ogCore - Documentacion Tecnica y Funcional

Equipo OpenGnsys

Octubre 2025

Índice

1 ogCore - Documentación Técnica y Funcional **5**

1.1 Tabla de Contenidos . . . 5
1.2 1. Introducción . . . 5
1.2.1 1.1 Propósito del Sistema . . . 6
1.2.2 1.2 Alcance . . . 6
1.3 2. Descripción del Proyecto . . . 6
1.3.1 2.1 ¿Qué es ogCore? . . . 6
1.3.2 2.2 Características Principales . . . 6
1.3.3 2.3 Versión Actual . . . 6
1.4 3. Arquitectura del Sistema . . . 7
1.4.1 3.1 Arquitectura General . . . 7
1.4.2 3.2 Capas de la Aplicación . . . 7
1.4.3 3.3 Patrones de Diseño Utilizados . . . 8
1.5 4. Tecnologías Utilizadas . . . 8
1.5.1 4.1 Backend . . . 8
1.5.2 4.2 Base de Datos . . . 8
1.5.3 4.3 Infraestructura . . . 9
1.5.4 4.4 Testing . . . 9
1.5.5 4.5 Desarrollo . . . 9
1.6 5. Requisitos del Sistema . . . 9
1.6.1 5.1 Requisitos de Hardware (Producción) . . . 9
1.6.2 5.2 Requisitos de Software . . . 10
1.6.3 5.3 Puertos Requeridos . . . 10
1.7 6. Instalación y Configuración . . . 10
1.7.1 6.1 Instalación con Docker . . . 10
1.7.2 6.2 Configuración . . . 11
1.7.3 6.3 Instalación en Producción . . . 12
1.8 7. Componentes del Sistema . . . 12
1.8.1 7.1 Controllers (Controladores) . . . 12
1.8.2 7.2 States (Procesadores de Estado) . . . 12
1.8.3 7.3 DTOs (Data Transfer Objects) . . . 13
1.8.4 7.4 Validators (Validadores) . . . 13
1.8.5 7.5 EventSubscribers (Suscriptores de Eventos) . . . 13
1.8.6 7.6 Factories (Fábricas) . . . 13
1.9 8. Guía de Desarrollo - Crear Nueva Entidad . . . 13
1.9.1 8.1 Visión General . . . 13

1.9.2 8.2 Paso 1: Crear la Entidad . . . 13
1.9.3 8.3 Paso 2: Crear el Repository . . . 15
1.9.4 8.4 Paso 3: Crear los DTOs . . . 16
1.9.5 8.5 Paso 4: Configuración YAML de API Platform . . . 20
1.9.6 8.6 Paso 5: Crear States (Provider y Processor) . . . 21
1.9.7 8.7 Paso 6: Crear Validadores Personalizados (Opcional) . . . 25
1.9.8 8.8 Paso 7: Crear Factory para Testing . . . 26
1.9.9 8.9 Paso 8: Crear Migración . . . 27
1.9.10 8.10 Paso 9: Crear Tests . . . 28
1.9.11 8.11 Paso 10: Documentar en Swagger . . . 31
1.9.12 8.12 Checklist de Creación de Entidad . . . 31
1.9.13 8.13 Comandos Útiles . . . 31
1.9.14 8.14 Buenas Prácticas . . . 32
1.10 9. API RESTful . . . 32
1.10.1 8.1 Documentación . . . 32
1.10.2 8.2 Formato de Respuestas . . . 32
1.10.3 8.3 Paginación . . . 33
1.10.4 8.4 Filtrado . . . 33
1.10.5 8.5 Ordenamiento . . . 33
1.10.6 8.6 Endpoints Principales . . . 34
1.10.7 8.7 Códigos de Estado HTTP . . . 34
1.11 9. Seguridad y Autenticación . . . 35
1.11.1 9.1 Sistema de Autenticación JWT . . . 35
1.11.2 9.2 Control de Acceso . . . 36
1.11.3 9.3 Roles de Usuario . . . 36
1.11.4 9.4 Seguridad de Contraseñas . . . 36
1.11.5 9.5 CORS (Cross-Origin Resource Sharing) . . . 36
1.11.6 9.6 SSL/TLS . . . 36
1.12 10. Servicios Principales . . . 36
1.12.1 10.1 CreatePartitionService . . . 36
1.12.2 10.2 CreateTraceService . . . 37
1.12.3 10.3 ChangeClientNetworkSettingsService . . . 37
1.12.4 10.4 ExternalGitRepositoryService . . . 37
1.12.5 10.5 StatusService (OgBoot/OgDhcp/OgRepository) . . . 37
1.12.6 10.6 UDSClient . . . 37
1.12.7 10.7 Trace/CreateService . . . 37
1.12.8 10.8 Utils/GetPartitionCodeService . . . 37
1.12.9 10.9 Utils/SimplifyOgLiveFilenameService . . . 37
1.12.10 10.10 Utils/GetIpAddressAndNetmaskFromCIDRService . . . 37
1.13 11. Comandos de Consola . . . 38
1.13.1 11.1 Comandos de Inicialización . . . 38
1.13.2 11.2 Comandos de Operación . . . 38
1.13.3 11.3 Comandos de Migración . . . 39
1.13.4 11.4 Comandos de Desarrollo . . . 39
1.13.5 11.5 Configuración de Cron . . . 40
1.14 12. Migraciones de Datos . . . 40
1.14.1 12.1 Sistema de Migraciones de Doctrine . . . 40
1.14.2 12.2 Migración desde OpenGnsys 1.1 . . . 40

1.14.3 12.3 Backup y Restauración . . . 41
1.15 13. Testing . . . 41
1.15.1 13.1 Framework de Testing . . . 41
1.15.2 13.2 Ejecutar Tests . . . 41
1.15.3 13.3 Tipos de Tests . . . 41
1.15.4 13.4 Factories para Testing . . . 42
1.15.5 13.5 Base de Datos de Testing . . . 42
1.16 14. Despliegue . . . 42
1.16.1 14.1 Despliegue con Docker Compose . . . 42
1.16.2 14.2 Despliegue con Jenkins . . . 42
1.16.3 14.3 Paquete Debian . . . 43
1.16.4 14.4 Configuración de Producción . . . 43
1.16.5 14.5 Monitoreo . . . 43
1.16.6 14.6 Escalabilidad . . . 43
1.17 15. Mantenimiento y Operaciones . . . 44
1.17.1 15.1 Reiniciar Base de Datos . . . 44
1.17.2 15.2 Limpiar Caché . . . 44
1.17.3 15.3 Actualizar Dependencias . . . 44
1.17.4 15.4 Regenerar Claves JWT . . . 44
1.17.5 15.5 Backups Automáticos . . . 44
1.17.6 15.6 Monitoreo de Salud . . . 44
1.17.7 15.7 Rotación de Logs . . . 45
1.18 16. Integración con Otros Servicios . . . 45
1.18.1 16.1 ogRepository . . . 45
1.18.2 16.2 ogDhcp . . . 45
1.18.3 16.3 ogBoot . . . 45
1.18.4 16.4 ogAgent . . . 46
1.18.5 16.5 UDS (Universal Desktop Services) . . . 46
1.18.6 16.6 Git (Versionado de Imágenes) . . . 46
1.18.7 16.7 Mercure (Notificaciones en Tiempo Real) . . . 46
1.19 17. Roadmap y Changelog . . . 46
1.19.1 17.1 Versión Actual: 0.5.0 . . . 46
1.19.2 17.2 Últimas Características (v0.25.1 - Octubre 2025) . . . 46
1.19.3 17.3 Próximas Características (Roadmap) . . . 47
1.19.4 17.4 Changelog Resumido . . . 47
1.20 18. Contribución . . . 48
1.20.1 18.1 Guía de Contribución . . . 48
1.20.2 18.2 Estándares de Código . . . 48
1.20.3 18.3 Proceso de Revisión . . . 48
1.20.4 18.4 Reportar Bugs . . . 48
1.21 19. Soporte y Contacto . . . 49
1.21.1 19.1 Documentación Adicional . . . 49
1.21.2 19.2 Recursos . . . 49
1.21.3 19.3 Equipo de Desarrollo . . . 49
1.21.4 19.4 Licencia . . . 49
1.22 Apéndices . . . 49
1.22.1 Apéndice A: Glosario de Términos . . . 49
1.22.2 Apéndice B: Puertos y Servicios . . . 49

1.22.3 Apéndice C: Estructura de Directorios 50
1.22.4 Apéndice D: Variables de Entorno Completas 50

Capítulo 1

ogCore - Documentación Técnica y Funcional

1.1 Tabla de Contenidos

1. Introducción
2. Descripción del Proyecto
3. Arquitectura del Sistema
4. Tecnologías Utilizadas
5. Requisitos del Sistema
6. Instalación y Configuración
7. Componentes del Sistema
8. Guía de Desarrollo - Crear Nueva Entidad
9. API RESTful
10. Seguridad y Autenticación
11. Servicios Principales
12. Comandos de Consola
13. Migraciones de Datos
14. Testing
15. Despliegue
16. Mantenimiento y Operaciones
17. Integración con Otros Servicios
18. Roadmap y Changelog
19. Contribución
20. Soporte y Contacto

1.2 1. Introducción

ogCore es el servicio central de **OpenGnsys**, una plataforma de código abierto diseñada para la gestión centralizada de aulas de informática, laboratorios y entornos educativos. Este servicio proporciona una API RESTful robusta que permite administrar de manera eficiente clientes (equipos), imágenes de sistemas operativos, perfiles de hardware y software, tareas programadas, y mucho más.

