

ATELIER DE PROFESSIONNALISATION.

Application web Symfony.



MediaTek86

Des formations pour tous
sur des outils numériques

Brevet de technicien supérieur - Services informatiques aux organisations 2023-2024

CNED

Sommaire

Préambule.....	3
Mission 0 : préparer l'environnement de travail.....	3
Mission 1 : nettoyer et optimiser le code existant.....	5
Tâche 1 : nettoyer le code.....	5
Tâche 2 : ajouter une fonctionnalité.....	7
VIEW (page des playlists).....	9
CONTROLLER.....	10
DAL.....	10
VIEW (page d'une playlist).....	12
Mission 2 : coder la partie back-office.....	13
Tâche 1 : gérer les formations.....	13
VIEW (page admin formation).....	14
CONTROLLER.....	15
VIEW (supprimer une formation).....	16
CONTROLLER.....	17
VIEW (ajouter une formation).....	18
CONTROLLER.....	19
VIEW (éditer une formation).....	20
CONTROLLER.....	21
Tâche 2 : gérer les playlists.....	21
VIEW (page admin playlists).....	22
CONTROLLER.....	24
Tâche 3 : gérer les catégories.....	27
VIEW (admin catégorie).....	27
CONTROLLER.....	28
SECURITE.....	30
Tâche 4 : ajouter l'accès avec authentification.....	31
Mission 3 : tester et documenter.....	37
Tâche 1 : gérer les tests.....	37
Tests unitaires.....	37
Tests d'intégration sur les règles de validation :.....	38
Tests d'intégration sur les Repository :.....	41
Tests fonctionnels :.....	43
Tests de compatibilité :.....	45
Tâche 2 : créer la documentation technique.....	47
Tâche 3 : créer la documentation utilisateur.....	47
Mission 4 : déployer le site et gérer le déploiement continu.....	47
Tâche 1 : déployer le site.....	47
Keycloak.....	47
Site en ligne.....	51
Tâche 2 : gérer la sauvegarde et la restauration de la BDD.....	52
Tâche 3 : mettre en place le déploiement continu.....	53
Bilan.....	54

Préambule.

Cette réalisation se fait dans le contexte suivant.

InfoTech Services 86 (ITS 86), est une Entreprise de Services Numériques (ESN) spécialisée dans le développement informatique (applications de bureau, web, mobile), l'hébergement de site web, l'infogérance, la gestion de parc informatique et l'ingénierie système et réseau. Elle répond régulièrement à des appels d'offres en tant que société d'infogérance et prestataire de services informatiques.

Parmi les marchés gagnés récemment par ITS 86, on trouve celui de la gestion du parc informatique du réseau des médiathèques de la Vienne, MediaTek86, ainsi que l'informatisation de plusieurs activités internes des médiathèques ou en lien avec le public.

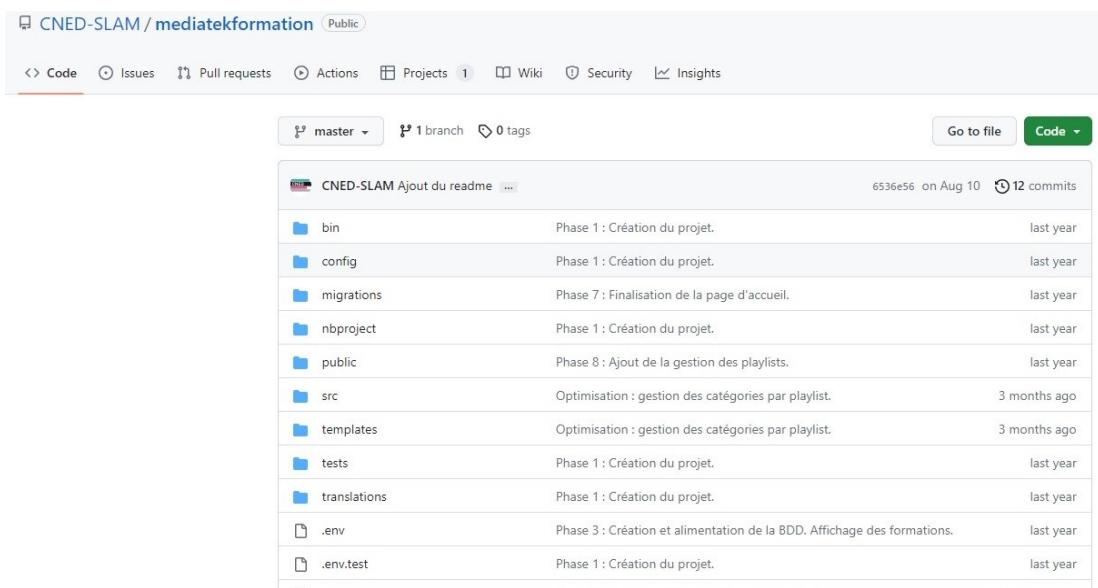
Je travail en tant que technicien junior développeur pour l'ESN InfoTech Services 86. Il m'est confié la réalisation d'une application Web d'accès aux informations en ligne.

Cette application Web est déjà commencée, elle est réalisé via le framework Symfony. Il m'est demandé de poursuivre son développement en gardant la structure actuelle et en suivant un cheminement se composant de quatre missions qui constituent les parties majeurs du process de développement de l'application.

Mission 0 : préparer l'environnement de travail.

Pour cette réalisation, j'utilise l'IDE Apache NetBeans, avec l'extension SonarLint4netbeans. Ainsi que tout le nécessaire pour faire fonctionner un projet sous Symfony, Composer, Wampserver et Git. En faisant particulièrement attention à la version de php car tout ces logicielles ne sont pas compatibles avec toutes les versions de php. Je choisi d'utiliser php8.0.26.

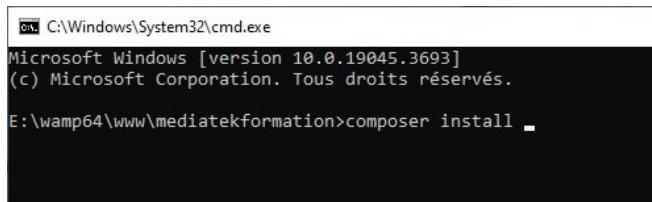
Je récupère le début de l'application depuis le dépôt GitHub du CNED.



The screenshot shows a GitHub repository page for 'CNED-SLAM / mediatekformation'. The repository has 1 branch and 0 tags. The commit history is as follows:

Commit	Message	Date
6536e56	CNED-SLAM Ajout du readme	on Aug 10
12 commits		
bin	Phase 1 : Création du projet.	last year
config	Phase 1 : Création du projet.	last year
migrations	Phase 7 : Finalisation de la page d'accueil.	last year
nbproject	Phase 1 : Création du projet.	last year
public	Phase 8 : Ajout de la gestion des playlists.	last year
src	Optimisation : gestion des catégories par playlist.	3 months ago
templates	Optimisation : gestion des catégories par playlist.	3 months ago
tests	Phase 1 : Création du projet.	last year
translations	Phase 1 : Création du projet.	last year
.env	Phase 3 : Création et alimentation de la BDD. Affichage des formations.	last year
.env.test	Phase 1 : Création du projet.	last year

Une fois l'application récupérée, je décomprime le tout dans un répertoire dédié de Wampserver. Ensuite je me positionne dans le répertoire de l'application via une console et j'installe le projet Symfony via Composer.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.19045.3693]
(c) Microsoft Corporation. Tous droits réservés.

E:\wamp64\www\mediatekformation>composer install -
```

Je configure la base de donnée via le script SQL fourni sur le GitHub du CNED et je m'assure que la BDD est bien présente.

Je consulte le Kanban du GitHub du CNED pour savoir ce qui a été fait et je lance l'application pour m'assurer du bon fonctionnement de l'installation.

Ensuite je crée un dépôt sur mon GitHub, je « commit » et « push » l'application.

Dans mon dépôt, je crée un projet Kanban et je le remplis de toutes les tâches prévues.

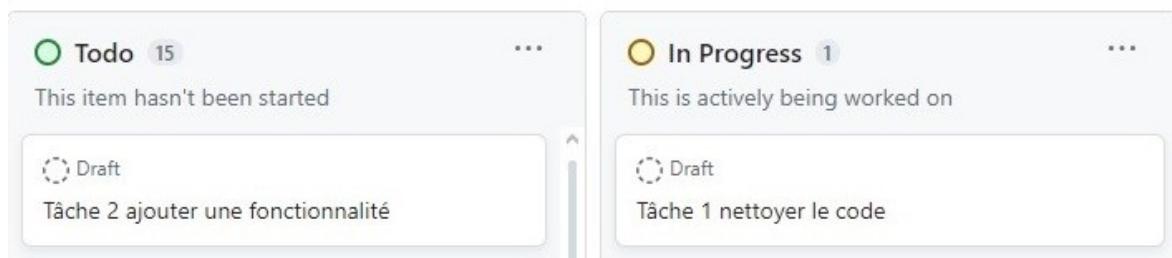
Mission 1 : nettoyer et optimiser le code existant

Tâche 1 : nettoyer le code.

Tâches à réaliser :

Nettoyer le code en suivant les indications de Sonarlint (ne nettoyer que les fichiers créés par le développeur, donc trier les "Action items" de Sonarlint par "Location" et s'arrêter au premier fichier dans "vendor").

Dans un premier temps, je place la tâche 1 nettoyer le code du kanban dans la partie in « progress ».



Ensuite j'affiche toutes les erreurs trouvées par l' extension sonarlit de l'IDE Netbean.

Action Items	
	Description
!	Syntax error unexpected:yield after!= expected:exit, integer, double, identifier, STRING_,
!	Syntax error unexpected:(expected:instanceof, as, =>, {, }, ', ', OR, XOR, &&, ?, ;, , &&,],
!	Unclosed 'block'
!	Unclosed 'if'
!	Unexpected 'endblock', expected 'endif'
!	Unclosed 'block'
!	Unclosed 'if'
!	Unexpected 'endblock', expected 'endif'
!	Syntax error unexpected:(expected:instanceof, as, =>, {, }, ', ', OR, XOR, &&, ?, ;, , &&,],
↑	php:S1192 = String literals should not be duplicated

Je trie les erreurs en fonction de leur emplacement pour modifier uniquement les fichiers qui me concerne.

Exemple de correction, il m'est indiqué une erreur « php:S1192 =String literals should not be duplicated » sur le fichier [FormationsController.php](#)

```

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

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

```

SonarLint interface showing the following details:

- Controller > FormationsController >**
- Action Items X**
- Description**
- SonarLint Critical (20)**
- php:S1192 = String literals should not be duplicated**
- File: FormationsController.php**

Donc je crée une constante contenant le chemin de la page pour quelle ne soit plus en dure et je remplace.

Ensute on peut constater que le nombre d'erreurs diminue et ne s'affiche plus pour [FormationsController.php](#).

```

public function sort($champ, $ordre, $stable=""): Response{
    $formations = $this->formationRepository->findAllOrderBy($champ, $ordre);
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_FORMATIONS, [
        'formations' => $formations,
        'categories' => $categories
    ]);
}

```

SonarLint interface showing the following details:

- Controller > FormationsController >**
- Action Items X**
- Description**
- SonarLint Critical (19)**
- php:S1192 = String literals should not be duplicated**
- File: PlaylistsController.php**

J'applique ce procédé pour toutes les erreurs de ce type sur les autres fichiers.

Pour les autres principales erreurs on retrouve, le nom de cheminImage dans [Formation.php](#), une erreur d'accolade dans [Playlist.php](#), un switch qui n'est pas utile dans [PlaylisteController.php](#), je le supprime dans un premier temps puis me ravise dans un second temps, en codant la fonction `sort()` concerné je découvre qu'en faire une alternatif pour supprimer l'affichage de l'erreur (un if imbriqué) me sera nécessaire pour la fonction.

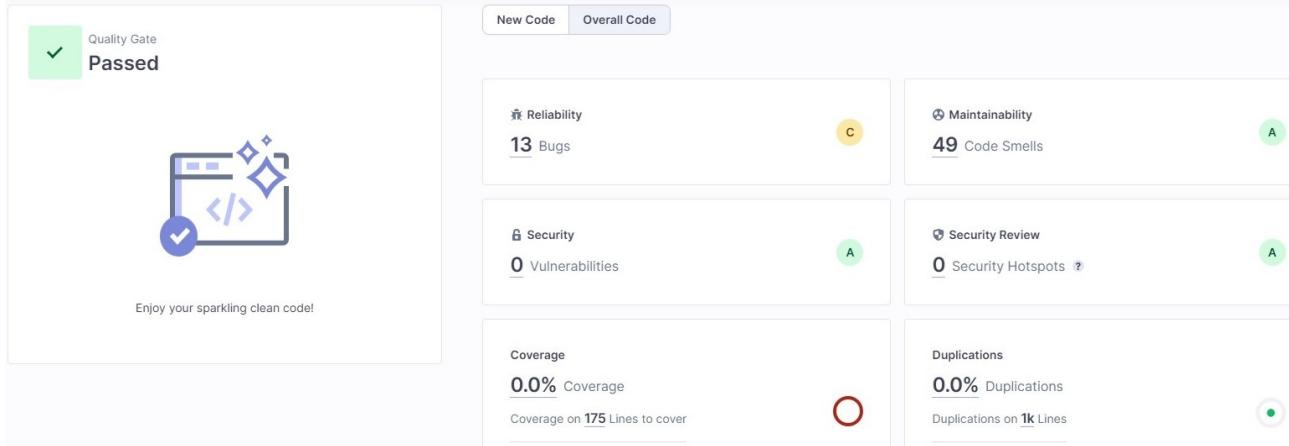
Ensute j'ajoute l'attribut alt aux images, pour rappel cet attribut est une description textuelle que l'on attribue à une image. Donc je recherche toutes les balises images via l'IDE et ensuite j'ajoute alt="image de formation".



J'ajoute aussi l'attribut description à toutes les tables. J'applique le même procédé de recherche que pour image et j'applique cet attribut.

Pour finir je lance l'application web pour m'assurer quelle fonctionne toujours et en plus je lance une analyse via sonarQube et je peux constater que tout est en ordre.

Les erreurs restantes affichées concernent des fichiers systèmes, je ne dois pas y toucher.



Le nettoyage du code étant terminé, je modifie mon kanban pour passer à la tâche suivante et je « commit » et « push » mon travail sur mon GitHub.



Sur les 2h estimés il m'a fallut 2h30 pour réaliser cette tâche.

Tâche 2 : ajouter une fonctionnalité.

Tâches à réaliser :

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.

Le but est d'ajouter dans la page des playlists 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.

Je comprends que les formations sont classés selon playlist_id et que chaque playlist possède un id. Capture de la table formation

id	published_at	title	description	video_id	playlist_id
1	2021-01-04 17:00:12	Eclipse n°8 : Déploiement	Exécution de l'application en dehors de l'IDE, en ...	Z4yTTXka958	1
2	2021-01-02 17:00:01	Eclipse n°7 : Tests unitaires	Intégration de JUnit dans l'application et créatio...	-nw42Xq6cYE	1
3	2020-12-30 17:00:07	Eclipse n°6 : Documentation technique	Intégration des commentaires normalisés et générat...	PjK_P3TKc00	1

Capture de la table playlist.

id	name	description
1	Eclipse et Java	Utilisation de l'IDE Eclipse et développement en J...
2	Visual Studio 2019 et C#	Plusieurs vidéos portant sur différents aspects de...
3	Programmation sous Python	Exercices progressifs pour apprendre à programmer

Avec cela quand je pointe le curseur sur un playlist_id dans la table formation, PhpMyAdmin m'indique le lien.

Exemple quand je point mon curseur sur un id il m'est indiqué la playlist concerné ici Eclipse et java

playlist_id
1
1
1
1
1

Donc je parcours les entités à la recherche du lien entre ces deux tables et je tombe sur la propriété **\$formation** dans [Playlist.php](#) qui lie les deux.

