DOCUMENTATION – ECF LIBRARY AFPA REST – BDD DE FILMS – CLEMENT MIERMONT

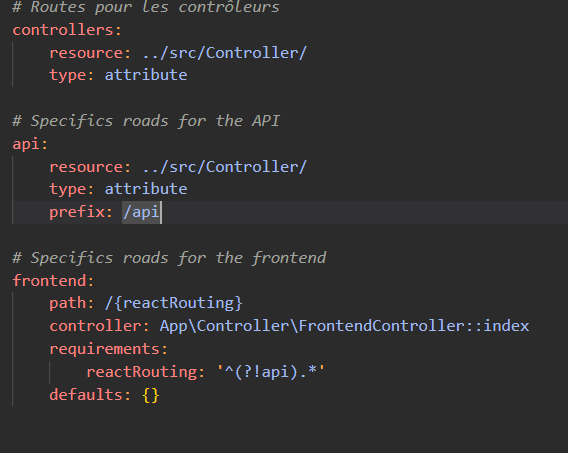
**1. Structure du Projet Backend**

La structure du projet backend contient les dossiers et fichiers clés suivants :

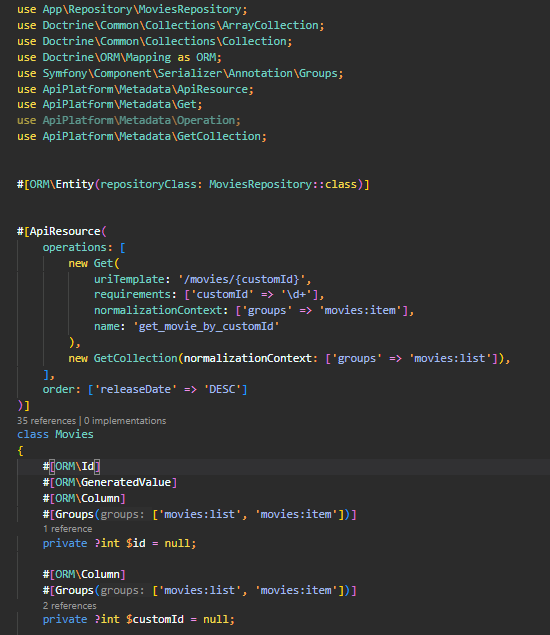
* **bin/** : Contient le script pour exécuter la console Symfony.
* **config/** : Contient tous les fichiers de configuration, y compris les paramètres de l'API.
* **migrations/** : Gère les migrations de base de données (Doctrine).
* **public/** : Contient le fichier d'entrée principal (index.php) et les ressources publiques.
* **src/** : Contient les classes de logique métier (contrôleurs, entités, services, etc.).
* **templates/** : Gère les fichiers Twig si nécessaire (l’affichage est géré par vue.JS)
* **var/** : Contient les fichiers temporaires et le cache.
* **vendor/** : Dossier des dépendances installées via Composer.
* **.env** : Fichier d'environnement pour les configurations (DB, JWT, etc.).

Ce projet utilise **API Platform** pour gérer l'API. Cela permet de créer des endpoints RESTful automatiquement à partir des entités définies.

* Les routes sont automatiques et générées par le fichier routes.yaml



Chaque entité représente une table dans une base de données



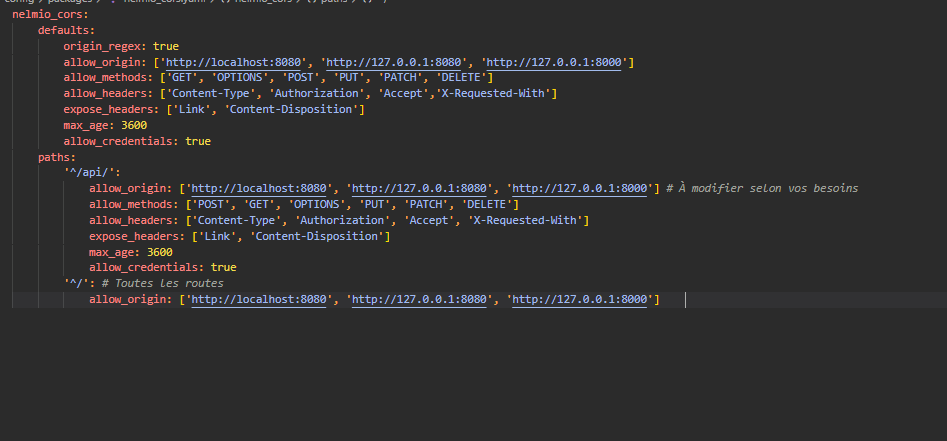
L'annotation #[ApiResource] génère automatiquement les endpoints CRUD pour cette entité.

