



DOCUMENTACIÓN TÉCNICA

PROYECTO GIMNASIO

Nicolás Martín Morcillo
Adrián Romero Romero
Alejandro Caparrós Zaplana
José Megía Guillén

[Enlace al repositorio de GitHub](#)



Universitat d'Alacant
Universidad de Alicante

ASIGNATURA: INGENIERÍA WEB

1. Introducción	2
a. Descripción del sistema	2
b. Tecnologías	3
1) Base de Datos: PostgreSQL	3
2) Backend: Java con Spring Boot	4
3) Frontend: Vue.js	4
c. Arquitectura	5
Cliente del sistema	5
Servidor del sistema	6
Ventajas de la arquitectura	7
d. Interoperación	7
1. Proyecto Tienda de Material deportivo	7
2. Proyecto TPV	8
2. Mockups	10
Página home	10
Solicitud de registro	12
Inicio de sesión	13
Editar mi perfil	14
Formulario de añadir fondos	15
Actividades disponibles	16
Asistentes a actividad	17
Actividades a las que estás apuntado	18
Apúntate a una actividad y detalle de actividades	19
Aceptar o rechazar solicitudes de socio por parte del webmaster	20
Crear una nueva actividad	21
Listado de informes	22
Listado de informes	23
3. Diagramas	24
4. Metodología y planificación	25
5. Descripción de la implementación	33
Implementación en el backend	33
Implementación en el frontend	38
Portabilidad del proyecto	39
6. Problemas encontrados y su solución	40
Uso de Data Transfer Objects	40
Problema con la estrategia de generación de IDs en Hibernate	41
Problema con la triple herencia	42
Interoperación con la tienda online de productos deportivos	43
7. Mejoras y ampliaciones	45
Reservas mejoradas para evitar solapamientos	45
Perfiles de usuario públicos	45
CRUD completo de todas las entidades	46
8. Referencias	47

1. Introducción

a. Descripción del sistema

El sistema para la gestión del gimnasio tiene como objetivo principal ofrecer una plataforma web que permita a los visitantes, socios, monitores y administradores interactuar con las funcionalidades y servicios del gimnasio de manera eficiente. Esta solución incluye funcionalidades específicas para la administración de socios, actividades y reservas, así como la interoperación con servicios externos como TPVV y una tienda en línea.

Funcionalidades principales:

En este apartado se recogen las funcionalidades principales del sistema, en ningún caso estas son las funcionalidades detalladas del sistema, las funcionalidades detalladas se recogen más adelante en el sistema.

- 1) Página pública, visibles para usuarios registrados y no registrados en el sistema:
 - a) Información sobre el gimnasio, incluyendo instalaciones, contacto, fotos, tarifas y cómo hacerse socio.
 - b) Formulario de solicitud para nuevos socios en el cual se capturan los datos personales de la persona que desea hacerse socio del gimnasio.
 - c) Catálogo de productos vinculados a una tienda online con la que debe interoperar el sistema.
- 2) Gestión de socios:
 - a) Creación y administración de cuentas de socios.
 - b) Control de cuotas mensuales, incluyendo dos modalidades: individual y familiar
 - c) Sistema de recarga de saldo mediante TPVV para realizar pagos por actividades. El sistema debe interoperar con el TPVV.
 - d) Reserva de actividades
- 3) Gestión de actividades:

- a) Listado de actividades disponibles en un calendario. Se permite filtrar el calendario de actividades por tipos de actividades.
 - b) Configuración del precio de actividades en función del tipo de actividad. Spinning 5€, Crossfit 15€, El Resto 10 €.
- 4) Perfil del monitor:
 - a) Visualización de actividades asignadas en su horario y asistentes.
 - b) Gestión de nuevas actividades, seleccionando tipo, horario, duración y sala.
- 5) Perfil del webmaster:
 - a) Administración de socios: aceptación de nuevos socios, bloqueo y reactivación.
 - b) Gestión de informes detallados como:
 - i) Asistencia a actividades
 - ii) Altas y bajas de socios
 - iii) Ingresos mensuales y anuales
 - c) Configuración de actividades y horarios

Con estas funcionalidades, el sistema busca ofrecer una solución integral que permita una gestión eficiente del gimnasio, mejorando la experiencia tanto de los socios como de los administradores. Adicionalmente, el sistema está basado en una interfaz web fácil de utilizar para todos los usuarios que interactúan con el sistema.

b. Tecnologías

Para el desarrollo del sistema de gestión del gimnasio, se han seleccionado tecnologías modernas y ampliamente utilizadas en la industria, que garantizan una arquitectura robusta, escalable y fácil de mantener. A continuación, se describen las tecnologías empleadas:

1) Base de Datos: PostgreSQL

PostgreSQL es el motor de base de datos relacional utilizado para almacenar y gestionar de manera eficiente la información del sistema. Razones para su elección:

- a) Es de código abierto y cuenta con una amplia comunidad de soporte.

- b) Adecuado para manejar relaciones complejas entre entidades del sistema como socios, actividades, reservas y saldos.
- c) Fácil de ejecutar sobre un contenedor Docker para facilitar su portabilidad y ejecución en cualquier máquina o sistema operativo.

2) Backend: Java con Spring Boot

El backend del sistema se ha desarrollado en Java utilizando el framework Spring Boot, una solución ágil para construir aplicaciones robustas y escalables. Razones para su elección:

- a) Contamos con mucha experiencia en el desarrollo de backend utilizando Spring Boot ya que lo hemos utilizado mucho en otras asignaturas.
- b) Ofrece un amplio soporte para integraciones, como el acceso a bases de datos relaciones.
- c) Soporte nativo para servicios RESTful, lo que permite crear una API eficiente y segura para la interacción con el frontend.
- d) Escalabilidad y modularidad para manejar las extensiones del sistema
- e) Útil para la implementación de la lógica de negocio.

3) Frontend: Vue.js

El frontend de la aplicación está desarrollado con Vue.js, un framework progresivo para construir interfaces de usuario interactivas y dinámicas. Razones para su elección:

- a) Es ligero, flexible y tenemos mucha experiencia en Vue ya que lo hemos utilizado en otras asignaturas del grado.
- b) Proporciona un sistema de componentes reutilizables que facilita el desarrollo y mantenimiento del frontend.
- c) Excelente soporte para el manejo de datos dinámicos mediante sus sistema de reactividad.
- d) Proporciona una integración muy sencilla con el API desarrollado para el backend.

Estas tecnologías trabajan juntas de manera armónica para ofrecer una experiencia fluida y confiable a los usuarios:

- a) El backend (Spring Boot) gestiona la lógica de negocio y provee servicios a través de un API RESTful.

- b) La base de datos PostgreSQL almacena y organiza la información estructurada, con acceso controlado por el backend.
- c) El frontend (Vue.js) consume el API RESTful para presentar una interfaz interactiva y atractiva para los usuarios finales.

c. Arquitectura

El sistema de gestión del gimnasio utiliza una arquitectura cliente-servidor, que se caracteriza por la separación de responsabilidades entre dos entidades principales: el cliente y el servidor. Esta arquitectura asegura escalabilidad, modularidad y facilidad de mantenimiento, y permite que el sistema sea eficiente y adaptable a futuras necesidades.

La arquitectura cliente-servidor divide el sistema en dos partes principales:

- 1) Cliente: Es la interfaz con la que interactúan los usuarios finales. Solicita servicios al servidor y presenta los resultados de manera comprensible y visualmente atractiva.
- 2) Servidor: Es el núcleo del sistema donde reside la lógica de negocio, se procesan las solicitudes y se gestionan los datos. Actúa como intermediario entre el cliente y la base de datos, garantizando la seguridad e integridad de la información.

Esta separación permite que las dos partes se desarrollen y mantengan de manera independiente, al tiempo que se comunican a través de un protocolo estándar como HTTP.

Cliente del sistema

El cliente es desarrollado con Vue.js y representa la capa de presentación del sistema. Su función principal es proporcionar una interfaz dinámica e interactiva para los usuarios finales.

Rol del cliente:

Mostrar al usuario información obtenida del servidor, como horarios de actividades, informes y saldos.

Permitir a los usuarios realizar acciones como registrarse, reservar actividades o gestionar su perfil.

Enviar solicitudes al servidor a través de llamadas a un API RESTful.

Comunicación con el servidor:

El cliente envía peticiones HTTP al servidor, que incluyen datos para operaciones como creación de reservas o consultas de saldo.

Recibe respuestas en formato JSON, las interpreta y las presenta al usuario de forma clara y amigable.

Servidor del sistema

El servidor, implementado con Java Spring Boot, representa la capa de lógica de negocio del sistema. Su función principal es procesar las solicitudes del cliente, interactuar con la base de datos y devolver respuestas estructuradas.

Rol del servidor:

Validar las solicitudes del cliente, como verificar que un socio tenga saldo suficiente para realizar una reserva.

Gestionar la interacción con la base de datos PostgreSQL, asegurando que las operaciones cumplan con las reglas de negocio.

Proveer una API RESTful para que el cliente pueda consumir los datos del sistema.

Funciones adicionales del servidor:

Seguridad: Proteger los datos sensibles mediante autenticación y autorización.

Manejo de transacciones: Asegurar la consistencia de las operaciones críticas, como la recarga de saldo o la reserva de actividades.

Generación de informes: Realizar cálculos y procesar datos para producir estadísticas y reportes solicitados por el webmaster.

