AP 1
App PHP Symfony

Cette synthèse présente le compte rendu des différentes missions réalisées de l’ap 1.
L'application d'origine disposait déjà d'un front office permettant de consulter les formations et les playlists. Les missions ont consisté à enrichir et compléter cette base.
L’objectif de ce projet est donc de développer la partie backend d’une application web PHP à l’aide du framework Symfony, afin de gérer un catalogue de formations portant sur le developpement.
Ce projet s’inscrit dans le cadre du BTS SIO SLAM, 2ème année.

Au cours de ce projet un certain nombre de compétences ont été abordées et travaillés notamment,

- Le développement back et front end,
- La mise en place d’une base de données relationnelle MySQL,
- La réalisation de tests unitaires, d’intégration, fonctionnels,
- La rédaction de documentation technique et utilisateur,
- Le déploiement de l’application web sur un serveur,
- Le déploiement continue de l’application via Github.

Les technologies qui on été utilisé principalement pour ce projet :

- PHP, Twig / Symfony
- Bootstrap 5
- MySQL, MySQL Workbench
- Phpstorm
- PHPUnit
- Git & Gitub
- SSH & WinSCP
- Let’s Encrypt & Certbot
- OBS Studio, Libre Office, notepadd++, notion

Sommaire :

Introduction p.1

Mission 1
Tâche I : Nettoyage du code p.3
Tâche II : Ajout fonctionnalité playlists p.5

Mission 2
Tâche I : Gestion des formations p.7
Tâche II : Gestion des playlists p.10
Tâche III : Gestion des catégories p.13
Tâche IV : Authentification p.16

Mission 3
Tâche I : Tests p.18
Tâche II : Documentation technique p.22
Tâche III : Documentation utilisateur p.22

Mission 4
Tâche I : Déploiement VPS p.23
Tâche II : Sauvegarde base de données p.24
Tâche III : Déploiement continue p.25

Bilan p.26

Mission 1;

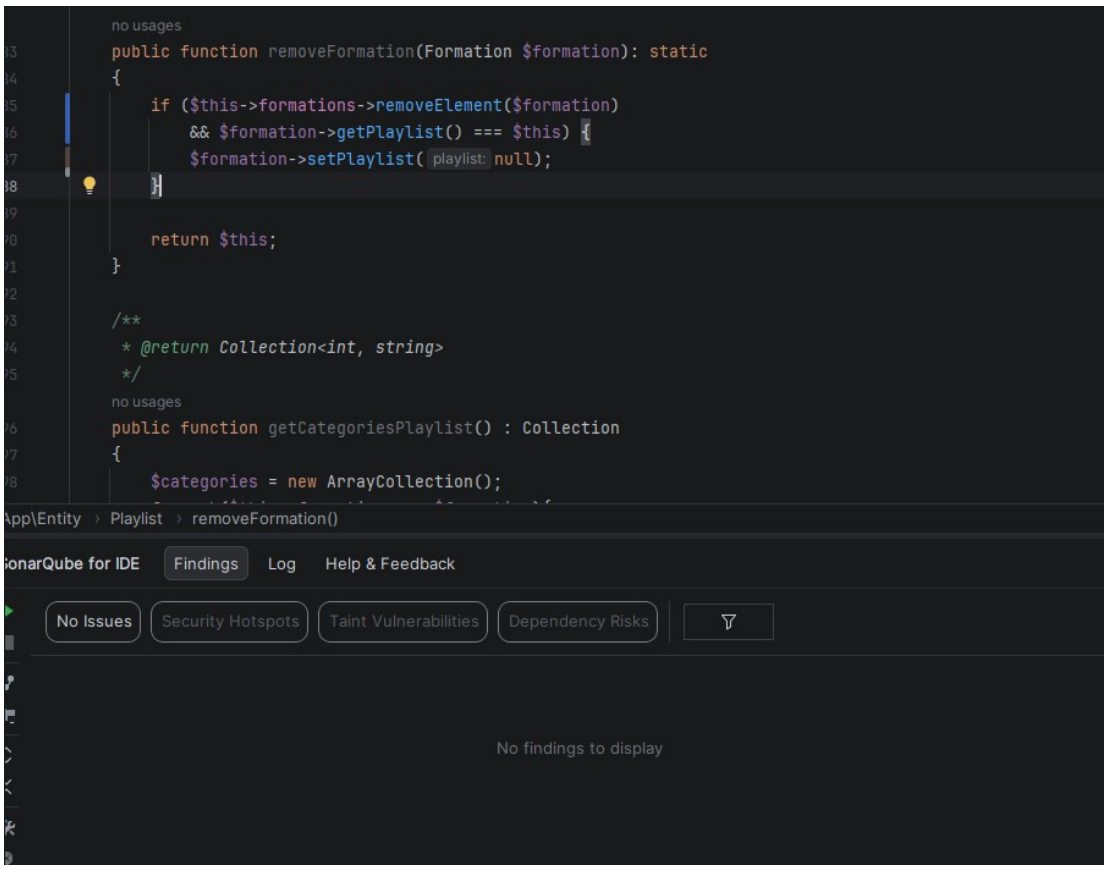
Tâche I

Nettoyage du code de la base du projet,
Une fois Sonarlint configuré sur PHPSTORM voici les modifications appliquées au code :

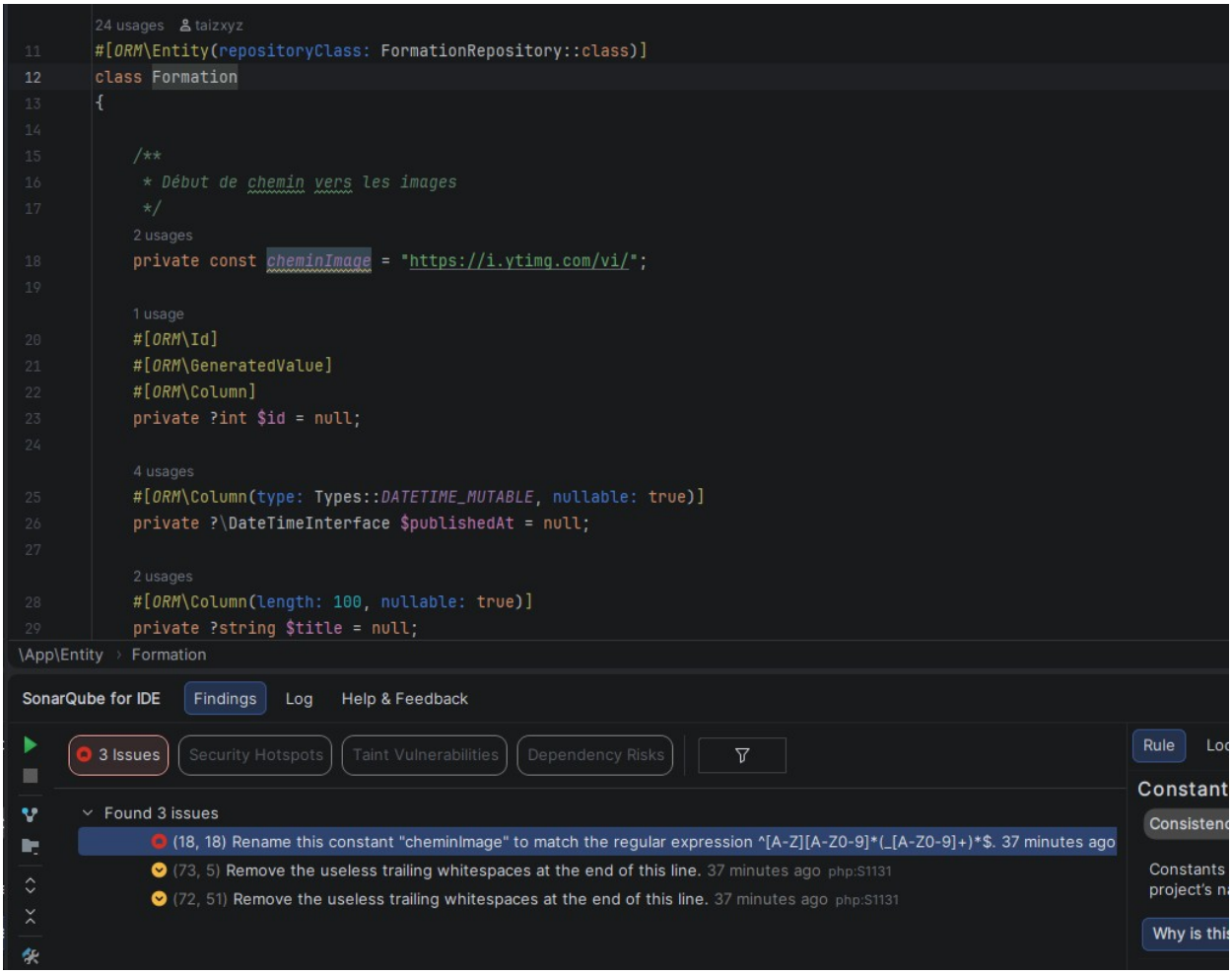
1) Un if imbriqué inutilement.



- voici ce bloc de code correspondant aux bonnes pratiques demandées et sonarlint n'affichant plus d'erreurs dans la console :



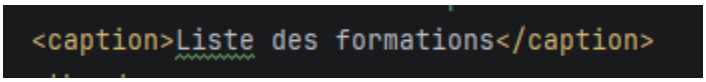
2) Le projet ne contient qu'une constante (src/Entity/Formation.php), qui n'était pas en majuscule. Sonarlint la repère après son analyse :



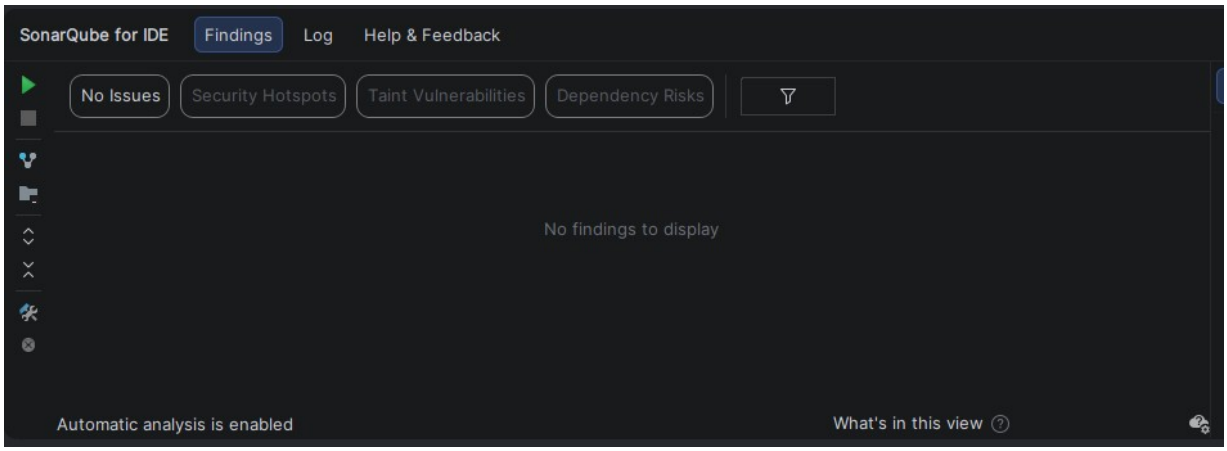
Celle-ci a ensuite été renommée CHEMIN_IMAGE pour correspondre aux conventions attendues. Cela permet au code d'être plus lisible et compréhensible.

