

DESARROLLO WEB EN ENTORNO SERVIDOR

# TEMA 8

## Doctrine, Formularios, Seguridad y Herencia

Dominando la gestión de datos, la interacción con el usuario y la protección de aplicaciones en **Symfony 8**.

Manuel Prieto Macias

Curso 2025/2026



# Conceptos Fundamentales

## Entity (Entidad)

Una clase PHP simple (POJO) que representa una tabla en la base de datos. Cada propiedad mapea a una columna.

## Repository (Repositorio)

Clase encargada de recuperar entidades desde la base de datos. Contiene métodos como `find()`, `findAll()` o consultas personalizadas.

## EntityManager

El "jefe". Gestiona el ciclo de vida de las entidades (persistir, eliminar) y sincroniza los cambios con la BD (`flush()`).

### OBJECT-RELATIONAL MAPPING


**class Product**

`$id`  
`$name`  
`$price`



**table products**

`id (INT)`  
`name (VARCHAR)`  
`price (DECIMAL)`

 Doctrine traduce tus objetos PHP a consultas SQL automáticamente, permitiéndote trabajar con POO en lugar de tablas.

# Entidades con Atributos PHP 8

Desde Symfony 6, las **Anotaciones** han sido reemplazadas por **Atributos nativos de PHP 8**. Son más limpios, rápidos y el estándar actual.



## **#[ORM\Entity]**

Marca la clase como una entidad de base de datos.



## **#[ORM\Id]**

Define la clave primaria de la tabla.



## **#[ORM\Column]**

Configura el tipo de columna (string, integer, etc.) y sus opciones.

```
namespace App\Entity;

use App\Repository\ProductRepository;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: ProductRepository::class)]
class Product
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 255)]
    private ?string $name = null;

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

    #[ORM\Column(type: 'text', nullable: true)]
    private ?string $description = null;

    // Getters y Setters...
```

# Creando Entidades Interactivamente

```
user@symfony:~$ symfony console make:entity Product
```

```
created: src/Entity/Product.php
```

```
created: src/Repository/ProductRepository.php
```

Entity generated! Now let's add some fields!

You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):

```
> name
```

Field type (enter ? to see all types) [string]:

```
> string
```

Field length [255]:

```
> 255
```

Can this field be null in the database (nullable) (yes/no) [no]:

```
> no
```

# ManyToOne y OneToMany

## Product.php

OWNING SIDE (FK)

```
class Product
{
    // ... id, name ...

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

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

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

1  N

## Category.php

INVERSE SIDE

```
class Category
{
    // ...

    #[ORM\OneToMany(mappedBy: 'category', targetEntity: Product::class)]
    private Collection $products;

    public function __construct()
    {
        $this->products = new ArrayCollection();
    }

    public function getProducts(): Collection
    {
        return $this->products;
    }
}
```

# ManyToMany (N:M)

## User.php (Owning Side)

```
class User
{
    #[ORM\ManyToMany(targetEntity: Group::class, inversedBy: 'users')]
    #[ORM\JoinTable(name: 'users_groups')]
    private Collection $groups;

    public function __construct() {
        $this->groups = new ArrayCollection();
    }

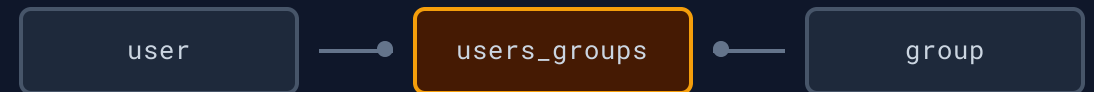
    public function addGroup(Group $group): static {
        if (!$this->groups->contains($group)) {
            $this->groups->add($group);
        }
    }
}
```

## Group.php (Inverse Side)

```
class Group
{
    #[ORM\ManyToMany(targetEntity: User::class, mappedBy: 'users')]
    private Collection $users;

    // ... constructor & getters ...
}
```

### ESTRUCTURA EN BASE DE DATOS



Doctrine crea automáticamente la **tabla intermedia** (users\_groups) con las claves foráneas de ambas entidades.

# Repositorios



## Acceso a Datos

El repositorio es la clase encargada de recuperar objetos desde la base de datos. Cada Entidad tiene su propio Repositorio.



## Métodos Mágicos

Doctrine genera métodos automáticos basados en tus campos: `findOneByEmail()`, `findByStatus()`.



## Personalización

Puedes crear métodos propios en el repositorio usando DQL o QueryBuilder para consultas complejas.

ProductController.php

Uso Básico