Le projet utilise **LexikJWTAuthenticationBundle** pour gérer l'authentification avec des tokens JWT.

**Configuration :**

* Les clés publiques et privées pour les JWT sont situées dans private.pem et public.pem.
* Les utilisateurs s'authentifient via un endpoint /login\_check, qui retourne un token JWT.
* Les routes sont protégées en utilisant des firewalls dans security.yaml.

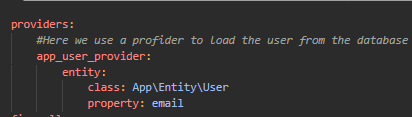
Le projet utilise **NelmioCorsBundle** pour gérer les requêtes cross-origin.



Le fichier security.yaml et le contrôleur AuthController collaborent pour gérer l'authentification via **JWT (JSON Web Tokens)**.

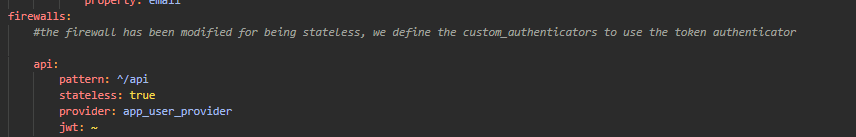
**1. Configuration dans security.yaml**

A. Fournisseur d'Utilisateurs (User Provider)



-> Définit que les utilisateurs sont chargés depuis l'entité User via leur email.

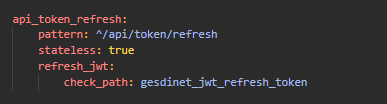
B. Firewalls



 **pattern: ^/api** : Protège toutes les routes qui commencent par /api.

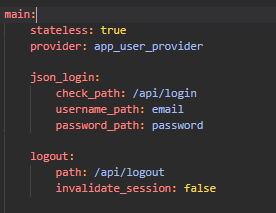
 **stateless: true** : Indique que le firewall n'utilise pas de session ; tout repose sur les tokens.

 **jwt: ~** : Active l'authentification par JWT via le bundle **LexikJWTAuthenticationBundle**.



api\_token\_refresh:

Permet de rafraîchir le token JWT à l'aide de **GesdinetJWTRefreshTokenBundle**.



**Section main du firewall**

La section main dans security.yaml configure le comportement de la sécurité pour gérer l'authentification de l'application.

**1. stateless: true**

* **Signification :**
* Cela indique que ce firewall est **stateless** (sans état).
* Aucun état (session, cookies) n'est conservé entre les requêtes.
* L'authentification repose entièrement sur les **tokens** (ex : JWT) envoyés dans les requêtes HTTP.
* **Avantage :**
* Idéal pour les API REST où chaque requête doit être authentifiée indépendamment.
* Cela rend l'API plus légère et scalable, car aucune gestion de session côté serveur n'est requise.

**1-provider: app\_user\_provider**

**Signification :**

* Définit le **fournisseur d'utilisateurs** à utiliser pour charger les informations utilisateur.
* Ici, le fournisseur app\_user\_provider charge les utilisateurs depuis l'entité User en utilisant l'email comme critère de recherche (défini plus haut dans providers).

**3-json\_login**

**Signification :**

* Permet l'authentification via une requête **JSON POST** à une route définie (ici /api/login).

**Options :**

* **check\_path: /api/login**
  + - Indique le chemin où l'utilisateur doit envoyer ses **identifiants** pour se connecter.
* **username\_path: email**
  + - Définit le champ JSON utilisé pour le **nom d'utilisateur** (ici email).
* **password\_path: password**
  + - Définit le champ JSON utilisé pour le **mot de passe** (ici password).