1.2.1 1.1 Propósito del Sistema

El propósito principal de ogCore es:

- **Centralizar la gestión** de equipos informáticos en entornos educativos y corporativos
- **Automatizar el despliegue** de sistemas operativos e imágenes
- **Gestionar inventarios** de hardware y software
- **Programar y ejecutar tareas** de manera automática
- **Proporcionar trazabilidad** de todas las operaciones realizadas
- **Facilitar la migración** desde versiones anteriores de OpenGnsys

1.2.2 1.2 Alcance

ogCore gestiona: - Clientes (equipos físicos) - Imágenes de sistemas operativos - Repositorios de imágenes - Perfiles de hardware y software - Unidades organizativas (aulas, grupos) - Redes y subredes - Plantillas PXE - Menús de arranque - Comandos y tareas programadas - Trazas de ejecución - Calendarios remotos - Integración con sistemas externos (UDS, ogRepository, ogDhcp, ogBoot)

1.3 2. Descripción del Proyecto

1.3.1 2.1 ¿Qué es ogCore?

ogCore es el núcleo central de OpenGnsys desarrollado con tecnologías modernas de PHP. Actúa como backend que expone una API RESTful completa, permitiendo la gestión de todos los aspectos relacionados con la administración de aulas informáticas.

1.3.2 2.2 Características Principales

- **API RESTful completa** con documentación Swagger/OpenAPI
- **Autenticación JWT** con refresh tokens
- **Arquitectura basada en eventos** con notificaciones en tiempo real (Mercure)
- **Sistema de colas** para la ejecución de tareas programadas
- **Integración con múltiples servicios** externos
- **Creación de imágenes** con integración Git y sistema monolítico
- **Sistema de trazabilidad** completo
- **Gestión de hardware** con inventario automático
- **Despliegues masivos** de imágenes (unicast, multicast, torrent, p2p)
- **Gestión de particiones** automática
- **Sistema de validación** robusto

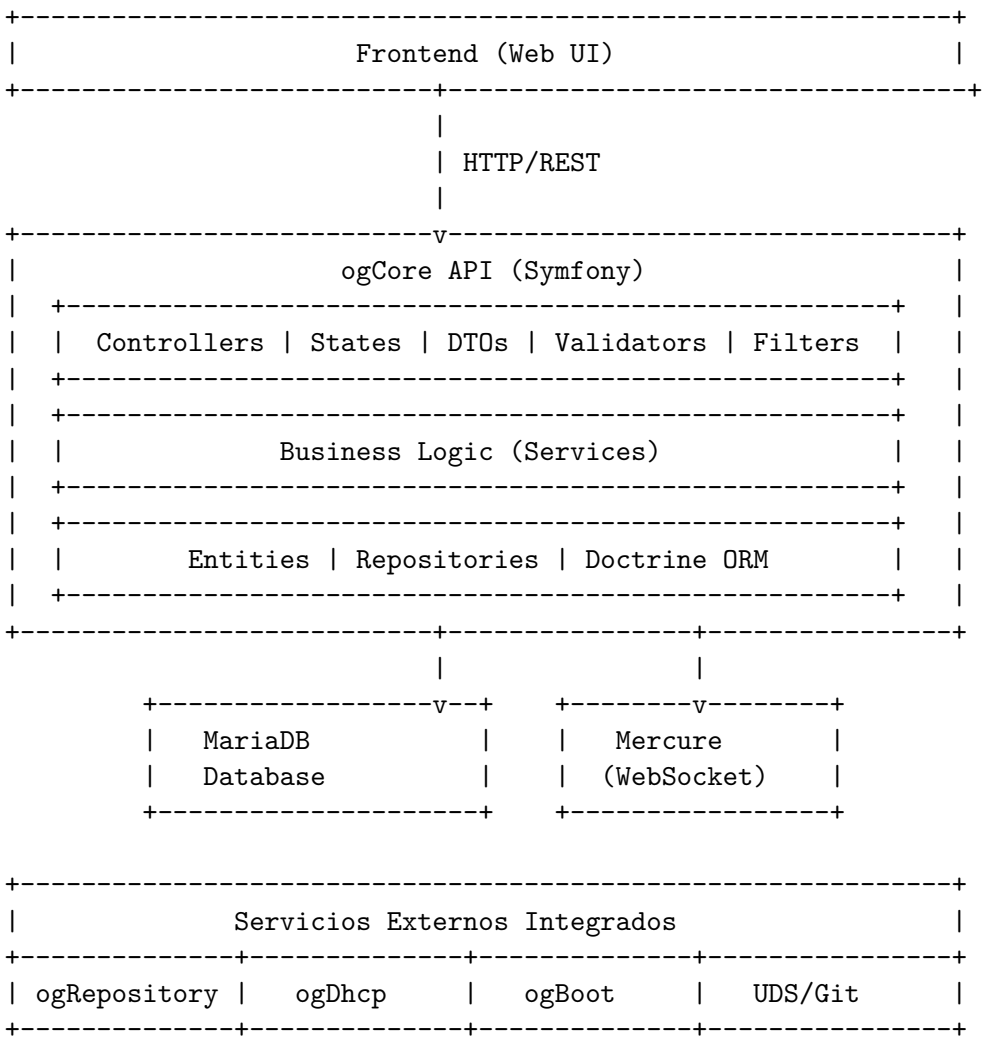
1.3.3 2.3 Versión Actual

- **Versión**: 1.0.0
- **Estado**: En desarrollo activo
- **Última actualización**: Octubre 2025

1.4 3. Arquitectura del Sistema

1.4.1 3.1 Arquitectura General

ogCore sigue una arquitectura de **microservicios** basada en el patrón **API-First**:



1.4.2 3.2 Capas de la Aplicación

1.4.2.1 3.2.1 Capa de Presentación (API)

- **API Platform**: Framework para crear APIs REST
- **Controllers**: Controladores personalizados (102 controladores)
- **DTOs (Data Transfer Objects)**: 93 objetos de transferencia de datos
- **OpenAPI/Swagger**: Documentación automática de la API

1.4.2.2 3.2.2 Capa de Negocio

- **Services**: Lógica de negocio (13 servicios principales)
- **Handlers**: Manejadores de eventos (2 handlers)
- **EventSubscribers**: Suscriptores de eventos (8 suscriptores)

- **Validators**: Validadores personalizados (18 validadores)

1.4.2.3 3.2.3 Capa de Datos

- **Entities**: 35 entidades del modelo de datos
- **Repositories**: 32 repositorios personalizados
- **Doctrine ORM**: Mapeo objeto-relacional
- **Migrations**: 85 migraciones de base de datos

1.4.2.4 3.2.4 Capa de Infraestructura

- **Docker**: Contenedorización de servicios
- **Nginx**: Servidor web reverse proxy
- **PHP-FPM**: Procesador de PHP
- **MariaDB**: Sistema de gestión de base de datos
- **Mercure**: Sistema de notificaciones en tiempo real

1.4.3 3.3 Patrones de Diseño Utilizados

1. **Repository Pattern**: Para abstracción de acceso a datos
2. **DTO Pattern**: Para transferencia de datos entre capas
3. **Factory Pattern**: Para creación de entidades complejas
4. **Event-Driven Architecture**: Para notificaciones y reactividad
5. **State Pattern**: Para gestión de estados de procesamiento
6. **Service Layer**: Para encapsulación de lógica de negocio

---

1.5 4. Tecnologías Utilizadas

1.5.1 4.1 Backend

Tecnología	Versión	Propósito
PHP	8.3	Lenguaje de programación principal
Symfony	6.4	Framework web principal
Doctrine ORM	2.19	Mapeo objeto-relacional
API Platform	3.2	Creación de APIs REST
Lexik JWT	3.0	Autenticación JWT
Gesdinet JWT Refresh Token	1.3	Refresh tokens
Stof Doctrine Extensions	1.10	Extensiones de Doctrine (timestampable, etc.)
Ramsey UUID	2.0	Generación de UUIDs

1.5.2 4.2 Base de Datos

Tecnología	Versión	Propósito
MariaDB	10.11	Sistema de gestión de base de datos
Doctrine DBAL	3.x	Capa de abstracción de base de datos
Doctrine Migrations	3.3	Gestión de migraciones

1.5.3 4.3 Infraestructura

Tecnología	Versión	Propósito
Docker	Latest	Contenedorización
Docker Compose	Latest	Orquestación de contenedores
Nginx	Latest	Servidor web / Reverse proxy
PHP-FPM	8.3	Procesador FastCGI
Mercure	0.3.9	Hub de notificaciones en tiempo real

1.5.4 4.4 Testing

Tecnología	Versión	Propósito
PHPUnit	9.5	Framework de testing
Symfony PHPUnit Bridge	7.0	Integración con Symfony
DAMA Doctrine Test Bundle	8.1	Transacciones de prueba
Zenstruck Foundry	1.37	Factories para testing

1.5.5 4.5 Desarrollo

Tecnología	Versión	Propósito
Symfony Maker Bundle	1.59	Generación de código
Symfony Web Profiler	6.4	Debugging y profiling
Monolog	3.x	Logging
PHPStan	Latest	Análisis estático

1.6 5. Requisitos del Sistema

1.6.1 5.1 Requisitos de Hardware (Producción)

- **CPU**: Mínimo 4 cores, recomendado 8+ cores
- **RAM**: Mínimo 8GB, recomendado 16GB+
- **Disco**:
  - Sistema: 20GB SSD
  - Base de datos: 50GB+ SSD
  - Logs: 10GB
- **Red**: 1Gbps mínimo

1.6.2 5.2 Requisitos de Software

- **Sistema Operativo**: Linux (Ubuntu 20.04+, Debian 11+, CentOS 8+)
- **Docker**: >= 20.10
- **Docker Compose**: >= 2.0
- **Git**: >= 2.25 (para gestión de código)

1.6.3 5.3 Puertos Requeridos

Puerto	Servicio	Propósito
8080	Nginx/HTTP	API y documentación
3306	MariaDB	Base de datos
9000	PHP-FPM	Procesador PHP
3000	Mercure	WebSocket/SSE

1.7 6. Instalación y Configuración

1.7.1 6.1 Instalación con Docker

1.7.1.1 6.1.1 Clonar el Repositorio

```
git clone <url-del-repositorio> ogcore
cd ogcore
```

1.7.1.2 6.1.2 Verificar Puertos Disponibles

Asegúrate de que los puertos 8080 y 3306 no estén en uso:

```
sudo lsof -i :8080
sudo lsof -i :3306
```

1.7.1.3 6.1.3 Desplegar Contenedores

```
docker compose up --build -d
```

1.7.1.4 6.1.4 Verificar Contenedores

```
docker ps
```

Deberías ver tres contenedores activos: - `ogcore-nginx` - `ogcore-php` - `ogcore-database`

1.7.1.5 6.1.5 Instalar Dependencias

```
docker exec ogcore-php composer install
```

1.7.1.6 6.1.6 Generar Claves JWT

```
docker exec ogcore-php php bin/console lexik:jwt:generate-keypair --overwrite
```

1.7.1.7 6.1.7 Inicializar Base de Datos

```
# Ejecutar migraciones
docker exec ogcore-php php bin/console doctrine:migrations:migrate --no-interaction

# Cargar datos iniciales (fixtures)
docker exec ogcore-php php bin/console doctrine:fixtures:load --no-interaction

# Cargar grupos de usuarios por defecto
docker exec ogcore-php php bin/console app:load-default-user-groups

# Cargar comandos por defecto
docker exec ogcore-php php bin/console app:load-default-commands
```

1.7.1.8 6.1.8 Acceder a la Aplicación

Abre tu navegador y accede a:

`http://127.0.0.1:8080/docs`

Deberías ver la documentación Swagger de la API de ogCore.