```
/**
 * @ORM\OneToMany(targetEntity=Formation::class, mappedBy="playlist")
 */
private $formations;
```

Explication :

@ORM\OneToOne: signifie qu'un enregistrement de l'entité à laquelle cette annotation est ajoutée peut être associé à plusieurs enregistrements de l'entité cible.

targetEntity=Formation::class: Spécifie l'entité cible de la relation donc l'entité Formation.

En résumé, cela signifie qu'un enregistrement de l'entité Playlist peut être associé à plusieurs enregistrements de l'entité Formation.

mappedBy="playlist": Indique le nom de la propriété dans l'entité Formation qui maintient la relation. Ce qui est le cas, dans [Formation.php](#) l'on retrouve bien la propriété `playlist`.

```
/**
 * @ORM\ManyToOne(targetEntity=Playlist::class, inversedBy="formations")
 */
private $playlist;
```

Avec tout ces éléments, je comprends le fonctionnement des entités [Formation.php](#) et [Playlist.php](#). Et donc je comprends que la méthode `getFormation()` de [Playlist.php](#) me retournera une collection

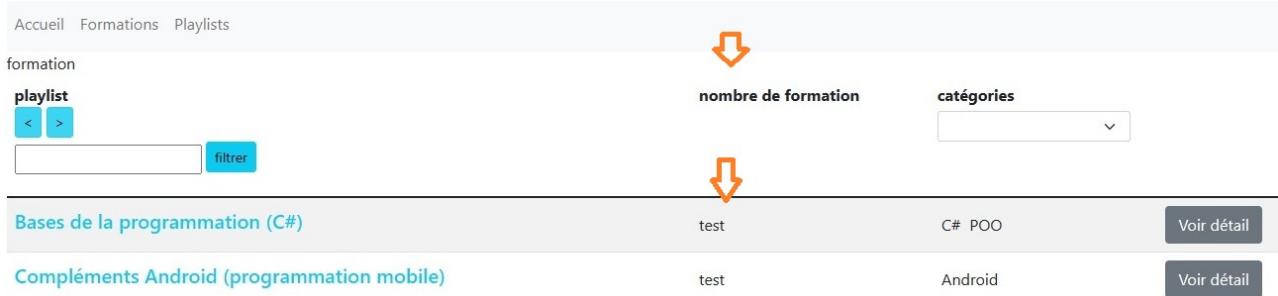
de formation. Donc il ne me reste plus qu'à faire appel à cette méthode pour obtenir le nombre de formation par playlist dans `playlist.html.twig`

```
/**  
 * @return Collection<int, Formation>  
 */  
public function getFormations(): Collection {  
    return $this->formations;  
}
```

VIEW (page des playlists)

Dans `playlist.html.twig` je crée une nouvelle colonne que j'appelle nombre de formation et je la remplis pour voir si l'affichage fonctionne.

```
<tbody>  
    <!-- boucle sur les playlists -->  
    {% if playlists|length > 0 %}  
        {% for k in 0..playlists|length-1 %}  
            <tr class="align-middle">  
                <td>  
                    <h5 class="text-info">  
                        {{ playlists[k].name }}  
                    </h5>  
                </td>  
  
                <td>  
                    test  
                </td>
```



Bases de la programmation (C#)	nombre de formation	catégories
test	C# POO	<button>Voir détail</button>
test	Android	<button>Voir détail</button>

J'utilise la boucle qui permet d'afficher les playlists pour faire appel à la méthode `getFormation()`, cependant cette méthode me retourne une collection de formation donc avec la propriété `length`, je peux obtenir la taille de la collection qui correspond au nombre de formation par playlist.

```
<!-- boucle sur les playlists -->  
    {% if playlists|length > 0 %}  
        {% for k in 0..playlists|length-1 %}  
            <tr class="align-middle">  
                <td>  
                    <h5 class="text-info">  
                        {{ playlists[k].name }}  
                    </h5>  
                </td>  
  
                <td>  
                    <h5 class="text-info">  
                        {{ playlists[k].getFormations()|length }}  
                    </h5>
```

Pour le tri, je crée des boutons pour nombre de formation en copiant ceux de playlist déjà présents.

```
<th class="text-left align-top" scope="col">
    nombre de formation<br/>
     <a href="#" class="btn btn-info btn-sm active" role="button" aria-pressed="true"><</a>
     <a href="#" class="btn btn-info btn-sm active" role="button" aria-pressed="true">>>/a>
</th>
```

CONTROLLER

Ensuite je me rends dans `PlaylistsControlleur.php`. Et là je réalise que le switch que j'ai supprimé pendant le nettoyage du code me permettait d'utiliser la même méthode `sort()` et de ne pas en créer une nouvelle, alors je le remets et le remplace par des if pour enlever l'erreur de sonnarLint. Je donne un nouveau nom de paramètre 'nombreDeFormation' pour déclencher le if. Et créer un nom de méthode `findAllOrderByNombre()`.

```
/**
 * @Route("/playlists/tri/{champ}/{ordre}", name="playlists.sort")
 * @param type $champ
 * @param type $ordre
 * @return Response
 */
public function sort($champ, $ordre): Response {
    if ($champ == "name") {
        $playlists = $this->playlistRepository->findAllOrderByName($ordre);
    }
    if ($champ == "nombreDeFormation") { 
        $playlists = $this->playlistRepository->findAllOrderByNombre($ordre); 
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_PLAYLISTS, [
        'playlists' => $playlists,
        'categories' => $categories
    ]);
}
```

Donc la méthode `sort()` reçoit un champ et un ordre en paramètre, envoyé par les boutons de `playlist.html.twig`. Si le champ est 'nombreDeFormation', la propriété `$playlists` sera valorisée par le retour de la méthode `findAllOrderByNombre()` qui elle-même reçoit en paramètre l'ordre des boutons et retourne un tri croissant ou décroissant.

Le tout est retourné via la clé 'playlists' en fin de méthode.

Ce qui donne dans la vue, les paramètres du contrôleur pour le fonctionnement des boutons.

```
nombre de formation<br/>
<a href="{{ path('playlists.sort', {champ:'nombreDeFormation', ordre:'ASC'}) }}>
<a href="{{ path('playlists.sort', {champ:'nombreDeFormation', ordre:'DESC'}) }}>
```

DAL

Dans PlaylistRepository.php, je crée la méthode `findAllOrderByNombre()`.

```
    }
    /**
     * retourne les playiste triées sur le nombre de vidéos
     * @param type $champ
     * @param type $ordre
     * @return Playlist[]
     */
    public function findAllOrderByNombre($ordre): array{
        return $this->createQueryBuilder('p')
            ->leftjoin('p.formations', 'f')
            ->groupBy('p.id')
            ->orderBy('count(f.id)', $ordre)
            ->getQuery()
            ->getResult();
    }
}
```

Explication de la méthode `findAllOrderByNombre()`

Cette méthode reçoit la propriété `$ordre` qui sera ‘ASC’ ou ‘DESC’ en fonction du bouton pressé et retournera un tableau de playlist.

`createQueryBuilder('p')` : permet de créer une requête de type "select" , le 'p' est l'alliasse de table playlist.

`leftJoin('p.formation','f')`:permet de faire une jointure gauche avec l'entité formation et f est l'alliasse de la requête.

`groupBy('p.id')` : permet de de grouper les résultats par id de playlist.

`orderBy('count(f.id)',$ordre)`:permet de les trier le résultat en appliquant le COUNT sur `leftJoin('p.formation','f')` en fonction de l'ordre reçu en paramètre.

`getQuery` : permet d'exécuter la requête.

`getResult` : permet de récupérer le résultat sous forme d'un tableau d'objets.

Ensuite j'essaye l'application pour m'assurer que tout fonctionne.

formation	nombre de formation
playlist	< >
<input type="text"/> filtrer	
Bases de la programmation (C#)	74
Programmation sous Python	19
MCD : exercices progressifs	18
TP Android (programmation mobile)	18
Compléments Android (programmation mobile)	13
Visual Studio 2019 et C#	11
Cours UML	10

VIEW (page d'une playlist)

Pour afficher cette information dans la page d'une playlist donc `playlist.html.twig`, j'utilise un simple compteur qui incrémente à chaque formation.

```
<div class="col">
    <h4 class="text-info mt-5">{{ playlist.name }}</h4>
    <strong>catégories : </strong>
    <!-- boucle pour afficher les catégories -->
    {% set anccategorie = '' %}
    {% for playlist in playlistcategories %}
        {{ playlist.name }}&ampnbsp
    {% endfor %}
    <br /><br />
    {% set compteur = 0 %}
    {% for formation in playlistformations %}
        {% set compteur = compteur + 1 %} ←
    {% endfor %}

    <strong>Nombre de vidéos :{{ compteur }}</strong>
    <br /><br />
    <strong>description :</strong><br />
    {{ playlist.description|nl2br }}
```

</div>

Et cela nous rend bien le nombre de vidéo

Accueil Formations Playlists

Compléments Android (programmation mobile)

catégories : Android

Nombre de vidéos :13 ←

description :

Chaque vidéo est indépendante et présente une notion spécifique.

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



Je « commit » et « push » sur mon dépôt GitHub et mets à jour mon kanban.

Todo	In Progress	Done
13	1	2
This item hasn't been started	This is actively being worked on	This has been completed
Tâche 4 gérer les playlists	Tâche 3 gérer les formations	Tâche 1 nettoyer le code
Tâche 5 gérer les catégories		Tâche 2 ajouter une fonctionnalité
Tâche 6 ajouter l'accès avec authentification		

Sur les 2h estimés, le temps réel pour réaliser cette fonctionnalité est de 5h, j'ai eu beaucoup de mal à trouver comment me servir de la méthode `getFormation()`.

Mission 2 : coder la partie back-office.

Tâche 1 : gérer les formations

Tâches à réaliser

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.

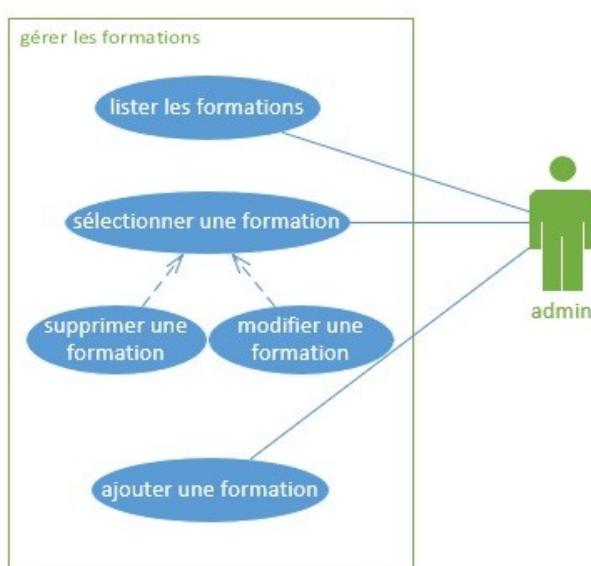
Si une formation est supprimée, il faut aussi l'enlever de la playlist où elle se trouvait.

Les mêmes tris et filtres présents dans le front office doivent être présents dans le back office.

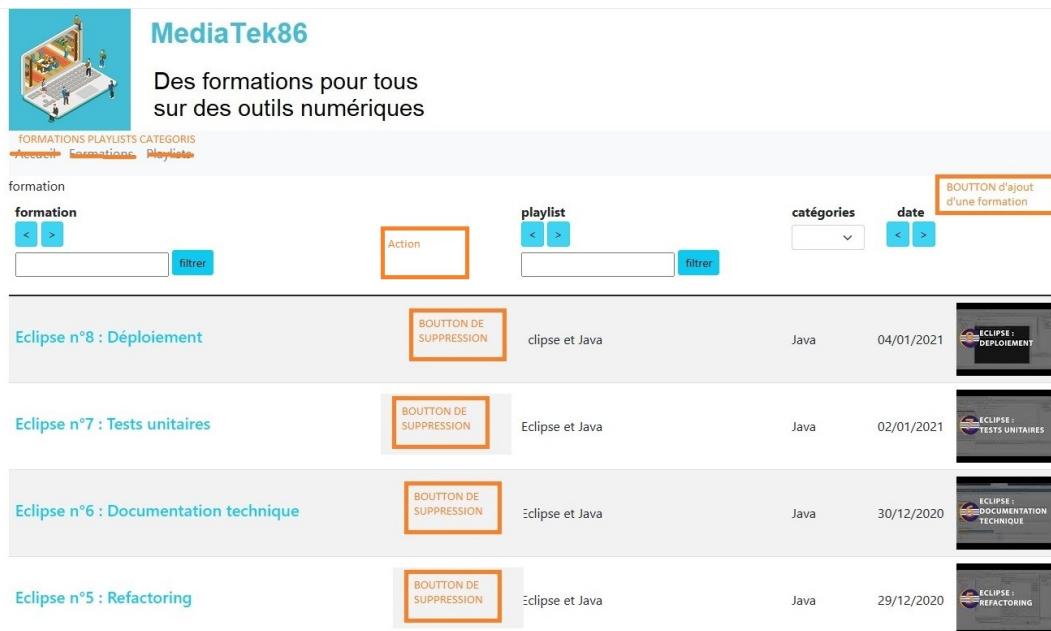
Un bouton doit permettre d'accéder au formulaire d'ajout d'une formation. Les saisies doivent être contrôlées. Seul le champ "description" n'est pas obligatoire ainsi que la sélection de catégories (une formation peut n'avoir aucune catégorie). La playlist et la ou les catégories doivent être sélectionnées 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.

Je commence par faire un diagramme de cas d'utilisation pour cette tâche.



Suivit d'une maquette en m'inspirant du front-office pour garder la structure.



MediaTek86

Des formations pour tous sur des outils numériques

FORMATIONS PLAYLISTS CATEGORISÉES

formation

formation

Action

playlist

catégories

date

BOUTON d'ajout d'une formation

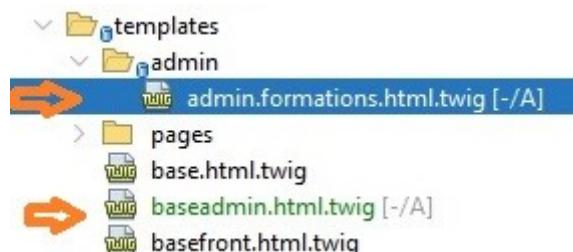
Eclipse n°8 : Déploiement	BOUTON DE SUPPRESSION	Eclipse et Java	Java	04/01/2021	
Eclipse n°7 : Tests unitaires	BOUTON DE SUPPRESSION	Eclipse et Java	Java	02/01/2021	
Eclipse n°6 : Documentation technique	BOUTON DE SUPPRESSION	Eclipse et Java	Java	30/12/2020	
Eclipse n°5 : Refactoring	BOUTON DE SUPPRESSION	Eclipse et Java	Java	29/12/2020	

VIEW (page admin formation)

Je commence par créer un dossier `admin` dans mon `templates` qui contiendra tout les fichiers vues liés au back-office.

Dans ce dossier je crée `admin.formations.html.twig` qui va contenir ma page pour lister les formations.

Dans le dossier `templates` je crée aussi le fichier `baseadmin.html.twig` qui va contenir tout ce qui est communs au back-office.



Pour respecter l'architecture du front, je copie le fichier `basefront.html.twig` et l'adapte à mon `baseadmin.html.twig` en modifiant les menus.

