Proyecto Final
VioletBox

Horas de Libre Configuración

Álvaro Castillero

DAW 2

IES Alixar

14/02/2026

1. Introducción

Descripción del proyecto

VioletBox es una aplicación web desarrollada como proyecto final de 2º de DAW. Se trata de un laboratorio básico de hacking orientado a principiantes, donde los usuarios pueden aprender sobre vulnerabilidades web a través de desafíos prácticos y guías explicativas integradas en la propia plataforma.

La aplicación está pensada para ejecutarse en entorno local (localhost) y simula un entorno controlado de aprendizaje en el que el usuario puede experimentar con conceptos fundamentales de seguridad web.

Objetivos de la aplicación

El objetivo principal de VioletBox es que el usuario aprenda los conceptos básicos relacionados con vulnerabilidades web de forma práctica. A través de laboratorios y desafíos, el usuario puede:

- Comprender cómo funcionan determinadas vulnerabilidades.
- Resolver retos prácticos aplicando los conocimientos aprendidos.
- Seguir tutoriales explicativos asociados a cada desafío.
- Visualizar su progreso dentro de la plataforma.

2. Análisis del sistema

Funcionalidades principales

Usuario no autenticado

- Acceder a las páginas de Login y Register.
- Registrarse en la aplicación.
- Iniciar sesión.

Usuario autenticado

- Acceder a la página principal.
- Navegar por los laboratorios y desafíos.
- Visualizar las guías asociadas a cada laboratorio.
- Enviar soluciones a los desafíos.
- Ver cuáles laboratorios ha completado.
- Acceder a la página de contacto.

Administrador

El administrador es un usuario especial que se identifica cuando el nombre de usuario es “admin”. Este usuario tiene acceso a un CRUD completo de usuarios, lo que le permite:

- Listar todos los usuarios.
- Editar usuarios (excepto el nombre del admin).
- Eliminar usuarios (no se puede eliminar la cuenta admin).

Casos de uso

Flujo típico de un usuario:

1. El usuario accede a la aplicación.
2. Se registra mediante el formulario de registro.
3. Inicia sesión.
4. Desde el index pulsa en “Comenzar” o accede a la sección de desafíos.
5. Visualiza los laboratorios disponibles.
6. Accede a un laboratorio concreto.
7. Consulta el tutorial asociado.
8. Envía una solución.
9. Si la respuesta es correcta, el laboratorio queda marcado como completado.

3. Diseño

Entidades y campos principales

El modelo de datos está compuesto por las siguientes entidades:

User

- id
- username (único)
- email (único)
- password
- birthDate
- roles

Representa a los usuarios registrados en la aplicación.

Module

- id
- name
- description

Agrupa los laboratorios por temática o categoría.

Laboratory

- id
- title
- description
- difficulty
- solution
- module (relación ManyToOne con Module)

Representa cada desafío práctico disponible en la plataforma.

Progress

- id
- user (ManyToOne con User)
- laboratory (ManyToOne con Laboratory)
- completed (boolean)
- createdAt

Permite almacenar qué laboratorios ha completado cada usuario.

Achievement

- id
- name
- description

Define los logros disponibles en el sistema.

UserAchievement

- user (ManyToOne)
- achievement (ManyToOne)
- earnedAt

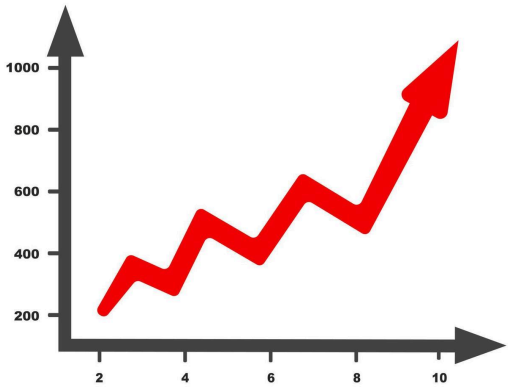
Entidad intermedia que implementa la relación N:M entre usuarios y logros.

Justificación del modelo de datos

Se diseñó el modelo con al menos seis entidades para estructurar correctamente la aplicación y separar responsabilidades.

