Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ESTUDIO Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN WEB PARA LA GESTIÓN DE PLANES DE ENTRENAMIENTO

Alumno: Mohamed Manuel El Karkri Izquierdo

Tutor: Prof. D. José Ramón Balsas Almagro
Dpto: Lenguajes y Sistemas Informáticos

11,2020

Índice

Tabla de Ilustraciones

1. Introducción

En este primer punto del Trabajo de Fin de Grado especificaré las distintas motivaciones y causas que me han llevado a desarrollar un prototipo de aplicación web. Para ello, realizaré una breve descripción del proyecto, añadiendo su correspondiente justificación, amén de los objetivos presentados en la propuesta inicial.

El sistema que se pretende desarrollar trata de la gestión de un gimnasio, pero con una funcionalidad añadida: la del seguimiento personal del usuario para ayudarle a obtener sus objetivos. Lo innovador es que el seguimiento no se realizará únicamente suministrando una dieta, sino proporcionando unos macros personalizados que definirán, cuanto debe ingerir una persona para alcanzar sus metas.

1.1. Motivación

Comer saludablemente, cuidar el cuerpo con actividad física, es un tema a la orden del día; la población en general está cada vez más concienciada con su propio cuerpo, lo entiende como el eje de todo cambio. Los gimnasios, antes considerados como centros casi de ocio, son ahora una parte fundamental de la rutina diaria para cuidar la salud. Centros donde uno aprende a cuidarse a sí mismo.

El proyecto que se va a desarrollar consiste en un prototipo de aplicación web, que se divide en dos partes:

***Backend*:** Es la parte del desarrollo que se encarga de que toda lógica de una aplicación web funcione, es decir es la parte del servidor. Esta será implementada mediante el *framework* Spring Boot, el cual es un conjunto de herramientas que nos permite crear una aplicación Spring mediante una sencilla configuración.

***Frontend*:** Es la parte del desarrollo que se encarga de toda la parte visible de la aplicación web, es decir, es la parte con la que estará en contacto el usuario, la interfaz. Esta será implementada mediante Angular, el cual es un *framework* para aplicaciones web basadas en navegador.

El motivo de escoger Spring Boot viene precedido de las asignaturas Desarrollo de Aplicaciones Web y Desarrollo de Aplicaciones Empresariales. Gracias a estas asignaturas he decidido a qué me quiero dedicar en mi futuro, además de que tengo la gran oportunidad de desarrollar este proyecto junto con el docente que comencé esta aventura. Por lo tanto, mi principal motivación profesional es poder desarrollarme aún más en este ámbito que es el que me interesa de cara al futuro.

El motivo de escoger Angular 2 es debido a que es algo nuevo, que no he utilizado nunca y me gustaría poder obtener conocimientos nuevos, además de que he sido seleccionado para realizar un curso de programación *full stack* de la empresa Indra y considero que esto me serviría en mi desarrollo como programador.

1.2. Objetivos

Nuestros principales objetivos a satisfacer para el desarrollo de nuestro prototipo de aplicación web son los siguientes:

- Realizar un estudio de necesidades y soluciones existentes en el contexto seleccionado.
- Diseñar un sistema informático especialmente orientado a la utilización de arquitecturas, principios de diseño y metodologías de desarrollo web.
- Seleccionar un entorno de desarrollo web e implementar un prototipo de aplicación web que satisfaga las necesidades que se hayan determinado.

1.3. Metodología

En primer lugar, hemos decidido realizar este desarrollo de Gestión de planes de entrenamiento mediante ingeniería de software. La ingeniería de software es una de las ramas de las ciencias de computación que estudia la creación de software confiable y de calidad, aplicando un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software. Dentro de este método nos hemos basado en el desarrollo iterativo, dónde hemos dividido el proyecto en diversos bloques temporales llamados iteraciones. Aunque he utilizado algunas técnicas de SCRUM [1], como las historias

de usuario, que son las que se proponen como objetivo cada iteración, decidí no trabajar con esta metodología debido a que no tiene sentido cuando solo trabaja una persona en el proyecto.

Para alcanzar los objetivos propuestos se realizarán las siguientes actividades:

- Análisis: en este apartado tratamos de obtener los requisitos del sistema mediante las historias de usuario. Las historias de usuario son una representación de un requisito del sistema a través del lenguaje del usuario. Son utilizadas en las metodologías ágiles su estructura es la siguiente “Como [rol del usuario], quiero … y/o para…” Esto es fantástico ya que te plantea el objetivo, pero tú o tu equipo de desarrollo sois los que debéis decidir cómo satisfacer este objetivo y ofrecer el mejor servicio al usuario.
- Diseño: consiste una vez has obtenido el modelo del dominio, en la fase del análisis, aplicando ciertas técnicas y principios obtener la definición del sistema a desarrollar con suficientes detalles como para permitir su interpretación y realización física. Para esto vamos a utilizar diagrama de Entidad-Relación, diagrama de clases, diagrama de secuencia, entre otros. Todo esto se realiza de forma que no se entre en detalles de implementación.
- Implementación: en esta fase trataremos de traducir a código lo que hemos desarrollado en la fase del diseño.
- Pruebas: Nos centraremos en los resultados obtenidos de las pruebas, además de explicar algunos detalles sobre estas pruebas
- Conclusiones: debemos realizar una retrospectiva de nuestro proyecto, indicando también que mejora se podrían realizar en el futuro sobre nuestro proyecto.
- Manual de instalación y de usuario: como fase final, pero no por ello menos importante, es necesario realizar un manual de instalación y de usuario. Por si es necesario continuar en un futuro con este proyecto, añadiendo nuevas funcionalidades, extendiéndolo o mejorando otras partes, para dar facilidad al desarrollador.

1.4. Estructura del documento

Hemos decidido organizar de la siguiente forma este documento:

- En el apartado 2 llamado análisis realizaremos un estudio detallado del proyecto navegando a través de diferentes apartados, donde trataremos un estudio inicial sobre este proyecto, seguido de nuestra propuesta de proyecto, presentaremos nuestros requisitos del sistema, inmediatamente realizaremos nuestra planificación inicial de tareas, acompañado del estudio de viabilidad, finalmente presentaremos nuestro modelo de dominio, bien detallado y desarrollo de los casos de uso.
- En el apartado 3 diseño, como su nombre indica está compuesto por el diseño de varios diagramas. Desarrollaremos y explicaremos el diagrama de Entidad-Relación, diagrama de clases, diagramas de secuencia. Asimismo, especificaremos el desarrollo de la interfaz utilizada en el proyecto. Finalmente idearemos el plan de pruebas, detallando qué pruebas vamos a realizar, cómo las vamos a realizar, dónde se van a aplicar, en qué momento del proyecto dependiendo de la metodología escogida.
- En el apartado 4 implementación detallaremos la arquitectura de nuestro sistema desarrollado, incluyendo todos los elementos que lo constituye. Finalmente, detallaremos la implementación que hemos realizado durante este proyecto para poder mostrar los recursos empleados para el desarrollo del mismo.
- En el apartado 5, comentamos los resultados obtenidos a través de las pruebas, ya planificadas durante la etapa de planificación, ya mencionada anteriormente, proporcionando una vista general de estos resultados.
- En el apartado 6 llamado conclusiones, detallaremos nuestra conclusión acerca de nuestro trabajo, añadiendo unas posibles mejoras y trabajos futuros.
- Finalmente, en el apartado 7 encontramos la bibliografía, que hemos ido completando a medida que desarrollamos nuestro proyecto.

2. Análisis

2.1. Análisis preliminar

Tomaremos como punto de partida que la gestión de un gimnasio tradicionalmente se ha desarrollado de forma manual. Cuando un cliente se da de alta en un gimnasio siempre persigue un objetivo, ya sea mantenerse en forma, ponerse en forma o simplemente como hobby. Estos objetivos se traducen en ganancia muscular, conocida en el mundo del *fitness* como *etapa de volumen*; mantenimiento del peso, conocida como *etapa normocalórica* o pérdida de grasa, conocida como *etapa de definición*.

Normalmente una persona que se establece un objetivo dentro del mundo del *fitness* suele estar asesorado por un entrenador personal. Este entrenador lo que hace es ayudar a conseguir este objetivo mediante una rutina y una dieta.

En un gimnasio existen monitores que pueden establecer a cada cliente su propia rutina personalizada. Sin embargo, difícilmente pueden hacer un estudio de cada cliente dependiendo de sus objetivos para establecer un plan específico. Es aquí donde radica el problema.

Actualmente todo se realiza presencialmente, es decir, cuando un cliente se da de alta en un gimnasio se le proporciona un horario de las clases que se imparten. Si el cliente desea apuntarse a alguna de ellas deberá hablar con el monitor para que le apunte. Además, si hay cambios de horarios, se debe avisar de este hecho a todos los usuarios por algún método de contacto, ya sea teléfono, email, etc.

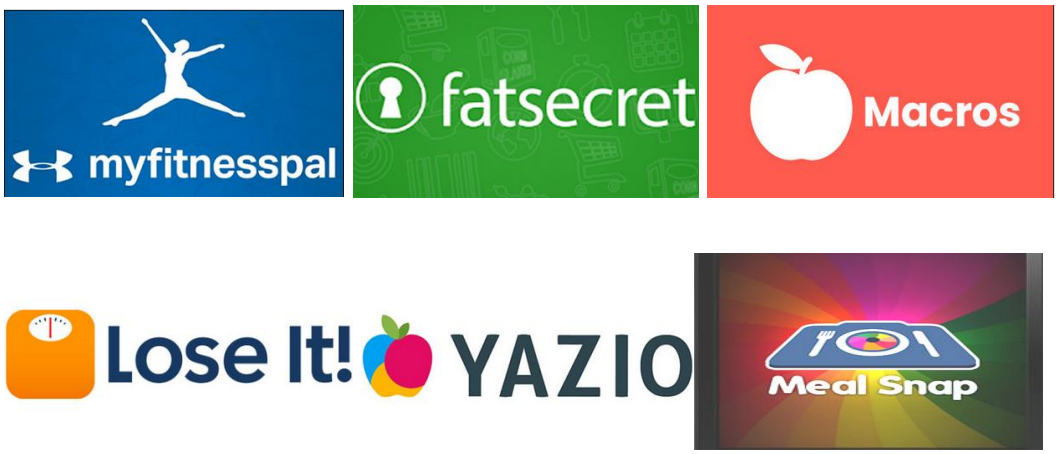
Lo que se pretende conseguir con este proyecto facilitar todo esto mediante una solución informática. Por ejemplo, estableciendo una gestión total del gimnasio, donde podamos consultar las clases, apuntarnos o desapuntarnos sin tener que ir personalmente. La facilidad que proporciona esta gestión, donde el monitor puede gestionar sus clases y usuarios sin tener que estar en el gimnasio y avisar de forma manual al usuario. Este es uno de los principales objetivos.

Debido a la necesidad de establecerse objetivos para acudir al gimnasio, he decidido añadir una funcionalidad novedosa. Gestionar un plan de nutrición para

alcanzar su objetivo sin necesidad de entrenador personal ni nutricionista. Este plan no trataría de una dieta cerrada durante unos meses/semanas, sino de unas recomendaciones sobre calorías, proteínas, grasas y carbohidratos que deben ingerirse día a día para satisfacer nuestro objetivo. Con este plan se persigue adquirir el hábito de comer de forma saludablemente. Por el contrario, no se pretende establecer una dieta estricta que desmotive a la persona o en la que encontremos alimentos que no sean del agrado, no sienten bien o incluso que no se puedan comer. Por lo tanto, este tipo de dieta no establece ningún número de comidas, tiempo entre comidas, ni ningún tipo de directiva ni *rest*ricciones. Solamente, al final del día, el resultado debe asemejarse lo máximo posible a lo establecido en el plan establecido. En definitiva, este sería el aspecto más novedoso del proyecto: incorporar un plan de entrenamiento individualizado, lo más ajustado posible, sin acudir a ningún nutricionista.

2.1.1. Estudio de soluciones existentes

Las aplicaciones que ahora mismo son utilizadas para el tema de seguir un registro de comida diario son las siguientes.



Estas aplicaciones te permiten llevar un contador de calorías en tu teléfono, conteniendo una amplia base de datos de alimentos. Se puede acceder a estos alimentos mediante el escaneo del código de barras y *poste*riormente introduces los gramos que vas a ingerir o has ingerido obteniendo como resultado las kilocalorías, proteínas, grasas y carbohidratos que tiene ese alimento.

Aspectos positivos que podemos observar en estas aplicaciones:

- Base de datos de alimentos: se pueden introducir directamente los alimentos mediante el código de barras e indicando los gramos que vas a consumir, a partir de los cuales obtienes kilocalorías, proteínas, grasas y carbohidratos.
- Son aplicaciones para móviles, por lo tanto son muy fáciles de usar.
- Se pueden definir tus kilocalorías para consumir a diario junto con el porcentaje de cada macro.

Incovenientes que se observan en estas aplicaciones:

- Objetivos: Para mí este es el mayor inconveniente puesto que si se marca un objetivo no trata de hacer una estimación durante un periodo de tiempo, como por ejemplo 2 semanas.
- No tienes ningún tipo de soporte como pueden ser los monitores de gimnasio, que pueden ayudarte en lo que necesites aportando sugerencias.

En conclusión, estas aplicaciones no son excluyentes de la nuestra, puede utilizarlas para introducir el registro de comida. Aunque te guíes por el plan de nutrición que hemos establecido en nuestra app. Además, nuestra app añade la gestión total de un gimnasio, que es bastante complejo.

2.2. Entrenamiento

En los siguientes apartados definiremos algunos conceptos básicos para sentar las bases sobre las que se desarrollará la solución propuesta. Se define el entrenamiento [2] como un conjunto de actividades planificadas con el objetivo de estimular y desarrollar las capacidades físicas progresivamente debido a las adaptaciones fisiológicas del organismo.

2.2.1. Terminología del entrenamiento

A continuación, se van a definir una serie de términos que van a servir de ayuda para comprender todo lo relacionado con el entrenamiento y para entender la estrategia tomada dependiendo de las necesidades del atleta:

- **Sesión de entrenamiento:** es el conjunto total de ejercicios hechos por el atleta en una ventana de tiempo cerrada. Se pueden hacer una o varias sesiones de entrenamiento en un día.
- **Repeticiones:** es el número de veces que efectuamos un movimiento.
- **Serie:** es una agrupación de repeticiones de un movimiento, separadas por tiempos de descanso entre ellas, ya sean del mismo o distinto ejercicio.
- **Grupo muscular:** son un conjunto de músculos que comparten la función que desempeñan en el organismo (estabilizadores, esfuerzos dinámicos, etc).
- **RM (Repetición Máxima):** es el número máximo de kilogramos que podemos levantar en un ejercicio determinado en un día en concreto con una técnica correcta. Varía de un día para otro.

2.2.2. Variables del entrenamiento

En el entrenamiento hay muchas variables a tener en cuenta para conseguir progresar adecuadamente. A continuación, se indican las más importantes:

- **Volumen de entrenamiento**. Es la variable más importante. En términos simp*list*as, se trata del número total de series y repeticiones hechas a lo largo de una sesión, semana, etc [1].

  Se suele tener muy en cuenta el grupo muscular para ajustar el número de ejercicios, series y repeticiones que hacemos a lo largo de una sesión y la semana, para garantizar un estímulo y recuperación óptimos a lo largo de toda la planificación.

  Podemos hacer referencia a dos conceptos básicos cuando hablamos de esta variable:

  - **MRV (Maximum Recoverable Volume):** máximo volumen recuperable o cantidad máxima de ejercicios, series y repeticiones que nuestro cuerpo es capaz de tolerar, teniendo en cuenta otros aspectos

como alimentación, intensidad, carga y cualesquiera otros conceptos que se explicarán más adelante.

- **MEV (Minimum Effective Volume):** volumen mínimo efectivo o cantidad mínima de ejercicios, series y repeticiones que nuestro cuerpo necesita para provocar un estímulo que lleve a las adaptaciones fisiológicas que buscamos.

VOLUME LANDMARKS	DESCRIPCIÓN
MV= Volumen de Mantenimiento	Mínimo volumen que necesitas para mantener tus ganancias
MEV = Volumen mínimo efectivo	Volumen mínimo a partir del cual se generan adaptaciones
MAV = Volumen máximo adaptativo	Volumen donde se generan las mejores adaptaciones
MRV = Volumen máximo recuperable	Máximo volumen del que te puedes recuperar. Sobrepasarlo constantemente = sobreentrenamiento

Ilustración 1: Volúmenes de entrenamiento y adaptaciones propuesto por Mike Israetel 2017 [2], [3]

- **Intensidad:** se entiende por intensidad el grado de esfuerzo desarrollado a la hora de hacer cada repetición de un ejercicio. Esta variable va íntimamente ligada a los kilogramos levantados, es decir, el porcentaje total sobre la RM [4], [5].

Es importante introducir un concepto muy relacionado con la intensidad y que ha tenido buena aceptación dentro de la comunidad en los últimos años de cara a la programación del entrenamiento de ciertos tipos de atletas, generalmente atletas de fuerza como los powerlifters.

- **RPE (Rate of Perceived Exertion)**. Es la escala del esfuerzo máximo llevado a cabo en una serie.

Ilustración 2: : Escala de esfuerzo máximo (RPE) [4], [5]

- **Frecuencia:** es el número de veces que entrenamos un grupo muscular en un tiempo determinado. Generalmente, se hace referencia a la frecuencia semanal, pero también podríamos hablar de frecuencia diaria.
- **Tiempos de descanso:** el tiempo de descanso dependerá directamente de la intensidad de cada serie.
- **Densidad:** se define la densidad de entrenamiento como el cociente entre el tiempo total de trabajo entre el tiempo total que dura la sesión.

El control de todas estas variables es indispensable para garantizar una buena evolución del atleta a lo largo de la preparación. A su vez, éstas estarán condicionadas dependiendo de cada persona, sus posibilidades y objetivos.

2.3. Nutrición

Según la OMS, la nutrición es la ingesta de alimentos en relación con las necesidades del organismo. Para mantener una buena nutrición, debemos realizar una dieta suficiente y equilibrada [6].

2.3.1. Ecuación de Harris-Benedict

La ecuación de Harris-Benedict es una ecuación empírica para estimar el metabolismo basal de una persona en función de su peso corporal, estatura y edad. Además de estos factores, se utilizan otros de actividad física para calcular la recomendación de consumo diario de calorías para un individuo.

Hombres	TMB = 66.5 + (13.8 x peso en kg) + (5 x altura en cm) - (6.8 x edad en años)
Mujeres	TMB = 655 + (9.6 x peso en kg) + (1.85 x altura en cm) - (4.7 x edad en años)

Ilustración 3: Fórmula de Harris-Benedict [7]

Para calcular exactamente la ingesta diaria de calorías recomendada de una persona para mantener su peso actual, utilizaremos la siguiente ilustración.

Poco o ningún ejercicio	Calorías diarias necesarias = TMB x 1,2
Ejercicio ligero (1-3 días a la semana)	Calorías diarias necesarias = TMB x 1,375
Ejercicio moderado (3-5 días a la semana)	Calorías diarias necesarias = TMB x 1,55
Ejercicio fuerte (6-7 días a la semana)	Calorías diarias necesarias = TMB x 1,725
Ejercicio muy fuerte (dos veces al día, entrenamientos muy duros)	Calorías diarias necesarias = TMB x 1,9

Ilustración 4: Factor de actividad [7]

2.3.2. Cálculo de macros

Una vez se calculen las calorías a consumir, se deberá establecer a cuántos gramos de proteína, grasa e hidratos de carbono se traduce esto. Por norma general, se trabaja con porcentajes muy estandarizados basados en la evidencia. La proteína suele ser la menos variable dentro de estos macros.

Hay diversos estudios [8, 9, 10, 11, 12] que confirman que, por norma general, un consumo de 1.6-2 gr/kg de peso corporal es más que suficiente para maximizar el proceso de síntesis proteica.

Por otra parte, para los hidratos de carbono, [8] hay evidencia de que un consumo inferior al 32% podría implicar pérdidas de masa muscular, con lo cual siempre se suelen rondar consumos superiores, desde 3gr/kg de peso hasta 6-7gr/kg de peso.

Por último, en cuanto a las grasas, no hay que descuidar su consumo, ya que intervienen en muchos procesos del organismo, como por ejemplo el control hormonal; un bajo consumo, además, podría tener graves consecuencias a medio plazo.

Generalmente, el consumo de proteínas se suele fijar en 1.6-2gr/kg de peso. Se juega con los porcentajes de hidratos y grasas para cuadrar las macros.

En este caso, hemos usado unos porcentajes comunes que garantizan esos consumos mínimos en cada uno de los macronutrientes para cuidar todos los procesos de los que dependen estos nutrientes: síntesis proteica, protección de la masa magra, regulación hormonal, etcétera.

2.3.3. Definición de conceptos

Nuestro cuerpo es básicamente un sistema como cualquier otro, entonces consecuentemente el sistema se basa en el primer principio de la termodinámica (*Energía que entra - Energía que sale = Variación de energía del sistema*). La kilocaloría es la unidad de energía que hace que nuestro sistema funcione. Según este principio podemos obtener diferentes escenarios:

- **Variación del sistema > 0:** se produce cuando ingerimos más kilocalorías de las que usamos día a día. Este estado en el que se encuentra nuestro cuerpo es llamado *superávit calórico*. Debido a este superávit calórico, tu cuerpo debe almacenar esta energía a través de la síntesis de nuevos tejidos, ya sean musculares o grasos.

- **Variación del sistema < 0:** se produce cuando ingerimos menos kilocalorías de las que utilizamos día a día, lo que provoca en nuestro cuerpo el llamado *déficit calórico*. Dada esta circunstancia, nuestro cuerpo comenzará a perder peso, ya que tendrá que extraer este déficit de energía de la masa corporal mediante la oxidación de grasas, aunque también sacrificaremos un porcentaje de músculo.
- **Variación del sistema = 0:** se produce cuando ingerimos las mismas kilocalorías que utilizamos durante el día, lo que proporciona a nuestro

cuerpo un estado llamado mantenimiento *calórico*. Debido a esto, nuestro cuerpo no sufriría ningún tipo de cambio en relación al peso.

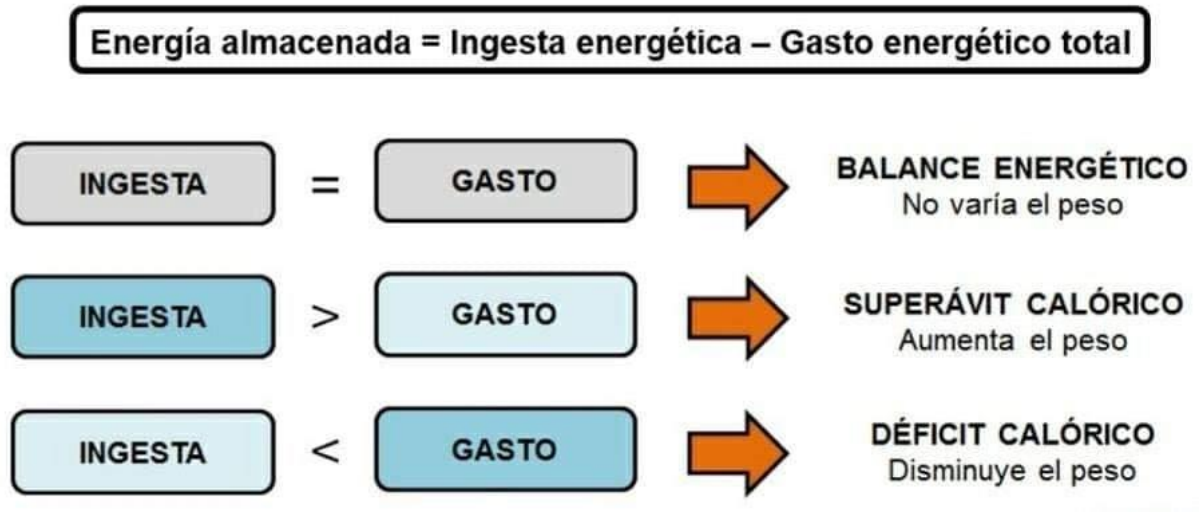


Ilustración 5: Energía almacenada [9]

2.3.4. Protocolo a seguir

Como a la hora de la definición es mucho más difícil de controlar y llevar a cabo para cualquiera usuario, vamos a seguir un plan específico. Si el usuario tiene como objetivo, por ejemplo, la ganancia muscular o el mantenimiento, nosotros le proporcionaremos unas Kcals basadas en la fórmula en el caso segundo; por lo tanto, no se añadirán Kcals de más, aunque, contrariamente, para un superávit calórico sí que añadiríamos unas 300 Kcals de más al día, teniendo en cuenta que este dato es meramente orientativo, puesto que el usuario, en sus objetivos, podría pretender ganar más peso por mes, en cuyo caso debería añadir más Kcals aún por día o, en el caso contrario, tener como objetivo ganar menos peso, por lo que, consecuentemente, debería recortar el consumo de Kcals por día. El proceso a seguir para establecer el protocolo de definición a cada usuario es el siguiente:

1. Con los datos que nos ha proporcionado, intentamos obtener el metabolismo basal del usuario, utilizando la fórmula de Harris-Benedict, y añadiendo además un coeficiente, el de actividad, mayor cuanto mayor sea el entrenamiento del usuario; se ilustra este coeficiente en la ilustración 7.
2. Una vez pasados 15 días, pedimos al usuario que nos indique su peso actual. Este ajuste lo hacemos en torno al factor de actividad. Una vez recibimos el

peso del usuario, en el caso de que su objetivo sea la pérdida de grasa hay dos opciones:

a) El usuario pesa más ahora que antes. Por lo tanto, suponiendo que ha seguido nuestro plan de nutrición, hay que arreglar el factor de actividad, ya que debe ser menor que el que estamos utilizando. Cada 0.5 kg que el usuario haya subido, 0.05 que se decrementa el factor actividad. Esto alega que el usuario está realizando un entreno de menor intensidad y que por lo tanto hay un menor gasto de energía. Este arreglo en el factor lo que implica es una bajada de Kcals en el protocolo para asemejarse lo máximo posible al usuario [10].
b) El usuario pesa menos que antes, pero la variación de peso ha sido de 1.5 kg o mayor. Esta pérdida de peso es excesiva, ya que tratamos de que el peso a perder durante una semana sea en torno a 0.5 kg. Por lo tanto, tratamos de arreglarlo igual que en el caso anterior: cada 1.5 kg que el usuario haya bajado, 0.05 que se le añade al factor de actividad. Esto implica un aumento de Kcals, que justificamos que es debido a que su actividad física es mucho más intensa [10].

3. Seguidamente, de esta revisión del metabolismo basal comenzamos con el protocolo de definición. Este protocolo hemos decidido realizarlo durante 12 semanas, 3 meses aproximadamente, ya que, este tiempo es la duración media de una definición para un usuario intermedio de gimnasio. La primera semana, empezamos con un déficit de 300 Kcals sobre su normo calórica que hemos obtenido en el paso anterior. Las demás 11 *rest*amos progresivamente 30 Kcals de la semana anterior. Con esto tratamos de que la pérdida de peso sea progresiva, manteniendo nuestro entrenamiento: aproximadamente 0.5 kg por semana. El *rest*ar Kcals por semana se debe a que si su cuerpo pesa 80 kilos y tiene una normo calórica de 2500, mientras que, si su cuerpo pesa actualmente 78 kilos, esta normo calórica pasa a ser unas 2400. Esto es básicamente el motivo de este decremento de Kcals semanalmente.

¿Por qué realizamos un déficit de 300 Kcals? Para perder 1 kilo de grasa necesitamos un déficit calórico de 1000 Kcals diario. Como nuestro objetivo es perder en torno a 300-400 gramos de grasa semanalmente, establecemos un déficit calórico de 300 Kcals.

2.4. Valoración del atleta

Para que el entrenador sea capaz de trazar una buena estrategia, debe conocer lo máximo posible al atleta. Para ello, es necesaria una valoración para poner en contexto la preparación. Hay que plantear muchas preguntas, entre ellas la edad del atleta, si tiene enfermedades médicas que puedan afectar a su rendimiento, o si ha practicado algún deporte previamente; también cuáles son sus hábitos diarios o en qué se basa su alimentación, si sufre de estrés por el trabajo y, ligado a esto, qué disponibilidad horaria tiene para p*rest*arle al entrenamiento Todas estas preguntas servirán al entrenador para plantear la mejor estrategia posible.

2.4.1. Concepto de adherencia

Antes de empezar a explicar por qué esas cuestiones son importantes, se va a desarrollar otro concepto relevante: la adherencia. Por adherencia [18] entendemos la capacidad del atleta de adaptarse al plan nutricional y físico que el entrenador ha diseñado. Es un concepto muy básico, pero de gran importancia. La gran mayoría de fracasos de las personas que empiezan un plan nutricional o de entrenamiento ocurre porque es un cambio tan brusco que acaba produciendo estrés y acaban por dejarlo. Es por eso que es importante conocer al atleta y fijar unos objetivos prácticos que hagan que poco a poco cambie su estilo de vida.

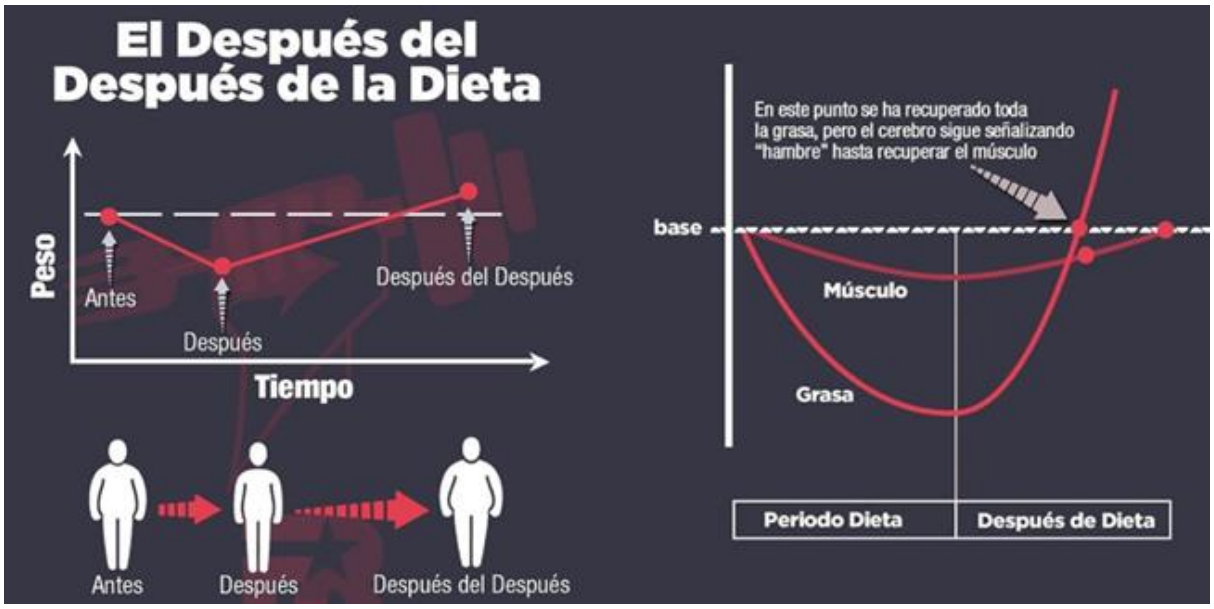


Ilustración 6: Infografía de mala adherencia con una dieta por Marcos Vázquez [11]

2.4.2. Casos prácticos

La situación de cada persona es distinta y hay que tener en cuenta su estilo de vida para conseguir que se adhiera correctamente al plan. A continuación, se van a exponer unos casos prácticos para dejar claro todo lo explicado anteriormente y constatar que hay que tener en cuenta hasta el más mínimo detalle en una preparación.

Caso práctico 1

Edad	Estatura	Peso	Enfermedad	Alimentación	¿Ha practicado/practica algún deporte?	¿Trabaja?	Objetivo
28	180 cm	105 kg	No	Mayoritariamente comida rápida y procesada	No	No	Perder grasa

Estamos ante un caso de una persona joven, sedentaria, con sobrepeso y problemas de alimentación cuyo objetivo es perder grasa, con las ventajas de que no tiene ninguna enfermedad que pueda limitar su entrenamiento; no trabaja.

Las primeras semanas del plan de entrenamiento se centrarán en hacer que la persona pase de tener un estilo de vida sedentario a una ligera actividad con una intensidad y volúmenes de entrenamiento bajos, que se irán incrementando conforme la persona se adapte tanto física como psicológicamente. Por ejemplo, tres días de

entrenamiento en los que se trabaje con ejercicios multiarticulares como una sentadilla, flexión de brazos, etc., que hagan que esa persona se mueva completamente, y otros dos días a la semana para salir a andar entre 5-10 km.

En cuanto al plan nutricional, nos basaremos en la ecuación de Harris-Benedict para obtener una estimación inicial de la normocaloría de esta persona, y se intentará, adaptando la cantidad de alimentos para cuadrar las calorías, que cambie paulatina, periódicamente, su alimentación, semana a semana.

Con tan solo hacer que esta persona se mueva un poco y corrija su alimentación estaremos consiguiendo que mejore su condición física a la vez que su composición corporal, lo cual permitirá que poco a poco el entrenador planifique entrenamientos más exigentes y con un control más exacto sobre las calorías.

Caso práctico 2

Edad	**Estatura**	**Peso**	**Enfermedad**	**Alimentación**	**¿Ha practicado/practica algún deporte?**	**¿Trabaja?**	**Objetivo**
24	180 cm	83 kg	No	Mayoritariamente sana	Si, 6 años de experiencia en entrenamiento de fuerza	Si	Preparar unas oposiciones de policía

En este otro caso tenemos a una persona joven, habituada al entrenamiento de fuerza y que está preparando unas oposiciones de policía mientras trabaja.

Partimos entonces de base con una persona que va a tener bastante estrés debido al trabajo y al estudio y que le puede afectar a la calidad del sueño y por tanto, repercutir negativamente en la recuperación y en los entrenamientos, incluso puede que llegar a una lesión por falta de descanso o desconcentración.