**Logout**

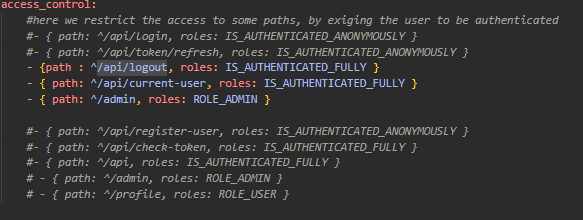
**Signification :**

Définit la route pour la **déconnexion**.

**Options :**

* **path: /api/logout**
  + - Indique le chemin où l'utilisateur peut se déconnecter.
* **invalidate\_session: false**
  + - Empêche l'invalidation de session (inutile ici car le firewall est stateless).

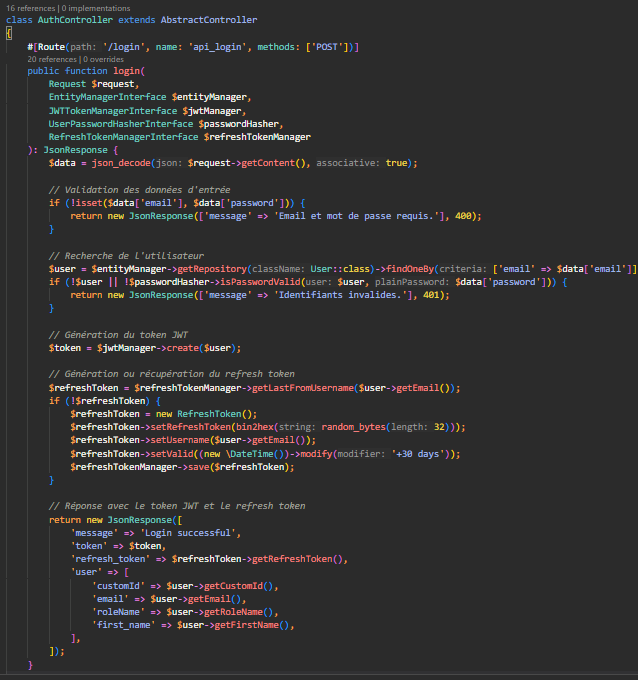
(n’ayant pour le moment pas réussi à faire fonctionner le logout avec un contrôlleur, je suis passé par un système de localstorage pour simuler une déconnexion via vue.JS, mais la connexion passe totalement par le contrôlleur)



Contrôle d’accès (beaucoup de cafouillage et de test, d’où le nombre faramineux de lignes passées en commentaire, le debug a été laborieux, remercions les anonymes des discussions github et de stack overflow)

Résumé rapide de la connexion :

* L'utilisateur se connecte via /api/login avec ses identifiants (JSON POST).
* => Retourne un token JWT.
* Ce **token JWT** est envoyé avec chaque requête protégée via l'en-tête Authorization: Bearer <TOKEN>



**Route /login** (Authentification de l'utilisateur)

Comme vous pouvez le voir, pas besoin de mettre API/devant la route, dans notre cas, la route est interprétée comme <http://localhost:8000/login>

Comment ça fonctionne ?

1 – Les données du corps de la requête JSON sont extraites et décodées

2 – On vérifie si l’utilisateur fournit un mail et un mot de passe (sinon retourne une erreur 400 Bad Request)

3 – On cherche l’utilisateur dans la base de données, s’il n’existe pas on retourne une erreur 401 unauthorized

4 - On génère le token avec le service JWTTokenManagerInterface

5 – On vérifie s’il existe un refresh token pour cet utilisateur, si non on crée un refresh token valide pour 30 jours et on sauvegarde le refresh dans la base de données

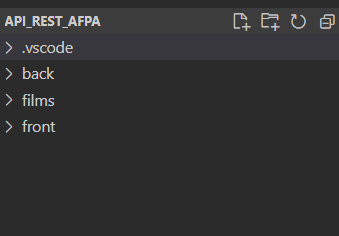
6 – On retourne une réponse JSON contenant :

