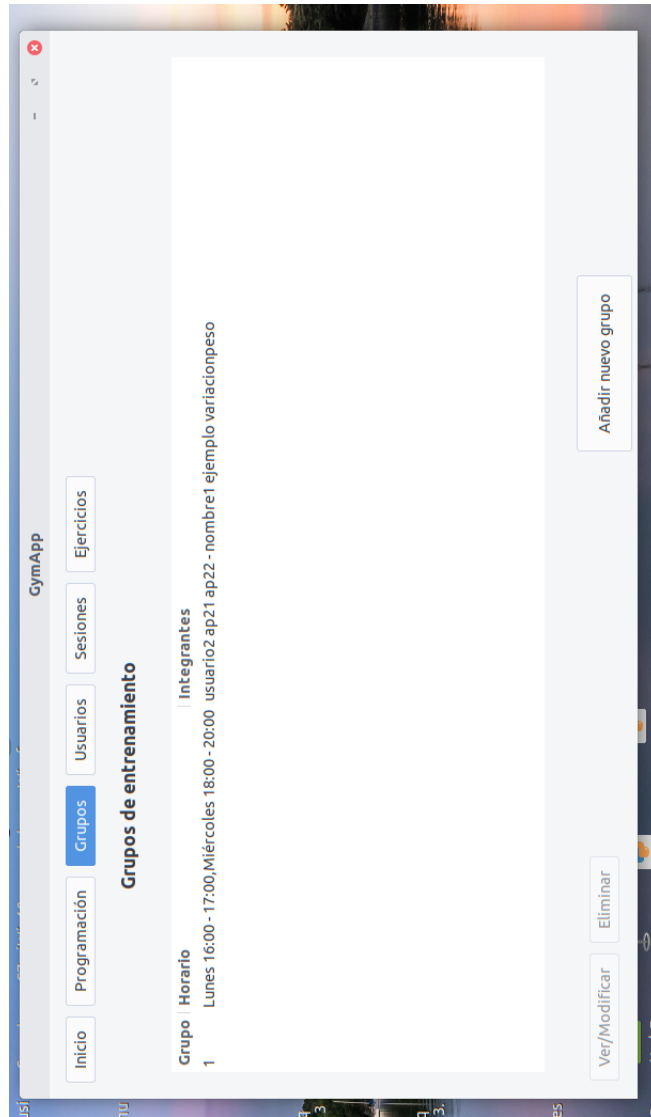


Figura A.14: Sección de grupos.

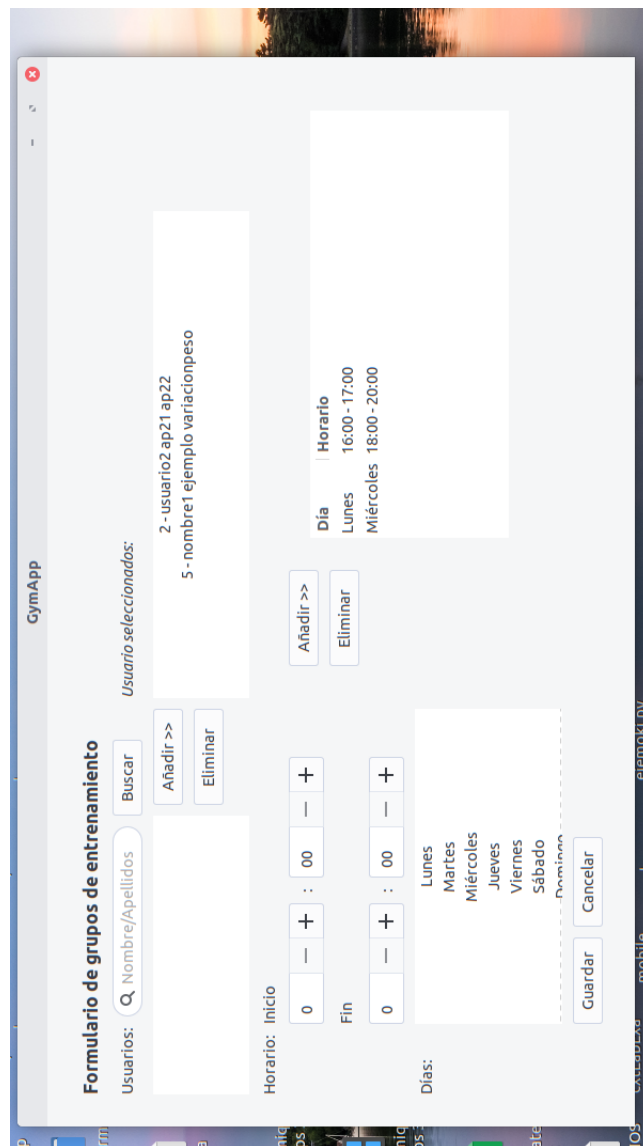


Figura A.15: Formulario de modificación de datos de grupos.

