COMPTE RENDU ATELIER 1

Sommaire

Contexte

MediaTek86 est une médiathèque située en Vienne (86) qui met à disposition de ses adhérents des ordinateurs connectés à Internet. Dans le cadre de sa politique de formation, MediaTek86 propose des auto-formations en ligne sur des outils numériques, accessibles via un site web dédié : mediatekformation.

Le chef de projet m’a confié plusieurs missions et m’a chargé de finaliser et déployer ce site qui a déjà été développé par un autre développeur. Ces missions concernent la correction du code existant, l'ajout de fonctionnalités, le développement d'un back office, la mise en place de tests, la documentation et le déploiement du site.

L’existant

Le premier développeur avait déjà réalisé une partie du site, constituant le front office. Cette partie permettait aux visiteurs de consulter la liste des formations et des playlists disponibles, d'effectuer des tris et des filtres sur ces listes, et d'accéder au détail d'une formation ou d'une playlist. Cependant, le code présentait quelques manquements aux bonnes pratiques de codage, et une fonctionnalité attendue dans le cahier des charges était absente : l'affichage du nombre de formations par playlist.

Langage et technologies utilisées

Le projet utilise les langages et technologies suivants :

Côté serveur : PHP 8.3 avec le framework Symfony 6.4, qui structure l'application selon le modèle de conception MVC (Modèle-Vue-Contrôleur). La gestion de la base de données est assurée par Doctrine, l'ORM intégré à Symfony, avec des requêtes construites via le QueryBuilder.

Côté client : HTML5, CSS3 et JavaScript, avec le framework Bootstrap 5 pour la mise en page et le rendu responsive.

Base de données : MySQL 8, gérée localement via WAMP et phpMyAdmin, et déployée sur le serveur OVHcloud.

Environnement de développement : NetBeans comme IDE, GitHub pour la gestion de versions et le déploiement continu via GitHub Actions.

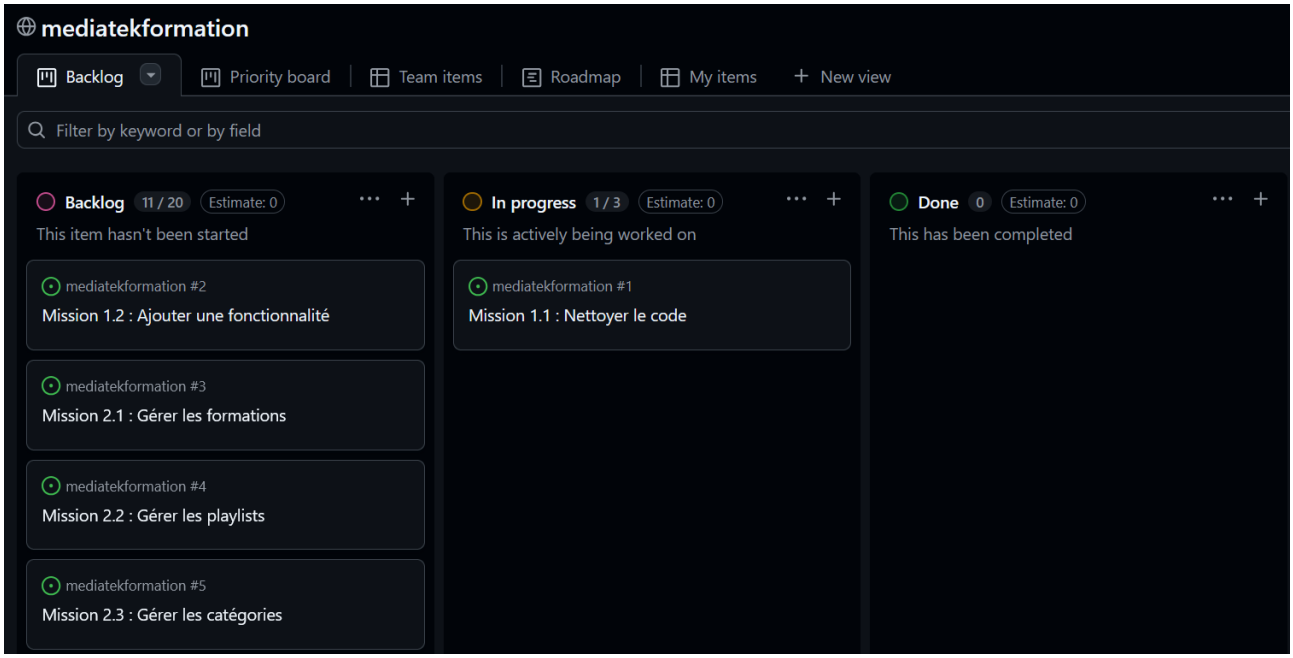
Serveur : Apache via WAMP en local, et un hébergement mutualisé OVHcloud en production, accessible à l'adresse https://mediatekformation.nathan-boudier.com.

Tests : PHPUnit pour les tests unitaires, d'intégration et fonctionnels. Et phpDocumentor pour la génération de la documentation technique.

Mission 1.1 : Nettoyage du code

Demandes de la mission :

Nettoyer le code en suivant les indications de Sonarlint (excepté le code généré automatiquement par Symfony).



Temps de réalisation estimé : 2h
Temps de réalisation réel : 1h

Erreur n°1

Intitulé SonarLint : Switch statement should have default clauses

```php
#[Route('/playlists/tri/{champ}/{ordre}', name: 'playlists.sort')]
public function sort($champ, $ordre): Response {
    switch($champ){
        case "name":
            $playlists = $this->playlistRepository->findAllOrderByName($ordre);
            break;
    }

    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_PLAYLISTS, [
                'playlists' => $playlists,
                'categories' => $categories
    ]);
}
```

Comme il n’y a qu’un seul cas possible dans le switch, je choisis de le supprimer complètement :

```php
#[Route('/playlists/tri/{champ}/{ordre}', name: 'playlists.sort')]
public function sort($champ, $ordre): Response {
    $playlists = $this->playlistRepository->findAllOrderByName($ordre);

    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_PLAYLISTS, [
                'playlists' => $playlists,
                'categories' => $categories
    ]);
}
```

Erreur n°2

Intitulé SonarLint : strings literals duplicated

```php
#[Route('/formations', name: 'formations')]
public function index(): Response {
    $formations = $this->formationRepository->findAll();
    $categories = $this->categorieRepository->findAll();
    return $this->render("pages/formations.html.twig", [
        'formations' => $formations,
        'categories' => $categories
    ]);
}
```

La chaîne "pages/formations.html.twig" est utilisée à plusieurs reprises dans le code. Je la déclare donc dans une constante :

```php
/**
 * Chemin vers la page 'formations'
 */
private const PAGE_FORMATIONS = "pages/formations.html.twig";
```

Puis je remplace la chaîne par self::PAGE_FORMATIONS

```php
#[Route('/formations', name: 'formations')]
public function index(): Response {
    $formations = $this->formationRepository->findAll();
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_FORMATIONS, [
        'formations' => $formations,
        'categories' => $categories
    ]);
}
```

Erreur n°3

Intitulé SonarLint : Method visibility should be exlicitly declared

```
function __construct(FormationRepository $formationRepository, CategorieRepository $categorieRepository) {
    $this->formationRepository = $formationRepository;
    $this->categorieRepository = $categorieRepository;
}
```

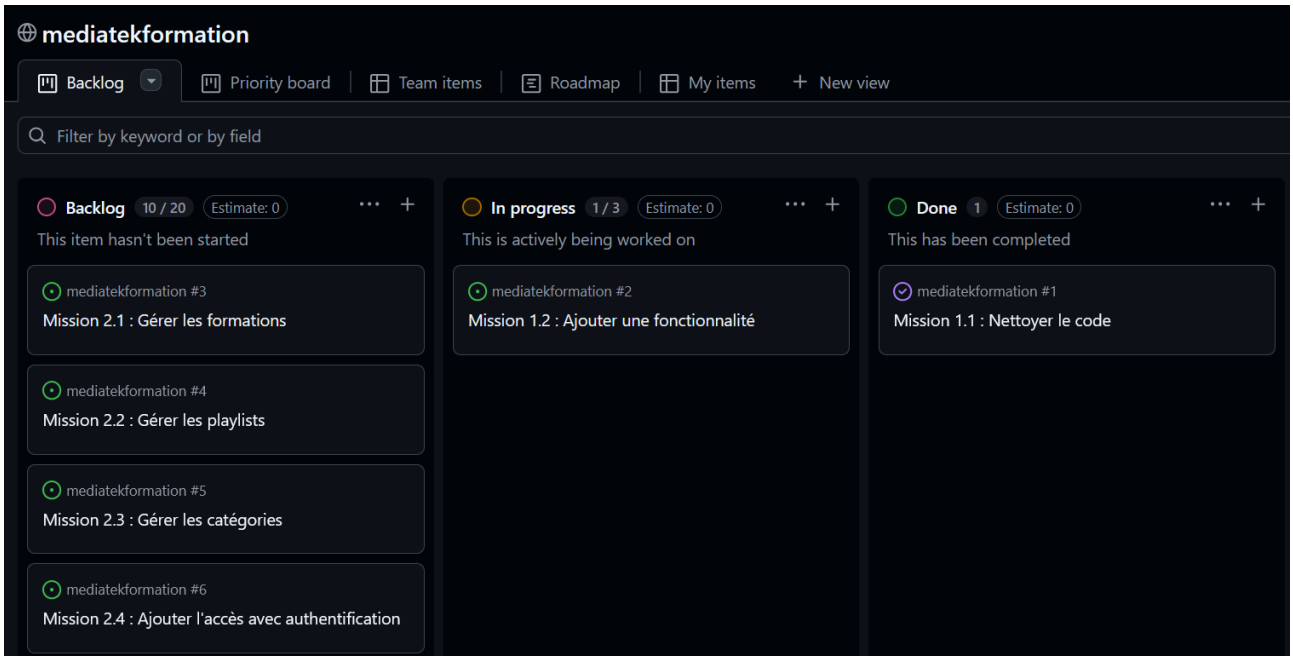
Je déclare donc la fonction en public :

```
public function __construct(FormationRepository $formationRepository, CategorieRepository $categorieRepository) {
    $this->formationRepository = $formationRepository;
    $this->categorieRepository = $categorieRepository;
}
```

Mission 1.2 : Ajouter une fonctionnalités

Demandes de la missions

Dans la page des playlists, ajouter une colonne pour afficher le nombre de formations par playlist et permettre le tri croissant et décroissant sur cette colonne. Cette information doit aussi s'afficher dans la page d'une playlist.



Temps de réalisation estimé : 1h
Temps de réalisation réel : 1h

Explication de code

J’ai ajouté une méthode dans PlaylistRepository nommée findAllOrderByNbFormations. Cette méthode retourne toutes les playlists triées sur le nombre de formations. Je reprend exactement le code de la méthode findAllOrderByName en remplaçant juste ‘p.name’ par 'COUNT(f.id)' pour effectuer le tri sur le nombre de formations :

```php
/**
 * Retourne toutes les playlists triées sur le nombre de formations
 * @param type $ordre
 * @return Playlist[]
 */
public function findAllOrderByNbFormations($ordre): array {
    return $this->createQueryBuilder('p')
        ->leftJoin('p.formations', 'f')
        ->groupBy('p.id')
        ->orderBy('COUNT(f.id)', $ordre)
        ->getQuery()
        ->getResult();
}
```

Ensuite, j’ai modifié la fonction sort dans PlaylistController en remettant le switch qui était présent dans le code d’origine, mais cette fois-ci avec plusieurs cas : un cas pour le tri sur le nom, un cas pour le tri sur le nombre de formations, et un cas par défaut qui tri aussi sur le nom.

```php
#[Route('/playlists/tri/{champ}/{ordre}', name: 'playlists.sort')]
public function sort($champ, $ordre): Response {
    switch($champ) {
        case "name":
            $playlists = $this->playlistRepository->findAllOrderByName($ordre);
            break;
        case "nbFormations":
            $playlists = $this->playlistRepository->findAllOrderByNbFormations($ordre);
            break;
        default:
            $playlists = $this->playlistRepository->findAllOrderByName('ASC');
            break;
    }

    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_PLAYLISTS, [
                'playlists' => $playlists,
                'categories' => $categories
    ]);
}
```

Puis j’ai modifié l’affichage dans playlists.html.twig pour ajouter l’entête nb formations et la colonne :

```
<th class="text-center align-top" scope="col">
    nb formations
    <a href="{{ path('playlists.sort', {champ:'nbFormations', ordre:'ASC'}) }}"
       class="btn btn-info btn-sm active" role="button" aria-pressed="true"><</a>
    <a href="{{ path('playlists.sort', {champ:'nbFormations', ordre:'DESC'}) }}"
       class="btn btn-info btn-sm active" role="button" aria-pressed="true">></a>
</th>
```

```
<td class="text-center">
    {{ playlists[k].formations|length }}
</td>
```

Enfin, j’ai ajouté le nombre de formations dans la page playlist.html.twig, sous le titre :

```
<p><strong>Nombre de formations :</strong> {{ playlistformations|length }}</p>
```

Résultat final

MediaTek86

Des formations pour tous
sur des outils numériques

Accueil Formations Playlists

playlist < >

filtrer

catégories

nb formations < >

Bases de la programmation (C#)	C# POO	74	Voir détail
Compléments Android (programmation mobile)	Android	13	Voir détail
Cours Composant logiciel	Cours	2	Voir détail
Cours Curseurs	SQL Cours POO	2	Voir détail

MediaTek86

Des formations pour tous
sur des outils numériques

Accueil Formations Playlists

Compléments Android (programmation mobile)

Nombre de formations : 13

catégories : Android

description :
Chaque vidéo est indépendante et présente une notion spécifique.
Prérequis : avoir des notions de base en programmation sous Android et en programmation objet (si vous ne connaissez pas du tout Android, commencez pas suivre la playlist "TP Android" qui part d'un niveau 0 et montre toutes les notions essentielles).