1.7.2 6.2 Configuración

1.7.2.1 6.2.1 Variables de Entorno

El archivo `.env` contiene las variables de configuración principales:

```
# Entorno
APP_ENV=dev
APP_SECRET=<tu-secreto>

# Base de datos
DATABASE_URL="mysql://user:password@ogcore-database:3306/ogcore"

# JWT
JWT_SECRET_KEY=%kernel.project_dir%/config/jwt/private.pem
JWT_PUBLIC_KEY=%kernel.project_dir%/config/jwt/public.pem
JWT_PASSPHRASE=<tu-passphrase>

# Mercure
MERCURE_URL=http://mercure:3000/.well-known/mercure
MERCURE_PUBLIC_URL=http://127.0.0.1:8080/.well-known/mercure
MERCURE_JWT_SECRET=<tu-secreto-mercure>

# Servicios externos
REMOTE_PC_URL=<url-uds>
REMOTE_PC_AUTH_LOGIN=<login>
REMOTE_PC_AUTH_USERNAME=<username>
REMOTE_PC_AUTH_PASSWORD=<password>
```

```
# SSL/TLS
SSL_ENABLED=false
```

1.7.2.2 6.2.2 Configuración de CORS

Edita `config/packages/nelmio_cors.yaml` para configurar CORS según tus necesidades.

1.7.2.3 6.2.3 Configuración de API Platform

La configuración de API Platform se encuentra en `config/packages/api_platform.yaml`.

1.7.3 6.3 Instalación en Producción

Para entornos de producción, utiliza el archivo `docker-compose-deploy.yml`:

```
docker compose -f docker-compose-deploy.yml up -d
```

Asegúrate de: - Cambiar todas las contraseñas por defecto - Configurar SSL/TLS - Configurar backups automáticos de base de datos - Configurar monitoreo y alertas - Revisar los límites de recursos de Docker

---

1.8 7. Componentes del Sistema

1.8.1 7.1 Controllers (Controladores)

El sistema cuenta con **102 controladores** organizados por dominio:

1.8.1.1 Controladores principales:

- **ClientController**: Gestión de clientes
- **ImageController**: Gestión de imágenes
- **OrganizationalUnitController**: Gestión de unidades organizativas
- **CommandController**: Gestión de comandos
- **TraceController**: Gestión de trazas
- **UserController**: Gestión de usuarios
- **SubnetController**: Gestión de subredes
- **RepositoryController**: Gestión de repositorios

1.8.2 7.2 States (Procesadores de Estado)

Los States implementan el patrón State de API Platform:

1.8.2.1 States principales (55 procesadores):

- **Providers**: Obtención de datos
- **Processors**: Procesamiento de escritura
- **Custom States**: Lógica personalizada

1.8.3 7.3 DTOs (Data Transfer Objects)

93 DTOs para transferencia de datos:

1.8.3.1 Tipos de DTOs:

- **Input DTOs**: Para recepción de datos
- **Output DTOs**: Para envío de datos
- **Transformation DTOs**: Para transformaciones

1.8.4 7.4 Validators (Validadores)

18 validadores personalizados para: - Validación de IPs y MACs - Validación de rangos DHCP - Validación de particiones - Validación de imágenes - Validación de comandos - Validación de usuarios

1.8.5 7.5 EventSubscribers (Suscriptores de Eventos)

8 suscriptores para: - Publicación en Mercure al cambiar estados - Limpieza de recursos - Validaciones pre/post persistencia - Logging de eventos

1.8.6 7.6 Factories (Fábricas)

24 factories para testing con Foundry: - Creación de entidades para tests - Datos de prueba realistas - Estados predefinidos

---

1.9 8. Guía de Desarrollo - Crear Nueva Entidad

Esta sección proporciona una guía paso a paso para añadir una nueva entidad al sistema ogCore, incluyendo todos los componentes necesarios.

1.9.1 8.1 Visión General

Para añadir una nueva entidad completa al sistema necesitarás crear:

1. **Entity** - La entidad Doctrine
2. **Repository** - Repositorio personalizado
3. **DTOs** - Input y Output DTOs
4. **Configuración YAML** - Para API Platform
5. **States** - Providers y Processors
6. **Validators** - Validaciones personalizadas (opcional)
7. **Factory** - Para testing
8. **Migración** - Cambios en base de datos

1.9.2 8.2 Paso 1: Crear la Entidad

Crea la entidad en `src/Entity/`. Ejemplo: `Product.php`

```
<?php
```

```php
namespace App\Entity;

use App\Repository\ProductRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: ProductRepository::class)]
#[ORM\Table(name: 'product')]
class Product extends AbstractEntity
{
    use NameableTrait; // Proporciona id, name, createdAt, updatedAt

    #[ORM\Column(length: 500, nullable: true)]
    #[Assert\Length(max: 500)]
    private ?string $description = null;

    #[ORM\Column]
    #[Assert\NotNull]
    #[Assert\Positive]
    private ?float $price = null;

    #[ORM\Column]
    private ?int $stock = 0;

    #[ORM\Column]
    private ?bool $active = true;

    #[ORM\ManyToOne(targetEntity: Category::class, inversedBy: 'products')]
    #[ORM\JoinColumn(nullable: false, onDelete: 'CASCADE')]
    private ?Category $category = null;

    // Getters y Setters
    public function getDescription(): ?string
    {
        return $this->description;
    }

    public function setDescription(?string $description): static
    {
        $this->description = $description;
        return $this;
    }

    public function getPrice(): ?float
    {
        return $this->price;
    }
```

```
    public function setPrice(float $price): static
    {
        $this->price = $price;
        return $this;
    }

    public function getStock(): ?int
    {
        return $this->stock;
    }

    public function setStock(int $stock): static
    {
        $this->stock = $stock;
        return $this;
    }

    public function isActive(): ?bool
    {
        return $this->active;
    }

    public function setActive(bool $active): static
    {
        $this->active = $active;
        return $this;
    }

    public function getCategory(): ?Category
    {
        return $this->category;
    }

    public function setCategory(?Category $category): static
    {
        $this->category = $category;
        return $this;
    }
}
```

1.9.3 8.3 Paso 2: Crear el Repository

Crea el repositorio en `src/Repository/ProductRepository.php`:

```
<?php

namespace App\Repository;

use App\Entity\Product;
```

```php
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<Product>
 */
class ProductRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Product::class);
    }

    /**
     * Encuentra productos activos por categoría
     */
    public function findActiveByCategory(int $categoryId): array
    {
        return $this->createQueryBuilder('p')
            ->where('p.category = :categoryId')
            ->andWhere('p.active = true')
            ->setParameter('categoryId', $categoryId)
            ->orderBy('p.name', 'ASC')
            ->getQuery()
            ->getResult();
    }

    /**
     * Encuentra productos con stock bajo
     */
    public function findLowStock(int $threshold = 10): array
    {
        return $this->createQueryBuilder('p')
            ->where('p.stock <= :threshold')
            ->andWhere('p.active = true')
            ->setParameter('threshold', $threshold)
            ->getQuery()
            ->getResult();
    }
}
```

1.9.4 8.4 Paso 3: Crear los DTOs

1.9.4.1 8.4.1 Input DTO

Crea `src/Dto/Input/ProductInput.php`:

```php
<?php
```

```php
namespace App\Dto\Input;

use ApiPlatform\Metadata\ApiProperty;
use App\Dto\Output\CategoryOutput;
use App\Entity\Product;
use Symfony\Component\Serializer\Annotation\Groups;
use Symfony\Component\Validator\Constraints as Assert;

final class ProductInput
{
    #[Assert\NotBlank(message: 'El nombre es obligatorio')]
    #[Assert\Length(
        min: 3,
        max: 255,
        minMessage: 'El nombre debe tener al menos {{ limit }} caracteres',
        maxMessage: 'El nombre no puede tener más de {{ limit }} caracteres'
    )]
    #[Groups(['product:write'])]
    #[ApiProperty(description: 'Nombre del producto', example: 'Laptop HP')]
    public ?string $name = null;

    #[Assert\Length(max: 500)]
    #[Groups(['product:write'])]
    #[ApiProperty(description: 'Descripción del producto')]
    public ?string $description = null;

    #[Assert\NotNull(message: 'El precio es obligatorio')]
    #[Assert\Positive(message: 'El precio debe ser positivo')]
    #[Groups(['product:write'])]
    #[ApiProperty(description: 'Precio del producto', example: 999.99)]
    public ?float $price = null;

    #[Assert\NotNull]
    #[Assert\PositiveOrZero]
    #[Groups(['product:write'])]
    #[ApiProperty(description: 'Stock disponible', example: 50)]
    public ?int $stock = 0;

    #[Groups(['product:write'])]
    #[ApiProperty(description: 'Producto activo', example: true)]
    public ?bool $active = true;

    #[Assert\NotNull(message: 'La categoría es obligatoria')]
    #[Groups(['product:write'])]
    #[ApiProperty(description: 'Categoría del producto')]
    public ?CategoryOutput $category = null;

    /**
```

```
     * Constructor que puede hidratar desde una entidad (para PUT/PATCH)
     */
    public function __construct(?Product $product = null)
    {
        if ($product) {
            $this->name = $product->getName();
            $this->description = $product->getDescription();
            $this->price = $product->getPrice();
            $this->stock = $product->getStock();
            $this->active = $product->isActive();
            $this->category = $product->getCategory()
                ? CategoryOutput::fromEntity($product->getCategory())
                : null;
        }
    }

    /**
     * Crea o actualiza una entidad Product desde este DTO
     */
    public function createOrUpdateEntity(?Product $product = null): Product
    {
        if (!$product) {
            $product = new Product();
        }

        $product->setName($this->name);
        $product->setDescription($this->description);
        $product->setPrice($this->price);
        $product->setStock($this->stock);
        $product->setActive($this->active ?? true);

        // La categoría se asigna en el Processor después de validarla

        return $product;
    }
}
```

