



# ATELIER PROFESSIONNEL MEDIATEK FORMATION

**COMPTE-RENDU**

Réalisé par **Bilel Ali Moussa**

## Mission 1: Nettoyer et optimiser le code existant

### Tâche 1: Nettoyer le code

1. Nettoyer le code en suivant les indications de Sonarlint.
2. Éviter les chaînes "en dur".
3. Nommer les constantes en majuscule.
4. Fusionner certains tests imbriqués inutilement.
5. Ajouter l'attribut "alt" à toutes les images.
6. Ajouter l'attribut "description" à toutes les tables.

### Tâche 2 : Ajouter une fonctionnalité

1. Ajouter une colonne dans la page des playlists pour afficher le nombre de formations par playlist.

## Mission 2 : Coder la partie back-office

### Tâche 1 : Gérer les formations

1. Créer une page pour gérer les formations avec ajout.
2. Afficher la liste des formations.
3. Permettre la suppression d'une formation.
4. Permettre la modification d'une formation.

### Tâche 2 : Gérer les playlists

1. Création de la page des playlists.
2. Afficher la liste des playlists.
3. Permettre la suppression d'une playlist.
4. Permettre la modification d'une playlist.
5. Permettre l'ajout d'une playlist.

### Tâche 3 : Gérer les catégories

1. Créer une page pour gérer les catégories avec ajout.
2. Afficher la liste des catégories.
3. Permettre la suppression d'une catégorie.
4. Permettre l'ajout direct d'une nouvelle catégorie.

### Tâche 4 : Ajouter l'accès avec authentification

1. Ajouter l'authentification avec Keycloak et gérer l'accès à la partie admin.

## Mission 3 : Effectuer les tests

### Tâche 1 : Gérer les tests

1. Effectuer différents types de tests (unitaires, d'intégration, fonctionnels, de compatibilité) sur l'application.

### Tâche 2 : Créer la documentation technique

1. Générer la documentation technique du site complet.

### Tâche 3 : Créer la documentation utilisateur

1. Créer une vidéo de démonstration des fonctionnalités du site.

## **Mission 4 : Déployer le site et gérer le déploiement continu**

### **Tâche 1 : Déployer le site**

1. Configuration du serveur keycloak en HTTPS.
2. Déployer la base de données.
3. Déployer le site.

### **Tâche 2 : Gérer la sauvegarde et la restauration de la base de données**

### **Tâche 3 : Mettre en place le déploiement continu**

#### **Contexte :**

MediaTek86 a pour rôle de fédérer les prêts de livres, DVD et CD et de développer la médiathèque numérique pour l'ensemble des médiathèques du département.

Afin de donner plus d'attractivité aux médiathèques, MediaTek86 veut se développer selon deux axes :

enrichir ses services en offrant aux adhérents la possibilité d'emprunter des films en VOD ; proposer, en complément de l'activité principale de la médiathèque, des formations aux outils numériques et des autoformations en ligne.

#### **Mission :**

Le chef de projet a contrôlé le travail du premier développeur et a constaté l'oubli de plusieurs fonctionnalités attendues dans le cahier des charges. Il m'a chargé de corriger ces problèmes, puis plusieurs autres missions m'ont été confiées (en particulier le back-office) pour finaliser et déployer le site.

#### **Langages et technologies :**

IDE : netbeans

Langages de programmation : php twig

Authentification : keycloak

Framework : symfony

Serveur : wampserver apache mysql php

VM : linux

Versionning : github

Hebergeur hostinger

# Mission 1 : Nettoyer et optimiser le code existant

## 1. Tâche 1 : Nettoyer le code

Temps estimé: 2h | Temps réalisé: 2h

Mission :

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" ).

**Tâche 1 : nettoyer le code #6** Edit New issue

Open Codeuraxe opened this issue 1 minute ago · 0 comments

**Codeuraxe** commented 1 minute ago Owner ...

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").

En rappel :

- Éviter les chaînes "en dur" (pour éliminer les "strings literals duplicated").
- Nommer les constantes en majuscule.
- Fusionner certains tests imbriqués inutilement.
- Ajouter l'attribut "alt" à toutes les images.
- Ajouter l'attribut "description" à toutes les tables.

**Assignees**  
No one — [assign yourself](#)

**Labels**  
None yet

**Projects**  
@Codeuraxe's untitled project  
Status: In Progress

**Milestone**  
No milestone

## 1 : CONSTANT NAMES SHOULD COMPLY WITH A NAMING CONVENTION

« Formation.php »

SonarLint suggère que le nom d'une constante dans le code ne respecte pas la convention de nommage.

```
/**
 * Début de chemin vers les images
 */
private const cheminImage = "https://i.ytimg.com/vi/";

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

Les constantes doivent toujours s'écrire en majuscule, je procède aux modifications :

```
/**
 * Début de chemin vers les images
 */
private const CHEMINIMAGE = "https://i.ytimg.com/vi/";
```

```

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

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

```

## 2.1 : STRING LITERALS SHOULD NOT BE DUPLICATED « FormationsController »

L'erreur suggère qu'il est préférable de ne pas répéter les mêmes valeurs de chaînes dans le code. Il faut donc définir des constantes pour les chaînes de caractères qui se répètent.

```

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, $stable);
    $categories = $this->categorieRepository->findAll();
    return $this->render("pages/formations.html.twig", [
        'formations' => $formations,
        'categories' => $categories
    ]);
}

```

"pages/formations.html.twig" se répète plusieurs fois. Je définis des constantes avec pour nom "FORMATIONS\_PATH" et "FORMATION\_PATH".

```

private const FORMATIONS_PATH = 'pages/formations.html.twig';
private const FORMATION_PATH = 'pages/formation.html.twig';

```

Ensuite remplaçons les chemins par les constantes affecté :

```

return $this->render(self::FORMATIONS_PATH, [
    'formations' => $formations,
    'categories' => $categories
]);

return $this->render(self::FORMATION_PATH, [
    'formation' => $formation
]);
}

```

## 2.2 : STRING LITERALS SHOULD NOT BE DUPLICATED « PlaylistsController »

Même erreur que dans le 2.1, le chemin "pages/playlists.html.twig" se répète plusieurs fois dans le code je corrige avec le même procédé.

```

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

```

"pages/playslists.html.twig" se répète plusieurs fois. Il faut définir des constantes avec pour nom "PLAYLISTS\_PATH" et "PLAYLIST\_PATH".

```

class PlaylistsController extends AbstractController {

    private const PLAYLISTS_PATH = 'pages/playlists.html.twig';
    private const PLAYLIST_PATH = 'pages/playlist.html.twig';
}

```

Ensuite je remplace les chemins par la constante affecté :

```

return $this->render(self::PLAYLISTS_PATH, [
    'playlists' => $playlists,
    'categories' => $categories
]);
}

```

### 3 : COLLAPSIBLE "IF" STATEMENTS SHOULD BE MERGED « Playlist.php »

Sonarlint indique que des blocs if consécutifs peuvent être combinés en un seul bloc pour rendre le code plus clair.

```

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

```

Je supprime le "if " et imbriquons "&& \$formation->getPlaylist() === \$this)"

```

public function removeFormation(Formation $formation): self
{
    if ($this->formations->removeElement($formation) && $formation->getPlaylist() === $this) {
        $formation->setPlaylist(null);
    }
}

```

### 3 : CONTROLE STRUCTURES SHOULD USE CURLY BRACES « Playlist.php »

Sonarlint suggère que les structures de contrôle (ici "if"), devraient toujours être entourées de crochets {} même lorsqu'elles ne contiennent qu'une seule instruction.

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

Modifications :

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

### 4 : "SWITCH" STATEMENTS SHOULD HAVE "DEFAULT" CLAUSES

« PlaylistsController.php »

Sonarlint recommande une clause default, même si elle est vide. Cela est utile pour traiter les cas où la valeur de \$champ ne correspond à aucune des valeurs spécifiées dans les case.

```
public function sort($champ, $ordre): Response{
    switch($champ){
        case "name":
            $playlists = $this->playlistRepository->findAllOrderByName($ordre);
            break;
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render(PAYLISTS_PATH, [
        'playlists' => $playlists,
        'categories' => $categories
    ]);
}
```

Modifications :

```
public function sort($champ, $ordre): Response{
    switch($champ){
        case "name":
            $playlists = $this->playlistRepository->findAllOrderByName($ordre);
            break;

        default:

            break;
    }
}
```

## 5. A CONDITIONALLY EXECUTED SINGLE LINE SHOULD BE DETONED BY INDENTATION « Playlist.php »

Sonarlint nous recommande de respecter la convention d'indentation.

```
public function removeFormation(Formation $formation): self
{
    {
        if ($this->formations->removeElement($formation) && $formation->getPlaylist() === $this) {
            $formation->setPlaylist(null);
        }
    }

    return $this;
}
```

Modifications :

```
public function removeFormation(Formation $formation): self
{

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


    return $this;
}
```

## 6. IMAGE TAGS SHOULD HAVE AN "ALT" ATTRIBUTES « accueil.html.twig »

Ce message recommande d'ajouter une description avec l'attribut "alt" à chaque balise img <img>.

```
{% if formation.picture %}
    <a href="{{ path('formations.showone', {id:formation.id}) }}">
    
```

Modifications :



```
{% if formation.picture %}
  <a href="{{ path('formations.showone', {id:formation.id}) }}">
    
  </a>
```

## Tâche 2 : ajouter une fonctionnalité :

Temps estimé: 2h | Temps réalisé: 3h

Mission :

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.

### Tâche 2 : ajouter une fonctionnalité #8

Open Codeuraxe opened last week

Codeuraxe now (edited)

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.

Edit

Assignees

Add assignees...

Labels

Add labels...

Milestone

Add milestone...

Status

In Progress

## 1. Affichage dans la vue

### Affichage du nombre de formations dans la page « playlist.html.twig »

Modification de la vue :

Ajout de la commande `{{playlistformations|length}}` pour afficher le nombre de formations dans chaque playlist individuelle.

```
<div class="row mt-3">
  <div class="col">
    <h4 class="text-info mt-5">{{ playlist.name }}</h4>
    <strong>nombre de formations :</strong>
    {{playlistformations|length}}
  <br /><br />
```

### Ajout des boutons de tri dans playlists.html.twig «playlists.html.twig »

Mise à jour de la vue playlists.html.twig :

Ajout des boutons `<` et `>` pour permettre le tri croissant et décroissant.

Chaque bouton pointe vers la route `playlists.sort` avec un paramètre "order" indiquant l'ordre de tri ("ASC" pour ascendant et "DESC" pour descendant).

Appel de la méthode `playlists.sort` pour gérer le tri de la colonne du nombre de formations par playlist.

Nombre Formation<br />

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

Catégories

Nombre Formation

Détail

## Bases de la programmation (C#)

**nombre de formations : 74**

**catégories : C# POO**

### Création de la méthode `findAllOrderByNbFormations` « [PlaylistRepository.php](#) »

Création d'une méthode pour récupérer les playlists triées en fonction du nombre de formations associées à chaque playlist.

