Tutoriel Symfony 6.4



Sommaire

[1. Introduction 3](#_gumhuvg9w8ha)

[2. Prérequis 3](#_41qomvpe1nms)

[2.1. Base de données 3](#_97nb45lcmhwh)

[2.2. PHP 3](#_gs469l795m02)

[2.2.1. Sous Windows 3](#_b9ns2ftwpk6g)

[2.2.2. Sous Mac 6](#_rolq3jjo712g)

[2.3. XDebug avec Xampp 6](#_pp2axzsn4u4u)

[2.4. Composer 7](#_cv2b2gair84h)

[2.5. CLI Symfony 7](#_iq8caaeyrqu6)

[2.5.1. Sous Windows 7](#_vib7eghldhel)

[2.5.2. Sous Mac 7](#_he08pnwb28xx)

[2.6. Symfony Requirements Checker 7](#_oao3fzmetgy9)

[2.7. VSCODE 8](#_9u50ilcmg5nf)

[3. Créer le projet Symfony 9](#_c2ehjouqm16w)

[4. Aperçu du projet 10](#_oz2jk2xkk2j7)

[5. Démarrer le serveur 14](#_meixbr77zlhv)

[6. Contrôleur 14](#_da5z81tk6c92)

[7. Templates: Twig 16](#_2ksa7bul4cgl)

[7.1. Afficher une variable 16](#_mq9lyaagusb0)

[7.2. if 16](#_dchqumur8cs2)

[7.3. for 16](#_j7cl54sefxzg)

[7.4. include 17](#_xgy20sw3cwf1)

[7.5. block 17](#_zi15pgmvdkp4)

[7.6. Afficher une image 17](#_jc3uj3b7m10i)

[7.7. Adaptation de la page d'accueil 17](#_qyky0uoa20y)

[7.8. Ajout d'en-tête et de pied de page 18](#_fb0ethnrnum)

[8. Créer la page à propos 21](#_bewmq8g1wy1o)

[9. Créer la base de données 23](#_udyhjt3kw6cg)

[10. Créer les entités 23](#_2mdpuuocks0z)

[11. Créer la migration et migrer 27](#_tk0qh6ak9ymo)

[12. Ajouter des données 27](#_ny4a9l8aepzn)

[13. Afficher les derniers jeux sur la page d'accueil 28](#_dahxt7j9841o)

[14. Création d'un backend d'administration 30](#_licy5pear7c4)

[14.1. Installation d'EasyAdmin 30](#_76zcq33elwgl)

[14.2. Modifier le tableau de bord 31](#_571fj5trkis8)

[14.3. Sécurisez l'administrateur 31](#_qc0mv2rinscm)

[14.4. Ajouter un système de connexion 32](#_i1rlu0tl6sxe)

[14.5. Ajouter un CRUD 34](#_aqmxca95i2fc)

[14.6. Personnaliser les CRUD 34](#_4k0toaoo2vdb)

[14.7. User CRUD 35](#_b1a8xehrl3cw)

[14.8. Autoriser le téléchargement d'images 36](#_cm7yqqw1wcg5)

[15. Mise à jour du menu 38](#_pfkk4pye0nd4)

[16. Liste des jeux 38](#_ctkyk7c1lpjh)

[17. Détails du jeu 40](#_qrkepetsrvos)

[18. Genre controller 42](#_3fz206ureyvd)

[19. Fonctionnalité Ajouter à la liste de souhaits 43](#_34okhi8ht5n1)

[20. Voir la liste de souhaits 47](#_mwxn3sefo9fm)

[21. Ajouter un avis 48](#_7tkfuroetznz)

[22. Commandes utiles 53](#_s8i7bgv4m91m)

# Introduction

Au cours de ce didacticiel, nous allons créer une application Web similaire à Steam/Epic Games.

Nous aurons des jeux avec des genres (fps, sport, puzzle game, tower defense etc.).

Les jeux peuvent avoir un éditeur.

Les utilisateurs (nom d'utilisateur, email, mot de passe) pourront ajouter des jeux à leur liste de souhaits.

# Prérequis

Avant de commencer à travailler sur le projet, nous devons vérifier que vous disposez d'un bon environnement de travail.

***Passez directement à la section 3. si vous avez déjà un environnement configuré.***

## Base de données

Assurez-vous d'avoir une base de données locale (mysql, mariadb, postgreSQL etc.) par exemple avec Xampp, Mamp ou Wamp.

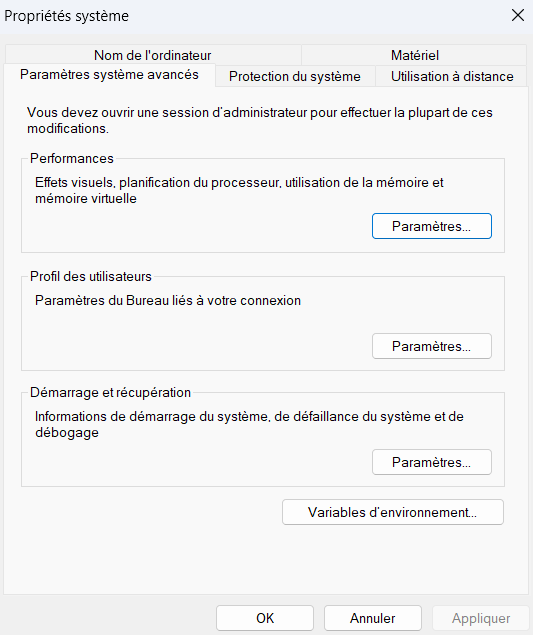
## PHP

Nous avons besoin de PHP 8.1 ou supérieur.  
Nous devons également avoir php disponible dans le terminal.

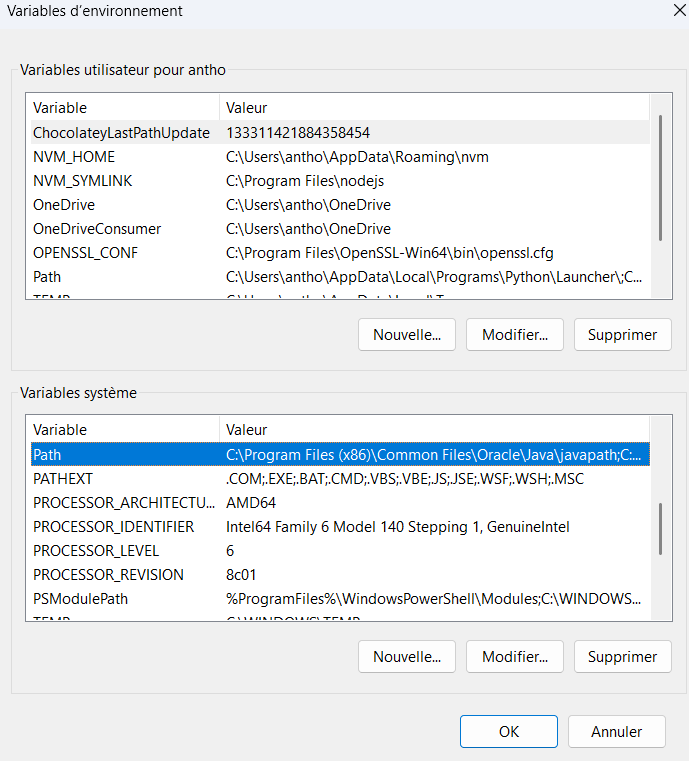
### Sous Windows

Localisez d’abord votre dossier de version php. Par exemple avec wamp, cela ressemblera à ça :  
C:\wamp64\bin\php\php8.2.21

Dans la barre de recherche, tapez « environnement» pour ouvrir la fenêtre de réglage des Propriétés système.  
  
Cela ouvrira la fenêtre Propriétés système. Sélectionnez l'onglet Avancé, puis le bouton Variables d'environnement.

Sous la section Variables système, faites défiler vers le bas et mettez en surbrillance la variable Path. Cliquez sur le bouton Modifier.



Assurez-vous que php n'est pas déjà dans la liste. Sinon, dans l'écran Modifier, cliquez sur Nouveau et ajoutez le chemin d'accès au répertoire Bin de php (par exemple : C:\wamp64\bin\php\php8.2.21).

Ouvrez le terminal et exécutez la commande suivante :

php -v

Vous devriez voir votre version php.

### Sous Mac

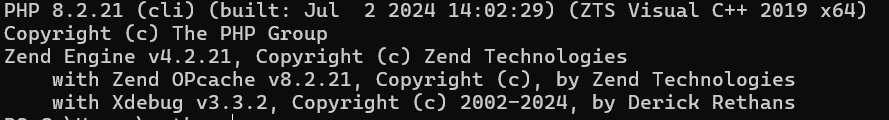
Dans le Finder, recherchez le dossier php, par exemple :  
/Applications/MAMP/bin/php/php8.2.0/bin/  
  
Dans le terminal, exécutez la commande suivante :  
**nano ~/.zshrc**  
  
Ajoutez la ligne suivante (en remplaçant par la version correcte)  
export PATH="/Applications/MAMP/bin/php/php8.2.0/bin/:$PATH"

Appuyez ensuite sur CTRL+X et confirmez avec Y.  
  
Exécutez la commande suivante :  
**source ~/.zshrc**  
Redémarrez le terminal et exécutez la commande suivante :

**php-v**  
Vous devriez voir votre version php.

## XDebug avec Xampp

Dans le terminal, lors de l'exécution de la commande suivante, vous devriez voir Xdebug :  
php -v

  
Si vous ne le voyez pas, vous devez modifier le php.ini pour l'activer et redémarrer le terminal.

Tout en bas du fichier ajouter:

***[xDebug]***

***zend\_extension = C:\xampp\php\ext\php\_xdebug.dll***

***xdebug.remote\_enable = 1***

***xdebug.remote\_autostart = On***  
Télécharger xdebug en fonction de votre version de php, renommer le fichier et le coller dans C:\xampp\php\ext\  
<https://xdebug.org/download>

## Composer

* + 1. Sous Windows

Téléchargez et installez le compositeur :  
<https://getcomposer.org/download/>

* + 1. Sous Mac

Installez d'abord Brew, ouvrez un terminal et exécutez :  
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

Puis exécutez :  
**brew installer composer**

## CLI Symfony

### Sous Windows

Tout d'abord, installez Scoop en ouvrant un terminal et exécutez :  
***Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser***

***Invoke-RestMethod -Uri https://get.scoop.sh | Invoke-Expression***

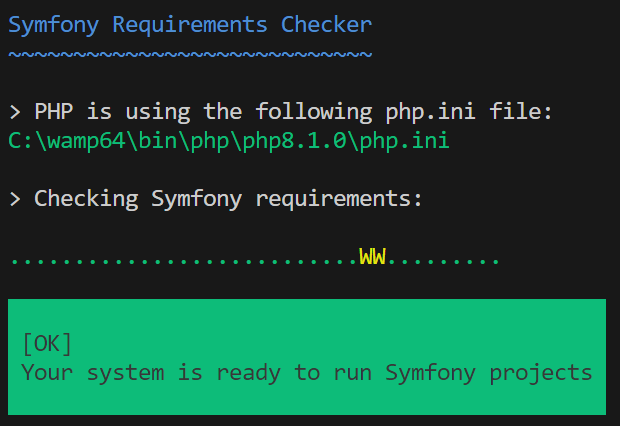
Puis exécutez :

