2025

Atelier professionnel Mediatek Formation

BTS SIO OPTION SLAM
ESTELLE FRIEDT

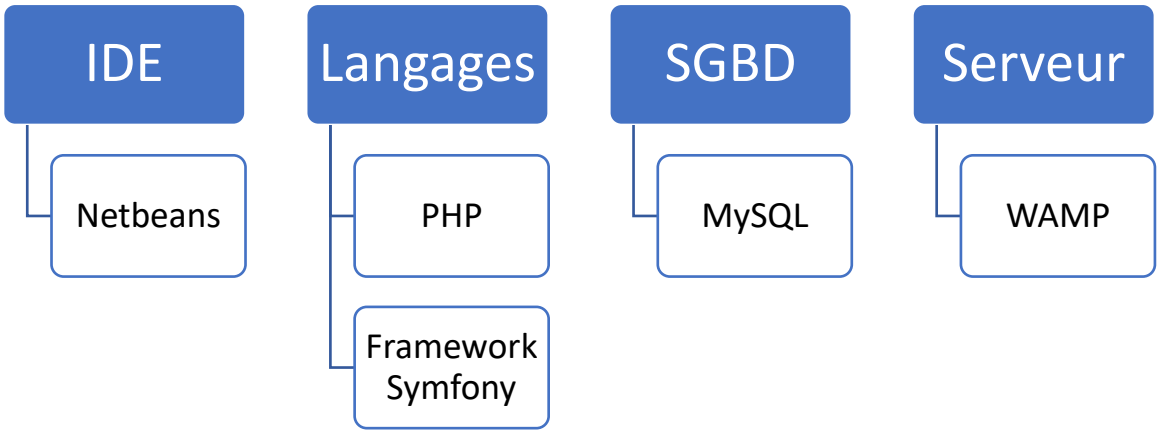
TABLE DES MATIERES

Contexte

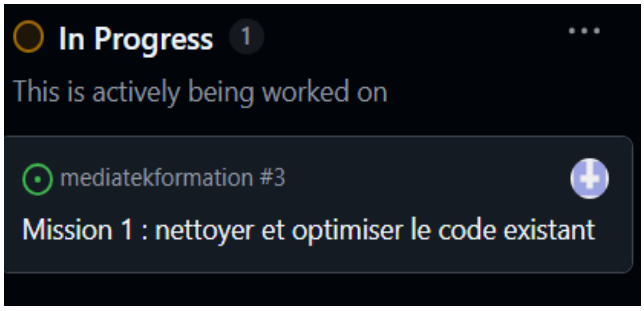
Le réseau des médiathèques de la Vienne, MediaTek86, a pour mission de centraliser les prêts de livres, DVD et CD, tout en développant les services numériques pour l'ensemble des médiathèques du département. Dans le but d'augmenter l'attractivité des médiathèques, MediaTek86 souhaite se développer autour de deux axes principaux :

- Enrichir ses services en permettant aux adhérents d'emprunter des films en VOD.
- Compléter l'offre traditionnelle des médiathèques en proposant des formations sur les outils numériques et des modules d'autoformation en ligne.

Langages et technologies utilisées



MISSION 1 : NETTOYER ET OPTIMISER LE CODE EXISTANT



TACHE 1 : NETTOYER LE CODE

Par convention il est nécessaire de nommer les constantes en majuscules :

× *Code avant correction* :

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

Intitulé de l'erreur Sonarlint :

✓ *Code après correction :*

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

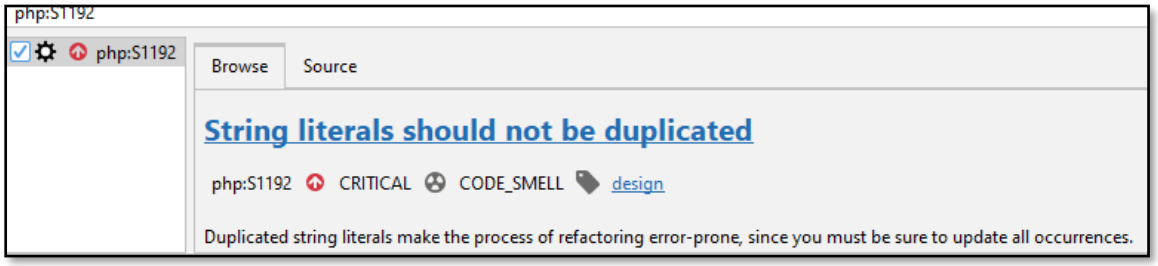
```
public function getMiniature(): ?string {
    return self::CHEMINIMAGE . $this->videoId . "/default.jpg";
}

public function getPicture(): ?string {
    return self::CHEMINIMAGE . $this->videoId . "/hqdefault.jpg";
}
```

Dans FormationController.php :

× *Code 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
    ]);
}
```

Erreur relevée par Sonarlint :



Il est donc recommandé de définir le chemin comme une constante afin de n'avoir à le modifier qu'une seule fois si nécessaire :

✓ *Code après correction :*

```php
class FormationsController extends AbstractController {

    const PAGES_FORMATIONS = "pages/formations.html.twig";
```

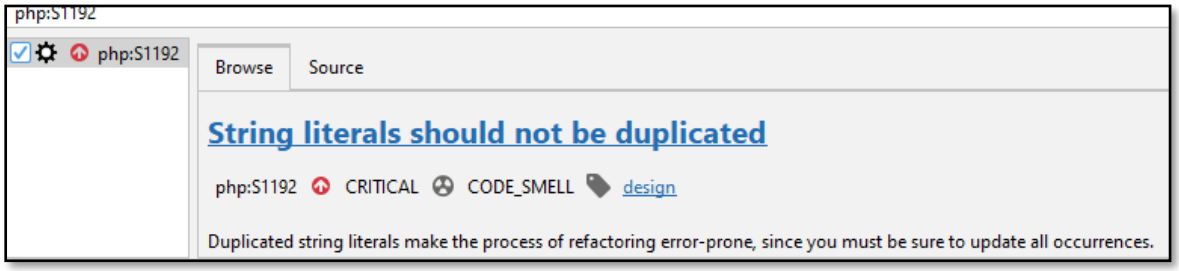
```php
#[Route('/formations', name: 'formations')]
public function index(): Response{
    $formations = $this->formationRepository->findAll();
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGES_FORMATIONS, [
        'formations' => $formations,
        'categories'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

Dans PlaylistsController.php :

× *Code avant correction :*

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

Erreur relevée par Sonarlint :



✓ *Code après correction :*

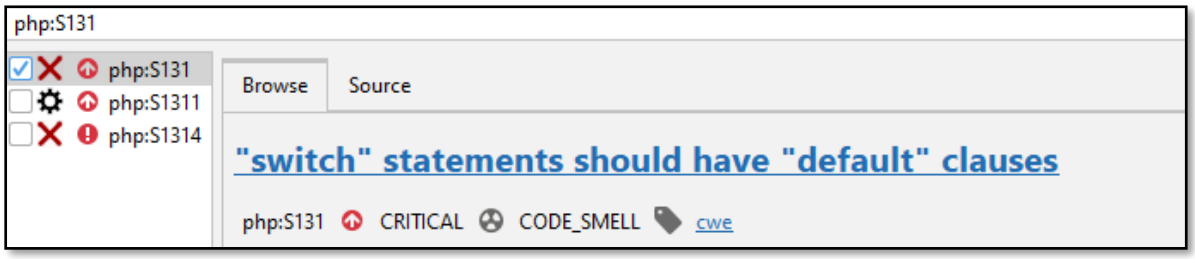
```php
class PlaylistsController extends AbstractController {

    const PAGES_PLAYLISTS = "pages/playlists.html.twig";
```

```php
    /**
     * @Route("/playlists", name="playlists")
     * @return Response
     */
    #[Route('/playlists', name: 'playlists')]
    public function index(): Response{
        $playlists = $this->playlistRepository->findAllOrderByName('ASC');
        $categories = $this->categorieRepository->findAll();
        return $this->render(self::PAGES_PLAYLISTS, [
            'playlists' => $playlists,
            'categories' => $categories
        ]);
```

× *Code avant correction :*

```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;
        }
```

Erreur relevée par Sonarlint :



Dans cette instruction switch, aucune clause default n'est présente. Cela signifie qu'aucune action n'est définie dans le cas où aucune des conditions spécifiées n'est remplie. Il est donc nécessaire d'ajouter une clause default pour garantir un comportement défini si un champ invalide est fourni :

✓ *Code après correction :*

```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;
        default:
            throw new \InvalidArgumentException("Invalid sorting parameter: $champ");
    }
```

Playlist.php :

× *Code avant correction :*

```php
public function getCategoriesPlaylist() : Collection
{
    $categories = new ArrayCollection();
    foreach($this->formations as $formation){
        $categoriesFormation = $formation->getCategories();
        foreach($categoriesFormation as $categorieFormation)
        if(!$categories->contains($categorieFormation->getName())){
            $categories[] = $categorieFormation->getName();
        }
    }
    return $categories;
}
```

Erreur relevée par Sonarlint :

Ici afin que le code soit plus clair il faut rajouter une accolade à la structure du foreach :

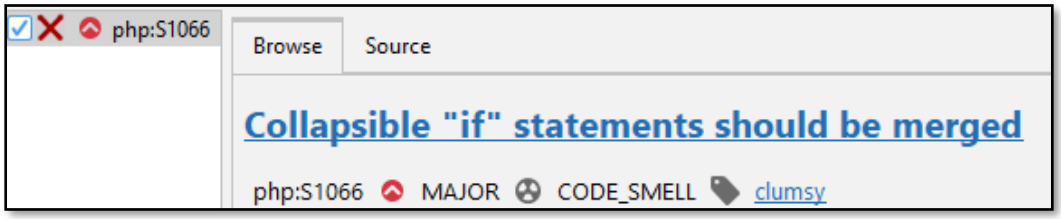
✓ *Code après correction :*

```php
/**
 * @return Collection<int, string>
 */
public function getCategoriesPlaylist(): Collection {

    $categories = new ArrayCollection();
    foreach ($this->formations as $formation) {
        $categoriesFormation = $formation->getCategories();
        foreach ($categoriesFormation as $categorieFormation) {
            if (!$categories->contains($categorieFormation->getName())) {
                $categories[] = $categorieFormation->getName();
            }
        }
    }
    return $categories;
}
```

× *Code avant correction :*

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

        return $this;
    }
```

Erreur relevée par Sonarlint :



L'erreur signalée ici indique que deux instructions if imbriquées peuvent être combinées.

Une modification du code est donc nécessaire en supprimant les 2ème if et en rajoutant &&.

✓ *Code après correction :*

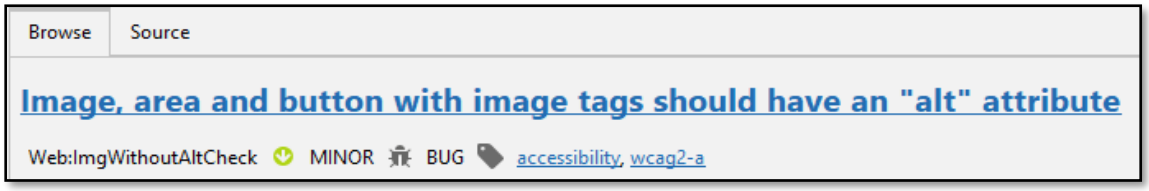
```php
    public function removeFormation(Formation $formation): static {
        if ($this->formations->removeElement($formation) && ($formation->getPlaylist() === $this)) {
            // set the owning side to null (unless already changed)

            $formation->setPlaylist(null);
        }
        return $this;
    }
```

Dossier templates :

× *Code avant correction :*

```
<td class="text-center">
    {% if formation.miniature %}
        <a href="{{ path('formations.showone', {id:formation.id}) }}">
            <img src="{{ formation.miniature }}">
        </a>
```

Erreur relevée par Sonarlint :



Cette erreur signifie que chaque balise <img> doit comporter un attribut alt. Il est donc nécessaire de modifier les fichiers concernés :

✓ *Code après correction :*

Dans **formations.html.twig** :

```
<td class="text-center">
    {% if formation.miniature %}
        <a href="{{ path('formations.showone', {id:formation.id}) }}">
            <img src="{{ formation.miniature }}" alt="Formations">
        </a>
```

Dans **accueil.html.twig** :

```
<img src="{{ formation.picture }}"
     class="card-img-top" style="width:100%;height:auto;">
```

```
<a href="{{ path('formations.showone', {id:formation.id}) }}">
<img src="{{ formation.picture }}" alt="image de la formation"
     class="card-img-top" style="width:100%;height:auto;">
```

Dans **playlists.html.twig** :

```
<a href="{{ path('formations.showone', {id:formation.id}) }}">
    <img src="{{ formation.miniature }}">
```

```
<a href="{{ path('formations.showone', {id:formation.id}) }}">
    <img src="{{ formation.miniature }}"alt="Formations">
```

Dans **basefront.html.twig** :

```
    <img src="{{ app.request.getBasePath()~'/banniere.jpg' }}" >
```

```
    <img src="{{ app.request.getBasePath()~'/banniere.jpg' }}"alt="Banniere image" >
```

× *Code avant correction :*

Dans **cgu.html.twig** :

```
(ci-après "<i>les Utilisateurs</i>")
```

```
(ci-après "<i>le site</i>").
```

Dans **basefront.html.twig** :

```
<p><small><i>
    Consultez nos <a class="link-secondary" href="{{ path('cgu') }}">Conditions Générales d'Utilisation</a>
</i></small></p>
```

Erreur relevée par Sonarlint :

Cette erreur indique que les balises strong et em doivent être utilisées pour marquer le texte qui dois être mis en avant.

✓ *Code après correction :*

Dans **cgu.html.twig** :

Les balises <i> (italique) sont remplacées par les balises :

```
(ci-après "<em>les Utilisateurs</em>")
```

```
(ci-après "<em>le site</em>")
```

Dans **basefront.html.twig** :

```
<p><small><em>
    Consultez nos <a class="link-secondary" href="{{ path('cgu') }}">Conditions Générales d'Utilisation</a>
</em></small></p>
```

× *Code avant correction :*

Dans **accueil.html.twig** :

```
Voici les <strong>deux dernières formations</strong> ajoutées au catalogue :
<table class="table">
```

Dans **playlists.html.twig** :

```
<table class="table table-striped">
```

Dans formations.html.twig :

```
<table class="table table-striped">
```

Erreur relevée par Sonarlint :

Cette erreur indique qu'il n'y a pas d'attributs pour décrire le contenu du tableau

Il faut donc rajouter la balise <caption> afin d'améliorer l'accessiiblité et fournir une explication du contenu du tableau :

✓ *Code après correction :*

Dans **accueil.html.twig** :

```
<table class="table">
    <caption> table pour les deux dernières formations ajoutées au catalogue</caption>
```

Dans **playlists.html.twig** :

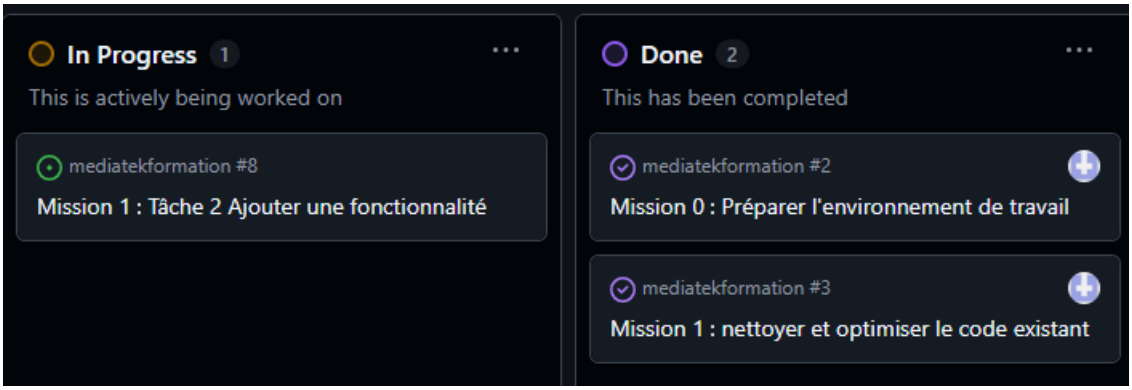
```
<table class="table table-striped">
    <caption>Liste des playlists</caption>
```

Dans **formations.html.twig** :

```
<table class="table table-striped">
    <caption>Liste des formations</caption>
    <thead>
```

TACHE 2 : AJOUTER UNE FONCTIONNALITE

⋙ 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.



- Ajout dans PlaylistRepository d'une méthode qui permet de récupérer toutes les playlists triées en fonction du nombre total de formations associées à chaque playlist :
  - Définition du paramètre $ordre pour spécifier le tri
  - Construction de la requête avec Doctrine :
    - Utilisation de createQueryBuilder('p') pour interroger l'entity Playlist puis ajout d'une jointure avec l'entité formations à l'aide de leftJoin('p.formations','f') pour récupérer les formations associées et grouper les résultats par l'ID de la playlist.
    - Utilisation de la méthode orderBy pour trier les playlists selon le nombre total de formations en intégrant la fonction COUNT(f.id) pour compter le nombre de formations.
    - Finalisation de la requête avec l'appel de getQuery() et getResult() afin d'obtenir les résultats.