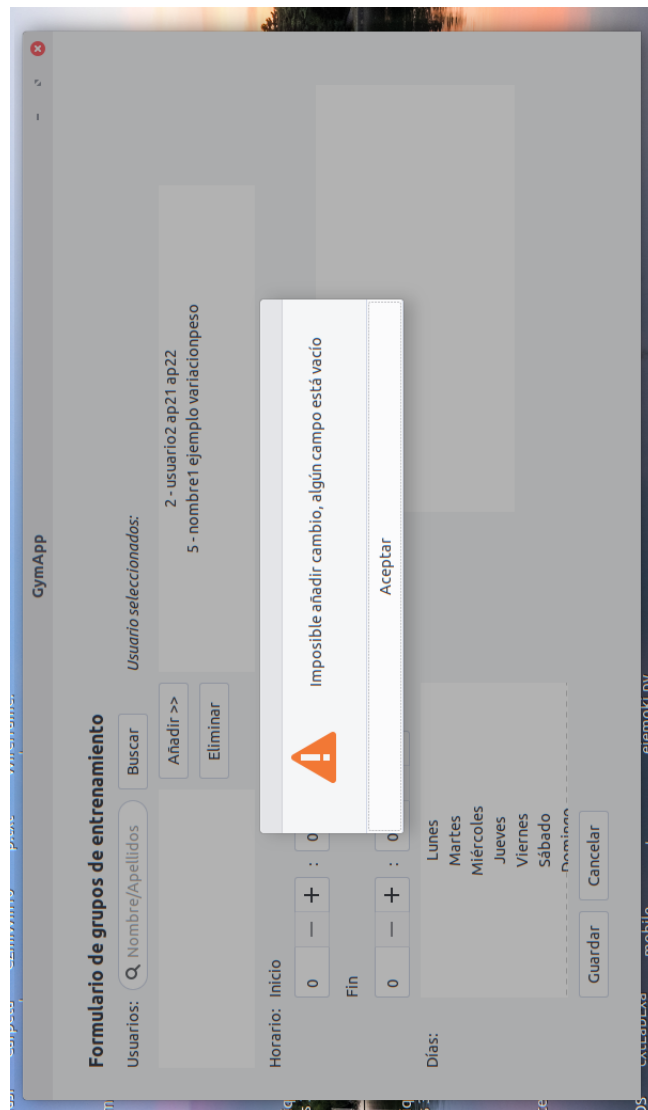


Figura A.16: Aviso de algún dato vacío (en este caso, asignación de horario).

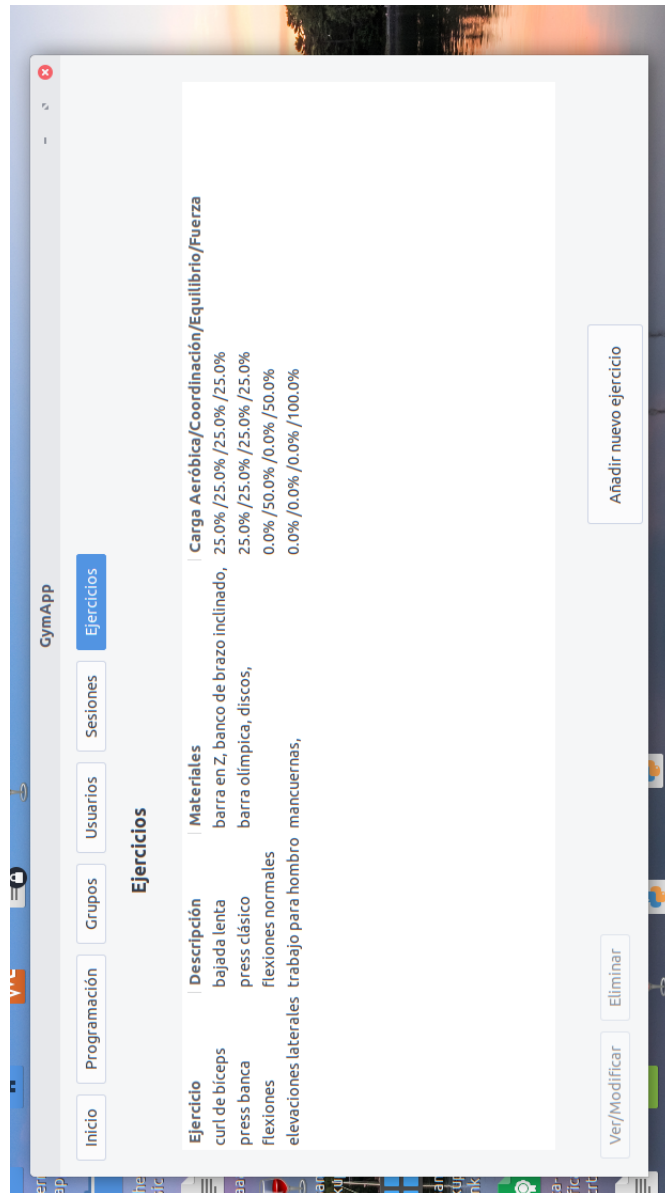


Figura A.17: Sección de ejercicios.

GymApp


### Formulario de Ejercicio

Nombre:

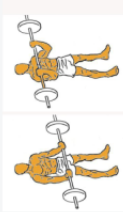
Descripción:

Materiales:

Realización:



*Enlaces añadidos:*



Evitar lesiones: muñeca, lesión pectoral

Evitar patologías: lumbago

Figura A.18: Formulario de modificación de ejercicios.

