Compte rendu du projet : MediatekFormation

Contexte :

Développement d'un site web avec une partie Front-end contenant des formations sur le domaine de l'informatique, et la partie Back-end permettant de gérer le site et ces formations.
Le site web : http://mediatekformation.go.yn.fr/

Langages et technologies utilisées :

- PHP / HTML / CSS / Twig / SQL

- Symfony (Pour gérer la liaison a la BDD)

- BootStrap 5 (Permet de styliser le site)

- Github (Déploiement continue et versioning)

- MySQL (Base de donnée)

- MyPlanetHoster (Hébergement du site)

Existant : Avant le début du projet, le site de MediatekFormation était incomplet, il manquer toute la partie Back-End, dans la partie Front-end il manquer certaines fonctionnalités comme les tris et les filtres, la documentation technique et la documentation utilisateur .

Sommaire

Mission 1

Tâche 1.1 : Nettoyer le code selon les recommandations de SonarLint

Description : Nettoyer le code en suivant les indications de SonarLint, sauf pour le code généré automatiquement par Symfony.

- **Kanban - État d'avancement :**

Estimation et temps réel

- Temps estimé : **2h00**
- Temps réel : **2h30**

Problèmes corrigés

1. **phpS115** : Les constantes doivent être en majuscules.
    - *Avant correction :*

```
private const cheminImage = "https://i.ytimg.com/vi/";
```

    - **Correction :** Les constantes ont été mises en majuscules, et les appels ont été mis à jour.
    - *Après correction :*

```
private const CHEMINIMAGE = "https://i.ytimg.com/vi/";
```

2. **phpS1192** : Répétitions dans l'appel des pages.
    - *Avant correction :*

```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
    ]);
}

#[Route('/formations/tri/{champ}/{ordre}/{table}', name: 'formations.sort')]
public function sort($champ, $ordre, $table=""): Response(
    $formations = $this->formationRepository->findAllOrderBy($champ, $ordre, $table);
    $categories = $this->categorieRepository->findAll();
    return $this->render("pages/formations.html.twig", [
        'formations' => $formations,
        'categories' => $categories
    ]);
}

#[Route('/formations/recherche/{champ}/{table}', name: 'formations.findallcontain')]
public function findAllContain($champ, Request $request, $table=""): Response(
    $valeur = $request->get("recherche");
    $formations = $this->formationRepository->findByContainValue($champ, $valeur, $table);
    $categories = $this->categorieRepository->findAll();
    return $this->render("pages/formations.html.twig", [
        'formations' => $formations,
        'categories' => $categories,
        'valeur' => $valeur,
        'table' => $table
    ]);
```

- **Correction :** Utilisation d'une variable `$lien` pour stocker les noms des fichiers `.twig`.
- *Après correction :*

```php
public $lien = "pages/formations.html.twig";

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

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

#[Route('/formations/tri/{champ}/{ordre}/{table}', name: 'formations.sort')]
public function sort($champ, $ordre, $table = ""): Response
{
    $formations = $this->formationRepository->findAllOrderBy($champ, $ordre, $table);
    $categories = $this->categorieRepository->findAll();
    return $this->render($this->lien, [
        'formations' => $formations,
        'categories' => $categories
    ]);
}

#[Route('/formations/recherche/{champ}/{table}', name: 'formations.findallcontain')]
public function findAllContain($champ, Request $request, $table = ""): Response
{
    $valeur = $request->get("recherche");
    $formations = $this->formationRepository->findByContainValue($champ, $valeur, $table);
    $categories = $this->categorieRepository->findAll();
    return $this->render($this->lien, [
        'formations' => $formations,
        'categories' => $categories,
        'valeur' => $valeur,
        'table' => $table
    ]);
}
```

3. **phpS121** : Les boucles `foreach` doivent utiliser des accolades {}.
    - *Avant correction :*

```php
foreach($categoriesFormation as $categorieFormation)
if(!$categories->contains($categorieFormation->getName())){
    $categories[] = $categorieFormation->getName();
}
```

- **Correction :** Ajout des accolades {} autour des blocs de code.

- *Après correction :*

```
foreach ($categoriesFormation as $categorieFormation) {
    if (!$categories->contains($categorieFormation->getName())) {
        $categories[] = $categorieFormation->getName();
    }
}
```

4. **phpS131** : Les instructions `switch` doivent inclure une clause `default`.

- *Avant correction :*

```
switch($champ){
    case "name":
        $playlists = $this->playlistRepository->findAllOrderByName($ordre);
        break;
}
```

- **Correction :** Le `switch` a été remplacé par un ensemble de `if`.
- *Après correction :*

```
if ($champ == "name") {
    $playlists = $this->playlistRepository->findAllOrderByName($ordre);
}
```

5. **phpS1066** : Les `if` imbriqués doivent être fusionnés.