```

1  {% extends "base.html.twig" %}| 
2
3  {% block title %}{% endblock %}
4  {% block stylesheets %}{% endblock %}
5  {% block top %}
6      <div class="container">
7          <!-- titre -->
8          <div class="text-left">
9              
10         </div>
11         <!-- menu -->
12         <nav class="navbar navbar-expand-lg navbar-light bg-light">
13             <div class="collapse navbar-collapse" id="navbarSupportedContent">
14                 <ul class="navbar-nav mr-auto">
15                     <li class="nav-item">
16                         <a class="nav-link" href="#">Formations</a> ←
17                     </li>
18                     <li class="nav-item">
19                         <a class="nav-link" href="#">Playlists</a> ←
20                     </li>
21                     <li class="nav-item">
22                         <a class="nav-link" href="#">Catégories</a> ←
23                     </li>
24                 </ul>
25             </div>
26         </nav>
27     </div>
28     ...
29 
```

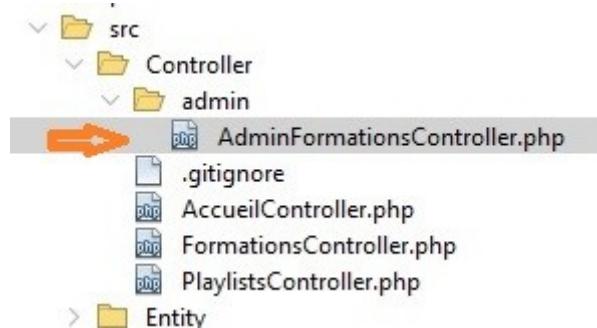
J'écris ma page `admin.formations.html` d'un simple titre pour un test dans un premier temps.

```

1  {% extends "baseadmin.html.twig" %} 
2
3  {% block body %}<h1>Page de gestion des formations</h1>{% endblock %}
4 
```

CONTROLLER

Je crée dans mon dossier `Controller` le dossier `admin` qui contiendra les contrôleurs de mon back-office et je crée le premier `AdminFormationsController.php`



Dans ce contrôleur je crée les constantes pour le chemin de la pages admin formation.

```
class AdminFormationsController extends AbstractController {  
  
    CONST PAGE_AFORMATIONS = "admin/admin.formations.html.twig";  
    CONST PAGE_AFORMATIONSBIS = 'admin.formations';
```

Deux propriétés et le constructeur qui valorise ces propriétés afin de pouvoir les utiliser dans les méthodes à venir.

Pour les premières méthodes je commence par `index()`, `findAllContain()` et `sort()` à l'image de `FormationsController.php` en modifiant les pages retournées ainsi que les noms que portent les routes.

```
public function __construct(FormationRepository $formationRepository,  
    $this->formationRepository = $formationRepository;  
    $this->categorieRepository = $categorieRepository;  
}  
  
/**  
 * Méthode pour l'affichage des formations en partie adminne.  
 * @Route("/admin", name="admin.formations")   
 * @return Response  
 */  
public function index(): Response {  
    $formations = $this->formationRepository->findAll();  
    $categories = $this->categorieRepository->findAll();  
    return $this->render(self::PAGE_AFORMATIONS, [  
        'formations' => $formations,  
        'categories' => $categories  
    ]);  
}
```

VIEW (supprimer une formation)

J'ajoute le lien dans le menu de `baseadmin.html.twig` pour avoir accès à la page maintenant que le chemin est créé dans le contrôleur.

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

Pour `admin.formations.html.twig`, je fais un copier coller du code de `formations.html.twig` afin de m'assurer de respecter l'architecture.

Je commence par modifier les liens des différents champs, boutons pour avoir accès aux bonnes pages quand j'interagis avec les divers champs.

Je teste les divers champs pour m'assurer de ne pas en oublier et de bien toujours être dans le back-office.

Ensute j'ajoute dans l'entête du tableau la colonne Action et je lui affecte les boutons éditer, supprimer avec confirmation et le bouton pour ajouter une formation.

```
<!-- Ajout des boutons éditer et supprimer avec confirmation -->
<td class="text-left">
    <a href="#" class="btn btn-secondary">Editer</a>
    <a href="#" class="btn btn-danger" onclick="confirm('Etes-vous sûr de vouloir supprimer {{formation.title}} ?')>Supprimer</a>
</td>
```

Je réalise que contrairement à la maquette il est plus esthétique d'ajouter le bouton pour ajouter une formation sous Action.

Formation	Action	playlist	catégories	Date
Eclipse n°7 : Tests unitaires	Ajouter une formation Supprimer Editer	Eclipse et Java	Java	02/01/2021
Eclipse n°6 : Documentation technique	Ajouter une formation Supprimer Editer	Eclipse et Java	Java	30/12/2020

Pour l'action d'un bouton supprimer, je crée un nom de lien et je récupère l'id de la formation en question.

```
<!-- Ajout des boutons éditer et supprimer avec confirmation -->
<td class="text-left">
    <a href="#" class="btn btn-secondary">Editer</a>
    <a href="{{ path('admin.formation.suppr', {id:formation.id}) }}" class="btn btn-danger">Supprimer</a>
</td>
```

CONTROLLER

Je retourne dans le contrôleur pour crée la méthode `suppr()`.

```
/**
 * @Route("admin/suppr/{id}", name="admin.formation.suppr")
 * @param type $id
 * @return Response
 */
public function suppr($id):Response{
    $formation= $this->formationRepository->find($id);
    $this->formationRepository->remove($formation, true);
    return $this->redirectToRoute('admin.voyages');
}
```

Explication de `suppr()`

Elle attend l'id de la formation envoyé par le bouton supprimer dans la vue.

`find($id)` appliquée sur `formationRepository` retrouve la bonne formation et valorise la propriété `$formation`.

`remove` de `formationRepository` qui attend en paramètre une formation et un booléen à true pour effacer.

Et pour finir on retourne la page pour avoir l'affichage qui se met à jour.

Toutes ces méthodes pour interagir avec l'entité formation se retrouvent dans `formationRepository.php` du dossier `Repository` qui correspond au DAL. Elles font le lien entre l'entité et la base de donnée.

VIEW (ajouter une formation)

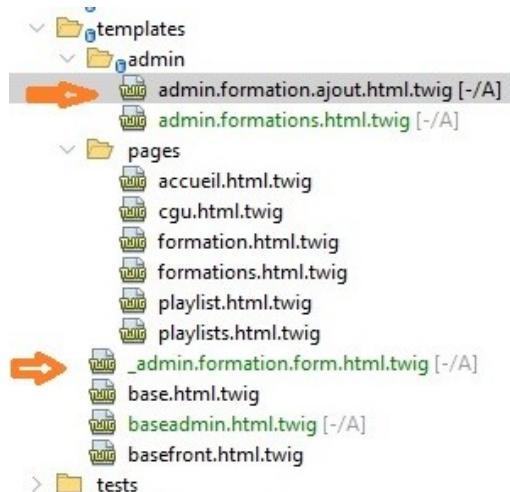
Pour l'action du bouton Ajouter une formation, je crée un nom de lien dans `admin.formations.html.twig`.

```
<!-- Ajout d'une colonne Action et le bouton Ajouter une formation -->
<th class="text-left align-top" scope="col">
    Action <br/>
    <a href="{{ path('admin.formation.ajout') }}>Ajouter une formation</a>
</th>
```

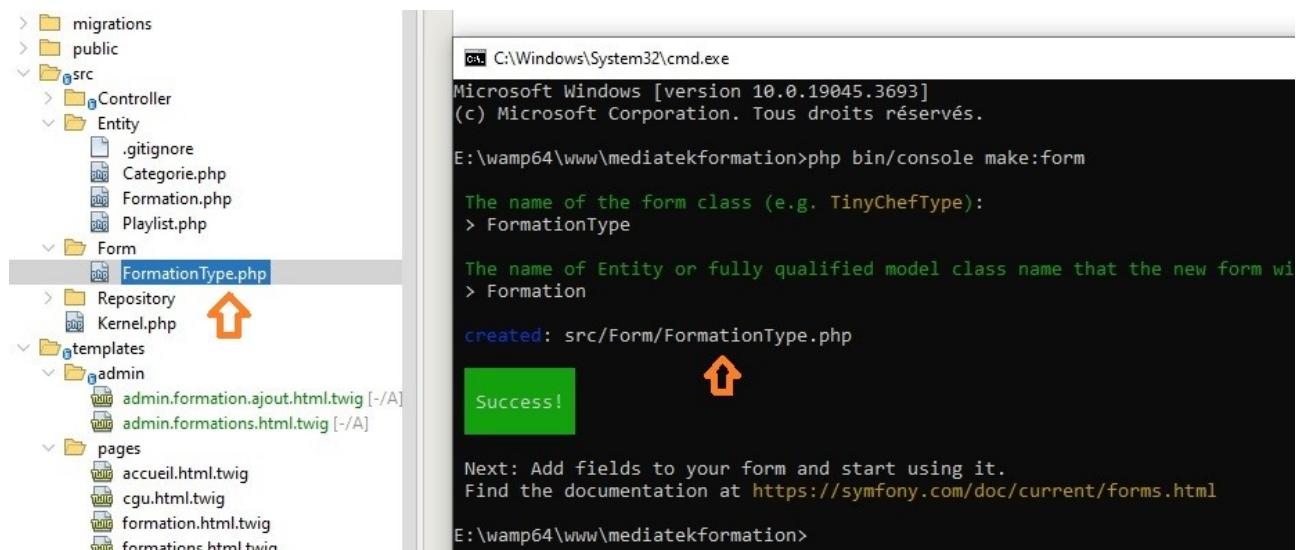


Dans le `templates` je crée le fichier `_admin.formation.form.html.twig` qui contiendra le formulaire pour ajouter ou éditer une formation.

Et je crée dans le dossier `admin` le fichier `admin.formation.ajout.html.twig`.



Ensuite j'utilise Symfony pour créer un formulaire à partir de l'entité Formation.



Je choisi un modèle de présentation du formulaire en le paramétrant dans le fichier twig.yml

```
twig:
    default_path: '%kernel.project_dir%/templates'
    form_themes: ['bootstrap_5_layout.html.twig']

when@test:
    twig:
        strict_variables: true
```

Je prépare l'appel du formulaire dans _admin.formation.form.html.twig

```
1 {{ form_start(formformation) }}
2 {{ form_end(formformation) }}
```

Et j'écris la page admin.formation.ajout.html.twig qui appelle le formulaire de _admin.formation.form.html.twig

```
1 {% extends "baseadmin.html.twig" %}
2 {%- block body %}
3     <h2>Nouvelle formation</h2>
4     {{ include('admin.formation.form.html.twig') }}
5 {%- endblock %}
```

CONTROLLER

Ensuite je fais la méthode ajout() du contrôleur.

```
/**
 * @Route("/admin/ajout", name="admin.formation.ajout")
 * @param Request $request
 * @return Response
 */
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, true);
        return $this->redirectToRoute('admin.formations');
    }
    return $this->render("admin/admin.formation.ajout.html.twig", [
        'formation'=> $formation,
        'formformation'=>$formformation->createView()
    ]);
}
```

Explication ajout()

La fonction attend une requête qui sera envoyé au moment où l'on presse le bouton.

La propriété \$formation va être valorisée par une formation « vide ».

La propriété `$formformation` va être valorisée par l'objet formulaire qui attend en paramètre une formation. Comme on lui donne une formation « vide » il va créer un formulaire « vide ».

Ensuite la méthode tente de récupérer la requête (handleRequest()).

Puis un test permet de contrôler si le formulaire soumis est valide.

Si il est valide, on va appeler la méthode `add()` qui va enregistrer les modification sur la bdd.

Et enfin nous rediriger sur la page admin des formations.

Cependant dans le cas où le formulaire n'est pas valide, cela va juste recréer un formulaire pour recommencer.

Ensuite je modifie le formulaire pour qu'il correspond aux demandes exactes des consignes.

VIEW (éditer une formation)

Pour l'action du bouton éditer, je crée un nom de lien `admin.formations.html.twig`. Sur l'action du bouton éditer.

```
<!-- Ajout des boutons éditer et supprimer avec confirmation -->
<td class="text-left">
    <a href="{{ path('admin.formation.edit', {id:formation.id}) }}" class="btn btn-secondary">Editer</a>
    <a href="{{ path('admin.formation.suppr', {id:formation.id}) }}" class="btn btn-danger" onclick="confirm('E
</td>
```

Je crée dans le dossier templates le fichier admin.formation.edit.html.twig



J'écris le fichier `admin/formation/edit.html.twig`



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

{# block body #}
    <h2>Edition d'une formation</h2>
    {{ include('_admin.formation.form.html.twig') }}
{# endblock #}
```

CONTROLLER

J'écris le contrôleur

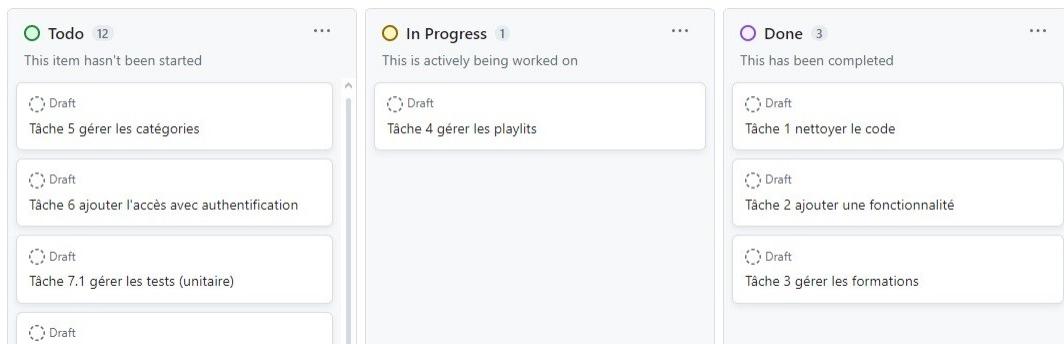
```
/*
 * @Route("/admin/edit/{id}", name="admin.formation.edit")
 * @param type $id
 * @param Request $request
 * @return Response
 */
public function edit($id, Request $request):Response{
    $formation = $this->formationRepository->find($id);
    $formformation = $this->createForm(FormationType::class, $formation);
    $formformation ->handleRequest($request);
    if($formformation->isSubmitted() && $formformation->isValid()){
        $this->formationRepository->add($formation, true);
        return $this->redirectToRoute('admin.formations');
    }
    return $this->render("admin/admin.formation.edit.html.twig", [
        'formation'=> $formation,
        'formformation'=>$formformation->createView()
    ]);
}
```

Explication `edit()`

Sont fonctionnement est identique à la méthode `ajout()`.

La différence est que je ne crée pas une formation vide mais que j'en récupère une via la méthode `find()` sur l'id envoyé en paramètre dans `admin.formations.html.twig`. Comme cela au moment de la création du formulaire, il sera rempli par les informations contenu dans la formation.

Une fois que tout fonctionne, je « commit » et « push » sur mon dépôt GitHub et mets à jour mon kanban pour passer à la prochaine tâche.



Sur les 5h estimés, le temps réel pour réaliser cette fonctionnalité est de 6h.

Tâche 2 : gérer les playlists

Tâches a réaliser

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.

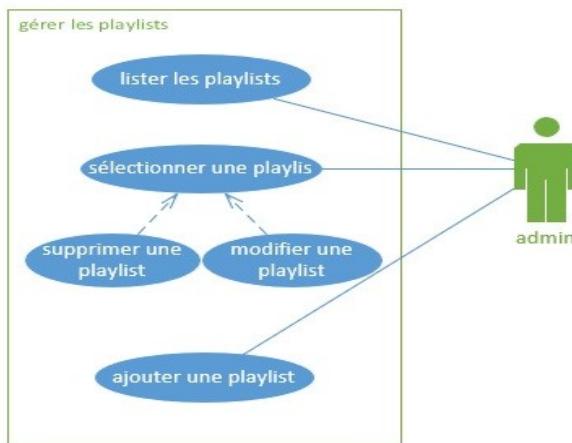
La suppression d'une playlist n'est possible que si aucune formation n'est rattachée à elle.

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

Je commence par faire un diagramme de cas d'utilisation pour cette tâche.



Je fais ensuite une maquette depuis la page d'affichage des playlists du front-office pour garder une cohérence avec le back-office et ma tâche précédente.