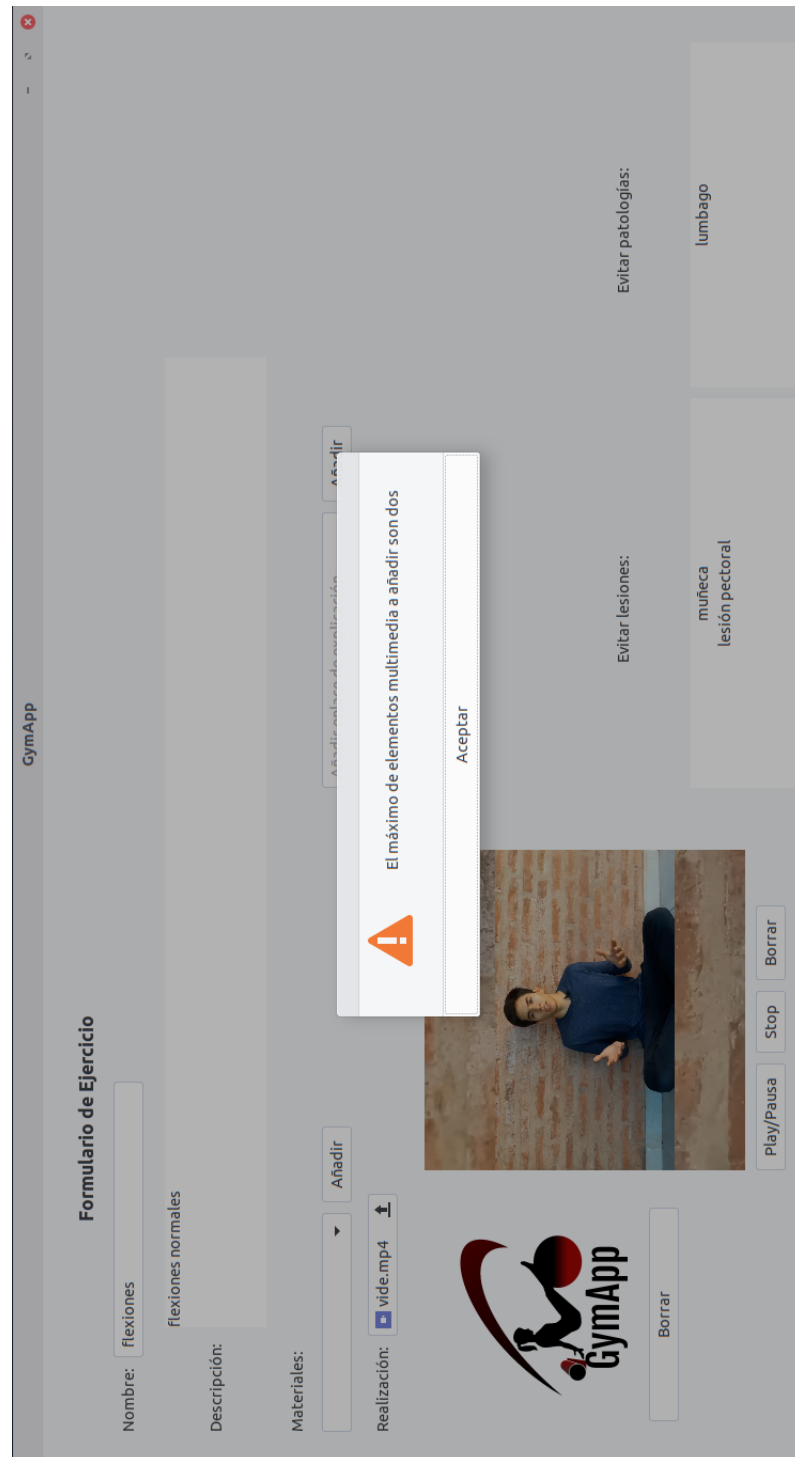


Figura A.19: Aviso al intentar introducir más de un elemento multimedia.

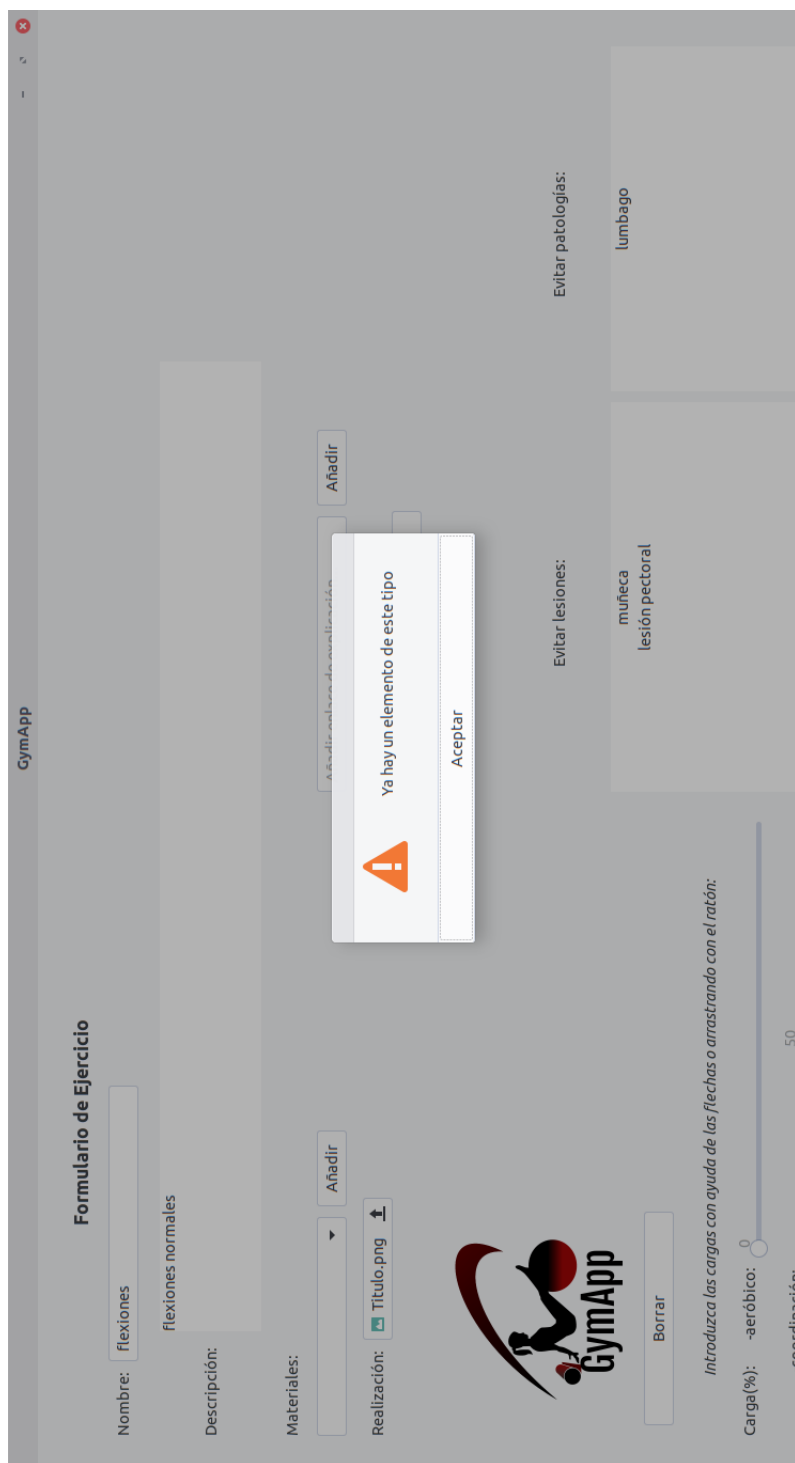


Figura A.20: Solo se permite un vídeo y/o una foto, uno de cada.