Ventajas de la arquitectura

Separación de responsabilidades: La interfaz y la lógica de negocio están claramente separadas, lo que facilita el mantenimiento y la actualización del sistema.

Escalabilidad: El cliente y el servidor pueden ampliarse o reemplazarse de manera independiente.

Interoperabilidad: La API RESTful permite que otros sistemas externos también interactúen con el servidor si es necesario.

d. Interoperación

El sistema de gestión del gimnasio está diseñado para interactuar con sistemas externos con el objetivo de ampliar su funcionalidad y mejorar la experiencia del usuario. Las integraciones previstas son las siguientes:

1. Proyecto Tienda de Material deportivo

La tienda de material deportivo proporciona un catálogo de productos que será visible desde la web del gimnasio. Los usuarios podrán explorar los productos disponibles y, al seleccionar uno, serán redirigidos a la tienda para realizar la compra.

Grupo responsable: Grupo 11.

Persona de contacto: Joan Climent Quiñones (jcqc3@alu.ua.es).

Detalles técnicos:

Los productos se consumirán mediante un servicio API proporcionado por el sistema de la tienda.

Este apartado de la memoria se está redactando el día 16 de Enero de 2025 cuando aún no tenemos noticias de si este proyecto está funcionando o si lo podemos utilizar de alguna manera. Hemos contactado con el responsable del grupo y nos ha transmitido que “Aun no lo tienen en producción”, que “Lo tendrá entre hoy y mañana” o incluso nos ha proporcionado un enlace a un repositorio GitHub inexistente. Por lo tanto a 1 día de la entrega **no vamos a interoperar con este proyecto**, no vamos a realizar cambios en el código a 1 día de la entrega por algo que no depende de nosotros.

2. Proyecto TPV

La integración con el sistema de TPV permitirá a los socios recargar su saldo mediante una pasarela de pago segura con tarjeta bancaria.

Grupo responsable: Grupo 01.

Persona de contacto: Mollá González, Tono (tmgm5@alu.ua.es).

Detalles técnicos:

La recarga de saldo se realizará mediante llamadas al servicio API RESTful del sistema TPV.

El sistema TPV se encargará de validar la transacción y devolver una confirmación al servidor del gimnasio, indicando el éxito o el fallo del pago.

Integración en nuestro sistema

Este TPV ha sido utilizado para recargar saldo a los socios. El usuario introduce la cantidad a recargar de saldo:

Cargar Saldo

Saldo a cargar en €

Confirmar Carga

Es al darle al botón de “Confirmar Carga”, cuando se abre una nueva ventana con el TPV para realizar el pago:

Carga de saldo

150.00€

Número de tarjeta

 4242 4242 4242 4242

Fecha de expiración

MM/YY

CVC

123

Titular de la tarjeta

NOMBRE APELLIDOS

Pagar 150.00€

 Pago seguro con cifrado SSL

Si el pago es correcto el depósito realizado se cargará en el saldo del socio y el usuario visualizará una página como esta:

Saldo correctamente cargado.

Pulsa [aquí](#) para volver a la página del gimnasio.

Pudiendo entonces volver a la página del gimnasio. En caso de que falle el proceso de carga de saldo, se le indica al usuario:

Error al cargar el saldo:

Este ingreso de saldo no se puede hacer

Por favor, intenta nuevamente o contacta con soporte.

2. Mockups

A continuación se muestran los mockups realizados y se justifican los patrones de diseño web aplicados.

Página home

- Patrón de página principal tipo Portal. Se centralizan todos los servicios, sitios y aplicativos que ofrece una empresa o entidad a clientes o empleados. Con un único punto de login y con la misma imagen corporativa. Se muestra información según permisos o roles de usuario
- Patrón de navegación de primer nivel. Dando una visión global del sitio, permiten saltar de un lugar a otro rápidamente. Ubicado arriba en horizontal (limitado al ancho de pantalla) ocupando un lugar y aspecto destacado. Se oculta el menú cuando se ejecutan tareas con navegación propia como el registro de usuarios.

¡CAMBIA TU VIDA CON NOSOTROS!

[Ver actividades](#)[¡Únete ahora!](#)

Servicios destacados



Instalaciones

Contamos con más de 5 salas a tu disposición, sala de musculación, etc etc. No te faltará de nada.



Actividades

Actividades de todo tipo a tu disposición para ejercitar como nunca!



Monitores

¡Nuestro personal estará dispuesto a ayudarte con cualquier problema que te surja!



Precio

Entrenarás en las mejores instalaciones por un precio insuperable

Promociones

Plan 1

Descripción ventajas plan etc Lorem ipsum dolor sit amet

X€/mes

[Registrarme](#)

Plan 2

Descripción ventajas plan etc Lorem ipsum dolor sit amet

X€/mes

[Registrarme](#)

Plan 3

Descripción ventajas plan etc Lorem ipsum dolor sit amet

X€/mes

[Registrarme](#)[Cookies y privacidad del usuario](#)[Terminos Legales](#)[Preferencias de privacidad](#)[Contacto](#)[Propietarios](#)

ST Torero José María Manzanares 2, Alicante 03005, Spain

© 2024 Gimnasio. All rights reserved.

Solicitud de registro

- Patrones de formulario, alineado con las etiquetas de los campos. Con el fin de facilitar/agilizar el trabajo con el formulario se deben alinear adecuadamente las etiquetas con sus correspondientes campos. No se han evitado las etiquetas incrustadas.
- Patrones de formulario, formato de los datos: se permiten varios formatos válidos para campos como fechas, horas, teléfonos.
- Patrones de formulario, navegación con el teclado. Usa tabindex para ordenar los saltos (tabulaciones) entre los campos del formulario
- Patrones de formulario, botones de comando: usando botones para enviar el formulario, resaltar el botón de la acción principal. Los botones se etiquetan apropiadamente con verbos - acciones. Se utiliza un botón de submit.
- Patrones de formulario, mensajes de error: se muestran mensajes de error cerca del campo.
- Patrones de identificación de usuarios, Captcha: para evitar la creación fraudulenta de cuentas falsas de forma automática se solicita al usuario escribir letras o números que aparecen en una imagen distorsionada o velada.

The screenshot shows a web browser window with the URL 'www.gimnasio.com'. The page has a navigation bar with links: 'Mi perfil', 'Mis clases', 'Instalaciones', 'Aspectos legales', 'Suscripciones', 'Tienda online', and 'Monedero'. A search bar and a user profile icon are also present. The main heading is 'Solicitud de Registro'. Below it, there are several input fields: 'Nombre', 'Apellidos', 'Dirección', 'Teléfono', 'Fecha de nacimiento', 'Correo electrónico', 'Contraseña', and 'Confirmar contraseña'. Below the fields, there is a link: 'Registrar usuario mediante Google, Apple, etc.'. Below that, there is a checkbox labeled 'Declaro aceptar y haber leído los Términos y Condiciones'. Below the checkbox, there is a CAPTCHA image with the text 'I'm not a robot' and a link 'Cambiar a captcha sonoro'. At the bottom, there are two buttons: 'Cancelar' and 'Registrar Usuario'. The footer contains links: 'Condiciones y privacidad del usuario', 'Terminos Legales', 'Preferencias de privacidad', 'Contacto', and 'Propietarios'. It also includes the address 'ST Torero José María Manzanares 2, Alicante 03005, Spain' and the copyright notice '© 2024 Gimnasio. All rights reserved.'

Inicio de sesión

- Patrones de formulario, campos simples que indican al usuario qué incluir para iniciar la sesión.
- Patrones de formulario, formulario breve con intención clara y mensaje de error amigable cuando el usuario introduce datos erróneos.
- Patrones de identificación de usuarios, al producirse un error, los campos del formulario se validan y cuenta con un mecanismo de identificación único (email y contraseña).
- Patrones de formulario, se utiliza un botón de submit.

Editar mi perfil

- Patrones de formulario, alineado con las etiquetas de los campos. Con el fin de facilitar/agilizar el trabajo con el formulario se deben alinear adecuadamente las etiquetas con sus correspondientes campos. No se han evitado las etiquetas incrustadas.
- Patrones de formulario, botones de comando: usando botones para enviar el formulario, resaltar el botón de la acción principal. Los botones se etiquetan apropiadamente con verbos - acciones. Se utiliza un botón de submit.
- Patrones de formulario, navegación con el teclado. Usa tabindex para ordenar los saltos (tabulaciones) entre los campos del formulario
- Patrones de formulario, mensajes de error: se muestran mensajes de error cerca del campo.

Gimnasio

← → ↻ www.gimnasio.com

Gimnasio

Mi perfil | Mis clases | Instalaciones | Aspectos legales | Suscripciones | Tienda online | Monedero

Q Search

Mi Perfil

Nombre

Nombre

Apellidos

Apellidos

Nombre de usuario

Nombre de usuario

Correo electrónico

Correo electrónico

Dirección

Dirección

Fecha de nacimiento

Fecha de nacimiento

Cambia tu contraseña

Contraseña

Confirmar contraseña

Recargar Saldo

Borrar cuenta

Cancelar

Guardar

[Cookies y privacidad del usuario](#) | [Terminos Legales](#) | [Preferencias de privacidad](#) | [Contacto](#) | [Propietarios](#)

ST Torero José María Manzanares 2, Alicante 03005, Spain

© 2024 Gimnasio. All rights reserved.