Android Studio (complément n°1) : Navigation Drawer et Fragment

Android Studio (complément n°2) : Récupérer les contacts du mobile

Android Studio (complément n°3) : Activity dépendante

Android Studio (complément n°4) : Envoyer un SMS

Android Studio (complément n°5) : Récupérer les photos du mobile

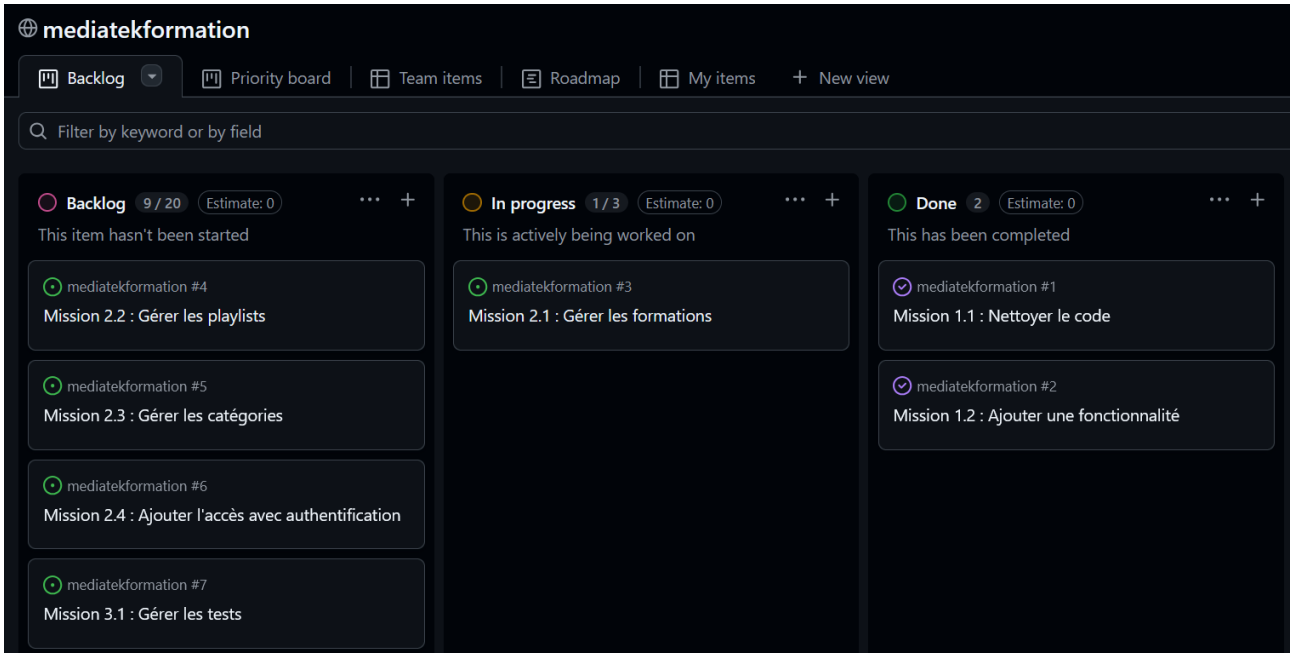
Android Studio (complément n°6) : Redimensionner

Mission 2.1 : Gérer les formations

Demandes de la mission

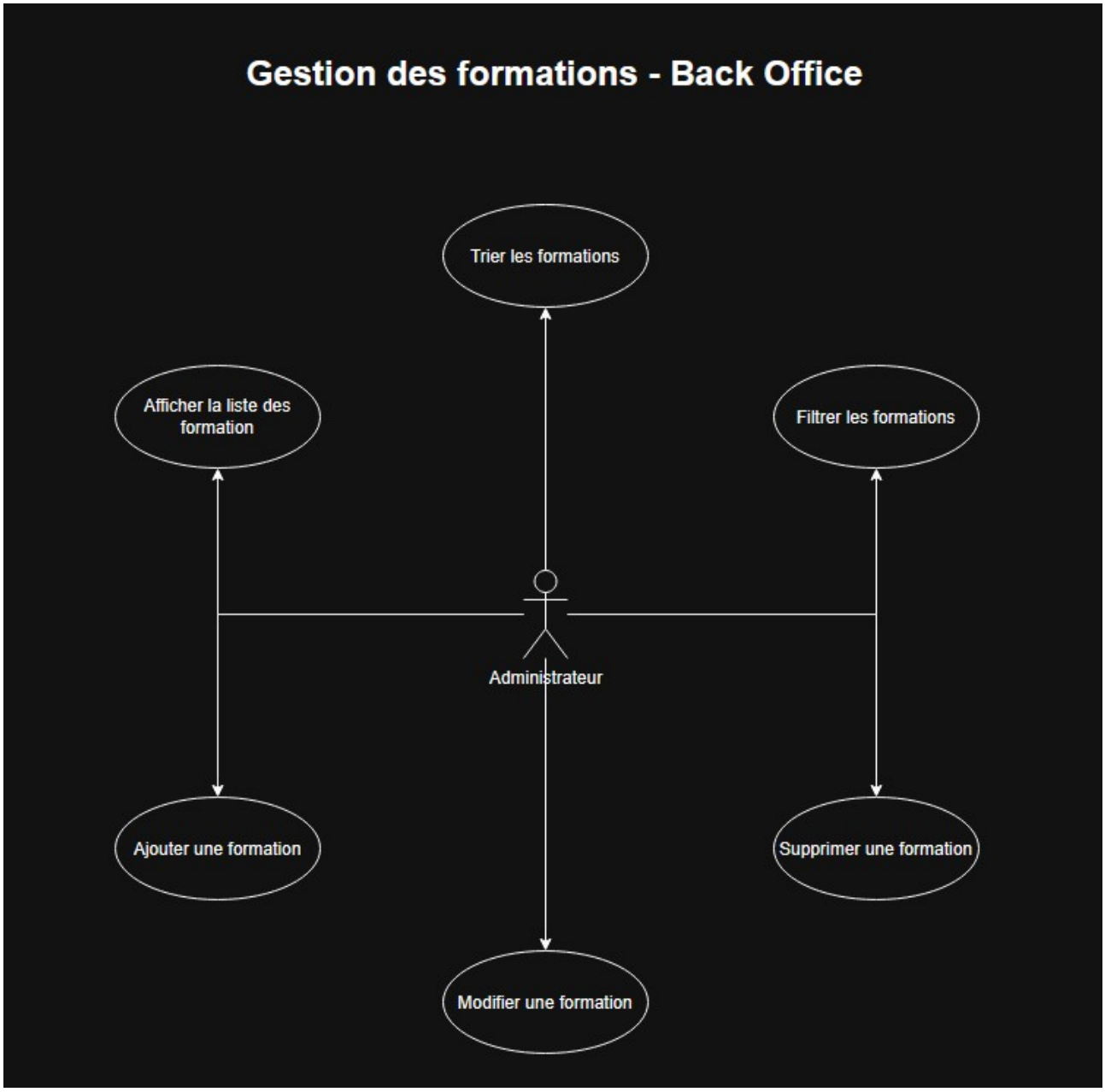
Une page doit permettre de lister les formations et, pour chaque formation, afficher un bouton permettant de la supprimer (après confirmation) et un bouton permettant de la modifier.

- Si une formation est supprimée, il faut aussi l'enlever de la playlist où elle se trouvait.
- Les mêmes tris et filtres présents dans le front office doivent être présents dans le back office.
- Un bouton doit permettre d'accéder au formulaire d'ajout d'une formation. Les saisies doivent être contrôlées. Seul le champ "description" n'est pas obligatoire ainsi que la sélection de catégories (une formation peut n'avoir aucune catégorie). La playlist et la ou les catégories doivent être sélectionnées dans une liste (une seule playlist par formation, plusieurs catégories possibles par formation). La date ne doit pas être saisie mais sélectionnée. Elle ne doit pas être postérieure à la date du jour.
- Le clic sur le bouton permettant de modifier une formation doit amener sur le même formulaire, mais cette fois prérempli.



Temps de réalisation estimé : 5h
Temps de réalisation réel : 4h30

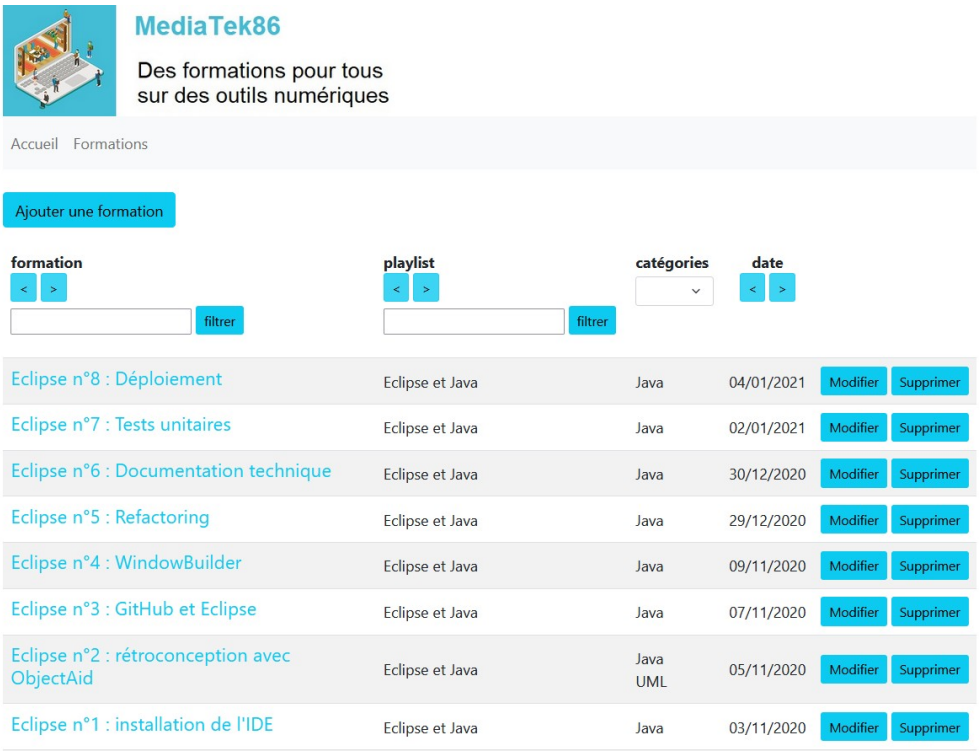
Diagramme de cas d’utilisation



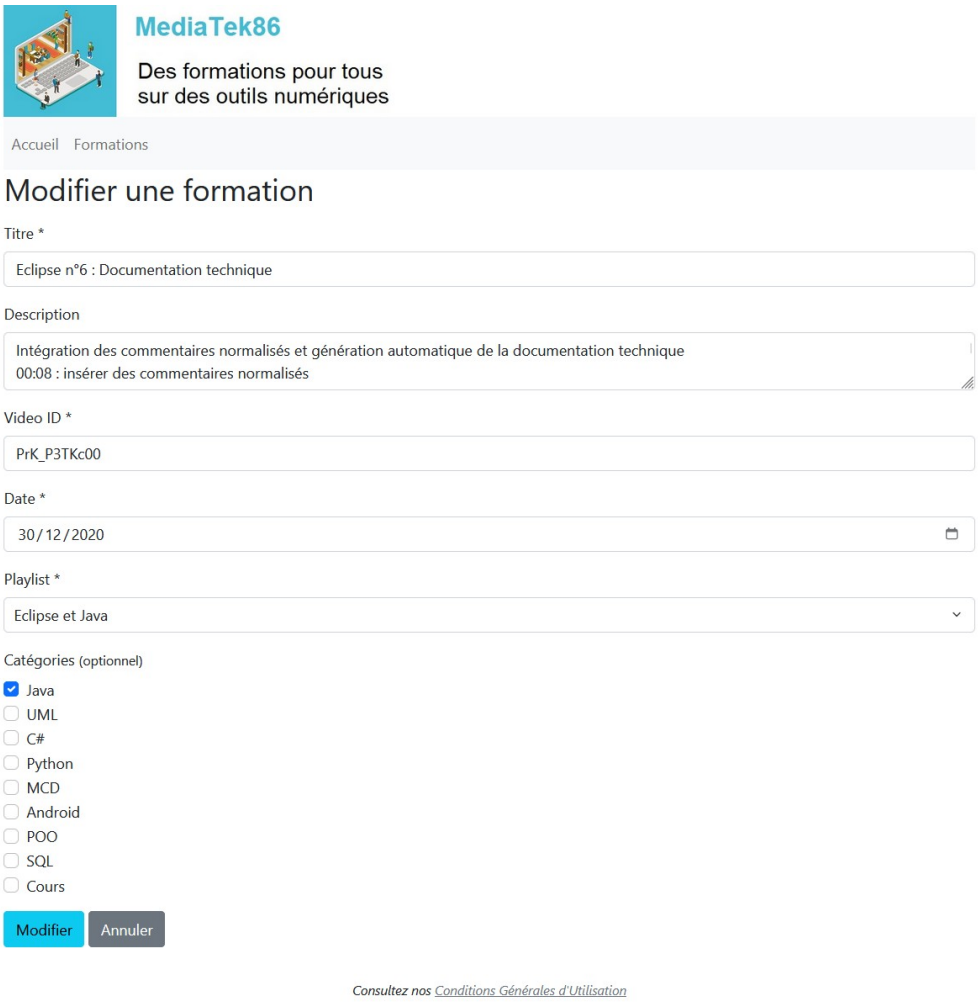
Nouvelles pages

Une page d'accueil admin :

Une page d'admin formations qui affiche les formations, permet de les trier, d'ajouter, modifier et supprimer une formation :

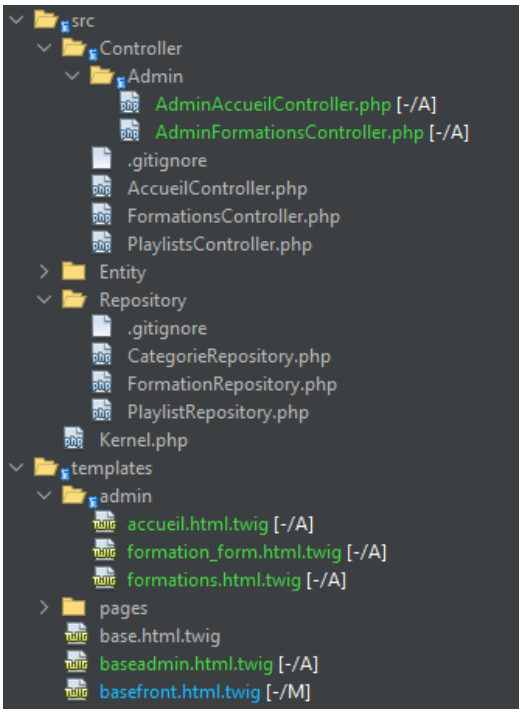


Une page d'admin formation_form qui est un formulaire d'ajout ou de modification d'une formation :



Explications de code

Voici l'arborescence du projet avec en vert les quatre fichier qui ont été créés :



J’ai commencé par créer une page d’accueil admin : admin/accueil.html.twig

```
{% extends "baseadmin.html.twig" %}

{% block body %}
    <br /><h2 class="mb-4">Bienvenue dans l'espace d'administration</h2>

{% endblock %}
```

J’ai donc dû créer un contrôleur très simple pour rediriger vers cette page :

```
<?php
namespace App\Controller\Admin;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

/**
 * Controleur admin de l'accueil
 */
class AdminAccueilController extends AbstractController {

    /**
     * Chemin vers la page admin 'accueil'
     */
    private const PAGE_ADMIN_ACCUEIL = "admin/accueil.html.twig";

    #[Route('/admin', name: 'admin')]
    public function index(): Response {
        return $this->render(self::PAGE_ADMIN_ACCUEIL);
    }
}
```

J’ai ensuite créé le template baseadmin.html.twig qui reprend le template de base en changeant juste le menu de navigation :

```
<!-- menu -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class="nav-link" href="{{ path('accueil') }}">Accueil</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('admin.formations') }}">Formations</a>
            </li>
        </ul>
    </div>
</nav>
```

J’ai aussi créé le template formations.html.twig dans le dossier admin qui reprend la même structure que le template formations de base en ajoutant un bouton pour ajouter une formation, et deux boutons modifier et supprimer dans une nouvelle colonne.

```
<br /><a href="{{ path('admin.formations.ajouter') }}" class="btn btn-info mb-3">Ajouter une formation</a>
```

```
<td class="text-center" style="white-space: nowrap">
    <a href="{{ path('admin.formations.modifier', {id:formations[k].id}) }}"
       class="btn btn-info btn-sm">Modifier</a>
    <a href="{{ path('admin.formations.supprimer', {id:formations[k].id}) }}"
       class="btn btn-info btn-sm"
       onclick="return confirm('Êtes-vous sûr de vouloir supprimer cette formation ?')">Supprimer</a>
</td>
```

J’ai également créé le template formation_form.html.twig dans le dossier admin qui est le formulaire pour ajouter ou modifier une formation.

