Formation Symfony 4
2ème séance

Symfony

Retour sur TP

- Au lieu de faire $this->get('session') pour récupérer l'objet session, mieux vaut mettre Session $session en argument de la méthode de votre Controller.
- Retourner une vue avec des données

```
/**
 * @Route("/color/{color}", name="color", requirements={"color": "[a-zA-Z]+"})
 */
public function colorAction($color){
    return $this->render('example/layout.html.twig', ['color' => $color ]);
}
```

- Afficher les données au niveau de la vue (templates/example/layout.html.twig) : {{ color }} dans notre exemple

Les namespaces

- Permet de résoudre les problèmes de collision pour deux classes portant le même nom. C'est un peu « la portée » d'une classe.
- Cela permet aussi d'organiser les sources de vos programmes (dossiers et fichiers).

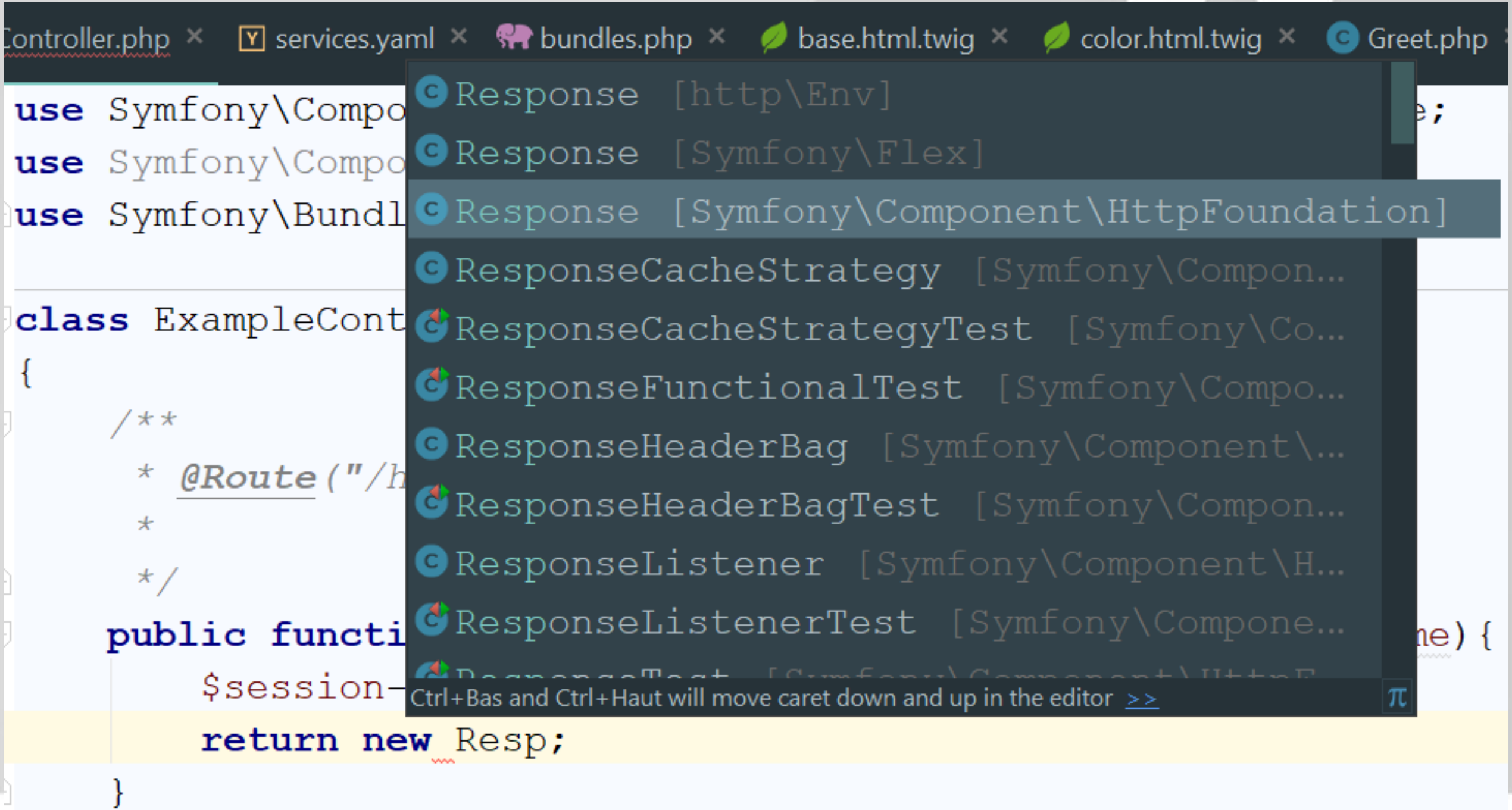
```php
<?php
//Equivalent du chemin src\Controller
//App est le "namespace source" (global) pour notre projet
namespace App\Controller;

//On importe des classes utilisant des namespaces
//Il suffit de taper: use A\MaClasse avec A le namespace de la classe qu'on importe
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
```