```
public function index($repository): Response
{
    // 1. Buscar por ID (retorna ?Product)
    $product = $repository->find($id);

    // 2. Buscar todos (retorna array)
    $allProducts = $repository->findAll();

    // 3. Búsqueda con criterios (WHERE, ORDER, LIMIT)
    $cheapProducts = $repository->findBy(
        ['category' => 'electronics'], // Criteria
        ['price' => 'ASC'],             // Order
        10                             // Limit
    );

    // 4. Búsqueda única (retorna ?Product)
    $one = $repository->findOneBy(['slug' => 'iphone-15']);
}
```

# Consultas Avanzadas

## DQL (DOCTRINE QUERY LANGUAGE)

### Consultas tipo SQL

Lenguaje similar a SQL pero orientado a objetos. Consultas **Entidades** en lugar de tablas. Ideal para consultas estáticas y legibles.

```
$query = $entityManager->createQuery(
    'SELECT p
     FROM App\Entity\Product p
     WHERE p.price > :price
     ORDER BY p.name ASC'
)->setParameter('price', $price);

$products = $query->getResult();
```

## QUERYBUILDER

### Construcción Programática

Interfaz fluida para construir consultas dinámicamente. Es la forma preferida dentro de los **Repositorios**.

```
// Dentro de ProductRepository
$qb = $this->createQueryBuilder('p')
    ->where('p.price > :price')
    ->setParameter('price', $price)
    ->orderBy('p.name', 'ASC');

$query = $qb->getQuery();
$products = $query->getResult();
```

# Ciclo de Vida y Eventos



## PrePersist

Se ejecuta antes de que el objeto se guarde por primera vez en la BD. Ideal para createdAt.



## PreUpdate

Se ejecuta justo antes de actualizar un registro existente. Ideal para updatedAt.



## PreRemove / PostRemove

Hooks para ejecutar lógica antes o después de eliminar una entidad (ej: borrar archivos).

Product.php

LifecycleCallbacks

```
#[ORM\Entity]
#[ORM\HasLifecycleCallbacks] // ¡Importante!
class Product
{
    #[ORM\Column(type: 'datetime')]
    private $createdAt;

    #[ORM\Column(type: 'datetime')]
    private $updatedAt;

    #[ORM\PrePersist]
    public function setCreatedAtValue(): void
    {
        $this->createdAt = new \DateTime();
        $this->updatedAt = new \DateTime();
    }

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

# Fixtures: Datos de Prueba

Las **Fixtures** son esenciales para el desarrollo. Permiten poblar la base de datos con datos falsos (dummy data) para probar la aplicación sin tener que introducir registros manualmente.



## DoctrineFixturesBundle



```
# 1. Crear una nueva clase de Fixture
$ php bin/console make:fixtures

# 2. Cargar los datos (;Borra la BD!)
$ php bin/console doctrine:fixtures:load
```

```
namespace App\DataFixtures;

use App\Entity\Product;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        // Crear 20 productos de prueba
        for ($i = 0; $i < 20; $i++) {
            $product = new Product();
            $product->setName('Producto ' . $i);
            $product->setPrice(mt_rand(10, 100));

            // Persistir (encolar)
            $manager->persist($product);
        }

        // Ejecutar consultas SQL
        $manager->flush();
    }
}
```

## COMPONENTES

# El Componente Form

Symfony abstrae la complejidad de los formularios HTML, gestionando la validación y el mapeo de datos automáticamente.



### Data Binding

Sincroniza automáticamente los datos del formulario con las propiedades de tu Entidad (y viceversa).



### Seguridad Integrada

Protección CSRF automática y saneamiento de datos por defecto.