**scoop install symfony-cli**

### Sous Mac

Dans l'exécution du terminal :  
**brew install symfony-cli/tap/symfony-cli**

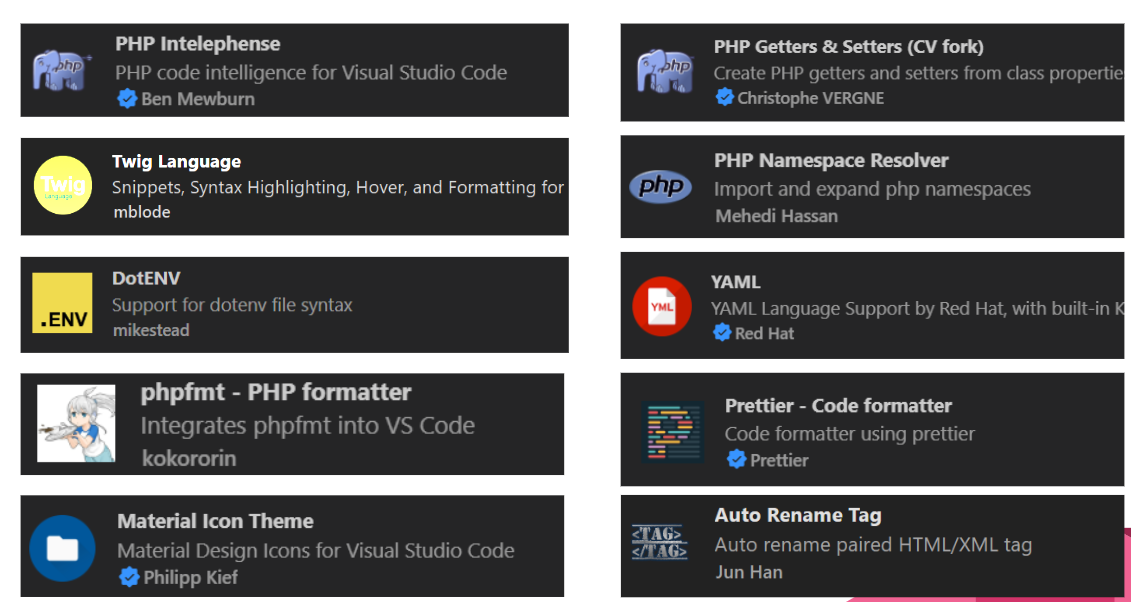
## Symfony Requirements Checker

Dans le terminal, exécutez la commande suivante :  
**symfony check:requirement**  
  
Le message [OK] indique que tous les prérequis sont respéctés.  


S'il n'est pas indiqué « Your system is ready », veuillez installer les dépendances manquantes.

## VSCODE

Installez les extensions suivantes pour VSCode :



# Créer le projet Symfony

Dans un terminal exécutez la commande suivante (vous pouvez remplacer all\_games par le nom de votre projet dans le cadre d’une autre application) :  
symfony new all\_games --version="6.4.\*" --webapp  
  
Ouvrez ensuite le dossier créé dans VSCode.

# Aperçu du projet

Nous allons créer une application Web similaire à Steam/Epic Games.

Nous aurons des **jeux** avec des **genres** (fps, sport, jeu de puzzle, tower defense etc.).

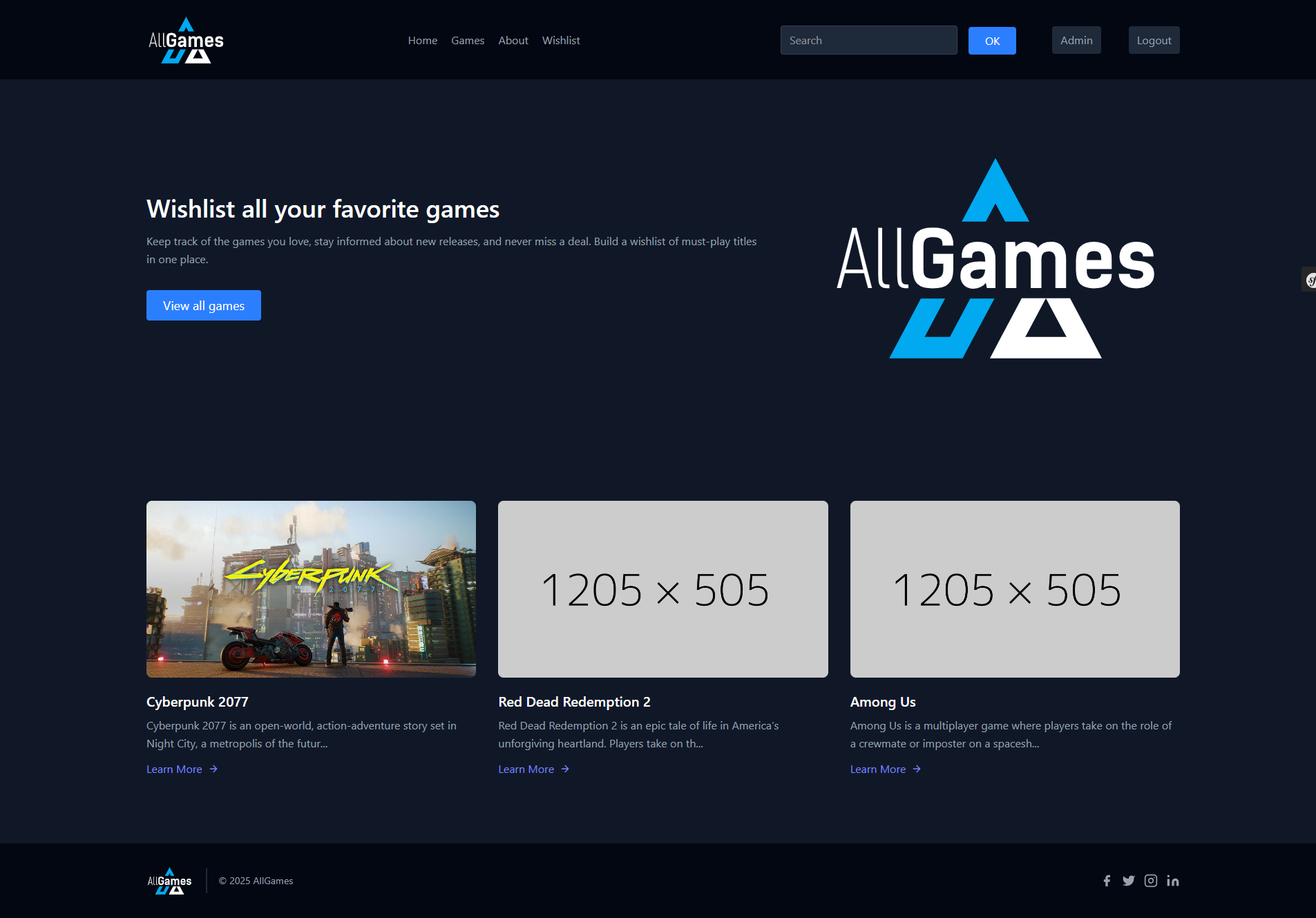
Les jeux peuvent avoir un **éditeur**.

Les **utilisateurs** (username, email, password) pourront ajouter des jeux à leur **liste de souhaits**.

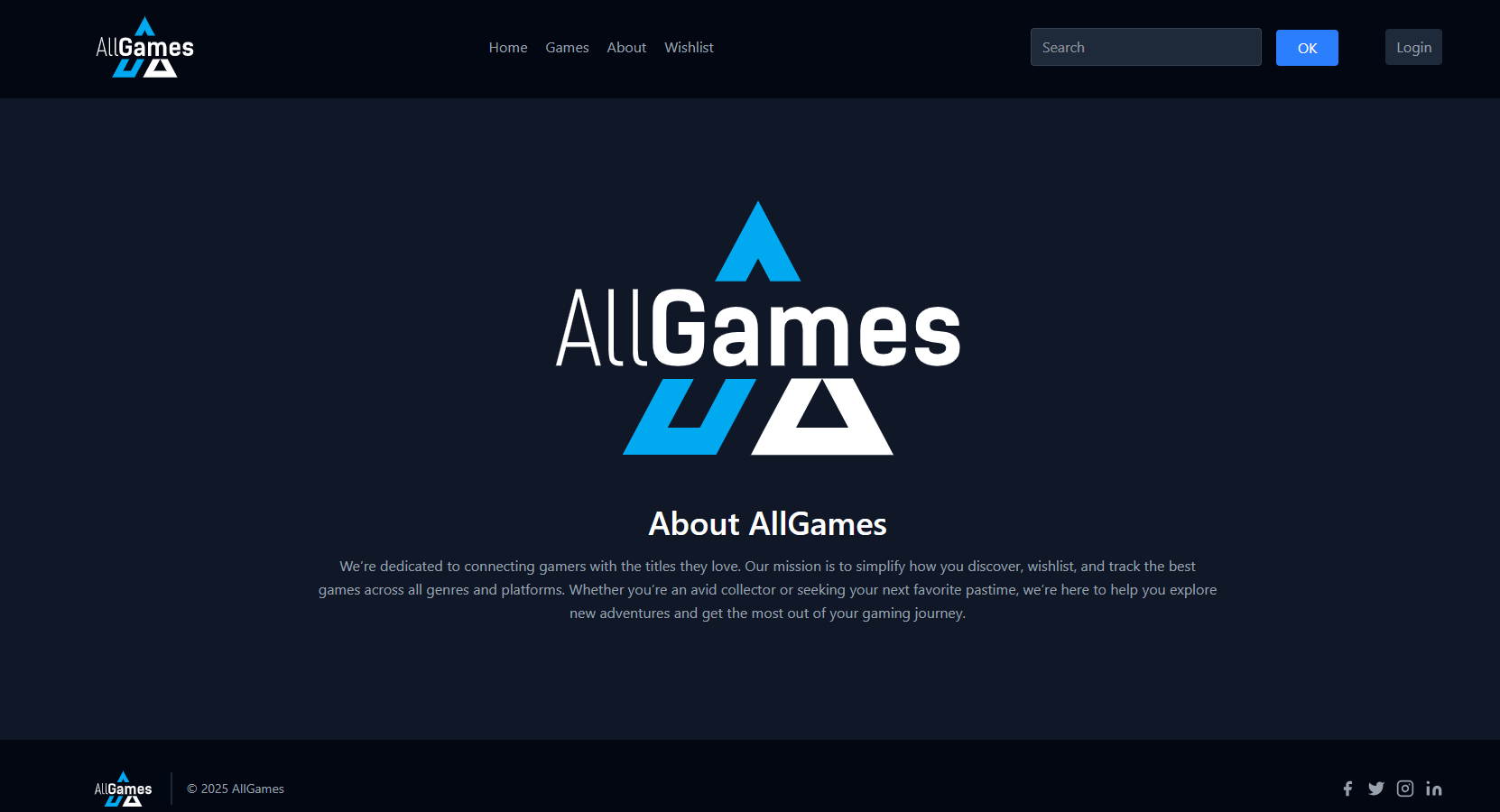
Les utilisateurs pourront donner un avis comme sur Steam (bool positif/négatif, commentaire).