En este caso, el entrenador deberá tener mucho cuidado con el volumen e intensidad de los entrenamientos, así como la carga, porque si no se tiene en cuenta

el estrés psicológico de esa persona a la hora de programar un entrenamiento pesado, quizá ese día esté desconcentrado y cansado y se lesione fallando el levantamiento.

En cuanto a la alimentación, el propio entrenador debe darse cuenta que debe partir de una normocalórica o superávit ligero para asegurar que se mantiene la masa muscular y que la alimentación no suponga un plus para el estrés psicológico.

A priori parecería que este segundo caso es más fácil de llevar que el primero, puesto que se parte de una base de que la persona tiene una buena alimentación y lleva años entrenando, pero factores tan simples como si está trabajando o está preparando unas oposiciones pueden cambiarlo radicalmente todo.

2.5. Propuesta de solución

Lo que vamos a tratar en este proyecto es dar un soporte personalizado a los clientes de un gimnasio, los cuales podrán gestionar su suscripción, además de que podrán obtener un plan de entrenamiento para alcanzar su objetivo. Todo esto, sin necesidad de ir a un dietista, nutricionista o entrenador personal. Además, se proporcionará una gestión total a los monitores del gimnasio. Los monitores van a poder crear sus clases, editarlas si les ha surgido algún tipo de inconveniente e incluso borrarlas. Respecto a la gestión de clientes, los monitores van a poder ver su información, borrarlos si estos ya no se encuentran apuntados al gimnasio e incluso poder ver la *lista* de participantes de sus clases.

Para facilitar estas funcionalidades me basaré en el uso de una aplicación basada en tecnologías web. Una aplicación web permite poder ser utilizada por cualquier cliente, independientemente de su sistema operativo, mediante el navegador. Otra característica esencial es la facilidad para actualizar y compartir aplicaciones sin instalar ningún tipo de software. La principal ventaja es la posibilidad de acceder desde cualquier plataforma sin tener que instalar nada, solo con el navegador que cualquier plataforma dispone. Esto implica que la aplicación deba ser *responsive,* ya que debe ser accesible para móviles también.

2.6. Historias de usuario

A continuación, se indicarán las historias de usuario realizadas para la especificación de requisitos. Además, tras un pequeño estudio sobre las GASP[1] que existen a día de hoy para la estimación de las tareas y planificación de esfuerzo he llegado a la conclusión de que voy a asignar a cada historia de usuario una talla de camiseta [20, 1]. Aunque, a su vez, también tiene asignado un número de puntos de historia equivalentes a la serie de Fibonacci. He establecido que cada día corresponde a 4 horas de trabajo [12].

Como podemos observar cada historia de usuario se compone de:

- Título: título de la historia de usuario junto con su numeración para tener un fácil acceso.
- Descripción: descripción breve de la historia de usuario.
- Puntos de historia: puntos que representa un valor, que será utilizado para estimar el tamaño de una tarea. En este caso nos hemos decantado por establecer estos puntos según las tallas de camiseta.
- Criterios de aceptación: criterios a satisfacer para completar las historias de usuario, de manera correcta.

Una vez establecida la historia de usuario, pasamos a la descomposición. La descomposición trata de fragmentar todo lo posible las tareas a realizar para cumplir cada historia de usuario. He decidido desglosarlo en parte del cliente (*Frontend*) y parte del servidor (*Backend*), ya que creo que cuando se cuenta con varios equipos de trabajo es más fácil repartir las tareas cuando ya están separadas se encuentran desglosadas.

Talla	**Puntos de Historia**	**Días**
XS	1	0.5
S	2	1
M	3	1.5
L	5	2.5
XL	8	4

[1] GASP: Serie de herramientas que sin ser parte de las reglas de Scrum, son adoptadas habitualmente por los equipos Scrum para llevar a cabo la estimación de tareas y la planificación del esfuerzo.

Título de HU01	*Login* Usuario
Descripción	Como cliente quiero loguearme para acceder a la aplicación.
Puntos de Historia	XL
Criterios de aceptación	● Existencia de una BBDD donde se puedan guardar mis datos para poder acceder a ellos cuando se necesiten. ● Existencia de un formulario de acceso para indicar mi correo y contraseña

Descomposición(*Backend*):

1. Crear prototipo de *Backend* inicial
2. Configuración del contenedor docker para la BBDD
3. Configuración de conexión de *Backend* con el contenedor
4. Añadir servicio de *Login* al *Backend*.
5. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

6. Crear la página de aterrizaje.
7. Crear un botón para el acceso a usuarios.
8. Formulario de *Login* con correo y contraseña.
9. Crear una página de bienvenida para usuarios.

Título de HU02	Registro
Descripción	Como cliente quiero registrarme para poder acceder a la aplicación con mis datos.
Puntos de Historia	L
Criterios de aceptación	● Existencia de una BBDD donde se puedan guardar mis datos para poder acceder a ellos cuando se necesiten. ● Existencia de un formulario para establecer mis datos.

Descomposición(*Backend*):

1. Añadir servicio de Registro al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón de acceso al registro
4. Formulario de Registro con datos a introducir.
5. Crear botón para enviar los datos del registro.

Título de HU03	Ver datos personales
Descripción	Como usuario quiero ver los datos personales que indiqué en el registro.
Puntos de Historia	M
Criterios de aceptación	● Existencia de una BBDD donde se puedan guardar mis datos para poder acceder a ellos cuando se necesiten. ● Existencia de una opción que permita acceder a consultar los datos del cliente.

Descomposición(*Backend*):

1. Añadir servicio de Obtener Usuario al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en página de bienvenida de usuarios para acceder a mi perfil.
4. Desarrollar vista para visualizar mis datos.

Título de HU04	Editar datos personales
Descripción	Como cliente quiero editar mis datos personales que indiqué en el registro.
Puntos de Historia	M
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a mis datos de registro y *poste*riormente guardar los datos modificados.

	• Existencia de una opción que permita mostrar los datos personales, con posibilidad de editar los campos de mi registro.

Descomposición(*Backend*):

1. Añadir servicio de Actualizar Usuario al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en la página de bienvenida de usuarios para editar perfil.
4. Desarrollar vista para visualizar mis datos y poder editarlos.
5. Crear botón para mandar los datos actualizados.

Título de HU05	Ver clases disponibles
Descripción	Como cliente quiero ver las clases que hay disponibles
Puntos de Historia	M
Criterios de aceptación	• Existencia de una BBDD donde se pueda acceder a la *lista* de clase que hay creadas.

Descomposición(*Backend*):

1. Añadir servicio de Obtener Clases Disponibles al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

1. Crear botón en la página de bienvenida de usuarios para acceder a las clases disponibles.
2. Desarrollar vista para visualizar las clases disponibles.

Título de HU06	Ver clases apuntadas
Descripción	Como cliente quiero ver las clases a las que estoy apuntado
Puntos de Historia	M

Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a la *lista* de clase que hay creadas. ● Existencia de una opción que permita mostrar una *lista* de clases a las que está apuntado.

Descomposición(*Backend*):

1. Añadir servicio de Obtener Clases Apuntadas al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en página de bienvenida de usuarios para acceder a las clases apuntadas.
4. Desarrollar vista para visualizar las clases apuntadas.

Título de HU07	Ver información de las clases (cliente)
Descripción	Como cliente quiero poder ver información más detallada de las clases disponibles.
Puntos de Historia	M
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a la información de la clase. ● Existencia de una opción que permita acceder a la información más detallada de la clase.

Descomposición(*Backend*):

1. Añadir servicio de Obtener Clase al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en la vista de *list*ado de clases disponibles, para ver información más detallada sobre la clase.
4. Desarrollar una vista para visualizar información más detallada sobre las clases.

Título de HU08	● Apuntarse a clase
Descripción	● Como cliente quiero poder apuntarme a las clases disponible
Puntos de Historia	● M
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a la *lista* de clase que hay creadas. ● Existencia de un botón que me apunte a la clase

Descomposición(*Backend*):

1. Añadir servicio de Apuntar Usuario a clase al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en vista de la información más detallada para apuntarme a la clase

Título de HU09	Desapuntarse de clase
Descripción	Como usuario quiero poder desapuntarse a las clases que estoy apuntado
Puntos de Historia	M
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a la *lista* de clase que estoy apuntado. ● Existencia de un botón que desapunte de la clase

Descomposición(*Backend*):

1. Añadir servicio de Desapuntar Usuario de clase al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en vista de la información más detallada para desapuntarse a la clase

Título de HU10	*Logout* Usuario
Descripción	Como cliente quiero poder cerrar sesión para salir de la aplicación.
Puntos de Historia	S
Criterios de aceptación	● Existencia de un botón que cierre la sesión

Descomposición(*Backend*):

1. Añadir servicio de Cerrar Sesión al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón para poder cerrar sesión, una vez esté logueado

Título de HU11	Entrenamiento
Descripción	Como cliente quiero poder obtener un plan de nutrición según mi objetivo.
Puntos de Historia	L
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a los datos del cliente. ● Existencia de una opción que permita mostrar el plan de nutrición a seguir.

Descomposición(*Backend*):

1. Añadir servicio de Obtener Entrenamiento al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en la página de bienvenida del usuario, para poder acceder al entrenamiento del usuario.

Título de HU12	*Login* Monitor
Descripción	Como monitor quiero loguearme para acceder a la aplicación.
Puntos de Historia	M
Criterios de aceptación	● Existencia de una BBDD donde se puedan guardar mis datos para poder acceder a ellos cuando se necesiten. ● Existencia de un formulario de acceso para indicar mi correo y contraseña

Descomposición(*Backend*):

1. Añadir servicio de *Login* para el monitor al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear un botón para el acceso a monitores.
4. Formulario de *Login* con correo y contraseña.
5. Crear una página de bienvenida para monitores.

Título de HU13	*List*ar clases a impartir
Descripción	Como monitor quiero poder ver una *list*a de las clases a impartir.
Puntos de Historia	M
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a las clases a impartir. ● Existencia de una opción que permita acceder a la *list*a de clases.

Descomposición(*Backend*):

1. Añadir servicio de *List*ar Clases a impartir por el monitor al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en página de bienvenida del monitor, para acceder al *list*ado de clases a impartir
4. Crear vista de *list*ado de clases a impartir

Título de HU14	Crear clase
Descripción	Como monitor quiero poder crear clases.
Puntos de Historia	L
Criterios de aceptación	● Existencia de una BBDD donde se pueda guardar los datos de la clase creada. ● Existencia de un formulario para establecer la información de la clase.

Descomposición(*Backend*):

1. Añadir servicio de Crear Clase al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en la vista de clases a impartir por el monitor, para acceder al formulario de crear nueva clase.
4. Formulario de crear clase
5. Crear botón, en el formulario de crear clase, para enviar los datos de la clase a crear.

Título de HU15	Ver información de las clases (monitor)
Descripción	Como monitor quiero poder ver información más detallada de las clases a impartir.
Puntos de Historia	S
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a la información de la clase. ● Existencia de una opción que permita mostrar la información más detallada de la clase.

Descomposición(*Backend*):

1. Añadir servicio de Obtener Clase al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en la vista de clases a impartir por el monitor, para acceder a información más detallada de las clases.
4. Crear vista para mostrar la información de la clase.

Título de HU16	Editar clase
Descripción	Como monitor quiero editar clases que han sido creadas por mí.
Puntos de Historia	M
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a los datos de la clase y *post*eriormente guardar los datos modificados. ● Existencia de una opción que permita mostrar los datos de la clase, con posibilidad de editar los campos.

Descomposición(*Backend*):

1. Añadir servicio de Actualizar Clase al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en la vista de clases a impartir por el monitor, para editar la información de las clases.
4. Crear vista para mostrar la información de la clase actual, pudiendo editar cualquier campo.
5. Crear botón para mandar la información actualizada de la clase.

Título de HU17	Borrar clase

Descripción	Como monitor quiero borrar las clases que tengo asignadas para impartir.
Puntos de Historia	M
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a las clases a impartir. ● Existencia de una opción que permita borrar una clase de la *list*a de clases a impartir.

Descomposición(*Backend*):

1. Añadir servicio de Borrar Clase al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en la vista de clases a impartir por el monitor, para borrar las clases.

Título de HU18	*Logout* Monitor
Descripción	Como monitor quiero poder cerrar sesión para salir de la aplicación.
Puntos de Historia	S
Criterios de aceptación	● Existencia de un botón que cierre la sesión

Descomposición(*Backend*):

1. Añadir servicio de Cerrar Sesión al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón para poder cerrar sesión, una vez esté logueado

Título de HU19	*List*ar participantes

Descripción	Como monitor quiero obtener una *list*a de participantes que hay apuntados a una clase.
Duración:	M
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a los usuarios que hay apuntados. ● Existencia de una opción que permita mostrarla *list*a de participantes.

Descomposición(*Backend*):

1. Añadir servicio de *List*ar Participantes de una clase al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en la vista de clases a impartir por el monitor, para acceder a la *list*a de participantes de las clases.
4. Crear vista para mostrar la *list*a de participantes de la clase.

Título de HU20	Borrar participante
Descripción	Como monitor quiero obtener una *list*a de usuario que hay ahora mismo activos en el gimnasio.
Puntos de Historia	S
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a los usuarios que hay creados. ● Existencia de una opción que permita mostrar la *list*a de usuarios.

Descomposición(*Backend*):

5. Añadir servicio de Borrar Usuario de una clase al *Backend*.
6. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio.

Descomposición(*Frontend*):

7. Crear botón en la vista del *list*ado de participantes, para borrar participantes de las clases.

Título de HU21	*List*ar usuarios
Descripción	Como monitor quiero obtener una *list*a de usuario que hay ahora mismo activos en el gimnasio.
Puntos de Historia	M
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a los usuarios que hay creados. ● Existencia de una opción que permita mostrar la *list*a de usuarios.

Descomposición(*Backend*):

1. Añadir servicio de *List*ar Usuarios creados al *Backend*.
2. Añadir tests para comprobar la correcta funcionalidad del nuevo servicio

Descomposición(*Frontend*):

3. Crear botón en la página de bienvenida del monitor, para acceder a la *list*a de usuarios que hay creados.
4. Crear vista para mostrar la *list*a de usuarios creados.

Título de HU22	Ver información de usuarios
Descripción	Como monitor quiero ver la información de mis usuarios.
Puntos de Historia	S
Criterios de aceptación	● Existencia de una BBDD donde se pueda acceder a la información de los usuarios. ● Existencia de una opción que permita mostrar la información de los usuarios.

Descomposición(*Frontend*):

1. Crear botón en la vista del *list*ado de usuarios creados, para acceder a la información más detallada del usuario.
2. Crear vista para mostrar la *list*a de usuarios creados.

Título de HU23	Borrar usuario

Descripción	Como monitor quiero borrar los usuarios que no estén apuntados al gimnasio.
Puntos de Historia	S
Criterios de aceptación	● Existencia de una opción que permita eliminar el usuario de la BBDD.

Descomposición(*Frontend*):

1. Crear botón en la vista del *list*ado de usuarios creados, para borrar al usuario que no esté apuntado al gimnasio.

2.7. Planificación inicial de tareas

Inmediatamente después de haber obtenido las historias de usuario del cliente junto con su descomposición, pasamos a realizar una planificación exacta mediante un modelo iterativo. Previamente realizaremos una tabla donde agruparemos las historias en categorías y estableceremos sus respectivas prioridades, para las que la prioridad máxima es 5, y la duración se indicará en días. He establecido que la duración en horas de cada día de trabajo sería de 4 horas y se trabajaría de lunes a viernes.

Categorías	HU	Prioridad	PH	Duración(Días)
Usuario	*Login* - HU01	5	8	4
	Registro - HU02	5	5	2.5
	Ver mis datos - HU03	3	3	1,5
	Editar mis datos - HU04	3	3	1,5
	Logout – HU10	3	2	1
	Entrenamiento - HU11	4	5	2.5
	Ver clases Apuntadas – HU06	3	3	1,5

	Ver clases disponibles - HU05	3	3	1,5
Clase	Crear clase – HU14	5	5	2.5
	Editar clase – HU16	3	3	1,5
	Borrar clase – HU17	3	3	1.5
	Ver información de las clases – HU07	2	5	2.5
	Apuntarse a clase - HU08	4	3	1.5
	Desapuntarse de clase - HU09	4	3	1.5
	*List*ar Participantes – HU19	2	3	1,5
	Borrar participante – HU20	2	2	1
Monitor	*Login* – HU12	5	3	1,5
	Logout – HU10	3	2	1
	*List*ar clases a impartir – HU13	3	3	1,5
	Ver información de usuarios – HU22	2	2	1
	*List*ar usuarios – HU21	2	3	1.5
	Borrar usuario – HU23	2	2	1

A continuación, muestro una tabla de las tareas a realizar durante el desarrollo del trabajo de fin de grado a modo de resumen:

Tareas	**Días**
Categoría Usuario	17
Categoría Clase	15,5
Categoría Monitor	8.5
Estudio de Angular	8
Estudio de entrenamiento	12
Redacción de documentación	20

Una vez finalizada la estimación de tiempo, pasamos a la organización de iteraciones.

Iteración	**Historia de usuario**	**Puntos de Historia**
Iteración 1	*Login* (Usuario) - HU01	8
	Registro (Usuario) - HU02	5
	Ver mis datos - HU03	3
	Editar mis datos - HU04	3
	Logout - HU10	2
	Entrenamiento - HU011	5
		Total: 26
	Crear clase - HU14	5
	Editar clase - HU16	3
	Borrar clase - HU17	3
	Ver información de las clases - HU07	5

Iteración 2	Apuntarme a clase - HU08	3
	Desapuntarme de clase - HU09	3
	*List*ar Participantes - HU19	3
	Borrar participante - HU20	2
		Total: 27
Iteración 3	*Login* (Monitor) - HU12	3
	Logout (Monitor) - HU10	2
	*List*ar usuarios - HU21	3
	Ver información de usuarios - HU22	2
	Borrar usuario - HU23	2
	Ver clases Apuntadas - HU06	3
	Ver clases disponibles - HU05	3
	*List*ar clases a impartir - HU13	3
		Total: 23

A continuación, adjunto el diagrama de Gantt resultante de esta planificación.

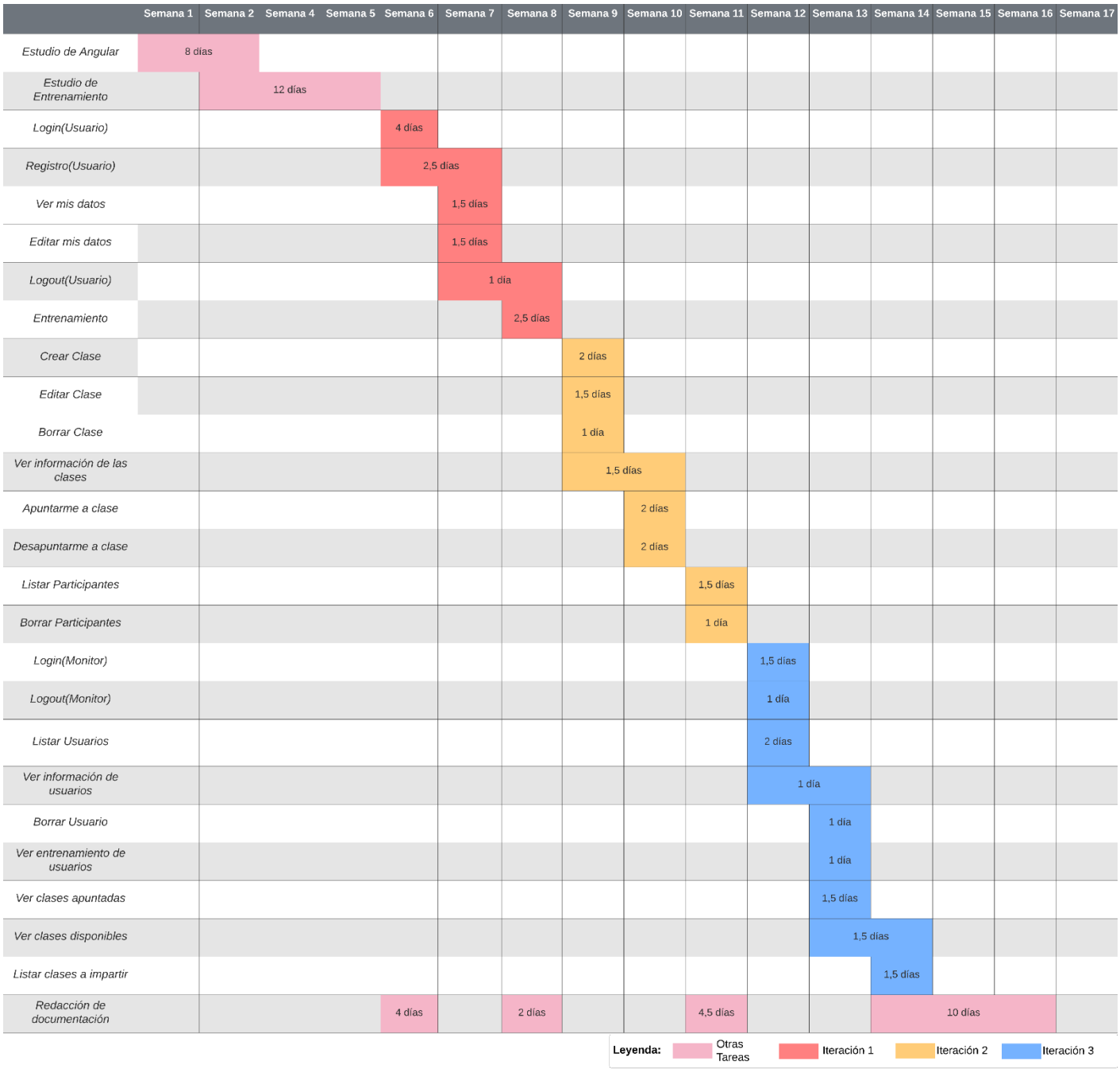


Ilustración 7: Diagrama de Gantt

2.8. Estudio de viabilidad

Una vez establecidas las historias de usuario y desarrollada la planificación inicial, obtenemos la duración del proyecto. La duración total es de 81 día, como hemos establecido que cada día son 4 horas, obtenemos el resultado de 324 horas. Para realizar el estudio de viabilidad hemos sacado de indeed [13] la media del sueldo

para Desarrolladores java en España. Para estudiar la viabilidad del desarrollo de este proyecto vamos a precisar de una tabla a modo de resumen.

Costes Software	Angular	0€
	Visual Code	0€
	Eclipse	0€
	Docker	0€
	SpringBoot	0€
Costes Personal	Desarrollador Java	4212€
Costes Desarrollo	Ordenador portátil	0€
	Conexión a internet	81€

Resultado total: 4293€

El coste 0 del software es debido a que es software libre y el cose 0 del portátil es debido a que ya disponía de él. A continuación, desgloso el resultado de los costes del desarrollador Java.

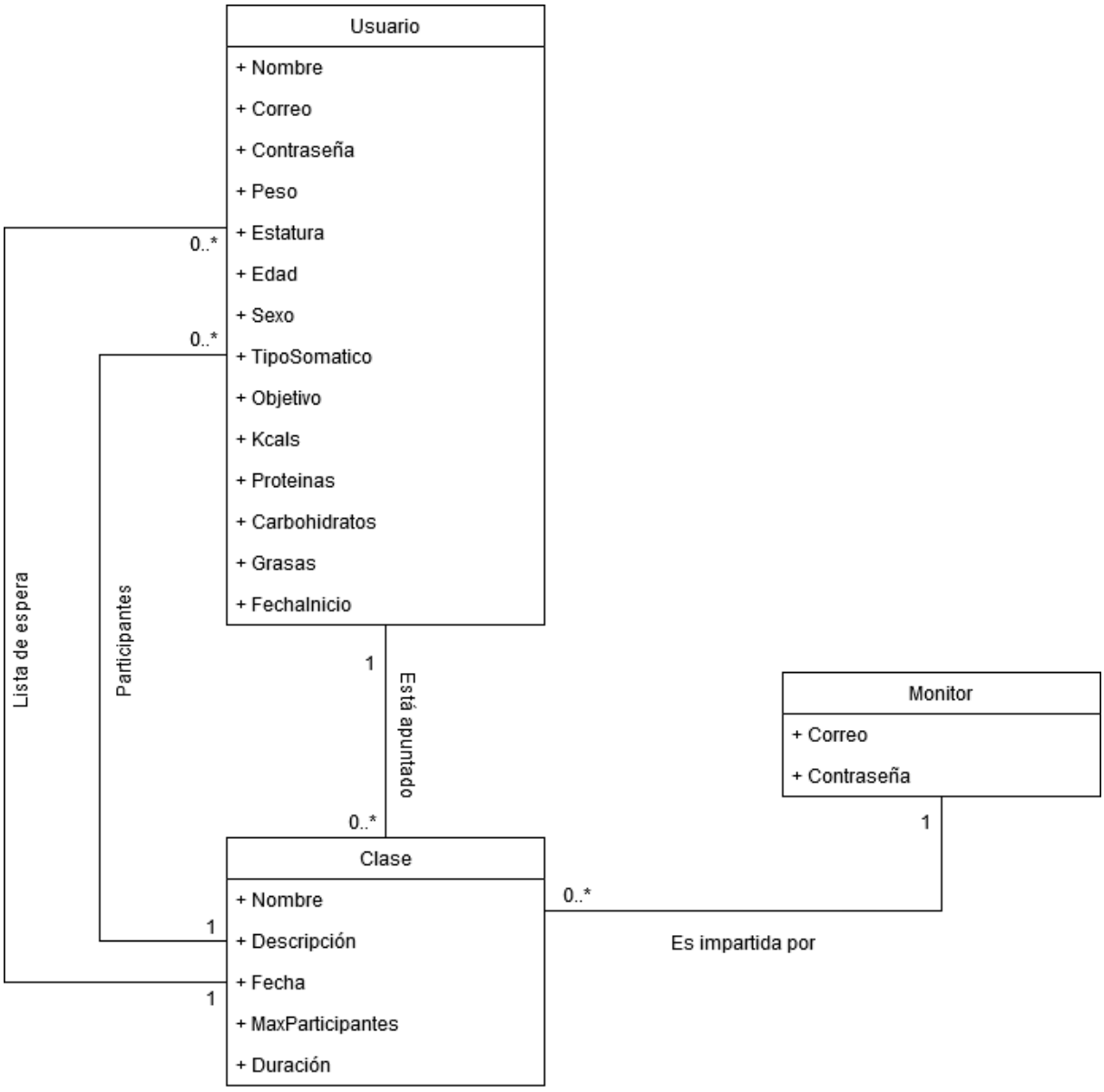
Precio por hora	Número de horas	Resultado
13	324	4212 €

El precio por hora resulta de dividir el salario al año entre los meses (en este caso son 14 debido a las pagas extras) y finalmente el resultado entre 8 que son las horas que se trabaja.

El número de horas resulta de *rest*ar las horas que se le han dedicado a los estudios y redacción del documento.

2.9. Modelo de dominio

Dentro del modelo del dominio, he decidido utilizar el diagrama de clases conceptuales, donde mostraremos las clases del sistema, sus atributos y sus relaciones.



A continuación, pasamos a explicar el diagrama:

Monitor: Es una clase sencilla, con solo 2 atributos, la única relación que tiene es con Clase, que indica las clases que impartiría el Monitor.

Clase: Como atributos sólo tiene nombre, descripción, fecha que se impartirá la clase, maxParticipantes que es el aforo máximo y la duración. Consta de 2 relaciones con usuario, una relación trata de la *lista* de participantes de la clase y la otra relación sería la *lista* de espera de esta.

Usuario: Como atributos más característicos estarían, TipoSomatico que es la forma física que tiene el cuerpo humano, esto depende de varios factores, Objetivo

que es el objetivo que establece en el gimnasio (Ganar Peso, Perder Peso), Kcals que debe ingerir el usuario para lograr su objetivo, se obtiene mediante una fórmula con los datos anteriores introducidos, Proteínas, Carbohidratos y Grasas que debe ingerir el usuario para lograr su objetivo, se obtiene a partir de las Kcals. FechaInicio es la fecha de alta que tiene el usuario, esto será necesario para poder realizar un seguimiento al usuario.

3. Diseño

3.1. Diagrama Entidad-Relación

Mediante este tipo de diagrama representamos las entidades relevantes de un sistema de información, así como sus interrelaciones y propiedades. Facilitando la representación de entidades en una base de datos. Este diagrama se ha obtenido mediante el módulo ORM[2] del *framework* de desarrollo empleado, el cual nos permite mapear las clases en tablas de una base de datos [14].

La normalización [15] es una técnica de modelado que consiste en aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo entidad-relación al modelo relacional. Existe una serie de formas normales, cuanto más alta sea la forma normal aplicable a la tabla, menos vulnerable será a inconsistencias y anomalías. Seguidamente se muestra las reglas a satisfacer de las primeras tres formas normales:

- Una tabla está en primera forma normal sí:
    a. Sus atributos contienen valores atómicos.
- Una tabla está en segunda forma normal sí:
    a. Está en primera forma normal
    b. Los atributos que no forman parte de ninguna clave dependen de forma completa de la clave principal. Básicamente, no dependen de subclaves.
- Una tabla está en tercera forma normal sí:
    a. Está en segunda forma normal
    b. Todos los atributos que no son clave primaria no dependen transitivamente de esta.

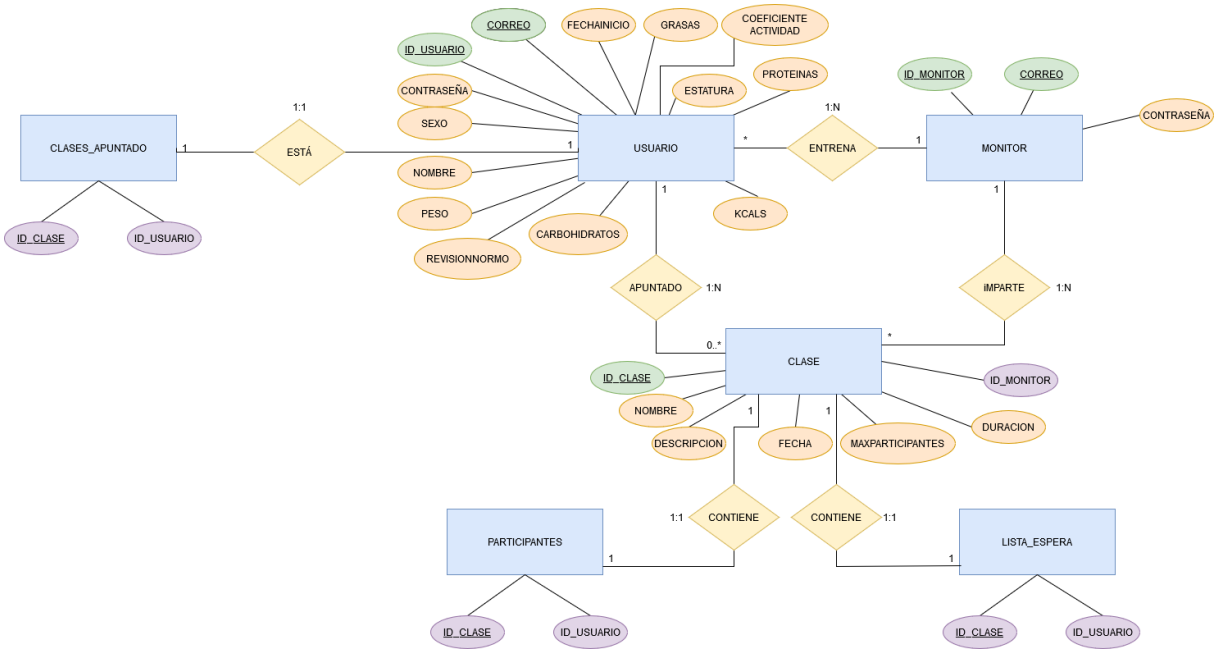
A continuación, adjunto el modelo entidad relación, donde podemos observar una serie de cambios con respecto al modelo del dominio. Los atributos propios de la clase están de color amarillo, mientras que las llaves primarias de cada clase están

[2] ORM: es un modelo de programación que permite mapear las estructuras de una base de datos relacional, sobre una estructura de entidades con el objeto de simplificar y acelerar el desarrollo de nuestras aplicaciones.

subrayadas y de color verde y finalmente las llaves foráneas están de color morado. Tras esto, los principales cambios son:

- Se añade 3 nuevas tablas:
  - Clases_Apuntado: Corresponde a las clases que el usuario está apuntado.
  - Participantes: Corresponde a la *lista* de participantes de una clase.
  - *Lista*_Espera: Corresponde a la *lista* de espera de una clase.
- Nuevo atributo de la tabla Clase, correspondiente al id del monitor que la ha creado e imparte.

El resultado de este modelo, es una tabla en tercera forma normal, donde los atributos que no son clave primaria no dependen transitivamente de esta.



3.2. Diagrama de Clases

A continuación, voy a mostrar el diagrama de clases realizado para este proyecto. He decidido ir adjuntando el resultado del diagrama de clases después de cada iteración, para así poder mostrar poder mostrar la evolución de este.

3.2.1. *Backend*

El diseño de este diagrama de clases es debido al uso de Spring[3] para el desarrollo del proyecto, el cual utiliza diferentes patrones de diseño: Singleton a la hora de definir los *beans*[4] en los archivos de configuración, el patrón *DTO*[5] para crear un objeto plano o POJO[6], con una serie de atributos que pueden ser enviados o recuperados del servidor, entre algunos otros.

Comenzaremos con el diagrama de clases del *backend*, donde utilizo Spring MVC.

