



ATELIER PROFESSIONNEL MEDIATEK FORMATION

COMPTE-RENDU RÉALISÉ PAR:
DROAI LOUIZA
BTS SIO-SLAM 2ÈME ANNÉE



CONTEXTE

MediaTek86 est un réseau qui administre les médiathèques de la Vienne. Son rôle principal est d'unifier les prêts de livres, DVD et CD tout en œuvrant au développement numérique des médiathèques dans tout le département. Afin d'accroître l'attrait des médiathèques, MediaTek86 souhaite élargir ses services en proposant des formations aux outils numériques pour les adhérents, ainsi que des ressources d'autoformation en ligne.

MISSION

Le chef de projet a contrôlé le travail du premier développeur et a constaté quelques manquements aux bonnes pratiques de codage et l'oubli d'une fonctionnalité attendue dans le cahier des charges. Dans un premier temps, il vous charge de corriger ces problèmes, puis plusieurs autres missions vous sont confiées(en particulier le back office) pour finaliser et déployer le site.

SOMMAIRE

MISSION 1: NETTOYER LE CODE ET AJOUTER UNE FONCTIONNALITE

TÂCHE 1 : GÉRER LES TESTS:

TACHE 1: NETTOYER LE CODE

TACHE 2: AJOUTER UNE FONCTIONNALITÉ

MISSION 2 : CODER LA PARTIE BACK-OFFICE

TACHE 1 : GÉRER LES FORMATIONS

TACHE 2: GÉRER LES PLAYLISTS

TACHE 3: GÉRER LES CATÉGORIES

TÂCHE 4 : AJOUTER L'ACCÈS AVEC AUTHENTIFICATION

MISSION 3 : TESTER ET DOCUMENTER

TÂCHE 1 : GÉRER LES TESTS:

TÂCHE 2: DOCUMENTATION TECHNIQUE

TÂCHE 3: DOCUMENTATION UTILISATEUR :

MISSION 4 : DÉPLOYER LE SITE ET GÉRER LE DEPLOIEMENT CONTINU

TÂCHE 1: DÉPLOYER LE SITE:

TÂCHE 2: DÉPLOYER LA BASE DE DONNÉES:

TÂCHE 3: DÉPLOYER LE SITE:

TÂCHE 4: GÉRER LA SAUVEGARDE ET LA RESTAURATION DE LA BDD:

TÂCHE 4: METTRE EN PLACE LE DÉPLOIEMENT CONTINU

BILAN

LANGAGES ET OUTILS

LANGAGES DE PROGRAMMATION

PHP
TWIG

SERVEUR

WAMPSEVER:
APACHE
MYSQL
PHP

AUTHENTIFICATION

KEYCLOCK

FRAMEWORD

SYMFONY

VERSIONNING

GITHUB

IDE

NETBEANS

MISSION 1: NETTOYER LE CODE ET AJOUTER UNE FOCTIONNALITE

TACHE 1: NETTOYER LE CODE

Tâche 1 : nettoyer le code (2h)

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.

Normalement, seul l'item demandant d'ajouter un header à une table, doit rester.

Ma première responsabilité était de nettoyer le code en me basant sur les recommandations de Sonarlint. Les instructions précisait aussi de ne procéder au nettoyage que des fichiers générés par le développeur, en triant les "Action items" de Sonarlint par "Location" et de stopper dès le premier fichier dans "vendor".

Pour identifier les erreurs à corriger, j'ai lancé une analyse avec Sonarlint dans chaque dossier du projet renfermant les fichiers initialement créés par le développeur. Une fois les erreurs relevées, j'ai procédé aux corrections nécessaires.

A. PREMIÈRE ERREUR "STRING LITERALS SHOULD NOT BE DUPLICATED" - FICHER "FORMATIONSCONTROLLER" (DOSSIER "CONTROLLER")

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

```

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

```

L'erreur "String literals should not be duplicated" signale la nécessité d'éviter les redondances de chaînes de caractères dans le code en utilisant des constantes.

Pour résoudre ce problème, il faut définir une constante, par exemple

"**FORMATIONPAGE**", et y affecter la chaîne de caractères

"pages/formations.html.twig".

```

define("FORMATIONPAGE", "pages/formations.html.twig" );
/**

```

-->Je modifie alors les occurrences de "pages/formations.html.twig" dans le code en utilisant le nom de ma constante "**FORMATIONSPATH**".

```

        $formations = $this->formationRepository->findAllOrderBy($champ, $ordre, $table);
        $categories = $this->categorieRepository->findAll();
        return $this->render(FORMATIONPAGE, [
            'formations' => $formations,

```

```

        $formations = $this->formationRepository->findAllOrderBy($champ, $ordre, $table);
        $categories = $this->categorieRepository->findAll();
        return $this->render(FORMATIONPAGE, [
            'formations' => $formations,
            'categories' => $categories,
            'valeur' => $valeur

```

"PLAYLISTSCONTROLLER"

La deuxième instance de l'erreur "String literals should not be duplicated", repérée dans le fichier "PlaylistsController" situé dans le répertoire "Controller", signale le même problème que celui rencontré précédemment,

```
        $playlists = $this->playlistRepository->findAllOrderByName (
            break;
        }
        $categories = $this->categorieRepository->findAll();
        return $this->render("pages/playlists.html.twig", [
            'playlists' => $playlists,
            'categories' => $categories
        ]);
    }
}
```

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

il faut définir une constante, par exemple "PLAYLISTSPAGE", et y affecter la chaîne de caractères "pages/playlists.html.twig".

```
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

define("PLAYLISTSPAGE", "pages/playlists.html.twig");
```

```
/**
 * Description of PlaylistsController
 */
```

-->Je modifie alors les occurrences de "pages/playlists.html.twig" dans le code en utilisant le nom de ma constante "PLAYLISTSPAGE".

```
}
$categories = $this->categorieRepository->findAll();
return $this->render(PLAYLISTSPAGE, [
    'playlists' => $playlists,
    'categories' => $categories
]);
```

```
public function findAllContain($champ, Request $request, $table=""): Response{
    $valeur = $request->get("recherche");
    $playlists = $this->playlistRepository->findByContainValue($champ, $valeur, $table);
    $categories = $this->categorieRepository->findAll();
    return $this->render(PLAYLISTSPAGE, [
        'playlists' => $playlists,
```

B-LES NOMS DE CONSTANTES DOIVENT RESPECTER UNE CONVENTION DE NOMMAGE.

En PHP, la convention de nommage habituelle pour les constantes est d'utiliser des lettres majuscules et des underscores pour séparer les mots.

Erreur : Fichier "Formation" (dans le répertoire "Entity").

Les noms de constantes doivent respecter une convention de nommage

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

Pour remédier à cette erreur, je dois convertir la constante en majuscules et remplacer toutes les occurrences de la constante "cheminImage" dans le code par "CHEMINIMAGE".

```
    /**
     * 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";
    }
```

C-LES INSTRUCTIONS CONDITIONNELLES IMBRIQUÉES DEVRAIENT ÊTRE FUSIONNÉES.

Cette erreur indique qu'il y a plusieurs instructions conditionnelles "if" dans une fonction qui pourraient être combinées en une seule instruction "if" pour améliorer la lisibilité du code.

Cette erreur a été repérée dans le fichier "Playlist" situé dans le répertoire "Entity".

```
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);
        }
    }

    return $this;
}
```

J'ai simplifié le code en fusionnant les deux conditions en une seule instruction "if" à l'aide de l'opérateur logique "&&". Cela rend le code plus lisible et réduit la complexité.

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

    return $this;
}
```

- **LES BALISES "" ET "" DEVRAIENT ÊTRE UTILISÉES.**

La première erreur indique que dans le fichier "cgu" du répertoire "templates > pages", les balises <i> ont été utilisées au lieu des balises , comme recommandé.

```
L.C.E.N., il est porté à la connaissance des utilisateurs et visiteurs du s  
></u> (ci-après "<i>le site</i>"). L'accès et l'utilisation du Site sont so
```

pour corriger cette erreur, je remplace <i> par

```
5  
6 L.C.E.N., il est porté à la connaissance de  
7 ></u> (ci-après "<em>le site</em>"). L'accès  
8
```

La seconde erreur indique que dans le fichier "cgu" situé dans le répertoire "templates > pages", les balises <i> ont été utilisées à la place des balises , comme recommandé.

```
(ci-après "<i>l'Éditeur</i>")</summary>
```

```
(ci-après "<i>l'Hébergeur</i>")</summary>
```

```
(ci-après "<i>les Utilisateurs</i>")</summary>  
utilisateurs tous les internautes qui naviguent, i
```

pour corriger cette erreur, je remplace <i> par

```
"<strong>l'Éditeur</strong>")</summary>
```

```
(ci-après "<strong>l'Hébergeur</strong>")</summary>
```

```
(ci-après "<strong>les Utilisateurs</strong>")</summary>  
utilisateurs tous les internautes qui naviguent, licent, visi
```


La même erreur indique que dans le fichier "basefront.html.twig" situé dans le répertoire "templates > pages", les balises `<i>` ont été utilisées à la place des balises ``, comme recommandé.

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

pour corriger cette erreur, je remplace `<i>` par ``

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

résultat:

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

D-AJOUTER L'ATTRIBUT "ALT" À TOUTES LES IMAGES.

L'erreur signalée est que les balises `` doivent inclure un attribut "alt" pour décrire l'image.

- le manque d'attribut "alt" dans les balises `` du fichier "accueil" situé dans le dossier "templates > pages".

```
<div class="col">
  <!-- emplacement photo -->
  {% if formation.picture %}
    <a href="{{ path('formations.showone', {id:formation.id}) }}">
      
    </a>
  {% endif %}
</div>
```

Pour corriger l'erreur "image tags should have an 'alt' attribute", on ajoute un attribut "alt" avec une description appropriée dans la balise ``.

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

- le manque d'attribut "alt" dans les balises du fichier "basefront.html.twig" situé dans le dossier "templates"

```

<!-- titre -->
<div class="text-left">
  
</div>

```

Pour corriger l'erreur "image tags should have an 'alt' attribute", on ajoute un attribut "alt" avec une description appropriée dans la balise .

```

<!-- titre -->
<div class="text-left">
  
</div>

```

- le manque d'attribut "alt" dans les balises du fichier "PLAYLIST" situé dans le dossier "templates > pages".

```

<div class="col-md-auto">
  {% if formation.miniature %}
    <a href="{{ path('formations.showone', {id:formation.id}) }}">
      
    </a>
  </div>

```

Pour corriger l'erreur "image tags should have an 'alt' attribute", on ajoute un attribut "alt" avec une description appropriée dans la balise .

```

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

```

- le manque d'attribut "alt" dans les balises du fichier "Formation" situé dans le dossier "templates > pages".

```

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

```

Pour corriger l'erreur "image tags should have an 'alt' attribute", on ajoute un attribut "alt" avec une description appropriée dans la balise .

```

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

```

E-AJOUTER L'ATTRIBUT "DESCRIPTION" À TOUTES LES TABLES.

L'erreur indique que les balises `<table>` ne sont pas suivies par une balise `<caption>` contenant une description de la table,

- l'absence de balise `<caption>` contenant une description après la balise `<table>`. Dans le fichier "accueil" du dossier "templates > pages"

```
Voici les <strong>deux dernières formations<
<table class="table">
  <tr>
```

Pour corriger cette erreur, on ajoute une balise `<caption>` avec une description juste après la balise `<table>`

```
<table class="table">
  <caption> table pour les 2 dernières formations ajoutées au catalogue</caption>
  <tr>
    {% for formation in formations %}
```

- l'absence de balise `<caption>` contenant une description après la balise `<table>`. Dans le fichier "playlist" du dossier "templates > pages"

```
{% extends "basefront.html.twig" %}
{% block body %}
  <table class="table table-striped">
    <thead>
```

Pour corriger cette erreur, on ajoute une balise `<caption>` avec une description juste après la balise `<table>`

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

- l'absence de balise `<caption>` contenant une description après la balise `<table>`. Dans le fichier "formation" du dossier "templates > pages"

```
% block body %}  
<table class="table table-striped">  
  <thead>
```

Pour corriger cette erreur, on ajoute une balise `<caption>` avec une description juste après la balise `<table>`

```
{% extends "basefront.html.twig" %}  
{% block body %}  
  <table class="table table-striped">  
    <caption>tableau de formation</caption>  
    <thead>
```

Mentions légales du site

Conformément aux dispositions des Articles 6-III et 19 de la Loi n°2004-575 du 21 juin 2004 pour la Confiance dans l'économie numérique, dite L.C.E.N., il est porté à la connaissance des utilisateurs et visiteurs du site mediatekformation.fr les présentes mentions légales.

Le site mediatekformation est accessible à l'adresse suivante : mediatekformation.fr (ci-après "le site"). L'accès et l'utilisation du Site sont soumis aux présentes "Mentions légales" détaillées ci-après ainsi qu'aux lois et/ou règlements applicables. La connexion, l'utilisation et l'accès à ce site impliquent l'acceptation intégrale et sans réserve de l'internaute de toutes les dispositions des présentes Mentions Légales.

Article 1 - Informations légales

En vertu de l'Article 6 de la Loi n° 2004-575 du 21 juin 2004 pour la confiance dans l'économie numérique, il est précisé dans cet article l'identité des différents intervenants dans le cadre de sa réalisation et de son suivi.

- ▶ A. Éditeur du site (ci-après "**l'Éditeur**")
- ▶ B. Hébergeur du site (ci-après "**l'Hébergeur**")
- ▶ C. Utilisateurs (ci-après "**les Utilisateurs**")

TACHE 2: AJOUTER UNE FONCTIONNALITÉ

Drlouiza commented 3 hours ago

Owner ...

Tâche 2 : ajouter une fonctionnalité (2h)

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.



Dans la page des playlists, intégrer une colonne permettant d'afficher le nombre de formations par playlist et d'activer le tri croissant et décroissant basé sur cette colonne. En rassurant que cette information soit visible dans la page détaillée de chaque playlist.

- **les playlists triées en fonction du nombre de formations**

pour récupérer les playlists triées en fonction du nombre de formations associées à chaque playlist. J'ai décidé de créer une nouvelle méthode dans la classe `PlaylistRepository` pour accomplir cette tâche. J'ai nommé cette méthode `findAllOrderByAmount` pour refléter son objectif. J'ai envisagé les paramètres nécessaires à la méthode, dans ce cas, le paramètre `$ordre` pour spécifier l'ordre de tri.

1. Construction de la requête Doctrine : J'ai utilisé le `QueryBuilder` de Doctrine pour construire la requête SQL nécessaire. J'ai ajouté les jointures et les conditions nécessaires pour récupérer les données souhaitées. Dans ce cas, j'ai ajouté une jointure avec la table des formations et j'ai groupé les résultats par l'ID de la playlist.
2. Définition de l'ordre de tri : J'ai utilisé la méthode `orderBy` pour spécifier l'ordre de tri des résultats. Dans ce cas, j'ai trié les playlists en fonction du nombre de formations associées à chaque playlist. J'ai utilisé la fonction de comptage (`COUNT`) pour compter le nombre de formations.
3. Exécution de la requête et récupération des résultats : J'ai finalisé la requête en appelant `getQuery()` pour obtenir l'objet `Query` correspondant, puis `getResult()` pour exécuter la requête et récupérer les résultats.

```

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

```

Pour optimiser le code:

```

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

```

Dans la méthode `getCategoriesPlaylist()` ajoutée à l'entité `Playlist`, j'ai créé une collection vide pour stocker les catégories. En parcourant les formations associées à la playlist, j'ai récupéré les catégories de chaque formation et les ai ajoutées à cette collection tout en évitant les doublons. Finalement, la méthode retourne cette collection contenant toutes les catégories associées à la playlist.

Je dois ajuster la méthode "sort" dans mon fichier PlaylistsController pour qu'elle puisse appeler à la fois les méthodes "findAllOrderByName" et "findAllOrderByAmount" du PlaylistRepository en fonction du champ de tri choisi, soit le nom ou le nombre de formations. Cela est réalisé en utilisant une structure de contrôle "switch case" pour sélectionner la méthode appropriée en fonction du critère de tri.

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


Dans la vue "playlists.html.twig", j'implémente la gestion du tri ascendant et descendant de la colonne du nombre de formations par playlist en appelant la méthode "playlists.sort". J'ajoute deux boutons, un pour le tri ascendant et l'autre pour le tri descendant, chacun avec un texte indiquant la direction du tri. Ces boutons sont liés au chemin vers la méthode "playlists.sort" pour gérer le tri.

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

<th class="text-center align-top" scope="col">
```

Dans la page "playlist.html.twig", j'ai ajouté le nombre de formations pour chaque playlist individuelle en utilisant la commande "playlistformations|length". Cela me permet d'afficher le nombre total de formations associées à chaque playlist. En plus, j'ai ajouté un titre "Nombre de formations" pour rendre cette information plus claire et facile à comprendre pour les utilisateurs.