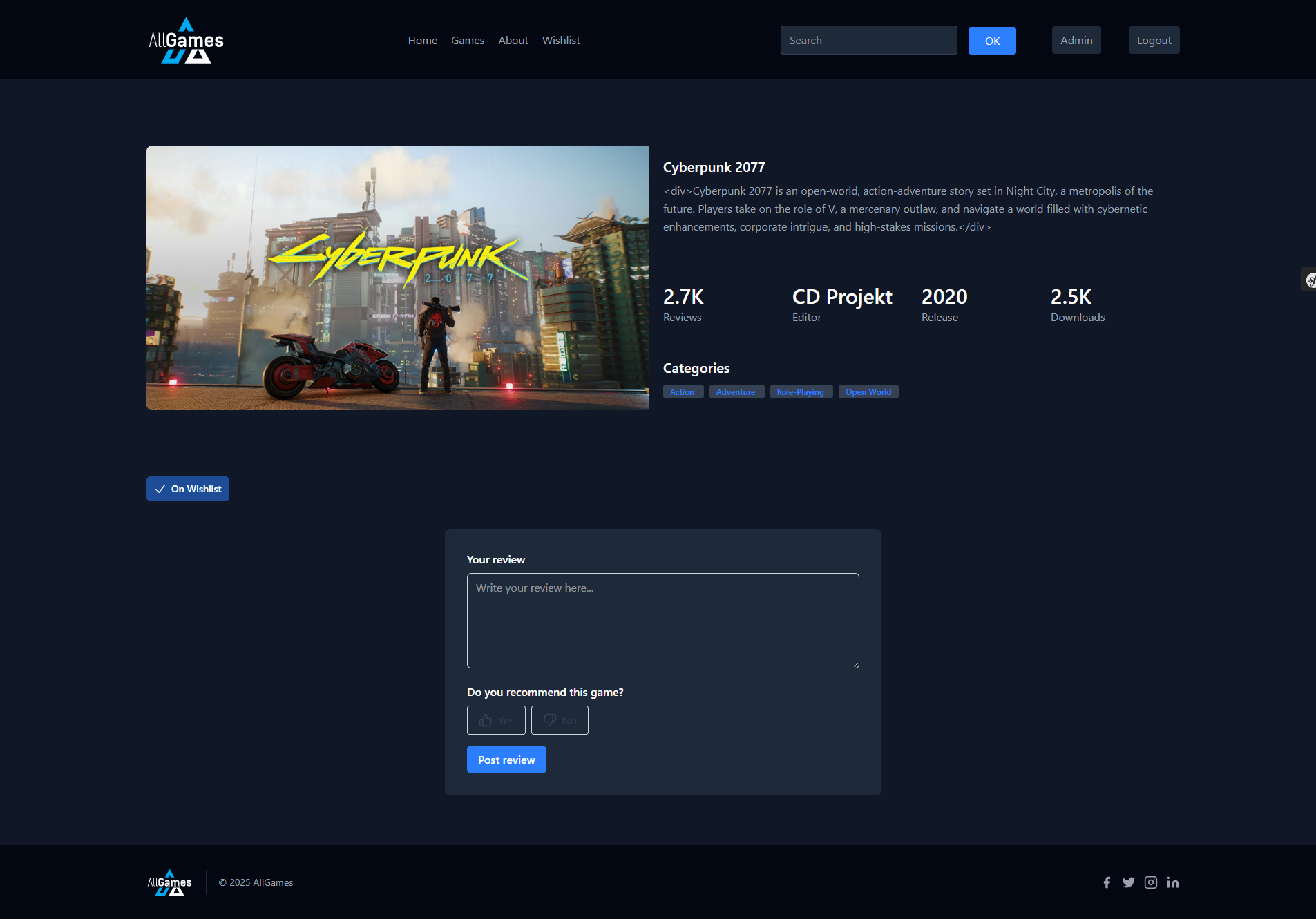
Page d’accueil:



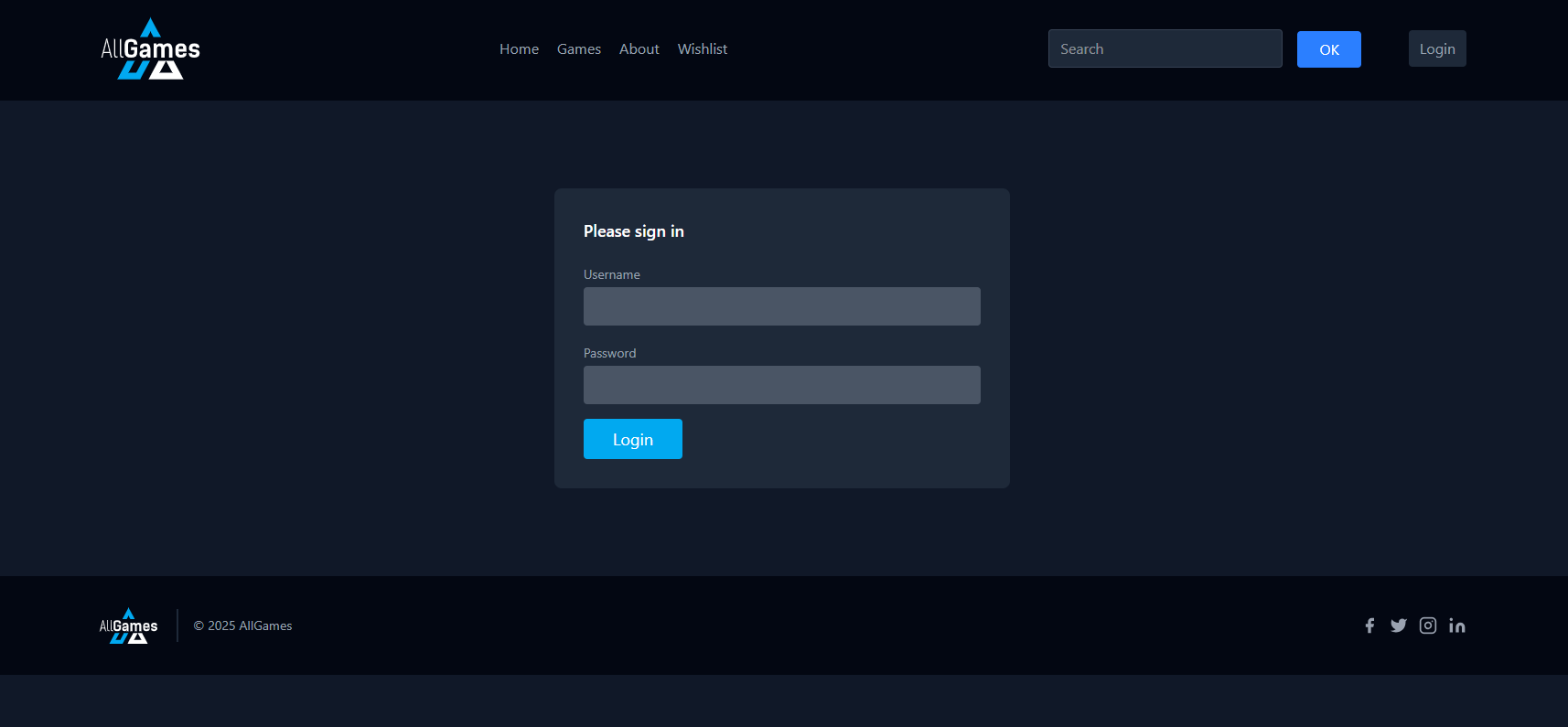
A propos:



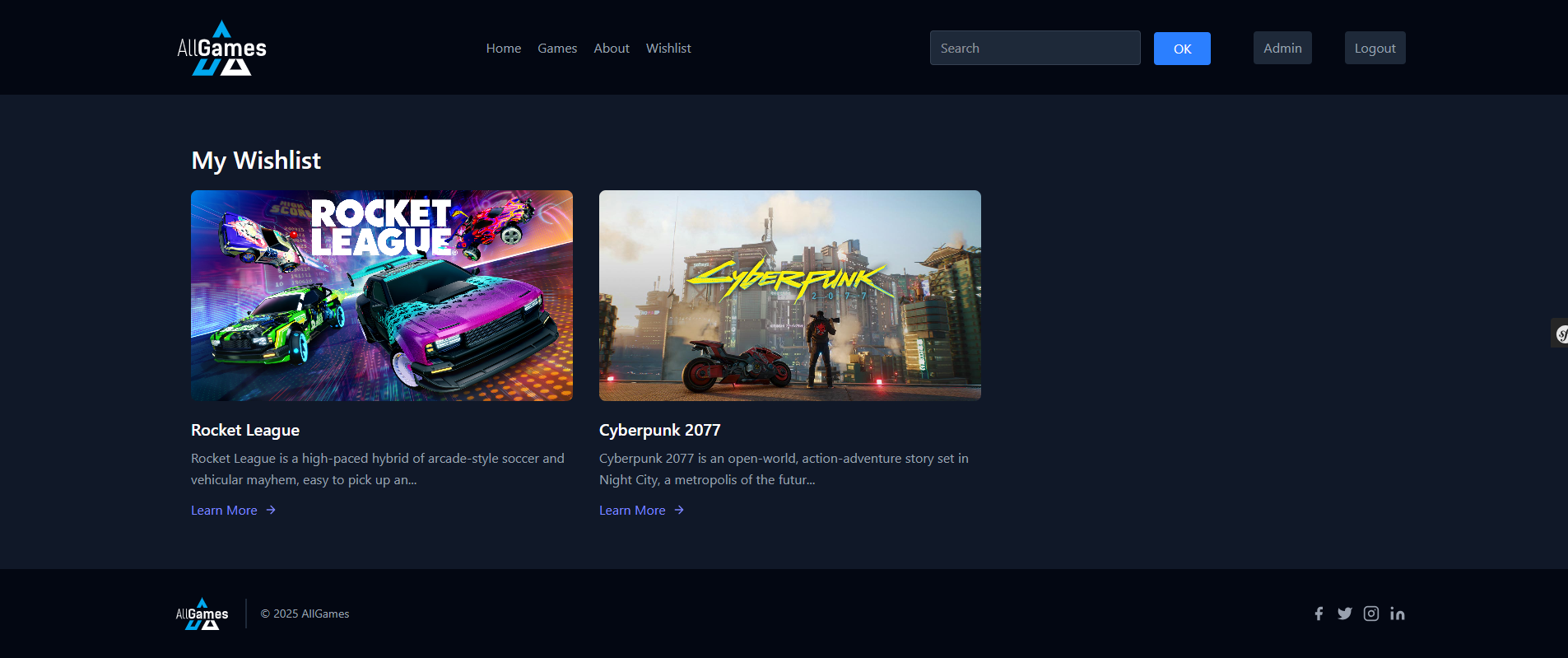
Détail d’un jeu



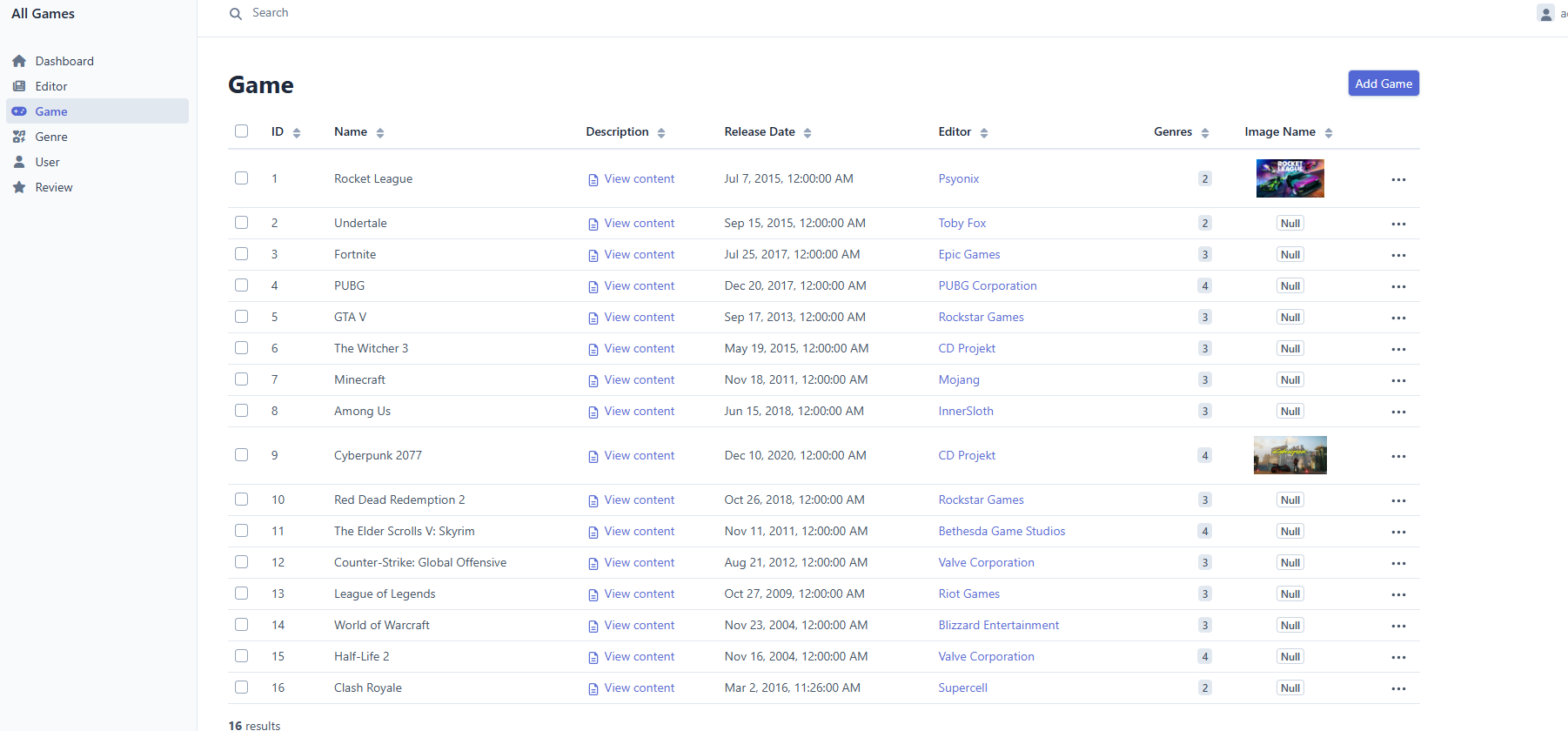
Login:



Affichage de la liste de souhaits:



Admin (easyadmin):

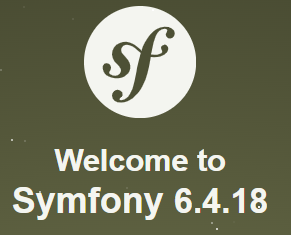


# Démarrer le serveur

Pour lancer un serveur pour exécuter notre application Symfony, nous pouvons exécuter la commande suivante :  
**symfony server:start**

Faites défiler vers le haut pour voir l’URL du serveur Web. Ensuite, vous pouvez copier-coller l'URL dans votre navigateur ou faire CTRL+Clic.

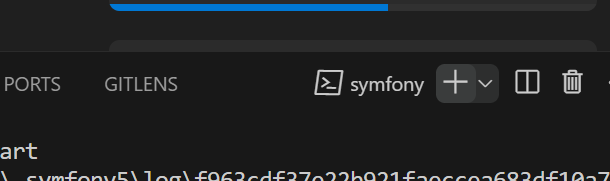
Vous devriez voir ceci dans votre navigateur :



Symfony affiche une page d'accueil temporaire, nous la modifierons lors de la prochaine étape.

# Contrôleur

Comme Symfony s'exécute dans notre terminal VSCode, nous ne pouvons saisir aucune commande.  
Il faut donc d'abord ajouter un autre terminal en cliquant sur le bouton + :



Créez votre premier Controller via la commande make:controller :

**symfony console make:controller PageController**

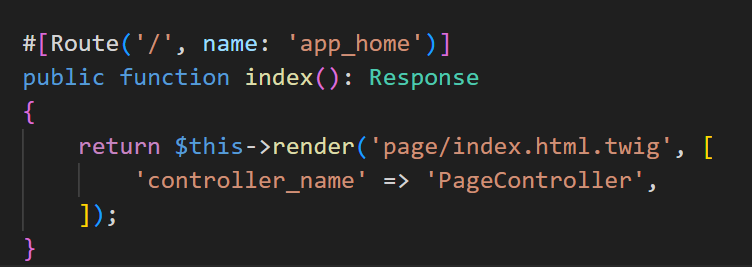
Nous utiliserons ce contrôleur pour les pages principales (accueil, à propos, mentions légales etc.).

Cette commande génère deux fichiers :  
src/Controller/PageController.php

templates/page/index.html.twig

Nous voulons maintenant modifier le PageController. Nous ouvrons PageController dans src -> Controller.

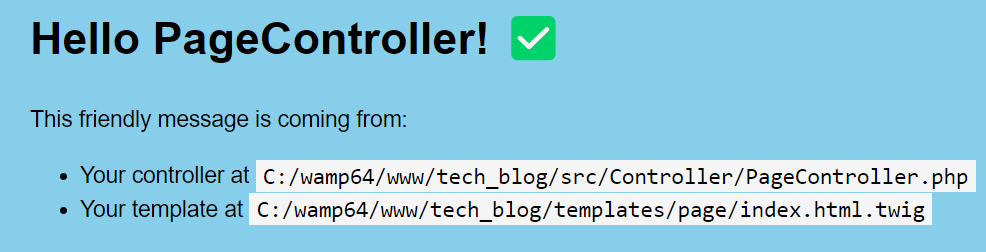
Nous voulons que la méthode index (action) soit la page d'accueil. Pour cela, nous devons changer la route « /page » en « / ».  
Nous pouvons également renommer « app\_page » en « app\_home »



Vous pouvez voir dans la méthode render :

* Le premier paramètre est le template que nous voulons charger
* Le deuxième paramètre est un tableau contenant toutes les variables que nous enverrons à la vue

Maintenant, si nous actualisons le navigateur, nous devrions avoir ceci :



# Templates: Twig

Twig est un moteur de template pour PHP, souvent utilisé avec le framework Symfony, bien qu'il puisse également être utilisé indépendamment.

Twig a été créé comme une alternative plus flexible, sécurisée et puissante aux templates PHP traditionnels.

Twig utilise son propre langage nous permettant d'utiliser

* variables
* conditions
* boucle
* etc.

## Afficher une variable

Pour afficher une variable provenant du contrôleur, on peut utiliser la syntaxe suivante en utilisant des doubles accolades :

**<p>{{ maVar }}</p>**

## if

L'instruction if de Twig est comparable aux instructions if de PHP.

{% if temperature > 15 et temperature < 30 %}

<p>C'est une belle journée pour une promenade dans le parc.</p>

{% else %}

<p>Mieux vaut rester à la maison</p>

{% endif %}

<https://twig.symfony.com/doc/3.x/tags/if.html>

## for

For permet de parcourir chaque élément d’un tableau. Par exemple, pour afficher une liste d'utilisateurs:

{% if utilisateurs %}

<ul>

{% for utilisateur dans utilisateurs %}

<li>{{ utilisateur.nom d'utilisateur }}</li>

{% endfor %}

</ul>

{% endif %}

<https://twig.symfony.com/doc/3.x/tags/for.html>

## include

L'instruction include inclut un template et affiche le contenu rendu de ce fichier :  
**{% include 'fichier.html.twig' %}**

<https://twig.symfony.com/doc/3.x/tags/include.html>

## block

Un bloc fait partie d'un template qui sera inclus dans l'enfant.

Chaque bloc peut être remplacé pour remplacer le contenu

**{% block titre %}Bonjour!{% endclock %}**

<https://twig.symfony.com/doc/3.x/tags/include.html>

## Afficher une image

Pour afficher une image, nous pouvons utiliser asset() pour renvoyer le chemin de l'image située dans le dossier assets.

<img width="120" src="{{ asset('images/logo.png') }}" alt="Nom de l'entreprise">

## Adaptation de la page d'accueil

Nous devons maintenant changer le html de cette page.  
Nous devons ouvrir templates/page/index.html.twig

Supprimons tout le code qui se trouve déjà entre le **block body**

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

{% block title %}Hello PageController!{% endblock %}

{% block body %}

{% endblock %}

Ajoutons maintenant du contenu à notre page d'accueil :

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

{% block title %}All Game{% endblock %}

{% block body %}

<h1>All Game</h1>

</div>

{% endblock %}

Maintenant, si vous actualisez la page, vous devriez voir votre titre.

## Ajout d'en-tête et de pied de page

Nous utiliserons le framework tailwind pour créer la page plus rapidement.  
  
Exécutez les commandes suivantes :  
composer require symfonycasts/tailwind-bundle

php bin/console tailwind:init

Le bundle fonctionne en modifiant automatiquement le contenu de styles/app.css avec le CSS compilé. Pour que cela fonctionne, vous devez exécuter la commande tailwind:build:

php bin/console tailwind:build --watch

Editez tailwind.config.js situé à la racine de votre projet pour qu'il corresponde à la couleur de la charte graphique :

theme: {

extend: {

colors: {

blue: {

500: '#01a9f0',

},

},

},

},

Téléchargez le logo :  
<https://drive.google.com/file/d/1eFdLhDSfRXtGbHWWSkHSrXn4ULEBlBqU/view?usp=sharing>

Dans le dossier assets, créez un dossier images et copiez-y le logo.

Pour ajouter un header et footer, nous allons modifier **templates/base.html.twig**

Ajoutez la classe class="bg-gray-900" sur le body:

<body class="bg-gray-900">

Dans ce projet, nous utiliserons tailwind et <https://tailblocks.cc/>.

Dans la balise body, nous pouvons voir le bloc twig body :

{% block body %}{% endblock %}

Avant le bloc de corps, insérons l'en-tête

<header class="text-gray-400 bg-gray-950 body-font">

<div class="container mx-auto flex flex-wrap p-5 flex-col md:flex-row items-center">

<a class="flex title-font font-medium items-center text-white mb-4 md:mb-0" href="#">

<img width="120" src="{{ asset('images/logo-all-games.svg') }}" alt="AllGames">

</a>

<nav class="md:ml-auto md:mr-auto flex flex-wrap items-center text-base justify-center">

<a class="mr-5 hover:text-white" href="#">Home</a>

<a class="mr-5 hover:text-white" href="#">Games</a>

<a class="mr-5 hover:text-white" href="#">About</a>

<a class="mr-5 hover:text-white" href="#">Contact</a>

</nav>

<div class="flex md:flex-nowrap flex-wrap justify-center items-end md:justify-start px-3">

<div class="relative sm:w-64 w-40 sm:mr-4 mr-2">

<input placeholder="Search" ="text" id="footer-field" name="footer-field" class="w-full bg-gray-800 bg-opacity-40 rounded border border-gray-700 focus:ring-2 focus:ring-blue-900 focus:bg-transparent focus:border-blue-500 text-base outline-none text-gray-100 py-1 px-3 leading-8 transition-colors duration-200 ease-in-out">

</div>

<button class="inline-flex text-white bg-blue-500 border-0 py-2 px-6 focus:outline-none hover:bg-blue-600 rounded">OK</button>

</div>

<a class="inline-flex items-center bg-gray-800 border-0 py-2 px-3 ml-10 focus:outline-none hover:bg-gray-700 rounded text-base mt-4 md:mt-0" href="#">Login

</a>

</div>

</header>

Après le bloc de corps, nous voulons ajouter le pied de page :

<footer class="text-gray-400 bg-gray-950 body-font">

<div class="container px-5 py-8 mx-auto flex items-center sm:flex-row flex-col">

<a class="flex title-font font-medium items-center md:justify-start justify-center text-white" href="#">