```php
/**
 * Retourne les playlists triées en fonction du nombre total de formations
 * @param type $ordre
 * @return array
 */
public function findAllOrderByTotalNb($ordre): array{
    return $this->createQueryBuilder('p')
            ->leftJoin('p.formations', 'f')
            ->groupBy('p.id')
            ->orderBy('COUNT(f.id)',$ordre)
            ->getQuery()
            ->getResult();

}
```

- Dans PlaylistsController :

Ajout d'un nouveau cas de tri dans le switch de la méthode sort. Ce tri permet de choisir la méthode de récupération des playlists en fonction du critère de l'utilisateur (name, ou nb formations) et de l'ordre (ASC ou DESC).Les playlists triées et les catégories sont ensuite envoyées à la vue pour affichage.

```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 = $this->playlistRepository->findAllOrderByTotalNb($ordre);
            break;
        default:
            throw new \InvalidArgumentException("Invalid sorting parameter: $champ");
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGES_PLAYLISTS, [
                'playlists' => $playlists,
                'categories' => $categories
    ]);
}
```

- Gestion de la vue : mise à jour du fichier playlist.html :

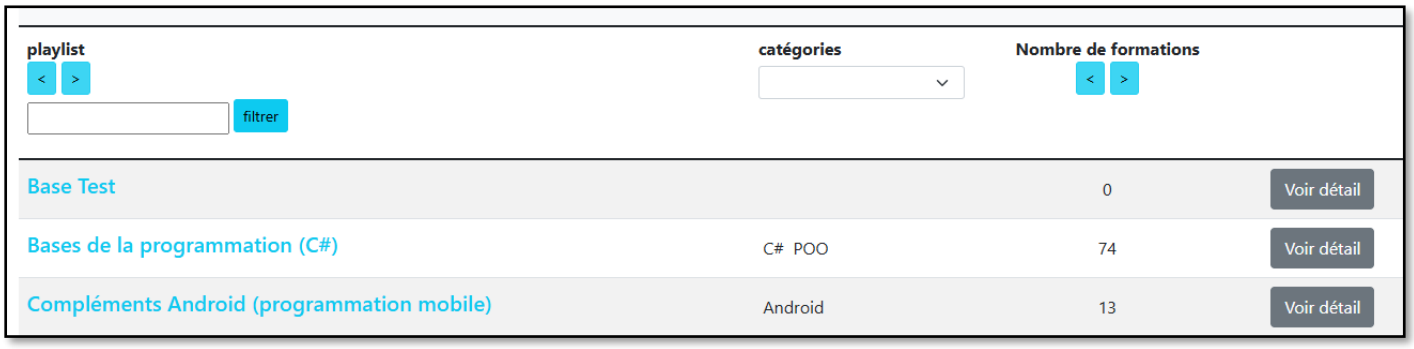
Ajout de l'information du nombre de formation par playlist avec l'utilisation de playlist[k].formations|length

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

Afin de permettre le tri croissant ou décroissant sur le nombre de formations présente dans la playlist on ajoute les boutons avec le path correspondant :

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

Résultat avec la colonne ajoutée dans la page qui liste les playlists :

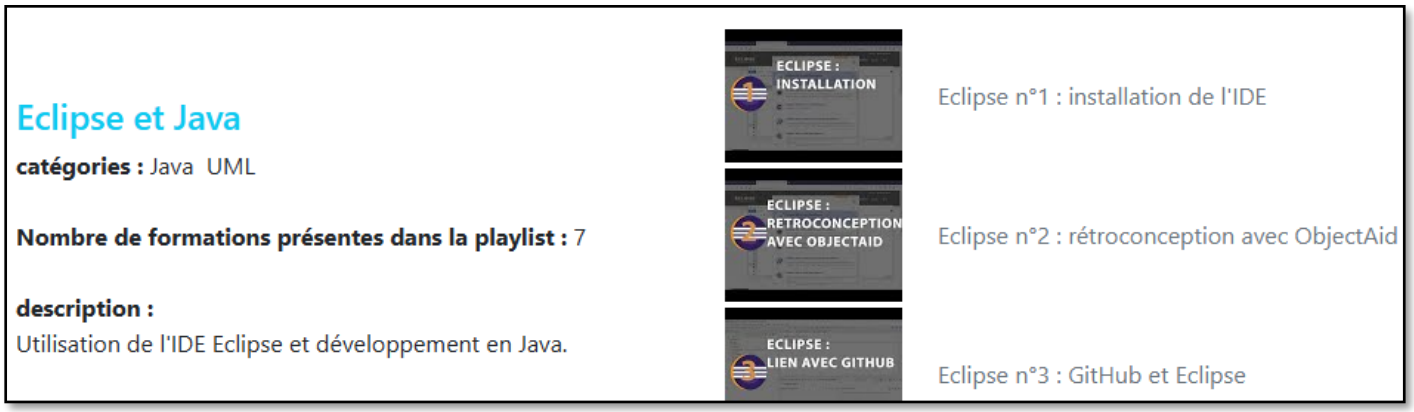


Il était également demandé d'afficher l'information dans la page d'une playlist :

- Ajout dans playlist.html.twig :

```
<strong>Nombre de formations présentes dans la playlist :</strong>
{{ playlist.formations|length }}
```

Résultat avec la colonne ajoutée sur la page d'une playlist en particulier :



MISSION 2 : CODER LA PARTIE BACK-OFFICE

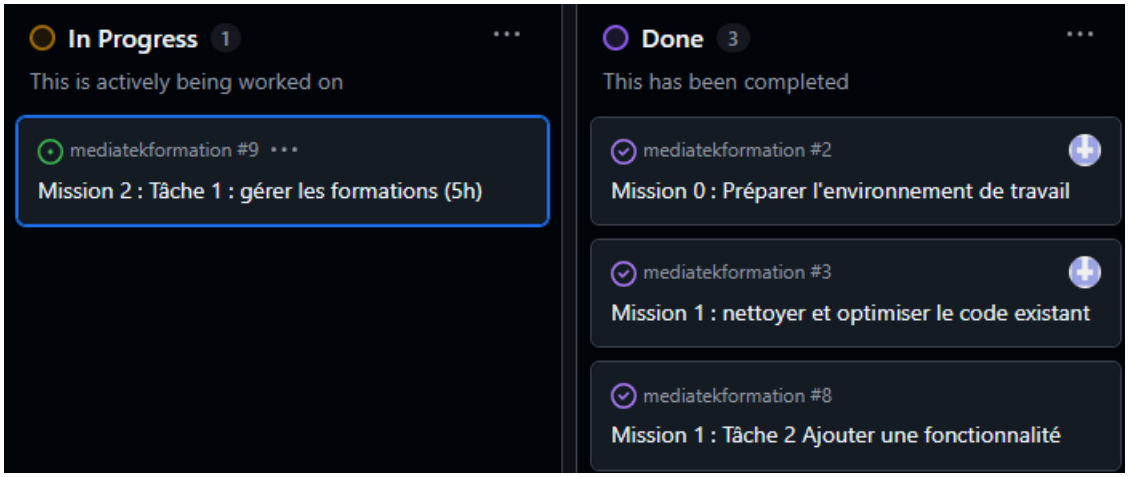
TACHE 1 : GERER LES FORMATIONS

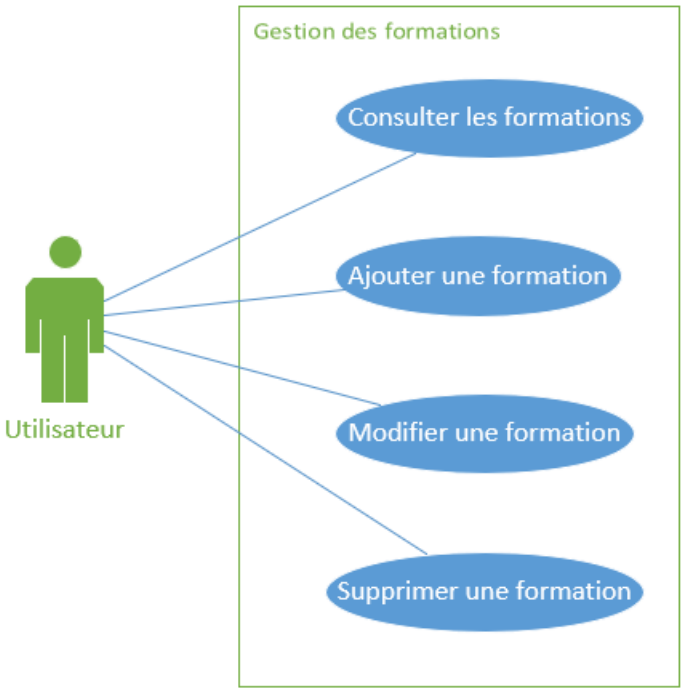
- Une page doit permettre de lister les formations et, pour chaque formation, afficher un bouton permettant de la supprimer (après confirmation) et un bouton permettant de la modifier. Si une formation est supprimée, il faut aussi l'enlever de la playlist où elle se trouvait.

- 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

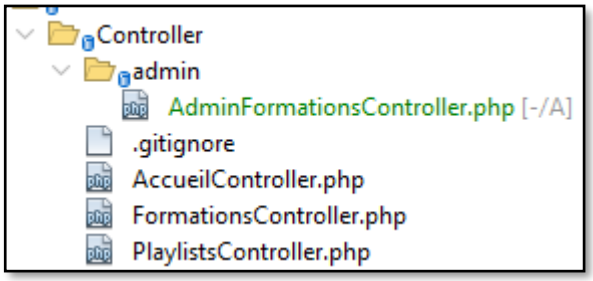
- Le clic sur le bouton permettant de modifier une formation doit amener sur le même formulaire, mais cette fois prérempli.



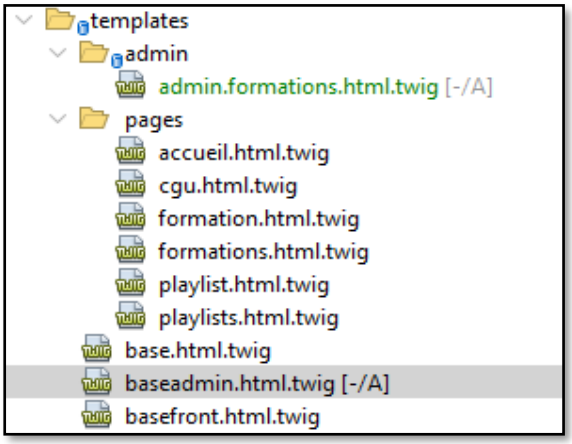


Création des dossiers et fichiers nécessaires :

- Ajout du dossier admin dans Controller et création du fichier AdminFormationsController



- Ajout d'un dossier admin dans templates et création du fichier admin.formations.html.twig et de baseadmin.html.twig



➢ Création de la classe AdminFormationsController :

- Création du constructeur pour accéder aux données des formations et des catégories.
- Création de la méthode index pour récupérer les enregistrements et les envoyer à la vue : admin.formations.html.twig

```php
class AdminFormationsController extends AbstractController {

    const PAGES_FORMATIONS_ADMIN = "admin/admin.formations.html.twig";

    /**
     * Accès aux formations
     * @var FormationRepository
     */
    private $formationRepository;

    /**
     * Accès aux catégories
     * @var CategorieRepository
     */
    private $categorieRepository;

    /**
     * Création du contructeur
     * @param FormationRepository $formationRepository
     * @param CategorieRepository $categorieRepository
     */
    function __construct(FormationRepository $formationRepository, CategorieRepository $categorieRepository) {
        $this->formationRepository = $formationRepository;
        $this->categorieRepository = $categorieRepository;
    }

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

➢ Dans la page admin.formations.html.twig :

Boucle for pour parcourir toutes les formations et affichage des détails de celles-ci : tire, playlist, catégories, date de publication.

```
{% for formation in formations %}
    <tr class="align-middle">
        <td>
            <h5 class="text-info">
                {{ formation.title }}
            </h5>
        </td>
        <td class="text-left">
            {{ formation.playlist.name }}
        </td>
        <td class="text-left">
            {% for categorie in formation.categories %}
                {{ categorie.name }}<br />
            {% endfor %}
        </td>
        <td class="text-center">
            {{ formation.publishedatstring }}
        </td>
        <td class="text-center">
            {% if formation.miniature %}
                <a href="{{ path('formations.showone', {id:formation.id}) }}">
                    <img src="{{ formation.miniature }}" alt="Formations">
                </a>
            {% endif %}
```

Création des boutons permettant supprimer et modifier la formation avec les path correspondants :

```
<a href="{{ path('admin.formation.edit', {id:formation.id}) }}" class="btn btn-secondary">Modifier</a>
<a href="{{ path('admin.formations.suppr', {id:formation.id}) }}" class="btn btn-danger" onclick="return confirm('Etes-vous sûr de vouloir supprimer
```

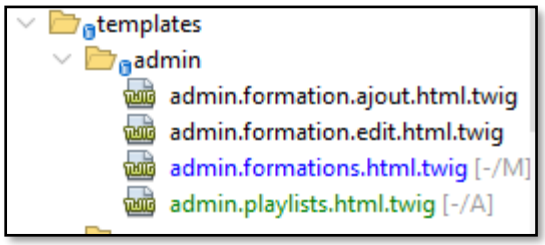
Création d’une nouvelle route dans AdminFormationsController :

Cette méthode permet de supprimer une formation.