Et enfin le contrôleur AdminFormationsController.php qui reprends exactement la même structure que FormationsController.php, en ajoutant les fonctions pour ajouter, modifier, et supprimer une formation :

```
#[Route('/admin/formations/ajouter', name: 'admin.formations.ajouter')]
public function ajouter(Request $request): Response {
    // Si le formulaire est bien soumis
    if($request->get("title")){
        $formation = new Formation();
        $formation->setTitle($request->get('title'));
        $formation->setDescription($request->get('description'));
        $formation->setVideoId($request->get('videoId'));

        // On définit la date de publication
        $date = new \DateTime($request->get('publishedAt'));
        $formation->setPublishedAt($date);

        // On définit la playlist
        $playlist = $this->playlistRepository->find($request->get('playlist'));
        $formation->setPlaylist($playlist);

        // On définit la catégorie
        $categorieIds = $request->get('categories', []);
        foreach($categorieIds as $categorieId){
            $categorie = $this->categorieRepository->find($categorieId);
            $formation->addCategory($categorie);
        }

        $this->formationRepository->add($formation);
        return $this->redirectToRoute('admin.formations');
    }

    // Sinon on affiche le formulaire vide
    return $this->render(self::PAGE_ADMIN_FORMATION_FORM, [
        'formation' => new Formation(),
        'playlists' => $this->playlistRepository->findAll(),
        'categories' => $this->categorieRepository->findAll(),
        'estModification' => false
    ]);
}
```

```php
#[Route('/admin/formations/modifier/{id}', name: 'admin.formations.modifier')]
public function modifier(int $id, Request $request): Response {
    $formation = $this->formationRepository->find($id);

    // Si le formulaire est bien soumis
    if($request->get("title")){
        $formation->setTitle($request->get('title'));
        $formation->setDescription($request->get('description'));
        $formation->setVideoId($request->get('videoId'));

        // On met à jours la date de publication
        $date = new \DateTime($request->get('publishedAt'));
        $formation->setPublishedAt($date);

        // On met à jours la playlist
        $playlist = $this->playlistRepository->find($request->get('playlist'));
        $formation->setPlaylist($playlist);

        // On met à jours les catégories
        $formation->getCategories()->clear();
        $categorieIds = $request->get('categories', []);
        foreach($categorieIds as $categorieId){
            $categorie = $this->categorieRepository->find($categorieId);
            $formation->addCategory($categorie);
        }

        $this->formationRepository->add($formation);
        return $this->redirectToRoute('admin.formations');
    }

    // Sinon on affiche le formulaire prérempli
    return $this->render(self::PAGE_ADMIN_FORMATION_FORM, [
        'formation' => $formation,
        'playlists' => $this->playlistRepository->findAll(),
        'categories' => $this->categorieRepository->findAll(),
        'estModification' => true
    ]);
}
```

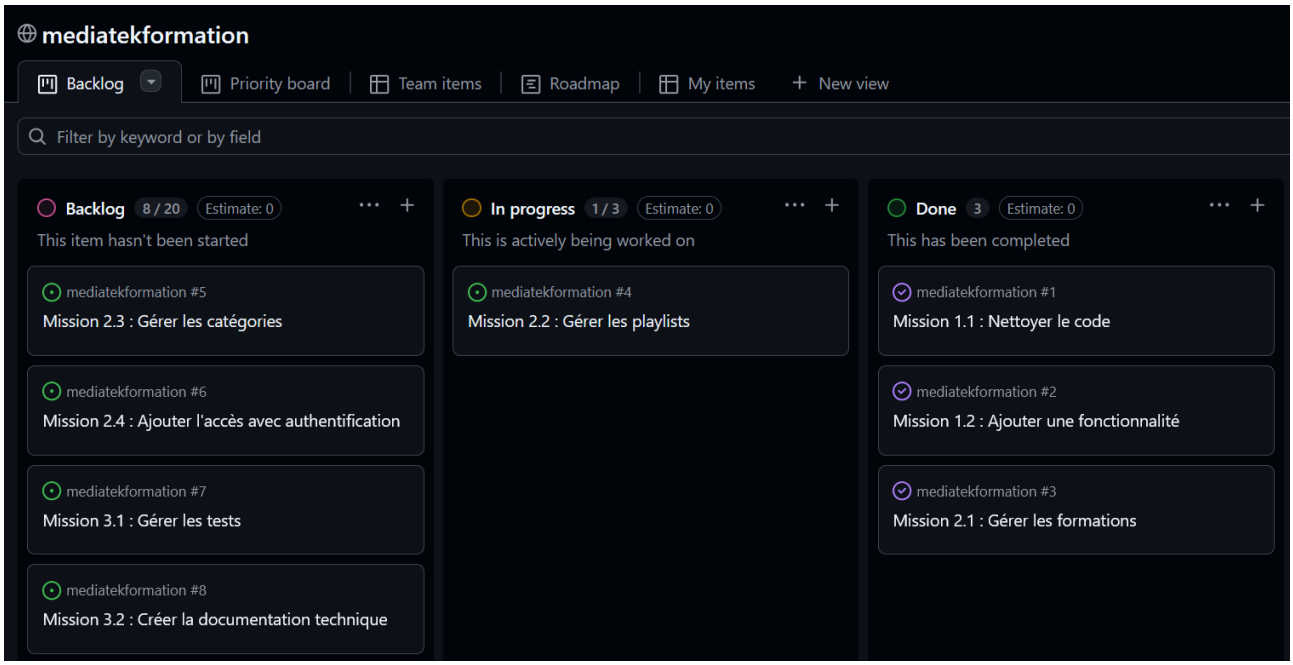
```php
#[Route('/admin/formations/supprimer/{id}', name: 'admin.formations.supprimer')]
public function supprimer(int $id): Response {
    $formation = $this->formationRepository->find($id);
    $this->formationRepository->remove($formation);
    return $this->redirectToRoute('admin.formations');
}
```

Mission 2.2 : Gérer les playlists

Demandes de la mission

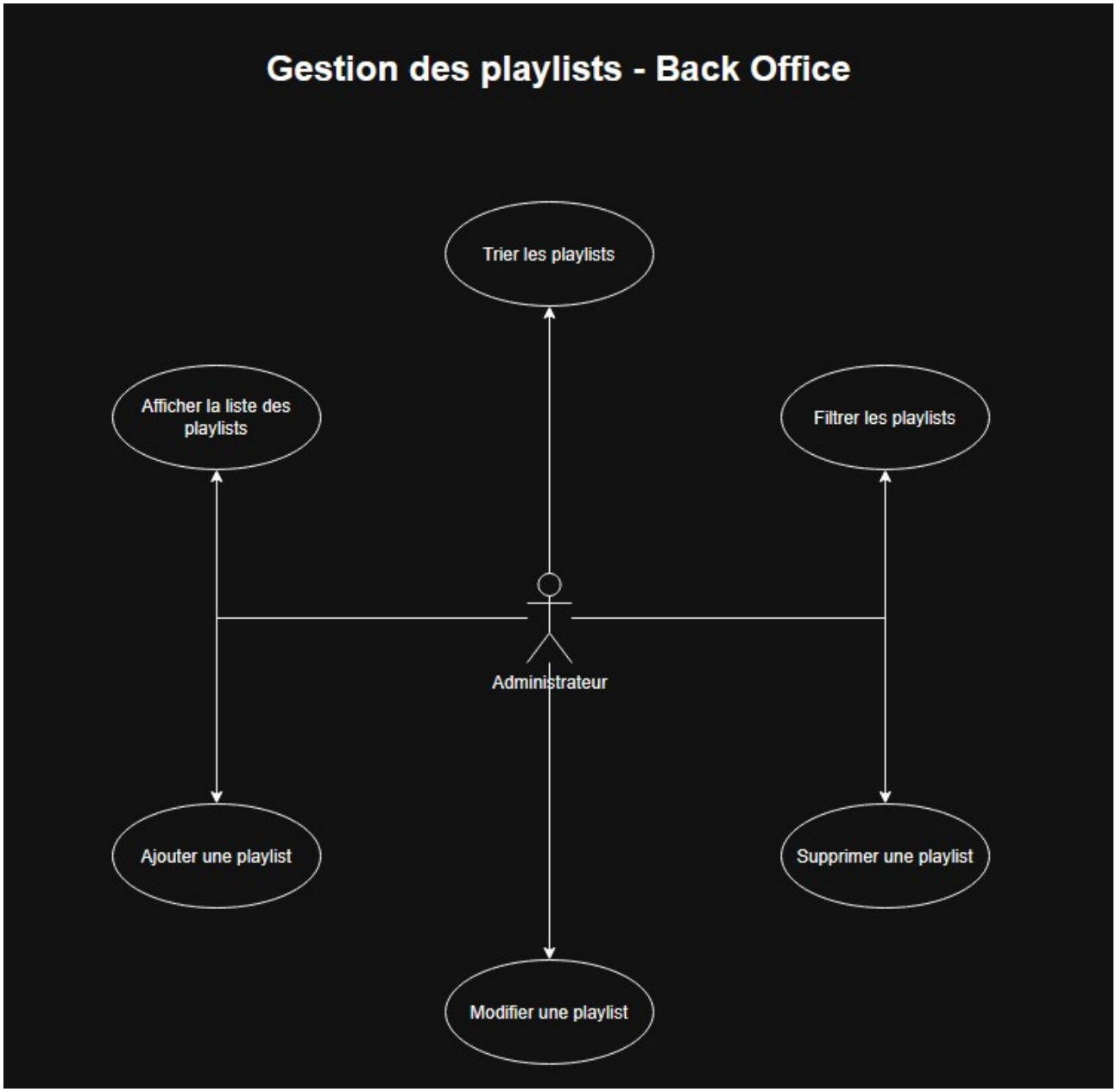
Une page doit permettre de lister les playlists et, pour chaque playlist, afficher un bouton permettant de la supprimer (après confirmation) et un bouton permettant de la modifier.

- La suppression d'une playlist n'est possible que si aucune formation n'est rattachée à elle.
- Les mêmes tris et filtres présents dans le front office doivent être présents dans le back office.
- Un bouton doit permettre d'accéder au formulaire d'ajout d'une playlist. Les saisies doivent être contrôlées. L'ajout d'une playlist consiste juste à saisir son nom et sa description. Seul le champ name est obligatoire.
- Le clic sur le bouton permettant de modifier une playlist doit amener sur le même formulaire, mais cette fois prérempli. Cette fois, la liste des formations de la playlist doit apparaître, mais il ne doit pas être possible d'ajouter ou de supprimer une formation : ce n'est que dans le formulaire de la formation qu'il est possible de préciser sa playlist de rattachement.



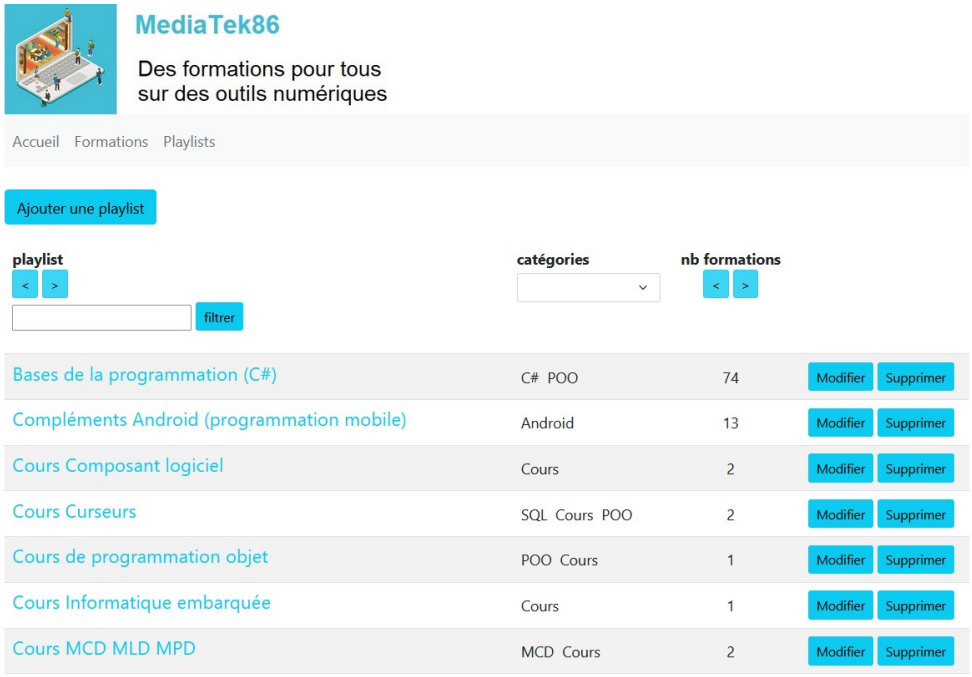
Temps de réalisation estimé : 5h
Temps de réalisation réel : 4h30

Diagramme de cas d’utilisation

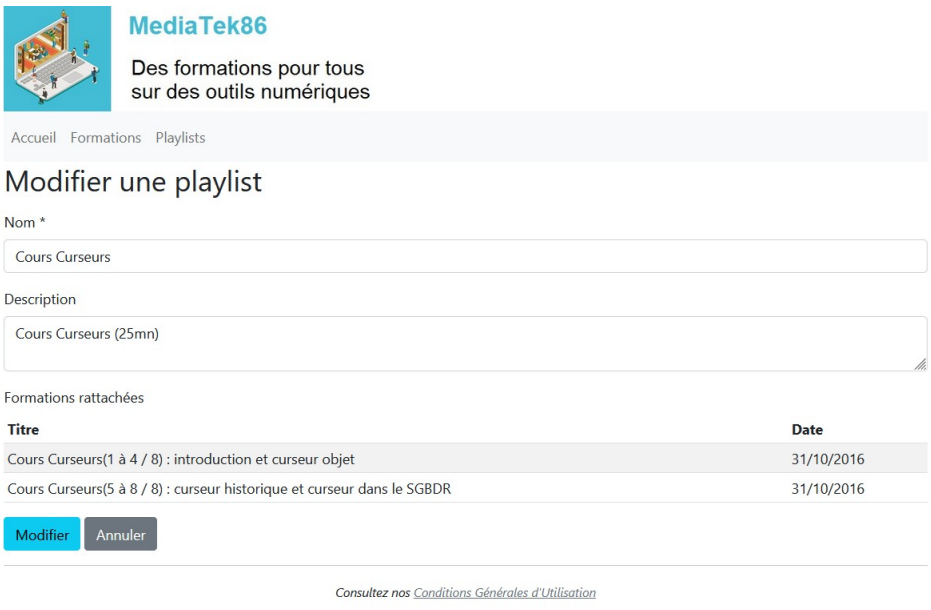


Nouvelles pages

Une page d’admin playlists qui affiche les playlists, permet de les trier, d’ajouter, modifier et supprimer une playlist :

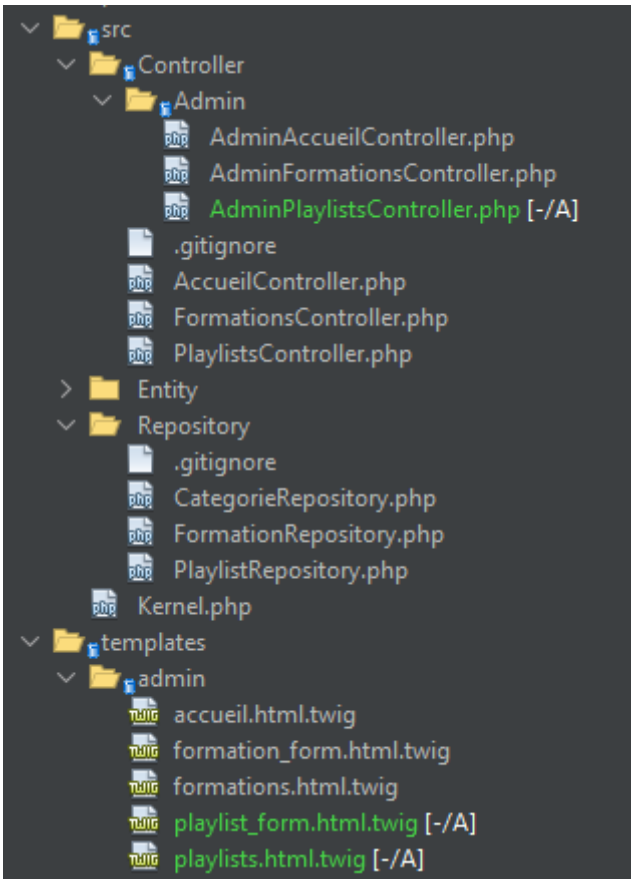


Une page d’admin playlist_form qui est un formulaire d’ajout ou de modification d’une playlist :



Explication de code

Voici l’arborescence du projet avec en vert les trois fichier qui ont été créés :



Le template baseadmin.html.twig est modifié pour ajouter l’option Playlists dans le menu :

```
<!-- menu -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class="nav-link" href="{{ path('accueil') }}">Accueil</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('admin.formations') }}">Formations</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('admin.playlists') }}">Playlists</a>
            </li>
        </ul>
    </div>
</nav>
```

Le template playlists.html.twig dans le dossier admin qui reprend la même structure que le template playlists de base en ajoutant un bouton pour ajouter une playlist, et deux boutons modifier et supprimer dans une nouvelle colonne.

```
<br /><a href="{{ path('admin.playlists.ajouter') }}" class="btn btn-info mb-3">Ajouter une playlist</a>
```

```
<td class="text-center" style="white-space: nowrap">
    <a href="{{ path('admin.playlists.modifier', {id:playlists[k].id}) }}"
       class="btn btn-info btn-sm">Modifier</a>
    <a href="{{ path('admin.playlists.supprimer', {id:playlists[k].id}) }}"
       class="btn btn-info btn-sm"
       onclick="return confirm('Êtes-vous sûr de vouloir supprimer cette playlist ?')">Supprimer</a>
</td>
```

Le template playlist_form.html.twig dans le dossier admin qui est le formulaire pour ajouter ou modifier une playlist.

