Pedro Vázquez Palomino

MEMORIA TÉCNICA

Proyecto: GymManager – Sistema de Gestión de Gimnasio

1. Introducción

1.1 Descripción del dominio del problema

He decidido realizarlo de un gimnasio deportivo donde se imparten clases concretas más allá de una libertad con las maquinarias, y muchas veces se necesitan gestionar información relacionada con sus socios, las clases que se imparten y las inscripciones a dichas clases.

En muchos casos, esta gestión se realiza manualmente o con herramientas poco automatizadas, lo que puede generar errores, duplicidades o pérdida de información.

El proyecto GymManager sirve para gestionar de manera estructurada y segura:

- Usuarios (administradores o socios).
- Clases disponibles en el gimnasio.
- Inscripciones de socios a clases.
- Control de acceso mediante login.

1.2 Objetivos de la aplicación

El objetivo principal es desarrollar una aplicación web que permite

- Gestionar clases del gimnasio y liminar inscripciones.
- Registrar socios e inscribir socios en clases.
- Restringir el acceso a usuarios autenticados.
- Utilizar base de datos SQLite integrada en el proyecto.
- Aplicar buenas prácticas con Doctrine y sistema de seguridad.

2. Análisis del sistema

2.1 Funcionalidades principales

La aplicación permite:

- Inicio de sesión de usuarios.
- Cierre de sesión.
- Listado de clases.
- Gestión de socios.
- Creación de inscripciones.
- Eliminación de inscripciones.
- Restricción de acceso a usuarios logueados.

2.2 Flujo de la aplicación

1. El usuario accede a la aplicación.
2. Si no está autenticado, puede irse solo añadirse como socio
3. Tras iniciar sesión correctamente:
    - Puede acceder al listado de clases.
    - Puede ver las inscripciones.
    - Puede crear nuevas inscripciones.
4. Puede cerrar sesión mediante la opción de logout.

3. Diseño

3.1 Entidades creadas

Usuario

Representa a la persona que puede acceder al sistema.

Campos principales:

- Id
- Email
- Password (encriptada)
- Roles

Socio

Representa a los miembros del gimnasio.

Campos principales:

- Nombre
- Apellido
- Email
- Edad
- Teléfono
- Fecha alta
- Tipo Membresía

Clase

Representa las actividades que ofrece el gimnasio.

Campos principales:

- Id
- Nombre
- Tipo
- Hora
- Entrenador

Inscripcion

Relaciona un Socio con una Clase.

Campos principales:

- Id
- Fecha
- Socio (ManyToOne)
- Clase (ManyToOne)

3.2 Justificación del modelo de datos

Se ha diseñado una relación:

- Un socio puede tener muchas inscripciones.
- Una clase puede tener muchas inscripciones.
- Una inscripción pertenece a un solo socio y a una sola clase.

Esto se implementa mediante relaciones ManyToOne en Doctrine.

El modelo permite:

- Evitar duplicación de datos.
- Mantener integridad referencial.
- Facilitar consultas eficientes.

3.3 Estructura general del proyecto

El proyecto sigue la arquitectura MVC de Symfony:

- src/Entity → Entidades del sistema.
- src/Controller → Controladores.
- src/Form → Formularios.
- templates/ → Vistas Twig.
- config/ → Configuración.
- var/ → Base de datos SQLite.

La base de datos utilizada es SQLite, almacenada como archivo dentro del proyecto, lo que permite portabilidad.

4. Implementación

4.1 Controladores principales

InscripcionController

Contiene:

- index() → Lista de inscripciones.
- new() → Crear nueva inscripción.
- delete() → Eliminar inscripción.

ClaseController

Permite:

- Listar clases.
- Gestionar información básica.

SecurityController

Gestiona:

- login()
- logout()

El logout es interceptado por el firewall de Symfony.

4.2 Uso de formularios y validaciones

Se han creado formularios utilizando:

AbstractType

Ejemplo: InscripcionType

Se utilizan:

- form_start()
- form_row()
- form_end()

Para relaciones (Socio, Clase) se emplea EntityType.

Se añadió el método __toString() en las entidades para evitar errores de conversión a string en Twig.

4.3 Acceso a base de datos mediante Doctrine

Se utiliza Doctrine ORM para:

- Persistir entidades.
- Consultar datos con Repository.
- Gestionar relaciones entre entidades.

Ejemplo de persistencia:

```
$em = $doctrine->getManager();
$em->persist($inscripcion);
$em->flush();
```

Se emplean migraciones y fixtures para crear estructura y datos iniciales.

Las contraseñas se codifican usando:

```
php bin/console security:hash-password
```

o mediante UserPasswordHasherInterface.

5. Conclusiones

5.1 Dificultades encontradas

Durante el desarrollo tuve varios problemas

- Configuración correcta del sistema de seguridad lo que me acusó varios problemas
- Codificación adecuada de contraseñas.
- Errores de rutas duplicadas y un bucle en el login.
- Configuración del logout.
- Gestión de relaciones entre entidades.
- Protección de rutas mediante roles.

5.2 Aprendizajes adquiridos

Gracias al desarrollo del proyecto se han adquirido conocimientos sobre:

- Arquitectura MVC en Symfony.
- Configuración del sistema de autenticación.
- Uso de Doctrine ORM.
- Creación de formularios dinámicos.
- Gestión de relaciones entre entidades.
- Uso de SQLite como base de datos portátil.
- Seguridad en aplicaciones web.
- Control de acceso por roles.

Además, he entendido la importancia de estructurar correctamente un proyecto profesional y aplicar buenas prácticas en desarrollo backend.