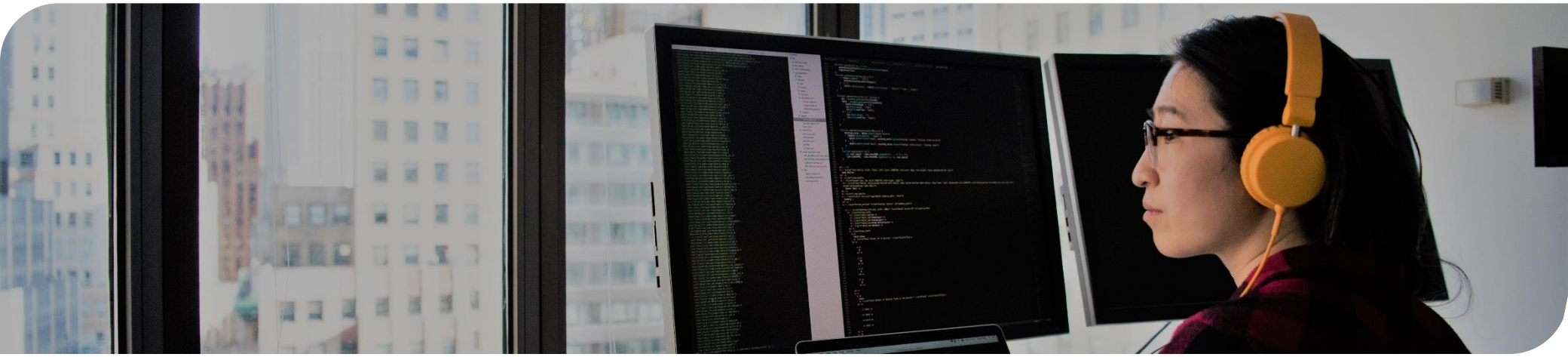
JUNIORISEP



Formation Symfony 5.0

Cours 2

Lundi 2 décembre | Pierre MARQUET & Florent FAVOLE & Laurianne PROGENT

Symfony

Récapitulatif cours 1

Vu la semaine dernière :

- **Modèle MVC :** modèles, vues, contrôleurs...
- **POO :** classe, objets, attributs, méthodes…
- **Bases de Symfony :** entités, routes, services (bundles)…
- **Les outils :** Composer, TWIG…
- **TP :** création de pages, routing, etc.

Des questions ?

1 Entités et Repository

Les entités

Entité : simple objet que l'**ORM** (Object-Relation Mapper) va manipuler et enregistrer dans la **base de données**. On dit « **persister** » une entité, plutôt que « enregistrer » une entité dans la BDD.

→ **Sur un projet :** il faut modéliser la **structure** de la base de données UML en avance, c'est-à-dire les entités (méthodes et attributs) et leurs relations.

Les entités se trouvent dans /src/Entity/

Les entités

→ Exemple d'UML

Ici nous avons 3 entités :

- **Gestionnaire** (avec 2 attributs et 2 méthodes)
- **Équipe** (avec 1 attribut)
- **Projet** (avec 3 attributs)

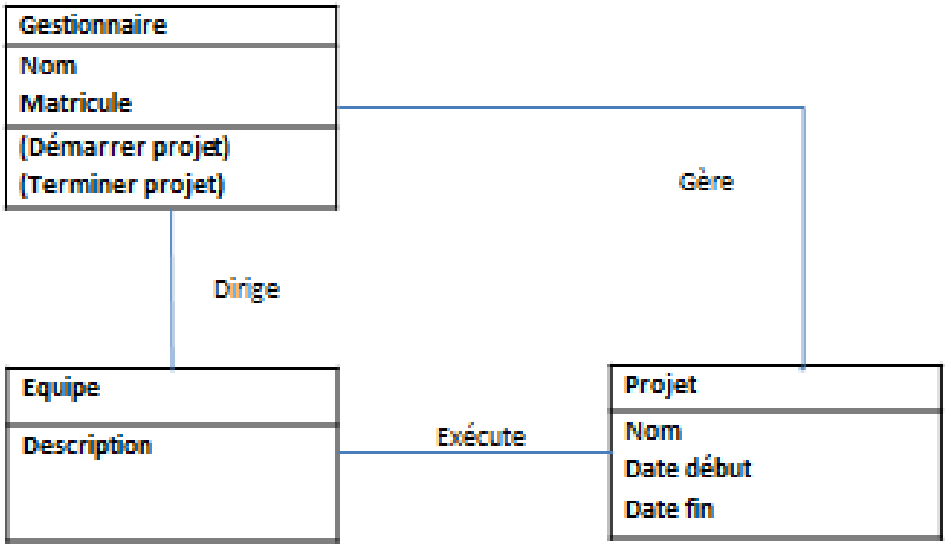


Figure 1.1. Gestionnaires, projets et équipes

EntityManager et Repository

EntityManager : gestionnaire d'entités, il sert à **manipuler** les entités (Ajouter/Modifier/Supprimer)