Et le contrôleur AdminPlaylistsController.php qui reprends exactement la même structure que PlaylistsController.php, en ajoutant les fonctions pour ajouter, modifier, et supprimer une playlist :

```
#[Route('/admin/playlists/ajouter', name: 'admin.playlists.ajouter')]
public function ajouter(Request $request): Response {
    // Si le formulaire est bien soumis
    if($request->get("name")){
        $playlist = new Playlist();
        $playlist->setName($request->get('name'));
        $playlist->setDescription($request->get('description'));
        $this->playlistRepository->add($playlist);
        return $this->redirectToRoute('admin.playlists');
    }

    // Sinon on affiche le formulaire vide
    return $this->render(self::PAGE_ADMIN_PLAYLIST_FORM, [
        'playlist' => new Playlist(),
        'formations' => [],
        'estModification' => false
    ]);
}
```

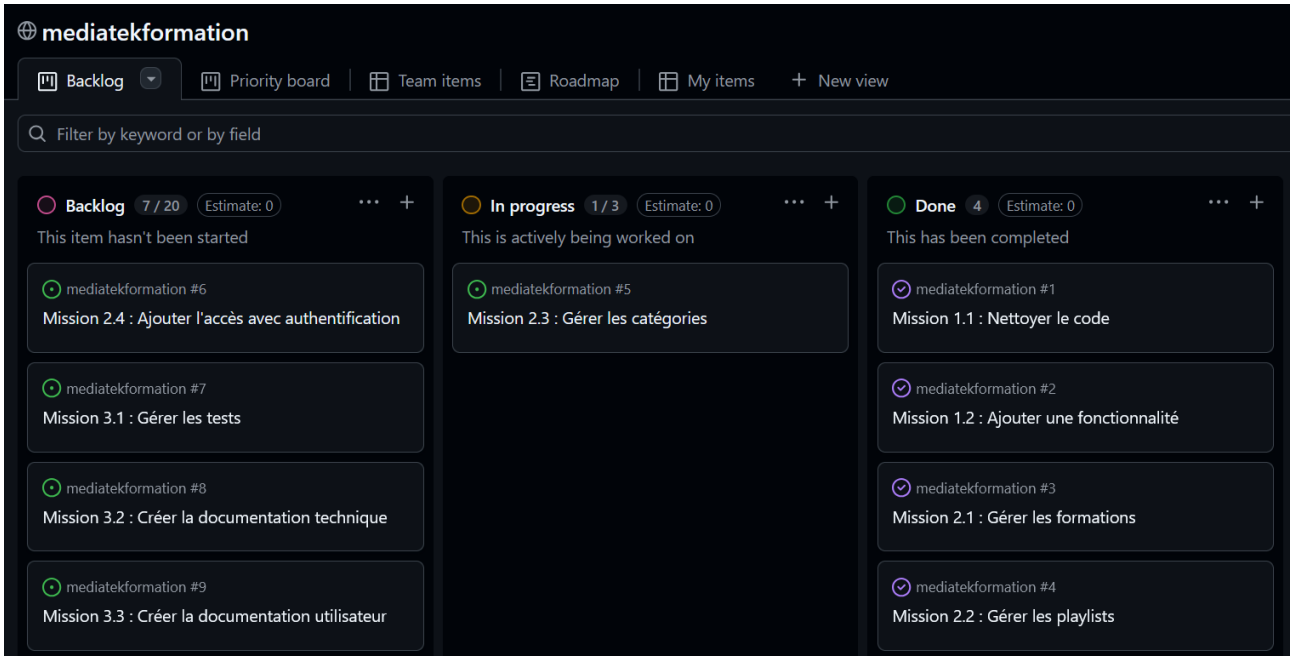
```php
#[Route('/admin/playlists/modifier/{id}', name: 'admin.playlists.modifier')]
public function modifier(int $id, Request $request): Response {
    $playlist = $this->playlistRepository->find($id);

    // Si le formulaire est bien soumis
    if($request->get("name")){
        $playlist->setName($request->get('name'));
        $playlist->setDescription($request->get('description'));
        $this->playlistRepository->add($playlist);
        return $this->redirectToRoute('admin.playlists');
    }

    // Sinon on affiche le formulaire prérempli avec la liste des formations
    $formations = $this->formationRepository->findAllForOnePlaylist($id);
    return $this->render(self::PAGE_ADMIN_PLAYLIST_FORM, [
        'playlist' => $playlist,
        'formations' => $formations,
        'estModification' => true
    ]);
}
```

```php
#[Route('/admin/playlists/supprimer/{id}', name: 'admin.playlists.supprimer')]
public function supprimer(int $id): Response {
    $playlist = $this->playlistRepository->find($id);
    // On vérifie qu'il n'y a pas de formations rattachées
    if($playlist->getFormations()->count() > 0){
        return $this->redirectToRoute('admin.playlists');
    }
    $this->playlistRepository->remove($playlist);
    return $this->redirectToRoute('admin.playlists');
}
```

Mission 2.3 : Gérer les catégories

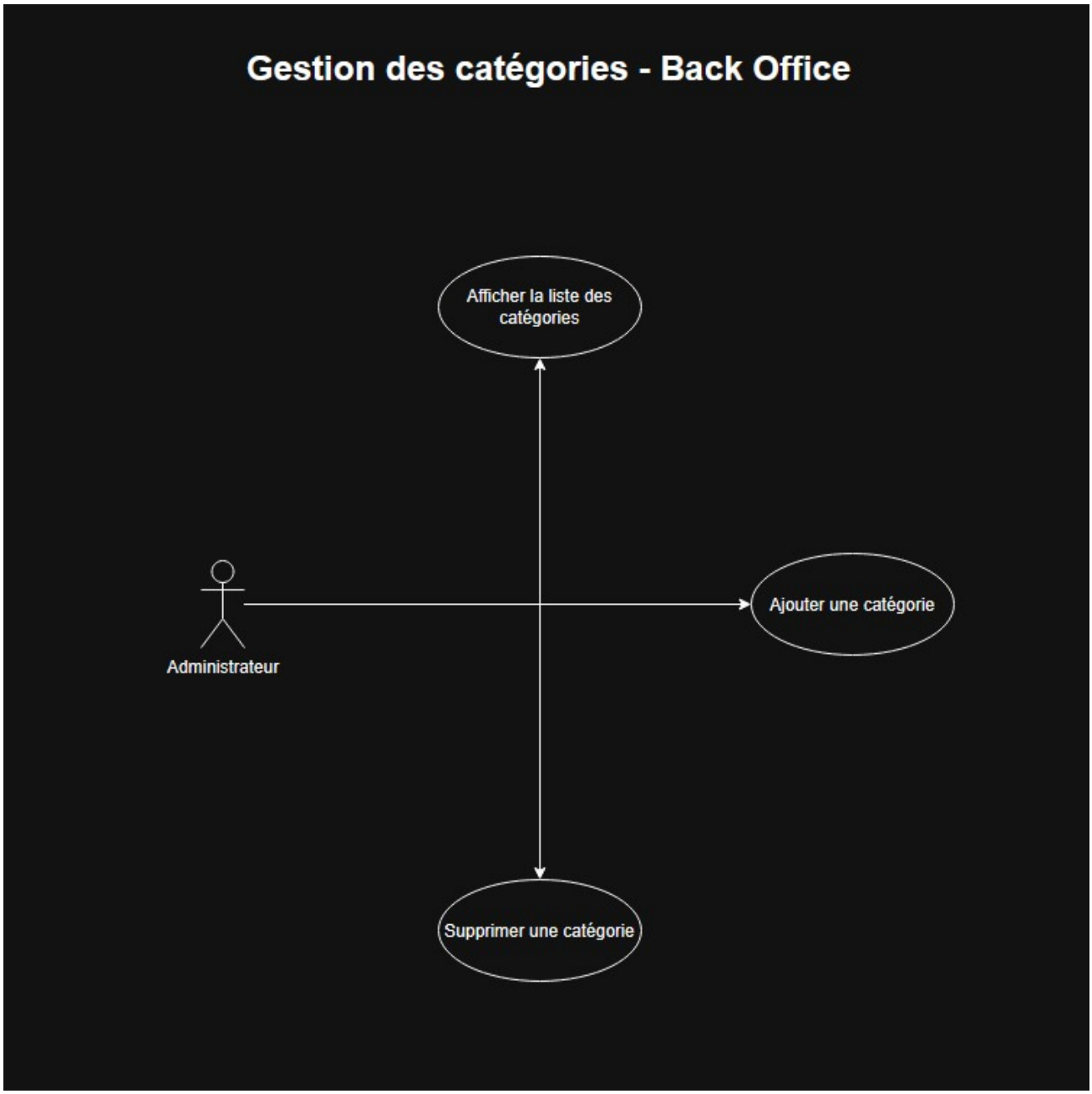
Demandes de la mission

- Une page doit permettre de lister les catégories et, pour chaque catégorie, afficher un bouton permettant de la supprimer. Attention, une catégorie ne peut être supprimée que si elle n'est rattachée à aucune formation.
- Dans la même page, un mini formulaire doit permettre de saisir et d'ajouter directement une nouvelle catégorie, à condition que le nom de la catégorie n'existe pas déjà.



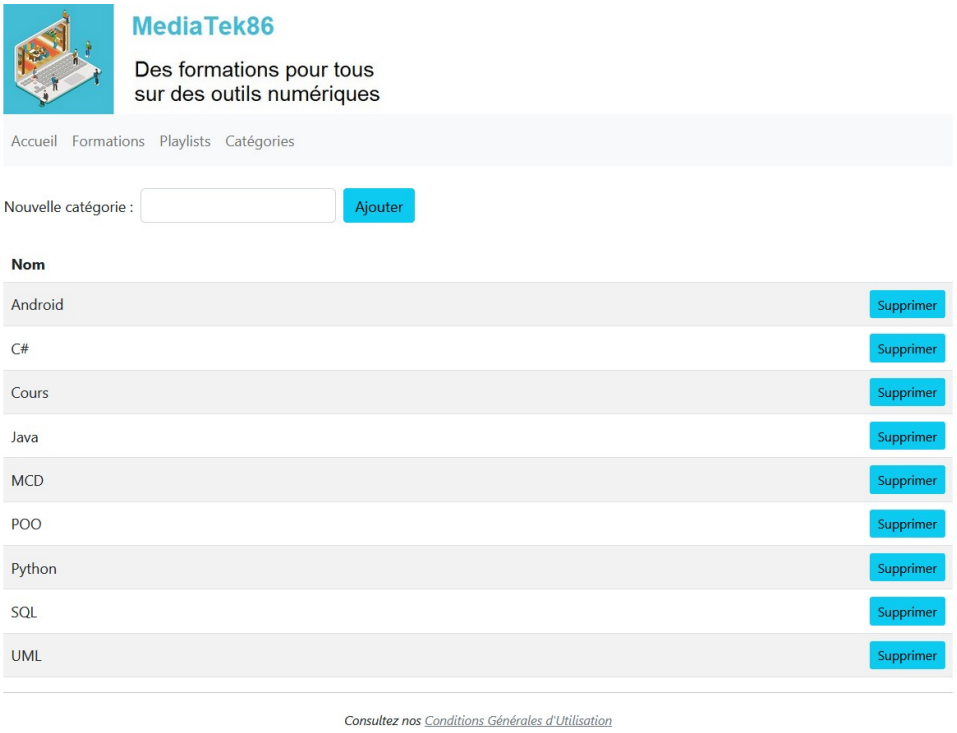
Temps de réalisation estimé : 3h
Temps de réalisation réel : 2h30

Diagramme des cas d’utilisation



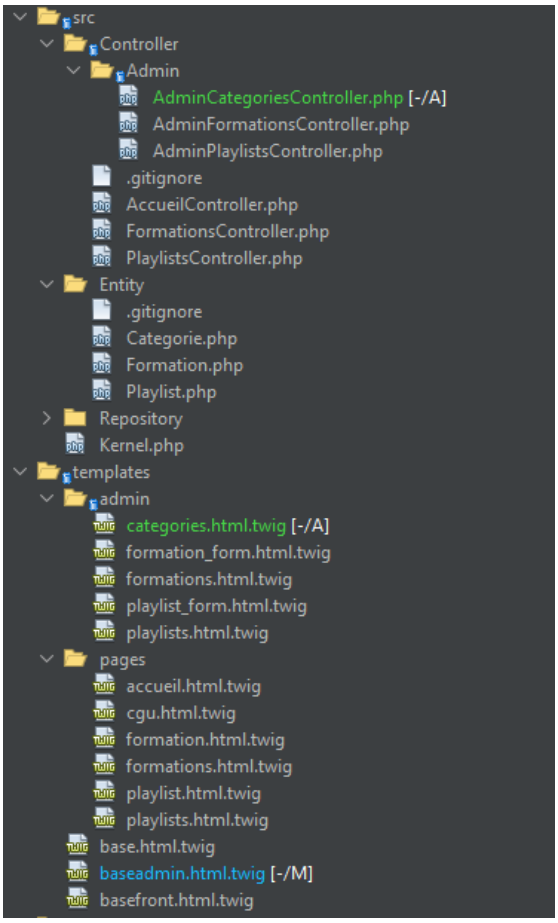
Nouvelle page

Une page d’admin catégories qui affiche les catégories, et permet d’ajouter et de supprimer une catégorie :



Explication de code

Voici l’arborescence du projet avec en vert les deux fichier qui ont été créés :



Le template baseadmin.html.twig est modifié pour ajouter l'option catégories dans le menu :

```
<!-- menu -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                <a class="nav-link" href="{{ path('accueil') }}">Accueil</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('admin.formations') }}">Formations</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('admin.playlists') }}">Playlists</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('admin.categories') }}">Catégories</a>
            </li>
        </ul>
    </div>
</nav>
```