1.9.4.2 8.4.2 Output DTO

Crea `src/Dto/Output/ProductOutput.php`:

```
<?php

namespace App\Dto\Output;

use ApiPlatform\Metadata\ApiProperty;
use App\Entity\Product;
use Symfony\Component\Serializer\Annotation\Groups;
```

```php
class ProductOutput extends AbstractOutput
{
    #[Groups(['product:read'])]
    #[ApiProperty(description: 'Nombre del producto')]
    public string $name;

    #[Groups(['product:read'])]
    #[ApiProperty(description: 'Descripción del producto')]
    public ?string $description;

    #[Groups(['product:read'])]
    #[ApiProperty(description: 'Precio del producto')]
    public float $price;

    #[Groups(['product:read'])]
    #[ApiProperty(description: 'Stock disponible')]
    public int $stock;

    #[Groups(['product:read'])]
    #[ApiProperty(description: 'Producto activo')]
    public bool $active;

    #[Groups(['product:read'])]
    #[ApiProperty(description: 'Categoría del producto')]
    public CategoryOutput $category;

    #[Groups(['product:read'])]
    #[ApiProperty(description: 'Fecha de creación')]
    public \DateTimeInterface $createdAt;

    #[Groups(['product:read'])]
    #[ApiProperty(description: 'Fecha de actualización')]
    public \DateTimeInterface $updatedAt;

    /**
     * Constructor que recibe la entidad y la almacena
     */
    public function __construct(Product $product)
    {
        parent::__construct($product);  // Guarda la entidad y establece id/uuid

        $this->name = $product->getName();
        $this->description = $product->getDescription();
        $this->price = $product->getPrice();
        $this->stock = $product->getStock();
        $this->active = $product->isActive();
        $this->category = new CategoryOutput($product->getCategory());
        $this->createdAt = $product->getCreatedAt();
```

```
        $this->updatedAt = $product->getUpdatedAt();
    }
}
```

1.9.5 8.5 Paso 4: Configuración YAML de API Platform

Crea `config/api_platform/Product.yaml`:

```yaml
App\Entity\Product:
  operations:
    get:
      provider: App\State\Provider\ProductProvider
      output: App\Dto\Output\ProductOutput
    getCollection:
      provider: App\State\Provider\ProductProvider
      output: App\Dto\Output\ProductOutput
      filters:
        - 'App\Filter\SearchFilter'
    post:
      input: App\Dto\Input\ProductInput
      processor: App\State\Processor\ProductProcessor
      output: App\Dto\Output\ProductOutput
      security: "is_granted('ROLE_ADMIN')"
    put:
      input: App\Dto\Input\ProductInput
      provider: App\State\Provider\ProductProvider
      processor: App\State\Processor\ProductProcessor
      output: App\Dto\Output\ProductOutput
      security: "is_granted('ROLE_ADMIN')"
    patch:
      input: App\Dto\Input\ProductInput
      provider: App\State\Provider\ProductProvider
      processor: App\State\Processor\ProductProcessor
      output: App\Dto\Output\ProductOutput
      security: "is_granted('ROLE_ADMIN')"
    delete:
      processor: App\State\Processor\ProductProcessor
      security: "is_granted('ROLE_ADMIN')"

  properties:
    id:
      identifier: true
    name:
      required: true
    price:
      required: true
    stock:
      required: true
    active:
```

```
      required: true
    category:
      required: true
    description:
      required: false
    createdAt:
      required: false
    updatedAt:
      required: false
```

1.9.6 8.6 Paso 5: Crear States (Provider y Processor)

ogCore usa **un único Provider** y **un único Processor** por entidad que manejan todas las operaciones HTTP.

1.9.6.1 8.6.1 Provider

Crea `src/State/Provider/ProductProvider.php`:

```php
<?php

namespace App\State\Provider;

use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\GetCollection;
use ApiPlatform\Metadata\Operation;
use ApiPlatform\Metadata\Patch;
use ApiPlatform\Metadata\Put;
use ApiPlatform\State\Pagination\TraversablePaginator;
use ApiPlatform\State\ProviderInterface;
use App\Dto\Input\ProductInput;
use App\Dto\Output\ProductOutput;
use App\Repository\ProductRepository;
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;

readonly class ProductProvider implements ProviderInterface
{
    public function __construct(
        private ProductRepository $productRepository,
        private ProviderInterface $collectionProvider,  // Inyección de API Platform
        private ProviderInterface $itemProvider         // Inyección de API Platform
    ) {}

    public function provide(Operation $operation, array $uriVariables = [], array $context = []
    {
        // Switch según el tipo de operación
        switch ($operation) {
            case $operation instanceof GetCollection:
                return $this->provideCollection($operation, $uriVariables, $context);
```

```php
        case $operation instanceof Patch:
        case $operation instanceof Put:
            return $this->provideInput($operation, $uriVariables, $context);
        case $operation instanceof Get:
            return $this->provideItem($operation, $uriVariables, $context);
    }
}

/**
 * Proporciona la colección de productos
 */
private function provideCollection(Operation $operation, array $uriVariables = [], array $c
{
    $filters = $context['filters'] ?? [];

    // Usar el provider de API Platform para paginación automática
    $paginator = $this->collectionProvider->provide($operation, $uriVariables, $context);

    // Convertir entidades a DTOs
    $items = new \ArrayObject();
    foreach ($paginator->getIterator() as $product) {
        $items[] = new ProductOutput($product);
    }

    return new TraversablePaginator(
        $items,
        $paginator->getCurrentPage(),
        $paginator->getItemsPerPage(),
        $paginator->getTotalItems()
    );
}

/**
 * Proporciona un producto individual
 */
private function provideItem(Operation $operation, array $uriVariables = [], array $context
{
    $product = $this->itemProvider->provide($operation, $uriVariables, $context);

    if (!$product) {
        throw new NotFoundHttpException('Producto no encontrado');
    }

    return new ProductOutput($product);
}

/**
 * Proporciona el Input DTO para PUT/PATCH
```

```
     */
    private function provideInput(Operation $operation, array $uriVariables = [], array $contex
    {
        if (isset($uriVariables['id'])) {
            $product = $this->itemProvider->provide($operation, $uriVariables, $context);

            return $product !== null ? new ProductInput($product) : null;
        }

        return new ProductInput();
    }
}
```

1.9.6.2 8.6.2 Processor

Crea `src/State/Processor/ProductProcessor.php`:

```
<?php

namespace App\State\Processor;

use ApiPlatform\Metadata\Delete;
use ApiPlatform\Metadata\Operation;
use ApiPlatform\Metadata\Patch;
use ApiPlatform\Metadata\Post;
use ApiPlatform\Metadata\Put;
use ApiPlatform\State\ProcessorInterface;
use ApiPlatform\Validator\ValidatorInterface;
use App\Dto\Input\ProductInput;
use App\Dto\Output\ProductOutput;
use App\Repository\CategoryRepository;
use App\Repository\ProductRepository;
use Symfony\Component\HttpKernel\Exception\BadRequestHttpException;

readonly class ProductProcessor implements ProcessorInterface
{
    public function __construct(
        private ProductRepository $productRepository,
        private CategoryRepository $categoryRepository,
        private ValidatorInterface $validator
    ) {}

    /**
     * Método principal que maneja todas las operaciones
     */
    public function process(mixed $data, Operation $operation, array $uriVariables = [], array
    {
        switch ($operation) {
            case $operation instanceof Post:
```

```
        case $operation instanceof Put:
        case $operation instanceof Patch:
            return $this->processCreateOrUpdate($data, $operation, $uriVariables, $context)
        case $operation instanceof Delete:
            return $this->processDelete($data, $operation, $uriVariables, $context);
    }
}

/**
 * Procesa creación y actualización (POST, PUT, PATCH)
 */
private function processCreateOrUpdate(
    mixed $data,
    Operation $operation,
    array $uriVariables = [],
    array $context = []
): ProductOutput {
    if (!($data instanceof ProductInput)) {
        throw new \Exception(sprintf('Data is not instance of %s', ProductInput::class));
    }

    $product = null;

    // Si es actualización, obtener la entidad existente
    if (isset($uriVariables['id'])) {
        $product = $this->productRepository->find($uriVariables['id']);
    }

    // Buscar categoría desde el Output DTO
    $category = $data->category ? $data->category->getEntity() : null;
    if (!$category) {
        throw new BadRequestHttpException('Categoría no encontrada');
    }

    // Crear o actualizar usando método del Input DTO
    $product = $data->createOrUpdateEntity($product);
    $product->setCategory($category);

    // Validar la entidad
    $this->validator->validate($product);

    // Guardar
    $this->productRepository->save($product);

    return new ProductOutput($product);
}

/**
```

```
     * Procesa eliminación (DELETE)
     */
    private function processDelete(
        mixed $data,
        Operation $operation,
        array $uriVariables = [],
        array $context = []
    ): null {
        $product = $this->productRepository->find($uriVariables['id']);

        if (!$product) {
            throw new BadRequestHttpException('Producto no encontrado');
        }

        $this->productRepository->delete($product);

        return null;
    }
}
```