On l'appelle de la manière suivante :

```
$em = $this->getDoctrine()->getManager();
```

Repositories (modèles en français) **:** servent à **récupérer** les entités dans la BDD. Ils sont créés automatiquement quand vous créez une entité. Ils se trouvent dans /src/Repository.

On appelle un repository dans une méthode d'un **contrôleur** en le mettant en argument, par exemple :

```
public function show(MovieRepository $movieRepository) { ... }
```

→ Un repository dispose toujours de quelques méthodes de base permettant de récupérer de façon très simple les entités. On peut cependant toujours customiser ses propres méthodes (par exemple pour ne récupérer que les entités les plus récentes).

Cheat sheet : Méthodes de l'EntityManager

Persister l'entité (il ne sert a rien de persister deux fois la même entité) :

persist($entity)

Exécuter la requête :

flush()

Supprimer l'entité :

remove($entity)

Cheat sheet : Méthodes des repositories

Ce sont des méthodes généralement définies dans la plupart des projets

Récupérer l'entité par son id :

find($id)

Récupérer toutes les entités d'une classe (on obtient un tableau d'objets) :

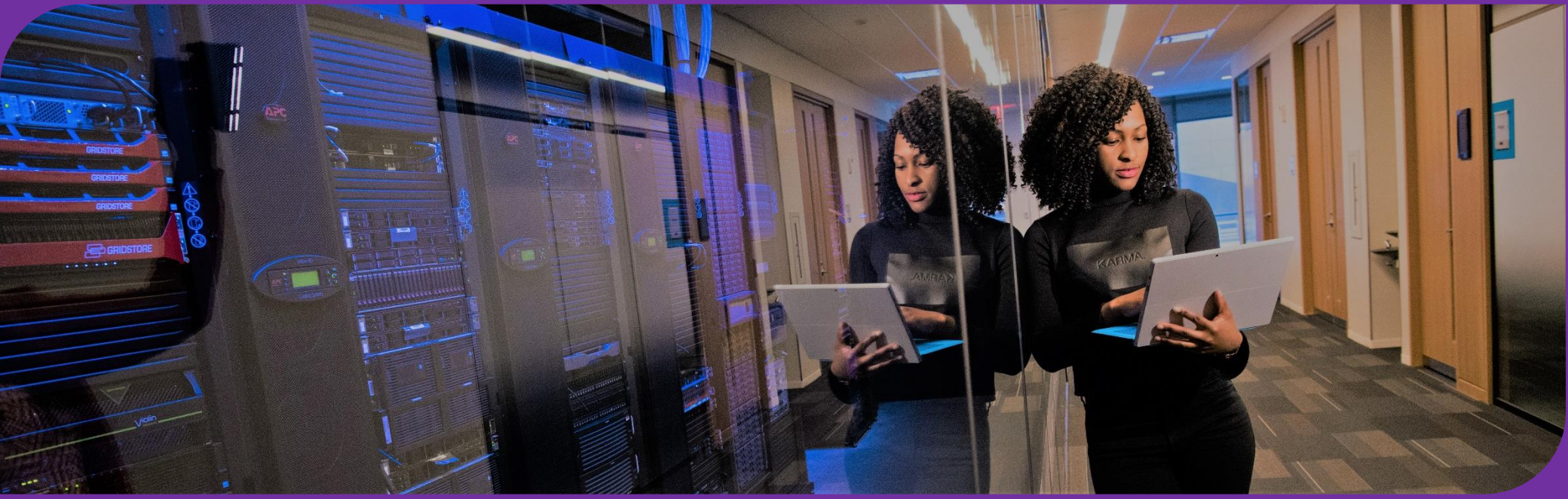
findAll()

Récupérer toutes les entités selon un filtre (on obtient un tableau d'objets) :

findBy($critere, $tri, $limit, $offset)

Comme findBy() mais pour une seule entité, on récupère donc un objet

findOneBy($critere)