A/ Le token JWT

B/ Le refresh token

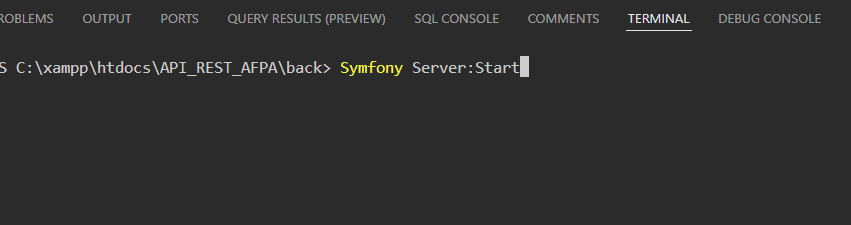
C/ les informations de l’utilisateur : customID, email, roleName, firstName

Comment Symfony permet d’interpréter du Vue.JS ?



Comme vous l’avez remarqué, il y a deux dossiers :

Back et Front



Symfony se lance ainsi depuis le terminal de Vscode.

Dans ce projet Symfony, l’interpétation de Vue.Js repose sur une architecture où le backend Symfony sert l’API REST, tandis que le frontend Vue.js consomme l’API.

Voyons ça comme un restaurant.

Symfony est ici notre cuisine. Les chefs sont les contrôleurs, ils préparent les plats, les données.

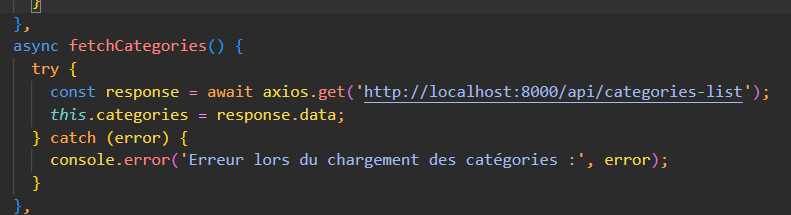
Les ingrédients sont la base de données. Les plats sont préparés par l’API sous forme de réponses pour être servis.

Le Frontend Vue. Js lui est le serveur, il va servir au client la commande, et prendre la commande au client, ainsi le serveur fait son job, et le cuisinier fait son job.

Back -> contient tout le backend de Symfony qui expose l’API

Front -> Contient tout le front end Vue.Js

Les structures sont séparées, mais comment communiquent-elles ?

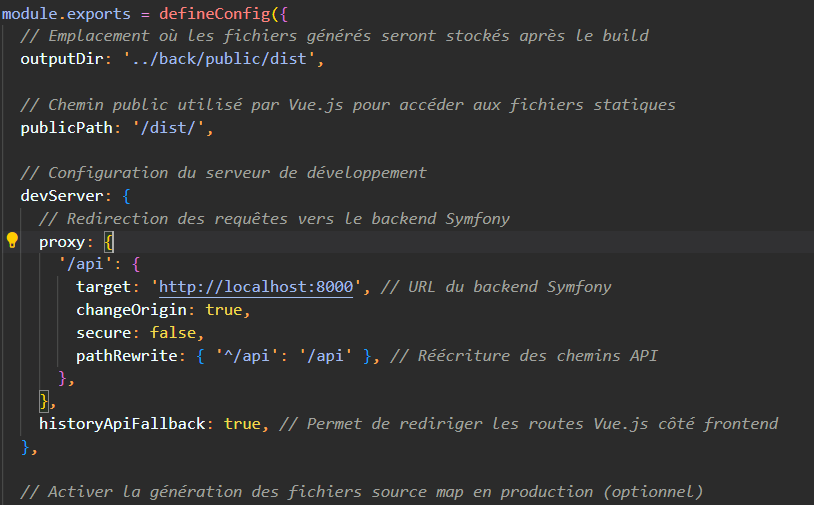
Dans le dossier front, on fait notre Vue.Js 

Dans la page vue, ici celle de MainCatalogue.view, on accède à l’API via Axios.get(‘l’URL de votre API)

On fait ce qu’on y à faire, c’està-dire le front-end.

