



Documentación Técnica: Restaurant API



Tabla de Contenidos

1. [Guía de Inicio Rápido](#)
 2. [Introducción y Arquitectura](#)
 3. [Configuración del Entorno](#)
 4. [Instalación Paso a Paso](#)
 5. [Estructura del Proyecto](#)
 6. [Implementación Detallada](#)
 7. [Dockerización](#)
 8. [Comandos de Mantenimiento](#)
 9. [Funcionalidades Avanzadas](#)
 10. [Patrones y Mejores Prácticas](#)
-



Guía de Inicio Rápido

⚡ Setup en 5 Minutos

Si acabas de descargar o clonar este proyecto, sigue estos pasos para tenerlo funcionando:

Paso 1: Verificar Prerrequisitos

```
# Verificar que Docker está instalado y funcionando
docker --version
# Debería mostrar: Docker version 20.10+

docker-compose --version
# Debería mostrar: docker-compose version 1.29+

# Verificar que Git está instalado (si clonaste el repo)
git --version
```

🔴 **Si no tienes Docker:** Instala Docker Desktop desde docker.com

Paso 2: Obtener el Proyecto

Opción A: Clonando desde GitHub

```
git clone <URL_DEL_REPOSITORIO>
cd backend
```

Opción B: Desde archivo ZIP

```
# Descomprimir el archivo ZIP
unzip restaurant-api.zip
cd restaurant-api/backend
```

Paso 3: Configurar Variables de Entorno

El proyecto **NO incluye** el archivo `.env` por seguridad. Necesitas crearlo:

```
# Crear archivo .env basado en .env.example (si existe)
cp .env.example .env

# O crear .env desde cero
touch .env
```

Contenido del archivo `.env`:


```
# =====
# CONFIGURACIÓN DE APLICACIÓN
# =====
APP_ENV=dev
# ⚠ IMPORTANTE: Generar un APP_SECRET único de 32 caracteres
APP_SECRET=tu_app_secret_de_32_caracteres_aqui

# =====
# BASE DE DATOS
# =====
DATABASE_URL="mysql://root:rootpassword@database:3306/restaurant_api?serverVersion=8.0.32"

# Variables para Docker Compose
MYSQL_ROOT_PASSWORD=rootpassword
MYSQL_DATABASE=restaurant_api

# =====
# CONFIGURACIÓN CORS (para desarrollo local)
# =====
CORS_ALLOW_ORIGIN="^https://(localhost|127\.\0\.\0\.\1)(:[0-9]+)?$"
NELMIO_CORS_ALLOW_ORIGIN="^https://(localhost|127\.\0\.\0\.\1)(:[0-9]+)?$"

# =====
# PUERTOS (opcional - cambiar si están ocupados)
# =====
NGINX_PORT=8080
MYSQL_PORT=3307
```

 **Generar APP_SECRET:** Ejecuta `openssl rand -hex 16` o usa [este generador](#)

Paso 4: Generar APP_SECRET

El **APP_SECRET** es **crítico** para la seguridad. Genera uno único:

```
# Opción 1: Con OpenSSL (recomendado)
openssl rand -hex 16

# Opción 2: Con herramientas online
# Visita: https://generate-secret.vercel.app/32

# Opción 3: Desde PHP
php -r "echo bin2hex(random_bytes(16)) . PHP_EOL;"

# Opción 4: Comando manual
echo $(LC_ALL=C tr -dc 'A-Za-z0-9' < /dev/urandom | head -c 32)
```

Ejemplo de APP_SECRET válido:

```
APP_SECRET=a1b2c3d4e5f6789012345678901abcde
```

Paso 5: Ejecutar Setup Automático

```
# Construir y levantar contenedores
docker-compose up -d --build

# Esperar a que MySQL esté listo (30-60 segundos)
echo "⌚ Esperando que MySQL esté listo..."
sleep 60

# Instalar dependencias de Composer
docker exec restaurant_api_php composer install

# Crear base de datos
docker exec restaurant_api_php php bin/console doctrine:database:create --if-not-exists

# Ejecutar migraciones para crear tablas
docker exec restaurant_api_php php bin/console doctrine:migrations:migrate -n

# Limpiar cache
docker exec restaurant_api_php php bin/console cache:clear
```

Paso 6: Verificar que Todo Funciona

```
# Verificar que los contenedores están corriendo
docker-compose ps

# Debería mostrar:
# restaurant_api_php      Up
# restaurant_api_nginx    Up      0.0.0.0:8080->80/tcp
# restaurant_api_mysql    Up      0.0.0.0:3307->3306/tcp

# Probar la API
curl http://localhost:8080/api

# Debería devolver información de la API Platform
```