Conseils et astuces

- Lors de la création d'un nouveau projet, installer les bundles principaux d'un coup :

$ composer req server --dev twig annotations maker-bundle profiler (req est une abréviation de require)

- Lorsque vous utilisez une classe que vous n'avez pas défini dans votre fichier vous pouvez facilement l'importer sans écrire du code, sur phpStorm c'est automatique. Cependant, vous pouvez toujours utiliser le raccourci pour importer une classe si nécessaire à savoir Alt + Enter.

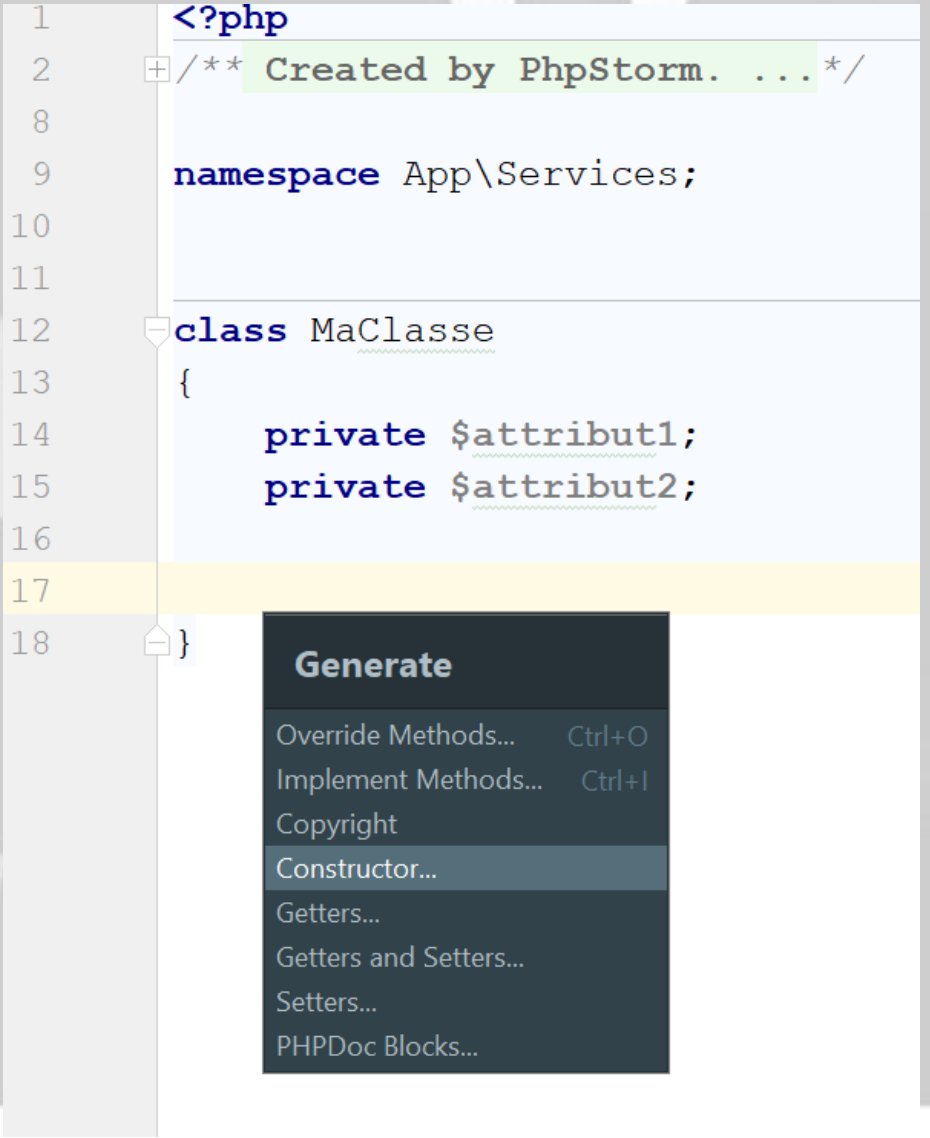

Controller.php
services.yaml
bundles.php
base.html.twig
color.html.twig
Greet.php
use Symfony\Compo
use Symfony\Compo
use Symfony\Bundl
class ExampleCont
{
/**
* @Route("/h
*
*/
public functi
$session-
return new Resp;
}
Response [http\Env]
Response [Symfony\Flex]
Response [Symfony\Component\HttpFoundation]
ResponseCacheStrategy [Symfony\Compon...
ResponseCacheStrategyTest [Symfony\Co...
ResponseFunctionalTest [Symfony\Compo...
ResponseHeaderBag [Symfony\Component\...
ResponseHeaderBagTest [Symfony\Compon...