MediaTek86

Des formations pour tous sur des outils numériques

formation playlists catégories

Accueil Formations Playlists

formation

playlist < >

Bases de la programmation (C#) BTN SUPP / MODIFICATION 74 C# POO Voir détail

Compléments Android (programmation mobile) BTN SUPP / MODIFICATION 13 Android Voir détail

VIEW (page admin playlists)

Je commence par créer un nom de lien dans mon fichier `baseadmin.html.twig` car c'est le fichier commun de mon back-office, il me permet de naviguer entre les différentes pages du back-office.

```

<li class="nav-item">
    <a class="nav-link" href="{{ path('admin.playlists') }}>Playlists</a>
</li>
    
```

Je crée ensuite dans le dossier `admin` le fichier `admin.playlists.html.twig` qui va me permettre de gérer les playlists. Je procède comme pour la tâche précédente.

Appeler le contenu du fichier `baseadmin.html.twig`

Copier la partie body du fichier playlists.html.twig,

Ajouter l'entête Action et les boutons supprimer, éditer et ajouter une playlist

Changer tout les liens de tout les boutons et champs.

admin.playlists.html.twig [-/A] x

Source History |

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

{# block body #}





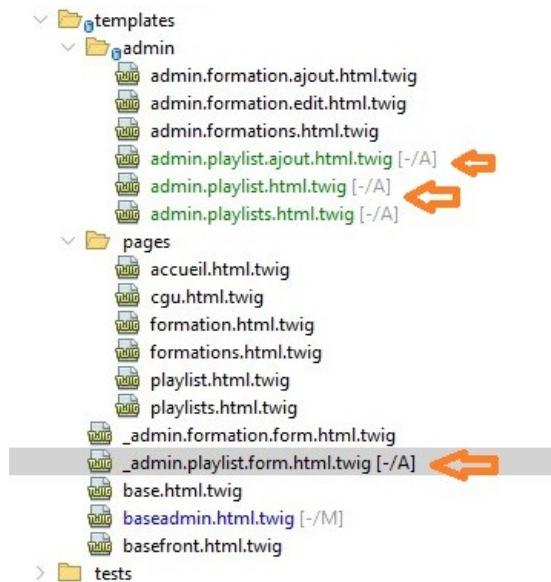
```

<!-- Ajout d'une colonne Action et le bouton Ajouter une playlist -->

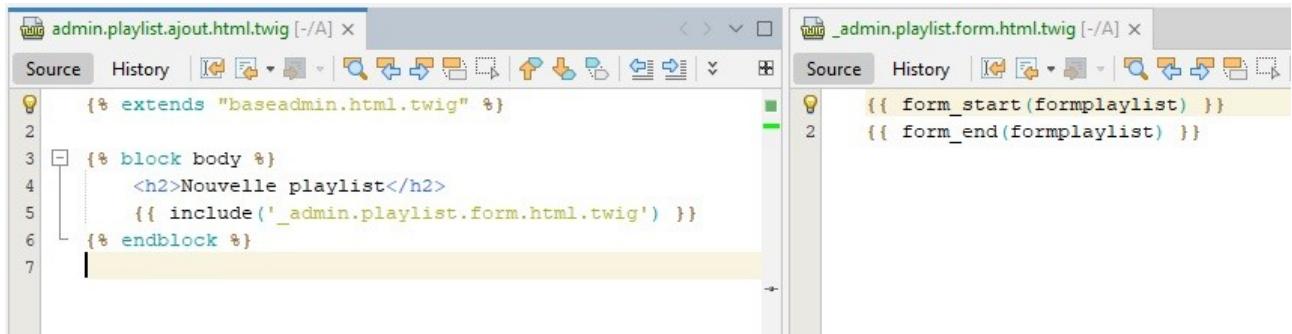
```
<th class="text-left align-top" scope="col">
    Action <br />
    <a href="{{ path('admin.playlist.ajout') }}" class="btn btn-primary">Ajouter une playlist</a>
</th>
```

```
<th class="text-left align-top" scope="col">
    nombre de formation<br />
    <a href="{{ path('admin.playlists.sort', {champ:'nombreDeFormation', ordre:'ASC'}) }}" class="btn btn-info" value="ASC" type="button"><-- Orange arrow pointing down to this line
    <a href="{{ path('admin.playlists.sort', {champ:'nombreDeFormation', ordre:'DESC'}) }}" class="btn btn-info" value="DESC" type="button"><-- Orange arrow pointing up to this line
</th>
```

Je crée ensuite les fichiers `_admin.playlist.form.html.twig` qui va contenir le formulaire, `admin.playlist.html.twig` qui va me permettre de modifier une playlist existante et `admin.playlist.ajout.html.twig` qui va me permettre d'ajouter une playlist.



Ces fichiers sont écrits de la même manière que ceux de admin.formation. La seul subtilité est que nous ne traitons pas de formations mais de playlists dans ce cas.



The screenshot shows a code editor with two tabs open. The left tab is titled "admin.playlist.ajout.html.twig" and contains the following code:

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

{%- block body %}
    <h2>Nouvelle playlist</h2>
    {{ include('_admin.playlist.form.html.twig') }}
{%- endblock %}
```

The right tab is titled "_admin.playlist.form.html.twig" and contains the following code:

```
 {{ form_start(formplaylist) }}
{{ form_end(formplaylist) }}
```

CONTROLLER

Je crée ensuite le contrôleur dans le dossier admin que je nomme `AdminPlaylistsController.php`

La méthode index du contrôleur est à l'image de la précédente, elle me permet de me rediriger vers la page `admin.playlists.html.twig`

```
/**
 * @Route("/admin/playlists", name="admin.playlists")
 * @return Response
 */
public function index(): Response {
    $playlists = $this->playlistRepository->findAllOrderBy('ASC');
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_APLAYLISTS, [
        'playlists' => $playlists,
        'categories' => $categories
    ]);
}
```

Les méthodes permettant de trier et de rechercher un éléments du contrôleur appelées depuis la pages `admin.playlists.html.twig` fonctionnent comme pour la tâche précédente.

```
public function sort($champ, $ordre): Response {
    if ($champ == "name") {
        $playlists = $this->playlistRepository->findAllOrderBy($ordre);
    }
    if ($champ == "nombreDeFormation") {
        $playlists = $this->playlistRepository->findAllOrderByNombre($ordre);
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_APLAYLISTS, [
        'playlists' => $playlists,
        'categories' => $categories
    ]);
}
```

Ensuite j'utilise Symfony pour créer un formulaire à partir de l'entité Playlist. J'adapte le formulaire aux demandes des consignes.

```

src
└── Controller
    ├── admin
    │   ├── AdminFormationsController.php
    │   └── AdminPlaylistsController.php [-/A]
    ├── Entity
    │   ├── .gitignore
    │   ├── Categorie.php
    │   ├── Formation.php
    │   └── Playlist.php
    └── Form
        ├── FormationType.php
        └── PlaylistType.php

```

Dans le contrôleur et j'écris la méthode ajout()

```

/**
 * Action du bouton Ajouter playlist dans l'onglet playlist du back-office
 * @Route("/admin/playlist/ajout", name="admin.playlist.ajout")
 * @param Request $request
 * @return Response
 */
public function ajout(Request $request): Response {
    $playlist = new Playlist();
    $formplaylist = $this->createForm(PlaylistType::class, $playlist);
    $formplaylist->handleRequest($request);
    if ($formplaylist->isSubmitted() && $formplaylist->isValid()) {
        $this->playlistRepository->add($playlist, true);
        return $this->redirectToRoute(self::PAGE_APLAYLISTSBIS);
    }
    return $this->render("admin/admin.playlist.ajout.html.twig", [
        'playlist' => $playlist,
        'formplaylist' => $formplaylist->createView()
    ]);
}

```

Elle est identique à la précédente sauf qu'elle traite des playlists et non des formations.
Ensuite je fais la méthode suppr().

```

/**
 * Action du bouton supprimer dans l'onglet playlist du back-office
 * @Route("admin/playlists/{id}", name="admin.playlist.suppr")
 * @param type $id
 * @return Response
 */
public function suppr($id): Response {
    $playlist = $this->playlistRepository->find($id);
    if (count($playlist->getFormations()) == 0) {
        $this->playlistRepository->remove($playlist, true);
        return $this->redirectToRoute(self::PAGE_APLAYLISTSBIS);
    } else {
        $this->addFlash('echec', 'impossible de supprimer une playlist qui contient des formations');
        return $this->redirectToRoute(self::PAGE_APLAYLISTSBIS);
    }
}

```

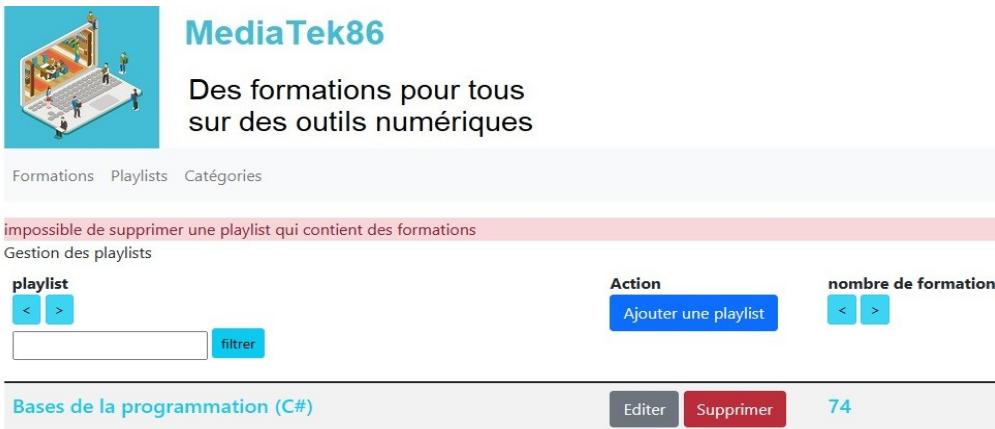
Il y a une subtilité pour cette méthode, il est demandé dans les consignes que la suppression d'une playlist n'est possible que si aucune formation n'est rattachée à elle. Donc il faut faire un test avant d'appliquer la suppression.

Et pour finir la méthode edit()

```
/**  
 * Action du bouton éditer dans l'onglet playlist du back-office  
 * @Route("/admin/playlist/edit/{id}", name="admin.playlist.edit")  
 * @param type $id  
 * @param Request $request  
 * @return Response  
 */  
  
public function edit($id, Request $request): Response {  
    $playlist = $this->playlistRepository->find($id);  
    $playlistFormations = $this->formationRepository->findAllForOnePlaylist($id);  
    $formplaylist = $this->createForm(PlaylistType::class, $playlist);  
    $formplaylist->handleRequest($request);  
    if ($formplaylist->isSubmitted() && $formplaylist->isValid()) {  
        $this->playlistRepository->add($playlist, true);  
        return $this->redirectToRoute(self::PAGE_APLAYLISTSBIS);  
    }  
    return $this->render("admin/admin.playlist.edit.html.twig", [  
        'playlist' => $playlist,  
        'formplaylist' => $formplaylist->createView(),  
        'playlistformations' => $playlistFormations  
    ]);  
}
```

Sont fonctionnement est identique à la méthode ajout(). La différence est que je ne crée pas une playlist vide mais que j'en récupère une via la méthode find() sur l'id envoyé en paramètre dans admin.playlists.html.twig

Ce qui donne au final pour les pages de gestion de playlist.



The screenshot shows the 'MediaTek86' website with a header featuring a laptop icon and the text 'MediaTek86 Des formations pour tous sur des outils numériques'. Below the header, there are navigation links for 'Formations', 'Playlists', and 'Catégories'. A red error message 'impossible de supprimer une playlist qui contient des formations' is displayed above the playlist list. The playlist list table has columns for 'Action', 'nombre de formation', and 'Bases de la programmation (C#)'. Buttons for 'Ajouter une playlist' and 'filtrer' are also visible.

Après divers essais pour m'assurer que tout fonctionne, je « commit » et « push » sur mon dépôt GitHub et mets à jour mon kanban pour passer à la prochaine tâche.



Sur les 5h estimés, le temps réel pour réaliser cette fonctionnalité est de 4h30.

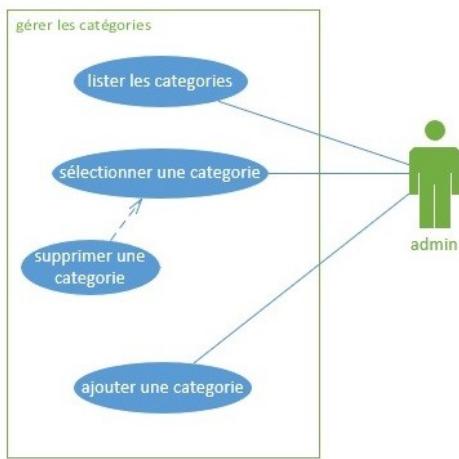
Tâche 3 : gérer les catégories.

Tâches à réaliser

Le but est de faire une page qui 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.

je commence par faire un diagramme de cas d'utilisation pour cette tâche. Suivi d'une maquette comme pour les tâches précédentes.



VIEW (admin catégorie)

Je crée un nom de lien dans mon fichier baseadmin.html.twig

```
    href="{{ path('admin.formations') }}>Formations
```



```
    href="{{ path('admin.playlists') }}>Playlists
```



```
    href="{{ path('admin.categories') }}>Catégories
```

Je crée ensuite le fichier qui va contenir le formulaire `_admin.categorie.form.html.twig` dans le dossier `templates`



```
Twig _admin.categorie.form.html.twig [-/A] ×
```

Source History |

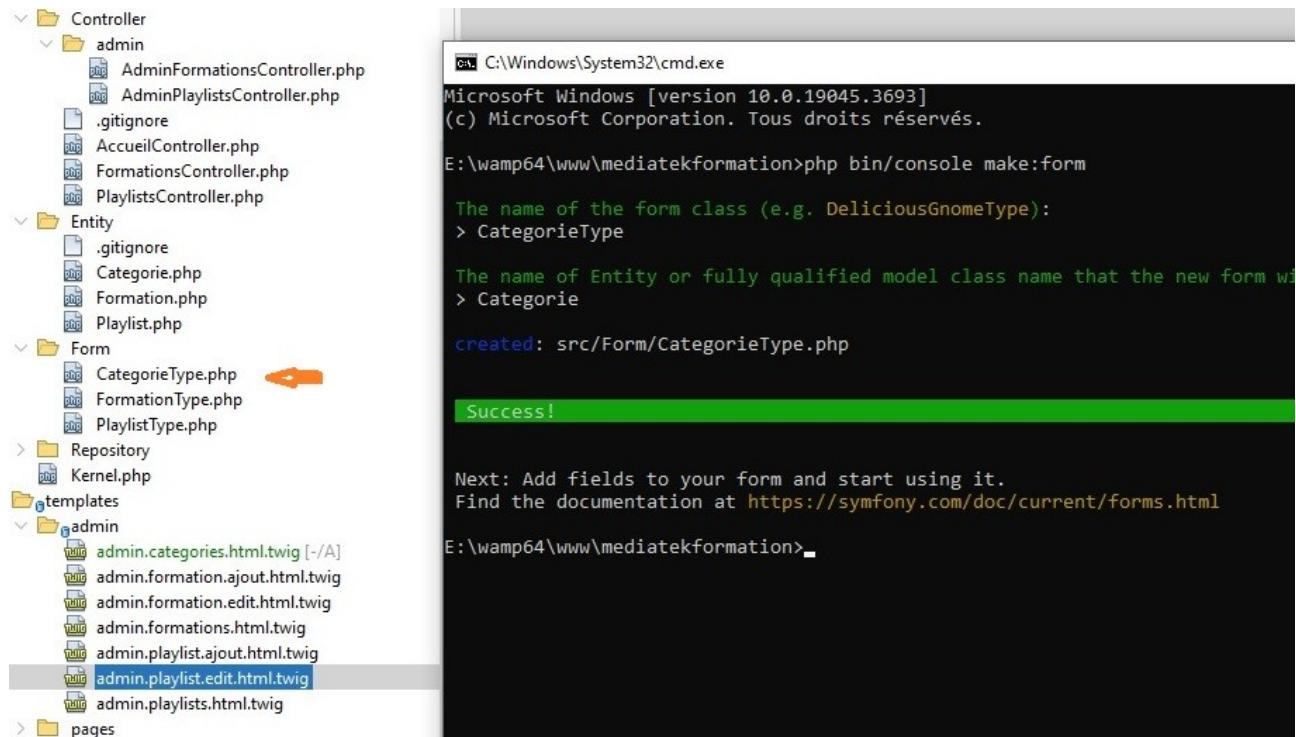
```
1 {{ form_start(formcategorie) }}
```

```
2 {{ form_end(formcategorie) }}
```

Ensuite dans le dossier `admin` de `templates` le fichier `admin.categories.html.twig` qui va être la page qui liste les catégories, j'intègre le formulaire et je crée les liens avec le contrôleur.

```
<!-- Ajout du bouton supprimer avec confirmation -->
<td class="text-left">
    <a href="{{ path('admin.categorie.suppr', {id: categorie.id}) }>
</td>
```

Ensuite j'utilise Symfony pour créer un formulaire à partir de l'entité Categorie. J'adapte le formulaire aux demandes des consignes.



