

Atelier 1

Application Web MediaTek86 (Symfony)

Table des matières

Mission 1.....	2
Tâche 1 : nettoyer le code.....	2
Tâche 2 : ajouter une fonctionnalité.....	4
Mission 2.....	7
Tâche 1 : Gérer les formations.....	7
Tâche 2 : Gérer les Playlists.....	14
Tâche 3 : gérer les catégories.....	20
Tâche 4 : ajouter l'accès avec authentification.....	24
Mission 3.....	30
Tâche 1 : gérer les tests.....	30
Tâche 2 : Générer la documentation technique.....	35
Tâche 3 : créer la documentation utilisateur.....	36
Mission 4 :.....	37
Tâche 1 : déployer le site.....	37
Tâche 2 : gérer la sauvegarde et la restauration de la BDD.....	38
Tâche 3 : mettre en place le déploiement continu.....	40

1. Contexte du projet

La société InfoTech Services 86 a été mandatée par le réseau de médiathèques **MediaTek86** (département de la Vienne). L'objectif est de faire évoluer une application web existante permettant aux usagers d'accéder à des vidéos d'auto-formation en ligne.

2. État actuel

Au démarrage de la mission, l'application était développée en **Symfony** . Elle permettait uniquement la consultation (Front-office) des formations, playlists et catégories. Aucune interface d'administration (Back-office) n'existait, rendant la mise à jour des données impossible pour le personnel de la médiathèque.

3. Technologies utilisées

Langage & Framework : PHP 8.2, Symfony 6.4 (Framework MVC).

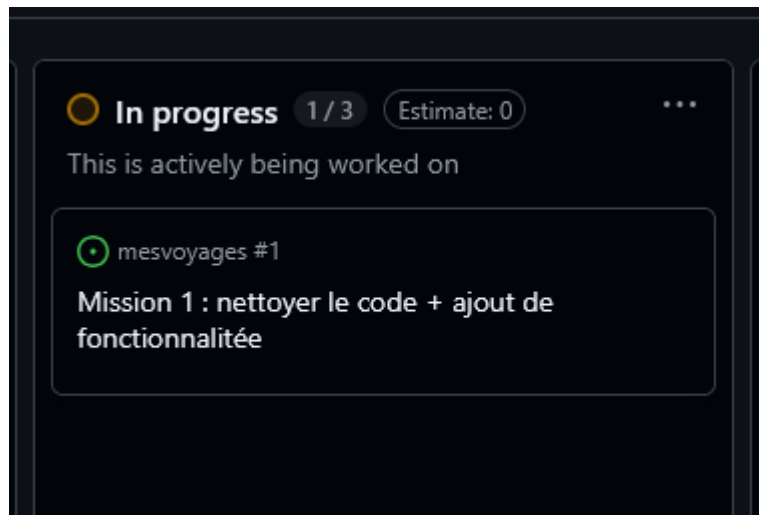
- **Base de données :** MySQL (MariaDB), gérée via l'ORM Doctrine.
- **Front-end :** Twig (Moteur de template), Bootstrap (CSS).
- **Outils de développement :** NetBeans (IDE), Git/GitHub (Versionning), WampServer (Serveur local).
- **Qualité & Tests :** SonarLint (Analyse statique), PHPUnit (Tests unitaires et fonctionnels).
- **Déploiement :** InfinityFree (Hébergement LAMP), GitHub Actions (Intégration continue).

Mission 1

Tâche 1 : nettoyer le code

Bilan :

Nettoyage du code jusqu'à que Sonarlint ne signale plus de problèmes



- ▼ ⚠ major (1 issue)
- ▼ ⚠ php:S1066 : Collapsible "if" statements should be merged (1)
 - 📄 87:12: Playlist.php

Code actuel :

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

Corrigé : suppression du deuxième if remplacé par &&

```
{  
    if ($this->formations->removeElement($formation) && $formation->getPlaylist() === $this) {  
        $formation->setPlaylist(null);  
    }  
}
```


- ▼ ⚠ php:S3973 : A conditionally executed single line should be denoted by indentation (1)
 - 📄 103:12: Playlist.php


Code actuel :

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

Corrigé : Correction de l'indentation et ajout d'un {

```
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;
}
```

✓  php:S131 : "switch" statements should have "default" clauses (1)


 61:8: PlaylistsController.php


Code actuel :

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

Corrigé: Ajout d'une clause default

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

✓  php:S115 : Constant names should comply with a naming convention (1)

 18:18: Formation.php


Code actuel :


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


Corrigé :

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

Il faut également changer les méthodes pour qu'elle ai ce nom

▼  php:S1192 : String literals should not be duplicated (2)

 39:29: FormationsController.php

 53:29: PlaylistsController.php

ajout d'une constante qu'on va réutiliser

```
private const PAGE_PLAYLISTS = "pages/playlists.html.twig";
```

Et remplacement de

```
return $this->render("pages/playlists.html.twig", [
```

Par

```
return $this->render(self::PAGE_PLAYLISTS, [
```

Même principe dans FormationsController.php

```
private const PAGE_FORMATIONS = "pages/formations.html.twig";
```

```
return $this->render("pages/formations.html.twig", [
```

```
return $this->render(self::PAGE_FORMATIONS, [
```

Tâche 1 : nettoyer le code (2h)	Temps réel = 1h30
--	-------------------

Tâche 2 : ajouter une fonctionnalité

Ajout d'une colonne pour afficher le nombre de formations par playlist et permettre le tri croissant et décroissant sur cette colonne.

Pour faire ça : Ajout d'une fonction dans PlaylistRepository pour récupérer les données requises dans la BDD. Ici, les playlist et leurs formations associées.

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

Ensuite dans PlaylistController modification de la méthode sort pour gérer le nouveau cas.

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

Ensuite dans le fichier playlists.html.twig ajout des deux boutons de tri

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

Dans le même fichier il faut également mettre dans la boucle du tbody l'affichage du nombre en utilisant |length pour compter le nombre d'éléments

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

Et pour finir dans playlist.html.twig ajout de l'affichage du nombre sur la page de détail.

```
<strong>{{ playlist.formations|length }} formations</strong><br>
```

Résultat :

playlist	catégories	Nombre de formations	
< >		< >	
<input type="text"/>	<input type="text"/>		<input type="button" value="filtrer"/>
Bases de la programmation (C#)	C# POO	74	<input type="button" value="Voir détail"/>
Programmation sous Python	Python POO	19	<input type="button" value="Voir détail"/>
MCD : exercices progressifs	MCD	18	<input type="button" value="Voir détail"/>
TP Android (programmation mobile)	Android SQL Java	18	<input type="button" value="Voir détail"/>
Compléments Android (programmation mobile)	Android	13	<input type="button" value="Voir détail"/>
Visual Studio 2019 et C#	C# POO	11	<input type="button" value="Voir détail"/>
Cours UML	UML Cours	10	<input type="button" value="Voir détail"/>
Eclipse et Java	Java UML	8	<input type="button" value="Voir détail"/>
Exercices objet (sujets EDC BTS SIO)	POO	8	<input type="button" value="Voir détail"/>
MCD exercices d'examen (sujets EDC BTS SIO)	MCD	8	<input type="button" value="Voir détail"/>

