Universitat Oberta de Catalunya

uoc.edu

# gGym: Aplicación móvil para la gestión de pequeños centros deportivos

**Jose Enrique Álvarez Rendal**
Máster Universitario en Desarrollo de Aplicaciones Móviles
Trabajo Final de Máster DADM

**Nombre Consultor/a:** David Escuer Latorre
**Nombre Profesor/a responsable de la asignatura:** Carles Garrigues Olivella

Junio 2019

**FICHA DEL TRABAJO FINAL**

| | |
|---|---|
| **Título del trabajo:** | *gGym: aplicación para la gestión de pequeños centros deportivos* |
| **Nombre del autor:** | *Jose Enrique Álvarez Rendal* |
| **Nombre del consultor/a:** | *David Escuer Latorre* |
| **Nombre del PRA:** | *Carles Garrigues Olivella* |
| **Fecha de entrega (mm/aaaa):** | *06/2019* |
| **Titulación:** | *Máster Universitario en Desarrollo de Aplicaciones Móviles* |
| **Área del Trabajo Final:** | *Trabajo Final de Máster DADM* |
| **Idioma del trabajo:** | *Castellano* |
| **Palabras clave** | *iOS / Gestión / Deporte* |


**Resumen del Trabajo (máximo 250 palabras):** *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Hoy en día es difícil para los centros deportivos pequeños pelear de igual a igual con las grandes cadenas deportivas, que cuentan con mayores medios tanto económicos como tecnológicos para la gestión de los centros y la captación de clientes. Este es el punto de partida de este Trabajo Final de Master: diseñar una aplicación para ayudar a estos pequeños empresarios con la gestión de sus centros, que sea sencilla y amigable y con un coste que puedan asumir.

La aplicación se desarrollará siguiendo una metodología de trabajo en espiral para garantizar que se puedan introducir nuevos requerimientos y cambios durante el desarrollo del trabajo, de forma que se itere de forma constante por los requisitos identificándolos, analizándolos y, si es posible, implementándolos.

El resultado del desarrollo de este Trabajo es una aplicación móvil para la plataforma iOS, que consume los datos a través de un backend disponible en la nube, con el que garantizamos la independencia, la reusabilidad y la extensibilidad de la solución. La aplicación desarrollada cubre la mayor parte de los requisitos expuestos, y además se han identificado una serie de mejoras para ampliar la base de público objetivo. Como conclusión, he de destacar como lección aprendida la importancia de las fases previas al desarrollo del producto, hecho que se remarca en las conclusiones, para conseguir el objetivo final alcanzado.

**Abstract (in English, 250 words or less):**

Today it's very difficult for the smallest sports centres to compete with the biggest companies of the sector, who have better economic and technological means for managing the centres and attracting new clients. This will be the starting point of this project: the need of design and develop a solution for helping this smallest centres with their management tasks, which will be based in a mobile application that must be easy to use and understand and with the smallest cost possible for them, so that they can afford it.

This application will be developed following a spiral methodology in order to guarantee that new requirements can be introduced during its development, so that we are always iterate over the requirements identifying new ones, making an analysis and, if possible, develop them so that they can be included in the final product.

The result of this project is a mobile application for the iOS platform, which is consuming and saving data in a server application available in the cloud. This way, we are guaranteeing the independence of both applications and the extensibility and reusability of the solution. The developed solution fulfils the majority of the requirements exposed, and we have identified a set of improvements in order to reach more public. Just to conclude, I have to highlight as the main lesson learned the importance of all the phases previous to the development of the product due to its importance for reaching the final objective.

# Índice de Contenidos

# Índice de Figuras

# Índice de Tablas

# Capítulo 1: **Introducción**

# Capítulo 1. Introducción

## 1.1. Contexto y justificación del Trabajo

Hoy en día es difícil para los centros deportivos pequeños, dirigidos por pequeños empresarios locales y emprendedores, pelear de igual a igual con las grandes cadenas deportivas, que cuentan con mayores medios económicos, estrategias de captación de socios más agresivas, y presencia en la mayoría de las redes sociales y tiendas de aplicaciones móviles para complementar su oferta de valor.
Estas grandes cadenas utilizan complejas soluciones de software para gestionar tanto el funcionamiento de los centros en sí, a semejanza de otras grandes empresas, como la gestión de las relaciones y la fidelización de los clientes. Para esta tarea suelen apoyarse en paquetes de software CRM cuyo coste es inasumible para un pequeño empresario. Además, tal y como se ha comentado, la mayoría de estas cadenas disponen de presencia en las principales tiendas de aplicaciones móviles, lo que les permite tener un trato muy cercano al usuario y ofrecerle una serie de servicios al alcance de su mano de una forma muy cómoda para él mismo.

Por el contrario, es habitual que los pequeños centros deportivos utilicen, a menudo, el papel para la gestión de los socios, los pagos, y otra serie de necesidades enmarcadas en su funcionamiento diario. En algunas ocasiones se apoyan también en software ofimático para facilitarles parte estas tareas. Este será el punto de partida de este Trabajo Final de Máster: ofrecer una solución a este tipo de pequeñas empresas (nuestro público objetivo) que les facilite la gestión diaria, con las premisas de diseñar una solución fácil de utilizar, asequible y cuya curva de aprendizaje sea lo más pequeña posible.

## 1.2. Estado del arte

Tal y como se ha comentado en el apartado anterior, la mayoría de grandes empresas del sector se apoyan en paquetes de software muy complejos para la gestión del funcionamiento. Estos paquetes de software suelen gestionar muchos aspectos, tales como el control de accesos al centro, la fidelización de clientes, la impresión de tarjetas de socio, etc.
Sin embargo, desde una mirada a alto nivel nuestros requisitos son distintos: la búsqueda en el mercado se centrará en una solución que gestione los socios, los pagos y la gestión de las salas y las pistas deportivas del centro. Es decir, no será necesario que la solución de software incluya algunos los puntos comentados en el apartado anterior: fidelización de clientes, campañas publicitarias, control de accesos, etc.

En los siguientes apartados detallaremos algunas de las soluciones existentes en el mercado, destacando los puntos en común con la solución que se pretende desarrollar en este Trabajo Final de Máster, y las diferencias con la misma.

### 1.2.1. Soluciones modulares

Software modulable GPA Sport *(Sitio web de GPA Sport 2019)*


*Ilustración 1: Software de gestión deportiva GPA Sport*

De acuerdo con la información del fabricante, este paquete de software es una solución modulable y parametrizable para adaptarse a las necesidades de cada centro deportivo. La solución parte de un paquete central encargado de la gestión de abonados, extensible con una serie de paquetes de gestión contable, control de accesos, puntos de venta, etc.
En este documento (*Dossier del software modulable de GPA Sport 2019)* se detallan cada uno de los módulos de los que dispone el software. Como puntos fuertes, podríamos destacar:

- Se parte de un módulo central de gestión de abonados, uno de los requisitos que se necesitaría cubrir.
- Se podría implantar una web para el centro deportivo con un gestor de contenidos.
- Se dispone de un paquete de terminal punto de venta (TPV) con el que el centro deportivo podría realizar los cobros a los abonados a través de tarjeta de crédito.
- Al ser un paquete modulable, podría acompañar el crecimiento del centro deportivo con nuevas funcionalidades soportadas por el mismo.

Sin embargo, se han detectado los siguientes inconvenientes:

- Aunque no aparecen precios en la web del fabricante, se asume un alto coste de implantación debido a la alta modularización y parametrización.
- La curva de aprendizaje de una solución de este tipo es muy alta. Además, se aprecian interfaces de usuario muy complejas en el folleto informativo.
- Las soluciones de este tipo “atarían” al empresario con el fabricante del software, puesto que no se tratan de soluciones universales basadas en aplicaciones web o móviles, si no en software cerrado o propietario.

### 1.2.2. Soluciones multiplataforma

Software de gestión Ismygym *(Sitio web del software IsMyGym 2019)*


*Ilustración 2: Software de gestión deportiva Ismygym*

Ismygym se anuncia como una solución integral para la gestión de clientes en gimnasios, basada en una aplicación web y/o móvil, muy orientada hacia los clientes del centro deportivo. De acuerdo a la web del fabricante, esta solución permite gestionar las reservas online, pagos, redes sociales y actividades del centro deportivo. Además, permite gestionar planes de fidelización y campañas publicitarias con los clientes.
Como ventajas de esta solución, podríamos destacar:

- Se trata de un sistema multiplataforma: web y móvil, por lo que no sería necesario una gran inversión inicial en hardware para adoptarla. La migración hacia otros sistemas sería sencilla también.
- La interfaz de usuario parece sencilla y amigable, y sería accesible a través de un navegador web estandar.
- El precio de la solución se adaptaría al volumen de usuarios de la misma, de acuerdo con los costes planteados en la web del fabricante.

La solución tendría los siguientes inconvenientes:

- No cubre las necesidades basicas que planteamos. Está muy orientada hacia el cliente, pero no hacia la gestión del centro en si.

- Para un centro deportivo pequeño es muy probable que se desaprovechasen la mayor parte de sus funcionalidades, por lo que la imagen de cara al cliente sería mala.

### 1.2.3. Tienda de aplicaciones de Android

La aplicación mas valorada (cinco estrellas) en la tienda de oficial de aplicaciones para Android, Google Play Store, se denomina Global Gym Software *(Aplicación móvil Global Gym Software - Google Play Store 2019).*

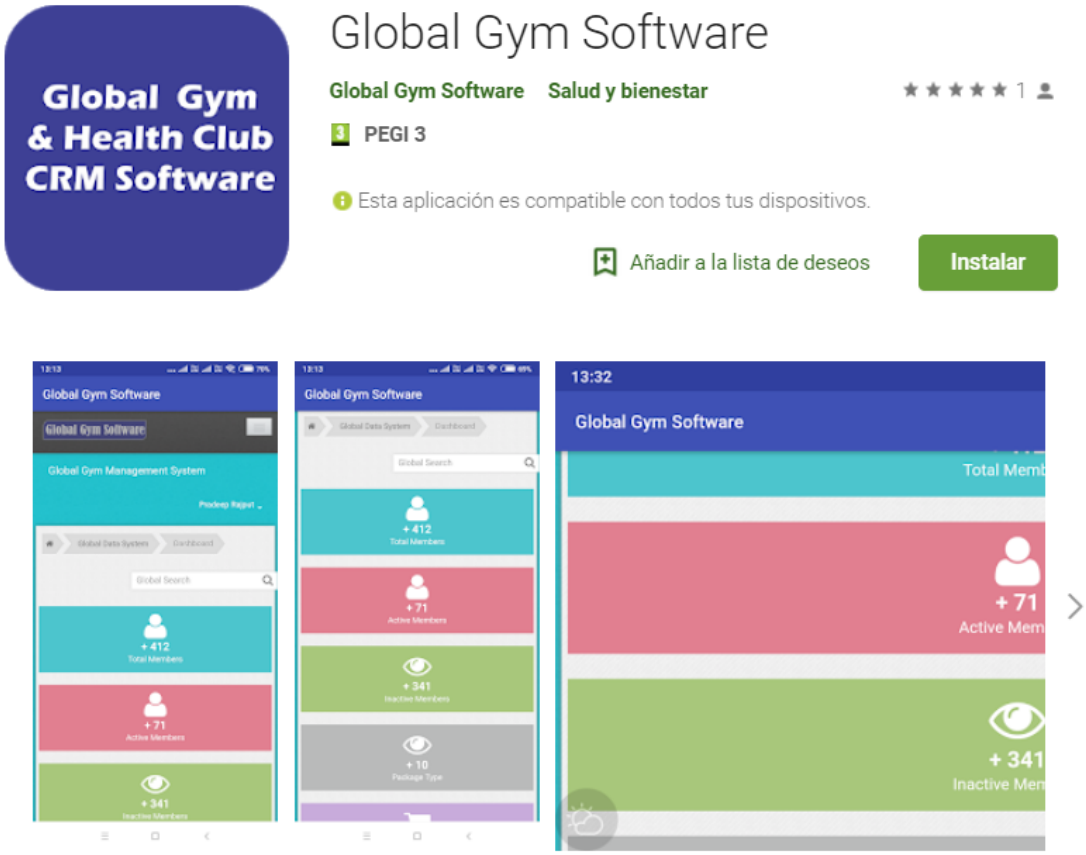



*Ilustración 3: Aplicación Global Gym Software para Android*

Esta aplicación, disponible para dispositivos móviles Android, se anuncia como una app para ahorrar costes y tiempo en las tareas administrativas de la gestión de un gimnasio. Entre las funcionalidades que anuncia se incluye el procesamiento de pagos, el seguimiento de clientes y la gestión de las rutinas de entrenamiento de los socios. Además, cuenta con una parte en la nube que intuimos amplia las funcionalidades de esta.

Como ventajas de esta solución, podríamos destacar:

- Se trata de una app móvil, por lo que no sería necesario una gran inversión inicial en hardware para adoptarla. Bastaría con disponer de una tableta o smartphone con este sistema operativo.
- Cubriría los requisitos expuestos de gestión de socios y pagos.

Entre los inconvenientes, hay que destacar:

- La interfaz de usuario está muy orientada a obtener datos de manera visual, con visualizaciones de tipo *dashboard*, pero no hacia la gestión en sí.
- La aplicación dispone de funcionalidades que no encajan en los requisitos planteados: cumpleaños recientes de los socios, gestión de los gastos en personal, etc. lo que acrecentaría la curva de aprendizaje de esta.
- No se dispone de información acerca de los precios de esta solución.

### 1.2.4. Conclusiones

Una vez analizado el estado del arte, habiendo detectado la existencia en el mercado de soluciones modulares, multiplataforma y para Android, se ha decido que la aplicación será un desarrollo para la plataforma iOS buscando cubrir este nicho de mercado, al no haber detectado durante este análisis una solución específica para esta plataforma. En los siguientes capítulos de la memoria de este Trabajo Final de Máster se desgranarán los requisitos y el enfoque para la consecución de este mismo.


*Ilustración 4: Logo de la plataforma iOS*

## 1.3. Objetivos del Trabajo

Los objetivos de este Trabajo Final de Máster son:

- El análisis de las necesidades de gestión de un pequeño centro deportivo para la toma de requisitos funcionales y no funcionales.
- La búsqueda de soluciones de software existentes en el mercado, y el análisis del número de requisitos cubiertos por las mismas para dar respuesta a las necesidades expuestas. Se deberá de incluir también como punto fundamental el coste de estas soluciones y la dificultad de implantación.
- El diseño y la construcción de una aplicación móvil para la plataforma iOS en base a los requisitos extraídos, que de respuesta a las necesidades detectadas. La solución de deberá de tener un coste de implantación asumible por un pequeño empresario y un alto grado de usabilidad con la menor curva de aprendizaje posible.
- La integración con sistemas externos que nos permitan extender las funcionalidades de la aplicación desarrollada en el futuro, en base a la reutilización de los datos generados por la aplicación móvil, de acuerdo

con las extensiones propuestas en la memoria de este Trabajo Final de Máster.

Para la consecución de los objetivos expuestos, será necesario que la solución desarrollada cubra los siguientes requisitos funcionales y no funcionales, que enumeraremos en los siguientes apartados.

### 1.3.1. Requisitos funcionales

| Código | Descripción |
|---|---|
| **RF1** | Se dispondrá de un fichero swagger para la descripción de la API REST expuesta por el backend a modo de contrato de comunicaciones. |
| **RF2** | El backend de la aplicación no permitirá la consulta de los datos almacenados a usuarios no autorizados por el sistema. |
| **RF3** | El backend de la aplicación permitirá al administrador del sistema el borrado de los datos almacenados en la misma. |
| **RF4** | El backend de la aplicación permitirá al administrador del sistema las operaciones de alta, baja y modificación de usuarios. |
| **RF5** | La aplicación móvil dispondrá de un sistema de registro/alta de nuevos usuarios. |
| **RF6** | La aplicación móvil dispondrá de un sistema de autenticación basado en usuario (correo electrónico) y contraseña. |
| **RF7** | En caso de que el sistema operativo permita otros métodos de autenticación de usuarios, como el uso de huella dactilar, se estudiará su uso como sustituto del usuario y la contraseña mencionados anteriormente. |
| **RF8** | La aplicación móvil no permitirá a usuarios sin registro el uso de las funcionalidades de esta misma. |
| **RF9** | La aplicación móvil permitirá las operaciones de gestión de abonados en el centro deportivo mediante las interfaces de usuarios específicas. |
| **RF10** | La aplicación móvil permitirá la captura y el almacenamiento de imágenes de los abonados del centro deportivo para su identificación por parte del gestor de este mismo. Este almacenamiento de imágenes se realizará en el dispositivo (y no en el backend) para la protección de las mismas. |
| **RF11** | La aplicación móvil permitirá la gestión de cobros mensuales y tarifas para el gestor del centro deportivo mediante las interfaces de usuarios específicas. |
| **RF12** | La aplicación móvil permitirá la gestión de equipamientos deportivos (pistas, salas, etc.) y la reserva de los espacios por parte del gestor del centro deportivo. |
| **RF13** | La aplicación móvil dispondrá de un apartado de configuración en el que se podrán eliminar todos los datos almacenados en la misma y borrar la cuenta del usuario. |
| **RF14** | Al apretar el botón de atrás del sistema operativo se accederá a la interfaz de usuario anterior a la actual. |

*Tabla 1: Requisitos funcionales de la aplicación*

### 1.3.2. Requisitos no funcionales

| Código | Descripción |
|---|---|
| **RNF1** | Se dispondrá de un backend alojado en una nube pública que gestionará la autenticación, el registro de usuarios y el almacenamiento de datos. |
| **RNF2** | La comunicación con el backend se basará en APIs REST, utilizando el protocolo de comunicaciones HTTP. Los datos se intercambiarán en formato JSON. |
| **RNF3** | Los datos generados por la aplicación serán reusables para garantizar la extensibilidad de la solución en el futuro. |
| **RNF4** | El almacenamiento de datos se realizará en un sistema gestor de bases de datos relacional. |
| **RNF5** | La aplicación se deberá de visualizar correctamente en dispositivos móviles y tabletas. |
| **RNF6** | La aplicación móvil se desarrollará siguiendo criterios de usabilidad y experiencia de usuario (UX) en el diseño y la construcción de las interfaces de usuario. |
| **RNF7** | Los usuarios interactuarán con la aplicación utilizando las capacidades estándar del dispositivo móvil (pantalla táctil y teclado para la introducción de datos). |
| **RNF8** | La aplicación móvil se testeará en las últimas versiones del sistema operativo iOS. |
| **RNF9** | La aplicación móvil desarrollada no dispondrá de publicidad en las interfaces de usuario para su monetización. |
| **RNF10** | La aplicación móvil deberá de garantizar tiempos de respuesta rápidos en las distintas interacciones del usuario. |
| **RNF11** | La aplicación móvil requerirá de la conexión a internet para su correcto funcionamiento, e informará al usuario en caso de que la conexión falle o tenga mala calidad. |
| **RNF12** | Se dispondrá de un manual de usuario que detalle el funcionamiento de la aplicación. |

*Tabla 2: Requisitos no funcionales de la aplicación*

## 1.4. Enfoque y método seguido

Una vez realizado el análisis del estado del arte y la toma de requisitos funcionales y no funcionales, se ha determinado que el mejor enfoque para dar respuesta a las necesidades detectadas es el desarrollo de una aplicación para dispositivos móviles, en concreto para la plataforma iOS.