```
public function findAllOrderByNbFormations($ordre): array  
{  
    return $this->createQueryBuilder('p')  
        ->leftJoin('p.formations', 'f')  
        ->groupBy('p.id')  
        ->orderBy('COUNT(f.title)', $ordre)  
        ->getQuery()  
        ->getResult();  
}
```

`createQueryBuilder('p')` : base de la requête pour récupérer les données à partir de la table playlist ('p').

`->leftJoin('p.formations', 'f')` : méthode utilisé pour faire une jointure entre la table des playlists ('p') et la table des formations ('f')

`->groupBy('p.id')` : Groupe les résultats par l'identifiant unique de chaque playlist.

`->orderBy('COUNT(f.title)', $ordre)` : Trie les résultats en fonction du nombre de formations associées à chaque playlist.

### Modification de la méthode `sort` dans « [PlaylistsController](#) »

Ajout de :

case "nombreformations":

`$playlists = $this->playlistRepository->findAllOrderByNbFormations($ordre);`

`break;`

Pour appeler la méthode "`findAllOrderByNbFormations`" et récupérer toutes les playlists triées par le nombre de formations, et le résultat est stocké dans la variable "`$playlists`"

```

public function sort($champ, $ordre): Response {
    switch ($champ) {
        case "name":
            $playlists = $this->playlistRepository->findAllOrderByName($ordre);
            break;
        case "nombreformations":
            $playlists = $this->playlistRepository->findAllOrderByNbFormations($ordre);
            break;
        default:
            break;
    }
}

```

La colonne retourne bien le nombre de formation par playlist et permet le tri croissant et décroissant.

Accueil Formations Playlists			
<b>Playlist</b> <div> <div>&lt; &gt;</div> <input type="text"/> <div>filtrer</div> </div>	<b>Catégories</b> <div> <input type="text"/> </div>	<b>Nombre Formation</b> <div> <div>&lt; &gt;</div> </div>	<b>Détail</b>
Cours Informatique embarquée	Cours	1	Voir détail
Cours Merise/2	MCD Cours	1	Voir détail
Cours Modèle relationnel et MCD	MCD Cours	1	Voir détail
Cours de programmation objet	POO Cours	1	Voir détail
Cours Composant logiciel	Cours	2	Voir détail
Cours MCD MLD MPD	MCD Cours	2	Voir détail

Accueil Formations Playlists			
<b>Playlist</b> <div> <div>&lt; &gt;</div> <input type="text"/> <div>filtrer</div> </div>	<b>Catégories</b> <div> <input type="text"/> </div>	<b>Nombre Formation</b> <div> <div>&lt; &gt;</div> </div>	<b>Détail</b>
Bases de la programmation (C#)	C# POO	74	Voir détail
Programmation sous Python	Python POO	19	Voir détail
MCD : exercices progressifs	MCD	18	Voir détail
TP Android (programmation mobile)	Android SQL Java	18	Voir détail
Compléments Android (programmation mobile)	Android	13	Voir détail
Visual Studio 2019 et C#	C# POO	11	Voir détail

## Mission 2 : Coder la partie back office

### Tâche 1 : Gérer les formations :

Temps estimé : 5h | Temps réalisé : 5h

Mission :

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

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

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

Un bouton doit permettre d'accéder au formulaire d'ajout d'une formation. Les saisies doivent être contrôlées. Seul le champ "description" n'est pas obligatoire ainsi que la sélection de catégories (une formation peut n'avoir aucune catégorie). La playlist et la ou les catégories doivent être sélectionnées dans une liste (une seule playlist par formation, plusieurs catégories possibles par formation). La date ne doit pas être saisie mais sélectionnée. Elle ne doit pas être postérieure à la date du jour.

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

Mission 2 : Tâche 1 : gérer les formations #11

Open

Codeuraxe opened now

Edit title

Codeuraxe

now (edited)

Edit

Assignees

Add assignees...

Labels

Add labels...

Milestone

Add milestone...

Status

In Progress

Linked pull requests

No linked pull requests

Repository

mediatekformation-master

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.

```

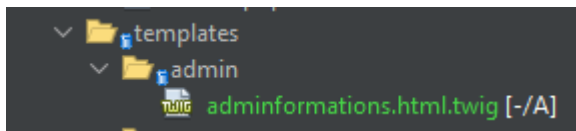
graph LR
    subgraph "Système de gestion des formations"
        direction TB
        UC1([Lister les formations])
        UC2([Supprimer une formation])
        UC3([Modifier une formation])
        UC4([Ajouter une formation])
        UC5([Tris et filtres])
        UC1 -.- UC2
    end
    Admin[Administrateur] --> UC1
    Admin --> UC2
    Admin --> UC3
    Admin --> UC4
    Admin --> UC5

```

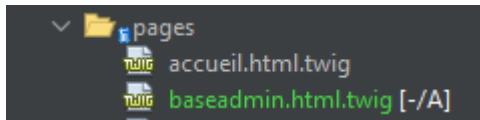
## 1.Lister les Formations

Dans le dossier "controller" je commence par créer un dossier admin qui contiendra le fichier "AdminFormationsController".

Création de la page "admininformations.html.twig " pour lister les formations :



Ensuite je crée le fichier "baseadmin.html.twig" qui va me permettre de définir la structure et le style de toutes les pages.



Dans "AdminFormationsController" je crée la route qui déterminera quelle méthode du contrôleur est exécuté en fonction de l'URL visité par l'utilisateur. Ensuite, je déclare les propriétés privées `$formationRepository`; et `$categorieRepository`; et créer la méthode `index` ainsi que le constructeur.

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

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

    /**
     * Constructeur de AdminFormationsController.
     *
     * @param FormationRepository $formationRepository
     * @param CategorieRepository $categorieRepository
     */
    public function __construct(FormationRepository $formationRepository, CategorieRepository $categorieRepository)
    {
        $this->formationRepository = $formationRepository;
        $this->categorieRepository = $categorieRepository;
    }

    /**
     * @Route("/admin", name="admin. formations")
     *
     * @return Response
     */
    public function index(): Response
    {
        $formations = $this->formationRepository->findAllOrderBy('title', 'ASC');
        $categories = $this->categorieRepository->findAll();

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

Maintenant je crée la structure de "baseadmin.html.twig" et j'intègre un lien vers la page de gestion des formations "admininformations.html.twig".

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

{% block title %}{% endblock %}
{% block stylesheets %}{% endblock %}
{% block top %}
    <div class="container">
        <!-- titre -->
        <div class="text-left">
            
        </div>
        <!-- menu -->
        <nav class="navbar navbar-expand-lg navbar-light bg-light">
            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <ul class="navbar-nav mr-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="{{ path('accueil') }}">Accueil</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ path('formations') }}">Formations</a>
                    </li>
                </ul>
            </div>
        </nav>
    </div>
{% endblock %}
```

## 2. Afficher la liste des formations

Pour parcourir toutes les formations disponibles, une boucle Twig doit être utilisée. Dans le code, cette boucle est définie par `{% for formation in formations %}`.

À l'intérieur de cette boucle, les détails de chaque formation doivent être affichés. Dans le code, chaque détail est présenté dans une cellule de tableau (`<td>...</td>`) ou dans une balise de titre (`<h5>...</h5>`), en utilisant des variables Twig telles que `{{ formation.title }}`, `{{ formation.playlist.name }}`, `{{ formation.categories }}`, `{{ formation.publishedAtString }}`, ainsi que des liens pour les miniatures et les boutons d'édition et de suppression.

```
<tbody>
    {% for formation in formations %}

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

## 3. Permettre la suppression d'une formation

Je commence par définir une route dans le contrôleur `AdminFormationsController.php` pour gérer la suppression d'une formation. `@Route("/formation/delete/{id}", name="admin.formation.delete")`.

Dans le fichier Twig (admin.formations.html.twig), il faut créer un bouton qui permettra la suppression d'une formation et insérer une balise avec un lien vers la route admin.formation.delete. Ensuite il faut une confirmation au bouton ajouté pour éviter les suppressions accidentelles. Pour cela j'utilise l'attribut onclick pour afficher une boîte de dialogue de confirmation.

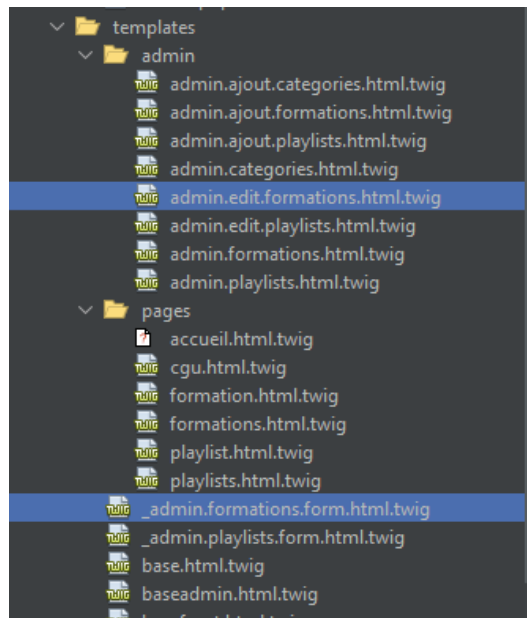


#### 4. Permettre la modification d'une formation

Dans la vue de la liste des formations (admin.formations.html.twig), je crée un bouton d'édition d'une formation.

Lorsque que l'on va cliquer sur le bouton "Editer", cela redirigera vers le formulaire d'édition de la formation avec l'identifiant de la formation passé en paramètre. A l'aide d'un formulaire d'édition (\_admin.formations.form.html.twig), les champs sont préremplis avec les informations de la formation sélectionnée. Lorsque que l'on soumet le formulaire d'édition, il met à jour la formation dans la base de données.

Je vais aussi créer un Formation.Type dans le dossier Form qui va me permettre d'éditer une formation.



\_admin.formations.form.html.twig :

{{ form\_start(form, {'attr': {'class': 'form-horizontal'}}) }} : Démarre le formulaire.

<div class="mt-3"></div> : Ajoute un espace vertical de 3 unités entre les éléments du formulaire.

{{ form\_row(form.title, {'label\_attr': {'class': 'col-sm-2 control-label'}, 'attr': {'class': 'form-control'}})

}} : Génère un champ de formulaire pour le titre de la formation avec un libellé.

Les autres lignes génèrent des champs de formulaire pour la description, la playlist, les catégories et la date de publication de la formation de manière similaire.

`{{ form_end(form) }}` : Termine le formulaire.

```

    {{ form_start(form, {'attr': {'class': 'form-horizontal'}}) }}

    <div class="mt-3"></div>

    <div class="form-group mb-3">
    |   {{ form_row(form.title, {'label_attr': {'class': 'col-sm-2 control-label'}, 'attr': {'class': 'form-control'}}) }}
    | </div>

    <div class="form-group mb-3">
    |   {{ form_row(form.description, {'label_attr': {'class': 'col-sm-2 control-label'}, 'attr': {'class': 'form-control'}}) }}
    | </div>

    <div class="form-group mb-3">
    |   {{ form_row(form.playlist, {'label_attr': {'class': 'col-sm-2 control-label'}, 'attr': {'class': 'form-control'}}) }}
    | </div>

    <div class="form-group mb-3">
    |   {{ form_row(form.categories, {'label_attr': {'class': 'col-sm-2 control-label'}, 'attr': {'class': 'form-control'}}) }}
    | </div>

    <div class="form-group mb-3">
    |   {{ form_row(form.publishedAt, {'label_attr': {'class': 'col-sm-2 control-label'}, 'attr': {'class': 'form-control'}}) }}
    | </div>