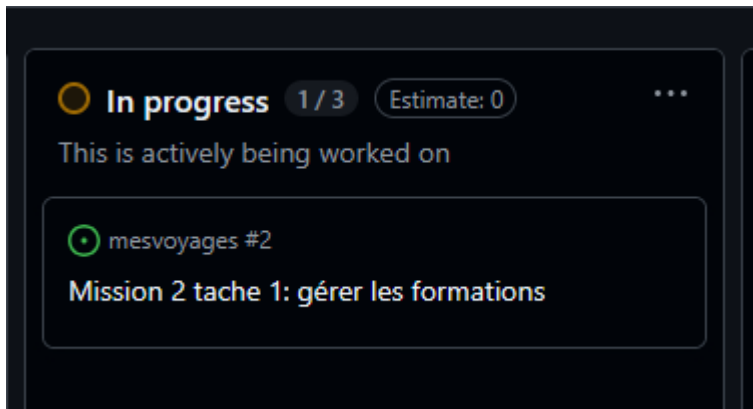
playlist	catégories	Nombre de formations	
< >		< >	
<input type="text"/>	<input type="text"/>		<input type="button" value="filtrer"/>
playlist test		0	<input type="button" value="Voir détail"/>
Cours Informatique embarquée	Cours	1	<input type="button" value="Voir détail"/>
Cours Merise/2	MCD Cours	1	<input type="button" value="Voir détail"/>
Cours Modèle relationnel et MCD	MCD Cours	1	<input type="button" value="Voir détail"/>
Cours de programmation objet	POO Cours	1	<input type="button" value="Voir détail"/>
Cours Composant logiciel	Cours	2	<input type="button" value="Voir détail"/>
Cours MCD MLD MPD	MCD Cours	2	<input type="button" value="Voir détail"/>
Cours MCD vs Diagramme de classes	MCD Cours	2	<input type="button" value="Voir détail"/>
Cours Curseurs	SQL Cours POO	2	<input type="button" value="Voir détail"/>
Sujet E5 SLAM 2018 Nouméa : cas LOCALUX	POO SQL	3	<input type="button" value="Voir détail"/>
Cours Transactions et verrou	SQL Cours	2	<input type="button" value="Voir détail"/>

Tâche 2 : ajouter une fonctionnalité (2h)

Temps réel =2h30

Mission 2

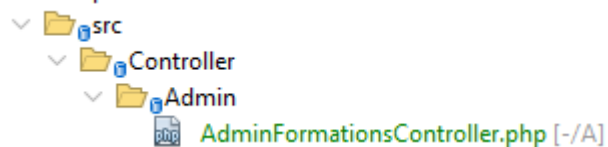
Tache 1 : Gérer les formations



Création d'un contrôleur spécifique pour l'admin

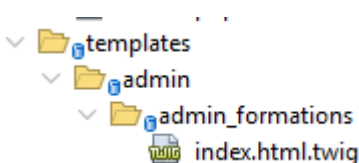
php bin/console make:controller Admin\AdminFormationsController

Ce qui crée



src/Controller/Admin/AdminFormationsController.php

Ainsi que



templates/admin/admin_formation/index.html.twig

Il faut ensuite coder l'accès à la base de données ainsi que la méthode pour afficher la page dans AdminFormationsController. Pour ça on copie le code présent dans FormationsController en changeant la route.


```

class AdminFormationsController extends AbstractController
{
    /**
     * @var FormationRepository
     */
    private $formationRepository;

    /**
     * @var CategorieRepository
     */
    private $categorieRepository;

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

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

        return $this->render('admin/admin_ formations/index.html.twig', [
            'formations' => $formations,
            'categories' => $categories
        ]);
    }
}

```

On copie également le contenu de la vue dans formations.html.twig vers index.html.twig

Ajout dans index.html.twig de la colonne et des boutons pour que l'admin puisse éditer et supprimer des formations.

```

</td>
<th class="text-center align-top" scope="col">
    actions
</th>

</td>
<td class="text-center">
    <button class="btn btn-secondary">Editer</button>
    <button class="btn btn-danger">Supprimer</button>
</td>

```

Suppression

Ensuite il faut coder la fonction de suppression des formations dans AdminFormationController.php avec un jeton CSRF pour la sécurité.

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

    if ($this->isCsrfTokenValid('delete'.$formation->getId(), $request->get('_token'))) {
        $this->formationRepository->remove($formation);
        $this->addFlash('success', 'La formation a été supprimée avec succès');
    }

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

Puis dans la vue on ajoute une demande de validation de la suppression lorsque l'on clique sur supprimer avec le token pour sécuriser.

```
<td class="text-center">
    <a href="#" class="btn btn-secondary btn-sm">Editer</a>
    <form method="POST" action="{{ path('admin.formation.suppr', {id: formation.id}) }}"
        onsubmit="return confirm('Êtes-vous sûr de vouloir supprimer cette formation ?');"
        style="display: inline-block;">
        <input type="hidden" name="_token" value="{{ csrf_token('delete' ~ formation.id) }}">
        <button type="submit" class="btn btn-danger btn-sm">Supprimer</button>
    </form>
</td>
```

On peut maintenant supprimer les formations depuis l'interface admin.

Modification et ajout

Pour l'ajout et la modification des formations je vais créer une classe FormType pour gérer les formulaires.

php bin/console make:form FormationType Formation

Cette commande crée le fichier src/Form/FormationType.php qui contient le code suivant.

```
<?php

namespace App\Form;

use App\Entity\Categorie;
use App\Entity\Formation;
use App\Entity\Playlist;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class FormationType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('publishedAt', null, [
                'widget' => 'single_text',
            ])
            ->add('title')
            ->add('description')
            ->add('videoId')
            ->add('playlist', EntityType::class, [
                'class' => Playlist::class,
                'choice_label' => 'id',
            ])
            ->add('categories', EntityType::class, [
                'class' => Categorie::class,
                'choice_label' => 'id',
                'multiple' => true,
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Formation::class,
        ]);
    }
}
```

Pour respecter les contraintes il va falloir modifier plusieurs parties.

- Ajout des champs adaptés (DateType pour le calendrier, EntityType pour les listes déroulantes)
- Pour la date de parution ajout d'une contrainte de validation LessThanOrEqual('today') pour empêcher la saisie d'une date future.
- Utilisation de EntityType sur l'entité Playlist pour afficher une liste déroulante des noms des playlists (via choice_label => 'name').
- Configuration du champ en multiple => true pour permettre la sélection de plusieurs catégories, tout en autorisant le champ vide (required => false).