3) Ajout de l'attribut "description" à toutes les tables (<table>), l'attribut <caption> a donc été ajouté dans les fichiers suivants :
- formations.html.twig
- playlists.html.twig
- categories.html.twig

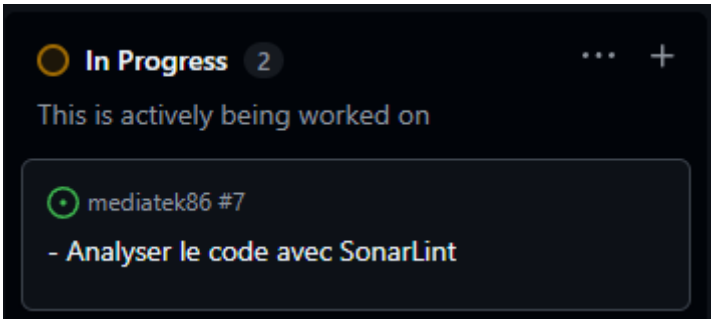
exemple dans pages/admin/formations.html.twig :



4) Pour finir avec Sonarlint voici une capture d'écran de sa console après la résolution de tous les problèmes énoncés. Je précise tout de même qu'il restait des espaces/sauts de lignes mais cela n'est pas intéressant à documenter.



5) Ensuite pour la partie Kanban (outil de journaling orienté sur le developpement d'un projet), voici la capture d'écran attendue dans "In Progress", cette tâche m'a pris environ 40 minutes.



Tâche II ;

Ajout d'une colonne "nb formations" dans la page des playlists avec l'option tri croissant et décroissant, affichage du nombre de formations dans les pages de détail d'une playlist.

Voici les méthodes ajoutées pour répondre à cette problématique :
/PlaylistRepository.php :

```
2 usages  taizxyz *
public function findAllOrderByFormationsCount($ordre): array{
    $playlists = $this->findAll();
    $nbrFormations = [];
    foreach($playlists as $p){
        $nbrFormations[] = count($p->getFormations());
    }
    if($ordre == 'ASC'){
        array_multisort( &array: $nbrFormations,  &...rest: SORT_ASC, $playlists);
    }else{
        array_multisort( &array: $nbrFormations,  &...rest: SORT_DESC, $playlists);
    }
    return $playlists;
}
```

findAllOrderByformationsCount($ordre) dans le repository.
La méthode récupère toutes les playlists avec un “findAll()” et ensuite pour chaque playlist elle compte le nombre de formations avec “count()”.
“array_multisort” est enfin utilisé pour trier selon l’ordre demandé, ascendant ou descendant.

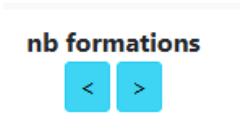
/playlistsController.php :

```
case "name":
    $playlists = $this->playlistRepository->findAllOrderByName($ordre);
    break;
case "nbrformations":
    $playlists = $this->playlistRepository->findAllOrderByFormationsCount($ordre);
    break;
```

Dans la méthode sort() du controller j’ai ajouté “case ‘nbrformations’”, son rôle étant d’appeler la nouvelle méthode présente dans le repository vu juste avant.

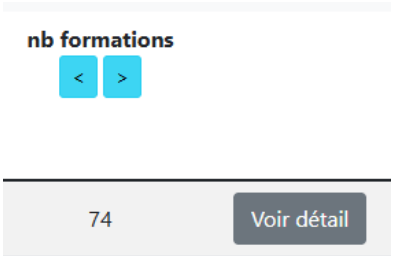
On bascule désormais dans la partie vue (front) de la problématique, c’est là que les changements seront visibles.

/playlists.html.twig :



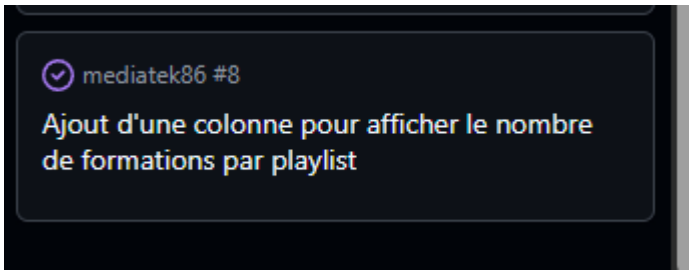
Ajout d’une colonne nb formations dans le tableau, cela comprend 2 boutons “< & >” pour trier dans les deux sens, en respectant la structure des colonnes déjà existantes.

/playlist.html.twig :



Ajout de l’affichage du nombre de formations avec “ {{ playlistformations|length }}”

Au niveau du Kanban je réalise le bilan après avoir réalisé la fonctionnalité il est donc en done (couleur violette),

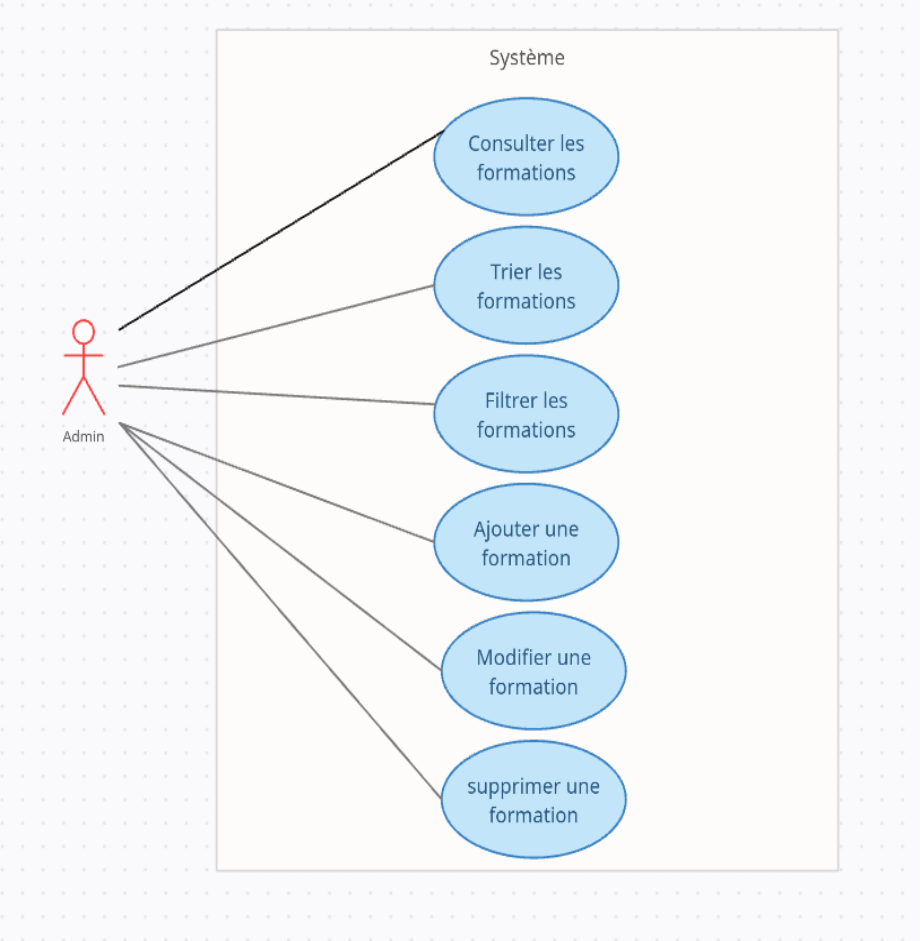


Cette tâche a duré environ 1 heure.

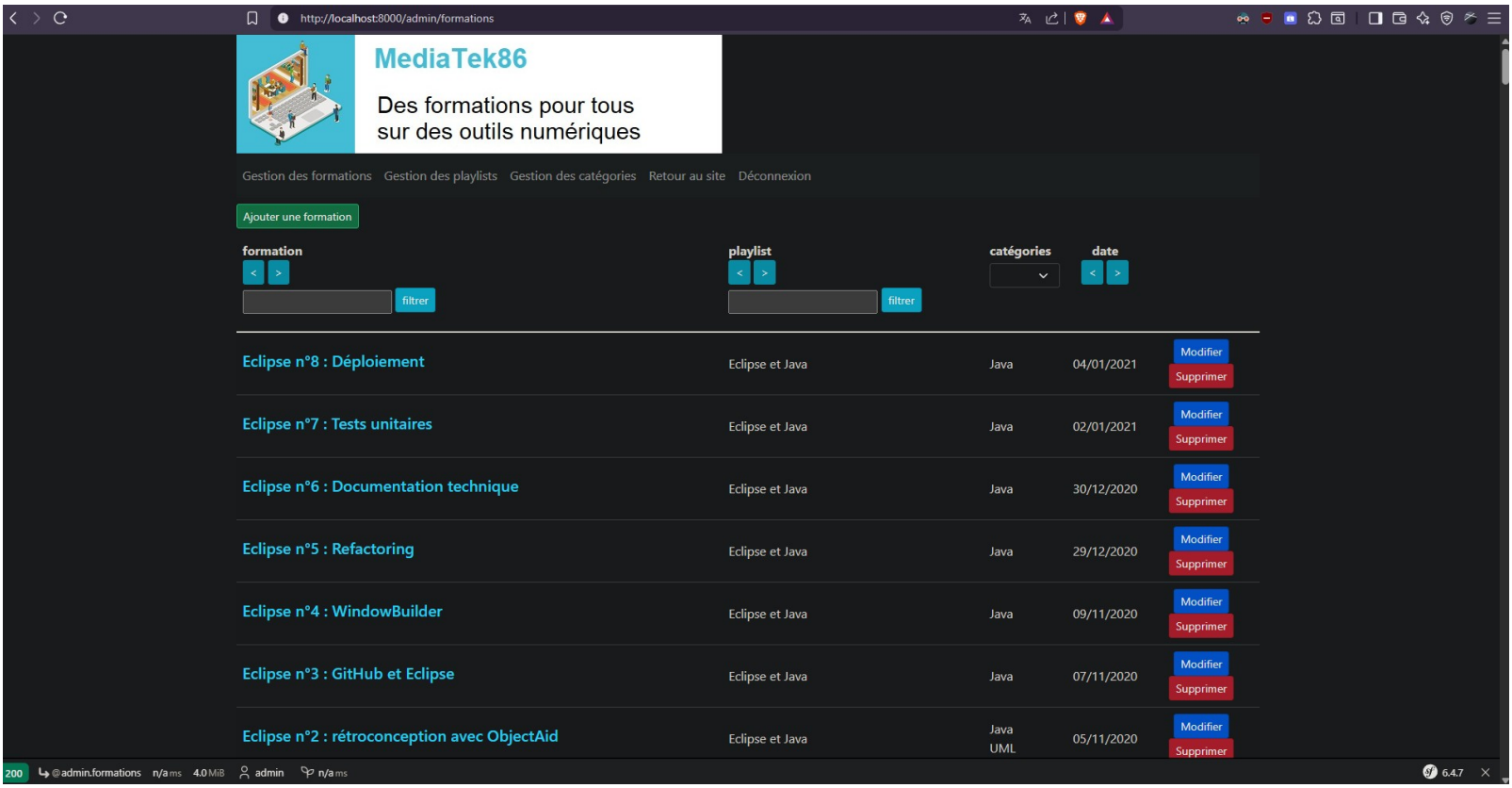
Mission 2 :