    {{ form_end(form) }}

class FormationType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title', TextType::class, [
                'label' => 'Titre de la Formation',
                'required' => true,
            ])
            ->add('description', TextareaType::class, [
                'label' => 'Description',
                'required' => false,
            ])
            ->add('categories', EntityType::class, [
                'class' => Categorie::class,
                'choice_label' => 'name',
                'multiple' => true,
                'required' => false,
            ])
            ->add('publishedAt', DateType::class, [
                'label' => 'Date',
                'widget' => 'single_text'
            ])
            ->add('playlist', EntityType::class, [
                'class' => Playlist::class,
                'choice_label' => 'name',
                'multiple' => false,
            ])
            ->add('submit', SubmitType::class, [
                'label' => 'Enregistrer'
            ]);
    }
}

```

->add('title', TextType::class, [...]): Ajoute un champ pour le titre de la formation.

->add('description', TextareaType::class, [...]): Ajoute un champ pour la description de la formation.

->add('categories', EntityType::class, [...]): Ajoute un champ pour les catégories de la formation, qui sont sélectionnées à partir de l'entité Categorie.

->add('publishedAt', DateType::class, [...]): Ajoute un champ pour la date de publication de la formation.

->add('submit', SubmitType::class, [...]) : Permet d'enregistrer les données.



Ensuite il faut créer la fonction d'édition dans (AdminFormationController.php)

La méthode va permettre de créer le formulaire de modification de formation en utilisant la classe FormationType, et en lui passant la formation existante récupérée à partir de l'identifiant fourni dans l'URL.

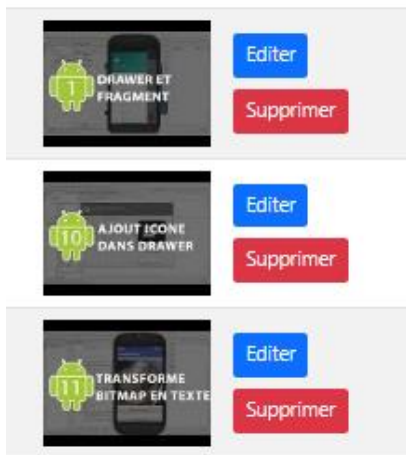
\$formformation->handleRequest(\$request);: Cette ligne traite la requête HTTP actuelle pour le formulaire. Cela permet au formulaire de récupérer les données soumises par l'utilisateur et de les lier à l'objet Formation. Ensuite il faut inclure un if qui va vérifier si le formulaire est soumis et valide.

Dans ce cas, les modifications sont enregistrées dans la base de données en appelant la méthode flush() sur l'EntityManager, qui est responsable de la gestion des entités.

Enfin, l'utilisateur est redirigé vers une autre page, vers la liste des formations.

Si le formulaire n'est pas encore soumis, la méthode affiche à nouveau le formulaire de modification avec les erreurs éventuelles pour que l'utilisateur puisse les corriger.

Ajoutons le bouton d'édition avec le chemin qui redirige vers (admin.edit.formations)



Titre de la Formation

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

Description

Apprendre à créer une navigation type drawer (menu qui apparaît à gauche) et y intégrer des fragments (pour l'affichage des différentes pages). Apprendre à insérer une Activity existante dans un fragment.

Playlist

Compléments Android (programmation mobile)

Categories

UML  
C#  
Python  
MCD  
Android

Date

09/07/2018

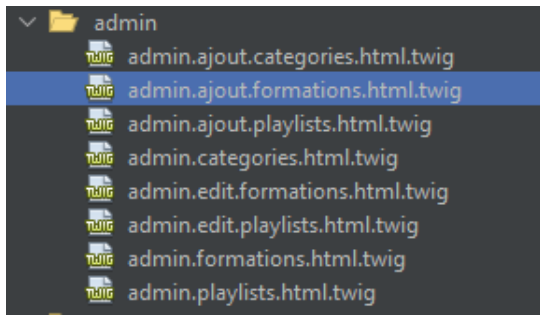
Enregistrer

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

## 5. Permettre l'ajout d'une formation

Avec le même procédé je commence par créer la vue (admin.ajout.formations.html.twig) qui va inclure (\_admin.formation.form.html.twig) ainsi que la méthode d'ajout dans (AdminFormationsController).

Enfin il faudra ajouter le bouton et le chemin de la fonction d'ajout, dans la page (admin.formations.html.twig).



Dans AdminFormationsController avec le même code en modifiant les routes :

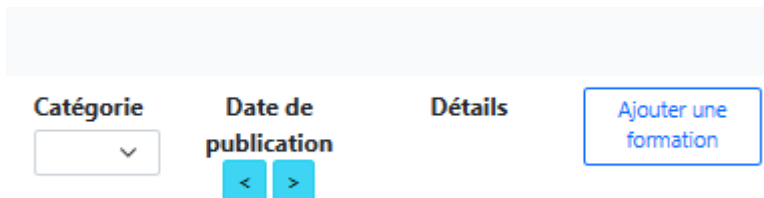
```
/**
 * @Route("/ajout", name="admin.ajout. formations")
 */
public function ajout(Request $request, EntityManagerInterface $entityManager): Response
{
    $formations = new Formation();
    $formformation = $this->createForm(FormationType::class, $formations);
    $formformation->handleRequest($request);

    if ($formformation->isSubmitted() && $formformation->isValid()) {
        $entityManager->persist($formations); // Ajoutez cette ligne pour persister l'entité
        $entityManager->flush();
        return $this->redirectToRoute('admin. formations');
    }

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

Dans (admin. formations.html.twig) j'insère le bouton d'ajout d'une formation et la route, ce qui donne :

```
<th class="text-center align-top" scope="col">
    Détails
</th>
<th class="text-align-top" scope="col" >
<a href="{{ path('admin.ajout. formations') }}" type="button" class="btn btn-outline-primary btn-sm">Ajouter une formation</a>
```



Titre de la Formation

Description

Playlist
Eclipse et Java - Test Ajout

Categories
Java
UML
C#
Python

Date
jj/mm/aaaa

Enregistrer

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

## 6. Permettre les fonctions de tris et filtres

### Tri Ascendant et Descendant

La fonctionnalité de tri pour les formations a été implémentée en créant une fonction `admin.formations.sort` appelée dans le fichier `admin.formations.html.twig`, qui s'active quand on clique sur les boutons "<" (pour le tri ascendant) et ">" (pour le tri descendant). Pour réaliser le tri ascendant et descendant selon le titre des formations, les méthodes `findAllOrderBy` ont été utilisées.

Dans `admin.formations.html.twig`, les boutons de tri sont configurés comme suit :

Chaque bouton contient un lien vers la route `admin.formations.sort` avec des paramètres `champ` et `ordre`. Le paramètre `champ` est défini sur 'title' pour indiquer le champ de tri. Le paramètre `ordre` est défini sur 'ASC' ou 'DESC' pour indiquer la direction du tri.

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

**Formation**

<
>

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

### Tri des Playlists par Nom

De manière similaire, le tri des playlists se fait en utilisant des liens qui pointent vers la même route de tri mais avec des paramètres différents, spécifiant la `table:'playlist'` et le `champ:'name'`, pour trier par le nom de la playlist.

```

<th class="text-left align-top" scope="col">
    Playlist<br />
    <a href="{{ path('admin.formations.sort', {table:'playlist', champ:'name', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"><</a>
    <a href="{{ path('admin.formations.sort', {table:'playlist', champ:'name', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true">></a>
    <form class="form-inline mt-1" method="POST" >
        <div class="form-group mr-1 mb-2">
            <input type="text" class="sm" name="recherche"
                value="{{ if valeur|default and table|default and table=='playlist' %}}{{ valeur }}{% endif % }}"
            <input type="hidden" name="_token" value="{{ csrf_token('filtre_name') }}" >
            <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
        </div>
    </form>

```

### Playlist

Compléments Android (programmation mobile)

## Tri par Date de Publication

Les dates de publication peuvent être triées de façon ascendant et descendant, en utilisant le champ `publishedAt` pour le tri.

```

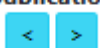
<th class="text-center align-top" scope="col">
    Date de publication<br />
    <a href="{{ path('admin.formations.sort', {champ:'publishedAt', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"><</a>
    <a href="{{ path('admin.formations.sort', {champ:'publishedAt',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

```

Date de  
publication



Date de  
publication



25/09/2016

04/01/2021

Dans `FormationsController.php`, la logique de tri est gérée par :

La méthode `sort` va capturer les paramètres `champ` et `ordre`, puis appelle `findAllOrderBy` sur `formationRepository` en passant ces paramètres. Cela récupère les formations triées selon les critères spécifiés. Ensuite, elle rend la vue `admin.formations.html.twig`

```

/**
 * @Route("/formations/tri/{champ}/{ordre}/{table}", name="formations.sort")
 * @param string $champ
 * @param string $ordre
 * @param string $table
 * @return Response
 */
public function sort($champ, $ordre, $table = ""): Response{
    $formations = $this->formationRepository->findAllOrderBy($champ, $ordre, $table);
    $categories = $this->categorieRepository->findAll();

    return $this->render(self::FORMATIONS_PATH, [
        'formations' => $formations,
    ]);
}

```

La méthode `findAllOrderBy` dans `FormationRepository.php` traite le tri :

```

public function findAllOrderBy($champ, $ordre, $table = ""): array
{
    if ($table === "") {
        return $this->createQueryBuilder('f')
            ->orderBy('f.' . $champ, $ordre)
            ->getQuery()
            ->getResult();
    }
}

```

## Filtres

### Filtrage des Formations par Titre

J'ai implémenté la fonctionnalité de filtrage des formations par titre en utilisant une fonction `findAllContain` dans `AdminFormationsController`. Cette fonction est conçue pour récupérer la valeur entrée dans la zone de recherche et pour filtrer les enregistrements en se basant sur les champs des différentes tables concernées.

Pour faciliter cette recherche côté utilisateur, j'ai ajouté un formulaire de recherche dans `admin.formations.html.twig`. Ce formulaire permet aux utilisateurs de saisir leur recherche

```

<div class="form-group">
    <input type="text" class="form-control" name="recherche" value="{{ valeur }}" />
    <input type="hidden" name="_token" value="{{ csrf_token('filtre_title') }}" />
    <button type="submit" class="btn btn-info">Filtrer</button>
</div>

```

Formation	Playlist	Catégorie	Date de publication	Détails	Ajouter une formation
<div> <div>&lt; &gt;</div> <div>Eclipse n°4 : WindowBuilder</div> <div>filtrer</div> </div>	<div> <div>&lt; &gt;</div> <div></div> <div>filtrer</div> </div>	<div> <div></div> <div></div> </div>	<div> <div>&lt; &gt;</div> <div>09/11/2020</div> </div>	<div> <div>ECLIPSE : WINDOWBUILDER</div> <div>Editer</div> <div>Supprimer</div> </div>	

## Filtrage des Playlists

J'ai également utilisé un formulaire similaire pour le filtrage des playlists, en ajoutant un champ 'name' et en ciblant la table 'playlist'. Cette approche assure une expérience utilisateur cohérente et intuitive à travers différentes sections du site.

```
<th class="text-left align-top" scope="col">
  Playlist<br />
  <a href="{{ path('admin.formations.sort', {table:'playlist', champ:'name', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"></a>
  <a href="{{ path('admin.formations.sort', {table:'playlist', champ:'name', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"></a>
  <form class="form-inline mt-1" method="POST" action="{{ path('admin.formations.findallcontain', {table:'playlist', champ:'name'}) }}">
    <div class="form-group mr-1 mb-2">
      <input type="text" class="sm" name="recherche" value="{{ if valeur|default and table|default and table=='playlist' %}}{{ valeur }}{% endif % }}">
      <input type="hidden" name="token" value="{{ csrf_token('filtre_name') }}">
      <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
    </div>
  </form>
</th>
```

formation	playlist	catégories	date
<div><div>&lt; &gt;</div><div><input type="text"/></div><div>filtrer</div></div>	<div><div>&lt; &gt;</div><div><input type="text" value="mcd"/></div><div>filtrer</div></div>	<div><div>&lt; &gt;</div><div><div></div></div></div>	<div><div>&lt; &gt;</div><div><div></div></div></div>
MCD exercice 18 : sujet 2006 (cas Credauto)	MCD : exercices progressifs	MCD	14/03/2019

## Filtrage des Catégories

Pour filtrer par catégorie, j'ai utilisé un formulaire utilisant un élément select, permettant à l'utilisateur de choisir parmi les catégories disponibles. L'action de ce formulaire est dirigée vers formations.findallcontain avec des paramètres spécifiant champ:'id' et table:'categories'.

```
Catégorie
<form class="form-inline mt-1" method="POST" action="{{ path('formations.findallcontain', {champ:'id', table:'categories'}) }}">
  <select class="form-select form-select-sm" name="recherche" id="recherche" onchange="this.form.submit();">
    <option value=""></option>
    {% for categorie in categories %}
      <option value="{{ categorie.id }}" {{ if valeur|default and valeur==categorie.id %}}selected{% endif %}>
        {{ categorie.name }}
      </option>
    {% endfor %}
  </select>
```

Dans AdminFormationsController :

la fonction findAllContain récupère la valeur de recherche soumise via le formulaire (\$request->get("recherche")), puis appelle la méthode findByContainValue sur le formationRepository avec cette valeur spécifique. Les résultats filtrés sont ensuite envoyés à la vue pour être affichés, permettant ainsi une visualisation efficace et ciblée des informations recherchées par l'utilisateur.

```

/**
 * Récupère les enregistrements selon le $champ et la $valeur
 * Et selon le $champ et la $valeur si autre $table
 * @Route("/formations/recherche/{champ}/{table}", name="formations.findallcontain")
 * @param type $champ
 * @param Request $request
 * @param type $table
 * @return Response
 */
public function findAllContain($champ, Request $request, $table=""): Response{
    $valeur = $request->get("recherche");
    if($table != ""){
        $formations = $this->formationRepository->findByContainValueTable($champ, $valeur, $table);
    }else{
        $formations = $this->formationRepository->findByContainValue($champ, $valeur);
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::FORMATIONS_PATH, [
        'formations' => $formations,
        'categories' => $categories,
        'valeur' => $valeur,
        'table' => $table
    ]);
}

```

## Tâche 2 : gérer les playlists

Temps estimé: 5h | Temps réalisé : 2h


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.