Si la formation est associée à une playlist, la méthode removeFormation() est utilisée pour la retirer de cette playlist avant de supprimer la formation de la base de données. Un message de confirmation est ensuite affiché et l'utilisateur est redirigé vers la page des formations

```php
/**
 * Suppression d'une formation
 * @param Formation $formation
 * @return Response
 */
#[Route('/admin/suppr/{id}', name: 'admin.formations.suppr')]
public function suppr(Formation $formation): Response {
    if ($formation) {
        //récupération de la playlist associée à la formation
        $playlist = $formation->getPlaylist();
        //si la formation est liée à une playlist retrait de celle ci
        if ($playlist) {
            $playlist->removeFormation($formation);
        }
        //Suppression dans la bdd
        $this->formationRepository->remove($formation);
    }
    $this->addFlash('danger','La formation' . $formation->getTitle() . ' a été supprimée avec succès.');

    return $this->redirectToRoute('admin.formations');
}
```

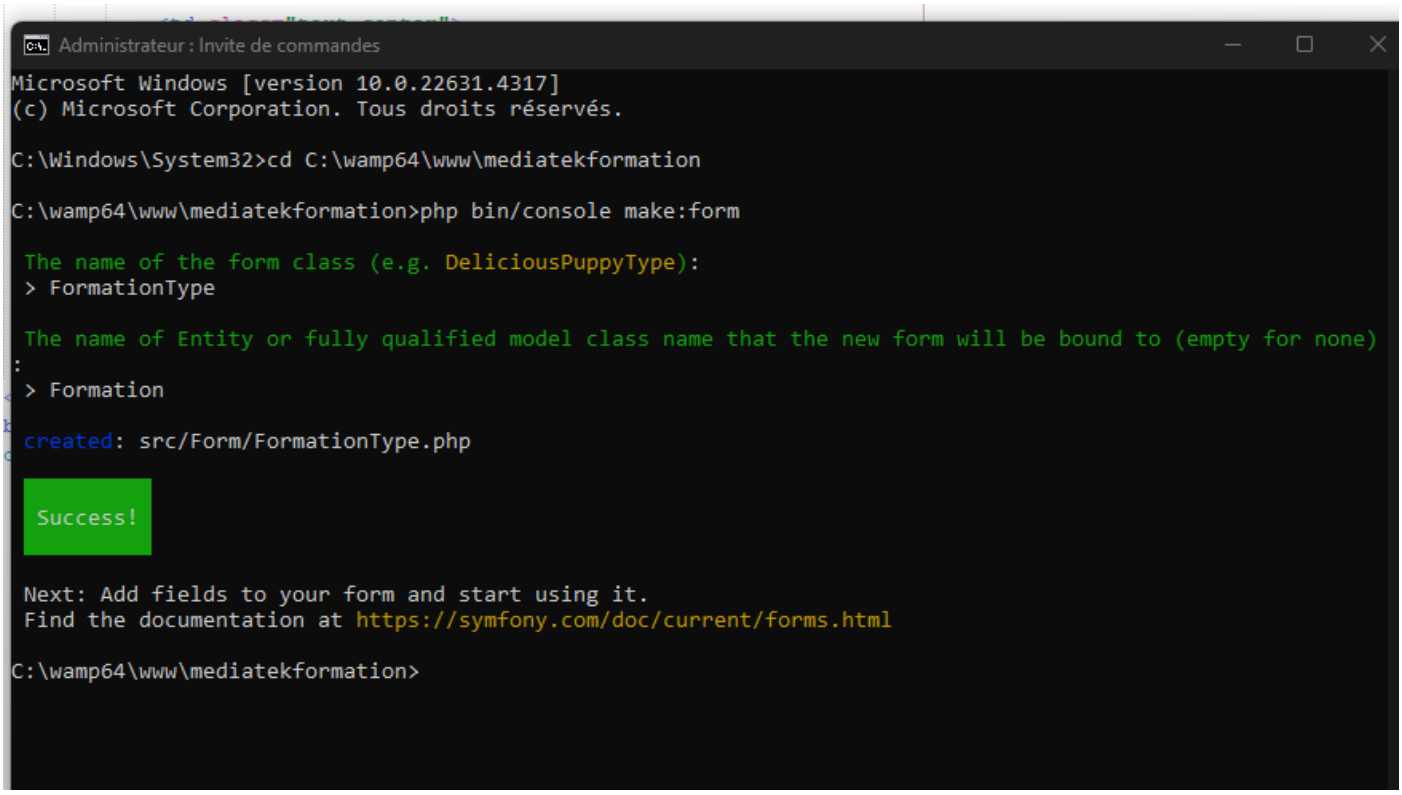
Création des nouveaux fichiers twig : admin.formation.edit et admin.formation.ajout:



Création du formulaire :

Dans une fenêtre de commande dans le dossier du projet je tape la commande :

Php bin/console make :form



Le formulaire va permettre la modification ou l'ajout d'une nouvelle formation avec le renseignement de plusieurs données qui vont également être contrôlées :

- La date de publication :
  - On utilise html 5 => true afin d'activer l'utilisation du sélecteur de date permettant ainsi à l'utilisateur de choisir une date via un calendrier interactif.
  - **required => true** garanti que la date doit obligatoirement être renseignée avant de soumettre le formulaire.
  - Pour s'assurer que la date de publication n'est pas postérieure à aujourd'hui, une contrainte de validation est ajoutée dans l'entité Formation.

```php
#[ORM\Column(type: Types::DATETIME_MUTABLE, nullable: true)]
#[Assert\LessThanOrEqual('today', message:"La date de publication ne peut pas être postérieure à aujourd'hui")]
private ?\DateTimeInterface $publishedAt = null;
```

  - On ajoute également un placeholder pour définir un texte d'exemple dans le champs de saisie ainsi qu'un onkeydown=> return false qui empêche l'utilisateur de saisir la date manuellement et limite la saisie au sélecteur de date.

- Seuls les champs description et la sélection de catégories ne sont pas obligatoires c'est pourquoi on utilise ici required=> false.

```php
$builder
        ->add('publishedAt', DateType::class, [
            'label' => 'Date de publication',
            'widget' => 'single_text',
            'data' => isset($options['data']) &&
            $options['data']->getPublishedAt() != null ? $options['data']->getPublishedAt() : new DateTime('now'),
            'html5' => true, // Utiliser le sélecteur de date natif HTML5
            'required' => true,
            'attr' => [
                'placeholder' => 'jj/mm/aaaa',
                'onkeydown' => 'return false;', // Empêche la saisie de texte
            ],
        ])
        ->add('title', TextType::class, [
            'label' => 'Titre',
            'required' => true,
                ]
        )
        ->add('description', TextareaType::class,[
            'required'=> false,
        ])
        ->add('categories', EntityType::class, [
            'class' => Categorie::class,
            'choice_label' => 'name',
            'multiple' => true,
            'expanded' => true,
            'required' => false,
        ])

        ->add('videoId', TextType::class, [
        'required' => false,

    ])
        ->add('playlist', EntityType::class, [
            'class' => Playlist::class,
            'choice_label' => 'name',
            'multiple' => false,
            'required' => true,
        ])
        ->add('submit', SubmitType::class, [
```

➢ Création d'une nouvelle route dans AdminFormationController :

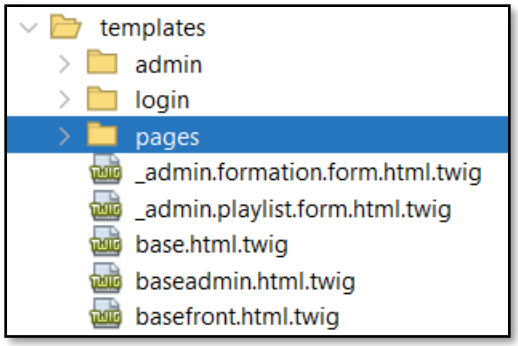
- Lorsqu'un utilisateur clique sur le bouton Modifier, symfony récupère l'ID de la formation à partir de l'URL.
  L'ID est alors utilisé pour récupérer l'objet Formation correspondant en base de données.
- Génération du formulaire avec createForm(FormationType :class, $formation) qui permet de pré remplir les champs avec les données existantes de la formation.
- Le formulaire est ensuite envoyé à la vue. Lors de la soumission ce celui-ci Symfony récupère et vérifie les données. Si celles-ci sont valides, la méthode add du repository est appelée pour enregistrer les modifications en base de données.
- Un message de confirmation est affiché à l'utilisateur et il est redirigé vers la liste des formations.

```php
/**
 * Modification d'une formation
 * @param Formation $formation
 * @param Request $request
 * @return Response
 */
#[Route('/admin/edit/{id}', name: 'admin.formation.edit')]
public function edit(Formation $formation, Request $request): Response {
    $formFormation = $this->createForm(FormationType::class, $formation);

    $formFormation->handleRequest($request);
    if ($formFormation->isSubmitted() && $formFormation->isValid()) {
        $this->formationRepository->add($formation);
        $this->addFlash('success', 'La modification de la formation ' . $formation->getTitle() . ' a bien été prise en compte.');
        return $this->redirectToRoute('admin.formations');
    }

    return $this->render("admin/admin.formation.edit.html.twig", [
                'formation' => $formation,
                'formFormation' => $formFormation->createView()
    ]);
}
```

Pour l'ajout d'une formation :

```php
/**
 * Ajout d'une formation
 * @param Request $request
 * @return Response
 */
#[Route('/admin/ajout', name: 'admin.formation.ajout')]
public function ajout(Request $request): Response {
    $formation = new Formation();
    $formFormation = $this->createForm(FormationType::class, $formation);

    $formFormation->handleRequest($request);
    if ($formFormation->isSubmitted() && $formFormation->isValid()) {
        $this->formationRepository->add($formation);
        $this->addFlash('success','La formation ' . $formation->getTitle() . ' a été ajoutée au catalogue.');
        return $this->redirectToRoute('admin.formations');
    }

    return $this->render("admin/admin.formation.ajout.html.twig", [
                'formation' => $formation,
                'formFormation' => $formFormation->createView()
    ]);
}
```

Création d'un nouveau fichier _admin.formation.form.html.twig qui contient la structure du formulaire. Ce fichier sera réutilisé à la fois pour la vue de modification et l'ajout de formations.




```php
/**
 * Tri des formaitons en fonction du champs et ordre spécifié
 * @param type $champ
 * @param type $ordre
 * @param type $table
 * @return Response
 */
#[Route('/admin/tri/{champ}/{ordre}/{table}', name: 'admin.formation.sort')]
public function sort($champ, $ordre, $table = ""): Response {
    $formations = $this->formationRepository->findAllOrderBy($champ, $ordre, $table);
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGES_FORMATIONS_ADMIN, [
                'formations' => $formations,
                'categories' => $categories
    ]);
}
```

Il est ensuite inclus dans la vue de modification de formation admin.formation.edit.html.twig.

```twig
{% extends "baseadmin.html.twig" %}
{% block body %}
    <h2>Détail de la formation</h2>
    {{ include ('_admin.formation.form.html.twig') }}
{% endblock %}
```

Ajouter les mêmes tris et filtres que dans le front office :

Création de nouvelles routes dans le controller des formations :

- Admin.formation.sort : tri des formations en fonction d'un champ et d'un ordre croissant ou décroissant

- Admin.formations.findallcontain : permet de trier les formations selon un critère saisi par l'utilisateur (titre, playlist, catégorie)

Ajout des boutons de tri dans la vue :

Tri par titre ASC / DESC

```
<a href="{{ path('admin.formation.sort', {champ:'title', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"><</a>
<a href="{{ path('admin.formation.sort', {champ:'title', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true">></a>
```

Tri par playlist

```
<a href="{{ path('admin.formation.sort', {table:'playlist', champ:'name', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button"
<a href="{{ path('admin.formation.sort', {table:'playlist', champ:'name', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button"
```

Tri par date

```
<a href="{{ path('admin.formation.sort', {champ:'publishedAt', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"><</a>
<a href="{{ path('admin.formation.sort', {champ:'publishedAt', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true">></a>
```

Ajout des formulaires :

Filtre par titre de formation avec ajout d'une protection CSRF et génération d'un token unique

```
<form class="form-inline mt-1" method="POST" action="{{ path('admin.formations.findallcontain', {champ:'title'}) }}">
    <div class="form-group mr-1 mb-2">
        <input type="text" class="sm" name="recherche"
               value="{% if valeur|default and not table|default %}{{ valeur }}{% endif %}">
        <input type="hidden" name="_token" value="{{ csrf_token('filtre_title') }}">
        <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
    </div>
</form>
```

```
/**
 * Filtre les formations en fonction de la valeur saisie par l'utilisateur
 * @param type $champ
 * @param Request $request
 * @param type $table
 * @return Response
 */
#[Route('/admin/recherche/{champ}/{table}', name: 'admin.formations.findallcontain')]
public function findAllContain($champ, Request $request, $table = ""): Response {
    $valeur = $request->get("recherche");
    $formations = $this->formationRepository->findByContainValue($champ, $valeur, $table);
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGES_FORMATIONS_ADMIN, [
                'formations' => $formations,
                'categories' => $categories,
                'valeur' => $valeur,
                'table' => $table
    ]);
}
```

Filtre par nom de playlist

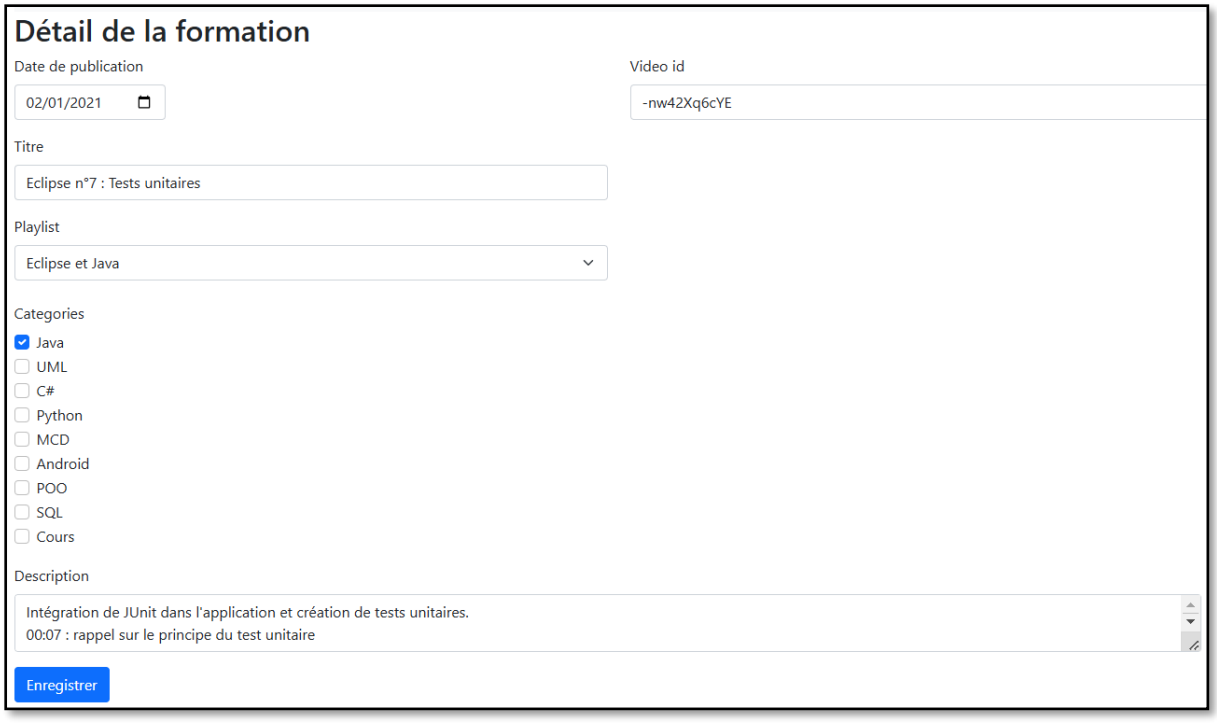
```
<form class="form-inline mt-1" method="POST" action="{{ path('admin.formations.findallcontain', {champ:'name', table:'playlist'}) }}">
    <div class="form-group mr-1 mb-2">
        <input type="text" class="sm" name="recherche"
               value="{% if valeur|default and table|default and table=='playlist' %}{{ valeur }}{% endif %}">
        <input type="hidden" name="_token" value="{{ csrf_token('filtre_name') }}">
        <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
    </div>
</form>
```