<img width="70" src="{{ asset('images/logo-all-games.svg') }}" alt="AllGames">

</a>

<p class="text-sm text-gray-400 sm:ml-4 sm:pl-4 sm:border-l-2 sm:border-gray-800 sm:py-2 sm:mt-0 mt-4">© {{ 'now' | date('Y') }} AllGames

</p>

<span class="inline-flex sm:ml-auto sm:mt-0 mt-4 justify-center sm:justify-start">

<a class="text-gray-400">

<svg fill="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" class="w-5 h-5" viewBox="0 0 24 24">

<path d="M18 2h-3a5 5 0 00-5 5v3H7v4h3v8h4v-8h3l1-4h-4V7a1 1 0 011-1h3z"></path>

</svg>

</a>

<a class="ml-3 text-gray-400">

<svg fill="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" class="w-5 h-5" viewBox="0 0 24 24">

<path d="M23 3a10.9 10.9 0 01-3.14 1.53 4.48 4.48 0 00-7.86 3v1A10.66 10.66 0 013 4s-4 9 5 13a11.64 11.64 0 01-7 2c9 5 20 0 20-11.5a4.5 4.5 0 00-.08-.83A7.72 7.72 0 0023 3z"></path>

</svg>

</a>

<a class="ml-3 text-gray-400">

<svg fill="none" stroke="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" class="w-5 h-5" viewBox="0 0 24 24">

<rect width="20" height="20" x="2" y="2" rx="5" ry="5"></rect>

<path d="M16 11.37A4 4 0 1112.63 8 4 4 0 0116 11.37zm1.5-4.87h.01"></path>

</svg>

</a>

<a class="ml-3 text-gray-400">

<svg fill="currentColor" stroke="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="0" class="w-5 h-5" viewBox="0 0 24 24">

<path stroke="none" d="M16 8a6 6 0 016 6v7h-4v-7a2 2 0 00-2-2 2 2 0 00-2 2v7h-4v-7a6 6 0 016-6zM2 9h4v12H2z"></path>

<circle cx="4" cy="4" r="2" stroke="none"></circle>

</svg>

</a>

</span>

</div>

</footer>

Modifions maintenant le contenu de la page d'accueil.  
Remplacez tout le code d’index.html.twig par celui-ci :

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

{% block title %}AllGames - Wishlist all your favorite games{% endblock %}

{% block body %}

<section class="text-gray-400 bg-gray-900 body-font">

<div class="container mx-auto flex px-5 py-24 md:flex-row flex-col items-center">

<div class="lg:flex-grow md:w-1/2 lg:pr-24 md:pr-16 flex flex-col md:items-start md:text-left mb-16 md:mb-0 items-center text-center">

<h1 class="title-font sm:text-4xl text-3xl mb-4 font-medium text-white">Wishlist all your favorite games

</h1>

<p class="mb-8 leading-relaxed">Keep track of the games you love, stay informed about new releases, and never miss a deal. Build a wishlist of must-play titles in one place.</p>

<div class="flex justify-center">

<a class="inline-flex text-white bg-blue-500 border-0 py-2 px-6 focus:outline-none hover:bg-blue-600 rounded text-lg">View all games</a>

</div>

</div>

<div class="lg:max-w-lg lg:w-full md:w-1/2 w-5/6">

<img class="object-cover object-center rounded" src="{{ asset('images/logo-all-games.svg') }}" alt="AllGames">

</div>

</div>

</section>

{% endblock %}

# Créer la page à propos

Pour créer la page à propos, nous devons d'abord modifier PageController.php (dans src/Controller).

On va ajouter la méthode suivante:

#[Route('/about', name: 'app\_about')]

public function about(): Response

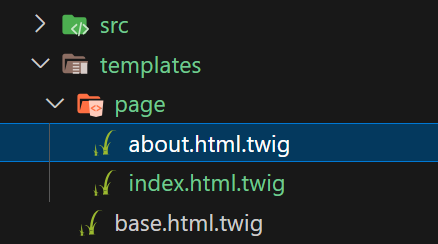
{

return $this->render('page/about.html.twig', [

]);

}

On doit maintenant créer un fichier about.html.twig dans templates/page



Ajouter le code suivant (ou composer la page vous même avec tailblock)

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

{% block title %}About us - AllGames{% endblock %}

{% block body %}

<section class="text-gray-400 bg-gray-900 body-font">

<div class="container mx-auto flex px-5 py-24 items-center justify-center flex-col">

<img

class="lg:w-2/6 md:w-3/6 w-5/6 mb-10 object-cover object-center rounded"

src="{{ asset('images/logo-all-games.svg') }}"

alt="AllGames" />

<div class="text-center lg:w-2/3 w-full">

<h1 class="title-font sm:text-4xl text-3xl mb-4 font-medium text-white">

A propos

</h1>

<p class="leading-relaxed mb-8">

Nous nous engageons à connecter les joueurs avec les titres qu'ils aiment. Notre

la mission est de simplifier la façon dont vous découvrez, listez vos souhaits et suivez les meilleurs

des jeux dans tous les genres et sur toutes les plateformes. Que vous soyez un collectionneur passionné

ou à la recherche de votre prochain passe-temps favori, nous sommes là pour vous aider à explorer

de nouvelles aventures et tirez le meilleur parti de votre parcours de jeu.

</p>

</div>

</div>

</section>

{% endblock %}

Modifions maintenant le menu pour pouvoir cliquer sur le lien à propos.  
Editez le fichier templates/base.html.twig  
  
Dans le href du lien à propos, remplacez le # par ce qui suit :

{{ path('app\_about') }}

Le lien devrait ressembler à ceci :

<a class="mr-5 hover:text-white" href="{{ path('app\_about') }}">A propos</a>

Faites de même pour le lien d'accueil :

<a class="mr-5 hover:text-white" href="{{ path('app\_home') }}">Accueil</a>

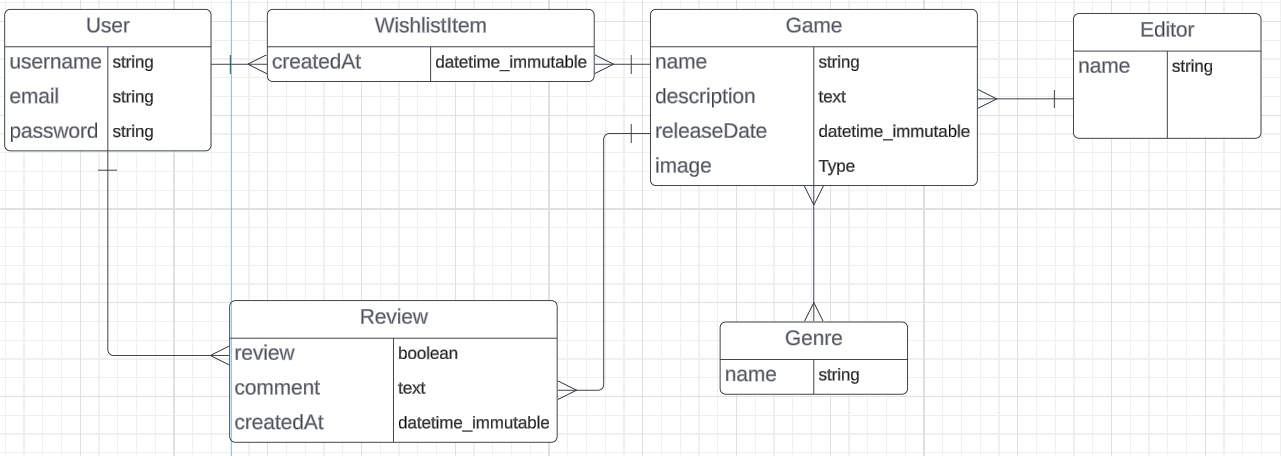
# Créer la base de données

Allez sur phpmyadmin et créez une nouvelle base de données appelée all\_games

Ajoutez ensuite le fichier un fichier .env.local et avec la configuration adaptée à votre environnement:

DATABASE\_URL="mysql://root:@127.0.0.1:3306/all\_game?serverVersion=5.7.36&charset=utf8mb4"

## Créer les entités

Nous devons maintenant créer toutes les entités basées sur le modèle conceptuel :  


Symfony créera pour nous des tables de jointure afin que nous n'ayons qu'à nous concentrer sur les entités principales.  
Nous n’avons également **pas** besoin de créer les clés primaires (id) et les clés étrangères.

Commençons par l'entité User qui est un peu particulière en utilisant la commande suivante :

symfony console make:user User

Et répondez comme ci-dessous :

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:

> yes

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:

> username

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).

Does this app need to hash/check user passwords? (yes/no) [yes]:

> yes

Ensuite, nous devons ajouter l'email sur l'entité Utilisateur :

symfony console make:entity User

Your entity already exists! So let's add some new fields!

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

> email

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

Pour chaque autre entité, exécutez cette commande :

symfony console make:entity

Entrez le nom de l'entité, commençons par Editor :

Class name of the entity to create or update (e.g. AgreeablePizza):

> Editor

Étape suivante, nous appuyons simplement sur Entrée :

Add the ability to broadcast entity updates using Symfony UX Turbo? (yes/no) [no]:

>

Nous ajoutons notre propriété (en minuscule) :

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

> name

Nous devons spécifier le type, s'il s'agit d'une chaîne, nous pouvons simplement appuyer sur Entrée :

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

>

Pour la longueur, si la valeur entre parenthèses est bonne on peut appuyer sur Entrée :

Field length [255]:

>

Si le champ est obligatoire, on peut appuyer sur Entrée :

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

>

Nous pouvons continuer à ajouter toutes les propriétés de cette façon.

Lorsque vous avez terminé, lorsque qu’il est demandé si nous souhaitons ajouter d’autres champs, appuyons sur Entrée pour quitter.

Faisons de même pour Game. Mais ici, vous devez ajouter 3 champs et un de plus pour la relation.  
  
Lorsque vous avez ajouté le nom, la description et la date de publication ainsi qu'un champ supplémentaire appelé éditeur :

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

> editor

Il faut préciser le type, type relation :

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

> **relation**

What class should this entity be related to?:

> **Editor**

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:

> **ManyToOne**

Is the Game.editor property allowed to be null (nullable)? (yes/no) [yes]:

> yes

Do you want to add a new property to Editor so that you can access/update Game objects from it - e.g. $editor->getGames()? (yes/no) [yes]:

> yes

New field name inside Editor [games]:

> games

⚠️Répétez ce processus pour **toutes les entités du projet et n'oubliez pas la relation.**

# Créer la migration et migrer

Notre projet contient maintenant toutes les entités avec les différents attributs.

Ensuite, nous devons créer les tables de base de données liées à ces entités PHP.

Doctrine Migrations est la solution idéale pour cela. Le paquet a déjà été installé dans le cadre de la dépendance orm.

Une migration est une classe qui décrit les changements nécessaires pour mettre à jour un schéma de base de données, de son état actuel vers le nouveau, en fonction des attributs de l'entité. Comme la base de données est vide pour l'instant, la migration devrait consister en la création de deux tables.

Nous devons maintenant créer le fichier de migration avec cette commande :

symfony console make:migration

Le fichier généré se trouve dans le dossier migrations. Vous pouvez jeter un oeil pour voir le code.

Nous devons maintenant exécuter cette migration pour créer toutes les tables.

symfony console doctrine:migrations:migrate

# Ajouter des données

Pour ajouter des données dans la base de données afin d'avoir un ensemble de données de départ, nous utiliserons des appareils.  
  