Esta solución nos aportará las siguientes ventajas:

- La aplicación tomará como punto de partida para el diseño los requisitos y las necesidades detectadas, con el objetivo de darles la mejor respuesta posible.

- La aplicación será diseñada teniendo en cuenta los criterios de usabilidad y buenas prácticas de experiencia de usuario aprendidos a lo largo de este máster, con el objetivo de que la curva de aprendizaje sea mínima.
- Gracias al alto grado de penetración de los dispositivos móviles inteligentes o *smartphones* en nuestro país, el público objetivo que potencialmente podría acceder a la aplicación desarrollada es muy alto.
- El uso de una aplicación móvil ahorrará costes a nuestro público objetivo ya que no será necesario que dispongan de un ordenador ni de conexión a internet en el propio centro deportivo, tan solo de un smartphone, con lo que ahorrarán costes de operación y de uso del software desarrollado.

Se ha definido iOS como la plataforma de destino ante la constatación durante el análisis del estado del arte de la existencia de un abanico de soluciones a medida, web y para Android. La aplicación será compatible con las últimas versiones del sistema operativo y se deberá de visualizar correctamente en smartphones y tabletas.

Además, se desarrollará un backend para la autenticación y el almacenamiento de datos con el que se comunicará la aplicación, y que será agnóstico de la misma, de forma que se garantice la extensibilidad del sistema diseñado.
Este backend se desarrollará como una aplicación Java utilizando el framework Spring Boot [1], debido a la velocidad de desarrollo que ofrece para este tipo de aplicaciones. Este backend expondrá una serie de APIs REST para la comunicación con los clientes, de forma que sea totalmente independiente de los mismos. Los datos generados se almacenarán en una base de datos de tipo relacional para garantizar la integridad referencial y la seguridad de los mismos. La estructura de esta base de datos se generará automáticamente por la capa de persistencia de la aplicación gracias al uso de un mapeador objeto-relacional, de forma que no sea necesario la gestión de scripts SQL, simplificando el proceso de despliegue y puesta en marcha de la aplicación.

Tanto la base de datos como la aplicación backend en sí se desplegarán de forma conjunta en el proveedor de nube pública Heroku.

### 1.4.1. Metodología para el desarrollo del proyecto

El proyecto se desarrollará por fases, siguiendo una metodología de trabajo en espiral. Esta metodología de trabajo nos garantizará que se puedan introducir nuevos requerimientos y mejoras durante el desarrollo del proyecto. Aquellos a los que no se pueda dar cumplimiento, se añadirán al backlog del proyecto como mejoras futuras.
La metodología de trabajo en espiral se divide en las siguientes fases de trabajo:

[1] Mas información acerca de Spring Boot: (Arquitectura Java - Que es Spring Boot 2019), (Página principal de Spring Boot 2019)

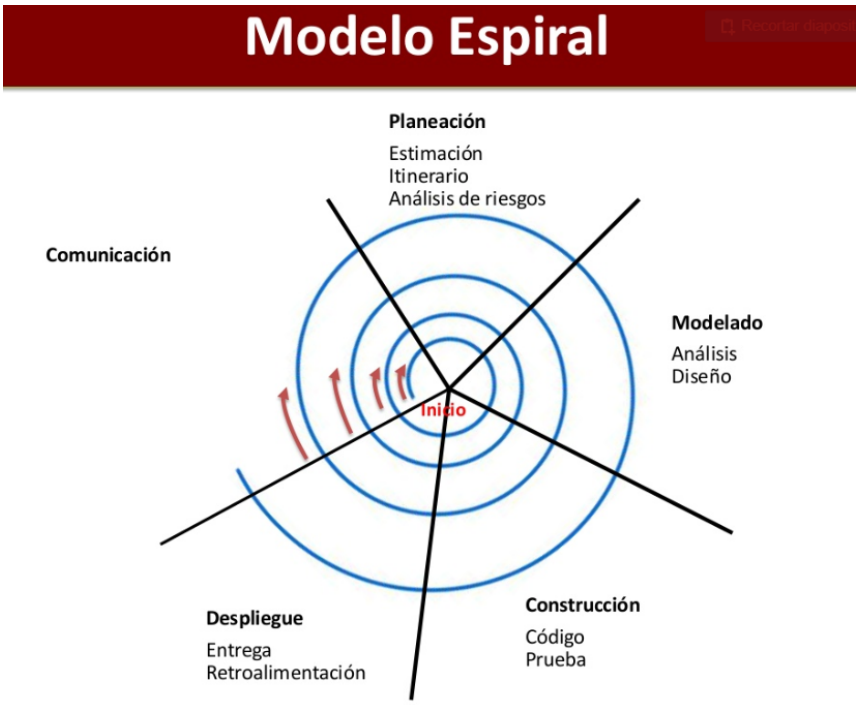



*Ilustración 5: Modelo de trabajo en espiral (Presentación "Modelos prescriptivos de proceso", disponible en Slide Share. s.f.)*

- Comunicación: toma de requisitos y decisiones iniciales del proyecto con el cliente.
- Análisis: Estimación, definición del plan de trabajo y análisis de riesgos.
- Modelado: Análisis y diseño de la solución.
- Construcción: Implementación y pruebas de la solución desarrollada.
- Despliegue: Entrega de la solución implementada y retroalimentación para la siguiente iteración.

Esta distribución en fases encaja a la perfección con las distintas entregas de este Trabajo Final de Máster y nos permitirá detectar mejoras y ampliaciones para su extensión durante todas las fases.

## 1.5. Planificación del Trabajo

Para la construcción de la aplicación, serán necesarios los siguientes recursos:

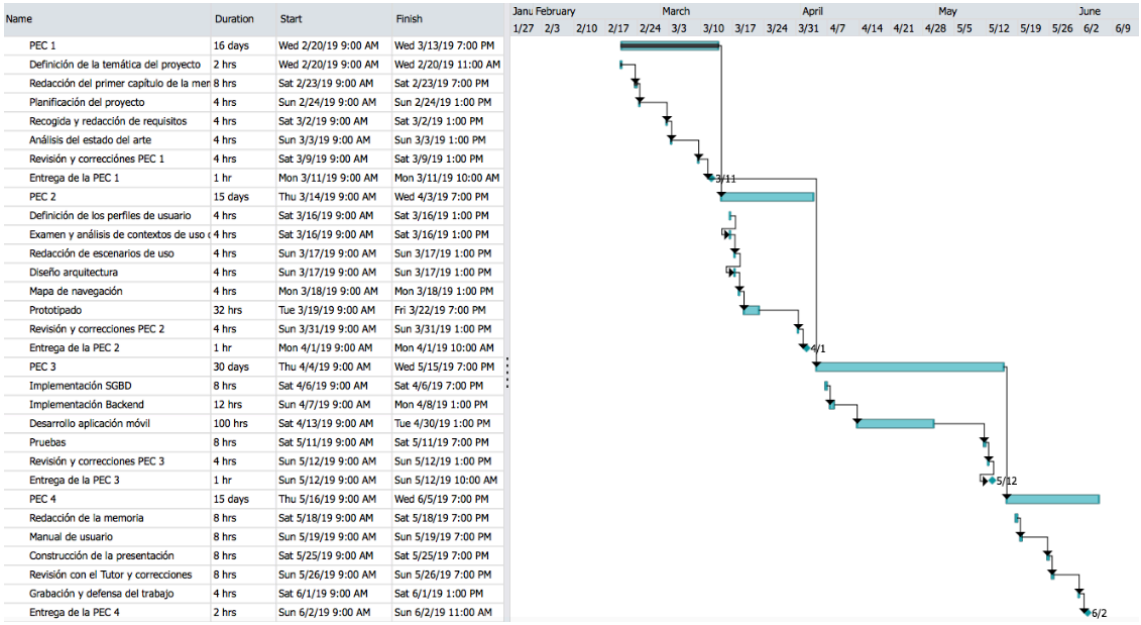
- Un backend que nos provea de una serie de funcionalidades avanzadas para la gestión de usuarios y de los datos del SGBD, desplegado en una nube pública.
- Un entorno de desarrollo Java, como por ejemplo Eclipse IDE, en caso de que sea necesario desarrollar a medida este backend.
- Una instancia de base de datos ejecutándose en una nube pública, tipo Google Cloud o Heroku, que nos provea de servicios de almacenamiento para los datos de los usuarios de la aplicación.
- Un ordenador cuyo sistema operativo sea Mac OS, debido a que el entorno de desarrollo para iOS (XCode) solo está disponible en esta plataforma.
- Opcionalmente, sería recomendable disponer de un dispositivo iOS (un smartphone o una tableta) para poder probar la aplicación sobre un dispositivo real.

A continuación, se presenta una tabla con el desglose de tareas a realizar para la consecución de este proyecto:

| Hito | Tareas para realizar (alto nivel) |
|---|---|
| Entrega de la PEC 1 (Conclusión fase análisis) | • Análisis y acuerdo con el tutor de la temática de la aplicación.<br>• Redacción del capítulo uno de la memoria, incluyendo planificación del trabajo.<br>• Recogida y redacción de requisitos funcionales y no funcionales.<br>• Análisis del estado del arte.<br>• Revisión con el tutor y correcciones de la PEC 1.<br>• Entrega de la PEC 1. |
| Entrega de la PEC 2 (Conclusión fase diseño) | • Definición de los perfiles de usuario.<br>• Examen y análisis de los contextos de uso de la aplicación.<br>• Redacción de escenarios de uso.<br>• Diseño y documentación de la arquitectura.<br>• Diseño y documentación del mapa de navegación de la aplicación.<br>• Construcción del prototipo de alto nivel.<br>• Revisión con el tutor y correcciones de la PEC 2.<br>• Entrega de la PEC 2. |
| Entrega de la PEC 3 (Conclusión fase implementación) | • Definición de modelo de datos, construcción y despliegue del SGBD en una nube pública.<br>• Desarrollo de la parte servidor de la aplicación y despliegue en nube pública.<br>• Desarrollo de la aplicación móvil.<br>• Pruebas de la aplicación móvil.<br>• Revisión con el tutor y correcciones PEC 3.<br>• Entrega de la PEC 3. |
| Entrega de la PEC 4 (Conclusión fase despliegue) | • Redacción final de la memoria.<br>• Redacción del manual de usuario de la aplicación.<br>• Construcción de la presentación del trabajo.<br>• Revisión con el tutor y correcciones de la PEC 4.<br>• Grabación y defensa del trabajo.<br>• Entrega final. |

*Tabla 3: Desglose de tareas por hito*

Una vez desglosadas las tareas, se presenta un diagrama de Gantt con la planificación temporal de las mismas para todo el proyecto:

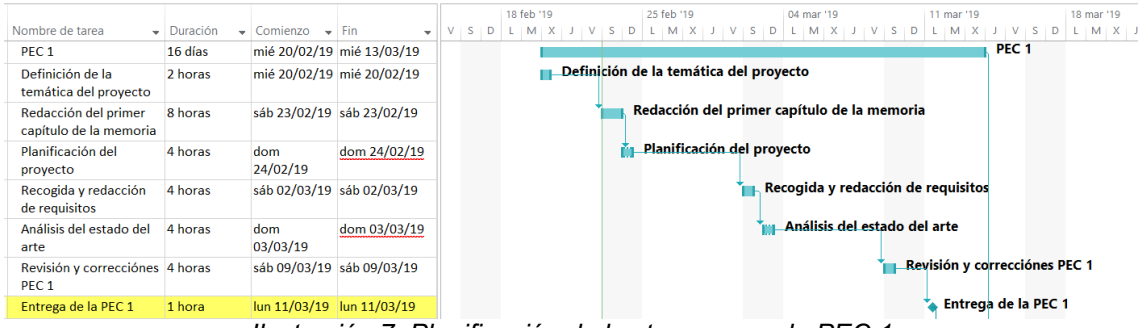


| Name | Duration | Start | Finish |
|---|---|---|---|
| PEC 1 | 16 days | Wed 2/20/19 9:00 AM | Wed 3/13/19 7:00 PM |
| Definición de la temática del proyecto | 2 hrs | Wed 2/20/19 9:00 AM | Wed 2/20/19 11:00 AM |
| Redacción del primer capítulo de la mer | 8 hrs | Sat 2/23/19 9:00 AM | Sat 2/23/19 7:00 PM |
| Planificación del proyecto | 4 hrs | Sun 2/24/19 9:00 AM | Sun 2/24/19 1:00 PM |
| Recogida y redacción de requisitos | 4 hrs | Sat 3/2/19 9:00 AM | Sat 3/2/19 1:00 PM |
| Análisis del estado del arte | 4 hrs | Sun 3/3/19 9:00 AM | Sun 3/3/19 1:00 PM |
| Revisión y correcciónes PEC 1 | 4 hrs | Sat 3/9/19 9:00 AM | Sat 3/9/19 1:00 PM |
| Entrega de la PEC 1 | 1 hr | Mon 3/11/19 9:00 AM | Mon 3/11/19 10:00 AM |
| PEC 2 | 15 days | Thu 3/14/19 9:00 AM | Wed 4/3/19 7:00 PM |
| Definición de los perfiles de usuario | 4 hrs | Sat 3/16/19 9:00 AM | Sat 3/16/19 1:00 PM |
| Examen y análisis de contextos de uso | 4 hrs | Sat 3/16/19 9:00 AM | Sat 3/16/19 1:00 PM |
| Redacción de escenarios de uso | 4 hrs | Sun 3/17/19 9:00 AM | Sun 3/17/19 1:00 PM |
| Diseño arquitectura | 4 hrs | Sun 3/17/19 9:00 AM | Sun 3/17/19 1:00 PM |
| Mapa de navegación | 4 hrs | Mon 3/18/19 9:00 AM | Mon 3/18/19 1:00 PM |
| Prototipado | 32 hrs | Tue 3/19/19 9:00 AM | Fri 3/22/19 7:00 PM |
| Revisión y correcciones PEC 2 | 4 hrs | Sun 3/31/19 9:00 AM | Sun 3/31/19 1:00 PM |
| Entrega de la PEC 2 | 1 hr | Mon 4/1/19 9:00 AM | Mon 4/1/19 10:00 AM |
| PEC 3 | 30 days | Thu 4/4/19 9:00 AM | Wed 5/15/19 7:00 PM |
| Implementación SGBD | 8 hrs | Sat 4/6/19 9:00 AM | Sat 4/6/19 7:00 PM |
| Implementación Backend | 12 hrs | Sun 4/7/19 9:00 AM | Mon 4/8/19 1:00 PM |
| Desarrollo aplicación móvil | 100 hrs | Sat 4/13/19 9:00 AM | Tue 4/30/19 1:00 PM |
| Pruebas | 8 hrs | Sat 5/11/19 9:00 AM | Sat 5/11/19 7:00 PM |
| Revisión y correcciones PEC 3 | 4 hrs | Sun 5/12/19 9:00 AM | Sun 5/12/19 1:00 PM |
| Entrega de la PEC 3 | 1 hr | Sun 5/12/19 9:00 AM | Sun 5/12/19 10:00 AM |
| PEC 4 | 15 days | Thu 5/16/19 9:00 AM | Wed 6/5/19 7:00 PM |
| Redacción de la memoria | 8 hrs | Sat 5/18/19 9:00 AM | Sat 5/18/19 7:00 PM |
| Manual de usuario | 8 hrs | Sun 5/19/19 9:00 AM | Sun 5/19/19 7:00 PM |
| Construcción de la presentación | 8 hrs | Sat 5/25/19 9:00 AM | Sat 5/25/19 7:00 PM |
| Revisión con el Tutor y correcciones | 8 hrs | Sun 5/26/19 9:00 AM | Sun 5/26/19 7:00 PM |
| Grabación y defensa del trabajo | 4 hrs | Sat 6/1/19 9:00 AM | Sat 6/1/19 1:00 PM |
| Entrega de la PEC 4 | 2 hrs | Sun 6/2/19 9:00 AM | Sun 6/2/19 11:00 AM |



*Ilustración 6: Planificación de las tareas para el Trabajo Final de Máster*

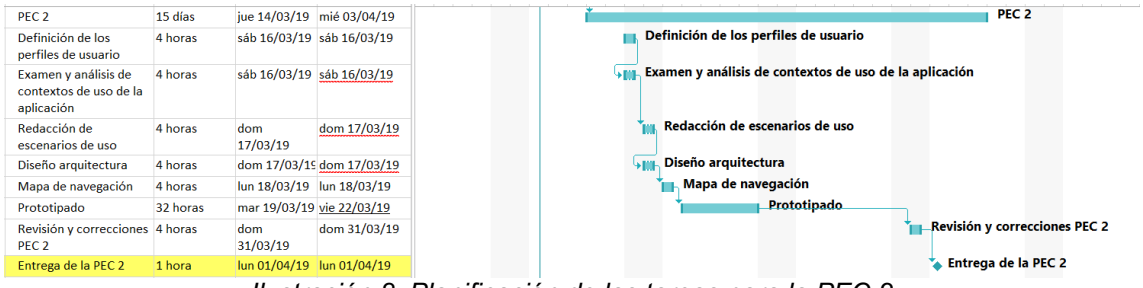
Y a continuación desglosado por cada PEC necesaria para la consecución del mismo:



| Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|
| PEC 1 | 16 días | mié 20/02/19 | mié 13/03/19 |
| Definición de la temática del proyecto | 2 horas | mié 20/02/19 | mié 20/02/19 |
| Redacción del primer capítulo de la memoria | 8 horas | sáb 23/02/19 | sáb 23/02/19 |
| Planificación del proyecto | 4 horas | dom 24/02/19 | dom 24/02/19 |
| Recogida y redacción de requisitos | 4 horas | sáb 02/03/19 | sáb 02/03/19 |
| Análisis del estado del arte | 4 horas | dom 03/03/19 | dom 03/03/19 |
| Revisión y correcciónes PEC 1 | 4 horas | sáb 09/03/19 | sáb 09/03/19 |
| Entrega de la PEC 1 | 1 hora | lun 11/03/19 | lun 11/03/19 |