Filtre par catégorie avec choix dans une liste

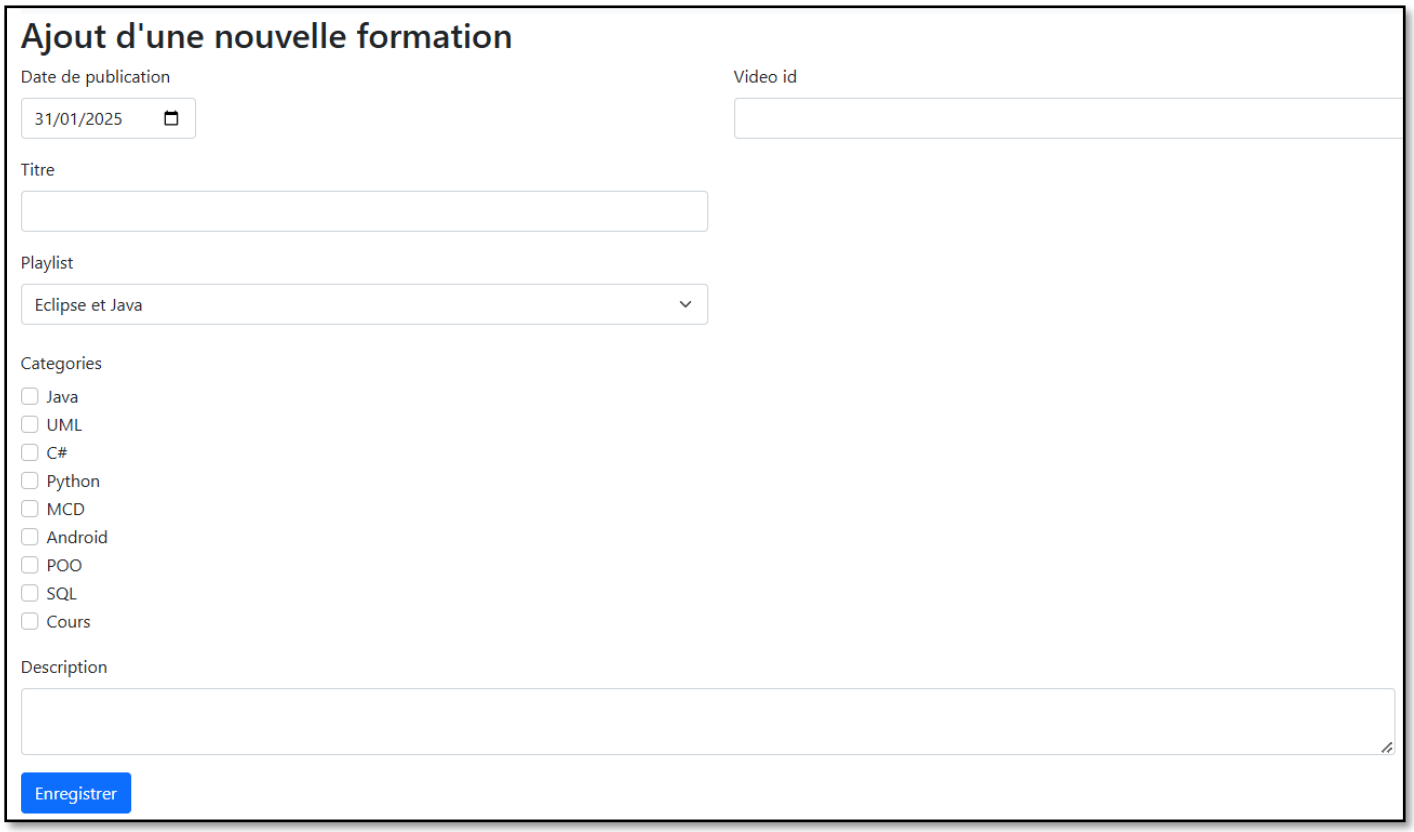
```
<form class="form-inline mt-1" method="POST" action="{{ path('admin.formations.findallcontain', {champ:'id', table:'categories'}) }}">
    <select class="form-select form-select-sm" name="recherche" id="recherche" onchange="this.form.submit()">
        <option value=""></option>
```

Page de gestion des formations :

Page de modification d'une formation avec le formulaire prérempli :



Page d’ajout d’une nouvelle formation avec le formulaire à renseigner :



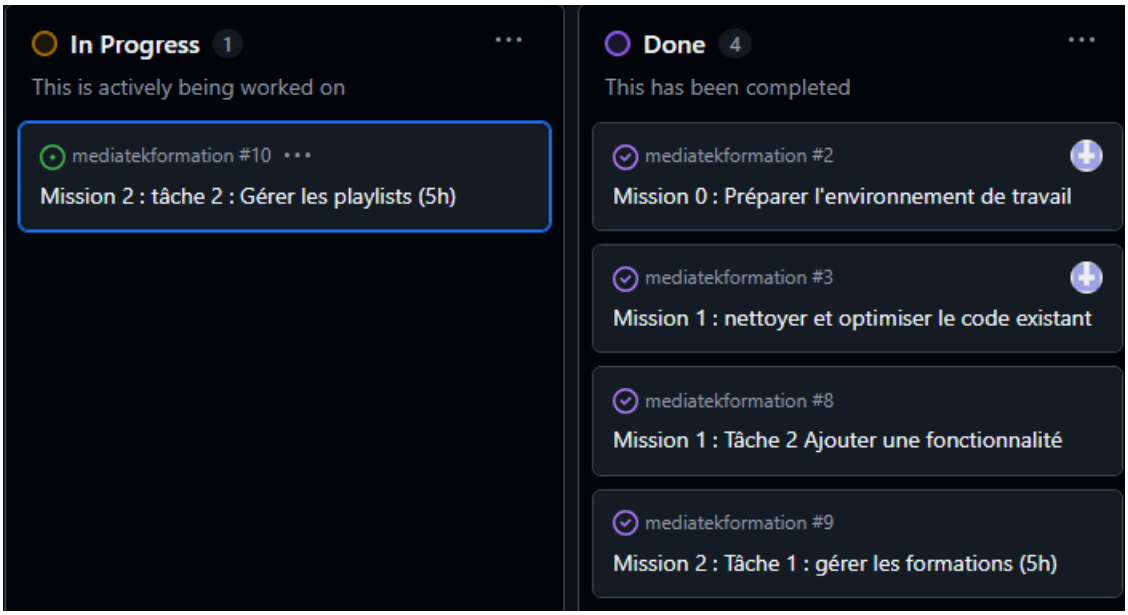
TACHE 2 : GERER LES PLAYLISTS

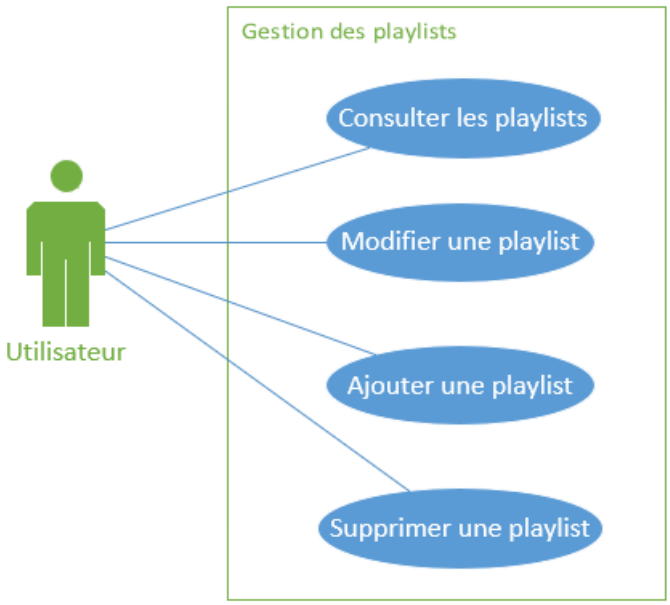
- Une page doit permettre de lister les playlists et, pour chaque playlist, afficher un bouton permettant de la supprimer (après confirmation) et un bouton permettant de la modifier. La suppression d'une playlist n'est possible que si aucune formation n'est rattachée à elle.

- 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.

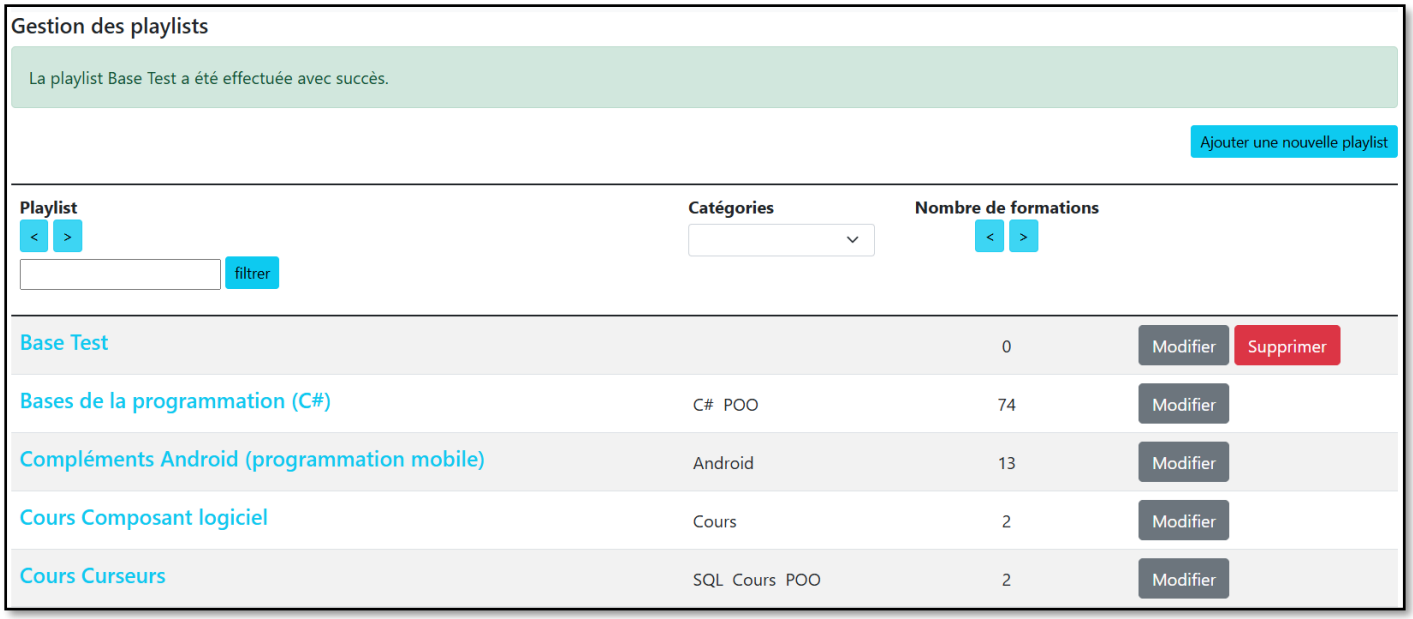
- Le clic sur le bouton permettant de modifier une playlist doit amener sur le même formulaire, mais cette fois prérempli. Cette fois, la liste des formations de la playlist doit apparaître, mais il ne doit pas être possible d'ajouter ou de supprimer une formation : ce n'est que dans le formulaire de la formation qu'il est possible de préciser sa playlist de rattachement.





Pour cette tâche, j'ai suivi un processus similaire à celui utilisé pour la gestion des formations. Une page est créée pour afficher la liste des playlists, avec des boutons permettant de les supprimer (après confirmation) ou de les modifier

Page de la gestion des playlists :

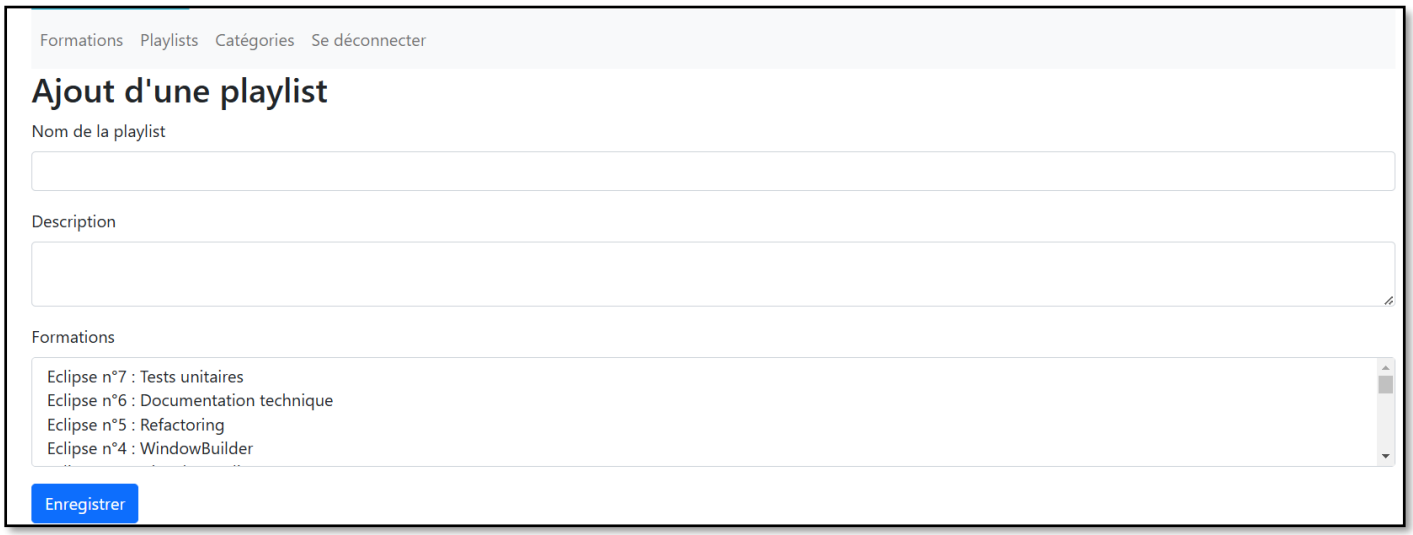


La suppression d'une playlist n'est possible que si aucune formation ne lui est rattachée. Pour garantir cette condition, le bouton "Supprimer" ne s'affiche que lorsque la playlist ne contient aucune formation.

```
{% if playlists[k].formations|length == 0 %}
    <a href="{{ path('admin.playlist.suppr',{id:playlists[k].id})  }}" class="btn btn-danger" onclick="return confirm('Etes-vous sûr de vouloir supprimer
{% endif %}
```