Exécutez la commande suivante pour installer le bundle :

composer require --dev orm-fixtures

Téléchargez les fichiers des appareils et placez-les dans votre dossier src DataFixtures :  
<https://drive.google.com/drive/folders/11fzunW5rzy_PRvU8s6Up5DUsajcG7pSy?usp=sharing>

Chargez ensuite les appareils avec la commande suivante

symfony console doctrine:fixtures:load

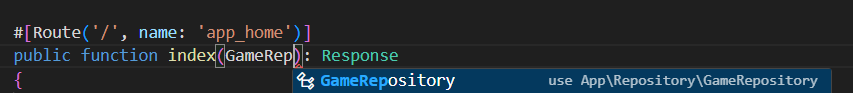
Dites oui, soyez prudent, cela supprimera toutes les données si vous les avez déjà.

Revenez maintenant à phpmyadmin et vous devriez voir les données dans les tableaux du jeu, de l'éditeur et des genres.

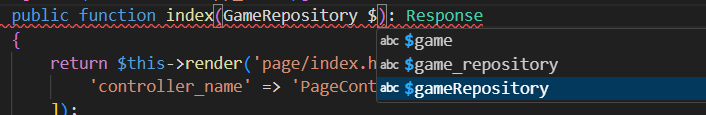
# Afficher les derniers jeux sur la page d'accueil

Dans VSCode, ouvrez PageController (src/Controller).

À l’intérieur du support d’index(), commencez à taper GameRep et cliquez sur GameRepository. Cela importera la dépendance (use) :



Mettez ensuite un espace et $ et cliquez sur $gameRepository :



Avant le render, nous souhaitons récupérer les 3 derniers jeux :

$games = $gameRepository->findBy([], ['releaseDate' => 'DESC'], 3);

Ensuite, nous avons besoin des paramètres de la méthode render pour passer les jeux :

return $this->render('page/index.html.twig', [

'games' => $games

]);

Modifions maintenant le page/index.html.twig et ajoutons une section avant la fin du bloc body :

<section class="text-gray-400 bg-gray-900 body-font">

<div class="container px-5 py-24 mx-auto">

<div class="flex flex-wrap sm:-m-4 -mx-4 -mb-10 -mt-4">

{% for game in games %}

{% include 'game/\_game\_item.html.twig' with {'game': game} %}

{% endfor %}

</div>

</div>

</section>

Ce code parcourt chaque jeu et inclut un fichier que nous devons créer.  
Dans le dossier templates, créez un dossier **game** et dans ce dossier, créez un fichier **\_game\_item.html.twig**.

Comme ce fichier ne sera pas un template complet mais seulement un morceau de code, nous commençons le nom par un underscore. Nous appelons ce type de fichier un “partial”.  
Nous pourrons réutiliser ce partial sur la page liste des jeux.

Dans ce fichier, ajoutez le code suivant :

<div class="p-4 md:w-1/3 sm:mb-0 mb-6">

<div class="rounded-lg h-64 overflow-hidden">

<img alt="content" class="object-cover object-center h-full w-full" src="https://dummyimage.com/1205x505">

</div>

<h2 class="text-xl font-medium title-font text-white mt-5">{{ game.name }}</h2>

<!-- We need to truncate the description and add ... at the end -->

<p class="text-base leading-relaxed mt-2">{{ game.description|slice(0, 100) }}...</p>

<a class="text-indigo-400 inline-flex items-center mt-3">Learn More

<svg fill="none" stroke="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" class="w-4 h-4 ml-2" viewBox="0 0 24 24">

<path d="M5 12h14M12 5l7 7-7 7"></path>

</svg>

</a>

</div>

Actualisez la page d'accueil et voyez le résultat. Comme nous ne gérons pas encore le téléchargement des images, nous avons un espace réservé pour l'image.

# Création d'un backend d'administration

## Installation d'EasyAdmin

Pour créer le backend d’administration, nous utiliserons EasyAdmin Bundle.  
  
Dans le terminal, exécutez la commande suivante :

composer require easycorp/easyadmin-bundle

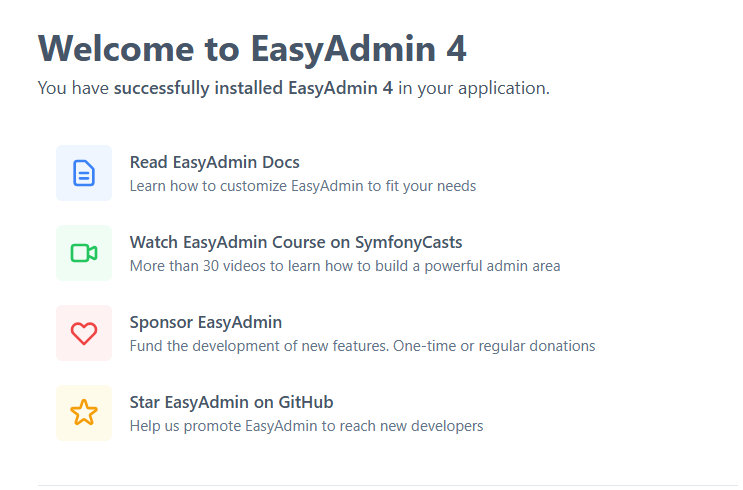
L'étape suivante consiste à créer le tableau de bord d'administration :

symfony console make:admin:dashboard

Appuyez sur Entrée pour toutes les étapes afin d'utiliser les paramètres par défaut

Allez maintenant dans l'admin (changez l'ip en fonction de votre projet) :  
<https://127.0.0.1:8000/admin>

Vous devriez voir une page d'accueil :



## Modifier le tableau de bord

Ouvrez le fichier suivant :

src/Controller/Admin/DashboardController.php

Dans la méthode index(), supprimez tout le code pour retourner un template.

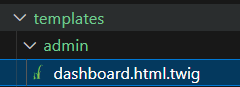
public function index(): Response

{

return $this->render('admin/dashboard.html.twig');

}

Dans le dossier templates, créez un dossier admin et à l'intérieur créez un fichier dashboard.html.twig



Dans ce fichier, ajoutez ce qui suit :

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

{% block content\_title %}Dashboard{% endblock %}

{% block main %}

<h2>Bienvenue dans l'administration AllGames !</h2>

{% endblock %}

## Sécurisez l'administrateur

Comme vous pouvez le constater, tout le monde peut accéder à l’admin.

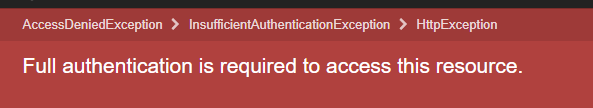
Editez le fichier suivant :  
config/packages/security.yaml

Faites défiler jusqu'à la partie access\_control et décommentez la ligne:

access\_control:

- { path: ^/admin, roles: ROLE\_ADMIN }

Actualisez maintenant la page d'administration, vous devriez cette erreur :



## Ajouter un système de connexion

Nous devons d’abord créer et administrer la base de données. Mais pour cela, nous avons besoin d’un hachage de mot de passe.  
  
Exécutez la commande suivante :

symfony console security:hash-password

Tapez le mot de passe de votre choix puis copiez le hachage du mot de passe.

Dans phpmyadmin, créez un nouvel utilisateur et ajoutez toutes les données.  
Pour le mot de passe, collez le hachage.  
Pour le rôle, mettez ceci :  
["ROLE\_ADMIN"]

Nous voulons maintenant avoir un formulaire de connexion.  
Exécutez la commande suivante :

symfony console make:security:form-login

Appuyez sur Entrée pour toutes les questions

Si vous actualisez l'administrateur, vous devriez être redirigé vers le formulaire de connexion :login



Accédez au fichier suivant :  
templates/security/login.html.twig

Remplacez le bloc body par ce code :

{% block body %}

<section class="text-gray-400 bg-gray-900 body-font">

<div class="container px-5 py-24 mx-auto flex flex-wrap items-center">

<div class="lg:w-2/6 md:w-1/2 mx-auto bg-gray-800 bg-opacity-50 rounded-lg p-8 flex flex-col md:ml-auto w-full mt-10 md:mt-0">

{% if app.user %}

<div class="text-white text-lg font-medium title-font mb-5">Vous êtes connecté en tant que {{ app.user.userIdentifier }}, <a class="text-blue-500" href="{{ path('app\_logout') }}">Logout</a></div>

{% else %}

<form method="post">

{% if error %}

<div class="p-4 mb-4 text-sm text-red-400 rounded-lg bg-gray-800 dark:text-red-400" role="alert">

<span class="font-medium">{{ error.messageKey|trans(error.messageData, 'security') }}</span>

</div>

{% endif %}

<h2 class="text-white text-lg font-medium title-font mb-5">Connexion</h2>

<div class="relative mb-4">

<label for="username" class="leading-7 text-sm text-gray-400">Pseudo</label>

<input type="text" value="{{ last\_username }}" name="\_username" id="username" class="w-full bg-gray-600 bg-opacity-20 focus:bg-transparent focus:ring-2 focus:ring-indigo-900 rounded border border-gray-600 focus:border-indigo-500 text-base outline-none text-gray-100 py-1 px-3 leading-8 transition-colors duration-200 ease-in-out" autocomplete="username" required autofocus>

</div>

<div class="relative mb-4">

<label for="password" class="leading-7 text-sm text-gray-400">Mot de passe</label>

<input type="password" name="\_password" id="password" class="w-full bg-gray-600 bg-opacity-20 focus:bg-transparent focus:ring-2 focus:ring-indigo-900 rounded border border-gray-600 focus:border-indigo-500 text-base outline-none text-gray-100 py-1 px-3 leading-8 transition-colors duration-200 ease-in-out" autocomplete="current-password" required>

</div>

<input type="hidden" name="\_csrf\_token"

value="{{ csrf\_token('authenticate') }}"

>

<button type="submit" class="text-white bg-indigo-500 border-0 py-2 px-8 focus:outline-none hover:bg-indigo-600 rounded text-lg">Se connecter</button>

</form>

{% endif %}

</div>

</div>

</section>

{% endblock %}

## Ajouter un CRUD

Pour générer un CRUD pour une entité, exécutez la commande suivante

php bin/console make:admin:crud

Appuyez sur Entrée pour toutes les étapes.

Pour ajouter ce CRUD au menu, éditez DashboardController.php et modifiez la méthode configureMenuItems :

public function configureMenuItems(): iterable

{

yield MenuItem::linkToDashboard('Dashboard', 'fa fa-newspaper');

yield MenuItem::linkToCrud('Editor', 'fas fa-list', Editor::class);

}

## Personnaliser les CRUD

Pour les entités ayant une relation telle que Game, nous devons personnaliser le CRUD.

Modifier:

src/Controller/Admin/GameCrudController.php

Voici les configureFields avec les associations :

public function configureFields(string $pageName): iterable

{

return [

IdField::new('id')->hideOnForm(),

TextField::new('name'),

TextEditorField::new('description')->setSortable(true),

DateTimeField::new('releaseDate'),

AssociationField::new('editor')->autocomplete(),

AssociationField::new('genres')->autocomplete(),

];

}

Si vous essayez de modifier un jeu, vous obtiendrez une erreur car nous n'avons pas ajouté la méthode magique \_\_toString dans l'éditeur et le genre.  
  
Modifiez les entités Editor et Genre et ajoutez les éléments suivants :

public function \_\_toString(): string

{

return $this->name;

}

## User CRUD

Si vous essayez d'utiliser l'utilisateur CRUD, vous verrez que le mot de passe est enregistré sous forme de texte brut.  
  