Codeuraxe commented on Jan 26 • edited

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.

Assignees

No one—[assign yourself](#)

Labels

None yet

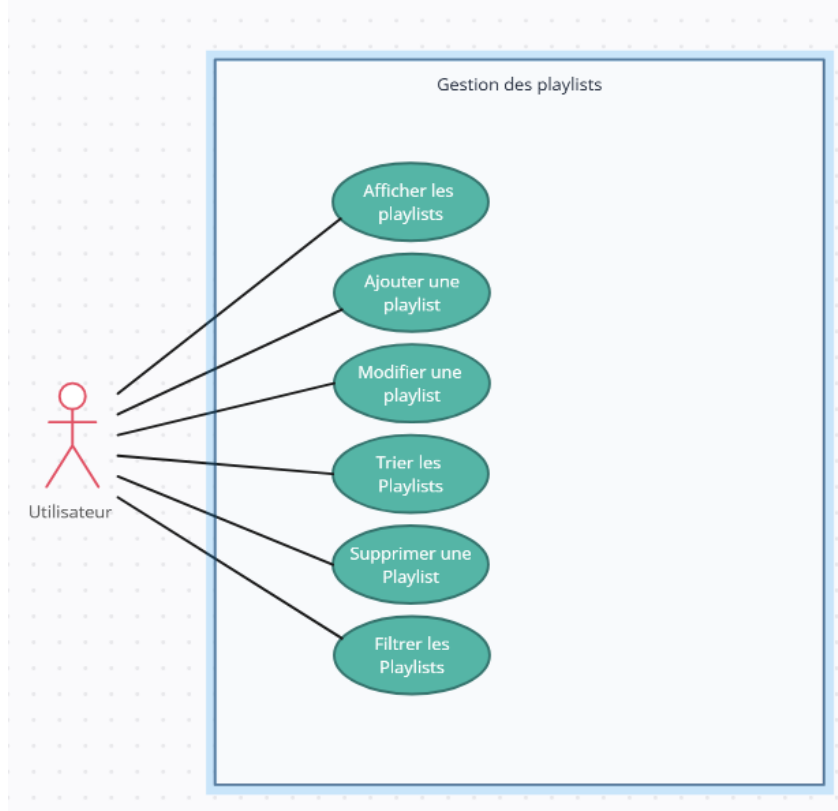
Projects

@Codeuraxe's untitled project

Status: In progress

+5 more

Milestone

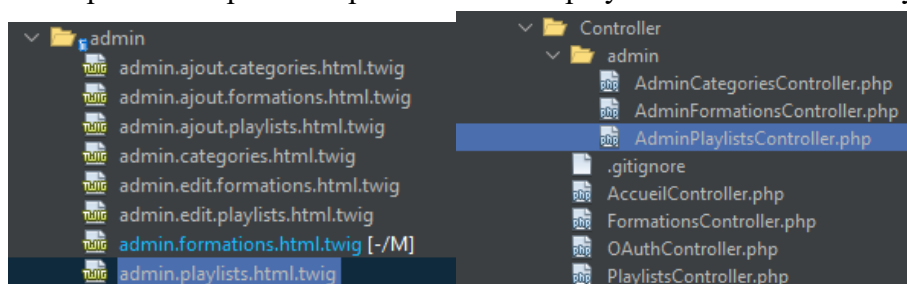


## 1. Création de la Page des Playlists

Pour débiter avec la gestion des playlists, j'ai d'abord créé une page `admin.playlists.html.twig` située dans le répertoire `templates/admin`. Cette page servira d'interface pour gérer (afficher, ajouter, modifier, supprimer) les playlists.

Dans le fichier de base `baseadmin.html.twig`, un lien dans la barre de navigation pointe vers la gestion des playlists (`admin.playlists`).

Je crée ensuite la page `AdminPlaylistsController`, la méthode `index()`, les variables et le constructeur ont été implémentée pour récupérer la liste des playlists en utilisant le `PlaylistRepository`.





```

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

```

```

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

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

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

```

## 2. Afficher la liste des playlists

Pour afficher les playlists, il est nécessaire d'effectuer une itération sur chaque élément de la liste playlists en utilisant `for k in 0..playlists|length-1`, où `k` sert d'indice pour accéder aux éléments.

admin.playlist.html.twig :

```

{% for k in 0..playlists|length-1 %}
    <tr class="align-middle">
        <td>
            <h5 class="text-info">
                {{ playlists[k].name }}
            </h5>
        </td>
        <td class="text-left">
            {% set categories = playlists[k].categoriesplaylist %}
            {% if categories|length > 0 %}
                {% for c in 0..categories|length-1 %}
                    &nbsp;{{ categories[c] }}
                {% endfor %}
            {% endif %}
        </td>
        <td class="text-center">
            {{ playlists[k].formations|length }}
        </td>
        <td class="text-center">
            <a href="{{ path('playlists.showone', {id:playlists[k].id}) }}" class="btn btn-secondary">Voir détail</a>
        </td>
        <td class="text-left">

```

## 3. Permettre la suppression d'une playlist

Pour la suppression d'une playlist, j'ai ajouté une méthode `delete` dans `AdminPlaylistsController`. Cette méthode vérifie si la playlist est vide avant de procéder à sa suppression. Si la condition est

remplie, la playlist est supprimée et l'utilisateur est redirigé vers la liste des playlists avec un message de confirmation.

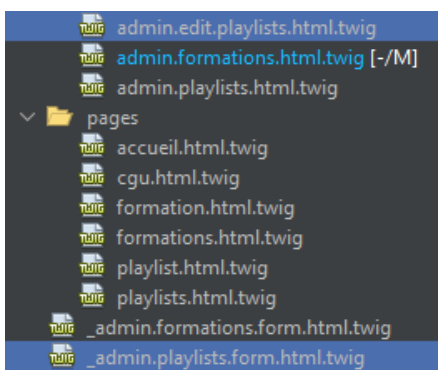
```
/**
 * @Route("/admin/playlist.delete/{id}", name="admin.playlist.delete")
 */
public function delete(Playlist $playlist, EntityManagerInterface $entityManager): Response
{
    $entityManager->remove($playlist);
    $entityManager->flush();
    return $this->redirectToRoute('admin.playlists');
}
```

Dans admin.playlist.html.twig j'ajoute un onclick permettant de demander la confirmation de suppression d'une playlist.

```
<div class="row">
    <div class="col">
        <a href="{{ path('admin.playlist.delete', {id:playlists[k].id}) }}" class="btn btn-danger btn-sm" onclick="return confirm('Confirmez la suppression');">Supprimer</a>
    </div>
</div>
```

#### 4. Permettre la modification d'une playlist

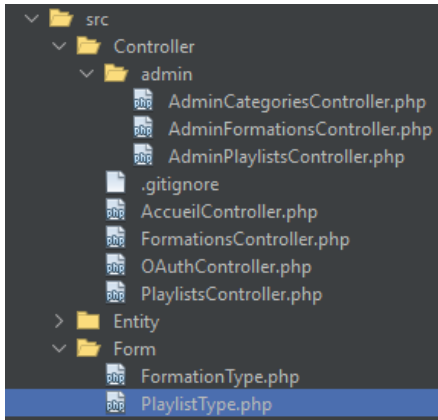
j'ai intégré un bouton "Éditer" qui redirige l'utilisateur vers un formulaire de modification spécifique. Ce formulaire est généré depuis une nouvelle page, nommée \_\_admin.playlists.form.html.twig, que j'ai créée spécifiquement pour héberger le formulaire d'édition d'une playlist. Pour l'intégrer et l'utiliser dans le contexte de l'édition, il est nécessaire de l'ajouter dans admin.edit.playlists.html.twig. Cette organisation me permet de réutiliser le formulaire pour l'édition d'autres playlists, optimisant ainsi la gestion du site.



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

{% block body %}
    {{ include ("__admin.playlists.form.html.twig") }}
{% endblock %}
```

Il faut maintenant créer PlaylistType.php qui va définir la structure du formulaire d'édition, incluant les champs à afficher (nom de la playlist, catégories, formations, etc.), avec un add de type text.



**Ajout du champ 'name' :** Pour permettre à l'utilisateur de saisir le nom de la playlist, j'intègre au constructeur un champ en utilisant la méthode `->add` avec comme premier argument 'name', de type `TextType`. Ensuite je spécifie l'option 'required'  $\Rightarrow$  `true`, pour rendre ce champ obligatoire. Pour guider l'utilisateur, je définis également un 'label' avec la valeur "Playlist", qui affichera le texte "Playlist" à côté du champ destiné à accueillir le nom de la playlist.

**Ajout du champ 'description' :** Pour recueillir des informations supplémentaires sur la playlist, j'ajoute un champ 'description' de type `TextareaType`, en utilisant la même méthode `->add`. Ce champ, contrairement au nom, n'est pas obligatoire, ce qui est indiqué par l'option 'required'  $\Rightarrow$  `false`. Afin d'informer l'utilisateur de l'objectif de ce champ, je lui attribue un 'label'  $\Rightarrow$  'Description'.

**Intégration du champ 'formations' :** Afin d'associer une ou plusieurs formations à la playlist, il est nécessaire d'inclure dans le formulaire un champ permettant de sélectionner ces formations. Ce champ, nommé 'formations', est ajouté via `->add`, en utilisant `EntityType::class` comme type. Cette spécification permet d'interagir avec la base de données pour lister les formations existantes. Je configure ce champ pour afficher les titres des formations ('choice\_label'  $\Rightarrow$  'title'), permettre des sélections multiples ('multiple'  $\Rightarrow$  `true`), et pour qu'il ne soit pas obligatoire ('required'  $\Rightarrow$  `false`). Ce dispositif facilite l'association des formations à la playlist lors de l'édition.

```
class PlaylistType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name', TextType::class, [
                'label' => 'Playlist',
                'required' => true,
            ])
            ->add('description', TextareaType::class, [
                'label' => 'Description',
                'required' => false,
            ])
            ->add('formations', EntityType::class, [
                'class' => Formation::class,
                'choice_label' => 'title',
                'multiple' => true,
                'required' => false,
            ])
            ->add('submit', SubmitType::class, [
                'label' => 'Enregistrer',
            ]);
    }
}
```

Playlist

Eclipse et Java - Test Ajout

Description

Utilisation de l'IDE Eclipse et développement en Java. - Test Ajout

Formations

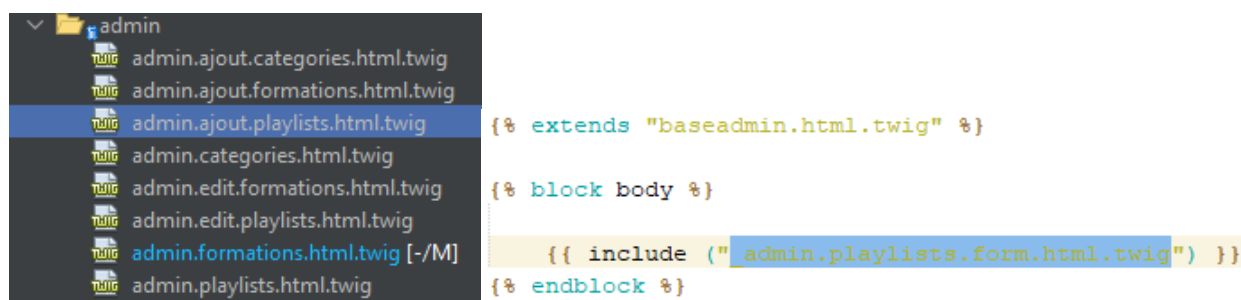
- Eclipse n°8 : Déploiement
- Eclipse n°7 : Tests unitaires
- Eclipse n°6 : Documentation technique
- Eclipse n°5 : Refactoring

Enregistrer

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

## 5. Permettre l'ajout d'une playlist

Grâce au formulaire d'édition déjà créé, la fonction d'ajout utilise le même formulaire mais sans préremplir les champs. Un bouton "Ajouter une playlist" redirige vers une page (admin.ajout.playlists.html.twig) où le formulaire vide est présenté à l'utilisateur. Je commence par créer (admin.ajout.html.twig) et j'inclus le chemin \_admin.playlists.form.html.twig contenant le formulaire.



Playlist

Description

Formations

- Eclipse n°8 : Déploiement
- Eclipse n°7 : Tests unitaires
- Eclipse n°6 : Documentation technique
- Eclipse n°5 : Refactoring

Enregistrer

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

Je mets en place une fonction d'ajout au sein du AdminPlaylistsController. Le processus est similaire à celui employé pour l'édition, puisque je réutilise le formulaire défini par PlaylistType.

La principale différence dans cette procédure d'ajout réside dans la nécessité d'initialiser une nouvelle instance de Playlist avant de procéder à la création du formulaire. Cela est accompli par l'ajout de la ligne `$playlist = new Playlist();` juste au début de la fonction. Pour capturer et traiter la demande de l'utilisateur, j'utilise le formulaire en appelant la méthode `handleRequest()` sur celui-ci, en lui passant l'objet Request en paramètre.