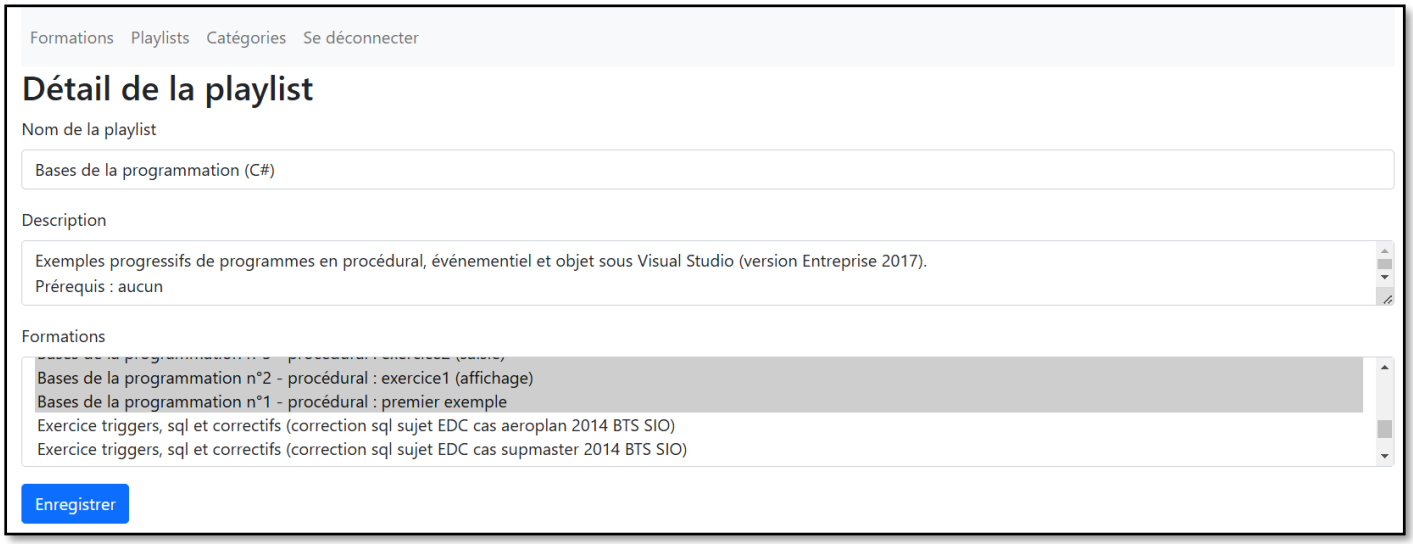
Le bouton "Modifier", quant à lui, est toujours visible et permet d'accéder au formulaire de modification de la playlist.

Les options de tri et de filtre présentes dans le front office sont également implémentées dans le back office. Un bouton permet d'accéder à un formulaire d'ajout de playlist, où les saisies sont validées.



L'ajout nécessite la saisie du nom (obligatoire) et de la description.

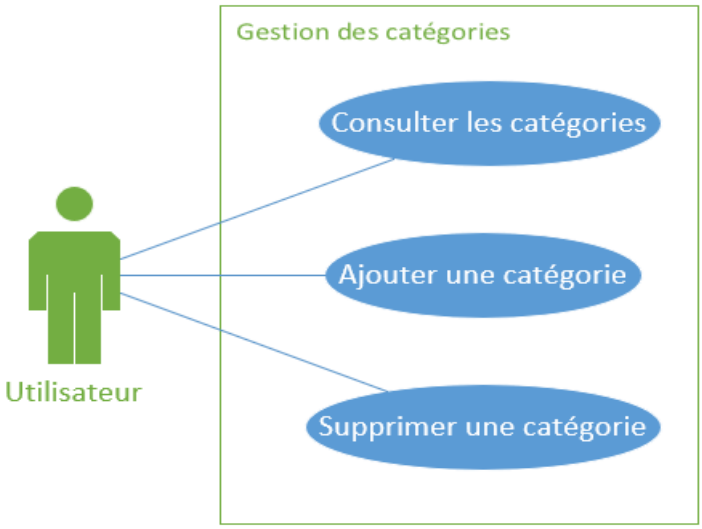
Lors de la modification d'une playlist, le formulaire est prérempli et affiche la liste des formations associées, sans possibilité d'ajouter ou de supprimer des formations, cette gestion étant réservée à l'édition de la formation elle-même.



TACHE 3 : GERER LES CATEGORIES

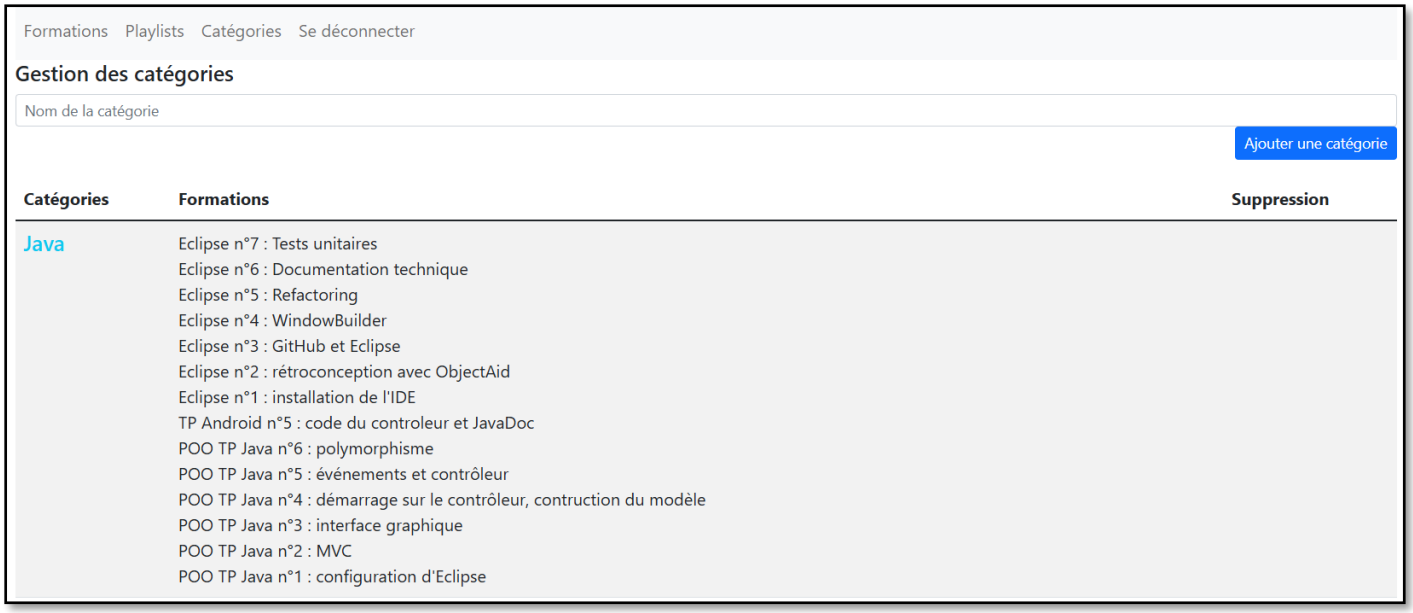
- Une page doit permettre de lister les catégories et, pour chaque catégorie, afficher un bouton permettant de la supprimer. Attention, une catégorie ne peut être supprimée que si elle n'est rattachée à aucune formation

- 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.



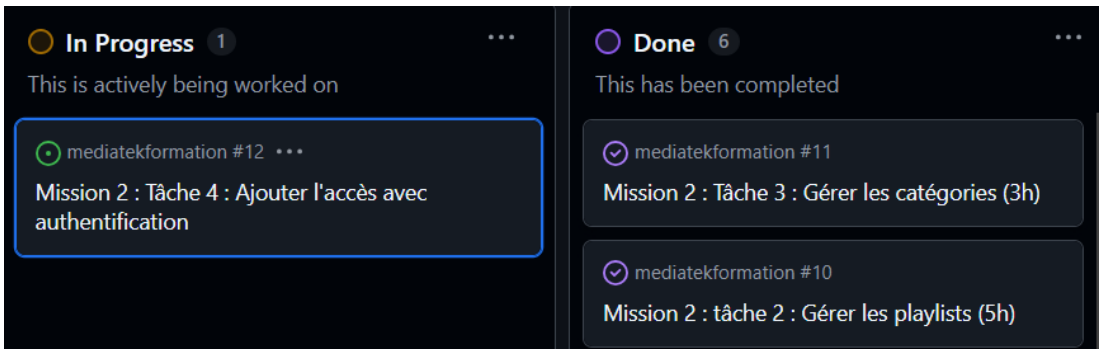
Pour cette tâche, j'ai suivi un processus similaire à celui utilisé pour la gestion des formations et des playlists. Une page est créée pour afficher la liste des catégories, avec des boutons permettant de les supprimer (après confirmation). Un mini formulaire est également ajouté pour permettre de saisir directement une nouvelle catégorie si son nom n'existe pas.

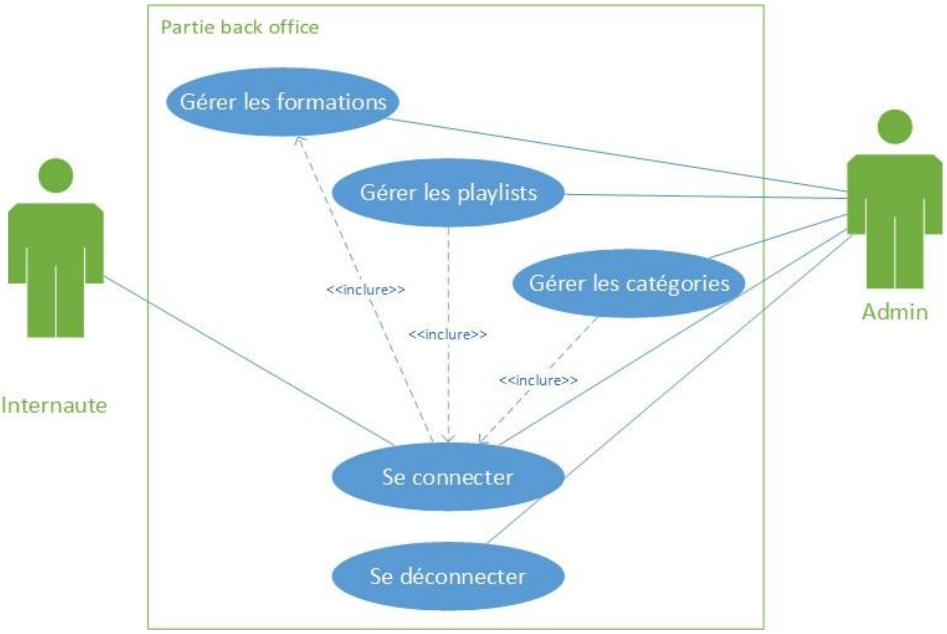
Page de gestion des catégories :



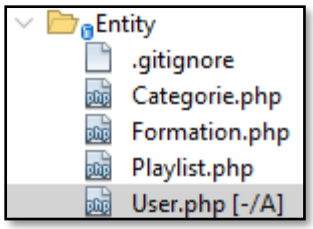
TACHE 4 : AJOUTER L'ACCES AVEC AUTHENTIFICATION

- Le back office ne doit être accessible qu'après authentification : un seul profil administrateur doit avoir le droit d'accès.
- Il doit être possible de se déconnecter, sur toutes les pages (avec un lien de déconnexion).





Le bundle Security étant déjà installé, une classe User est créée et enregistrée en base données afin de mémoriser les utilisateurs qui auront des autorisations d'accès.



En suivant la documentation de Symfony (partie sécurité) on effectue ensuite les modifications nécessaire dans le fichier « security.yaml » afin de paramétrer l'authentification :

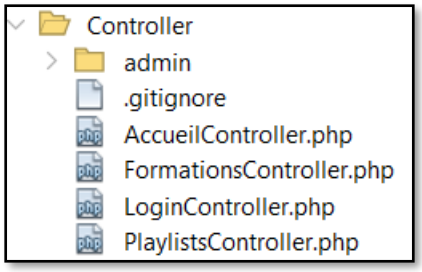
On précise ici que pour accéder au chemin « /admin », il faut avoir le « ROLE_ADMIN ».

```
access control:

    - { path: ^/admin, roles: ROLE_ADMIN }
```

Création d'un formulaire d'authentification en utilisant la commande suivante :

```
Php bin/console
make :controller Login
```

Création de LoginController.php dans le dossier Controller et de la page index.html.twig dans le dossier templates>login :



Au niveau du contrôleur la route permettant d'afficher le formulaire de connexion est définie comme suit :

```
class LoginController extends AbstractController
{
    #[Route('/login', name: 'app_login')]
```

Activation de l'authentification par formulaire :

Dans un premier temps, les paramètres d'authentification par formulaire doivent être activés dans le fichier « security.yaml ».Cette configuration fait référence à la route définie dans le controleur pour l'affichage du formulaire de connexion :

```
form_login:
    #app_login : nom de la route définie dans le controleur
    login_path: app_login
    check_path: app_login
    default_target_path: admin.formations
    enable_csrf: true
```

- **Login_path** définit la route vers laquelle l'utilisateur est dirigé pour accéder au formulaire de connexion. Elle correspond à la route définie dans le contrôleur, ici **app_login**.
- **Check_path** est la route où les informations du formulaire sont envoyées pour être vérifiées.
- **Default_target_path** indique la route vers laquelle l'utilisateur est redirigé après une authentification réussie, ici la page de gestion des formations (**admin.formations**).
- **Enable_csrf** active la protection CSRF pour sécuriser le formulaire. Il est donc nécessaire d'ajouter un champ de token en masqué dans la vue.

Modification du contrôleur pour gérer les erreurs d'authentification :

Dans un second temps, la modification du contrôleur est nécessaire afin d'afficher les éventuelles erreurs si l'utilisateur ne saisit pas des informations correctes et réafficher le dernier login.

Pour cela on utilise AuthentificationUtils fourni par Symfony afin de récupérer les informations liées à l'authentification :

- getLastAuthentificationError() : permet d'obtenir l'erreur d'authentification la plus récente, par exemple en cas de saisie d'un mot de passe ou login incorrect.
- getLastUsername() : récupère le dernier nom d'utilisateur saisi lors de la dernière tentative de connexion afin de le réafficher dans le formulaire.