### Renderizado Flexible

Genera el HTML completo con una sola línea en Twig o personaliza cada campo individualmente.

```
$ symfony console make:form
```

```
class ProductType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name')
            ->add('price')
            ->add('description')
            ->add('category', EntityType::class, [
                'class' => Category::class,
                'choice_label' => 'name',
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Product::class,
        ]);
    }
}
```

# Tipos de Campos (Form Types)

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('title', TextType::class)

        ->add('content', TextareaType::class, [
            'attr' => ['rows' => 5],
        ])

        ->add('publishedAt', DateTimeType::class, [
            'widget' => 'single_text',
        ])

        ->add('category', EntityType::class, [
            'class' => Category::class,
            'choice_label' => 'name',
        ])

        ->add('save', SubmitType::class, [
            'label' => 'Guardar Post'
        ]);
}
```

## A Texto

- TextType
- TextareaType
- EmailType
- PasswordType

## ☰ Selección

- ChoiceType
- **EntityType (Doctrine)**
- CheckboxType
- RadioType

## 📅 Fecha/Hora

- DateType
- DateTimeType
- TimeType
- BirthdayType

## # Números/Otros

- IntegerType
- MoneyType
- FileType
- SubmitType

# Generando Formularios (make:form)

```
$ symfony console make:form
```

The name of the form class (e.g. TinyPuppyType):

```
> ProductType
```

The name of Entity or fully qualified model class name

```
> Product
```

```
created: src/Form/ProductType.php
```

```
class ProductType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name')
            ->add('price')
            ->add('description')
        ;
    }


    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Product::class,
        ]);
    }
}
```

# Validación de Datos


## Lógica Centralizada


En Symfony, las reglas de validación se definen directamente en la **Entidad** (o DTO), no en el controlador ni en el formulario.

## Constraints Comunes

 NotBlank

 Length

 Email

 Positive

 Date

 Regex

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
// Importante: Alias 'Assert' es el estándar
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity]
class Product
{
    #[ORM\Column(length: 255)]
    #[Assert\NotBlank(message: "El nombre es obligatorio")]
    #[Assert\Length(min: 3, max: 255)]
    private ?string $name = null;

    #[ORM\Column]
    #[Assert\NotBlank]
    #[Assert\Positive(message: "El precio debe ser mayor a 0")]
    private ?float $price = null;

    #[ORM\Column(length: 255)]
    #[Assert>Email]
    private ?string $supplierEmail = null;
```

# El Flujo de Procesamiento

1

## Crear Formulario

Instanciar el form type asociado a la entidad.

2

## Handle Request

Procesar los datos POST y mapearlos a la entidad.

3

## Validar

Verificar si fue enviado y si cumple las constraints.

4

## Persistir

Guardar en BD si todo es correcto.

5

## Redirigir

```
#[Route('/product/new', name: 'app_product_new')]
public function new(Request $request, EntityManagerInterface $em): Response
{
    $product = new Product();

    // 1. Crear el formulario
    $form = $this->createForm(ProductType::class, $product);

    // 2. Procesar la petición (¡Magia!)
    $form->handleRequest($request);

    // 3. Verificar envío y validez
    if ($form->isSubmitted() && $form->isValid()) {

        // 4. Guardar en base de datos
        $em->persist($product);
        $em->flush();

        // 5. Redirigir (Post-Redirect-Get)
        return $this->redirectToRoute('app_product_index');
    }

    // Renderizar formulario (GET o POST con errores)
```

# CRUD Automático (make:crud)

```
$ symfony console make:crud Product
```



**i** Genera un CRUD completo y funcional. ¡Ideal para prototipar o paneles de administración!

## FORMULARIOS

# Personalización (Form Theming)

## Renderizado en Twig

Evita usar `{{ form(form) }}`. Para mayor control, renderiza fila por fila.

```
{{ form_start(form) }}

<div class="my-4">
    {{ form_row(form.name, {
        'label': 'Nombre del Producto',
        'attr': {'class': 'bg-gray-100'}
    }) }}
</div>

{{ form_row(form.category) }}

<button type="submit">Guardar</button>

{{ form_end(form) }}
```

## Configuración Global

Aplica estilos de Tailwind o Bootstrap a **todos** los formularios automáticamente.

```
# config/packages/twig.yaml

twig:
    form_themes:
        # Elige uno:
        # - 'bootstrap_5_layout.html.twig'
        # - 'foundation_6_layout.html.twig'

        # Para Tailwind CSS:
        - 'tailwind_2_layout.html.twig'

        # O tu propio tema personalizado:
        - 'form/my_theme.html.twig'
```

# Conceptos Fundamentales



## Autenticación

¿Quién eres?

El proceso de verificar la identidad de un usuario (Login).

En Symfony: **Firewalls**, **User Providers** y **Authenticators**.



## Autorización

¿Qué puedes hacer?

El proceso de verificar si el usuario tiene permiso para acceder a un recurso.

En Symfony: **Roles** (ROLE\_ADMIN), **Voters** y **Access Control**.