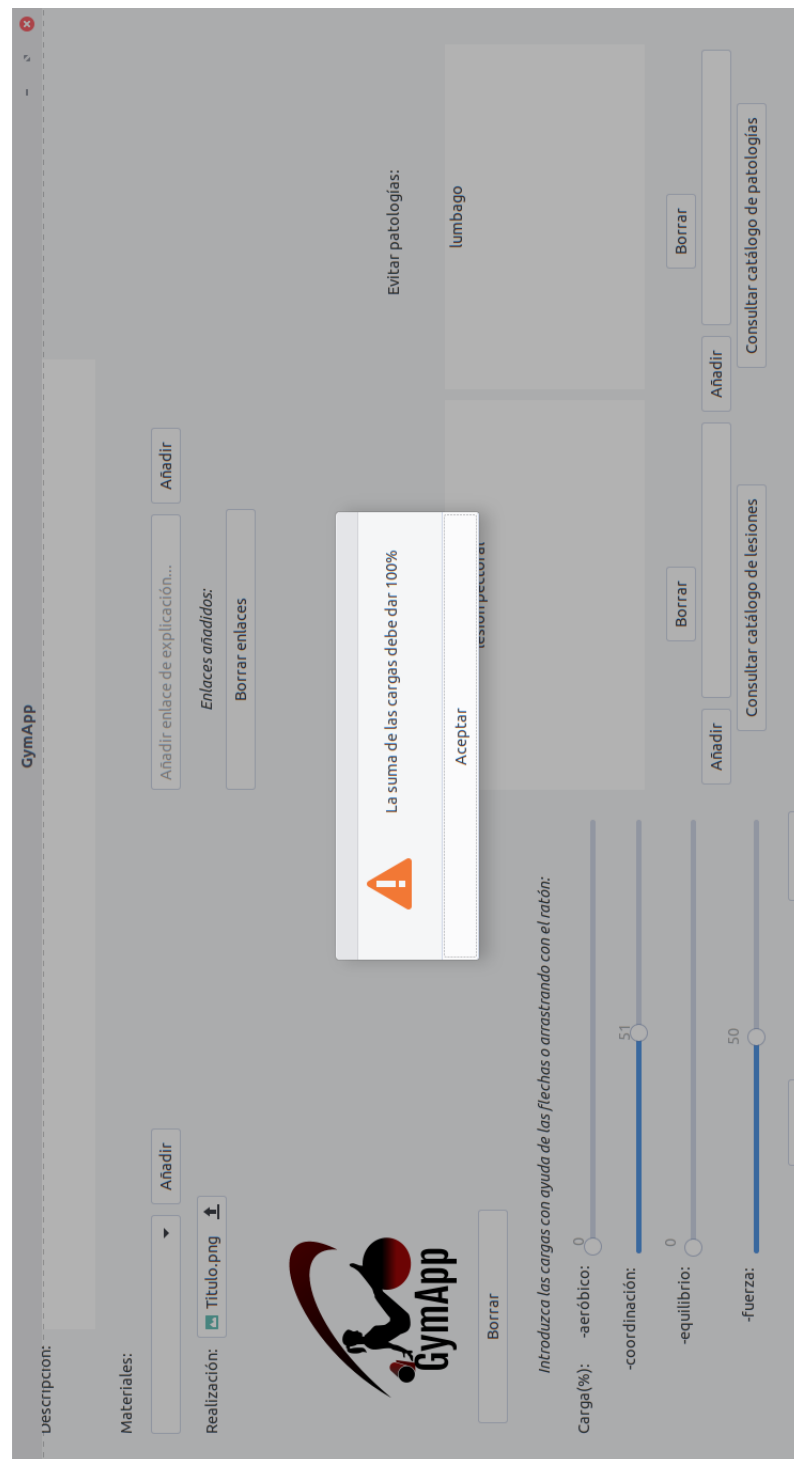


Figura A.21: La suma de las cargas no debe exceder el 100%.

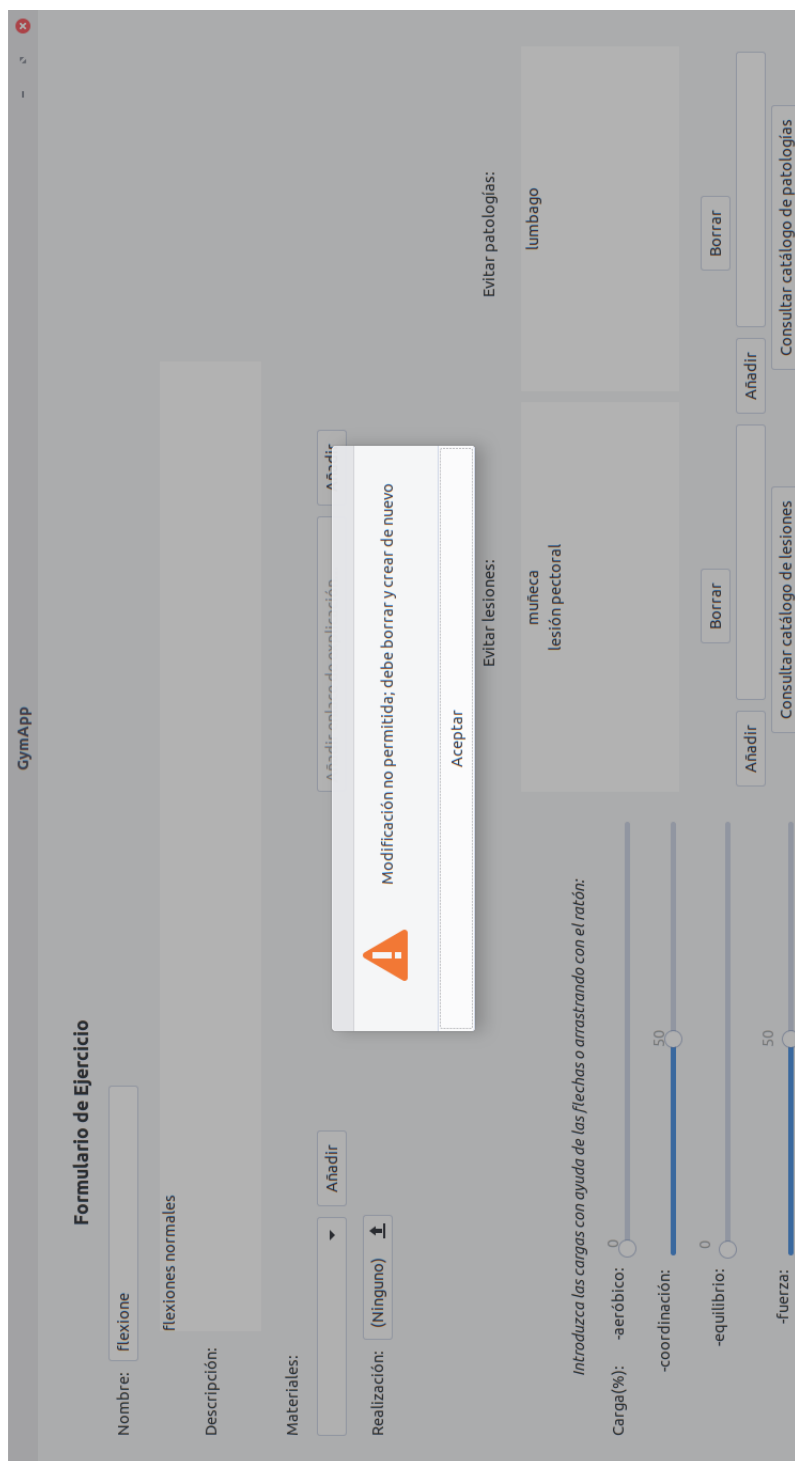


Figura A.22: Por consistencia, para modificar el nombre de un ejercicio se debe borrar y crearlo de nuevo.