- *Avant correction :*

```
if ($this->formations->removeElement($formation)) {
    // set the owning side to null (unless already changed)
    if ($formation->getPlaylist() === $this) {
        $formation->setPlaylist(null);
    }
}
```

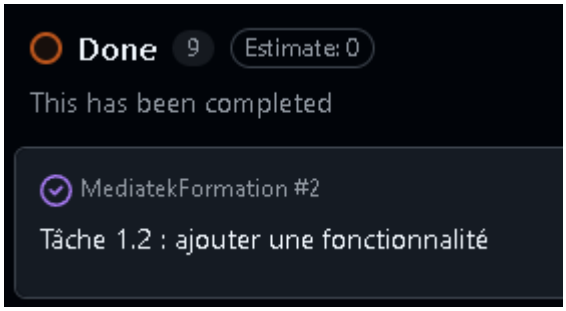
- **Correction :** Fusion des conditions dans un seul `if`.
- *Après correction :*

```php
if ($this->formations->removeElement($formation) && $formation->getPlaylist() === $this) {
    // set the owning side to null (unless already changed)
    $formation->setPlaylist(null);
}
```

Tâche 1.2 : Affichage et tri du nombre de formations par playlist

Description : Ajouter une colonne affichant le nombre de formations par playlist et permettre le tri croissant/décroissant.

- **Kanban - État d'avancement :**



Estimation et temps réel

- Temps estimé : **2h00**
- Temps réel : **1h10**

Changements dans le code

1. **Cette migration ajoute une colonne** `nbrdeformation` **de type entier dans la table** `playlist` **pour stocker le nombre de formations associées à chaque playlist :**

migrations/Version20241022123825.php

```
@@ -0,0 +1,31 @@
+ <?php
+
+ declare(strict_types=1);
+
+ namespace DoctrineMigrations;
+
+ use Doctrine\DBAL\Schema\Schema;
+ use Doctrine\Migrations\AbstractMigration;
+
+ /**
+  * Auto-generated Migration: Please modify to your needs!
+  */
+ final class Version20241022123825 extends AbstractMigration
+ {
+     public function getDescription(): string
+     {
+         return '';
+     }
+
+     public function up(Schema $schema): void
+     {
+         // this up() migration is auto-generated, please modify it to your needs
+         $this->addSql('ALTER TABLE playlist ADD nbrdeformation INT NOT NULL');
+     }
+
+     public function down(Schema $schema): void
+     {
+         // this down() migration is auto-generated, please modify it to your needs
+         $this->addSql('ALTER TABLE playlist DROP nbrdeformation');
+     }
+ }
```

2. **Ajout de la possibilité de trier les playlists par le nombre de formations (`nbrdeformation`) :**

   Code ajouté dans la méthode `sort()` de `PlaylistsController` :

src/Controller/PlaylistsController.php

```
@@ -75,6 +75,9 @@ public function sort($champ, $ordre): Response
75  75          if ($champ == "name") {
76  76              $playlists = $this->playlistRepository->findAllOrderByName($ordre);
77  77          }
    78  +       if ($champ == "nbrdeformation") {
    79  +           $playlists = $this->playlistRepository->findAllOrderByNbrFormation($ordre);
    80  +       }
78  81          $categories = $this->categorieRepository->findAll();
79  82          return $this->render($this->lien, [
80  83              'playlists' => $playlists,
```

3. **Ajout de la méthode `findAllOrderByNbrFormation()` pour permettre le tri des playlists par le nombre de formations associées. :**

src/Repository/PlaylistRepository.php

```
@@ -44,6 +44,16 @@ public function findAllOrderByName($ordre): array
44  44                  ->getResult();
45  45          }
46  46
47  +       public function findAllOrderByNbrFormation($ordre): array
48  +       {
49  +           return $this->createQueryBuilder('p')
50  +               ->leftjoin('p.formations', 'f')
51  +               ->groupBy('p.id')
52  +               ->orderBy('p.nbrdeformation', $ordre)
53  +               ->getQuery()
54  +               ->getResult();
55  +       }
56  +
```

4. **Mise à jour des templates :**

`playlists.html.twig` :

- Afficher une colonne supplémentaire dans la liste des playlists.
- Fournir une fonctionnalité de tri basée sur la colonne `nbrdeformation`. .

```
<th class="align-top" scope="col">
    Nombre<br>de formation
    <br>
    <a href="{{ path('playlists.sort', {champ:'nbrdeformation', ordre:'ASC'}) }}"
        class="btn btn-info btn-sm active" role="button" aria-pressed="true">▲ </a>
    <a href="{{ path('playlists.sort', {champ:'nbrdeformation', ordre:'DESC'}) }}"
        class="btn btn-info btn-sm active" role="button" aria-pressed="true">▼</a>
</th>
```

```
<td>
    {{ playlists[k].nbrdeformation }}
</td>
```

`playlist.html.twig` :

- Affiche le nombre de formations associé à une playlist dans la page de détail d'une playlist.
- Utilise la variable `playlist.nbrdeformation` pour afficher dynamiquement le contenu.

templates/pages/playlist.html.twig

```
@@ -12,6 +12,9 @@
12  12          <br /><br />
13  13          <strong>description :</strong><br />
14  14          {{ playlist.description|nl2br }}
    15  +       <br /><br />
    16  +       <strong>Nombre de formation :</strong><br />
    17  +       {{ playlist.nbrdeformation}}
```

Mission 2

Tâche 2.1 : Gestion CRUD des formations

Description : (Back-office)Ajouter des boutons pour créer, éditer et supprimer des formations. Créer les fichiers . `twig` et routes nécessaires.