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

Gestion de la vue **index.html.twig** :

- Le formulaire vérifie d'abord si des erreurs sont à afficher suite à une soumission incorrecte (comme défini dans le controleur).
- La valeur {{ last_username }} permet de préremplir la zone saisie avec le nom d'utilisateur lors d'une précédente soumission.

```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">Login:</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 %}
```

➢ Gestion de la déconnexion :

Pour gérer la déconnexion il est nécessaire d'ajouter une méthode qui récupère la route dans le contrôleur :

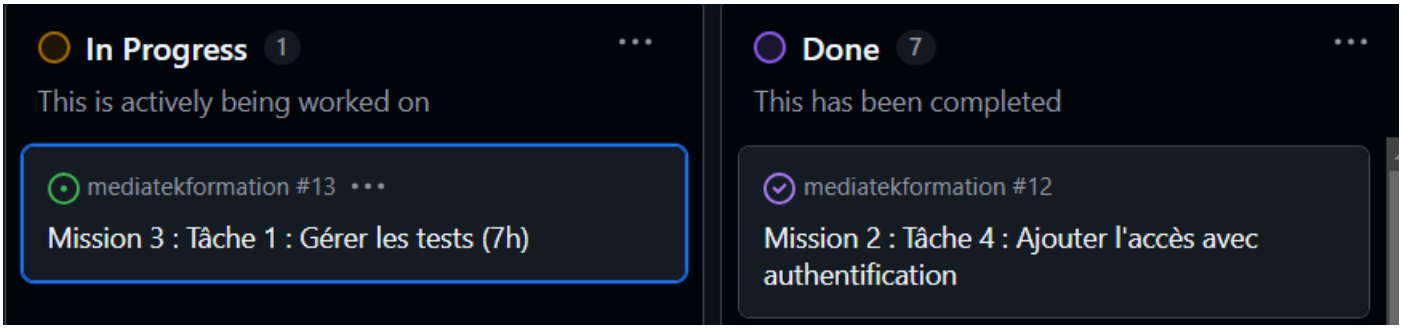
```php
//Gerer la deconnexion
#[Route('/logout',name:'logout')]
public function logout(){
```

Une fois la route fixée il faut configurer logout dans le firewall du fichier security afin que Symfony gère automatiquement la déconnexion de l'utilisateur :

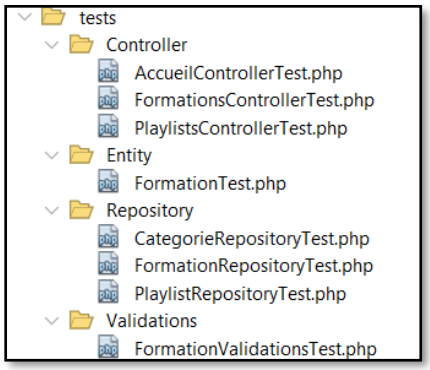
```yaml
logout:
    path: logout
```

MISSION 3 : TESTER ET DOCUMENTER

TACHE 1 : GERER LES TESTS



Pour réaliser les différents tests, j'ai mis en place un dossier tests contenant plusieurs sous-dossiers :



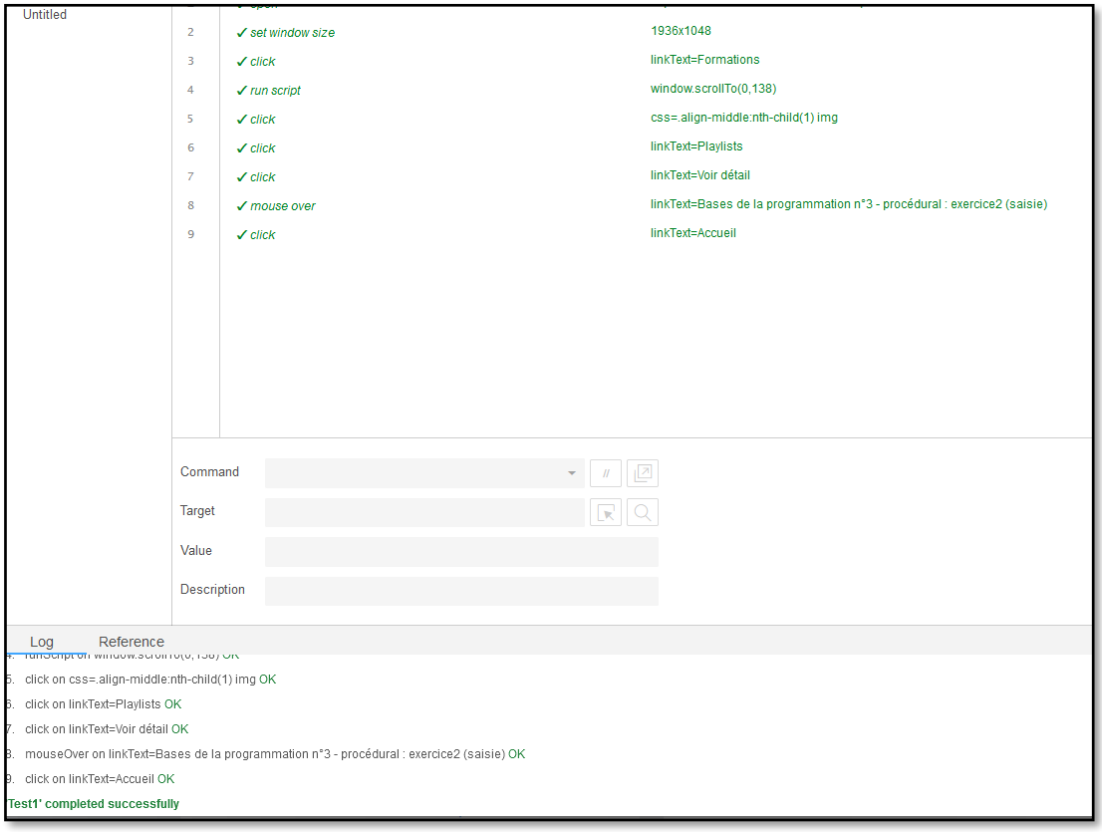
Controller : regroupe les tests fonctionnels. Les classes de test héritent de WebTestCase.

Entity : contient un test unitaire pour vérifier le bon fonctionnement de la méthode qui retourne la date de parution sous forme de chaîne. Ici, la classe de test hérite de TestCase.

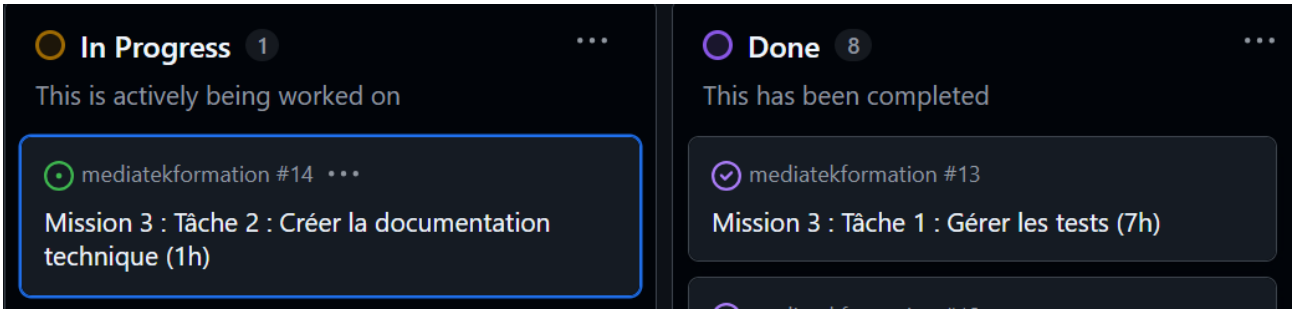
Repository : regroupe les tests d'intégration pour les méthodes des repositories Catégorie, Formation et Playlist. Ces tests nécessitant un accès au noyau de l'application, ils utilisent la classe KernelTestCase.

Validations : inclut un test d'intégration pour vérifier que lors de l'ajout ou de la modification d'une formation, la date de publication ne peut pas être postérieure à aujourd'hui. Ce test utilise également KernelTestCase.

Afin de tester la compatibilité avec Chrome et Firefox j'ai utilisé Selenium qui est un Framework de test permettant d'enregistrer un scénario et de le rejouer.

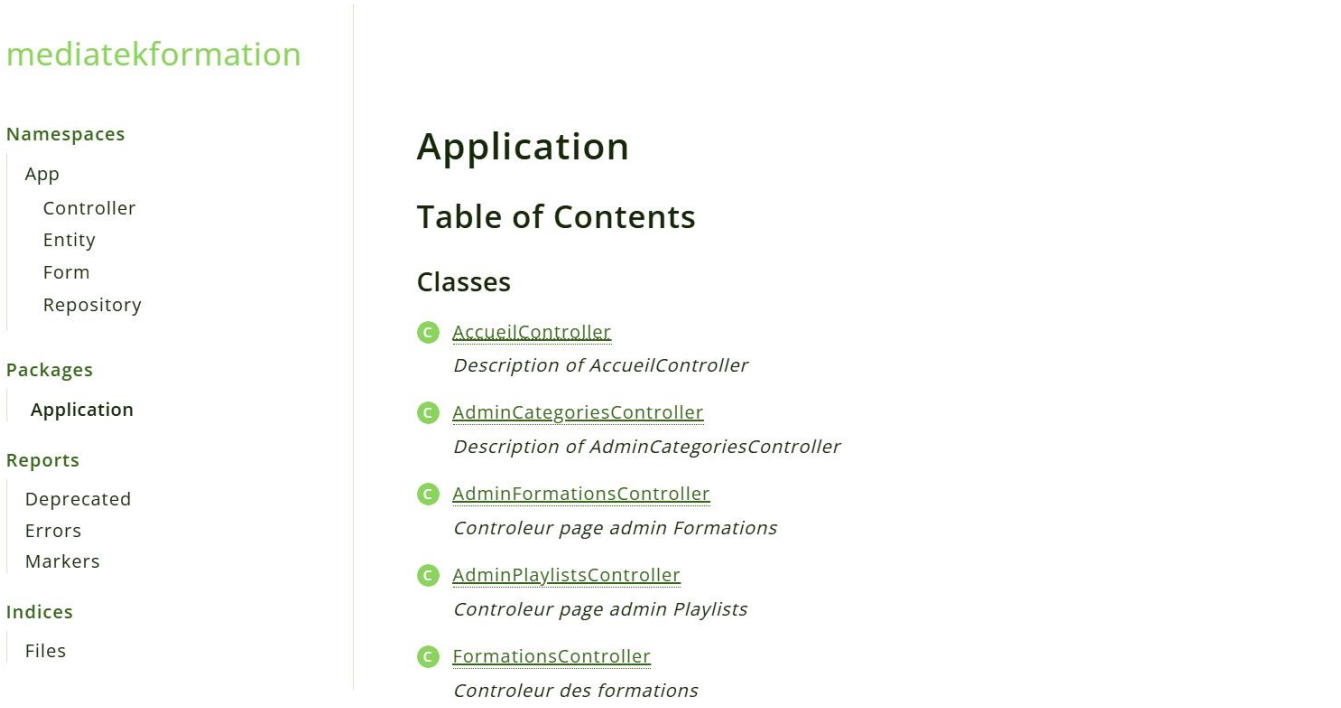


TACHE 2 : CREER LA DOCUMENTATION TECHNIQUE

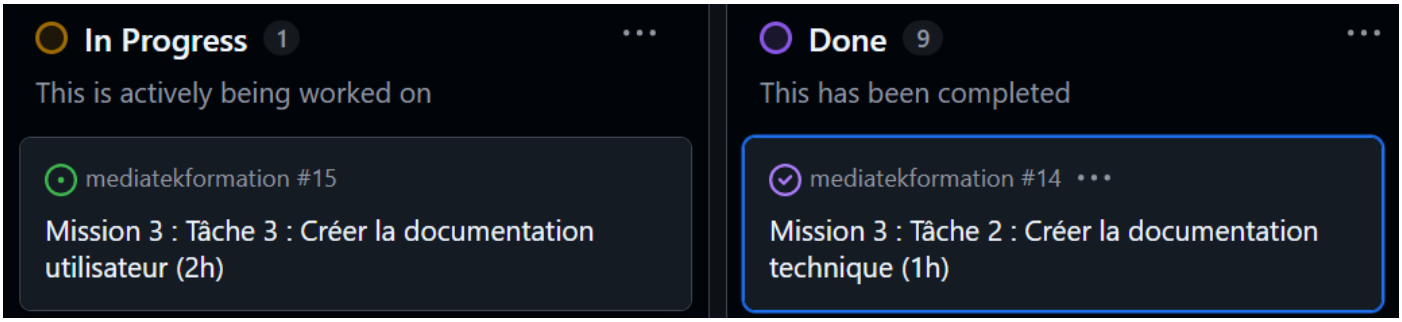


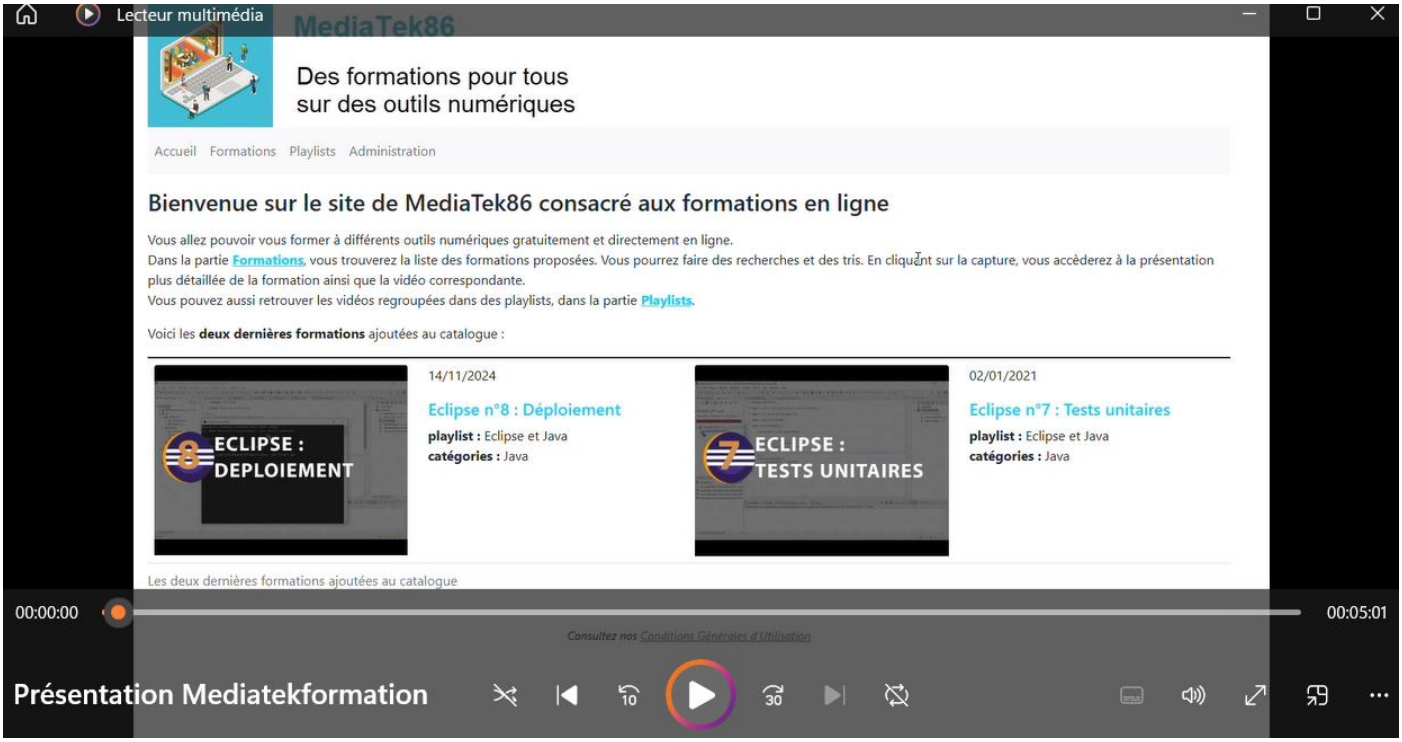
Afin de créer la documentation technique et ne prendre en compte que les classes ajoutées par le développeur j'ouvre une fenêtre de commande en mode admin :

```
"C:\wamp64\bin\php\php8.2.13\php.exe"
"C:\wamp64\bin\php\php8.2.13\ext\phpDocumentor.phar" "run" "--ansi" "--directory"
"C:/wamp64/www/mediatekformation/src" "--target"
"C:/wamp64/www/mediatekformation_doc" "--title" "mediatekformation"
```