Tâche I :

Mise en place du back office pour la gestion des formations (liste, ajout, modif, suppression).
Voici le diagramme de cas d'utilisation pour illustrer la tâche :



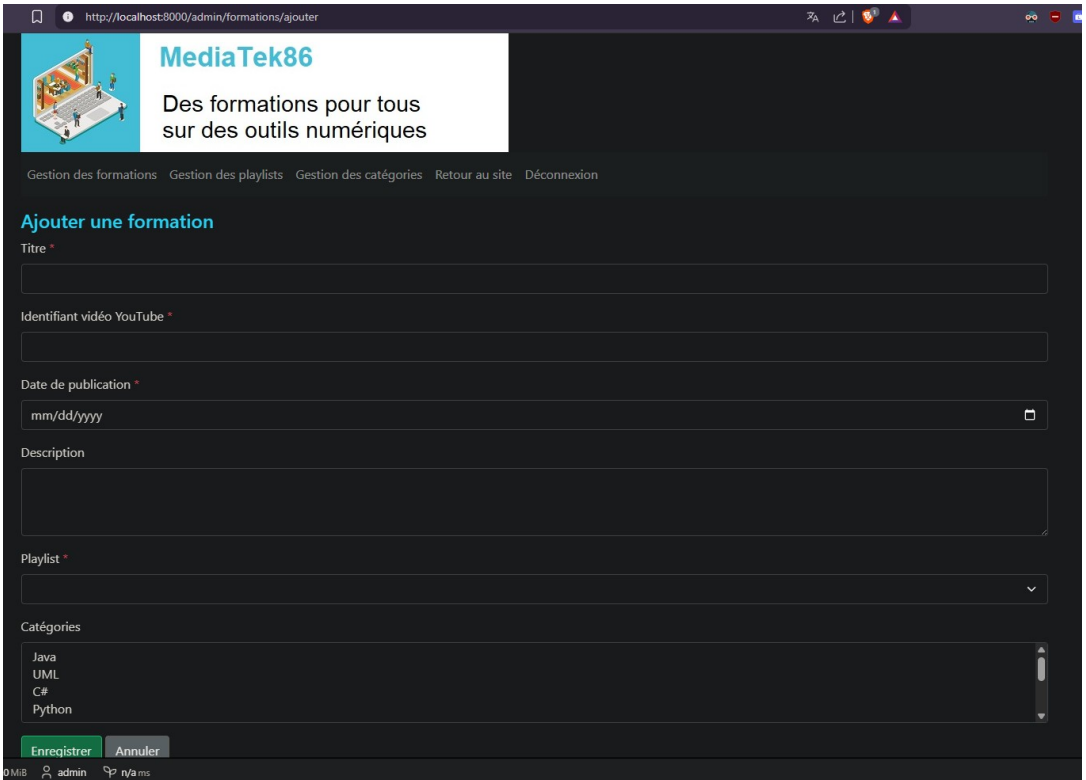
Page admin/formations :



(compte rendu de chaque tâches effectuées à la fin de la mission 2 tâche 4 d'ou la présence de "deconnexion" sur la capture d'écran.)

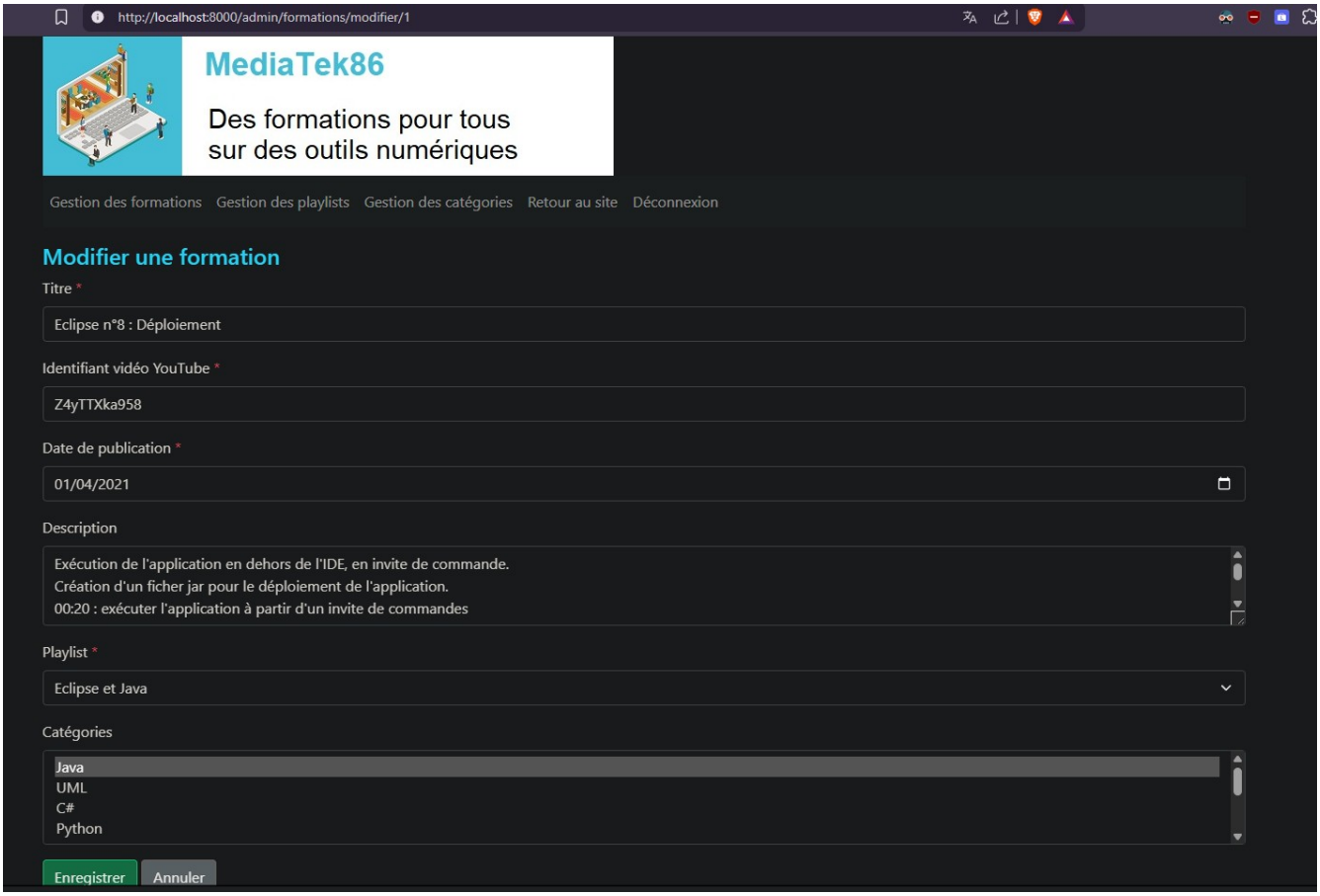
Cette page affiche sous forme de tableau toutes les formations avec les mêmes tris et filtres que le front office.
Deux boutons sont ajoutés par ligne "Modifier & Supprimer", la suppression demande une confirmation en guise de sécurité.

Page admin/formations/ajouter :



C’est donc le formulaire d’ajout, il permet de créer une nouvelle formation. Le titre, date et playlist sont obligatoires pour valider le formulaire.
La date est sélectionnable uniquement jusqu’à aujourd’hui et le reste des formations demandées est optionnelles.
Un token CSRF est présent pour sécuriser le formulaire.

Page admin/formations/modifier/id :



Cette fois ci il s’agit du formulaire de modification, il est identique à celui d’ajout mais prérempli avec les informations liées à l’id de la formation sélectionnée.

Arborescence des fichiers ajoutés :

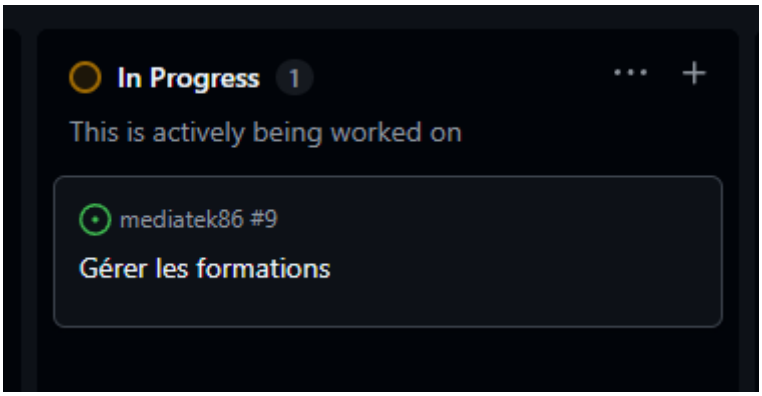
```
src/
└── Controller/
    └── AdminFormationsController.php

templates/
└── pages/
    └── admin/
        ├── formations.html.twig
        └── formation_form.html.twig
```

AdminFormationsController.php, contient 6 méthodes (index, sort, findAllcontain, supprimer, ajouter, modifier). Il suit la même architecture que le controller du front office.

formations.html.twig, reprend la structure du tableau déjà présent dans le front office en ajoutant les boutons mentionnés plus tôt.