Pour une vue claire et précise du nombre de formations disponibles pour chaque playlist directement dans le tableau, on doit ajouté une nouvelle colonne affichant le nombre de formations associées à chaque playlist

```
11 v class="col" /
  <h4 class="text-info mt-5">{{ playlist.name }}</h4>
  <strong>Nombre de formations : </strong>{{
    playlistformations|length}} <br>
  <strong>Catégories : </strong>
```

```

</thead>
<tbody>
  <!-- boucle sur les playlists -->
  {% if playlists|length > 0 %}
    {% for k in 0..playlists|length-1 %}
      <tr class="align-middle">
        <td>
          <h5 class="text-info">
            {{ playlists[k].name }}
          </h5>
        </td>
        <td 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>
          <h5 class="text-center">
            {{ playlists[k].formations|length }}
          </h5>
        </td>
        <td class="text-center">
          <a href="{{ path('playlists.showone', {id:playlists[k].id}) }}" class="btn btn-secondary">Voir détail</a>
        </td>
      </tr>
    {% endfor %}
  {% endif %}
</tbody>

```

Et pour finir, je teste le bon fonctionnement de ces fonctionnalités en exécutant l'application. Je vérifie d'abord que le nombre de formations s'affiche correctement sur la page "playlists" et que les boutons "croissant" et "décroissant" permettent de trier les playlists par le bon ordre. Ensuite, je m'assure que le nombre de formations d'une playlist est affiché lorsque je clique sur le bouton "Voir détail" pour accéder à la page de la playlist correspondante.

- L'afficher de nombre de formations par playlist et permettre le tri croissant et décroissant sur cette colonne.

playlist Croissant Décroissant filtrer

catégories

Nombre de formations Croissant Décroissant

Détail

Bases de la programmation (C#) C# POO 74 [Voir détail](#)

- L'afficher de nombre de formations par playlist dans la page d'une playlist



MediaTek86

Des formations pour tous sur des outils numériques

[Accueil](#) [Formations](#) [Playlists](#)

Bases de la programmation (C#)

Nombre de formations : 74

Catégories : C# POO

Description :

Exemples progressifs de programmes en procédural, événementiel et objet sous Visual Studio (version Entreprise 2017).

Prérequis : aucun

1ère partie : programmation procédurale en mode console (non graphique)

n°1 à 30 : procédural, notions élémentaires (variables, saisie/affichage, affectations/calculs, alternatives (if/switch), itérations (while/do-while/for))

n°31 à 42 : procédural, tableaux (1 et 2 dimensions, manipulations, tris, recherches)

n°43 à 59 : procédural, modules et paramètres (procédures et fonctions)



Bases de la programmation n°1 - procédural : premier exemple

Bases de la programmation n°2 - procédural : exercice1 (affichage)

Bases de la programmation n°3 - procédural : exercice2 (saisie)

Bases de la programmation n°4 - procédural : exercice3 (calculs)

Bases de la programmation n°5 - procédural : exercice4 (calcul dans

MISSION 2 : CODER LA PARTIE BACK-OFFICE

TACHE 1 : GÉRER LES FORMATIONS

La tâche consiste à mettre en place la gestion des formations, avec les points suivants :

- Créer une page pour lister les formations.
- Chaque formation doit être accompagnée de boutons pour la suppression (avec confirmation) et la modification.
- Supprimer une formation doit également la retirer de la playlist associée.
- Assurer que les tris et filtres du front office sont également disponibles dans le back office.
- Ajouter un bouton pour accéder au formulaire d'ajout de formation.
- Contrôler les saisies dans le formulaire, en s'assurant que seuls les champs obligatoires sont renseignés.
- Permettre la sélection d'une playlist et de catégories depuis une liste déroulante.
- La date doit être sélectionnée plutôt que saisie et ne peut pas être postérieure à la date du jour.
- Lorsque l'utilisateur souhaite modifier une formation, rediriger vers le même formulaire prérempli.

A-CREATION DE PAGE DE GESTION DE FORMATION

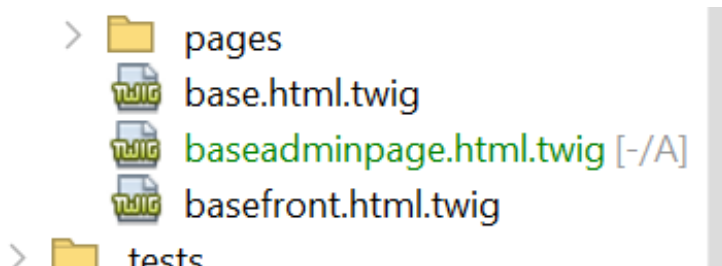
Je commence par créer un dossier "admin" dans le dossier "templates", pour pouvoir ensuite créer une page "formations.html.twig"



formations.html.twig

je crée un dossier “admin” dans le dossier “controller”, pour pouvoir ensuite créer un fichier “AdminFormationsController”

je crée un fichier “baseadminpage.html.twig” dans le dossier “templates> pages”



- faire le lien entre les pages

on ajoute un lien dans l'interface utilisateur principale (frontend) qui redirigera vers la page de gestion des formations. on place ce lien dans la barre de navigation de fichier “basefront.html.twig”

```
<div class="nav-item justify-content-md-end navbar-nav">
|   <a class="btn btn-outline-success" href="{ path('admin.formations') }" >##128274; Espace administrateur</a>
</div>
```

on ajoute ensuite un chemin vers la page d'administration dans le fichier baseadminpage.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('admin.formations') }}">Formations</a>
]                     </li>
]                 </ul>
]             </div>
]         </nav>
]     </div>
] {% endblock %}
```

dans ma page admininformationsController.php, je crée les variables, constructeur, l'index:

```
1  private $formationRepository;
2
3  /**
4   *
5   * @var categorieRepository
6   */
7  private $categorieRepository;
8
9  public function __construct(formationRepository $formationRepository, categorieRepository $categorieRepository) {
10     $this->formationRepository = $formationRepository;
11     $this->categorieRepository = $categorieRepository;
12 }
13
14 /**
15  * @Route("/admin", name="admin.formations")
16  * @return Response
17  */
18 public function index(): Response{
19     $formations = $this->formationRepository->findAll();
20     $categories = $this->categorieRepository->findAll();
21     return $this->render(self::PAGE_FORMATIONS, [
22         'formations' => $formations,
23         'categories' => $categories
24     ]);
25 }
26
27 /**
28  * @Route("/admin/formations/tri/{champ}/{ordre}/{table}", name="admin.formations.sort")
29  * @param type $champ
30  * @param type $ordre
31  * @param type $table
32  * @return Response
33  */
34 public function sort($champ, $ordre, $table=""): Response{
35     $formations = $this->formationRepository->findAllOrderBy($champ, $ordre, $table);
36     $categories = $this->categorieRepository->findAll();
37     return $this->render(self::PAGE_FORMATIONS, [
38         'formations' => $formations,
39         'categories' => $categories
40     ]);
41 }
```

affichage des listes des formations

```
<tbody>
  {% for formation in formations %}
  <tr class="align-middle">
    <td><h5 class="text-info">{{ formation.title }}</h5</td>
    <td class="align-middle">{{ formation.playlist }}</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('admin.showone', {id:formation.id}) }}">
    <td><a href="{{ path('admin.formation.edit', {id:formation.id}) }}" class="btn btn-warning">Editer</a></td>
    <td><a href="{{ path('admin.formation.suppr', {id:formation.id}) }}" class="btn btn-outline-danger"
      onclick="return confirm('Confirmer la suppression de {{ formation.title }} ?')">Supprimer </a></td>
  </tr>
  {% endfor %}
</tbody>
</table>
```

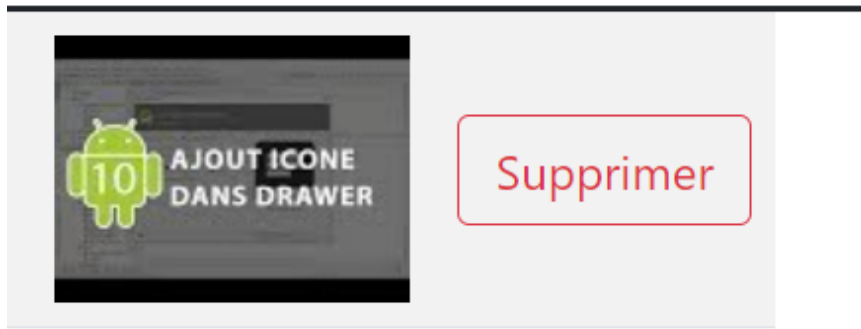
AFFICHER UN BOUTON PERMETTANT DE LA SUPPRIMER (APRÈS CONFIRMATION)

Pour autoriser la suppression d'une formation, je dois définir une fonction dans le contrôleur "AdminFormationsController". Cette fonction va récupérer la méthode "remove" du "FormationsRepository". Ensuite, je vais configurer une redirection vers la route correspondant à ma page "formations.html.twig" après la suppression.

```
/**
 * Suppression d'une formation
 * @Route("/admin/formations/suppr/{id}", name="admin.formation.suppr")
 * @param Formation $formation
 * @return Response
 */
public function suppr(Formation $formation):
    Response{
        $this->formationRepository->remove($formation, true);
        $this->addFlash(
            'alert',
            'Suppression de la formation ' . $formation->getTitle() . ' prise en compte');
        return $this->redirectToRoute('admin.formations');
    }
```

J'ajoute les boutons demandés dans le fichier twig.

je teste mon bouton "supprimer" :



localhost indique

Confirmer la suppression de Eclipse n°8 : Déploiement ?

OK

Annuler

UN BOUTON PERMETTANT DE LA MODIFIER.

Pour pouvoir modifier une formation, je dois ajouter un nouveau bouton "Éditer". Lorsque ce bouton est cliqué, il ouvre un formulaire permettant de modifier les détails de la formation. Pour créer ce formulaire, je dois créer une nouvelle page nommée "formations.html.twig" dans le répertoire racine des modèles. Cette page contiendra le formulaire d'édition de la formation.

En plaçant le formulaire dans cette nouvelle page, je pourrai le réutiliser dans d'autres pages.

Pour créer le formulaire dans ma page "formations.form.html.twig", je dois suivre ces étapes :

1. Créer un nouveau dossier nommé "Form" dans le répertoire "src".
2. Concevoir une nouvelle classe PHP appelée "FormationType" à l'intérieur de ce dossier.
3. Cette classe servira à définir les champs du formulaire d'ajout ou d'édition d'une formation.

Le formulaire doit récupérer le champ "formation", avec le titre et la description de la formation. Ensuite, il doit inclure le champ "catégories" avec le nom de la catégorie associée et la possibilité de sélectionner une ou plusieurs catégories lors de l'ajout ou de l'édition. De plus, il doit contenir le champ avec la date de création de la formation, le champ de la playlist qui concerne la formation et la possibilité d'en sélectionner une parmi celles qui existent. Enfin, le formulaire doit contenir un bouton "submit", qui permettra de soumettre le formulaire et d'enregistrer les informations dans la base de données.

Pour que le builder puisse récupérer le champ avec le titre de la formation sélectionnée et sa description, nous ajoutons au builder un `"->add('title')"` de type `"Text"`. On lui rajoute un `"label => Formation"`, qui affichera le texte "Formation" à côté du champ qui contiendra le titre de la formation. Nous précisons également `"required' => true"` car il est obligatoire pour une formation d'avoir un titre. Ensuite, nous ajoutons un `"->add('description')"` de type `"Textarea"`, avec `"required' => false"` car la description n'est pas obligatoire. Nous lui donnons également un `"label => Description"`.

Le formulaire doit également inclure le champ "playlist", qui affiche une liste déroulante des playlists et sélectionne la playlist de la formation lorsque le bouton d'édition est cliqué. Il doit également permettre de sélectionner une nouvelle playlist dans cette liste si l'on souhaite éditer une formation. Pour que le formulaire d'édition d'une formation prenne en compte les informations de la classe "Playlist" afin de les enregistrer dans la liste déroulante, nous devons ajouter la classe "Playlist" au builder. La commande permettant de réaliser cela est la suivante : "->add('playlist', EntityType::class)". Ensuite, nous ajoutons la classe "Playlist" et lui précisons un label "playlist". Pour créer la liste des choix de playlist possibles, nous ajoutons "choice_label => name" afin de récupérer les noms des playlists. Comme nous ne devons pouvoir sélectionner qu'une seule playlist par formation, nous définissons "'multiple' => false", et comme il est obligatoire d'avoir une playlist, nous définissons "required" comme "true".

```
class FormationType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $opt:
    {
        $builder
            ->add('publishedAt', DateType::class, [
                'widget' => 'single_text',
                'data' => isset($options['data']) &&
                    $options['data']->getPublishedAt() != null ? $option
                'label' => 'Date'
            ])
            ->add('title', TextType::class, [
                'required' => true
            ])
            ->add('description')
            ->add('videoId')
            ->add('playlist', EntityType::class, [
                'class' => Playlist::class,
                'choice_label' => 'name',
                'required' => false
            ])
            ->add('categories', EntityType::class, [
                'class' => Categorie::class,
                'choice_label' => 'name',
                'multiple' => true,
                'required' => false
            ])
            ->add('submit', SubmitType::class, [
                'label' => 'Valider'
```

Le formulaire doit également permettre l'ajout ou l'édition d'une catégorie. Pour cela, j'ajoute un "->add('categories)" de type Entity, qui fait référence à la classe "Categorie". Je lui attribue le label "Catégorie", et en choice_label, je précise "name" pour afficher le nom de chaque catégorie dans la liste. Puisque plusieurs catégories peuvent être affectées à une formation, je définis "multiple" comme "true", et comme il est possible de ne pas affecter de catégorie du tout, je définis 'required' comme 'false'.

```
->add('categories', EntityType::class, [  
    'class' => Categorie::class,  
    'choice_label' => 'name',  
    'multiple' => true,  
    'required' => false  
]);
```

j'ajoute un bouton "enregistrer" qui va permettre d'enregistrer les données

```
        ->add('submit', SubmitType::class, [  
            'label' => 'Enregistrer'  
        ]);  
    }  
}
```

je peux passer à la création de la fonction pour le bouton "éditer" dans mon fichier "AdminFormationsController". Cette fonction crée le formulaire en utilisant la classe "FormationType" qui contient la définition des champs du formulaire, et récupère les formations. La méthode "\$formFormation->handleRequest(\$request)" gère la soumission du formulaire.

J'ajoute une condition "if" pour vérifier si le formulaire a été soumis et s'il est valide. Si c'est le cas, il enregistre les informations du formulaire dans la base de données en appelant la méthode "add" du FormationRepository. Ensuite, l'utilisateur est redirigé vers la page de gestion des formations.

Si le formulaire n'est pas soumis ou s'il n'est pas valide, la fonction maintient l'utilisateur sur la page d'édition du formulaire.

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

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

Dans la page "_admin.formations.form.html.twig", je dois afficher le formulaire de modification des formations. Je commence par déclarer le début du formulaire avec "{{ form_start(formformation) }}" et je le termine avec "{{ form_end(formformation) }}". Ensuite, je construis une colonne pour chaque champ du formulaire "FormationType". Les données des champs sont affichées à l'aide de la syntaxe "{{ form_row(formformation.title) }}", où "formformation" est le formulaire créé dans le AdminFormationsController.