```

/**
 * @Route("/admin/playlist/ajout", name="admin.ajout.playlists")
 */
public function ajout(Request $request, EntityManagerInterface $entityManager): Response
{
    $playlist = new Playlist();
    $form = $this->createForm(PlaylistType::class, $playlist);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager->persist($playlist);
        $entityManager->flush();
        return $this->redirectToRoute('admin.playlists');
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render('admin/admin.ajout.playlists.html.twig', [
        'formplaylist' => $form->createView(),
        'categories' => $categories
    ]);
}

```

Enfin, j'intègre un bouton intitulé "Ajouter une playlist" dans ma page "admin.playlists.html.twig". Ce bouton est configuré pour diriger vers la fonction associée à la route "admin.ajout.playlists". Lorsqu'un utilisateur clique sur ce bouton, il est redirigé vers la page permettant d'ajouter une nouvelle playlist.

```

<th class="text-center align-top" scope="col">
    Détails
</th>
<th class="text-align-top" scope="col">
    <a href="{{ path('admin.ajout.playlists') }}" type="button" class="btn btn-outline-primary btn-sm">Ajouter une playlist</a>
</th>
</tr>

```

## 6. Permettre les fonctions de tris et filtres

### Partie Tri

Pour activer le tri sur la page des playlists, j'intègre des boutons dans le fichier (admin.playlists.html.twig). Les critères de tri souhaités sont le nom de la playlist et le nombre de formations. Lorsqu'un utilisateur clique sur un des boutons admin.playlists.sort va être invoquée. Dans admin.playlists.html.twig, pour chaque critère de tri, j'ai ajouté des boutons "<" et ">" qui permettent de trier en ordre ascendant ou descendant.

En fonction des valeurs reçues (champ et ordre), la méthode appropriée du PlaylistRepository va être appelée pour effectuer le tri. J'ai créé deux méthodes dans le repository : findAllOrderByNom pour le tri par nom, et findAllOrderByNbFormations pour le tri par nombre de formations.

```

/**
 * @Route("/admin/playlists/tri/{champ}/{ordre}/{table}", name="admin.playlists.sort")
 */
public function sort($champ, $ordre, $table = ""): Response
{
    if ($champ == 'name') {
        $playlists = $this->playlistRepository->findAllOrderByName($ordre);
    } elseif ($champ == 'nbformations') {
        $playlists = $this->playlistRepository->findAllOrderByNbFormations($ordre);
    } else {
        // Gestion si le champ ne correspond à aucun des cas précédents
        $playlists = $this->playlistRepository->findAll();
    }

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

```

Tris sur les playlists :

```

Playlist<br />
<a href="{{ path('admin.playlists.sort', {table: 'playlist', champ: 'name', ordre: 'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"></a>
<a href="{{ path('admin.playlists.sort', {table: 'playlist', champ: 'name', ordre: 'DESC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"></a>

```

Tris sur le nombre de formation :

```

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

```

## Filtrage

Dans mon contrôleur AdminPlaylistsController, j'ai créé la méthode findAllContain. Pour gérer les requêtes de filtrage que les utilisateurs peuvent spécifier via la zone de recherche.

Dans la méthode findAllContain, le processus commence par la récupération du terme de recherche que l'utilisateur a entré. Ce terme est obtenu à partir de l'objet Request grâce à \$request->get("recherche").

Je vérifie si des paramètres spécifiant une table et un champ sont inclus dans la requête. Si c'est le cas, et la table est spécifiée, j'appelle findByContainValueTable de PlaylistRepository pour filtrer les données en fonction de \$valeur dans le \$champ spécifié de la \$table donnée.

Si aucun nom de table n'est spécifié, j'utilise findByContainValue, pour rechercher \$valeur dans le \$champ indiqué sans se limiter à une table spécifique.

En plus du filtrage, la fonction récupère la liste de toutes les catégories pour les utiliser dans le formulaire de filtrage par catégorie dans la vue.

```

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

```

Ensuite il faut renvoyer les résultats à la vue admin/admin.playlists.html.twig, où les playlists filtrées sont présentées à l'utilisateur. J'inclus également la liste des catégories.

Pour permettre la recherche par nom, je crée un formulaire en utilisant la méthode POST. Ce formulaire pointe vers la fonction admin.playlists.findallcontain pour le traitement des données.

À l'intérieur du formulaire, j'insère un `<input>` pour la saisie du texte de recherche et un autre `<input>` pour générer un token CSRF, améliorant ainsi la sécurité du formulaire.

```

Playlist<br />
<a href="{{ path('admin.playlists.sort', {table: 'playlist', champ: 'name', ordre: 'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"><<
<a href="{{ path('admin.playlists.sort', {table: 'playlist', champ: 'name', ordre: 'DESC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true">>>
<form class="form-inline mt-1" method="POST" action="{{ path('admin.playlists.findallcontain', {champ: 'name', table: 'playlist'}) }}">
    <div class="form-group mr-1 mb-2">
        <input type="text" class="sm" name="recherche" value="{{ if valeur|default and table|default and table=='playlist' %}} {{ valeur }}{% endif %}">
        <input type="hidden" name="_token" value="{{ csrf_token('filtre_name') }}">
        <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
    </div>
</form>

```

Pour le filtrage par catégories, je crée un formulaire simple contenant une balise `<select>` remplie avec les catégories obtenues depuis le contrôleur. Ce formulaire utilise également admin.playlists.findallcontain, mais avec \$champ défini à "id" et \$table à "categories".

Lorsqu'une catégorie est sélectionnée, le formulaire est soumis automatiquement, permettant ainsi de filtrer les playlists par la catégorie choisie.

À l'intérieur de la balise `<select>`, j'utilise une boucle for pour générer des `<option>` pour chaque catégorie, permettant aux utilisateurs de choisir parmi toutes les catégories disponibles.

<b>Playlist</b> <div> <div>&lt;</div> <div>&gt;</div> </div> <div> <input type="text"/> <div>filtrer</div> </div>	<b>Catégorie</b> <div> <div></div> <div>▼</div> </div>	<b>Nombre de formations</b> <div> <div>&lt;</div> <div>&gt;</div> </div>	<b>Détails</b>	<div>Ajouter une playlist</div>
--	---	---	----------------	---------------------------------

Eclipse et Java - Test Ajout	Java UML	8	<div>Voir détail</div>	<div>Editer</div> <div>Supprimer</div>
------------------------------	----------	---	------------------------	--

```

<th class="text-left align-top" scope="col">
  Catégorie
  <form class="form-inline mt-1" method="POST" action="{{ path('admin.playlists.findallcontain', {champ: 'id', table: 'categories'}) }}">
    <select class="form-select form-select-sm" name="recherche" id="recherche" onchange="this.form.submit();">
      <option value=""></option>
      {% for categorie in categories %}
        <option
          {% if valeur|default and valeur==categorie.id %}
            selected
          {% endif %}
          value="{{ categorie.id }}">{{ categorie.name }}
        </option>
      {% endfor %}
    </select>
  </form>
</th>

```

### Tâche 3: Gérer les catégories

Temps estimé: 3h | Temps réalisé: 2h

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

Dans la même page, un mini formulaire doit permettre de saisir et d'ajouter directement une nouvelle catégorie, à condition que le nom de la catégorie n'existe pas déjà.

Mission 2 : Tâche 3 : gérer les catégories #5

Edit
Copy
Share
More
Close

Open
Codeuraxe/Kanban2
Private

Codeuraxe
opened this issue on Jan 27

edited by Codeuraxe
Edits
More

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

Dans la même page, un mini formulaire doit permettre de saisir et d'ajouter directement une nouvelle catégorie, à condition que le nom de la catégorie n'existe pas déjà

+
Codeuraxe
added this to @Codeuraxe's untitled project on Jan 27

Assignees
No one - [Assign yourself](#)

Labels
No labels

Projects
@Codeuraxe's untitled project
Status In progress

```

graph LR
    subgraph "Gestion des catégories"
        UC1([Consulter les catégories])
        UC2([Ajouter une catégorie])
        UC3([Supprimer une catégorie])
    end
    U((Utilisateur)) --- UC1
    U --- UC2
    U --- UC3

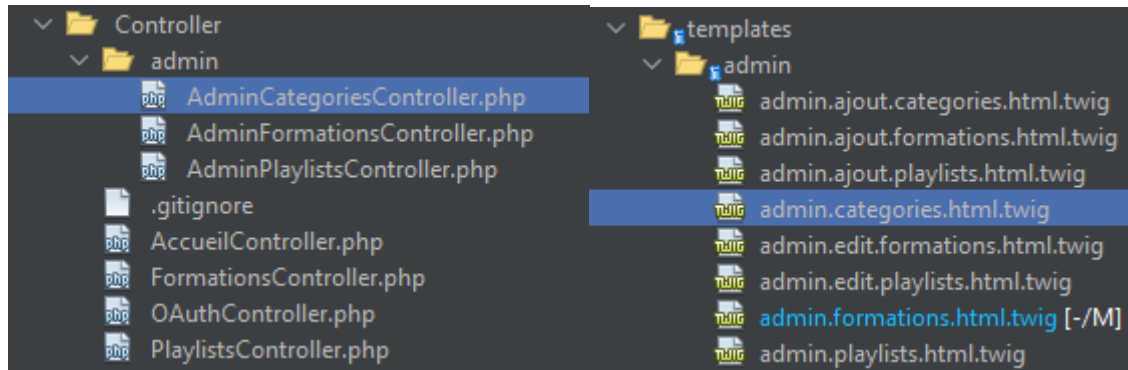
```

The diagram illustrates the requirements for the 'Gestion des catégories' (Category Management) system. It features a central box labeled 'Gestion des catégories' containing three use cases: 'Consulter les catégories' (View categories), 'Ajouter une catégorie' (Add a category), and 'Supprimer une catégorie' (Delete a category). A stick figure labeled 'Utilisateur' (User) is connected to each of these three use cases, indicating that the user is the actor for all these actions.



## 1. Création de la Page des Catégories

Je commencé par développer un fichier "admin.categories.html.twig" . Pour relier cette page à l'application, j'ai créé une classe AdminCategoriesController dans le dossier "controller". Dans "baseadmin.html.twig", j'ai ajouté un lien vers la page d'administration des catégories.