[3] Spring: Es un framework para la creación de aplicaciones empresariales Java. https://spring.io/
[4] Bean: Clase destinada a encapsular información, para reutilizar código fuente, estructurando este código fuente en unidades lo más sencillas posibles.
[5] DTO: Es un objeto plano que es utilizado para la transferencia de datos.
[6] POJO: Es un objeto simple, que no extiende ni implementa ningún tipo de framework

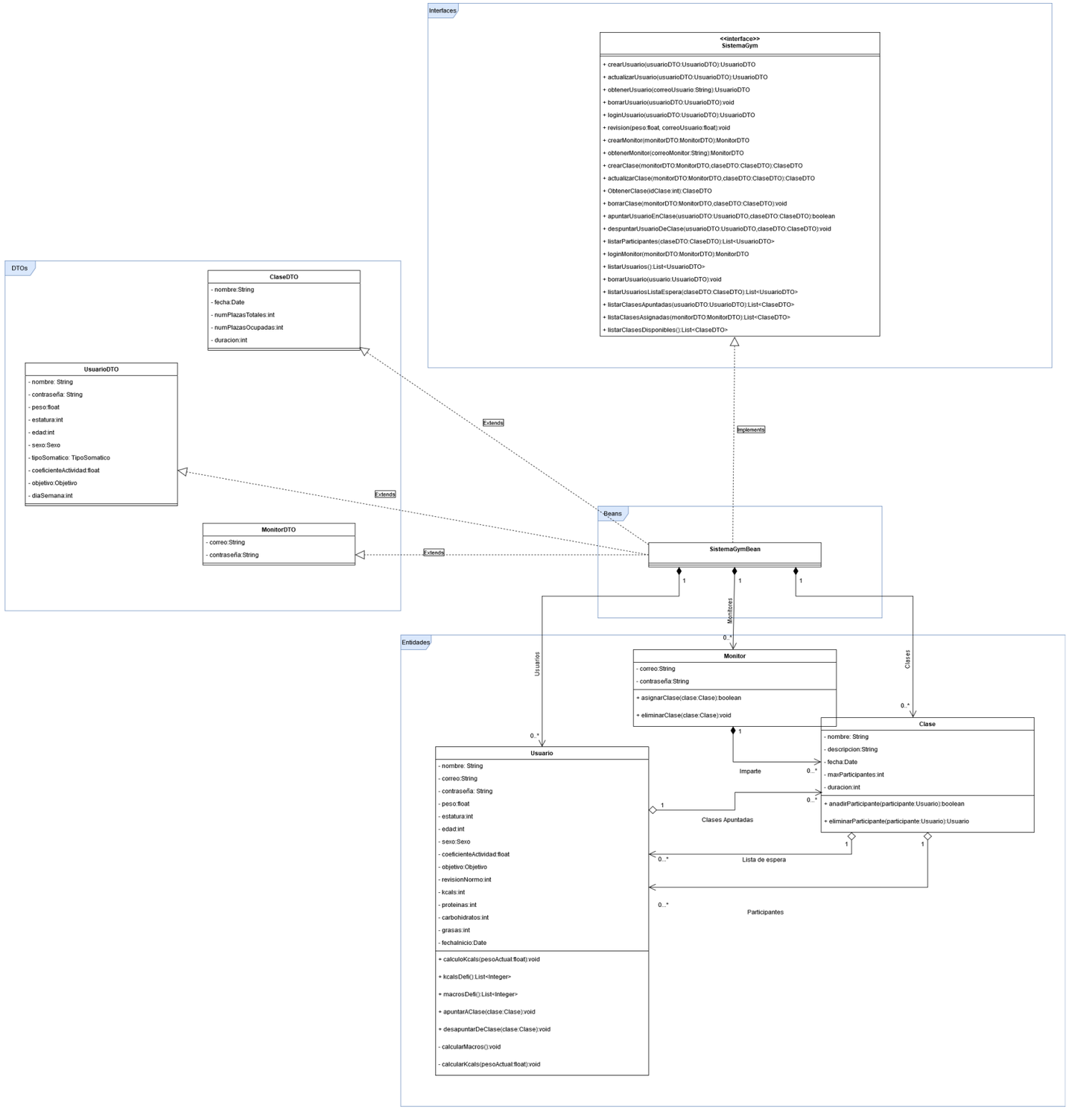


Ilustración 8: Diagrama de clases final (*Backend*)

El cambio más aparente podemos observarlo en la interfaz, que a medida que vamos avanzando en las iteraciones, incorpora nuevos métodos que son necesarios para satisfacer las historias de usuario dedicadas a cada iteración.

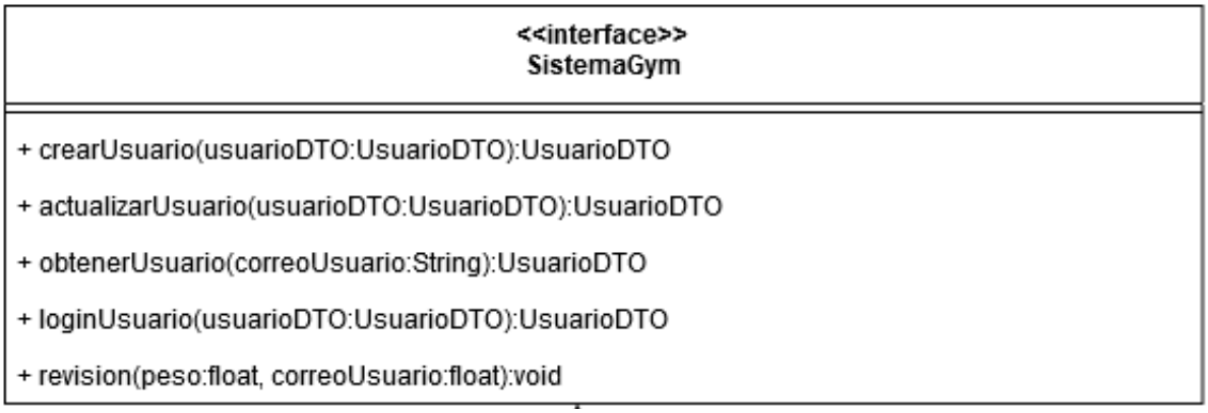


Ilustración 9: Interfaz iteración 1

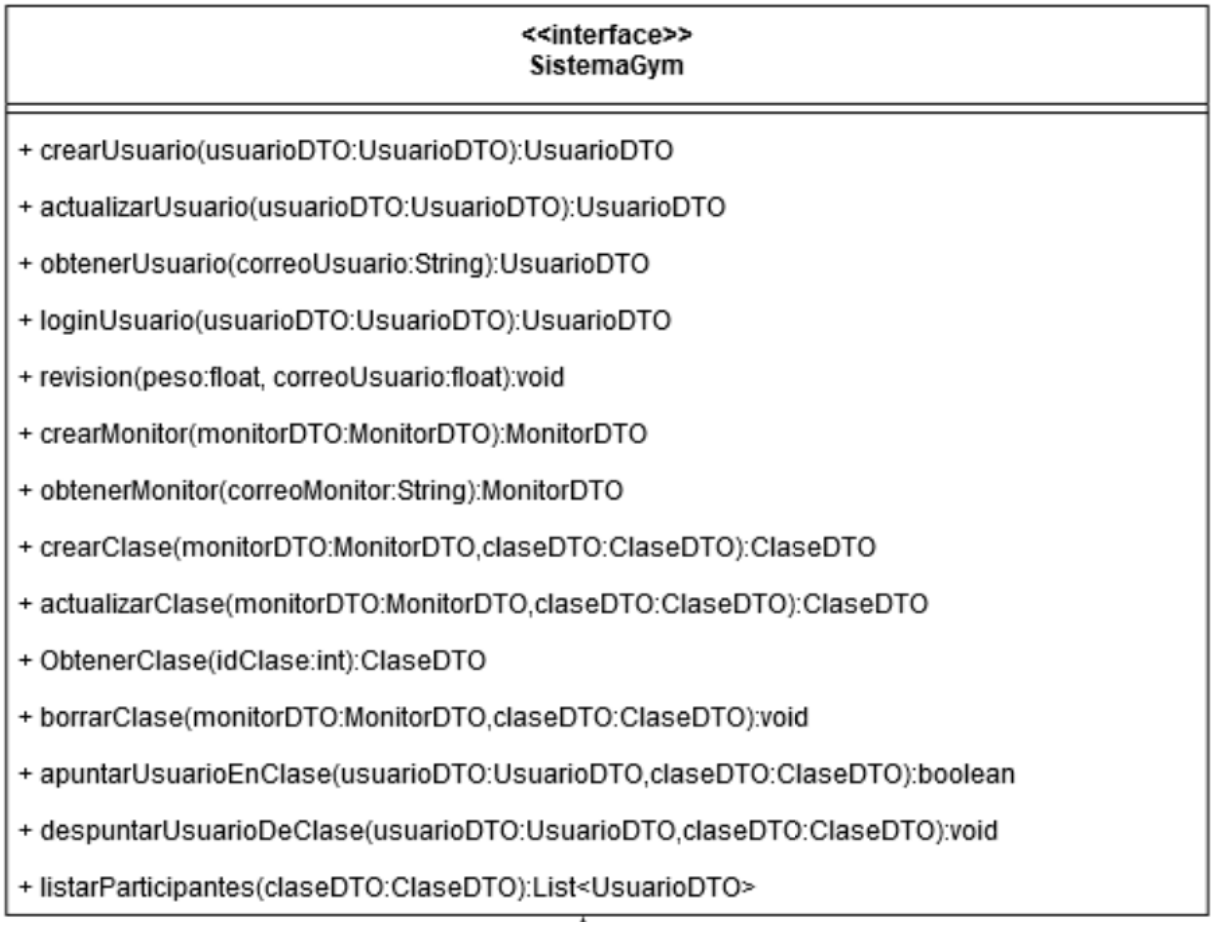


Ilustración 10: Interfaz iteración 2

Ilustración 11: Interfaz iteración 3

Además de la interfaz, otro paquete que evoluciona a medida que pasan las iteraciones, es el paquete de las entidades. Nos damos cuenta que a medida que implementamos operaciones en el *bean*, es necesario la implementación de otras funciones en las entidades.

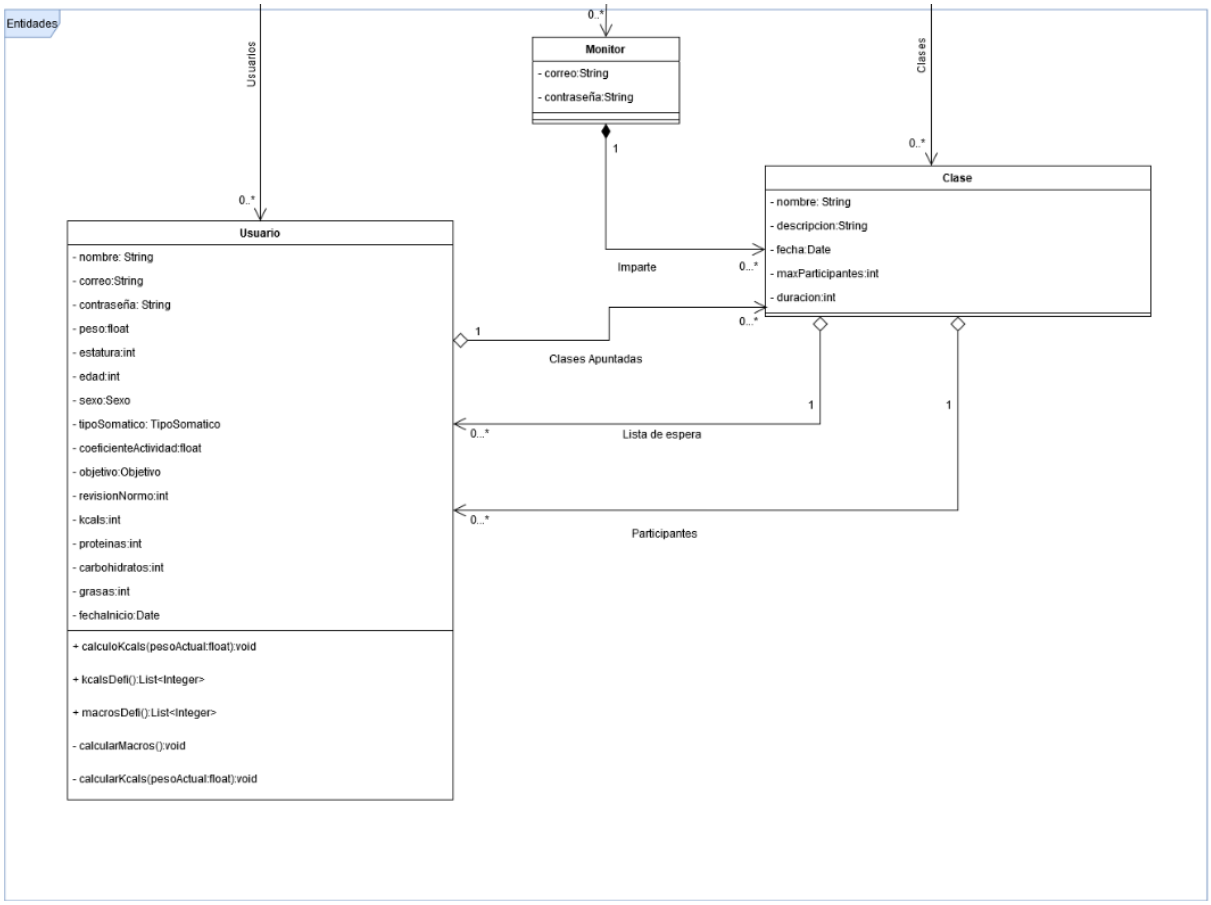


Ilustración 12: Entidades iteración 1

En esta iteración nos centramos más bien en la parte del usuario, por lo tanto, como podemos observar, dentro del paquete entidades, solo hemos desarrollado funciones en la entidad usuario.

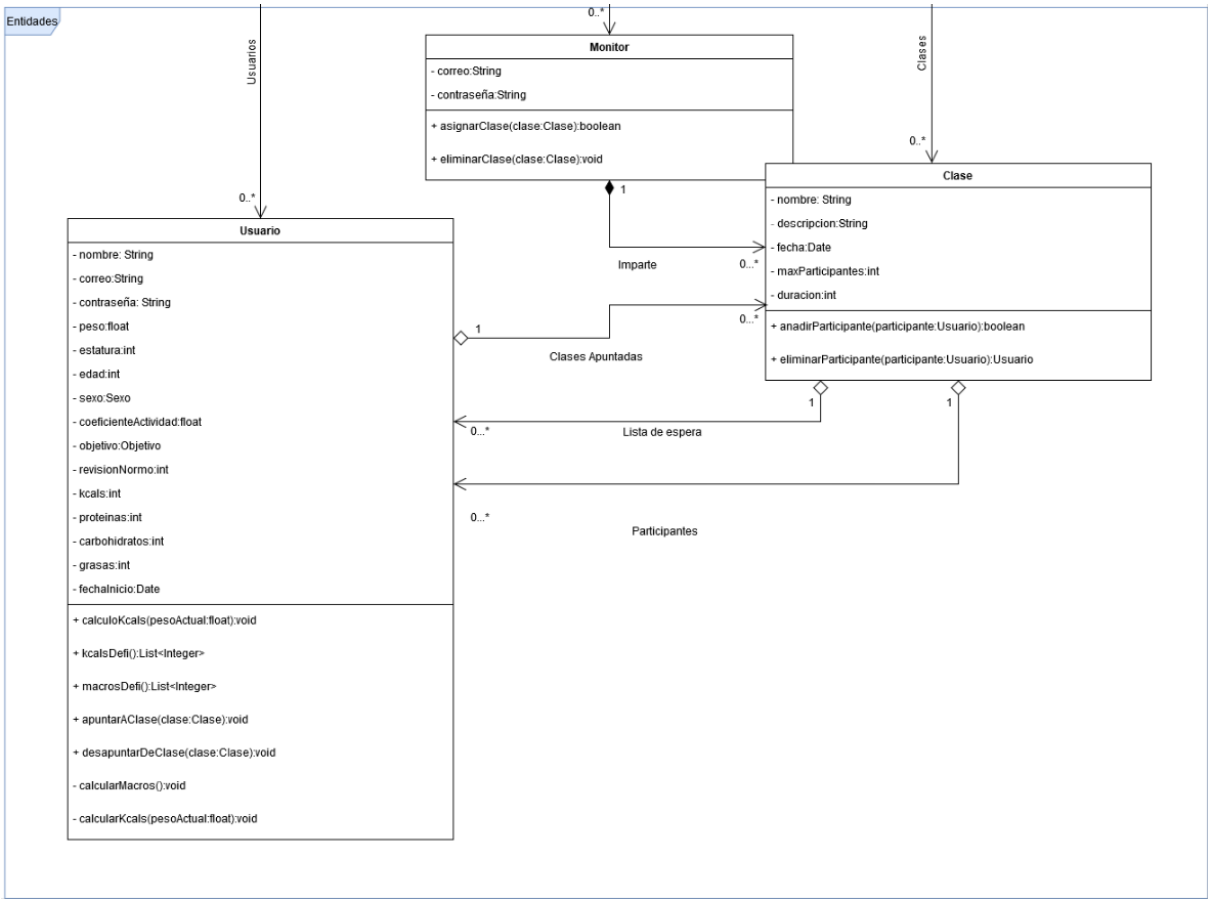


Ilustración 13: Entidades iteración 2

En esta iteración terminamos las funciones que necesitaremos como apoyo al *bean*, es decir en la iteración 3 no añadimos ninguna operación nueva en el paquete entidades.

3.2.2. *Frontend*

Seguidamente incorporaré el diagrama de clases con respecto al *Frontend*, dónde la tecnología a usar es Angular. Angular utiliza el patrón de diseño MVC(Modelo-Vista-Controlador) repartiendo el código del programa en componentes, fragmentándolo según su función dentro del proyecto.

Este diagrama está caracterizado por el software que utilizamos, Angular, y su estructura basada en componentes.

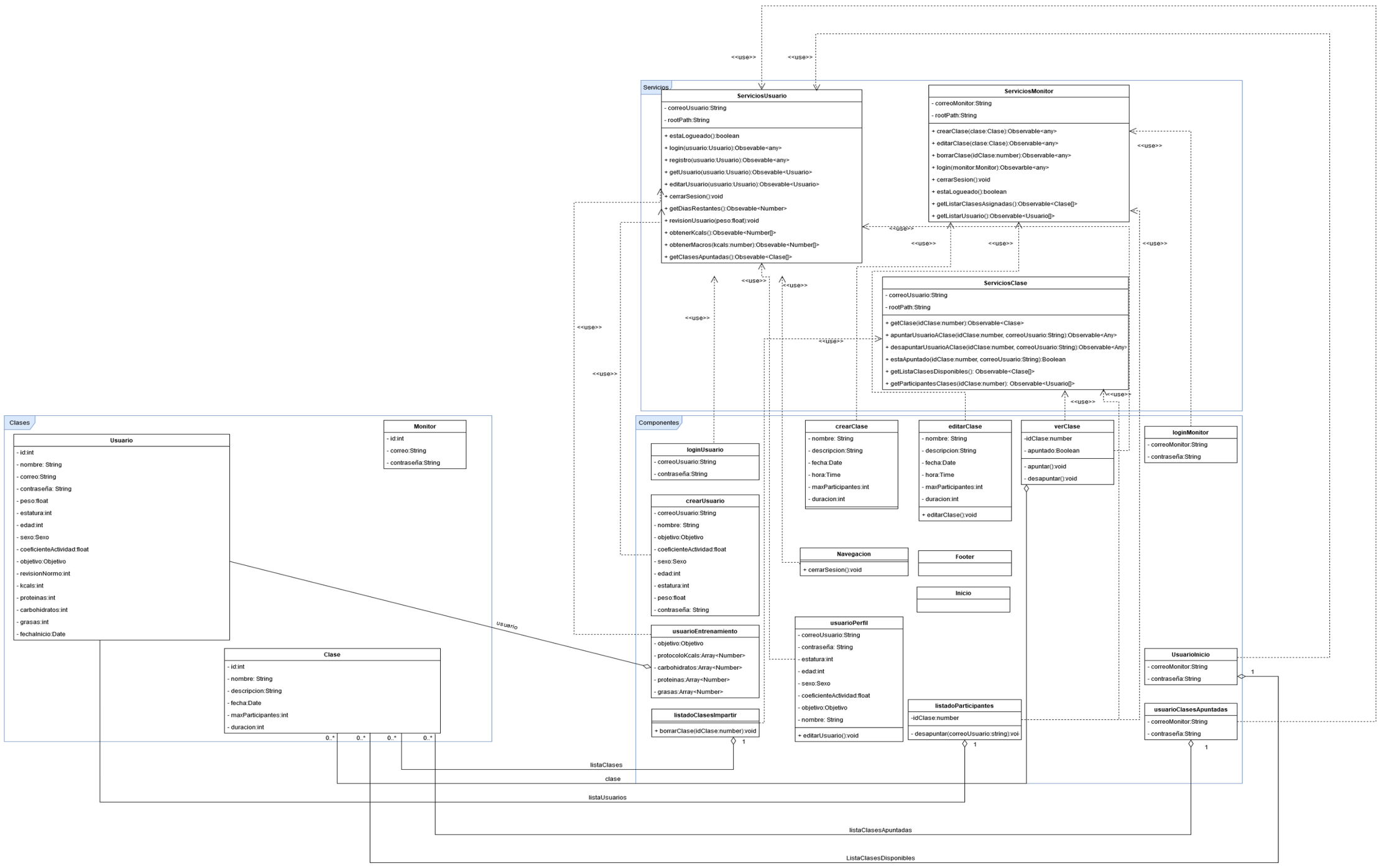


Ilustración 14: Diagrama de clases final (*Frontend*)

3.3. Diagramas de secuencia

En este apartado vamos a utilizar los diagramas de secuencia del sistema, mediante los cuales obtenemos la representación de la interacción entre los diferentes objetos de nuestro sistema reflejados en los diagramas de clases del apartado anterior. En estos diagramas incluimos la interacción de nuestro *Frontend* que corresponde al color azul en el diagrama con nuestro *Backend* que corresponde al color verde, para ver cómo funcionaría realmente.

Acerca del desarrollo de diagramas de secuencia, he decidido hacer los más complejos y llamativos para poder observar el correcto funcionamiento. Además, solo se incorporan aquellos más representativos para aclarar el funcionamiento del diseño propuesto. El *resto* serian similares a los presentados. no desarrollados son bastante similares a los realizados. A continuación, añado un diagrama genérico, donde podemos comprender el funcionamiento básico de este proyecto.

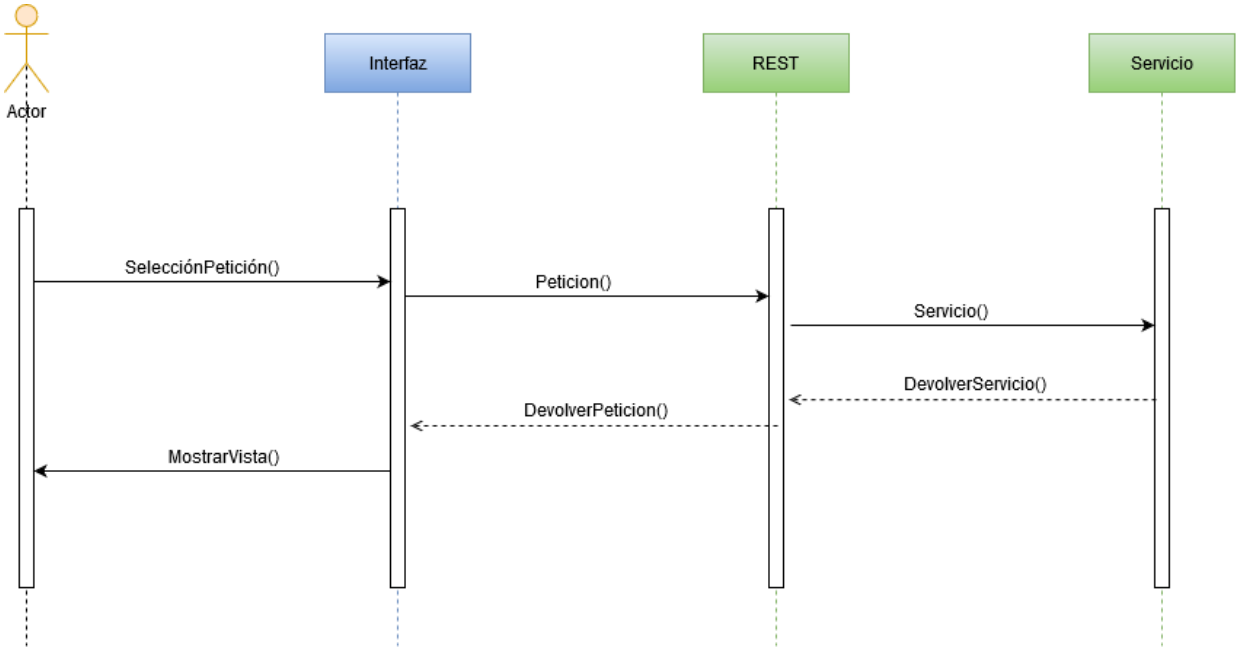


Ilustración 15: Diagrama de secuencia (Genérico)

Tras este primer ejemplo genérico, donde muestro el similar funcionamiento que tendrán los demás diagramas a representar. Pasamos a mostrar los diagramas seleccionados para ser representados detalladamente.

En primer lugar, adjuntamos el diagrama de secuencia del “*Login* de un monitor”. En este diagrama intentamos cubrir las diferentes posibilidades, tales como: no existe el monitor, contraseña incorrecta y finalmente el escenario exitoso.

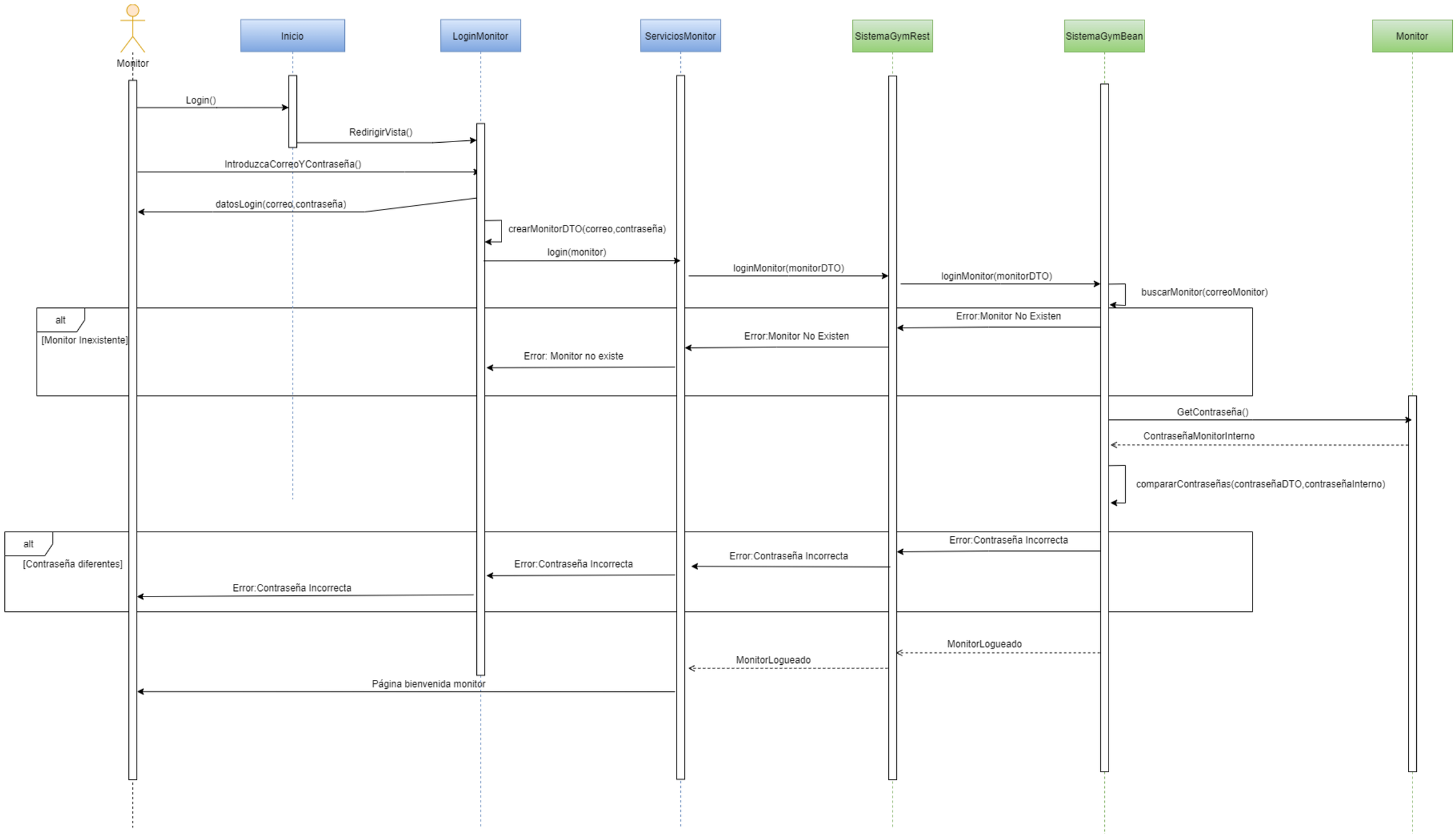


Ilustración 16: Diagrama de secuencia (*Login* de Monitor)

El siguiente diagrama es el registro de un Usuario. Mostramos desde cero cómo crear un usuario y cómo intervienen los diferentes componentes de nuestro sistema.

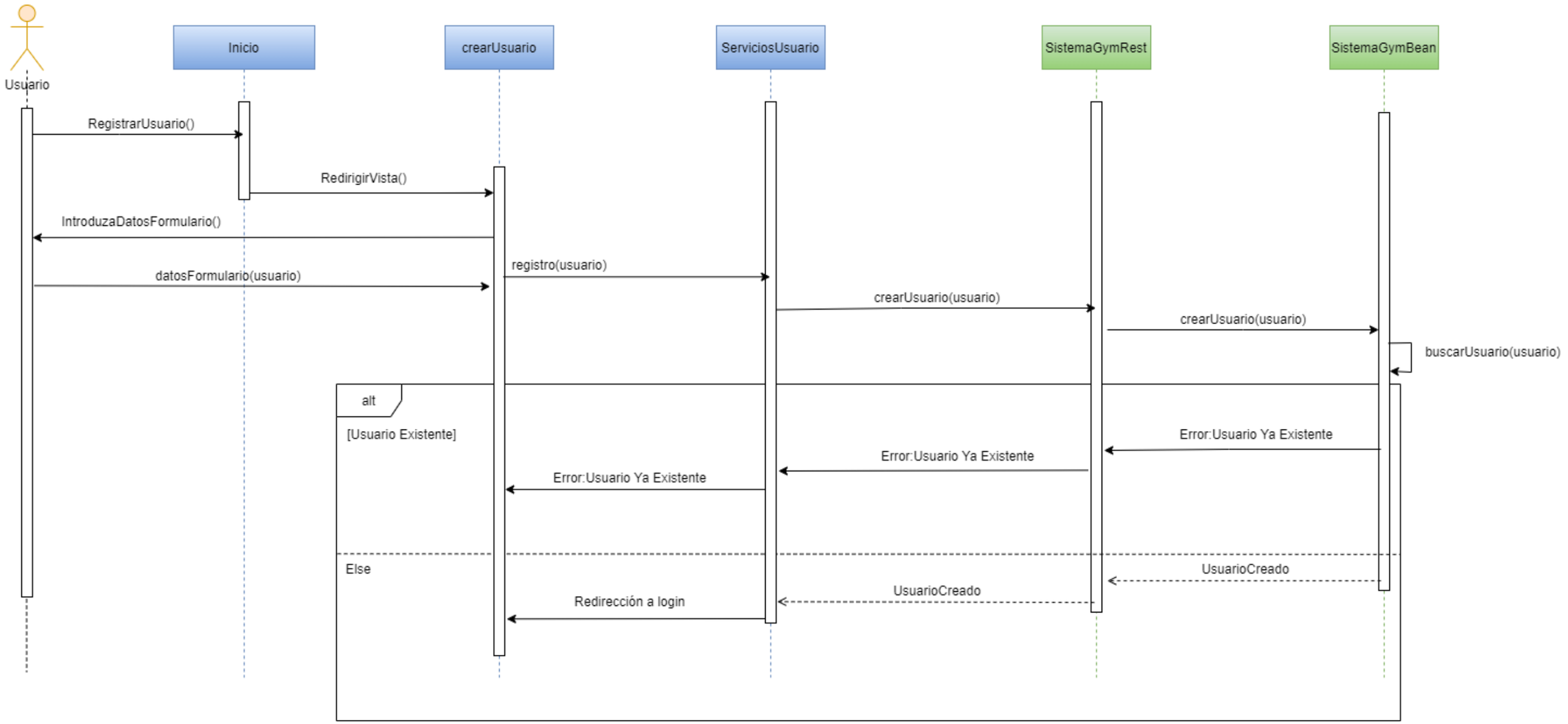


Ilustración 17: Diagrama de secuencia (Registro de Usuario)

En el diagrama de secuencia de borrar una clase, intervienen varios factores: primero como monitor tiene que estar registrado previamente, debe existir el monitor y la clase y para finalizar la clase a borrar debe haber sido creada por el monitor.