Le template categories.html.twig dans le dossier admin qui comporte un formulaire et un tableau qui affiche les catégories :

```
{% extends "baseadmin.html.twig" %}

{% block body %}
    <!-- Formulaire d'ajout -->
    <br /><form method="POST" class="mb-4 d-flex gap-2 align-items-center">
        <label for="name" class="form-label mb-0">Nouvelle catégorie :</label>
        <input type="text" class="form-control w-auto" id="name" name="name" required>
        <button type="submit" class="btn btn-info">Ajouter</button>
    </form>

    <!-- Liste des catégories -->
    <table class="table table-striped">
        <thead>
            <tr>
                <th class="text-left align-top" scope="col">Nom</th>
                <th class="text-end align-top" scope="col"> </th>
            </tr>
        </thead>
        <tbody>
            {% if categories|length > 0 %}
                {% for k in 0..categories|length-1 %}
                    <tr class="align-middle">
                        <td>{{ categories[k].name }}</td>
                        <td class="text-end">
                            <a href="{{ path('admin.categories.supprimer', {id:categories[k].id}) }}"
                               class="btn btn-info btn-sm"
                               onclick="return confirm('Êtes-vous sûr de vouloir supprimer cette catégorie ?')">Supprimer</a>
                        </td>
                    </tr>
                {% endfor %}
            {% endif %}
        </tbody>
    </table>
{% endblock %}
```

Et le contrôleur AdminCategoriesController.php qui reprends à peu près la structure des autres contrôleurs, mais cette fois-ci comme le formulaire est dans la même page, on gère l'ajout dans index(), pas besoin de créer une méthode séparée. index() récupère les catégories, et gère aussi l'ajout d'une catégorie en vérifiant que le nom est bien rempli et qu'il n'existe pas déjà.

```php
#[Route('/admin/categories', name: 'admin.categories')]
public function index(Request $request): Response {
    // Si le formulaire d'ajout est bien soumis
    if($request->get("name")){
        $name = $request->get('name');
        // On vérifie que la catégorie n'existe pas déjà
        $existante = $this->categorieRepository->findOneBy(['name' => $name]);
        if(!$existante){
            $categorie = new Categorie();
            $categorie->setName($name);
            $this->categorieRepository->add($categorie);
        }
        return $this->redirectToRoute('admin.categories');
    }

    $categories = $this->categorieRepository->findBy([], ['name' => 'ASC']);
    return $this->render(self::PAGE_ADMIN_CATEGORIES, [
        'categories' => $categories
    ]);
}
```

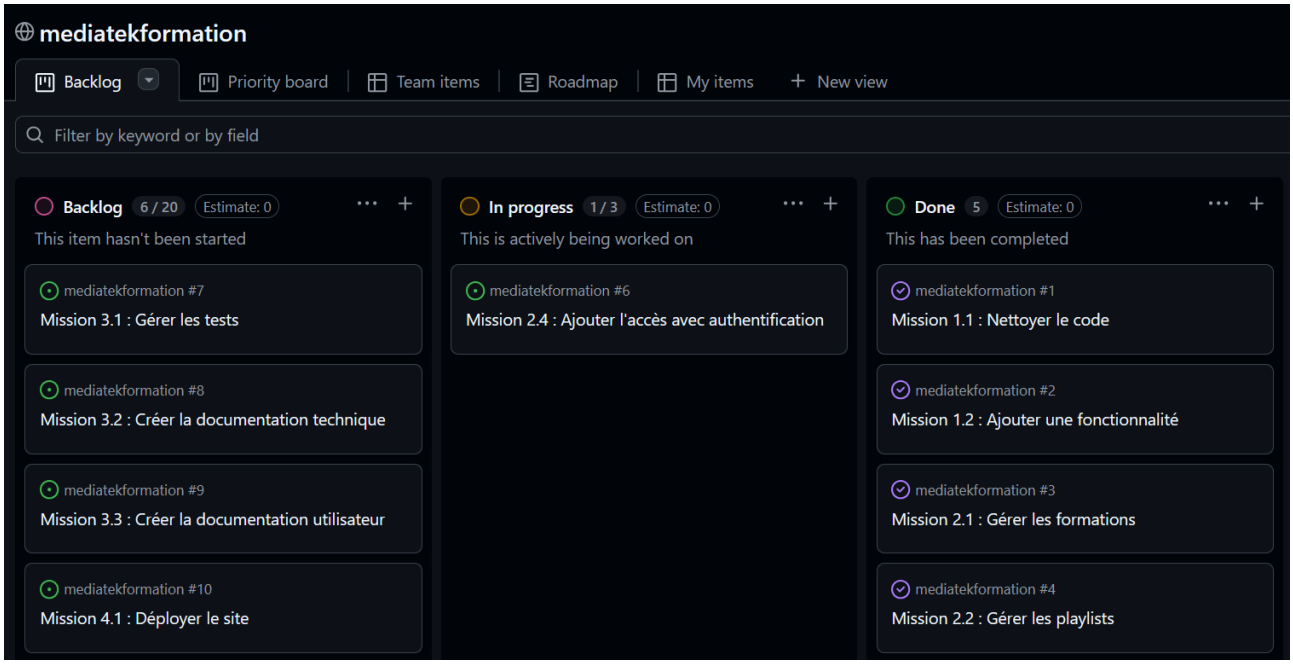
Et aussi la fonction supprimer qui gère la suppression d'une catégorie en vérifiant qu'elle n'est rattachée à aucune formation :

```php
#[Route('/admin/categories/supprimer/{id}', name: 'admin.categories.supprimer')]
public function supprimer(int $id): Response {
    $categorie = $this->categorieRepository->find($id);
    // On vérifie qu'aucune formation n'est rattachée
    if($categorie->getFormations()->count() > 0){
        return $this->redirectToRoute('admin.categories');
    }
    $this->categorieRepository->remove($categorie);
    return $this->redirectToRoute('admin.categories');
}
```

Mission 2.4 : Ajouter l’accès avec authentification

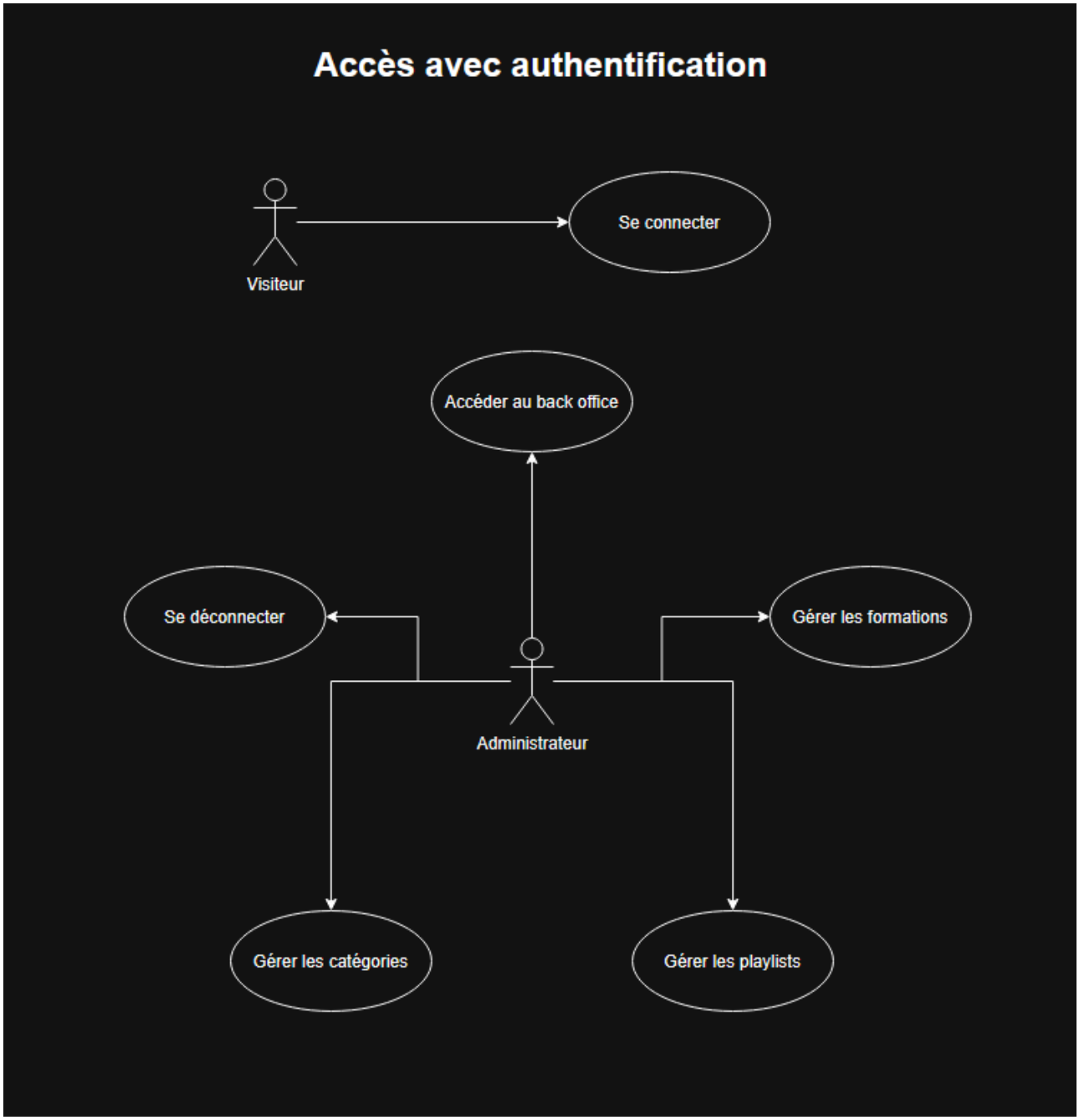
Demandes de la mission

- Le back office ne doit être accessible qu'après authentification : un seul profil administrateur doit avoir le droit d’accès.
- L'accès au back office doit se faire en ajoutant "/admin" à la fin de l'URL.
- Il doit être possible de se déconnecter, sur toutes les pages (avec un lien de déconnexion).



Temps de réalisation estimé : 4h
Temps de réalisation réel : 4h

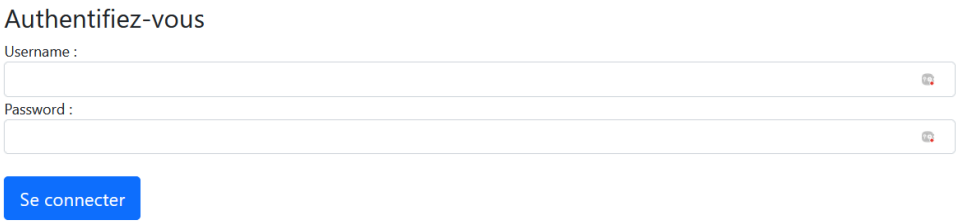
Diagramme de cas d’utilisation



Nouvelles pages

Nouvelle page :

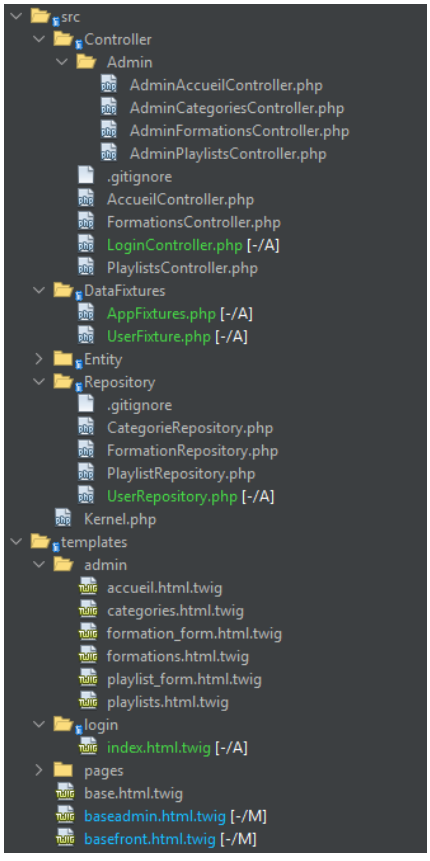
Une page de connexion quand on clique sur Administration :



Une page d'accueil admin, avec une option Déconnexion dans le menu présente sur toutes les pages si l'utilisateur est connecté :

Explication de code

Voici l'arborescence du projet avec en vert les fichier qui ont été créés :



Pour ajouter la couche d'authentification, j'ai suivi la méthode vue en cours.

Étape 1 : Création de l'entité User

La commande php bin/console make:user génère automatiquement deux fichiers :
- src/Entity/User.php qui est l'entité qui représente l'administrateur dans la base de données
- src/Repository/UserRepository.php qui est le repository associé

Étape 2 : Mise à jours de la base de données

Les commandes php bin/console make:migration et php bin/console doctrine:migrations:migrate créent automatiquement la table user dans la base de données.

Étape 3 : Création du compte admin

J'ai créé manuellement le fichier src/DataFixtures/AppFixtures.php pour que Symfony crée automatiquement un compte admin avec la commande suivante :
php bin/console doctrine:fixtures:load --append --group=UserFixture

Étape 4 : Création du formulaire de connexion

On crée le contrôleur qui gère l'authentification :
php bin/console make:controller Login

On configure la route d'authentification :

```yaml
form_login:
    # app_login est le nom de la route défini dans le contrôleur
    login_path: app_login
    check_path: app_login
```

On modifie la fonction index() du contrôleur :

```php
#[Route('/login', name: 'app_login')]
public function index(AuthenticationUtils $authenticationUtils): Response {
    // Récupération éventuelle de l'erreur
    $error = $authenticationUtils->getLastAuthenticationError();
    // Récupération du dernier login utilisé
    $lastUsername = $authenticationUtils->getLastUsername();
    return $this->render('login/index.html.twig', [
        'last_username' => $lastUsername,
        'error' => $error
    ]);
}
```

On modifie ensuite la vue du formulaire :

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

{% block body %}
    {% if error %}
        <div>{{ error.messageKey|trans(error.messageData, 'security') }}</div>
    {% endif %}

    <form action="{{ path('app_login') }}" method="post">
        <h3>Authentifiez-vous</h3>
        <label for="username">Username :</label>
        <input type="text" id="username" name="_username"
               value="{{ last_username }}"
               class="form-control" required autofocus />

        <label for="password">Password :</label>
        <input type="password" id="password" name="_password"
               class="form-control" required />

        <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">

        <br />
        <button class="btn btn-lg btn-primary" type="submit">Se connecter</button>
    </form>
{% endblock %}
```

On protège le formulaire en ajoutant enable_csrf : true dans security.yaml

Pour finir on gère la déconnexion en ajoutant ceci dans security.yaml :

```yaml
logout:
    path: logout
```

Et on ajoute l'option Déconnexion dans le menu de navigation, il n'apparaît que si l'utilisateur est connecté :