Paso 7: Acceder a la Aplicación

- ☒ **API Base:** http://localhost:8080/api
 - ☒ **Documentación Swagger:** http://localhost:8080/api/docs
 - ☒ **API Platform Admin:** http://localhost:8080/api
-

Solución de Problemas Comunes

Puerto 8080 ocupado

```
# Ver qué proceso usa el puerto
netstat -ano | findstr :8080 # Windows
lsof -i :8080                # Linux/Mac

# Cambiar puerto en docker-compose.yml
ports:
  - "8081:80" # Usar 8081 en lugar de 8080
```

Error "APP_SECRET not set"

```
# Verificar que .env tiene APP_SECRET
cat .env | grep APP_SECRET

# Si no existe, agregarlo
echo "APP_SECRET=$(openssl rand -hex 16)" >> .env
```

MySQL no conecta

```
# Ver logs de MySQL
docker-compose logs database
```

```
# Reiniciar MySQL
docker-compose restart database

# Verificar que la red funciona
docker exec restaurant_api_php ping database
```

Composer install falla

```
# Si hay problemas de permisos
docker exec restaurant_api_php chown -R www-data:www-data /var/www/html

# Si hay problemas de memoria
docker exec restaurant_api_php composer install --no-dev --optimize-autoloader
```

Cache no se limpia

```
# Limpiar cache manualmente
docker exec restaurant_api_php rm -rf var/cache/*
docker exec restaurant_api_php php bin/console cache:clear --no-warmup
```

Primeros Pasos Después del Setup

1. Crear Usuario de Prueba

```
# Usando la API (recomendado)
curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "email": "admin@test.com",
    "name": "Administrador Test"
  }'

# Guardar la API_KEY que devuelve la respuesta
```

2. Probar Autenticación

```
# Usar la API_KEY obtenida en el paso anterior
curl -H "X-API-KEY: tu_api_key_aqui" \
  http://localhost:8080/api/restaurants
```

3. Crear Primer Restaurante

```
curl -X POST http://localhost:8080/api/restaurants \
-H "Content-Type: application/json" \
-H "X-API-KEY: tu_api_key_aqui" \
-d '{
  "name": "Mi Primer Restaurante",
  "address": "Calle Principal 123",
  "phone": "123456789"
}'
```

4. Explorar Swagger UI

Visita <http://localhost:8080/api/docs> para:

- Ver todos los endpoints disponibles
- Probar la API directamente desde el navegador
- Ver ejemplos de request/response
- Configurar autenticación con el botón "Authorize"

Checklist de Verificación

Marca cada ítem cuando lo completes:

- ☐ Docker y Docker Compose instalados
- ☐ Proyecto descargado/clonado
- ☐ Archivo `.env` creado con `APP_SECRET`
- ☐ Contenedores construidos y corriendo (`docker-compose ps`)
- ☐ Dependencias instaladas (`composer install`)
- ☐ Base de datos creada
- ☐ Migraciones ejecutadas
- ☐ Cache limpiado
- ☐ API responde en `http://localhost:8080/api`
- ☐ Swagger UI accesible en `http://localhost:8080/api/docs`
- ☐ Usuario de prueba creado
- ☐ Autenticación funcionando
- ☐ Primer restaurante creado