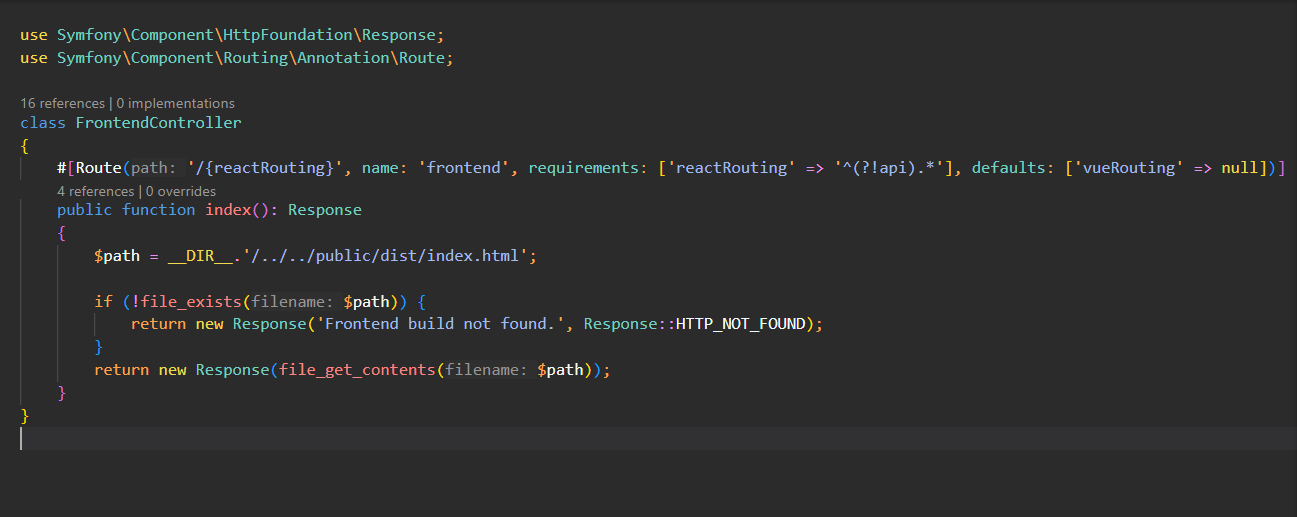


Dans VScode on ouvre le dossier dans le terminal de commande, on exécute npm run build.



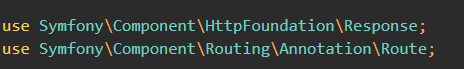
La configuration du projet nous permet de déplacer la compilation de Vue.Js dans /\*racine de votre projet \*/back/public/dist

(On a également configuré un serveur de développement au début du projet mais ici il n’est pas utile vu qu’on fait les tests et le debug dans symfony. )



Le FrontendController.php sert à rediriger toutes les routes du frontend gérées par Vue.JS vers le fichier index.html généré dans public/dist. Cela permet d’utiliser frontend single-page application (SPA) en backend.

**Namespace et use statements** :



Ces use permettent d'importer des classes nécessaires :

* Response : pour retourner des réponses HTTP.
* Route : pour définir des routes via des annotations.

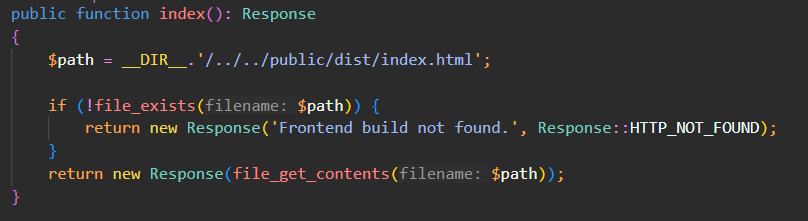


**Annotation de la route** :

Cette annotation définit une route **catch-all** pour capturer toutes les requêtes qui ne commencent pas par /api.

 **Détails des options :**

* path: '/{reactRouting}' : capture toutes les routes après /.
* Requirements: ['reactRouting' => '^(?!api).\*'] : exclut toutes les routes qui commencent par /api.
* defaults: ['vueRouting' => null] : fournit une valeur par défaut pour vueRouting (utile pour éviter des erreurs si aucune valeur n'est capturée).



Cette fonction construit le chemin absolu vers le fichier index. Html généré lors de la compilation du frontend vers back/public/dist. Et nous retourne ensuite le tout sous forme de avec file\_get\_contest.

Ce fichier permet donc à Symfony de rediriger toutes les routes non-API vers l’index.HTML.