TACHE 3 : CREER LA DOCUMENTATION UTILISATEUR



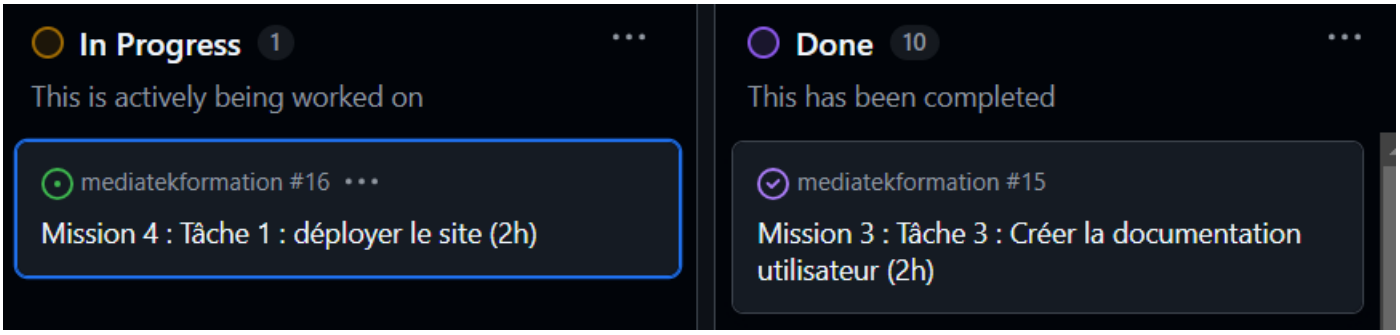


La vidéo de présentation de l’application est disponible sur la page dédiée à l’atelier professionnel du portfolio.

MISSION 4 : DEPLOYER LE SITE ET GERER LE DEPLOIEMENT CONTINU

TACHE 1 : DEPLOYER LE SITE

- 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.

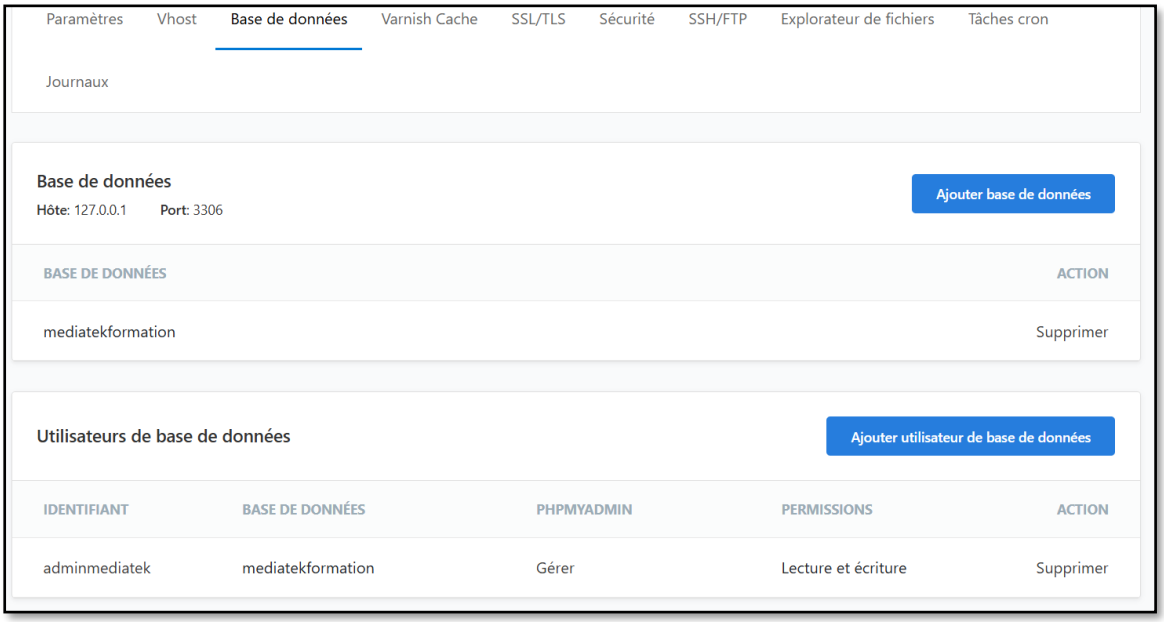


Pour le déploiement du site je me suis connectée au panel CloudPanel :

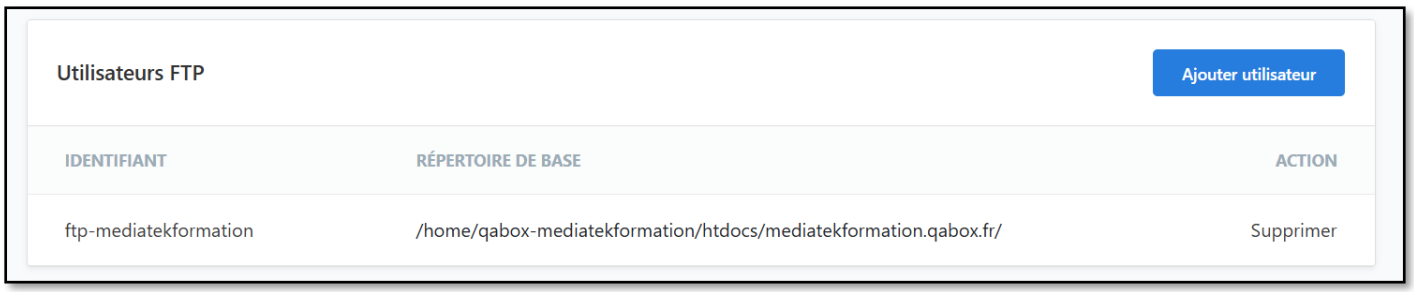
Qu'est-ce que **CloudPanel** ? C'est un panneau de contrôle d'hébergement web spécialement conçu pour faciliter la gestion des sites web, des applications et des serveurs dans un environnement cloud. Il s'agit d'un solution complète qui combine les fonctions de serveur, base de données, administration de domaine.

1. On commence par exporter la base de donnée locale « mediatek formation » au format SQL pour l'importe vers l'hébergeur. Un utilisateur de base donnée est également créé pour celle-ci.

2. Sur le panel qabox, on créé une nouvelle base de données puis on ajoute un utilisateur associé à celle-ci avec les permissions nécessaires.



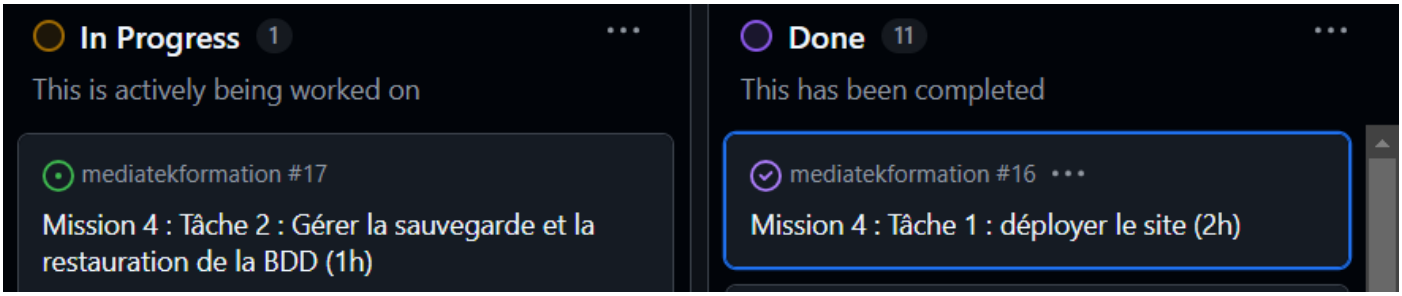
3. Configuration ftp : Afin de transférer les fichiers du site, on configure un accès FTP qui implique la création d'un compte utilisateur FTP ainsi que l'attribution d'un mot de passe pour celui-ci.



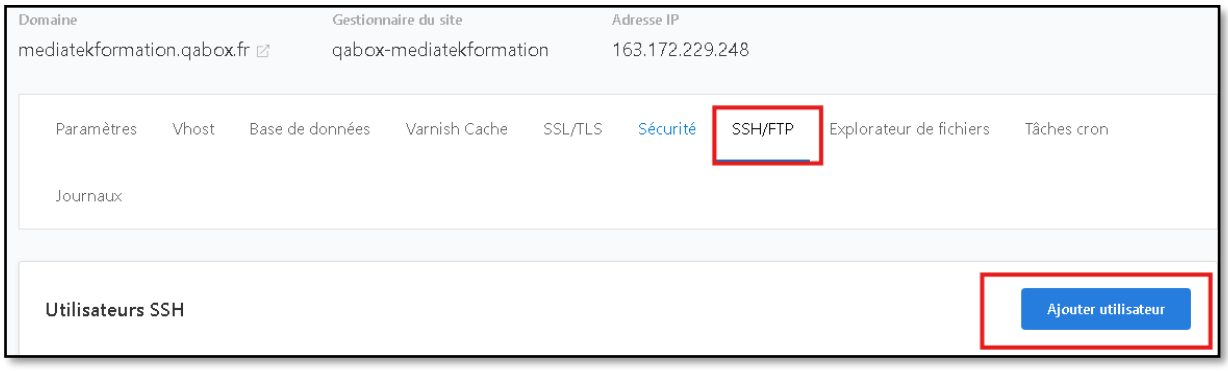
4. Transfert des fichiers via filezilla : On utilise le logiciel FileZilla pour transférer tous les fichiers du site depuis l'environnement local vers le serveur.

TACHE 2 : GERER LA SAUVEGARDE ET LA RESTAURATION DE LA BDD

- Une sauvegarde journalière automatisée doit être programmée pour la BDD
- La restauration pourra se faire manuellement, en exécutant le script de sauvegarde.

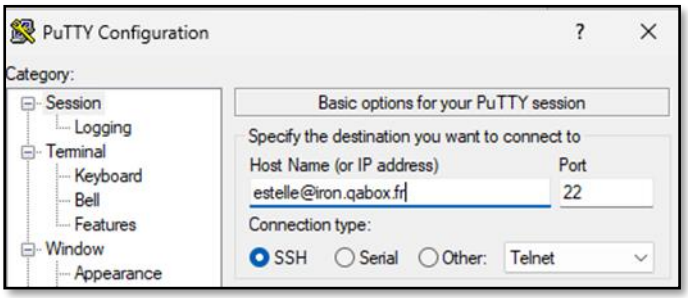


1. Créer un accès SSH via le panel Cloud Panel :

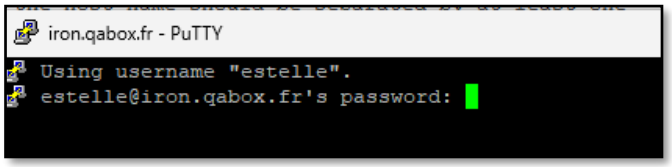


2. Télécharger le logiciel Putty qui est un client SSH pour pouvoir se connecter au serveur.

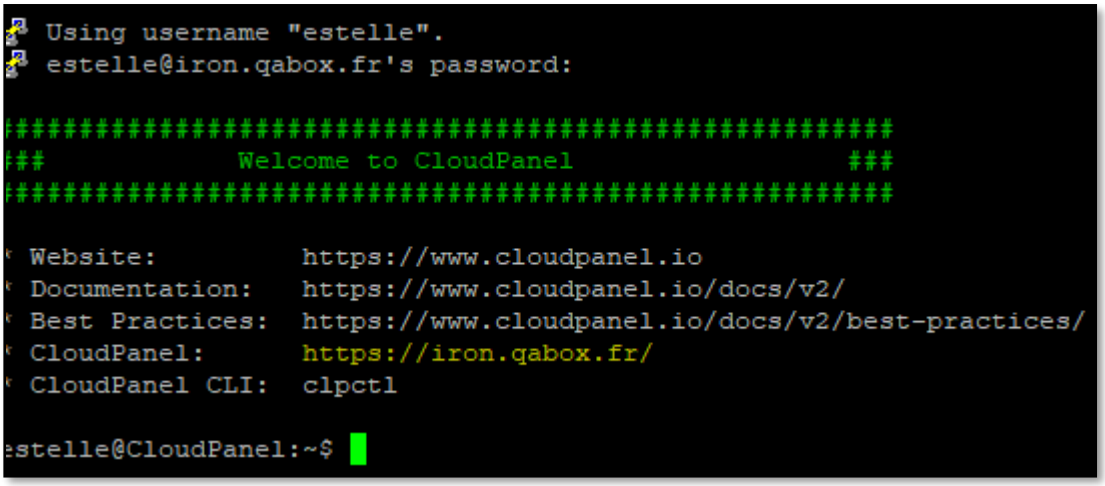
3. Dans « Host name » renseigner le nom d'hôte du serveur avec l'utilisateur : estelle@iron.qabox.fr



4. Renseigner le mot de passe indiqué lors de la création de l'utilisateur :



Nous sommes maintenant connectés au serveur et à notre compte en SSH :



5. Créer le fichier à l'aide de l'éditeur de texte « Nano » : nano backup.sh

Puis copier le script de backup :

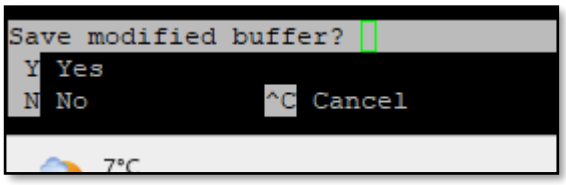
```
#!/bin/sh DATE=`date -I`
find /home/estelle/savebdd/bdd* -mtime -1 -exec rm {} \; clpctl db:export
--databaseName=mediatekformation --
file=/home/estelle/savebdd/bddbackup_${DATE}.sql.gz
```

Ici, nous n'utilisons pas mysqldump, mais la commande « clpctl» comme indiqué dans la documentation du panel :

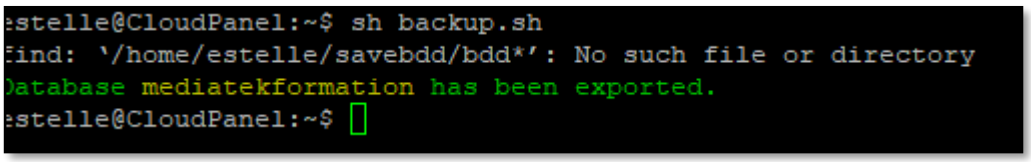
https://www.cloudpanel.io/docs/v2/cloudpanel-cli/site-user-commands/

Le panel dispose de son propre outil de backup.