- **Kanban - État d'avancement :**

Estimation et temps réel

- Temps estimé : **5h00**
- Temps réel : **4h30**

Changements dans le code

1. **Arborescence des fichiers :**

src/

├── Controller/

│ ├── Admin/

│ │ ├── AdminFormationsController.php (Gère les CRUD des formations pour l'admin)

├── Form/

│ ├── FormationsFormType.php (Formulaire pour les opérations d'ajout/édition)

templates/

├── admin/

│ ├── admin.formation.add.html.twig (Page d'ajout)

│ ├── admin.formation.edit.html.twig (Page d'édition)

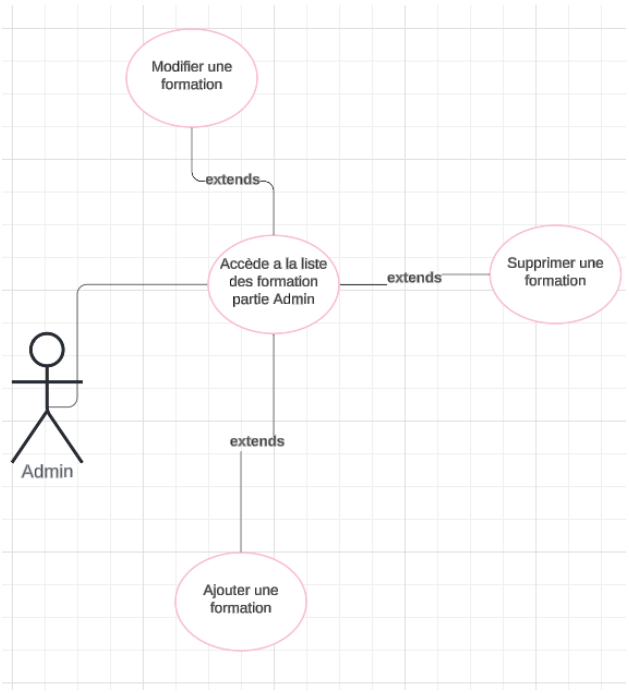
│ ├── admin.formations.html.twig (Page principale pour l'admin)

templates/

├── baseadmin.html.twig (Ajout d'un lien pour accéder aux formations dans le menu)

2. **Maquettage et diagrammes :**

- *Diagramme cas d'utilisation :*



Acteurs :

- Administrateur.
- **Cas d'utilisation :**
- Ajouter une formation.
- Modifier une formation.
- Supprimer une formation.

- *Maquettage :*

Page de gestion des formations (admin) :

- Présente la liste des formations avec les boutons "Ajouter", "Modifier", "Supprimer".

Page du formulaire d'ajout d'une formation :

- Affiche un formulaire avec les champs nécessaires pour ajouter une nouvelle formation.

Tâche 2.2 : Gestion CRUD des playlists

Description :
Créer une gestion CRUD des playlists dans le back-office, comprenant :

- Un bouton pour **ajouter** une playlist via un formulaire (avec validation des saisies : `name` obligatoire, `description` facultative).
- Un bouton pour **modifier** une playlist avec un formulaire pré-rempli (affiche également la liste des formations associées, sans modification possible).
- Un bouton pour **supprimer** une playlist, avec une confirmation (seulement si elle n'a pas de formations associées).
- Les mêmes tris et filtres que dans le front-office doivent être présents.

Kanban - État d'avancement :

Estimation et temps réel

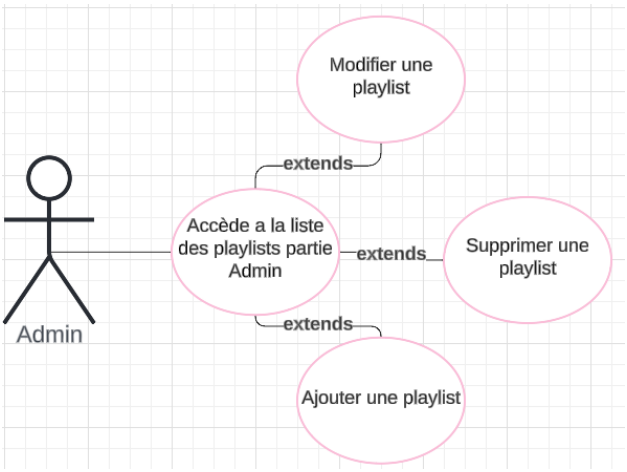
- Temps estimé : **5h00**
- Temps réel : **3h00**

Changements dans le code

1. Arborescence des fichiers :

src/
├── Controller/
│ ├── Admin/
│ │ ├── AdminPlaylistsController.php (Gère les CRUD des Playlists pour l'admin)
├── Form/
│ ├── PlaylistsType.php (Formulaire pour les opérations d'ajout/édition)
templates/
├── admin/
│ ├── admin.Playlists.add.html.twig (Page d'ajout)
│ ├── admin.Playlists.edit.html.twig (Page d'édition)
│ ├── admin.Playlists.html.twig (Page principale pour l'admin)
templates/
├── baseadmin.html.twig (Ajout d'un lien pour accéder aux playlists dans le menu)

Maquettage et diagrammes :



- **Diagramme cas d'utilisation :**

Acteurs :

• Administrateur.

