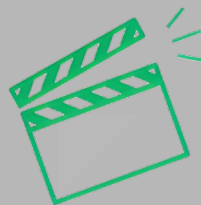


Projet - Symfony



Moviz

Déroulé

- 7 lives d'une heure sur la mise en place d'un projet Symfony.
 - On peut déjà s'inscrire aux prochains.
- Ce live s'adresse aux débutants :
 - Parcours Spé Symfony : un live est prévu fin novembre.
 - Le même projet sera développé en php vanilla (POO) démarre fin novembre
- Prérequis :
 - Avoir les connaissances de php et les notions de classe/objet et du modèle MVC.
 - Il est également conseillé d'avoir les connaissances de base de Symfony.
- Questions/Réponse à la moitié (25min) et fin du live (55min).
- Sources du projet sur github :
 - https://github.com/arirangz/symfony_moviz

Programme des 7 lives

- Présentation du projet
- Conception de la base de données
- Installation et configuration de Symfony
- Création des Entity
- Génération des tables
- Mise en place de l'administration
- Sécurisation de l'interface d'administration
- Génération des formulaires de connexion
- Création du formulaire d'inscription
- Création des pages et actions front (twig, html, css, contrôleur, repository)




Environnement technique

- PHP \geq 8.1 et MYSQL \geq 5.7 (WAMP)
- Symfony 6.3
- Bootstrap 5.3
- Easyadmin 4.7
- VSCode



Besoin

- La société Moviz veut proposer un site de notation et critique de films.
 - Elle souhaite que les utilisateurs puissent s'inscrire et noter un film et en même temps poster une critique (optionnel).
 - Elle souhaite pouvoir administrer les films (nom, année de sortie) qui seront catégorisés (thriller, action etc.) et pourront être associés à plusieurs réalisateurs (nom, prénom).
 - Elle souhaite également pouvoir administrer les critiques (validation requise) mais la note sera prise en compte automatiquement.
 - Eventuellement on souhaite pouvoir permettre aux visiteurs de filtrer les films (par catégorie, par réalisateur, recherche par mot clé).
 - Une charte graphique a été définie.
- 

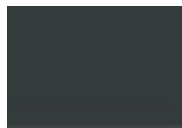
Charte graphique

- Police : **SFMono-Regular**

- Couleurs :



#09b468



#323b3c



#deb887

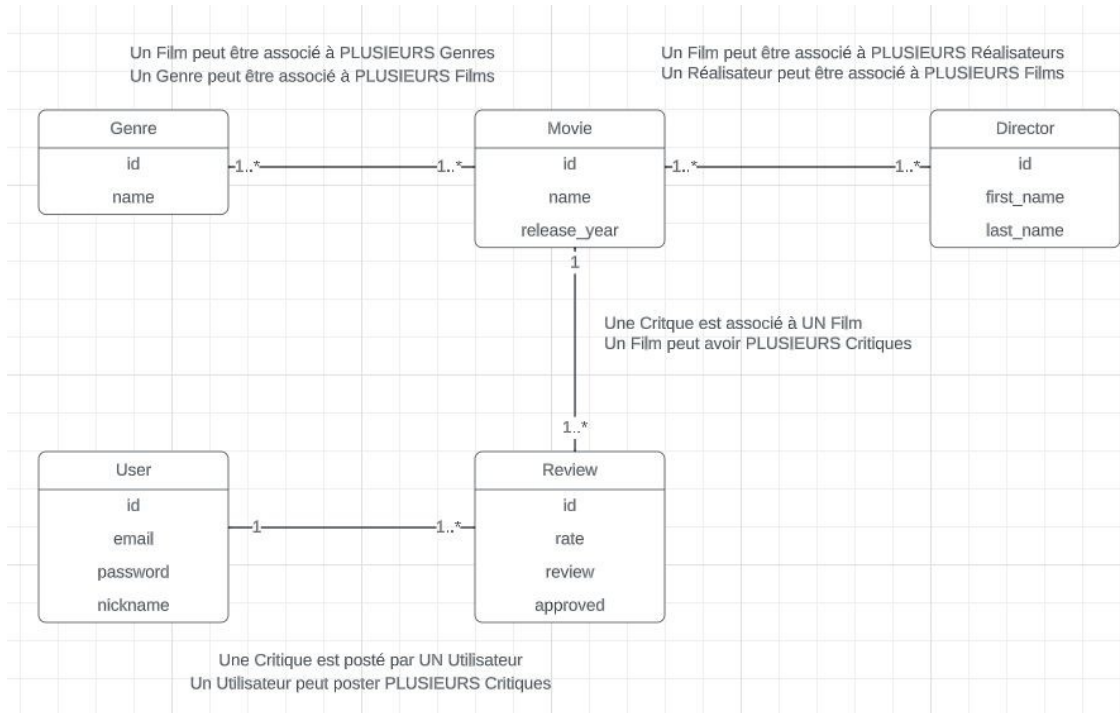
- Logo :