Pour modifier ce comportement, suivez les étapes décrites dans le lien suivant :

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

Vous devrez également ajouter tous les champs nécessaires tels que le nom d'utilisateur et les rôles :

$fields = [

IdField::new('id')->hideOnForm(),

EmailField::new('email'),

TextField::new('username'),

ChoiceField::new('roles')

->setChoices(['ROLE\_ADMIN' => 'ROLE\_ADMIN', 'ROLE\_USER' => 'ROLE\_USER'])

->allowMultipleChoices()

->renderExpanded(),

];

## Autoriser le téléchargement d'images

Nous voulons maintenant pouvoir télécharger une image pour notre jeu.  
  
Exécutez la commande suivante :

composer require vich/uploader-bundle

Si on vous demande “Do you want to execute this recipe?”», appuyez sur **y**

Cela installera le bundle vich/upload.

Dans assets/images, créez un nouveau dossier games ».

Editez le fichier suivant :  
config/packages/vich\_uploader.yaml

Décommentez les mappings et renommez les produits par jeux :

mappings:

**games**:

uri\_prefix: /images/**games**

upload\_destination: '%kernel.project\_dir%/public/images/**games'**

namer: Vich\UploaderBundle\Naming\SmartUniqueNamer

Suivez les étapes de la documentation :  
<https://github.com/dustin10/VichUploaderBundle/blob/master/docs/usage.md#step-1-configure-an-upload-mapping>

Puis exécuter  
**symfony console make:migration**  
  
et  
**symfony console d:m:m**

Il faut ensuite modifier GameCrudController pour gérer l'image :

public function configureFields(string $pageName): iterable

{

$mappingsParams = $this->getParameter('vich\_uploader.mappings');

$gamesImagePath = $mappingsParams['games']['uri\_prefix'];

return [

IdField::new('id')->hideOnForm(),

TextField::new('name'),

TextEditorField::new('description')->setSortable(true),

DateTimeField::new('releaseDate'),

AssociationField::new('editor')->autocomplete(),

AssociationField::new('genres')->autocomplete(),

TextareaField::new('imageFile')->setFormType(VichImageType::class)->hideOnIndex(),

ImageField::new('imageName')->setBasePath($gamesImagePath)->hideOnForm(),

];

}

N'oubliez pas d'importer toutes les classes (CTRL+Espace puis cliquez sur la suggestion).

Essayez d'ajouter une image au dernier jeu sorti puis mettez à jour templates/game/\_game\_item.html.twig et remplacez l'image par

<img src="{% if game.imageName %}{{ vich\_uploader\_asset(game, 'imageFile') }}{% else %}https://dummyimage.com/1205x505{% endif %}" class="object-cover object-center h-full w-full" alt="{{ game.name }}">

# Mise à jour du menu

{% if app.user %}

<a class="inline-flex items-center bg-gray-800 border-0 py-2 px-3 ml-10 focus:outline-none hover:bg-gray-700 rounded text-base mt-4 md:mt-0" href="{{ path('app\_logout') }}">Déconnexion</a>

{% else %}

<a class="inline-flex items-center bg-gray-800 border-0 py-2 px-3 ml-10 focus:outline-none hover:bg-gray-700 rounded text-base mt-4 md:mt-0" href="{{ path('app\_login') }}">Connexion </a>

{% endif %}

{% if is\_granted('ROLE\_ADMIN') %}

<a class="inline-flex items-center bg-gray-800 border-0 py-2 px-3 ml-10 focus:outline-none hover:bg-gray-700 rounded text-base mt-4 md:mt-0" href="{{ path('admin') }}">Admin</a>

{% endif %}

# Liste des jeux

Nous allons maintenant créer un contrôleur Game.  
  
Exécutez la commande suivante :  
symfony console make:controller GameController

Dans GameController.php, supprimez la méthode index et ajoutez ce qui suit à la place

#[Route('/game/list', name: 'app\_game\_list')]

public function list(GameRepository $gameRepository): Response

{

//get all games

$games = $gameRepository->findAll();

return $this->render('game/list.html.twig', [

'games' => $games,

]);

}

Dans templates/game, renommez le fichier index.html.twig en list.html.twig.  
  
Ajoutez le code suivant :

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

{% block title %}Games list{% endblock %}

{% block body %}

<section class="text-gray-400 bg-gray-900 body-font container px-5 py-14 mx-auto">

<div class="container px-5 py-24 mx-auto">

<div class="flex flex-wrap sm:-m-4 -mx-4 -mb-10 -mt-4">

{% for game in games %}

{% include 'game/\_game\_item.html.twig' with {'game': game} %}

{% endfor %}

</div>

</div>

</section>

{% endblock %}

Essayez maintenant d'accéder à la page dans votre navigateur :  
<https://127.0.0.1:8000/game/list>

Vous devriez voir la liste des jeux.

Nous devons maintenant mettre à jour le menu pour pouvoir accéder à cette page. Editez templates/base.html.twig et modifiez le lien Games :

<a class="mr-5 hover:text-white" href="{{ path('app\_game\_list') }}">Games</a>

# Détails du jeu

Nous souhaitons permettre à l'utilisateur de pouvoir cliquer sur le lien « En savoir plus » pour voir les détails complets du jeu.  
  
Nous devons d’abord ajouter une nouvelle méthode à notre GameController.php.

#[Route('/game/{id}', name: 'app\_game\_show')]

public function show(Game $game): Response

{

return $this->render('game/show.html.twig', [

'game' => $game,

]);

}

Dans templates/jeu, créez un nouveau fichier show.html.twig

Ajoutez le code suivant :

{% extends "base.html.twig" %}

{% block title %}{{ game.name }}{% endblock %}

{% block body %}

<section class="text-gray-400 bg-gray-900 body-font">

<div class="container px-5 py-24 mx-auto flex flex-wrap">

<div class="lg:w-1/2 sm:w-1/3 w-full rounded-lg overflow-hidden mt-6 sm:mt-0 sm:pr-5">

<img src="{% if game.imageName %}{{ vich\_uploader\_asset(game, 'imageFile') }}{% else %}https://dummyimage.com/1205x505{% endif %}" class="object-cover object-center w-full h-full" alt="{{ game.name }}">

</div>

<div class="flex flex-wrap -mx-4 lg:w-1/2 sm:w-2/3 content-between">

<div class="w-full sm:p-4 px-4 mb-6">

<h1 class="title-font font-medium text-xl mb-2 text-white">{{ game.name }}</h1>

<div class="leading-relaxed">

{{ game.description|raw }}

</div>

</div>

<div class="p-4 sm:w-1/2 lg:w-1/4 w-1/2">

<h2 class="title-font font-medium text-3xl text-white">2.7K</h2>

<p class="leading-relaxed">Evaluations</p>

</div>

<div class="p-4 sm:w-1/2 lg:w-1/4 w-1/2">

<h2 class="title-font font-medium text-3xl text-white">{{ game.editor.name }}</h2>

<p class="leading-relaxed">Editeur</p>

</div>

<div class="p-4 sm:w-1/2 lg:w-1/4 w-1/2">

<h2 class="title-font font-medium text-3xl text-white">{{ game.releaseDate|date('Y') }}</h2>

<p class="leading-relaxed">Sortie</p>

</div>

<div class="p-4 sm:w-1/2 lg:w-1/4 w-1/2">

<h2 class="title-font font-medium text-3xl text-white">2.5K</h2>

<p class="leading-relaxed">Téléchargements</p>

</div>

<div class="p-4">

<h2 class="title-font font-medium text-xl text-white pb-2">Catégories</h2>

{% for genre in game.genres %}

<a class="bg-gray-700 text-blue-500 text-xs font-medium me-2 px-2.5 py-0.5 rounded-sm dark:bg-blue-900 dark:text-blue-300 hover:bg-blue-500 hover:text-white" href="#">{{ genre.name }}</a>

{% endfor %}

</div>

</div>

</div>

</section>

{% endblock %}

Mettons maintenant à jour le lien « En savoir plus » en éditant le fichier templates/game/\_game\_item.html.twig

Sur l'élément de lien « a », ajoutez l'attribut href suivant :

href="{{ path('app\_game\_show', {'id': game.id}) }}

# Genre controller

Nous voulons pouvoir cliquer sur un genre pour voir tous les jeux liés à ce genre.  
  
Nous devons d'abord ajouter un GenreController

symfony console make:controller GenreController

Dans GenreController, supprimez la méthode index et ajoutez ce qui suit :

#[Route('/genre/{id}', name: 'app\_genre\_show')]

public function show(Genre $genre): Response

{

return $this->render('genre/show.html.twig', [

'genre' => $genre

]);

}

Dans templates/genre, renommez index.html.twig en show.html.twig et remplacez tout le contenu par :

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

{% block title %}Games list{% endblock %}

{% block body %}

<section class="text-gray-400 bg-gray-900 body-font container px-5 py-14 mx-auto">

<div class="container px-5 py-24 mx-auto">

<h1 class="title-font font-medium text-4xl mb-8 text-white">{{ genre.name }}</h1>

<div class="flex flex-wrap sm:-m-4 -mx-4 -mb-10 -mt-4">

{% for game in genre.games %}

{% include "game/\_game\_item.html.twig" %}

{% endfor %}

</div>

</div>

</section>

{% endblock %}

Dans templates/game/show.html.twig, ajoutez le href suivant sur le lien du genre

{{ path('app\_genre\_show', {'id': genre.id }) }}

# Fonctionnalité Ajouter à la liste de souhaits

Nous allons maintenant créer un contrôleur Wishlist.  
  
Exécutez la commande suivante :  
symfony console make:controller WishlistController

Ajoutez les deux méthodes suivantes :

#[Route('/game/{id}/wishlist/toggle', name: 'app\_game\_wishlist\_toggle', methods: ['POST'])]

public function wishlistToggle(

Request $request,

Game $game,

EntityManagerInterface $entityManager ,

WishlistRepository $wishlistRepository

): Response {

$user = $this->getUser();

if (!$user) {

throw $this->createAccessDeniedException('You need to be connected.');

}

$wishlist = $wishlistRepository->findOneBy(['game' => $game, 'user' => $user]);

if ($wishlist) {

$entityManager ->remove($wishlist);

$wishlist = null;

} else {

$wishlist = new Wishlist();

$wishlist->setUser($user);

$wishlist->setGame($game);

$entityManager ->persist($wishlist);

}

$entityManager ->flush();

return $this->render('game/\_wishlist\_button.html.twig', [

'game' => $game,

'wishlist' => $wishlist,

]);

}

Nous devons modifier le **GameController** et la méthode show pour vérifier s'il existe déjà une liste de souhaits existante et la retourne à la vue :

#[Route('/game/{id}', name: 'app\_game\_show')]

public function show(Game $game, Request $request, EntityManagerInterface $entityManager, WishlistRepository $wishlistRepository): Response

{

$user = $this->getUser();

$wishlist = $wishlistRepository->findOneBy(['game' => $game, 'user' => $user]);

return $this->render('game/show.html.twig', [

'game' => $game,

'wishlist' => $wishlist,

]);

}

Côté template, nous allons utiliser des turbo-frame. Cela permet de gérer le refresh d’un élément dans la page sans avoir à actualiser toute la page et sans avoir à coder la partie javascript (comme on le ferait avec ajax ou fetch).  
  