1.9.7 8.7 Paso 6: Crear Validadores Personalizados (Opcional)

Si necesitas validaciones complejas, crea un validador. Ejemplo: `src/Validator/ProductStockValidator.php`:

```
<?php

namespace App\Validator;

use Symfony\Component\Validator\Constraint;
use Symfony\Component\Validator\ConstraintValidator;
use Symfony\Component\Validator\Exception\UnexpectedTypeException;

class ProductStockValidator extends ConstraintValidator
{
    public function validate($value, Constraint $constraint): void
    {
        if (!$constraint instanceof ProductStock) {
            throw new UnexpectedTypeException($constraint, ProductStock::class);
        }

        if (null === $value || '' === $value) {
            return;
        }

        if ($value < 0) {
            $this->context->buildViolation($constraint->message)
                ->setParameter('{{ value }}', $value)
                ->addViolation();
        }
```

```
        // Validación adicional: stock máximo
        if ($value > $constraint->maxStock) {
            $this->context->buildViolation('El stock no puede exceder {{ limit }}')
                ->setParameter('{{ limit }}', $constraint->maxStock)
                ->addViolation();
        }
    }
}
```

Y la constraint `src/Validator/ProductStock.php`:

```
<?php

namespace App\Validator;

use Symfony\Component\Validator\Constraint;

#[\Attribute]
class ProductStock extends Constraint
{
    public string $message = 'El stock "{{ value }}" no es válido';
    public int $maxStock = 10000;

    public function validatedBy(): string
    {
        return static::class.'Validator';
    }
}
```

1.9.8 8.8 Paso 7: Crear Factory para Testing

Crea `src/Factory/ProductFactory.php`:

```
<?php

namespace App\Factory;

use App\Entity\Product;
use Zenstruck\Foundry\ModelFactory;

/**
 * @extends ModelFactory<Product>
 */
final class ProductFactory extends ModelFactory
{
    protected function getDefaults(): array
    {
        return [
            'name' => self::faker()->words(3, true),
```

```
            'description' => self::faker()->sentence(),
            'price' => self::faker()->randomFloat(2, 1, 1000),
            'stock' => self::faker()->numberBetween(0, 100),
            'active' => self::faker()->boolean(80), // 80% activos
            'category' => CategoryFactory::new(),
        ];
    }

    protected static function getClass(): string
    {
        return Product::class;
    }

    /**
     * Estado: producto sin stock
     */
    public function outOfStock(): self
    {
        return $this->addState(['stock' => 0]);
    }

    /**
     * Estado: producto inactivo
     */
    public function inactive(): self
    {
        return $this->addState(['active' => false]);
    }

    /**
     * Estado: producto de lujo (precio alto)
     */
    public function luxury(): self
    {
        return $this->addState([
            'price' => self::faker()->randomFloat(2, 500, 5000)
        ]);
    }
}
```

1.9.9 8.9 Paso 8: Crear Migración

Genera la migración automáticamente:

```
docker exec ogcore-php php bin/console make:migration
```

Revisa y edita la migración generada en `migrations/`:

```
<?php
```

```php
declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

final class Version20251013120000 extends AbstractMigration
{
    public function getDescription(): string
    {
        return 'Crea la tabla product';
    }

    public function up(Schema $schema): void
    {
        $this->addSql('CREATE TABLE product (
            id CHAR(36) NOT NULL COMMENT \'(DC2Type:uuid)\',
            category_id CHAR(36) NOT NULL COMMENT \'(DC2Type:uuid)\',
            name VARCHAR(255) NOT NULL,
            description VARCHAR(500) DEFAULT NULL,
            price DOUBLE PRECISION NOT NULL,
            stock INT NOT NULL,
            active TINYINT(1) NOT NULL,
            created_at DATETIME NOT NULL COMMENT \'(DC2Type:datetime_immutable)\',
            updated_at DATETIME NOT NULL COMMENT \'(DC2Type:datetime_immutable)\',
            INDEX IDX_PRODUCT_CATEGORY (category_id),
            INDEX IDX_PRODUCT_ACTIVE (active),
            PRIMARY KEY(id)
        ) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');

        $this->addSql('ALTER TABLE product ADD CONSTRAINT FK_PRODUCT_CATEGORY
            FOREIGN KEY (category_id) REFERENCES category (id) ON DELETE CASCADE');
    }

    public function down(Schema $schema): void
    {
        $this->addSql('ALTER TABLE product DROP FOREIGN KEY FK_PRODUCT_CATEGORY');
        $this->addSql('DROP TABLE product');
    }
}
```

Ejecuta la migración:

```
docker exec ogcore-php php bin/console doctrine:migrations:migrate --no-interaction
```

1.9.10 8.10 Paso 9: Crear Tests

Crea `tests/Functional/ProductTest.php`:

```php
<?php

namespace App\Tests\Functional;

use App\Factory\CategoryFactory;
use App\Factory\ProductFactory;
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Zenstruck\Foundry\Test\Factories;
use Zenstruck\Foundry\Test\ResetDatabase;

class ProductTest extends WebTestCase
{
    use ResetDatabase, Factories;

    public function testGetCollection(): void
    {
        $client = static::createClient();

        // Crear productos de prueba
        ProductFactory::createMany(5);

        $client->request('GET', '/products', [
            'headers' => ['Accept' => 'application/json']
        ]);

        $this->assertResponseIsSuccessful();
        $this->assertJsonContains(['hydra:totalItems' => 5]);
    }

    public function testPostProduct(): void
    {
        $client = static::createClient();
        $category = CategoryFactory::createOne();

        $client->request('POST', '/products', [
            'json' => [
                'name' => 'Producto Test',
                'description' => 'Descripción de prueba',
                'price' => 99.99,
                'stock' => 50,
                'active' => true,
                'categoryId' => $category->getId()
            ],
            'headers' => ['Content-Type' => 'application/json']
        ]);

        $this->assertResponseStatusCodeSame(201);
        $this->assertJsonContains([
```

```
            'name' => 'Producto Test',
            'price' => 99.99
        ]);
    }

    public function testPutProduct(): void
    {
        $client = static::createClient();
        $product = ProductFactory::createOne();
        $newCategory = CategoryFactory::createOne();

        $client->request('PUT', '/products/' . $product->getId(), [
            'json' => [
                'name' => 'Producto Actualizado',
                'price' => 149.99,
                'stock' => 30,
                'categoryId' => $newCategory->getId()
            ],
            'headers' => ['Content-Type' => 'application/json']
        ]);

        $this->assertResponseIsSuccessful();
        $this->assertJsonContains([
            'name' => 'Producto Actualizado',
            'price' => 149.99
        ]);
    }

    public function testDeleteProduct(): void
    {
        $client = static::createClient();
        $product = ProductFactory::createOne();

        $client->request('DELETE', '/products/' . $product->getId());

        $this->assertResponseStatusCodeSame(204);
    }

    public function testValidation(): void
    {
        $client = static::createClient();

        $client->request('POST', '/products', [
            'json' => [
                'name' => 'AB', // Muy corto
                'price' => -10,  // Negativo
            ],
            'headers' => ['Content-Type' => 'application/json']
```

```
        ]);

        $this->assertResponseStatusCodeSame(422);
    }
}
```

1.9.11 8.11 Paso 10: Documentar en Swagger

La configuración YAML de API Platform generará automáticamente la documentación en Swagger. Puedes añadir descripciones adicionales:

```yaml
App\Entity\Product:
  description: 'Entidad que representa un producto del catálogo'
  operations:
    get:
      summary: 'Obtener un producto por ID'
      description: 'Retorna la información detallada de un producto'
      provider: App\State\Product\ProductProvider
      output: App\Dto\Output\ProductOutput
    getCollection:
      summary: 'Obtener listado de productos'
      description: 'Retorna una colección paginada de productos'
      provider: App\State\Product\ProductCollectionProvider
      output: App\Dto\Output\ProductOutput
    # ... resto de operaciones
```

1.9.12 8.12 Checklist de Creación de Entidad

Usa este checklist para asegurarte de no olvidar ningún paso:

- ☐ **Entidad creada** en `src/Entity/`
- ☐ **Repository creado** en `src/Repository/`
- ☐ **Input DTO creado** en `src/Dto/Input/` con constructor y método `createOrUpdateEntity()`
- ☐ **Output DTO creado** en `src/Dto/Output/` con método estático `fromEntity()`
- ☐ **Configuración YAML** en `config/api_platform/`
- ☐ **Provider creado** en `src/State/Provider/` (un solo archivo que maneja Get y GetCollection)
- ☐ **Processor creado** en `src/State/Processor/` (un solo archivo que maneja Post, Put, Patch, Delete)
- ☐ **Validators** personalizados (si aplica) en `src/Validator/`
- ☐ **Factory** para testing en `src/Factory/`
- ☐ **Migración generada** con `make:migration` y ejecutada
- ☐ **Tests funcionales** creados en `tests/Functional/`
- ☐ **Documentación Swagger** verificada en `/docs`
- ☐ **Permisos de seguridad** configurados en YAML con `security: "is_granted('ROLE_X')"`

1.9.13 8.13 Comandos Útiles

```
# Generar entidad con maker
docker exec ogcore-php php bin/console make:entity Product
```

```
# Generar migración
docker exec ogcore-php php bin/console make:migration

# Ejecutar migración
docker exec ogcore-php php bin/console doctrine:migrations:migrate

# Validar esquema
docker exec ogcore-php php bin/console doctrine:schema:validate

# Ejecutar tests
docker exec ogcore-php php bin/phpunit tests/Functional/ProductTest.php

# Ver rutas de API
docker exec ogcore-php php bin/console debug:router | grep product
```