Moviz

Modèle Conceptuel de Données

<https://lucid.app/>



Symfony

- Symfony est un framework open source écrit en PHP qui permet aux développeurs de créer des applications web rapidement et efficacement.
- Symfony suit souvent le modèle de conception MVC (Modèle-Vue-Contrôleur) et d'autres design pattern et fournit des outils et des composants prêts à l'emploi pour faciliter le développement d'applications web.



Extensions VSCODE



PHP Intelephense

PHP code intelligence for Visual Studio Code



Ben Mewburn



Twig Language 2

Snippets, Syntax Highlighting, Hover, and Formatting
mblode



DotENV

Support for dotenv file syntax
mikestead



phpfmt - PHP formatter

Integrates phpfmt into VS Code
kokororin



Material Icon Theme

Material Design Icons for Visual Studio Code
Philipp Kief



PHP Getters & Setters (CV fork)

Create PHP getters and setters from class properties



Christophe VERGNE



PHP Namespace Resolver

Import and expand php namespaces
Mehedi Hassan



YAML

YAML Language Support by Red Hat, with built-in K...
Red Hat



PHPDoc Comment

Add phpdoc @param and @return tag for selected func...
Rex Shi




Prettier - Code formatter

Code formatter using prettier
Prettier

Installation

- Avoir défini php dans les variables d'environnement et avoir Xdebug d'activé
- Installer composer
 - <https://getcomposer.org/download/>
- Installer symfony cli
 - <https://symfony.com/download>
- Vérifier les requirements
 - `symfony check:requirement`
- Créer un nouveau projet symfony
 - `symfony new my_project --webapp`
- Relancer l'indexation dans VSCODE : CTRL+MAJ+P -> Intelephense: index
- Lancer un serveur
 - `symfony server:start`
- Lien vers le git
 - https://github.com/arirangz/symfony_moviz

Structure d'une application Symfony

- bin/ : Ce dossier contient des scripts exécutables en ligne de commande pour l'application.
 - config/ : Ce dossier contient la configuration principale de l'application, le fichier de configuration des routes, le fichier de configuration des services, les fichiers de configuration des packages.
 - migrations: classes de migrations générées pour ajouter/modifier des tables
 - public/ : Ce dossier contient les fichiers publics de l'application, tels que les fichiers CSS, JavaScript et les images.
 - src/ : Ce dossier contient le code source de l'application, les contrôleurs, les entités, les classes de formulaires, etc.
 - templates/ : Ce dossier contient les fichiers de templates Twig utilisés pour générer les pages de l'application.
 - tests/ : Ce dossier contient les fichiers de tests unitaires et fonctionnels pour l'application.
 - var/ : Ce dossier contient les fichiers de cache et les fichiers journaux de l'application.
 - vendor : Contient tous les composants installés avec composer
- 

Première page Symfony

- <https://symfony.com/doc/6.2/the-fast-track/fr/6-controller.html>



Routing annotation vs attribute

```
/**
 * @Route("/blog")
 */
class BlogController extends AbstractController
{
```



```
#[Route('/blog')]
class BlogController extends AbstractController
{
```

```
/**
 * @Route("/posts/{slug}", methods="GET", name="blog_post")
 */
public function postShow(Post $post): Response
{
```



```
#[Route('/posts/{slug}', name: 'blog_post', methods: ['GET'])]
public function postShow(Post $post): Response
{
```

Routing annotation vs attribute

config/routes.yaml

```
controllers:  
  resource:  
    path: ../src/Controller/  
    namespace: App\Controller  
    type: annotation
```

config/routes.yaml

```
controllers:  
  resource:  
    path: ../src/Controller/  
    namespace: App\Controller  
    type: attribute
```