2 Doctrine

La BDD et Doctrine

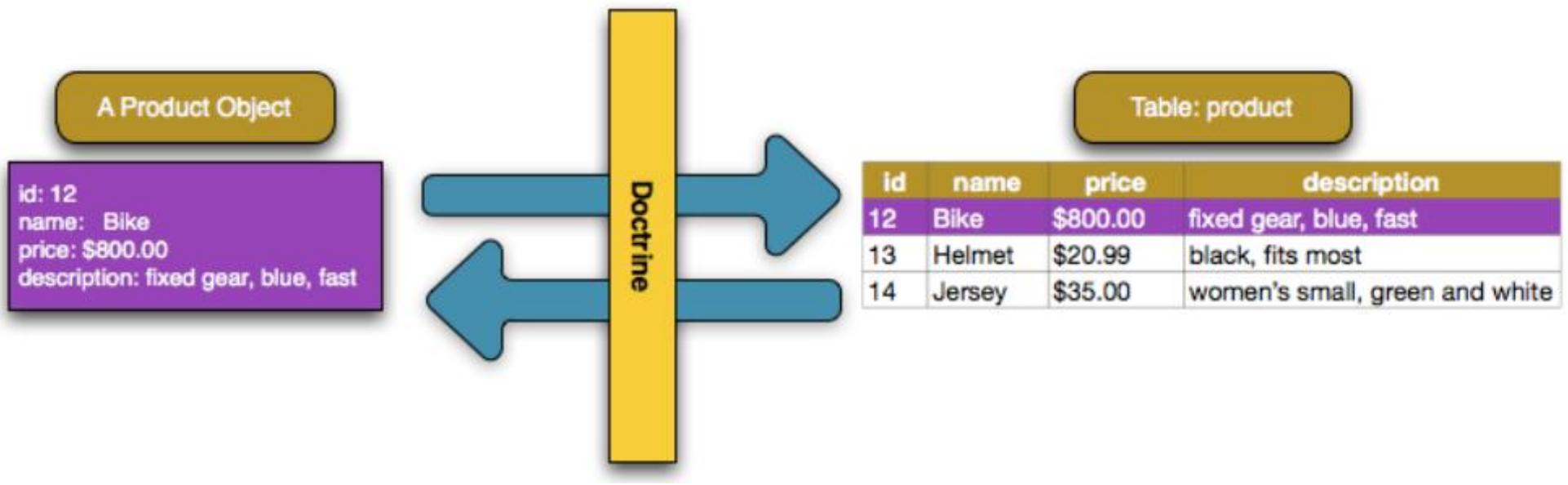
Doctrine : ORM (Object-Relation Mapper) permettant de **stocker** des informations variées.

→ On interagit avec notre **BDD** (ici MySQL) grâce à l'ORM (Object Relational-Mapping) **Doctrine2**.

C'est un **service**, le rôle de l'ORM est de **convertir des entités** (classes PHP) **en tables** dans la BDD. On parle de **« persistance »** ou « mapping ».

➔ Plus besoin de créer ses tables dans phpMyAdmin dorénavant !

La BDD et Doctrine



id	name	price	description
12	Bike	$800.00	fixed gear, blue, fast
13	Helmet	$20.99	black, fits most
14	Jersey	$35.00	women's small, green and white

Paramétrer sa BDD

Si ce n'est pas déjà fait, require le bundle orm :

$ composer require orm

Modifier le fichier **.env** à la racine du projet :

```
###> doctrine/doctrine-bundle ###
# Format described at http://docs.doctrine-project.org/projects/doctrine-dbal/
# For an SQLite database, use: "sqlite:///%kernel.project_dir%/var/data.db"
# Configure your db driver and server_version in config/packages/doctrine.yaml
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
###< doctrine/doctrine-bundle ###
```

→ (il faut modifier **db_user**, **db_password** et **db_name**)

Et enfin **créer** la base de donnée :

$ php bin/console doctrine:database:create

Créer une Entité

Pour **créer** une entité :

$ php bin/console make:entity NomEntité

Il vous sera ensuite demandé de spécifier les **attributs** de l'entité (nom, type, longueur et si nullable).

→ L'entité sera ensuite ajoutée à votre projet, avec son repository, ses attributs, ses getters et ses setters.

On a créé note entité, c'est bien, mais il faut actualiser la BDD maintenant !

Migrer

Étape 1 :

Générer un **fichier** contenant les **requêtes SQL** dans le dossier Migrations.