CONTROLLER

Je crée ensuite le contrôleur `AdminController.php` dans le dossier `admin` de `Controller`. Ce contrôleur fonctionne comme les précédents et ne contient que deux méthodes, `index()` et `supr()`.

```
/**
 * @Route("/admin/categories", name="admin.categories")
 * @param Request $request
 * @return Response
 */
public function index(Request $request): Response {
    $categories = $this->categorieRepository->findAll();
    $categorie = new Categorie();
    $formcategorie = $this->createForm(CategorieType::class, $categorie);
    $formcategorie->handleRequest($request);
    if ($formcategorie->isSubmitted() && $formcategorie->isValid()) {
        $this->categorieRepository->add($categorie, true);
        return $this->redirectToRoute(self::PAGE_ACATEGORIESBIS);
    }
    return $this->render(self::PAGE_ACATEGORIES, [
        'categories' => $categories,
        'formcategorie' => $formcategorie->createView()
    ]);
}
```

Comme le formulaire est intégré à la page qui liste les catégories, il faut le créer dans la méthode `index()`. Son fonctionnement est identique aux précédents sauf que dans ce cas, on manipule des catégories.

Pour la fonction `suppr()` il est demandé dans les consignes qu'une catégorie ne peut être supprimée que si elle n'est rattachée à aucune formation donc avant d'appliquer `remove()`, je procède comme pour la fonction `suppr()` du contrôleur de playlist, avec un test avant la suppression.

```
/*
 * Action du bouton supprimer dans l'onglet categorie du back-office
 * @Route("admin/categorie/suppr/{id}", name="admin.categorie.suppr")
 * @param type $id
 * @return Reponse
 */
public function suppr($id): Response {
    $categorie = $this->categorieRepository->find($id);
    if (count($categorie->getFormations()) == 0) {
        $this->categorieRepository->remove($categorie, true);
        return $this->redirectToRoute(self::PAGE_ACATEGORIESBIS);
    } else {
        $this->addFlash('echec', 'impossible de supprimer une categorie qui contient des formations');
        return $this->redirectToRoute(self::PAGE_ACATEGORIESBIS);
    }
}
```

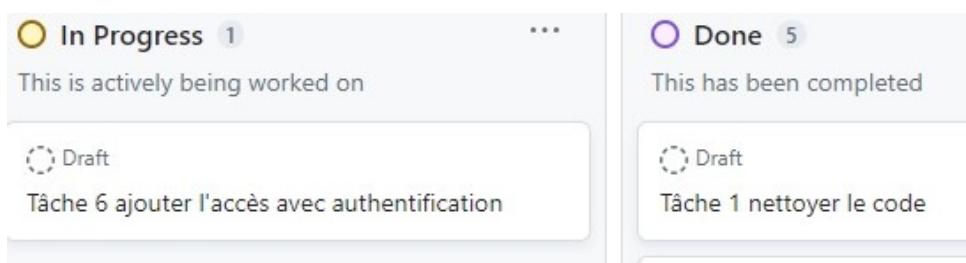
Ce qui donne pour résultat final la page suivante

The screenshot shows a web application interface for managing categories. At the top, there's a header with the logo 'MediaTek86' and a subtitle 'Des formations pour tous sur des outils numériques'. Below the header, there's a navigation bar with links for 'Formations', 'Playlists', and 'Catégories'. The main content area is titled 'Gestion des catégories' and contains a table with the following data:

Categories	
Java	<button>Supprimer</button>
UML	<button>Supprimer</button>
C#	<button>Supprimer</button>
Python	<button>Supprimer</button>
MCD	<button>Supprimer</button>
Android	<button>Supprimer</button>
POO	<button>Supprimer</button>
SQL	<button>Supprimer</button>
Cours	<button>Supprimer</button>

To the right of the table, there's a search bar labeled 'Nom de la catégorie' and a blue 'Valider' button.

Une fois que tout fonctionne, je « commit » et « push » sur mon dépôt GitHub et mets à jour mon kanban pour passer à la prochaine tâche.



Sur les 3h estimés, le temps réel pour réaliser cette fonctionnalité est de 3h.

SECURITE

Avant de commencer la tâche 4, je vais m'atteler à sécuriser le site.

Token pour contrer la CSRF (cross-site request forgery)

C'est la prise de contrôle d'une session ouverte d'un utilisateur.

Cela consiste à exécuter une requête HTTP qui va réaliser une action normalement possible qu'après authentification, en profitant d'une session ouverte permettant de réaliser des actions sensibles grâce à une seule authentification de départ, ou aucune authentification.

Pour se faire quand j'ai crée des formulaires dans ma « vue » j'ai systématiquement pris la partie qui ajoute le champ caché du token.

```
<th class="text-left align-top" scope="col">
    playlist<br />
    <a href="{{ path('admin.playlists.sort', {champ:'name', ordre:'ASC'}) }}" class="btn btn-info btn-sm">
        <a href="{{ path('admin.playlists.sort', {champ:'name', ordre:'DESC'}) }}" class="btn btn-info btn-sm">
            <form class="form-inline mt-1" method="POST" action="{{ path('admin.playlists.findallcontain', {c
                <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_name') }}"/>
                    <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
                </div>
            </form>
        </th>
```

Donc il ne me reste qu'à le vérifier dans le contrôleur.

```
public function findAllContain($champ, Request $request, $table = ""): Response {
    if ($this->isCsrfTokenValid('filtre_' . $champ, $request->get('_token'))) { ←
        $valeur = $request->get("recherche");
```

J'ai effectué cette modification pour [AdminFormationsController.php](#) et [AdminPlaylistsController.php](#).

Pour les formulaires créés par symfony, cette partie est automatiquement gérée dans les formulaires.

Pour éviter les injections SQL, je m'assure d'avoir des « setParameter » et non une concaténation simple pour les valeurs dynamiques provenant d'entrées utilisateurs.

Exemple

```
public function findByContainValue($champ, $valeur, $table=""): array{
    if($valeur==""){
        return $this->findAllOrderByName('ASC');
    }
    if($table==""){
        return $this->createQueryBuilder('p')
            ->leftjoin('p.formations', 'f')
            ->where('p.'.$champ.' LIKE :valeur')
            ->setParameter('valeur', '%'.$valeur.'%') →
            ->groupBy('p.id')
            ->orderBy('p.name', 'ASC')
            ->getQuery()
            ->getResult();
    }else{
```

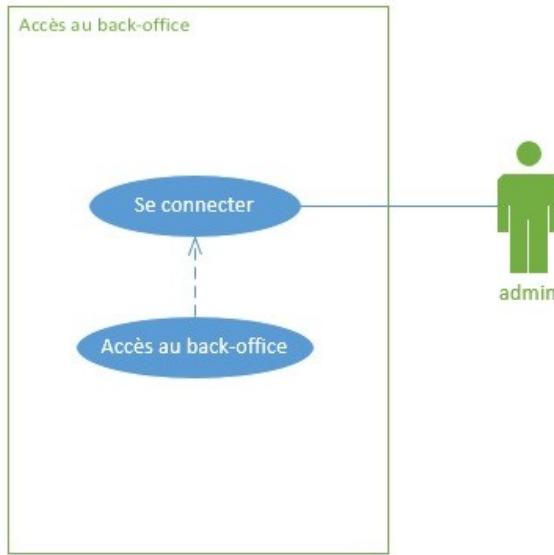
Tâche 4 : ajouter l'accès avec authentification

Tâches à réaliser

Le back-office ne doit être accessible qu'après authentification : un seul profil administrateur doit avoir le droit d'accès. Pour gérer l'authentification, utiliser Keycloak.

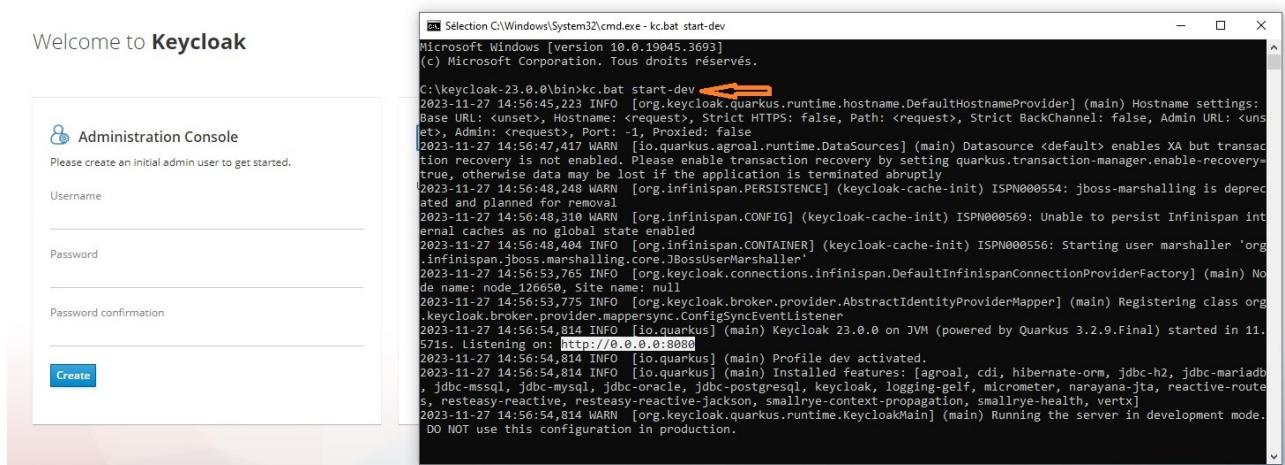
Il doit être possible de se déconnecter, sur toutes les pages

je commence par faire un diagramme de cas d'utilisations



Je compte utiliser la page de connexion par défaut de keycloak donc je ne fais pas de maquette.

Après avoir téléchargé et installé à la racine keycloak en version 19,0,1 (la dernière version en date ne fonctionne pas sur ce projet) il suffit de le lancer via la commande kc.bat start-dev dans le dossier bin de keycloak pour démarre le service.



Après avoir crée un compte administrateur, on doit se créer un « royaume » qui va contenir les applications que nous allons gérer via keycloak.

Realm name * myapplis 

Enabled  On

Create **Cancel**

Ensuite dans ce royaume on se crée un client qui correspond à l'application.

Manage Clients are applications and services that can request authentication of a user.

Clients Client type ⓘ OpenID Connect

Client scopes

Realm roles

Users Client ID * ⓘ mediatek86 
Required field

Groups

Et enfin pour cette application, on crée un utilisateur pour se loger.

Clients Username * adminmediatek86 

Client scopes Email

Realm roles

Users

Pendant les différentes étapes de création il faut configurer de la manière suivante :

- Client type : OpenID Connect
- Always display in console : Off
- Client authentication : On
- Authorisation : Off
- Standard flow : On
- Implicit flow : On
- Direct access grants : On
- Service accounts roles : Off
- OAuth 2.0 Device Authorization Grant : Off
- OIDC CIBA Grant : Off
- Enabled : On (en haut à droite)
- Valid redirect URIs (dans Access settings) : *
- Consent required (dans Login settings) : On

- Display client on screen : On
- Front channel logout (dans logout settings) : Off
- Backchannel logout session required : On
- Backchannel logout revoke offline sessions : Off
- username : adminmesmediatek86
- email : aaa@aaa.com
- Enabled : ON
- Email Verified : OFF

Une fois keycloak configuré, il faut préparer l'application symfony en fonction de la configuration de keycloak.

Pour commencer, le fichier .env

```

tests
translations
var
vendor
.env [-/M] ←
.env.test
.gitignore
README.md
composer.json [-/M]
composer.lock [-/M]

40      #***< symfony/messenger ***
41
42      #***> symfony/mailer ***
43      # MAILER_DSN=null://null
44      #***< symfony/mailer ***
45 KEYCLOAK_SECRET=LgVGygV5GXXusSP10gcmjLMJRjpOshT1
46 KEYCLOAK_CLIENTID=mediatek86
47 KEYCLOAK_APP_URL=http://localhost:8080
48

```

La configuration correspond aux paramètres entrés dans keycloak lors de la création.

L'authentification va se faire avec Keycloak, par rapport à l'utilisateur créé dans Keycloak.

Cependant, pour faciliter l'expérience de navigation, l'idée est d'enregistrer l'utilisateur dans la BDD du site, dès qu'il est connecté, pour pouvoir gérer sa déconnexion, sans dépendre de Keycloak.

Donc on commence par créer une table user via symfony.

```
E:\wamp64\www\mediatekformation>php bin/console make:user
The name of the security user class (e.g. User) [User]:
```

Ensuite on crée l'entité User.

```
E:\wamp64\www\mediatekformation>php bin/console make:entity User
Your entity already exists! So let's add some new fields!
New property name (press <return> to stop adding fields):
> keycloakId
```

Et ensuite on crée les fichiers de migrations et on effectue la migration. Il ne reste cas s'assurer que la BDD est rempli correctement.

Pour l'usage de Keycloak, il suffit de suivre les indication de la documentation et juste adapter notre code à notre configuration. (<https://github.com/knpu/university/oauth2-client-bundle>)

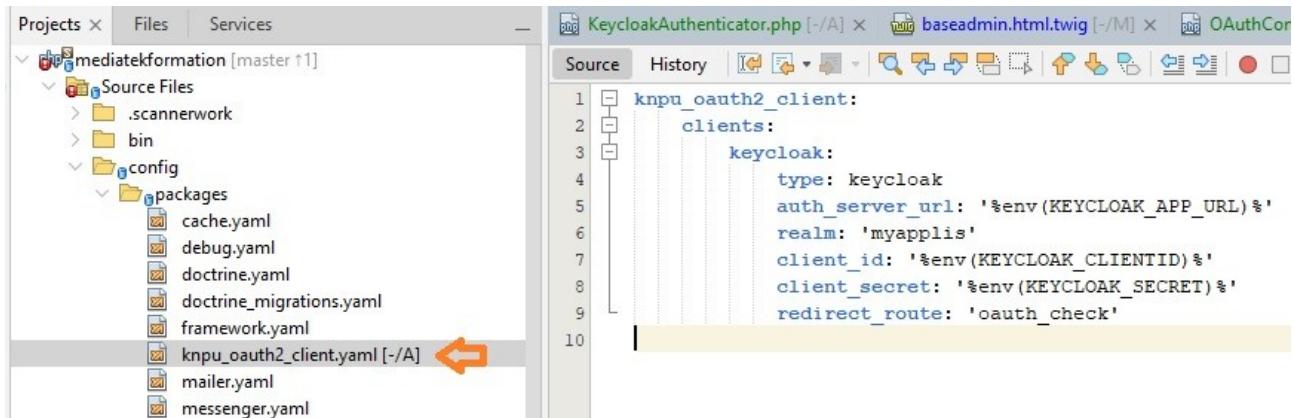
Donc ici on installe deux bundles nécessaires aux fonctionnement.
knpu/university/oauth2-client-bundle et stevenmaguire/oauth2-keycloak

```

run "composer audit" for a full list of advisories.
using version ^2.16 for knpu/university/oauth2-client-bundle ←
: \wamp64\www\mediatekformation>composer require stevenmaguire/oauth2-keycloak --with-all-dependencies ←
/composer.json has been updated
running composer update stevenmaguire/oauth2-keycloak --with-all-dependencies

```

Une fois installés, il faut configurer knpu_oauth2_client.yaml via les paramètres du .env



Ensuite on configure le fichier security.yaml

Une fois fait, on crée le contrôleur via symfony

```

E:\wamp64\www\mediatekformation>php bin/console make:controller OAuthController --no-template
created: src/Controller/OAuthController.php
Success!