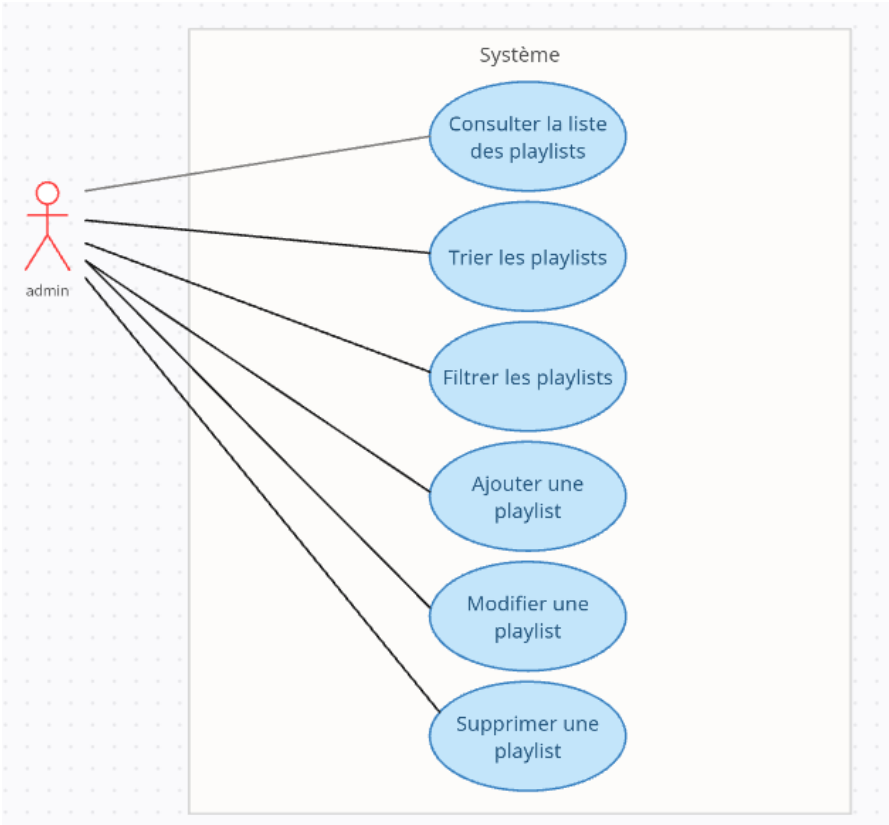
formation_form.html.twig, c'est un formulaire partagé entre la modification et l'ajout. Il est chargé via les deux routes correspondantes.



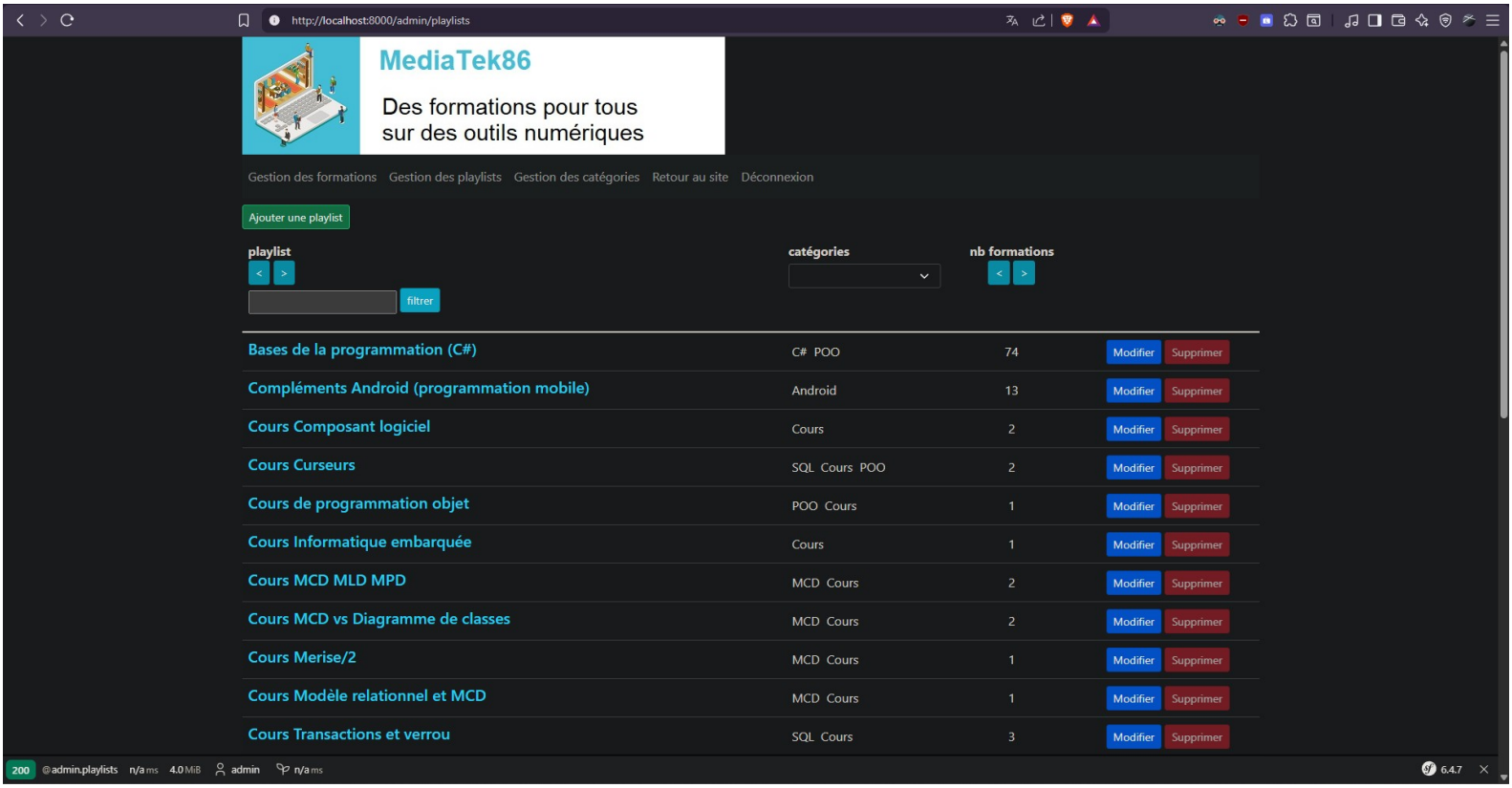
Cette tâche a duré 3h.

Tâche II :

Mise en place du back office pour la gestion des playlists (liste, ajout, modif, suppression).
Voici le diagramme de cas d'utilisation pour illustrer la tâche :

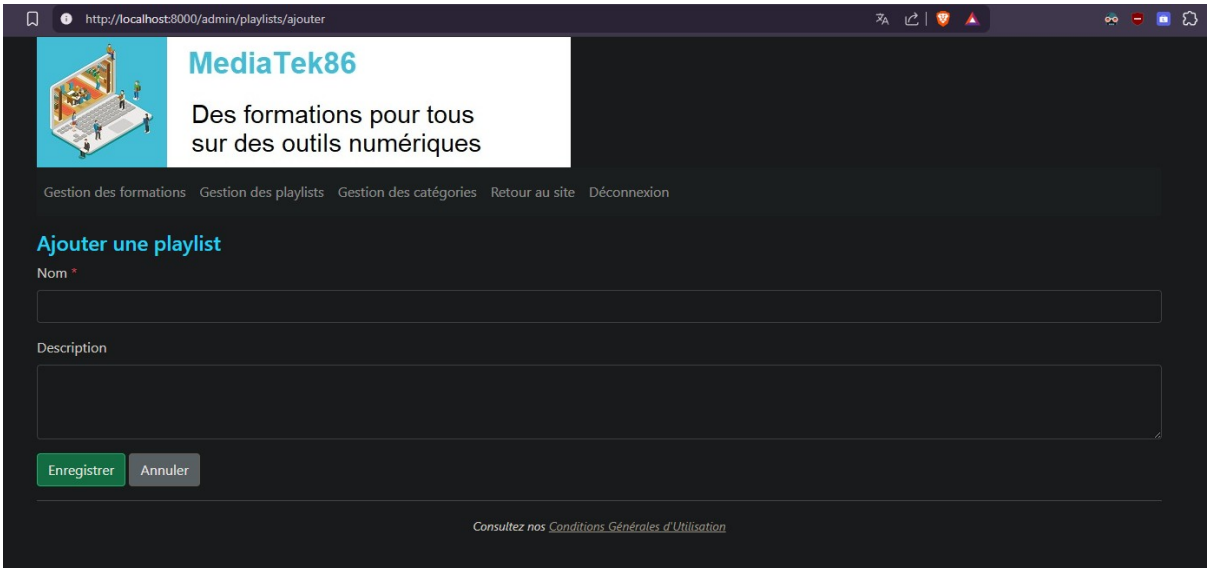


Page admin/playlists :



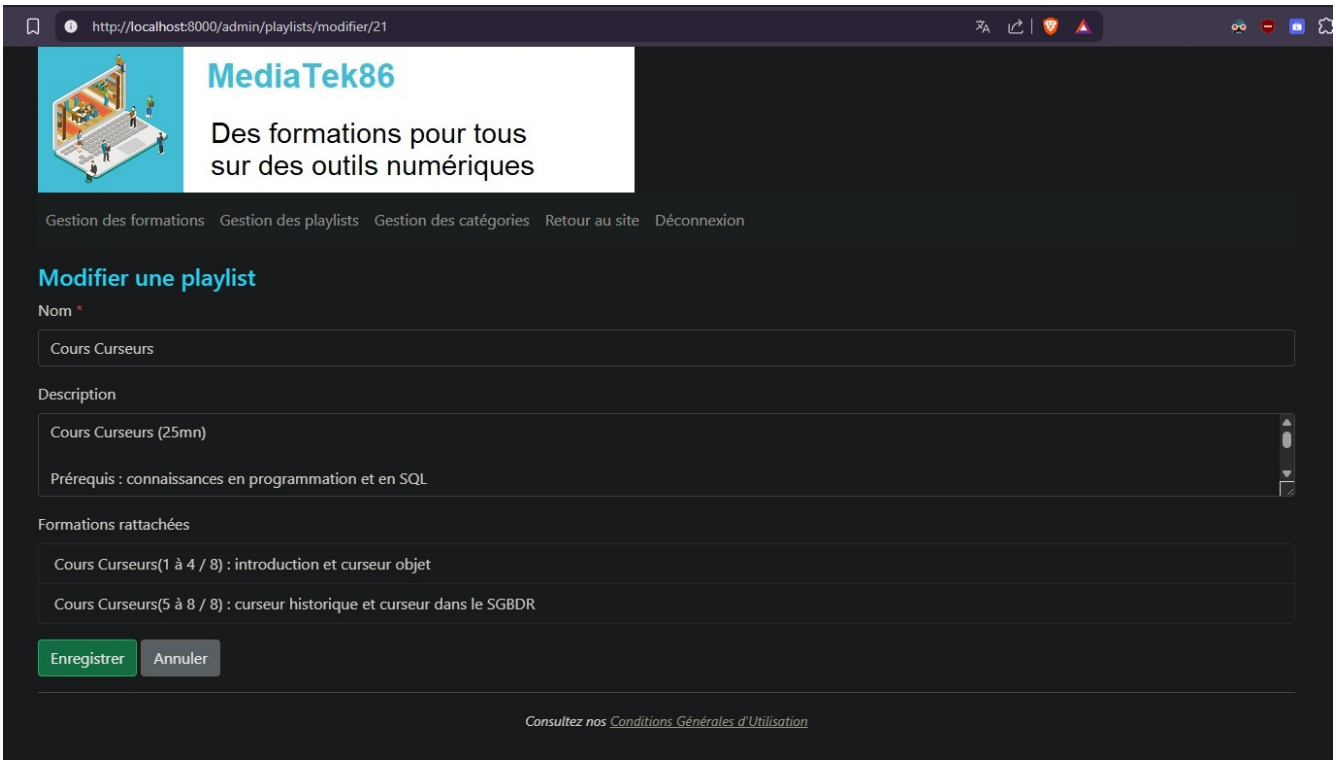
Cette page affiche toutes les playlists avec les mêmes tris et filtres que le front office.
Deux boutons sont ajoutés par ligne “modifier & supprimer”.
Comme sur la page “admin/formations” le bouton supprimer contient une sécurité, celui-ci est désactivé si la playlist contient encore des formations. Cela permet d’éviter la suppression accidentelle d’une playlist non vide.