$ php bin/console make:migration

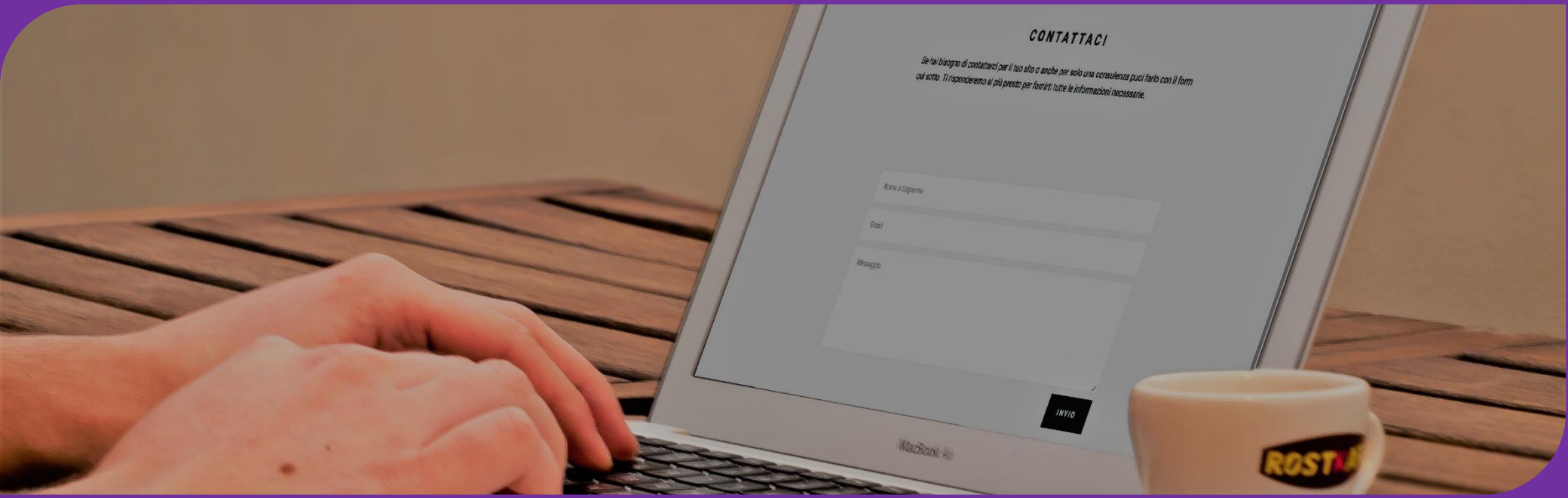
→ le fichier créé permet de **vérifier** la requête SQL avant de migrer, vous pouvez même directement changer la requête SQL dans ce fichier une fois ce dernier généré.

Étape 2 :

Exécuter le fichier précédemment créé pour **persister** les entités.

$ php bin/console doctrine:migrations:migrate

➔ Réitérer ces deux commandes pour chaque changements dans vos entités !


CONTATTACI
Se hai bisogno di contattarci per il tuo sito o anche per solo una consulenza puoi farlo con il form
qui sotto. Ti risponderemo al più presto per fornirti tutte le informazioni necessarie.
Nome e Cognome
Email
Messaggio
INVIO

3 Le CRUD

Le C.R.U.D.

C.R.U.D.: les quatre **opérations** de base pour la **persistance des données**, en particulier le stockage d'informations en base de données.

- **Create :** créer
- **Read :** lire
- **Update :** mettre à jour
- **Delete :** supprimer

Nous verrons rapidement comment exécuter ces actions en brut, puis nous créerons un C.R.U.D. automatiquement en ligne de commande

Create

Dans le contrôleur :

```php
/**
 * @Route("/addMovie", name="add.movie")
 */
public function new()
{
    $movie = new Movie(); // On créé un nouvel objet Movie
    $movie->setTitle( title: "The Grand Budapest Hotel");
    $movie->setDescription( description: "Super film.");

    $em = $this->getDoctrine()->getManager(); // On appelle l'EntityManager
    $em->persist($movie); // On persiste le film
    $em->flush(); // On exécute la persistance

    return new Response( content: "Film ajouté");
}
```

Read

Pour un seul objet

Dans le contrôleur :

```php
/**
 * @Route("/movie/{id}", name="show.movie")
 */
public function show(Movie $movie) // On récupère directement le film dont l'id est spécifié dans l'URI
{
    return $this->render( view: "movie/movie.html.twig", ['movie' => $movie]);
    // Une vue est retournée avec un objet en donnée
}
```