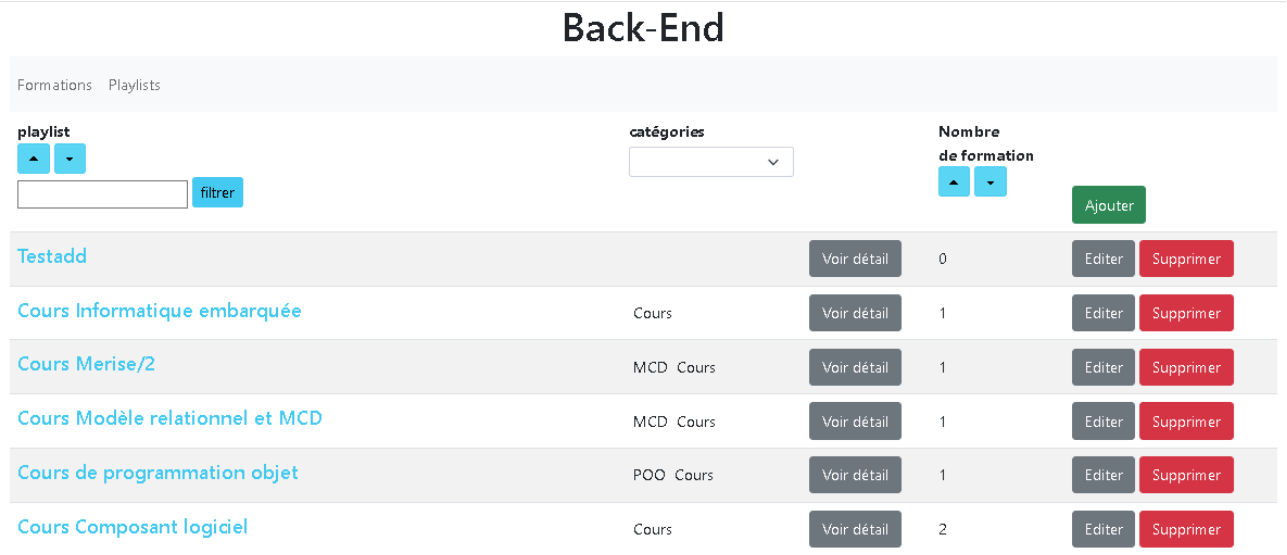
Cas d'utilisation :

• Ajouter une playlist.
• Modifier une playlist.
• Supprimer une playlist.

2. Maquettage :

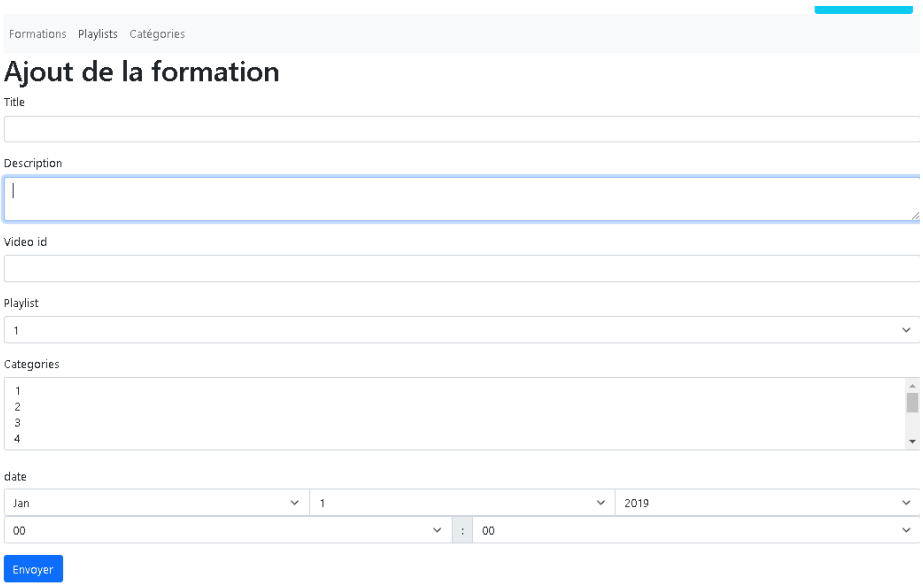
Page de gestion des playlists (admin) :

- Présente la liste des playlists avec les boutons "Ajouter", "Modifier", et "Supprimer".



Page du formulaire d'ajout d'une playlist :

- Affiche un formulaire avec les champs nécessaires pour ajouter une nouvelle playlist.



Tâche 2.3 : Gérer les catégories

Description :
Créer une gestion CRUD simplifiée pour les catégories dans le back-office comprenant :

- Une page pour lister les catégories existantes.
- Un bouton permettant de supprimer une catégorie (uniquement si aucune formation n'y est rattachée).
- Un mini formulaire pour ajouter une nouvelle catégorie directement dans la page (avec validation pour éviter les doublons).

Kanban - État d'avancement :

Estimation et temps réel :