FLUJO DE PETICIÓN:

Request



Firewall (AuthN)



Access Control (AuthZ)



Controller

# La Entidad User

Para que Symfony pueda autenticar usuarios, tu entidad debe implementar interfaces específicas.

## **UserInterface**

Obligatoria. Define quién es el usuario y qué permisos tiene.

`getRoles()``eraseCredentials()``getUserIdentifier()`

## **PasswordAuthenticated...**

Necesaria si usas contraseñas. Permite al sistema obtener el hash.

```
namespace App\Entity;

use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 180, unique: true)]
    private ?string $email = null;

    #[ORM\Column]
    private array $roles = [];

    #[ORM\Column]
    private ?string $password = null;
```

# Seguridad en Dos Pasos

```
$ make:user
```

## Crear la Entidad de Usuario

Genera la clase que implementa `UserInterface` y `PasswordAuthenticatedUserInterface`.

- ✓ Crea `User.php` (Entidad)
- ✓ Crea `UserRepository.php`
- ⚙ Configura el encoder de contraseñas en `security.yaml`

```
$ make:auth
```

## Sistema de Login

Crea el formulario de login, el controlador y configura el firewall automáticamente.

- 🛡 Genera el **Authenticator** (Guard)
- </> Crea `SecurityController` y templates
- 🔥 Actualiza el **firewall** en `security.yaml`

# Configuración: security.yaml

```
security:
    # 1. Algoritmos de hash
    password_hashers:
        App\Entity\User: 'auto'

    # 2. Proveedores de usuarios (¿De dónde salen?)
    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email

    # 3. Firewalls (Autenticación)
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: app_user_provider
            form_login:
                login_path: app_login
                check_path: app_login
            logout:
                path: app_logout

    # 4. Control de Acceso (Autorización por URL)
```



## Password Hashers

Define cómo se cifran las contraseñas. 'auto' selecciona el mejor algoritmo disponible (Bcrypt/Argon2).



## Providers

Indica a Symfony cómo cargar usuarios. Generalmente desde una Entidad Doctrine buscando por email o username.



## Firewalls

El núcleo de la autenticación. Define zonas (dev, main) y mecanismos (form\_login, json\_login, logout).



## Access Control

Reglas globales de autorización basadas en patrones de URL. Es la primera línea de defensa.

# Autenticación y Login

config/packages/security.yaml

```
security:
    firewalls:
        main:
            # ...
            form_login:
                login_path: app_login
                check_path: app_login
                enable_csrf: true

            logout:
                path: app_logout
                # where to redirect after logout
                target: app_home
```

**i** Symfony intercepta automáticamente las peticiones a `check_path` y `logout`. No necesitas escribir lógica para verificar contraseñas o destruir la sesión manualmente.

src/Controller/SecurityController.php

```
class SecurityController extends AbstractController
{
    #[Route(path: '/login', name: 'app_login')]
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        // Obtener error de login si lo hay
        $error = $authenticationUtils->getLastAuthenticationError();

        // Último nombre de usuario introducido
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('security/login.html.twig', [
            'last_username' => $lastUsername,
            'error' => $error
        ]);
    }

    #[Route(path: '/logout', name: 'app_logout')]
    public function logout(): void
    {
        // Este método puede estar vacío.
        // La key 'logout' del firewall lo intercepta.
        throw new \LogicException('This method can be blank.');
```

# Roles y Jerarquía

ROLE\_SUPER\_ADMIN



ROLE\_ADMIN



ROLE\_USER

Un usuario con **ROLE\_ADMIN** tiene automáticamente todos los permisos de **ROLE\_USER**.

```
security:
  role_hierarchy:
    # El Admin hereda del User
    ROLE_ADMIN: [ROLE_USER]

    # El Super Admin hereda del Admin (y por tanto del User)
    # y además puede suplantar identidad
    ROLE_SUPER_ADMIN: [ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]
```

## Reglas de Roles

- Deben empezar siempre por ROLE\_.
- Son simples cadenas de texto (Strings).
- Se asignan en la base de datos (columna JSON).
- ROLE\_USER suele ser el rol base por defecto.