Dans la vue :

```twig
{% block body %}
    <p>Titre du film : {{ movie.title }}</p>
    <p>Description du film : {{ movie.description }}</p>
{% endblock %}
```

Read

Pour un tableau d'objets

Dans le contrôleur :

```php
/**
 * @Route("/movies", name="show.all.movies")
 */
public function showMovies(MovieRepository $movieRepository) // On récupère directement le repository
{
    $movies = $movieRepository->findAll();
    return $this->render( view: "movie/movies.html.twig", ['movies' => $movies]);
    // Une vue est retournée avec un tableau d'objets en donnée
}
```

Dans la vue :

```twig
{% block body %}
    {% for movie in movies %}
        <p>Titre du film : {{ movie.title }}</p>
        <p>Description du film : {{ movie.description }}</p>
    {% endfor %}
{% endblock %}
```

Update

Dans le contrôleur :

```php
/**
 * @Route("/edit/{id}", name="edit.movie")
 */
public function edit(Movie $movie) // On récupère le film dont l'id est spécifié dans l'URL
{
    $movie->setDescription( description: "Une nouvelle description."); // Exemple de changement en brut

    $em = $this->getDoctrine()->getManager(); // On appelle l'EntityManager
    $em->persist($movie); // On persiste le film
    $em->flush(); // On exécute la persistence

    return $this->redirectToRoute( route: "show.movie", ['id' => $movie->getId()]);
    // On redirige vers la page du film qu'on vient de modifier.
}
```

Delete

Dans le contrôleur :

```php
/**
 * @Route("/delete/{id}", name="delete.movie")
 */
public function delete(Movie $movie) // On récupère le film dont l'id est spécifié dans l'URL
{
    $em = $this->getDoctrine()->getManager(); // On appelle l'EntityManager
    $em->remove($movie); // On supprime le film
    $em->flush(); // On exécute la persistence

    return $this->redirectToRoute( route: "show.all.movies");
    // On redirige vers la page présentant tous les films.
}
```

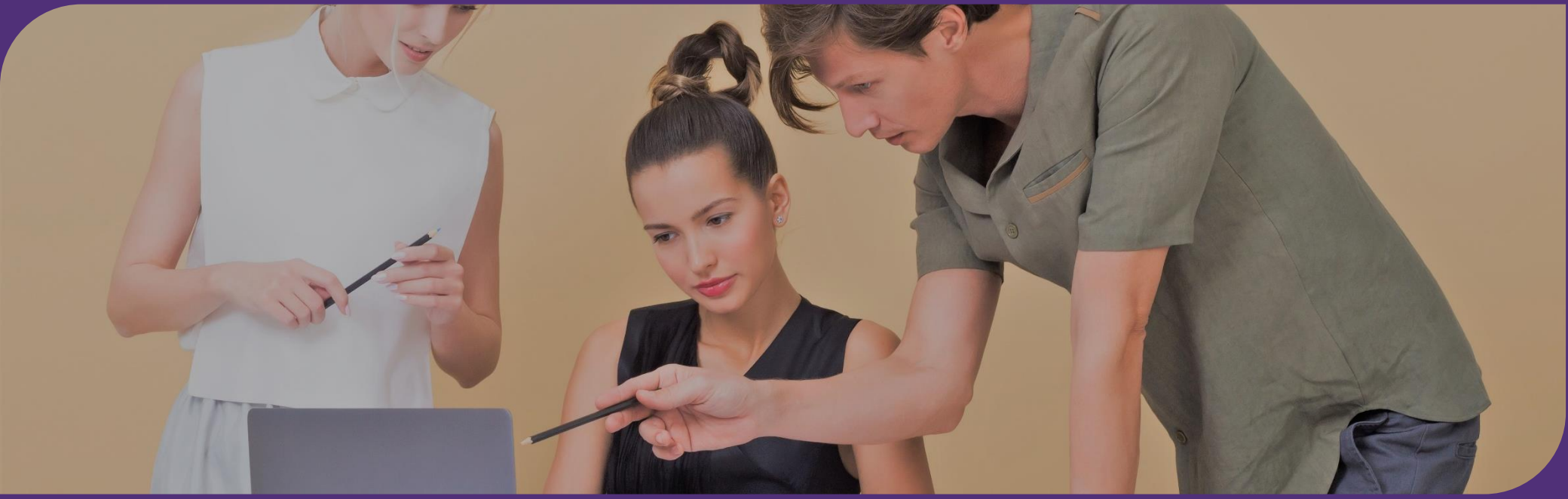
Mettre en place un C.R.U.D. facilement

En ligne de commande :