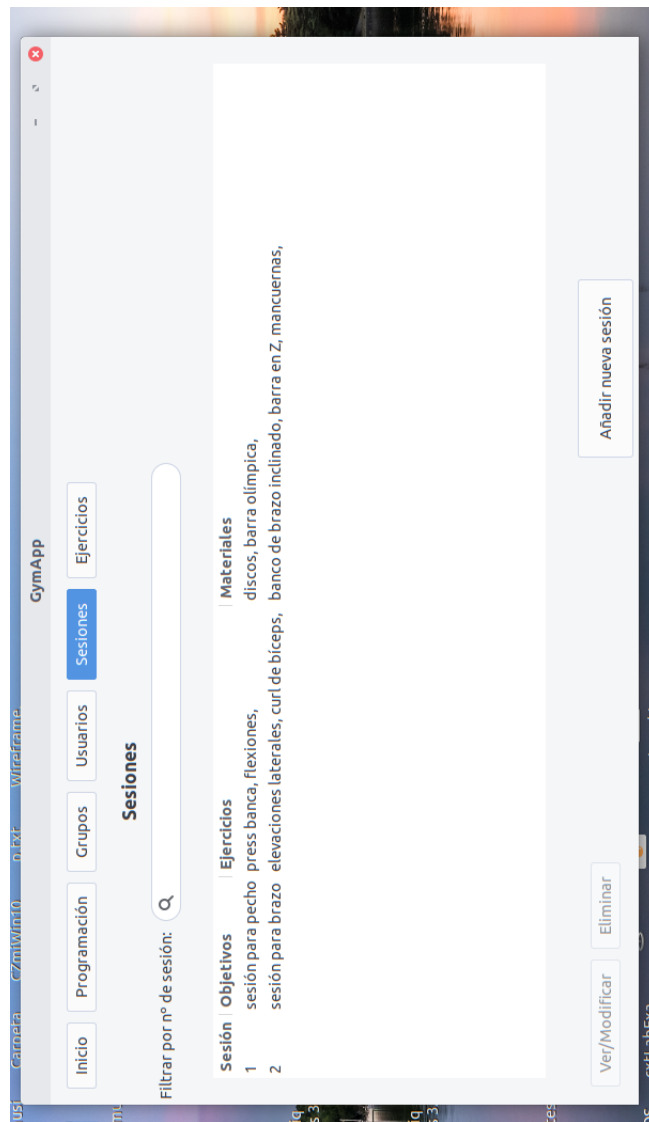


Figura A.23: Sección de sesiones.

GymApp

### Formulario de Sesión

Objetivos: sesión para pecho

Ejercicio:  ▼

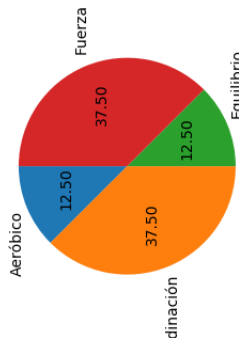
flexiones  
press banca

Materiales:

discos  
barra olimpica

Incompatibilidades:

muñeca  
lesión pectoral  
lumbago



Categoría	Valor
Fuerza	37.50
Coordinación	37.50
Aeróbico	12.50
Equilibrio	12.50

Figura A.24: Formulario de modificación de sesiones.

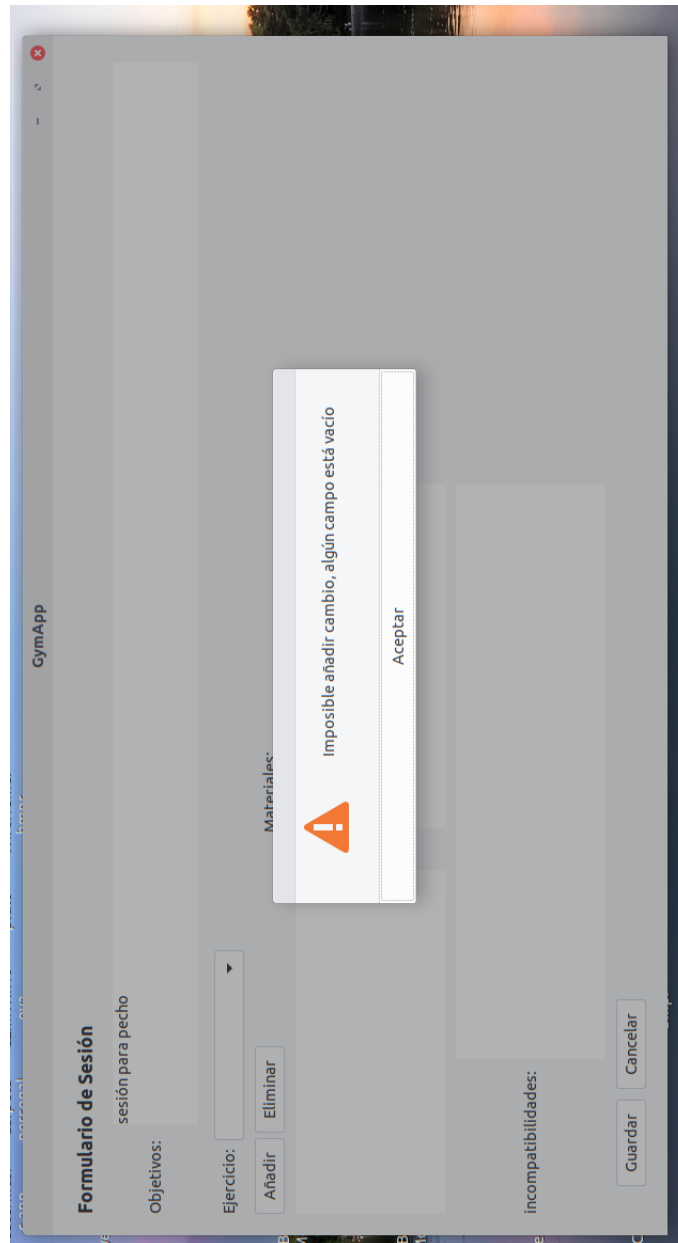


Figura A.25: Aviso de sesión sin ejercicios.