*Ilustración 7: Planificación de las tareas para la PEC 1*



| | | | |
|---|---|---|---|
| PEC 2 | 15 días | jue 14/03/19 | mié 03/04/19 |
| Definición de los perfiles de usuario | 4 horas | sáb 16/03/19 | sáb 16/03/19 |
| Examen y análisis de contextos de uso de la aplicación | 4 horas | sáb 16/03/19 | sáb 16/03/19 |
| Redacción de escenarios de uso | 4 horas | dom 17/03/19 | dom 17/03/19 |
| Diseño arquitectura | 4 horas | dom 17/03/19 | dom 17/03/19 |
| Mapa de navegación | 4 horas | lun 18/03/19 | lun 18/03/19 |
| Prototipado | 32 horas | mar 19/03/19 | vie 22/03/19 |
| Revisión y correcciones PEC 2 | 4 horas | dom 31/03/19 | dom 31/03/19 |
| Entrega de la PEC 2 | 1 hora | lun 01/04/19 | lun 01/04/19 |



*Ilustración 8: Planificación de las tareas para la PEC 2*

| | | | |
|---|---|---|---|
| PEC 3 | 30 días | jue 04/04/19 | mié 15/05/19 |
| Implementación SGBD | 8 horas | sáb 06/04/19 | sáb 06/04/19 |
| Implementación Backend | 12 horas | dom 07/04/19 | lun 08/04/19 |
| Desarrollo aplicación móvil | 100 horas | sáb 13/04/19 | mar 30/04/19 |
| Pruebas | 8 horas | sáb 11/05/19 | sáb 11/05/19 |
| Revisión y correcciones PEC 3 | 4 horas | dom 12/05/19 | dom 12/05/19 |
| Entrega de la PEC 3 | 1 hora | dom 12/05/19 | dom 12/05/19 |



*Ilustración 9: Planificación de las tareas para la PEC 3*

| | | | |
|---|---|---|---|
| PEC 4 | 15 días | jue 16/05/19 | mié 05/06/19 |
| Redacción de la memoria | 8 horas | sáb 18/05/19 | sáb 18/05/19 |
| Manual de usuario | 8 horas | dom 19/05/19 | dom 19/05/19 |
| Construcción de la presentación | 8 horas | sáb 25/05/19 | sáb 25/05/19 |
| Revisión con el Tutor y correcciones | 8 horas | dom 26/05/19 | dom 26/05/19 |
| Grabación y defensa del trabajo | 4 horas | sáb 01/06/19 | sáb 01/06/19 |
| Entrega de la PEC 4 | 2 horas | dom 02/06/19 | dom 02/06/19 |

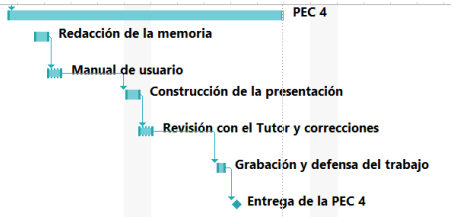



*Ilustración 10: Planificación de las tareas para la PEC 4*

Para la construcción de la planificación se ha tenido en cuenta la siguiente dedicación al Trabajo Final de Máster:

- Semana laborable: dedicación de entre dos y cuatro horas totales.
- Fines de semana y festivos: dedicación de hasta ocho horas diarias.

## 1.6. Breve sumario de productos obtenidos

Una vez concluido este Trabajo Final de Máster, obtendremos los siguientes productos:

- Una aplicación móvil para la plataforma iOS que de respuesta los requisitos y necesidades expuestos en esta memoria.
- La memoria del Trabajo Final de Máster, en el que se incluirá como anexo un manual de usuario de la aplicación y un análisis de las posibles extensiones de este trabajo.

## 1.7. Breve descripción de los otros capítulos de la memoria

Los siguientes capítulos de la memoria abordarán una serie de aspectos importantes en el diseño y el desarrollo de la solución, así como un apartado final con las conclusiones y las lecciones aprendidas.

En el siguiente apartado de este Trabajo Final de Máster abordaremos el diseño de la aplicación a desarrollar: desde el análisis de los usuarios de la aplicación y los casos de uso, pasando por la arquitectura y el prototipo de las interfaces de usuario de esta. Ese proceso, que se desarrollará bajo el hilo conductor del Diseño Centrado en el Usuario (DCU) será iterativo y el resultado final sentará las bases para iniciar el desarrollo de la aplicación.
En el capítulo 3 de esta memoria abordaremos el proceso de desarrollo y pruebas de la aplicación teniendo en cuenta la experiencia de esa PEC 3. Y, por último, en el capítulo 4 abordaremos las conclusiones y ampliaciones propuestas para este proyecto.

# Capítulo 2: **Diseño de la aplicación**

# Capítulo 2. Diseño de la aplicación

En los siguientes apartados de la memoria de este Trabajo Final de Máster presentaremos el diseño de la aplicación y su arquitectura utilizando el Diseño Centrado en el Usuario como hilo conductor de los distintos apartados.
Para empezar, se realizará un análisis de los usuarios objetivo de la aplicación, seguido por los escenarios de uso, la presentación del primer prototipo de la aplicación y la evaluación de este mismo. Una vez analizados todos estos aspectos, finalizaremos este capítulo con la definición de los casos de uso y la presentación de la arquitectura de la aplicación a desarrollar.

## 2.1. Análisis de los usuarios

La aplicación que proponemos se diseñará para satisfacer las necesidades de un público muy concreto, tal y como se comentaba en la introducción de esta memoria. Nuestros usuarios serán empresarios o gestores de centros deportivos (polideportivos, gimnasios, centros de ocio, etc.) de un tamaño pequeño, quizá con presencia tan solo localizada en un barrio de cualquier ciudad, cuya gestión se ha realizado de forma manual durante muchos años.

Nuestro objetivo es ayudarles en el día a día, y para ello asumiremos que:

- Los usuarios de nuestra aplicación actualmente gestionan la actividad de los centros con papel o utilizando alguna herramienta ofimática.
- La mayoría de ellos están familiarizados con el uso de smartphones o tabletas, aunque no para el uso profesional de los mismos. Sin embargo, al utilizarlos habitualmente con otras aplicaciones podemos asumir que estarán familiarizados con los métodos de interacción habituales de estos dispositivos.
- Los usuarios serán reticentes al cambio, puesto que llevan años utilizando el mismo sistema de gestión y estarán acostumbrados.
- Nuestros usuarios objetivo serán de mediana edad. Nuestra aplicación no tendrá una gran base de público joven, mas familiarizado quizá con otro tipo de soluciones en la nube.

Una vez que hemos descrito las características de nuestro público objetivo, pasaremos a detallar las necesidades que intentaremos solucionar:

- La aplicación permitirá gestionar de una forma muy sencilla las fichas de los abonados al centro deportivo. Para ello, se almacenará la información relativa al cliente en sí, sus datos de contacto y, si fuera necesario, una fotografía de este para identificarle más fácilmente.
- Sobre la base de abonados gestionaremos las cuotas de cada uno de ellos. Sin embargo, se deberá de manejar de una forma sencilla el hecho de que no todos los usuarios tienen la misma cuota, puesto que pueden existir ofertas y/o campañas promocionales, y que no todos los usuarios tienen la misma periodicidad en el pago de sus cuotas (no tienen porque pagar al mes, pueden existir otro tipo de periodicidades en los recibos).
- La información de los clientes, las cuotas y los pagos se deberá de explotar de una forma sencilla en beneficio del gestor del centro, en el sentido que este deberá de poder identificar que usuarios están al

corriente de pago y cuales no, así como los próximos vencimientos de las cuotas.
- En caso de que el centro deportivo disponga de instalaciones o equipamientos cuyo uso sea bajo reserva, se deberá de gestionar las mismas almacenando y explotando la información de la disponibilidad y el uso por parte de los abonados.

## 2.2. Escenario y contexto de uso de la aplicación

En el apartado anterior abordábamos las características y las necesidades de los usuarios de la aplicación, sin entrar a detallar en que contexto harán uso de ella. A continuación, detallaremos los escenarios de uso de la aplicación teniendo en cuenta las necesidades descritas en el apartado anterior.

**Escenario de uso nº1: El usuario utiliza la aplicación en su puesto de trabajo**

| | |
|---|---|
| **Contexto de uso:** | El usuario utiliza la aplicación en su puesto de trabajo para dar de alta un nuevo abonado. |
| **Características del dispositivo:** | Smartphone con buena conexión a internet. |
| **Descripción del escenario:** | El usuario ya utiliza la aplicación para la gestión del centro y está intentando dar de alta un nuevo abonado. Para ello, deberá de recoger una serie de datos (algunos obligatorios) y almacenar la ficha mientras el cliente está esperando. |
| **Necesidades detectadas:** | A la hora de recoger datos, la aplicación deberá de ser lo más ágil posible. Para ello, solo serán requeridos los campos mínimos en los formularios y se permitirán ediciones posteriores para ampliar la información recogida. |

**Escenario de uso nº2: El usuario utiliza la aplicación en su domicilio en un segundo dispositivo**

| | |
|---|---|
| **Contexto de uso:** | El usuario utiliza la aplicación en su domicilio para ver el listado de pagos pendientes, en un dispositivo distinto al que usa en su puesto de trabajo. |
| **Características del dispositivo:** | Tableta con buena conexión a internet. |
| **Descripción del escenario:** | El usuario está en su domicilio y quiere visualizar el listado de próximos vencimientos. Para ello, dispone de una tableta con conexión a internet. Sin embargo, este no es el dispositivo con el que utiliza habitualmente la aplicación para trabajar. |
| **Necesidades detectadas:** | La información que se generé por la aplicación se deberá de poder consumir en cualquier dispositivo que cumpla una serie de requisitos:<br>• Tener conexión a internet. |

| | |
|---|---|
| | • Disponer de la aplicación instalada.<br>• Que el usuario se autentique correctamente en la aplicación. |

**Escenario de uso nº3: El usuario sufre una interrupción durante el uso de la aplicación**

| | |
|---|---|
| **Contexto de uso:** | El usuario está utilizando la aplicación en el centro deportivo, durante su trabajo. |
| **Características del dispositivo:** | Smartphone con conexión a internet. |
| **Descripción del escenario:** | El usuario está utilizando la aplicación en el centro deportivo durante su trabajo cuando sufre una interrupción y tiene que bloquear el dispositivo. |
| **Necesidades detectadas:** | El usuario deberá de poder retomar la interacción con la aplicación en cualquier momento y en el mismo punto en el que lo dejó cuando abandonó el dispositivo. Por tanto, no se dispondrá de un mecanismo de caducidad de las sesiones de los usuarios para no interrumpirles su trabajo, siempre y cuando no cierren la aplicación completamente. |

**Escenario de uso nº4: El usuario se queda sin conexión a Internet durante el uso de la aplicación**

| | |
|---|---|
| **Contexto de uso:** | El usuario está utilizando la aplicación en el centro deportivo, durante su trabajo. |
| **Características del dispositivo:** | Smartphone con mala cobertura. |
| **Descripción del escenario:** | El usuario se encuentra realizando una interacción con la aplicación, por ejemplo, un nuevo alta de abonado. Cuando intenta almacenar los datos una vez completado el formulario, la aplicación le informa de que no dispone de conexión a internet y no puede completar el trabajo. |
| **Necesidades detectadas:** | Será necesario que, en caso de corte de conexión, la aplicación almacene localmente la información hasta que se pueda volver a enviar al servidor, para que los usuarios puedan seguir trabajando con la misma. |

## 2.3. Prototipo de la aplicación

A continuación, presentaremos uno a uno los prototipos de las distintas pantallas de la aplicación, empezando por la pantalla de login en la aplicación:

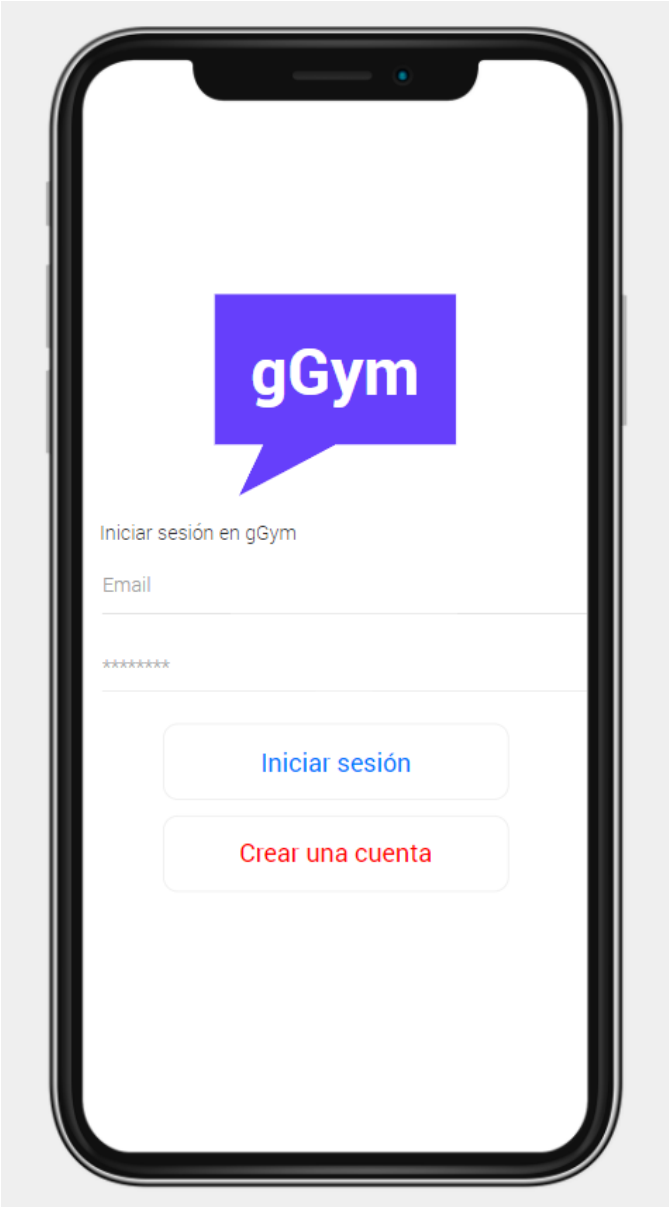



*Ilustración 11: Prototipo de la pantalla de login*

En la pantalla de login de la aplicación el usuario deberá de introducir el usuario y la contraseña de la cuenta que haya creado. En caso de que sea la primera vez que acceda a la aplicación, el usuario también dispondrá de la opción de crear una nueva cuenta.

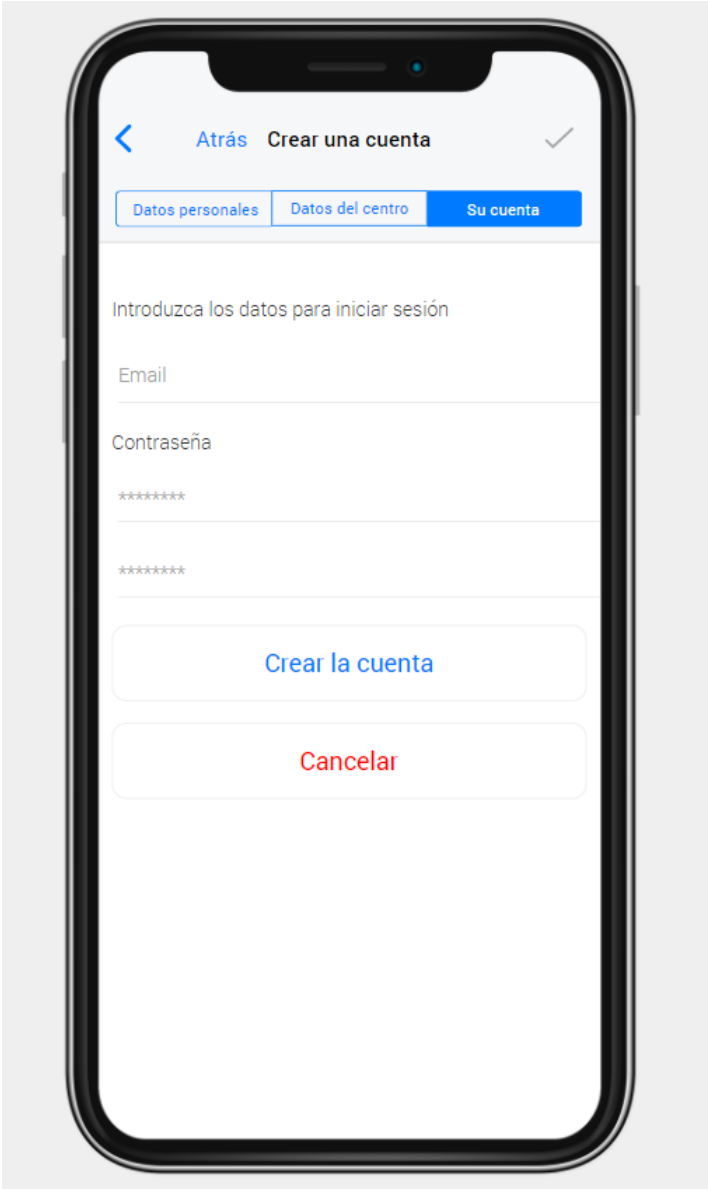



*Ilustración 12: Prototipo de la pantalla de registro*

La pantalla de registro de usuarios consta de un formulario con tres apartados diferenciados en los que el usuario deberá de introducir sus datos personales, los datos del centro deportivo y los datos de la cuenta con los que iniciará sesión