$ php bin/console make:crud NomEntité

→ Génère le contrôleur avec les méthodes associées, le formulaire et les vues

Pratique pour tout générer en un instant, il faut tout de même personnaliser après selon ses besoins



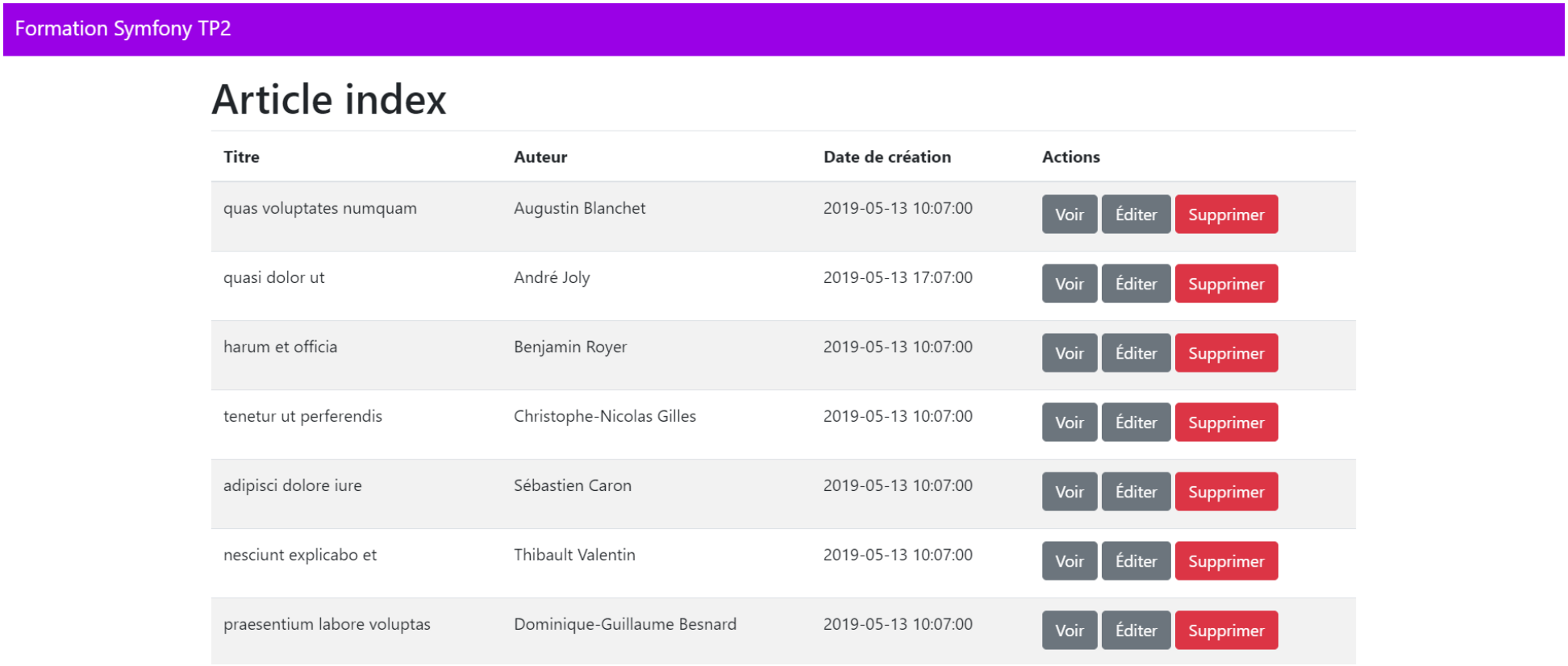
7

TP

TP

But du TP : Réalisation d'un CRUD

On créera pour cela une entité puis son CRUD associé.