- **Temps estimé :** 3h
- **Temps réel :** 3h

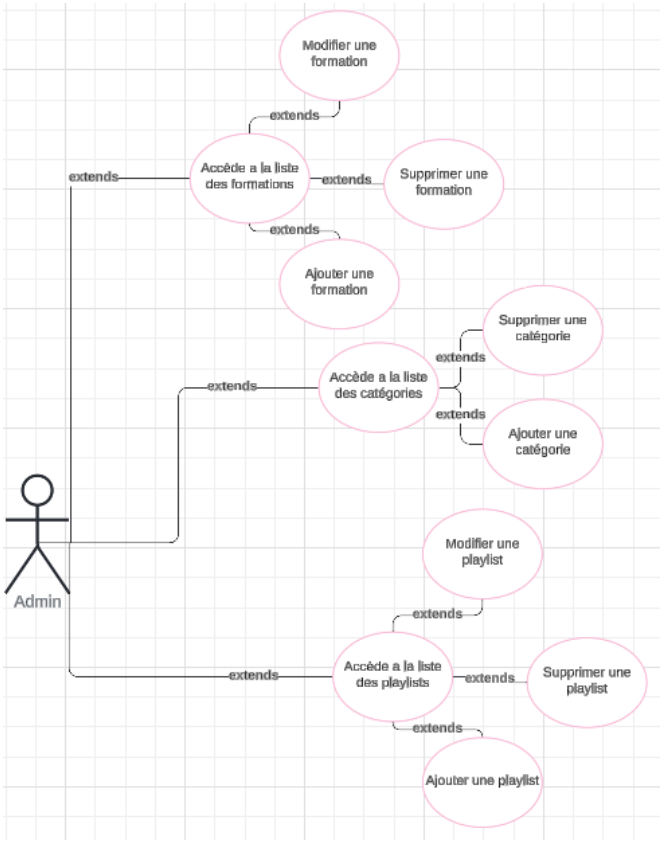
Changements dans le code

1. **Arborescence des fichiers :**

src/
├── Controller/
│ ├── Admin/
│ ├── AdminCategoriesController.php (Gère les actions liées aux catégories dans le back-office)
templates/
├── baseadmin.html.twig (Ajout d'un lien dans le menu pour accéder à la gestion des catégories)
├── admin/
├── admin.categories.html.twig (Page principale pour lister, ajouter et supprimer les catégories)

Maquettage et diagrammes :

Cas
•
•



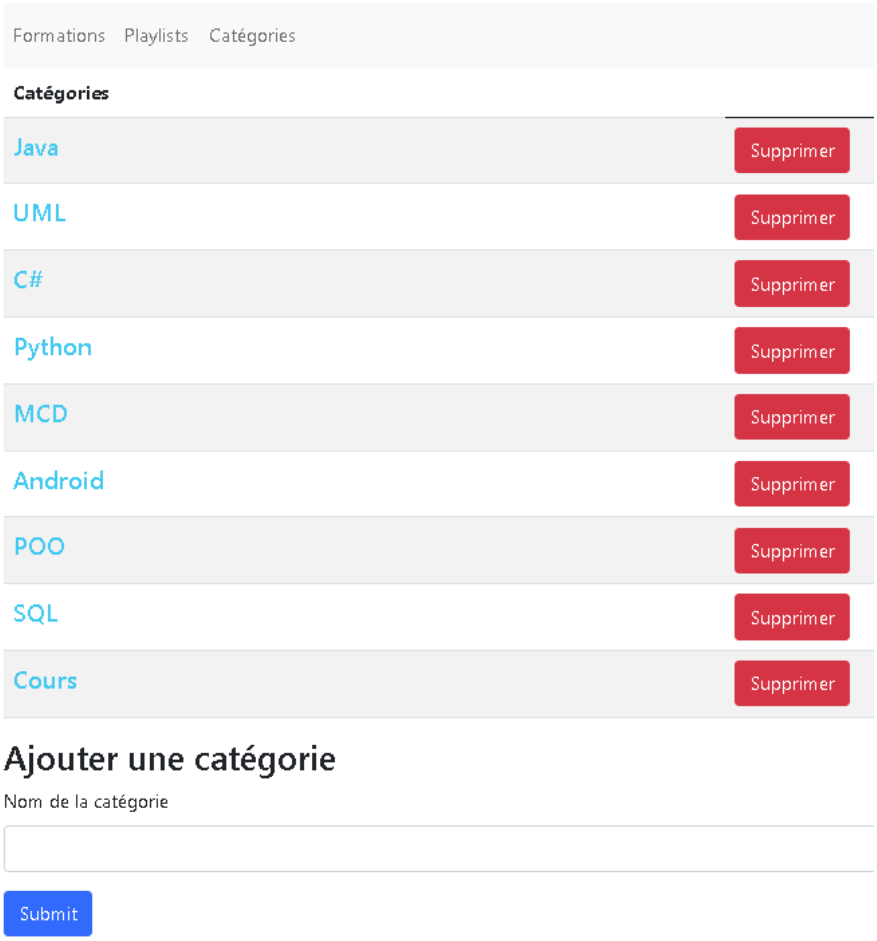
• **Diagramme cas d'utilisation :**
Acteurs :
• Administrateur.

d'utilisation :
Ajouter une catégorie.
Supprimer une catégorie.

2. Maquettage :

Page de gestion des catégories (admin) :

- **La liste des catégories avec un bouton "Supprimer" pour chaque catégorie**
- **Un formulaire simple en bas de page pour ajouter une nouvelle catégorie**

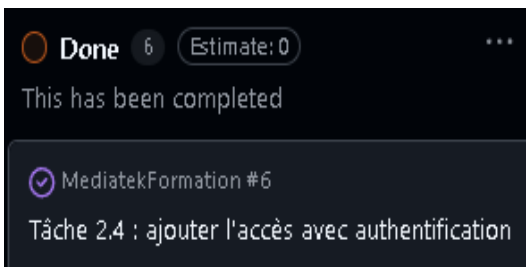


Tâche 2.4 : Ajouter l’accès avec authentification

Description :

- Le back-office ne doit être accessible qu'après authentification.
- Un seul profil administrateur doit avoir accès.
- Un lien "Se déconnecter" doit être présent sur toutes les pages pour permettre la déconnexion.