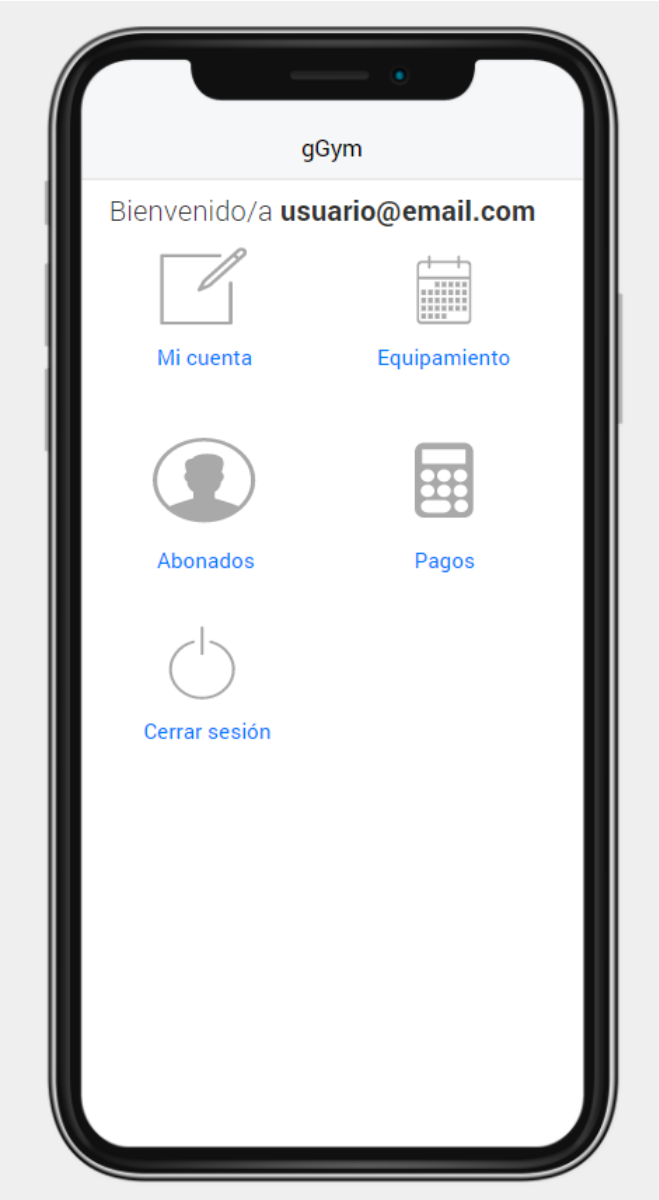



*Ilustración 13: Prototipo de la pantalla con el menú principal*

La pantalla de menú da acceso a las distintas opciones y funcionalidades de la aplicación, una vez que el usuario se haya logueado correctamente.

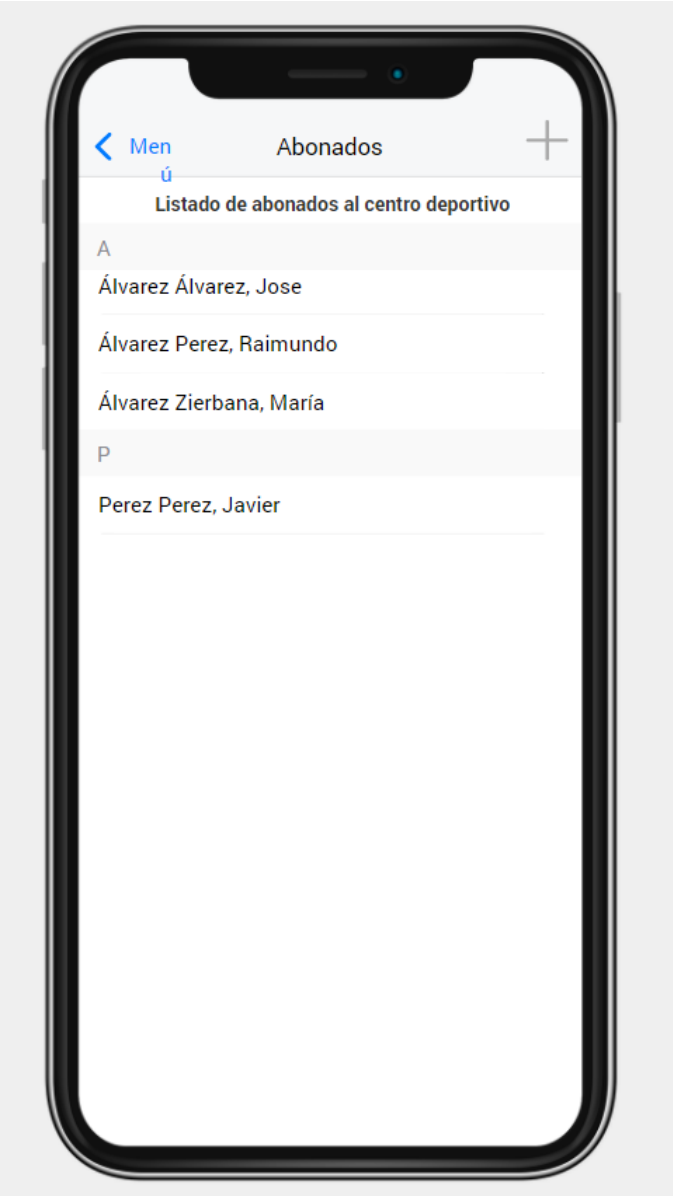



*Ilustración 14: Prototipo de la pantalla con el listado de abonados*

Al acceder a la sección de abonados desde el menú principal de la aplicación se presentará un listado ordenado alfabéticamente con todos los clientes del centro deportivo. Desde esta pantalla se podrá:

- Crear un nuevo abonado en el centro
- Acceder al detalle de un abonado

Ambas acciones ser realizarán en la siguiente pantalla de detalle:


*Ilustración 15: Prototipo de la pantalla de gestión de abonados*

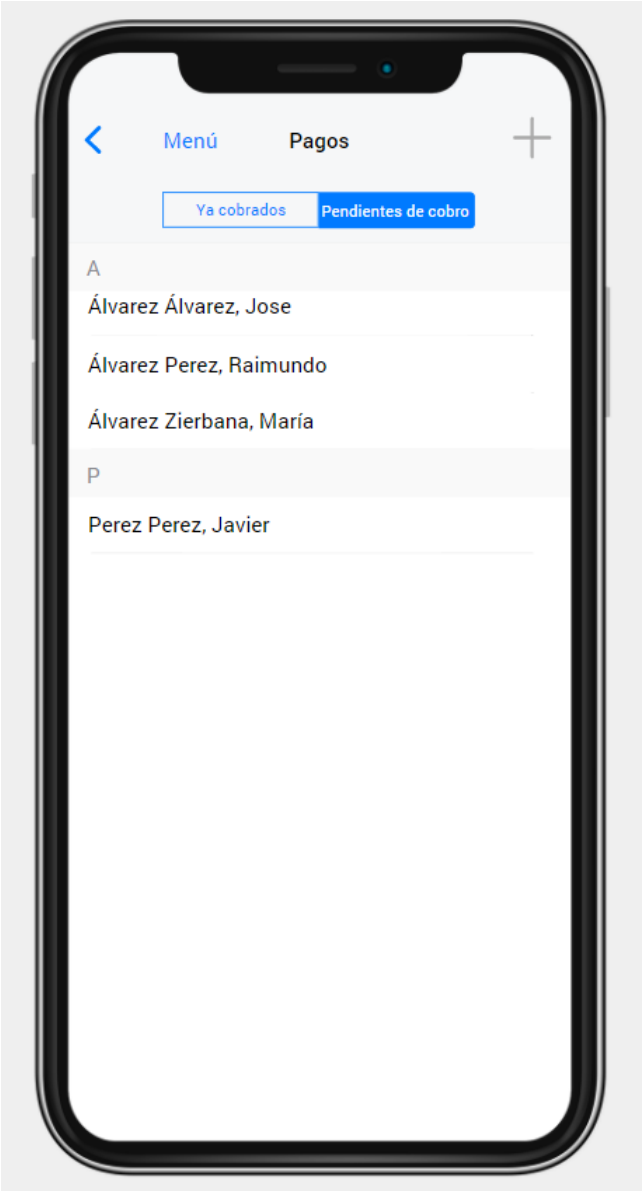



*Ilustración 16: Prototipo de la pantalla con el listado de pagos*

Al acceder a la sección de pagos desde el menú principal de la aplicación se presentará un listado ordenado alfabéticamente con todos los clientes del centro deportivo, separado en dos categorías: abonados que ya han pagado su cuota para el mes en curso, y abonados que todavía no han realizado el pago.

Desde esta pantalla se podrá además registrar un nuevo pago por parte de un abonado. Esta acción se realizará en la siguiente pantalla de detalle:

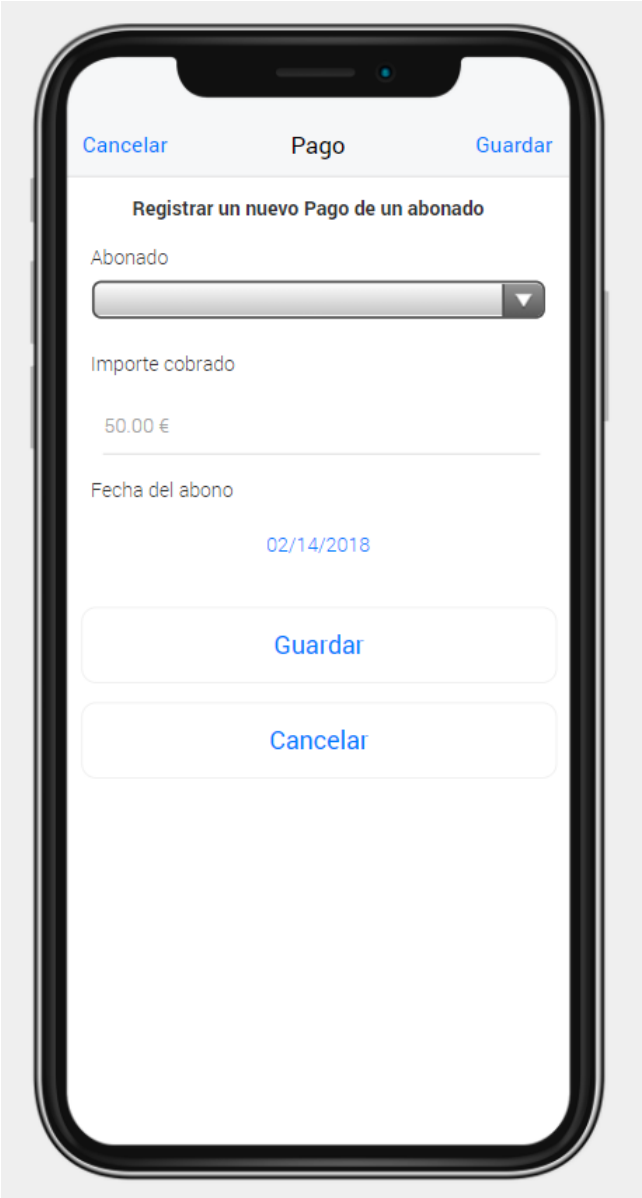



*Ilustración 17: Prototipo de la pantalla de registro de un pago*

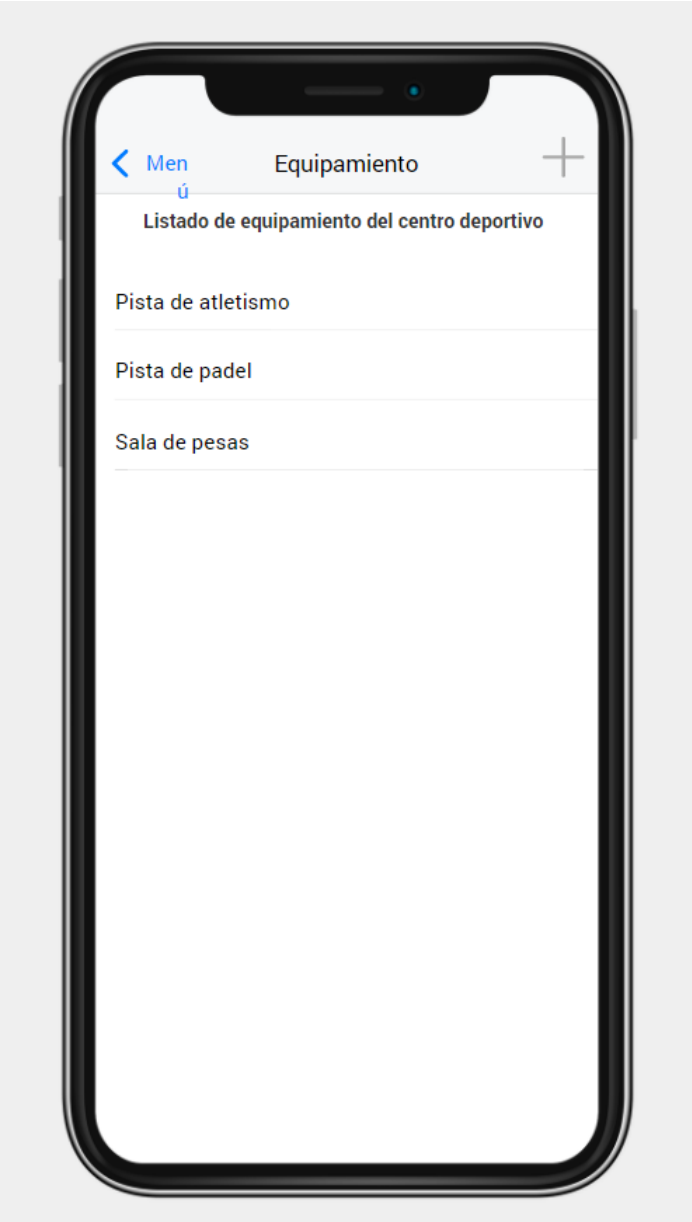



*Ilustración 18: Prototipo de la pantalla con el listado de equipamiento*

Al acceder a la sección de equipamiento desde el menú principal de la aplicación se presentará un listado ordenado alfabéticamente con los equipamientos registrados por parte del usuario. Desde esta pantalla se podrá:

- Registrar un nuevo equipamiento
- Modificar un equipamiento existente
- Reservar un equipamiento

Todas las acciones se realizarán en la siguiente pantalla de detalle:


*Ilustración 19: Prototipo de la pantalla con el detalle de un equipamiento*

## 2.4. Evaluación del prototipo

Para evaluar la calidad del prototipo construido y seguir con el proceso de refinado del diseño del mismo, se deberán de verificar los siguientes puntos sobre cada interfaz de usuario de la aplicación. A continuación, se presenta el listado de puntos a verificar:

### 2.4.1. Navegación por la aplicación:

- El usuario dispone de información contextual del sitio de la aplicación en el que se encuentra, la acción que va a tomar y la información que le es requerida para continuar.
- No existen mas de cinco niveles de profundidad en el mapa de navegación de la aplicación. Es decir, para cualquiera de las acciones a tomar en la

aplicación el usuario no podrá navegar por mas de cuatro pantallas hasta llegar al destino.
- El usuario dispone siempre de la opción volver atrás en la pantalla. Está opción se presenta además siempre en el mismo sitio.

### 2.4.2. Botones de acción:

- Los botones se llaman igual en todas las pantallas, no se utilizan nomenclaturas distintas para las mismas acciones y se presentan siempre en la misma ubicación en las interfaces de usuario.
- Los botones se presentan siempre en el mismo orden en las interfaces de usuario: la opción mas restrictiva es la última siempre.
- La gama de colores para los botones es homogénea: la opción mas restrictiva de las que puede tomar está coloreada en rojo y la opción que ejecuta la acción utiliza el color por defecto del sistema operativo.
- Al tomar la opción mas restrictiva se utiliza un diálogo modal de confirmación para evitar que el usuario se confunda. Este diálogo dispone de un mensaje de confirmación y de un botón para cancelar la acción.

### 2.4.3. Gama de colores:

- No se utilizan mas de mas de cuatro colores distintos en la misma interfaz de usuario.
- Se deberá de primar en todo momento la gama de colores predeterminada de los compontes visuales del sistema operativo, así como los estilos para los diálogos y los botones.
- Se respetan las combinaciones de colores que los usuarios con problemas de visión pueden apreciar sin dificultades.

### 2.4.4. Interfaz de usuario:

- Se sigue en todo momento el principio de “una pantalla una acción a realizar”.
- Las pantallas con listados de información son homogéneas y utilizan la misma estructura para la presentación de la información.
- Las pantallas de visualización de detalles permitirán la edición de los mismos.
- Se utilizan en todo momento los componentes por defecto del sistema operativo para la introducción de datos, de forma que se facilite al usuario la comprensión de la interfaz.
- En caso de que la pantalla necesite mostrar muchos campos de información se primará el uso de pestañas para agrupar información relacionada sobre el uso de scroll vertical.
- Se aprovechará el ancho de la pantalla para los campos, sin que existan márgenes laterales con respecto a los bordes de esta.

## 2.5. Definición de los casos de uso de la aplicación

Mediante el diagrama de casos de uso mostraremos de una forma gráfica todas las funcionalidades de la aplicación y como un usuario interacciona con ellas, antes de entrar en el detalle del listado de casos de uso en sí.