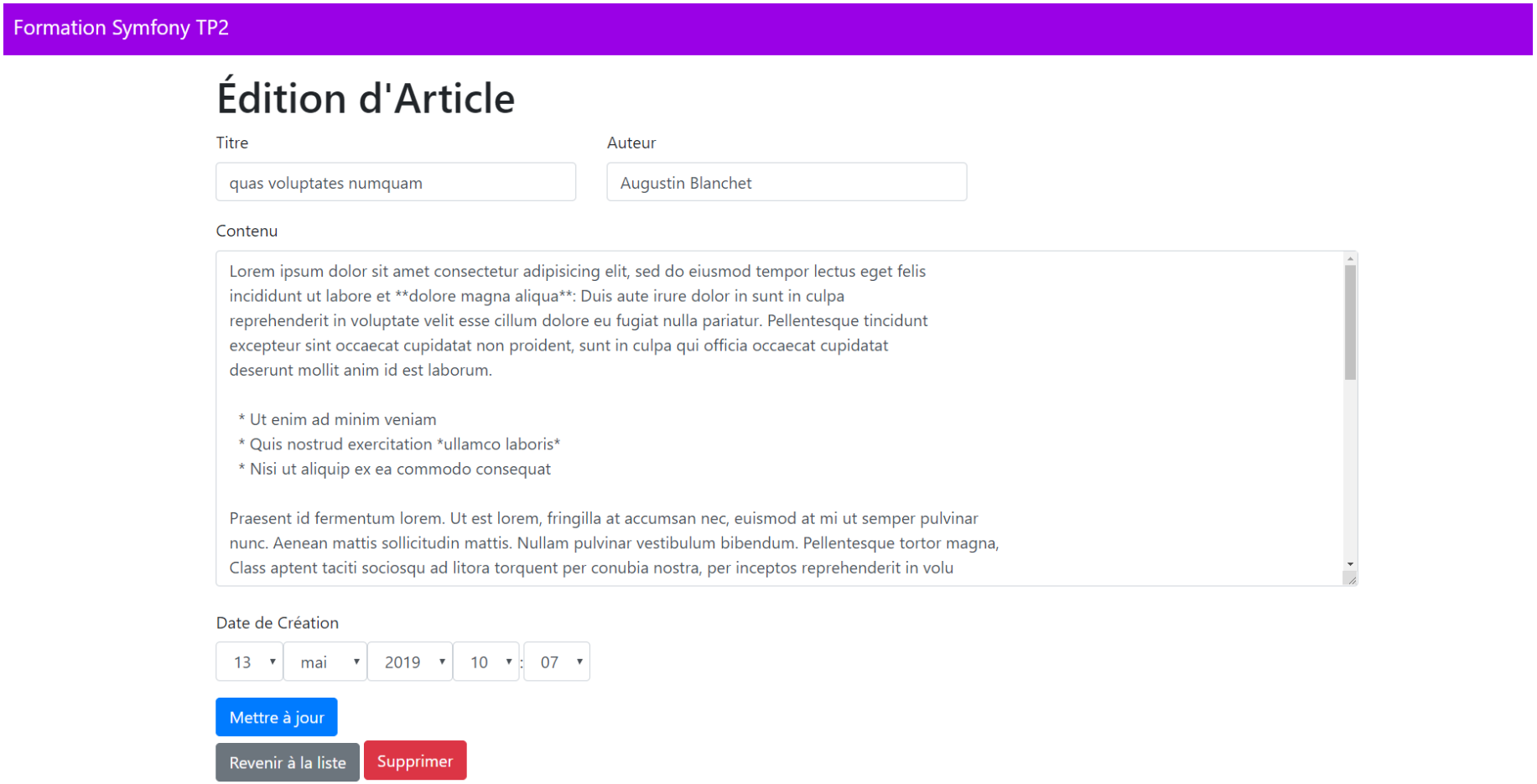


TP

Page de création :

TP

Page d'édition :



TP

Page de lecture :

Formation Symfony TP2

quas voluptates numquam

Auteur : Augustin Blanchet Date de publication : 2019-05-13

Lorem ipsum dolor sit amet consectetur adipisicing elit, sed do eiusmod tempor lectus eget felis incididunt ut labore et **dolore magna aliqua**: Duis aute irure dolor in sunt in culpa reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Pellentesque tincidunt excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia occaecat cupidatat deserunt mollit anim id est laborum.

* Ut enim ad minim veniam
* Quis nostrud exercitation *ullamco laboris*
* Nisi ut aliquip ex ea commodo consequat

Praesent id fermentum lorem. Ut est lorem, fringilla at accumsan nec, euismod at mi ut semper pulvinar nunc. Aenean mattis sollicitudin mattis. Nullam pulvinar vestibulum bibendum. Pellentesque tortor magna, Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos reprehenderit in volu himenaeos. Fusce nulla purus, gravida ac interdum ut, blandit eget ex. Duis a sunt in culpa qui offici luctus dolor.

Integer auctor massa maximus nulla scelerisque accumsan. Aliquam ac malesuada blandit elit vel viverra ex. Pellentesque tortor magna, vulputate eu vulputate ut, venenatis ac lectus. velit esse cillum dolore

JUNIORISEP

Merci d'avoir participé à cette formation Junior ISEP !