ResponseListener [Symfony\Component\H...
ResponseListenerTest [Symfony\Compone...
Ctrl+Bas and Ctrl+Haut will move caret down and up in the editor >>

Conseils et astuces 2

Pour créer un constructeur et des getter/setter d'une classe automatiquement sur phpStorm, aller dans:

Code>Generate… (un raccourci par défaut sera visible, vous pouvez le changer dans:

File>Settings>Keymap>Main menu>Code>Generate)

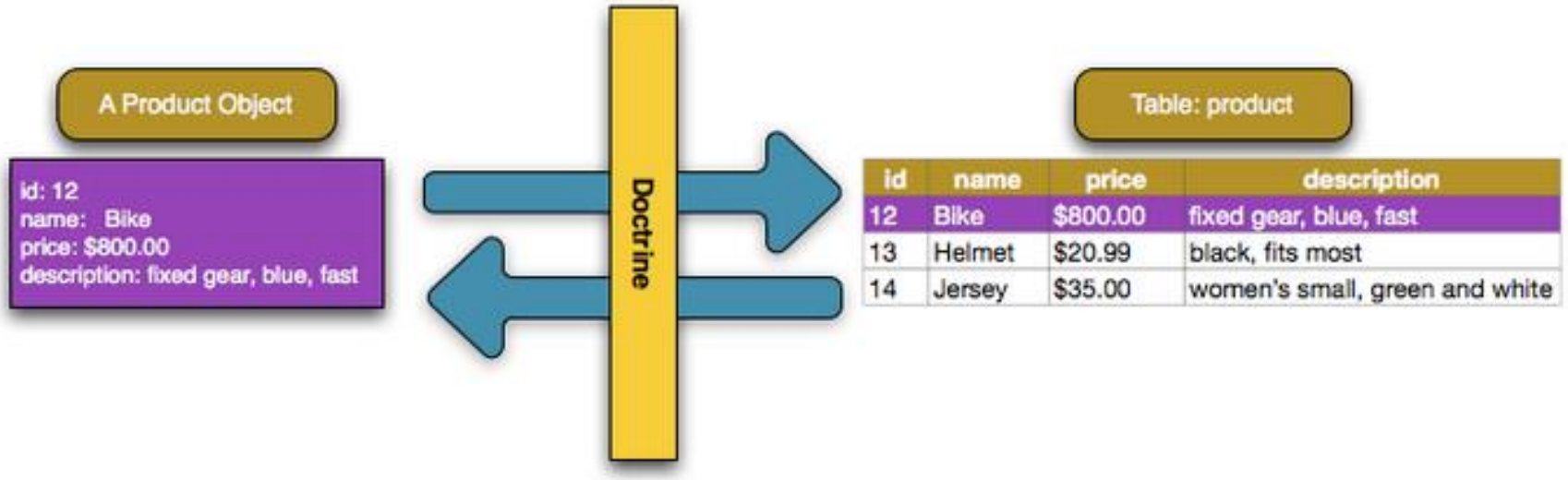


Conseils et astuces 3

- Pour chercher n'importe quel fichier dans votre projet, utilisez le raccourci Shift + Shift
- Pour afficher une console dans la fenêtre de PhpStorm aller dans View>Tool Windows>Terminal (pratique pour ne pas switcher entre terminal et fenêtre pour coder, seul bémol : ne comprends pas les commandes linux)

La base de données (BDD)

- Permets de stocker des informations : utilisateurs, commentaires, articles, etc.
- On interagit avec notre BDD (ici MySQL) grâce à l'ORM (Object Relational Mapping) **Doctrine 2.** 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.