Pour plus d’information sur Turbo vous pouvez lire la documentation:

<https://symfony.com/bundles/ux-turbo/current/index.html#usage>

Dans templates, créez un nouveau dossier de **wishlist** et créez à l'intérieur un nouveau fichier «\_wishlist\_button.html.twig»

<turbo-frame id="wishlist-button-{{ game.id }}">

{% if app.user %}

{% if wishlist %}

<form

action="{{ path('app\_game\_wishlist\_toggle', { id: game.id }) }}"

method="POST"

data-turbo-frame="wishlist-button-{{ game.id }}"

>

<div class="relative inline-block group">

<button class="inline-flex items-center rounded-md bg-blue-500/50 px-3 py-2 text-sm font-semibold text-white shadow-xs hover:bg-indigo-500 focus-visible:outline-2 focus-visible:outline-offset-2 focus-visible:outline-indigo-600">

<svg class="mr-1.5 -ml-0.5 size-5" viewBox="0 0 20 20" fill="currentColor" aria-hidden="true" data-slot="icon">

<path fill-rule="evenodd" d="M16.704 4.153a.75.75 0 0 1 .143 1.052l-8 10.5a.75.75 0 0 1-1.127.075l-4.5-4.5a.75.75 0 0 1 1.06-1.06l3.894 3.893 7.48-9.817a.75.75 0 0 1 1.05-.143Z" clip-rule="evenodd"></path>

</svg>

Dans la liste de souhait</button>

<!-- Tooltip -->

<div class="absolute left-1/2 bottom-full mb-2 w-max -translate-x-1/2 rounded bg-gray-700 px-2 py-1 text-xs text-white opacity-0 transition-opacity duration-300 group-hover:opacity-100">

Supprimer de la liste de souhait

</div>

</div>

</form>

{% else %}

<form

action="{{ path('app\_game\_wishlist\_toggle', { id: game.id }) }}"

method="POST"

data-turbo-frame="wishlist-button-{{ game.id }}"

>

<button class="inline-flex items-center rounded-md bg-blue-500/50 px-3 py-2 text-sm font-semibold text-white shadow-xs hover:bg-indigo-500 focus-visible:outline-2 focus-visible:outline-offset-2 focus-visible:outline-indigo-600">

Ajouter à la liste de souhait</button>

</form>

{% endif %}

{% else %}

Se connecter pour ajouter à la liste de souhait

{% endif %}

</turbo-frame>

Dans templates/game/show.html.twig ajouter le code suivant avant le body endblock:

<section class="text-gray-400 bg-gray-900 body-font pb-10">

<div class="container px-5 mx-auto flex flex-wrap" id="wishlist-button">

{% include game/\_wishlist\_button.html.twig' with {

game: game,

wishlist: wishlist

} %}

</div>

</section>

De plus, nous souhaitons que la date de création à soit définie automatiquement. Pour cela, nous devons modifier l'entité Wishlist et ajouter l'attribut php #[ORM\HasLifecycleCallbacks] avant la classe :

**#[ORM\HasLifecycleCallbacks]**

#[ORM\Entity(repositoryClass: WishlistRepository::class)]

class Wishlist

{

Nous devons également ajouter la méthode suivante :

#[ORM\PrePersist]

public function setCreatedAtValue(): static

{

$this->createdAt = new \DateTimeImmutable();

return $this;

}

# Voir la liste de souhaits

Dans le contrôleur Wishlist, ajoutez la méthode suivante :

#[Route('/wishlist/', name: 'app\_wishlist')]

public function list(WishlistRepository $wishlistRepository): Response

{

$wishlistItems = $wishlistRepository->findBy(["user" => $this->getUser()]);

return $this->render('wishlist/list.html.twig', [

'wishlistItems' => $wishlistItems,

]);

}

Dans templates/wishlist ajoutez un nouveau fichier list.html.twig et ajoutez le code suivant :

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

{% block title %}Wishlist{% endblock %}

{% block body %}

<section class="text-gray-400 bg-gray-900 body-font container px-5 py-14 mx-auto">

<h1>My Wishlist</h1>

<div class="container px-5 py-24 mx-auto">

<div class="flex flex-wrap sm:-m-4 -mx-4 -mb-10 -mt-4">

{% for item in wishlistItems %}

{% include 'game/\_game\_item.html.twig' with {'game': item.getGame} %}

{% endfor %}

</div>

</div>

</section>

{% endblock %}

# Ajouter un avis

Pour pouvoir gérer la révision, nous devons utiliser ce qu'on appelle un FormType. Un type de formulaire est une classe qui va définir les champs nécessaires.

Vous pouvez en savoir plus sur les formulaires Symfony ici :  
<https://symfony.com/doc/current/forms.html>

Exécutez la commande suivante

**symfony console make:form ReviewType**

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):

**Review**

Dans src/Form/ReviewType.php, remplacez le code par le suivant :

<?php

namespace App\Form;

use App\Entity\Review;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\Extension\Core\Type\ChoiceType;

use Symfony\Component\Form\Extension\Core\Type\TextareaType;

use Symfony\Component\Form\FormBuilderInterface;

use Symfony\Component\OptionsResolver\OptionsResolver;

use Symfony\Component\Validator\Constraints\NotNull;

class ReviewType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options): void

{

$builder

->add('comment', TextareaType::class, [

'label' => 'Your review',

'required' => false,

'attr' => [

'rows' => 5,

'class' => 'bg-gray-800 w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent',

'placeholder' => 'Write your review here...',

],

])

->add('review', ChoiceType::class, [

'label' => 'Do you recommend this game?',

'choices' => [

'Yes' => true,

'No' => false,

],

'expanded' => true,

'multiple' => false,

'constraints' => [

new NotNull([

'message' => 'Please select an option.',

]),

],

]);

}

public function configureOptions(OptionsResolver $resolver): void

{

$resolver->setDefaults([

'data\_class' => Review::class,

]);

}

}

Nous allons maintenant créer un contrôleur de jeu.  
Exécutez la commande suivante :  
symfony console make:controller ReviewController

Dans ReviewController.php, supprimez tout le code et ajoutez plutôt ce qui suit :

<?php

namespace App\Controller;

use App\Entity\Game;

use App\Entity\Review;

use App\Form\ReviewType;

use App\Repository\ReviewRepository;

use Doctrine\ORM\EntityManagerInterface;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Bundle\SecurityBundle\Security;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Attribute\Route;

final class ReviewController extends AbstractController

{

#[Route('/review/{id}/form', name: 'app\_review\_form')]

public function form(Game $game, Request $request, ReviewRepository $reviewRepository,

Security $security, EntityManagerInterface $em): Response

{

$user = $security->getUser();

$review = $reviewRepository->findOneBy(['game' => $game, 'user' => $user]);

if (!$review) {

$review = new Review();

$review->setGame($game);

$review->setUser($user);

}

$form = $this->createForm(ReviewType::class, $review);

$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {

$em->persist($review);

$em->flush();

}

return $this->render('review/\_form.html.twig', [

'form' => $form->createView(),

'gameId' => $game->getId(),

'review' => $review

]);

}

}

Dans templates/review, créez un nouveau fichier \_form.html.twig et ajoutez le code suivant :

<turbo-frame id="review-form">

<div class="container mb-8 px-5 mx-auto max-w-2xl ">

{% if review.id %}

<div class="p-4 mb-5 text-sm text-blue-100 rounded-lg bg-gray-700 dark:bg-gray-800 dark:text-blue-500" role="alert">

<span class="font-medium">Thanks for the review!</span>

</div>

{% else %}

<form method="post" action="{{ path('app\_review\_form', { id: gameId }) }}" class="text-white bg-gray-800 p-8 rounded-lg" data-turbo-frame="review-form">

<div class="mb-4">

<label for="{{ form.comment.vars.id }}" class="block text-white font-semibold mb-2">

Your review

</label>

{{ form\_widget(form.comment) }}

</div>

<div class="mb-4">

{% if form\_errors(form.review) %}

<div class="p-4 mb-4 text-sm text-red-800 rounded-lg bg-red-50 dark:bg-gray-800 dark:text-red-400" role="alert">

{{ form\_errors(form.review) }}

</div>

{% endif %}

<span class="block text-white font-semibold mb-2">

Do you recommend this game?

</span>

<div class="flex space-x-2">

{% set yes\_field = form.review[0] %}

{% set no\_field = form.review[1] %}

<input

type="radio"

id="{{ yes\_field.vars.id }}"

name="{{ yes\_field.vars.full\_name }}"

value="{{ yes\_field.vars.value }}"

class="hidden peer/yes"

{% if yes\_field.vars.checked %} checked {% endif %} />

<label

for="{{ yes\_field.vars.id }}"

class="flex items-center px-4 py-2 border border-gray-300 rounded cursor-pointer

text-gray-700 hover:bg-blue-100

peer-checked/yes:bg-blue-500 peer-checked/yes:text-white

peer-checked/yes:border-blue-500 transition-colors">

<svg class="w-5 h-5 mr-2" viewBox="0 0 24 24" width="24" height="24" stroke="currentColor" stroke-width="2" fill="none" stroke-linecap="round" stroke-linejoin="round">

<path d="M14 9V5a3 3 0 0 0-3-3l-4 9v11h11.28a2 2 0 0 0 2-1.7l1.38-9a2 2 0 0 0-2-2.3zM7 22H4a2 2 0 0 1-2-2v-7a2 2 0 0 1 2-2h3"></path>

</svg>

Yes

</label>

<input

type="radio"

id="{{ no\_field.vars.id }}"

name="{{ no\_createdfield.vars.full\_name }}"

value="{{ no\_field.vars.value }}"

class="hidden peer/no"

{% if no\_field.vars.checked %} checked {% endif %} />

<label

for="{{ no\_field.vars.id }}"

class="flex items-center px-4 py-2 border border-gray-300 rounded cursor-pointer

text-gray-700 hover:bg-blue-100

peer-checked/no:bg-blue-500 peer-checked/no:text-white

peer-checked/no:border-blue-500 transition-colors">

<svg class="w-5 h-5 mr-2" viewBox="0 0 24 24" width="24" height="24" stroke="currentColor" stroke-width="2" fill="none" stroke-linecap="round" stroke-linejoin="round">

<path d="M10 15v4a3 3 0 0 0 3 3l4-9V2H5.72a2 2 0 0 0-2 1.7l-1.38 9a2 2 0 0 0 2 2.3zm7-13h2.67A2.31 2.31 0 0 1 22 4v7a2.31 2.31 0 0 1-2.33 2H17"></path>

</svg>

No

</label>

</div>

</div>

{{ form\_row(form.\_token) }}

<button type="submit"

class="cursor-pointer px-4 py-2 bg-blue-500 text-white font-semibold rounded-md hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:ring-offset-2">

Post review

</button>

</form>

{% endif %}

</div>

</turbo-frame>

Nous voulons que la date de création soit définie automatiquement. Pour cela nous devons modifier l'entité Review et ajouter l'attribut php #[ORM\HasLifecycleCallbacks] avant la classe :

**#[ORM\HasLifecycleCallbacks]**

#[ORM\Entity(repositoryClass: ReviewRepository::class)]

class Review

{

Nous devons également ajouter la méthode suivante :

#[ORM\PrePersist]

public function setCreatedAtValue(): static

{

$this->createdAt = new \DateTimeImmutable();

return $this;

}

# 

# Commandes utiles

**Démarre le serveur Web local Symfony.**

symfony server:start

**Arrêt du serveur web local Symfony (même effet que CTRL+C).**

symfony server:stop

**Vider le cache de l'application.**

symfony console cache:clear

**Créer un contrôleur :**

symfony console make:controller ControllerName

**Génère une nouvelle entité Doctrine.**

symfony console make:entity EntityName

**Créer un formulaire :**

symfony console make:form FormName

**Génère une nouvelle migration basée sur le schéma de base de données actuel.**

symfony console make:migration

**Exécute les migrations pour mettre à jour votre schéma de base de données.**

symfony console doctrine:migrations:migrate

**Lister tous les itinéraires**

symfony console debug:router

**Tailwind watch**

php bin/console tailwind:build --watch