Page admin/playlists/ajouter :



Cette page est un formulaire permettant de créer une nouvelle playlist. Le champ “nom” est obligatoire, la description est optionnelle. Un token CSRF est présent pour sécuriser le formulaire.

Page admin/playlists/modifier/id :



Cette page est également un formulaire comme celui d’ajout mais cette fois ci prérempli avec les données de la playlist sélectionnée.
La liste des formations rattachées à la playlist est affichée en lecture seule car c’est dans le formulaire d’une formation que l’on choisit sa playlist.

Arborescence des fichiers ajoutés :

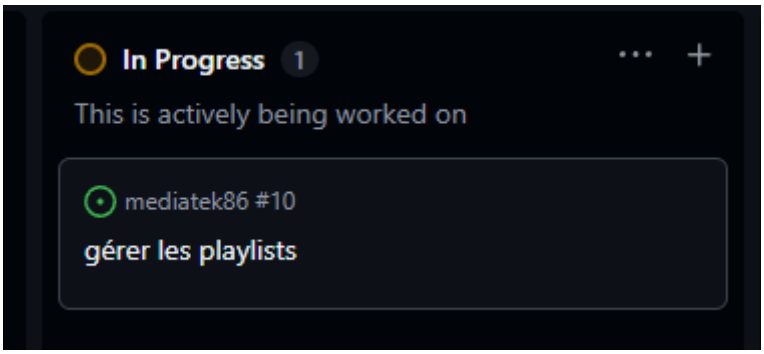
```
src/
└── Controller/
    └── AdminPlaylistsController.php

templates/
└── pages/
    └── admin/
        ├── playlists.html.twig
        └── playlist_form.html.twig
```

AdminPlaylistsController.php, contient 6 méthodes, (index, sort, findAllContain, supprimer, ajouter, modifier et avant de supprimer une playlist la méthode supprimer() vérifie que la playlist ne contient aucune information avec “isEmpty()”).
On remarque d’ailleurs que c’est globalement la même structure que “AdminFormationsController.php”.

playlists.html.twig, reprend la structure du tableau présent dans le front office en ajoutant les boutons d’action.

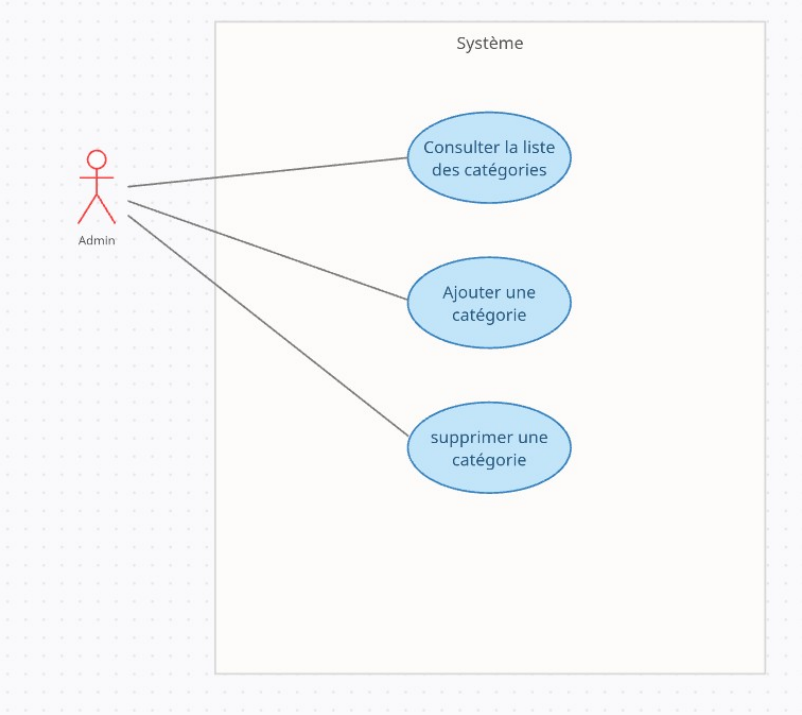
playlist_form.html.twig, est le formulaire partagé entre l’ajout et la modification.



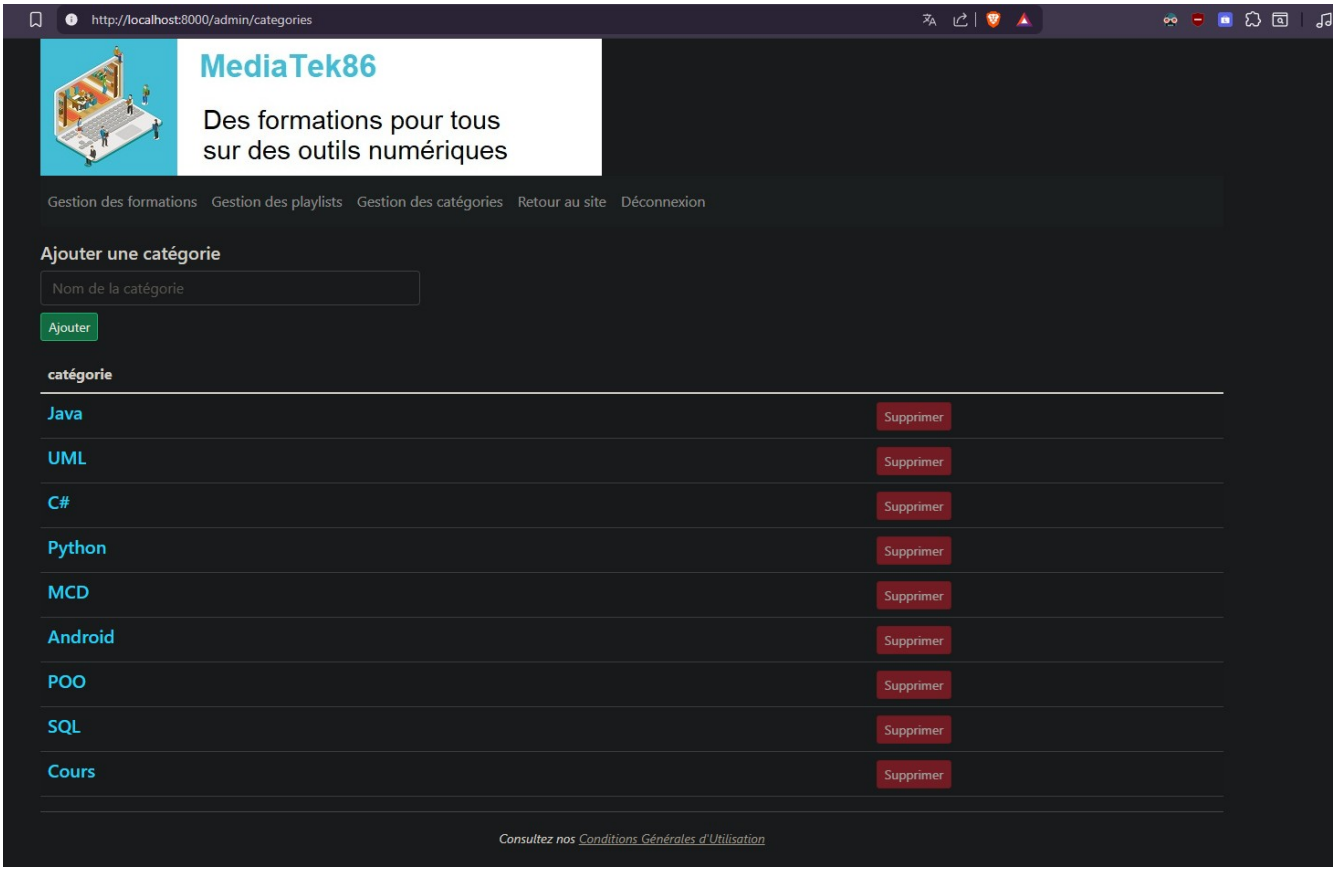
Cette tâche a duré environ 2h30.

Tâche III :

Mise en place du back office pour la gestion des catégories, liste ajout et suppression.
Voici le diagramme de cas d’utilisation pour illustrer la tâche :



Page admin/categories :



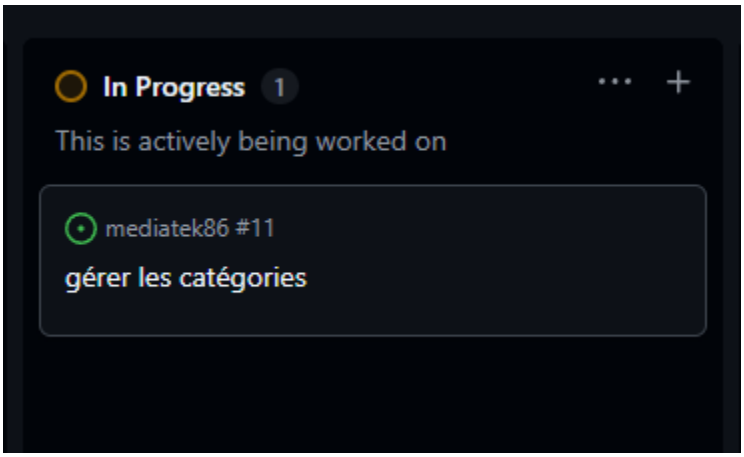
Cette page affiche la liste de toutes les catégories avec un bouton “supprimer” par ligne, celui-ci est désactivé si la catégorie est encore rattachée à au moins une formation.
Un formulaire en bas de page permet d’ajouter directement une nouvelle catégorie sans changer de page. De plus si le nom existe déjà en base de données la catégorie n’est pas ajoutée.