Figura A.26: Sección de programación de sesiones.

**Programación de Sesión**  
*Busque por nº de grupo o por nombre/apellidos de usuario:*

Grupo nº: 1   Nombre/Apellidos

Usuario seleccionados: 4-usuario4 ap41 ap42

**Búsqueda de sesiones:**

Filtro

- ▶ Materiales
- ▶ Aeróbico
- ▶ Equilibrio
- ▶ Coordinación
- ▶ Fuerza

Mostrar sesiones >> 1

Seleccione una sesión:

Objetivos: Materiales Ejercicios  
 sesión para pecho

*Rellene las duraciones indicando su unidad (min ó s).*

Vueltas: 3  Calentamiento: 10min  Descanso: 3min  Duración ejercicio: 10min  Reposo: 1min  Tiempo de la sesión: 81 min 0 s

Figura A.27: Formulario de creación de sesiones programadas.

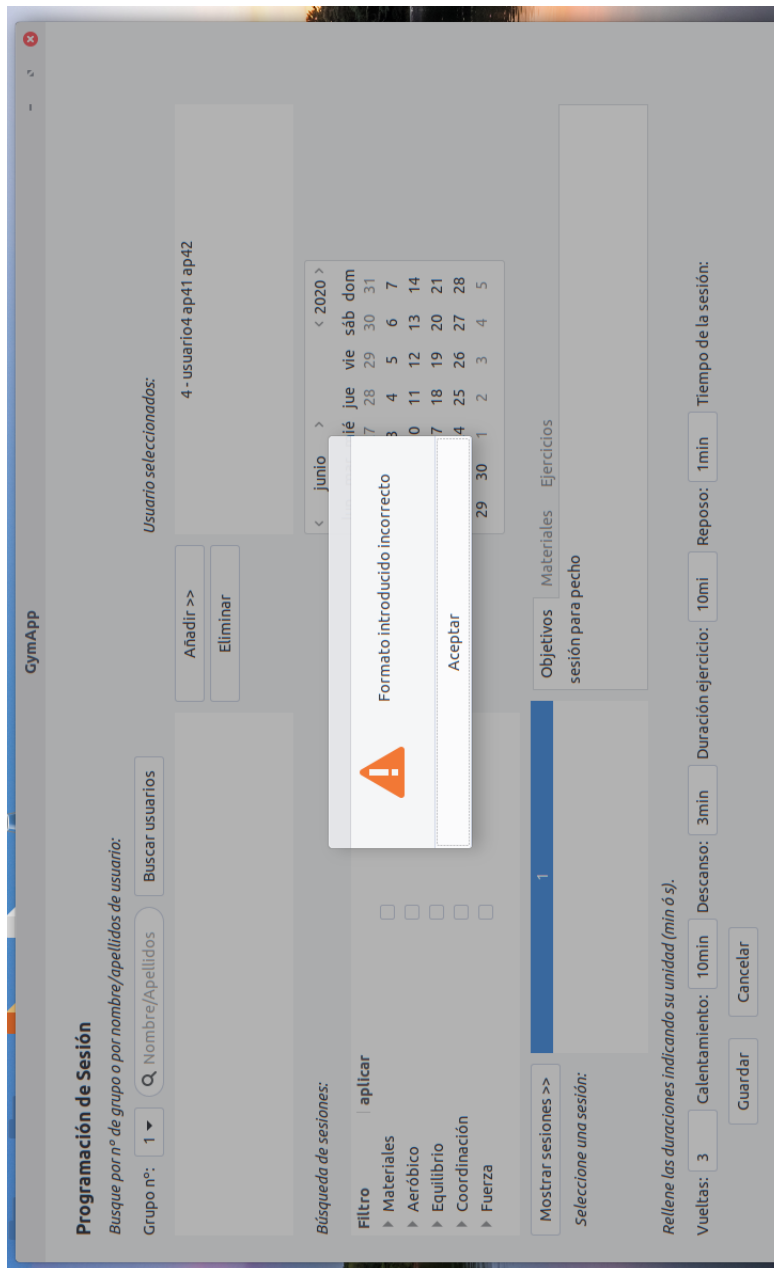


Figura A.28: Ejemplo de mal formato en los tiempos.