Script de Setup Automatizado

Si prefieres un setup completamente automatizado, crea este script:

```
#!/bin/bash
# setup.sh - Script de configuración automática

echo "🚀 Configurando Restaurant API..."
```

```
# Verificar prerequisites
if ! command -v docker &> /dev/null; then
    echo "❌ Docker no está instalado"
    echo "    Instala Docker Desktop desde: https://www.docker.com/products/docker-desktop"
    exit 1
fi

if ! command -v docker-compose &> /dev/null; then
    echo "❌ Docker Compose no está instalado"
    exit 1
fi

# Crear .env si no existe
if [ ! -f .env ]; then
    echo "📄 Creando archivo .env..."
    cat > .env << EOL
APP_ENV=dev
APP_SECRET=$(openssl rand -hex 16)
DATABASE_URL="mysql://root:rootpassword@database:3306/restaurant_api?serverVersion=8.0.32"
MYSQL_ROOT_PASSWORD=rootpassword
MYSQL_DATABASE=restaurant_api
CORS_ALLOW_ORIGIN="^https?://(localhost|127\.\0\.\0\1)(:[0-9]+)?$"
NELMIO_CORS_ALLOW_ORIGIN="^https?://(localhost|127\.\0\.\0\1)(:[0-9]+)?$"
EOL
    echo "✅ Archivo .env creado con APP_SECRET único"
fi

# Construir y levantar contenedores
echo "🏗️ Construyendo contenedores..."
docker-compose build --no-cache

echo "🚀 Iniciando servicios..."
docker-compose up -d

# Esperar MySQL
echo "⌚ Esperando que MySQL esté listo..."
until docker exec restaurant_api_mysql mysqladmin ping -h localhost --silent; do
    sleep 3
    echo "    Esperando MySQL..."
done

# Instalar dependencias
echo "📦 Instalando dependencias..."
docker exec restaurant_api_php composer install

# Setup de base de datos
echo "🗄️ Configurando base de datos..."
docker exec restaurant_api_php php bin/console doctrine:database:create --if-not-exists
docker exec restaurant_api_php php bin/console doctrine:migrations:migrate -n
```

```
# Limpiar cache
echo "🧹 Limpiando cache..."
docker exec restaurant_api_php php bin/console cache:clear

# Crear usuario de prueba
echo "👤 Creando usuario de prueba..."
RESPONSE=$(curl -s -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "email": "admin@test.com",
    "name": "Administrador Test"
  }')

API_KEY=$(echo $RESPONSE | grep -o '"api_key": "[^"]*"' | cut -d'"' -f4)

echo ""
echo "✅ ¡Setup completado exitosamente!"
echo ""
echo "📌 URLs importantes:"
echo "  🌐 API Base: http://localhost:8080/api"
echo "  📖 Documentación: http://localhost:8080/api/docs"
echo ""
echo "🔑 Usuario de prueba creado:"
echo "  📧 Email: admin@test.com"
echo "  🔑 API Key: $API_KEY"
echo ""
echo "🔧 Comando de prueba:"
echo "  curl -H \"X-API-KEY: $API_KEY\" http://localhost:8080/api/restaurants"
echo ""
echo "🌱 ¡Listo para desarrollar!"
```

Para usar el script:

```
# Hacer ejecutable
chmod +x setup.sh

# Ejecutar
./setup.sh
```

🏗️ Introducción y Arquitectura

Visión General del Proyecto

Esta API RESTful para gestión de restaurantes fue desarrollada utilizando **Symfony 7.3** con **API Platform**, implementando una arquitectura robusta que sigue los principios **SOLID** y patrones de diseño modernos.

Decisiones Arquitectónicas

Stack Tecnológico Elegido

Tecnologías:

Framework: **Symfony 7.3**

API Framework: **API Platform 4.x**

ORM: **Doctrine 3.x**

Base de Datos: **MySQL 8.0**

Contenedorización: **Docker + Docker Compose**

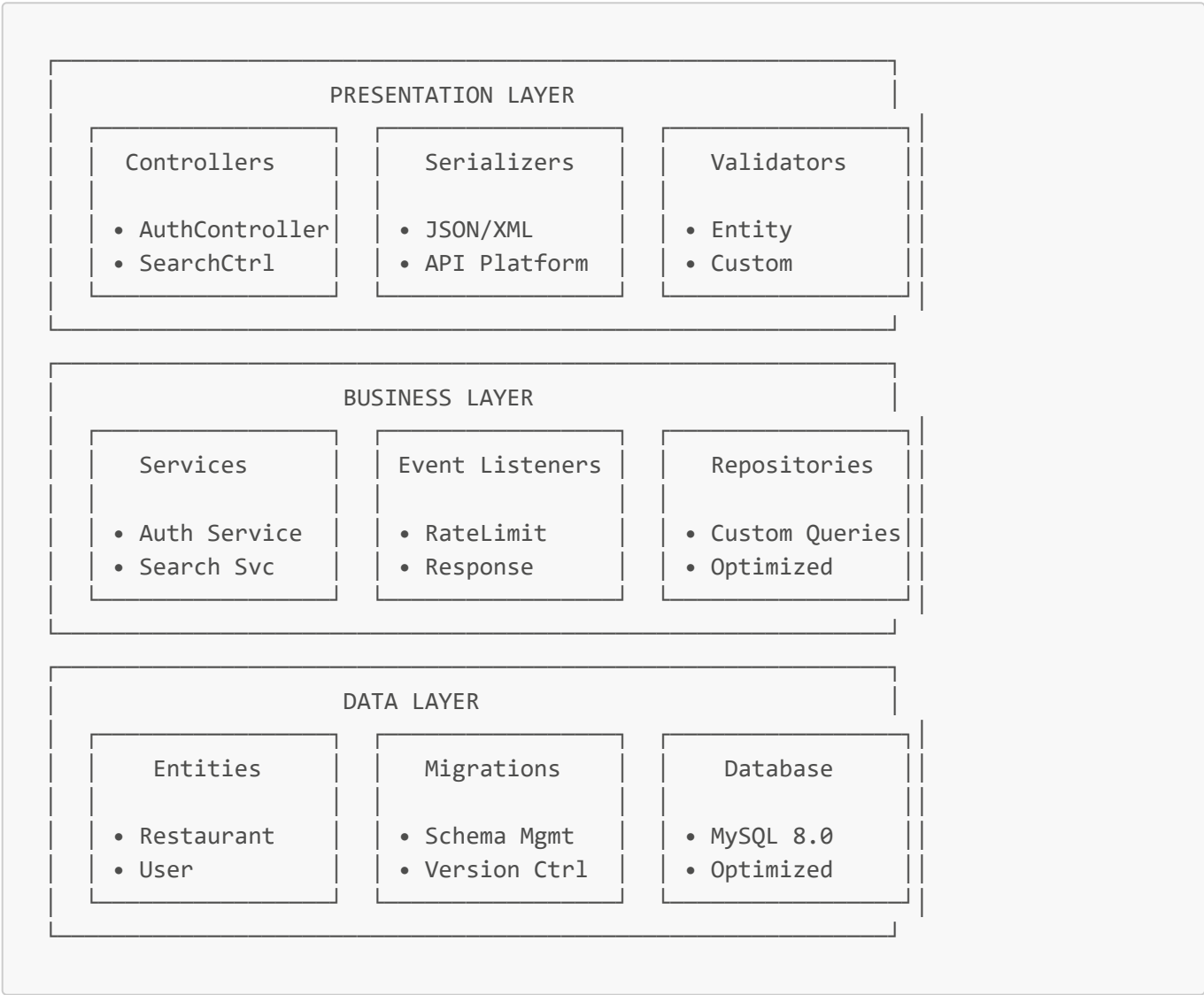
Servidor Web: **Nginx + PHP-FPM**

Documentación: **NelmioApiDocBundle (OpenAPI 3.0)**

Autenticación: **API Keys + Cookies HttpOnly**

Rate Limiting: **Symfony RateLimiter**

Arquitectura de Capas



Principios de Diseño Aplicados

SOLID Principles

- **Single Responsibility:** Cada clase tiene una responsabilidad única
- **Open/Closed:** Extensible pero cerrado a modificaciones
- **Liskov Substitution:** Interfaces bien definidas

- **Interface Segregation:** Interfaces específicas
- **Dependency Inversion:** Inyección de dependencias

Patrones Implementados

- **Repository Pattern:** Abstracción de acceso a datos con queries optimizadas
- **Observer Pattern:** Event listeners para rate limiting y headers de respuesta
- **Authenticator Pattern:** Autenticación API Key personalizada
- **MVC Pattern:** Controllers, Entities, Repositories separados

Configuración del Entorno

Prerrequisitos

```
# Verificar versiones
docker --version          # >= 20.10
docker-compose --version  # >= 1.29
git --version              # >= 2.25
```

Creación del Proyecto Base

```
# Paso 1: Crear proyecto Symfony
symfony new restaurant-api --version=7.3

# Paso 2: Navegar al directorio
cd restaurant-api

# Paso 3: Inicializar Git
git init
git add .
git commit -m "feat: initial Symfony 7.3 project setup"
```

Variables de Entorno

Archivo **.env**

```
# =====
# CONFIGURACIÓN DE APLICACIÓN
# =====
APP_ENV=dev
APP_SECRET=e8b5f7c2d9a4e6f1b8c3d7e9f2a5b8c1d4e7f0a3b6c9e2f5a8b1d4e7f0a3b6c9

# =====
# BASE DE DATOS
# =====
```

```
# Formato: mysql://user:password@host:port/database?serverVersion=version
DATABASE_URL="mysql://root:rootpassword@database:3306/restaurant_api?
serverVersion=8.0.32"

# Variables para Docker Compose
MYSQL_ROOT_PASSWORD=rootpassword
MYSQL_DATABASE=restaurant_api

# =====
# CONFIGURACIÓN CORS
# =====
CORS_ALLOW_ORIGIN="^https?:/(localhost|127\.0\.0\.1)(:[0-9]+)?$"

# =====
# API PLATFORM
# =====
API_PLATFORM_TITLE="Restaurant API"
API_PLATFORM_DESCRIPTION="API RESTful para gestión de restaurantes"
API_PLATFORM_VERSION="1.0.0"

# =====
# NELMIO API DOC
# =====
NELMIO_CORS_ALLOW_ORIGIN="^https?:/(localhost|127\.0\.0\.1)(:[0-9]+)?$"
```