Formulario de añadir fondos

- Patrones de formulario, mensajes de error: se muestran mensajes de error cerca del campo. Claros y concisos y fáciles de identificar para el usuario.
- Patrones de formulario, al usuario se le indica para qué sirven los datos que enviará, y cuánto dinero tiene en el momento.
- Patrones de formulario, alineado con las etiquetas de los campos. Con el fin de facilitar/agilizar el trabajo con el formulario se deben alinear adecuadamente las etiquetas con sus correspondientes campos. No se han evitado las etiquetas incrustadas.
- Patrones de formulario, botones de comando: usando botones para enviar el formulario, resaltar el botón de la acción principal. Los botones se etiquetan apropiadamente con verbos - acciones. Se utiliza un botón de submit.
- Patrones de formulario, navegación con el teclado. Usa tabindex para ordenar los saltos (tabulaciones) entre los campos del formulario



Monedero - Fondos actuales: x €

Añadir fondos

Inserta la cantidad de euros a añadir al monedero

Selecciona el método de pago

Select one

▼

Acceder a la pasarela de pago



Monedero - Fondos actuales: x €

Añadir fondos (Mostrando mensajes de error)

Inserta la cantidad de euros a añadir al monedero

Es obligatorio introducir la cantidad de euros a añadir al monedero

Selecciona el método de pago

Es obligatorio introducir el método de pago

▼

Acceder a la pasarela de pago

Actividades disponibles

- Patrones de listados, en este caso hemos utilizado un listado de eventos. Mostramos las actividades que existen en el gimnasio disponibles para el usuario y puede obtener información útil sobre cada una de ellas como su tipo, monitor a cargo o el horario.
- Patrones de listados, cada ítem tiene asociada la acción de poder apuntarse a una actividad determinada.
- Patrones de listados, se ha utilizado una lista tabular con varios atributos ordenables en función de su tipo u horario.
- Patrones de listados, se han añadido utilidades como la acción de apuntarse, además de una botonera que nos permite movernos entre las páginas de la misma.
- Patrones de listados, al usar una lista tabular hemos permitido al usuario mostrar la lista en formato calendario también.



Asistentes a actividad

- Patrones de listados, se ha utilizado una lista tabular.
- Patrones de listados, cada ítem tiene varios datos sobre los usuarios que asisten a una actividad concreta.
- Patrones de listados, se ha utilizado una lista tabular con varios atributos ordenables en función de su ID.



Actividades a las que estás apuntado

- Patrones de listados, se ha utilizado una lista de eventos o calendario. Al tener cada actividad una fecha concreta hemos implementado la vista de calendario.
- Patrones de listados, se le permite al usuario cambiar a la vista normal para poder buscar ítems.
- Patrones de listados, se ha permitido (en producción) cambiar la vista a diaria, semanal o mensual y los días con eventos están resaltados.

Gimnasio

www.gimnasio.com

Gimnasio

Mi perfil

Mis clases

Instalaciones

Aspectos legales

Suscripciones

Tienda online

Monedero

Search

Actividades a las que estás apuntado

November 25, 2024

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
		Clase Zumba 18:00				
10	11	12	13	14	15	16
				Deporte al aire libre		Clase Musculación
17	18	19	20	Clase Spinning 17:00	22	23
24	Taller estiramiento	26	27	28	29	30
		Clase Crossfit		Clase Spinning 17:00		

Cookies y privacidad del usuario

Terminos Legales

Preferencias de privacidad

Contacto

Propietarios

ST Torero José María Manzanares 2, Alicante 03005, Spain

© 2024 Gimnasio. All rights reserved.

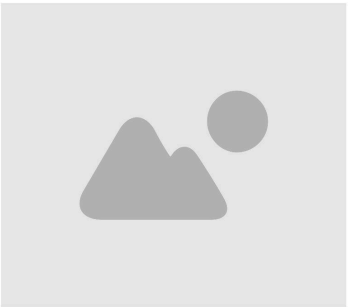
Apúntate a una actividad y detalle de actividades

- Patrones de listados, se ha utilizado una lista simple. En ella mostramos todos los datos de la actividad en cuestión.
- Patrones de navegación, se incluye un botón que pide la confirmación del usuario para formar parte de la actividad. El usuario es redirigido a la página que le confirma su reserva.



Apúntate a NOMBREACTIVIDAD

Inicio



(Descripción) Clase de Yoga, apúntate para estar en paz contigo mismo y disfrutar de una compañía maravillosa etc etc

(Fecha) 14 diciembre 2024, 18:30

(Lugar) Sala multiusos

(Nivel) Principiante

(Monitor) John Doe

Me apunto!



Spinning

15€ por clase 12 plazas restantes

La clase tendrá lugar el día **15 de diciembre** en el **pabellón principal** del gimnasio.

La clase comenzará a las **16:00** con una duración de **2 horas**.

Monitor a cargo:



Nombre:

E-mail:

Teléfono de contacto:

[Condiciones y privacidad](#) | [Terminos Legales](#) | [Preferencias de privacidad](#) | [Contacto](#) | [Propietarios](#)

ST Torero José María Manzanares 2,
Alicante 03005, Spain

Aceptar o rechazar solicitudes de socio por parte del webmaster

- Patrones de listados, se ha utilizado una lista tabular. Cuenta con row stripping para hacer más legible la tabla y se alinean los datos de las celdas en base a su tipo de valor.
- Patrones de navegación, abajo del listado se ha incluido una botonera útil para navegar entre registros de la tabla.
- Patrones de listados, se permite al webmaster filtrar por nombre de usuario en caso de necesitar encontrar más rápido al solicitante.
- Patrones de listado, añadidas acciones dedicadas a cada ítem, se incluye un botón que permite al webmaster encargado rechazar o aceptar una solicitud de un nuevo usuario, así como ver los detalles del mismo



Crear una nueva actividad

- Patrones de formulario, alineado con las etiquetas de los campos. Con el fin de facilitar/agilizar el trabajo con el formulario se deben alinear adecuadamente las etiquetas con sus correspondientes campos. No se han evitado las etiquetas incrustadas.
- Patrones de formulario, formato de los datos: se permiten varios formatos válidos para campos como fechas u horas.
- Patrones de formulario, navegación con el teclado. Usa tabindex para ordenar los saltos (tabulaciones) entre los campos del formulario
- Patrones de formulario, botones de comando: usando botones para enviar el formulario, resaltar el botón de la acción principal. Los botones se etiquetan apropiadamente con verbos - acciones. Se utiliza un botón de submit.
- Patrones de formulario, mensajes de error: se muestran mensajes de error cerca del campo.




¡Crea una nueva actividad!