Résultat :

```
class FormationType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('title', null, ['label' => 'Titre'])
            ->add('description', null, ['required' => false, 'label' => 'Description'])
            ->add('publishedAt', DateTime::class, [
                'widget' => 'single_text',
                'label' => 'Date',
                'constraints' => [new LessThanOrEqual(['value' => 'today', 'message' => 'Pas de date future'])]
            ])
            ->add('videoId', null, ['label' => 'ID Vidéo'])
            ->add('playlist', EntityType::class, [
                'class' => Playlist::class,
                'choice_label' => 'name'
            ])
            ->add('categories', EntityType::class, [
                'class' => Categorie::class,
                'choice_label' => 'name',
                'multiple' => true,
                'required' => false
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults(['data_class' => Formation::class]);
    }
}
```

Ajout de deux routes dans AdminFormationsController avec utilisation de la méthode `$form->handleRequest($request)` qui permet à Symfony de récupérer les données POST et de remplir automatiquement l'objet Formation.

```
#[Route('/admin/formations/ajout', name: 'admin.formation.ajout')]
public function ajout(Request $request): Response
{
    $formation = new Formation();
    $form = $this->createForm(FormationType::class, $formation);

    $form->handleRequest($request);
    if($form->isSubmitted() && $form->isValid()){
        $this->formationRepository->add($formation);
        return $this->redirectToRoute('admin.formations');
    }

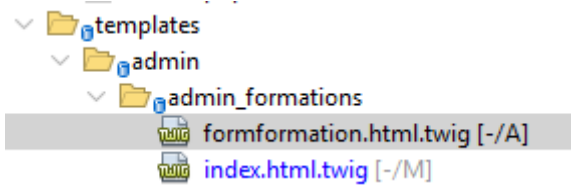
    return $this->render('admin/admin_formation/form.formation.html.twig', [
        'formformation' => $form->createView()
    ]);
}

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

    $form->handleRequest($request);
    if($form->isSubmitted() && $form->isValid()){
        $this->formationRepository->add($formation);
        return $this->redirectToRoute('admin.formations');
    }

    return $this->render('admin/admin_formation/form.formation.html.twig', [
        'formformation' => $form->createView()
    ]);
}
```

Pour finir création d'un fichier Twig `form.formation.html.twig` pour l'affichage des ajouts et modifications.



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

{% block body %}
<div class="container mt-4">
    <h2>
        {% if app.request.attributes.get('_route') == 'admin.formation.ajout' %}
            Ajouter une formation
        {% else %}
            Modifier une formation
        {% endif %}
    </h2>


    {{ form_start(formformation) }}

    <div class="row">
        <div class="col-md-6">
            {{ form_row(formformation.title) }}
            {{ form_row(formformation.playlist) }}
            {{ form_row(formformation.categories) }}
            {{ form_row(formformation.publishedAt) }}
            {{ form_row(formformation.videoId) }}
        </div>
        <div class="col-md-6">
            {{ form_row(formformation.description, {'attr': {'rows': 10}}) }}
        </div>
    </div>

    <div class="mt-3">
        <button type="submit" class="btn btn-primary">Enregistrer</button>
    </div>

    {{ form_end(formformation) }}
</div>
{% endblock %}
```

Résultat :



MediaTek86
Des formations pour tous
sur des outils numériques

Accueil Formations Playlists

Ajouter une formation

Titre

Playlist

Eclipse et Java

Java

UML

C#

Categories

Python

Date

jj/mm/aaaa

ID Vidéo

Description

Enregistrer

[Consultez nos Conditions Générales d'Utilisation](#)



MediaTek86
Des formations pour tous
sur des outils numériques

Accueil Formations Playlists

Modifier une formation

Titre

Eclipse n°8 : Déploiement

Playlist

Eclipse et Java

Java

UML

C#

Categories

Python

Date

04/01/2021

ID Vidéo

Z4yTTXka958

Description

Exécution de l'application en dehors de l'IDE, en invite de commande.
Création d'un fichier jar pour le déploiement de l'application.
00:20 : exécuter

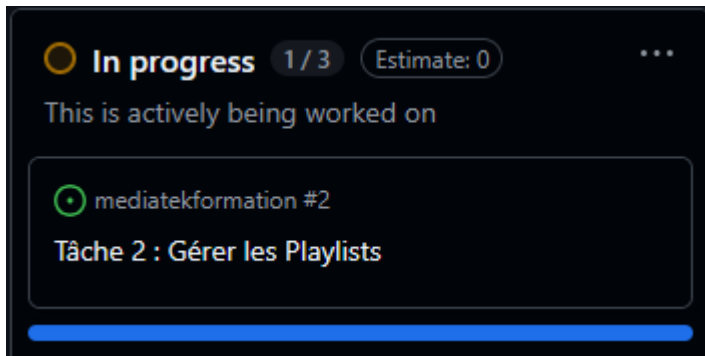
Enregistrer

[Consultez nos Conditions Générales d'Utilisation](#)

Tâche 1 : gérer les formations (5h)

Temps réel : 8-9h

Tâche 2 : Gérer les Playlists



Tout d'abord pour gérer les formations on crée un contrôleur dédié.

php bin/console make:controller "Admin\AdminPlaylistsController"

Ce qui crée src/Controller/Admin/AdminPlaylistsController.php.

Suppression

Premièrement nous allons nous occuper du code pour la suppression d'une playlist. En respectant les contraintes imposées.

```
class AdminPlaylistsController extends AbstractController
{
    private $playlistRepository;
    private $categorieRepository;

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

    #[Route('/admin/playlists', name: 'admin.playlists')]
    public function index(): Response
    {
        $playlists = $this->playlistRepository->findAllOrderByName('ASC');
        $categories = $this->categorieRepository->findAll();

        return $this->render('admin/admin_playlists/index.html.twig', [
            'playlists' => $playlists,
            'categories' => $categories
        ]);
    }

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

        if ($this->isCsrfTokenValid('delete.'.$playlist->getId(), $request->get('_token'))) {
            if (count($playlist->getFormations()) > 0) {
                $this->addFlash('danger', 'Vous ne pouvez pas supprimer cette playlist car elle contient des formations.');
```

Création du constructeurs qui accède a Playlist et Catégorie et de l'index de la route. La fonction de suppression vérifie le token CSRF puis il vérifie avec `if (count($playlist->getFormations()) > 0)` si la playlist est vide ou non. Si elle l'est elle la supprime sinon non puis recharge la page.

Affichage dans la vue dans `templates/admin/admin_playlists/index.html.twig`

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

[ {% block body %}
[   <div class="container mt-4">
[       <h2>Gestion des playlists</h2>

[       <a href="#" class="btn btn-info mb-3">Ajouter une playlist</a>

[       {% for message in app.flashes('success') %}
[       <div class="alert alert-success mt-3">
[           {{ message }}
[       </div>
[       {% endfor %}

[       {% for message in app.flashes('danger') %}
[       <div class="alert alert-danger mt-3">
[           {{ message }}
[       </div>
[       {% endfor %}

[       <table class="table table-striped">
[       <thead>
[       <tr>
[           <th>Playlist</th>
[           <th>Catégories</th>
[           <th class="text-center">Formations</th>
[           <th class="text-center">Actions</th>
[       </tr>
[       </thead>
```

Utilisation de « `for message in app.flashes` » pour afficher une barre verte ou rouge.