```

{{ form_start(formFormation, {'attr': {'class': 'form'}}) }}
<div class="row">
  <div class="col">
    <div class="row">
      <div class="col-md-4">
        <div class="form-group">
          {{ form_row(formFormation.publishedAT, {'attr': {'class': 'form-control', 'placeholder': 'Date de publication'}}) }}
        </div>
      </div>
      <div class="col-md-4">
        <div class="form-group">
          {{ form_row(formFormation.title, {'attr': {'class': 'form-control', 'placeholder': 'Titre'}}) }}
        </div>
      </div>
      <div class="col-md-4">
        <div class="form-group">
          {{ form_row(formFormation.videoId, {'attr': {'class': 'form-control', 'placeholder': 'ID de la vidéo'}}) }}
        </div>
      </div>
    </div>
    <div class="form-group">
      {{ form_row(formFormation.description, {'attr': {'class': 'form-control', 'placeholder': 'Description'}}) }}
    </div>
    <div class="form-group">
      {{ form_row(formFormation.playlist, {'attr': {'class': 'form-control'}}) }}
    </div>
    <div class="form-group">
      {{ form_row(formFormation.categories, {'attr': {'class': 'form-control'}}) }}
    </div>
  </div>
</div>

```


j'ajoute le bouton "éditer" avec le lien vers ma fonction "admin.edit.formations"

```

</td>
<td><a href="{{ path('admin.edit.formations', {id:formation.id}) }}" class="btn btn-outline-info">Editer</a></td>
<td><a href="{{ path('admin.suppr.formation', {id:formation.id}) }}" class="btn btn-outline-danger"
  onclick="return confirm('Confirmer la suppression de {{ formation.title }} ?')">Supprimer </a></td>

```

je teste maintenant mon application

18/10/2018  [Editer](#) [Supprimer](#)

18/12/2018  [Editer](#) [Supprimer](#)

Date	Title	Video id
<input type="text" value="04/01/2021"/>	<input type="text" value="Eclipse n°8 : Déploiement"/>	<input type="text" value="Z4yTTXka958"/>
Description		
<input type="text" value="Exécution de l'application en dehors de l'IDE, en invite de commande. Création d'un fichier jar pour le déploiement de l'application."/>		
Playlist		
<input type="text" value="Eclipse et Java"/>		
Categories		
<input type="text" value="Android"/> <input type="text" value="C#"/> <input type="text" value="Cours"/> <input type="text" value="Java"/>		
<input type="button" value="Valider"/>		

AJOUTER UNE NOUVELLE FORMATION

La fonctionnalité d'édition de formation évite la recréation de formulaires existants. Ainsi, les modifications dans le builder ne sont pas nécessaires. Pour ajouter une nouvelle formation, il suffit de créer la fonction correspondante dans AdminFormationsController. Ensuite, il faut ajouter le bouton et le chemin d'ajout dans la page formations.html.twig.

Dans le "AdminFormationsController", je crée la fonction d'ajout, qui est presque identique à celle d'édition puisque j'utilise le même formulaire. La seule différence est l'attribution d'une nouvelle formation à la table des formations avec "\$formation = new Formation();"

```
/**
 * Ajouter une formation
 * @Route("/admin/formation/ajout", name="admin.formation.ajout")
 * @param Request $request
 * @return Response
 */
public function ajout(Request $request):
    Response{
        $formation = new Formation();
        $formFormation = $this->createForm(FormationType::class, $formation);

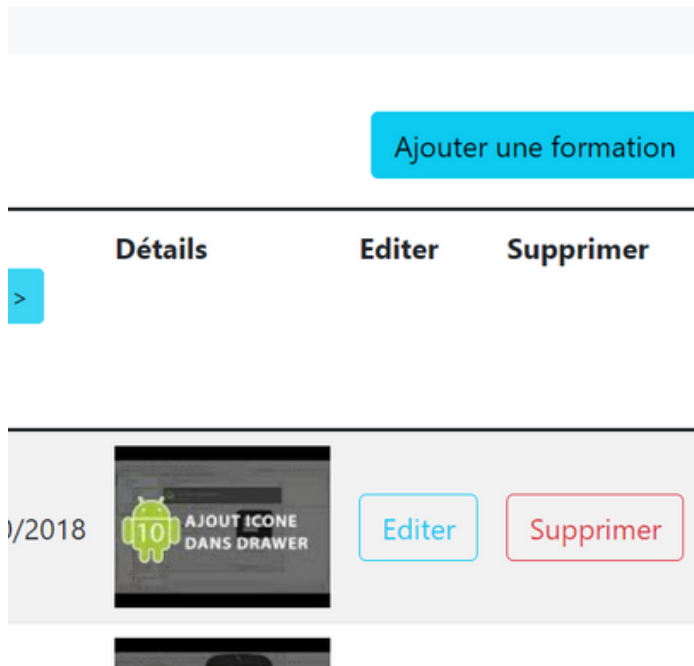
        $formFormation->handleRequest($request);
        if($formFormation->isSubmitted() && $formFormation->isValid()){
            $this->formationRepository->add($formation, true);
            $this->addFlash(
                'success',
                'Ajout de la formation ' . $formation->getTitle() . '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

        return $this->render(self::PAGE_FORMATION, [
            'formation' => $formation,
            'formFormation' => $formFormation->createView()
        ]);
    }
}
```

j'ajoute le bouton "Ajouter une formation" à la page "admin/formations.html.twig". Ce bouton est lié à la route de la fonction ajout formations, de sorte que cliquer dessus exécute la méthode correspondante.

```
<h3>GESTION DES FORMATIONS</h3>
<p class="text-end">
    <a href="{{ path('admin.formation.ajout') }}" class="btn btn-info">
        Ajouter une formation
    </a>
</p>
```

Après avoir exécuté mon application, je me rends sur la page des formations pour vérifier que le bouton "Ajouter une formation" s'affiche correctement.



Date: 01/04/2024

Title: Titre

Video id: ID de la vidéo

Description: Description

Playlist: [Dropdown menu]

Categories: Android, C#, Cours, Java

Valider

TRIS ET FILTRES

Tris ascendante et descendante

Pour commencer, je vais créer une fonction "admin.playlists.sort" qui sera invoquée lorsque les boutons "<" et ">" seront cliqués dans le fichier "playlists.html.twig". Cette fonction effectuera le tri ascendant et descendant des playlists en fonction de leur nom.

Dans ma fonction, je vais inclure deux paramètres (\$champ et \$ordre).

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

Pour appeler la méthode dans la page "admin/playlists.html.twig" lors de la création des boutons Croissant et Décroissant, je vais ajouter le chemin "admin.playlists.sort" à chacun des deux boutons. Ensuite, je vais définir le tri en insérant "ASC" comme valeur pour le paramètre \$ordre du bouton croissant, et "DESC" pour le bouton décroissant. Le champ recherché sera "name", car c'est le champ contenant le nom des playlists et je veux trier les playlists par leur nom.

```
<th class="text-left align-top" scope="col">
  Formation<br />
  <a href="{{ path('admin.formationen.sort', {champ:'title', ordre:'ASC'}) }}" class="btn btn-info" role="button" aria-pressed="true">
  <a href="{{ path('admin.formationen.sort', {champ:'title', ordre:'DESC'}) }}" class="btn btn-info" role="button" aria-pressed="true">
  <form class="form-inline mt-1" method="POST" action="{{ path('admin.formationen.findallcontain', {champ:'title'}) }}">
    <div class="form-group mr-1 mb-2">
      <input type="text" class="sm" name="recherche"
        value="{% if valeur|default and not table|default %}{{ valeur }}{% endif %}">
      <input type="hidden" name="_token" value="{{ csrf_token('filtre_title') }}">
      <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
    </div>
  </form>
</th>
```

Je dois aussi implémenter un tri ascendant et descendant basé sur le nombre de formations. Je procède de la même manière que pour le tri des playlists, puis je modifie la valeur de \$champ pour être "nbformations", afin que le tri se fasse correctement selon ce critère.

```
<th class="text-left align-top" scope="col">
  Playlist<br />
  <a href="{{ path('admin.formationen.sort', {table:'playlist', champ:'name', ordre:'ASC'}) }}" class="btn btn-info" role="button" aria-pressed="true">
  <a href="{{ path('admin.formationen.sort', {table:'playlist', champ:'name', ordre:'DESC'}) }}" class="btn btn-info" role="button" aria-pressed="true">
  <form class="form-inline mt-1" method="POST" action="{{ path('admin.formationen.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>
</th>
```

je fait la même chose pour les dates

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

Filtres

Pour filtrer les playlists et les catégories selon une valeur saisie dans un formulaire de recherche, je dois créer une fonction "findAllContain" dans le "AdminPlaylistsController". Cette fonction récupère la valeur saisie dans la zone de recherche, puis la compare avec les champs des différentes tables.

En commençant par extraire la valeur de la zone de recherche avec une requête "get("recherche");", stockée dans la variable \$valeur, la fonction détermine si une table est spécifiée en paramètre. Si c'est le cas, elle appelle la méthode "findByContainValue" du "PlaylistRepository" pour retourner les enregistrements de la table et du champ correspondant à \$valeur. En l'absence de désignation de table, la fonction utilise la méthode "findByContainValue" du "PlaylistRepository" pour retourner les enregistrements du champ correspondant à \$valeur.

La fonction récupère également la liste de toutes les catégories.

```
public function findAllContain($champ, Request $request, $table=""):
    Response{
        $valeur = $request->get("recherche");
        $playlists = $this->playlistRepository->findByContainValue($champ, $valeur, $table);
        $categories = $this->categorieRepository->findAll();
        return $this->render(self::PAGE_PLAYLISTS, [
            'playlists' => $playlists,
            'categories' => $categories,
            'valeur' => $valeur,
            'table' => $table
        ]);
    }
```

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

Pour filtrer par catégories, je vais créer un formulaire avec une liste déroulante contenant les catégories. Je vais utiliser la méthode "admin.playlists.findallcontain" en remplaçant le champ par "id" et en ajoutant le paramètre \$table "categories". La balise <select> dans le formulaire permettra de choisir une catégorie dans la liste déroulante de recherche. Une fois qu'une catégorie est sélectionnée et le formulaire soumis, les enregistrements correspondants à cette catégorie seront renvoyés. Pour que la recherche par catégorie fonctionne, je vais ajouter la boucle "for" nécessaire pour traiter les résultats du formulaire.

```
<th class="text-left align-top" scope="col" style="padding-bottom: 10px;">
  <span style="font-weight: bold;">Catégorie</span><br />
  <form class="form-inline mt-1" method="POST" action="{{ path('admin.formations.findallcontain', {champ:'id', table:'categories'}) }}">
    <select class="form-select form-select-sm" name="recherche" id="recherche" onchange="this.form.submit();">
      <option value=""></option>
      {% for categorie in categories %}
        <option
          {% if valeur|default and valeur==categorie.id %}
            selected
          {% endif %}
          value="{{ categorie.id }}">{{ categorie.name }}
        </option>
      {% endfor %}
    </select>
  </form>
```

Je vais exécuter l'application pour confirmer que les filtres de recherche sont visibles et opérationnels sur la page de gestion des playlists.

Formation	Playlist	Catégorie	Date
Croissant Décroissant <input type="text" value="android"/> <input type="button" value="filtrer"/>	Croissant Décroissant <input type="text"/> <input type="button" value="filtrer"/>	<input type="text" value="v"/>	<input type="button" value="Ancien"/> <input type="button" value="Récent"/>
Android Studio (complément n°13) : Permissions	Compléments Android (programmation mobile)	Android	29/09/2019
Android Studio (complément n°12) : Positionner texte sur photo	Compléments Android (programmation mobile)	Android	17/09/2019

MISSION 2 : CODER LA PARTIE BACK-OFFICE

TACHE 2: GÉRER LES PLAYLISTS

Tâche 2 : gérer les playlists (5h)

- 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.

1-CRÉATION DE LA PAGE

Pour mettre en place une interface de gestion des playlists, je vais concevoir une page nommée "admin/playlists.html.twig" dans le répertoire "templates>admin". En parallèle, je vais élaborer la classe "AdminPlaylistsController" dans le dossier "controller" pour gérer les itinéraires menant à la page "admin/playlists.html.twig".

Dans mon fichier "baseadmin.html.twig", je vais inclure le lien vers la gestion des playlists dans la barre de navigation sous le nom "admin.playlists".

```

baseadmin.html.twig
basefront.html.twig
</li>
<li class="nav-item">
  <a class="nav-link" href="{{ path('admin.playlists') }}">Playlists</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="#">Catégories</a>
</li>
</div>

```

Je vais créer le canevas initial de ma page d'administration des playlists. Cette structure servira ensuite à afficher le tableau des playlists ainsi que leurs catégories, et à offrir les fonctionnalités d'ajout, de modification et de suppression.

```

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

{% block body %}
    <h5>Gestion des playlists</h5>

```

Dans la classe "AdminPlaylistsController", je vais déclarer mes variables, écrire le constructeur et inclure la méthode "index" qui contient le chemin d'accès vers la nouvelle page de l'application.

```

private $formationRepository;

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

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

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

/**

```


2-AFFICHAGE DE LA LISTE DES PLAYLISTS :

Pour afficher la liste des playlists, je vais dans la page "admin/playlists.html.twig" et je vais créer une boucle "for" pour récupérer le nom des playlists et les placer dans une première colonne. Ensuite, j'afficherai les catégories, le nombre de formations par playlist et un bouton "Editer"

```

-
<!-- boucle sur les playlists -->
{% if playlists|length > 0 %}
  {% for k in 0..playlists|length-1 %}
    <tr class="align-middle">
      <td>
        <h5 class="text-info">
          {{ playlists[k].name }}
        </h5>
      </td>
      <td 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>
        <h5 class="text-center">
          {{ playlists[k].formations|length }}
        </h5>
      </td>
      <td>
        <a href="{{ path('admin.playlist.edit', {id:playlists[k].
          Editer
        </a>

```

Je crée mes fichiers twig pour l'ajout et la modification de playlist.

3-SUPPRESSION DES PLAYLISTS:

Pour supprimer une playlist, je crée une fonction dans "AdminPlaylistsController" qui utilise la méthode "remove" du "PlaylistsRepository", puis je redirige vers "admin.playlists.html.twig".

```

/**
 * Suppression d'une playlist
 * @Route("/admin/playlist/suppr/{id}", name="admin.playlist.suppr")
 * @param Playlist $playlist
 * @return Response
 */
public function suppr(Playlist $playlist):
    Response
{
    if(count($playlist->getFormations()) > 0){
        $this->addFlash('alert', 'Impossible de supprimer la playlist ' . $playlist->getName() . ' car elle contient des formations');
        return $this->redirectToRoute('admin.playlists');
    }

    $this->playlistRepository->remove($playlist, true);
    $this->addFlash('alert', 'La suppression de la playlist ' . $playlist->getName() . ' a été effectuée avec succès');
    return $this->redirectToRoute('admin.playlists');
}

```

Pour ajouter la fonction de suppression des playlists, je crée une commande dans "admin.playlists.html.twig" pour le bouton "Supprimer". Cette commande sera placée à la fin de la boucle "for" pour chaque playlist et précisera la route vers la fonction "admin.suppr.playlist" dans "AdminPlaylistsController", qui effectue la suppression dans la base de données. Pour assurer la suppression uniquement des playlists sans formation, j'ajoute une condition "if" dans la colonne de suppression pour afficher le bouton "Supprimer" uniquement lorsque le nombre de formations est égal à zéro. De plus, je vais ajouter un "onclick" pour demander confirmation avant la suppression.

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

4- ÉDITION DES PLAYLISTS

Pour ajouter la fonctionnalité d'édition des playlists, je vais créer un nouveau bouton "Éditer" dans "admin.playlists.html.twig". Lorsque ce bouton est cliqué, il redirigera vers un formulaire d'édition de la playlist. Pour créer ce formulaire, je vais réaliser une nouvelle page "admin.playlists.html.twig" dans le répertoire racine des modèles, contenant le formulaire d'édition.

Pour créer le formulaire dans la page "admin.playlists.form.html.twig", je vais concevoir une nouvelle classe PHP nommée "PlaylistType" dans le dossier "src/Form". Cette classe servira à définir les champs du formulaire d'ajout ou d'édition d'une playlist.

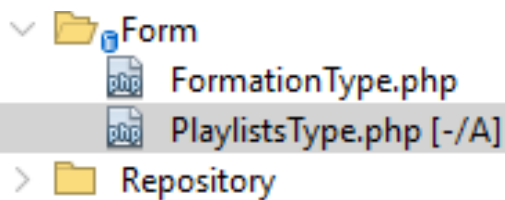
Pour créer le formulaire, nous allons ajouter deux champs : "name" pour le nom et la description de la playlist, et "formations" pour afficher la liste des formations associées. Ensuite, nous incluons un bouton "submit" pour enregistrer les informations dans la base de données.

Pour récupérer le champ avec le nom de la playlist sélectionnée et sa description, nous devons ajouter au builder une instruction "->add name" de type "Text" avec un "label => Playlist" pour afficher "Playlist" à côté du champ. Nous précisons "'required' => true" car le nom de la playlist est obligatoire. Ensuite, nous ajoutons un "->add description" de type "TextArea" avec "'required' => false" car la description n'est pas obligatoire, et nous lui donnons un "label => Description".

Pour inclure le champ "formations" dans le formulaire, nous devons afficher une liste déroulante des formations associées à la playlist en cours d'édition. Pour cela, nous ajoutons la classe "Formation" au builder du formulaire en utilisant la commande suivante : "->add('formation', EntityType::class,".

Ensuite, nous spécifions la classe "Formation" avec un label "formation" et nous utilisons "choice_label => title" pour récupérer les titres des formations associées. Puisque plusieurs formations peuvent être sélectionnées par playlist, nous ajoutons "'multiple' => true". Comme il n'est pas obligatoire d'avoir une formation, nous fixons "required" à "false".

Enfin, je vais ajouter "->add('submit', SubmitType::class)" pour permettre l'enregistrement des données dans la base de données lorsque l'utilisateur clique sur un bouton "Enregistrer".