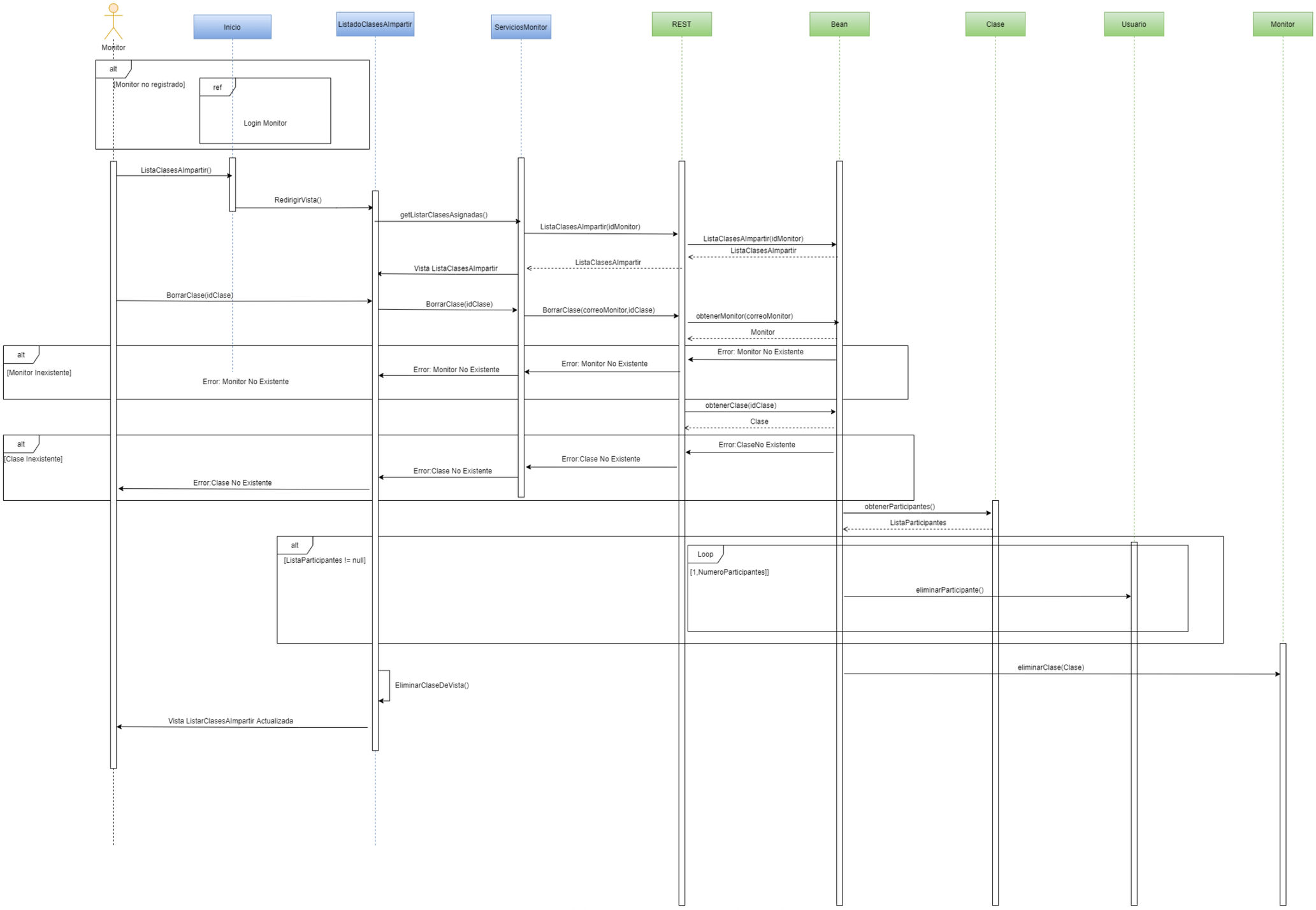


Ilustración 18:Diagrama de secuencia (Borrar Clase)

El siguiente diagrama, simula el comportamiento de la aplicación a la hora de apuntar a un usuario en una clase.

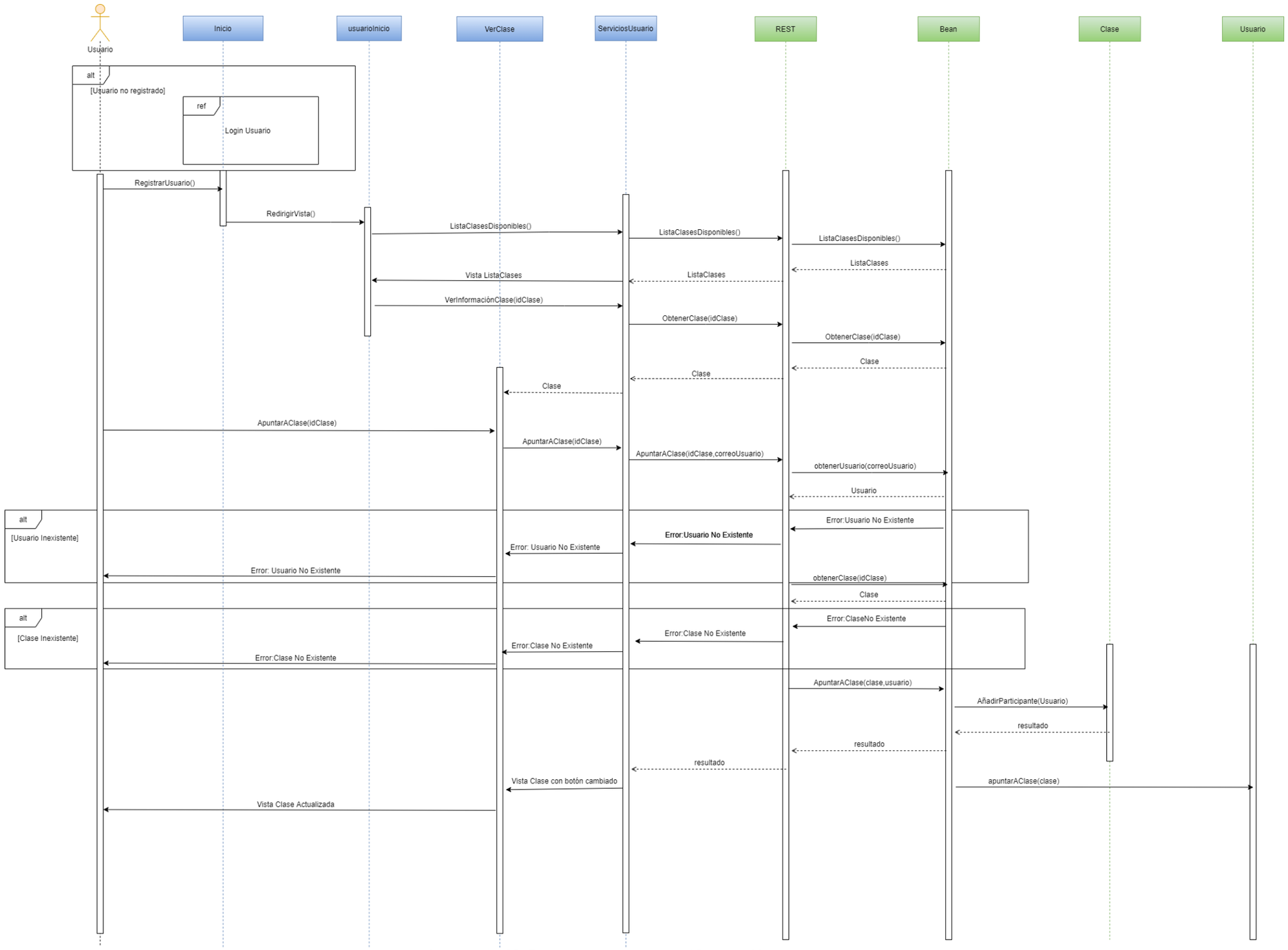


Ilustración 19: Diagrama de secuencia (Apuntar Usuario a clase)

El último diagrama de secuencia debido a su complejidad, he decidido hacerlo en el caso de que el objetivo del usuario sea la pérdida de grasa. Los objetivos ganar músculo y mantenimiento, serían similares excepto que en vez de mostrar un plan específico durante X semanas, solo se mostrarían las calorías, proteínas, grasas y carbohidratos que deben ingerirse. Esto se debe a que en un plan de mantenimiento o ganancia de músculo no es necesario establecer una fecha, el usuario puede seguir mientras quiera, siempre debe ser aconsejado por el monitor.

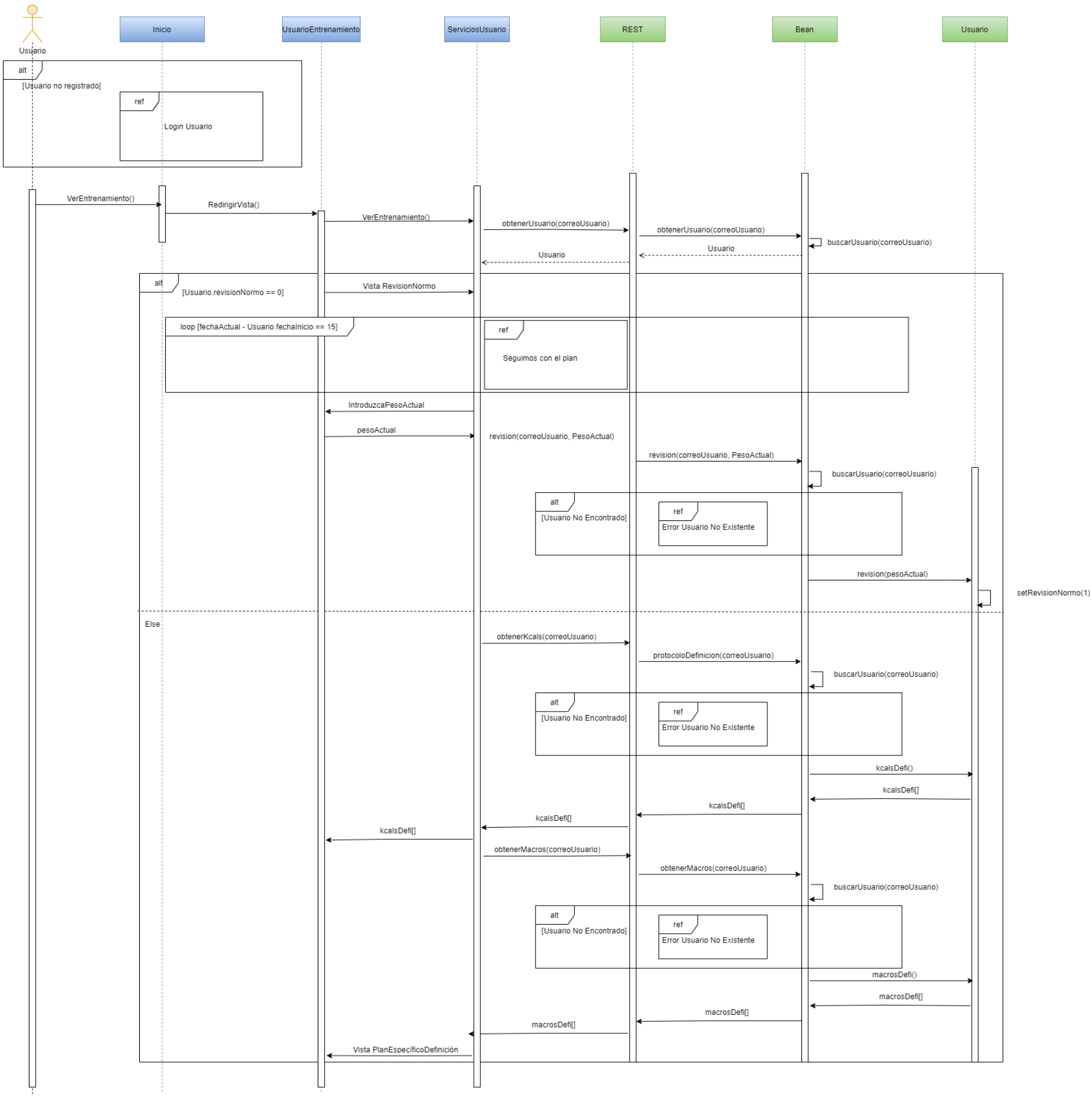


Ilustración 20: Diagrama de secuencia (Entrenamiento)

3.4. Diseño del API *REST*

Api *REST* [16], es un estilo de arquitectura que es utilizado para diseñar aplicaciones en red, basándose en la creación de servicios. El servicio *REST* se rige mediante *HTTP*, y las operaciones mediante podremos manipular los recursos de la aplicación son las siguientes: *GET* para consultar y leer, *POST* para crear, PUT para editar y *DELETE* para eliminar. Nuestra API ha sido desarrollada mediante el *framework* Spring y la componen los siguientes elementos:

Endpoints: son las URLs que retornan valores de una API. Los *endpoints* disponibles de nuestra API, son los siguientes (todos constan como prefijo de *http*://localhost:8080/sistemaGym):

- /usuario
- /usuario/{correoUsuario}
- /usuario/{correoUsuario}/protocoloDefinicion
- /usuario/{correoUsuario}/dias*Rest*antes
- /usuario/{correoUsuario}/revision
- /usuario/*login*
- /usuario/{correoUsuario}/macros
- /monitor/{correoMonitor}/clase
- /monitor/{correoMonitor}/clase/{idClase}
- /monitor/{correoMonitor}
- /monitor/*login*
- /clase/{idClase}
- /clase/{idClase}/apuntadas/{correoUsuario}
- /clase/{idClase}/apuntadasEspera/{correoUsuario}
- /clase/{idClase}/participantes
- /clase/{idClase}/*list*aDeEspera
- /usuario/{correoUsuario}/clasesApuntadas
- /usuario/{correoUsuario}/clasesAsignadas

URL	Operaciones	Datos de retorno	Códigos de retorno
/usuario	CrearUsuario (*POST*)	Usuario*DTO*, en el caso de que sea realice correctamente el registro.	*CREATED* (201), en el caso de un registro correcto. *CONFLICT* (409), en el caso de que el usuario ya esté registrado.
/usuario/{correoUsuario}	obtenerUsuario (*GET*)	Usuario*DTO*, en el caso de que exista el usuario.	*OK* (200), en el caso de obtener el usuario. *NOT_FOUND* (404), en el caso de no encontrar al usuario.
	actualizarUsuario (UPDATE)	Usuario*DTO*, en el caso de que sea realice correctamente el actualizado de datos.	*OK* (200), en el caso de actualizar el usuario. *NOT_FOUND* (404), en el caso de no encontrar al usuario.
	borrarUsuario (*DELETE*)	No devuelve ningún tipo de datos.	*OK* (200), en el caso de borrar el usuario. *NOT_FOUND* (404), en el caso de no encontrar al usuario.

			CONFLICT (409), en el caso de que la contraseña del usuario a borrar sea incorrecta.
/usuario/{correoUsuario}/protocoloDefinicion	obtenerKcals (*GET*)	Devuelve una *lista* de enteros, que se corresponde al plan de Kcals a ingerir durante la definición.	*OK* (200), en el caso de obtener el plan. *NOT_FOUND* (404), en el caso de no encontrar al usuario.
/usuario/{correoUsuario}/dias*Resta*ntes	obtenerDias*Resta*ntes (*GET*)	Devuelve un entero con los días *resta*ntes para actualizar el peso.	*OK* (200), en el caso de obtener los días. *NOT_FOUND* (404), en el caso de no encontrar al usuario.
/usuario/{correoUsuario}/revision	revisionUsuario (*POST*)	No devuelve ningún valor	*OK* (200), en el caso realizar correctamente la revisión. *NOT_FOUND* (404), en el caso de no encontrar al usuario.
/usuario/*login*	*login*Usuario (*POST*)	Usuario*DTO* en el caso de que se procese	*OK* (200), en el caso de credenciales

		correctamente el *login*.	correctos del usuario. *NOT_FOUND* (404), en el caso de no encontrar al usuario. *CONFLICT* (409), en el caso de credenciales incorrectos del usuario
/usuario/{correoUsuario}/macros	obtenerMacros (*GET*)	Devuelve una *lista* con los macros	*OK* (200), en el caso realizar correctamente la devolución de macros. *NOT_FOUND* (404), en el caso de no encontrar al usuario.
/monitor/{correoMonitor}/clase	crearClase (*POST*)	Clase*DTO*, en el caso de que sea realice correctamente la creación.	*CREATED* (201), en el caso de una creación correcta. *CONFLICT* (409), en el caso de que la clase ya esté creada. *CONFLICT* (409), en el caso de que el monitor esté

			ocupado para asignarse esa clase. *NOT_FOUND* (404), en el caso de no encontrar al monitor.
/monitor/{correoMonitor}/clase/{idClase}	actualizarClase (UPDATE)	Clase*DTO*, en el caso de que sea realice correctamente el actualizado de datos.	*OK* (200), en el caso de actualizar la clase. *CONFLICT* (409), en el caso de que el monitor no imparta esa clase. *CONFLICT* (409), en el caso de que el monitor esté ocupado para asignarse esa clase. *NOT_FOUND* (404), en el caso de no encontrar al monitor. *NOT_FOUND* (404), en el caso de no encontrar la clase.

	borrarClase (*DELETE*)	No devuelve ningún tipo de datos.	*OK* (200), en el caso de borrar la clase. *CONFLICT* (409), en el caso de que el monitor no imparta esa clase. *CONFLICT* (409), en el caso de que el monitor esté ocupado para asignarse esa clase. *NOT_FOUND* (404), en el caso de no encontrar al monitor. *NOT_FOUND* (404), en el caso de no encontrar la clase.
/monitor/{correoMonitor}	obtenerMonitor (*GET*)	Monitor*DTO*, en el caso de que exista el monitor.	*OK* (200), en el caso de obtener el monitor. *NOT_FOUND* (404), en el caso de no encontrar al monitor.
/monitor/*login*	*login*Monitor	Monitor*DTO* en el caso de que se	*OK* (200), en el caso de

	(*POST*)	procese correctamente el *login*.	credenciales correctos del monitor. *NOT_FOUND* (404), en el caso de no encontrar al monitor. *CONFLICT* (409), en el caso de credenciales incorrectos del monitor
/clase/{idClase}	obtenerClase (*GET*)	Clase*DTO*, en el caso de que exista la clase.	*OK* (200), en el caso de obtener la clase. *NOT_FOUND* (404), en el caso de no encontrar la clase.
/clase/{idClase}/apuntadas/{correoUsuario}	apuntarUsuarioEnClase (*POST*)	*Boolean*, indicando true (se ha apuntado) o false (se ha apuntado a la *lista* de espera)	*OK* (200), en el caso apuntar al usuario. *NOT_FOUND* (404), en el caso de no encontrar la clase. *NOT_FOUND* (404), en el caso de no encontrar al usuario.

			CONFLICT (409), en el caso de que el usuario ya esté apuntado a esa clase.
	estaApuntado (*GET*)	*Boolean*, indicando true (está apuntado a la clase) o false (si no lo está)	*OK* (200), en el caso de realizar la consulta correctamente. *NOT_FOUND* (404), en el caso de no encontrar la clase. *NOT_FOUND* (404), en el caso de no encontrar al usuario.
	despuntarUsuario EnClase (*POST*)	No devuelve nada	*OK* (200), en el caso desapuntar al usuario. *NOT_FOUND* (404), en el caso de no encontrar la clase. *NOT_FOUND* (404), en el caso de no encontrar al usuario.

			CONFLICT (409), en el caso de que el usuario no esté apuntado a esa clase.
/clase/{idClase}/apuntadasEspera/{correoUsuario**}**	estaApuntado*Lista*Espera (*GET*)	*Boolean*, indicando *true* (está apuntado a la *lista* de espera) o *false* (si no lo está)	*OK* (200), en el caso de realizar la consulta correctamente. *NOT_FOUND* (404), en el caso de no encontrar la clase. *NOT_FOUND* (404), en el caso de no encontrar al usuario.
/clase/{idClase}/participantes	*lista*rParticipantes (*GET*)	*List*<Usuario*DTO*>, *lista* de los participantes de la clase.	*OK* (200), en el caso de realizar la consulta correctamente. *NOT_FOUND* (404), en el caso de no encontrar la clase.
/clase/{idClase}/*lista*DeEspera	*lista*rUsuarios*Lista*Espera (*GET*)	*List*<Usuario*DTO*>, *lista* de espera de la clase.	*OK* (200), en el caso de realizar la consulta correctamente.

			NOT_FOUND (404), en el caso de no encontrar la clase.
/usuario/{correoUsuario}/clasesApuntadas	*lista*rClasesApuntadas (*GET*)	*List*<Clase*DTO*>, *lista* de las clases a las que el usuario está apuntado.	*OK* (200), en el caso de realizar la consulta correctamente. *NOT_FOUND* (404), en el caso de no encontrar al usuario.
/usuario/{correoUsuario}/clasesAsignadas	*lista*rClasesAsignadas (*GET*)	*List*<Clase*DTO*>, *lista* de las clases que el monitor tiene asignadas.	*OK* (200), en el caso de realizar la consulta correctamente. *NOT_FOUND* (404), en el caso de no encontrar al monitor.

3.5. Diseño de la interfaz

La interfaz es un punto importante a la hora del diseño. A día de hoy, una aplicación web debe ser totalmente *responsive*[7], ya que normalmente el móvil es el principal método de acceso a cualquiera. Para realizar un diseño correcto, debemos satisfacer los criterios de aceptación definidos anteriormente en las historias de

[7] Responsive: es una técnica de diseño web que tiene como misión la correcta visualización de una página web en distintos dispositivos.

usuario. Hay diversas formas de plantear este diseño de interfaz [12], pero me he decantado por el desarrollo mediante *storyboards*.

Esta técnica se desarrolla mediante el diseño *wireframe* de las pantallas de usuario. Es una técnica muy útil, ya que permite cambios rápidos sin tener que rehacer desde cero la pantalla, como sería en el caso de prototipo a papel. Además, no es necesario un trabajo excesivo como en el caso de realizar una maqueta digital.

He realizado un storyboard completo, incluyendo todas las pantallas que va a contener a priori nuestro proyecto. Asimismo, he incluido pantallas tanto del rol usuario como del rol monitor.

- **Página de bienvenida a la web.**

En esta página, hay poca funcionalidad. Intentaremos realizar algún elemento más vistoso, como un *slider*. Principalmente observamos que va a haber un área de usuarios y otro de monitores, esto enlaza el *Login* de cada rol.

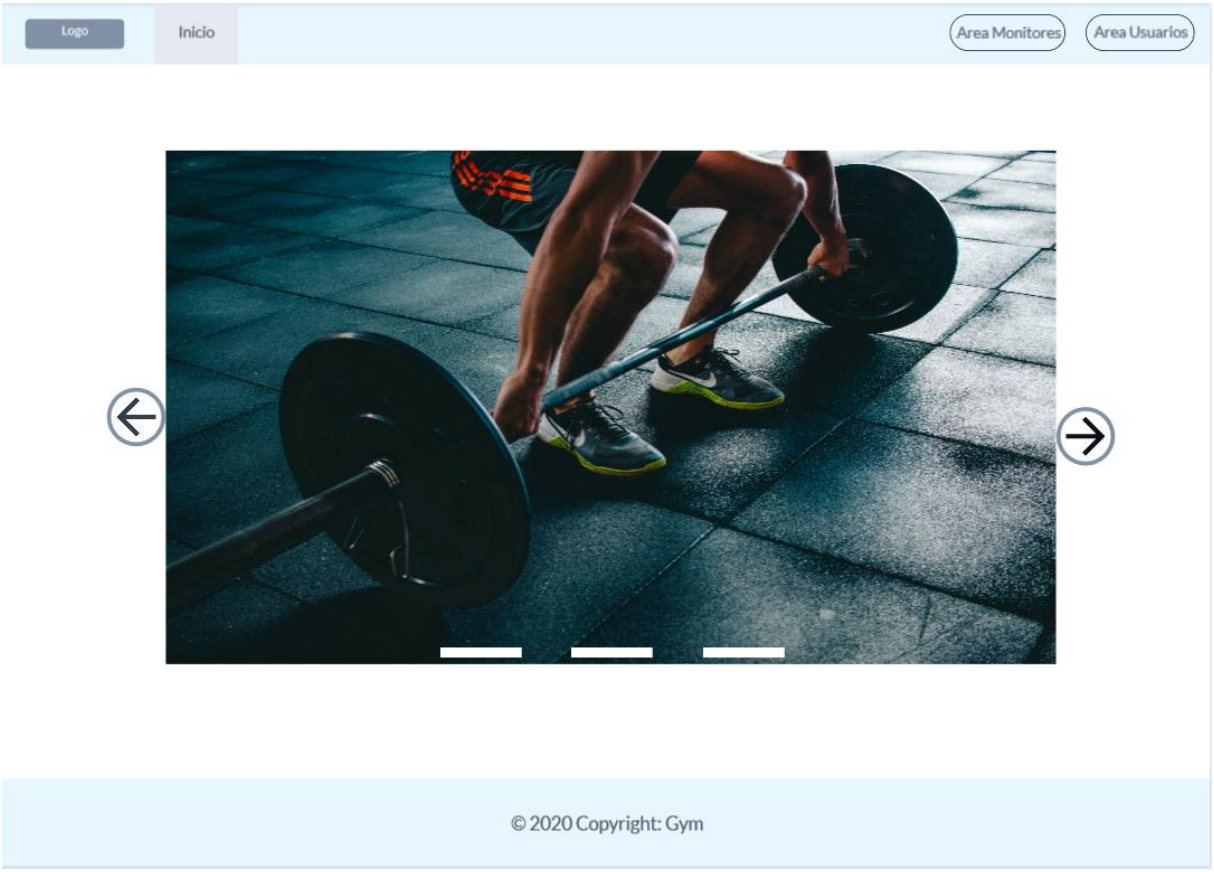


Ilustración 21: Página de bienvenida a la web

- ***Login* Usuario**

Una vez hacemos clic en Área Usuarios, nos redirige a la siguiente página, donde si no estamos registrados, tenemos un acceso directo debajo del botón de iniciar sesión para acceder de una forma rápida.

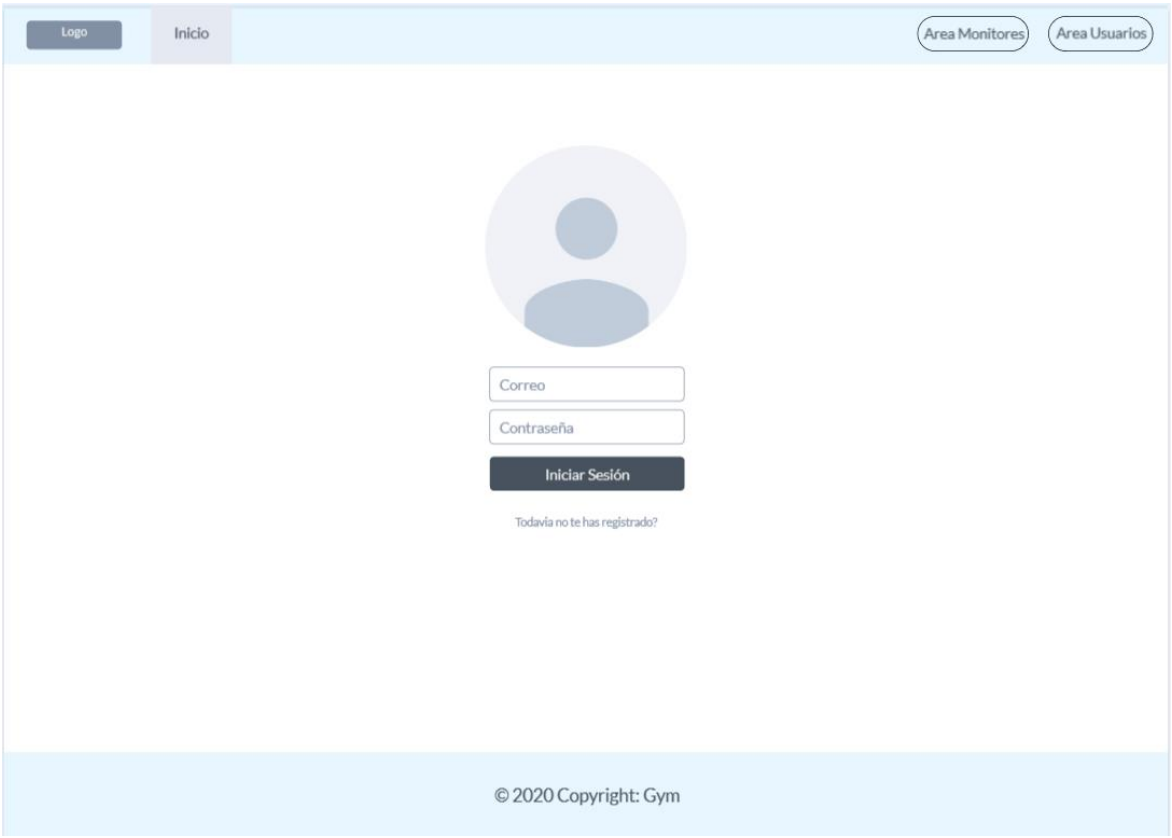


Ilustración 22: *Login* Usuario

- **Registro Usuario**

Para usuarios no registrados, después de acceder mediante el enlace que hemos comentado anteriormente, hemos creado el siguiente formulario.

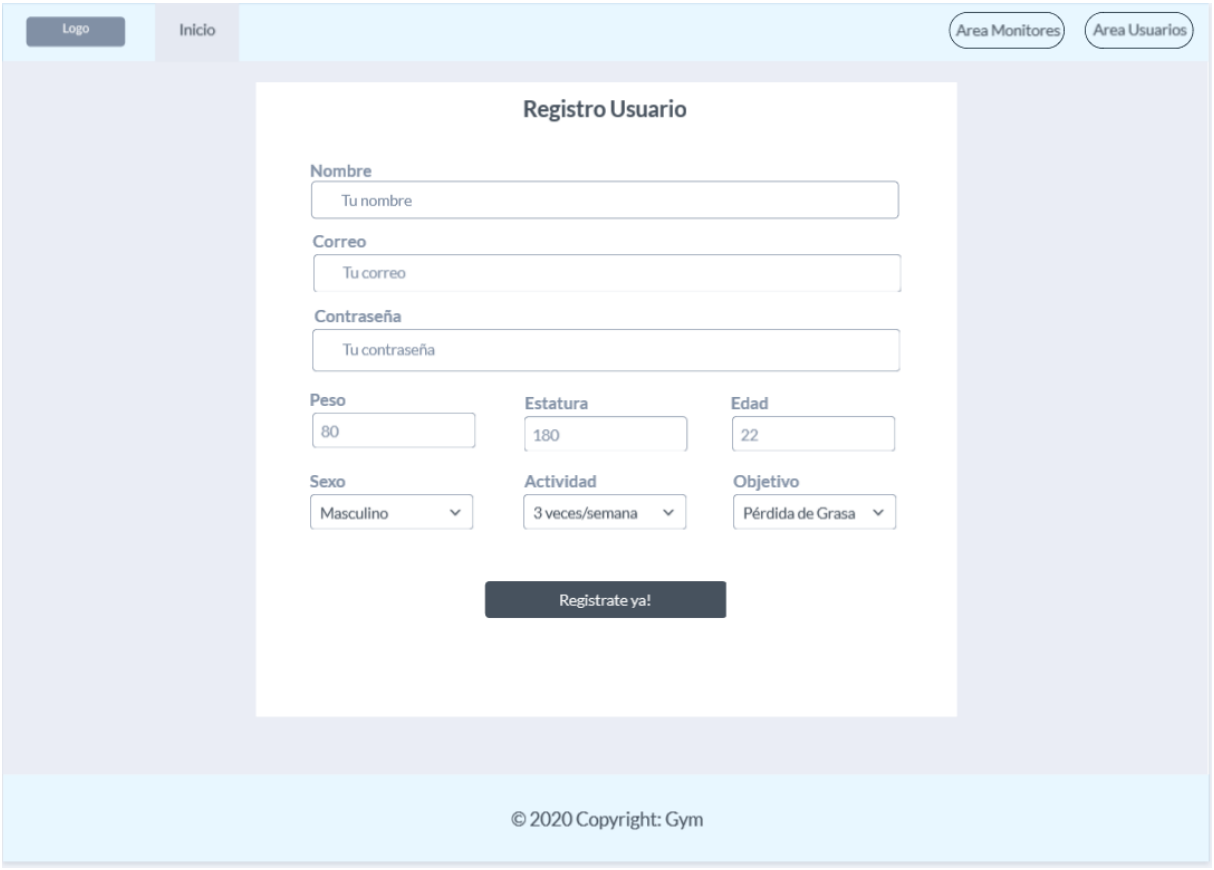


Ilustración 23: Registro Usuario

- **Página de bienvenida usuario**

En esta página podemos observar diversos cambios. El principal es la navegación, ya que no encontramos Área Monitores ni Área Usuarios. Sin embargo, hemos añadido otras funcionalidades como son: Perfil, Entrenamiento y Mis Clases. En esta ventana, muestro las clases que hay disponibles para que accedan a su información detallada y poder apuntarse si interesa.

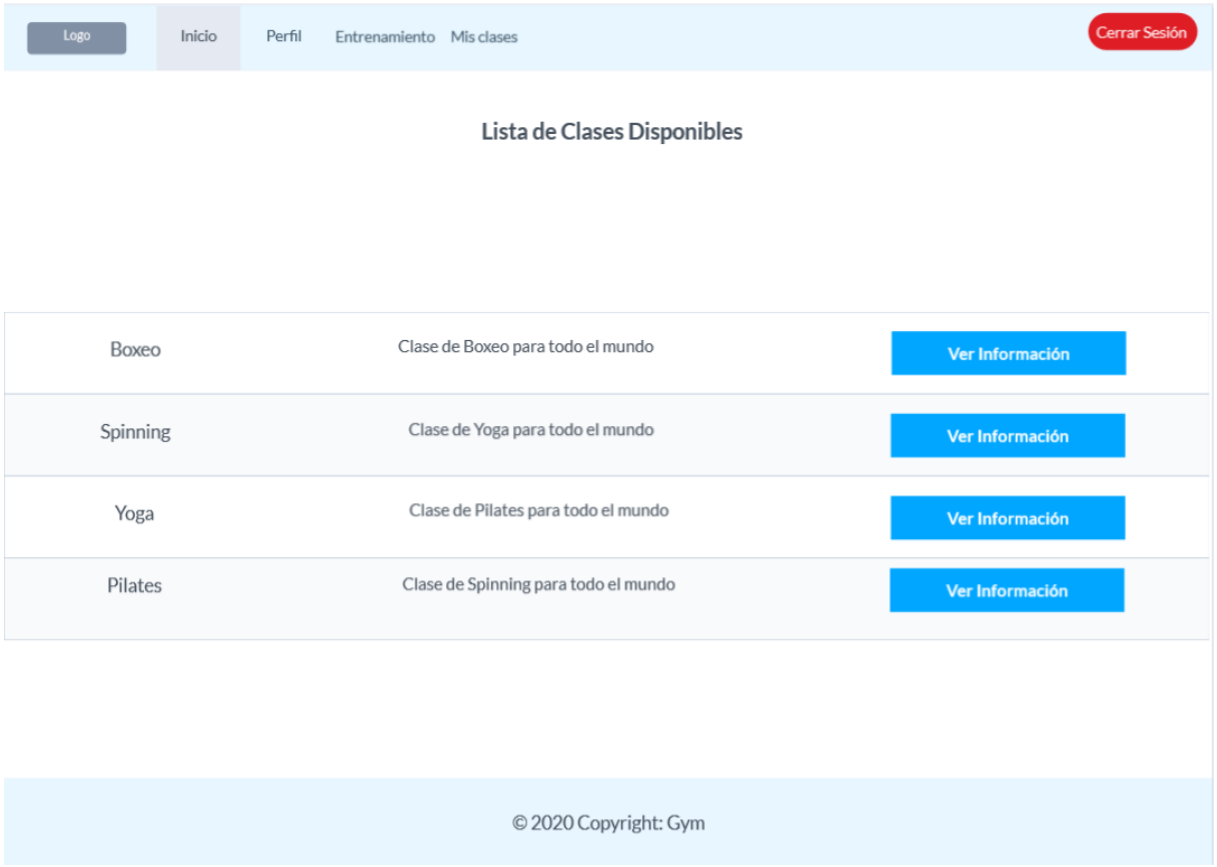


Ilustración 24: Página de bienvenida usuario

- **Información detallada de una clase**

Una vez hemos clicado en ver información, de la anterior ventana. Obtenemos la información detallada de la clase que hemos elegido. Si estamos apuntados a esta clase, aparecerá el botón de desapuntarme, mientras que si no estás apuntado saldrá el botón de apuntarme.