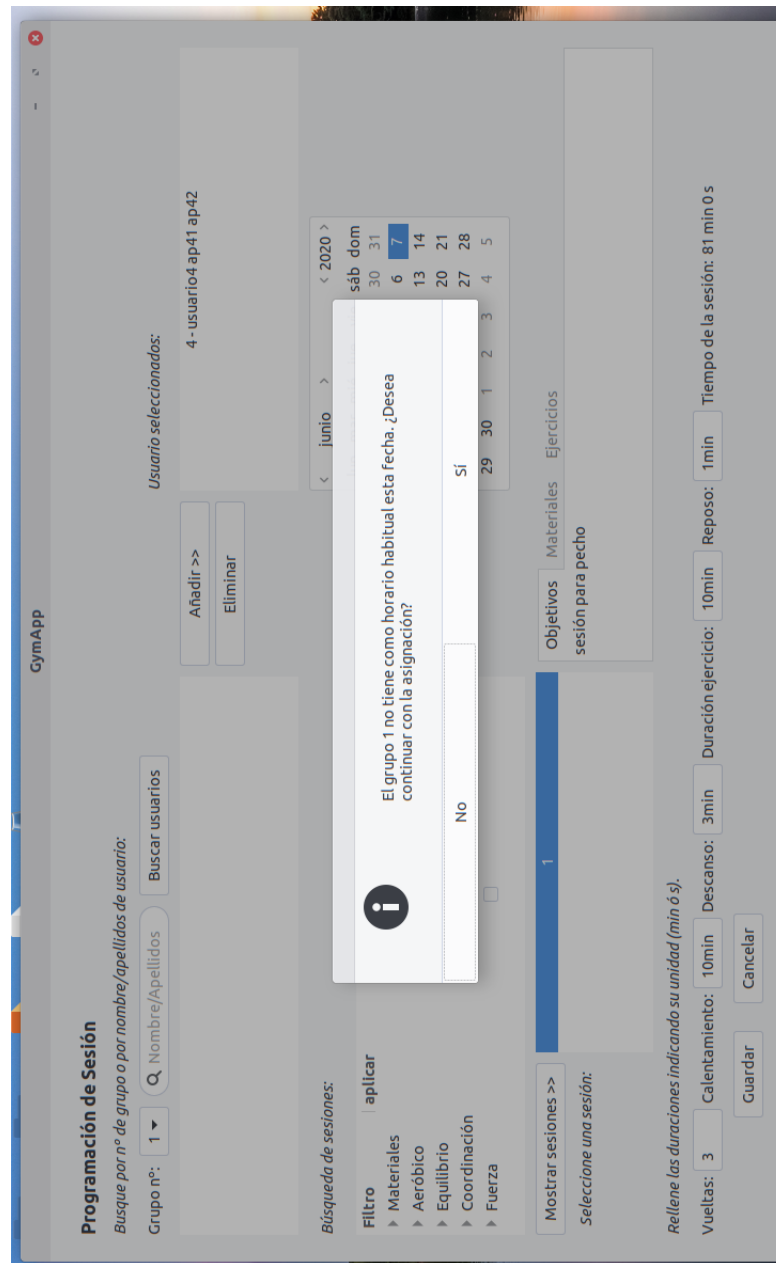


Figura A.29: Aviso de día que no corresponde al grupo asignado.

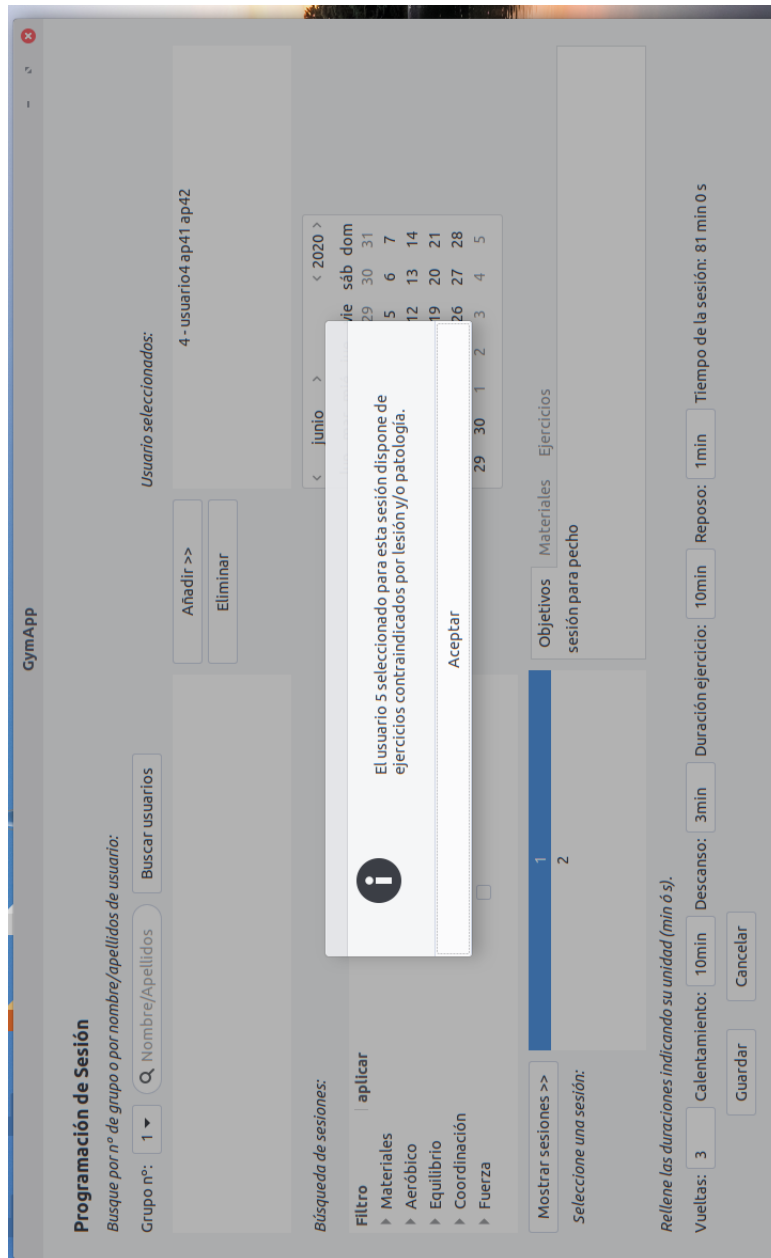


Figura A.30: Advertencia de usuario con lesión incluido en la programación con algún ejercicio contraproducente.