Vous ne pouvez pas supprimer cette playlist car elle contient des formations.

Et utilisation d'une boucle pour afficher les données envoyées par le controlleur avec demande de confirmation lors d'une tentative de suppression.

```
<tbody>
  {% for playlist in playlists %}
    <tr class="align-middle">
      <td>
        <h5 class="text-info">{{ playlist.name }}</h5>
      </td>
      <td>
        {% for categorie in playlist.categoriesplaylist %}
          {{ categorie }}
        {% endfor %}
      </td>
      <td class="text-center">
        {{ playlist.formations|length }}
      </td>
      <td class="text-center">
        <a href="#" class="btn btn-secondary btn-sm">Editor</a>

        <form method="POST" action="{{ path('admin.playlist.suppr', {id: playlist.id}) }}"
          onsubmit="return confirm('Êtes-vous sûr ?');"
          style="display: inline-block;">
          <input type="hidden" name="_token" value="{{ csrf_token('delete' ~ playlist.id) }}">
          <button type="submit" class="btn btn-danger btn-sm">Supprimer</button>
        </form>
      </td>
    </tr>
  {% endfor %}
</tbody>
</table>
</div>
{% endblock %}
```

Ajout et Modification

Premièrement création d'un formulaire FormType.

php bin/console make:form PlaylistType Playlist

Ce qui crée src/Form/PlaylistType.php

```
class PlaylistType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name', TextType::class, [
                'label' => 'Nom de la playlist',
                'required' => true
            ])
            ->add('description', TextareaType::class, [
                'label' => 'Description',
                'required' => false,
                'attr' => ['rows' => 5]
            ]);
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Playlist::class,
        ]);
    }
}
```

Dans ce fichier il faut configurer les champs pour respecter les contraintes donc en rendant le name required => true et le champ description comme facultatif.

Ensuite dans le controlleur AdminPlaylistsController il faut ajouter la méthode d'ajout et de modification (edit).

```
#[Route('/admin/playlists/ajout', name: 'admin.playlist.ajout')]
public function ajout(Request $request): Response
{
    $playlist = new Playlist();
    $form = $this->createForm(PlaylistType::class, $playlist);

    $form->handleRequest($request);
    if($form->isSubmitted() && $form->isValid()){
        $this->playlistRepository->add($playlist);
        return $this->redirectToRoute('admin.playlists');
    }

    return $this->render('admin/admin_playlists/formplaylist.html.twig', [
        'formplaylist' => $form->createView()
    ]);
}
```

Elle crée une nouvelle playlist vide puis la valide et l'ajoute dans la BDD via le repository

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

    $form->handleRequest($request);
    if($form->isSubmitted() && $form->isValid()){
        $this->playlistRepository->add($playlist);
        return $this->redirectToRoute('admin.playlists');
    }

    return $this->render('admin/admin_playlists/formplaylist.html.twig', [
        'formplaylist' => $form->createView(),
        'playlist' => $playlist
    ]);
}
```

La méthode edit récupère l'ID de la playlist existante et la passe à la vue via la méthode render.

Pour la vue création d'un fichier twig formplaylist.html.twig dans templates/admin/admin_playlists/formplaylist.html.twig

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

{% block body %}
<div class="container mt-4">
    <h2>
        {% if app.request.attributes.get('_route') == 'admin.playlist.ajout' %}
            Ajouter une playlist
        {% else %}
            Modifier une playlist
        {% endif %}
    </h2>

    {{ form_start(formplaylist) }}

    <div class="row">
        <div class="col-md-6">
            {{ form_row(formplaylist.name) }}
        </div>
        <div class="col-md-6">
            {{ form_row(formplaylist.description) }}
        </div>
    </div>

    <div class="mt-3">
        <button type="submit" class="btn btn-primary">Enregistrer</button>
    </div>
</div>
```

```

{{ form_end(formplaylist) }}
{% if playlist is defined and playlist. formations|length > 0 %}
    <hr class="mt-5">
    <h4 class="text-info">Formations dans cette playlist (Lecture seule)</h4>

    <table class="table table-striped mt-3">
        <thead>
            <tr>
                <th>Titre de la formation</th>
                <th>Date</th>
            </tr>
        </thead>
        <tbody>
            {% for formation in playlist. formations %}
                <tr>
                    <td>{{ formation.title }}</td>
                    <td>{{ formation.publishedatstring }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
{% endif %}

</div>
{% endblock %}

```

Utilisation de `app.request.attributes.get('_route')` pour détecter si nous sommes sur la route d'ajout ou d'édition, et adapter le titre de la page.

« if playlist is defined » vérifie si une playlist existe. Si c'est le cas boucle sur `playlist. formations` pour afficher un tableau HTML simple contenant les titres et dates des formations.

La possibilité d'ajouter, modifier et supprimer les playlist à été ajouté en respectant les contraintes.

Tâche 2 : gérer les playlists (5h)	Temps réel : 6h
------------------------------------	-----------------

Tâche 3 : gérer les catégories

On commence par créer un formulaire.

php bin/console make:form CategoryType Category

Ce qui crée `src/Form/CategoryType.php`

```

class CategorieType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name', TextType::class, [
                'label' => 'Nom de la catégorie',
                'required' => true
            ])
            ->add('submit', SubmitType::class, [
                'label' => 'Ajouter',
                'attr' => ['class' => 'btn btn-success mt-2']
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Categorie::class,
        ]);
    }
}

```

Dedans on a le champ name et submit avec le bouton de submit directement dans le formulaire.