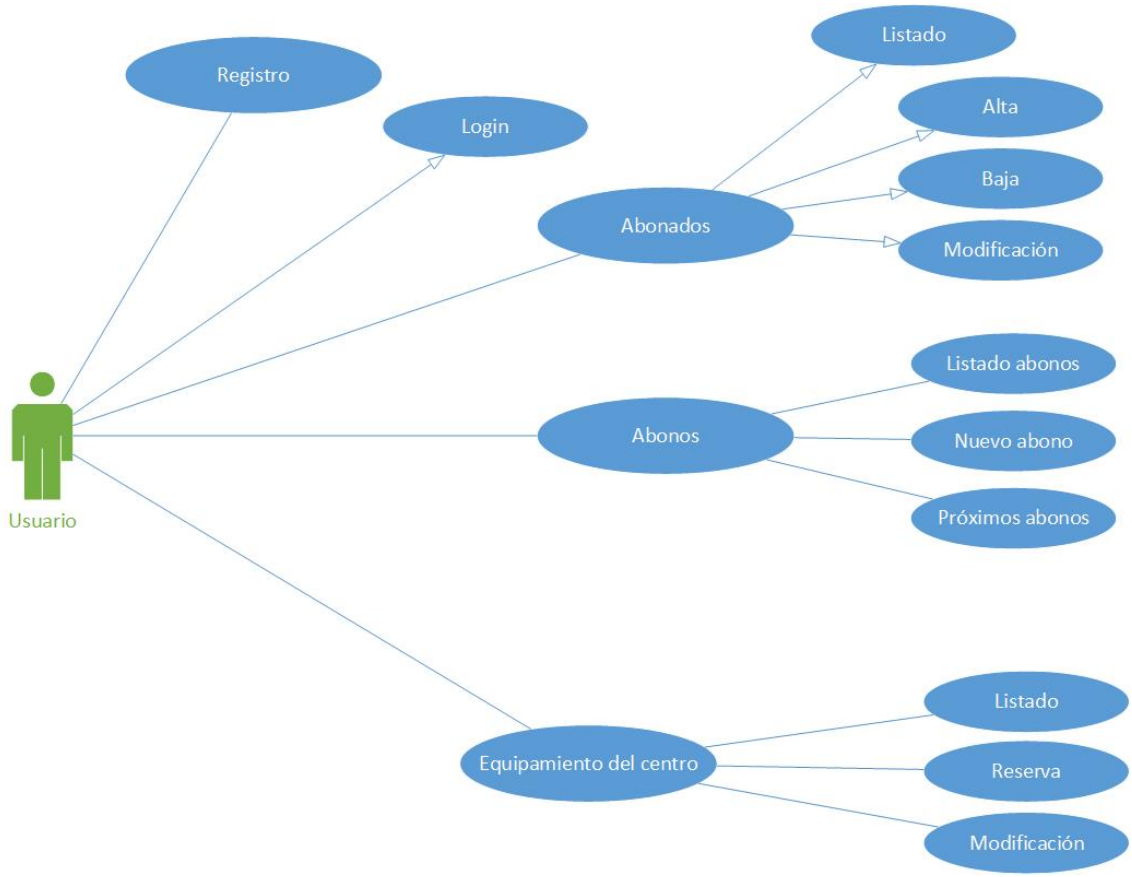



*Ilustración 20: Diagrama de casos de uso de la aplicación*

A continuación se presenta el listado de casos de uso de la aplicación:

| **Caso de Uso nº 1: Registro de nuevo usuario** | |
|---|---|
| **Descripción** | El usuario descarga la aplicación y quiere crear una cuenta |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha descargado la aplicación |
| **Postcondiciones** | El usuario dispone de una cuenta para acceder a la aplicación en futuras ocasiones |
| **Flujo Principal** | 1. El usuario accede a la aplicación y visualiza la pantalla de login. Pulsa sobre la opción “Crear una cuenta”.<br>2. El usuario accede a un formulario en el que introduce sus datos personales, los datos del centro deportivo, su email y la contraseña por duplicado.<br>3. El usuario pulsa el botón registrar y accede al menú principal de la aplicación. |

| | |
|---|---|
| **Flujo alternativo** | N/A |

| **Caso de Uso nº 2: Login de usuario** | |
|---|---|
| **Descripción** | El usuario quiere acceder a la aplicación |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha registrado correctamente y dispone de una cuenta activa |
| **Postcondiciones** | El usuario accede a la aplicación |
| **Flujo Principal** | 1. El usuario accede a la aplicación y visualiza la pantalla de login. Introduce su email y su contraseña y pulsa sobre la opción “Iniciar sesión”.<br>2. El usuario visualiza el menú principal de la aplicación, en caso de que sus datos de acceso sean correctos. |
| **Flujo alternativo** | En caso de que el usuario introduzca mal sus datos de acceso, se mostrará un mensaje de error y será necesario que repita el paso 1. |

| **Caso de Uso nº 3: Consulta del listado de abonados** | |
|---|---|
| **Descripción** | El usuario quiere visualizar el listado de abonados al centro deportivo |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | El usuario visualiza el listado de abonados, ordenado alfabéticamente |
| **Flujo Principal** | 1. El usuario accede desde el menú principal a la sección de Abonados.<br>2. El usuario visualiza el listado de abonados. |
| **Flujo alternativo** | N/A |

| **Caso de Uso nº 4: Alta de nuevo abonado** | |
|---|---|
| **Descripción** | El usuario quiere registrar un nuevo abonado en el centro deportivo |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | Se ha registrado un nuevo abonado en la aplicación. |

| | |
|---|---|
| **Flujo Principal** | 1. El usuario accede desde el menú principal a la sección de Abonados.<br>2. El usuario visualiza el listado de abonados y pulsa sobre la opción “Nuevo”.<br>3. El usuario accede a un formulario en el que introducirá los datos del abonado y pulsará el botón “Guardar”.<br>4. El usuario vuelve al listado de abonados. |
| **Flujo alternativo** | Si el usuario pulsa el botón “Cancelar” en el paso 3, volverá al listado de abonados y no se registrarán los datos que haya introducido en la aplicación. |

| **Caso de Uso nº 5: Baja de abonado** | |
|---|---|
| **Descripción** | El usuario quiere dar de baja un abonado en la aplicación. |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | Se eliminan los registros asociados al abonado en la aplicación. |
| **Flujo Principal** | 1. El usuario accede desde el menú principal a la sección de Abonados.<br>2. El usuario visualiza el listado de abonados y pulsa sobre uno de ellos.<br>3. El usuario accede a los detalles del abonado y pulsa sobre el botón “Dar de baja”<br>4. El usuario vuelve al listado de abonados. |
| **Flujo alternativo** | Si el usuario pulsa el botón “Cancelar” en el paso 3, volverá al listado de abonados y no se realizarán modificaciones de los datos almacenados en la aplicación. |

| **Caso de Uso nº 6: Modificar los datos de un abonado** | |
|---|---|
| **Descripción** | El usuario quiere modificar los datos de un abonado en la aplicación. |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | Se almacenan los datos modificados en la aplicación. |
| **Flujo Principal** | 1. El usuario accede desde el menú principal a la sección de Abonados.<br>2. El usuario visualiza el listado de abonados y pulsa sobre uno de ellos.<br>3. El usuario accede a los detalles del abonado, realiza las modificaciones que estime y pulsa sobre el botón “Guardar”. |

| | |
|---|---|
| | 4. El usuario vuelve al listado de abonados. |
| **Flujo alternativo** | Si el usuario pulsa el botón “Cancelar” en el paso 3, volverá al listado de abonados y no se realizarán modificaciones de los datos almacenados en la aplicación. |

| **Caso de Uso nº 7: Listado de abonos (pagos)** | |
|---|---|
| **Descripción** | El usuario quiere visualizar el listado de abonos del mes en curso. |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | El usuario accede al listado con la información de los abonados que ya han pagado el mes en curso. |
| **Flujo Principal** | 1. El usuario accede a la sección de abonos desde el menú principal de la aplicación.<br>2. El usuario visualiza el listado de abonados que ya han pagado el mes en curso. |
| **Flujo alternativo** | N/A |

| **Caso de Uso nº 8: Registrar un nuevo abono (pago)** | |
|---|---|
| **Descripción** | El usuario quiere registrar un nuevo pago en la aplicación. |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | El pago queda registrado correctamente en la aplicación. |
| **Flujo Principal** | 1. El usuario accede a la sección de abonos desde el menú principal de la aplicación.<br>2. El usuario visualiza el listado de abonados que ya han pagado el mes en curso y pulsa sobre el botón “Nuevo”.<br>3. El usuario introduce los datos del abonado y la fecha de abono y pulsa sobre el botón “Guardar”.<br>4. El usuario vuelve al listado de abonos. |
| **Flujo alternativo** | Si el usuario pulsa el botón “Cancelar” en el paso 3, volverá al listado de pagos y no se realizarán modificaciones de los datos almacenados en la aplicación. |

| **Caso de Uso nº 9: Listado de abonos pendientes** | |
|---|---|
| **Descripción** | El usuario quiere consultar que abonados no han pagado su cuota para el mes en curso. |

| | |
|---|---|
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | El usuario accede al listado. |
| **Flujo Principal** | 1. El usuario accede a la sección de abonos desde el menú principal de la aplicación.<br>2. El usuario visualiza el listado de abonados que ya han pagado el mes en curso y filtra la información por “impagados”. |
| **Flujo alternativo** | N/A. |

| Caso **de Uso nº 10: Equipamiento del centro** | |
|---|---|
| **Descripción** | El usuario quiere consultar el listado de equipamiento que tiene registrado en la aplicación. |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | El usuario accede al listado. |
| **Flujo Principal** | 1. El usuario accede al listado de equipamiento desde el menú principal de la aplicación.<br>2. El usuario visualiza el listado de equipamiento que tiene registrado en la aplicación. |
| **Flujo alternativo** | N/A |

| **Caso de Uso nº 11: Reserva de equipamiento** | |
|---|---|
| **Descripción** | El usuario quiere registrar una reserva de equipamiento en la aplicación. |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | La reserva de equipamiento queda registrada. |
| **Flujo Principal** | 1. El usuario accede al listado de equipamiento desde el menú principal de la aplicación.<br>2. El usuario visualiza el listado de equipamiento que tiene registrado en la aplicación y accede a uno en concreto.<br>3. El usuario visualiza los datos del equipamiento, el listado de reservas y pulsa sobre el botón “Reservar”.<br>4. El usuario introduce los datos del abonado, las fechas de la reserva y pulsa sobre el botón “Guardar”. |

| | |
|---|---|
| | 5. El usuario vuelve al detalle del equipamiento. |
| **Flujo alternativo** | Si el usuario pulsa el botón “Cancelar” en el paso 4, volverá al detalle del equipamiento y no se almacenarán los cambios que haya hecho en la aplicación. |

| **Caso de Uso nº 12: Modificar un equipamiento** | |
|---|---|
| **Descripción** | El usuario quiere modificar algún dato del equipamiento que tiene registrado en la aplicación. |
| **Actores** | Usuario |
| **Precondiciones** | El usuario se ha autenticado correctamente en la aplicación |
| **Postcondiciones** | Los datos se almacenan en la aplicación. |
| **Flujo Principal** | 1. El usuario accede al listado de equipamiento desde el menú principal de la aplicación.<br>2. El usuario visualiza el listado de equipamiento que tiene registrado en la aplicación y accede a uno en concreto.<br>3. El usuario visualiza los datos del equipamiento, realiza las modificaciones que considere y pulsa sobre el botón “Guardar”<br>4. El usuario vuelve al listado de equipamiento. |
| **Flujo alternativo** | Si el usuario pulsa el botón “Cancelar” en el paso 3, volverá al listado de equipamiento y no se almacenarán los cambios que haya hecho en la aplicación. |

## 2.6. Arquitectura de la solución

Tal y como se ha mencionado en capítulos anteriores de esta memoria, la solución a desarrollar se basará en dos productos distintos:

- Una aplicación para dispositivos móviles iOS
- Un backend desplegado en la nube de Heroku

En los siguientes apartados se detallará el diseño de cada uno de estos productos para su posterior desarrollo.

### 2.6.1. Vista general de la solución

A alto nivel, la solución desarrollada tendrá el siguiente aspecto:

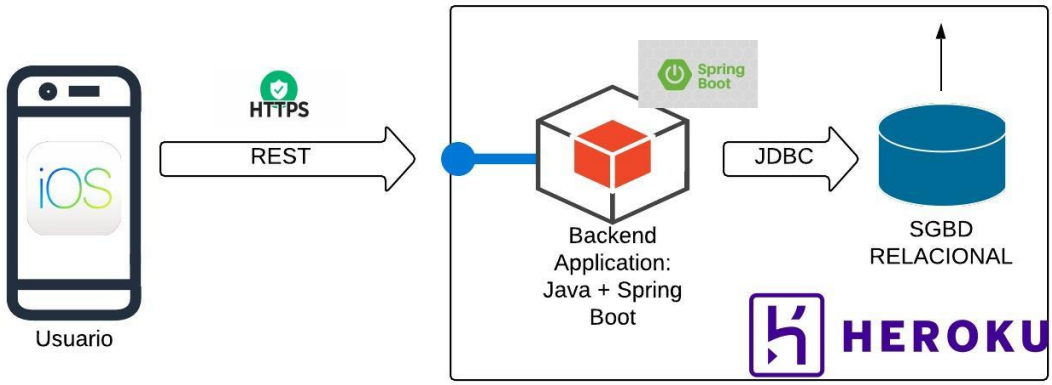



*Ilustración 21: Diagrama de alto nivel de la arquitectura*

En este diagrama general queremos destacar los siguientes elementos:

- Los usuarios accederán a través de sus dispositivos móviles gracias a la aplicación desarrollada, que deberán descargar e instalar.
- La aplicación móvil se comunicará a través de un canal seguro (protocolo https) con el backend. La aplicación consumirá las APIs REST expuestas por este mismo, intercambiando los datos en formato JSON.
- Todos los elementos del backend de la aplicación estará desplegados de forma conjunta en la nube de Heroku.
- La base de datos será de tipo relacional, y se utilizará el SGBD que provea esta nube pública. Al utilizar Heroku como proveedor, la base de datos será Postgres.
- La aplicación backend utilizará el lenguaje de programación Java y el framework Spring Boot.

### 2.6.2. Modelo de datos

Ambas aplicaciones (móvil y backend) utilizarán el siguiente modelo de datos:

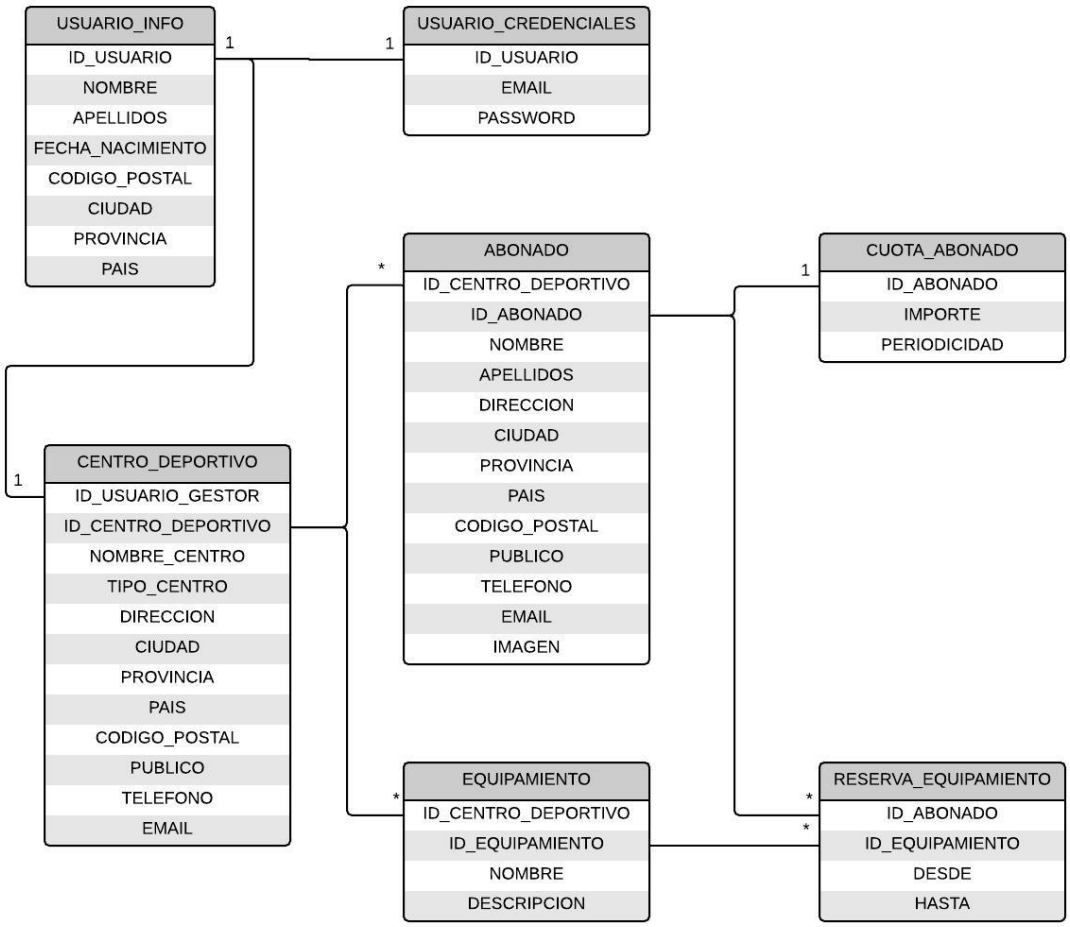



*Ilustración 22: Modelo de datos de la solución*

Se dispondrá de las siguientes entidades:

- USUARIO_CREDENCIALES: Tabla para almacenar los datos de acceso a la aplicación de los distintos usuarios (email y contraseña).
- USUARIO_INFO: Tabla para almacenar los datos de los usuarios cuando se registran en la aplicación.
- CENTRO_DEPORTIVO: Tabla para almacenar los datos del centro deportivo en sí.
- ABONADO: Tabla para almacenar los clientes abonados en los distintos centros deportivos.
- CUOTA_ABONADO: Tabla para almacenar la cuota de los distintos abonados a los centros deportivos.
- EQUIPAMIENTO: Tabla para almacenar la información del equipamiento reservable en los distintos centros deportivos.
- RESERVA_EQUIPAMIENTO: Tabla para almacenar las reservas de equipamiento de los abonados al centro deportivo.