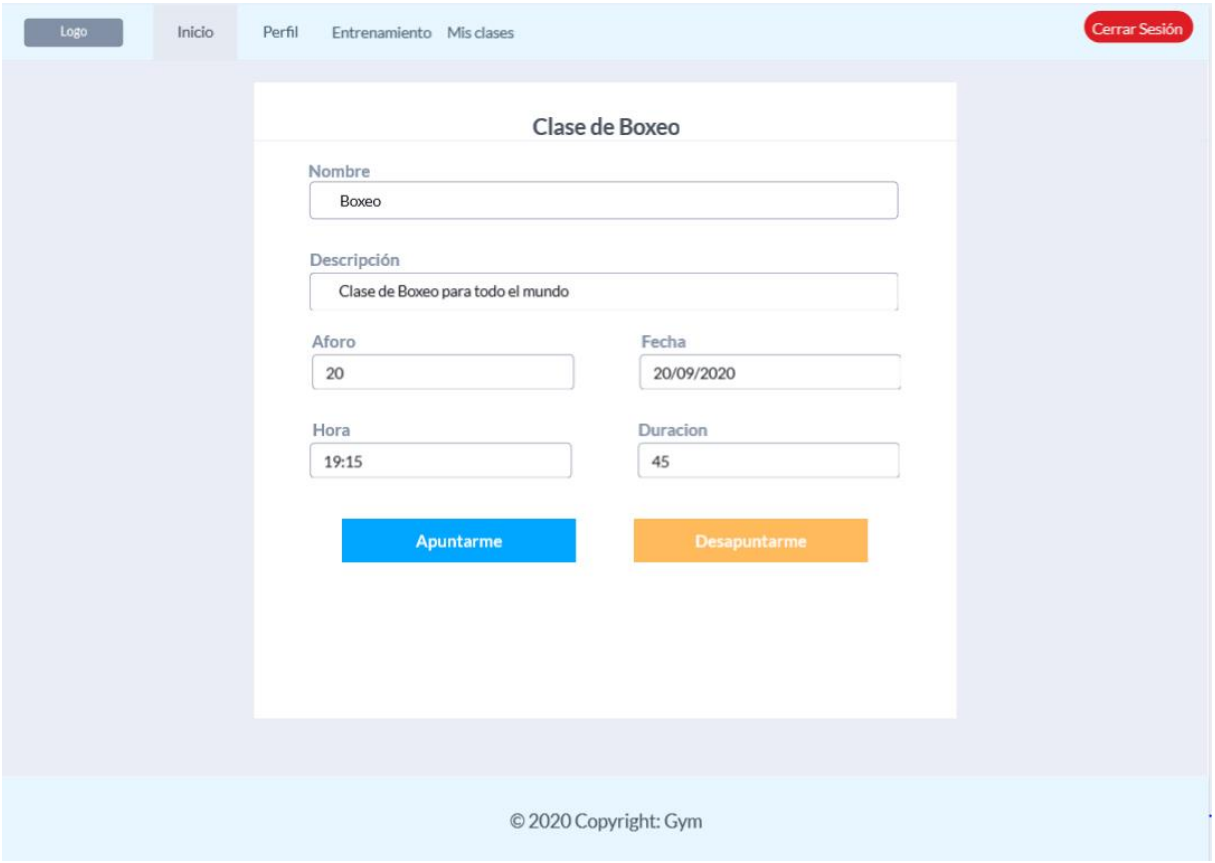


Ilustración 25: Información detallada de una clase

- **Perfil de usuario**

En este caso, he decidido englobar 2 historias de usuario dentro de esta sección. Una vez hemos iniciado sesión, si vamos a la sección de mi perfil encontramos una sección donde aparecen nuestros datos que indicamos en el registro. Además de esto he incorporado la funcionalidad de poder editar estos datos y mandarlos o cancelar en el caso de que no queramos modificar nada.

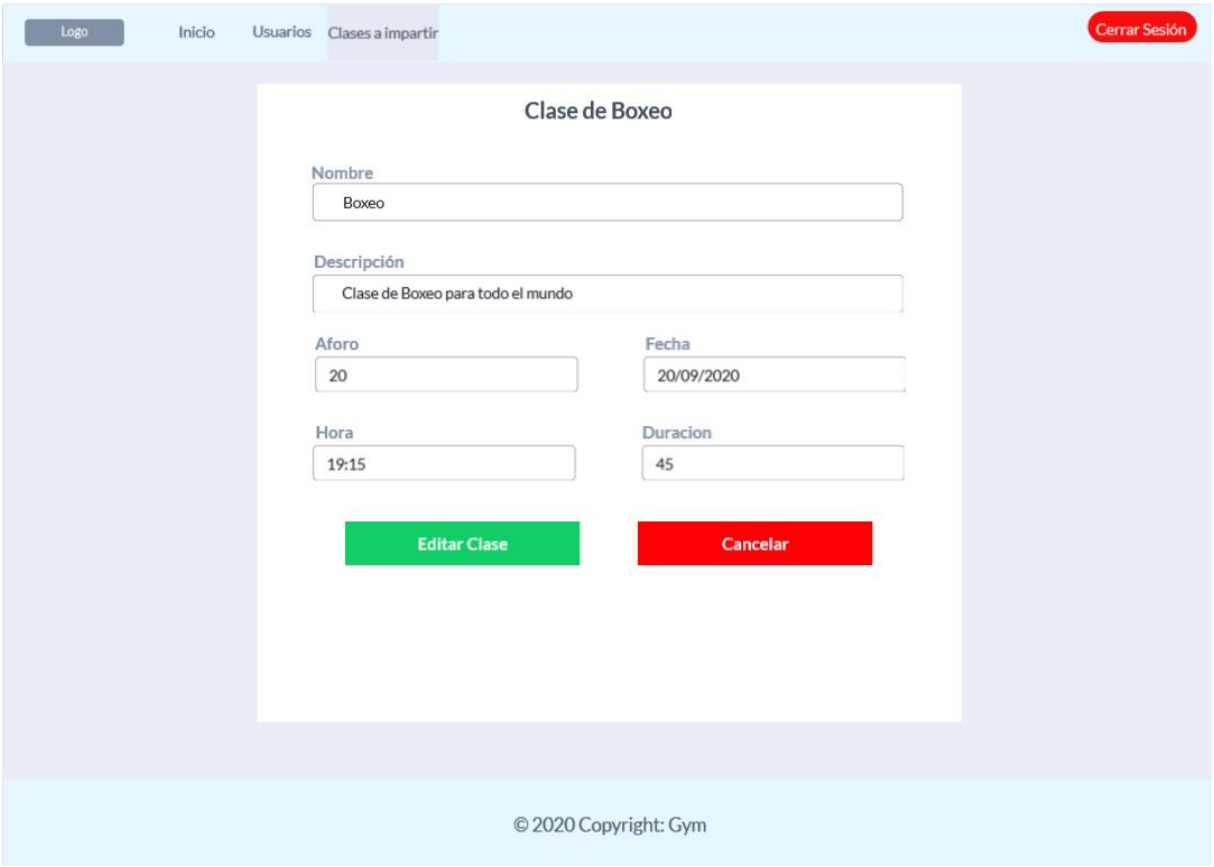


Ilustración 26: Perfil de usuario

- **Entrenamiento**

Este apartado es lo innovador del proyecto. Según en qué parte estés del protocolo, te aparecerá una pantalla u otra. Principalmente, lo que intentamos hacer es obtener la normo caloría del usuario, entonces te aparecerá esta pantalla. Además, esta pantalla la parte de ‘actualizar el peso’, no te saldrá hasta que no hayas realizado este plan durante 15 días.

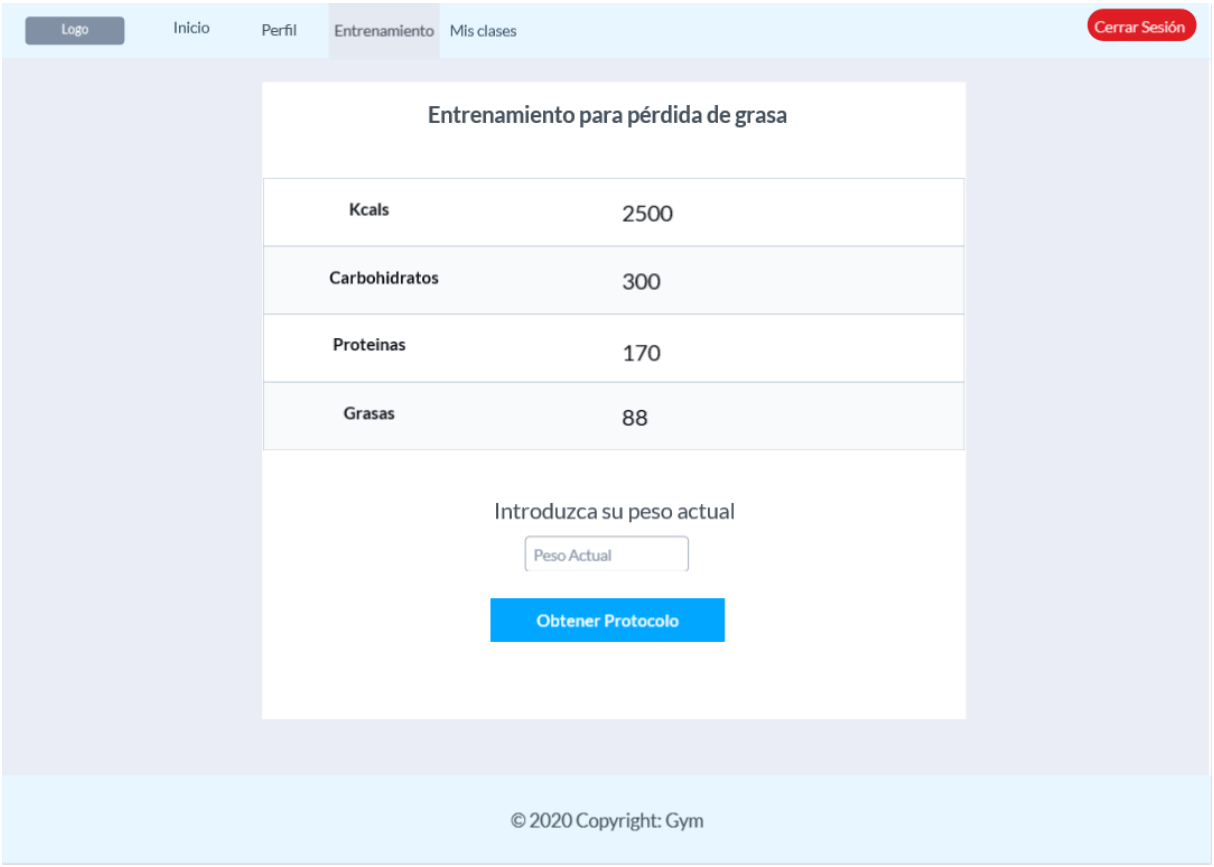


Ilustración 27: Entrenamiento

Una vez han pasado los 15 días y hemos introducido el peso actual, obtenemos la normo caloría del usuario y establecemos el protocolo de definición a seguir durante unas semanas.

Ilustración 28: Plan de entrenamiento

- **Mis clases**

En esta sección, aparecen las clases a las que el usuario está apuntado en modo de tabla.

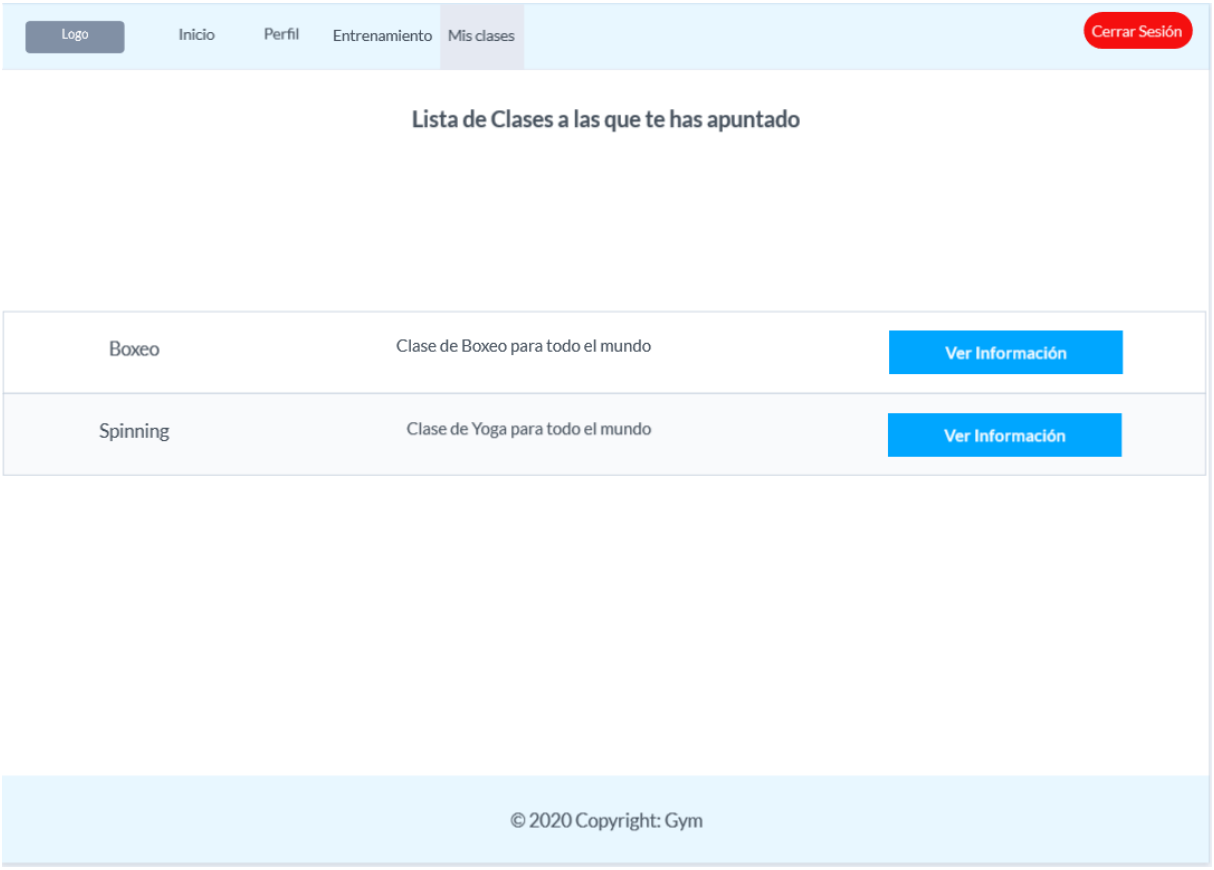


Ilustración 29: Mis clases

- ***Login* Monitor**

Este *Login* es básicamente igual al del usuario, con dos únicos cambios de que un monitor no se puede registrar desde la interfaz; además, incluimos un enlace para usuarios que hayan hecho clic sin intención.

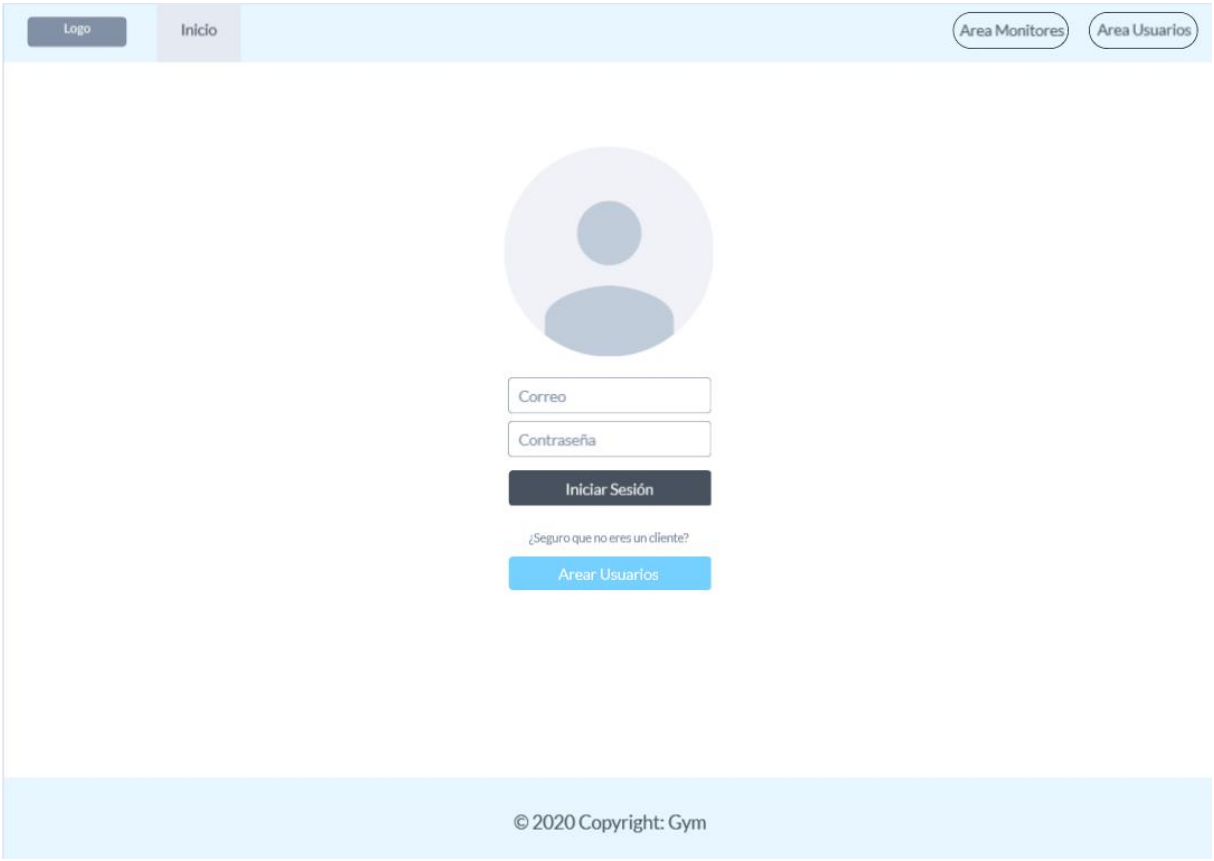


Ilustración 30: *Login* monitor

- ***Lista* usuarios**

En este *list*ado aparecen los usuarios que están registrados en la aplicación. Aquí tenemos la funcionalidad de ver información detallada del usuario, editar el usuario y borrar al usuario.

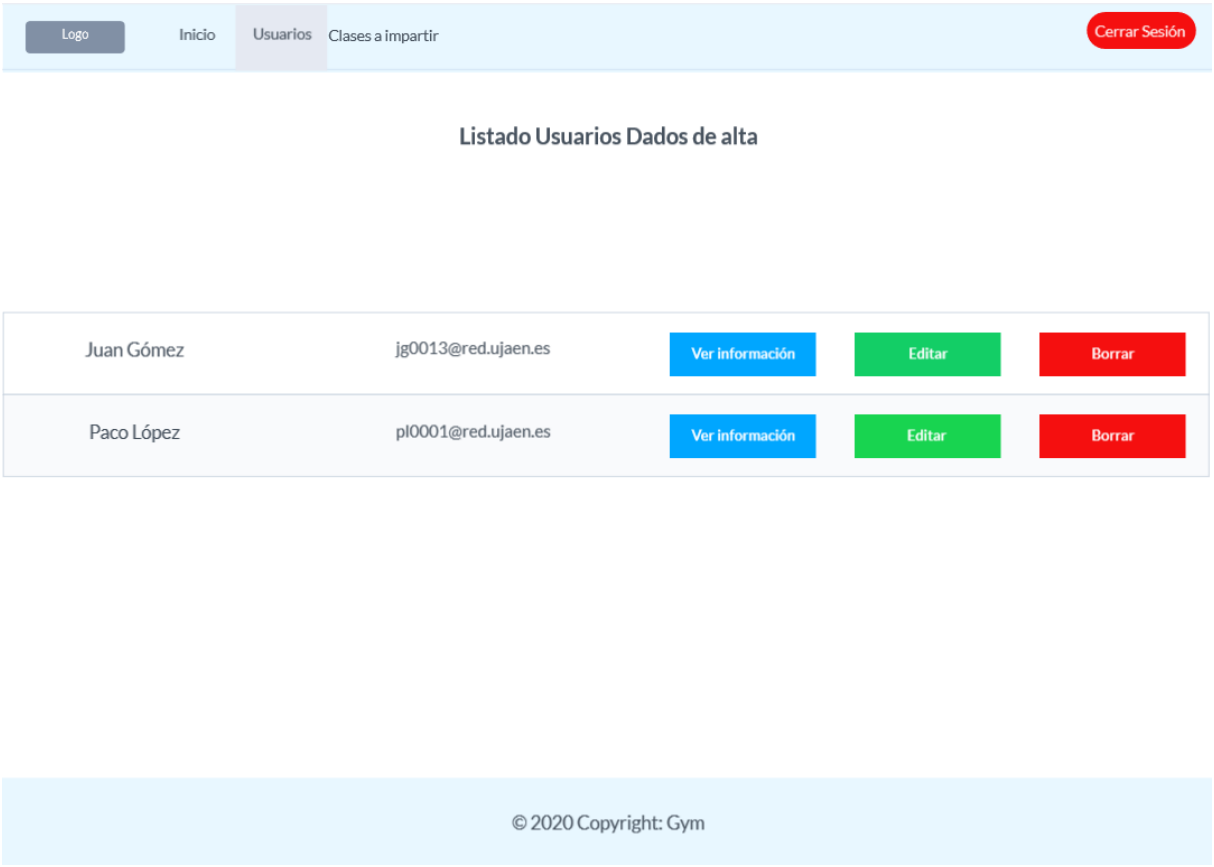


Ilustración 31: *Lista* usuarios

- ***Lista* clases a impartir**

El principal cambio que se observa al iniciar sesión como monitor es en la navegación, donde solo aparecen Inicio, Usuarios y Clases a impartir. Esta vista es muy importante, añade la funcionalidad de *list*ar los participantes de las clases, editar clase y borrar la clase.

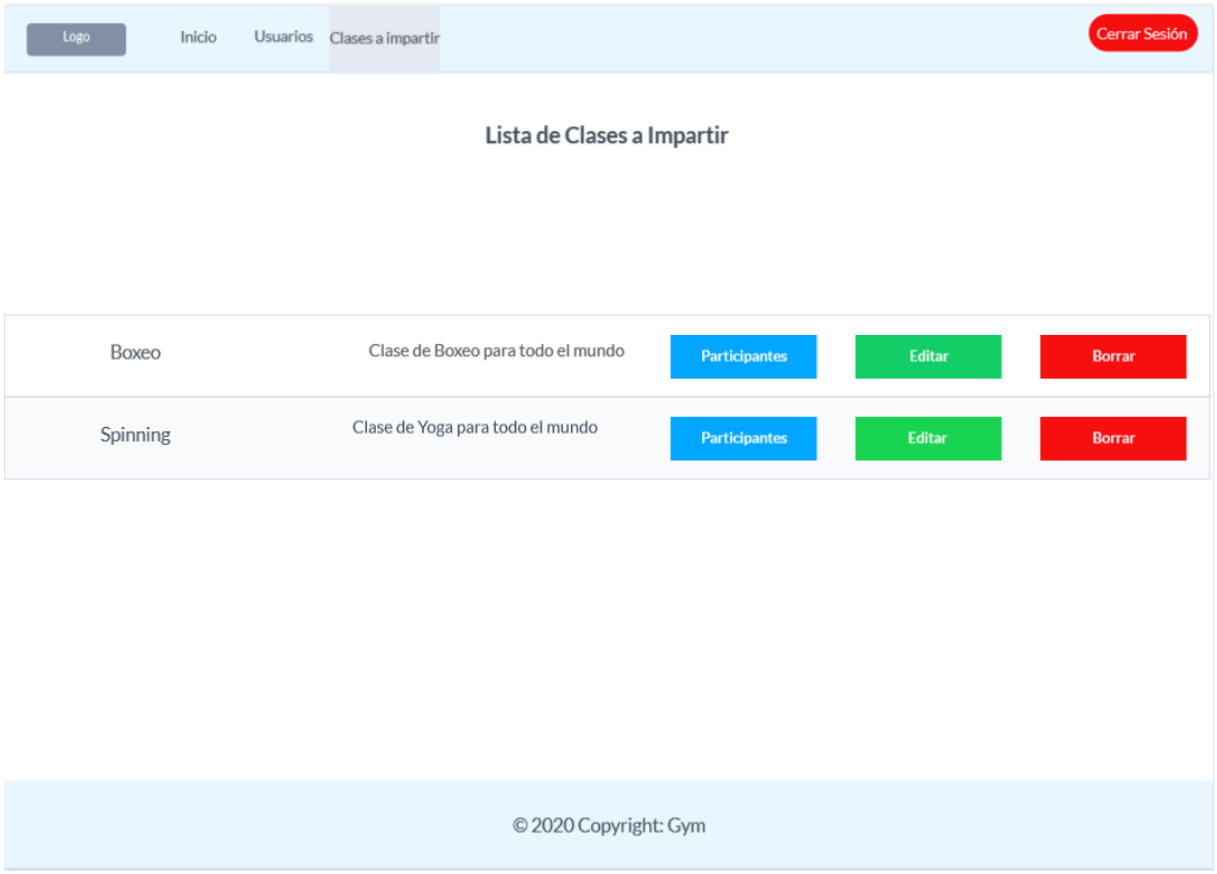


Ilustración 32: *Lista* clases a impartir

- ***Lista* Participantes de una clase**

Cuando clicamos en participantes de la clase que queremos, nos redirige a un *list*ado de participantes, donde podremos borrar alguno si el monitor lo desea.

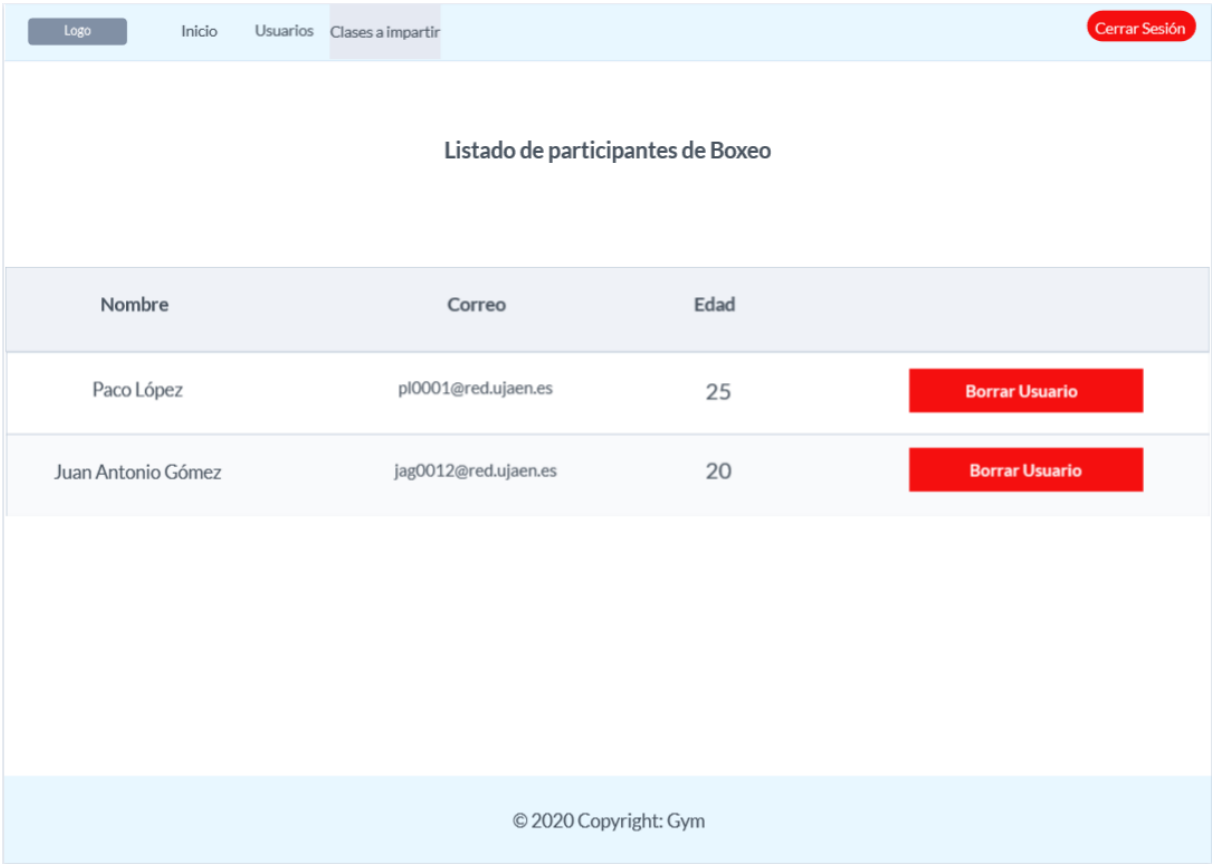


Ilustración 33: *Lista* Participantes de una clase

- **Editar Clase**

A esta vista accedemos mediante el botón de editar clase, de las clases a impartir por el monitor. Esta vista es un formulario que recoge los datos actuales de la clase y podemos o bien guardar los cambios o bien cancelarlos.

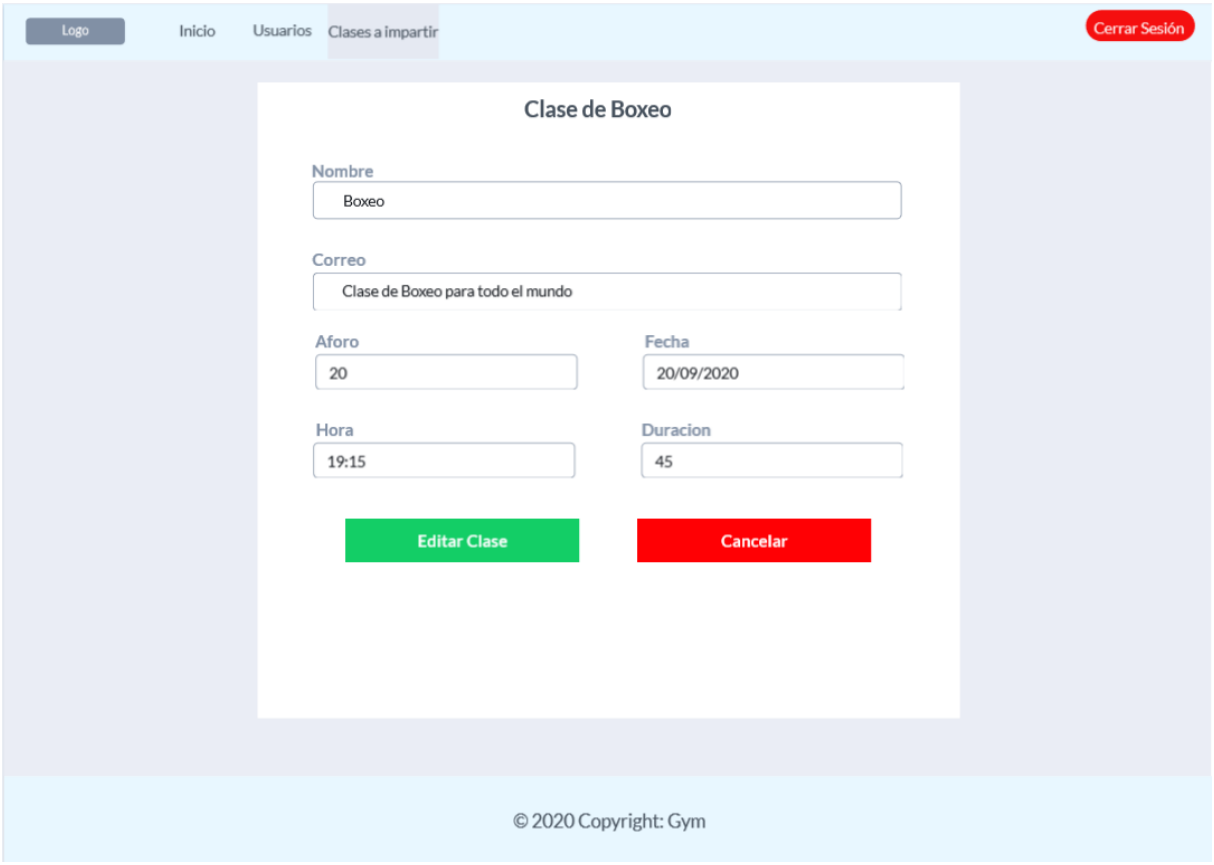


Ilustración 34: Editar Clase

3.6. Plan de pruebas

Para establecer el plan de pruebas correctamente, hemos tomado una serie de decisiones que expondremos en el siguiente apartado. En el *Backend* nos hemos decantado por un desarrollo basado en *TDD*, mientras que en el *Frontend* hemos decidido decantarnos por la aceptación de criterios una vez desarrollada la historia de usuario.