```twig
<!-- menu -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto">
            <li class="nav-item">
                <a class="nav-link" href="{{ path('accueil') }}">Accueil</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('formations') }}">Formations</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ path('playlists') }}">Playlists</a>
            </li>
        </ul>
        <ul class="navbar-nav ms-auto">
            <li class="nav-item">
                <a class="nav-link" href="{{ path('admin.accueil') }}">Administration</a>
            </li>
            {% if app.user %}
                <li class="nav-item">
                    <a class="nav-link text-danger" href="{{ path('logout') }}">Déconnexion</a>
                </li>
            {% endif %}
        </ul>
    </div>
</nav>
```

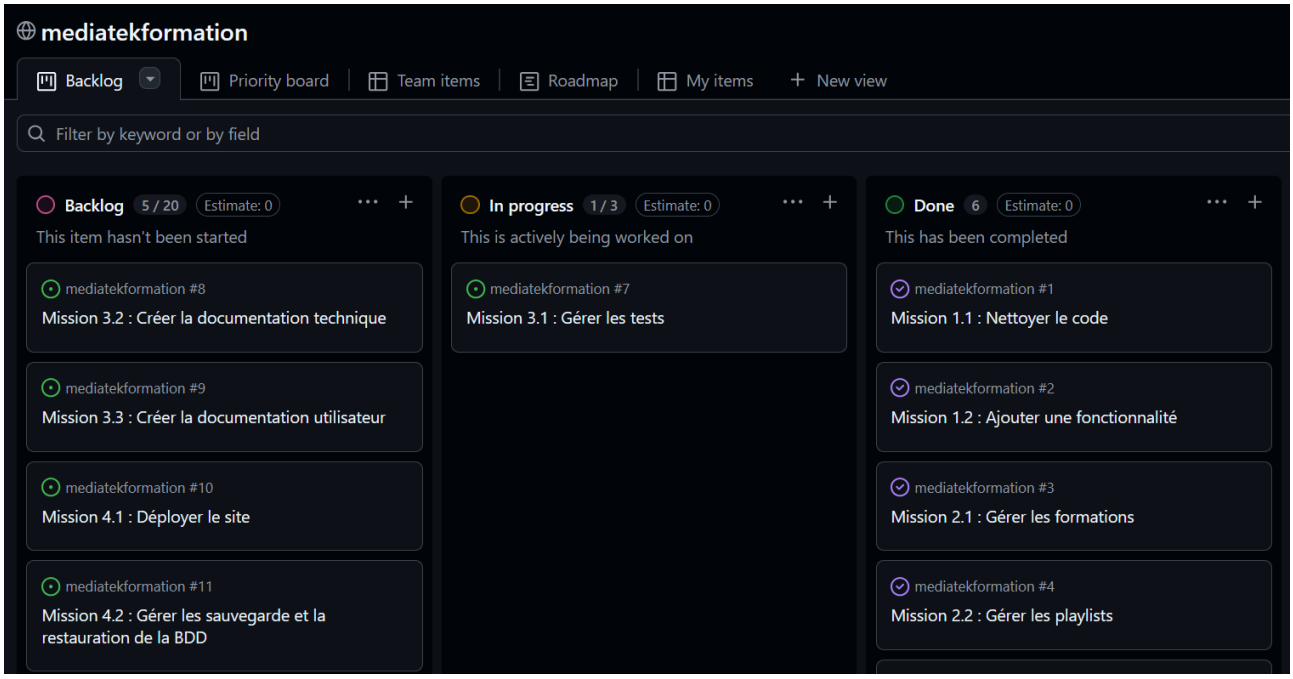
Mission 3.1 : Gérer les tests

Demandes de la mission

Tests unitaires :
Contrôler le fonctionnement de la méthode qui retourne la date de parution au format string.
Tests d'intégration sur les règles de validation :
Lors de l'ajout ou de la modification d'une formation, contrôler que la date n'est pas postérieure à aujourd'hui.
Tests d'intégration sur les Repository :
Contrôler toutes les méthodes ajoutées dans les classes Repository (pour cela, créer une BDD de test).
Tests fonctionnels :
Contrôler que la page d'accueil est accessible.
Dans chaque page contenant des listes :

- contrôler que les tris fonctionnent (en testant juste le résultat de la première ligne) ;
- contrôler que les filtres fonctionnent (en testant le nombre de lignes obtenu et le résultat de la première ligne) ;
- contrôler que le clic sur un lien (ou bouton) dans une liste permet d'accéder à la bonne page (en contrôlant l'accès à la page mais aussi le contenu d'un des éléments de la page).

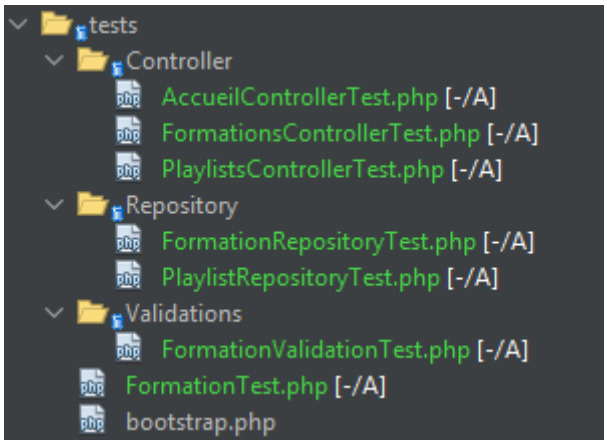
Tests de compatibilité :
Tester le site sur plusieurs navigateurs pour contrôler la compatibilité.



Temps de réalisation estimé : 7h
Temps de réalisation réel : 7h30

Explication de code

Voici les classes de tests créées en vert :



La classe FormationTest.php pour contrôler la méthode getPublishedAtString() :

```php
<?php
namespace App\Tests;

use App\Entity\Formation;
use PHPUnit\Framework\TestCase;

class FormationTest extends TestCase {

    public function testGetPublishedAtString() {
        $formation = new Formation();
        $formation->setPublishedAt(new \DateTime("2024-04-22"));
        $this->assertEquals("22/04/2024", $formation->getPublishedAtString());
    }
}
```

La classe FormationValidationTest.php pour tester le contrôle sur la date qui ne doit pas être postérieure à la date du jour.

```php
<?php
namespace App\Tests\Validations;

use App\Entity\Formation;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

class FormationValidationTest extends KernelTestCase {

    // Méthode qui vérifie le nombre d'erreurs
    public function assertErrors(Formation $formation, int $nbErreurs) {
        self::bootKernel();
        $validator = self::getContainer()->get('validator');
        $errors = $validator->validate($formation);
        $this->assertCount($nbErreurs, $errors);
    }

    // Test avec une date correcte (aujourd'hui) : aucune erreur attendue
    public function testDateValide() {
        $formation = new Formation();
        $formation->setPublishedAt(new \DateTime("now"));
        $this->assertErrors($formation, 0);
    }

    // Test avec une date future : une erreur attendue
    public function testDateNonValide() {
        $formation = new Formation();
        $formation->setPublishedAt(new \DateTime("2030-01-01"));
        $this->assertErrors($formation, 1);
    }
}
```

La classe PlaylistRepositoryTest.php pour tester les méthodes findAllOrderByName(), findAllOrderByNbFormations() et findByContainValue() :

```php
// Test findAllOrderByName : vérifie que le premier résultat est bien trié
public function testFindAllOrderByName() {
    $repository = $this->recupRepository();
    $playlists = $repository->findAllOrderByName('ASC');
    $this->assertEquals("Bases de la programmation (C#)", $playlists[0]->getName());
}

// Test findAllOrderByNbFormations : vérifie que le tri fonctionne
public function testFindAllOrderByNbFormations() {
    $repository = $this->recupRepository();
    $playlists = $repository->findAllOrderByNbFormations('ASC');
    $this->assertEquals(0, count($playlists[0]->getFormations()));
}

// Test findByContainValue : vérifie le nombre de résultats
public function testFindByContainValue() {
    $repository = $this->recupRepository();
    $playlists = $repository->findByContainValue('name', 'a', '');
    $this->assertGreaterThan(0, count($playlists));
}
```

La classe FormationRepositoryTest.php pour tester les méthodes findAllOrderBy(), findByContainValue(), findAllLasted() et findAllForOnePlaylist() :

```php
public function recupRepository(): FormationRepository {
    self::bootKernel();
    return self::getContainer()->get(FormationRepository::class);
}

// Test findAllOrderBy : vérifie que le tri sur le titre fonctionne
public function testFindAllOrderBy() {
    $repository = $this->recupRepository();
    $formations = $repository->findAllOrderBy('title', 'ASC');
    $this->assertEquals("Android Studio (complément n°1) : Navigation Drawer et Fragment", $formations[0]->getTitle());
}

// Test findByContainValue : vérifie le nombre de résultats
public function testFindByContainValue() {
    $repository = $this->recupRepository();
    $formations = $repository->findByContainValue('title', 'a', '');
    $this->assertGreaterThan(0, count($formations));
}

// Test findAllLasted : vérifie qu'on obtient bien le bon nombre de formations
public function testFindAllLasted() {
    $repository = $this->recupRepository();
    $formations = $repository->findAllLasted(3);
    $this->assertCount(3, $formations);
}

// Test findAllForOnePlaylist : vérifie qu'on obtient les formations d'une playlist
public function testFindAllForOnePlaylist() {
    $repository = $this->recupRepository();
    $formations = $repository->findAllForOnePlaylist(2);
    $this->assertGreaterThan(0, count($formations));
}
```

Pour les tests fonctionnels j'ai d'abord créé AccueilControllerTest.php pour tester l'accès à la page d'accueil :

```php
// Test que la page d'accueil est accessible
public function testAccesPageAccueil() {
    $client = static::createClient();
    $client->request('GET', '/');
    $this->assertResponseStatusCodeSame(200);
}
```

La classe FormationsControllerTest.php pour tester que la page est accessible, et que les tris, les filtres et les clics sur les liens fonctionnent :

```php
// Test que la page formations est accessible
public function testAccesPageFormations() {
    $client = static::createClient();
    $client->request('GET', '/formations');
    $this->assertResponseStatusCodeSame(200);
}

// Test le tri par titre ASC : vérifie le titre de la première ligne
public function testTriTitreAsc() {
    $client = static::createClient();
    $client->request('GET', '/formations/tri/title/ASC');
    $this->assertResponseStatusCodeSame(200);
    $this->assertSelectorTextContains('h5', 'Android Studio (complément n°1) : Navigation Drawer et Fragment');
}

// Test le tri par titre DESC : vérifie le titre de la première ligne
public function testTriTitreDesc() {
    $client = static::createClient();
    $client->request('GET', '/formations/tri/title/DESC');
    $this->assertResponseStatusCodeSame(200);
    $this->assertSelectorTextContains('h5', 'UML : Diagramme de paquetages');
}

// Test le filtre par titre : vérifie le nombre de résultats et la première ligne
public function testFiltreParTitre() {
    $client = static::createClient();
    $crawler = $client->request('GET', '/formations/recherche/title', ['recherche' => 'Eclipse']);
    $this->assertResponseStatusCodeSame(200);
    $nbFormations = $crawler->filter('h5')->count();
    $this->assertGreaterThan(0, $nbFormations);
    $this->assertSelectorTextContains('h5', 'Eclipse');
}

// Test le clic sur une formation : vérifie l'accès à la page détail
public function testClicFormation() {
    $client = static::createClient();
    $crawler = $client->request('GET', '/formations');
    $link = $crawler->filter('td a')->first()->link();
    $client->click($link);
    $this->assertResponseStatusCodeSame(200);
}
```

La classe PlaylistsControllerTest.php pour tester que la page est accessible, et que les tris, les filtres et les clics sur les liens fonctionnent :

```php
// Test que la page playlists est accessible
public function testAccesPagePlaylists() {
    $client = static::createClient();
    $client->request('GET', '/playlists');
    $this->assertResponseStatusCodeSame(200);
}

// Test le tri par nom ASC : vérifie le nom de la première ligne
public function testTriNomAsc() {
    $client = static::createClient();
    $client->request('GET', '/playlists/tri/name/ASC');
    $this->assertResponseStatusCodeSame(200);
    $this->assertSelectorTextContains('h5', '(C#)');
}

// Test le tri par nom DESC : vérifie le nom de la première ligne
public function testTriNomDesc() {
    $client = static::createClient();
    $client->request('GET', '/playlists/tri/name/DESC');
    $this->assertResponseStatusCodeSame(200);
    $this->assertSelectorTextContains('h5', '2019');
}

// Test le filtre par nom : vérifie le nombre de résultats et la première ligne
public function testFiltreParNom() {
    $client = static::createClient();
    $crawler = $client->request('GET', '/playlists/recherche/name', ['recherche' => 'Android']);
    $this->assertResponseStatusCodeSame(200);
    $nbPlaylists = $crawler->filter('h5')->count();
    $this->assertGreaterThan(0, $nbPlaylists);
    $this->assertSelectorTextContains('h5', 'Android');
}