Depuis php 8, on peut utiliser les attributs php

<https://www.php.net/manual/fr/language.attributes.overview.php>

Webpack Encore

- **Combiner** : Webpack regroupe plusieurs fichiers source en un seul fichier, ce qui permet de réduire le nombre de requêtes réseau nécessaires pour charger une page web. Cela améliore les performances de chargement de votre site.
- **Transformer**: Webpack prend en charge différents types de fichiers source, tels que ES6/ESNext JavaScript, TypeScript, Sass, LESS, CoffeeScript, etc. Il peut les transpiler dans des formats plus anciens ou en JavaScript natif pour une meilleure compatibilité.
- **Minifier** : Webpack peut réduire la taille des fichiers en supprimant les espaces vides, les commentaires, et en obfusquant le code. Cela permet de réduire la taille des fichiers et d'accélérer le chargement de la page.
- **Optimiser** : Webpack peut optimiser automatiquement les images en les comprimant et en les convertissant dans des formats plus légers, réduisant ainsi le temps de chargement des images.
- etc.



Webpack Encore

<https://symfony.com/doc/current/frontend/encore/installation.html>

composer require symfony/webpack-encore-bundle

npm install

Pour compiler en dev

npm run dev

Pour compiler en prod

npm run build

Pour compiler en temps réel les changements :

npm run watch

<https://symfony.com/doc/current/frontend/encore/simple-example.html>



Webpack Encore : SASS et Bootstrap

npm install sass-loader sass --save-dev

Modifier webpack.config.js et ajouter

```
.enableSassLoader()
```

npm install bootstrap --save-dev

<https://symfony.com/doc/current/frontend/encore/bootstrap.html>



Webpack Encore : Images

npm install file-loader --save-dev

Modifier webpack.config.js et ajouter

```
.copyFiles({  
  from: './assets/images',  
  to: 'images/[path][name].[ext]',  
})
```



Easyadmin

Installation du bundle easyadmin

```
composer require easycorp/easyadmin-bundle
```



Easyadmin - Création du dashboard

<https://symfony.com/bundles/EasyAdminBundle/current/dashboards.html>

On modifie index() de DashboardController pour afficher une page

```
public function index(): Response
{
    return $this->render('admin/dashboard.html.twig');
}
```

On ajoute le fichier templates/admin/dashboard.html.twig

```
{% extends '@EasyAdmin/page/content.html.twig' %}

{% block content_title %}Tableau de bord{% endblock %}

{% block main %}
    <h2>Bienvenue sur l'administration Moviz !</h2>
{% endblock %}
```

Base de donnée

On va devoir créer une base de données puis modifier notre fichier .env (ou .env.local) pour préciser la connexion

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/studi_moviz?serverVersion=5.7.36&charset=utf8mb4"
```



Ajouter un login

Admin

<https://symfony.com/doc/current/the-fast-track/fr/15-security.html>

```
symfony console make:user User
```

```
symfony console make:auth
```



Création des entités

La commande suivante va nous permettre de générer une classe dans Entity

```
symfony console make:entity
```

<https://symfony.com/doc/current/doctrine.html>




Création des tables

La commande suivante va générer un fichier de migration (une classe php avec des requêtes SQL)

```
symfony console make:migration
```

La commande suivante va exécuter les requêtes SQL

```
php bin/console doctrine:migrations:migrate
```



Easyadmin

Au lieu d'afficher le dashboard demo, on peut rediriger vers un CRUD

```
public function index(): Response
{
    $adminUrlGenerator = $this->container->get(AdminUrlGenerator::class);
    return $this->redirect($adminUrlGenerator->setController(MovieCrudController::class)->generateUrl());
}
```

Afficher en français par défaut

Il faut définir la “locale” par défaut.

```
# config/packages/translation.yaml
framework:
    default_locale: fr
```



Easyadmin

Création des CRUD

```
php bin/console make:admin:crud
```

Pour User, il faut rajouter le hashage du mot de passe :

<https://dev.to/nabbisen/symfony-6-and-easyadmin-4-hashing-password-3eec>

On va rajouter nickname et roles :

```
TextField::new('nickname'),  
ChoiceField::new('roles')  
->setChoices(['ROLE_ADMIN' => 'ROLE_ADMIN', 'ROLE_USER' => 'ROLE_USER'])  
->allowMultipleChoices()  
->renderExpanded(),
```

Easyadmin

Gérer les champs type association:

<https://symfony.com/doc/4.x/EasyAdminBundle/fields/AssociationField.html>

Lorsque Easyadmin va tenter d'afficher les informations des tables liées, cela va provoquer une erreur car il ne sait pas quel information afficher.

Pour corriger cela on va devoir ajouter une méthode `__toString` sur chaque entité qui va retourner une valeur :

```
public function __toString()
{
    return $this->getName();
}
```