Arborescence des fichiers ajoutés :

```
src/
└── Controller/
 └── AdminCategoriesController.php
templates/
└── pages/
 └── admin/
 └── categories.html.twig
```

AdminCategoriesController.php, contient 2 méthodes (index et supprimer). La méthode index() gère à la fois l’affichage de la liste et le traitement du formulaire d’ajout.
Avant d’ajouter la moindre catégorie elle vérifie avec “findOneBy()” que le nom n’existe pas déjà.
Avant de supprimer la méthode vérifie avec “isEmpty()” que la catégorie n’est rattachée à aucune formation.

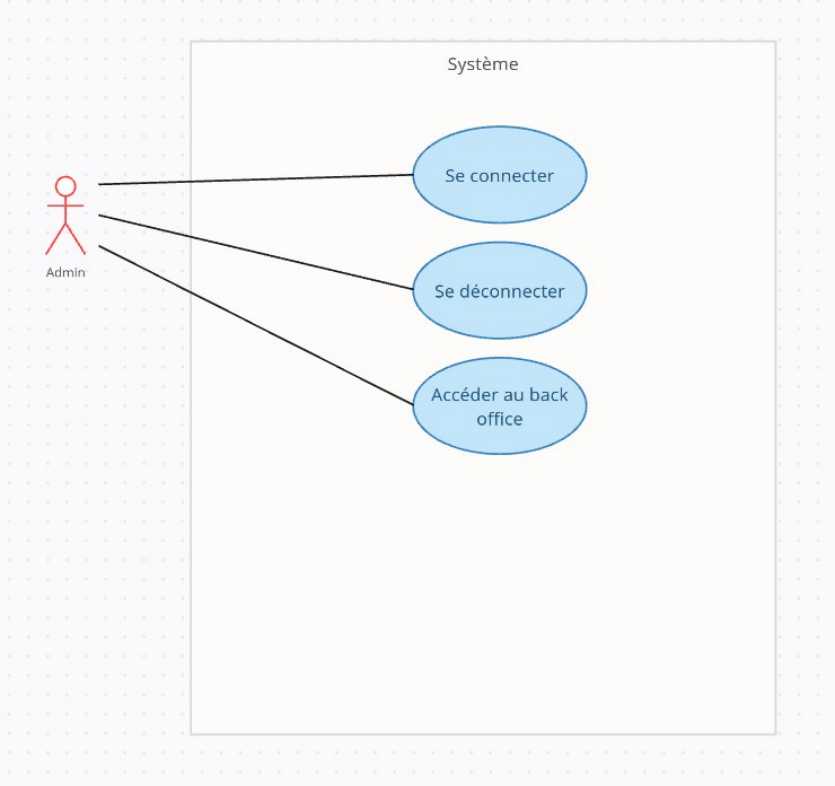
categories.html.twig, affiche la liste des catégories et contient le formulaire d’ajout sur la même page.



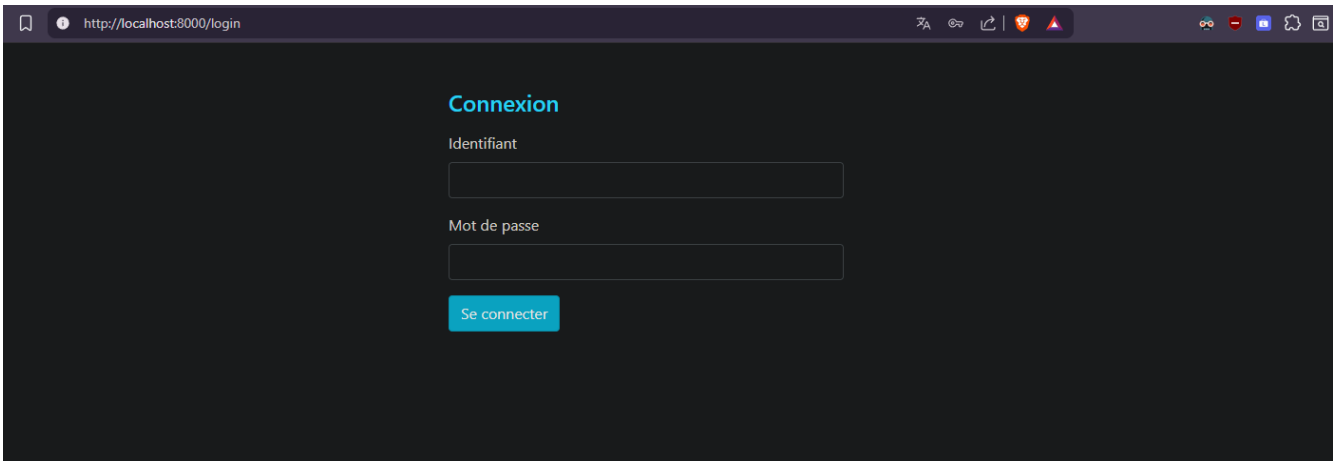
Cette Tâche a duré 2h.

Tâche IV :

Mise en place de l’authentification pour sécuriser l’accès au back office.
Voici le diagramme de cas d’utilisation pour illustrer la tâche :

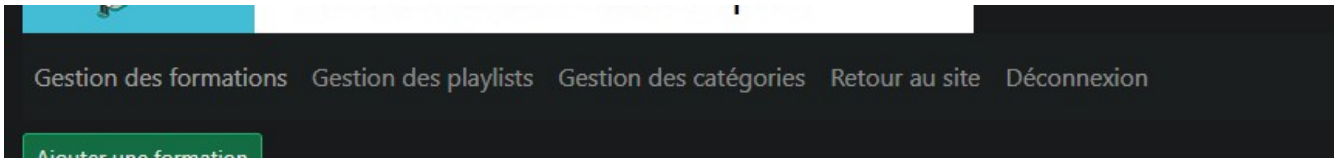


Page /login :



C’est la page de connexion avec les champs “identifiant & mot de passe”, un token CSRF est présent dans le formulaire. Si les identifiants sont incorrects Symfony affiche alors un message d’erreur.
Une fois connecté l’utilisateur est redirigé vers les différentes parties du back office vu précédemment.

Option de déconnexion dans la navbar :



La case déconnexion est présente sur toutes les pages du back office via “baseback.html.twig”

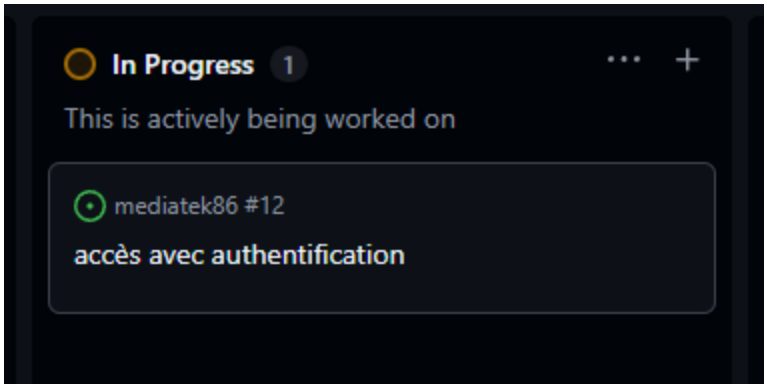
Modification de “config/packages/security.yaml” :
C’est le fichier principal pour cette tâche, il contient l’utilisateur en mémoire avec le role “ROLE_ADMIN” et un mot de passe.
“form_login” pointe vers la route login pour gérer la connexion et “logout” pointe vers la route “logout” pour procéder à la déconnexion du compte administrateur.
“access_control” lui protège toutes les routes suivant /admin/ et exige le “ROLE_ADMIN”.

Arborescence des fichiers ajoutés :

```
src/
└── Controller/
    └── LoginController.php
templates/
└── pages/
    └── login.html.twig
```

LoginController.php, contient 2 méthodes, “login()” qui affiche le formulaire et “logout()” dont le corps est vide car c’est symfony qui gère la déconnexion automatiquement.

Login.html.twig, contient le formulaire avec les champs (_username, _password, _csrf_token)



Cette tâche a duré 3h.

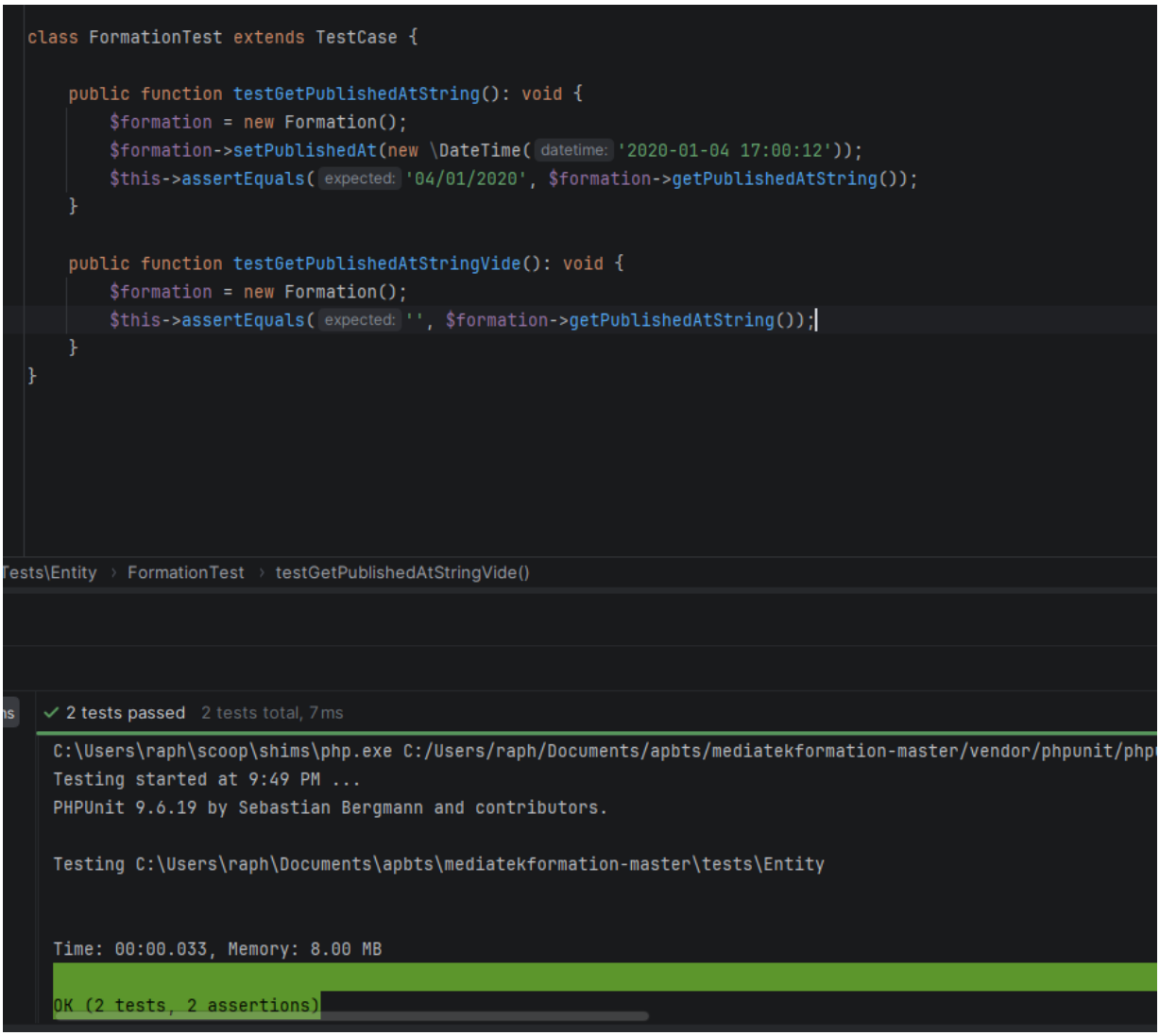
Mission 3 :

Tâche I :

Mise en place des tests unitaires d'intégration fonctionnels et de compatibilité de l'application.