# Control de Acceso (Access Control)

```
security:
  # ...
  access_control:
    # 1. Login es público (¡Importante!)
    - { path: ^/login, roles: PUBLIC_ACCESS }

    # 2. Panel de administración solo para admins
    - { path: ^/admin, roles: ROLE_ADMIN }

    # 3. Perfil de usuario para cualquier logueado
    - { path: ^/profile, roles: ROLE_USER }

    # 4. API protegida
    - { path: ^/api, roles: IS_AUTHENTICATED_FULLY }
```

## Reglas de Coincidencia

- 1 Symfony comprueba la URL de la petición contra el **path** (Regex).
- 2 Se detiene en la **PRIMERA** coincidencia encontrada (de arriba a abajo).
- 3 Si coincide, verifica si el usuario tiene el **rol** requerido.
- 4 Si no tiene el rol, lanza **AccessDeniedException** (403).

**⚠ ¡Cuidado con el orden!** Si pones una regla genérica (ej: ^/) al principio, bloquearás o permitirás todo antes de llegar a las reglas específicas.

# Voters vs Roles

## Roles Simples

- ✓ Estáticos y globales (definidos en BD o código).
- ✓ No dependen del objeto específico que se intenta acceder.
- ✓ Ideal para permisos generales de la aplicación.

```
is_granted('ROLE_ADMIN')  
// ¿Es administrador? SÍ/NO
```

**VS**

## Voters (Lógica)

- ✓ Dinámicos y contextuales.
- ✓ Evalúan reglas de negocio complejas sobre un objeto específico.
- ✓ Permiten lógica granular ("Dueño", "Creador", "Horario").

```
is_granted('POST_EDIT', $post)  
// ¿Es el dueño de ESTE post?
```

# Implementando un Voter

## supports()

Filtra si este Voter debe actuar. Verifica el atributo (ej. 'POST\_EDIT') y el tipo de objeto (ej. Post).

## voteOnAttribute()

Contiene la lógica de negocio real. Devuelve true si se concede el acceso, false si no.

```
>_ symfony console make:voter
```

```
class PostVoter extends Voter
{
    const EDIT = 'POST_EDIT';

    protected function supports(string $attribute, mixed $subject): bool
    {
        // Solo votar si es POST_EDIT y el sujeto es un Post
        return $attribute === self::EDIT
            && $subject instanceof Post;
    }

    protected function voteOnAttribute(string $attribute, mixed $subject, TokenInterface $token): bool
    {
        $user = $token->getUser();

        // Si no está logueado, denegar
        if (!$user instanceof User) {
            return false;
        }

        // Lógica: ¿Es el dueño del post?
        // $subject es el objeto Post
        return $subject->getAuthor() === $user;
    }
}
```

# Esqueleto de Seguridad (make:voter)

```
>_ $ symfony console make:voter PostVoter
```

```
class PostVoter extends Voter
{
    protected function supports(string $attribute, $subject): bool
    {
        // replace with your own logic
        return in_array($attribute, ['POST_EDIT', 'POST_VIEW'])
            && $subject instanceof \App\Entity\Post;
    }

    protected function voteOnAttribute(string $attribute, $subject, TokenInterface $token): bool
    {
        $user = $token->getUser();
        // anonymous users are denied
        if (!$user instanceof UserInterface) {
            return false;
        }

        switch ($attribute) {
            case 'POST_EDIT':
                // check EDIT permission
                break;
            case 'POST_VIEW':
                // check VIEW permission

```

# Usando Voters

## </> En el Controlador

```
// Opción A: Método Imperativo (Clásico)
public function edit(Post $post): Response
{
    // Lanza excepción 403 si devuelve false
    $this->denyAccessUnlessGranted('POST_EDIT', $post);

    // ... lógica de edición
}
```


```
// Opción B: Atributo (Moderno y Limpio)
#[Route('/post/{id}/edit')]
#[IsGranted('POST_EDIT', subject: 'post')]
public function edit(Post $post): Response
{
    // El código aquí solo se ejecuta si el Voter aprueba
}
```


## 🛡 En la Vista (Twig)

```
{% if is_granted('POST_EDIT', post) %}

    <a href="...">
        <i class="fas fa-edit"></i> Editar Post
    </a>

{% endif %}
```

 **Automático:** Symfony inyecta automáticamente el usuario actual en el Voter. No necesitas pasarlo manualmente.

 **Seguridad en Capas:** Usa `is_granted` en Twig para UX (ocultar botones), pero SIEMPRE protege el controlador para seguridad real.