### 2.6.3. Diagrama de clases para la aplicación backend

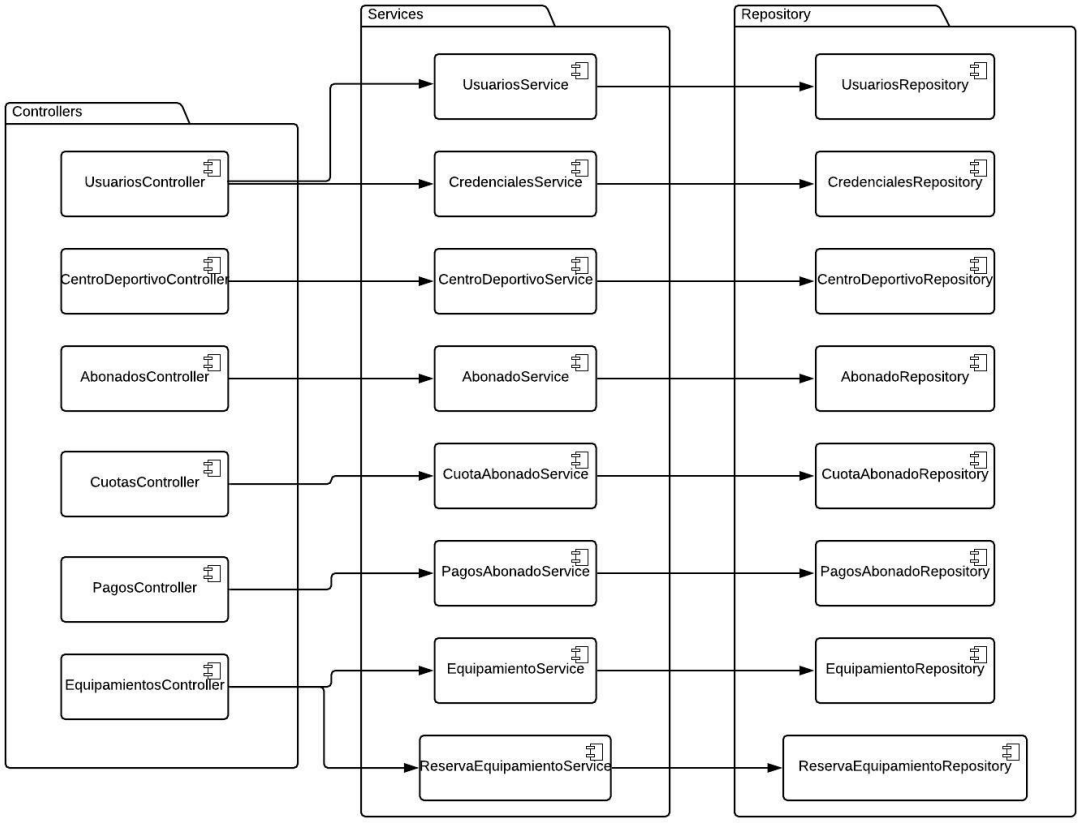



*Ilustración 23: Diagrama de clases del Backend*

La aplicación backend se desarrollará siguiendo el patrón de desarrollo tres capas: presentación, negocio y datos.
Las responsabilidades de cada capa serán las siguientes:

- Capa de controladores: expondrán al exterior los endpoints de los servicios REST de nuestras APIs, y se encargarán además de la validación de los datos de entrada y del formateo de los datos de salida.
- Capa de negocio (servicios): reciben los datos de entrada de la capa de presentación y pueden operar con la capa de datos para almacenar o recoger datos del SGBD. En esta capa se dispondrá de toda la lógica de negocio de la aplicación.
- Capa de datos (repositorios): contiene las clases encargadas de interactuar con la base de datos (o otros sistemas de persistencia).

### 2.6.4. Diagrama de clases para la aplicación móvil

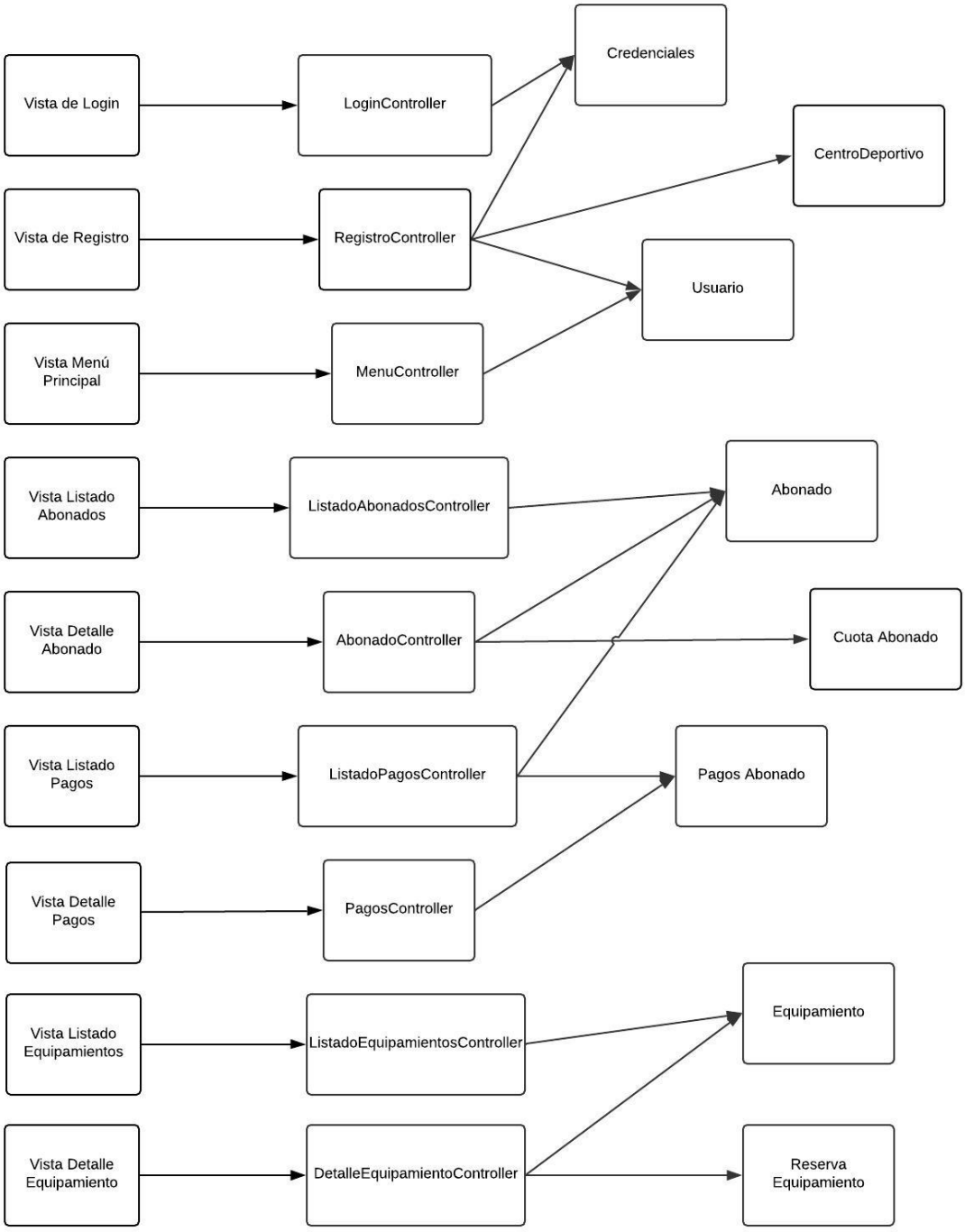



*Ilustración 24: Diagrama de clases de la aplicación móvil*

La aplicación móvil se desarrollará siguiendo el patrón Modelo Vista Controlador (MVC). Este patrón de arquitectura nos permitirá separar en los siguientes componentes la lógica de la aplicación:

- Las vistas constituyen las distintas interfaces de usuario definidas. En ellas, el usuario interactuará con la aplicación registrando datos y acciones.
- Los controladores son los intermediarios entre el modelo y las distintas vistas. En ellos gestionaremos el flujo de información entre ambos, la

lógica de presentación de las vistas y el tratamiento de las acciones de los usuarios.

- El modelo es la representación de la estructura de datos del sistema. En el definimos toda la lógica de negocio de la aplicación y la comunicación con el backend.

### 2.6.5. Estructura de las APIs del sistema

Para la correcta comunicación entre la aplicación móvil y la aplicación backend se definirán las siguientes APIs REST. En todos los casos los datos serán intercambiados en formato JSON.

| Endpoint | Método HTTP | Acción | Datos de entrada | Datos de salida |
|---|---|---|---|---|
| **/credenciales/login** | POST | Login | Usuario/password | N/A |
| **/credenciales/crear** | POST | Crear credenciales | Usuario/password | N/A |
| **/usuarios** | POST | Crear usuario | Datos del usuario | Usuario creado |
| **/centros/crear** | POST | Crear centro deportivo | Datos del centro deportivo | Centro creado |
| **/centros/{id}** | GET | Obtener los datos de un centro deportivo | ID Usuario, ID Centro deportivo | Detalles del centro deportivo |
| **/abonados** | GET | Obtener el listado de abonados | ID Centro Deportivo, ID Usuario | Listado de abonados |
| **/abonados/{id}** | GET | Obtener los detalles de un abonado | ID Abonado, ID Usuario | Detalles del abonado |
| **/abonados/{id}** | POST | Crear un abonado | ID Usuario, datos del abonado | Detalles del abonado |
| **/abonados/{id}** | PUT | Actualizar un abonado | ID Usuario, ID abonado | Detalles del abonado |
| **/cuotas/{id}** | GET | Obtener la cuota de un abonado | ID Usuario, ID abonado | Detalles de la cuota |
| **/cuotas/{id}** | POST | Crear una cuota para un abonado | ID Usuario, datos de la cuota | Detalles de la cuota |
| **/cuotas/{id}** | PUT | Actualizar la cuota de un abonado | ID Usuario, ID abonado | Detalles de la cuota |
| **/pagos/pendientes** | GET | Obtener el listado de pagos pendientes | ID Usuario, ID Centro | Listado de pagos |

| | | | | |
|---|---|---|---|---|
| | | | deportivo, Mes, Año | |
| **/pagos/cobrados** | GET | Obtener el listado de pagos recibidos | ID Usuario, ID Centro deportivo, Mes, Año | Listado de pagos |
| **/pagos/{id}** | GET | Obtener los detalles de un pago | ID Usuario, ID pago | Detalles del pago |
| **/pagos/{id}** | POST | Registra el pago de un usuario | ID Usuario, ID abonado | Detalles del pago |
| **/equipamiento** | GET | Obtiene el listado de equipamiento de un centro deportivo | ID Usuario, ID Centro deportivo | Listado de equipamiento del centro deportivo |
| **/equipamiento/{id}** | GET | Obtiene el detalle de un equipamiento | ID Usuario, ID equipamiento | Detalles de un equipamiento |
| **/equipamiento/{id}** | POST | Crea un nuevo equipamiento en el centro deportivo | ID Usuario, ID Centro deportivo | Detalles de un equipamiento |
| **/equipamiento/{id}** | PUT | Actualiza el equipamiento de un centro deportivo | ID Usuario, ID equipamiento | Detalles de un equipamiento |
| **/equipamiento/{id}/reservar** | POST | Reserva un equipamiento en el centro deportivo | ID Usuario, ID equipamiento | Detalles de un equipamiento |

*Tabla 4: Listado de APIs de la aplicación*

# Capítulo 3: **Desarrollo**

# Capítulo 3. Desarrollo de la aplicación

Una vez hemos abordado el diseño de la aplicación pasaremos a detallar como ha sido el proceso de desarrollo de la misma, además de las modificaciones de alcance que hemos tenido que realizar.

## 3.1. Desarrollo de la aplicación

### 3.1.1. Desarrollo del backend

Tal y como se ha comentado en el apartado de diseño, el backend de la aplicación se ha desarrollado en Java utilizando el framework Spring Boot.
A continuación presentamos una captura de la estructura de paquetes del proyecto, que sigue el diseño propuesto para el mismo:

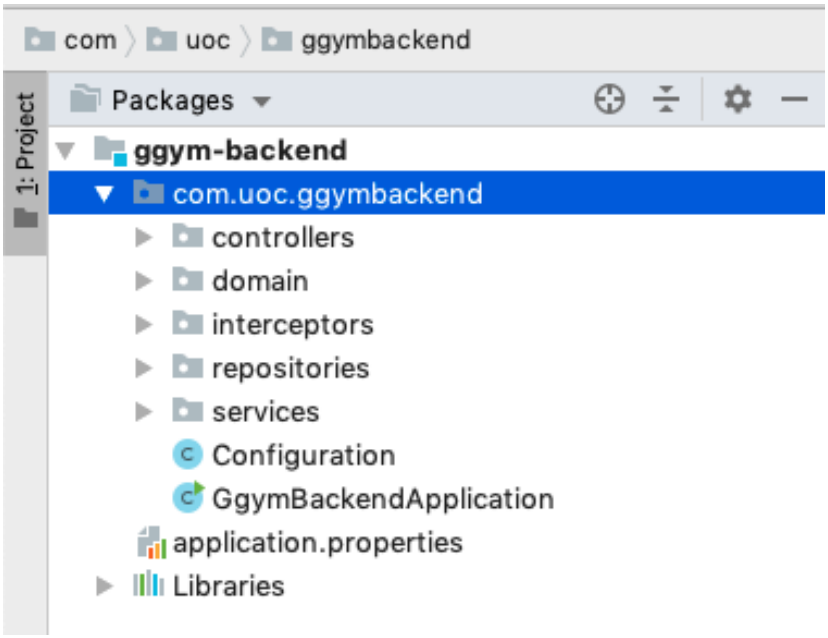



*Ilustración 25: Estructura de paquetes de la aplicación backend.*

La aplicación cuenta con una clase principal para lanzarla, así como una clase de configuración y un fichero de propiedades.
A continuación detallamos el contenido de cada paquete:

- **Controllers:** contiene las clases que exponen los distintos endpoints de las APIs REST. Suponen el punto de entrada y de salida de la aplicación: se exponen las distintas operaciones, se reciben los datos y se envía la respuesta a las peticiones. Los distintos controladores se comunican con la capa de servicios.
  Este sería el controlador que expone las operaciones relacionadas con los abonados:

```
@RestController
public class AbonadosController {

    @Autowired
    private AbonadosService abonadosService;
```

```
    @PostMapping(value = "/abonados/crear", produces = "application/json",
consumes = "application/json")
    public AbonadoVO crearAbonado(@RequestBody @Valid AbonadoVO abonadoVO) {
        return abonadosService.crearAbonado(abonadoVO);
    }

    @GetMapping(value = "/abonados/centro/{idCentroDeportivo}", produces =
"application/json")
    public List<AbonadoVO> obtenerListadoAbonados(@PathVariable Long
idCentroDeportivo) {
        return abonadosService.obtenerListadoAbonados(idCentroDeportivo);
    }

    @GetMapping(value = "/abonados/{id}", produces = "application/json")
    public AbonadoVO obtenerAbonado(@PathVariable Long id) {
        return abonadosService.obtenerAbonado(id);
    }

    @PutMapping(value = "/abonados/actualizar", produces = "application/json",
consumes = "application/json")
    public void actualizarAbonado(@RequestBody @Valid AbonadoVO abonadoVO) {
        abonadosService.actualizarAbonado(abonadoVO);
    }

    @DeleteMapping(value = "/abonados/{id}", produces = "application/json")
    public void borrarAbonado(@PathVariable Long id) {
        abonadosService.borrarAbonado(id);
    }

}
```

- **Domain:** Contiene el modelo de datos de la aplicación, tanto los objetos que viajan por la red al consumir la API (denominados VOs), como las entidades de base de datos.
  Esta sería la entidad que modela los datos de los abonados en nuestra base de datos:

```
@Entity
public class Abonado {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idAbonado;

    @NotBlank
    private String nombre;

    @NotBlank
    private String apellidos;

    @NotNull
    private LocalDate fechaNacimiento;

    private String direccion;

    private String ciudad;

    private String provincia;

    private String pais;

    private String codigoPostal;

    private String email;
```

```
    @NotBlank
    private String numeroTelefono;

    private String rutaImagen;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_centro")
    private CentroDeportivo centroDeportivo;

    @OneToMany(mappedBy = "abonado", cascade = CascadeType.ALL)
    private List<Reserva> reservas;

}
```

- **Interceptors:** Contiene la capa de seguridad de la aplicación. Los interceptores leen todas las peticiones que llegan a la aplicación y las filtran. En nuestro caso, existe un interceptor de autorización que verifica que el acceso a datos de la aplicación es utilizado por usuarios registrados en la misma.
  Este sería el código del mismo:

```
@Component
public class AuthorizationInterceptor implements HandlerInterceptor {

    @Autowired
    private UsuarioService usuarioService;

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        // Comprobar que existe la cabecera idUsuario
        String idUsuario = request.getHeader("idUsuario");
        // Si no existe la cabecera no dejaremos pasar
        if (idUsuario == null || idUsuario.isEmpty()) {
            throw new IllegalAccessException("No se ha especificado un ID de
Usuario");
        }
        // Comprobar que el usuario existe llamando al service.
        try {
            usuarioService.obtenerUsuario(Long.parseLong(idUsuario));
        } catch (NoSuchElementException e) {
            // El usuario no existe. No dejaremos pasar.
            throw new IllegalAccessException("ID de Usuario no válido");
        }

        return true;
    }
}
```

- **Repositories:** la capa de repositorios nos permite el acceso y la comunicación con la base de datos. Gracias a Hibernate, esta capa se compone de una serie de interfaces que extienden del framework, el cual nos otorga ya de una serie de métodos predefinidos.
  Por tanto, una interfaz de esta capa tendría el siguiente aspecto:

```
@Repository
public interface AbonadosRepository extends JpaRepository<Abonado, Long> {

}
```

- **Services:** la capa de servicios contiene toda la lógica de la aplicación. Esta capa recibe las peticiones de los controladores y realiza el procesamiento oportuno, en el que se puede incluir el acceso a datos a través de la capa de repositorios.
  Este sería el aspecto del servicio de abonados:

```
@Service
public class AbonadosService {

    @Autowired
    private AbonadosRepository abonadosRepository;

    @Autowired
    private CentrosService centrosService;

    @Autowired
    private ModelMapper modelMapper;

    public AbonadoVO crearAbonado(AbonadoVO abonadoVO) {
        // Mapear a entidad
        Abonado abonado = modelMapper.map(abonadoVO, Abonado.class);
        // Obtener los datos del centro deportivo
        CentroDeportivoVO centroVO =
centrosService.obtenerCentro(abonadoVO.getIdCentroDeportivo());
        CentroDeportivo centro = modelMapper.map(centroVO,
CentroDeportivo.class);
        abonado.setCentroDeportivo(centro);
        // Guardar en la base de datos
        abonado = abonadosRepository.save(abonado);
        // Mapear a VO y devolver
        return modelMapper.map(abonado, AbonadoVO.class);
    }

    public AbonadoVO obtenerAbonado(Long idAbonado) {
        // Obtener el abonado solicitado
        Optional<Abonado> optUsuario = abonadosRepository.findById(idAbonado);
        if (!optUsuario.isPresent()) {
            throw new NoSuchElementException("No se encuentra el abonado
solicitado");
        }
        // Mapear el usuario a VO y devolverlo
        return modelMapper.map(optUsuario.get(), AbonadoVO.class);
    }

    public void actualizarAbonado(AbonadoVO abonadoVO) {
        // Mapear a entidad
        Abonado abonado = modelMapper.map(abonadoVO, Abonado.class);
        // Obtener el abonado solicitado
        Optional<Abonado> optUsuario =
abonadosRepository.findById(abonado.getIdAbonado());
        if (!optUsuario.isPresent()) {
            throw new NoSuchElementException("No se encuentra el abonado
solicitado");
        }
        // Actualizar el usuario
        abonadosRepository.saveAndFlush(abonado);
    }

    public void borrarAbonado(Long idAbonado) {
        // Obtener el abonado solicitado
        Optional<Abonado> optUsuario = abonadosRepository.findById(idAbonado);
        if (!optUsuario.isPresent()) {
            throw new NoSuchElementException("No se encuentra el abonado
solicitado");
```

```
        }
        // Borrar el usuario
        abonadosRepository.deleteById(idAbonado);
    }

    public List<AbonadoVO> obtenerListadoAbonados(Long idCentroDeportivo) {
        // Obtener los datos del centro deportivo
        CentroDeportivoVO centro =
centrosService.obtenerCentro(idCentroDeportivo);
        // Devolver el listado de abonados del centro
        List<AbonadoVO> abonados = centro.getAbonados();
        // Ordenar alfabéticamente
        Collections.sort(abonados);
        return abonados;
    }
}
```

### 3.1.2. Despliegue de la aplicación backend

La aplicación se ha desplegado en Heroku gracias a los automatismos que esta nube pública nos ofrece: a través de un proceso de Integración Continua que se activa cada vez que se sube código a la rama *reléase* del repositorio.
La aplicación se ha desplegado junto con una base de datos *Postgresql* ofrecida y gestionada por el proveedor. Los esquemas de la base de datos se han creado automáticamente por la aplicación, por lo que no ha sido necesario una intervención manual.