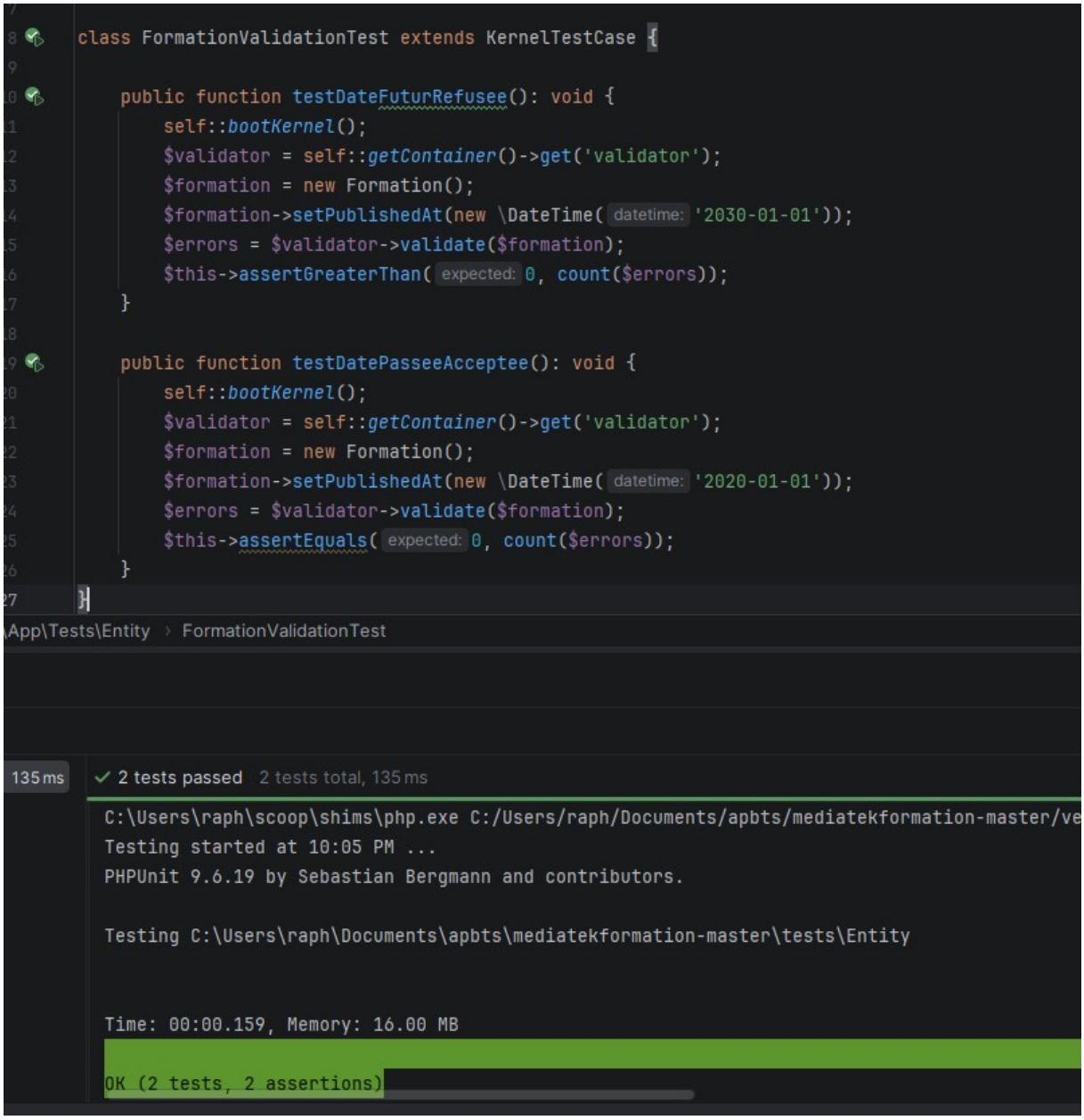
Tests unitaires :

tests/Entity/FormationTest.php, ce test controle la méthode "getPublishedAtString()" de l'entity "Formation".
Pour cela on crée une formation avec la date 2020-01-04 et on vérifie que la méthode retourne bien la date exacte donc : 2020-01-04.
Un second test vérifie que la méthode retourne une chaine vide si aucune date n'est renseignée.



Tests d’intégration sur les règles de validation :

tests/Entity/FormationValidationTest.php, pour pouvoir tester la validation de la date j’ai ajouté “#[Assert\LessThanOrEqual(“today”)]” sur le champ “publishedAt” de l’entity “Formation”. Deux tests sont effectués, un avec une date dans le futur qui doit retourner une erreur de validation et un avec une date passée qui doit être acceptée.



Tests d’intégration sur les Repository :

tests/Repository/FormationRepositoryTest.php & tests/Repository/PlaylistRepositoryTest.php, utilisent KernelTestCase avec une bdd de test (mediatekformation_test), ils controlent les méthodes ajoutées dans les repository.
- findAllOrderBy() en ASC et DESC vérifie que le premier résultat est bien trié
- findByContainValue() vérifie que tous les résultats contiennent la valeur recherchée
- findAllLasted() vérifie que le bon nombre de formation est retourné
- findAllOrderByFormationsCount() en ASC & DESC vérifie l’ordre par nombre de formations

```php
class FormationRepositoryTest extends KernelTestCase {

    public function testFindAllOrderByTitleAsc(): void {
        $formations = $this->repo->findAllOrderBy( champ: 'title',  ordre: 'ASC');
        $this->assertGreaterThan( expected: 0, count($formations));
        $premier = $formations[0]->getTitle();
        $deuxieme = $formations[1]->getTitle();
        $this->assertTrue( condition: $premier <= $deuxieme);
    }

    public function testFindAllOrderByTitleDesc(): void {
        $formations = $this->repo->findAllOrderBy( champ: 'title',  ordre: 'DESC');
        $this->assertGreaterThan( expected: 0, count($formations));
        $premier = $formations[0]->getTitle();
        $deuxieme = $formations[1]->getTitle();
        $this->assertTrue( condition: $premier >= $deuxieme);
    }

    public function testFindByContainValue(): void {
        $formations = $this->repo->findByContainValue( champ: 'title',  valeur: 'Eclipse');
        $this->assertGreaterThan( expected: 0, count($formations));
        foreach($formations as $f){
            $this->assertStringContainsString( needle: 'Eclipse', $f->getTitle());
        }
```

\App\Tests\Repository › FormationRepositoryTest

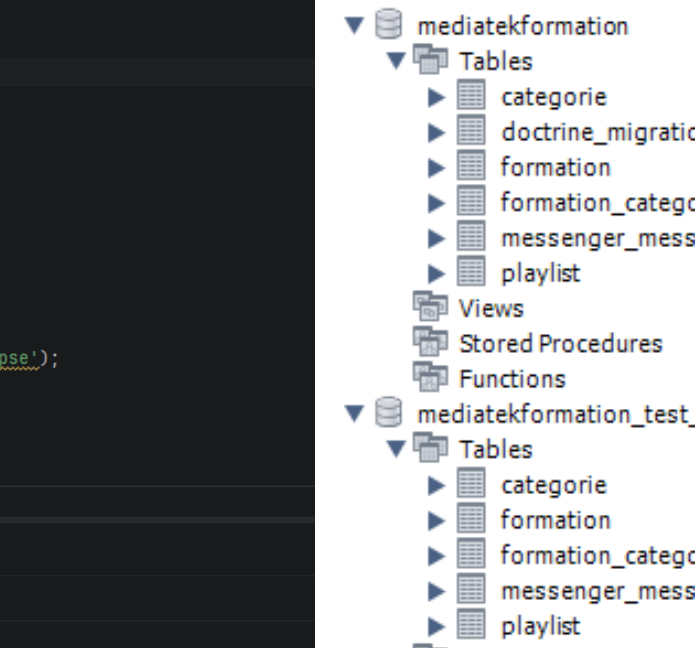
239 ms ✓ 4 tests passed 4 tests total, 239 ms

```
C:\Users\raph\scoop\shims\php.exe C:/Users/raph/Documents/apbts/mediatekformation-master/vendor/phpunit/phpuni
Testing started at 11:03 PM ...
PHPUnit 9.6.19 by Sebastian Bergmann and contributors.

Testing C:\Users\raph\Documents\apbts\mediatekformation-master\tests\Repository


Time: 00:00.267, Memory: 20.00 MB

OK (4 tests, 15 assertions)
```



Tests fonctionnels :

tests/Controller/* , ces controllers utilisent tous “WebTestCase”, cela permet de vérifier que la page d’accueil répond avec un code, que les tris fonctionnent en controlant la première ligne du tableau, que les filtres fonctionnent en controlant le nombre de lignes et la première ligne, que le clic sur un lien dans la liste amène bien sur la bonne page avec un code 200.