Tipo de actividad

▼


Fecha



Hora de inicio



Duración



Nº de plazas

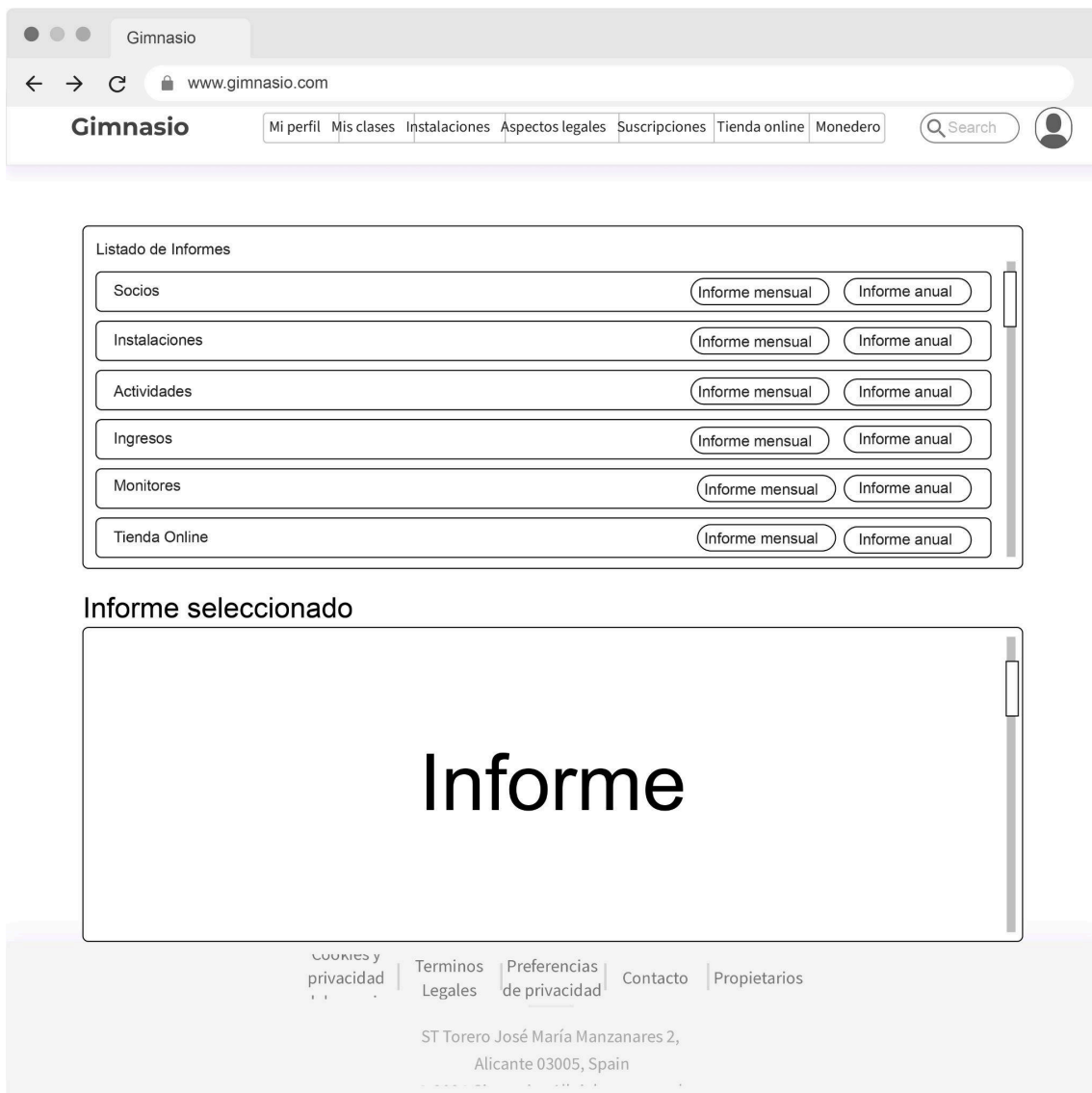
Dónde se llevará a cabo

▼

Confirmar

Listado de informes

- Patrones de búsqueda avanzada, el webmaster puede seleccionar en función de diversos parámetros (tipo de informe, mensual, anual...) el informe que desee.
- Patrones de búsqueda en el sitio, el resultado se obtiene de forma clara y ordenada para evitarle al usuario una segunda búsqueda. Se muestra información descriptiva de lo que desea.
- Patrones de formulario, se muestran instrucciones de los informes que se le permite solicitar al usuario, así como botones para enviar la solicitud de información.
- Patrones de listados, se ha utilizado una lista simple con dos acciones asociadas por ítem (informe mensual y anual).



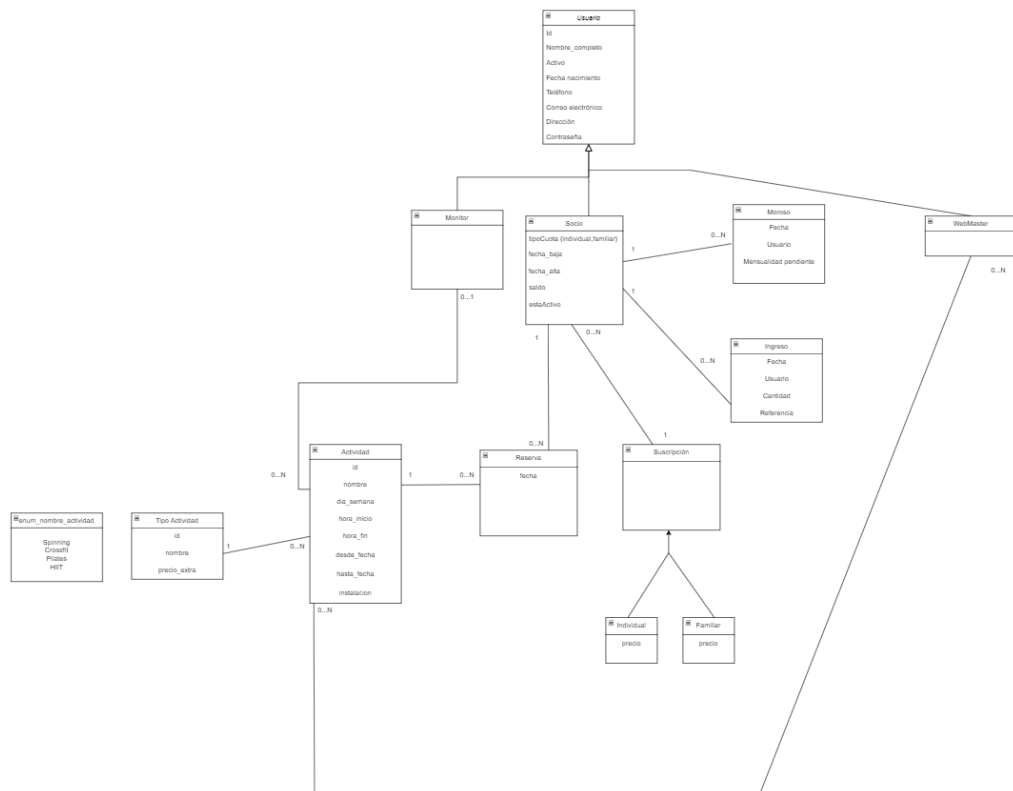
Listado de informes

- Patrones de listados, se ha utilizado una lista simple con datos como el título de la cuota, el precio y un botón asociado que indica al usuario que se apunte.
- Patrones de formulario, botones de comando: usando botones para enviar el formulario, resaltar el botón de la acción principal. Los botones se etiquetan apropiadamente con verbos - acciones. Se utiliza un botón de submit.



3. Diagramas

A continuación se presenta el diagrama UML que representa todo el sistema y que representa también nuestro esquema de bases de datos.



Si no se puede visualizar bien la imagen debido a su contenido, se puede descargar el diagrama UML desde este [enlace](#). El fichero descargado es un HTML en el que se puede hacer zoom sin problema alguno para visualizar al completo todo el diagrama.

Continuando con la explicación del diagrama podemos ver las siguientes clases:

- **Usuario**: representa la entidad genérica a partir de la cual se distingue entre tres entidades distintas:
 - **Monitor**: representa a los monitores del gimnasio, pudiendo crear nuevas actividades, consultar la asistencia a clase, o consultar su propio horario de clases.
 - **Socio**: representa a uno de los usuarios estrella del sistema, siendo quien realiza reservar, carga saldo o consulta su horario.

- WebMaster: representa la entidad webmaster, siendo este equivalente a un administrador del sistema, pudiendo: bloquear o admitir usuarios, crear nuevas actividades, visualizar informes, ver impagos de usuarios, crear nuevas cuotas mensuales.
- Actividad: representa la entidad actividad, son eventos a los que se puede apuntar un socio. Todas las actividades tienen un:
 - TipoActividad: representa un conjunto numerado de tipos de actividades: Spinning, Crossfit, Pilates y HIT. Establecen el precio extra de las actividades.
- Reserva: representa la entidad reserva, lo que se refiere a la relación entre un socio y una actividad en una fecha determinada. Se ha seguido la siguiente estrategia de reservas: indicar en cada actividad su día de inicio (ejemplo 1/1/2025), su día de finalización (ejemplo 31/12/2025) y el día de la semana que toma lugar (ejemplo Martes). De este modo siguiendo el ejemplo que hemos planteado, durante el año 2025 tendremos la clase todos los martes, entonces cuando el usuario seleccione una fecha en el calendario (ejemplo 21/10/2025) tendremos una reserva de la actividad en una fecha determinada para un socio determinado.
- Suscripción: representa los tipos de suscripciones del sistema. En nuestro caso:
 - Individual: con un precio de 30 euros
 - Familiar: con un precio de 60 euros
- Ingreso: representa las operaciones satisfactorias de ingresos realizados, almacenando también la referencia que se guarda en el TPV para tener todos los pagos controlados.
- Moroso: representa un impago de cuota de un socio, almacenando la cuota no pagada.

4. Metodología y planificación

Para el desarrollo del proyecto se ha seguido una metodología ágil siendo la elegida una variación de SCRUM en la que nos hemos organizado por historias de usuario. Aprovechando que habíamos elegido como sistema de control de versiones la plataforma GitHub, decidimos utilizar las issues como historias de usuario y los GitHub Projects como tableros de tareas.

En primer lugar decidimos organizar todas las tareas en Issues de tipo “Epic”, estas issues son usadas en GitHub para organizar pequeñas tareas en una gran tarea que las englobe a todas. Siguiendo esta idea creamos las siguiente 5 Issues “Epic”, creadas por los casos de uso o perfiles de uso:

- Usuario no registrado: agrupamos aquí las issues o historias necesarias para lograr completar los requisitos de los usuarios no registrados en el sistema.
 - Crear un formulario de alta para nuevos socios
 - Visualización de la tienda online con productos de terceros



- Usuarios WebMaster: agrupamos aquí las issues o historias necesarias para lograr completar los requisitos de los webmasters.
 - Aprobar o rechazar solicitudes
 - Activar o bloquear socios
 - Añadir actividades asignando horario, sala y monitor
 - Visualización de informes
 - Ver un listado de morosidades

Epic Usuario WebMaster #12

Open



jmgm62-ua opened last month · edited by jmgm62-ua

Edits ...

Se deben completar las siguientes issues para poder cerrar esta issue epic:

- Aprobar o rechazar las solicitudes de nuevos socios - [Aprobar o rechazar solicitudes #26](#)
- Activar o bloquear socios - [Gestion de cuentas de usuario #9](#)
- Añadir actividades asignando horario sala y monitor - [Crear nueva actividad #23](#)
- Visualización de informes - [Informes #28](#)
- [Listado de usuarios "morosos" #49](#)

Create sub-issue



- Usuarios Socios: agrupamos aquí las issues o historias necesarias para lograr completar los requisitos de los socios.
 - Recargar saldo
 - Reservar actividades
 - Gestión de reservas, en modo lista o calendario
 - Modificar cuota
 - Ver tienda

Epic Usuario Socio #13

Open



jmgm62-ua opened last month · edited by jmgm62-ua

Edits ...

