Documentación Técnica — API RESTful para Gestión de Restaurantes

Realizado por: Daniel Bracamonte

Índice

Documentación Técnica

1. Introducción

Este proyecto consiste en una API RESTful desarrollada en PHP con Symfony 6 y el componente API Platform, diseñada para gestionar una base de datos de restaurantes. Incluye un sistema de autenticación de usuarios (registro e inicio de sesión),que permite una personalizada para los usuarios. La API está documentada automáticamente con OpenAPI/Swagger, facilitando su uso y mantenimiento.

2. Tecnologías Utilizadas

- **Backend:** Symfony 6 (PHP 8+), Doctrine ORM para persistencia en base de datos.
- **Base de datos:** MySQL, gestionada mediante Doctrine.
- **Frontend:** HTML5, CSS, JavaScript (Fetch API).
- **Documentación API:** API Platform (OpenAPI/Swagger).
- **Servidor Web:** Apache/Nginx (según entorno).

3. Funcionalidades

El proyecto ofrece las siguientes funcionalidades principales:

1. **Gestión de Restaurantes:**
    - Crear, leer, actualizar y eliminar (CRUD) registros de restaurantes, incluyendo información como nombre, dirección y teléfono.
    - Exportar la lista de restaurantes a un archivo PDF para referencia o impresión.
2. **Autenticación de Usuarios:**
    - Registro de nuevos usuarios con correo electrónico y contraseña, utilizando encriptación segura (JWT para autenticación).

    - Inicio de sesión seguro para acceder a las funcionalidades protegidas de la API.
3. **Interfaz de Usuario:**
    - Interfaz web sencilla para listar, añadir, editar y eliminar restaurantes mediante una tabla interactiva y formularios modales.
    - Formularios de registro e inicio de sesión para la autenticación de usuarios, con manejo de errores en el cliente.
4. **API RESTful:**
    - Endpoints para gestionar restaurantes (GET, POST, PUT, DELETE) con respuestas en formato JSON.
    - Endpoints para registro e inicio de sesión de usuarios, integrados con autenticación JWT.
5. **Validación y Manejo de Errores:**
    - Validación de datos en el backend para asegurar la integridad de los campos obligatorios y sus formatos.
    - Respuestas con códigos HTTP apropiados (200, 201, 400, 404, etc.) y mensajes de error claros.
    - Manejo de errores en el frontend con alertas para el usuario.
6. **Documentación Automática:**
    - Generación automática de documentación OpenAPI/Swagger mediante API Platform, accesible en /api/docs, para facilitar la integración y prueba de los endpoints.

4. Modelo de Datos

4.1. Entidad Restaurante

Campo	Tipo	Restricciones	Descripción
id	int	PK, auto-increment	Identificador único
nombre	varchar	longitud 100,not null	Nombre del sitio
dirección	varchar	longitud 255,not null	Dirección física
telefono	varchar	longitud 20, not null	Contacto

Esta entidad se mapea mediante Doctrine ORM en Symfony.

4.2. Entidad Usuario

Campo	Tipo	Restricciones	Descripción
id	int	PK, auto-increment	Identificador único
email	varchar	longitud 180,not null	Correo del user
password	varchar	longitud 255,not null	Contraseña
roles	json	not null	Rol del user

5. API REST

5.1. Endpoints Disponibles

Método	Ruta	Funcionalidad	Descripción	Código de respuesta
GET	/api/restaurantes	Listar restaurantes	Retorna lista en json	200 Ok
GET	/api/restaurantes/{id}	Obtener un restaurante	Retorna datos del restaurante	200 Ok /404 no encontrado
POST	/api/restaurantes	Crear un restaurante	Crea nuevo restaurante	201 Creado/400 Datos invalidos
PUT	/api/restaurantes/{id}	Actualizar restaurante	Actualizar campos	200 Ok /404 no encontrado
DELETE	/api/restaurantes/{id}	Borrar restaurante	Borrar registro de id	200 Ok/ 404 no encontrado

5.2. Endpoints Disponibles

POST /api/restaurantes
Content-Type: application/json

```
{
  "nombre": "Restaurante El Buen Sabor",
  "dirección": "Calle Falsa 123",
  "teléfono": "123456789"
}
```

5.3. Respuesta exitosa (201)

Cuando se crea un restaurante correctamente mediante el endpoint POST /api/restaurantes, el servidor responde con un código HTTP 201 Created indicando que el recurso fue creado satisfactoriamente.

La respuesta tiene el siguiente formato JSON:


```
{
  "mensaje": "Restaurante creado",
  "id": 5
}
```


5.4. Manejo de errores

La API utiliza códigos de estado HTTP adecuados para indicar errores en las solicitudes. A continuación, se describen los principales errores manejados:

- **400 Bad Request**
  Indica que la solicitud es incorrecta debido a que faltan datos obligatorios, los datos enviados no cumplen con el formato esperado, o existen validaciones fallidas.
  Ejemplo de respuesta JSON:


```
{

  "error": "Faltan datos"

}
```


- **404 Not Found**

  Se devuelve cuando se intenta acceder, actualizar o eliminar un restaurante que no existe en la base de datos.Ejemplo de respuesta JSON:


```
{

 "error": "Restaurante no encontrado."

}
```


- **Otros errores**

En caso de errores inesperados o internos del servidor, se puede devolver un código 500 Internal Server Error con un mensaje genérico para evitar exponer detalles sensibles.

6. Validación y Manejo de Errores

- Se validan campos obligatorios (nombre, dirección, teléfono) para POST y PUT.
- Se utilizan códigos HTTP adecuados (400, 404, 201, 200).
- Mensajes claros y consistentes en respuesta JSON.
- En el frontend se manejan alertas para informar errores al usuario.

7. Frontend

- HTML simple con tabla para listar restaurantes.
- Botones para añadir, editar y borrar restaurantes.
- Formularios modales para entrada de datos.
- Consumo de API con fetch.
- Adaptación para parsear respuesta JSON según formato API Platform.
- Manejo básico de errores y validaciones en cliente**.**

8. Documentación Automática

Se utiliza API Platform para:

- Generar documentación OpenAPI/Swagger.
- Permitir probar endpoints desde interfaz web (/api/docs).
- Facilitar integración con terceros y mantenimiento.

9.Conclusión

Este proyecto proporciona una solución robusta y escalable para la gestión de restaurantes mediante una API RESTful desarrollada con Symfony 6.Una interfaz de usuario intuitiva y la documentación automática con OpenAPI/Swagger garantizan una experiencia accesible tanto para usuarios finales como para desarrolladores. La aplicación es ideal para entornos

que requieran administrar información de restaurantes de manera eficiente, con un enfoque en la usabilidad y la facilidad de mantenimiento