```
*/
class PlaylistsType 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'
            ]);
    }
}
```

Dans votre fichier "AdminPlaylistsController", vous pouvez désormais créer la fonction associée au bouton "Éditer". Cette fonction utilisera la classe "PlaylistType" pour définir les champs du formulaire et récupérera les playlists existantes.

Ensuite, la méthode "\$formPlaylist->handleRequest(\$request)" gèrera la soumission du formulaire. Vous ajouterez une condition "if" pour vérifier si le formulaire a été soumis et s'il est valide. Si c'est le cas, les informations du formulaire seront enregistrées dans la base de données en utilisant la méthode "add" du PlaylistRepository, puis l'utilisateur sera redirigé vers la page de gestion des playlists.

Si le formulaire n'est pas soumis ou s'il n'est pas valide, la fonction maintiendra l'utilisateur sur la page d'édition du formulaire.

```

/**
 * Edition d'une playlist
 * @Route("/admin/edit.playlists/{id}", name="admin.edit.playlist")
 * @param Playlist $playlists
 * @param Request $request
 * @return Response
 */
public function edit(Playlist $playlists, Request $request): Response{
    $formPlaylist = $this->createForm(PlaylistType::class, $playlists);

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

    return $this->render("admin/admin.edit.playlists.html.twig", [
        'playlists' => $playlists,
        'formplaylist' => $formPlaylist->createView()
    ]);
}

```

Pour afficher le formulaire dans la page "admin.playlists.form.html.twig", vous devez inclure "{{ form_start(formplaylist) }}" au début du code pour indiquer le début du formulaire, et "{{ form_end(formplaylists) }}" à la fin. Ensuite, vous pouvez construire une colonne pour chaque champ du "PlaylistsType".

Pour afficher les informations dans les champs, utilisez "{{ form_row(formplaylist.name) }}". Comme les informations proviennent du formulaire créé dans AdminPlaylistsController, vous devez précéder chaque champ avec "formplaylists".

```

{{ form_start(formplaylist, {'attr': {'class': 'form'}}) }}
<div class="row">
  <div class="col">
    <div class="row">
      <div class="col-md-4">
        <div class="form-group">
          {{ form_row(formplaylist.name, {'attr': {'class': 'form-control', 'placeholder': 'Playlist'}}) }}
        </div>
      </div>
      <div class="col-md-4">
        <div class="form-group">
          {{ form_row(formplaylist.description, {'attr': {'class': 'form-control', 'placeholder': 'Description'}}) }}
        </div>
      </div>
      <div class="col-md-4">
        <div class="form-group">
          {{ form_row(formplaylist.formation, {'attr': {'class': 'form-control'}}) }}
        </div>
      </div>
    </div>
    <div class="col">
      <div class="form-group">
        {{ form_row(formplaylist.submit, {'attr': {'class': 'btn btn-success'}}) }}
      </div>
    </div>
  </div>
</div>
{{ form_end(formplaylist) }}

```

Dans la page "admin/playlists.html.twig", vous allez ajouter un bouton "Éditer" avec le chemin vers votre fonction "admin.edit.playlists". Lorsque ce bouton est cliqué, la méthode associée pourra s'exécuter.

```
</td>  
<td class="text-center">  
  <a href="{{ path('admin.edit.playlists', (id:playlists[k].id) }}" class="btn btn-warning">Editer</a>  
</td>
```

J'exécute l'application

The screenshot shows a web application interface. At the top, there is a table with the following content:

Bases de la programmation (C#)	C# POO	74	Editer	Supprimer
--------------------------------	--------	----	------------------------	---------------------------

Below the table, there is a search bar with the text "Bases de la programmation (C#)". To the right of the search bar, there is a dropdown menu with the text "Exemples progressifs de programmes en procédural, événementiel et objet sous". Below the search bar, there is a list of formations:

- POO TP Java n°4 : démarrage sur le contrôleur, construction du modèle Playlist : POO TP Java
- POO TP Java n°3 : interface graphique Playlist : POO TP Java
- POO TP Java n°2 : MVC Playlist : POO TP Java
- POO TP Java n°1 : configuration d'Eclipse Playlist : POO TP Java
- Bases de la programmation n°74 - POO : collections Playlist : Bases de la programmation (C#)

At the bottom of the list, there is a green button labeled "Valider".

5-AJOUTER UNE PLAYLIST

Grâce à la fonctionnalité d'édition d'une playlist, je n'ai pas besoin de recréer un nouveau formulaire, car il a déjà été construit. Je n'ai pas non plus besoin de faire des modifications dans le builder.

Grâce à la fonctionnalité d'édition d'une playlist, je n'ai pas besoin de recréer un nouveau formulaire, car il a déjà été construit. Je n'ai pas non plus besoin de faire des modifications dans le builder.

Pour ajouter une fonction d'ajout de playlist dans mon AdminPlaylistsController, je vais créer une nouvelle page `admin.ajout.playlists.html.twig` dans le dossier `templates>admin`. Je vais en profiter pour y insérer l'include de la page `admin.playlists.form.html.twig` contenant le formulaire.

Ensuite, je vais ajouter le bouton et le chemin de la fonction d'ajout dans la page `admin/playlist.html.twig`.

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

{% block body %}
    {{ include ('admin.playlists.form.html.twig') }}
{% endblock %}
```

Je réalise ensuite ma fonction d'ajout dans le `AdminPlaylistsController`. Le code est quasiment identique à celui de la fonction d'édition, puisque je récupère le même formulaire.

En guise de changements, je dois simplement affecter une nouvelle playlist à la table des playlists. Donc, j'ajoute la commande `"$playlists = new Playlist();"` juste avant de créer le formulaire.

```
public function ajout(Request $request):
    Response
{
    $playlist = new Playlist();
    $formPlaylist = $this->createForm(PlaylistType::class, $playlist);

    $formPlaylist->handleRequest($request);
    if ($formPlaylist->isSubmitted() && $formPlaylist->isValid()){
        $this->playlistRepository->add($playlist, true);
        $formations = $playlist->getFormations()->toArray();
        foreach($formations as $formation) {
            $formation->setPlaylist($playlist);
            $this->formationRepository->add($formation, true);
        }
        $this->addFlash(
```

Enfin, j'ajoute mon bouton "Ajouter une playlist" à ma page "admin/playlists.html.twig", en précisant la route vers la fonction "admin.ajout.playlists", pour que le clic sur le bouton puisse exécuter la méthode.

Je place mon bouton dans une balise `<p class="text-end">` juste en dessous du bloc body, pour qu'il se situe tout en haut à droite de ma page afin d'obtenir un meilleur visuel.

Ensuite, j'exécute mon application et je me dirige sur la page des playlists pour vérifier l'affichage du bouton "Ajouter une playlist", la redirection vers le formulaire, ainsi que le bon fonctionnement de l'ajout d'une nouvelle playlist.

Gestion des playlists

Ajouter une nouvelle playlist

Playlist Description

Formations

Eclipse n°8 : Déploiement Playlist : Eclipse et Java
Eclipse n°7 : Tests unitaires Playlist : Eclipse et Java
Eclipse n°6 : Documentation technique Playlist : Eclipse et Java
Eclipse n°5 : Refactoring Playlist : Eclipse et Java

Valider

6-TRIS ET FILTRES:

Dans la gestion des playlists sur la page dédiée, ma première tâche consiste à permettre le tri ascendant et descendant de la liste des playlists. Pour ce faire, je vais créer une fonction nommée "admin.playlists.sort", qui sera invoquée lorsqu'un utilisateur clique sur les boutons "Croissant" et "Décroissant" dans le fichier "admin/playlists.html.twig".

Cette fonction aura pour objectif de trier les playlists en fonction de leur nom et du nombre de formations qu'elles contiennent. Pour cela, je ferai appel aux

méthodes "findAllOrderByName" et "findAllOrderByAmount" du "PlaylistRepository".

La fonction prendra deux paramètres, \$champ et \$ordre. Si \$champ est égal à "name", la fonction utilisera la méthode "findAllOrderByName" pour récupérer la liste des playlists, qu'elle triera ensuite en fonction de leur nom, respectant l'ordre spécifié dans la page "admin/playlists.html.twig".

Si \$champ est égal à "nombre", la méthode appelée sera "findAllOrderByAmount". Dans ce cas, le tableau des playlists sera récupéré et trié en fonction du nombre de formations qu'elles contiennent, suivant l'ordre choisi dans la page "admin/playlists.html.twig".

une fois le tri effectué, la fonction récupérera la liste des catégories avant de rediriger l'utilisateur vers la page des playlists lorsqu'il cliquera sur les boutons "croissant" ou "décroissant".

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

Ensuite, il faut utiliser la méthode dans la page "admin/playlists.html.twig" lors de la création des boutons "Croissant" et "Décroissant". Pour ce faire, j'ajoute le chemin "admin.playlists.sort" à chacun des deux boutons et je détermine le type de tri en incluant "ASC" comme paramètre \$ordre pour le bouton croissant et "DESC" pour le bouton décroissant. Le champ recherché est défini comme "name", car il contient le nom des playlists et je souhaite que le tri se fasse en fonction de ce champ.

```

<th class="text-left align-top" scope="col">
  Playlist<br>
  <a href="{{ path('admin.playlists.sort', {champ:'name', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button"
  <a href="{{ path('admin.playlists.sort', {champ:'name', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button"
  <form class="form-inline mt-1" method="POST" action="

```

Je procède de la même manière que pour le tri des playlists, puis je ajuste la valeur du \$champ pour qu'il soit "nbformations", car le tri doit être effectué en fonction du champ "nbformations".

```

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

```

J'effectue une vérification sur le fonctionnement

Playlist

Croissant Décroissant

 [filtrer](#)

Visual Studio 2019 et C#

Nombre de formations	Détail	Editer
Croissant Décroissant		
1	détail	Editer

Playlist

Croissant Décroissant

 [filtrer](#)

Bases de la programmation (C#)

Nombre de formations	Détail	Editer
Croissant Décroissant		
74	détail	Editer

Je dois maintenant mettre en place le filtrage des playlists et des catégories via un formulaire de recherche permettant de restreindre le contenu affiché dans le tableau en fonction de la valeur saisie.

Pour cela, je vais créer une méthode "findAllContain" dans le contrôleur "AdminPlaylistController". Cette méthode sera chargée de récupérer la valeur saisie dans le formulaire de recherche et de la comparer aux champs des différentes tables.

Tout d'abord, je vais extraire la valeur de la zone de recherche à l'aide de la requête "get("recherche")", que je vais stocker dans la variable \$valeur.

Si une table est spécifiée en paramètre, la méthode appellera la fonction "findByContainValue" du "PlaylistRepository" pour retourner les enregistrements correspondants à la table et au champ donnés pour la valeur saisie.

Si aucune table n'est spécifiée, la méthode appellera la fonction "findByContainValue" du "PlaylistRepository" pour retourner les enregistrements correspondant au champ donné pour la valeur saisie.

Enfin, la méthode récupérera également la liste de toutes les catégories.

```
/* Tri des enregistrements selon le nombre des playlists
 * @Route("/admin/playlists/recherche/{champ}/{table}", name="admin.playlists.findAllContain")
 * @param type $champ
 * @param Request $request
 * @param type $table
 * @return Response
 */
public function findAllContain($champ, Request $request, $table=""):
    Response{
        $valeur = $request->get("recherche");
        $playlists = $this->playlistRepository->findByContainValue($champ, $valeur, $table);
        $categories = $this->categorieRepository->findAll();
        return $this->render(self::PAGE_PLAYLISTS, [
            'playlists' => $playlists,
            'categories' => $categories,
            'valeur' => $valeur,
            'table' => $table
        ]);
    };
```

Maintenant que j'ai créé la méthode permettant d'enregistrer les saisies d'un formulaire de recherche, je dois créer les formulaires de recherche dans la page "admin/playlists.html.twig" et leur attribuer un champ et éventuellement une table.

Pour créer un formulaire de recherche, je vais utiliser les balises <form> en spécifiant la méthode "POST" et en définissant le chemin vers la fonction à appeler, ici "admin.playlists.findallcontain".

À l'intérieur de la balise <div>, je vais ajouter un champ de saisie (input) pour permettre à l'utilisateur de saisir une valeur dans la zone de recherche. Je vais également ajouter un second champ pour générer un jeton CSRF afin de sécuriser le formulaire.

```
Playlist<br>
<a href="{{ path('admin.playlists.sort', {champ:'name', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button">ASC</a>
<a href="{{ path('admin.playlists.sort', {champ:'name', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button">DESC</a>
<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="form-control" name="recherche" value="{{ if valeur|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>
```

Maintenant, je vais mettre en place le filtre pour les catégories. Pour ce faire, je vais créer un formulaire d'une seule ligne qui affichera la liste des catégories. Cela se fera en appelant la méthode "admin.playlists.findallcontain", en remplaçant le paramètre \$champ par "id" et en ajoutant le paramètre \$table "categories".

Ensuite, j'ajouterai une balise <select> qui permettra à l'utilisateur de sélectionner une catégorie dans la liste déroulante de recherche.

Une fois qu'une catégorie est sélectionnée, le formulaire renverra les enregistrements correspondant à cette catégorie.

Enfin, je vais inclure la boucle "for" nécessaire au bon fonctionnement de la recherche par catégorie.

```
Categories<br>
<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>
```

Je lance l'application pour vérifier que les filtres de recherche s'affichent correctement

Gestion des playlists

Ajouter une nouvelle playlist

Playlist	Catégories	Nombre de formations	Détail	Editer
Croissant Décroissant	<input type="text" value=""/>	Croissant Décroissant		
<input type="text" value="c#"/> filtrer				
Bases de la programmation (C#)	C# POO	74	détail	Editer
Visual Studio 2019 et C#	C# POO	11	détail	Editer

Playlist	Catégories	Nombre de formations	Détail	Editer
Croissant Décroissant	<input type="text" value="MCD"/> <input type="text" value=""/>	Croissant Décroissant		
<input type="text" value=""/> filtrer				
Cours MCD MLD MPD	MCD Cours	2	détail	Editer
Cours MCD vs Diagramme de classes	MCD Cours	2	détail	Editer
Cours Merise/2	MCD Cours	1	détail	Editer
Cours Modèle relationnel et MCD	MCD Cours	1	détail	Editer
MCD : exercices progressifs	MCD	18	détail	Editer
MCD exercices d'examen (sujets EDC BTS SIO)	MCD	8	détail	Editer

MISSION 2 : CODER LA PARTIE BACK-OFFICE

TACHE 3: GÉRER LES CATÉGORIES

Tâche 3 : gérer les catégories (3h)

- 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.

1-PAGE CATÉGORIES

Pour mettre en place une page de gestion des catégories, je vais créer un modèle nommé "admin/categories.html.twig" dans le répertoire "templates/admin". Ce modèle sera dédié à la gestion des catégories.

Ensuite, je vais écrire une classe nommée "AdminCategoriesController" dans le dossier "Controller". Cette classe va gérer les routes vers la page "admin.categories.html.twig", ainsi que toutes les opérations nécessaires pour la gestion des catégories.