# Estrategias de Herencia

## Single Table Inheritance

Todas las clases de la jerarquía se guardan en **una sola tabla** gigante.

| person |      |
|--------|------|
| id     | INT  |
| type   | STR  |
| name   | STR  |
| salary | NULL |
| level  | NULL |

- ✓ Muy rápido (sin JOINS).
- ✓ Simple de consultar.
- ✗ Muchos valores NULL.
- ✗ No normalizado.

RECOMENDADO

## Class Table Inheritance (Joined)

Cada clase tiene **su propia tabla**. Se unen mediante JOINS automáticos.

| person |     |
|--------|-----|
| id     | PK  |
| name   | STR |

| employee |    |
|----------|----|
| id       | FK |
| salary   |    |

| student |    |
|---------|----|
| id      | FK |
| level   |    |

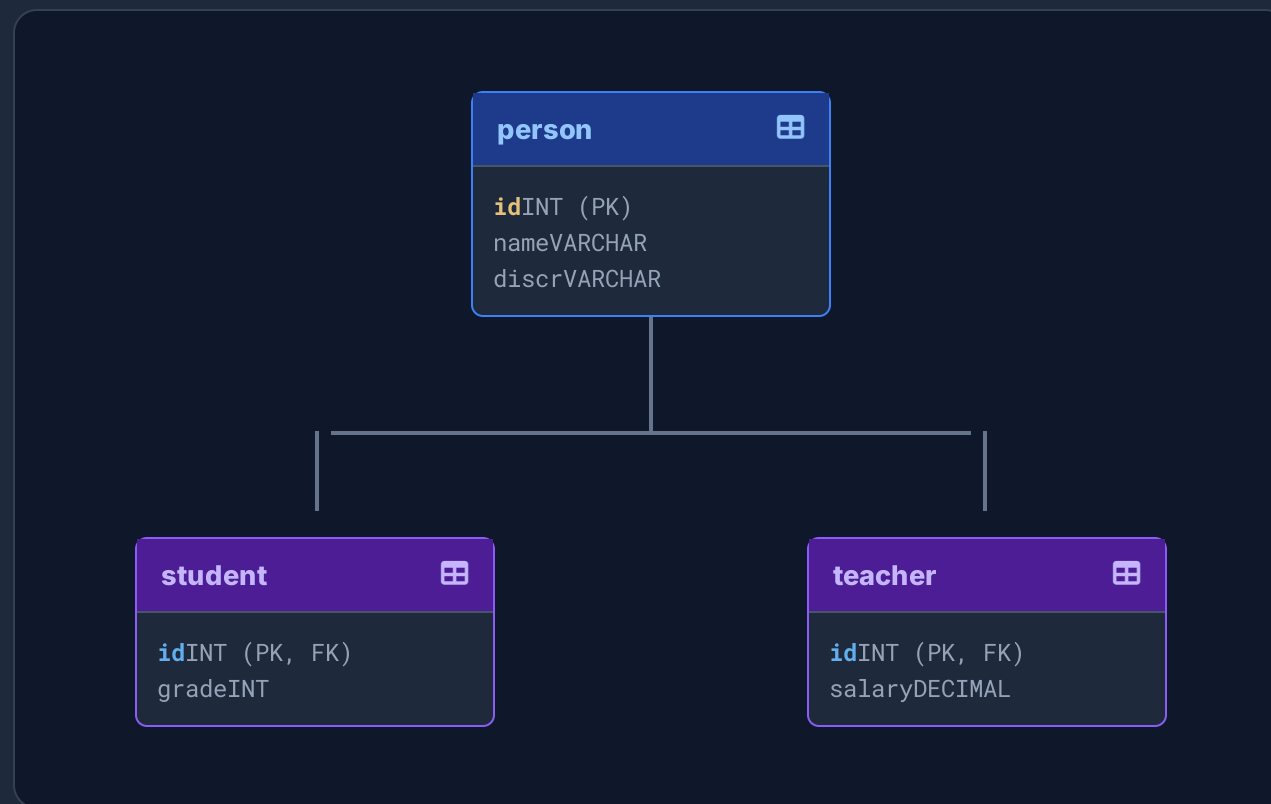
- ✓ Base de datos normalizada.
- ✓ Sin columnas NULL innecesarias.
- ✓ Estructura limpia y lógica.
- ⚠ Requiere JOINS (ligero impacto).