Une fois le script copié, il faut enregistrer le fichier :

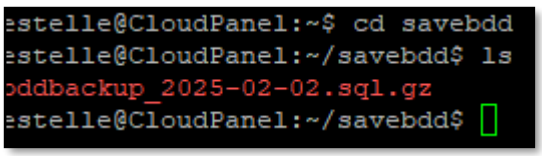


6. Le script enregistre le backup de la base de donnée dans le dossier « **savebdd**», mais ce fichier n'existe pas, il faut donc le créer à l'aide de la commande : « **mkdir savebdd**».



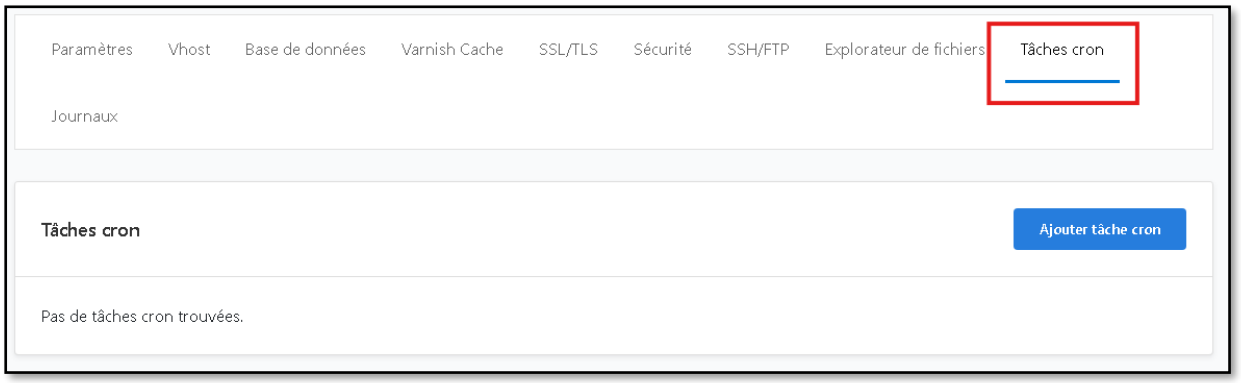
7. Ensuite le script peut être lancé à l'aide de la commande « sh backup.sh» :

8. Afin de vérifier que le backup a bien fonctionné, on se rends dans le dossier « **savebdd**» avec la commande « **cd savebdd** » . Ensuite on liste les fichiers avec la commande « ls » :



Automatisation de la tâche à l'aide d'un « cron » :

1. Se connecter au cloud panel et aller dans le menu « tâches cron » :



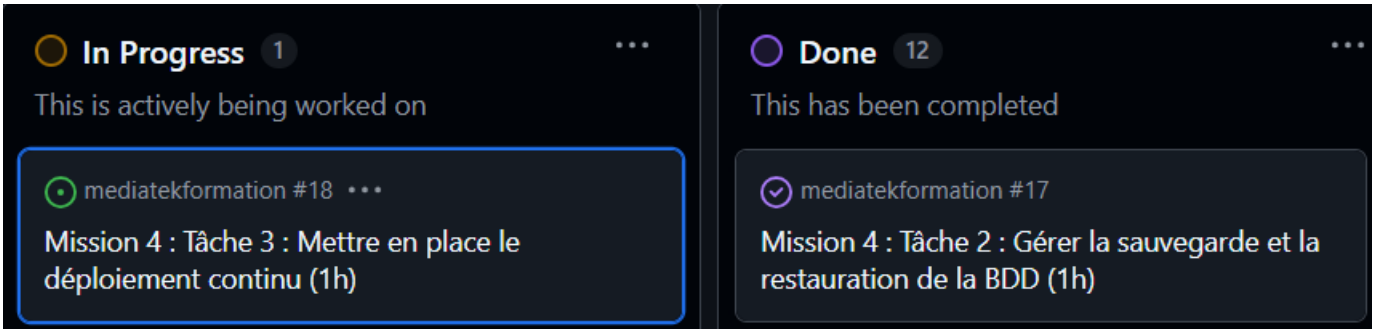
2. Cliquer sur ajouter une tâche cron :

Modèle Crontab → Choisir la fréquence d'exécution de la tâche.

Commande à exécuter → Indiquer le chemin du script précédemment créé.

TACHE 3 : METTRE EN PLACE LE DEPLOIEMENT CONTINU

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



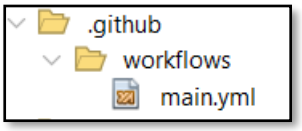
Afin de garantir que chaque push vers GitHub entraîne une mise à jour automatique du site en ligne, j'ai suivi plusieurs étapes :

1. Création du fichier yml d'automatisation : Sur GitHub dans l'onglet « Actions » → Set up a workflow yourself je créé le script de déploiement continu :

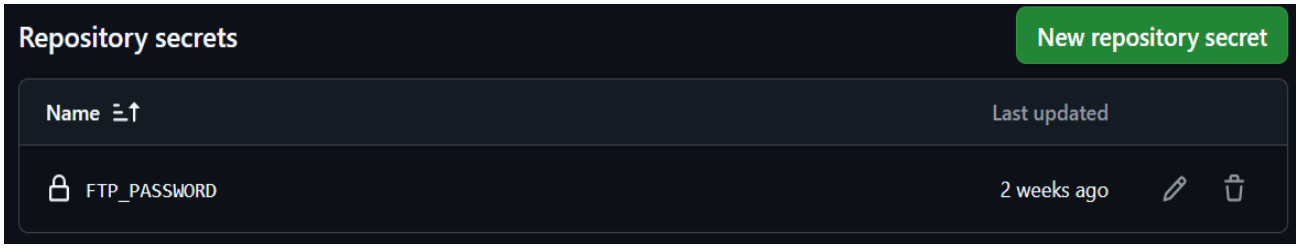
```yaml
on: push
name: Deploy website on push
jobs:
  web-deploy:
    name: Deploy
    runs-on: ubuntu-latest
    steps:
    - name: Get latest code
      uses: actions/checkout@v3

    - name: Sync files
      uses: SamKirkland/FTP-Deploy-Action@v4.3.4
      with:
        server: iron.qabox.fr
        server-dir:
        username: ftp-mediatekformation
        password: ${{ secrets.ftp_password }}
        timeout: 360000
        protocol: ftp
        port: 21
```

Après avoir rencontré plusieurs difficultés avec l'adresse du « server-dir », j'ai d'abord tenté d'utiliser l'adresse du répertoire de base du site. Cependant, cela entrainait la création d'un nouveau dossier contenant tous les fichiers au mauvais emplacement. Après plusieurs essais et ajustements, j'ai constaté que pour modifier correctement les fichiers existants, il fallait laisser l'adresse du server-dir vide.

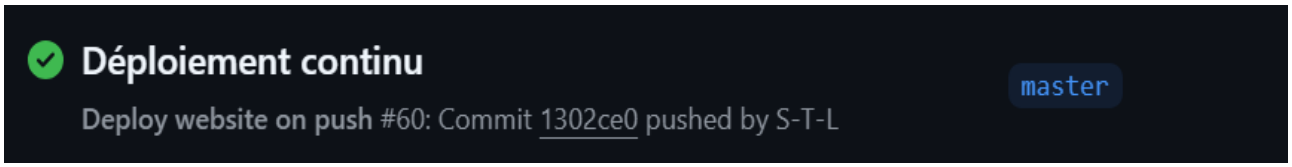
2. Une fois le script créé un « Commit changes » est effectué ce qui ajoute le dossier .github/workflows à la racine du dépôt, contenant le fichier YML nouvellement créé.



3. Dans le dépôt GitHub sous Secrets and variables → actions → new repository secret on renseigne le nom donné au fichier yml ainsi que la valeur correspondante, qui contient le mot de passe permettant d'accéder au ftp du site.



4. Test du déploiement continu : une modification mineure est effectuée sur une page HTML, suivie d'un commit et d'un push.
   La modification est bien prise en compte sur le site en ligne, et dans l'onglet « workflow run » de GitHub on peut vérifier que le processus à été exécuté et validé avec succès.



BILAN

Cet atelier m'a permis d'approfondir mes connaissances avec Symfony en explorant de nouvelles syntaxes et en recherchant des informations complémentaires. J'ai particulièrement apprécié la possibilité de travailler simultanément sur les aspects front end et back end ce qui m'a offert une vision plus globale d'un projet web.

De plus ce travail m'a aidé à améliorer mon organisation de travail. En effet sur ce projet j'ai d'abord focalisé mon travail sur l'aspect pratique et le code en reportant la rédaction du bilan à la fin. Cette approche m'a malheureusement fait perdre beaucoup de temps car j'ai du revenir sur les premières missions pour retracer exactement les méthodes que j'avais mises en place.

J'ai rencontré quelques difficultés à mettre en place le déploiement continu. L'erreur provenait d'une adresse server dir incorrecte ce qui m'a conduit à de nombreux essais infructueux avant d'identifier et corriger le problème.

Contexte : MediaTek86

Situation professionnelle : Symfony

Application : mediatekformation (site de mise à disposition des auto-formations).

ANNEXE 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 : 16/01/2025 09 : 20 : 00 Test avec une date nulle : null	La méthode doit retourner 16/01/2025 La méthode doit renvoyer une chaine vide	OK

TESTS D'INTEGRATION

But du test	Action de contrôle	Résultat attendu	Bilan
Lors de l'ajout ou de la modification d'une formation, contrôler que la date n'est pas postérieure à aujourd'hui.	Test avec une date dans le futur (+1 year) avec une erreur attendue	La validation échoue car il y a bien une erreur avec la date dans le futur	OK
FormationRepositoryTest			

Vérifier que le nombre d'enregistrements dans la table Formation est correct	Compter les enregistrements de la table Formation et le comparer avec le nombre attendu (237)	Nombre d'enregistrements présent en BDD : 237	OK
Vérifier que l'ajout d'une nouvelle formation avec add() mets bien à jour la base de donnée	Ajouter une formation et comparer le nombre total avant et après l'ajout	Le nombre total augmente de 1 après l'ajout de la formation	OK
Vérifier que la suppression d'une formation avec remove() met bien à jour la base de données	Ajouter une formation, puis la supprimer, et comparer le nombre total avant l'ajout et après la suppression	Le nombre total diminue de 1 après la suppression de la formation	OK
Vérifier le tri des formations par un champ en ordre ASC ou DESC	Appeler findAllOrderBy() avec un champ et un ordre, puis vérifier que le tri respecte l'ordre spécifié	ASC : "Android Studio (complément n°1) : Navigation Drawer et Fragment" DESC : "UML : Diagramme de paquetages"	OK
Vérifier le filtrage des formations par une valeur spécifique dans un champ	Appeler findByContainValue() sur le champ title avec la valeur "Eclipse" et compter le nombre de résultats	Résultat = 9 "Eclipse n°8 : Déploiement"	OK
Vérifier la récupération des formations les plus récentes	Test de findAllLasted avec la date "2025/01/17 00:00:00 "	2025/01/17 00:00:00	OK
Vérifier la récupération des formations d'une playlist	Test de findAllForOnePlaylist() avec l'id 2	"C# : présentation des objets graphiques"	OK
Vérifier que le nombre d'enregistrements dans la table Playlist est correct	Compter les enregistrements de la table Playlist et le comparer avec le nombre attendu (27)	Nombre d'enregistrements présent en BDD : 27	OK

Vérifier que l'ajout d'une nouvelle playlist avec add() met bien à jour la base de données	Ajouter une playlist et comparer le nombre total avant et après l'ajout	Le nombre total augmente de 1 après l'ajout de la playlist	OK
Vérifier que la suppression d'une playlist avec remove() met bien à jour la base de données	Ajouter une playlist, puis la supprimer, et comparer le nombre total avant l'ajout et après la suppression	Le nombre total diminue de 1 après la suppression de la playlist	OK
Vérifier le tri des playlists par nom en ordre	Test de findAllOrderByName() avec "DESC"	'Visual Studio 2019 et C#'	OK

TESTS FONCTIONNELS

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler que la page d'accès est accessible	Test fonctionnel : assertResponseStatusCodeSame(Response::HTTP_OK)	HTTP_OK	OK
Contrôler que la page des formations est accessible	assertResponseStatusCodeSame(Response::HTTP_OK);	HTTP_OK	OK
PAGE FORMATIONS			

Vérifier le tri ASC selon le nom des playlists	Test avec un ordre ASC	Android Studio (complément n°1) : Navigation Drawer et Fragment'	OK
Vérifier le tri DESC selon le nom des playlists	Test avec un ordre DESC	C# : ListBox en couleur	OK
Vérifier le tri ASC sur le titre des formations	Test avec un ordre ASC	Android Studio (complément n°1) : Navigation Drawer et Fragment'	OK
Vérifier le tri DESC sur le titre des formations	Test avec un ordre DESC	UML : Diagramme de paquetages	OK
Vérifier le tri ASC sur la date de publication de la formation	Test avec un ordre ASC	Cours UML (8 à 11 / 33) : diagramme de classes	OK
Vérifier le tri DESC sur la date de publication de la formation	Test avec un ordre DESC	Test Formation	OK
Vérifier le filtre sur les formations	Test avec Python	Python n°18 : Décorateur singleton	OK
Vérifier le filtre sur les playlists	Test avec MCD	MCD exercice 18 : sujet 2006 (cas Credauto)	OK
Vérifier le filtre sur les catégories	Test avec id catégorie 2 (UML)	Eclipse n°2 : rétroconception avec ObjectAid'	OK
Vérifier que le lien vers les détails d'une formation fonctionne	Test du clic sur l'image de la première formation	/formations/formation/1	OK

Contrôler que la page des playlists est accessible	assertResponseStatusCodeSame(Response::HTTP_OK)	HTTP_OK	OK
Vérifier le tri Asc selon le nom des playlists	Test avec un ordre ASC	Bases de la programmation (C#)	OK
Vérifier le tri DESC selon le nom des playlists	Test avec un ordre DESC	Visual Studio 2019 et C#	OK
Vérifier le tri ASC selon le nombre de formations	Test avec un ordre ASC	Cours Informatique embarquée	OK
Vérifier le tri DESC selon le nombre de formations	Test avec un ordre DESC	Bases de la programmation (C#	OK
Vérifier le filtre sur les playlists	Test avec programmation	Bases de la programmation (C#)	OK
Vérifier le filtre sur les catégories	Test avec la catégorie id 8 (SQL)	Cours Curseurs	OK
Vérifier que le lien vers le détail d'une playlist fonctionne	Test du clic sur le lien -> Voir le détail	/playlists/playlist/13	OK

TESTS DE COMPATIBILITE

But du test	Action de contrôle	Résultat attendu	Bilan
Vérifier la compatibilité sur firefox et chrome	Création d'un scénario avec Selenium	Aucune erreur	OK