Instalación Paso a Paso

Fase 1: Dependencias Core

API Platform y Doctrine

```
# Instalar API Platform (incluye Symfony Serializer)
docker exec restaurant_api_php composer require api-platform/core

# Instalar Doctrine ORM pack
docker exec restaurant_api_php composer require symfony/orm-pack

# Instalar migraciones de Doctrine
docker exec restaurant_api_php composer require doctrine/doctrine-migrations-
bundle

# Verificar instalación
docker exec restaurant_api_php php bin/console debug:container doctrine
```

Sistema de Seguridad

```
# Instalar Security Bundle
docker exec restaurant_api_php composer require symfony/security-bundle

# Instalar Validator para validaciones
docker exec restaurant_api_php composer require symfony/validator

# Instalar Serializer (si no está incluido)
docker exec restaurant_api_php composer require symfony/serializer

# Generar configuración inicial de seguridad
docker exec restaurant_api_php php bin/console make:user
```

Fase 2: Documentación y API

NelmioApiDocBundle

```
# Instalar Nelmio API Doc Bundle
docker exec restaurant_api_php composer require nelmio/api-doc-bundle

# Instalar CORS Bundle
docker exec restaurant_api_php composer require nelmio/cors-bundle

# Verificar bundles registrados
docker exec restaurant_api_php php bin/console debug:config
```

Templates y Assets (Para Swagger UI)

```
# Instalar Twig Bundle (necesario para Swagger UI)
docker exec restaurant_api_php composer require symfony/twig-bundle

# Instalar Asset Component
docker exec restaurant_api_php composer require symfony/asset

# Limpiar cache después de instalación
docker exec restaurant_api_php php bin/console cache:clear
```

Fase 3: Funcionalidades Avanzadas

Rate Limiting

```
# Instalar Rate Limiter Component
docker exec restaurant_api_php composer require symfony/rate-limiter

# Verificar servicios de rate limiting
docker exec restaurant_api_php php bin/console debug:container rate_limiter
```

Herramientas de Desarrollo

```
# Web Profiler (solo desarrollo)
docker exec restaurant_api_php composer require --dev symfony/web-profiler-bundle

# Maker Bundle para generación de código
docker exec restaurant_api_php composer require --dev symfony/maker-bundle

# Debug Bundle (incluido en web-profiler)
docker exec restaurant_api_php composer require --dev symfony/debug-bundle
```

Comandos de Verificación Post-Instalación

```
# Verificar todos los bundles registrados
docker exec restaurant_api_php php bin/console debug:container --show-deprecated

# Verificar rutas disponibles
docker exec restaurant_api_php php bin/console debug:router

# Verificar configuración de servicios
docker exec restaurant_api_php php bin/console debug:container --parameters

# Verificar configuración de Doctrine
docker exec restaurant_api_php php bin/console doctrine:mapping:info
```

📁 Estructura del Proyecto

Árbol de Directorios Implementado

```
restaurant-api/
├── config/                                # Configuraciones del framework
│   ├── bundles.php                       # Registro de bundles
│   ├── packages/                         # Configuración por bundle
│   │   ├── api_platform.yaml            # Config API Platform
│   │   ├── doctrine.yaml                # Config ORM
│   │   ├── framework.yaml               # Config core Symfony
│   │   ├── nelmio_api_doc.yaml           # Config documentación
│   │   ├── nelmio_cors.yaml              # Config CORS
│   │   ├── rate_limiter.yaml             # Config rate limiting
│   │   ├── routing.yaml                  # Config rutas
│   │   ├── security.yaml                 # Config seguridad
│   │   └── twig.yaml                     # Config templates
│   └── routes/                           # Definición de rutas
│       └── framework.yaml                # Rutas del framework
```

```

├── services.yaml      # Container de servicios
├── routes.yaml        # Rutas principales
├── docker/           # Configuración Docker
│   └── nginx/
│       └── default.conf # Config Nginx
├── migrations/       # Migraciones de base de datos
│   └── VersionXXXXXXXXX.php # Archivos de migración
├── public/           # Punto de entrada web
│   └── index.php      # Front controller
├── src/              # Código fuente de la aplicación
│   ├── Controller/   # Controladores HTTP
│   │   ├── AuthController.php # Autenticación y login
│   │   ├── RestaurantSearchController.php # Búsqueda avanzada
│   │   └── ErrorController.php # Manejo de errores
│   ├── Entity/       # Entidades Doctrine
│   │   ├── Restaurant.php # Entidad Restaurant
│   │   └── User.php      # Entidad User
│   ├── EventListener/ # Event Listeners
│   │   ├── RateLimitListener.php # Rate limiting
│   │   └── RateLimitResponseListener.php # Headers de rate limit
│   ├── Repository/   # Repositorios custom
│   │   ├── RestaurantRepository.php # Queries Restaurant
│   │   └── UserRepository.php      # Queries User
│   ├── Security/     # Componentes de seguridad
│   │   ├── ApiKeyAuthenticator.php # Autenticador API Key
│   │   └── UserChecker.php        # Verificador de usuarios
│   └── Kernel.php     # Kernel de la aplicación
├── templates/        # Templates Twig
│   └── bundles/       # Override de templates
├── var/              # Archivos generados
│   ├── cache/         # Cache de la aplicación
│   ├── log/           # Logs
│   └── sessions/      # Sesiones
├── vendor/           # Dependencias de Composer
├── .env              # Variables de entorno
├── .env.local        # Override local de variables
├── .gitignore        # Archivos ignorados por Git
├── composer.json     # Dependencias y configuración
├── composer.lock     # Lock de versiones
├── docker-compose.yml # Definición de servicios Docker
├── Dockerfile        # Imagen Docker personalizada
└── symfony.lock      # Lock de recetas Symfony