El *TDD* [17] o *test-drive-development* es una práctica de programación que utiliza la siguiente metodología: escribir las pruebas primero y refactorizar el código.

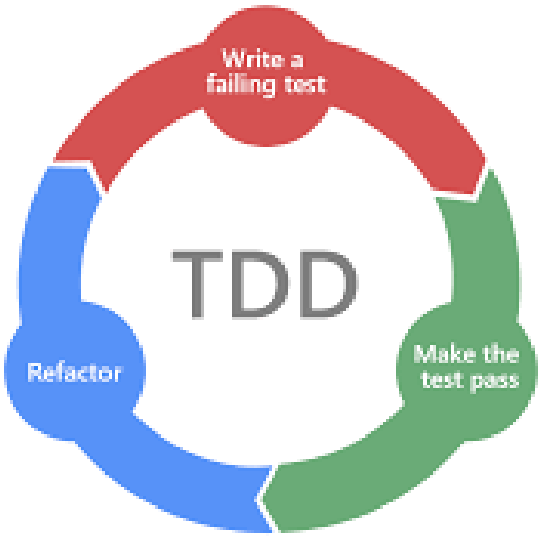


Ilustración 35: *TDD* [18]

Para el desarrollo de *TDD* hemos decidido utilizar la librería de Junit[8] para la que hemos encontrado numerosas herramientas que faciliten el desarrollo de estos tests. La metodología de creación de estos test ha sido realizar el caso base o exitoso primero. A partir de ahí, continuar por los demás escenarios posibles.

Para el *Backend* en mi opinión creo que es de vital importancia utilizar estos tests, ya que observar el comportamiento que realiza tu aplicación frente a diversos escenarios, distintas entradas, etc.

En el *Frontend* me he basado en garantizar la aceptación de criterios después de haber realizado las historias de usuario, por lo tanto, todas las pruebas que se desarrollen sobre el *Frontend* se realizarán teniendo en cuenta los criterios de aceptación definidos en las historias de usuario.

[8] Junit: podemos observas más información aquí https://junit.org/junit5/

4. Implementación

Dentro del apartado de implementación, vamos a tratar varios aspectos tales como: herramientas que he utilizado para el desarrollo, diagrama arquitectónico del sistema que hemos desarrollado, detalles a destacar sobre la implementación de nuestra app.

4.1. Herramientas de desarrollo

4.1.1. Eclipse

Eclipse es un entorno de desarrollo compuesto por un conjunto de herramientas de programación de código abierto multiplataforma. Este entorno ha sido utilizado para el desarrollo del *Backend*, ya que es particularmente útil debido a la gran cantidad de plugins disponibles, de acceso inmediato.

Ilustración 36: Logo de Eclipse

4.1.2. Docker

Docker es una aplicación que permite automatizar el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. En el proyecto ha sido utilizado para crear un contenedor de MySQL y alojar nuestra base de datos.

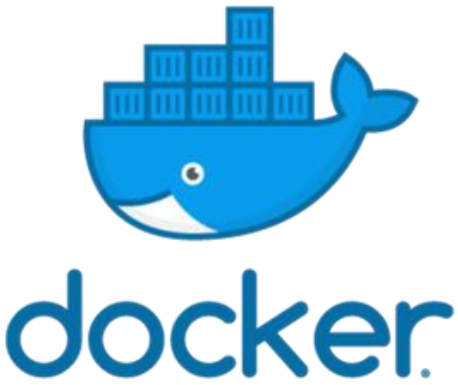


Ilustración 37: Logo de Docker

4.1.3. MySQL

MySQL es un sistema de gestión de bases de datos relacionales, que nos permite almacenar y administrar datos utilizando tablas, vistas, funciones, etc. MySQL ha sido utilizado en el proyecto para la gestión de datos del *Backend*.

Ilustración 38: Logo de MySQL

4.1.4. Spring Boot

Spring Boot es una estructura que elimina la mayor parte del trabajo realizado para la configuración de las aplicaciones basadas en el *framework* Spring. En este proyecto ha sido utilizado para el desarrollo del *Backend*.

Ilustración 39: Logo de Spring Boot

4.1.5. Angular

Angular es un *framework* para aplicaciones web desarrollado en TypeScript, para el desarrollo de aplicaciones web, móviles web, móviles nativas y escritorio nativas. En este proyecto este *framework* ha sido utilizado para el desarrollo de la parte del *Frontend*.

Ilustración 40: Logo de Angular

4.2. Arquitectura

El diagrama arquitectónico del sistema trata de ofrecernos una visión resumida del propio sistema. Tratamos que, de un simple vistazo, cualquier persona pueda entender su funcionamiento [19].

Como ya hemos indicado anteriormente, esta aplicación se compone de dos partes: el *Backend* y el *Frontend*.

4.2.1. Diseño arquitectónico *Backend*

A continuación, voy a comentar algunos aspectos claves para entender el funcionamiento de nuestro *Backend*, que se ha desarrollado como ya he comentado anteriormente, con Spring Boot.

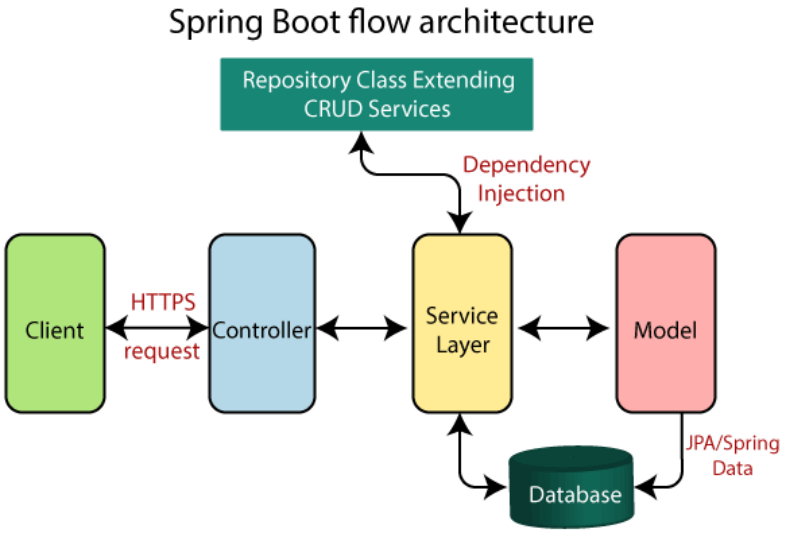


Ilustración 41: Arquitectura *Backend* [20]

Como podemos observar en la ilustración 40, cuando el cliente necesita algún recurso del *Backend*, se comunica a través de llamadas *HTTPS*[9] [21]. A la misma vez, es el controlador el que devuelve la respuesta de la aplicación. Dentro del controller, se encuentran todas las funcionalidades implementadas que internamente también trabaja con todas las clases. *Service Layer*, junto con *Model*, es el encargado de realizar las operaciones *CRUD*[10], en la BBDD. Más adelante mostraremos algunos detalles de cómo se utilizan todos estos aspectos desde *SpringBoot*.

4.2.2. Diseño arquitectónico *Frontend*

Seguidamente, continúo con las características sobre el funcionamiento del *Frontend*.

[9] HTTPS: HTTPS es un protocolo destinado a la transferencia segura de datos de hipertexto.
[10] CRUD: Trata sobre las operaciones crear, leer, actualizar y borrar, que se usan para referirse a las operaciones básicas de la BBDD.

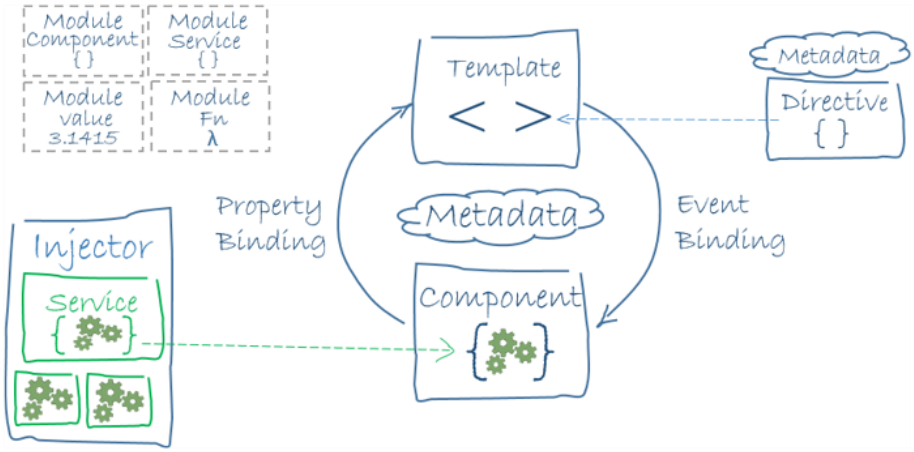


Ilustración 42: Arquitectura *Frontend* [21]

Como ya hemos indicado anteriormente en el punto 3.2.2, Angular está compuesto por varios tipos de elementos: componentes, plantillas, decoradores, servicios, etc.. La vista está compuesta por el componente y la plantilla. La plantilla es una clase *HTML*, donde damos el aspecto visual a la vista. El componente es el que controla a esta vista. Al componente se le inyectan los servicios, que son los encargados de realizar las peticiones *HTTPS* al servidor para obtener los recursos necesarios o emprender las acciones que necesite realizar esta vista [21]. En el siguiente apartado veremos algunos detalles sintácticos del *Framework* para su correcta utilización.

4.3. Detalles sobre implementación

A continuación, comenzamos con los detalles de implementación a tener en cuenta dentro de nuestro desarrollo. Estos detalles como ya hemos indicado anteriormente, al haber dos proyectos (Frontend y *Backend*), he decidido dividir estos detalles en dos.

4.3.1. *Backend*

Empezamos por lo esencial, la configuración de la BBDD. Este archivo denominado *application.yml,* se encuentra situado en la carpeta resources. Indica la configuración de nuestra BBDD, que en este caso está implementada en un contenedor Docker.

```yaml
## Application YML
---
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.driver-class-name: com.mysql.cj.jdbc.Driver
spring.datasource.url: jdbc:mysql://${MYSQL_HOST:localhost}:3306/backend
spring.datasource.username: usuario
spring.datasource.password: password
---
# Estrategia de mantenimiento del esquema
spring.jpa.properties.javax.persistence.schema-generation.database.action: drop-and-create
```

Ilustración 43: Configuración de BBDD

Otra parte esencial en nuestro proyecto, que está relacionada con nuestra BBDD son los *DAOs*. Estos extienden *JPARepository* y son los encargados de comunicarse con la BBDD, mediante las operaciones *CRUD*. En la siguiente imagen podemos observar como facilita, cualquier tipo de búsqueda ya que solo tienes que indicar cómo quieres que busque y que tipo de dato necesitas de retorno.

```java
package es.uja.curso1920.DAOs;

import java.time.LocalDateTime;
import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import es.uja.curso1920.objetos.Clase;

public interface ClaseDAO extends JpaRepository<Clase, Integer> {

    Clase findByNombreEquals(String nombre);

    Clase findByNombreEqualsAndFechaEquals(String nombre, LocalDateTime fecha);

    List<Clase> findByDescripcionContainsIgnoreCase(String descripcion);

    List<Clase> findByNombreContainsIgnoreCase(String nombre);


}
```

Estos *DAOs*, son inyectados en el SistemaGym*Bean* para ser utilizado *post*eriormente.

```java
// DAO de usuario
@Autowired
UsuarioDAO usuariosDAO;
// DAO de clase
@Autowired
ClaseDAO clasesDAO;
// DAO de monitor
@Autowired
MonitorDAO monitoresDAO;
```

Ilustración 44: Inyección de DAOs

```java
@Override
@Transactional
public UsuarioDTO crearUsuario(UsuarioDTO usuarioDTO) throws UsuarioRepetidoEnSistema {
    Usuario usuario = findUser(usuarioDTO.getCorreo());
    if (usuario != null) {
        throw new UsuarioRepetidoEnSistema();
    } else {
        usuario = modelMapper.map(usuarioDTO, Usuario.class);
        usuario.calculoKcals(0);
        usuario.setFechaInicio(LocalDate.now());
        usuariosDAO.save(usuario);
        usuarioDTO = modelMapper.map(usuario, UsuarioDTO.class);
        return usuarioDTO;
    }

}
```

Ilustración 45: Crear Usuario (SistemaGymBean)

Continuamos con los *DTOs*, estos objetos transportan datos y son utilizados para la comunicación entre cliente y servidor [22]. Además de lo indicado en la foto contiene los métodos *getters* y *setters*.

```java
public class MonitorDTO implements Serializable {

    private static final long serialVersionUID = -1263730089727502496L;
    @NotNull
    private int id;
    @NotNull
    @Email
    @Size(min = 0, max = 200)
    private String correo;
    @NotNull
    @Size(min = 0, max = 200)
    private String contraseña;

    public MonitorDTO() {
        super();
    }

    public MonitorDTO(String correo, String contraseña) {
        super();
        this.id = 0;
        this.correo = correo;
        this.contraseña = contraseña;
    }
```

Ilustración 46: Monitor*DTO*

Para el mapeo de *DTO* a entidad y viceversa, utilizo el *ModelMapper*. Este es inyectado como los *DTOs*.

```java
// ModelMapper para mapeo DTO-Entidad
@Autowired
private ModelMapper modelMapper;
```

Ilustración 47: ModelMapper

Así pasamos de *DTO* a entidad.

```
usuario = modelMapper.map(usuarioDTO, Usuario.class);
```

Ilustración 48: DTO a entidad

Así pasamos de entidad a *DTO*.

```
usuarioDTO = modelMapper.map(usuario, UsuarioDTO.class);
```

Ilustración 49: Entidad a DTO

A la hora de lanzar excepciones, utilizo excepciones propias. Estas han sido configuradas para poder recibirlas desde el servicio *REST* con la anotación ResponseStatus.

```
@ResponseStatus(code = HttpStatus.NOT_FOUND, reason = "Error, Clase no encontrada")
public class ClaseNoEncontrada extends RuntimeException {

    private static final long serialVersionUID = -5620238526989473391;

}
```

Ilustración 50: Excepción ClaseNoEncontrada

Finalizo con la clase que lanza el contexto de aplicación. Aquí podemos ver cómo se indican qué paquetes deben escanearse para localizar dónde se encuentra lo necesario para que funcione el proyecto.

```
@SpringBootApplication(scanBasePackages = { "es.uja.curso1920.beans", "es.uja.curso1920.REST" })
@EntityScan(basePackages = "es.uja.curso1920.objetos")
@EnableJpaRepositories(basePackages = "es.uja.curso1920.DAOs")
public class App {
    @Bean
    public ModelMapper modelMapper() {
        return new ModelMapper();
    }

    Run | Debug
    public static void main(String[] args) {
        // Iniciar contenedor de Spring y obtener contexto
        SpringApplication.run(App.class, args);
    }
```

Ilustración 51: Clase App

*Poste*riormente he desarrollado un *CommanLineRunner* para que en cuanto arranquemos la aplicación, se creen las siguientes entidades con el fin de facilitar una prueba.

```java
@Bean
public CommandLineRunner initApp(SistemaGym sistemaGym) {
    return (args) -> {
        sistemaGym.crearUsuario(new UsuarioDTO("Juan Hernández", "juan@red.ujaen.es", "juan", 78, 180, 20,
                Sexo.Hombre, Constantes.FACTOR_MUY_FUERTE_ACTIVIDAD, Objetivo.Perder_grasa));
                sistemaGym.crearUsuario(new UsuarioDTO("Francisco Liébana", "francisco@red.ujaen.es", "francisco", 65,
                Sexo.Hombre, Constantes.FACTOR_MODERADO_ACTIVIDAD, Objetivo.Ganar_musculo));
        MonitorDTO monitorCreado = sistemaGym.crearMonitor(new MonitorDTO("monitor1@red.ujaen.es", "secreto"));
        LocalDateTime date = LocalDateTime.of(2020, 8, 27, 17, 30);
        ClaseDTO claseDTO = sistemaGym.crearClase(monitorCreado,
                new ClaseDTO("Boxeo", "Clase para aprender a boxear dentro de un cuadrilátero", date, 10, 45));
        sistemaGym.crearClase(monitorCreado,
                new ClaseDTO("Yoga", "Clase para conseguir relajarte totalmente gracias al Yoga",
                        claseDTO.getFecha().plusDays(1), 1, 30));
    };
}
```

Ilustración 52: CommanLineRunner

4.3.2. *Frontend*

Dentro de los detalles sobre la implementación, en el *Frontend* me gustaría comenzar detallando la estructura de contenidos del proyecto. Dentro de este proyecto, he decidido clasificar los archivos en 4 carpetas: clases, componentes, enum y servicios.

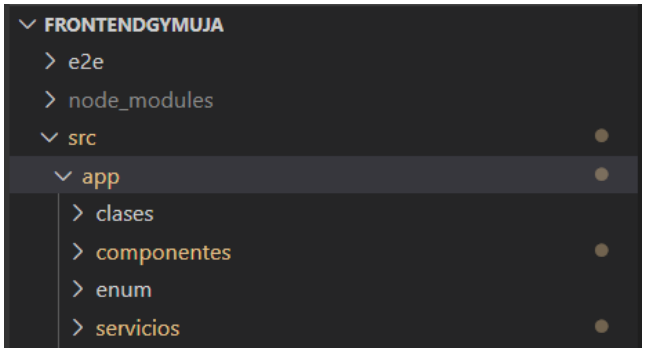


Ilustración 53: Estructura de contenidos del proyecto (Frontend)

En la carpeta clases podemos encontrar las entidades que necesitamos para comunicarnos con el *Backend*, ya sea para recibir como para enviar.

```
import { Sexo } from '../enum/sexo.enum'
import { Objetivo } from '../enum/objetivo.enum'

export class Usuario {
  id:number
  nombre:String
  correo:String
  contrasena:String
  peso:Float32Array
  estatura:number
  edad:number
  sexo:Sexo
  coeficienteActividad:Float32Array
  objetivo:Objetivo
  revisionNormo:number
  kcals:number
  proteinas:number
  carbohidratos:number
  grasas:number

}
```

Ilustración 54: Clase Usuario (Frontend)

En la carpeta servicios podemos encontrar los servicios de monitor, de clase y de usuario. Estos servicios son los encargados de realizar las llamadas *HTTP* al *Backend*. Estos son alguno de los servicios de usuario:

```
registro(usuario:Usuario): Observable<any>{
 return this.http.post(this.rootPath, usuario)
}

getUsuario():Observable<Usuario>{
  return this.http.get<Usuario>(this.rootPath + "/" + this.correoUsuario)
}

obtenerUsuario(correoUsuario:String):Observable<Usuario>{
  return this.http.get<Usuario>(this.rootPath+ "/" + correoUsuario)
}

editarUsuario(usuario:Usuario):Observable<Usuario>{
  return this.http.put<Usuario>(this.rootPath + "/" + this.correoUsuario, usuario)

}
```

Ilustración 55: Servicios Usuario (Frontend)

Los componentes están formados por un archivo .css que es el estilo, *.html* que es la vista, .spect.ts que son utilizados para el testeo y .ts que es el encargado de realizar las peticiones, navegar entre vistas, etc.

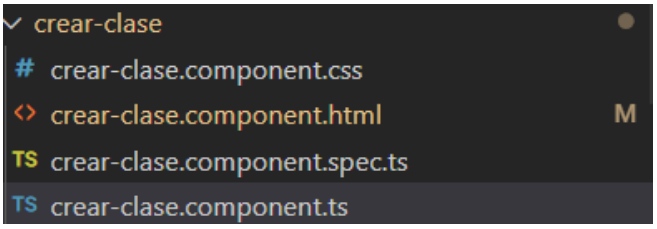


Ilustración 56: Componente crear clase (Frontend)

La estructura del fichero .ts es la siguiente: primero declaramos las variables a utilizar, en este caso necesitaremos los atributos de una clase; a continuación definimos el constructor, donde declaramos los servicios a utilizar, en este caso los de monitor y el Router, para redirigir *poste*riormente. Para finalizar, creamos la función crearClase que crea una Clase con los datos recogidos del formulario e invoca al crearClase de monitorServicios.

```
export class CrearClaseComponent implements OnInit {
  public nombre:String
  public descripcion:String
  public maxParticipantes:number
  public duracion:number

  public  fecha:Date
  public hora:Time

  constructor(private monitorServicios:ServiciosMonitorService, private ruta:Router) { }

  ngOnInit() {
  }

  crearClase(){
    var fechaFinal = new Date(this.fecha + ' ' +  this.hora)

    let clase:Clase = new Clase()
    clase.nombre = this.nombre
    clase.descripcion = this.descripcion
    clase.fecha = fechaFinal
    clase.maxParticipantes = this.maxParticipantes
    clase.duracion = this.duracion

    this.monitorServicios.crearClase(clase).subscribe(data =>{
      this.ruta.navigate(['/clasesImpartir'])
    })
  }

}
```

Ilustración 57: Componente crearclase.ts (Frontend)

```
crearClase(clase:Clase): Observable<any>{
  return this.http.post(this.rootPath + "/" + this.correoMonitor + "/clase", clase)
}
```

Ilustración 58: crearClase de monitorServicios (Frontend)

Para recoger los atributos del formulario hemos utilizado ngModel.

```
<div class="form-group">
    <label for="claseNombre">Título de la clase</label>
    <input [(ngModel)]='nombre' type="text" class="form-control " name="claseNombre" id="claseNombre" placeholder="" />
</div>
<div class="row">
    <div class="form-group col-md">
        <label for="claseAforo">Aforo de la clase</label>
        <input [(ngModel)]='maxParticipantes' type="number" class="form-control" name="claseAforo" id="claseAforo" min="
    </div>
    <div class="form-group col-md">
        <label for="claseFecha">Fecha de la clase</label>
        <input [(ngModel)]='fecha' class="form-control" type="date" value="2020-08-19" name="claseFecha" id="claseFecha
    </div>
    <div class="form-group col-md">
        <label for="claseHora">Hora de la clase</label>
        <input [(ngModel)]='hora' class="form-control" type="time" value="18:00" name="claseHora" id="claseHora">
    </div>
    <div class="form-group col-md">
        <label for="claseDuracion">Duración</label>
        <input [(ngModel)]='duracion' type="number" class="form-control" name="claseDuracion" id="claseDuracion" min="0
    </div>
</div>
<div class="form-group">
```

Ilustración 59: Componente crearclase.html (Frontend)

Para finalizar vamos a hablar sobre el app-routing, como su nombre indica es donde inyectamos los componentes y les asignamos la ruta para acceder a ellos.

```
const routes: Routes = [
  { path: '', component: InicioComponent},
  {path: 'clase/:idClase', component: VerClaseComponent},
  {path: 'crearClase', component: CrearClaseComponent},
  {path: 'crearUsuario', component: CrearUsuarioComponent},
  {path: 'login', component:LoginComponent},
  {path: 'loginMonitor', component:MonitorLoginComponent},
  {path: 'clasesImpartir', component:ListadoClasesImpartirComponent},
  {path: 'listarParticipantes/:idClase', component:ListadoParticipantesComponent},
  {path: 'usuarioInicio', component:UsuarioInicioComponent},
  {path: 'usuarioClasesApuntadas', component:UsuarioClasesComponent},
  {path: 'editarClase/:idClase',component:EditarClaseComponent},
  {path: 'usuarioPerfil', component: UsuarioPerfilComponent},
  {path: 'usuarioEntrenamiento', component:UsuarioEntrenamientoComponent},
  {path: 'usuariosRegistrados', component:ListaUsuariosRegistradosComponent},
  {path: 'usuarioPerfil/:correoUsuario', component: UsuarioPerfilComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Ilustración 60: App-rounting (Frontend)

5. Pruebas

A la hora de realizar las pruebas a la aplicación, he decidido hacer la parte del *Backend* con Junit, mientras que en el *Frontend* me he basado en, tras realizar el desarrollo de lo necesario en el *Frontend* (Vista y servicio), revisar los criterios de aceptación para que se cumplan correctamente esta historia de usuario.

5.1. Junit

Junit es un conjunto de librerías desarrolladas para testear el funcionamiento de las clases y métodos que componen nuestra aplicación, para poder observar que se comporta como debe ante distintas situaciones. En nuestro proyecto ha sido utilizado para el testeo de SistemaGym*Bean* y de SistemaGym*REST*, aunque esto a la vez testea las demás clases, ya que se apoya en ellas para su correcto funcionamiento.

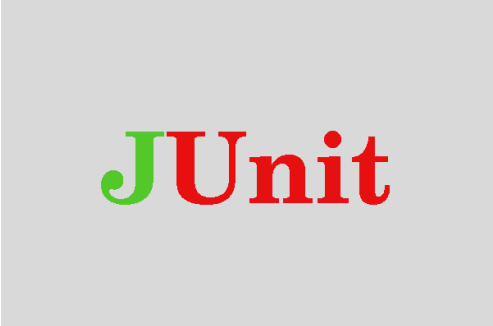


Ilustración 61: Logo Junit

5.1.1. Configuración de Junit

Primero, debemos añadir la dependencia al pom.xml.

```
<!-- JUnit -->
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <scope>test</scope>
</dependency>
```

Ilustración 62: Dependencia Junit

Lo siguiente, tras crear la clase donde implementaremos los tests, en este caso los tests de integración del SistemaGym*Bean*, añadimos las siguientes anotaciones.

```
@SpringBootTest(classes = { App.class })
@DirtiesContext(classMode = ClassMode.BEFORE_EACH_TEST_METHOD)
@TestInstance(Lifecycle.PER_CLASS)
Run Test | Debug Test
public class SistemaGymIntegrationTest {
```

Ilustración 63: Configuración SistemaGymIntegrationTest

***SpringBoot*Test:** Es utilizado para indicar la clase utilizada para iniciar el contexto de aplicación, siendo utilizado para una prueba. [24]

DirtiesContext: Esta anotación la utilizo para indicar que la prueba modifica el contexto de aplicación, diciendo al marco de prueba que cierre y vuelva a crear el contexto para las siguientes pruebas [25].

TestInstance: Esta anotación nos permite configurar el ciclo de vida de las pruebas [26].

Lo siguiente seria la inyección del *bean* en la clase donde vamos a realizar los tests.

```
public class SistemaGymIntegrationTest {

    @Autowired
    SistemaGym sistemaGym;
```

Ilustración 64: Inyección de SistemaGym

5.1.2. Pruebas de SistemaGym*Bean*

La clase SistemaGymIntegrationTest, como su propio nombre indica, trata sobre la realización de test de integración. Esta clase comienza por una primera parte donde defino los *DTOs* a utilizar en estos tests, como podemos observar en la imagen.

```
UsuarioDTO usuarioTest1 = new UsuarioDTO("Juan Hernández", "juan@red.ujaen.es", "juan", 78, 180, 20, Sexo.Hombre,
        FACTOR_LIGERO_ACTIVIDAD, Objetivo.Ganar_musculo);
UsuarioDTO usuarioTest1Correo = new UsuarioDTO("Pepe", "juan@red.ujaen.es", "pepe", 78, 180, 20, Sexo.Hombre,
        FACTOR_LIGERO_ACTIVIDAD, Objetivo.Ganar_musculo);
UsuarioDTO usuarioTest2 = new UsuarioDTO("Paco López", "paco@red.ujaen.es", "paco", 120, 190, 30, Sexo.Hombre,
        FACTOR_MODERADO_ACTIVIDAD, Objetivo.Perder_grasa);
UsuarioDTO usuarioTest3 = new UsuarioDTO("Juan Antonio Rubio", "juanantonio@red.ujaen.es", "juanantonio", 60, 170,
       27, Sexo.Hombre, FACTOR_FUERTE_ACTIVIDAD, Objetivo.Ganar_musculo);

MonitorDTO monitorTest1 = new MonitorDTO("monitor1@red.ujaen.es", "secreto");
MonitorDTO monitorTest1Correo = new MonitorDTO("monitor1@red.ujaen.es", "secreto");
MonitorDTO monitorTest2 = new MonitorDTO("monitor2@red.ujaen.es", "secreto");

ClaseDTO claseTest1 = new ClaseDTO("Boxeo", "Clase para aprender a boxear", LocalDateTime.now(), 10, 45);
ClaseDTO claseTest2 = new ClaseDTO("Yoga", "Clase para conseguir relajarte totalmente gracias al Yoga",
       claseTest1.getFecha().plusDays(1), 1, 30);
ClaseDTO claseTest3 = new ClaseDTO("Pilates", "Clase para conseguir estirar, fortalecer y equilibrar el cuerpo ",
       claseTest1.getFecha().plusDays(2), 2, 35);
```

Ilustración 65: Definición de *DTOs* a utilizar

Como ya indiqué en el apartado 3.6., mi metodología para el *TDD* es probar el caso base o correcto y *poste*riormente casos donde deban lanzarse excepciones para observar el correcto funcionamiento.

Esta clase la componen 14 tests, cada uno desarrollado para cada funcionalidad del *bean*. Y esto es el resultado de la ejecución completa de esta clase.

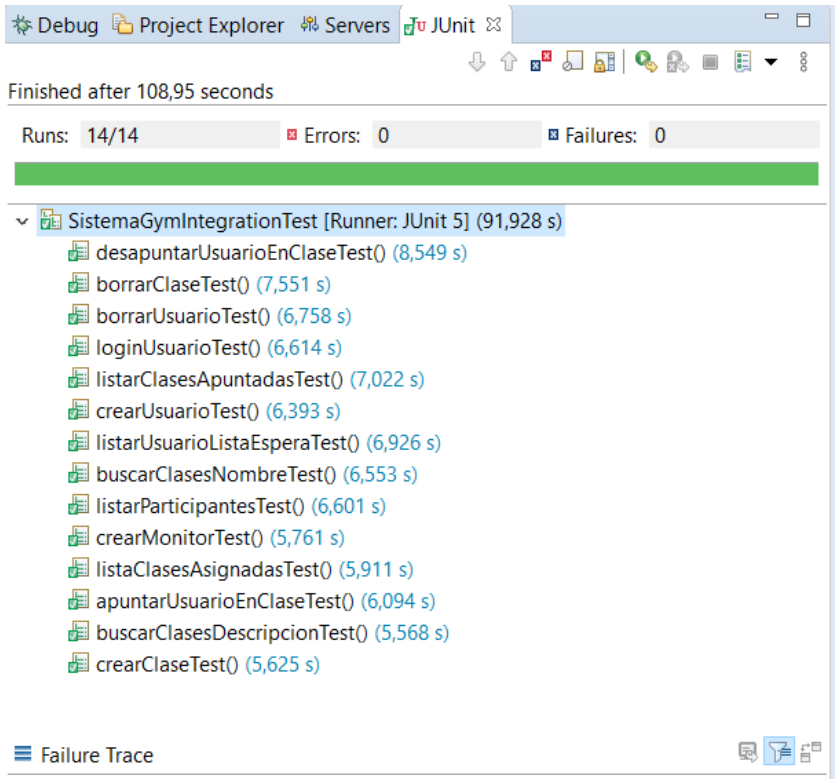


Ilustración 66: Resultado tests integración

Pasamos a ver más en detalle estos tests, solo viendo algunos de los más complejos, ya que los demás serían bastantes similares.

Comenzaremos por un test sencillo, crearClaseTest. Para poder crear una clase, debemos crear posteriormente un monitor, ya que es impartida por este. Observamos En primer lugar el caso base, el cual no provoca ninguna excepción y se crea todo correctamente. Después pasamos al caso donde queremos crear una clase, pero el monitor no existe, por lo tanto, lanzaría una excepción. Seguidamente, el monitor que ha creado claseTest1, quiere establecer otra clase, claseTest4, diez minutos después que la anterior. Como podemos ver en la ilustración 66, la claseTest1 tiene una duración de cuarenta y cinco minutos, por lo tanto, este monitor no estaría disponible. Para finalizar el mismo monitor, quiere crear una clase ya existente.

```
@Test
Run Test | Debug Test | ✓
void crearClaseTest() {
    sistemaGym.crearMonitor(monitorTest1);
    sistemaGym.crearClase(monitorTest1, claseTest1);

    // Monitor inexistente
    Assertions.assertThrows(MonitorNoEncontrado.class, () -> sistemaGym.crearClase(monitorTest2, claseTest2));

    // Monitor no disponible debido a que tiene una clase a la misma hora o durante
    // esa hora
    ClaseDTO claseTest4 = new ClaseDTO("Pilates", "Consigue estirar, fortalecer y equilibrar el cuerpo ",
            claseTest1.getFecha().plusMinutes(10), 2, 35);
    Assertions.assertThrows(MonitorNoDisponible.class, () -> sistemaGym.crearClase(monitorTest1, claseTest4));

    // Clase ya existente
    Assertions.assertThrows(ClaseRepetidaEnSistema.class, () -> sistemaGym.crearClase(monitorTest1, claseTest1));
}
```

Ilustración 67: Ejemplo de crearClaseTest

Pasamos a un test más complejo, el borrado de un usuario en el sistema. Comenzamos con el caso base, es decir, eliminar un usuario que solo se ha creado. Pasamos a un usuario que se ha apuntado a otras clases, en este caso claseTest1. Comprobamos que se ha apuntado correctamente con el primer *assertEquals* y finalmente lo borramos. *Poste*riormente, volvemos a comprobar si hay algún participante en la clase y nos encontramos con que ha borrado correctamente al usuario. El último caso es cuando queremos borrar un usuario que no existe.