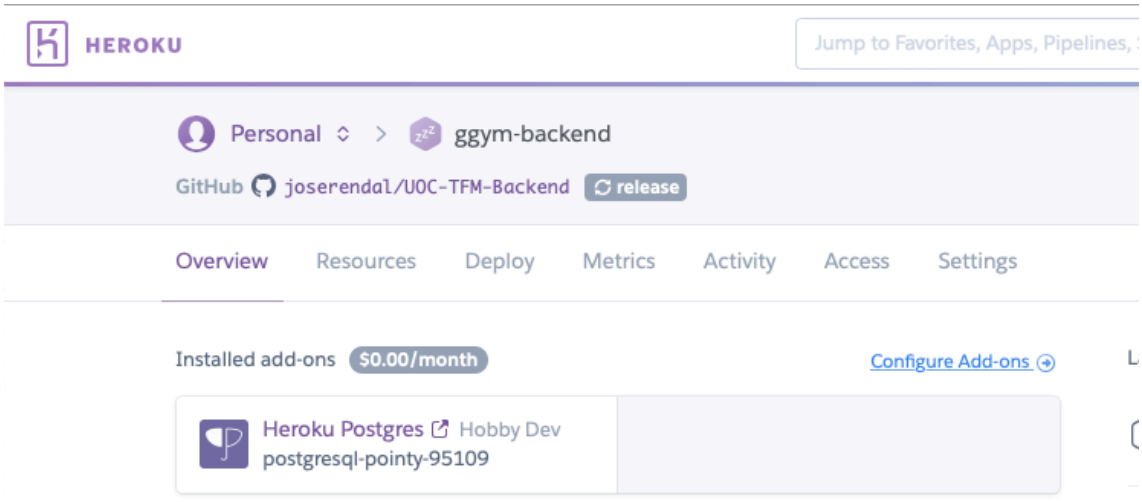



*Ilustración 26: Captura del backend desplegado en Heroku, junto a la base de datos.*

### 3.1.3. Cambios en el diseño de las APIs

Durante el proceso de desarrollo se han realizado las siguientes modificaciones sobre el diseño de las APIs para adaptarlas mejor a las necesidades de la aplicación móvil. En la siguiente tabla detallamos el endpoint afectado, el cambio realizado y la motivación del mismo:

| Endpoint | Cambio realizado | Motivación |
|---|---|---|
| **/credenciales/login** | Se modifican los parámetros de entrada para aceptar un JSON | Homogeneizar la entrada de este endpoint con el resto. |

| | | |
|---|---|---|
| | en el cuerpo de la petición en vez de parámetros normales. | |
| **/abonados** | Se ordena la salida de esta llamada por orden alfabético apellidos/nombre. | Asegurar la correcta presentación en el dispositivo móvil. |
| **/centros/usuario/{idUsuario}** | Se crea este nuevo endpoint para devolver el centro deportivo asociado a un usuario en concreto. | Se detectó como necesidad durante el desarrollo de la app. |
| **/pagos** | Se modifican todos los endpoints de gestión de pagos porque se incluye la información completa del abonado en la entrada y salida de estos métodos. Previamente se había definido que era necesario solo enviar el id del abonado. | En el diseño de la app se incluye que en la ventana de pagos pendientes se incluya toda la información del usuario. El backend no lo contemplaba así. |
| **/pagos/pendientes /pagos/cobrados** | Se añaden como parámetro el mes y el año en las llamadas para filtrar el listado. | En el diseño original el usuario solo podía ver los pagos pendientes y cobrados del mes actual. Sin embargo, se ha implementado esta mejora sobre el diseño. |
| **/cuota/id** | Se modifica el parámetro de entrada. Antes se recibida el ID de la Cuota en sí y ahora se recibe el ID del abonado. | Durante la construcción de la app no se dispone de ID de la cuota, solo del ID del usuario. |
| **/equipamiento/centro/id** | Se crea este nuevo endpoint en la API. | En el diseño original de la API no se contemplaba un endpoint para obtener el equipamiento de un centro en base al ID de este último. Sin embargo, es necesario para la vista de gestión de equipamiento. |
| **/equipamiento/**** | Cuando se obtienen los detalles de los equipamientos, se incluye la información de la reserva filtrada por el día actual. | Necesario para presentar el detalle de las reservas junto con el detalle del equipamiento. |

*Tabla 5: Cambios realizados sobre el diseño de las APIs*

### 3.1.4. Desarrollo de la app móvil

La aplicación móvil se ha desarrollado inicialmente utilizando el lenguaje de programación Swift 3. Durante el desarrollo de la aplicación se liberó la actualización pertinente de Mac OS y XCode, por lo que el proyecto entero se actualizó a Swift 5.
No se han incluido librerías de terceros ni dependencia externas mas allá de las ofrecidas por el propio ecosistema iOS.

La aplicación móvil dispone de la siguiente estructura interna:

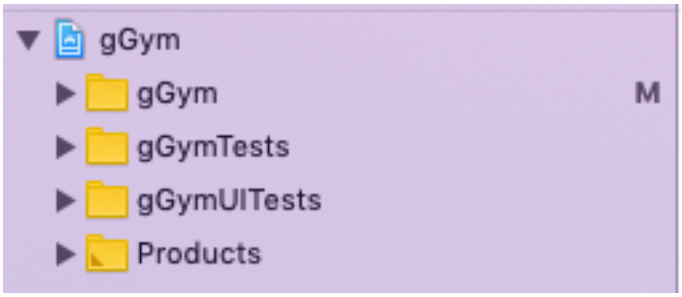



*Ilustración 27: Contenido de la app móvil*

Estos son los distintos proyectos que contiene:

- **gGym:** aplicación móvil principal. Contiene el código fuente y las pantallas de la misma.
- **gGymTests:** pruebas unitarias de la aplicación móvil.
- **gGymUITests:** pruebas de la interfaz de usuario de la aplicación móvil.

Centrándonos en la aplicación móvil principal, esta sería la estructura de la misma:

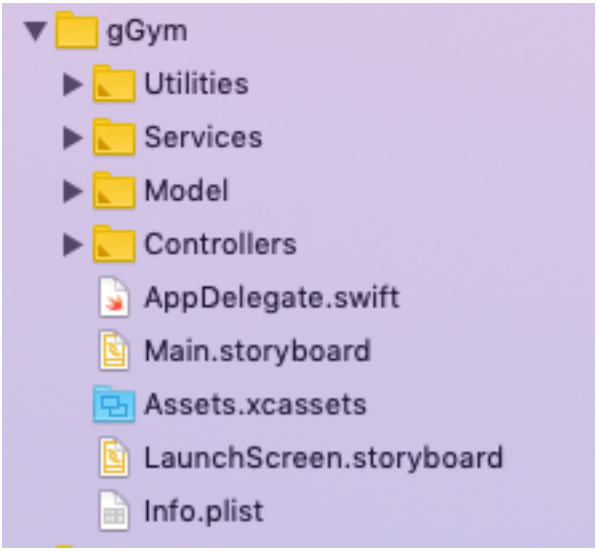



*Ilustración 28: estructura de paquetes de la app móvil*

El fichero *Main.storyboard* contiene el storyboard principal de la aplicación, con todas las pantallas de la misma.
Centrándonos en el contenido de cada paquete:

- **Utilities:** Contiene una serie de clases con utilidades reutilizables dentro de la aplicación. En nuestro caso, se dispone de una clase con funcionalidades para la validación de los formularios de entrada de la aplicación, así como para mostrar diálogos en la interfaz de usuario.
  Este sería el contenido de la clase de validación de formularios, donde se dispone de dos funciones para validar campos de texto y de email:

```
class FormValidator {

    // Validate whether a field is fulfilled with data. If not,
display alert to the user.
    class func
validateRequiredTextFieldAndDisplayAlert(textField: UITextField,
fieldName: String, callerController: UIViewController) -> Bool {
        // Check if the field is present
        if (textField.text?.isEmpty)! {
            // Display validation error
            DialogHelper.displayErrorDialogWithoutAction(title:
"Error", message: "El campo " + fieldName + " es requerido",
button: "Cerrar", callerController: callerController)
            // Validation not succeded
            return false
        }
        // Validation OK
        return true
    }

    // Validate email address
    class func validateEmailAddressAndDisplayAlert(textField:
UITextField, callerController: UIViewController) -> Bool {
        // Email regular expression
        let emailRegEx = "[A-Z0-9a-z._%+-]+@[A-Za-z0-9.-]+\\.[A-
Za-z]{2,64}"
        // Predicate for validation
        let emailTest = NSPredicate(format:"SELF MATCHES %@",
emailRegEx)
        // Validate email address with predicate
        if(!emailTest.evaluate(with: textField.text)) {
            // Display validation error
            DialogHelper.displayErrorDialogWithoutAction(title:
"Error", message: "El email introducido no es válido", button:
"Cerrar", callerController: callerController)
            // Validation not succeded
            return false
        }
        // Validation OK
        return true;
    }

}
```

- **Services:** contiene todas las clases necesarias para la comunicación con las APIs expuestas por el backend, ya sea para enviar datos o para realizar lectura de los mismos.
  Este sería el contenido de la clase encargada de la comunicación con el servicio de Usuarios:

```
class UsuariosService {

    // Call to the Backend service for creating the User
    class func createUser(user: User!) -> User {
        // Semaphore for controlling execution
        let semaphore = DispatchSemaphore(value: 0)
        // Build HTTP Request
        let request : NSMutableURLRequest =
NSMutableURLRequest()
        request.url = URL(string: Constants.apiHost +
Constants.usuariosPath + "/crear")
        request.httpMethod = "POST"
        request.timeoutInterval = 30
        // Request headers
        request.addValue("application/json", forHTTPHeaderField:
"Content-Type")
        // Request body
        request.httpBody = user.jsonRepresentation
        // Send request
        let dataTask = URLSession.shared.dataTask(with: request
as URLRequest, completionHandler: {data, response, error in
            // Get the HTTP Response
            let httpResponse = response as? HTTPURLResponse
            // Return true or false
            if httpResponse?.statusCode == 200{
                do {
                    // get the user ID
                    let json = try
JSONSerialization.jsonObject(with: data!,
options:.allowFragments)  as! [String:Any]
                    user.idUsuario = json["idUsuario"] as! CLong
                } catch let parseError as NSError {
                        print("JSON Error
\(parseError.localizedDescription)")
                }
            } else {
                user.idUsuario = 0
            }
            // End semaphore
            semaphore.signal()
        })
        // Execute and wait
        dataTask.resume()
        semaphore.wait()
        // Return the result
        return user
    }
}
```

- **Model:** contiene las clases con el modelo de datos de la aplicación, que se asemeja al modelo de datos del backend.
  Este sería la clase encargada de modelar los datos de los abonados:

```
class Client: NSObject {

    var idAbonado: CLong
    var idCentroDeportivo: CLong
    var nombre: String
    var apellidos: String
    var direccion: String
    var ciudad: String
    var provincia: String
    var pais: String
    var codigoPostal: String
    var email: String
    var numeroTelefono: String
    var fechaNacimiento: Date

    init?(nombre: String, apellidos: String, direccion: String,
ciudad: String, provincia: String, pais: String, codigoPostal:
String, email: String, numeroTelefono: String, fechaNacimiento:
Date) {
        self.idAbonado = 0
        self.idCentroDeportivo = 0
        self.nombre = nombre
        self.apellidos = apellidos
        self.direccion = direccion
        self.ciudad = ciudad
        self.provincia = provincia
        self.pais = pais
        self.codigoPostal = codigoPostal
        self.email = email
        self.numeroTelefono = numeroTelefono
        self.fechaNacimiento = fechaNacimiento
    }

    var jsonRepresentation : Data {
        // Date formatter
        let dateformatter = DateFormatter()
        dateformatter.dateFormat = "yyyy-MM-dd"
        // JSON dictionary
        let dict = ["idAbonado" : idAbonado,
                    "idCentroDeportivo" : idCentroDeportivo,
                    "nombre" : nombre,
                    "apellidos" : apellidos,
                    "direccion" : direccion,
                    "ciudad" : ciudad,
                    "provincia" : provincia,
                    "pais" : pais,
                    "codigoPostal" : codigoPostal,
                    "email" : email,
                    "numeroTelefono" : numeroTelefono,
```

```
                    "fechaNacimiento" :
dateformatter.string(from: fechaNacimiento)] as [String : Any]
        return try! JSONSerialization.data(withJSONObject: dict,
options: .prettyPrinted)
    }
}
```

- **Controllers:** capa encargada de la comunicación con las interfaces de usuario de la aplicación. Contiene además toda la lógica de validación de la capa de presentación, y se comunica con la capa de servicios para recoger y enviar datos al backend.
  Este sería el contenido del controlador de login de la aplicación:

```
class LoginViewController: UIViewController, UITextFieldDelegate
{

    // Outlets
    @IBOutlet weak var usernameField: UITextField!
    @IBOutlet weak var passwordField: UITextField!
    @IBOutlet weak var activityIndicator:
UIActivityIndicatorView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Clean password
        usernameField.text = nil
        passwordField.text = nil
        // Check if credentials stored. If so, display on screen
        let userDefaults: UserDefaults = UserDefaults.standard
        if let data = userDefaults.object(forKey: "credentials")
{
            let credentials =
NSKeyedUnarchiver.unarchiveObject(with: data as! Data) as?
Credentials
            usernameField.text = credentials?.email
            passwordField.text = credentials?.password
        }
        // Set deletegates for text field
        self.usernameField.delegate = self
        self.passwordField.delegate = self
    }

    // When the user clicks outside text fields, hide the
keyboard
    override func touchesBegan(_ touches: Set<UITouch>, with
event: UIEvent?) {
        self.view.endEditing(true)
    }

    @IBAction func loginTapped(_ sender: UIButton) {
        // Get the username and password introduced by the user
        let username:String = usernameField.text!
        let password:String = passwordField.text!
        // Check both fields are fulfilled
```

```
        if(username.isEmpty || password.isEmpty) {
            // Display an error message
            let alertController = UIAlertController(title:
"Error", message: "Por favor, revise sus credenciales",
preferredStyle: .alert)
            alertController.addAction(UIAlertAction(title:
"Cerrar", style: .default))
            self.present(alertController, animated: true,
completion: nil)
            // End the execution of the action
            return
        }
        // Show loading dialog
        activityIndicator.startAnimating()
        // Create a credentials object
        let credentialsObject = Credentials(idUsuario: 0, email:
username, password: password)
        // Call the API and get the response
        let credentials = CredentialsService.login(credentials:
credentialsObject)
        // If request was wrong
        if(credentials.idUsuario == 0) {
            // Hide loading dialog
            activityIndicator.stopAnimating()
            // Display an error message
            DialogHelper.displayErrorDialogWithoutAction(title:
"Error", message: "No se ha podido autenticar en el sistema",
button: "Cerrar", callerController: self)
            // End the execution of the action
            return
        }
        // Save the credentials object in the database
        let encodedData =
NSKeyedArchiver.archivedData(withRootObject: credentialsObject!)
        let userDefaults: UserDefaults = UserDefaults.standard
        userDefaults.set(encodedData, forKey: "credentials")
        userDefaults.synchronize()
        // Hide loading dialog
        activityIndicator.stopAnimating()
        // Send the user to the Main Menu
        self.performSegue(withIdentifier: "mainMenuSegue",
sender: self)
    }

}
```

### 3.1.5. Cambios sobre el diseño de la app móvil

Durante el proceso de desarrollo se han realizado las siguientes modificaciones y mejoras no previstas en el diseño inicial de la aplicación:

- Se ha definido una pantalla de carga para que se muestre cuando el usuario inicia la aplicación en su dispositivo, antes de visualizar la pantalla de login. Se trata de una funcionalidad ofrecida por el propio ecosistema y que mejora la experiencia del usuario.
  Esta pantalla se ha desarrollado en consonancia con la pantalla de login (mismo icono de la app y orientación de los elementos).