```

Descripción de Componentes Implementados

src/Entity/

Las entidades implementadas representan el modelo de datos:

```

<?php
// src/Entity/Restaurant.php - Ejemplo de implementación real

```

```
namespace App\Entity;

use ApiPlatform\Metadata\ApiResource;
use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\GetCollection;
use ApiPlatform\Metadata\Post;
use ApiPlatform\Metadata\Put;
use ApiPlatform\Metadata\Patch;
use ApiPlatform\Metadata\Delete;
use App\Repository\RestaurantRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: RestaurantRepository::class)]
#[ORM\HasLifecycleCallbacks]
#[ApiResource(
    operations: [
        new GetCollection(),
        new Get(),
        new Post(security: "is_granted('ROLE_USER')"),
        new Put(security: "is_granted('ROLE_USER')"),
        new Patch(security: "is_granted('ROLE_USER')"),
        new Delete(security: "is_granted('ROLE_USER')")
    ]
)]
class Restaurant
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 255)]
    #[Assert\NotBlank(message: 'El nombre del restaurante es obligatorio')]
    #[Assert\Length(
        min: 2,
        max: 255,
        minMessage: 'El nombre debe tener al menos {{ limit }} caracteres',
        maxMessage: 'El nombre no puede exceder {{ limit }} caracteres'
    )]
    private ?string $name = null;

    #[ORM\Column(length: 500)]
    #[Assert\NotBlank(message: 'La dirección es obligatoria')]
    #[Assert\Length(max: 500)]
    private ?string $address = null;

    #[ORM\Column(length: 20, nullable: true)]
    #[Assert\Regex(
        pattern: '/^[\\d\\s\\-\\+\\(\\)]+$/ ',
        message: 'El teléfono solo puede contener números, espacios, guiones y paréntesis'
    )]
```

```
private ?string $phone = null;

#[ORM\Column]
private ?\DateTimeImmutable $createdAt = null;

#[ORM\Column]
private ?\DateTimeImmutable $updatedAt = null;

public function __construct()
{
    $this->createdAt = new \DateTimeImmutable();
    $this->updatedAt = new \DateTimeImmutable();
}

#[ORM\PreUpdate]
public function setUpdatedAtValue(): void
{
    $this->updatedAt = new \DateTimeImmutable();
}

// Getters y Setters implementados...
}
```

Implementación Detallada

Sistema de Autenticación Implementado

Configuración de Seguridad

```
# config/packages/security.yaml - Configuración real del proyecto
security:
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
            'auto'

    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

    auth:
        pattern: ^/api/auth
        security: false
```



```

api:
  pattern: ^/api
  stateless: true
  custom_authenticators:
    - App\Security\ApiKeyAuthenticator
  user_checker: App\Security\UserChecker

access_control:
  - { path: ^/api/auth, roles: PUBLIC_ACCESS }
  - { path: ^/api/docs, roles: PUBLIC_ACCESS }
  - { path: ^/api, roles: ROLE_USER }

role_hierarchy:
  ROLE_ADMIN: ROLE_USER

```

Autenticador Implementado

```

<?php
// src/Security/ApiKeyAuthenticator.php - Código real implementado

namespace App\Security;

use App\Entity\User;
use App\Repository\UserRepository;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;
use Symfony\Component\Security\Core\Exception\AuthenticationException;
use
Symfony\Component\Security\Core\Exception\CustomUserMessageAuthenticationException
;
use Symfony\Component\Security\Http\Authenticator\AbstractAuthenticator;
use Symfony\Component\Security\Http\Authenticator\Passport\Badge\UserBadge;
use Symfony\Component\Security\Http\Authenticator\Passport\Passport;
use Symfony\Component\Security\Http\Authenticator\Passport\SelfValidatingPassport;

class ApiKeyAuthenticator extends AbstractAuthenticator
{
    public function __construct(
        private UserRepository $userRepository
    ) {}

    public function supports(Request $request): ?bool
    {
        return $request->headers->has('X-API-KEY') ||
            $request->headers->has('Authorization');
    }

    public function authenticate(Request $request): Passport
    {