// Test le clic sur une playlist : vérifie l'accès à la page détail
public function testClicPlaylist() {
    $client = static::createClient();
    $crawler = $client->request('GET', '/playlists');
    $client->clickLink('Voir détail');
    $this->assertResponseStatusCodeSame(200);
}
```

Plan de tests

Contexte : MediaTek86
Situation professionnelle : Symfony
Application : mediatekformation (site de mise à disposition des auto-formations).

Plan de tests

Tests unitaires

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler la méthode getPublishedAtString() de la classe Formation pour voir si elle retourne la bonne date au bon format.	Test unitaire lancé avec la date : 2024-04-22	22/04/2024	OK

Tests d'intégration

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler que la date d'une formation est valide (non postérieure à aujourd'hui)	Test d'intégration lancé avec la date : DateTime("now") Test d'intégration lancé avec la date : 2030-01-01	La date est inférieure ou égale à aujourd'hui La date est supérieure à aujourd'hui	OK OK
FormationRepository : findAllOrderBy() vérifier que le tri sur le titre fonctionne	Test d'intégration lancé avec le paramètre « ASC »	La première formation retournée est bien "Android Studio (complément n°1) : Navigation Drawer et Fragment"	OK
FormationRepository : findByContainValue() vérifier le nombre de résultats	Test d'intégration lancé avec les paramètres 'title', 'a', ''	Retourne bien plusieurs formations	OK
FormationRepository : findAllLasted() vérifier qu'on obtient bien le bon nombre de formations	Test d'intégration lancé avec le paramètre 3	Retourne bien les 3 formations les plus récentes	OK
FormationRepository : findAllForOnePlaylist() vérifier qu'on obtient les formations d'une playlist	Test d'intégration lancé avec le paramètre 2	Retourne bien les formations de la playlist d'id 2	OK
PlaylistRepository : findAllOrderByName() vérifier que le premier résultat est bien trié	Test d'intégration lancé avec le paramètre « ASC »	La première playlist retournée est bien "Bases de la programmation (C#)"	OK
PlaylistRepository : findAllOrderByNbFormations() vérifier que le tri fonctionne	Test d'intégration lancé avec le paramètre « ASC »	La première playlist retournée a bien 0 formations	OK
PlaylistRepository : findByContainValue() vérifier le nombre de résultats	Test d'intégration lancé avec les paramètres 'name', 'a', ''	Retourne bien plusieurs formations	OK

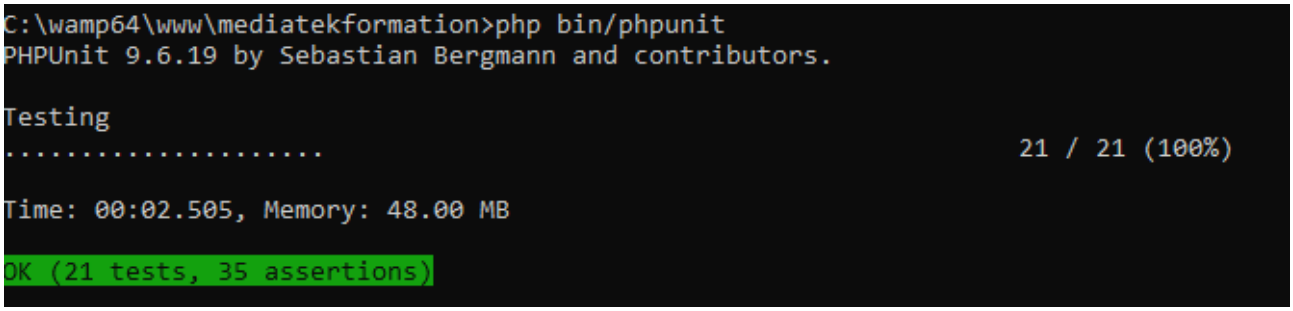
Tests fonctionnels

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler que la page d'accueil est accessible	Requête GET sur /	La page répond	OK
Contrôler que la page formations est accessible	Requête GET sur /formations	La page répond	OK
Contrôler le tri par titre ASC	Requête GET sur /formations/tri/title/ASC	La première formation est « Android Studio (complément n°1) : Navigation Drawer et Fragment »	OK
Contrôler le tri par titre DESC	Requête GET sur /formations/tri/title/DESC	La première formation est « UML : Diagramme de paquetages »	OK
Contrôler le filtre par titre	Requête POST sur /formations/recherche/title avec la valeur « Eclipse »	9 résultats, la première formation est « Eclipse n°8 : Déploiement »	OK
Contrôler le clic sur une formation	Clic sur le premier lien de la liste	La page de détail est accessible	OK
Contrôler que la page playlists est accessible	Requête GET sur /playlists	La page répond	OK
Contrôler le tri par nom ASC	Requête GET sur /playlists/tri/name/ASC	La première playlist est « Bases de la programmation (C#) »	OK
Contrôler le tri par nom DESC	Requête GET sur /playlists/tri/name/DESC	La première playlist est « Visual Studio 2019 et C# »	OK
Contrôler le filtre par nom	Requête POST sur /playlist/recherche/name avec la valeur « Android »	2 résultats, la première playlist est « Compléments Android (programmation mobile) »	OK
Contrôler le clic sur une playlist	Clic sur le premier lien de la liste	La page de détail est accessible	OK

Tests de compatibilité

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler la compatibilité avec Firefox	Navigation sur toutes les pages de l'application avec Firefox	Toutes les pages s'affichent correctement et toutes les fonctionnalités sont opérationnelles	OK
Contrôler la compatibilité avec Chrome	Navigation sur toutes les pages de l'application avec Chrome	Toutes les pages s'affichent correctement et toutes les fonctionnalités sont opérationnelles	OK

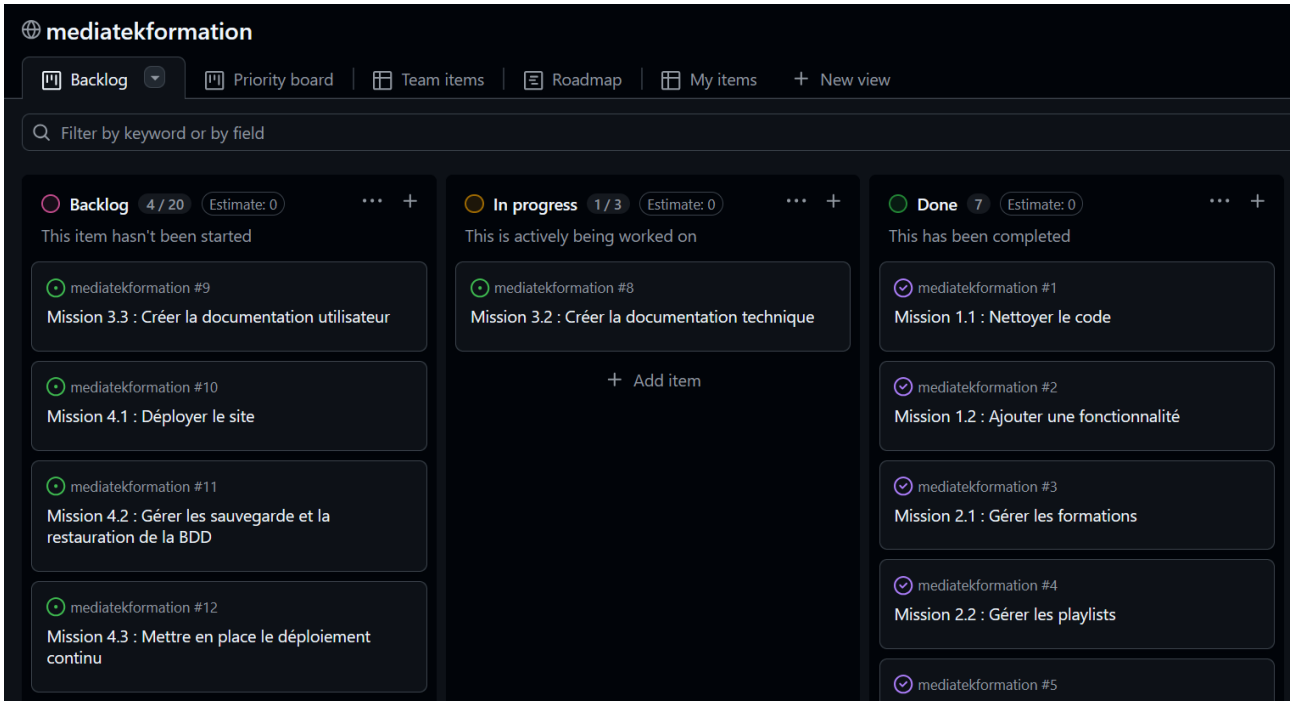
Et voici la preuve que tous les tests réussissent :



Mission 3.2 : Créer la documentation technique

Demandes de la mission

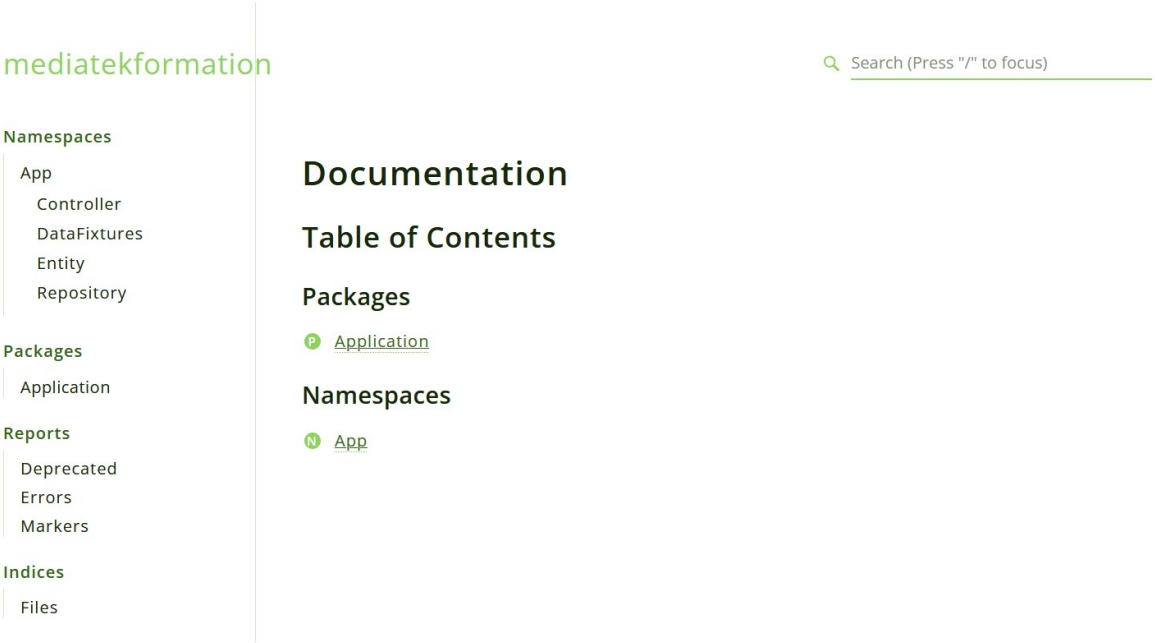
- Contrôler que tous les commentaires normalisés nécessaires à la génération de la documentation technique ont été correctement insérés.
- Générer la documentation technique du site complet : front et back office excluant le code automatiquement généré par Symfony (voir l'article "Génération de la documentation technique sous NetBeans" dans le wiki du dépôt).



Temps de réalisation estimé : 1h
Temps de réalisation réel : 1h45

Aperçu

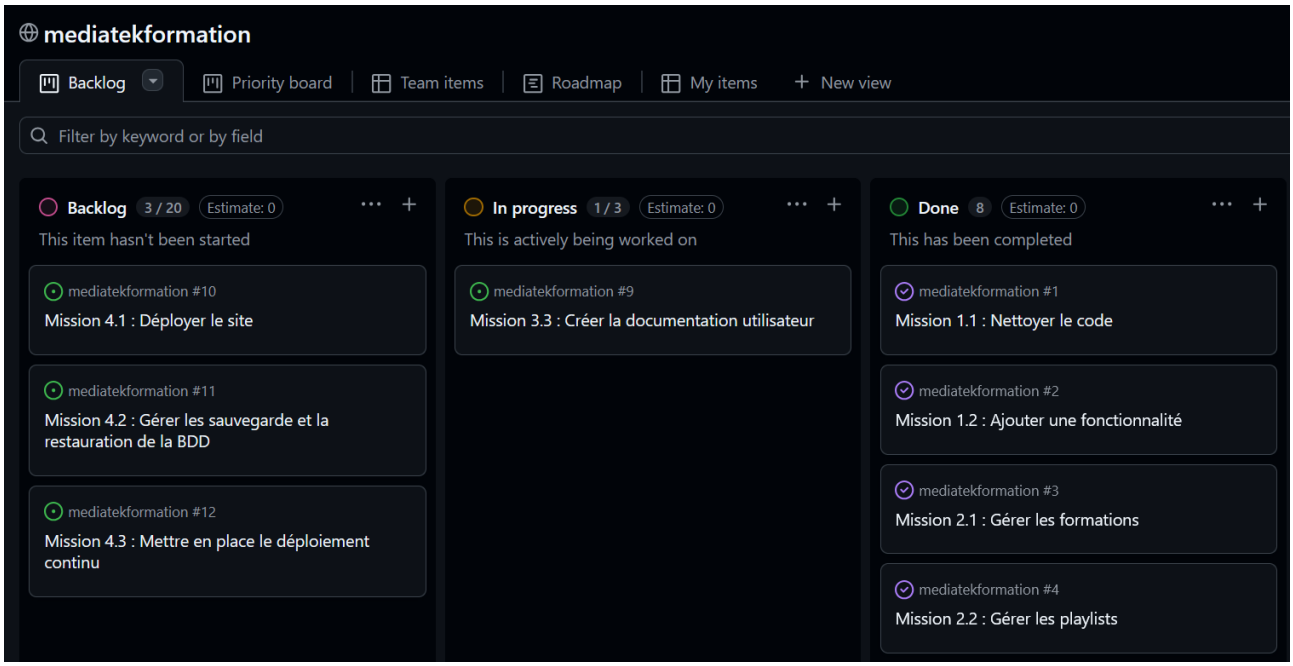
Première page de la documentation technique :



Mission 3.3 : Créer la documentation utilisateur

Demandes de la mission

Créer en vidéo qui permet de montrer toutes les fonctionnalités du site (front et back office).
Cette vidéo ne doit pas dépasser les 5mn et doit présenter clairement toutes les fonctionnalités, en montrant les manipulations qui doivent être accompagnées d'explications orales.



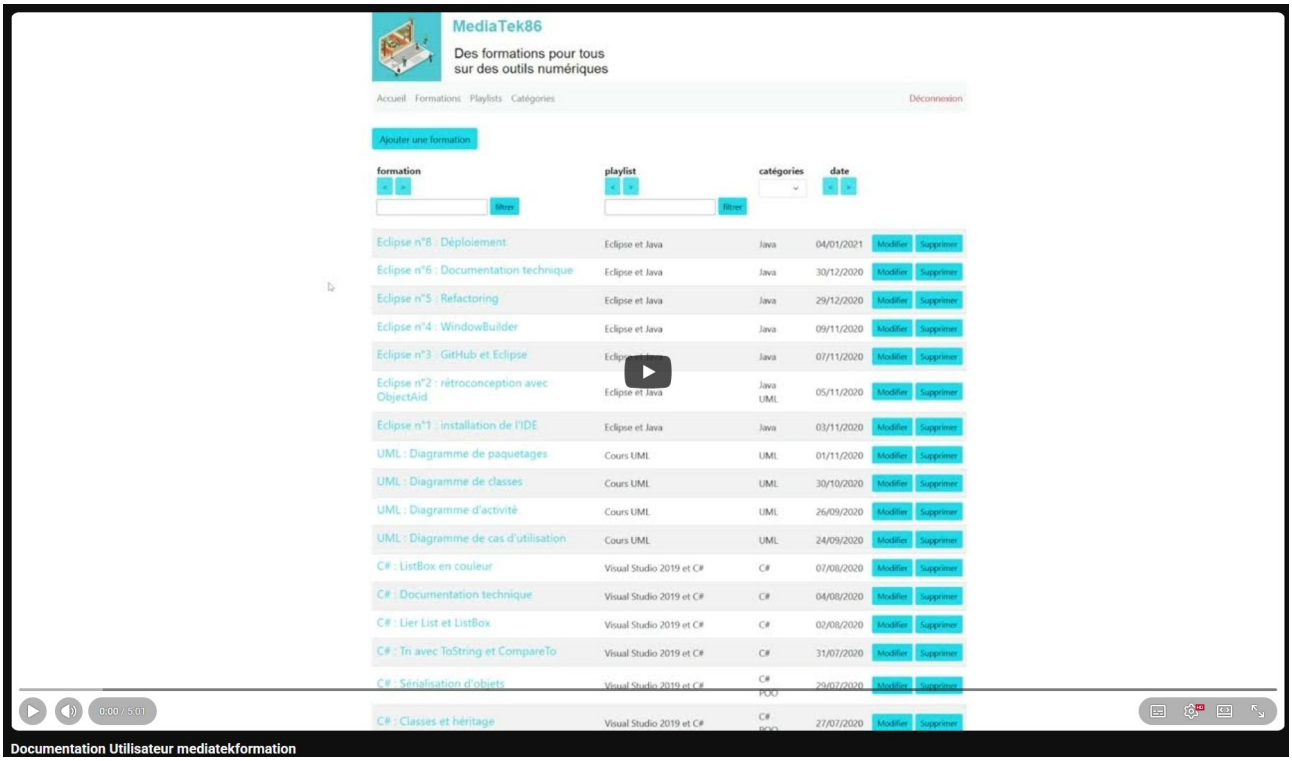
Temps de réalisation estimé : 2h
Temps de réalisation réel : 1h

Vidéo

La documentation utilisateur est accessible ici :

Documentation Utilisateur MediatekFormation

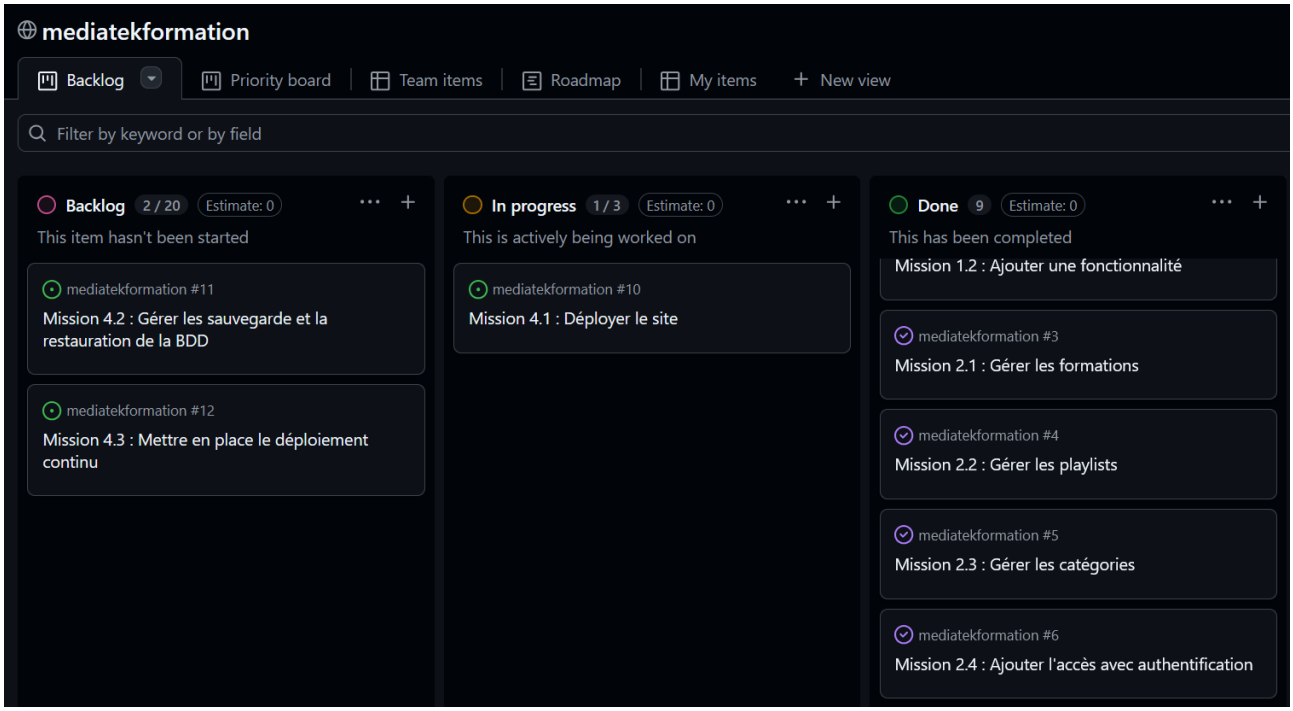
Capture d’écran de la vidéo :



Mission 4.1 : Déployer le site

Demandes de la mission

- Déployer le site, la BDD et la documentation technique chez un hébergeur.
- Mettre à jour la page de CGU avec la bonne adresse du site.



Temps de réalisation estimé : 2h
Temps de réalisation réel : 2h30

Procédé du déploiement

J’ai choisit d’utiliser OVHcloud pour héberger le site car c’est déjà là que mon portfolio est hébergé. J’ai donc créé un sous-domaine mediatekformation.nathan-boudier.com et créé la base de données.

J’ai fait une copie du fichier .env en modifiant la variable DATABASE_URL avec les informations de connexion à la base de données OVH.

J’ai importé la base de données dans le phpMyAdmin d’OVH.

J’ai ensuite transféré le projet sur le serveur d’OVH via FileZilla.

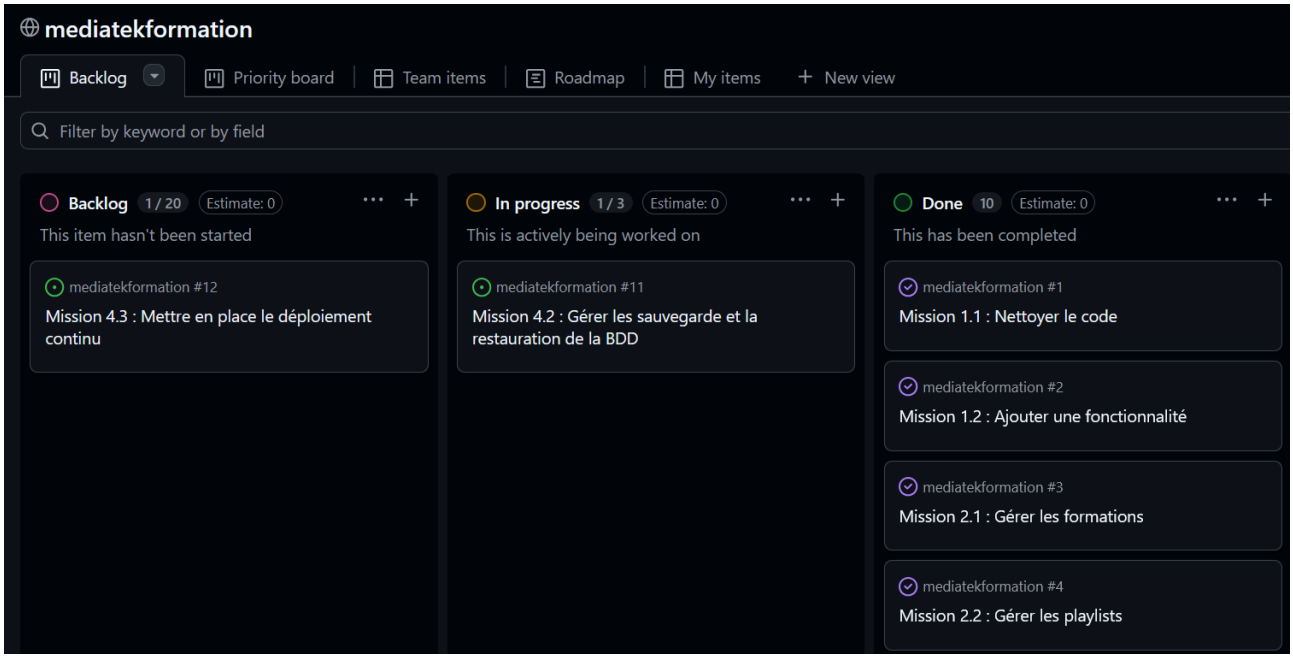
J’ai ajouté la doc dans un dossier doc dans le dossier public du projet via FileZilla.
URL de la doc : https://mediatekformation.nathan-boudier.com/doc/index.html

Enfin j’ai mis à jours la page de CGU avec la bonne adresse et les informations de l’hébergeur.

Mission 4.2 : Gérer mes sauvegarde et la restauration de la BDD

Demandes de la mission

- Une sauvegarde journalière automatisée doit être programmée pour la BDD (voir l'article "Automatiser la sauvegarde d'une BDD" dans le wiki du dépôt).
- La restauration pourra se faire manuellement, en exécutant le script de sauvegarde.



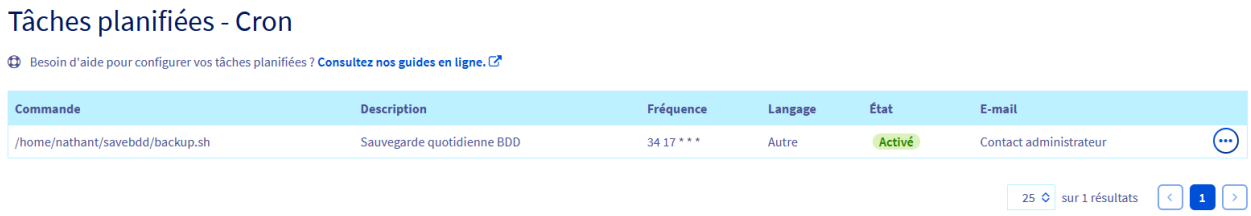
Temps de réalisation estimé : 1h
Temps de réalisation réel : 1h

Tâche Cron

J’ai créé un script backup.sh dans le dossier savebdd à la racine du compte OVH. Ce script permet de sauvegarder la BDD dans un fichier compressé nommé avec la date du jour :