Kanban - État d'avancement :



Estimation et temps réel :

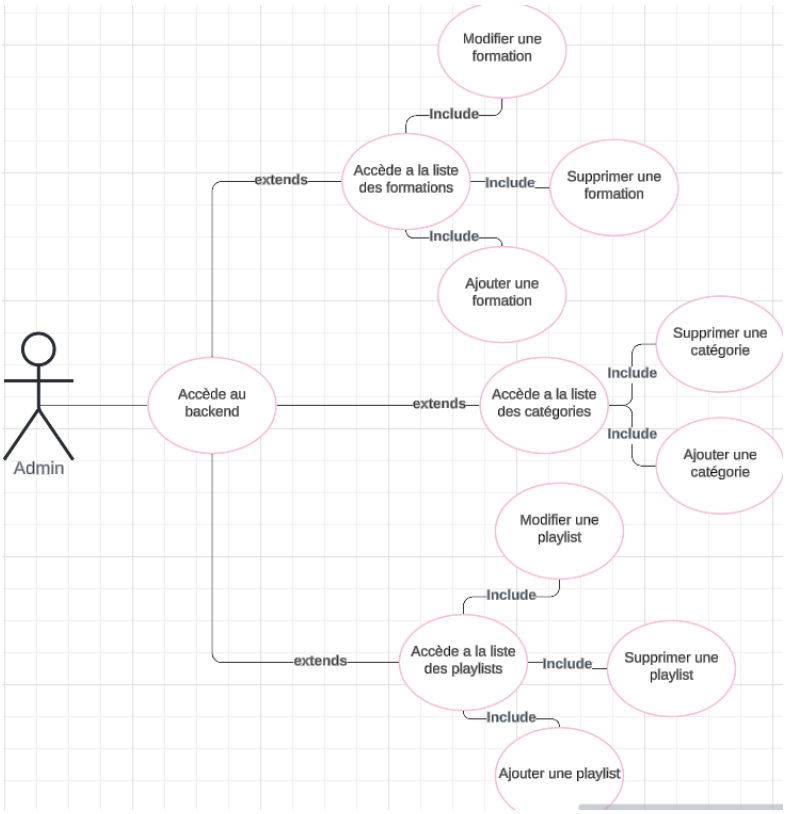
- **Temps estimé :** 4h
- **Temps réel :** 4h

Changements dans le code

2. **Arborescence des fichiers :**

src/
├── Controller/
│ ├── LoginController.php (Gère les routes de connexion et déconnexion)
templates/
├── baseadmin.html.twig (Ajout du lien "Se déconnecter" dans le back-office)
├── login/
│ ├── login.html.twig (Page de connexion avec formulaire)

Maquettage et diagrammes :



Acteurs :

- Administrateur.

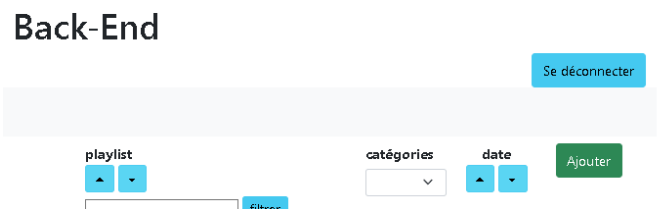
Cas d'utilisation :

- Se connecter au back-office.
- Accéder au contenu après authentification.
- Se déconnecter du back-office.

2. Maquettage :

Fenêtre de connexion :

- Affiche un formulaire simple avec les champs `Username` et `Password`.
- En cas d'erreur, un message est affiché.



Mission 3

Tâche 3.1 : Gérer les tests

Description : Réalisation de tests pour vérifier le bon fonctionnement de l'application selon plusieurs catégories de tests :

- **Tests unitaires :** Contrôler le format de la date de parution d'une formation.
- **Tests de validation :** Contrôler que la date de publication d'une formation n'est pas postérieure à aujourd'hui.
- **Tests d'intégration :** Vérifier les méthodes ajoutées dans les Repository via une base de données de test.
- **Tests fonctionnels :** Vérifier l'accessibilité des pages, le bon fonctionnement des tris, filtres, et liens de navigation dans les listes.
- **Tests de compatibilité :** Utiliser Selenium pour vérifier la compatibilité du site sur plusieurs navigateurs.

Kanban - État d'avancement :

Estimation et temps réel :

- **Temps estimé :** 7h
- **Temps réel :** 8h

Résultats des tests par catégorie

1. Tests unitaires :

- Test de la méthode `getPublishedAtString()` : **OK**. La méthode retourne bien la date au format attendu.

2. Tests de validation :

- Validation des dates : **OK**. Les formations avec des dates valides sont acceptées, et celles avec des dates non valides sont rejetées correctement.

3. Tests d'intégration :

- Méthodes des Repository (`CategorieRepository`, `FormationRepository`, `PlaylistsRepository`) : **OK**. Tous les tests ajoutés, supprimés, et vérifiés fonctionnent correctement avec une base de données de test.

4. Tests fonctionnels :

- Accès aux pages (Accueil, Formations, Playlists) : **OK**. Toutes les pages sont accessibles avec le contenu attendu.
- Tests des tris et filtres : **OK**. Les tris affichent correctement les résultats, et les filtres fonctionnent sur les champs.
- Navigation entre les pages via les liens : **OK**. Les clics sur les boutons ou liens mènent aux pages attendues avec le contenu correct.