Paramétrer la base de données

JUNIORISEP

$ composer require orm

Dans le fichier .env à la racine du projet :

```
###> doctrine/doctrine-bundle ###
# Format described at http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/
# 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://root:@127.0.0.1:3306/tp_seance2_symfony
```

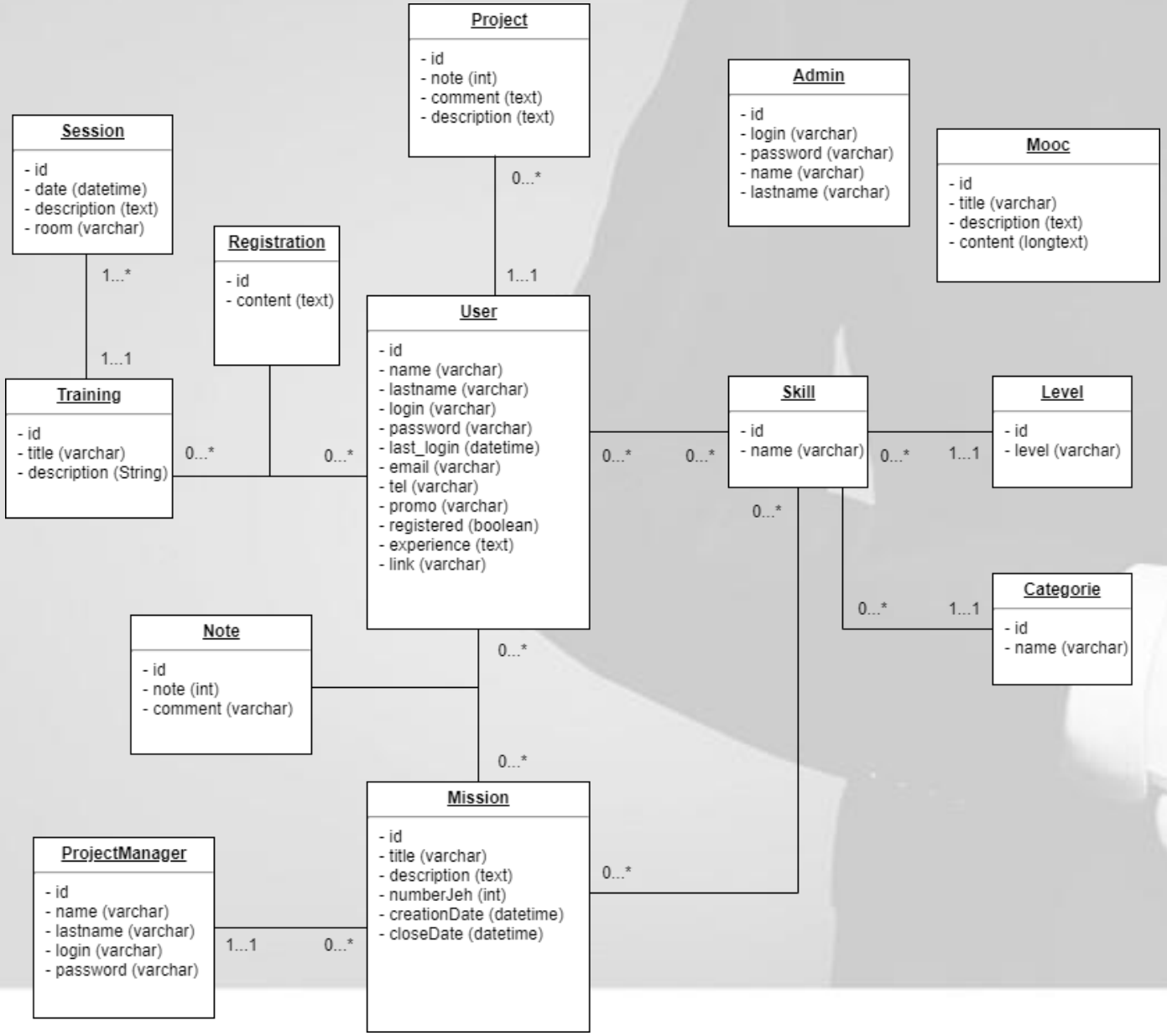
$ php bin/console doctrine:database:create

Les Entités

- Un objet dont vous confiez l'enregistrement à l'ORM s'appelle une **entité.** On dit également **persister** une entité, plutôt qu'enregistrer une entité.
- Sur un projet: il faut modéliser **la structure de la base de données UML** en avance, c'est-à-dire les entités (avec méthodes et attributs) et leur relations.
- Créer une entité avec le maker-bundle:

  $ php bin/console make:entity EntityName
- Les entités se trouvent dans **/src/Entity/**

Exemple de base de données en UML avec cardinalité



Mapper vos entités avec la BDD

$ php bin/console make:migration => génère un fichier php contenant les requêtes SQL dans le dossier migrations

$ php bin/console doctrine:migrations:migrate => exécute le fichier php précédemment crée pour persister les entités

- Réitérer ces deux commandes pour chaque changements dans vos entités
- On peut également utiliser $ php bin/console doctrine:schema:update --force pour mettre à jour directement la BDD depuis vos entités sans passer par le fichier de migration php.