Puis on crée le contrôleur

php bin/console make:controller "Admin\AdminCategoriesController"

Ce qui crée src/Controller/Admin/AdminCategoriesController et /templates/admin/admin_categories/index.html.twig

```

class AdminCategoriesController extends AbstractController
{
    private $categorieRepository;

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

    #[Route('/admin/categories', name: 'admin.categories')]
    public function index(Request $request): Response
    {
        $categorie = new Categorie();
        $form = $this->createForm(CategorieType::class, $categorie);

        $form->handleRequest($request);
        if($form->isSubmitted() && $form->isValid()){
            $existing = $this->categorieRepository->findOneBy(['name' => $categorie->getName()]);

            if ($existing) {
                $this->addFlash('danger', 'Cette catégorie existe déjà.');
            } else {
                $this->categorieRepository->add($categorie);
                $this->addFlash('success', 'Catégorie ajoutée avec succès.');
                return $this->redirectToRoute('admin.categories');
            }
        }

        $categories = $this->categorieRepository->findAll();

        return $this->render('admin/admin_categories/index.html.twig', [
            'categories' => $categories,
            'formcategorie' => $form->createView()
        ]);
    }
}

```

La méthode index gère l’affichage de la liste et traite le formulaire d’ajout.

-Création d’une catégorie vide que l’on remplit avec les données du formulaires

-Validation avec vérification des doublons. Si la catégorie existe un message d’erreur apparaît et sinon un message de succès.

```

#[Route('/admin/categories/suppr/{id}', name: 'admin.categorie.suppr')]
public function suppr(int $id, Request $request): Response
{
    $categorie = $this->categorieRepository->find($id);

    if ($this->isCsrfTokenValid('delete.'.$categorie->getId(), $request->get('_token'))) {
        if (count($categorie->getFormations()) > 0) {
            $this->addFlash('danger', 'Impossible de supprimer cette catégorie car elle est utilisée dans des formations.');
        } else {
            $this->categorieRepository->remove($categorie);
            $this->addFlash('success', 'Catégorie supprimée.');
        }
    }

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

```

La méthode suppr gère la suppression des catégories.

- Récupération de la catégorie ciblée par l'id dans l'url.
- Verification du token CSRF
- Vérification si la catégorie fait partie d'un formation ou non. Si non on supprime avec succès sinon message d'erreur.

Pour finir la vue du fichier templates/admin/admin_categories/index.html.twig

```
{% block body %}
    <div class="container mt-4">
        <h2>Gestion des catégories</h2>

        {% for message in app.flashes('success') %}
            <div class="alert alert-success">{{ message }}</div>
        {% endfor %}
        {% for message in app.flashes('danger') %}
            <div class="alert alert-danger">{{ message }}</div>
        {% endfor %}

        <div class="row mt-4">
            <div class="col-md-4">
                <div class="card">
                    <div class="card-header bg-info text-white">
                        Ajouter une catégorie
                    </div>
                    <div class="card-body">
                        {{ form_start(formcategorie) }}
                        {{ form_row(formcategorie.name) }}
                        {{ form_row(formcategorie.submit) }}
                        {{ form_end(formcategorie) }}
                    </div>
                </div>
            </div>
        </div>
    </div>
{% endblock %}
```

- L'affichage de success ou d'échec est reçu depuis le contrôleur et affiché à l'écran.
- Pour la mise en page utilisation d'une grille bootstrap pour diviser la répartition a 1/3 pour le formulaire et 2/3 pour la liste.
- On génère le token CSRF caché

```
<div class="col-md-8">
    <table class="table table-striped">
        <thead>
            <tr>
                <th>Nom</th>
                <th class="text-center">Actions</th>
            </tr>
        </thead>
        <tbody>
            {% for categorie in categories %}
                <tr>
                    <td>{{ categorie.name }}</td>
                    <td class="text-center">
                        <form method="POST" action="{{ path('admin.categorie.suppr', {id: categorie.id}) }}"
                            onsubmit="return confirm('Êtes-vous sûr ?');">
                            <input type="hidden" name="_token" value="{{ csrf_token('delete' ~ categorie.id) }}">
                            <button type="submit" class="btn btn-danger btn-sm">Supprimer</button>
                        </form>
                    </td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
</div>
{% endblock %}
```


Cette partie affiche la liste sur la colonne de droite avec une boucle for pour parcourir le tableau envoyé par le contrôleur.

- Suppression de l'id choisit avec demande de confirmation.
- Génération du token CSRF pour cette catégorie

Gestion des catégories

Ajouter une catégorie

Nom de la catégorie

Ajouter

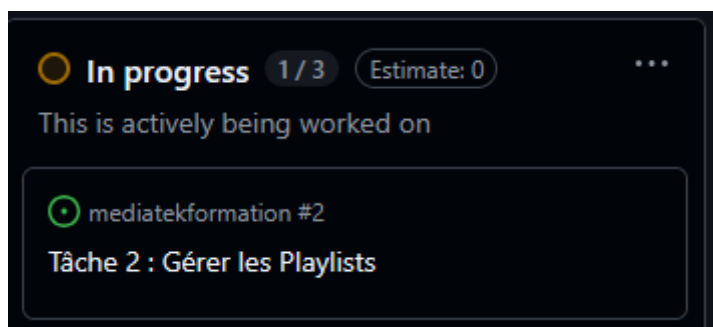
Nom	Actions
Java	Supprimer
UML	Supprimer
C#	Supprimer
Python	Supprimer
MCD	Supprimer
Android	Supprimer
POO	Supprimer
SQL	Supprimer
Cours	Supprimer

Tâche 3 : gérer les catégories (3h)

Temps réel : 3h

Tâche 4 : ajouter l'accès avec authentification

Premièrement nous allons créer la table qui contiendra le login et le mot de passe hashé.



```

PS C:\wamp64\www\mediatekformation> php bin/console make:user

The name of the security user class (e.g. User) [User]:
> User

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
> yes

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
> username

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).

Does this app need to hash/check user passwords? (yes/no) [yes]:
> yes

created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml

Success!

```

Ensuite mise à jour de la bdd avec

php bin/console make:migration
php bin/console doctrine:migrations:migrate

Avec

php bin/console make:security:form-login
On créer le contrôleur de sécurité.

```

PS C:\wamp64\www\mediatekformation> php bin/console make:security:form-login

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
>

Do you want to generate a '/logout' URL? (yes/no) [yes]:
> yes

Do you want to generate PHPUnit tests? [Experimental] (yes/no) [no]:
> no

created: src/Controller/SecurityController.php
created: templates/security/login.html.twig
updated: config/packages/security.yaml

Success!

Next: Review and adapt the login template: security/login.html.twig to suit your needs.

```

Dans le fichier security.yaml situé à **config/packages/security.yaml** on ajoute la ligne définissant le path par défaut sur l'url de la page des formations admin.

```

firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        lazy: true
        provider: app_user_provider
        form_login:
            login_path: app_login
            check_path: app_login
            enable_csrf: true
            default_target_path: admin.definitions
        logout:
            path: app_logout
            # where to redirect after logout
            # target: app_any_route

```

On dé-commente également la ligne admin plus bas dans le fichier.

```

access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }

```

Donc maintenant pour toute url commençant par admin l'utilisateur doit avoir le badge ROLE_ADMIN sinon il sera renvoyé vers la page de login

Pour générer un premier utilisateur j'ai utilisé le bundle DoctrineFixtures pour gérer l'ajout d'un utilisateur dont le mot de passe est hashé.