Exemple avec tests/Controller/AccueilControllerTest.php :

```php
<?php

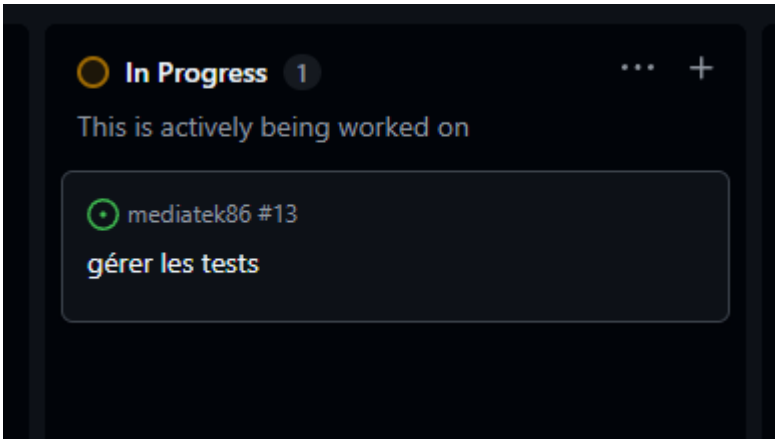
namespace App\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class AccueilControllerTest extends WebTestCase {

    public function testPageAccueilAccessible(): void {
        $client = static::createClient();
        $client->request( method: 'GET', uri: '/');
        $this->assertResponseIsSuccessful();
    }
}
```

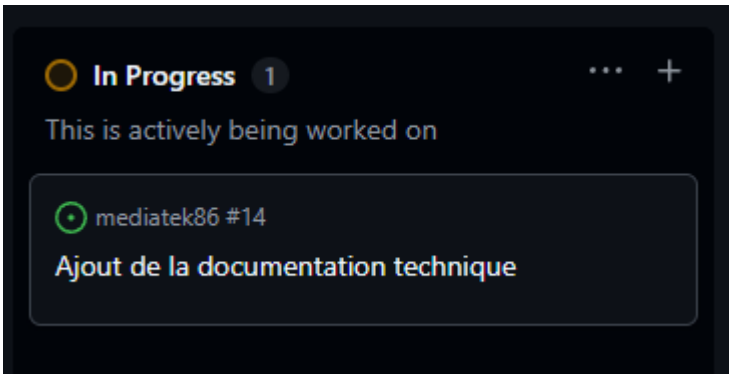
Pour la partie front office du site je l’ai testé sur (firefox, brave, safari), ce qui m’a permis de faire le test sur les 3 moteurs les plus utilisés dans le monde (Gecko, Blink et WebKit) et je n’ai pas rencontré de problème.



Pour la durée je ne sais pas trop n'ayant pas effectué cette tâche en une journée mais je pense entre 5 et 8 heures.

Tâche II :

Ajout de la documentation technique dans le code.
La documentation technique a ensuite été générée via phpDocumentor avec la commande :
"php phpDocumentor.phar -d src/ -t docs/"
L'ide utilisé au cours de cette activité a été phpstorm de mon coté je crois ne pas l'avoir mentionnée dans ce bilan.

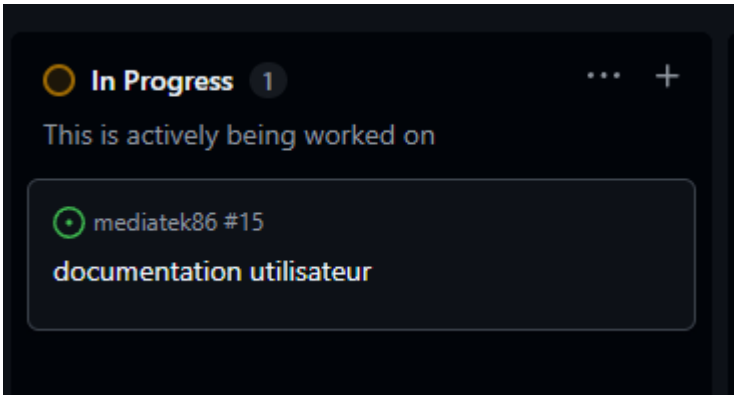


Cette tâche a duré 40 minutes.

Tâche III :

Ajout de la documentation utilisateur.
Lien vers la vidéo :

https://www.youtube.com/watch?v=WEkONaJWiqM



Cette tâche m’a duré environ 30 minutes.

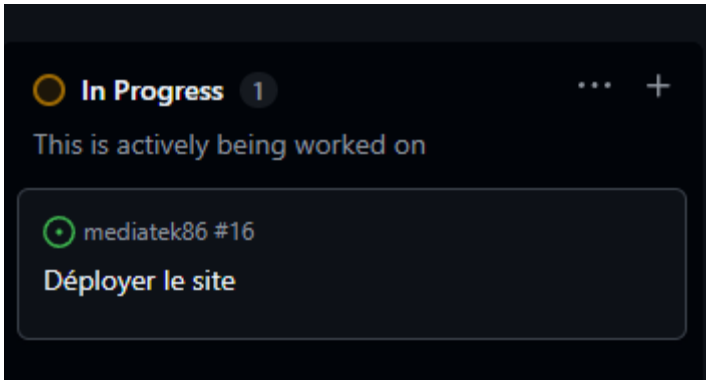
Mission 4

Tâche I :

- Le site a été déployé sur un VPS OVH Ubuntu avec ssh et l’application WinSCP.
- Voici l’ordre des manipulations réalisés :
- Installation de nginx, PHP 8.4, MySQL 8.4 et composer sur le serveur.
- Création de la base de données et d’un utilisateur MySQL.
- Upload du projet via WinSCP dans /var/www/mediatek (l’ap est isolé tu restes du portfolio).
- Installation des dépendances avec la commande “composer install” pour ma part j’ai utilisé “composer install –no-dev ayant disable le dev mod dans Symfony.
- Import de la base de données mediatek86.sql
- Configuration du virtual host nginx pointant vers le fichier public.
- Activation du certificat HTTPS sur les nouveaux sous domaines (mediatek.taizsys.xyz & docs.mediatek.taizsys.xyz).
- Après analyse, ajout des headers de sécurité (xframe options, xcontent-type options; HSTS)

Comme énoncé plus tôt la documentation technique est disponible via l’url :
https://docs-mediatek.taizsys.xyz

La page CGU a également été mise à jour le site étant maintenant hébergé.



Cette tâche a duré environ 1h30.

Tâche II :

A la suite du déploiement du site j'ai enchainé avec la sauvegarde de la BDD, et voici le script qui a été utilisé pour cette tâche :

```
#!/bin/bash
DATE=$(date +%Y-%m-%d)
mysqldump --no-tablespaces -u mediatek -pKozeno1338! mediatekformation >
/var/backups/mediatek/bdd_$DATE.sql
```

Grâce à ce script la restauration est déjà possible manuellement.

Mais pour obtenir une exécution automatique chaque jour il faut utiliser un cron job :
0 2 * * * /var/backups/mediatek/backup.sh

Maintenant chaque fichier est nommé avec la date du jour (bdd_2026_03_04.sql) ce qui permet de tenir un historique clair dans le temps.

Voici le mode opératoire utilisé en cas de restauration :

mysql -u mediatek -pmdp mediatekformation < /var/backups/mediatek/bdd_AAAA-MM-JJ.sql



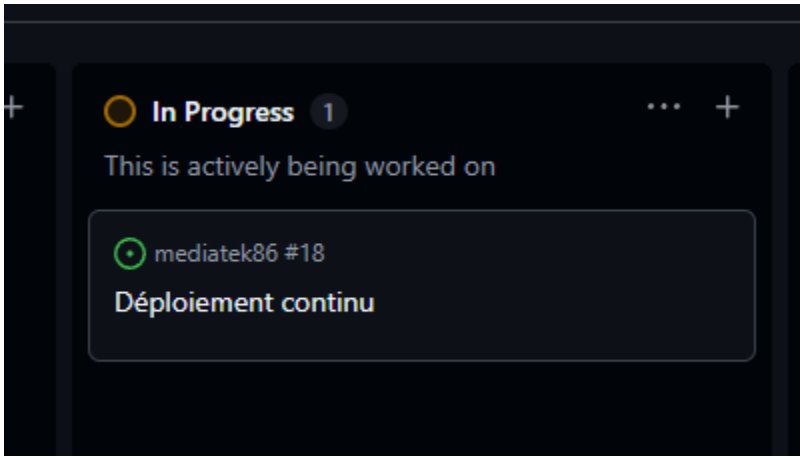
Cette tâche a duré 1h.

Tâche III :

J'ai créé un workflow Github actions dans ".github/workflows/deploy.yml". De manière à ce que chaque push effectué sur la branche main Github se connecte automatiquement en SSH au VPS et exécute les commandes de mise à jour:

- git pull origin main

La connexion SSH est sécurisée et effectuée à l’aide d’une key SSH, celle-ci est stockée dans les secrets du dépôt Github.



Cette tâche a duré 1h.

Bilan :

Dans l’ensemble, toutes les missions demandées ont été réalisées et le projet est fonctionnel.

J’ai d’abord retravaillé le code existant en suivant les recommandations de Sonarlint. Ça m’a permis de rendre le code plus propre et plus compréhensible notamment en mettant les constantes en majuscules, en simplifiant certaines conditions et en ajoutant les attributs alt et caption qui manquaient.

Ensuite, j’ai ajouté une colonne qui affiche le nombre de formations par playlist. J’ai aussi mis en place un système de tri (croissant et décroissant), ce qui rend l’affichage plus pratique à utiliser.J’ai développé un back office complet qui permet de gérer les formations, les playlists et les catégories. On peut ajouter, modifier et supprimer facilement les données.
L’accès à ce back office est sécurisé avec un système de connexion basé sur un compte administrateur, ce qui évite que n’importe qui puisse modifier les données.

Concernant les tests, j’en ai réalisé plusieurs types (unitaires, intégration, fonctionnels et compatibilité). Tous les tests passent correctement, avec 20 tests et 54 assertions validées.

J’ai aussi généré la documentation technique avec phpDocumentor, ce qui permet d’avoir une vue claire du projet et de son fonctionnement. Une vidéo de présentation a été réalisée pour montrer comment utiliser l’application et présenter ses principales fonctionnalités.

Enfin, l’application a été déployée sur un VPS OVH. Le site est sécurisé en HTTPS, avec une sauvegarde automatique tous les jours et un système de déploiement continu via GitHub Actions.

Problèmes rencontrés :

Cache de Symfony qui n’actualisait pas la page, j’ai perdu beaucoup de temps à debug des choses fonctionnelles à cause de ça.

Connexion à l’utilisateur root sur OVH, par défaut OVH bloque les connexions en root. Il faut donc l’autoriser dans nano /etc/ssh/sshd_config. L’utilisateur root est obligatoire pour téléverser des fichiers dans le VPS avec WinSCP et accorder des droits à un utilisateur ne marche pas dans ce cas.

Conflit de fichiers lors du git pull sur le VPS, j’ai dû faire une sauvegarde du .env puis git reset --hard.

Cette fois-ci, c’est un problème lié à la sécurité, lorsque le site est passé en déploiement continu je n’avais pas pensé au fait que security.yaml serait lui aussi importé sur le VPS, ce qui a donc causé un problème de sécurité évident, (mot de passe en claire sur la branche main Github).