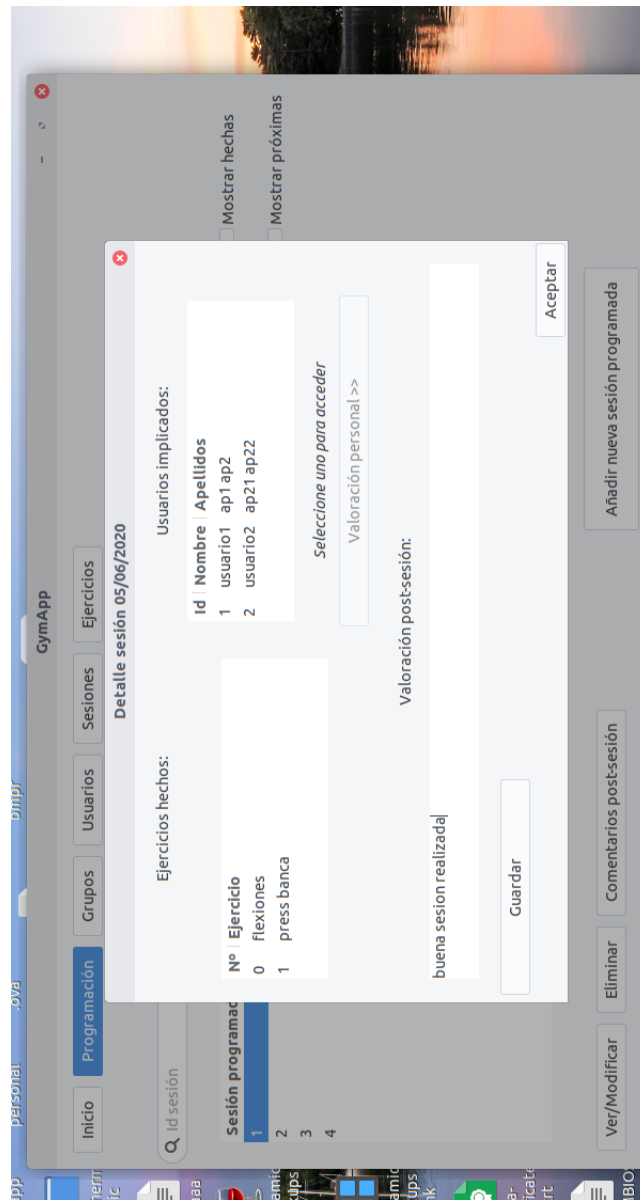


Figura A.31: Valoración post-sesión.

---

# Contenido del directorio de la aplicación, enlace de gitHub y dependencias (requisitos no funcionales)

---

## B.1 Enlace de gitHub y contenido del directorio

El enlace del repositorio de gitHub es de uso público:

<https://github.com/pablocollazo-fic/pablo.collazoTFG.git>

El directorio de la aplicación llamado **"GymApp"** contiene una serie de elementos:

- Archivos *main.py*, *controller.py*, *view.py* y *model.py*.
- Lista de materiales utilizados dentro de la base de datos (archivo solo de referencia).
- Archivo *database.db* que contiene las tablas de la base datos con datos de prueba y tabla de materiales completa.
- 3 fotos (Titulo.png, add.svg y foto.png) de uso interno de la aplicación.
- 1 elemento multimedia de ejemplo para los ejercicios (1366\_2000.jpg)

## B.2 Ejecución

Para la ejecución de el programa simplemente es necesarios situarse en el directorio GymApp y por medio de terminal ejecutar "python3 main.py".

## B.3 Requisitos no funcionales

Su producción y prueba fue realizado en un S.O. Linux Ubuntu 18.04 LTS en un entorno Gnome.

Para la correcta ejecución del programa es necesarios disponer de una serie de dependencias instaladas:

-python 3. Con librerías extra:

- - - pygobject (3.26.1).

- - - python-vlc (3.0.9113).

- - - numpy (1.16.3).

- - - matplotlib (2.1.1).

- - - Pillow (5.3.0).

-python3-matplotlib. (También desde el repositorio apt para correcto funcionamiento)

-sqlite3.

# Lista de acrónimos

---

**UX** *User eXperience.*

**GUI** *Graphical User Interface.*

**MVC** *Model-View-Controller.*

**GTK** *Gimp ToolKit.*

**(Xerox) PARC** *Palo Alto Research Center*





# Glosario

---

**Wireframe** Referido a un sitio web, es una guía visual que representa el esqueleto o estructura de un sitio web utilizado en la fase de diseño.

**Framework** También conocido como **marco de trabajo** se refiere a un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia para enfrentar y resolver nuevos problemas de índole similar.[20]



# Bibliografía

---

- [1] Developer.chrome, “Mvc.” [En línea]. Disponible en: [https://developer.chrome.com/apps/app\\_frameworks](https://developer.chrome.com/apps/app_frameworks)
- [2] Trainingym, “¿qué es trainingym?” [En línea]. Disponible en: <https://trainingym.com/que-es-trainingym/>
- [3] Virtuagym, “Virtuagym.” [En línea]. Disponible en: <https://business.virtuagym.com/es/software-gestion-gimnasio/#manage>
- [4] Train2Go, “Train2go.” [En línea]. Disponible en: <https://train2go.com/>
- [5] Wikipedia, “Gui.” [En línea]. Disponible en: [https://es.wikipedia.org/wiki/Interfaz\\_gráfica\\_de\\_usuario#Historia](https://es.wikipedia.org/wiki/Interfaz_gráfica_de_usuario#Historia)
- [6] Balsamiq, “Balsamiq.” [En línea]. Disponible en: <https://balsamiq.com/company/>
- [7] S. Lab, “How we use python at spotify.” [En línea]. Disponible en: <https://labs.spotify.com/2013/03/20/how-we-use-python-at-spotify/>
- [8] DjangoProject, “Django.” [En línea]. Disponible en: <https://www.djangoproject.com>
- [9] Matplotlib.pyplot, “Matplotlib.pyplot.” [En línea]. Disponible en: [https://matplotlib.org/3.2.1/api/\\_as\\_gen/matplotlib.pyplot.html](https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.html)
- [10] G. GTK, “Gtk.” [En línea]. Disponible en: <https://www.gtk.org/docs/>
- [11] pyobject, “pyobject.” [En línea]. Disponible en: <https://pyobject.readthedocs.io/en/latest/>
- [12] Widget, “Widget.” [En línea]. Disponible en: <https://python-gtk-3-tutorial.readthedocs.io/en/latest/basics.html>
- [13] SQLite, “About sqlite.” [En línea]. Disponible en: <https://www.sqlite.org/about.html>

- [14] Python, “Sqlite3.” [En línea]. Disponible en: <https://docs.python.org/3/library/sqlite3.html>
- [15] T. M. H. Reenskaug, “Mvc xerox parc 1978-79.” [En línea]. Disponible en: <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>
- [16] —, “Models - views - controllers.” [En línea]. Disponible en: <http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf>
- [17] ScrumManager, “Modelo original de scrum para desarrollo de software.” [En línea]. Disponible en: [https://www.scrummanager.net/bok/index.php?title=Modelo\\_original\\_de\\_Scrum\\_para\\_desarrollo\\_de\\_software](https://www.scrummanager.net/bok/index.php?title=Modelo_original_de_Scrum_para_desarrollo_de_software)
- [18] Scrum, “The scrum guide.” [En línea]. Disponible en: <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100>
- [19] Vlc, “Vlc lib.” [En línea]. Disponible en: <http://www.olivieraubert.net/vlc/python-ctypes/doc/vlc.MediaListPlayer-class.html>
- [20] Wikipedia, “Framework.” [En línea]. Disponible en: <https://es.wikipedia.org/wiki/Framework>