N'hésitez pas à nous contacter :

- lprogent@juniorisep.com
- ffavole@juniorisep.com
- pmarquet@juniorisep.com

8 Bonus

Créer des relations entre les entités

Bases sur la **cardinalité** :

Un **modèle**, lorsqu'il met en **relation** deux entités A et B, doit toujours stipuler à **combien d'éléments** de l'entité B chaque élément de A peut **correspondre**, et inversement. Pour chaque élément d'une entité, on doit stipuler à combien d'éléments de l'autre entité celui-ci est susceptible de correspondre.

Des exemples pour comprendre :

- Une structure se situe dans **une seule** ville
- Dans une ville peut se situer **aucune ou plusieurs** structures.

Créer des relations entre les entités

Les différents types de relation :

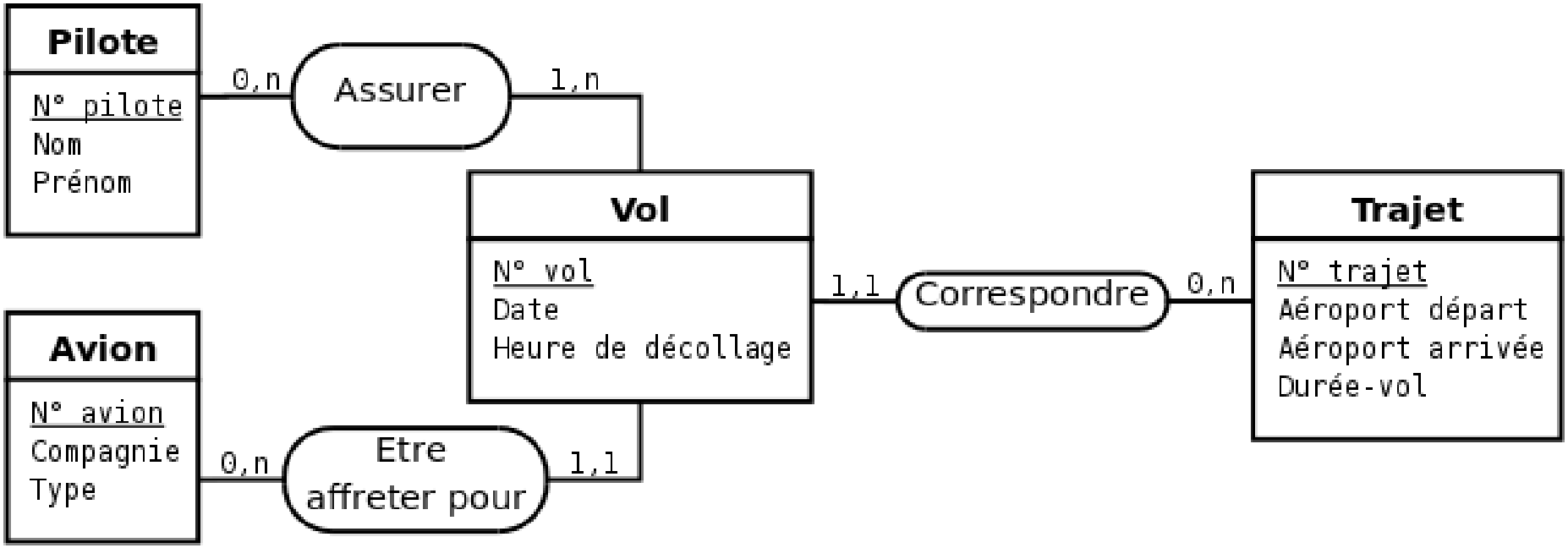
Relation **OneToOne**	
0..1 ↔ 1..1	**1..1 ↔ 1..1**

Relations **OneToMany** ou **ManyToOne**	
0..1 ↔ 1..n	**1..n ↔ 0..1**

Relation **ManyToMany**	
0..n ↔ 0..n	**1..n ↔ 0..n**

Créer des relations entre les entités

Exemple complet :



Créer des relations entre les entités

Pour **créer** une relation entre deux entités :

$ php bin/console make:entity NomEntité

(on peut l'utiliser même si l'entité est déjà créée)

→ Lorsque l'invité de commande demande le **type** de l'attribut, tapez : « **relation** » puis précisez à quelle entité elle est liée. Il faut aussi préciser le **type** de relation (ManyToOne, OneToOne, ...).