# Class Table Inheritance (Joined)

Cada clase en la jerarquía se mapea a su **propia tabla** en la base de datos. Las tablas hijas están vinculadas a la tabla padre mediante una Foreign Key.

## Análisis

- ✓ **Normalización:** Estructura de BD limpia y canónica.
- ✓ **Espacio:** Sin columnas NULL innecesarias.
- ✗ **Rendimiento:** Requiere operaciones JOIN para hidratar las entidades, lo que puede ser más lento.



## Ejemplo Práctico (Parte 2)

Student.php

```
#[ORM\Entity]
class Student extends User
{
    #[ORM\Column(type: 'float')]
    private ?float $averageGrade = null;

    // Getters & Setters...
}
```

Teacher.php

```
#[ORM\Entity]
class Teacher extends User
{
    #[ORM\Column(type: 'decimal', scale: 2)]
    private ?string $salary = null;

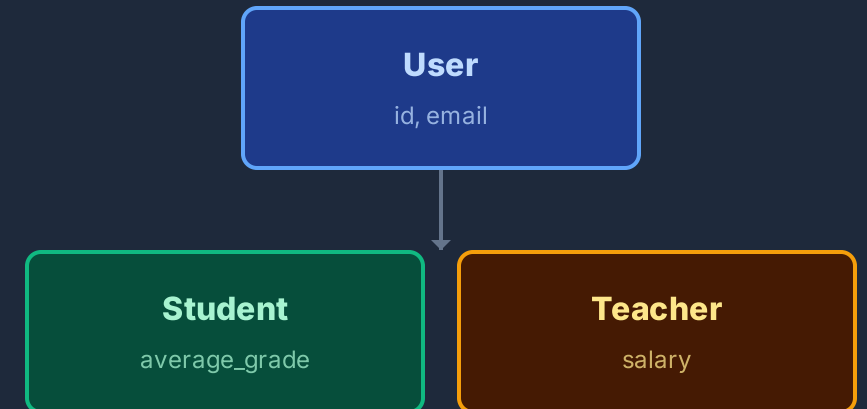
    // Getters & Setters...
}
```

# Ejemplo Práctico (Parte 1)

User.php (Clase Padre)

```
#[ORM\Entity]
#[ORM\InheritanceType('JOINED')]
#[ORM\DiscriminatorColumn(name: 'discr', type: 'string')]
#[ORM\DiscriminatorMap([
    'user' => User::class,
    'student' => Student::class,
    'teacher' => Teacher::class
])]
class User
{
    #[ORM\Id, ORM\GeneratedValue, ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 180)]
    private ?string $email = null;
}
```



## HERRAMIENTAS

# Maker Bundle: Tu Mejor Amigo

### Core & Datos

#### **make:entity**

Crea una nueva entidad o añade campos a una existente.

#### **make:controller**

Genera un controlador vacío con una ruta y template Twig.

#### **make:form**

Crea una clase FormType basada en una Entidad existente.

#### **make:crud**

Genera controlador, formulario y vistas para un CRUD completo.

### Seguridad & Lógica

#### **make:user**

Configura la entidad User y security.yaml automáticamente.

#### **make:auth**

Crea un sistema de login (Form Login o Guard Authenticator).

#### **make:voter**

Genera la estructura de una clase Voter de seguridad.

#### **make:fixtures**

Crea una clase para cargar datos de prueba.



### Modo Interactivo

No necesitas memorizar los argumentos. Simplemente ejecuta el comando (ej: `symfony console make:entity`) y Maker te hará preguntas paso a paso.

# Buenas Prácticas



## Validación en Entidades

Centraliza las reglas (Constraints) en la Entidad, no en el Controlador ni en el FormType.



## Lógica en Controladores

Evita "Fat Controllers". Mueve la lógica de negocio a Servicios o al Modelo.



## Inyección de Dependencias

Usa inyección por constructor. Evita llamar al contenedor de servicios directamente.



## Consultas en Bucles

Cuidado con el problema N+1. Usa JOIN en DQL para traer relaciones (Eager Loading).

# Preparación para el Seminario

---

1

## Revisar el PDF del Seminario

Descarga el documento "Seminario 1: Blog Tecnológico" del campus virtual.

2

## Entorno Docker Listo

Asegúrate de tener los contenedores levantados y funcionando (Nginx, PHP, MySQL).

3

## Investigación Previa

Lee sobre `make:entity` y relaciones OneToMany en la documentación oficial.



Escanea para acceder  
a la documentación