```java
@Test
Run Test | Debug Test | ✓
void borrarUsuarioTest() {
    sistemaGym.crearUsuario(usuarioTest1);
    sistemaGym.borrarUsuario(usuarioTest1);

    // Al borrar el usuario, debe borrarse como participante de todas las clases
    sistemaGym.crearUsuario(usuarioTest1);
    sistemaGym.crearMonitor(monitorTest1);
    ClaseDTO aux1 = sistemaGym.crearClase(monitorTest1, claseTest1);

    //Apuntamos a la claseTest1 el usuarioTest1
    sistemaGym.apuntarUsuarioEnClase(usuarioTest1, aux1);
    Assertions.assertEquals(1, sistemaGym.listarParticipantes(aux1).size());

    //Borramos al usuario
    sistemaGym.borrarUsuario(usuarioTest1);
    Assertions.assertEquals(0, sistemaGym.listarParticipantes(aux1).size());

    // Borrar un usuario inexistente
    Assertions.assertThrows(UsuarioNoEncontrado.class, () -> sistemaGym.borrarUsuario(usuarioTest1));
}
```

Ilustración 68: Ejemplo de borrarUsuarioTest

Como ya he comentado anteriormente, esta es la metodología que he seguido durante todos los tests. A continuación, paso a mostrar los tests del servicio *REST*.

5.1.3. Pruebas de SistemaGym*Rest*

La clase SistemaGymRESTTest, como su propio nombre indica, trata sobre la realización de test de integración. Esta clase comienza por una primera parte donde defino los *DTOs* a utilizar en estos tests, como podemos observar en la imagen.

```java
UsuarioDTO usuarioTest1 = new UsuarioDTO("Juan Hernández", "juan@red.ujaen.es", "juan", 78, 180, 20, Sexo.Hombre,
        FACTOR_LIGERO_ACTIVIDAD, Objetivo.Ganar_musculo);
UsuarioDTO usuarioTest2 = new UsuarioDTO("Paco López", "paco@red.ujaen.es", "paco", 120, 190, 30, Sexo.Hombre,
        FACTOR_MODERADO_ACTIVIDAD, Objetivo.Perder_grasa);
UsuarioDTO usuarioTest3 = new UsuarioDTO("Juan Antonio Rubio", "juanantonio@red.ujaen.es", "juanantonio", 60, 170,
        27, Sexo.Hombre, FACTOR_FUERTE_ACTIVIDAD, Objetivo.Ganar_musculo);
// Usuario con idInexistente
UsuarioDTO usuarioTest4 = new UsuarioDTO("Juan Manuel Ramos", "juanmanuel@red.ujaen.es", "juanmanuel", 90, 195, 23,
        Sexo.Hombre, FACTOR_FUERTE_ACTIVIDAD, Objetivo.Ganar_musculo);

MonitorDTO monitorTest1 = new MonitorDTO("monitor1@red.ujaen.es", "secreto");
MonitorDTO monitorTest1Correo = new MonitorDTO("monitor1@red.ujaen.es", "hola");
MonitorDTO monitorTest2 = new MonitorDTO("monitor2@red.ujaen.es", "secreto");
MonitorDTO monitorTest3 = new MonitorDTO("monitor3@red.ujaen.es", "secreto");

ClaseDTO claseTest1 = new ClaseDTO("Boxeo", "Clase para aprender a boxear", LocalDateTime.now(), 10, 45);
ClaseDTO claseTest2 = new ClaseDTO("Yoga", "Clase para conseguir relajarte totalmente gracias al Yoga",
        claseTest1.getFecha().plusDays(1), 1, 30);
ClaseDTO claseTest3 = new ClaseDTO("Pilates", "Clase para conseguir estirar, fortalecer y equilibrar el cuerpo ",
        claseTest1.getFecha().plusDays(2), 2, 35);
```

Ilustración 69: Definición de *DTOs* a utilizar

En este caso, además de necesitar los *DTOs*, incorporo dos atributos más: *localPort*, que es el puerto local generado automáticamente, y *restTemplate* que nos permite personalizar nuestras llamadas *HTTP* al servicio *REST*.

```
@LocalServerPort
int localPort;

TestRestTemplate restTemplate;
```

Ilustración 70: *LocalPort* y *restTemplate*

Para utilizar Test*RestTemplate* es necesario incorporar el siguiente fragmento al pom.xml [27].

```
<!-- Spring Boot -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Ilustración 71: Dependencia de *SpringBoot*Starter

Estos atributos son inicializados en una operación crear*Rest*emplate(). El ID del usuario es para utilizar el caso de que se cree un usuario inexistente.

```
@PostConstruct
public void crearRestTemplate() {
    RestTemplateBuilder restTemplateBuilder = new RestTemplateBuilder()
            .rootUri("http://localhost:" + localPort + "/sistemaGym");
    restTemplate = new TestRestTemplate(restTemplateBuilder);

    usuarioTest4.setId(-1);
}
```

Ilustración 72: Inicialización de atributos *REST*

Esta clase la componen 21 tests, cada uno desarrollado para cada funcionalidad del servicio *REST*. Y esto es el resultado de la ejecución completa de esta clase.

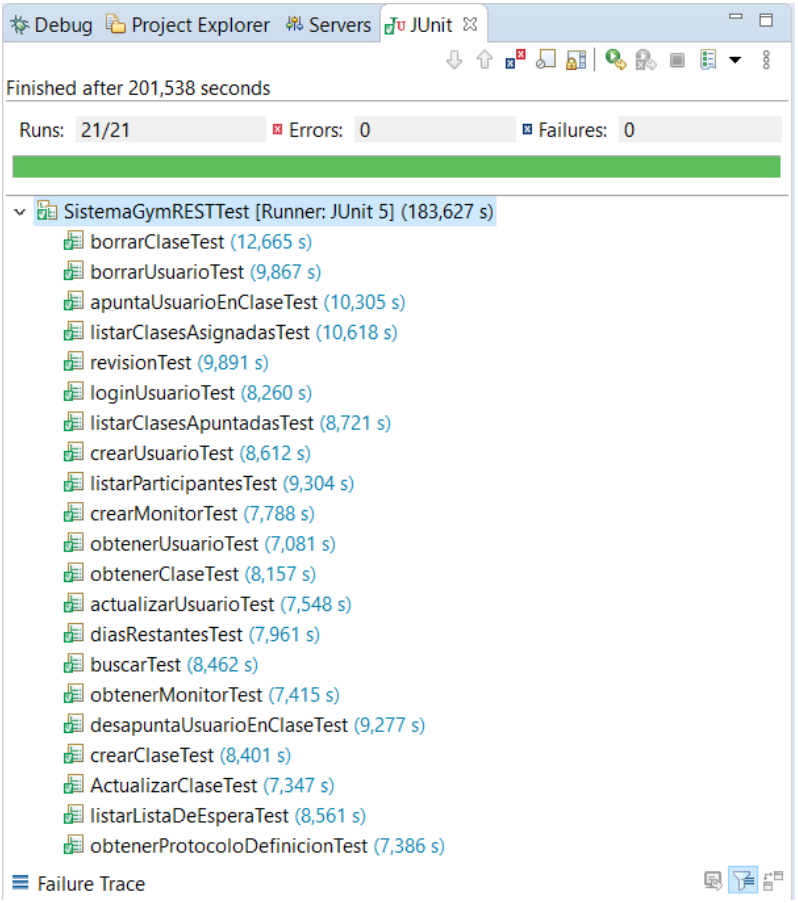


Ilustración 73: Resultado tests del servicio *REST*

Comienzo con un ejemplo sencillo, como es el crearUsuarioTest(). En este test realizamos un *post* con la ruta ya inicializada en la anterior ilustración, añadiendo además '/usuario'; por añadidura le pasamos el usuarioTest1 para crearlo. Con el fin de crear un usuario, en el segundo caso creamos un usuario con el mismo email que el que acabamos de crear, esto por lo tanto genera un error.

```
@Test
Run Test | Debug Test |
public void crearUsuarioTest() {
    ResponseEntity<UsuarioDTO> respuesta1 = restTemplate.postForEntity("/usuario", usuarioTest1, UsuarioDTO.class);
    Assertions.assertEquals(HttpStatus.CREATED, respuesta1.getStatusCode());

    // Crear un usuario con email repetido
    UsuarioDTO usuarioErroneo = new UsuarioDTO("paco", "juan@red.ujaen.es", "paco", 90, 170, 30, Sexo.Hombre,
     FACTOR_LIGERO_ACTIVIDAD, Objetivo.Ganar_musculo);
    ResponseEntity<UsuarioDTO> respuesta2 = restTemplate.postForEntity("/usuario", usuarioErroneo,
            UsuarioDTO.class);
    Assertions.assertEquals(HttpStatus.CONFLICT, respuesta2.getStatusCode());
}
```

Ilustración 74: Ejemplo de crearUsuarioTest

Continuamos con un test más complejo, como puede ser el *login*UsuarioTest(). Básicamente es igual que el anterior, ya que realizamos un *post*, añadiendo a la ruta '/usuario/*login*' y además le pasamos el usuarioTest1 para poder verificar las credenciales. Los siguientes casos son cuando se quiere realizar un *login* de un usuario que no existe; por lo tanto, devuelve un *NOT_FOUND*. Finalizamos este tests para cuando la contraseña del usuario es incorrecta, que devuelve un *CONFLICT*.

```
@Test
Run Test | Debug Test | ✓
public void loginUsuarioTest() {
    ResponseEntity<UsuarioDTO> rUsuarioCreado1 = restTemplate.postForEntity("/usuario", usuarioTest1,
            UsuarioDTO.class);

    // Login Correcto
    ResponseEntity<UsuarioDTO> respuestaCorrecta = restTemplate.postForEntity("/usuario/login", rUsuarioCreado1.getBody(),
            UsuarioDTO.class);
    Assertions.assertEquals(HttpStatus.OK, respuestaCorrecta.getStatusCode());

    // Login Incorrecto, usuarioInexistente
    ResponseEntity<Void> respuestaErronea1 = restTemplate.postForEntity("/usuario/login", usuarioTest4, Void.class);
    Assertions.assertEquals(HttpStatus.NOT_FOUND, respuestaErronea1.getStatusCode());
```

Ilustración 75: Ejemplo de *login*UsuarioTest

```
// Login Incorrecto, contraseña incorrecta
UsuarioDTO usuarioIncorrecto = new UsuarioDTO(rUsuarioCreado1.getBody().getNombre(),
        rUsuarioCreado1.getBody().getCorreo(), "HOLA", rUsuarioCreado1.getBody().getPeso(),
        rUsuarioCreado1.getBody().getEstatura(), rUsuarioCreado1.getBody().getEdad(),
        rUsuarioCreado1.getBody().getSexo(),
        rUsuarioCreado1.getBody().getCoeficienteActividad(), rUsuarioCreado1.getBody().getObjetivo());
usuarioIncorrecto.setId(rUsuarioCreado1.getBody().getId());
ResponseEntity<UsuarioDTO> respuestaErronea2 = restTemplate.postForEntity("/usuario/login", usuarioIncorrecto, UsuarioDTO.class);
Assertions.assertEquals(HttpStatus.CONFLICT, respuestaErronea2.getStatusCode());
```

Ilustración 76: Ejemplo de *login*UsuarioTest (2)

5.2. Pruebas de Front

Como colofón, finalizo con las pruebas realizadas en el *Frontend*, que, como indiqué en el apartado 3.6, voy a basarme en el desarrollo de todo lo que necesitamos indicado en las historias de usuario y, *post*eriormente, comprobar si cumplo o no los criterios de aceptación. Por ejemplo, el registro de usuario.

A continuación, ilustro la historia de usuario para el registro de usuario.

Título de HU02	Registro
Descripción	Como cliente quiero registrarme para poder acceder a la aplicación con mis datos.
Puntos de Historia	L
Criterios de aceptación	● Existencia de una BBDD donde se puedan guardar mis datos para poder acceder a ellos cuando se necesiten. ● Existencia de un formulario para establecer mis datos.

El criterio de aceptación es el siguiente: existencia de un formulario para establecer mis datos. Adjunto el formulario creado.

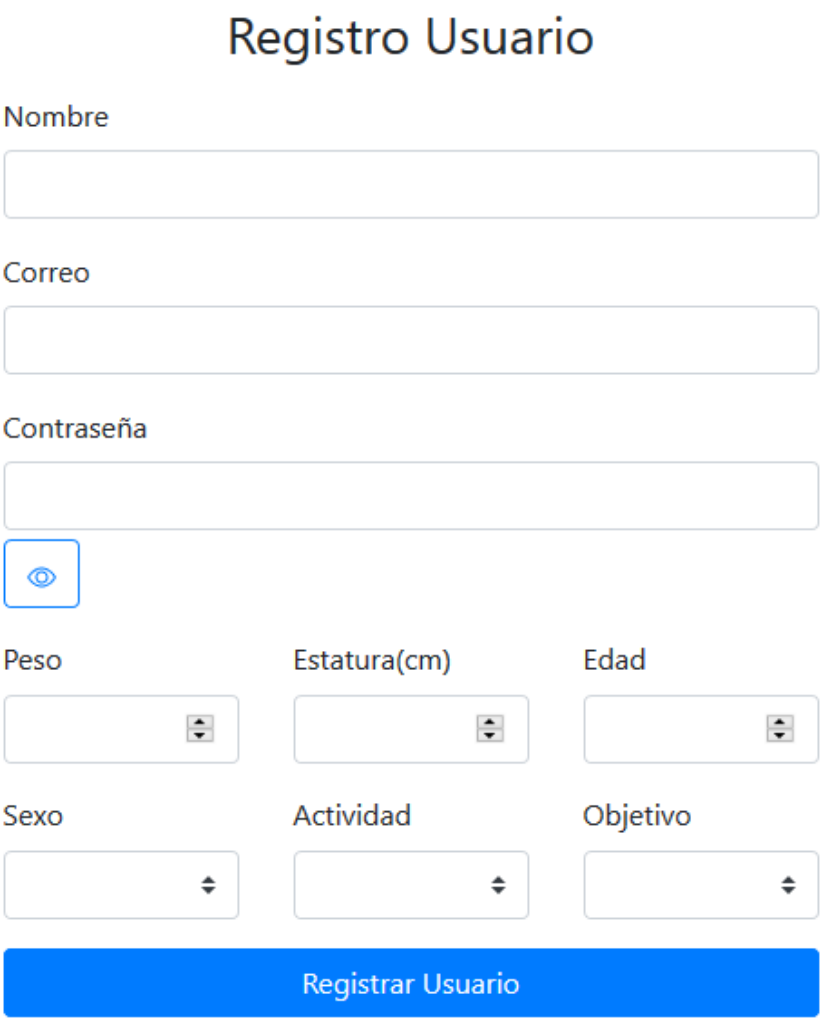


Ilustración 77: Registrar Usuario

Al revisar si cumplimos o no el criterio, podemos observar realmente si existe este formulario y que efectivamente lo guarda en la BBDD. Por lo tanto, ha pasado con éxito los criterios de aceptación.

5.3. Resultado de las pruebas

En la siguiente tabla se muestra un resumen de todas las pruebas realizadas.

Proyecto	Tipo de pruebas	Número de pruebas	Resultado de las pruebas
Backend	Tests con Junit	35	Satisfactorio

Frontend	Aceptación de criterios	23	Satisfactorio

Con estas pruebas hemos conseguido corregir cualquier tipo de errata o dato no esperado. Con los tests en el *Backend* usando *Junit* hemos podido comprobar que el servidor funcione correctamente según lo esperado. Por ejemplo, podría haber errores tales como cuando borra a un usuario, pero no borra todo lo relacionado con este. También puede detectar errores como que un monitor pueda crear una clase cuando está impartiendo otra durante esa hora.

Con respecto al *Frontend*, son pruebas sencillas que básicamente sirven para saber si están funcionando correctamente las peticiones *HTTP* y por lo tanto están siendo reflejados los cambios en la base de datos.

6. Conclusiones

Para finalizar este trabajo, expondremos diversas conclusiones obtenidas a lo largo de su desarrollo. A continuación, desarrollaré punto por punto aquellas que considero más relevantes respecto al alcance de los objetivos que han definido las líneas principales de mi proyecto.

a) En primer lugar, he realizado un estudio en profundidad de conceptos específicos del mundo del fitness para contextualizarlo a las necesidades concretas del proyecto. Para ello se han recopilado y estudiado numerosas fuentes de información como se refleja en el apartado de análisis. También se han estudiado soluciones software existentes que nos ayudarán a determinar algunas ideas relevantes para las funcionalidades del sistema propuesto.
b) A continuación, he desarrollado el diseño de un sistema informático cliente-servidor basado en tecnologías web. Para ello he recurrido a la utilización de diferentes tipos de diagramas habituales en el desarrollo web: Entidad-Relación, diagramas de Clases, etcétera. Con esto se pretende facilitar la comprensión del funcionamiento del sistema para facilitar su desarrollo y mantenimiento futuro.
c) Otros dos apartados del proyecto a los que se les ha dedicado un gran esfuerzo han sido la implementación y las pruebas. En el primer caso, he presentado tanto las herramientas de desarrollo como los detalles sobre la implementación más relevantes para facilitar su comprensión y mantenimiento al igual que ya se comentó en la parte de diseño. En el caso de las pruebas, se ha planteado un desarrollo basado en una metodología TDD para garantizar que a medida que se introducen cambios que todo funcione correctamente.

Considero que han sido esenciales las tecnologías que he utilizado en el desarrollo para finalizar de forma correcta y satisfactoria el proyecto; tecnologías que, sin duda, nos han facilitado el trabajo, y entre las que se encuentran, como ya se mencionara anteriormente, Spring Boot, que ha sido de gran ayuda, sobre todo por la facilidad que aportan sus configuraciones automáticas; o Junit, que nos permite testear el backend y ver su comportamiento en diversos escenarios. También merece

una mención especial Angular que, aunque es un framework que no hemos llegado a utilizar a lo largo del Grado, gracias a su arquitectura basada en componentes y a su fácil configuración y uso mediante el gestor de paquetes Npm me ha parecido muy versátil.

Quizá la mayor desventaja de Junit y Angular es su complejidad inicial, es decir, que al principio se puede tardar en comprender cómo y de qué manera funcionan.

No he encontrado ningún problema para el tema de mantener o ampliar el proyecto, puesto que todo queda bien reflejado en la documentación presentada, sujeta a cualquiera cambio.

Es importante aclarar un aspecto: a idea original de este proyecto pueda parecer sencilla, no lo ha sido en absoluto. Tenemos que tener en cuenta algo que considero clave: el mundo del fitness está sujeto a muchas variantes; quiero decir, como ya quedará reflejado a lo largo de mi proyecto, las mismas direcciones u orientaciones deportivas no convienen o se adecúan del todo y de forma uniforme u homogénea a todo el público. Por el contrario, un mismo entrenamiento o rutina no alcanzaría los mismos objetivos en personas con distintas modalidades biológicas o rutinarias, lo que implica un estudio o atención especial a cada caso. No siempre suele existir una ecuación infalible, una suma simplificada del tipo uno más uno es igual a dos para cualquier problema. A pesar de todo, quería ser capaz de proponer este punto de vista personal al problema, definir unos límites y proponer una posible solución mediante el prototipo desarrollado. Al fin y al cabo, pienso que el resultado que hemos obtenido es el adecuado: llevar al siguiente nivel tecnológico la gestión de un gimnasio por medio de tecnologías web. Es decir, todo aquello relacionado con la gestión de usuarios, clases a impartir y entrenamientos, sin descartar que, tal vez, un nutricionista o monitor especializado modifique algún dato dentro del plan de entrenamiento, algún porcentaje de macros, la duración de la definición o puede que otros detalles; aunque esto sería, sin duda, otro punto de vista, distinto e igual de lícito que el aquí expuesto.

Como colofón, huelga decir que el desarrollo de este TFG, complejo sin duda, ha sido el broche de oro que cierra el aprendizaje dilatado del Grado. Lo considero un hilo conductor hacia mi maduración personal y profesional, puesto que es el trasunto

de todo aquello que me define como ingeniero informático, en la medida en que he canalizado dentro de mis posibilidades todo conocimiento previamente adquirido, engrosado con otras lecturas o aprendizajes obligatorios y complementarios, para su completo desarrollo.

Me siento realizado y satisfecho por haber afrontado y superado lo que considero sin duda un reto intelectual con la ayuda de mi tutor, a quien sin duda, como últimas palabras, le dedico mi más sincero agradecimiento.

6.1. Mejoras y trabajos futuros

En primer lugar, una de las mejoras que puede aportar una gran funcionalidad a la aplicación es un buscador para los monitores, donde puedan filtrar por nombre o por correo. Esta funcionalidad sería fácil de implementar, ya que solo debemos implementar el buscado en el *Frontend*, ya que en el *Backend* se encuentra implementada, pero por motivos de planificación no ha podido ser desarrollada.

En segundo lugar, implementar filtros para la búsqueda de clases, filtros por nombre, fecha, duración, etc. En este caso no hay nada de este tipo implementado en el proyecto, por lo tanto, tendría que implementarse desde cero.

Otro cambio, bastante importante es introducir la rutina de entrenamiento al usuario mediante la aplicación. Como monitor poder enviar una rutina al usuario y que sea estilo Wiki, para poder modificarla con facilidad. La integración de esto creo que supondría una dificultad mayor, pero no debemos modificar nada de lo ya implementado, sino incorporar nuevo contenido.

Como última mejora, añadiría una gestión de perfiles para el tema de navegación de vistas. Añadiendo también un nuevo rol como sería el de ADMIN, que es el encargado de gestionarlo todo, incluyendo dar de alta, dar de baja o actualizar un monitor.

Bibliografía

[1] G. F. Ángel Luis, *Apuntes de Desarrollo ágil,* Universidad de Jaén, 2020.

[2] Sanitas, «Conceptos básicos del entrenamiento,» [En línea]. Available: https://www.sanitas.es/sanitas/seguros/es/particulares/biblioteca-de-salud/ejercicio-deporte/Consejos-para-correr/conceptos-entrenamiento.html. [Último acceso: Octubre 2020].

[3] D. M. López, «VOLUMEN DE ENTRENAMIENTO PARA HIPERTROFIA MUSCULAR,» Gaman Training, 23 Julio 2020. [En línea]. Available: https://www.gamantraining.com/volumen-de-entrenamiento-para-hipertrofia-muscular/. [Último acceso: Octubre 2020].

[4] D. M. Israetel, «The hypertrophy training guide central hub,» RP, 27 Abril 2017. [En línea]. Available: https://renaissanceperiodization.com/hypertrophy-training-guide-central-hub/. [Último acceso: Octubre 2020].

[5] D. M. Israetel, «Training volume landmarks for muscle growth,» RP, 04 Enero 2017. [En línea]. Available: https://renaissanceperiodization.com/training-volume-landmarks-muscle-growth/. [Último acceso: Octubre 2020].

[6] J. M. M. Morales, «La percepción subjetiva del esfuerzo como parte de la evaluación de la intensidad del entrenamiento,» efdeportes, [En línea]. Available: https://www.efdeportes.com/efd73/percep.htm. [Último acceso: Octubre 2020].

[7] N. Williams, «Occupational Medicine,» de *The Borg Rating of Perceived Exertion (RPE) scale*, Oxford academic, 2017, pp. 404-405.

[8] «Nutrición,» OMS, [En línea]. Available: https://www.who.int/topics/nutrition/es/. [Último acceso: Octubre 2020].

[9] «Ecuación de Harris-Benedict,» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/Ecuaci%C3%B3n_de_Harris-Benedict. [Último acceso: Octubre 2020].

[10] E. R. Helms, «High-protein, low-fat, short-term diet results in less stress and fatigue than moderate-protein moderate-fat diet during weight loss in male weightlifters: a pilot study,» Abril 2015. [En línea]. Available: https://pubmed.ncbi.nlm.nih.gov/25028958/. [Último acceso: Noviembre 2020].

[11] J. R. Hoffman, «Effect of Protein Intake on Strength, Body Composition and Endocrine Changes in Strength/Power Athletes,» Diciembre 2006. [En línea]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2129168/. [Último acceso: Noviembre 2020].

[12] M. Tarnopolsky, «Evaluation of protein requirements for trained strength athletes,» [En línea]. Available: https://pubmed.ncbi.nlm.nih.gov/1474076/. [Último acceso: Noviembre 2020].

[13] P. W. Lemon, «Protein requirements and muscle mass/strength changes during intensive training in novice bodybuilders,» [En línea]. Available: https://pubmed.ncbi.nlm.nih.gov/1400008/. [Último acceso: Noviembre 2020].

[14] J. Antonio, «The effects of consuming a high protein diet (4.4 g/kg/d) on body composition in resistance-trained individuals,» 2014. [En línea]. Available: https://jissn.biomedcentral.com/articles/10.1186/1550-2783-11-19. [Último acceso: Noviembre 2020].

[15] M. Muñoz, «CANTIDAD MÍNIMA DE HIDRATOS PARA MANTENER MASA MUSCULAR,» Powerexplosive, 30 Enero 2017. [En línea]. Available: https://powerexplosive.com/cantidad-minima-de-hidratos-para-mantener-masa-muscular/. [Último acceso: Octubre 2020].

[16] S. P. Ramírez, «SUPERÁVIT CALÓRICO: Todo lo que debes saber,» SaludDiez, Julio 13 2020. [En línea]. [Último acceso: Octubre 2020].

[17] F. Royano., «MÉTODO PARA CALCULAR TUS CALORÍAS DE MANTENIMIENTO,» Wellness Real, 13 Julio 2020. [En línea]. Available: https://wellnessreal.es/metodo-para-calcular-tus-calorias-de-mantenimiento/. [Último acceso: Octubre 2020].

[18] Vital sport, «Adherencia al entrenamiento ¿Qué es y cómo desarrollarla?,» Noviembre 2019. [En línea]. Available: https://aptavs.com/articulos/adherencia-entrenamiento. [Último acceso: Octubre 2020].

[19] M. Vázquez, «Instagram,» Septiembre 2020. [En línea]. Available: https://www.instagram.com/p/CFT3aVhoIwy/. [Último acceso: Octubre 2020].

[20] C. Álvaro, «SCRUM y los puntos de historia, ¿Cómo funcionan?,» Enero 2020. [En línea]. Available: https://www.incentro.com/es-es/blog/stories/scrum-puntos-de-historia-como-funcionan/. [Último acceso: Octubre 2020].

[21] Indeed, «Indeed,» Noviembre 2020. [En línea]. Available: https://es.indeed.com/salaries/desarrollador-java-Salaries. [Último acceso: Noviembre 2020].

[22] Deloitte, «¿Qué es un ORM?.,» Deloitte, [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html. [Último acceso: Septiembre 2020].

[23] M. Valero, «Modelo relacional y formas normales.,» Bigeek, 18 Diciembre 2016. [En línea]. Available: https://blog.bi-geek.com/modelo-relacional-formas-normales/. [Último acceso: Septiembre 2020].

[24] G. Escobar, «¿Qué es un API?,» MakeItReal, 15 Septiembre 2015. [En línea]. Available: https://blog.makeitreal.camp/que-es-un-api/. [Último acceso: Octubre 2020].

[25] R. Martínez, «TDD, una metodología para gobernarlos a todos,» Paradigma, 2017. [En línea]. Available: https://www.paradigmadigital.com/techbiz/tdd-una-metodologia-gobernarlos-todos/. [Último acceso: Octubre 2020].

[26] Software Development, «Marsner,» [En línea]. Available: https://marsner.com/blog/why-test-driven-development-tdd/. [Último acceso: Octubre 2020].

[27] «Diagrama de arquitectura del sistema,» Bittarcorp S.L., [En línea]. Available: https://bittacorp.wordpress.com/2008/11/22/diagrama-de-arquitectura-del-sistema/. [Último acceso: Noviembre 2020].

[28] JavaTpoint, «Spring Boot Architecture,» [En línea]. Available: https://www.javatpoint.com/spring-boot-architecture. [Último acceso: Octubre 2020].

[29] «Protocolo seguro de transferencia de hipertexto,» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/Protocolo_seguro_de_transferencia_de_hipertexto. [Último acceso: Octubre 2020].

[30] Angular, «Introduction to Angular concepts,» [En línea]. [Último acceso: Octubre 2020].

[31] O. Blancarte, «Data Transfer Object (DTO) – Patrón de diseño.,» Oscar Blancarte Software Architect., 30 Noviembre 2018. [En línea]. Available: https://www.oscarblancarteblog.com/2018/11/30/data-transfer-object-dto-patron-diseno/.. [Último acceso: Septiembre 2020].

[32] T. Hombergs, «Integration Tests with Spring Boot and @SpringBootTest,» [En línea]. Available: https://reflectoring.io/spring-boot-test/. [Último acceso: Octubre 2020].

[33] Baeldung, «A Quick Guide to @DirtiesContext,» Octubre 2019. [En línea]. Available: https://www.baeldung.com/spring-dirtiescontext. [Último acceso: Octubre 2020].

[34] V. Balasubramaniam, «@TestInstance Annotation in JUnit 5,» Baeldung, Agosto 2020. [En línea]. Available: https://www.baeldung.com/junit-testinstance-annotation. [Último acceso: Octubre 2020].

[35] Baeldung, «Exploring the Spring Boot TestRestTemplate,» Marzo 2020. [En línea]. Available: https://www.baeldung.com/spring-boot-testresttemplate . [Último acceso: Octubre 2020].

[36] C. Larman, UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado, Madrid: Pearson Educacion, 2010.

[37] «Normalización de bases de datos,» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_bases_de_datos.. [Último acceso: Septiembre 2020].

[38] C. Pesquera, «¿QUÉ ES UN POJO, EJB Y UN BEAN?,» Carlospesquera.com, 10 Marzo 2014. [En línea]. Available: https://carlospesquera.com/que-es-un-pojo-ejb-y-un-bean/. [Último acceso: Septiembre 2020].

[39] «CRUD,» Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/CRUD. [Último acceso: 2020 Octubre].

[40] «Introduction to components and templates,» Angular, [En línea]. Available: https://angular.io/guide/architecture-components. [Último acceso: 2020 Octubre].

Apéndice I. Manual de Instalación del sistema

- Instalación de Docker:
  - https://www.docker.com/get-started
  - Creación BBDD:
    - docker pull mysql → para instalar la imagen oficial de mysql
    - docker run –d –p 3306:3306 –name mysql-db –e MYSQL_ROOT_PASSWORD=secret mysql →para iniciar un contenedor llamado mysql-db con el servidor mysql escuchando en el puerto 3306.
    - docker exect –it mysql-db mysql –p → para iniciar un cliente mysql con el usuario ROOT, su contraseña se dió al iniciar el servidor, es secret
    - Desde el cliente mysql y como ROOT, ejecutar las siguientes órdenes para crear la base de datos, el usuario y darle privilegios a este:
      - CREATE DATABASE Backend;
      - CREATE USER ‘usuario’@’%’ IDENTIFIED BY ‘password’;
      - GRANT ALL ON Backend.* TO ‘usuario’@’%’
- Instalación visual code
  - https://code.visualstudio.com/

- Descarga Eclipse
  - https://www.eclipse.org/downloads/packages/release/2020-09/r/eclipse-ide-enterprise-java-developers
- Instalación angular
  - Instalar node versión estable: https://nodejs.org/es/
  - npm install -g @angular/cli.
- Para lanzar el Backend:
  - Importamos el proyecto al eclipse
  - Run as > Spring Boot Aplication
- Para lanzar el Frontend
  - El Frontend lo abrimos con visual code
  - Npm install para las dependencias
  - Npm start para lanzarlo

Configurar BBDD

Para la configuración de BBDD hemos utilizado un contenedor en Docker que podemos encontrarlo en su página principal totalmente gratis: https://www.docker.com/get-started.

Tras instalar docker pasamos a configurar el contenedor de MySQL, mediante los siguientes comandos:

- *docker pull mysql* → para instalar la imagen oficial de MySQL.
- *docker run –d –p 3306:3306 –name mysql-db –e MYSQL_ROOT_PASSWORD=secret mysql* → para iniciar un contenedor llamado mysql-db con el servidor MySQL escuchando en el puerto 3306.
- *docker exect –it mysql-db mysql –p* → para iniciar un cliente MySQL con el usuario ROOT, su contraseña se estableció al iniciar el servidor, es secret.
- Desde el cliente MySQL y como ROOT, ejecutar las siguientes órdenes para crear la base de datos, el usuario y darle privilegios a este:
  - *CREATE DATABASE Backend;*
  - *CREATE USER ‘usuario’@’%’ IDENTIFIED BY ‘password’;*

- *GRANT ALL ON Backend.* TO 'usuario'@'%'*

Configurar entorno para Backend.

Para el Backend necesitamos el entorno *Eclipse IDE for Enterprise Java Developers,* que podemos encontrarlo en su página oficial: https://www.eclipse.org/downloads/packages/release/2020-09/r/eclipse-ide-enterprise-java-developers.

En segundo lugar, importamos el proyecto de la siguiente forma:

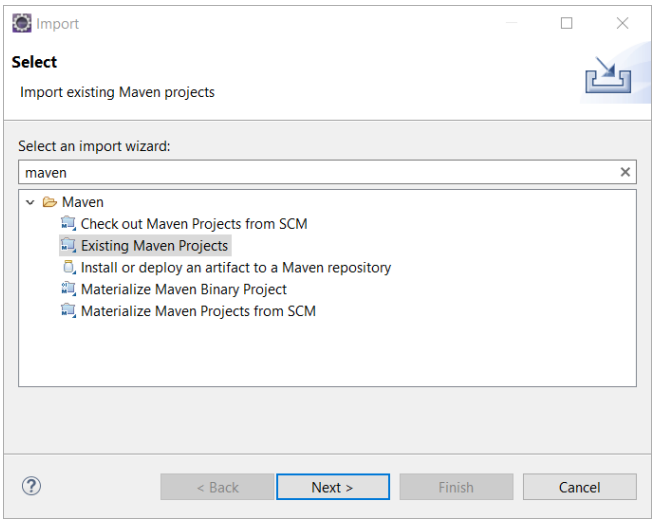


Ilustración 78: Importar proyecto

Buscamos la ruta donde tenemos el proyecto y lo abrimos, quedaría tal que así:

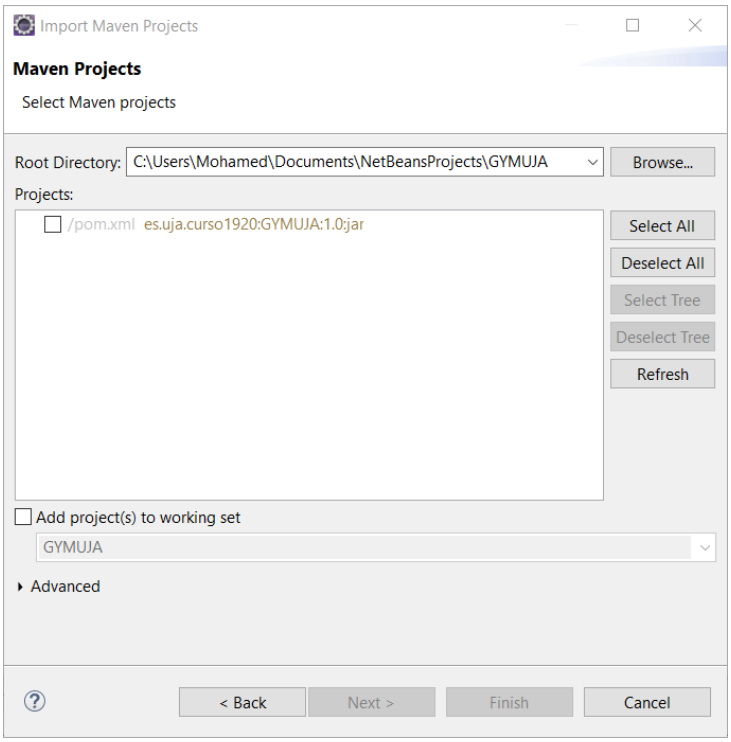


Ilustración 79: Importar proyecto

A continuación, seleccionamos el pom.xml y pulsamos *finish.* Y ya tendríamos nuestro Backend configurado.