1.9.14 8.14 Buenas Prácticas

1. **Nomenclatura consistente**: Usa el mismo nombre base para entidad, DTOs, States y Factory
2. **Validaciones**: Siempre valida en el Input DTO y en el Processor si es necesario
3. **Seguridad**: Define permisos en la configuración YAML (`security: "is_granted('ROLE_ADMIN')"`)
4. **Tests**: Crea tests para todos los endpoints (GET, POST, PUT, DELETE)
5. **DTOs separados**: Nunca expongas la entidad directamente, siempre usa DTOs
6. **Transacciones**: Doctrine maneja transacciones automáticamente, pero para operaciones complejas considera usar `$entityManager->transactional()`
7. **Lazy loading**: Configura correctamente las relaciones (fetch LAZY/EAGER)
8. **Índices**: Añade índices en campos que se usan frecuentemente en WHERE/ORDER BY
9. **Soft Delete**: Si necesitas "borrado suave", usa `SoftDeleteableTrait` de Gedmo
10. **Eventos**: Usa EventSubscribers para lógica transversal (logging, notificaciones Mercure)

1.10 9. API RESTful

1.10.1 8.1 Documentación

La API está completamente documentada con **OpenAPI 3.0** y accesible en:

`http://127.0.0.1:8080/docs`

1.10.2 8.2 Formato de Respuestas

1.10.2.1 Formato estándar (JSON-LD):

```
{
  "@context": "/contexts/Client",
  "@id": "/clients/123e4567-e89b-12d3-a456-426614174000",
  "@type": "Client",
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "name": "PC-01",
  "mac": "00:11:22:33:44:55",
```

```
  "ip": "192.168.1.100",
  "status": "active"
}
```

1.10.2.2 Formato alternativo (JSON):

```
{
  "id": "123e4567-e89b-12d3-a456-426614174000",
  "name": "PC-01",
  "mac": "00:11:22:33:44:55",
  "ip": "192.168.1.100",
  "status": "active"
}
```

1.10.3 8.3 Paginación

Todas las colecciones soportan paginación:

```
GET /clients?page=1&itemsPerPage=30
```

Respuesta:

```
{
  "@context": "/contexts/Client",
  "@id": "/clients",
  "@type": "hydra:Collection",
  "hydra:member": [...],
  "hydra:totalItems": 150,
  "hydra:view": {
    "@id": "/clients?page=1",
    "@type": "hydra:PartialCollectionView",
    "hydra:first": "/clients?page=1",
    "hydra:last": "/clients?page=5",
    "hydra:next": "/clients?page=2"
  }
}
```

1.10.4 8.4 Filtrado

Soporta filtros avanzados:

```
GET /clients?status=active
GET /clients?organizationalUnit.name=Aula1
GET /traces?client.id=123&status=pending
```

1.10.5 8.5 Ordenamiento

```
GET /clients?order[name]=asc
GET /traces?order[createdAt]=desc
```

1.10.6 8.6 Endpoints Principales

1.10.6.1 9.6.1 Autenticación

```
POST /auth/login
POST /auth/refresh
```

1.10.6.2 9.6.2 Clientes

```
GET    /clients
POST   /clients
GET    /clients/{id}
PUT    /clients/{id}
PATCH  /clients/{id}
DELETE /clients/{id}
POST   /clients/batch
POST   /clients/{id}/power-on
POST   /clients/{id}/power-off
POST   /clients/{id}/reboot
```

1.10.6.3 9.6.3 Imágenes

```
GET    /images
POST   /images
GET    /images/{id}
PUT    /images/{id}
DELETE /images/{id}
POST   /images/{id}/deploy
POST   /images/{id}/create
POST   /images/{id}/backup
```

1.10.6.4 9.6.4 Comandos y Tareas

```
GET    /commands
POST   /commands
GET    /command-tasks
POST   /command-tasks
PUT    /command-tasks/{id}
DELETE /command-tasks/{id}
```

1.10.6.5 9.6.5 Trazas

```
GET    /traces
GET    /traces/{id}
PUT    /traces/{id}
PATCH  /traces/{id}/complete
PATCH  /traces/{id}/cancel
```

1.10.7 8.7 Códigos de Estado HTTP

Código	Significado
200	OK - Operación exitosa
201	Created - Recurso creado
204	No Content - Eliminación exitosa
400	Bad Request - Datos inválidos
401	Unauthorized - No autenticado
403	Forbidden - Sin permisos
404	Not Found - Recurso no encontrado
409	Conflict - Conflicto (cliente ocupado, constraint violation)
422	Unprocessable Entity - Validación fallida
500	Internal Server Error - Error del servidor

1.11 9. Seguridad y Autenticación

1.11.1 9.1 Sistema de Autenticación JWT

ogCore utiliza **JSON Web Tokens (JWT)** para autenticación:

1.11.1.1 10.1.1 Obtener Token

```
POST /auth/login
Content-Type: application/json

{
  "username": "admin",
  "password": "password"
}
```

Respuesta:

```
{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGc...",
  "refresh_token": "def50200a54b7b..."
}
```

1.11.1.2 10.1.2 Usar Token

Incluir el token en el header `Authorization`:

```
GET /clients
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGc...
```

1.11.1.3 10.1.3 Refresh Token

```
POST /auth/refresh
Content-Type: application/json

{
```

```
  "refresh_token": "def50200a54b7b..."
}
```

1.11.2 9.2 Control de Acceso

1.11.2.1 Rutas Públicas:

- `/auth/login`
- `/auth/refresh`
- `/docs`
- `/opengnsys/rest/*` (webhooks)
- `/og-repository/webhook`
- `/menu-browser`

1.11.2.2 Rutas Protegidas:

- Todo lo demás requiere `IS_AUTHENTICATED_FULLY`

1.11.3 9.3 Roles de Usuario

Los roles se gestionan en la entidad `User`:

```
$user->setRoles(['ROLE_USER']);
$user->setRoles(['ROLE_ADMIN']);
```

1.11.4 9.4 Seguridad de Contraseñas

- Hashing con **bcrypt** automático
- Validación de fortaleza
- Cambio de contraseña seguro

1.11.5 9.5 CORS (Cross-Origin Resource Sharing)

Configurado en `nelmio_cors.yaml` para permitir acceso desde frontend.

1.11.6 9.6 SSL/TLS

Para producción, habilitar SSL:

```
SSL_ENABLED=true
```

Configurar certificados en `/certs/`.

1.12 10. Servicios Principales

1.12.1 10.1 CreatePartitionService

Propósito: Crear y actualizar particiones de clientes.

Funcionalidad: - Recibe información de particiones del agente - Crea/actualiza particiones en base de datos - Detecta tipo de firmware (BIOS/UEFI) - Calcula códigos de partición

1.12.2 10.2 CreateTraceService

Propósito: Crear trazas de ejecución para tareas programadas.

Funcionalidad: - Genera trazas para cada combinación cliente-comando - Gestiona comandos agrupados - Establece estado inicial y fecha de ejecución

1.12.3 10.3 ChangeClientNetworkSettingsService

Propósito: Cambiar configuración de red de clientes.

Funcionalidad: - Actualiza subnet, IP, configuración de red - Sincroniza con servicios externos (DHCP)

1.12.4 10.4 ExternalGitRepositoryService

Propósito: Gestión de repositorios Git para imágenes.

Funcionalidad: - Backup de imágenes a Git - Versionado de imágenes - Sincronización con repositorios remotos

1.12.5 10.5 StatusService (OgBoot/OgDhcp/OgRepository)

Propósito: Verificar estado de servicios externos.

Funcionalidad: - Health checks de servicios - Manejo de errores de conexión - Logging de estado

1.12.6 10.6 UDSClient

Propósito: Integración con UDS (Universal Desktop Services).

Funcionalidad: - Autenticación con UDS - Obtención de service pools - Cálculo de asientos disponibles - Gestión de calendarios remotos

1.12.7 10.7 Trace/CreateService

Propósito: Servicio especializado para creación de trazas.

Funcionalidad: - Validación de parámetros de entrada - Creación de trazas individuales o masivas

1.12.8 10.8 Utils/GetPartitionCodeService

Propósito: Obtener código de partición según tipo y filesystem.

1.12.9 10.9 Utils/SimplifyOgLiveFilenameService

Propósito: Simplificar nombres de archivos OgLive para mostrar al usuario.

1.12.10 10.10 Utils/GetIpAddressAndNetmaskFromCIDRService

Propósito: Convertir notación CIDR a IP y máscara de red.

1.13 11. Comandos de Consola

ogCore incluye **18 comandos de consola** para tareas administrativas:

1.13.1 11.1 Comandos de Inicialización

1.13.1.1 12.1.1 Cargar Grupos de Usuario por Defecto

```
php bin/console app:load-default-user-groups
```

Crea los grupos de usuarios predeterminados del sistema.

1.13.1.2 12.1.2 Cargar Comandos por Defecto

```
php bin/console app:load-default-commands
```

Carga los comandos básicos del sistema.

1.13.1.3 12.1.3 Cargar Usuario Admin

```
php bin/console app:load-default-user-admin
```

Crea el usuario administrador por defecto.

1.13.1.4 12.1.4 Cargar Menú por Defecto

```
php bin/console app:load-default-menu
```

Carga menús de arranque predeterminados.

1.13.1.5 12.1.5 Cargar Tipos de Hardware

```
php bin/console app:load-hardware-types
```

Inicializa la tabla de tipos de hardware.

1.13.1.6 12.1.6 Cargar Unidad Organizativa por Defecto

```
php bin/console app:load-organizational-unit-default
```

Crea la unidad organizativa raíz.