Je vais créer le cadre initial de ma page d'administration des catégories. Cette page permettra dans un premier temps d'afficher un tableau présentant les catégories ainsi que leurs formations associées. Par la suite, elle offrira la possibilité d'ajouter, de modifier ou de supprimer des catégories.

```

{% extends "baseadmin.html.twig" %}
{% block body %}
<h5>Gestion des catégories</h5>
{% for message in app.flashes('success') %}
<div class="alert alert-success mt-4">
  {{ message }}
</div>
{% endfor %}
{% for message in app.flashes('alert') %}
<div class="alert alert-warning mt-4">
  {{ message }}
</div>
{% endfor %}
<p class="text-end">
  <a href="{{ path('admin.categorie.ajout') }}" class="btn btn-success">
    Ajouter une categorie
  </a>
</p>
<table class="table table-striped">
  <caption>tableau des categories</caption>
  <thead>
  <tr>
  <th class="text-left align-top" scope="col">
    Catégories<br>
    <a href="{{ path('admin.categories.sort', {champ:'name', ordre:'AS'}) }}">
    <a href="{{ path('admin.categories.sort', {champ:'name', ordre:'DE'}) }}">
    <form class="form-inline mt-1" method="POST" action="{{ path('admin.categories.sort', {champ:'name', ordre:'AS'}) }}">
      <div class="form-group mr-1 mb-2">
        <input type="text" class="sm" name="recherche"
          value="{% if valeur|default and not table|default %}{{ valeur }}{% else %}</div>
      <input type="hidden" name="_token" value="{{ csrf_token('admin.categories.sort') }}">
      <button type="submit" class="btn btn-info btn-sm active">Filtrer
    </form>
  </th>
  </thead>
  </table>

```

Dans mon fichier "baseadmin", je vais inclure le lien vers ma page d'administration des catégories dans la barre de navigation.

```

<li class="nav-item">
  <a class="nav-link" href="{{ path('admin.categories') }}">Catégories</a>
</li>
v>

```

Dans ma classe "AdminCategoriesController", je vais initialiser mes variables, définir le constructeur et mettre en place la méthode index qui contient la route vers la nouvelle page de l'application. Cela permettra de gérer efficacement les données et les actions liées à la gestion des catégories.

```

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

/**
 * Affiche la liste des catégories.
 * Cette méthode récupère le terme de recherche (s'il existe) à partir de la requête,
 * @Route("/admin/categories", name="admin.categories")
 * @return Response
 */

public function index(Request $request): Response
{
    $searchTerm = $request->query->get('search');

    if ($searchTerm) {
        $categories = $this->categorieRepository->findBySearchTerm($searchTerm);
    } else {
        $categories = $this->categorieRepository->findAll();
    }

    $formations = $this->formationRepository->findAll();
}

```

2-AFFICHAGE DES CATÉGORIES

Pour afficher la liste des catégories dans la page "admin.categories.html.twig", je vais utiliser une boucle "for" pour parcourir les catégories. Dans cette boucle, je vais récupérer le nom de chaque catégorie et l'afficher dans une première colonne du tableau. Ensuite, j'afficherai les formations associées à chaque catégorie dans une seconde colonne du tableau.

```

<tbody>
{% for categorie in categories %}
    <tr class="align-middle">
        <td>
            <h5 class="text-info">
                {{ categorie.name }}
            </h5>
        </td>
        <td class="text-left">
            {% for formation in categorie.formations %}

```


3-SUPPRIMER UNE CATÉGORIE

Pour permettre la suppression d'une catégorie, je vais créer une méthode dans "AdminCategoriesController" qui récupère la méthode "remove" du "CategorieRepository". Ensuite, je vais effectuer une redirection vers la route correspondant à ma page "admin.categories.html.twig".

```
/**
 * Suppression d'une catégorie
 * Redirection vers la page d'administration
 * @Route("/admin/categories/suppr/{id}", name="admin.categories.suppr")
 * @param Categorie $categorie
 * @return Response
 */
public function suppr(Categorie $categorie): Response{
    $this->categorieRepository->remove($categorie, true);
    return $this->redirectToRoute('admin.categories');
}
```

Je crée ensuite le bouton "Supprimer" dans la page "admin/categories.html.twig". Pour cela, je place cette commande à la fin de ma boucle "for" afin d'ajouter une colonne qui affichera les boutons "Supprimer" pour chaque catégorie.

Je dois spécifier la route de ma fonction "admin.suppr.categories" dans le "AdminCategoriesController", qui permet de supprimer les catégories de la base de données.

Afin de ne permettre la suppression que des catégories ne contenant aucune formation, j'ajoute une condition "if" dans ma colonne de suppression. Si le nombre de formations associées à une catégorie est égal à zéro, le bouton "Supprimer" est activé pour cette catégorie, sinon il reste désactivé.

J'ajoute un événement "onclick" pour demander la confirmation de la suppression lors du clic sur le bouton "Supprimer".

```
<td>
    {% if categorie.formations|length == 0 %}
        <a href="{{ path('admin.categories.suppr', {id:categorie.id}) }}" class="btn btn-danger" onclick=
    {% else %}
        <button type="button" class="btn btn-danger disabled">Supprimer</button>
    {% endif %}
</td>
```

4-AJOUTER UNE CATÉGORIE

Pour ajouter une catégorie qui n'existe pas déjà dans la liste des catégories, je dois créer une méthode "findAllEqual" dans le "CategorieRepository". Cette méthode prendra le nom de la catégorie en paramètre.

La requête contenue dans la méthode aura pour but de sélectionner le nom des catégories présentes dans la table des catégories, puis de les comparer avec la valeur saisie par l'utilisateur lors de l'ajout d'une catégorie dans le formulaire de recherche. Cela permettra de vérifier si une catégorie avec le même nom existe déjà dans la base de données.

```
public function findAllEqual($name) : array {  
    return $this->createQueryBuilder('c')  
        ->select('c.name name')  
        ->where('c.name=:name')  
        ->setParameter('name', $name)  
        ->getQuery()  
        ->getResult();  
}
```

Ensuite, je crée la fonction d'ajout d'une catégorie "admin.ajout.categorie" dans la classe "AdminCategoriesController". Cette fonction appellera la méthode "findAllEqual" du "CategorieRepository" pour comparer les noms des catégories avec celui saisi dans le formulaire.

En ajoutant une condition "if", je vérifie si le nom de la catégorie n'est pas déjà présent dans la liste. Si c'est le cas, la nouvelle catégorie est créée.

Enfin, je redirige l'utilisateur vers la page d'administration des catégories après l'ajout de la catégorie.

```

/**
 * Ajout d'une catégorie
 * @Route("/admin/categorie/ajout", name="admin.categorie.ajout")
 * @param Request $request
 * @return Response
 */
public function ajout(Request $request): Response
{
    $categorie = new Categorie();
    $formCategorie = $this->createForm(CategorieType::class, $categorie);

    $formCategorie->handleRequest($request);
    if ($formCategorie->isSubmitted() && $formCategorie->isValid()){
        $this->categorieRepository->add($categorie, true);
        $formations = $categorie->getFormations()->toArray();
        foreach($formations as $formation) {
            $this->categorieRepository->addFormationCategorie($formation->getId(), $categorie->getId());
        }
        $this->addFlash(
            'success',
            'Ajout de la categorie ' . $categorie->getName() . " prise en compte"
        );
        return $this->redirectToRoute('admin.categories');
    }

    return $this->render(self::PAGE_CATEGORIE, [
        'categorie' => $categorie,
        'formCategorie' => $formCategorie->createView()
    ]);
}

```

Ensuite, je dois construire un mini-formulaire permettant d'ajouter une catégorie dans la page "admin.categories.html.twig".

```

</div>
</form>
</th>
</tr>
</table>
<button type="submit" class="btn btn-success">Ajouter une catégorie</button>

```

la méthode edit:

```
public function edit(Categorie $categorie, Request $request): Response
{
    // Récupération des formations initiales liées à la catégorie
    $formationsIni = $categorie->getFormations()->toArray();

    // Création du formulaire de modification de catégorie
    $formCategorie = $this->createForm(CategorieType::class, $categorie);
    $formCategorie->handleRequest($request);

    // Traitement du formulaire s'il est soumis et valide
    if ($formCategorie->isSubmitted() && $formCategorie->isValid()) {
        // Enregistrement des modifications de la catégorie
        $this->categorieRepository->add($categorie, true);

        // Récupération des formations après modification
        $formations = $categorie->getFormations()->toArray();

        // Ajout des nouvelles formations liées à la catégorie
        foreach ($formations as $formation) {
            if (!in_array($formation, $formationsIni)) {
                $this->categorieRepository->addFormationCategorie($formation->getId(), $categorie->getId());
            }
        }

        // Suppression des formations retirées de la catégorie
        foreach ($formationsIni as $formation) {
            if (!in_array($formation, $formations)) {
                $this->categorieRepository->delFormationCategorie($formation->getId(), $categorie->getId());
            }
        }

        // Ajout d'un message de succès
        $this->addFlash(
            'success',
            'Modification de la catégorie ' . $categorie->getName() . ' prise en compte.'
        );

        // Redirection vers la liste des catégories
        return $this->redirectToRoute('admin.categories');
    }
}
```

```
<td>
    <a href="{( path('admin.categorie.edit', {id:categories[k].id }) )}" class="btn btn-warning">
        Editer
    </a>
</td>
<+A>
```

Je lance l'application pour vérifier que tous les éléments fonctionnent correctement.

Ajouter une categorie

Catégories	Formations	Edition/Suppression	
<p>Croissant Décroissant</p> <input type="text"/> <p>Filter</p>	<p>Android Studio (complément n°13) : Permissions Android Studio (complément n°12) : Positionner texte sur photo Sujet E5 SLAM 2019 : cas RESTILOC mission3 (SQL et Android) Android Studio (complément n°11) : Transformer une image en texte Android Studio (complément n°10) : Ajout icone dans menu Android Studio (complément n°9) : Ajout texte sur photo Android Studio (complément n°8) : Enregistrer une photo Android Studio (complément n°7) : Prendre une photo Android Studio (complément n°6) : Redimensionner des photos Android Studio (complément n°5) : Récupérer les photos du mobile Android Studio (complément n°4) : Envoyer un SMS Android Studio (complément n°3) : Activity dépendante Android Studio (complément n°2) : Récupérer les contacts du mobile Android Studio (complément n°1) : Navigation Drawer et Fragment TP Android n°18 : liste adapter interactive (4) TP Android n°17 : liste adapter interactive (3) TP Android n°16 : liste adapter interactive (2) TP Android n°15 : liste adapter interactive (1) TP Android n°14 : plusieurs interfaces TP Android n°13 : formatage de la date TP Android n°12 : base de données distante MySQL (4) TP Android n°11 : base de données distante MySQL (3) TP Android n°10 : base de données distante MySQL (2) TP Android n°9 : base de données distante MySQL (1)</p>	<p>Editer</p>	<p>Supprimer</p>
Android			

Croissant Décroissant

catégorie teste	Aucune formation associée	Supprimer
-----------------	---------------------------	-----------

Etes-vous sûr de vouloir supprimer catégorie teste ?

OK Annuler

Android

- Python n°14 : Message Playlist : Programmation
- Python n°13 : Encapsulation Playlist : Program
- Python n°12 : Classe et liste d'objets Playlist : F
- Python n°11 : dictionnaire et IDE PyCharm Play
- Python n°10 : gestion des exceptions et utilis
- Python n°9 : Listes Playlist : Programmation so
- Python n°8 : Fonctions et bibliothèques Playlis
- Python n°7 : Conversion binaire et menu Playli
- Python n°6 : Nombre premier, test et booléen
- Python n°5 : Produit de valeurs Playlist : Progr
- Python n°4 : bibliothèque math Playlist : Progr
- Python n°3 : Somme entre 2 bornes Playlist : P
- Python n°2 : boucle for Playlist : Programmatic
- Python n°1 : boucle simple, saisie, affichage Pl
- Python n°0 : installation de Python Playlist : Pr
- Android Studio (complément n°13) : Permiss

Valider

MISSION 2 : CODER LA PARTIE BACK-OFFICE

TÂCHE 4 : AJOUTER L'ACCÈS AVEC AUTHENTIFICATION

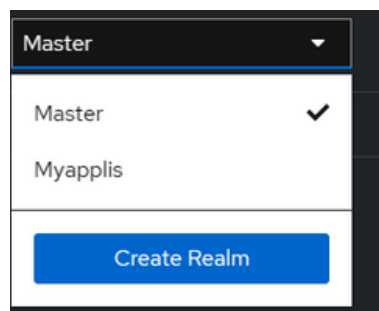
Tâche 4 : ajouter l'accès avec authentification (4h)

- 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).

Pour commencer, je vais configurer Keycloak afin de le relier à l'application et permettre l'authentification de l'utilisateur pour accéder aux pages d'administration. Pour cela, je vais démarrer le serveur Keycloak en exécutant la commande "kc.bat start-dev" dans l'invite de commandes Windows en mode Administrateur. Pour effectuer cette opération, je dois me positionner dans le dossier "C:/keycloak/bin".

```
C:\keycloak-24.0.1\bin>kc.bat start-dev
```

Il est nécessaire de mettre en place un compte administrateur pour pouvoir se connecter au logiciel en local. Ensuite, il sera possible de procéder à la configuration de Keycloak pour l'application. La première étape consistera à créer le royaume "Myapplis".



Puis je crée un nouveau client, mediatek pour l'application Symfony Mediatek Formation

Réglages généraux

Identité du client *

Nom

Description

Toujours afficher dans l'interface utilisateur Désactivé

- Aller à la section
- Réglages généraux
 - Accéder aux paramètres
 - Configuration des capacités
 - Paramètres de connexion
 - Paramètres de déconnexion

Accéder aux paramètres

URL racine

URL d'accueil

URI de redirection valides [Ajouter des URI de redirection valides](#)

URI de redirection après déconnexion valides [Ajouter des URI de redirection de déconnexion valides](#)

Origines du Web [Ajouter des origines Web](#)

URL d'administration

Configuration des capacités

Authentification client Sur

Autorisation Désactivé

Flux d'authentification Débit standard Subventions d'accès direct

Flux implicite Rôles des comptes de service

Octroi d'autorisation d'appareil OAuth 2.0

Subvention OIDC CIBA

- Paramè
- Paramè

Paramètres de connexion

Thème de connexion

Consentement requis Sur

Afficher le client à l'écran Sur

Texte de l'écran de consentement

Je configure le client et je récupère « l'app secret »

Client secret: W3HB7HkAMMdeOH3q4CWgMH1beffMAJpE 🗑️ 📄 [Regenerate](#)

Je vais saisir ces informations, ainsi que celles du client, dans le fichier .env situé à la racine de mon projet Symfony

```
KEYCLOAK_SECRET=W3HB7HkAMMdeOH3q4CWgMH1beffMAJpE
KEYCLOAK_CLIENTID=mediatek
KEYCLOAK_APP_URL=http://localhost:8080
```

Je crée un utilisateur pour l'application Mediatek Formation

admin2 Action ▾

< Details Attributes Credentials Role mapping Groups Consents Identity prov >

ID * c304447a-a7cb-49a1-8172-630c7484c905

Created at * 3/31/2024, 7:36:46 PM

Username * admin2

Email

Email verified Off

First name

Last name

Enabled On

Required user actions

[Save](#) [Revert](#)

🔔	Type	User label	Data	
☰	Password	My password	Show data	Reset password ⋮

Je crée une classe User dans le projet Symfony.

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

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

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

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

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

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

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

Success!

Next Steps:
- Review your new App\Entity\User class.
- Use make:entity to add more fields to your User entity and then run make:migration.
- Create a way to authenticate! See https://symfony.com/doc/current/security.html

C:\wamp64\www\mediatekformation>
```

```
C:\wamp64\www\mediatekformation>php bin/console make:entity User

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
> keycloakId

Field type (enter ? to see all types) [integer]:
> string

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
> yes

updated: src/Entity/User.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>

Success!

Next: When you're ready, create a migration with php bin/console make:migration

C:\wamp64\www\mediatekformation>
```

```
C:\wamp64\www\mediatekformation>php bin/console make:migration
```

```
created: migrations/Version20240329215111.php
```

Success!

Review the new migration then run it with `php bin/console doctrine:migrations:migrate`
See <https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html>

```
C:\wamp64\www\mediatekformation>
```

```
C:\wamp64\www\mediatekformation>php bin/console doctrine:migrations:migrate
```

```
WARNING! You are about to execute a migration in database "mediatekformation" that could result in schema changes  
and data loss. Are you sure you wish to continue? (yes/no) [yes]:
```

```
>
```

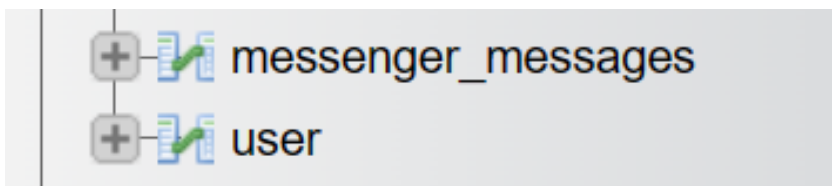
```
[notice] Migrating up to DoctrineMigrations\Version20240329215111
```

```
[notice] finished in 344.8ms, used 20M memory, 1 migrations executed, 8 sql queries
```

```
[OK] Successfully migrated to version: DoctrineMigrations\Version20240329215111
```

```
C:\wamp64\www\mediatekformation>
```

Je contrôle que la table "user" a été créée.



Il existe des bundles pour insérer dans Symfony tout le nécessaire pour le lien entre Symfony et Keycloak. Deux bundles doivent être installés.

Je lance les deux commandes :

- `composer require knpuniversity/oauth2-client-bundle 2.10`
- `composer require stevenmaguire/oauth2-keycloak 3.1 --with-all-dependencies`

Configuration du fichier config/packages/knpu_oauth2_client.yaml créé avec oauth2-client-bundle.

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

Configuration du firewall config/packages/security.yaml.