Se deben completar las siguientes issues para poder cerrar esta issue epic:

- Recargar saldo - [Recarga de saldo #16](#)
- Reservar actividades - [Reservar actividad #18](#)
- Gestión de reservas - [Gestion de reservas #17](#)
- Modificar cuota - [Modificar cuota #20](#)
- Ver tienda - [Visualización de la tienda online con productos de terceros #8](#)

Create sub-issue

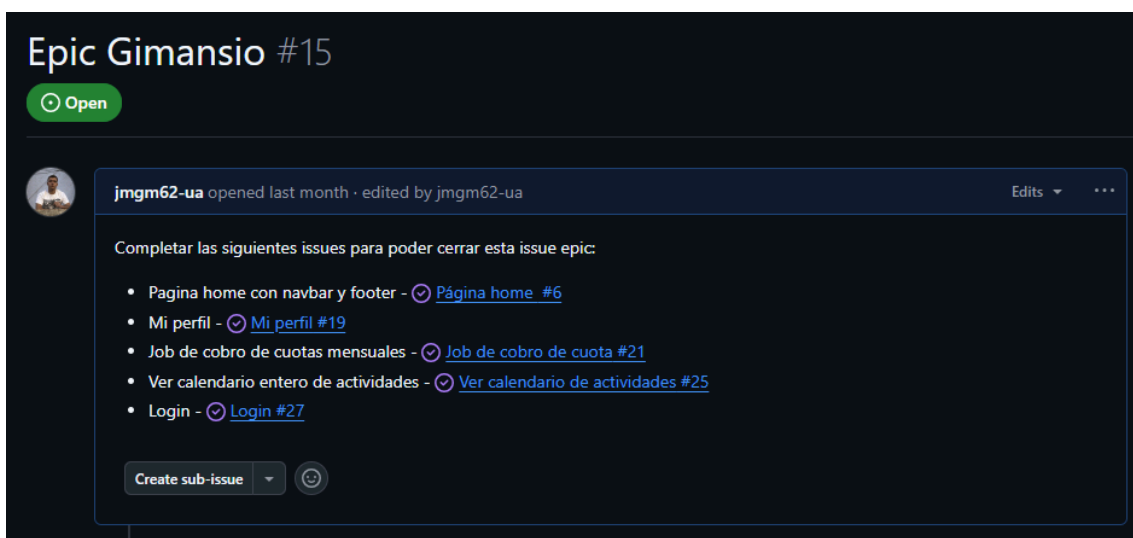


- Usuarios Monitor: agrupamos aquí las issues o historias necesarias para lograr completar los requisitos de los monitores.
 - Ver mis actividades en calendario o lista
 - Añadir nuevas actividades
 - Ver asistentes a mis clases



Esta agrupación de tareas Gimnasio es una agrupación de tareas que no eran para un perfil específico sino que eran generales a todos los perfiles:

- Página home
- Mi perfil, que es distinto para cada rol.
- Job de cobro de cuotas mensuales, para cobrar las mensualidades automáticamente
- Ver calendario entero de actividades
- Formulario de inicio de sesión

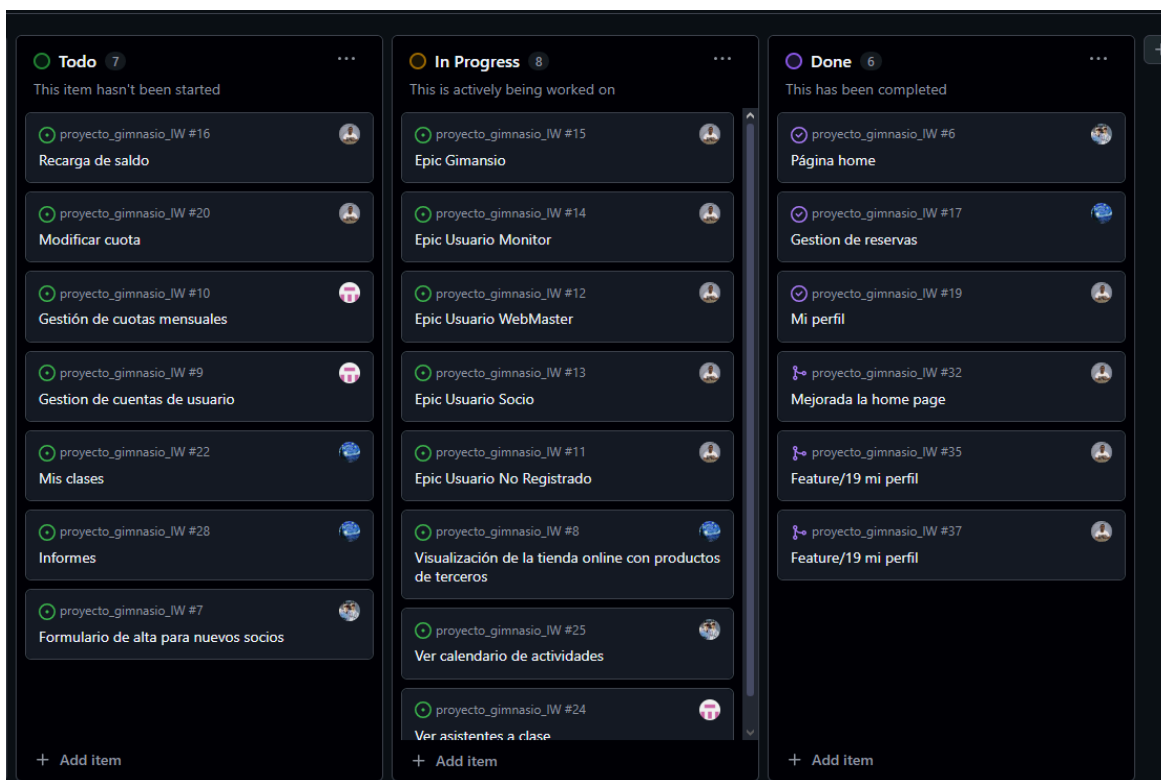


Esta manera de organizar las tareas ha sido muy útil para Jose Megía que ha asumido el rol de “Líder de equipo” y ha podido seguir fácilmente el progreso en conjunto de las tareas.

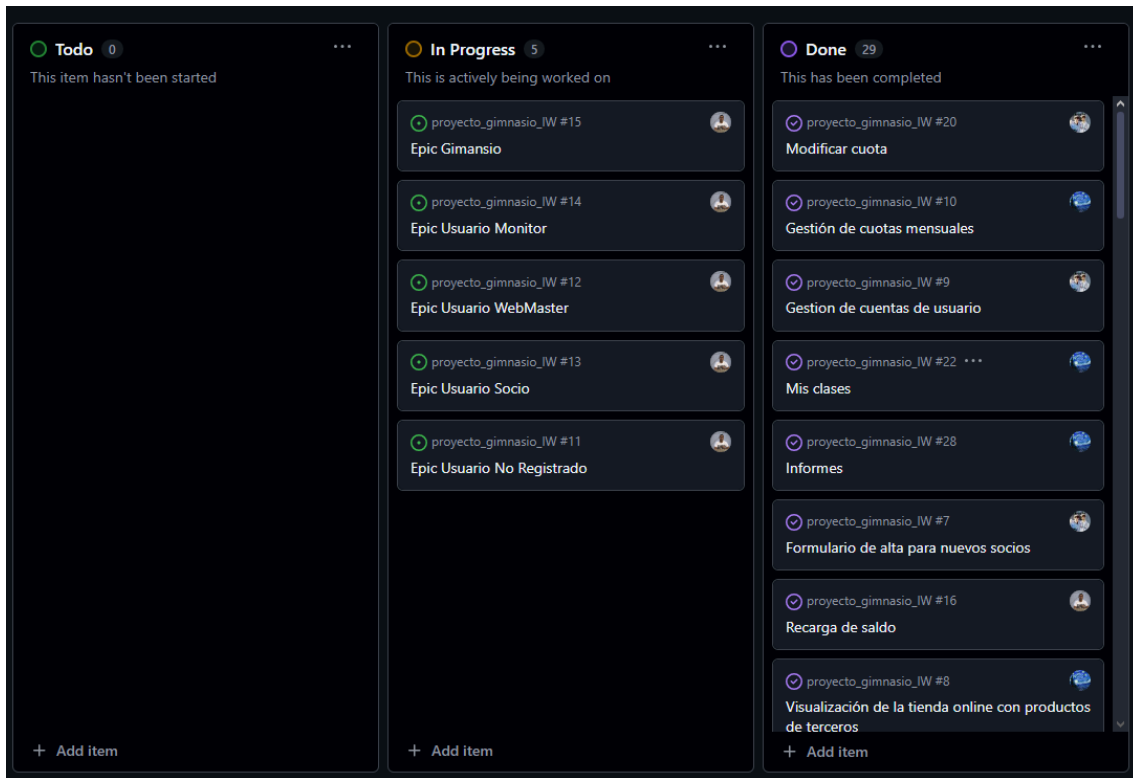
Continuando con la planificación, José Megía realizó la asignación de tareas entre los desarrolladores en función del perfil de desarrollo de cada persona y del tiempo disponible que iban a tener. En un principio se realizó la asignación a 4 desarrolladores: Alejandro, Nicolas, Adrián y Jose, sin embargo, Alejandro nos comunicó que no podría contribuir al desarrollo del proyecto por problemas personales. Fue entonces cuando Jose asignó el trabajo entre 3 desarrolladores: Nicolás, Adrián y Jose.

A los perfiles más creativos como el de Nicolás y Adrián se les asignó tareas que requerían interfaces importantes del sistema. Adrián, fue asignado también con tareas relacionadas con el mantenimiento de la base de datos por su facilidad para trabajar con bases de datos relacionales, por ejemplo, creó el seeder de la base de datos e implantó el uso de hibernate en el SpringBoot. A Jose se le asignaron actividades más críticas del sistema, como la reserva de actividades o el cobro de cuotas mensuales.