1.13.2 11.2 Comandos de Operación

1.13.2.1 12.2.1 Verificar Disponibilidad de Clientes

```
php bin/console app:check-client-availability
```

Ejecución: Cada 1 minuto (cron)

Verifica el estado de conectividad de los clientes y actualiza su estado si no responden en un tiempo determinado.

1.13.2.2 12.2.2 Ejecutar Trazas Pendientes

```
php bin/console app:execute-pending-traces
```

Ejecución: Cada 1 minuto (cron)

Procesa las trazas en estado `PENDING` y las ejecuta en los clientes correspondientes.

1.13.2.3 12.2.3 Ejecutar Tareas Programadas

```
php bin/console app:run-scheduled-command-tasks
```

Ejecución: Cada 1 minuto (cron)

Ejecuta tareas programadas que han llegado a su hora de ejecución.

1.13.3 11.3 Comandos de Migración

Comandos para migrar datos desde OpenGnsys 1.1:

1.13.3.1 12.3.1 Migrar Unidades Organizativas

```
php bin/console opengnsys:migration:organizational-unit
```

1.13.3.2 12.3.2 Migrar Perfiles de Hardware

```
php bin/console opengnsys:migration:hardware-profile
```

1.13.3.3 12.3.3 Migrar Clientes

```
php bin/console opengnsys:migration:clients
```

1.13.3.4 12.3.4 Migrar Sistemas Operativos

```
php bin/console opengnsys:migration:os
```

1.13.3.5 12.3.5 Migrar Imágenes

```
php bin/console opengnsys:migration:image
```

1.13.3.6 12.3.6 Migrar Perfiles de Software

```
php bin/console opengnsys:migration:software-profile
```

1.13.3.7 12.3.7 Migrar Particiones de Clientes

```
php bin/console opengnsys:migration:partition-client
```

1.13.4 11.4 Comandos de Desarrollo

1.13.4.1 12.4.1 Crear Repositorios de Imágenes

```
php bin/console app:create-image-repositories
```

Crea repositorios de imágenes para desarrollo/testing.

1.13.4.2 12.4.2 Cargar Trazas de Ejemplo

```
php bin/console app:charge-example-trace
```

Carga trazas de ejemplo para testing.

1.13.5 11.5 Configuración de Cron

Agregar al crontab:

```
* * * * * docker exec ogcore-php php bin/console app:check-client-availability
* * * * * docker exec ogcore-php php bin/console app:execute-pending-traces
* * * * * docker exec ogcore-php php bin/console app:run-scheduled-command-tasks
```

1.14 12. Migraciones de Datos

1.14.1 12.1 Sistema de Migraciones de Doctrine

ogCore utiliza Doctrine Migrations para gestionar cambios en el esquema de base de datos.

1.14.1.1 13.1.1 Crear una Migración

```
docker exec ogcore-php php bin/console make:migration
```

1.14.1.2 13.1.2 Ejecutar Migraciones

```
docker exec ogcore-php php bin/console doctrine:migrations:migrate
```

1.14.1.3 13.1.3 Ver Estado de Migraciones

```
docker exec ogcore-php php bin/console doctrine:migrations:status
```

1.14.1.4 13.1.4 Revertir Migración

```
docker exec ogcore-php php bin/console doctrine:migrations:migrate prev
```

1.14.2 12.2 Migración desde OpenGnsys 1.1

Para migrar datos desde una instalación de OpenGnsys 1.1:

1.14.2.1 13.2.1 Crear Base de Datos Temporal

```
docker exec ogcore-php php bin/console doctrine:database:create --connection=og_1
```

1.14.2.2 13.2.2 Cargar Dump de OpenGnsys 1.1

```
docker exec -i ogcore-database mysql -u user -p ogcore_old_og < dump_og_1.1.sql
```

1.14.2.3 13.2.3 Ejecutar Migraciones

Ejecutar los comandos de migración en orden:

```
docker exec ogcore-php php bin/console opengnsys:migration:organizational-unit
docker exec ogcore-php php bin/console opengnsys:migration:hardware-profile
docker exec ogcore-php php bin/console opengnsys:migration:clients
docker exec ogcore-php php bin/console opengnsys:migration:os
docker exec ogcore-php php bin/console opengnsys:migration:image
docker exec ogcore-php php bin/console opengnsys:migration:software-profile
```

1.14.3 12.3 Backup y Restauración

1.14.3.1 13.3.1 Backup de Base de Datos

```
docker exec ogcore-database mysqldump -u user -p ogcore > backup_$(date +%Y%m%d).sql
```

1.14.3.2 13.3.2 Restaurar Base de Datos

```
docker exec -i ogcore-database mysql -u user -p ogcore < backup_20251008.sql
```

1.15 13. Testing

1.15.1 13.1 Framework de Testing

ogCore utiliza **PHPUnit** con integración de Symfony y Doctrine.

1.15.2 13.2 Ejecutar Tests

1.15.2.1 14.2.1 Todos los Tests

```
docker compose exec php bin/phpunit
```

1.15.2.2 14.2.2 Tests Específicos

```
docker compose exec php bin/phpunit tests/Functional/ClientTest.php
```

1.15.2.3 14.2.3 Tests con Coverage

```
docker compose exec php bin/phpunit --coverage-html coverage/
```

1.15.3 13.3 Tipos de Tests

1.15.3.1 14.3.1 Tests Funcionales

Ubicación: `tests/Functional/`

20 archivos de tests funcionales que prueban: - Endpoints de API - Flujos completos - Integraciones

Ejemplo:

```php
public function testCreateClient(): void
{
    $client = static::createClient();
    $client->request('POST', '/clients', [
        'json' => [
            'name' => 'Test Client',
            'mac' => '00:11:22:33:44:55',
            'ip' => '192.168.1.100'
        ]
    ]);

    $this->assertResponseStatusCodeSame(201);
}
```

1.15.4 13.4 Factories para Testing

Se utilizan **Zenstruck Foundry** factories (24 factories) para crear datos de prueba:

```php
ClientFactory::createOne([
    'name' => 'Test PC',
    'status' => 'active'
]);
```

1.15.5 13.5 Base de Datos de Testing

Los tests utilizan **DAMA Doctrine Test Bundle** para: - Ejecutar cada test en una transacción - Rollback automático después de cada test - Aislamiento completo entre tests

---

1.16 14. Despliegue

1.16.1 14.1 Despliegue con Docker Compose

1.16.1.1 15.1.1 Desarrollo

```bash
docker compose up -d
```

1.16.1.2 15.1.2 Producción

```bash
docker compose -f docker-compose-deploy.yml up -d
```

1.16.2 14.2 Despliegue con Jenkins

El proyecto incluye **Jenkinsfile** para CI/CD.

1.16.2.1 Pipeline stages:

1. **Checkout**: Clonar repositorio
2. **Build**: Construir imagen Docker
3. **Test**: Ejecutar tests

4. **Package**: Crear paquete Debian
5. **Deploy**: Desplegar en servidor

1.16.3 14.3 Paquete Debian

El proyecto puede empaquetarse como `.deb`:

`./package.sh`

Estructura del paquete en `debian/`: - Control files - Postinst/preinst scripts - Systemd service file - Configuración

1.16.4 14.4 Configuración de Producción

1.16.4.1 15.4.1 Variables de Entorno

Crear `.env.local` con configuración de producción:

```
APP_ENV=prod
APP_DEBUG=0
DATABASE_URL="mysql://user:pass@localhost:3306/ogcore"
# ... más configuración
```

1.16.4.2 15.4.2 Optimizaciones

```
# Limpiar caché
docker exec ogcore-php php bin/console cache:clear --env=prod

# Calentar caché
docker exec ogcore-php php bin/console cache:warmup --env=prod

# Optimizar autoloader
docker exec ogcore-php composer dump-autoload --optimize --classmap-authoritative
```

1.16.5 14.5 Monitoreo

1.16.5.1 15.5.1 Logs

Logs ubicados en `var/log/`: - `dev.log` / `prod.log`: Logs de aplicación - `nginx/access.log`: Accesos a nginx - `nginx/error.log`: Errores de nginx

1.16.5.2 15.5.2 Syslog

Los logs también se envían a syslog para centralización.

1.16.6 14.6 Escalabilidad

Para escalar horizontalmente:

1. **Base de datos**: Usar MariaDB con replicación master-slave
2. **PHP**: Aumentar réplicas de contenedor PHP
3. **Load Balancer**: Usar nginx como balanceador de carga
4. **Caché**: Implementar Redis para sesiones y caché

5. **Mercure**: Escalar hub de Mercure

1.17 15. Mantenimiento y Operaciones

1.17.1 15.1 Reiniciar Base de Datos

```
docker exec ogcore-php php bin/console doctrine:database:drop --force
docker exec ogcore-php php bin/console doctrine:database:create
docker exec ogcore-php php bin/console doctrine:migrations:migrate --no-interaction
docker exec ogcore-php php bin/console doctrine:fixtures:load --no-interaction
```

1.17.2 15.2 Limpiar Caché

```
docker exec ogcore-php php bin/console cache:clear
docker exec ogcore-php php bin/console cache:warmup
```

1.17.3 15.3 Actualizar Dependencias

```
docker exec ogcore-php composer update
```

1.17.4 15.4 Regenerar Claves JWT

```
docker exec ogcore-php php bin/console lexik:jwt:generate-keypair --overwrite
```

1.17.5 15.5 Backups Automáticos

Script ejemplo para backup automático:

```
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/backups"

# Backup de base de datos
docker exec ogcore-database mysqldump -u user -p password ogcore > $BACKUP_DIR/db_$DATE.sql

# Backup de archivos
tar -czf $BACKUP_DIR/files_$DATE.tar.gz /var/www/html/ogcore/var

# Limpiar backups antiguos (>30 días)
find $BACKUP_DIR -name "*.sql" -mtime +30 -delete
find $BACKUP_DIR -name "*.tar.gz" -mtime +30 -delete
```

1.17.6 15.6 Monitoreo de Salud

1.17.6.1 Health Check Endpoint

```
GET /health
```

1.17.6.2 Verificar Estado de Servicios

```
# PHP
docker exec ogcore-php php -v

# Nginx
docker exec ogcore-nginx nginx -t

# MariaDB
docker exec ogcore-database mysqladmin -u user -p status
```

1.17.7 15.7 Rotación de Logs

Configurar logrotate:

```
/var/www/html/ogcore/var/log/*.log {
    daily
    rotate 14
    compress
    delaycompress
    notifempty
    create 0640 www-data www-data
    sharedscripts
}
```

1.18 16. Integración con Otros Servicios

1.18.1 16.1 ogRepository

Propósito: Gestión de repositorios de imágenes.