Configurar entorno para Frontend.

Para el Frontend necesitamos el entorno de desarrollo llamado *Visual Studio Code*, que podemos encontrarlo en su página oficial: https://code.visualstudio.com/.

El siguiente paso es configurar Angular que para eso necesitamos lo siguiente:

- Instalar el *nodeJS* → https://nodejs.org/es/
- Instalar Angular, abrimos una terminal y ejecutamos el siguiente comando → *npm install -g @angular/cli*

Para finalizar abrimos el *Visual Studio Code* y abrimos el proyecto de la siguiente forma: File > Open Folder y seleccionamos la carpeta del proyecto. Para finalizar la configuración abrimos un terminal y lo situamos dentro del proyecto y ejecutamos el siguiente comando para instalar las dependencias: *npm install.*

Ejecutar aplicación.

En primer lugar, necesitamos lanzar el contenedor de Docker con la BBDD. Abrimos *Docker*, una vez se haya iniciado hacemos *click* sobre el icono.

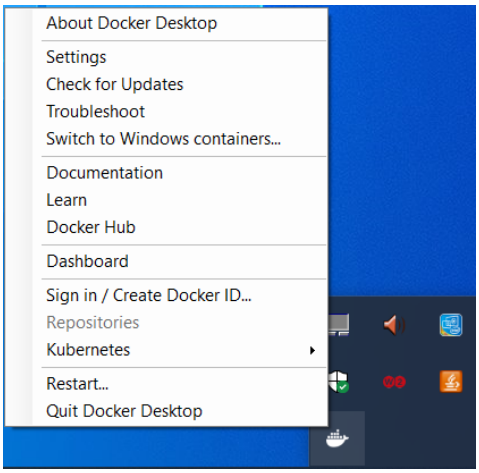


Ilustración 80: Opciones Docker

Cuando nos encontramos en este menú, pinchamos en *Dashboard* que nos lleva a los distintos contenedores que tenemos creados y pulsamos en el *play* del contenedor *mysql-db*.

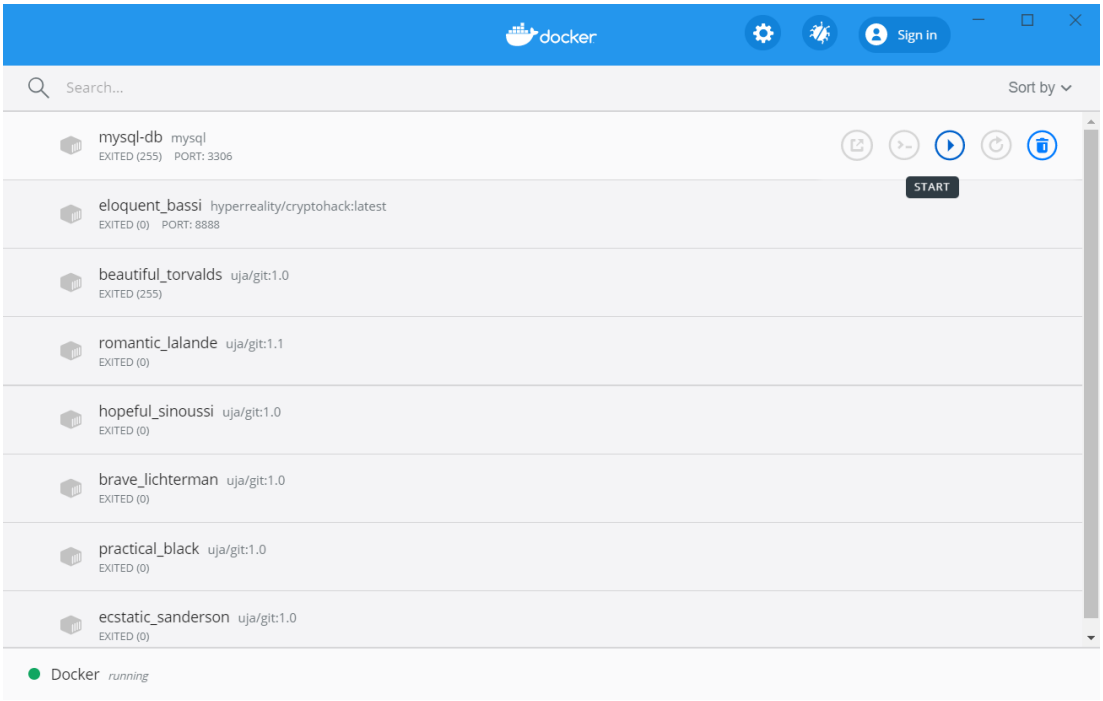


Ilustración 81: Dashboard Docker

Una vez tenemos la BBDD funcionando, pasamos al Backend. Abrimos eclipse donde tenemos importado el proyecto y clicamos sobre nuestro proyecto nos vamos a Run as > Spring Boot App.

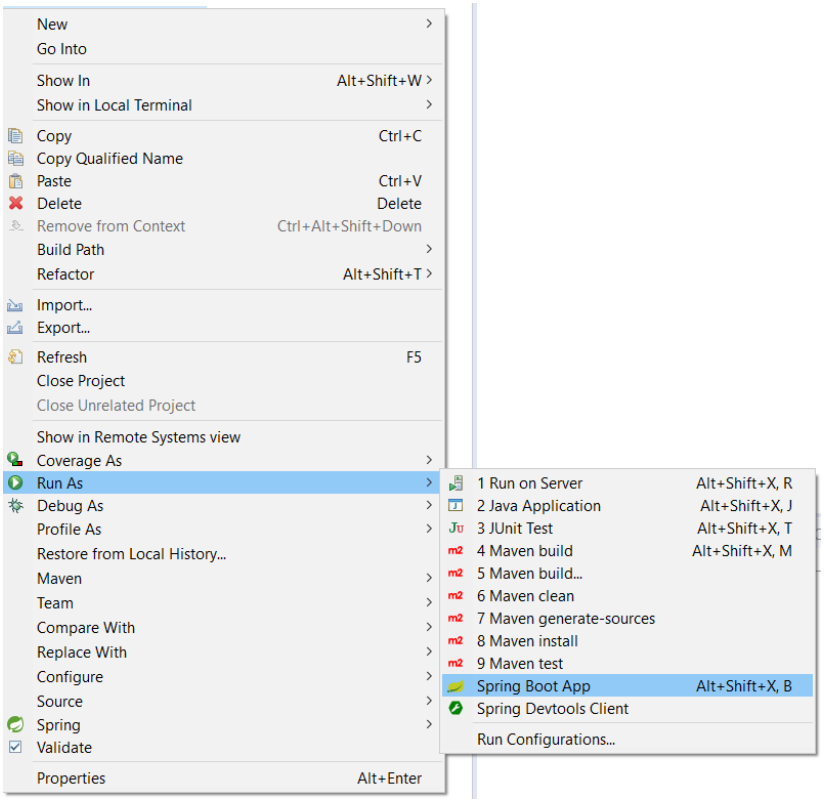


Ilustración 82: Run as Spring Boot App

Para finalizar abrimos el *Visual Studio Code,* dónde tenemos el proyecto importado y en la consola del *Visual* lanzamos el siguiente comando: *npm start*. Una vez ha compilado y lanzado correctamente el proyecto, podemos abrir el siguiente enlace para observar el proyecto: http://localhost:4200/.

Apéndice II. Manual de Usuario

Una vez iniciamos la aplicación, nos encontramos en la página principal donde podemos observar un *slider*.

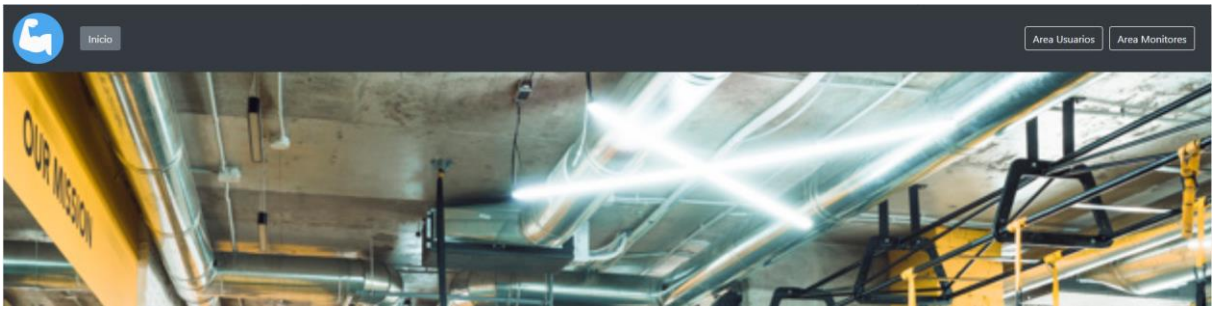


Ilustración 83: Página principal

Si nos vamos al Área de monitores, nos encontramos con el *Login*.

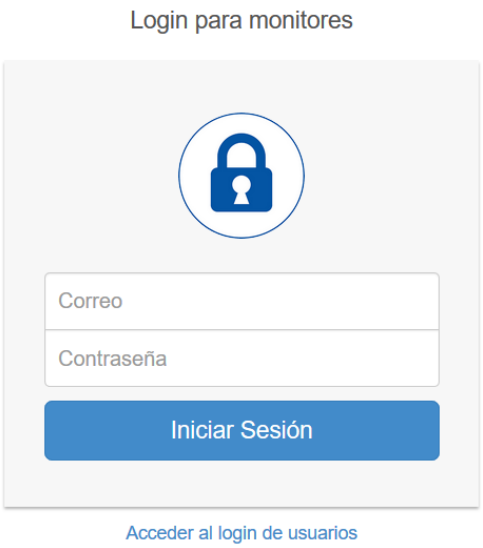


Ilustración 84: Login Monitor

Cuándo se ha *logueado* el monitor, nos encontramos con el *slider* del inicio pero con diferentes botones en la barra de navegación.

Ilustración 85: Inicio monitor

Continuamos con usuarios registrados, que muestra una lista con los usuarios que hay actualmente creados en la BBDD.

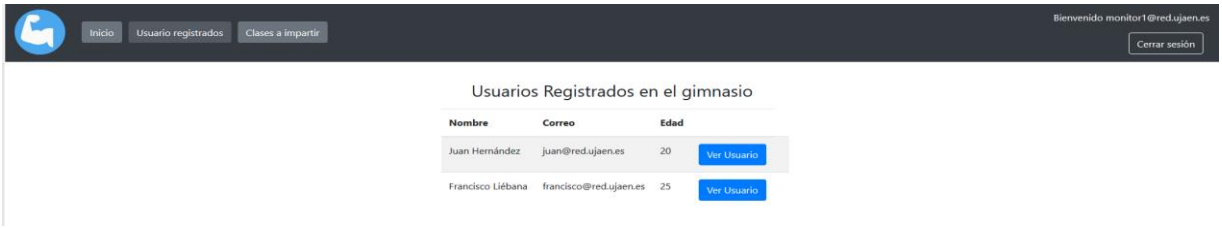


Ilustración 86: Usuarios Registrados

Si pulsamos en Ver Usuario donde está el usuario Juan Hernández, nos proporciona una vista con todos los datos de Juan, donde incluso podemos llegar a borrar este usuario.

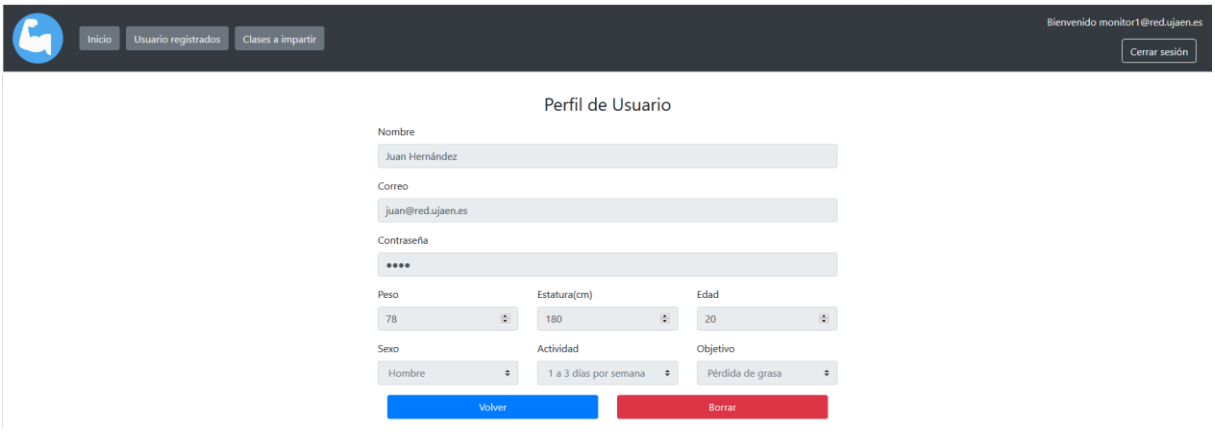


Ilustración 87: Perfil usuario

Pasamos a Clases a impartir, donde encontramos una lista de las clases que va a impartir el monitor. Dentro de este apartado podemos crear una clase nueva, listar los participantes de una clase, editar la clase y borrar la clase.

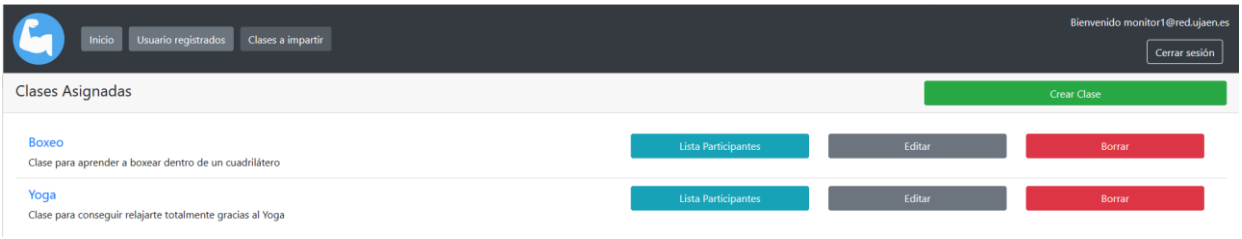


Ilustración 88: Clases a impartir

Comenzamos con crear una nueva clase, esta es la vista a la que te redirige.

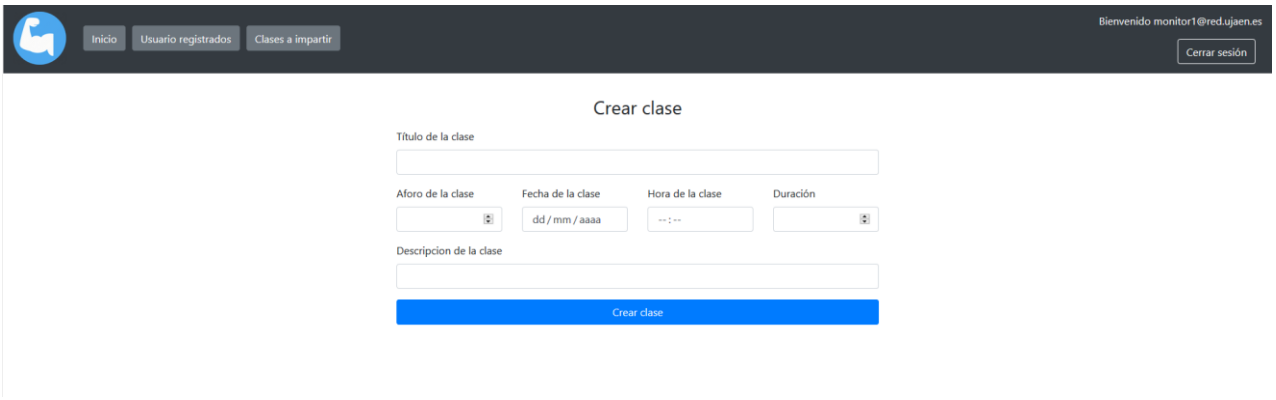


Ilustración 89: Crear Clase

Si listamos los participantes de una clase, existen dos casos: que no haya ningún participante o en el caso de si los haya muestra una lista de los participantes, dónde el monitor puede borrarlo como participante.

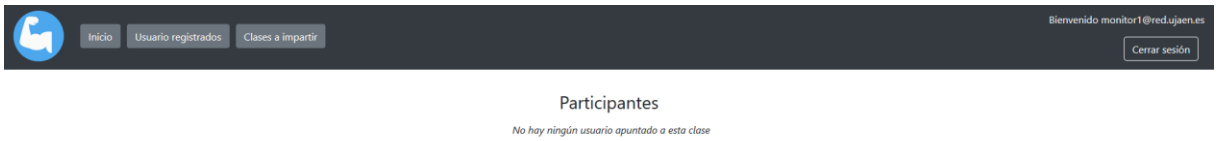


Ilustración 90: Participantes

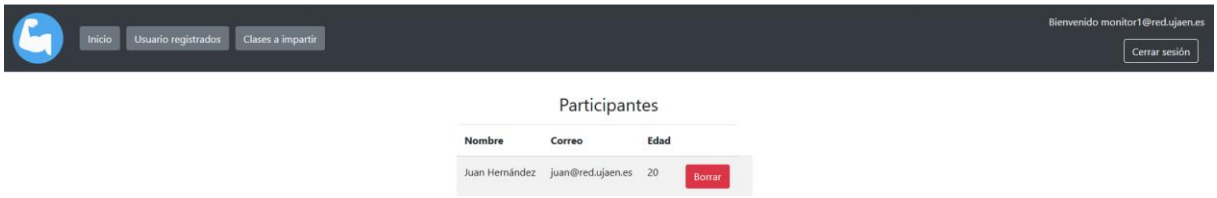


Ilustración 91: Participantes

Al hacer click en editar una clase, te salen todos los campos en relación a esa clase y puedes modificar cualquiera y finalizar dando a Editar clase.

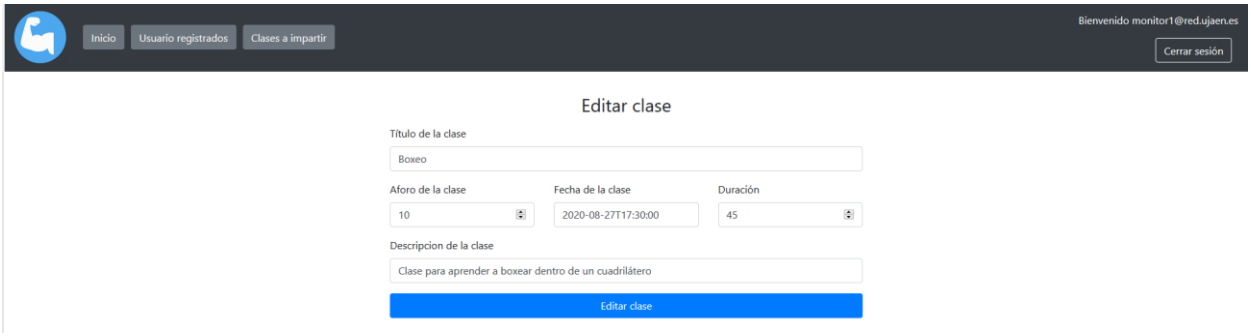


Ilustración 92: Editar Clase

Con todo esto hemos finalizado en la parte de monitor. Continuamos con el usuario, en este caso el formulario de creación de un usuario.

En el registro de usuario, nos encontramos con una serie de campos: Nombre, Correo, Contraseña, Peso, Estatura, Edad, Sexo, Actividad, Objetivo. Hemos decidido incorporar un botón que permite observar la contraseña, para ver es correcta.

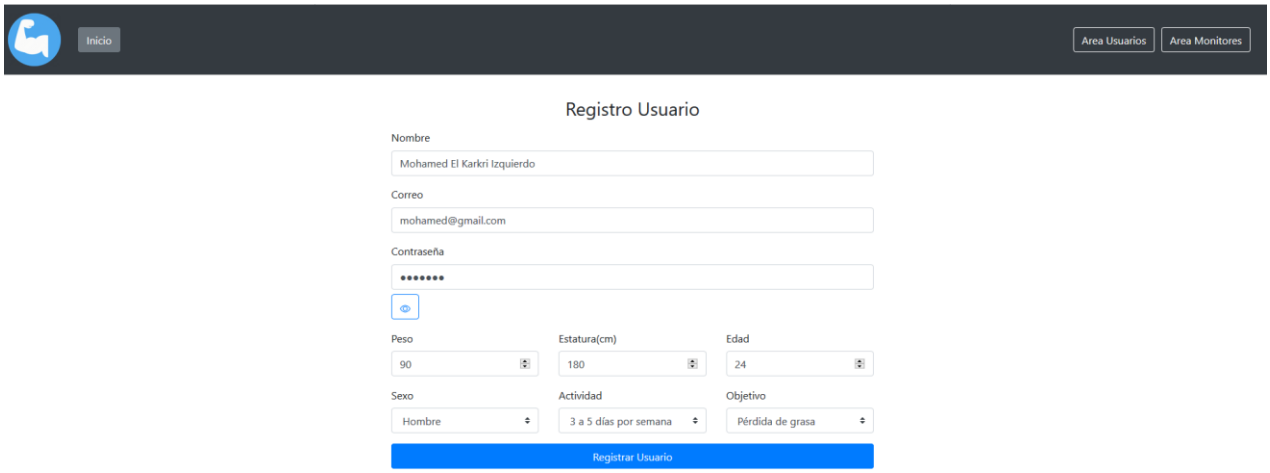


Ilustración 93: Registro Usuario

El siguiente paso es el login del usuario, dónde si introducimos algún dato erróneo nos lo indicará.

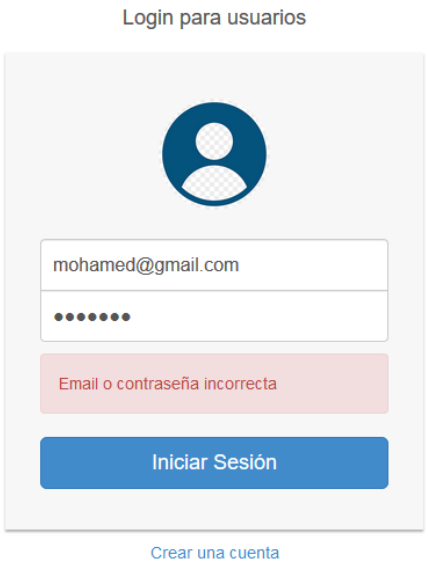


Ilustración 94: Login Usuario

Para tener un mejor acceso a las clases he decidido que la página de bienvenida al usuario se corresponda a la lista de clases disponibles. En esta vista, podemos acceder a información más detallada mediante el botón o el nombre de la clase.

Ilustración 95: Inicio Usuario

Si queremos acceder a ver más información, nos encontraremos con la vista completa de la clase, dónde podremos apuntarnos o desapuntarnos en el caso de que ya estemos apuntados.

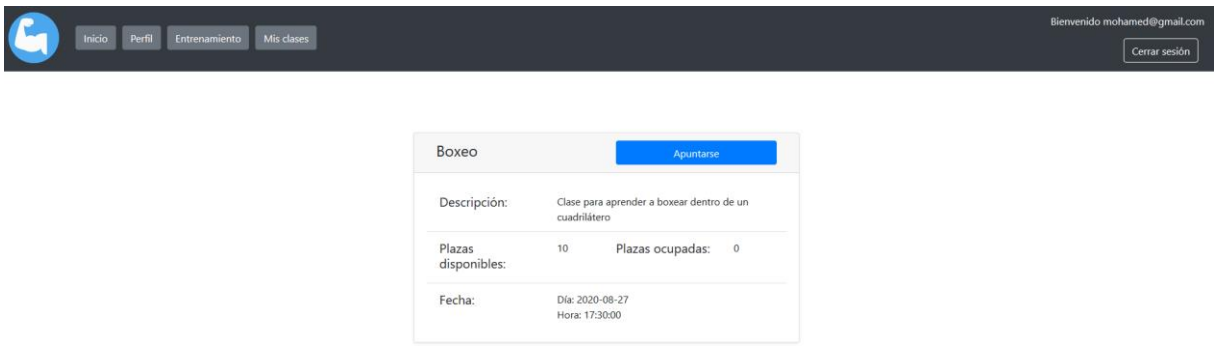


Ilustración 96: Ver Clase

Al apuntarnos en una clase, se registra esta clase como apuntada y si nos dirigimos a Mis clases la encontramos en la lista.

Ilustración 97: Mis clases

Si hacemos clic en su nombre, nos indica que estamos apuntados actualmente.

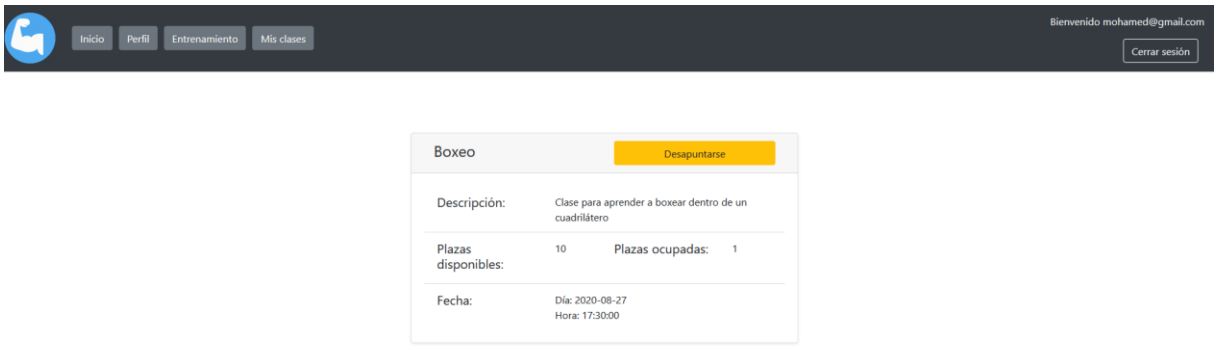


Ilustración 98: Vista clase apuntado

Para acceder al perfil de usuario, necesitamos utilizar el botón de Perfil. En esta vista podemos modificar los distintos campos que hemos introducido en el registro y podemos guardarlo mediante el botón de Editar Usuario o deshacer todo mediante el botón de Cancelar.

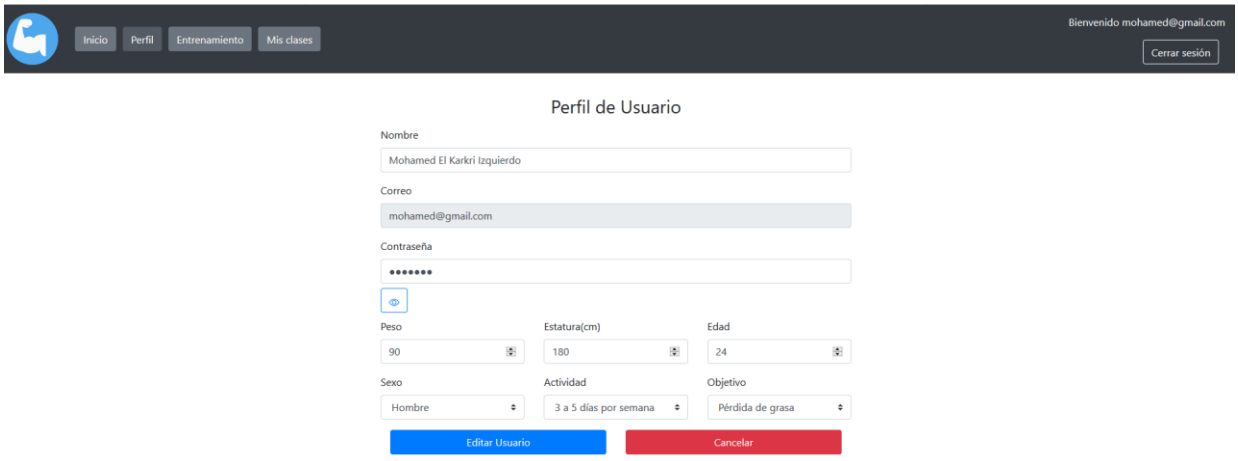


Ilustración 99: Perfil usuario

Para finalizar pasamos a la parte del Entrenamiento. Donde existen tres posibles casos: Pérdida de grasa, Ganancia muscular y Mantenimiento. En primer lugar, comenzamos con Ganancia muscular, que te proporciona una tabla de la nutrición a llevar cada día durante el tiempo que sigas con este objetivo.

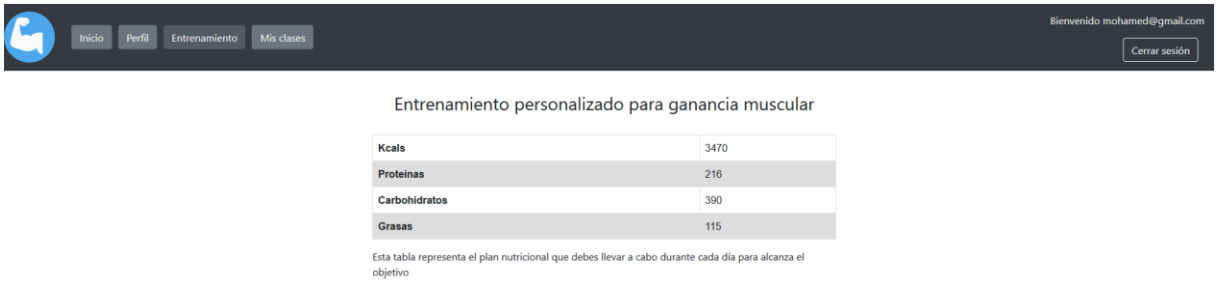


Ilustración 100: Ganancia muscular

En segundo lugar, continuamos con Mantenimiento que es similar a la vista anterior.

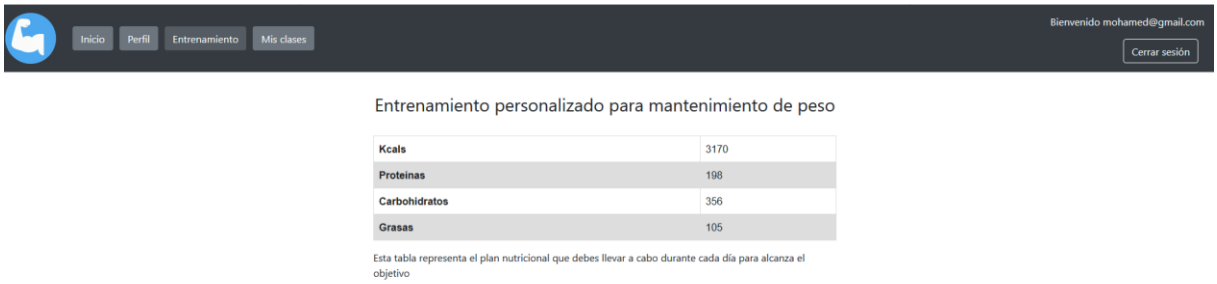


Ilustración 101: Mantenimiento

Para finalizar, seguimos con Pérdida de grasa dónde es más complejo. Primero te proporciona una nutrición a seguir cada día durante 15 días, tras esos 15 días te permite añadir el peso para poder ajustar la fórmula lo máximo posible.

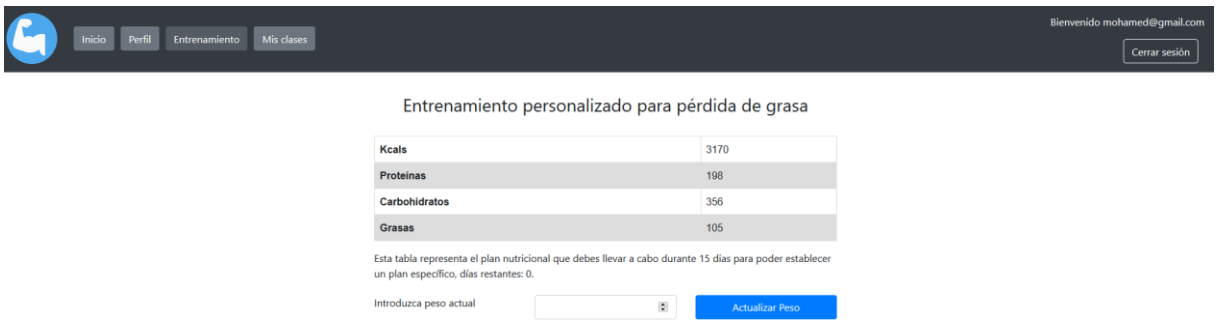


Ilustración 102: Pérdida de grasa

En este caso le introducimos el mismo peso y pasamos a la nutrición a seguir cada día durante 12 semanas.

Ilustración 103: Pérdida de grasa

Apéndice III. Descripción de contenidos suministrados

1. Memoria en PDF del trabajo.
2. Archivo zip con los dos proyectos: Backend y Frontend.