Durante el desarrollo, los desarrolladores indicaban el progreso actual en un GitHub Projects usado a modo de tablero. Aquí se podía ver que tareas estaban por hacer, cuales estaban en progreso y cuales estaban terminadas. Las tareas “Epic” siempre permanecen en progreso.

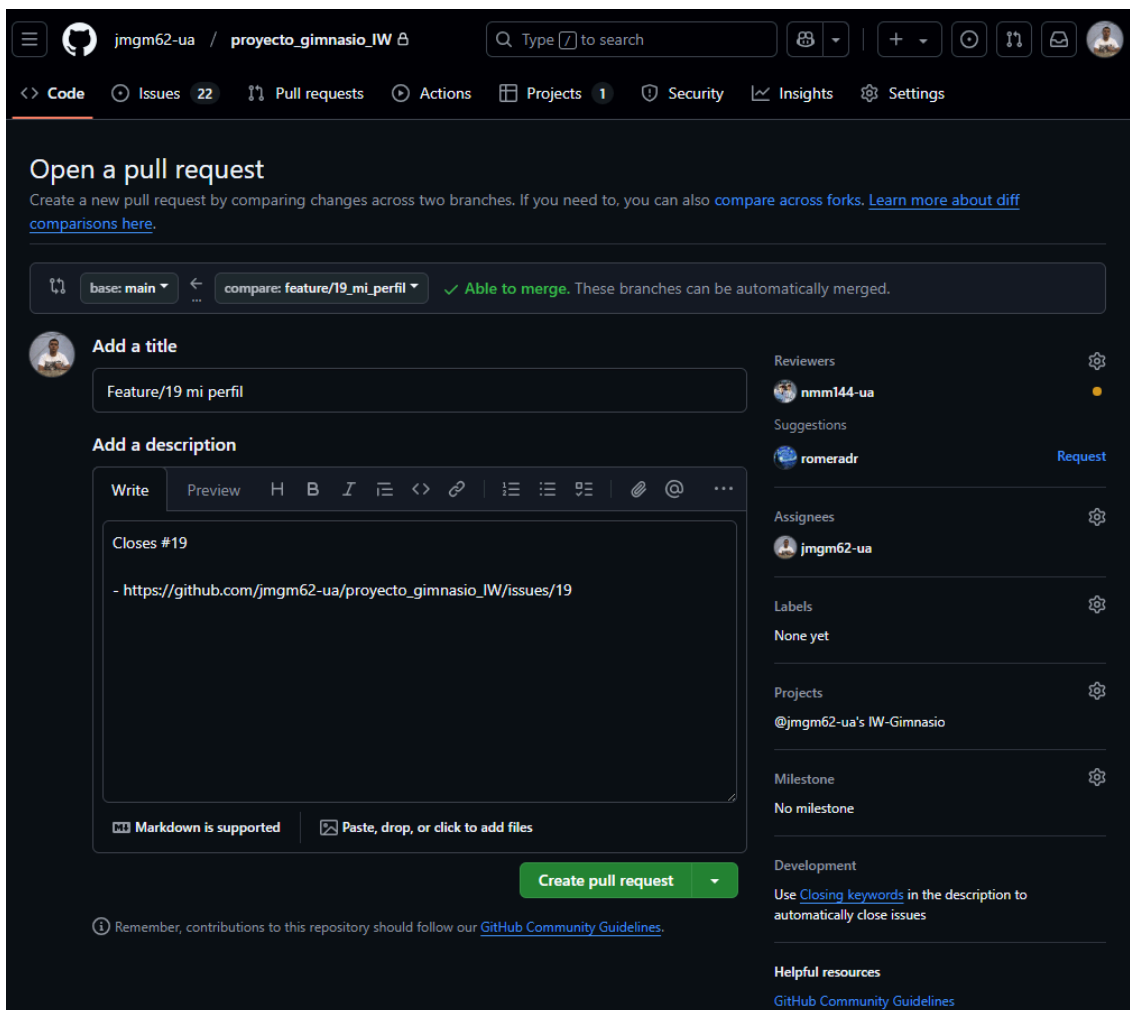


Una vez finalizado el proyecto, todas las tareas quedaban en la columna de “terminadas”, a excepción de las “Epic” que por norma interna siempre iban a estar “En progreso”. Así quedaba el tablero de tareas:

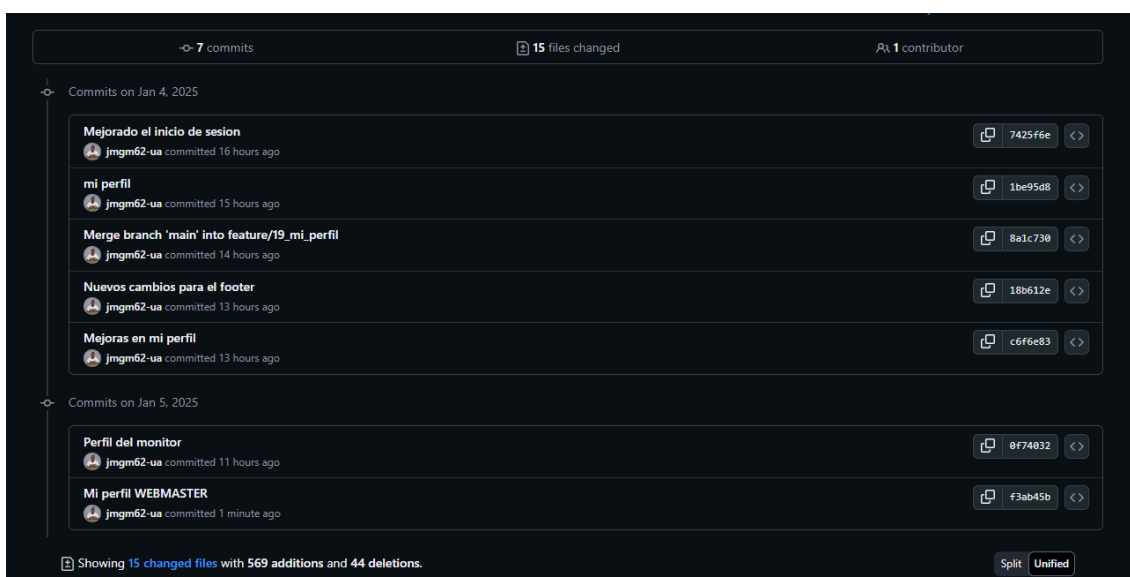


Para facilitar la implementación del proyecto y disminuir la aparición de errores nuestro código ha tenido que ser revisado por nuestros compañeros haciendo uso de los “Pull Requests” de Github. Describimos a continuación el proceso:

- 1) Creamos un “Pull Request” y seleccionamos a un compañero como revisor de código. Este compañero revisa el código y permite o no introducirlo en la rama principal de desarrollo:



2) El revisor puede ver que y cuantos commits nuevos se han introducido:



3) El revisor puede ver que cambios se han introducido en cada fichero modificado:

Showing 15 changed files with 569 additions and 44 deletions.

```
@@ -14,15 +14,15 @@ public class ManagerUserSession {
14 14 // una autorización sencilla. En los métodos de controllers
15 15 // comprobamos si el id del usuario logeado coincide con el obtenido
16 16 // desde la URL
17 - public void logearUsuario(Long idUsuario) {
18 - session.setAttribute("idUsuarioLogeado", idUsuario);
17 + public void logearUsuario(String email) {
18 + session.setAttribute("emailUsuarioLogeado", email);
19 19 }
20 20
21 21 public Long usuarioLogeado() {
22 - return (Long) session.getAttribute("idUsuarioLogeado");
22 + return (Long) session.getAttribute("emailUsuarioLogeado");
23 23 }
24 24
25 25 public void logout() {
26 - session.setAttribute("idUsuarioLogeado", null);
26 + session.setAttribute("emailUsuarioLogeado", null);
27 27 }
28 28 }
```

- 4) Por último el revisor indica si aprueba los nuevos cambios o indica si se deben realizar correcciones de código:

Finish your review

Write

Preview

H

B

I

≡

<>

↗

≡

≡

≡

📎

@

↗

←

Todo correcto

Markdown is supported | Paste, drop, or click to add files

☐ Comment

Submit general feedback without explicit approval.

☒ Approve

Submit feedback and approve merging these changes.

☐ Request changes

Submit feedback that must be addressed before merging.

Submit review

De esta manera hemos conseguido disminuir la cantidad de errores introducidos en el proyecto.

5. Descripción de la implementación

Implementación en el backend

Para llevar a cabo la implementación en el backend hemos tomado varias decisiones que justificamos a continuación:

- 1) Uso de hibernate y JPA: Como ya hemos mencionado anteriormente desde el backend realizamos los accesos a la base de datos, aprovechando que utilizabamos el framework SpringBoot decidimos utilizar JPA e hibernate para facilitar la persistencia de datos y la gestión de la base de datos PostgreSQL. De esta manera con la capa de modelo conseguimos la creación de tablas y relaciones en la propia base de datos.