```
class AdminCategoriesController extends AbstractController
{
    private $categorieRepository;

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

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

## 2. Affichage de la Liste des Catégories

Pour présenter les catégories existantes, j'ai utilisé une boucle "for" dans "admin.categories.html.twig", permettant d'afficher le nom de chaque catégorie. J'ai également mis en place une colonne pour lister les formations liées à chaque catégorie, fournissant une vue d'ensemble utile.

```
<table class="table table-striped">
    {# Affichage des catégories #}
    <thead>
        <tr>
            <th class="text-left align-top" scope="col">Catégorie</th>
            <th class="text-left align-top" scope="col">Formations</th>
            <th class="text-align-top" scope="col"></th>
        </tr>
    </thead>
```

```

<tbody>
    {% for categorie in categories %}
        <tr class="align-middle">
            <td class="align-middle">{{ categorie.name }}</td>
            <td class="text-left">
                {% for formation in categorie.formations %}
                    {{ formation.title }}<br/>
                {% endfor %}
            </td>
            <td>

```

### 3. Permettre la suppression d'une catégorie

CategorieRepository exécute la suppression, puis redirige l'utilisateur vers la page des catégories. Pour chaque catégorie listée, j'ai ajouté un bouton "Supprimer" qui appelle cette méthode. Afin d'éviter la suppression de catégories associées à des formations, j'ai intégré une condition qui permet l'affichage du bouton "Supprimer" uniquement pour les catégories sans formations. Enfin, une confirmation de l'action est demandée via un dialogue de confirmation, activé par un "onclick".

AdminCategoriesController.php :

```

/**
 * @Route("/admin/categorie/delete/{id}", name="admin.categorie.delete")
 */
public function delete(Categorie $categorie, EntityManagerInterface $entityManager): Response
{
    $entityManager->remove($categorie);
    $entityManager->flush();
    return $this->redirectToRoute('admin.categories');
}

```

### 4. Permettre l'ajout d'une Catégorie

Pour introduire une nouvelle catégorie, j'ai élaboré un processus d'ajout qui commence par la saisie d'un nom de catégorie dans un formulaire sur "admin.categories.html.twig". Avant d'ajouter la catégorie, j'utilise la méthode findBy de CategorieRepository pour vérifier si le nom soumis est déjà pris. Si le nom est unique, la catégorie est créée et ajoutée à la base de données, et je redirige ensuite l'utilisateur vers la page d'administration des catégories, en confirmant l'action par un message flash de succès.

AdminCategoriesController.php :

```

/**
 * @Route("/admin/categorie/ajout", name="admin.ajout.categories")
 */
public function ajout(Request $request, EntityManagerInterface $entityManager): Response
{
    $nomCategorie = $request->request->get('nom_categorie');

    if ($nomCategorie) {
        $categorie = new Categorie();
        $categorie->setName($nomCategorie);

        // Vérifier si la catégorie existe déjà
        $existingCategories = $this->categorieRepository->findBy(['name' => $nomCategorie]);

        if (empty($existingCategories)) {
            // Ajouter une nouvelle catégorie
            $entityManager->persist($categorie);
            $entityManager->flush();

            $this->addFlash('success', 'La catégorie a été ajoutée avec succès.');
```

admin.categories.html.twig :

```

<form method="post" action="{{ path('admin.ajout.categories') }}" class="mb-3">
    {# Ajout explicite du token CSRF #}
    <input type="hidden" name="_csrf_token" value="{{ csrf_token('ajout_categorie') }}">

    <input type="text" id="nom_categorie" name="nom_categorie" required>

    <button type="submit" class="btn btn-primary">Ajouter la catégorie</button>
</form>

```

Accueil Formations Playlists Catégories Gestion

Ajouter la catégorie

Catégorie	Formations
Java	Eclipse n°8 : Déploiement
	Eclipse n°7 : Tests unitaires
	Eclipse n°6 : Documentation technique
	Eclipse n°5 : Refactoring
	Eclipse n°4 : WindowBuilder
	Eclipse n°3 : GitHub et Eclipse
	Eclipse n°2 : rétroconception avec ObjectAid
	Eclipse n°1 : installation de l'IDE
	TP Android n°5 : code du contrôleur et JavaDoc
	POO TP Java n°6 : polymorphisme
	POO TP Java n°5 : événements et contrôleur
	POO TP Java n°4 : démarrage sur le contrôleur, construction du modèle
	POO TP Java n°3 : interface graphique
	POO TP Java n°2 : MVC
	POO TP Java n°1 : configuration d'Eclipse

## Tâche 4: Ajouter l'accès avec authentification

Temps estimé: 4h | Temps réalisé : 4h

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 (avec un lien de déconnexion).

# Configuration et Intégration de Keycloak pour l'Authentification

## 1. Mise en place de Keycloak

Initialisation de Keycloak : J'installe keycloak-19, ensuite je démarre le serveur via la commande `kc.bat start-dev`, en me plaçant dans le dossier `C:/keycloak/bin`.

Configuration dans l'interface de Keycloak : Une fois le serveur démarré, je configure un royaume nommé "mediatek-formation". Ensuite je crée le client "admin-id".

Paramétrage du client : J'ai spécifié les options de flux d'authentification et désactivé les cookies pour l'authentification par navigateur.

The screenshot displays the Keycloak Admin Console configuration for a client named "adminID". The interface is divided into two main panels: "Capability config" on the left and "General Settings" on the right.

**Capability config (Left Panel):**

- Client authentication:** On (toggle switch).
- Authorization:** Off (toggle switch).
- Authentication flow:** Standard flow (checked), Direct access grants (checked), Implicit flow (checked), Service accounts roles (unchecked), OAuth 2.0 Device Authorization Grant (unchecked), and OIDC CIBA Grant (unchecked).
- Login settings:**
  - Login theme: Choose...
  - Consent required: On (toggle switch).
  - Display client on screen: On (toggle switch).
  - Client consent screen text: (empty text area).
- Logout settings:**
  - Front channel logout: On (toggle switch).
  - Front-channel logout URL: (empty text field).
  - Backchannel logout URL: (empty text field).
  - Backchannel logout session required: On (toggle switch).
  - Backchannel logout revoke offline sessions: Off (toggle switch).

**General Settings (Right Panel):**

- Client ID:** adminID
- Name:** adminID
- Description:** (empty text area).
- Always display in console:** Off (toggle switch).
- Access settings:**
  - Root URL: (empty text field).
  - Home URL: (empty text field).
  - Valid redirect URIs: \* (with a link to "Add valid redirect URIs").
  - Valid post logout redirect URIs: (empty text field, with a link to "Add valid post logout redirect URIs").
  - Web origins: (empty text field, with a link to "Add web origins").
  - Admin URL: (empty text field).
- Capability config (Bottom):** Client authentication (On), Authorization (Off), Authentication flow (Standard flow and Direct access grants checked).

Je crée un utilisateur avec les droits nécessaires et un mot de passe.

## 2. Configuration du Projet Symfony

### Gestion de l'Authentification

Préparation de l'environnement : Dans le fichier `.env` de NetBeans, j'ai ajouté les configurations nécessaires pour relier Symfony à Keycloak, incluant l'ID client et le secret client.

```
KEYCLOAK_SECRET=TKRvin37ETGytOU6MWnoZwW6g6LiQrYx
KEYCLOAK_CLIENTID=adminID
KEYCLOAK_APP_URL=http://localhost:8080
```

Création et migration de l'utilisateur : À l'aide des commandes `php bin/console make:user` et `php bin/console make:migration`, j'ai préparé la structure de la base de données pour gérer les utilisateurs.

Installation des bundles : Les commandes `composer require` m'ont permis d'intégrer les bundles nécessaires pour le lien entre Symfony et Keycloak, configurant ainsi le fichier `knpu_oauth2_client.yaml`.

```
knpu_oauth2_client:
  clients:
    keycloak:
      type: keycloak
      auth_server_url: '%env(KEYCLOAK_APP_URL)%'
      realm: 'mediatek-formation'
      client_id: '%env(KEYCLOAK_CLIENTID)%'
      client_secret: '%env(KEYCLOAK_SECRET)%'
      redirect_route: 'oauth_check'
```

J'ai mis à jour le fichier `security.yaml` pour définir les politiques d'accès et le processus d'authentification avec Keycloak.

Configuration du Firewall:

J'ouvre le fichier "`security.yaml`" et j'ajoute la route de redirection "`oauth_login`" ainsi que les chemins nécessitant une authentification.

```
entry_point: form_login
#provider: app_user_provider
form_login:
  login_path: oauth_login
custom_authenticators:
  - App\Security\KeycloakAuthenticator
logout:
  path: logout
```

Création du Contrôleur OAuthController:

J'utilise la commande `php bin/console make:controller OAuthController --no-template` pour créer le contrôleur. Ce contrôleur gère les routes liées à l'authentification.

```

class OAuthController extends AbstractController
{
    /**
     * Création de la route qui redirige vers l'authentification
     * @Route("/oauth/login", name="oauth_login")
     */
    public function index(ClientRegistry $clientRegistry): RedirectResponse
    {
        return $clientRegistry->getClient('keycloak')->redirect();
    }

    /**
     * Création de la route qui prend en charge la redirection du retour
     * @Route("/oauth/callback", name="oauth_check")
     */
    public function connectCheckAction(Request $request, ClientRegistry $clientRegistry)
    {
    }

    /**
     * Création de la route vers logout
     * @Route("/logout", name="logout")
     */
    public function logout()
    {
    }
}

```

Création de la Classe KeycloakAuthenticator:

Dans le dossier "src", je crée un dossier "Security" avec une nouvelle classe "KeycloakAuthenticator.php".

J'implémente les méthodes nécessaires à la gestion de l'authentification.

```

/**
 * Gère l'authentification
 */
class KeycloakAuthenticator extends OAuth2Authenticator implements AuthenticationEntryPointInterface
{
    private ClientRegistry $clientRegistry;
    private EntityManagerInterface $entityManager;
    private RouterInterface $router;

    public function __construct(ClientRegistry $clientRegistry, EntityManagerInterface $entityManager, RouterInterface $router)
    {
        $this->clientRegistry = $clientRegistry;
        $this->entityManager = $entityManager;
        $this->router = $router;
    }
}

```

Méthode start

La méthode spécifie comment démarrer une authentification. Elle redirige vers une route temporaire définie dans le contrôleur précédent (OAuthController), qui s'occupe de solliciter Keycloak.

```

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

```

Méthode supports

La méthode détermine si le système d'authentification doit se déclencher pour une URL donnée. Et Vérifie si l'URL correspond à 'oauth\_check'.

```

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

```

## Méthode authenticate

La méthode gère l'authentification en récupérant le client correspondant dans Keycloak et le token à partir des informations fournies.

```
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) {
            $keycloakUser = $client->fetchUserFromToken($accessToken);

            $existingUser = $this->entityManager
                ->getRepository(User::class)
                ->findOneBy(['keycloakId' => $keycloakUser->getId()]);

            if ($existingUser) {
                return $existingUser;
            }

            $email = $keycloakUser->getEmail();
            $userInDatabase = $this->entityManager
                ->getRepository(User::class)
                ->findOneBy(['email' => $email]);

            if ($userInDatabase) {
                $userInDatabase->setKeycloakId($keycloakUser->getId());
                $this->entityManager->persist($userInDatabase);
                $this->entityManager->flush();

                return $userInDatabase;
            }

            $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;
        })
    );
}
```

1. La méthode tente de récupérer un utilisateur existant dans la base de données.

Si un utilisateur est trouvé, il est retourné directement.

2. Si aucun utilisateur n'est trouvé pour le keycloakId, la méthode vérifie si l'email de l'utilisateur est déjà enregistré dans la base de données.

Si l'email est trouvé, l'algorithme met à jour le keycloakId de cet utilisateur avec l'ID récupéré depuis Keycloak, puis retourne l'utilisateur mis à jour.

3. Si ni le keycloakId ni l'email ne sont présents dans la base de données, l'algorithme crée un nouvel utilisateur.

Ce nouvel utilisateur est enregistré avec son keycloakId, son email et un rôle par défaut (dans cet exemple, le rôle est ROLE\_ADMIN). Enfin, l'utilisateur nouvellement créé est retourné.

Les Méthodes onAuthenticationFailure et onAuthenticationSuccess.

Elle Gèrent respectivement les cas d'échec et de succès de l'authentification.

onAuthenticationFailure: Retourne une réponse d'erreur.