```
security:
  enable_authenticator_manager: true
  # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
  password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
  # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
  providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
      entity:
        class: App\Entity\User
        property: email
  firewalls:
    dev:
      pattern: ^/(_(profiler|wdt)|css|images|js)/
      security: false
    main:
      lazy: true
      provider: app_user_provider
      form_login:
        login_path: oauth_login

      # activate different ways to authenticate
      # https://symfony.com/doc/current/security.html#the-firewall

      # https://symfony.com/doc/current/security/impersonating_user.html
      # switch_user: true

  # Easy way to control access for large sections of your site
  # Note: Only the *first* access control that matches will be used
  access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }

when@test:
```

Le chemin menant au formulaire d'authentification a été défini comme étant `oauth_login`. De plus, le contrôle d'accès a été ajusté : en retirant le commentaire `#` au début de la ligne, je précise à Symfony que chaque contrôleur ayant une route contenant `/admin` exigera un statut "ROLE_ADMIN" de l'utilisateur pour y accéder, ce qui signifie qu'une connexion authentifiée sera nécessaire.

Créer le contrôleur qui va gérer l'authentification avec cette commande:
php bin/console make:controller OAuthController --no-template

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

Success!

Next: Open your new controller class and add some pages!

C:\wamp64\www\mediatekformation>
```

Configuration du contrôleur qui va gérer l'authentification

```
<?php

namespace App\Controller;

use KnpU\OAuth2ClientBundle\Client\ClientRegistry;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;

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

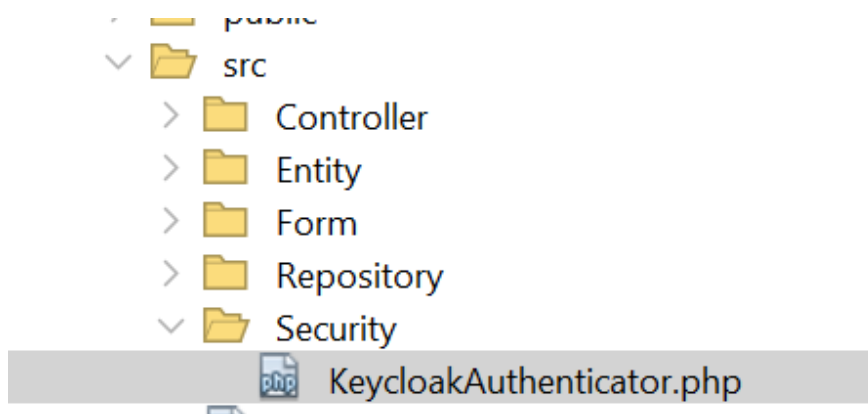
    /**
     * @Route("/oauth/callback", name="oauth_check")
     */
    public function connectCheckAction(Request $request, ClientRegistry $clientRegistry){

    }

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

    }
}
```

Dans le dossier "src", je crée un nouveau dossier "Security" et, dans ce dossier, une nouvelle classe PHP "KeycloakAuthenticator.php".



```

<?php
namespace App\Security;

use App\Entity\User;
use Doctrine\ORM\EntityManagerInterface;
use KnpU\OAuth2ClientBundle\Client\ClientRegistry;
use KnpU\OAuth2ClientBundle\Security\Authenticator\OAuth2Authenticator;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\RouterInterface;
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;
use Symfony\Component\Security\Core\Exception\AuthenticationException;
use Symfony\Component\Security\Http\Authenticator\Passport\Badge\UserBadge;
use Symfony\Component\Security\Http\Authenticator\Passport\Passport;
use Symfony\Component\Security\Http\Authenticator\Passport\SelfValidatingPassport;
use Symfony\Component\Security\Http\EntryPoint\AuthenticationEntryPointInterface;

/**
 * Description of KeycloakAuthenticator
 *
 * @author louiza
 */
class KeycloakAuthenticator extends OAuth2Authenticator implements AuthenticationEntryPointInterface {

    private $clientRegistry;
    private $entityManager;
    private $router;

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

    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){

```

Je dois lancer l'application, qui me dirige vers ma page d'accueil, si j'ajoute "/admin" à la fin de l'URL pour accéder à la partie administration, je suis redirigée vers l'authentification Keycloak avec la demande de saisie du "username or email" et du "password"

Sign in to your account

Username or email

Password

Sign In

Je vais maintenant m'occuper de configurer ma déconnexion. Je vais ajouter un lien dans la partie "admin" pour pouvoir me déconnecter. Dans le dossier "templates", j'ouvre le fichier "baseadmin.html.twig". Dans ce fichier, au-dessus du titre, j'ajoute un lien "se déconnecter" qui redirige vers la route 'logout'.

```
<div class="text-end">
  <a href="{{ path('logout') }}" class="link-dark">se déconnecter</a>
</div>
```

Dans le contrôleur "OAuthController", situé dans le dossier "src > Controller", je vais ajouter la méthode suivante, qui sera appelée avec la route 'logout'. Cette méthode sera vide car c'est le firewall qui va prendre la relève.

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

Dans security.yaml (config > packages"), j'ajoute le logout.

```
custom_authenticators:
    - App\Security\KeycloakAuthenticator
logout:
    path: logout

# activate different ways to authenticate
# https://symfony.com/doc/current/security.html
```



MediaTek86

Des formations pour tous sur des outils numériques

se déconnecter

Back office

Retour à la page d'accueil

Ajouter une formation

Formation	Playlist	Catégorie	Date	Détails	Editer	Supprimer
Eclipse n°8 : Déploiement	Eclipse et Java	Java	04/01/2021		Editer	Supprimer
Eclipse n°7 : Tests unitaires	Eclipse et Java	Java	02/01/2021		Editer	Supprimer
Eclipse n°6 : Documentation					Editer	Supprimer

MISSION 3 : TESTER ET DOCUMENTER

TÂCHE 1 : GÉRER LES TESTS:

TEST UNITAIRES:

La tâche consiste à créer une classe de test appelée `FormationTest` pour vérifier le bon fonctionnement de la méthode renvoyant la date de parution au format string.

```
<?php
use App\Entity\Formation;
use PHPUnit\Framework\TestCase;

class FormationTest extends TestCase
{
    public function testGetPublishedAtString()
    {
        $Formation = new Formation();
        $Formation->setTitle("Formation test");
        $Formation->setPublishedAt(new \DateTime("2022-05-17"));
        $this->assertEquals("17/05/2022", $Formation->getPublishedAtString());
    }
}
```

je le lance avec la commande php bin/phpunit.

```
PHPUnit 9.6.16 by Sebastian Bergmann and contributors.

Testing
.                                                                    1 / 1 (100%)

Time: 00:00.115, Memory: 8.00 MB

OK (1 test, 1 assertion)

C:\wamp64\www\mediatekformation>
```

TESTS D'INTÉGRATION SUR LES RÈGLES DE VALIDATION :


```

<?php
namespace App\Tests\Validations;

use App\Entity\Formation;
use DateTime;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;
use Symfony\Component\Validator\Validator\ValidatorInterface;

/**
 * Assurez-vous que la date de la formation n'est pas ultérieure à la date actuelle lors de l'ajout ou de la modification d'une formation.
 */
class FormationValidationTest extends KernelTestCase{

    public function getFormation(): Formation{
        return (new Formation())
            ->setTitle('Nouvelle formation')
            ->setPublishedAt(new DateTime("2026/01/18"));
    }

    public function testValidDateFormation(){
        $this->assertErrors($this->getFormation()->setPublishedAt(new DateTime('yesterday')), 0, "04/10/2023 devrait réussir");
        $this->assertErrors($this->getFormation()->setPublishedAt(new DateTime("first day of January 2008")), 0, "01/01/2008 devrait réussir");
        $this->assertErrors($this->getFormation()->setPublishedAt(new DateTime("last sat of July 2008")), 0, "26/07/2008 devrait réussir");
    }

    public function testNonValidDateFormation()
    {
        $this->assertErrors($this->getFormation()->setPublishedAt(new DateTime('06/08/2026')), 1, "devrait échouer");
        $this->assertErrors($this->getFormation()->setPublishedAt(new DateTime('11/02/2025')), 1, "devrait échouer");
        $this->assertErrors($this->getFormation()->setPublishedAt(new DateTime('06/11/2028')), 1, "devrait échouer");
        $this->assertErrors($this->getFormation()->setPublishedAt(new DateTime('09/07/2025')), 1, "devrait échouer");
    }

    public function testValidationDateFormation(){
        $formation = $this->getFormation()->setPublishedAt(new DateTime("2026/01/18"));
        $this->assertErrors($formation, 1);
    }

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

```

J'ajoute une assertion `LessThanOrEqual('today')` à la propriété `publishedAt` de la classe `Formation`. Si une date renseignée est postérieure à la date actuelle, la donnée ne sera pas acceptée et le test ne sera pas valide.

```

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

```

```

C:\wamp64\www\mediatekformation>php bin/phpunit --filter FormationValidationTest
PHPUnit 9.6.16 by Sebastian Bergmann and contributors.

```

```

Testing

```

```

...

```

```

3 / 3 (100%)

```

```

Time: 00:02.220, Memory: 26.00 MB

```

```

OK (3 tests, 8 assertions)

```

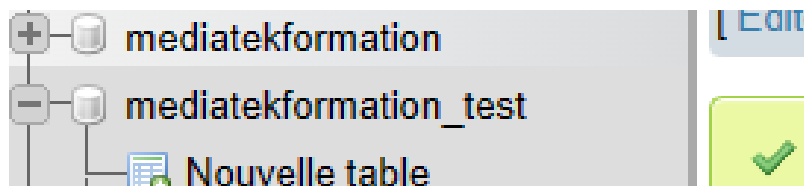
```

C:\wamp64\www\mediatekformation>

```


TESTS D'INTÉGRATION SUR LES REPOSITORY

Pour tester la classe `FormationRepository`, je crée une BDD de test "mediatekformation_test", je crée une classe de test appelée `FormationRepositoryTest`, qui hérite de `KernelTestCase`.



```
use App\Entity\Formation;
use App\Repository\FormationRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

class FormationRepositoryTest extends KernelTestCase
{
    private EntityManagerInterface $entityManager;
    private FormationRepository $repository;

    protected function setUp(): void
    {
        self::bootKernel();
        $this->entityManager = self::$container->get(EntityManagerInterface::class);
        $this->repository = self::$container->get(FormationRepository::class);
    }

    public function testNbFormations(): void
    {
        $nbFormation = $this->repository->count([]);
        $this->assertEquals(237, $nbFormation);
    }

    private function createFormation(): Formation
    {
        return (new Formation())
            ->setTitle("Un titre")
            ->setDescription("Description blabla")
            ->setPublishedAt(new \DateTime("yesterday"));
    }

    public function testAddFormation(): void
    {
        $nbFormationBefore = $this->repository->count([]);

        $formation = $this->createFormation();
        $this->entityManager->persist($formation);
        $this->entityManager->flush();
    }
}
```

php bin/phpunit --filter FormationRepositoryTest

```
C:\wamp64\www\mediatekformation>php bin/phpunit --filter FormationRepositoryTest
PHPUnit 9.6.16 by Sebastian Bergmann and contributors.

Testing
...
3 / 3 (100%)

Time: 00:01.655, Memory: 26.00 MB

OK (3 tests, 3 assertions)

C:\wamp64\www\mediatekformation>
C:\wamp64\www\mediatekformation>
```

J'effectue les mêmes tests pour CatégorieRepository

```
<?php

namespace App\tests\Repository;

use App\Entity\Categorie;
use App\Repository\CategorieRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

class CategorieRepositoryTest extends KernelTestCase
{
    private EntityManagerInterface $entityManager;
    private CategorieRepository $repository;

    protected function setUp(): void
    {
        self::bootKernel();
        $this->entityManager = self::$container->get(EntityManagerInterface::class);
        $this->repository = self::$container->get(CategorieRepository::class);
    }

    public function testNbCategorie(): void
    {
        $nbCategorie = $this->repository->count([]);
        $this->assertEquals(9, $nbCategorie);
    }

    private function createCategorie(): Categorie
    {
        return (new Categorie())
            ->setName("Un nom");
    }

    public function testAddCategorie(): void
    {
        $nbCategorieBefore = $this->repository->count([]);

        $categorie = $this->createCategorie();
        $this->entityManager->persist($categorie);
        $this->entityManager->flush();
    }
}
```

php bin/phpunit --filter CategorieRepositoryTest

```
PHPUnit 9.6.16 by Sebastian Bergmann and contributors.

Testing
... 3 / 3 (100%)

Time: 00:01.768, Memory: 26.00 MB

OK (3 tests, 3 assertions)
```

J'effectue les mêmes tests pour PlaylistsRepository

```
<?php

namespace App\tests\Repository;

use App\Entity\Playlist;
use Doctrine\ORM\EntityManagerInterface;
use App\Repository\PlaylistRepository;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

class PlaylistRepositoryTest extends KernelTestCase
{
    private EntityManagerInterface $entityManager;

    protected function setUp(): void
    {
        self::bootKernel();

        $this->entityManager = self::$container->get('doctrine')->getManager();
    }

    public function recupRepository(): PlaylistRepository
    {
        return self::$container->get(PlaylistRepository::class);
    }

    public function testNbPlaylist(): void
    {
        $repository = $this->recupRepository();
        $nbPlaylist = $repository->count([]);
        $this->assertEquals(27, $nbPlaylist);
    }

    public function newPlaylist(): Playlist
    {
        $playlist = (new Playlist())
            ->setName("Un nom");
        return $playlist;
    }

    public function testAddPlaylist(): void
    {

```

```
php bin/phpunit --filter PlaylistRepositoryTest
```

```
PHPUnit 9.6.16 by Sebastian Bergmann and contributors.

Testing
...                                                    3 / 3 (100%)

Time: 00:01.202, Memory: 26.00 MB

OK (3 tests, 3 assertions)

C:\wamp64\www\mediatekformation>
```

TESTS FONCTIONNELS :

Je commence par organiser mes tests fonctionnels en créant un répertoire nommé "Controller" à l'intérieur de mon répertoire "tests".

Contrôler que la page d'accueil est accessible.

```
<?php

namespace App\tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Component\HttpFoundation\Response;

class AccueilControllerTest extends WebTestCase
{
    public function testAccesPage(): void
    {
        $client = static::createClient();
        $client->request('GET', '/');
        $response = $client->getResponse();
        $this->assertSame(200, $response->getStatusCode());
    }
}
```

```
C:\wamp64\www\mediatekformation>php bin/phpunit --filter AccueilControllerTest
PHPUnit 9.6.16 by Sebastian Bergmann and contributors.

Testing
.                                                       1 / 1 (100%)

Time: 00:01.164, Memory: 30.00 MB

OK (1 test, 1 assertion)

C:\wamp64\www\mediatekformation>
```

Controller la page formations:

```
<?php

namespace App\tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Component\HttpFoundation\Response;

class FormationsControllerTest extends WebTestCase
{
    public function testAccessPage(): void
    {
        $client = static::createClient();
        $client->catchExceptions(false);

        $client->request('GET', '/formations');
        $response = $client->getResponse();

        $this->assertEquals(200, $response->getStatusCode());
    }

    public function testLinkFormations()
    {
        $client = static::createClient();
        $client->request('GET', '/formations');
        $crawler = $client->getCrawler();
        $link = $crawler->selectLink('image miniature')->link();
        $client->click($link);
        $this->assertResponseStatusCodeSame(Response::HTTP_OK);
        $suri = $client->getRequest()->getUri();
        $this->assertStringContainsString('/formations/formation/', $suri);
    }

    public function testFilterFormation(): void
    {
        $client = static::createClient();
        $client->request('GET', '/formations');
        $crawler = $client->submitForm('filtrer', [
            'recherche' => 'Eclipse n°3 : GitHub et Eclipse'
        ]);
    }
}
```

```

c:\wamp64\www\mediatekformation>php bin/phpunit --filter FormationsControllerTest
PHPUnit 9.6.18 by Sebastian Bergmann and contributors.

Testing
.....                                     5 / 5 (100%)

Time: 00:02.748, Memory: 40.00 MB

OK (5 tests, 15 assertions)

c:\wamp64\www\mediatekformation>

```

Contrôler la page playlists

```

<?php
namespace App\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Component\HttpFoundation\Response;

class PlaylistsControllerTest extends WebTestCase
{
    public function testAccessPage()
    {
        $client = static::createClient();
        $client->request('GET', '/playlists');
        $response = $client->getResponse();
        $this->assertEquals(Response::HTTP_OK, $response->getStatusCode());
    }

    public function testContentPage()
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/playlists');
        $this->assertSelectorTextContains('th', 'playlist');
        $this->assertCount(4, $crawler->filter('th'));
    }

    public function testTriPlaylists()
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/playlists/tri/name/ASC');

        // Vérifiez que la réponse est OK
        $this->assertEquals(Response::HTTP_OK, $client->getResponse()->getStatusCode());

        // Vérifiez la présence du tableau avec la classe "table-striped"
        $this->assertGreaterThan(
            0,
            $crawler->filter('table.table-striped')->count(),
            'La page ne contient pas de tableau avec la classe "table-striped"
        );

        // Vérifiez la présence des titres de playlist
        $this->assertGreaterThan(

```

```
c:\wamp64\www\mediatekformation>php bin/phpunit --filter PlaylistsControllerTest
PHPUnit 9.6.18 by Sebastian Bergmann and contributors.

Testing
.....                                                    7 / 7 (100%)

Time: 00:03.136, Memory: 38.00 MB

OK (7 tests, 24 assertions)

c:\wamp64\www\mediatekformation>
```

TESTS DE COMPATIBILITÉ :

Nommez votre nouveau projet

Veuillez fournir un nom pour votre nouveau projet.