Endpoints integrados: - Crear imagen - Desplegar imagen - Backup imagen - Eliminar imagen - Convertir a imagen virtual - Importar imagen externa - Verificar integridad - Imagen global

Webhook: `/og-repository/webhook`

1.18.2 16.2 ogDhcp

Propósito: Gestión de DHCP dinámico.

Funcionalidades: - Agregar clientes a DHCP - Eliminar clientes de DHCP - Actualizar configuración - Gestión de subredes

1.18.3 16.3 ogBoot

Propósito: Gestión de archivos de arranque PXE.

Funcionalidades: - Crear archivos de arranque - Actualizar plantillas PXE - Eliminar configuraciones - Sincronización de menús

1.18.4 16.4 ogAgent

Propósito: Agente instalado en clientes para ejecutar comandos.

Operaciones: - Power on/off/reboot - Crear/desplegar imágenes - Particionar discos - Ejecutar scripts - Obtener inventario hardware - Verificar tamaño de particiones - Kill jobs

1.18.5 16.5 UDS (Universal Desktop Services)

Propósito: Integración con sistema de escritorios remotos.

Funcionalidades: - Autenticación - Obtener service pools - Calcular disponibilidad - Gestión de reservas

1.18.6 16.6 Git (Versionado de Imágenes)

Propósito: Versionado de imágenes con Git.

Funcionalidades: - Backup a repositorio Git - Versionado automático - Recuperación de versiones - Sincronización

1.18.7 16.7 Mercure (Notificaciones en Tiempo Real)

Propósito: Notificaciones push en tiempo real.

Eventos publicados: - Cambio de estado de clientes - Actualización de trazas - Cambios en comandos - Alertas y notificaciones

Suscripción del cliente:

```
const eventSource = new EventSource('http://localhost:8080/.well-known/mercure?topic=/clients/{
eventSource.onmessage = event => {
    const data = JSON.parse(event.data);
    console.log('Client updated:', data);
};
```

1.19 17. Roadmap y Changelog

1.19.1 17.1 Versión Actual: 0.5.0

1.19.2 17.2 Últimas Características (v0.25.1 - Octubre 2025)

- **Tareas programadas**: Sistema completo de colas y scheduling
- **Inventario hardware**: Obtención automática de inventario
- **Clonado de imágenes**: Mejoras en el servicio de clonado
- **Repositorios Git**: Backup y gestión de repositorios
- **Particionado**: Integración con agente para scripts de particionado
- **Gestión de trazas**: Marcar como completadas, cancelación
- **Estados de cliente**: Nuevo estado “enviado” para Windows/Linux
- **Validación de particiones**: Comprobación de tamaños
- **Imagen versionada**: Sistema de versionado de imágenes

- **Calendario remoto**: Integración con UDS
- **Mercure**: Notificaciones en tiempo real
- **Despliegue multicast**: Modos torrent y UDPCAST

1.19.3 17.3 Próximas Características (Roadmap)

1.19.3.1 v0.6.0 (Q4 2025)

- ☐ Dashboard de administración mejorado
- ☐ Reportes y estadísticas avanzadas
- ☐ API GraphQL complementaria
- ☐ Mejoras en rendimiento de queries

1.19.3.2 v0.7.0 (Q1 2026)

- ☐ Soporte para múltiples lenguajes (i18n completo)
- ☐ Sistema de plugins
- ☐ Mejoras en clustering
- ☐ API de webhooks salientes

1.19.3.3 v1.0.0 (Q2 2026)

- ☐ Versión estable de producción
- ☐ Documentación completa de API
- ☐ Guías de migración finalizadas
- ☐ Certificación de seguridad

1.19.4 17.4 Changelog Resumido

Ver `CHANGELOG.md` para el historial completo de cambios.

Versiones destacadas:

- **0.25.1** (2025-10-01): Correcciones en tareas programadas
- **0.25.0** (2025-09-23): Inventario hardware
- **0.24.0** (2025-09-09): Servicio de eliminación de repositorios Git
- **0.23.0** (2025-09-08): Backups a repositorios Git
- **0.20.0** (2025-08-25): Sistema de colas y tareas
- **0.15.0** (2025-06-26): Integración ogGit y sistema de cola
- **0.14.0** (2025-06-02): Mover equipos, imagen cache, inicio sesión
- **0.13.0** (2025-05-20): Comunicación TLS con agente
- **0.12.0** (2025-05-13): Tareas programadas, integración ogGit
- **0.11.0** (2025-04-11): Versionado de imágenes, ejecución de scripts
- **0.10.0** (2025-03-25): Imagen virtual, importar imágenes
- **0.9.0** (2025-03-04): Mercure, notificaciones en tiempo real
- **0.8.0** (2025-01-10): Imagen global, jerarquía de aulas
- **0.7.3** (2025-01-03): Multiselección, import/export, torrent/udpcast

---

1.20 18. Contribución

1.20.1 18.1 Guía de Contribución

Para contribuir al proyecto:

1. **Fork** el repositorio
2. Crea una **rama** para tu feature: `git checkout -b feature/nueva-funcionalidad`
3. **Commit** tus cambios: `git commit -am 'Añade nueva funcionalidad'`
4. **Push** a la rama: `git push origin feature/nueva-funcionalidad`
5. Crea un **Pull Request**

1.20.2 18.2 Estándares de Código

- Seguir **PSR-12** para PHP
- Usar **Type Hints** en PHP 8.3
- Documentar con **PHPDoc**
- Tests para nuevas funcionalidades
- Commits descriptivos en español

1.20.3 18.3 Proceso de Revisión

1. CI/CD ejecuta tests automáticamente
2. Revisión de código por maintainers
3. Aprobación requerida antes de merge
4. Merge a rama develop
5. Release periódicos a main

1.20.4 18.4 Reportar Bugs

Usar el sistema de Issues del repositorio:

Template de Bug:

```
**Descripción del bug**
Descripción clara y concisa del bug.

**Pasos para reproducir**
1. Ir a '...'
2. Hacer clic en '...'
3. Ver error

**Comportamiento esperado**
Lo que esperabas que sucediera.

**Screenshots**
Si aplica, añadir screenshots.

**Entorno**
- OS: [ej. Ubuntu 22.04]
- Versión: [ej. 0.5.0]
```

- Browser: [ej. Firefox 118]

1.21 19. Soporte y Contacto

1.21.1 19.1 Documentación Adicional

- **API Docs**: http://localhost:8080/docs
- **Postman Collection**: `swagger-assets/ogCore.postman_collection.zip`
- **Diagramas**: `swagger-assets/img_bbdd.png`

1.21.2 19.2 Recursos

- **Repositorio**: [URL del repositorio]
- **Wiki**: [URL de la wiki]
- **Issues**: [URL de issues]

1.21.3 19.3 Equipo de Desarrollo

Proyecto desarrollado por el equipo de OpenGnsys en colaboración con universidades participantes.

1.21.4 19.4 Licencia

Proyecto propietario. Ver archivo `LICENSE` para más información.

1.22 Apéndices

1.22.1 Apéndice A: Glosario de Términos

- **Cliente**: Equipo físico gestionado por el sistema
- **Imagen**: Copia bit a bit de una partición de disco
- **Traza**: Registro de ejecución de un comando
- **Unidad Organizativa**: Agrupación lógica de clientes (aula, grupo)
- **PXE**: Preboot Execution Environment, arranque por red
- **ogLive**: Imagen Live de arranque de OpenGnsys
- **Partición**: División lógica de un disco duro
- **Repositorio**: Servidor de almacenamiento de imágenes

1.22.2 Apéndice B: Puertos y Servicios

Puerto	Servicio	Protocolo	Descripción
8080	nginx	HTTP	API y docs
3306	MariaDB	MySQL	Base de datos
9000	PHP-FPM	FastCGI	Procesador PHP
3000	Mercure	HTTP/SSE	Notificaciones

1.22.3 Apéndice C: Estructura de Directorios

```
ogcore/
|-- bin/              # Ejecutables (console, phpunit)
|-- config/           # Configuración de Symfony
|   |-- api_platform/ # Configuración de entidades API
|   |-- packages/     # Configuración de bundles
|   +-- routes/       # Rutas
|-- migrations/       # Migraciones de base de datos
|-- public/           # Punto de entrada web
|-- src/              # Código fuente
|   |-- Command/      # Comandos de consola
|   |-- Controller/   # Controladores
|   |-- Dto/          # Data Transfer Objects
|   |-- Entity/       # Entidades Doctrine
|   |-- EventListener/# Event Listeners
|   |-- EventSubscriber/ # Event Subscribers
|   |-- Factory/      # Factories para testing
|   |-- Filter/       # Filtros de API Platform
|   |-- Repository/   # Repositorios Doctrine
|   |-- Security/     # Seguridad y autenticación
|   |-- Service/      # Servicios de negocio
|   |-- State/        # States de API Platform
|   +-- Validator/    # Validadores personalizados
|-- tests/            # Tests
|-- translations/     # Traducciones
|-- var/              # Archivos variables (cache, logs)
+-- vendor/           # Dependencias
```

1.22.4 Apéndice D: Variables de Entorno Completas

Ver archivo `.env` para todas las variables de entorno disponibles.

Fecha de actualización: Octubre 2025
Versión del documento: 1.0
Mantenedor: Equipo OpenGnsys