```

Le contrôleur va contenir trois méthodes, `index()` qui sera appelé au moment de la demande de connexion, `connectCheckAction()` qui prend en charge la route de redirection du retour et la méthode `logout()` qui se charge de la déconnexion

```

/**
 * @Route("/oauth/login", name="oauth_login")
 */
public function index(ClientRegistry $clientRegistry): RedirectResponse
{
    return $clientRegistry->getClient('keycloak')->redirect();
}

/**
 * @Route("/oauth/callback", name="oauth_check")
 */
public function connectCheckAction(Request $request, ClientRegistry $clientRegistry) {
}

/**
 * @Route("logout", name="logout")
 */
public function logout() {
}

```

Ensuite on crée dans un dossier security, une nouvelle classe PHP qui va gérer l'authentification, `KeycloakAuthenticator.php`

Toujours en suivant la documentation, cette classe va contenir cinq méthodes.

La méthode `start()` permet de spécifier comment démarrer une authentification

```

public function start(Request $request, AuthenticationException $authException = null): Response {
    return new RedirectResponse(
        '/oauth/login',
        Response::HTTP_TEMPORARY_REDIRECT
    );
}

```

La méthode `supports()` doit renvoyer true si le système d'authentification doit se déclencher pour l'url donnée.

```

public function supports(Request $request): bool {
    return $request->attributes->get('_route') === 'oauth_check';
}

```

La méthode `onAuthenticationFailure()` s'exécute en cas de problème

```

public function onAuthenticationFailure(Request $request, AuthenticationException $exception): ?Response {
    $message = strtr($exception->getMessageKey(), $exception->getMessageData());
    return new Response($message, Response::HTTP_FORBIDDEN);
}

```

La méthode `onAuthenticationSuccess()` s'exécute quand tout s'est bien passé

```

public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response {
    $targetUrl = $this->router->generate('admin.formations');
    return new RedirectResponse($targetUrl);
}

```

La méthode `authenticate()` s'occupe de récupérer le client correspondant, dans Keycloak, et le token, à partir des informations données.

Une fois ces informations obtenues, il est possible de récupérer l'utilisateur : le but est de l'enregistrer dans la base de données pour une meilleure expérience de navigation.

Donc, il y a 3 possibilités

- soit l'utilisateur existe déjà dans la BDD et s'est déjà connecté avec Keycloak

```

public function authenticate(Request $request): Passport {
    $client = $this->clientRegistry->getClient('keycloak');
    $accessToken = $this->fetchAccessToken($client);

    return new SelfValidatingPassport(
        new UserBadge($accessToken->getToken(), function() use($accessToken, $client) {
            /**
             * @var KeycloakUser $keycloakUser
             */
            $keycloakUser = $client->fetchUserFromToken($accessToken);
            // I recherche user dans bdd par son id keycloak
            $existingUser = $this->entityManager
                ->getRepository(User::class)
                ->findOneBy(['keycloakId'=>$keycloakUser->getId()]);
            if($existingUser){
                return $existingUser;
            }
        })
    );
}

```

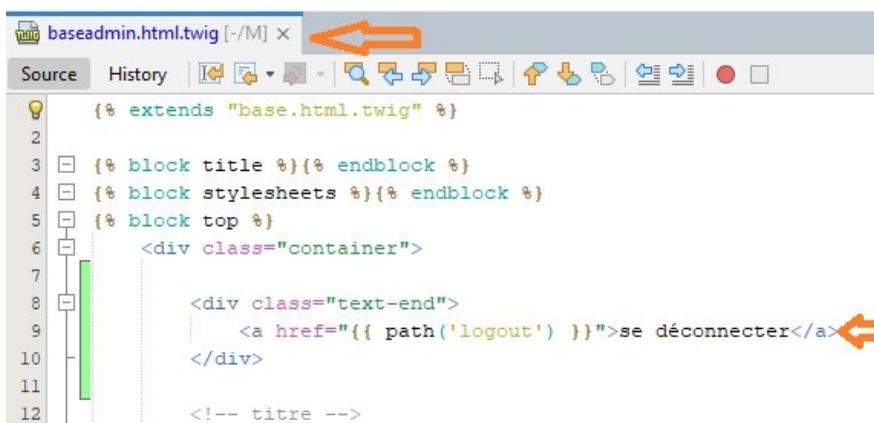
- soit l'utilisateur existe dans la BDD mais ne s'est jamais connecté avec Keycloak

```
//2 user existe mais pas encore connecté avec keycloak
$email = $keycloakUser->getEmail();
/**
 * @var User $userInDatabase
 */
$userInDatabase = $this->entityManager
    ->getRepository(User::class)
    ->findOneBy(['email'=>$email]);
if($userInDatabase){
    $userInDatabase->setKeycloakId($keycloakUser->getId());
    $this->entityManager->persist($userInDatabase);
    $this->entityManager->flush();
    return $userInDatabase;
}
```

— soit l'utilisateur n'existe pas encore dans la BDD et il faut alors le créer

```
//3 user n'existe pas encore dans la bdd
$user = new User();
$user->setKeycloakId($keycloakUser->getId());
$user->setEmail($keycloakUser->getEmail());
$user->setPassword("");
$user->setRoles(['ROLE_ADMIN']);
$this->entityManager->persist($user);
$this->entityManager->flush();
return $user;
})
```

Ensuite je met en place la déconnexion pour cela il suffit d'ajouter un lien dans le back-office qui appelle la méthode `logout()` du contrôleur.



Il suffit de lancer l'application et d'entrer l'url de la partie admin.
<http://localhost/mediatekformation/public/index.php/admin>

Cela nous présentera la page de connexion de keycloak.

Sign in to your account

Username or email
mediatek86

Password

Sign In

Il suffit d'entrer les informations paramétrés dans keycloak pour se connecter.
Pour se déconnecter il suffit de cliquer sur le lien prévu à cet effet.

The screenshot shows a web application interface. At the top right is a blue button labeled "se déconnecter". Below it is a user profile icon with the name "MediaTek86". The main content area has a teal header with the text "Des formations pour tous sur des outils numériques". Below the header are sections for "Formation" (with buttons for adding, deleting, and filtering), "Playlist" (with buttons for adding, deleting, and filtering), and "Catégories" (with a dropdown and date range buttons). A course card for "Eclipse n°7 : Tests unitaires" is displayed, showing details like "Editer", "Supprimer", "Eclipse et Java", "Cours", "02/01/2021", and a thumbnail image.

Symfony nous confirme que nous sommes bien déconnecté.



Une fois que tout fonctionne, je « commit » et « push » sur mon dépôt GitHub et mets à jour mon kanban pour passer à la prochaine tâche.

The screenshot shows a Kanban board. A card is visible with the status "In Progress 1", the text "This is actively being worked on", and a sub-card labeled "Draft" with the text "Tâche 7.1 gérer les tests (unitaire)".

Sur les 4h estimés le temps réel pour réaliser cette tâche est de 6h, elle ne demandait qu'à suivre un process depuis la documentation officiel mais j'ai eu beaucoup de blocages car je n'ai pas utilisé une version Keycloak compatible avec mon projet symfony au début.

Mission 3 : tester et documenter

Tâche 1 : gérer les tests

Tests unitaires

Tâches à réaliser

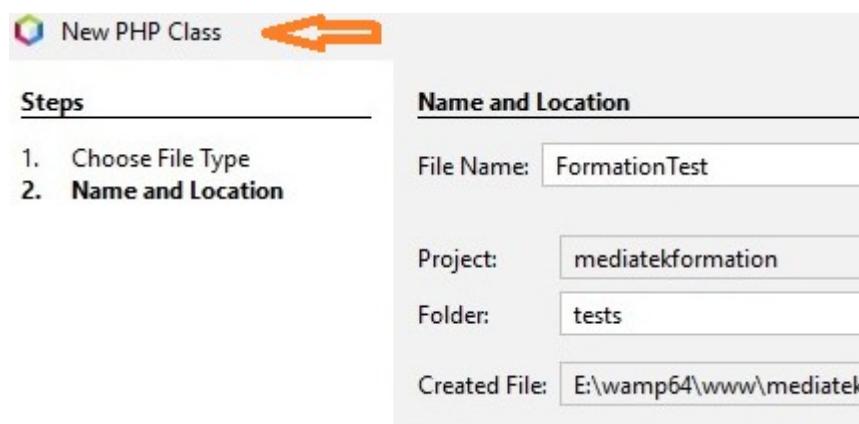
Le but est de contrôler le fonctionnement de la méthode qui retourne la date de parution au format string.

La méthode qui retourne la date de parution au format string est `getPublishedAtString()` de l'entité `Formation.php`.

J'entre la commande `php bin/phpunit` qui installe les composants nécessaires pour réaliser les tests.

```
E:\wamp64\www\mediatekformation>php bin/phpunit
PHPUnit 9.5.23 #StandWithUkraine
No tests executed!
```

Ensuite je crée une nouvelle classe `FormationTest.php` dans le dossier `tests`, par convention on donne à la classe de test le nom de la classe à tester suivi de `Test`.



je crée une méthode `testGetPublishedAtString()` qui va tester la méthode `getPublishedAtString()` de l'entité `Formation`.

```
/**
 * Test de la méthode getPublishedAtString() de Formation
 */
public function testGetPublishedAtString(){
    $formation = new Formation();
    $formation->setPublishedAt(new \DateTime("2023-01-04 17:00:12"));
    $this->assertEquals("04/01/2023", $formation->getPublishedAtString());
}
```

Explication `testGetPublishedAtString()`

Elle crée une instance de la classe Formation.

Cette instance valorise la propriété `setPublishedAt()` avec une date comme indiqué dans le plan test.

La méthode `assertEquals()` compare la date obtenu avec la date attendu.

J'entre de nouveau la commande `php bin/phpunit` pour avoir le résultat du test.

```
E:\wamp64\www\mediatekformation>php bin/phpunit
PHPUnit 9.5.23 #StandWithUkraine

Testing
.

Time: 00:00.098, Memory: 10.00 MB

OK (1 test, 1 assertion)
```

Tout est ok, je mets à jour mon travail sur GitHub et passe aux prochains tests.

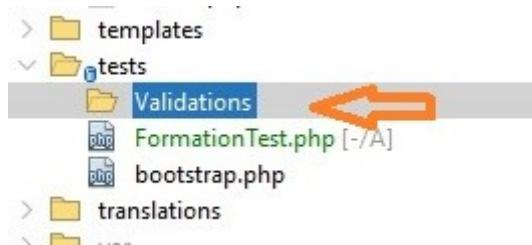
Tests d'intégration sur les règles de validation :

Tâches à réaliser

Le but est de contrôler que lors de l'ajout ou de la modification d'une formation, contrôler que la date n'est pas postérieure à aujourd'hui.

Pour cloisonner les tests je crée un nouveau dossier `Validations` dans `tests`.

Je crée une nouvelle classe `FormationValidationsTest.php`



Dans un premier temps je crée une méthode `getFormation()` qui va me retourner une formation dont je vais avoir besoin pour les tests.

```
class FormationValidationsTest extends KernelTestCase {

    public function getFormation(): Formation{
        return(new Formation())
            ->setTitle("Titre de la formation");
    }
}
```

Ensuite je crée la méthode `assertErrors()`

```

public function assertErrors(Formation $formation, int $nbErreursAttendues) {
    self::bootKernel();
    $validator = self::getContainer()->get(ValidatorInterface::class);
    $error = $validator->validate($formation);
    $this->assertCount($nbErreursAttendues, $error);
}

```

Explication

Cette méthode récupère l'objet formation et un nombre d'erreur attendu.

Ensuite, il faut faire appel au noyau (kernel) pour accéder au validateur afin de récupérer les éventuelles erreurs de validation de l'entity (ici de l'objet `$formation`) dans un tableau (`$error`). Ensuite il me suffit d'utiliser la méthode `assertErrors()` dans mes méthodes de test.

Premier test, que la date ne peut pas être postérieure à aujourd'hui.

```

/**
 * Essai qu'une date ne doit pas être postérieure à la date du jour
 * (Doit rendre une erreur)
 */
public function testValideDateFormationA(){
    $formation=$this->getFormation()->setPublishedAt((new \DateTime("2025-01-04")));
    $this->assertErrors($formation, 1);
}

```

J'entre la commande `php bin/phpunit --filter FormationValidationsTest` pour effectuer ce test. Et j'obtiens un échec (failure)

```

E:\wamp64\www\mediatekformation>php bin/phpunit --filter FormationValidationsTest
PHPUnit 9.5.23 #StandWithUkraine

Testing
F..                                         3 / 3 (100%)

Time: 00:00.952, Memory: 28.00 MB

There was 1 failure:

1) App\tests\Validations\FormationValidationsTest::testValideDateFormationA
Failed asserting that actual size 0 matches expected size 1.

E:\wamp64\www\mediatekformation\tests\Validations\FormationValidationsTest.php:48
E:\wamp64\www\mediatekformation\tests\Validations\FormationValidationsTest.php:26

FAILURES!
Tests: 3, Assertions: 3, Failures: 1. ←

```

Bien que ce n'est pas le résultat espéré, cela est normal dans mon cas car j'empêche dans le formulaire d'entrer une date postérieur pas dans l'entité.

Donc je dois ajouter une restriction à la propriété `publishedAt()` dans `Formation.php`.

```

use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;


```

/**
 * @ORM\Entity(repositoryClass=FormationRepository::class)
 */
class Formation
{
 /**
 * Début de chemin vers les images
 */
 private const CHEMIN_IMAGE = "https://i.ytimg.com/vi/";

 /**
 * @ORM\Id
 * @ORM\GeneratedValue
 * @ORM\Column(type="integer")
 */
 private $id;

 /**
 * @ORM\Column(type="datetime", nullable=true)
 * @Assert\LessThanOrEqual("now")
 */
 private $publishedAt;

```



```

Pour se faire il suffit d'intégrer une assertion sur la date.

Malheureusement pour moi il ne m'est pas possible de le faire de manière classique avec une annotation car cela bug, je dois avoir une mauvaise combinaison de version entre php, netbean et symfony. On peut voir que l'Assert n'est pas pris en compte.

Et pour une raison inconnu si j'utilise un attribut l'IDE m'indique que ce n'est pas compatible mais en fait cela marche bien...

```

use Symfony\Component\Validator\Constraints as Assert;


```

/**
 * @ORM\Entity(repositoryClass=FormationRepository::class)
 */
class Formation
{
 /**
 * Début de chemin vers les images
 */
 private const CHEMIN_IMAGE = "https://i.ytimg.com/vi/";

 /**
 * @ORM\Id
 * @ORM\GeneratedValue
 * @ORM\Column(type="integer")
 */
 private $id;

 /**
 * @ORM\Column(type="datetime", nullable=true)
 */
 private $publishedAt;

```



Language feature not compatible with PHP version indicated in project settings  

----  

(Alt-Enter shows hints)



```

#[Assert\LessThanOrEqual("now")]
private $publishedAt;

```


```

Une fois le test corrigé, je fais les méthodes pour tester une date antérieure et la date du jour.

```
/*
 * Essai une date antérieure à la date du jour
 * (Doit rendre une réussite)
 */
public function testValideDateFormationB() {
    $formation=$this->getFormation()->setPublishedAt((new \DateTime("1900-01-04")));
    $this->assertErrors($formation, 0);
}

/*
 * Essai la date du jour
 * (Doit rendre une réussite)
 */
public function testValideDateFormationC() {
    $formation=$this->getFormation()->setPublishedAt((new \DateTime("now")));
    $this->assertErrors($formation, 0);
}
```

Et les trois tests donnent les résultats attendus.

je mets à jour mon travail sur Github et passe aux prochains tests.

```
E:\wamp64\www\mediatekformation>php bin/phpunit --filter FormationValidationsTest
PHPUnit 9.5.23 #StandWithUkraine

Testing
...
3 / 3 (100%)

Time: 00:01.005, Memory: 28.00 MB

OK (3 tests, 3 assertions) ←
```

Tests d'intégration sur les Repository :

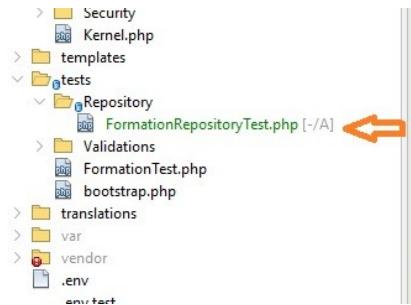
Tâches à réaliser

Le but est de contrôler toutes les méthodes ajoutées dans les classes Repository.