```
#!/bin/sh
DATE=`date -I`
find /home/nathant/savebdd/bdd* -mtime -1 -exec rm {} \;
mysqldump -u nathantmediatek -pTON_MOT_DE_PASSE --databases
nathantmediatek --single-transaction | gzip >
/home/nathant/savebdd/bddbackup_${DATE}.sql.gz
```

J’ai ensuite configuré une tâche CRON dans l’espace OVH pour exécuter ce script une fois par jour.



Cependant j’ai constaté que cette tâche ne fonctionnait pas, sûrement à cause de la commande mysqldump qui semble ne pas être accessible dans OVH, ce qui empêche l’exécution du script.

Il y a une alternative dans OVH qui propose des sauvegardes automatiques de la base de données, qui sont consultables dans l’espace client.

Mode opératoire pour la restauration :

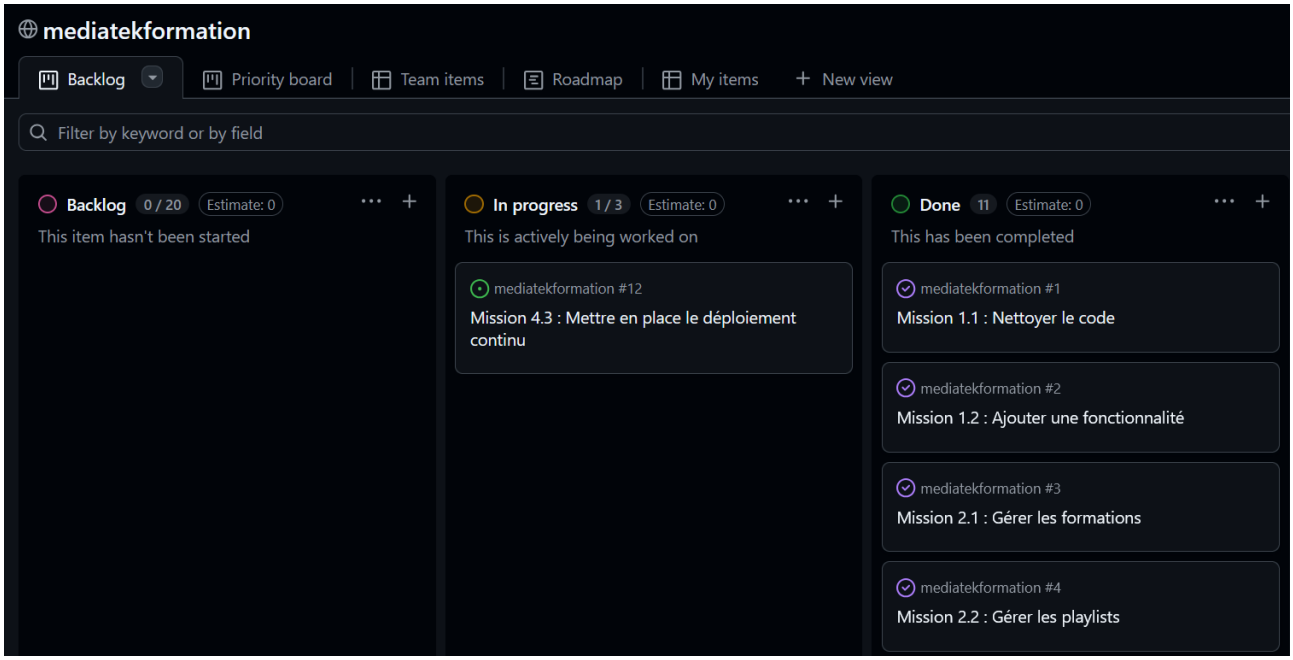
Pour restaurer la BDD, il faudrait d’abord récupérer le fichier de sauvegarde de la BDD dans le dossier savebdd via FileZilla. Il faudrait ensuite dézipper ce fichier pour obtenir le fichier .sql, puis se connecter au phpMyAdmin OVH et supprimer toutes les tables. Enfin il n’y aurait plus qu’a importer le fichier .sql récupéré pour restaurer la BDD.

Avec les sauvegardes automatiques OVH, il faut cliquer sur les 3 points à droite de la BDD, choisir l’option Restaurer une sauvegarde, choisir la date de sauvegarde souhaitée, et confirmer la restauration.

Mission 4.3 : Mettre en place le déploiement continu

Demandes de la mission

Configurer le dépôt Github pour que le site en ligne soit mis à jour à chaque push reçu dans le dépôt.



Temps de réalisation estimé : 1h
Temps de réalisation réel : 1h

Démarche

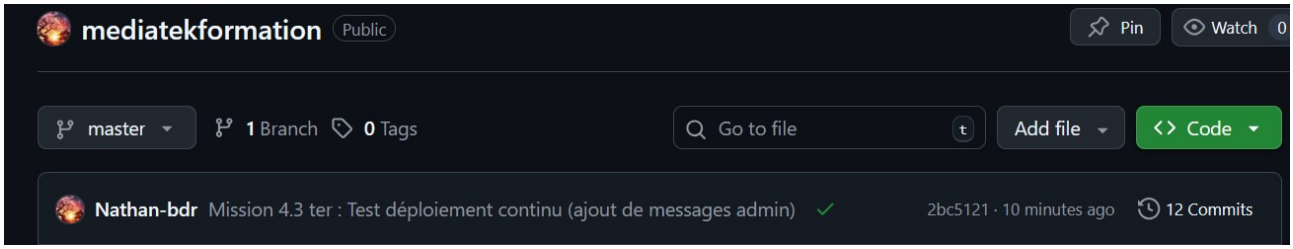
Il faut commencer par ajouter .env au début du fichier .gitignore.
Il faut ensuite taper la commande suivante dans la console Git Bash :
git rm --cached .env

Sur GitHub, il faut ensuite créer un fichier yml qui contient le code suivant :

```yaml
on: push
name: Deploy website on push
jobs:
  web-deploy:
    name: Deploy
    runs-on: ubuntu-latest
    steps:
      - name: Get latest code
        uses: actions/checkout@v2

      - name: Sync files
        uses: SamKirkland/FTP-Deploy-Action@4.3.0
        with:
          server: ftp.cluster100.hosting.ovh.net
          server-dir: /mediatekformation/
          username: nathant
          password: ${{ secrets.ftp_password }}
```

Enfin, on enregistre le mot de passe d'accès au FTP dans GitHub, dans les settings.

J'ai effectué un push pour tester en ajoutant des messages dans la partie admin, on voit que le script a bien été exécuté :



Bilan final

Objectifs atteints

L'ensemble des missions confiées ont été réalisées avec succès.

La mission 1 a permis de nettoyer le code existant en suivant les recommandations de SonarLint, puis d'ajouter la fonctionnalité manquante : l'affichage du nombre de formations par playlist avec la possibilité de trier sur cette colonne.

La mission 2 a consisté à développer un back office complet permettant à un administrateur authentifié de gérer le contenu de la base de données. Il est désormais possible d'ajouter, modifier et supprimer des formations, des playlists et des catégories, avec des contrôles de saisie et des messages d'information à chaque action. L'accès au back office est sécurisé par un formulaire d'authentification avec un profil administrateur unique, et un lien de déconnexion est disponible sur toutes les pages.

La mission 3 a permis de mettre en place une suite de tests complète : un test unitaire sur la méthode de formatage de date, des tests d'intégration sur les règles de validation et les méthodes des Repository, et des tests fonctionnels sur les pages de liste avec vérification des tris, des filtres et des liens. La documentation technique a été générée avec phpDocumentor et est accessible en ligne. Une vidéo de présentation de 5 minutes a également été réalisée pour documenter l'ensemble des fonctionnalités.

La mission 4 a permis de déployer le site, la base de données et la documentation technique sur un hébergement mutualisé OVHcloud. Une sauvegarde journalière automatisée de la base de données a été mise en place via une tâche Cron. Un déploiement continu a été configuré via GitHub Actions, permettant de mettre à jour automatiquement le site en ligne à chaque push vers le dépôt GitHub.

Problèmes rencontrés

Plusieurs difficultés ont été rencontrées au cours du développement.

Lors de la mise en place de la contrainte de validation sur la date de publication, l'utilisation de LessThanOrEqual("today") provoquait une erreur car Symfony compare la date avec l'heure exacte de minuit. Il a fallu utiliser LessThanOrEqual("tomorrow") pour obtenir le comportement attendu.

Lors du déploiement sur OVHcloud, le fichier .htaccess nécessaire à l'URL rewriting de Symfony était absent du projet. Il a fallu le générer avec la commande composer require symfony/apache-pack. De plus, le dossier var/cache devait être vidé sur le serveur après le transfert des fichiers pour éviter des erreurs de timeout.

Pour l'hébergement de la documentation technique, il n'était pas possible d'accéder directement au dossier doc car Symfony interceptait toutes les URLs. Il a fallu ajouter une exception dans le fichier .htaccess pour permettre l'accès direct à ce dossier.