- Se ha modificado el diseño de la vista de gestión de pagos para que permita seleccionar la fecha y el año. Esto no estaba previsto en el diseño inicial y limitaba al usuario, ya que solo podría ver los pagos cobrados y pendientes del mes en curso y no de meses pasados. Para ello, se ha implementado un botón en la barra de navegación que enseña un dialogo modal de selección de fechas. El listado de pagos se refresca automáticamente en base a ese parámetro.

- Se ha modificado el diseño de la vista de gestión de equipamiento. En el prototipo original se definían dos pantallas: una para consultar el listado de equipamiento disponible, y otra para acceder a los detalles del equipamiento y realizar la reserva.
  Se ha procedido a separar en dos pantallas distintas esta última (consulta de detalles del equipamiento y pantalla de reserva) para facilitar la comprensión al usuario del proceso y evitar la sobrecarga de información en una única pantalla.

## 3.2. Pruebas de la aplicación

### 3.2.1. Pruebas unitarias

Dentro del módulo gGymTests de la aplicación se han desarrollado una batería de pruebas unitarias diseñadas para testear la correcta funcionalidad de la aplicación, así como la comunicación con el backend.
Todas las pruebas se han ejecutado en el entorno local para poder utilizar el emulador de iOS que viene incorporado con XCode, ya que es necesario para poder lanzar las pruebas de rendimiento y las pruebas automatizadas sobre la interfaz de usuario.

**Batería para el modelo de datos**

Se ha desarrollado una única prueba para esta capa: una prueba unitaria para la clase *Credenciales* del modelo. Se ha testeado esta y no otras clases del modelo ya que esta instancia se almacena en el dispositivo del usuario para que no tenga que escribir las credenciales cada vez que abra la aplicación. La prueba unitaria desarrollada testea esta funcionalidad específica que es clave para la experiencia del usuario.
El resto de las clases del modelo se cubrirán íntegramente al testear la capa de servicios, que intercambia estos objetos con el backend de la aplicación.

**Batería para la capa de servicios**

Para testear la capa de servicios se han desarrollado un total de 31 pruebas unitarias que cubren por si solas el 59.3% del código de la aplicación (al excluir la capa de controladores), de ahí la importancia de testear esta capa.
Para construir las pruebas, se ha seguido la siguiente guía:

- Para las operaciones de lectura, probaremos a llamar al backend con parámetros de entrada que no devuelvan datos, así como con parámetros de entrada que si los devuelvan. Estas últimas peticiones se someterán además a pruebas de rendimiento (se explicarán en el siguiente apartado).
- Para las operaciones de escritura, se realizarán llamadas al backend con datos inválidos para provocar errores y no modificar el estado del mismo, salvo para las operaciones de actualización que si se testearán realmente.

### 3.2.2. Pruebas de rendimiento de las operaciones de lectura de datos

Aprovechándonos de las funcionalidades para realizar pruebas de rendimiento ofrecidas por el IDE XCode, se ha sometido a prueba de rendimiento las siguientes operaciones de lectura de la capa de servicios, que se comunican con el backend y realizan el procesamiento de los datos recibidos:

| Status | Tests | Duration | Time |
|---|---|---|---|
| ▼ | CentersServiceTest › gGymTests 4 passed (100%) in 2s | | |
| ✓ | testGetCenterPerformance() | 1s | 0,145s |
| ▼ | ClientsServiceTest › gGymTests 7 passed (100%) in 4s | | |
| ✓ | testGetClientDetailsPerformance() | 1s | 0,125s |
| ✓ | testGetClientsofValidCenterPerformance() | 1s | 0,167s |
| ▼ | CredentialsServiceTest › gGymTests 4 passed (100%) in 2s | | |
| ✓ | testLoginPerformance() | 1s | 0,163s |
| ▼ | EquipmentsServiceTest › gGymTests 4 passed (100%) in 2s | | |
| ✓ | testGetEquipmentsForCenterPerformance() | 1s | 0,137s |
| ▼ | PaymentsServiceTest › gGymTests 5 passed (100%) in 3s | | |
| ✓ | testGetPaidPaymentsPerformance() | 1s | 0,138s |
| ✓ | testGetPendingPaymentsPerformance() | 1s | 0,152s |
| ▼ | ReceiptsServiceTest › gGymTests 4 passed (100%) in 2s | | |
| ✓ | testGetReceiptsPerformance() | 1s | 0,169s |

*Ilustración 29: Resultado de las pruebas de rendimiento de la capa de servicios de la app*

- Login (autenticación con el backend)
- Lectura de los datos del centro deportivo asociado al usuario
- Lectura de los clientes de un centro deportivo, y de un cliente en concreto
- Lectura de todos los recibos pendientes y pagados de un centro deportivo
- Lectura de la cuota de un socio
- Lectura de todo el equipamiento de un centro deportivo

El objetivo de estas pruebas es detectar a tiempo problemas de rendimiento en las llamadas al backend que puedan ocasionar al usuario una mala experiencia de usuario.

### 3.2.3. Pruebas automatizadas sobre la interfaz de usuario (UI)

Dentro del módulo gGymUITests de la aplicación se ha desarrollado una batería de pruebas que simulan las interacciones de un usuario sobre la pantalla real de la aplicación. Esta batería ha sido desarrollada con la ayuda de las funcionalidades del IDE XCode.

Estas pruebas están orientadas a simular el comportamiento real de un usuario interaccionando con las distintas vistas de la aplicación. En ellas probamos:

- Que la respuesta de los distintos componentes visuales ante las interacciones del usuario es la correcta.

- Que el usuario recibe los mensajes de error adecuados cuando introduce datos erróneos o intenta ejecutar acciones sin rellenar todos los datos necesarios.
- Que la navegación entre las pantallas de la aplicación respeta lo definido en el prototipo de la misma.

Para ser mas concisos, se han probado los siguientes flujos que harían los usuarios normales:

- Login y logout
- Login con credenciales inválidas
- Navegación por la pantalla de creación de usuarios
- Intento de registro de usuario sin los campos requeridos
- Intento de registro de usuario con todos los campos requeridos, pero contraseñas distintas
- Creación de un equipamiento
- Reserva del equipamiento
- Actualización de los datos de un abonado
- Creación de un abonado y pago del recibo del mes en curso

Gracias a toda la batería de pruebas desarrollada, el proyecto alcanza un 91.8% de cobertura de código.


*Ilustración 30: Cobertura de código final después de la ejecución de la batería de pruebas*

### 3.2.4. Pruebas de usabilidad en distintos dispositivos

Se ha procedido a probar la aplicación en varios dispositivos iPhone e IPad para comprobar la correcta usabilidad de la misma con varios tamaños de pantalla. Entre ellos, se ha probado la aplicación en los siguientes dispositivos emulados:

- iPhone 6 – pantalla de 4,7 pulgadas y resolución de 750x1334 px
- iPhone 8 Plus – pantalla de 5.5 pulgadas y resolución de 1080x1920 px
- iPhone XS Max – pantalla de 6,5 pulgadas y resolución de 1242x2688 px
- iPad 6ª generación – pantalla de 9,7 pulgadas y resolución de 1536x2048px

## 3.3. Revisión de la planificación

Con respecto a la planificación inicial se produjeron las siguientes desviaciones durante el desarrollo de la solución propuesta:

- El tiempo de desarrollo del backend, estimado originalmente en 12h, fue ligeramente superior. Además, fue necesario realizar las modificaciones descritas en la *Tabla 5: Cambios realizados sobre el diseño de las APIs* durante el desarrollo de la aplicación móvil, por lo que el tiempo final de desarrollo del backend fue de unas 20h.
- Durante el desarrollo de la aplicación móvil se produjeron retrasos debido a la actualización a Swift 5, así como por el desarrollo de las mejoras detalladas en la *sección 3.1.5 Cambios sobre el diseño de la app móvil.* Sin embargo, se definieron estas mejoras como necesarias para mejorar la usabilidad y la calidad de la aplicación móvil con respecto al diseño final.
- También se produjeron retrasos durante las primeras fases del desarrollo de la aplicación móvil debido a la necesidad de refrescar los conocimientos adquiridos en la plataforma móvil elegida.

La desviación producida durante el desarrollo del backend, mencionada en el primer punto de la lista, fue compensada debido a que la previsión de horas para el desarrollo del sistema gestor de base de datos (8h) fue innecesaria, ya que esta fue aprovisionada automáticamente por el proveedor de nube pública utilizado y los esquemas creados por la propia aplicación gracias a Hibernate.

Sin embargo, para corregir los retrasos producidos durante el desarrollo de la aplicación móvil fue necesario aumentar la dedicación al proyecto en tres días laborables, no previstos inicialmente, en el que se invirtieron cerca de 20h no previstas. Además, fue necesario también aumentar la dedicación prevista para las pruebas de la aplicación, estimadas originalmente en 8h, para desarrollar pruebas sobre la interfaz de usuario que, aunque no estaban contempladas inicialmente han servido para mejorar la calidad del producto y detectar fallos en la aplicación.

# Capítulo 4: **Conclusiones**

# Capítulo 4. Conclusiones

Concluido el desarrollo y las pruebas de la aplicación, es el momento de reflexionar acerca del desarrollo del proyecto de este Trabajo Final de Master, de las lecciones aprendidas y de exponer las líneas de trabajo futuras.

Durante el desarrollo del proyecto he aprendido la importancia de la fase de planificación y del ejercicio de diseñar y estimar las tareas sobre el calendario para la consecución de los objetivos. Durante esta fase fijamos las metas, que han de ser ambiciosas pero realistas, pero sobre todo acotamos el alcance y fijamos las prioridades para ofrecer el mejor producto posible. Gracias al seguimiento de la planificación he podido identificar en todo momento en que punto del desarrollo me encontraba, para poder incluir medidas para corregir desvíos y conseguir el objetivo final.

Además, gracias también a la fase de diseño los tiempos de desarrollo se han acortado sensiblemente al tener claro el objetivo que alcanzar. Disponer de toda la información de diseño de la arquitectura de la solución, las APIs necesarias y las interfaces de usuario facilita enormemente el trabajo de desarrollo.

Al respecto de los objetivos planteados, se ha conseguido alcanzar e implementar la mayor parte de ellos, con algunas excepciones. Por ejemplo, los requisitos iniciales planteaban la necesidad de incluir fotografías de los abonados en la aplicación móvil. Lamentablemente, no se ha podido dar respuesta a este requisito puesto que el tiempo de desarrollo era muy grande y excedía la capacidad disponible. Por ello, se han priorizado el resto de los requisitos para intentar alcanzar el mejor producto posible, dentro del seguimiento de la planificación realizado.
Se han tenido que introducir medidas para corregir desvíos en el seguimiento de la misma, tal y como se ha expuesto en el capítulo anterior de esta memoria, tales como aumentar la dedicación al proyecto durante la fase de desarrollo. Este tiempo extra no estaba incluido en la planificación inicial y se compensó en parte con los ahorros conseguidos en el desarrollo del backend.

A modo de resumen, hay que destacar la lección aprendida acerca de la importancia de las fases previas la desarrollo de un proyecto de este tipo. Centrándonos en este Trabajo Final de Master, destacar que se han cumplido la mayor parte de requisitos expuestos, aunque ha sido necesario la introducción de correcciones para corregir los desvíos en la planificación inicial, que por otra parte no han sido excesivos.

## 4.1. Ampliaciones propuestas

Una vez expuestas las conclusiones de este Trabajo Final de Master, es el momento de exponer las líneas de trabajo futuro sobre las que continuar enriqueciendo la solución propuesta.
Para ello, podemos definir tres grandes líneas de trabajo a alto nivel:

- Añadir nuevas funcionalidades a la aplicación móvil desarrollada.

- Portar la aplicación a otros ecosistemas.
- Mejorar las funcionalidades existentes en el backend.

A continuación detallaremos las propuestas para cada línea de trabajo.

### 4.1.1. Mejorar la aplicación móvil desarrollada

Existen funcionalidades adicionales que pueden añadirse en la aplicación existente para mejorar la experiencia del usuario:

- Cobro de recibos a través de tarjeta bancaria, paypal u otras plataformas de pago virtuales.
- Publicación y gestión de clases dirigidas. Para ello, sería necesario ampliar el sistema de reservas existente.
- Inventariado de material disponible en el centro.
- Incluir fotografías en la gestión de abonados y del equipamiento.
- Lanzamiento de campañas publicitarias vía email, sms, etc.
- Utilizar mas el almacenamiento local del dispositivo para reducir el volumen de peticiones dirigidas al backend.

Además, podría desarrollarse también una segunda aplicación móvil que complemente a la ya existente, pero desde el punto de vista del cliente. A esta nueva aplicación traspasaríamos algunas de las funcionalidades principales desarrolladas, tales como:

- La reserva de equipamientos
- El pago de recibos

### 4.1.2. Portar la aplicación a otros ecosistemas

La aplicación desarrollada solo está disponible para la plataforma iOS. Para ampliar la base del público objetivo se podrían desarrollar aplicaciones para el ecosistema Android y web.
Con ello, ampliaríamos la base de usuarios y dispositivos en los que se podría utilizar la aplicación. Gracias al backend desarrollado, el usuario podría cambiar de dispositivo sin ningún perjuicio, ya que sus datos estarían siempre disponibles y actualizados.

### 4.1.3. Mejorar el backend

La última de las líneas de trabajo sería la de mejorar el backend. Actualmente, se encuentra desplegado en la capa gratuita de la nube de Heroku, por lo que la escalabilidad de la solución es limitada.
Sería necesario, por tanto, buscar un proveedor de nube pública alternativo que nos ofrezca capacidades de escalabilidad automática con un buen sistema de precios, tanto para la aplicación en sí como para la base de datos.
Además, sería necesario también añadir nuevas funcionalidades en el backend que soporten las ampliaciones propuestas en las otros dos líneas de trabajo, puesto que deberá de ser el backend quien las soporte.

# Capítulo 5: **Glosario**

# Capítulo 5. Glosario

| Termino | Definición |
|---|---|
| **Android** | Sistema operativo móvil desarrollado por Google, que está presente en la mayoría de los dispositivos presentes en el mercado. |
| **Backend** | Aplicación sin interfaz de usuario, desplegada en un servidor, encargada de gestionar las comunicaciones con el sistema de almacenamiento de datos y la lógica de negocio, entre otras. Normalmente, dispone de una capa para exponer distintos servicios a otras aplicaciones. |
| **CRM** | Abreviatura de *Customer Relationship Management,* software específico para la gestión de clientes, con funcionalidades tipo campañas publicitarias y de fidelización para empresas. |
| **Dashboard** | Interfaces de usuario cuya misión es ofrecer información rápida acerca de la marcha del negocio. Suelen estar formadas por gráficas y números principalmente, prescindiendo de textos. |
| **iOS** | Sistema operativo móvil desarrollado por Apple, presente únicamente en sus dispositivos móviles y tabletas. |
| **REST** | Acrónimo de *Representational State Transfer,* patrón de arquitectura de comunicaciones entre el cliente y el servidor basado en servicios sin estado, con los que nos comunicamos vía HTTP. |
| **SGBD** | Abreviatura de Sistema Gestor de Base de Datos, software encargado de gestionar almacenamiento de datos en formato relacional (tablas) o no relacional (otros, como documentos, JSON, grafos, etc.) |
| **Spring Boot** | Framework basado en Java para simplificar el desarrollo de aplicaciones servidor. |
| **Swift** | Lenguaje de programación nacido en 2014 utilizado para la programación de aplicaciones, principalmente destinadas al ecosistema iOS. La última versión disponible es Swift 5. |
| **XCode** | Entorno de desarrollo de Apple para la programación de aplicaciones (tanto móviles como de escritorio) para su ecosistema. |

*Tabla 6: Glosario de términos*

# Capítulo 6: **Bibliografía**

# Capítulo 6. Bibliografía

2019. *Aplicación móvil Global Gym Software - Google Play Store.* 2 de Marzo. https://play.google.com/store/apps/details?id=in.globalcrm.global.gym.software.

2019. *Arquitectura Java - Que es Spring Boot.* 3 de Marzo. https://www.arquitecturajava.com/que-es-spring-boot/.

2019. *Dossier del software modulable de GPA Sport.* 2 de Marzo. http://www.gpasport.com/sites/gpasport.com/files/docs/general_gpasport.pdf.

2019. *Página principal de Spring Boot.* 3 de Marzo. https://spring.io/projects/spring-boot.

s.f. «Presentación "Modelos prescriptivos de proceso", disponible en Slide Share.» Coesi Consultoría. Último acceso: 2 de Marzo de 2019. https://es.slideshare.net/coesiconsultoria/sesin-3-modelos-prescriptivos-de-proceso.

s.f. *Que es REST.* https://desarrolloweb.com/articulos/que-es-rest-caracteristicas-sistemas.html.

2019. *Sitio web de GPA Sport.* 2 de Marzo. http://www.gpasport.com/soluciones.

2019. *Sitio web del software IsMyGym.* 2 de Marzo. http://www.ismygym.com/software-web-iphone-android-gestion-reservas/.

s.f. *What is Heroku.* https://www.heroku.com/what.

# Capítulo 7: **Anexos**

# Capítulo 7. Anexos

Adjunto a esta Memoria del Trabajo Final de Master se incluyen los siguientes documentos:

- **Manual de usuario** de la aplicación móvil, dirigido al usuario final de la misma para facilitar la comprensión de todas las funcionalidades y capacidades del producto desarrollado.
- **Guía de arranque del Backend,** con las instrucciones necesarias para importar en el IDE la aplicación desarrollada para ofrecer las APIs REST que son consumidas por la aplicación móvil.
- **Guía de uso de la colección Postman,** con las instrucciones necesarias para importar la colección Postman y realizar llamadas a las APIs REST disponibles en el proveedor de nube pública escogida.
- **Guía de arranque de la app móvil,** con las instrucciones necesarias para importar en el IDE la aplicación móvil desarrollada y arrancarla en un emulador de iOS.