```

```
        $apiKey = $this->extractApiKey($request);

        if (null === $apiKey) {
            throw new CustomUserMessageAuthenticationException('No API key
provided');
        }

        return new SelfValidatingPassport(
            new UserBadge($apiKey, function($apiKey) {
                $user = $this->userRepository->findOneBy(['apiKey' => $apiKey]);

                if (!$user) {
                    throw new CustomUserMessageAuthenticationException('Invalid
API key');
                }

                if (!$user->isActive()) {
                    throw new CustomUserMessageAuthenticationException('User
account is disabled');
                }

                return $user;
            })
        );
    }

    public function onAuthenticationSuccess(Request $request, TokenInterface
$token, string $firewallName): ?Response
    {
        return null;
    }

    public function onAuthenticationFailure(Request $request,
AuthenticationException $exception): ?Response
    {
        return new JsonResponse([
            'error' => true,
            'message' => strstr($exception->getMessageKey(), $exception-
>getMessageData()),
            'code' => Response::HTTP_UNAUTHORIZED
        ], Response::HTTP_UNAUTHORIZED);
    }

    private function extractApiKey(Request $request): ?string
    {
        if ($request->headers->has('X-API-KEY')) {
            return $request->headers->get('X-API-KEY');
        }

        $authHeader = $request->headers->get('Authorization');
        if ($authHeader && str_starts_with($authHeader, 'Bearer ')) {
            return substr($authHeader, 7);
        }
    }
}
```

```
        return $request->cookies->get('api_token');  
    }  
}
```

Sistema de Rate Limiting Implementado

Configuración de Rate Limiters

```
# config/packages/rate_limiter.yaml - Configuración real  
framework:  
    rate_limiter:  
        login_ip:  
            policy: 'sliding_window'  
            limit: 10  
            interval: '15 minutes'  
  
        register_ip:  
            policy: 'sliding_window'  
            limit: 5  
            interval: '1 hour'  
  
        write_operations:  
            policy: 'sliding_window'  
            limit: 30  
            interval: '10 minutes'  
  
        authenticated_user:  
            policy: 'sliding_window'  
            limit: 200  
            interval: '1 hour'  
  
        anonymous_user:  
            policy: 'sliding_window'  
            limit: 50  
            interval: '1 hour'
```

Sistema de Búsqueda Implementado

Repository con Queries Optimizadas

El `RestaurantRepository.php` implementado incluye métodos avanzados de búsqueda:

- `findWithAdvancedSearch()` - Búsqueda con múltiples filtros
- `findSimilarRestaurants()` - Algoritmo de similitud
- `quickSearch()` - Búsqueda rápida para autocompletado
- `getStatistics()` - Estadísticas del sistema

Dockerfile Optimizado

```
# Dockerfile real del proyecto
FROM php:8.3-fpm-alpine

RUN apk add --no-cache \
    git \
    curl \
    libpng-dev \
    libxml2-dev \
    zip \
    unzip

RUN docker-php-ext-install pdo pdo_mysql

COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

WORKDIR /var/www/html

COPY . .

RUN composer install --no-dev --optimize-autoloader

RUN chown -R www-data:www-data /var/www/html/var

EXPOSE 9000
CMD ["php-fpm"]
```

Docker Compose Implementado

```
# docker-compose.yml real del proyecto
services:
  php:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: restaurant_api_php
    restart: unless-stopped
    volumes:
      - ./:/var/www/html:cached
    depends_on:
      - database
    networks:
      - restaurant_network

  nginx:
    image: nginx:alpine
    container_name: restaurant_api_nginx
    restart: unless-stopped
    ports:
```

```
- "8080:80"
volumes:
  - ./:/var/www/html:cached
  - ./docker/nginx/default.conf:/etc/nginx/conf.d/default.conf
depends_on:
  - php
networks:
  - restaurant_network

database:
  image: mysql:8.0
  container_name: restaurant_api_mysql
  restart: unless-stopped
  environment:
    MYSQL_DATABASE: restaurant_api
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
  volumes:
    - db_data:/var/lib/mysql
  networks:
    - restaurant_network
  ports:
    - "3307:3306"

networks:
  restaurant_network:
    driver: bridge

volumes:
  db_data:
```