5. Tests de compatibilité (Selenium) :

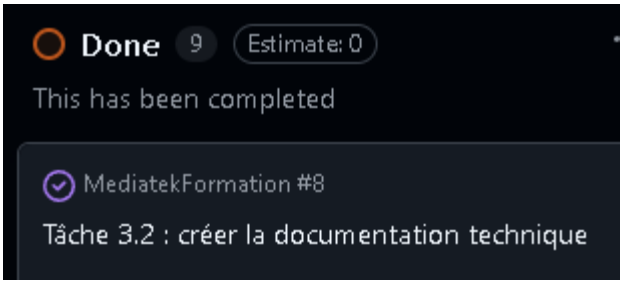
- **Navigateurs testés :** Chrome, Firefox, Edge.
- **Scénario :** Accès à la partie front-office, navigation, tests des liens et formulaires.
- **Bilan :** Compatibilité validée sur tous les navigateurs testés.

Tâche 3.2 : Créer la documentation technique

Description : Vérifier que tous les commentaires normalisés nécessaires pour la génération de la documentation technique ont été correctement ajoutés.

- Générer la documentation technique du site complet (front et back office) en excluant :
- Le code automatiquement généré par Symfony.
- Les fichiers du dossier `vendor`.

Kanban - État d'avancement :



Estimation et temps réel :

- **Temps estimé :** 1h
- **Temps réel : 1h**

Processus réalisé :

1. **Vérification des commentaires :**
    - Contrôle de chaque fichier pour s'assurer que les commentaires respectent les normes.
    - Vérification effectuée sur :
        - Les contrôleurs (front et back office).
        - Les formulaires.
        - Les Repository.
2. **Génération de la documentation technique :**
    - Outil utilisé : **NetBeans**.
    - Exclusions appliquées :
        - Le code automatiquement généré par Symfony.
        - Les fichiers du dossier `vendor`.

Tâche 3.3 : Créer la documentation utilisateur

Description : Créer une vidéo de démonstration montrant toutes les fonctionnalités du site (front et back office).

- Durée maximale : 5 minutes.
- Présentation claire des fonctionnalités avec manipulations et explications orales

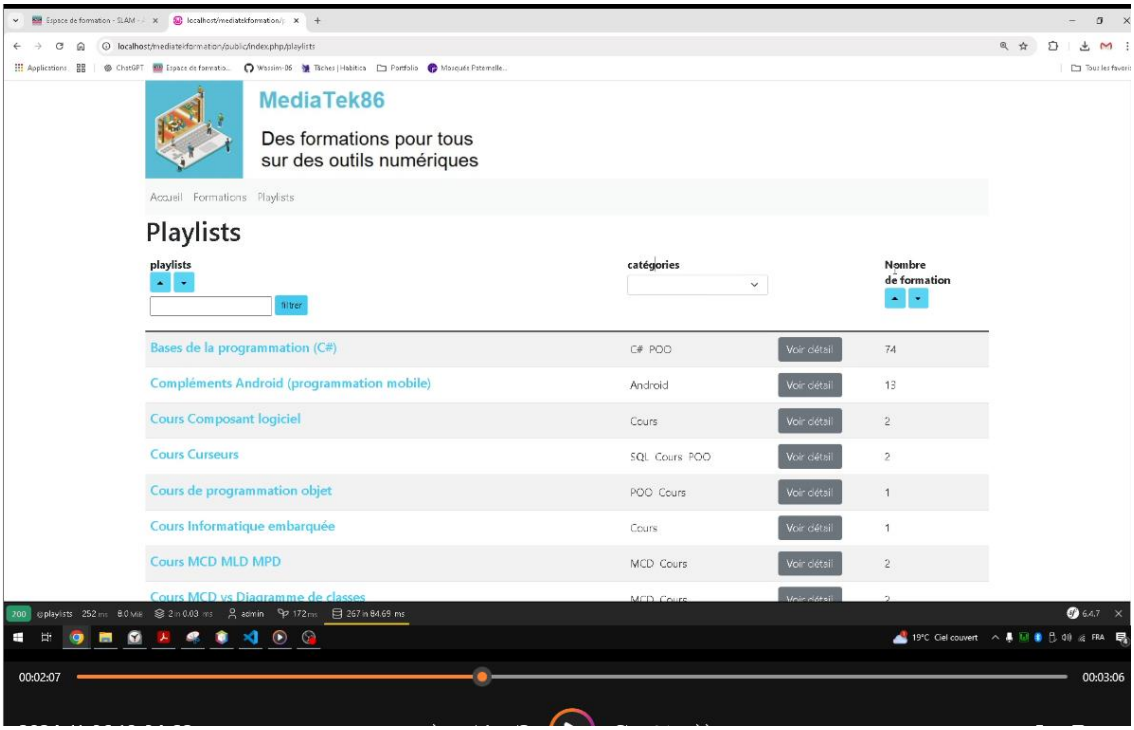
Kanban - État d'avancement :

Estimation et temps réel :

- **Temps estimé :** 2h
- **Temps réel : 1h**

Processus réalisé :

1. **Préparation :**
    - Identification des fonctionnalités à présenter :
        - **Front office :** Navigation, recherche, tri, filtres, détails des formations.
        - **Back office :** CRUD des formations, playlists, catégories, gestion de l'authentification.
2. **Enregistrement de la vidéo :**
    - Outil utilisé : **OBS Studio**
    - Parcours des fonctionnalités du site en temps réel.
    - Explications orales accompagnant chaque manipulation pour clarifier les actions.



Mission 4

Tâche 4.1 : Déployer le site

- **Description :**
- Déployer le site en ligne avec sa base de données opérationnelle.
- Assurer la mise à jour et le déploiement de la page "CGU".
- Rendre accessible la partie "admin" de manière sécurisée.
- Finaliser et documenter toutes les étapes nécessaires au déploiement.