Ejemplo de elemento de la capa de modelo para crear una tabla en la base de datos.

```
@Entity
@Data
@Table(name = "actividades")
public class Actividad {

    @Id
    private Long id;
```

Ejemplo de atributos de la clase Actividad de la capa de modelo, estos atributos de la clase son luego las columnas de la tabla Actividad:

```
@Id
private Long id;

private String nombre;
private String diaSemana;

@Column(name = "fecha_inicio")
private Date fechaInicio;

@Column(name = "fecha_fin")
private Date fechaFin;

private String horaInicio;
```

id	dia_semana	fecha_fin	fecha_inicio	hora_fin	hora_inicio
1	Lunes	2026-12-31 23:59:00.000000	2025-01-01 00:00:00.000000	20:00	19:00
2	Martes	2026-12-31 23:59:00.000000	2025-01-01 00:00:00.000000	20:00	19:00
3	Miércoles	2026-12-31 23:59:00.000000	2025-01-01 00:00:00.000000	20:00	19:00
4	Jueves	2026-12-31 23:59:00.000000	2025-01-01 00:00:00.000000	20:00	19:00
5	Viernes	2026-12-31 23:59:00.000000	2025-01-01 00:00:00.000000	20:00	19:00

Ejemplo de relaciones entre entidades, en el Socio de la capa de modelo añadimos las reservas de esta manera:

```
@Getter
@OneToMany(mappedBy = "socio")
private List<Reserva> reservas;
```

Y en la entidad Reserva, añadimos lo siguiente:

```
@ManyToOne
@JoinColumn(name = "socio_id")
private Socio socio;
```

Hemos creado de esta manera una relación uno a muchos entre Reserva y Socio. Vemos entonces que la gestión entre los modelos y la base de datos se facilita mucho.

- 2) Separación en capas de: Modelo, Data Transfer Object, Repository, Controller y Servicios. Esta decisión de separar el código en capas responde al principio de responsabilidad única y busca una arquitectura limpia y mantenible.

- a) Modelo:

Representa las entidades del dominio y su estructura.

Define cómo se relacionan las tablas de la base de datos con objetos Java.

Esta capa debe enfocarse únicamente en representar datos.

- b) Data Transfer Object

Facilita la transferencia de datos entre capas, evitando exponer directamente el modelo.

Permite mapear y manipular datos para casos de uso específicos sin modificar las entidades principales

c) Repository

Abstrae el acceso a la base de datos.

Se encarga exclusivamente de las operaciones CRUD y consultas.

Mantiene el código de persistencia separado de la lógica de negocio.

d) Servicio

Contiene la lógica de negocio.

Orquesta la interacción entre el controlador y el repositorio.

Garantiza que las reglas y procesos específicos del dominio se apliquen correctamente.

e) Controlador

Es el punto de entrada para las peticiones del cliente.

Maneja las solicitudes HTTP, delegando las operaciones de negocio al servicio y regresando las respuestas apropiadas.

Logramos de esta manera maximizar la modularidad de nuestro código, reutilizar la lógica de negocio en tareas similares y facilitar la escalabilidad del sistema.

- 3) Creación de un seeder. Decidimos crear un seeder para trabajar todos en base a los mismos datos y obtener resultados coherentes entre todos durante el desarrollo. El script del seeder está en lenguaje SQL.

```
-- Instalaciones (Mantenido igual)

INSERT INTO public.instalaciones (id, nombre) VALUES

(1, 'Sala principal'),

(2, 'Sala de ciclo'),

(3, 'Sala general 1'),
```

```
(4, 'Sala general 2'),  
(5, 'Vestuarios'),  
(6, 'Recepción'),  
(7, 'Cafetería'),  
(8, 'Sala para estiramientos');
```

- 4) Creación de un “job” para el cobro de cuotas mensuales. Al principio no sabíamos muy bien cómo abordar la tarea del cobro de cuotas mensuales hasta que decidimos usar un “job”, es decir un hilo de ejecución independiente del principal de la aplicación que es ejecutado el día 1 de cada mes para efectuar el cobro de cuotas.

```
@Scheduled(cron = "0 0 0 1 * *")  
  
public void descontarCuota() {  
  
    LocalDate fechaActual = LocalDate.now();  
  
    if (fechaActual.getDayOfMonth() == 1) {  
  
        List<Socio> lista_socios = socioRepository.findAll();  
  
        for (Socio socio : lista_socios){  
  
            if (socio.getActivo()){  
  
                String cuota_usuario = socio.getTipoCuota();  
  
                float price_cuota = get_precio_cuota(cuota_usuario);  
  
                descontar_saldo(socio, price_cuota);  
  
            }  
  
        }  
  
    }  
  
}
```

Explicación: haciendo uso de `@Scheduled(cron = "0 0 0 1 * *")` podemos ejecutar el código de la función "descontarCuota" cada día 1 de cada mes. Conseguimos así automatizar esta tarea sin necesidad de ser ejecutada por algún usuario. Además al descontar el saldo si se detecta que un socio no tiene saldo suficiente será desactivado y considerado como un moroso, almacenando el mes que no ha pagado (código completo en la carpeta de servicios en el fichero CuotaService).

```
if(socio.getSaldo() < price_cuota){  
  
    Moroso nuevoMoroso = new Moroso();  
  
    nuevoMoroso.setId_socio(socio.getId());  
  
    String fechaMesAño = currentDate.format(formatter);  
  
    nuevoMoroso.setMensualidad_no_pagada(fechaMesAño);  
}
```

- 5) Manejo propio de las IDs asignadas: mientras que jpa trataba de seguir su comportamiento por defecto, trataba de asignar él las IDs sin embargo, al utilizar nosotros un script de seeder, JPA no tenían en cuenta las IDs introducidas por el seeder ya que no las había introducido él mismo. Fue entonces cuando tuvimos que gestionar nosotros manualmente la asignación de ID, buscando entre todos los elementos de la tabla la ID más alta y asignando entonces la siguiente.

```
Long nextId = 1L; // Valor por defecto si no hay reservas  
  
List<Moroso> allMorosos = morosoRepository.findAll();  
  
if (!allMorosos.isEmpty()) {  
  
    Long maxId = allMorosos.stream()  
  
        .mapToLong(Moroso::getId)  
  
        .max()  
  
        .orElse(0L);  
  
    nextId = maxId + 1;  
  
}  
  
nuevoMoroso.setId(nextId);  
  
morosoRepository.save(nuevoMoroso);
```

Implementación en el frontend

El frontend ha sido desarrollado en el framework Vue, y hemos tomado varias decisiones claves:

- Desarrollo en componentes: el funcionamiento de vue se basa en los componentes, de esta manera hemos podido reutilizar código y trabajar de manera independiente fácilmente.
- Uso del router de Vue: el router es una pieza clave de nuestro desarrollo ya que nos permite cambiar fácilmente entre componentes.

```
import { useRouter } from 'vue-router';

this.$router = useRouter();

this.$router.push('/');
```

```
{

  path: '/',

  name: 'home',

  component: HomeView,

},
```

En este caso podemos observar cómo definimos la ruta "/" e indicamos que se corresponde con el componente HomeView (página principal) y vemos también cómo con 3 líneas podemos cambiar entre componentes haciendo uso de push.

- Uso de almacén centralizado de Pinia: de esta manera hemos podido almacenar las credenciales del usuario para identificarlo fácilmente en cada petición realizada al backend. Este almacén se podía modificar desde cualquier pantalla del sistema.

```
import { defineStore } from 'pinia';

import piniaPersist from 'pinia-plugin-persistedstate';

export const useUserStore = defineStore('user', {

  state: () => ({
```

```
    userType: null,  
  
    isLoggedIn: false,  
  
    id: null,  
  
    email: null,  
  
    nombre: null,  
  
    fechaNacimiento: null,  
  
    tipo_suscripcion: null,  
  
    direccion: null,  
  
    actividad_a_reservar: null,  
  
    fecha_seleccionada: null,  
  
  }),
```

Portabilidad del proyecto

Una decisión clave del desarrollo de nuestro proyecto fue utilizar Docker, de esta manera conseguimos que nuestro proyecto sea fácilmente portable entre máquinas y sistemas operativos. Así podemos desplegar nuestra aplicación en cualquier situación y máquina con un par de comandos.

6. Problemas encontrados y su solución

Durante el desarrollo del proyecto, nos han surgido varios problemas relacionados con la implementación. En este apartado los describiremos uno por uno y adjuntamos piezas de código para ilustrar cómo han sido solucionados.

Uso de Data Transfer Objects

Al comenzar la implementación de nuestro backend en SpringBoot, no teníamos demasiado claro cómo devolver los datos que obteníamos de la base de datos directamente al cliente que hacía la petición. Para ello, hicimos uso de los DTOs o Data Transfer Objects, que nos permitieron abstraer y estructurar la información de manera que fuera más adecuada para su consumo por parte del cliente, sin exponer directamente los modelos de la base de datos.

Este enfoque no solo simplifica la implementación, sino que también evita conflictos con los modelos de la base de datos y garantiza que los datos se presenten de manera consistente y segura. Utilizamos el mapeo para convertir los objetos recuperados en una lista de DTOs, lo que nos permitió devolver únicamente la información necesaria para la vista, optimizando así el rendimiento y la claridad del código.