Pour installer le bundle

composer require --dev doctrine/doctrine-fixtures-bundle

```

<?php

namespace App\DataFixtures;

use App\Entity\User;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;

class AppFixtures extends Fixture
{
    private $passwordHasher;

    public function __construct(UserPasswordHasherInterface $passwordHasher)
    {
        $this->passwordHasher = $passwordHasher;
    }

    public function load(ObjectManager $manager): void
    {
        $user = new User();
        $user->setUsername('admin');
        $user->setRoles(['ROLE_ADMIN']);

        $hashedPassword = $this->passwordHasher->hashPassword(
            $user,
            'admin'
        );
        $user->setPassword($hashedPassword);

        $manager->persist($user);
        $manager->flush();
    }
}

```

Création d'un dossier DataFixtures dans /src/controller et dedans un fichier AppFixtures.php.

Ce fichier sert donc de scrip d'insertion d'un premier admin. Il récupère l'outil UserPasswordHasherInterface avec le constructeur. Ensuite il crée un objet lui donne le ROLE_ADMIN, hash le mdp et l'enregistre en BDD

Il faut ensuite faire cette commande pour ajouter dans la BDD l'utilisateur admin. Ne pas oublier --append sinon ça efface la BDD.

php bin/console doctrine:fixtures:load --append

Le contrôleur contient ce code qui à été généré automatiquement par la commande **php bin/console make:security:form-login**

```

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;

class SecurityController extends AbstractController
{
    #[Route(path: '/login', name: 'app_login')]
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        // get the login error if there is one
        $error = $authenticationUtils->getLastAuthenticationError();

        // last username entered by the user
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('security/login.html.twig', [
            'last_username' => $lastUsername,
            'error' => $error,
        ]);
    }

    #[Route(path: '/logout', name: 'app_logout')]
    public function logout(): void
    {
        throw new LogicException('This method can be blank - it will be intercepted by the logout key on your firewall.');
```

Il gère l'affichage du formulaire de connexion et la route de déconnexion.

- Si l'utilisateur est déjà connecté et essaie de retourner sur /login, ça le renvoie directement vers l'admin.

- `$error = $authenticationUtils->getLastAuthenticationError();` sert à stocker un identifiant invalide et à afficher un message d'erreur.

- Username préremplie si jamais le mdp est erronée pour gagner du temps.

Pour l'affichage dans la vue il faut modifier `basefront.html.twig`

On récupère l'utilisateur connecté via `app.user`. Le bouton de déconnexion pointe vers `app_logout` du contrôleur et Admin vers l'url de formations admin. Sinon vers la page de login

```
{% if app.user %}
    <li class="nav-item">
        <a class="nav-link text-danger" href="{{ path('app_logout') }}">Se déconnecter</a>
    </li>
    <li class="nav-item">
        <a class="nav-link fw-bold" href="{{ path('admin. formations') }}">Admin</a>
    </li>
{% else %}
    <li class="nav-item">
        <a class="nav-link" href="{{ path('app_login') }}">Login</a>
    </li>
{% endif %}
```

Please sign in

Username

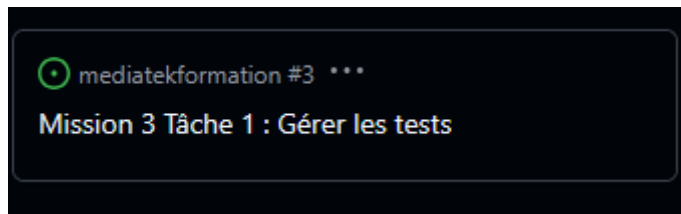
Password

Sign in

**Tâche 4 : ajouter l'accès avec authentification
(4 h)**

Temps réel : 4h30

Mission 3



Tâche 1 : gérer les tests

Nous allons d'abord installer le pack de test Symfony avec **composer require --dev symfony/test-pack**

Ensuite dans le fichier `.env.test` nous allons ajouter le lien vers l'url de test de la BDD

```
# define your env variables for the test env here
KERNEL_CLASS='App\Kernel'
APP_SECRET='Secretfort3st'
SYMFONY_DEPRECATIONS_HELPER=999999
PANTHER_APP_ENV=panther
PANTHER_ERROR_SCREENSHOT_DIR=./var/error-screenshots
DATABASE_URL="mysql://root:@127.0.0.1:3306/mediatekformation_test?serverVersion=9.1.0&charset=utf8mb4"
```

php bin/console doctrine:database:create --env=test

php bin/console doctrine:schema:create --env=test

php bin/console doctrine:fixtures:load --env=test --append (pour l'admin)

Ces commandes vont servir à peupler la BDD de test pour que les tests unitaires aient des données à tester.

Ensuite on ajoute le `.sql` pour avoir les données de la BDD

Création du fichier **tests/Unit/FormationTest.php**

```
class FormationTest extends TestCase
{
    /**
     * Vérifie que la date s'affiche dans le bon format
     * @return void
     */
    public function testGetPublishedAtString(): void
    {
        $formation = new Formation();
        $date = new \DateTime("2023-01-04 17:00:12");
        $formation->setPublishedAt($date);

        $this->assertEquals("04/01/2023", $formation->getPublishedAtString());
    }
}
```

Ce test se concentre sur la méthode `getPublishedAtString()`.

La date est stockée en base au format `DateTime` (ex: 2023-01-04 17:00:00), mais

l'affichage demande un format français string (04/01/2023). On affecte une date précise et on la test avec \$this → assertEquals.

Création du fichier **tests/Integration/FormationValidationsTest.php**

```
class FormationValidationsTest extends KernelTestCase
{
    /**
     * Test la date de demain et doit renvoyer une erreur
     * @return void
     */
    public function testDatePosteriorToTodayIsInvalid(): void
    {
        self::bootKernel();
        $container = static::getContainer();
        $validator = $container->get('validator');

        $formation = new Formation();
        $formation->setTitle("Test Future Date");
        $formation->setPublishedAt(new \DateTime('+1 day'));

        $formation->setPlaylist(new Playlist());

        $errors = $validator->validate($formation);

        $this->assertCount(1, $errors, "La validation aurait dû échouer pour une date future.");
    }
}
```

Le but est de tester les règles de validation.