- La relación entre Laboratory y Module permite organizar los desafíos por categorías.
- La entidad Progress permite registrar el avance del usuario sin modificar directamente la entidad Laboratory.
- La relación N:M entre User y Achievement mediante UserAchievement permite asignar el progreso de cada usuario.
- Se separó claramente la lógica de usuarios, progreso y desafíos para mantener una estructura escalable y mantenible.

Este modelo facilita la ampliación futura de la aplicación, permitiendo añadir más módulos, logros o tipos de seguimiento sin modificar la estructura principal.



Estructura general del proyecto

El proyecto sigue el patrón MVC de Symfony y se organiza en las siguientes carpetas principales:

- Controller: contiene los controladores que gestionan las rutas y la lógica de negocio.
- Entity: define las entidades del sistema y su mapeo con la base de datos.
- Repository: gestiona las consultas a la base de datos.
- Form: define los formularios y sus validaciones.
- Templates: vistas desarrolladas con Twig.
- DataFixtures: carga de datos iniciales obligatorios, como laboratorios.
- Migrations: control de versiones de la base de datos.

4. Implementación

Controladores más relevantes

ChallengesController

Se encarga de mostrar el listado de laboratorios disponibles. Además, consulta el progreso del usuario autenticado para determinar qué laboratorios han sido completados y marcarlos visualmente en la vista.

Utiliza:

- LaboratoryRepository para obtener todos los laboratorios.
- ProgressRepository para consultar el progreso del usuario.

LaboratoryController

Gestiona la visualización individual de cada laboratorio y el envío de soluciones.

En el método submit:

- Se compara la respuesta enviada por el usuario con la solución almacenada en la base de datos.
- Se crea o actualiza una entidad Progress.
- Se persiste la información mediante EntityManager (persist y flush).

Este controlador es clave porque implementa la lógica principal del sistema de desafíos.

Uso de formularios y validaciones

Se utilizan formularios de Symfony para el registro y la administración.

En el registro:

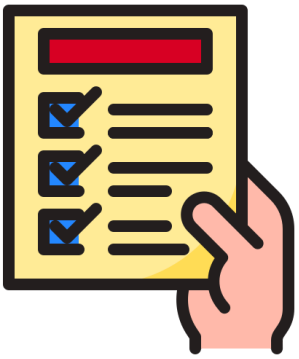
- Se validan campos obligatorios.
- Se evita la duplicidad de username y email.
- Se comprueba la coincidencia de contraseñas.

En el login:

- Se validan credenciales incorrectas mediante el sistema de seguridad de Symfony.

En el panel de administración:

- No se permite modificar el nombre del usuario administrador.
- No se permite eliminar la cuenta admin.



Acceso a base de datos con Doctrine

El acceso a la base de datos se realiza mediante Doctrine ORM.

Se utilizan:

- Repositorios para consultas (find, findAll, findOneBy, findBy).
- Relaciones ManyToOne para vincular entidades como Progress con User y Laboratory.
- Una relación N:M implementada mediante la entidad UserAchievement.

Además:

- Se utilizan Doctrine Migrations para generar y versionar las tablas.
- Se emplean Doctrine Fixtures para cargar datos iniciales, como los laboratorios, garantizando datos persistentes tras reinicios de la base de datos.

5. Conclusiones

Dificultades encontradas

Durante el desarrollo surgieron varios problemas técnicos:

- Errores con migraciones duplicadas.
- Problemas con la persistencia de datos que se resolvieron utilizando fixtures.
- Conflictos con dependencias de PHP.
- Problemas con el entorno de desarrollo.
- Corrupción del repositorio Git.

Estas dificultades obligaron a comprender mejor la gestión del entorno y la estructura interna de Symfony.

Aprendizajes adquiridos

Este proyecto permitió consolidar conocimientos sobre:

- Arquitectura Modelo Vista Controlador en Symfony.
- Uso de Doctrine ORM.
- Gestión de migraciones y fixtures.
- Control de roles y autenticación.
- Organización estructurada del código.
- Buenas prácticas en el diseño de bases de datos.

Además, se adquirió mayor seguridad en el desarrollo de aplicaciones web completas utilizando Symfony.