Comandos de Mantenimiento

Comandos Docker Básicos

```
# Gestión de contenedores
docker-compose up -d --build
docker-compose ps
docker-compose logs -f
docker-compose logs -f php
docker-compose logs -f nginx
docker-compose logs -f database
docker-compose down
docker-compose restart php
docker exec -it restaurant_api_php sh
docker exec -it restaurant_api_mysql mysql -u root -p
```

Comandos Symfony/PHP

```
# Instalación y gestión de dependencias
docker exec restaurant_api_php composer require vendor/package
docker exec restaurant_api_php composer require --dev vendor/package
docker exec restaurant_api_php composer update
docker exec restaurant_api_php composer validate
docker exec restaurant_api_php composer dump-autoload --optimize

# Gestión de cache
docker exec restaurant_api_php php bin/console cache:clear
docker exec restaurant_api_php php bin/console cache:clear --env=prod
docker exec restaurant_api_php php bin/console cache:warmup
docker exec restaurant_api_php rm -rf var/cache/dev/*

# Gestión de base de datos
docker exec restaurant_api_php php bin/console doctrine:database:create
docker exec restaurant_api_php php bin/console doctrine:database:drop --force
docker exec restaurant_api_php php bin/console make:migration
docker exec restaurant_api_php php bin/console doctrine:migrations:migrate
docker exec restaurant_api_php php bin/console doctrine:schema:validate

# Debugging y análisis
docker exec restaurant_api_php php bin/console debug:router
docker exec restaurant_api_php php bin/console debug:container
docker exec restaurant_api_php php bin/console debug:config
docker exec restaurant_api_php php bin/console doctrine:mapping:info
```

⚡ Funcionalidades Avanzadas Implementadas

Event-Driven Architecture

Event Listener para Rate Limiting (Implementado)

```
// Extracto del código real en src/EventListener/RateLimitListener.php
class RateLimitListener
{
    public function onKernelRequest(RequestEvent $event): void
    {
        if (!$event->isMainRequest()) {
            return;
        }

        $request = $event->getRequest();
        $route = $request->attributes->get('_route');

        if (!$route || str_starts_with($route, '_')) {
            return;
        }

        $clientId = $request->getClientIp();
```

```
$user = $this->security->getUser();

$limitType = $this->determineLimitType($request, $user);
$limiter = $this->getLimiterForType($limitType, $clientId, $user);

$limit = $limiter->consume();

if (!$limit->isAccepted()) {
    $this->throwRateLimitException($limitType, $limit);
}
}
```

Event Listener para Headers de Rate Limiting (Implementado)

```
// Extracto del código real en src/EventListener/RateLimitResponseListener.php
class RateLimitResponseListener
{
    public function onKernelResponse(ResponseEvent $event): void
    {
        if (!$event->isMainRequest()) {
            return;
        }

        $request = $event->getRequest();
        $response = $event->getResponse();

        if (!str_starts_with($request->getPathInfo(), '/api/')) {
            return;
        }

        // Agregar headers informativos de rate limiting
        $response->headers->set('X-RateLimit-Limit', $limit->getLimit());
        $response->headers->set('X-RateLimit-Remaining', $limit-
>getRemainingTokens());
        // ... más headers
    }
}
```

Patrones y Mejores Prácticas Implementadas

Repository Pattern Implementado

El proyecto implementa el Repository Pattern con:

- **RestaurantRepository**: Queries optimizadas para búsqueda avanzada
- **UserRepository**: Métodos para gestión de usuarios y autenticación

Ejemplo Real del RestaurantRepository

```
// Extracto de src/Repository/RestaurantRepository.php
class RestaurantRepository extends ServiceEntityRepository
{
    public function findWithAdvancedSearch(
        ?string $search = null,
        ?string $name = null,
        ?string $address = null,
        // ... otros parámetros
    ): array {
        $qb = $this->createQueryBuilder('r');

        if ($search) {
            $qb->andWhere('(r.name LIKE :search OR r.address LIKE :search OR
r.phone LIKE :search)')
                ->setParameter('search', "%{$search}%");
        }

        // ... más lógica de filtros

        return [
            'results' => $qb->getQuery()->getResult(),
            'pagination' => [
                'total' => $total,
                'page' => $page,
                'limit' => $limit,
                'pages' => ceil($total / $limit)
            ]
        ];
    }
}
```

Controller Pattern Implementado

El proyecto tiene controllers especializados:

- **AuthController:** Maneja login, register, logout, refresh API key
- **RestaurantSearchController:** Búsqueda avanzada, estadísticas, similares
- **ErrorController:** Manejo centralizado de errores

Event Listener Pattern Implementado

- **RateLimitListener:** Aplica rate limiting antes de procesar requests
- **RateLimitResponseListener:** Agrega headers informativos a las respuestas