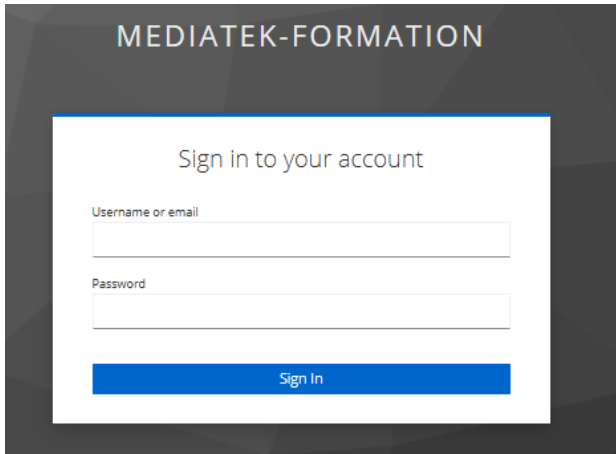
```
public function onAuthenticationFailure(Request $request, AuthenticationException $exception): ?Response
{
    $message = strtr($exception->getMessageKey(), $exception->getMessageData());

    return new Response($message, Response::HTTP_FORBIDDEN);
}
```

onAuthenticationSuccess: Redirige vers la route prévue à l'origine ou vers la partie 'admin' du site.

```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
{
    $targetUrl = $request->get('_target_path') ?? $this->router->generate('admin.formation');

    return new RedirectResponse($targetUrl);
}
```



## Gestion de la déconnexion

Je vais maintenant configurer la déconnexion de l'utilisateur. Mon objectif est de faciliter la déconnexion depuis la section "admin". Pour ce faire, je me dirige vers le dossier "templates" afin d'éditer le fichier "baseadmin.html.twig". Juste au-dessus de la section du titre, j'insère un lien intitulé "Se déconnecter". Ce lien pointe vers la route 'logout', qui n'existe pas encore.

```
{% block top %}
<div class="container">
  <div class="text-end">
    <a href="{{ path('logout') }}">Se déconnecter</a>
  </div>
  <div class="text-left">
    
  </div>
</div>
```

j'ajoute la méthode suivante qui réagira à la route 'logout'. Cette méthode est intentionnellement laissée vide puisque c'est le firewall de Symfony qui va gérer la logique de déconnexion.

```
/**
 * Création de la route vers logout
 * @Route("/logout", name="logout")
 */
public function logout()
{
}
```

Dans le fichier de configuration "security.yaml", situé dans "config > packages", je me focalise sur la section du firewall nommée "main". C'est ici que je peux ajouter la configuration nécessaire pour le logout. Ceci inclut la spécification du chemin vers lequel les utilisateurs seront redirigés lorsqu'ils se déconnectent.



```
entry_point: form_login
#provider: app_user_provider
form_login:
  login_path: oauth_login
custom_authenticators:
  - App\Security\KeycloakAuthenticator
logout:
  path: logout
```

## Mission 3: Effectuer les tests

Temps estimé: 7h | Temps réalisé: 9h

### Tâche 1: Gérer les tests

#### 1. Effectuer différents types de tests (unitaires, d'intégration, fonctionnels, de compatibilité) sur l'application.

Tests unitaires :

Contrôler le fonctionnement de la méthode qui retourne la date de parution au format string.

Tests d'intégration sur les règles de validation :

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

Tests d'intégration sur les Repository :

Contrôler toutes les méthodes ajoutées dans les classes Repository (pour cela, créer une BDD de test).

Tests fonctionnels :

Contrôler que la page d'accueil est accessible.

Dans chaque page contenant des listes :

contrôler que les tris fonctionnent (en testant juste le résultat de la première ligne) ;

contrôler que les filtres fonctionnent (en testant le nombre de lignes obtenu et le résultat de la première ligne) ;

contrôler que le clic sur un lien (ou bouton) dans une liste permet d'accéder à la bonne page (en contrôlant l'accès à la page mais aussi le contenu d'un des éléments de la page).

Tests de compatibilité :

Créer un scénario avec Selenium, sur la partie front office, et le jouer sur plusieurs navigateurs pour tester la compatibilité du site.

Codeuraxe opened this issue on Feb 3

...

Tests unitaires :

Contrôler le fonctionnement de la méthode qui retourne la date de parution au format string.

Tests d'intégration sur les règles de validation :

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

Tests d'intégration sur les Repository :

Contrôler toutes les méthodes ajoutées dans les classes Repository (pour cela, créer une BDD de test).

Tests fonctionnels :

Contrôler que la page d'accueil est accessible.

Dans chaque page contenant des listes :

contrôler que les tris fonctionnent (en testant juste le résultat de la première ligne) ;

contrôler que les filtres fonctionnent (en testant le nombre de lignes obtenu et le résultat de la première ligne) ;

contrôler que le clic sur un lien (ou bouton) dans une liste permet d'accéder à la bonne page (en contrôlant l'accès à la page mais aussi le contenu d'un des éléments de la page).

Tests de compatibilité :

Créer un scénario avec Selenium, sur la partie front office, et le jouer sur plusieurs navigateurs pour tester la compatibilité du site.

Assignees

No one - Assign yourself

Labels

No labels

Projects

@Codeuraxe's untitled project

Status

In progress

Priority

Filter options

Size

Filter options

Estimate

Enter number...

Start date

No date

End date

No date

Tests unitaires

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler la méthode <code>getPublishedAtString()</code> de la classe <code>Formation</code> pour voir si elle retourne la bonne date au bon format.	Test unitaire lancé avec la date : 2024-02-06 17:00:12	06/01/2024	OK

Tests d'intégration

But du test	Action de contrôle	Résultat attendu	Bilan
Lors de l'ajout ou de la modification d'une formation, contrôler que la date n'est pas postérieure à aujourd'hui.	Appeler la méthode <code>count()</code> : 2033/03/16	Erreur : 1	OK
Tester la méthode <code>add()</code> de la classe <code>FormationRepository</code> pour vérifier l'ajout d'une nouvelle formation.	Ajouter une nouvelle formation à la base de données et compter le nombre total de formations après l'ajout.	Le nombre total de formations doit augmenter de 1.	OK
Tester la méthode <code>remove()</code> de la classe <code>FormationRepository</code> pour vérifier la suppression d'une formation.	Supprimer une formation de la base de données et compter le nombre total de formations après la suppression.	Le nombre total de formations doit diminuer de 1.	OK
Tester la méthode <code>findAllOrderBy()</code> de la classe <code>FormationRepository</code> pour vérifier le tri des formations par titre.	Obtenir la liste de formations triées par titre et vérifier que la première formation est correcte.	La première formation de la liste doit être correcte et le nombre total de formations doit être correct.	OK
Tester la méthode <code>findAllOrderByTable()</code> de la classe <code>FormationRepository</code> pour vérifier le tri des formations par table.	Obtenir la liste de formations triées par table et vérifier que la première formation est correcte.	La première formation de la liste doit être correcte et le nombre total de formations doit être correct.	OK
Tester la méthode <code>findByContainValue()</code> de la classe <code>FormationRepository</code> pour vérifier la recherche de formations par une valeur partielle.	Ajouter une formation avec un titre contenant "C#" puis rechercher les formations contenant "C#" dans le titre.	Obtenir une liste de formations contenant "C#" dans le titre.	OK
Tester la méthode <code>findByContainValueTable()</code> de la classe <code>FormationRepository</code> pour vérifier la recherche de formations par une valeur partielle dans une table spécifique.	Ajouter une formation et rechercher les formations contenant une valeur spécifique dans une table donnée. <code>&gt;findByContainValueTable("name", "MCD exercices d'examen (sujets EDC BTS SIO)"</code>	Obtenir une liste de formations contenant la valeur spécifique dans la table spécifiée.	OK



# mediatekformation-master3

## Namespaces

- App
  - Controller
  - Entity
  - Form
  - Repository
  - Security

## Packages

- Application

## Reports

- Deprecated
- Errors
- Markers

## Indices

- Files

## Documentation

### Table of Contents







#### Packages

 [Application](#)

#### Namespaces

 [App](#)

#### Constants

 [CNAME](#) = "c.name"  
 [CNCATEGORIENAME](#) = "c.name categoriename"  
 [FCATEGORIES](#) = "f.categories"  
 [PFORMATIONS](#) = "p.formations"  
 [PIDID](#) = "p.id id"  
 [PNAMENAME](#) = "p.name name"

## C.Tâche 3: Créer la documentation utilisateur

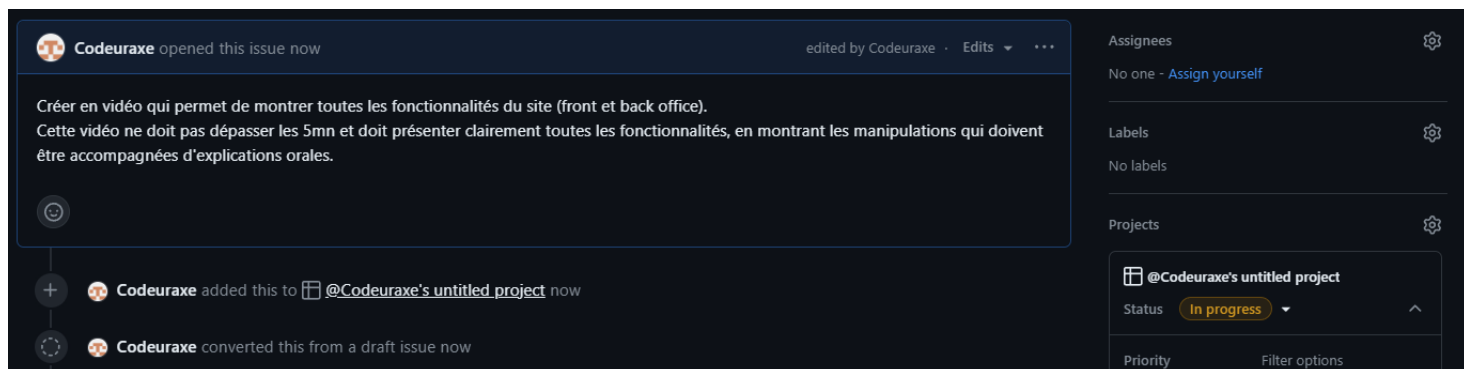
Créer une vidéo qui permet de montrer toutes les fonctionnalités du site (front et back office). Cette vidéo ne doit pas dépasser les 5mn et doit présenter clairement toutes les fonctionnalités, en montrant les manipulations qui doivent être accompagnées d'explications orales

Temps estimé: 2h | Temps réalisé: 2h

<https://www.youtube.com/watch?v=pv2BTnJchmM>

### 1.Créer une vidéo de démonstration des fonctionnalités du site

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



## Mission 4 : Déployer le site et gérer le déploiement continu

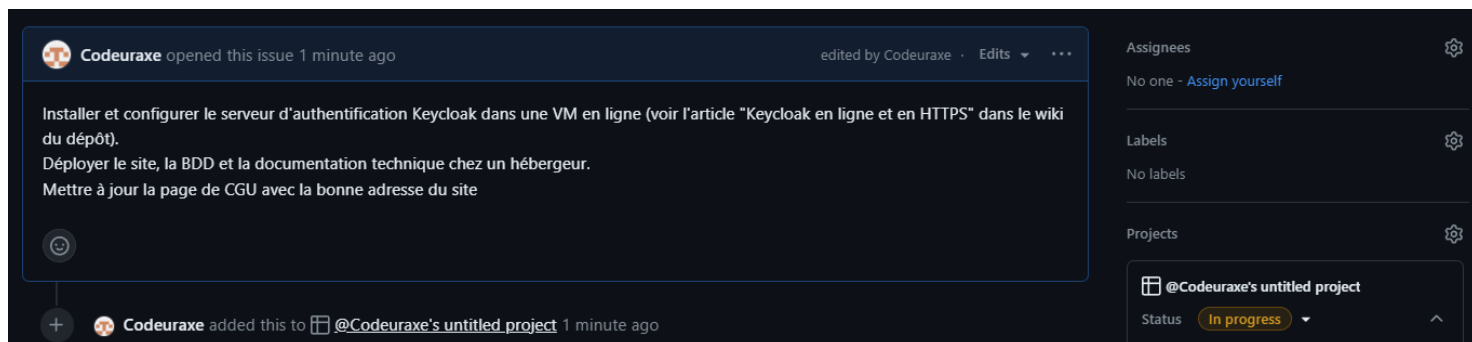
Temps estimé: 1h | Temps réalisé : 2h

### Tâche 1 - Déployer le site

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.