Pour commencer je crée via phpMyAdmin une BDD qui va contenir les différents tests qui va se nommer mediatekformation_test. En effet par défaut symfony effectue les tests sur une BDD dont le nom est celui de la BDD suivi de _test.



Je crée dans le dossier **tests** un dossier **Repository** dans le quel je crée la classe **FormationRepositoryTest.php** qui va tester toutes les méthodes de **FormationRepository**.



```
> Security
  Kernel.php
> templates
> tests
  <-- FormationRepositoryTest.php [-/A] ← Red arrow
  <-- FormationTest.php
  <-- bootstrap.php
> translations
> var
> vendor
  .env
  env_start
```

```
12   * @author hedri
13   */
14  class FormationRepositoryTest extends KernelTestCase {
15    /**
16     * Permet d'accéder au kernel et de récupérer l'instance du repository
17     */
18
19    public function recupererRepository(): FormationRepository {
20      self::bootKernel();
21      $repository = self::getContainer()->get(FormationRepository::class);
22      return $repository;
23    }
24
```

La première méthode **recupererRepository()** me permet d'accéder au kernel et de récupérer l'instance du repository.

La méthode **newFormation()** me permet de créer une formation qui va être utilisée pour les tests.

```
public function newFormation(): Formation {
    $formation = (new Formation())
        ->setTitre("Formation test");
    return $formation;
```

La méthode **testAddFormation()** me permet de tester l'ajout d'une formation.

```
public function testAddFormation() {
    $repository = $this->recupererRepository();
    $formation = $this->newFormation();
    $nbFormation = $repository->count([]);
    $repository->add($formation, true);
    $this->assertEquals($nbFormation + 1, $repository->count([]), "erreur sur testAddFormation()");
}
```

Explication

Pour se faire la méthode récupère le repository de la méthode **recupererRepository()**

Crée une formation.

Compte le nombre de formation contenu dans la bdd.

Ajoute la formation créée.

Compare le nombre de formation dans la bdd en sachant qu'il doit être incrémenté de 1. Dans le cas contraire retourne un message d'erreur personnalisé, ici le nom de la méthode de test.

La méthode **testRemoveFormation()** fonctionne sur le même principe que la précédente.

```
public function testRemoveFormation() {
    $repository = $this->recupererRepository();
    $formation = $this->newFormation();
    $repository->add($formation, true);
    $nbFormation = $repository->count([]);
    $repository->remove($formation, true);
    $this->assertEquals($nbFormation - 1, $repository->count([]), "erreur sur testRemoveFormation()");
}
```

La différence est que le test doit être décrémenté de 1.

```

public function testFindAllOrderBy() {
    $repository = $this->recupRepository();
    $nbformation = $repository->count([]);
    $formation = $repository->findAllOrderBy('title', 'ASC', '');
    $nb2formation = count($formation);
    $this->assertEquals($nbformation, $nb2formation, "erreur sur testFindAllOrderBy()");
}

```

La méthode `testFindAllOrderBy()`

Récupère le repository de la méthode `recupRepository()`

Compte le nombre de formation.

Fait appelle à la méthode `findAllOrderBy()` en lui donnant des paramètres normalement attendu.

Compte le nombre de formation retourné

Compare les deux résultats.

```

public function testFindByContainValue() {
    $repository = $this->recupRepository();
    $formation = $this->newFormation()
        ->setTitle("Essai de containeValue");
    $repository->add($formation, true);
    $uneformation = $repository->findByContainValue('title', 'Essai de containeValue', '');
    $nbformation = count($uneformation);
    $this->assertEquals($nbformation, 1, "erreur sur testFindAllOrderBy()");
    $repository->remove($formation, true);
}

```

La méthode `TestfindByContainValue()`

Récupère le repository de la méthode `recupRepository()`

Crée une formation avec un titre unique à cette méthode.

Ajoute la formation.

Cherche cette formation au titre unique en utilisant la méthode `findByContainValue()`

Compte le résultat.

Compare pour savoir si est trouvé la formation en question

J'efface cette formation spécifique de la bdd pour ne pas fausser un prochain test.

Tout les tests sont valides.

```

E:\wamp64\www\mediatekformation>php bin/phpunit --filter FormationRepositoryTest
PHPUnit 9.5.23 #StandWithUkraine

Testing
....                                         4 / 4 (100%)

Time: 00:01.411, Memory: 30.00 MB

OK (4 tests, 4 assertions)

E:\wamp64\www\mediatekformation>

```

Je procède de la même manière pour les méthodes de `PlaylistRepository` et `CategorieRepository` en prenant en compte leur caractéristiques.

Tests fonctionnels :

Tâche à réaliser

Le but est de contrôler que la page d'accueil est accessible. Dans chaque page contenant des listes, que les tris fonctionnent, que les filtres fonctionnent et que le clic sur un lien (ou bouton) dans une liste permet d'accéder à la bonne page.

Pour les faire, je crée dans le dossier [tests](#), un dossier [Controller](#) dans lequel je fais la classe [AccueilControllerTest.php](#).

```
class AccueilControllerTest extends WebTestCase {

    /**
     * Contrôle que la page d'accueil est accessible.
     */
    public function testAccesAccueil() {
        $client = static::createClient();
        $client->request('GET', '/');
        $this->assertResponseStatusCodeSame(Response::HTTP_OK)
    }
}
```

Explication :

On crée un client.

On envoie une requête sur le lien de la page d'accueil « / »

L'appel de [assertResponseStatusSame\(\)](#) permet de comparer le code http renvoyé avec celui mis en parenthèse.

Je procède de la même manière pour les playlists et formations.

Ce test sur les formations me fait corriger mon fichier [formations.html.twig](#)

```
@@ -65,7 +65,7 @@
65   65           </h5>
66   66           </td>
67   67           <td class="text-left">
68   -           {{ formation.playlist.name }} 
68   +           {% if formation.playlist.name is defined %} {{ formation.playlist.name }} {% endif %}
```

Pour le contrôle des tris je fais une méthode [testTriFormation\(\)](#).

```
public function testTriFormation(){
    $client = static::createClient();
    $client->request('GET', '/formations/tri/title/ASC');
    $this->assertSelectorTextContains('h5', 'Android Studio (complément n°1)');
}
```

Explication :

Créer un client.

Il fait une requête sur un tri, dans cette exemple le tri sur les titres dans l'ordre alphabétique.

Compare le résultat obtenu sur la balise h5 ici le titre Android Studio..

J'utilise le même process pour les autres tris, envoyer un tri et comparer que le résultat attendu est cohérent.

Pour le contrôle des recherche, je fais une méthode `testFiltreFormation()`

```
public function testFiltreFormation() {
    $client = static::createClient();
    $client->request('GET', '/formations');
    $crawler = $client->submitForm('filtrer', [
        'recherche' => 'Eclipse n°7 : Tests unitaires'
    ]);
    $this->assertCount(1, $crawler->filter('h5'));
    $this->assertSelectorTextContains('h5', 'Eclipse n°7 : Tests unitaires');
}
```

Explication :

Créer un client.

Il fait une requête sur la page formations.

On simule la soumission d'un formulaire en précisant le contenu (Eclipse n°...)

Je vérifie le nombre de ligne obtenu

Je vérifie que le contenu soit bien (Eclipse n°...)

Je procède de même pour playlist avec un contenu compatible.

Pour le test du clic sur un lien je fais une méthode `testLinkFormtion()`

```
public function testLinkFormtion() {
    $client = static::createClient();
    $client->request('GET', '/formations');
    $crawler = $client->getCrawler();
    $link = $crawler->selectLink('image de formation')->link();
    $client->click($link);
    $response = $client->getResponse();
    $this->assertEquals(Response::HTTP_OK, $response->getStatusCode())
}
```

Explication

Créer un client pour simuler une requête.

Effectue la requête GET sur /formation.

Récupère le Crawler, qui permet de parcourir et manipuler le contenu.

Sélectionne le lien.

Effectue un clic sur le lien.

Récupère la réponse HTTP.

Vérifie que le statut de la réponse est HTTP_OK.

Ensuite pareille pour playlist.

Test final :

```
E:\wamp64\www\mediatekformation>php bin/phpunit
PHPUnit 9.5.23 #StandWithUkraine

Testing
.....
24 / 24 (100%)

Time: 00:07.838, Memory: 42.00 MB

OK (24 tests, 42 assertions)
```

Tests de compatibilité :

Tâche à réaliser

Le but est de créer un scénario avec Selenium sur la partie front office et le jouer sur plusieurs navigateurs pour tester la compatibilité du site.

Je crée un scénario qui consiste en la manipulation de divers pages, divers recherches, divers click...

Search tests...		http://192.168.1.108/mediatekformation/public/index.php	
Untitled		Command	Target
	✓ test1	12	✓ click css=.text-center > .b Id(3)
		13	✓ click css=.align-middle:n (2) img
		14	✓ click linkText=Playlists
		15	✓ click linkText=>
		16	✓ click linkText=<
		17	✓ click css=.text-left:nth-chi n:nth-child(2)
		18	✓ click css=.text-left:nth-chi n:nth-child(3)
		19	✓ click id=recherche
		20	✓ select id=recherche
		21	✓ click css=option:nth-chilc
		22	✓ click linkText=Voir détail

Puis sélenium me permet de répéter ce test sur différent navigateur de manière automatisé, ici avec firefox

```

C:\Users\GT630M>selenium-side-runner -s http://192.168.1.108:4444 -c "browserName=firefox"
mediatek.side
info: Running test Untitled
info: Building driver for firefox
info: Driver has been built for firefox
info: Finished test Untitled Success
PASS AppData/Roaming/npm/node_modules/selenium-side-runner/dist/main.test.js (9.414 s)
  Running project testmediatek
    Running suite Default Suite
      ✓ Running test Untitled (6829 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        9.654 s
Ran all test suites within paths "C:\Users\GT630M\AppData\Roaming\npm\node_modules\selenium-side-runner".
C:\Users\GT630M>

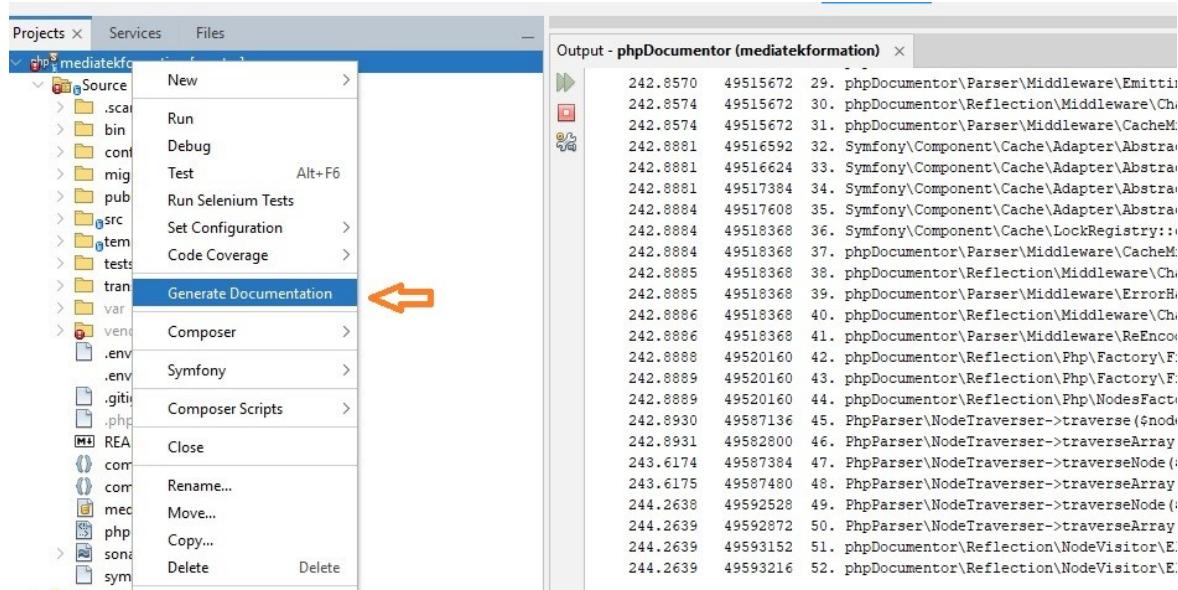
```

Une fois que tout les tests fonctionnent, je « commit » et « push » sur mon dépôt GitHub et mets à jour mon kanban pour passer à la prochaine tâche.

Sur les 7h estimés le temps réel pour réaliser cette tâche est de 11h. Le bug sur l'assertion et l'échec du test fonctionnel sur l'accès de la page formations étaient chronophages à comprendre et résoudre.

Tâche 2 : créer la documentation technique

Pour générer la documentation technique sous netBean, il faut ajouter phpDocumentor.phar puis configurer netBean



Sur les 1h estimés le temps réel pour réaliser cette tâche est de 1h.

Tâche 3 : créer la documentation utilisateur

Pour se faire j'utilise le logiciel OBS Studio, cette documentation est téléchargeable depuis le GitHub du projet.

Mission 4 : déployer le site et gérer le déploiement continu.

Tâche 1 : déployer le site.

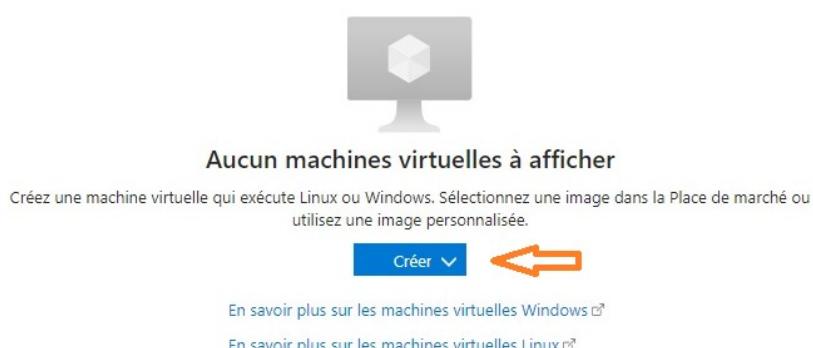
Tâches à réaliser

- Installer et configurer le serveur d'authentification Keycloak dans une VM en ligne (voir l'article "Keycloak en ligne et en HTTPS" dans le wiki du dépôt).
- 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

Keycloak

Dans un premier temps il faut installer et configurer le serveur d'authentification Keycloak dans une VM en ligne.

Pour se faire, j'utilise la solution de Microsoft Azure pour héberger et créer une machine virtuelle.



Je configure une machine virtuelle sous Ubuntu Server 20.04LTS.

The screenshot shows the Azure portal's configuration interface for creating a new virtual machine. It includes fields for 'Abonnement' (Subscription) set to 'Azure for Students', 'Groupe de ressources' (Resource Group) set to '(Nouveau) GroupeRessourceRealisationPro1', and 'Nom de la machine virtuelle' (Virtual machine name) set to 'machineVirtuelKeycloak'. Under 'Détails de l'instance' (Instance details), 'Région' (Region) is set to '(Europe) France Central', 'Options de disponibilité' (Availability options) shows 'Zone de disponibilité' (Availability zone) as 'Zones 1', and 'Type de sécurité' (Security type) is 'Standard'. In the 'Image' (Image) section, 'Ubuntu Server 20.04 LTS - x64 de 2e génération' is selected. A note at the bottom right of this section says: 'Vous pouvez désormais sélectionner plusieurs zones. La sélection de plusieurs zones crée une machine virtuelle par zone. En savoir plus'.