L'application doit interdire la saisie d'une date de parution postérieure à la date du jour. J'ai utilisé KernelTestCase pour accéder au service Validator. Le test instancie une formation avec une date à J+1. L'assertion \$this->assertCount(1, \$errors) valide que le système rejette bien cette donnée (le test réussit si une erreur est détectée).

```
/**
 * Test la date de hier et doit renvoyer un succès
 * @return void
 */
public function testDatePriorToTodayIsValid(): void
{
    self::bootKernel();
    $validator = static::getContainer()->get('validator');

    $formation = new Formation();
    $formation->setTitle("Test Past Date");
    $formation->setPublishedAt(new \DateTime('-1 day'));
    $formation->setPlaylist(new Playlist());

    $errors = $validator->validate($formation);

    $this->assertCount(0, $errors, "La validation aurait dû réussir pour une date passée.");
}
```

La même chose est faite mais pour la date de hier.

Création du fichier `tests/Integration/PlaylistRepositoryTest.php`

```
class PlaylistRepositoryTest extends KernelTestCase
{
    /**
     * Test l'envoi et le tri des objets
     * @return void
     */
    public function testFindAllOrderByNbFormations(): void
    {
        self::bootKernel();
        $repository = static::getContainer()->get(PlaylistRepository::class);

        $playlists = $repository->findAllOrderByNbFormations('ASC');

        $this->assertIsArray($playlists);

        if (count($playlists) >= 2) {
            $first = $playlists[0]->getFormations()->count();
            $last = $playlists[count($playlists) - 1]->getFormations()->count();
            $this->assertLessThanOrEqual($last, $first);
        }
    }
}
```

On vérifie que la méthode renvoie bien un tableau. Ensuite on vérifie si la logique de tri fonctionne en vérifiant que le premier élément du tableau ai moins (ou autant) de vidéos que le dernier.

Création du fichier `tests/Functional/FrontTest.php`

```
class FrontTest extends WebTestCase
{
    /**
     * Teste l'accès à la page d'accueil et le lien vers les formations
     * @return void
     */
    public function testPageAccueilAndLink(): void
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/');

        $this->assertResponseIsSuccessful();
        $this->assertSelectorTextContains('nav', 'Accueil');

        $link = $crawler->selectLink('Formations')->link();
        $client->click($link);
        $this->assertResponseIsSuccessful();
        $this->assertRouteSame('formations');
    }
}
```

On crée un client http virtuel. Ensuite on demande la page d'accueil que l'on stock dans \$crawler. On vérifie que l'on recoit un 200 OK et que le mot Accueil est présent dans la barre de navigation. On simule un click sur le texte Formations et on vérifie qu'on arrive sur la bonne route.

```
/**
 * Teste le filtre de recherche des formations (Page Formations)
 * @return void
 */
public function testFilterFormations(): void
{
    $client = static::createClient();
    $crawler = $client->request('GET', '/formations');

    $form = $crawler->selectButton('filtrer')->form([
        'recherche' => 'Java'
    ]);

    $client->submit($form);

    $this->assertResponseIsSuccessful();
    $this->assertSelectorTextContains('table tbody tr', 'Java');
}
```

TestFilterFormations teste que l'accès à la liste de filtre fonctionne en récupérant le formulaire via le bouton « filtrer ». Il remplit le champ texte avec la valeur "Java" et soumet le formulaire. L'assertion \$this->assertSelectorTextContains() valide que le tableau de résultats contient bien le terme recherché.

Les 3 autres méthodes fonctionnent de la même façon « request -> click -> assert »

```

/**
 * Teste le tri des Playlists (Page Playlists)
 * @return void
 */
public function testSortPlaylists(): void
{
    $client = static::createClient();
    $client->request('GET', '/playlists/tri/name/ASC');
    $this->assertResponseIsSuccessful();
    $this->assertSelectorExists('table tbody tr');
}

/**
 * Teste le clic sur une playlist pour voir le détail
 * @return void
 */
public function testClickPlaylistDetail(): void
{
    $client = static::createClient();
    $crawler = $client->request('GET', '/playlists');

    $link = $crawler->selectLink('Voir détail')->link();
    $client->click($link);

    $this->assertResponseIsSuccessful();
    $this->assertRouteSame('playlists.showone');
    $this->assertSelectorExists('h4');
}

```

```

/**
 * Teste le tri des formations (Page Formations)
 * @return void
 */
public function testSortFormations(): void
{
    $client = static::createClient();
    $crawler = $client->request('GET', '/formations/tri/title/ASC');

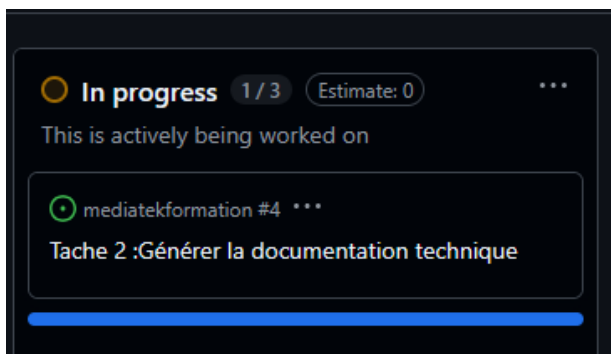
    $this->assertResponseIsSuccessful();

    $this->assertSelectorExists('table tbody tr');
}

```

Tâche 1 : gérer les tests (7h)	Temps réel : 6-7h
--------------------------------	-------------------

Tache 2 : Générer la documentation technique



Installation de phpDocumentor dans Netbeans pour générer la documentation à partir des commentaires depuis l'interface graphique.

Lors de la génération une erreur de permission bloque la génération.

Received error "mkdir(): Permission denied", while attempting to create directory ".phpdoc/cache"

On peut contourner le problème en lançant la commande manuellement depuis
 C:\wamp64\www\mediatekformation> dans powershell.

mediatekformation

Namespaces

DoctrineMigrations

App

Controller

DataFixtures

Entity

Form

Repository

Tests

ContainerFPJvYUN

ContainerXlrXHsS

Proxies

__CG__

Symfony

Config

Component

Contracts

Bundle

Bridge

Flex

Polyfill

UX

ContainerMxBcLP9

Composer

Autoload

Documentation

Table of Contents

Packages

Application

Namespaces

DoctrineMigrations

App

ContainerFPJvYUN

ContainerXlrXHsS

Proxies

Symfony

ContainerMxBcLP9

Composer

Doctrine

Command

Fixtures

Egulias

Masterminds

Monolog

DeepCopy

PhpParser

Search (Press "/" to focus)

Tâche 2 : créer la documentation technique (1h)	Temps réel : 1h
--	-----------------

Tâche 3 : créer la documentation utilisateur

In progress

1 / 3

Estimate: 0

...

This is actively being worked on

mediatekformation #5

...

Création video pour documentation utilisateur

Mediatek 86 backoffice

Vidéo de présentation du backoffice du projet mediatek86

3:31

Une vidéo de présentation du back office du projet accessible à l’url suivant : <https://www.youtube.com/watch?v=VrW9WoQzqDU>

Tâche 3 : créer la documentation utilisateur (2h)	Temps réel : 1h
--	-----------------

Mission 4 :

Tâche 1 : déployer le site



J'ai fini par déployer sur Infinity Free suite a des problèmes avec Planet Hoster.