### 1. Configuration du serveur keycloak en HTTPS.

Pour déployer le serveur d'authentification je commence par mettre en place Keycloak sur une VM Linux dans Azure, j'ai suivi plusieurs étapes détaillées que je vais résumer.

Tout d'abord, je me suis connecté à Azure avec mon compte. J'ai navigué jusqu'à la section "Machines virtuelles" et j'ai cliqué sur "Créer > Machine virtuelle Azure". J'ai nommé la machine virtuelle "Media", sélectionné la région "France Central", et opté pour l'image "Ubuntu Server 20.04 LTS – x64 de la 2e génération" avec une architecture x64. J'ai configuré la taille de la VM pour avoir 1 processeur virtuel et 1 Go de mémoire, j'ai choisi l'authentification par mot de passe avec un nom d'utilisateur "sio" et un mot de passe sécurisé. J'ai sélectionné tous les ports d'entrée nécessaires (http, HTTPS, SSH, RDP) avant de valider et créer la VM.

Une fois la VM déployée, je suis passé à la configuration du DNS. Dans la vue d'ensemble de la VM, j'ai configuré le nom DNS "monkeycloakmedia" pour que mon domaine soit accessible via

"monkeycloak.francecentral.cloudapp.azure.com". Ensuite, j'ai paramétré les ports en ajoutant une règle pour le port 443 dans la section "Mise en réseau" de la configuration de la VM.

Pour utiliser la VM, j'ai accédé à celle-ci en SSH en utilisant Putty, où j'ai saisi l'adresse IP de la VM et sélectionné SSH. Après m'être connecté avec les identifiants fournis lors de la création de la VM, j'ai procédé à l'installation des différents logiciels nécessaires.

J'ai commencé par installer JDK 18.0.1 avec une série de commandes pour mettre à jour les paquets, télécharger et extraire le JDK, et configurer les variables d'environnement.

Ensuite, j'ai installé Keycloak 19.0.1 en téléchargeant et en extrayant l'archive depuis le site officiel de Keycloak.

J'ai aussi installé Apache pour servir les pages web et vérifié son fonctionnement en accédant à la VM depuis un navigateur web en utilisant HTTP.

Pour sécuriser la communication, j'ai installé Certbot pour obtenir un certificat SSL et configuré Apache pour utiliser HTTPS.

J'ai vérifié l'installation de Screen pour pouvoir lancer des processus qui restent actifs après la fermeture de la session SSH, et j'ai installé Screen car il n'était pas présent.

Pour accéder directement au serveur Keycloak et le configurer pour mon application, j'ai suivi un processus détaillé qui m'a permis de sécuriser l'accès à mon application et de gérer les utilisateurs et leurs droits.

Ensuite, je me suis rendu à l'adresse correspondant à mon nom de DNS configuré précédemment, c'est-à-dire <https://monkeycloak.francecentral.cloudapp.azure.com>. Là, j'ai pu accéder à la page de Keycloak. J'ai configuré Keycloak en créant d'abord le compte admin, en entrant un nom d'utilisateur et un mot de passe sécurisé, puis en cliquant sur "Create". Après cette étape, j'ai accédé à la console d'administration de Keycloak en saisissant mes identifiants.

Pour structurer mon espace de gestion dans Keycloak, j'ai commencé par créer un nouveau royaume, que j'ai nommé "myapplis", pour isoler la gestion de mon application. Ce nom doit correspondre à celui que j'utilise dans la configuration de mon application pour qu'ils puissent communiquer correctement.

J'ai ensuite procédé à la création d'un client dans Keycloak, qui représente mon application "mediatekformation". J'ai veillé à ce que l'ID du client et les différents paramètres soient configurés correctement pour permettre l'authentification et l'autorisation via OpenID Connect.

Après avoir configuré le client, j'ai créé un utilisateur spécifique qui aura accès à certaines parties de mon application. J'ai renseigné les informations de base de l'utilisateur, défini un mot de passe sécurisé et ajusté les paramètres selon les besoins.

J'ai également pris soin de désactiver les cookies dans les paramètres d'authentification de Keycloak pour permettre une déconnexion correcte.

Enfin, j'ai mis à jour les variables d'environnement de mon application pour y intégrer l'URL de Keycloak, le "client secret" et l'ID du client. Ces informations permettent à mon application de communiquer avec Keycloak pour gérer l'authentification et l'autorisation des utilisateurs.

```
KEYCLOAK_SECRET=3rraBJ1ET3dvuBuqAjsx51I0iIbQr8vh  
KEYCLOAK_CLIENTID=adminID  
KEYCLOAK_APP_URL=https://monkeycloakmedia.francecentral.cloudapp.azure.com
```

## 2. Déployer la base de données

Pour déployer la base de données, j'ai choisi l'hébergeur Hostinger. Pour commencer, j'exporte la base de données de l'application et copie le contenu dans un fichier. Ensuite, je me rends dans l'onglet "Bases de données > phpMyAdmin" sur Hostinger, où je crée une nouvelle base de données et un utilisateur associé. Les paramètres que j'utilise pour ma base de données sont : nom de la base de données : u695746505\_mediatekbase, utilisateur : u695746505\_mediatekform, et mot de passe : Senseoaxe95\*\*. Une fois la base de données ouverte, je colle le contenu de mon fichier dans l'onglet "SQL" de phpMyAdmin et clique sur "Exécuter". Ma base de données apparaît ensuite avec toutes les tables correctement configurées.

J'attribue également un site web à ma base de données, "http://mediatekformation.online/public", qui est le site où l'application sera déployée. Puis, dans Netbeans, je modifie les informations d'accès à la base de données dans le fichier ".env".

```
# DATABASE_URL="sqlite:///kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://u695746505_mediatekform:Senseoaxe95**@127.0.0.1:3306/u695746505_mediatekbase"
# DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=14&charset=utf8"
###< doctrine/doctrine-bundle ###
```

### 3. Déployer le site

Pour ce qui est du déploiement du site, je procède à quelques configurations dans Netbeans. Je vais dans la fenêtre de commandes et tape : composer require symfony/apache-pack, ce qui est nécessaire pour créer le fichier ".htaccess" dans le dossier "public". Je modifie également la variable "APP\_ENV" de "dev" à "prod" dans mon fichier ".env". Pour déployer le site sur Hostinger, je navigue jusqu'à l'onglet "Hébergement" et clique sur "Gérer" à côté de mon nom de domaine. Je me rends ensuite dans le "Gestionnaire de fichiers" et y dépose le dossier du projet, préalablement compressé en .zip pour gagner du temps. Après avoir décompressé les fichiers dans "public\_html", je lance le site. Constatant qu'il ne redirige pas directement vers ma page d'accueil, je retourne dans le gestionnaire Hostinger et procède à une redirection du nom de domaine de "http://mediatekformation.online" vers "

http://mediatekformation.online/public".

Après avoir déployé mon site chez un hébergeur et effectué les configurations nécessaires, j'ai testé l'accès aux parties de mon application nécessitant une authentification pour m'assurer que tout fonctionnait comme prévu.

### Tâche 2 - Gérer la sauvegarde et la restauration de la base de données

Temps estimé: 1h | Temps réalisé: 1h

Une sauvegarde journalière automatisée doit être programmée pour la BDD (voir l'article "Automatiser la sauvegarde d'une BDD" dans le wiki du dépôt).

La restauration pourra se faire manuellement, en exécutant le script de sauvegarde.





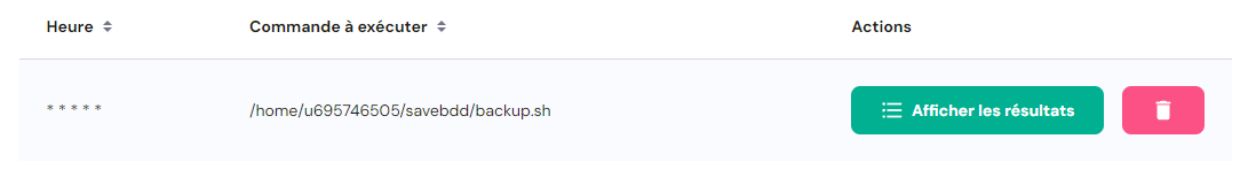
## 1. Automatisation de la sauvegarde

Pour créer une sauvegarde journalière automatisée, je crée un fichier de script pour sauvegarder ma base de données sur mon site d'hébergement. Je nomme ce fichier `backup.sh` et le configure pour enregistrer le script de la base de données dans un fichier intitulé `bddbbackup_`, suivi de la date et de l'extension `.sql.gz`, dans le dossier `savebdd` situé à la racine de mon compte.

Après avoir créé le script en local, je le convertis au format Unix en utilisant `dos2unix` pour m'assurer qu'il s'exécutera correctement sous Linux.

Pour transférer le fichier `backup.sh` chez l'hébergeur, j'utilise FileZilla. Je me connecte à mon compte, crée le dossier `savebdd` à la racine, puis transfère le fichier `backup.sh` dans ce dossier. Je m'assure ensuite de définir les permissions nécessaires pour permettre l'exécution du script.

Pour automatiser le processus de sauvegarde, je crée une tâche Cron depuis le panneau de gestion de Hostinger. Je configure la tâche pour exécuter le script `backup.sh` une fois par jour, en spécifiant le chemin complet du script dans la commande de la tâche Cron.



Je reçois une sauvegarde de ma base de données dans le dossier `savebdd` tous les jours, portant le nom `bddbbackup_` suivi de la date du jour et de l'extension `.sql.gz`.

## Tâche 3 - Mettre en place le déploiement continu

Temps estimé: 1h | Temps réalisé: 1h

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

Le but est de mettre en place un déploiement continu, avec l'objectif qu'à chaque fois que je pousse des modifications vers GitHub, mon site en ligne se mette également à jour. Ce processus nécessite que je configure GitHub pour qu'il sache vers quel serveur FTP envoyer les fichiers.

Je me dirige vers le dépôt Github qui contient mon projet et sélectionne "Actions". Sous le titre, je clique sur "set up a workflow yourself". Je met le script suivant :

```
on: push
name: Deploy website on push
jobs:
  web-deploy:
```



```
name: Deploy
runs-on: ubuntu-latest
steps:
  - name: Get latest code
    uses: actions/checkout@v2
  - name: Sync files
    uses: SamKirkland/FTP-Deploy-Action@4.3.0
    with:
      server: 'mediatekformation.online'
      server-dir: /public_html/
      username: 'u695746505.mediatekformation'
      password: '${ secrets.ftp_PASSWORD }
```

Ce script définit les actions à exécuter lors d'un push. Je remplace les informations sensibles par les données réelles de mon serveur FTP. Après avoir ajusté le script, je commets le nouveau fichier. Je constate que le dossier ".github/workflows" contenant mon fichier YML de déploiement a été ajouté à la racine de mon dépôt. Ensuite, dans Netbeans, je fais un pull du dépôt pour récupérer le dossier en local.

Dans mon dépôt GitHub, je navigue vers Settings > Secrets > Actions et crée un nouveau secret de dépôt. Ce secret, nommé "ftp\_password", contient le mot de passe d'accès au FTP de mon site, conformément à la référence dans le fichier YML.

Après avoir modifié le code de mon site en local, chaque commit et push vers GitHub déclenche automatiquement la mise à jour des fichiers sur le serveur FTP, mettant ainsi à jour le site en ligne.

## Bilan

J'ai rencontré des difficultés avec des erreurs techniques fréquentes tout au long du projet, par exemple lors du déploiement sur une VM Windows, qui ne fonctionnait pas comme attendu ou encore une erreur lors du déploiement du site. Cette situation m'a contraint à passer d'une VM Windows à Linux. Ces défis ont mis à l'épreuve ma capacité à résoudre des problèmes et à m'adapter à de nouveaux environnements techniques.

Globalement, cet atelier m'a permis de consolider et d'apprendre de nouvelles compétences en développement web. Les obstacles rencontrés ont été des occasions d'apprentissage. Je suis satisfait des résultats obtenus et confiant dans l'application de ces connaissances à des projets futurs. J'ai pu réaliser toutes les tâches demandées.

## Portfolio :

**Dépôt distant :** <https://github.com/Codeuraxe/Kanban2.git>

**Lien du site en ligne :** <https://mediatekformation.online/>