NOM DU PROJET

Vous pouvez modifier le nom de votre projet à tout moment en cliquant dessus et en saisissant un nouveau nom.

D'ACCORD ANNULER

Je crée un scénario de test sur Sélénium avec chrome

Projet: TestMediatekformation*

Essais - +

Rechercher des tests...

http://192.168.1.59/mediatekformation/public/index.php

Commande	Cible	Valeur
1 ✓ ouvrir	http://192.168.1.59/mediatekformation/public/index.php	
2 ✓ définir la taille de la fenêtre	1456x928	
3 ✓ Cliquez sur	linkText=Formations	
4 ✓ Cliquez sur	lienText=Croissant	
5 ✓ Cliquez sur	linkText=Décroissant	
6 ✓ Cliquez sur	css=.text-left:nth-child(2) > .btn:nth-child(2)	
7 ✓ Cliquez sur	css=.text-left:nth-child(2) > .btn:nth-child(3)	
8 ✓ Cliquez sur	name=recherche	
9 ✓ Cliquez sur	name=recherche	
dix ✓ taper	name=recherche	c#
11 ✓ Cliquez sur	css=.text-left:nth-child(1) > .form-inline .btn	
12 ✓ Cliquez sur	css=.text-left:nth-child(2) .sm	
13 ✓ taper	css=.text-left:nth-child(2) .sm	Android

Commande

Cible

Valeur

Description

Enregistrer Référence

30	cliquez sur css=.text-center > .btn:nth-child(3) D'ACCORD	13:00:04
31	cliquez sur linkText=Voir détail D'ACCORD	13:00:06
32	cliquez sur linkText=Formations D'ACCORD	13:00:08
33	fermer OK	13:00:08

Je relance le scénario avec Firefox pour tester la comptabilité

Welcome to Selenium IDE! Version 3.17.4

What would you like to do?

- Record a new test in a new project
- Open an existing project
- Create a new project
- Close Selenium IDE

To learn more on Selenium IDE and I

Envoi du fichier

Téléchargements

Rechercher dans : Téléchargem...

Organiser Nouveau dossier

louiza : personnel

- Bureau
- Documents
- Images

Aujourd'hui

TestMediatekfor mation.side

Hier

Nom du fichier : Firefox Installer.exe

Tous les fichiers (*.*)

Ouvrir Annuler

Extension : Selenium IDE - Selenium IDE - TestMediatekformation - Mozilla Firefox

Project: TestMediatekformation

Tests - +

Search tests... Q

http://192.168.1.59/mediatekformation/public/index.php

Command	Target	Value
1 ✓ open	http://192.168.1.59/mediatekformation/public/index.php	
2 ✓ set window size	1456x928	
3 ✓ click	linkText=Formations	
4 ✓ click	linkText=Croissant	
5 ✓ click	linkText=Décroissant	
6 ✓ click	css=.text-left:nth-child(2) > .btn:nth-child(2)	
7 ✓ click	css=.text-left:nth-child(2) > .btn:nth-child(3)	
8 ✓ click	name=recherche	
9 ✓ click	name=recherche	
10 ✓ type	name=recherche	ca#
11 ✓ click	css=.text-left:nth-child(1) > .form-inline .btn	
12 ✓ click	css=.text-left:nth-child(2) .sm	
13 ✓ type	css=.text-left:nth-child(2) .sm	android
14 ✓ click	css=.text-left:nth-child(2) > .form-inline .btn	

Command #

Target

Value

Description

Log	Reference	
29. click on css=.text-center > .btn:nth-child(2) OK		13:10:12
30. click on css=.text-center > .btn:nth-child(3) OK		13:10:12
31. click on linkText=Voir détail OK		13:10:15
32. click on linkText=Formations OK		13:10:17
33. close OK		13:10:18
'test' completed successfully		13:10:18

TÂCHE 2: DOCUMENTATION TECHNIQUE

Tâche 2 : créer la documentation technique (1h)

- Contrôler que tous les commentaires normalisés nécessaires à la génération de la documentation technique ont été correctement insérés.
- Générer la documentation technique du site complet : front et back office excluant le code automatiquement généré par Symfony (voir l'article "Génération de la documentation technique sous NetBeans" dans le wiki du dépôt).

Pour générer la documentation, je télécharge le fichier phpDocumentor.phar, puis je le copie dans le dossier "ext" de mon installation PHP. Ensuite, j'ouvre une fenêtre d'invite de commandes en mode administrateur et j'insère la commande suivante.

```
C:\Windows\System32>"C:\wamp64\bin\php\php8.2.13\php.exe" "C:\wamp64\bin\php\php8.2.13\ext\phpDocumentor.phar" "run" "--ansi" "--directory" "C:/wamp64/www/mediatekformation/src" "--target" "C:\wamp64\www\mediatekformation_doc" "--title" "mediatekformation"
```

Espaces de noms

- Application
- Manette
- Entité
- Formulaire
- Dépôt
- Sécurité

Paquets

- Application

Rapports

- Obsolète
- les erreurs
- Marqueurs

Indices

- Des dossiers

Documentation

Table des matières

Paquets

[Application](#)

Espaces de noms

[Application](#)

Constantes

[CHEMIN DES FORMATIONS](#) = "pages/formations.html.twig"

[CHEMIN DE LISTE DE LECTURE](#) = "pages/playlists.html.twig"

Constantes

CHEMIN DES FORMATIONS

[FormationsController.php](#) : 11

```
public mixed FORMATIONSPATH = "pages/formations.html.twig"
```

CHEMIN DE LISTE DE LECTURE

[PlaylistsController.php](#) : 12

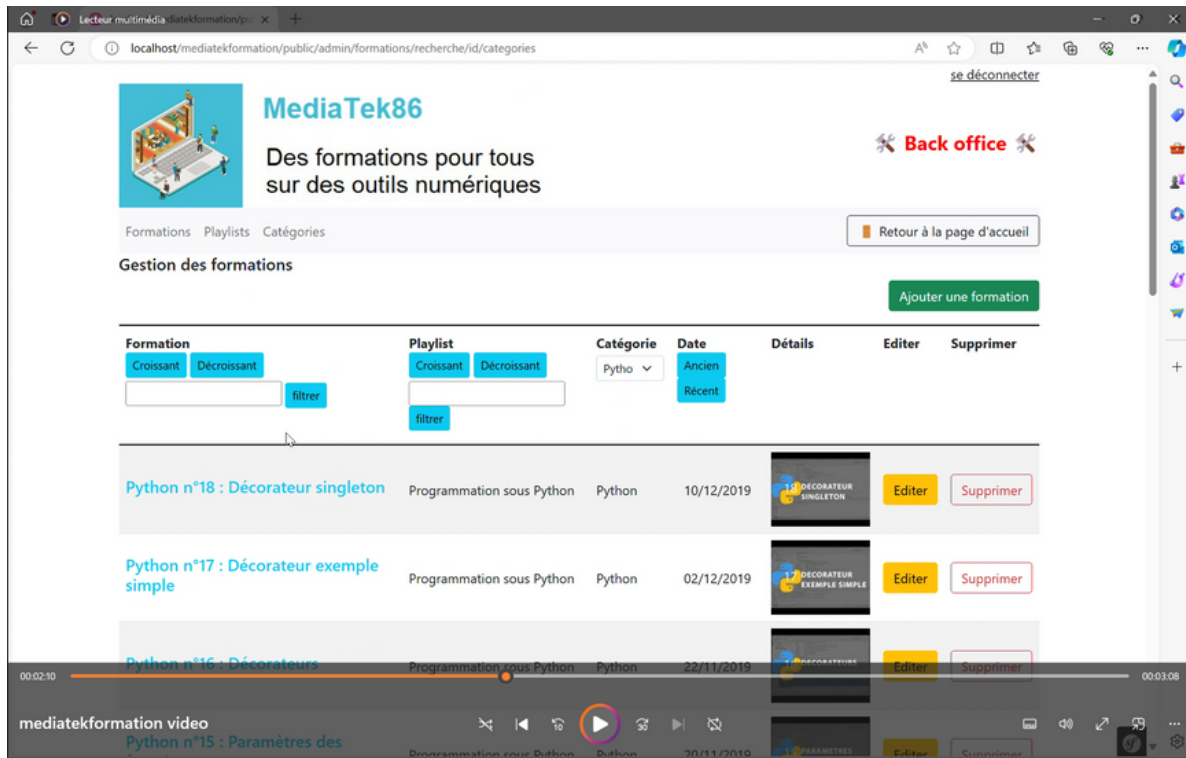
```
public mixed PLAYLISTSPATH = "pages/playlists.html.twig"
```

TÂCHE 3: DOCUMENTATION UTILISATEUR :

Tâche 3 : créer la documentation utilisateur (2h)

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 DEPLOIEMENT CONTINU

TÂCHE 1: DÉPLOYER LE SITE:

Tâche 1 : déployer le site (2h)

- 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.

Après m'être connecté à mon compte Azure, j'ai lancé la création d'une nouvelle machine virtuelle linux, que j'ai nommée vmKeycloak.

Une fois la VM prête, j'ai procédé à la configuration de son nom DNS pour faciliter l'accès. J'ai choisi vmkeycloak.francecentral.cloudapp.azure.com comme adresse, ce qui rendrait la connexion ultérieure à Keycloak beaucoup plus simple. Ensuite, j'ai configuré la VM pour autoriser le trafic sur le port 443, essentiel pour sécuriser les connexions à Keycloak via HTTPS.

L'étape suivante a été d'accéder à la VM en utilisant SSH, ce qui m'a permis d'entrer dans l'environnement Linux de la VM. J'ai utilisé Putty pour cela, une fois connecté, j'ai commencé à préparer le système pour Keycloak.

Le premier logiciel à installer était le JDK, version 18.0.1, crucial pour faire fonctionner Keycloak. J'ai exécuté une série de commandes pour mettre à jour le système, télécharger le JDK, extraire les fichiers, et configurer les variables d'environnement nécessaires. Ces étapes ont assuré que Java était correctement installé et prêt à l'emploi.

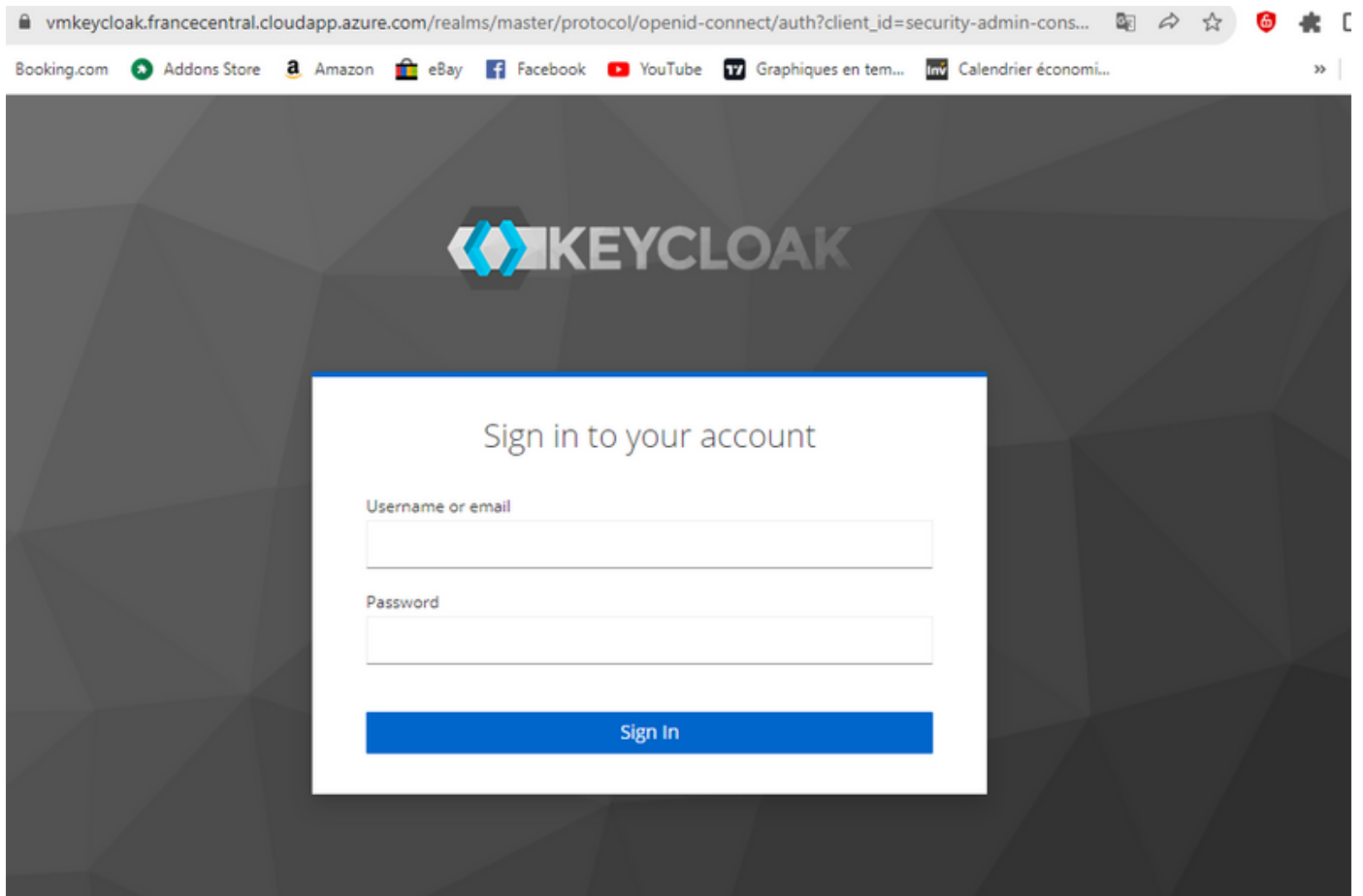
Avec Java en place, j'ai téléchargé et décompressé la dernière version de Keycloak, la 19.0.1, dans /usr. C'était assez simple, juste une question de téléchargement et d'extraction des fichiers.

Pour m'assurer que Keycloak serait accessible via le web, j'ai installé Apache. Après l'installation, j'ai démarré le service Apache et vérifié son statut pour m'assurer qu'il fonctionnait correctement. À ce stade, j'ai pu tester l'accès à la VM de l'extérieur en utilisant HTTP.

Sachant que Keycloak nécessite une connexion sécurisée, j'ai installé Certbot pour obtenir un certificat SSL gratuit de Let's Encrypt. Après avoir configuré le certificat avec Apache, j'ai pu accéder à la VM en utilisant HTTPS dans mon navigateur, confirmant que la configuration était correcte.

Avant de démarrer Keycloak, j'ai installé screen pour pouvoir exécuter Keycloak en arrière-plan sans interruption. Enfin, j'ai configuré et démarré Keycloak avec les bons paramètres, notamment en indiquant le nom de domaine et en utilisant les fichiers de certificat SSL. Une fois Keycloak lancé, j'ai vérifié que je pouvais y accéder via HTTPS en utilisant le nom DNS que j'avais configuré plus tôt.

Après m'être connecté à mon compte Azure, j'ai lancé la création d'une nouvelle machine virtuelle linux, que j'ai nommée vmKeycloak.



Dans l'application "mediatek-formation", il faut donner les bonnes valeurs aux 3 variables:

`KEYCLOAK_SECRET=BoTwymyKPOHv0gNtk7uT7ueupftmQLaI`

`KEYCLOAK_CLIENTID=mediatek`

`KEYCLOAK_APP_URL=https://vmkeycloak.francecentral.cloudapp.azure.com`

TÂCHE 2: DÉPLOYER LA BASE DE DONNÉES:

J'ai opté pour Hostinger pour héberger toute la partie en ligne de mon projet, y compris la base de données. Pour commencer, j'ai dû exporter la base de données locale de mon application. J'ai fait ça en sauvegardant tout dans un fichier SQL.

Après, je suis allé sur Hostinger et dans leur section "bases de données > phpMyAdmin", j'ai créé une nouvelle base de données, avec un utilisateur spécifique pour y accéder. J'ai configuré ma base avec les paramètres requis.

Une fois ma base prête, j'ai ouvert phpMyAdmin sur Hostinger, je me suis dirigé vers l'onglet "SQL", et là, j'ai collé le contenu de mon fichier SQL. J'ai cliqué sur "Exécuter" et toutes mes tables et données ont été importées. Ma base de données était désormais en ligne, avec toutes les tables correctement configurées, exactement comme elles étaient en local.

Base de données MySQL ⚡	Utilisateur MySQL ⚡
u575234902_media 1 Mo	u575234902_usermedi a

Ensuite, je lance Netbeans et je m'occupe de mettre à jour les informations de connexion à la base de données dans le fichier ".env".