L’Entity manager et les Repositories

- L'EntityManager est un gestionnaire d’entités, il sert à **manipuler** les entités => Ajouter/Modifier/Supprimer. On l’appelle avec : **$em = $this->getDoctrine()->getManager();**

- Les Repositories servent à **récupérer** les entités. Ils sont crées automatiquement grâce au maker-bundle quand vous avez crée une entité. Ils se trouvent dans **/src/Repository**. On appelle un repository en le mettant en argument de la méthode du Controller (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

Les principales méthodes de l'entity manager

- persist($entity) => persiste l'entité (il ne sert a rien de persister deux fois la même entité)
- flush() => exécute la requête
- remove($entity) => supprime l'entité

Les méthodes de récupération de base du Repository

- find($id) => récupère l'entité par son id
- findAll() => récupère toutes les entités d'une même classe, on obtient donc un tableau d'objets
- findBy($critere, $tri, $limit, $offset) => récupère toutes les entités selon un filtre, donc on a au final un tableau d'objets
- findOneBy($critere) => comme findBy() mais pour une seule entité, on récupère donc un objet

Ajouter une entité

```php
/**
 * @Route("/movie", name="movie")
 */
public function index()
{
    $movie = new Movie();
    $movie->setTitle( title: "The Notorious");
    $movie->setContent( content: "C'est un documentaire sur Conor McGregor, champion de MMA"

    //L'Entity Manager sert à manipuler les entités, il fait le lien avec la bdd
    $em = $this->getDoctrine()->getManager();
    $em->persist($movie); //on persiste le film
    $em->flush(); //on execute la persistence

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

Afficher une entité

Au niveau du Controller

```php
/**
 * @Route("/movie/{id}", name="showMovie")
 */
//récupère directement le film correspondant à l'id indiqué dans l'url
public function show(Movie $movie)
{
    //attention, une vue est retournée avec un objet en donnée et pas une variable
    return $this->render( view: 'layout/layout.html.twig', ['movie' => $movie]);
}
```

Au niveau de la Vue

```twig
{% extends 'base.html.twig' %}

{% block body %}

    <p>Film : {{ movie.title }}</p>
    <p>Contenu : {{ movie.content }}</p>

{% endblock %}
```

Afficher des entités (ici toutes les entités d'une même table)

Au niveau du Controller

```php
/**
 * @Route("/movies", name="showMovies")
 */
public function showMovies(MovieRepository $movieRepository)
{
    $movies = $movieRepository->findAll();
    return $this->render( view: 'layout/layout1.html.twig', ['movies' => $movies]);
}
```

Au niveau de la Vue

```twig
{% extends 'base.html.twig' %}

{% block body %}

    {% for movie in movies %}
        <p>Film : {{ movie.title }}</p>
        <p>Contenu : {{ movie.content }}</p>
    {% endfor %}

{% endblock %}
```

Modifier une entité

```php
/**
 * @Route("/edit/{id}", name="editMovie")
 */
public function editMovie(Movie $movie)
{
    $movie->setContent( content: "Mon nouveau contenu");

    $em = $this->getDoctrine()->getManager();
    $em->flush();
    return $this->redirectToRoute( route: 'showMovie', ['id' => $movie->getId()]);
}
```

Supprimer une entité

```php
/**
 * @Route("/delete/{id}", name="deleteMovie")
 */
public function deleteMovie(Movie $movie)
{
    $em = $this->getDoctrine()->getManager();
    $em->remove($movie);
    $em->flush();
    return new Response( content: "Film bien supprimé");
}
```

Projet : Site web

Annexe : Cardinalités en UML -> Relations

0…1 – 1…1, 1…1 – 1…1, etc => relation OneToOne

0…1 – 1…*, 1…* – 0…1, etc => relation OneToMany ou ManyToOne

0…* – 0…*, 1…* – 0…*, etc => relation ManyToMany

Annexe : Créer des relations entre les entités

- $ php bin/console make:entity EntityName (on peut l'utiliser même si l'entité est déjà crée).
- Lorsque l'invite de commande demande le type de l'attribut, tapez : « relation » puis précisez a quelle entité elle est liée. Il faut aussi préciser le type de relation (ManyToOne, OneToOne, …).