Este uso de DTOs no solo mejoró la organización del código, sino que también facilitó la mantenibilidad y escalabilidad de nuestra aplicación, al separar claramente la lógica de negocio de la presentación de los datos. En la imagen se muestra una pieza de código de cómo fueron utilizados estos DTOs para la obtención de datos:

```
public ArrayList<ActividadData> findActividadesByMonitorEmail(String email) {
    Optional<Monitor> monitor = monitorRepository.findByEmail(email);

    return monitor.map(value -> actividadRepository.findById(value.getId())
        .stream()
        .map(actividad -> new ActividadData(
            actividad.getId(),
            actividad.getNombre(),
            actividad.getDiaSemana(),
            actividad.getHoraInicio(),
            actividad.getHoraFin(),
            actividad.getFechaInicio(),
            actividad.getFechaFin(),
            actividad.getTipoActividad().getNombre(),
            actividad.getTipoActividad().getPrecioExtra()
        ))
        .collect(Collectors.toCollection(ArrayList::new)))
        .orElse(null);
}
```

Problema con la estrategia de generación de IDs en Hibernate

Inicialmente, al diseñar los modelos de datos para la base de datos en PostgreSQL utilizando Hibernate JPA en Spring Boot, configuramos la estrategia de generación de IDs como "identity". Sin embargo, decidimos eliminarla debido a conflictos que surgían al introducir datos manualmente, como en el caso del seeder, lo que provocaba fallos de consistencia en la base de datos. Decidimos como equipo manejar manualmente la asignación de IDs porque de lo contrario sería imposible introducir datos para una demostración en vivo.

Antes:

```
@Entity
@Data
@Table(name = "actividades")
public class Actividad {

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

Después:

```
@Entity
@Data
@Table(name = "actividades")
public class Actividad {

    @Id
    private Long id;
```

Algunos de los errores que nos surgían sin este necesario cambio eran del estilo: “el ID introducido ya existe en la base de datos, la primary key se ha violado”, dado que Hibernate no estaba asignando correctamente el ID de los objetos a insertar.

Problema con la triple herencia

Como se puede apreciar en nuestro diagrama de clases, la clase Usuario actúa como clase padre de las clases Socio, Monitor y Webmaster, las cuales heredan sus atributos y comportamientos. En la implementación con Spring Boot y Hibernate JPA, optamos por utilizar la estrategia de mapeo Table per Class, que genera una tabla independiente para cada una de las clases hijas, junto con una tabla principal users que almacena los datos comunes a todos los tipos de usuarios.

Sin embargo, durante el desarrollo nos encontramos con un problema relacionado con la generación de IDs. En Hibernate, cuando se utiliza el algoritmo de generación Identity, las IDs se generan de manera única por tabla. Esto significa que cada tabla hija (socios, monitores y webmasters) generaba sus propias IDs de forma independiente, lo que inicialmente parecía funcional. El inconveniente surgió al intentar realizar inserciones o establecer relaciones mediante Foreign Keys, ya que, al ser todos tipos de usuarios, existía la posibilidad de que una ID generada en una tabla hija coincidiera con una ID ya existente en otra tabla hija. Esto provocaba errores de duplicación, dado que, en esencia, todos los registros representaban usuarios, aunque fueran de distintos tipos.

Para resolver este problema, implementamos una solución basada en una secuencia de SQL. Creamos una secuencia compartida por todas las tablas de usuarios (socios, monitores y webmasters), la cual se autoincrementaba de manera única y global. En Spring Boot, modificamos la estrategia de generación de IDs a SEQUENCE, indicando el nombre de la secuencia creada. Este enfoque garantizó que las IDs fueran únicas en todas las tablas, evitando conflictos y asegurando la consistencia de los datos al realizar inserciones o establecer relaciones entre las entidades.

```

-- Reset sequences
ALTER SEQUENCE public.user_id_seq RESTART WITH 1;

-- Eliminar la secuencia si ya existe (usando CASCADE para eliminar dependencias)
DROP SEQUENCE IF EXISTS public.user_id_seq CASCADE;

-- Crear la secuencia para generar IDs únicos
CREATE SEQUENCE public.user_id_seq START WITH 1 INCREMENT BY 1;

-- Configurar la tabla socios para usar la secuencia
ALTER TABLE public.socios ALTER COLUMN id SET DEFAULT nextval('public.user_id_seq');

-- Configurar la tabla monitores para usar la secuencia
ALTER TABLE public.monitores ALTER COLUMN id SET DEFAULT nextval('public.user_id_seq');

-- Configurar la tabla web_masters para usar la secuencia
ALTER TABLE public.web_masters ALTER COLUMN id SET DEFAULT nextval('public.user_id_seq');

```

Esta solución no sólo resolvió el problema de duplicación de IDs, sino que también mejoró la integridad y escalabilidad del sistema, permitiendo una gestión más robusta de los usuarios y sus tipos asociados. Además, destacamos la importancia de comprender cómo las estrategias de generación de IDs pueden afectar el diseño de la base de datos y la lógica de negocio, especialmente en modelos de herencia como el implementado.

Interoperación con la tienda online de productos deportivos

El caso de la tienda online nos fue un poco más complejo. Intentamos colaborar con el equipo encargado de su desarrollo, pero no se nos proporcionaron los recursos necesarios, como endpoints válidos o un Dockerfile, lo que hizo imposible implementar la funcionalidad completa de la tienda online.

```
// Datos mock expandidos
const mockCategories = [
  { id: 1, nombre: "Fútbol" },
  { id: 2, nombre: "Tenis" },
  { id: 3, nombre: "Pádel" }
]

const mockSubcategories = {
  1: [
    { id: 1, nombre: "Botas" },
    { id: 2, nombre: "Balones" },
    { id: 3, nombre: "Equipaciones" }
  ],
  2: [
    { id: 4, nombre: "Raquetas" },
    { id: 5, nombre: "Pelotas" },
    { id: 6, nombre: "Complementos" }
  ],
  3: [
    { id: 7, nombre: "Palas" },
    { id: 8, nombre: "Pelotas" },
    { id: 9, nombre: "Accesorios" }
  ]
}
```

Para evitar mostrar una sección vacía o incompleta durante la demostración, decidimos crear la página de la tienda y, basándonos en la documentación de la API que nos facilitaron nuestros compañeros, desarrollamos unos mocks que simulaban los datos que, en teoría, devolvería la API si estuviera operativa. Esto nos permitió presentar una interfaz funcional y coherente, aunque los datos fueran simulados.

Además, con la expectativa de que eventualmente podríamos integrar la API real, implementamos un objeto en la vista llamado `API_CONFIG`. Este objeto incluía un atributo `useRealAPI`, inicialmente configurado en `false`, que determinaría si la aplicación debía conectarse a la URL de la API real (especificada en su atributo `url`) en caso de estar en `true`. De esta manera, si el equipo de la tienda online lograba poner en producción su aplicación, solo necesitaríamos modificar un par de líneas de código para activar la conexión con la API real, lo que facilitaría una transición rápida y eficiente.

```
// Configuración de API
const API_CONFIG = {
  // Cambiar a true para usar la API real
  useRealAPI: false,
  baseUrl: 'https://api-ejemplo.com'
}
```

7. Mejoras y ampliaciones

Aunque nuestro proyecto consideramos que está muy completo, siempre se puede mejorar cualquier cosa que creas, así que en este apartado nos encargamos de proponer nuevas funcionalidades y mejoras del producto creado durante esta segunda parte de la asignatura.

Reservas mejoradas para evitar solapamientos

En esta versión final de nuestra aplicación, las reservas han sido consideradas para poder solaparse. Esto quiere decir que un usuario, por ejemplo, puede añadir varias reservas para una misma actividad.

La funcionalidad ha sido implementada así pensando en un usuario que quiera efectuar una reserva para que sus acompañantes o amigos que entrenan con él puedan ir a la clase con él, en vez de tener que registrar cada uno su reserva.

Dado que esta funcionalidad es un poco extraña y al monitor en el listado de asistentes le aparecerá un usuario en varias ocasiones, consideramos que sería una buena opción mejorar este proceso para o bien limitar la reserva de clases a una por persona; o bien añadir la funcionalidad de “reserva múltiple” que permita seleccionar a más socios acompañantes y que sean notificados para aceptar.

Perfiles de usuario públicos

Actualmente, la única forma de ver un perfil es siendo usuario y accediendo a nuestro perfil. Pero, ¿y si un monitor quiere saber más información de un asistente a su clase? Por ello consideramos que sería de gran utilidad una vista que muestre los datos de un usuario públicamente, para que tanto monitor como demás usuarios pudieran acceder a los datos de un perfil como nombre, foto de perfil, etc.

Esto nos permitiría en un futuro crear una especie de red social de deporte en la que nuestros usuarios pudieran compartir utilizando sus perfiles públicos del gimnasio información o rutinas que les sean útiles.

CRUD completo de todas las entidades

Por falta de tiempo, no hemos implementado un CRUD (Create, Read, Update, Delete) de todas las entidades de nuestra aplicación. Por ejemplo, creemos que sería muy útil permitir al usuario eliminar una reserva si ya no puede ir, o actualizar los datos de una actividad.

Son pequeños detalles que, pese a su fácil implementación, dado el tiempo reducido con el que hemos contado, no nos ha sido posible implementar, y que serían de gran utilidad en la web.

8. Referencias

- **Documentación de Spring Boot:**
<https://docs.spring.io/spring-boot/documentation.html>
- **Documentación de Vue:**
<https://vuejs.org/guide/introduction.html>
- **Configuración de Hibernate JPA:**
<https://certidevs.com/tutorial-hibernate-jpa>
- **Documentación de PostgreSQL:**
<https://www.postgresql.org/docs/>
- **Documentación de JavaScript:**
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>
- **Temario de la asignatura**
<https://moodle2024-25.ua.es/moodle/course/view.php?id=1109§ion=11#tabs-tree-start>
- **Documentación de Docker:**
<https://docs.docker.com/>
- **[Enlace a nuestro repositorio GitHub](#)**