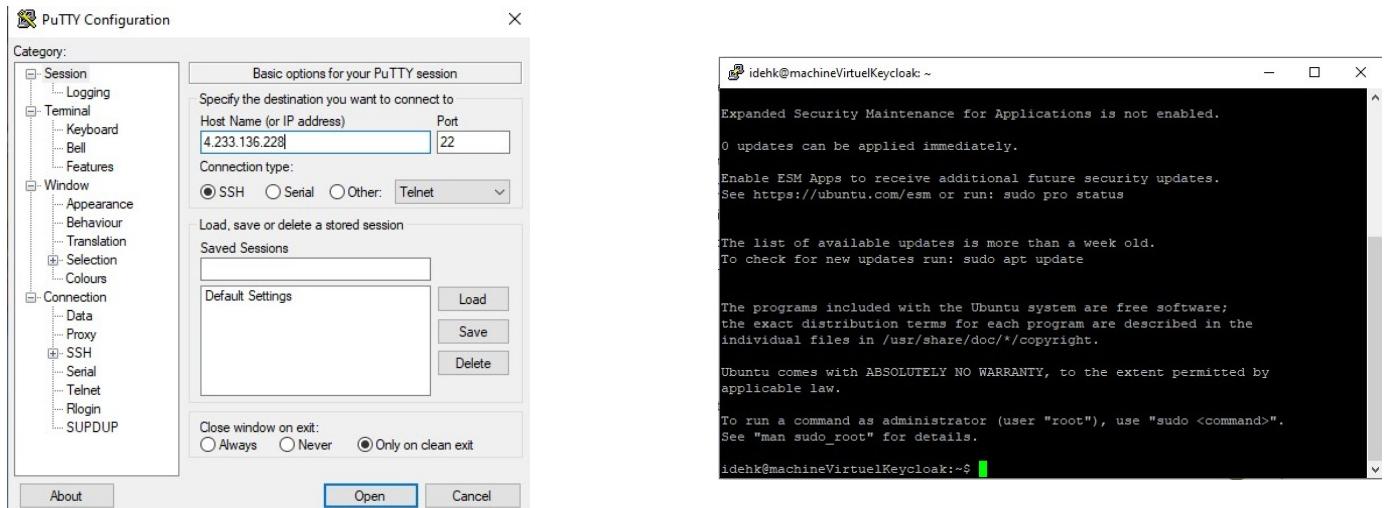
Configure le DNS et ouvre les ports nécessaires sur le serveur.

```

Système d'exploitation : Linux (ubuntu 20.04) ←
Taille : Standard B1s (1 processeur virtuel, 1 Gio de mémoire)
Adresse IP publique : 20.19.39.71 ←
Réseau/sous-réseau virtuel : vmKeycloak-vnet/default
Nom DNS : vmkeycloakdns.francecentral.cloudapp.azure.com ←
État d'intégrité : -

```

Une fois le serveur configuré et en ligne, je me connecte via putty sur l'ip de la VM et je me log avec l'id et le mot de passe que j'ai créé quand j'ai configuré la vm.



Dans la machine virtuelle j'installe et configure le nécessaire pour faire fonctionner keycloak, JDK18.0.8, keycloak 19.0.1, Apache2, Cerbot et Screen.

<http://vmkeycloakdns.francecentral.cloudapp.azure.com/>



Cerbot me permet d'avoir un certificat pour mon dns.

```

Deploying certificate
Successfully deployed certificate for vmkeycloakdns.francecentral.cloudapp.azure.com to /etc/apache2/sites-available/
000-default-le-ssl.conf
Congratulations! You have successfully enabled HTTPS on https://vmkeycloakdns.francecentral.cloudapp.azure.com

-----
If you like Certbot, please consider supporting our work by:
 * Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
 * Donating to EFF:                   https://eff.org/donate-le
-----
idehk@vmKeycloak:~$ 

```

Ensuite il est m'est permis de lancer le service keycloak.

je me connecte à keycloak avec l'adresse du DNS configuré dans Azure et je configure royaume, client et utilisateur comme pour le serveur local.

The screenshot shows the Keycloak Admin Console with the URL <https://vmkeycloakdns.francecentral.cloudapp.azure.com/admin/master/console/#/master>. The top navigation bar includes links for Ressources cours, A1, A3, Site dev., Connexion à votre e..., GitHub, Portfolio, and Brevet de tech. The main header features the Keycloak logo and the text "KEYCLOAK". Below the header, there's a "Master" dropdown menu and the text "Master realm". An orange arrow points from the left side of the screen towards the Keycloak logo.

Je crée un royaume.

The screenshot shows the "Create realm" form. It includes fields for "Resource file" (with a placeholder "Drag a file here or browse..."), "Realm name" (set to "myapplis" with an orange arrow pointing to it), and "Enabled" (set to "On"). To the right, a code editor displays a YAML configuration file named "knpu_oauth2_client.yaml". The file contains a section for "clients" under "keycloak" with the following values:

```
knpu_oauth2_client:
  clients:
    keycloak:
      type: keycloak
      auth_server_url: '$env(KEYCLOAK_APP_URL)'
      realm: 'myapplis'
      client_id: '$env(KEYCLOAK_CLIENTID)'
      client_secret: '$env(KEYCLOAK_SECRET)'
      redirect_route: 'oauth_check'
```

An orange arrow points from the right side of the screen towards the "realm" value in the configuration file.

Je crée un client.

The screenshot shows the "Create client" form. It has a step indicator "1 General Settings" and a "Client type" field set to "OpenID Connect". In the "Client ID" field, the value "mediatek86" is shown with an orange arrow pointing to it. Below the form, a snippet of configuration code is displayed:

```
#keycloak en ligne sur AZURE
#KEYCLOAK_SECRET=fHjdzPS7CjgtpmGsWIPrPDuzgINjovGO
#KEYCLOAK_CLIENTID=mediatek86
#KEYCLOAK_APP_URL=https://vmkeycloak.francecentral.cloudapp.azure.com
```

An orange arrow points from the right side of the screen towards the "Client ID" value in the configuration snippet.

Le client est configuré comme cela :

Client type : OpenID Connect

- Always display in console : Off
- Client authentication : On

- Authorisation : Off
- Standard flow : On
- Implicit flow : On
- Direct access grants : On
- Service accounts roles : Off
- OAuth 2.0 Device Authorization Grant : Off
- OIDC CIBA Grant : Off
- Enabled : On (en haut à droite)
- Valid redirect URIs (dans Access settings) : *
- Consent required (dans Login settings) : On
- Display client on screen : On
- Front channel logout (dans logout settings) : Off
- Backchannel logout session required : On
- Backchannel logout revoke offline sessions : Off

Et je crée un utilisateur.

Je mets à jour dans mon application les paramètres de keycloak nécessaires.

Site en ligne

Pour déployer le site, j'utilise planethoster, cet hébergeur propose assez d'espace de stockage pour un projet symfony en version gratuite.

Après avoir créé un compte, il me suffit de transférer le dossier de l'application web via un client FTP (FileZilla)

Planethoster fourni PhpMyAdmin, il me suffit de créer une BDD et de modifier .ENV dans l'application pour configurer l'accès à cette BBD et d'importer le script SQL.



Je modifie la CGU sur mon adresse.

Je déploie la documentation technique sur mon GitHub.
Finalement le site est accessible en ligne.

<https://mattek.go.yj.fr>

A screenshot of the MediaTek86 website. The header shows the URL https://mattek.go.yj.fr/mediatekformation/public/. Below the header is a large image of a laptop screen displaying a room full of people, symbolizing online learning. To the right of the image, the text "MediaTek86" is written in a large, blue, sans-serif font. Below this, the tagline "Des formations pour tous sur des outils numériques" is displayed in a smaller, black font. At the bottom of the header, there is a navigation bar with links for "Accueil", "Formations", and "Playlists".

Bienvenue sur le site de MediaTek86 consacré aux formations en ligne

Vous allez pouvoir vous former à différents outils numériques gratuitement et directement en ligne.

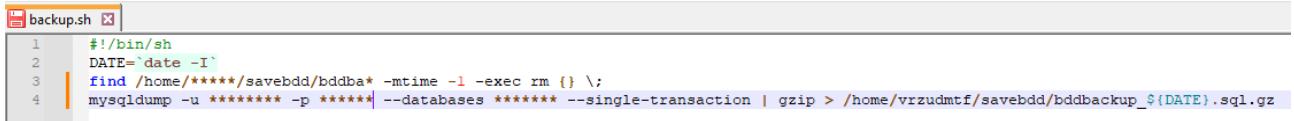
Dans la partie [Formations](#), vous trouverez la liste des formations proposées. Vous pourrez faire des recherches et des tris. En cliquant plus détaillée de la formation ainsi que la vidéo correspondante.

Vous pouvez aussi retrouver les vidéos regroupées dans des playlists, dans la partie [Playlists](#).

Une fois que tout fonctionne, je mets à jour mon kanban pour passer à la prochaine tâche.
Sur les 2h estimés le temps réel pour réaliser cette tâche est de 2h

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

Pour se faire il faut créer un scripte



```
backup.sh
1 #!/bin/sh
2 DATE=`date -I`
3 find /home/*****/savebdd/bddba* -mtime -1 -exec rm {} \;
4 mysqldump -u ***** -p ***** --databases ***** --single-transaction | gzip > /home/vrzudmtf/savebdd/bddbackup_${DATE}.sql.gz
```

Et configurer une exécution de ce scripte quand on souhaite faire une sauvegarde de la BDD.

Une fois que tout fonctionne, je mets à jour mon kanban pour passer à la prochaine tâche.
Sur les 1h estimé le temps réel pour réaliser cette tâche est de 1h

PARAMÈTRES COMMUNS

Une fois par jour(0 0 ***)

MINUTE

0 0

HEURE

0 0

JOUR DU MOIS

* Chaque jour (*)

MOIS

* Chaque mois (*)

JOUR DE LA SEMAINE

* Chaque jour (*)

COMMANDÉ

/home/[logincompte]/savebdd/backup.sh

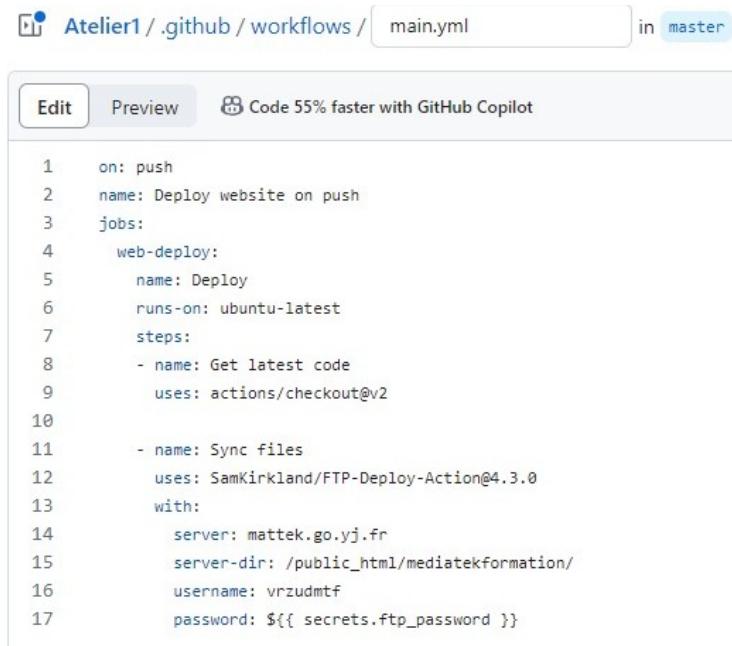
Tâche 3 : mettre en place le déploiement continu

Le but est de faire en sorte qu'à chaque push vers GitHub, le site en ligne soit aussi mis à jour.

Dans un premier temps je crée le fichier yml d'automatisation.

Dans le dépôt GitHub qui contient le projet, je sélectionner "Actions" puis "set up a workflow yourself et j'entre

Puis dans mon IDE je récupère cette ajout.



```
Atelier1/.github/workflows/main.yml in master

Edit Preview Code 55% faster with GitHub Copilot

1 on: push
2   name: Deploy website on push
3   jobs:
4     web-deploy:
5       name: Deploy
6       runs-on: ubuntu-latest
7       steps:
8         - name: Get latest code
9           uses: actions/checkout@v2
10
11        - name: Sync files
12          uses: SamKirkland/FTP-Deploy-Action@4.3.0
13          with:
14            server: mattek.go.yj.fr
15            server-dir: /public_html/mediatekformation/
16            username: vrzudmtf
17            password: ${{ secrets.ftp_password }}
```

Et dans le dépôt GitHub j'enregistre le mot de passe pour accéder au ftp du site

Actions secrets / Update secret

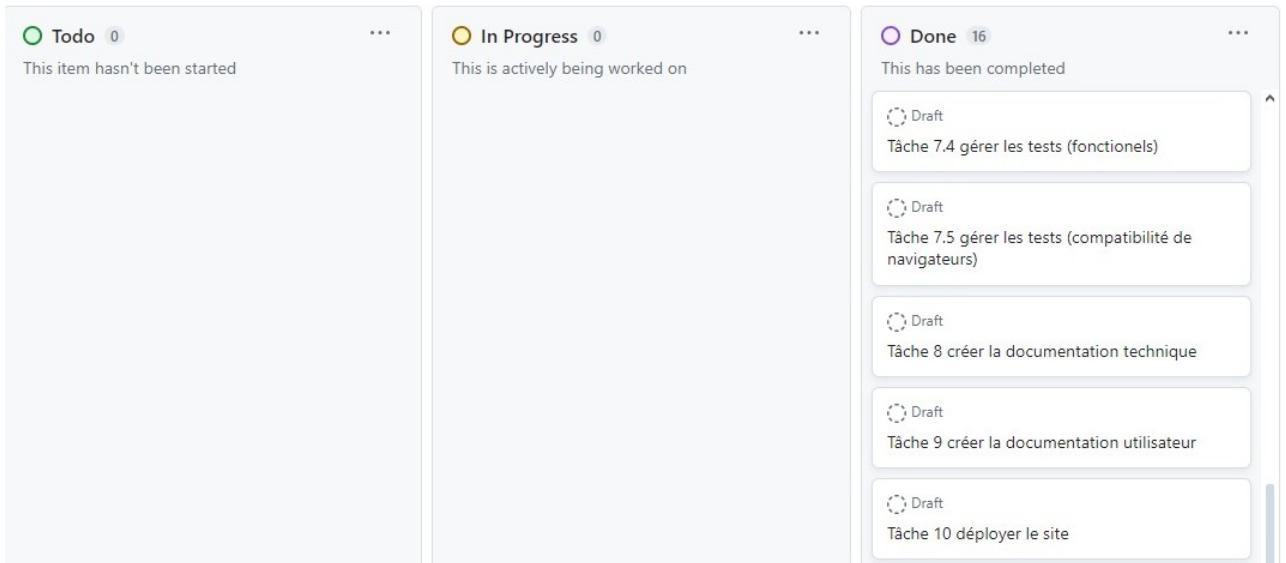
FTP_PASSWORD

Value
Password pour accéder au ftp du site.

Update secret

Ensuite tout les prochains commit et push mettrons le site à jour automatiquement.

Une fois que tout fonctionne, je mets à jour mon kanban



Bilan

Tout les objectif ont étaient atteints, le site est fonctionnel, la VM pour keycloak aussi. Cette réalisation était relativement chronophage, du à des problèmes de comptabilité entre mes divers outils, ma version de PHP ma porté préjudice. Cependant cela n'enlève rien au plaisir de faire cette réalisation et d'avoir un résultat pleinement fonctionnelle !