Kanban - État d'avancement :

Estimation et temps réel :

- **Temps estimé :** 2h
- **Temps réel :** 2**h**

Préparation :

- Fichiers transférés sur le serveur via FTP.
- Configuration de la base de données (`DATABASE_URL` dans `.env`).
- **Mise à jour des CGU** :
- Modification et transfert sur le serveur.
- **Sécurisation admin** :
- Configuration d’un accès sécurisé.
- **Tests** :
- Vérifications des fonctionnalités du site, front et back.

Résultat final

Le site est en ligne, sécurisé et fonctionnel, avec une base de données connectée, une page CGU mise à jour, et un accès "admin" sécurisé.

Tâche 4.2 : Gérer la sauvegarde et la restauration de la bdd

- **Description :** La demande était de programmer une sauvegarde journalière automatisée de la base de données `ucjdscww_mediatekformation` et d'assurer une méthode de restauration à partir de cette sauvegarde.

Kanban - État d'avancement :

Estimation et temps réel :

- **Temps estimé :** 1h
- **Temps réel :** 2**h (en raison des limitations liées à l'hébergement et des ajustements nécessaires).**

Sauvegarde manuelle via phpMyAdmin
En raison de la version Lite de PlanetHoster, qui ne supporte pas les tâches Cron, nous avons opté pour une méthode manuelle d'exportation de la base de données via phpMyAdmin.

Étapes pour effectuer la sauvegarde manuelle :

1. Connectez-vous à phpMyAdmin.
2. Sélectionnez la base de données `ucjdscww_mediatekformation`.
3. Cliquez sur l'onglet **Exporter**.
4. Choisissez le format **SQL** et cliquez sur **Exécuter** pour télécharger la sauvegarde.
5. Sauvegardez le fichier SQL dans un dossier local dédié (ex. : `D:¥Sauvegardes`).

Étapes pour restaurer manuellement via phpMyAdmin :

1. Connectez-vous à phpMyAdmin.
2. Sélectionnez la base de données cible ou créez-en une nouvelle.
3. Cliquez sur l'onglet **Importer**.
4. Téléversez le fichier de sauvegarde SQL et cliquez sur **Exécuter**.
5. Vérifiez que les données et les tables ont été restaurées correctement.

Tâche 4.3 : Mise en place du déploiement continu

- **Description :**Configurer un système de déploiement continu pour que le site en ligne soit mis à jour automatiquement à chaque push effectué sur le dépôt GitHub.

Kanban - État d'avancement :

Estimation et temps réel :

- **Temps estimé :** 1h
- **Temps réel : 1h**

Résumé des étapes suivies :

1. **Configuration GitHub Actions :**
    - Création d'un fichier `main.yml` pour automatiser la synchronisation via FTP avec l'action **FTP-Deploy-Action**.
    - Configuration des chemins et ajout des secrets pour sécuriser les identifiants FTP.
2. **Création des secrets GitHub :**
    - Ajout du mot de passe FTP dans **Settings > Secrets and Variables**.
3. **Test et débogage :**
    - Résolution des erreurs sur les chemins, identifiants FTP et protocole.
    - Tests validés avec plusieurs commits.
4. **Validation finale :**
    - Déploiement testé avec succès : les modifications poussées dans GitHub sont automatiquement déployées en ligne

Script du déploiement continu :

```
name: Deploy website on push


on:
  push:
    branches:
```

```
    - master
jobs:
  web-deploy:
    name: Deploy
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2
    - name: Sync files via FTP
      uses: SamKirkland/FTP-Deploy-Action@4.3.0
      with:
        server: mediatekformation.go.yn.fr
        username: ucjdscww
        password: ${{ secrets.FTP_PASSWORD }}
        local-dir: ./ # Chemin vers ton projet local.
        server-dir: /public_html/mediatekformation/ # Chemin vers ton projet sur le serveur.
```

Bilan final du projet MediatekFormation

Objectifs atteints :

Fonctionnalités développées :

- **Back-end complet :** Toutes les fonctionnalités de gestion (CRUD) pour les formations, playlists et catégories sont implémentées.
- **Front-end amélioré :** Les tris et filtres permettent une navigation fluide pour l'utilisateur.
- **Authentification sécurisée :** Mise en place d’un système d’accès pour sécuriser les parties sensibles.

Mise en production :

- **Déploiement continu :** Le site est automatiquement mis à jour à chaque push sur le dépôt GitHub.
- **Hébergement :** Le site est fonctionnel en ligne grâce à l'hébergeur PlanetHoster.
- **Sauvegarde et restauration :** La base de données peut être sauvegardée manuellement pour garantir la sécurité des données.

Documentation complète :

- **Technique :** Fournit les détails nécessaires pour maintenir et faire évoluer le projet.
- **Utilisateur :** Simplifie l'utilisation des fonctionnalités du site.

Problèmes rencontrés :

1. **Accès à la base de données :**
    - Difficultés avec les identifiants MySQL et les permissions, nécessitant des ajustements dans `phpMyAdmin`.
2. **Déploiement continu :**
    - Erreurs liées à la configuration FTP et aux chemins du serveur, corrigées après plusieurs tests et modifications du fichier `.yml`.
3. **Limites de l'hébergeur :**
    - Absence de support pour les tâches cron, obligeant à gérer les sauvegardes manuellement.

Note : De nombreux autres problèmes ont également été rencontrés, mais ils ont tous été résolus au fil du projet grâce à des ajustements techniques et des recherches approfondies.