Etapas nécessaire :

- Passage de dev a prod dans le .env APP_ENV=prod
- Optimisation des dépendances via composer install --no-dev --optimize-autoloader
- Génération du cache de production en local avant transfert.

Après avoir exporter la BDD je l'ai importé dans le phpmyadmin du fournisseur.

Les fichiers ont été transféré en FTP via WinSCP. Création d'un fichier .htaccess dans le dossier

/public pour rediriger le trafic entrant vers le contrôleur frontal de Symfony (index.php).

```
php_value display_errors Off
php_value mbstring.http_input auto
php_value date.timezone America/New_York
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_URI} !^/public/
    RewriteRule ^(.*)$ public/$1 [L]
</IfModule>
```

Tâche 1 : déployer le site (2 h)	Temps réel : 5h
----------------------------------	-----------------

Tâche 2 : gérer la sauvegarde et la restauration de la BDD

Infinity Free n'a malheureusement pas de Cron Jobs pour sa version gratuite j'ai donc contourné le problème avec un scrip de sauvegarde php et le site cron-job.org

```
<?php
// 1. SÉCURITÉ
$cle_securite = "MediaTek86_Backup_key";

if (!isset($_GET['key']) || $_GET['key'] !== $cle_securite) {
    header('HTTP/1.0 403 Forbidden');
    die("Accès non autorisé.");
}

// 2. CONFIGURATION
$host = 'sql202.infinityfree.com';
$user = 'if0_41191427';
$pass = 'jNDG8Jar0l99';
$db = 'if0_41191427_mediatek';

// 3. DOSSIER DE SAUVEGARDE
$dossier_backups = __DIR__ . '/backups_db';
if (!is_dir($dossier_backups)) {
    mkdir($dossier_backups, 0755, true);
}

$nom_fichier = "backup_" . date("Y-m-d_H-i") . ".sql";
$chemin_complet = $dossier_backups . '/' . $nom_fichier;
```

La partie sécurité vérifie que la demande contient bien la backup key et refuse l'accées si non.

Configuration contient les données pour se connecter à la BDD.

La partie 3 créer un dossier de sauvegarde s'il n'existe pas déjà et définit le format du backup.sql.

```

try {
    $conn = new mysqli($host, $user, $pass, $db);
    $conn->set_charset("utf8");

    $tables = array();
    $result = $conn->query("SHOW TABLES");
    while($row = $result->fetch_row()) { $tables[] = $row[0]; }

    fwrite($handle, "-- Sauvegarde MediaTek - " . date("Y-m-d H:i") . "\n\n");

    foreach($tables as $table) {
        // Structure de la table
        $row2 = $conn->query("SHOW CREATE TABLE $table")->fetch_row();
        fwrite($handle, "\n\n" . $row2[1] . ";\n\n");

        // Données de la table
        $result = $conn->query("SELECT * FROM $table");
        while($row = $result->fetch_row()) {
            $sql = "INSERT INTO $table VALUES(";
            for($j=0; $j<$conn->field_count; $j++) {
                if (isset($row[$j])) {
                    $sql .= "'" . $conn->real_escape_string($row[$j]) . "'";
                } else {
                    $sql .= 'NULL';
                }
                if ($j < ($conn->field_count - 1)) { $sql .= ','; }
            }
            $sql .= ");\n";
            fwrite($handle, $sql);
        }
    }

    fclose($handle);
    echo "Sauvegarde réussie : $nom_fichier";

} catch (Exception $e) {
    fclose($handle);
    echo "Erreur : " . $e->getMessage();
}

```

Cette partie parcourt toutes les tables pour en extraire toutes les données.

Tâche 2 : gérer la sauvegarde et la restauration de la BDD (1h)

Temps réel : 2h30

Tâche 3 : mettre en place le déploiement continu



Création d'un secret dans le github nommé FTP_PASSWORD avec comme valeur le mdp ftp de infinity free.

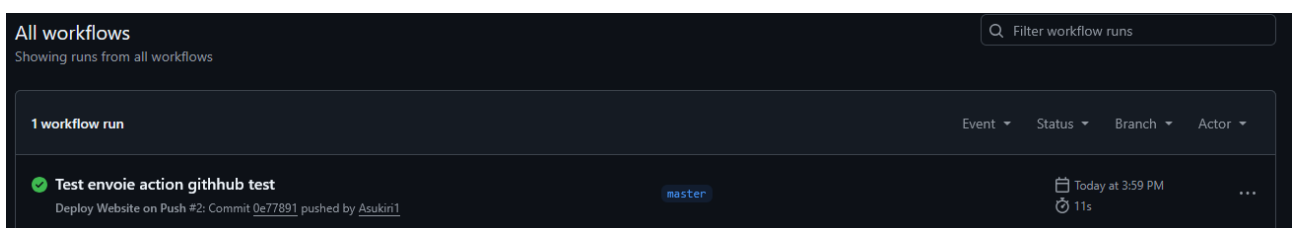
Création d'un dossier deploy.yaml qui a pour but de dire à github d'envoyer les mise a jours du repo sur le site web.

```
on:
  push:
    branches:
      - master

name: Deploy Website on Push

jobs:
  web-deploy:
    name: Deploy
    runs-on: ubuntu-latest
    steps:
      - name: Get latest code
        uses: actions/checkout@v4

      - name: Sync files
        uses: SamKirkland/FTP-Deploy-Action@v4.3.5
        with:
          server: ftpupload.net
          username: if0_41191427
          password: ${ secrets.FTP_PASSWORD }
          server-dir: /htdocs/
          exclude: |
            **/.git*
            **/.git*/**
            **/node_modules/**
            **/tests/**
            .env.test
            .env
            config/packages/doctrine.yaml
```



Tâche 3 : mettre en place le déploiement

Temps réel : 2h

Bilan Final

L'ensemble des missions confiées a été réalisé. L'application dispose désormais d'un Back-office sécurisé et fonctionnel permettant la gestion complète des données. Le code a été fiabilisé par des tests unitaires et fonctionnels, et documenté. Enfin, le site est accessible publiquement et bénéficie d'un processus de déploiement automatisé.

Problèmes rencontrés et solutions apportées :

Contraintes d'hébergement : L'hébergeur gratuit choisi (InfinityFree) ne proposant pas d'accès SSH, l'installation des dépendances (composer install) et le vidage du cache n'ont pas pu être réalisés sur le serveur.

- *Solution :* J'ai préparé une version "production" en local (optimisation de l'autoloader, suppression des dev-dependencies) avant de transférer les fichiers via FTP.

Même problèmes avec l'absence de Cron Jobs pour backup la base de données.

- *Solution :* Utilisation d'un site tier (cronjobs.org) pour réaliser des backups journalier.