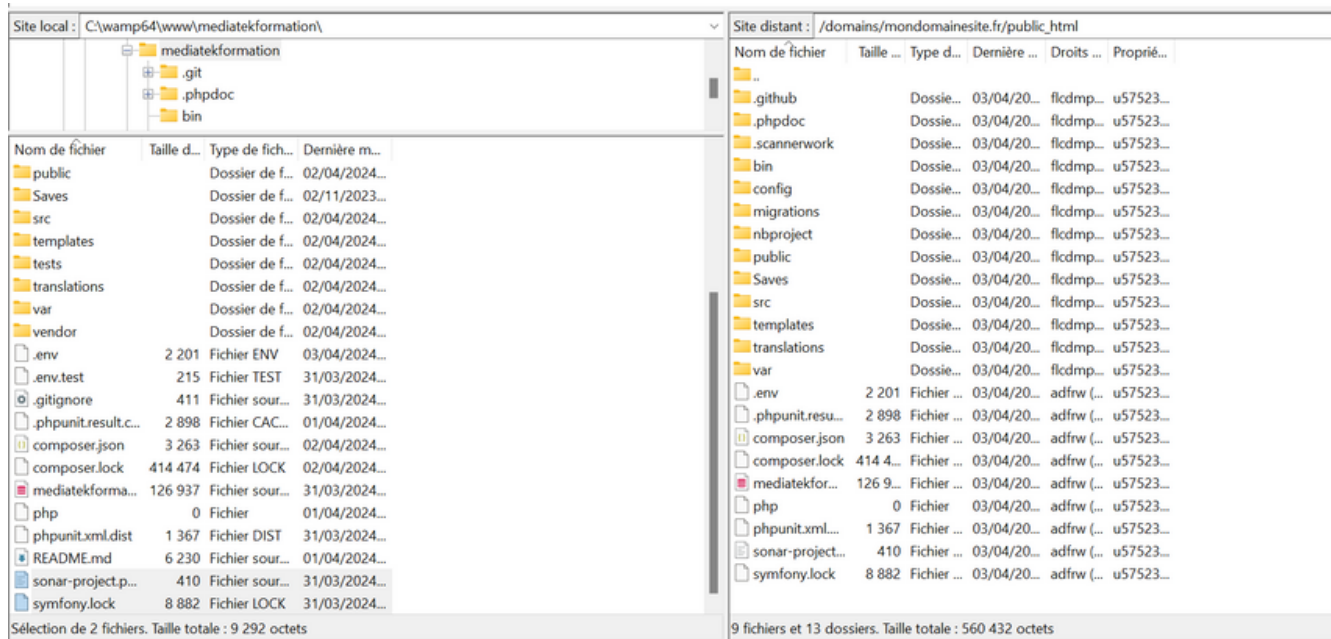
```

31 # DATABASE_URL="sqlite:///kernel.project_dir%/var/data.db"
32 DATABASE_URL="mysql://u575234902_usermedia:Adminuser4004@localhost:3306/u575234902_media?serverVersion=8&charset=utf8mb4"
33
34 # DATABASE_URL="postgresql://app:!ChangeMe!@127.0.0.1:5432/app?serverVersion=14&charset=utf8"
35 ###< doctrine/doctrine-bundle ###
36
37 ###> symfony/messenger ###
38 # Choisissez un des transports ci-dessous
39 # MESSENGER_TRANSPORT_DSN=doctrine://default
40 # MESSENGER_TRANSPORT_DSN=amqp://guest:guest@localhost:5672/%2f/messages
41 # MESSENGER_TRANSPORT_DSN=redis://localhost:6379/messages
42 ###< symfony/messenger ###
43
44 ###> symfony/mailer ###
45 MAILER_DSN=smtplib://contact@mondomainesite.fr:Azure48@louiza@smtp.hostinger.com:465
46 ###< symfony/mailer ###

```

TÂCHE 3: DÉPLOYER LE SITE:

Pour déployer mon site, j'ai utilisé FileZilla, un logiciel libre qui facilite la connexion FTP. Après avoir saisi les informations de mon FTP, je me suis connecté au serveur distant. Ensuite, j'ai navigué jusqu'au dossier "domains" > "mondomainesite.fr" > "public_html". J'ai transféré mon projet sur le serveur distant en veillant à exclure les dossiers "tests" et "vendor", ainsi que le dossier ".git" et le fichier ".gitignore".



Ensuite, j'ai mis à jour mon PHP en passant à la version 8.2.

Pour reconstruire le dossier "vendor", j'ai dû me connecter à Hostinger en SSH. Après m'être connecté au terminal avec une connexion SSH.

Composer est maintenant à jour, et j'ai utilisé la commande suivante pour télécharger ce qui manquait :

```
php composer.phar require symfony/apache-pack
```

Ensuite, pour mettre Symfony en production, j'ai modifié le fichier ".env" pour définir le mode de production ("prod") et j'ai également mis mon serveur en production en définissant "prod" pour "APP_ENV". Après cela, je suis retourné sur SSH et j'ai exécuté la commande suivante :

```
php composer.phar install --no-dev --optimize-autoloader APP_ENV=prod APP_DEBUG=0 php bin/console cache:clear
```

J'ai rencontré une erreur 404 en essayant d'accéder à la page admin de mon site. Pour résoudre ce problème, j'ai ajouté un fichier ".htaccess" à la racine de mon projet, dans le dossier "public_html". Ensuite, j'ai ajouté un deuxième fichier ".htaccess" dans le dossier "public" pour indiquer à Apache d'afficher mon fichier "index.php".

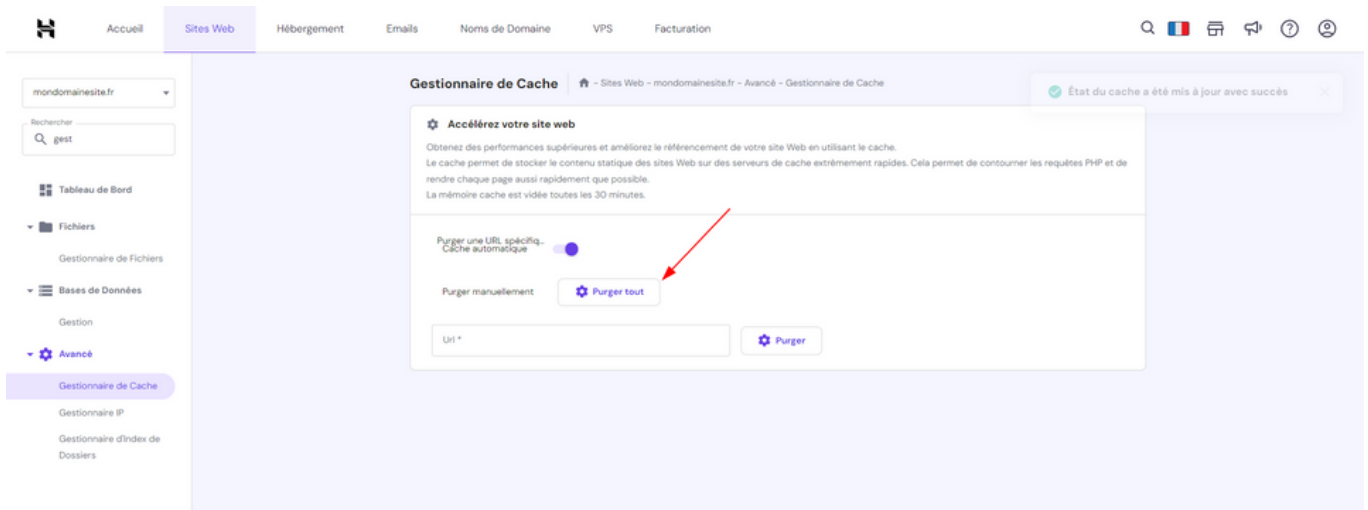
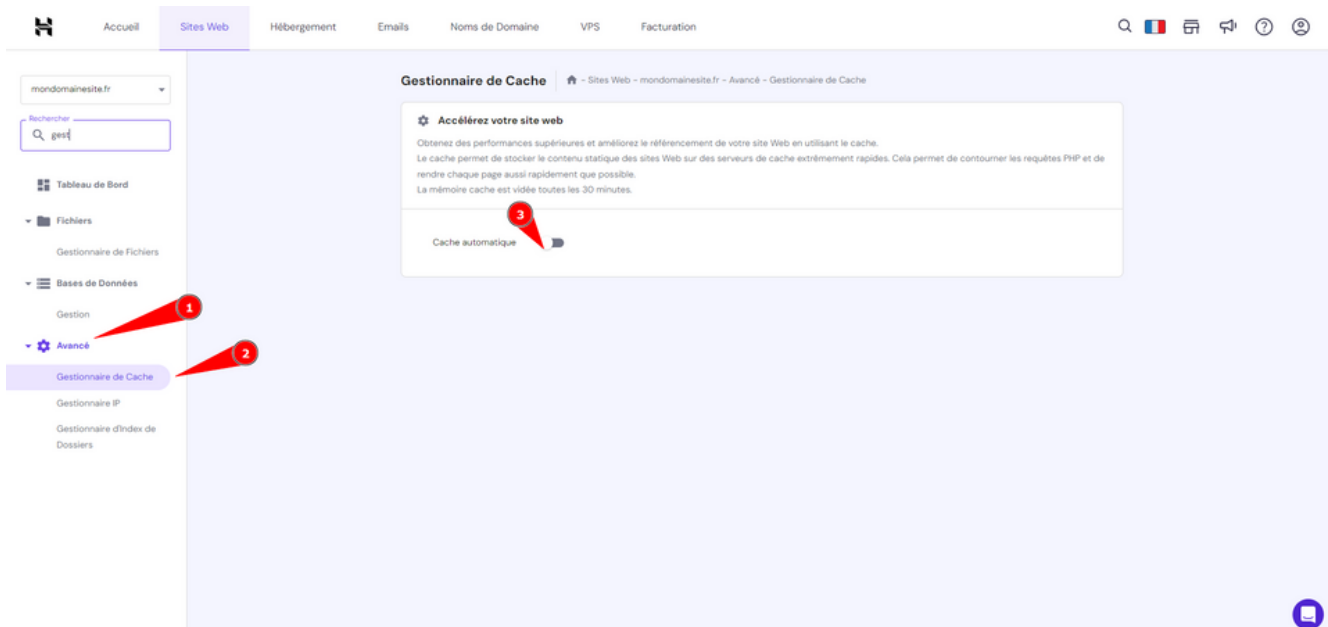
Après ces étapes, mon application était en ligne. Pour résoudre l'erreur 404 lors de l'accès à la page admin, j'ai vidé le cache à partir du panneau de contrôle Hostinger

Graphiques en tem... Calendrier économi... Calendrier économi... Sentiment Forex | M... Corrélation Forex -... Calculateur de lot j... Calendrier économi... Currency Strength C... » | Tous les favoris

Oops! Une erreur s'est produite

Le serveur a renvoyé un « 404 Not Found ».

Quelque chose est cassé. Veuillez nous faire savoir ce que vous faisiez lorsque cette erreur s'est produite. Nous le réparerons dès que possible. Désolé pour tout inconvénient causé.



TÂCHE 4: GÉRER LA SAUVEGARDE ET LA RESTAURATION DE LA BDD:

Pour mettre en place une sauvegarde automatique quotidienne de ma base de données, je commence par accéder au gestionnaire de fichiers de mon compte utilisateur FTP sur Hostinger. Là, je crée un dossier nommé "savebdd" à la racine de mes répertoires. À l'intérieur de ce dossier, je crée un script nommé "backup.sh". Ce script contient toutes les instructions nécessaires pour extraire la base de données et la sauvegarder dans un autre dossier nommé "bddbackup_", suivi de la date du jour, avec l'extension ".sql.gz".



backup.sh



bddbbackup_2024-01-01.sql.gz

Ensuite, je me rends dans l'onglet "Tâches Cron" de mon hébergeur Hostinger. Une fois sur cette page, j'ai la possibilité d'ajouter une commande à exécuter. Sous cette section, je trouve des paramètres qui me permettent de choisir la fréquence d'exécution de la tâche, comme "sauvegarde quotidienne", "sauvegardes toutes les minutes" ou "toutes les heures", etc.

Pour ce projet spécifique, j'ajoute la commande suivante :
"/home/u575234902/savebdd/backup.sh". Cette commande sera exécutée lors de la tâche cron.

Ensuite, je sélectionne l'option "sauvegarde quotidienne" et, en bas de la page, je peux vérifier que ma tâche cron a bien été créée. Cette sauvegarde sera enregistrée dans un nouveau fichier, dans le dossier "savebdd" que j'ai précédemment créé dans le gestionnaire de fichiers, au même niveau que "public_html".

⚙ Liste des tâches Cron

Heure ⇅

Commande à exécuter ⇅

Actions

/home/u575234902/savebdd/backup.sh

☰ Afficher les résultats



TÂCHE 4: METTRE EN PLACE LE DÉPLOIEMENT CONTINU

Tâche 3 : mettre en place le déploiement continu (1h) Configurer le dépôt Github pour que le site en ligne soit mis à jour à chaque push reçu dans le dépôt.

Pour déployer l'application sur GitHub, plusieurs étapes sont nécessaires. L'objectif est d'automatiser la mise à jour du site en ligne à chaque push vers GitHub. Pour cela, nous devons configurer GitHub afin de lui indiquer vers quel serveur FTP envoyer les fichiers. Pour commencer, je dois créer un fichier YAML d'automatisation. Pour ce faire, je me rends dans le dépôt GitHub de mon projet, puis je sélectionne l'onglet "Actions". Ensuite, je clique sur le lien "Set up a workflow yourself" et je supprime le contenu de "Edit new file". Je le remplace ensuite par le contenu approprié pour spécifier le déploiement vers le serveur FTP. Une fois le fichier YAML créé et sauvegardé, GitHub sera configuré pour effectuer le déploiement automatique à chaque push vers la branche principale.

```
1   on: push
2   name: Deploy website on push
3   jobs:
4     web-deploy:
5       name: Deploy
6       runs-on: ubuntu-latest
7       steps:
8         - name: Get latest code
9           uses: actions/checkout@v2
10
11        - name: Sync files
12          uses: SamKirkland/FTP-Deploy-Action@4.3.0
13          with:
14            server:
15              server-dir: /public_html/
16              username: u575234902
17              password: ${ secrets.ftp_password }
```

Après avoir cliqué sur "start" puis sur "commit new file", le dossier ".github/workflows" est ajouté et contient le fichier YAML de déploiement nouvellement créé. Ensuite, dans Netbeans, je vais dans l'onglet "Git", puis je sélectionne "Remote" > "Pull" > "Next" > "Finish" pour récupérer le dossier localement. Pour sécuriser l'accès FTP, je sauvegarde le mot de passe dans le dépôt GitHub. Pour cela, je vais dans "Settings" > "Secrets" > "Actions" > "New repository secret", à la racine du dépôt. Je remplis les champs "name" et "value", puis je clique sur "Add secret" pour enregistrer le mot de passe.

Secrets and variables

Actions

Codespaces

Dependabot

Name ↕↑

Last updated

FTP_PASSWORD

35 minutes ago



Annotations

1 error

- Invalid workflow file: .github/workflows/main.yml#L4
You have an error in your yaml syntax on line 4

Annotations

1 error and 2 warnings

- Deploy
Error: getaddrinfo ENOTFOUND ftp://mondomaine.fr (control socket)

Maintenant, je peux utiliser le déploiement continu. Pour cela, je modifie la couleur du bouton d'accès à la page admin dans Netbeans, puis je fais un commit suivi d'un push. Ensuite, je vérifie dans l'onglet "Actions" de mon dépôt GitHub pour voir si tout s'est bien déroulé. Au début, j'ai rencontré une erreur d'adresse, alors j'ai corrigé cela en remplaçant l'adresse par l'adresse IP du serveur FTP.

← Deploy website on push

✔ Update main.yml #4

Summary

Jobs

- ✔ Deploy

Run details

- Usage
- Workflow file

Deploy Beta Give feedback 🔍

succeeded 14 minutes ago in 2m 56s

- > ✔ Set up job
- > ✔ Get latest code
- > ✔ Sync files
- > ✔ Post Get latest code
- > ✔ Complete job

BILAN

Lors de la réalisation de ce projet, j'ai atteint les objectifs liés au nettoyage et à l'optimisation du code, ainsi qu'à l'ajout de plusieurs fonctionnalités. Cependant, j'ai rencontré des difficultés au début du projet pour comprendre la tâche 2 et la tâche 3 de la première mission. En effet, le code avait déjà été créé par un autre développeur, ce qui m'a quelque peu désorienté lors de sa relecture. De plus, j'ai rencontré une importante difficulté liée à une erreur lors de l'authentification avec Keycloak : "Class "App\Entity\User" is not a valid entity or mapped superclass." Cela m'a pris trois jours pour en comprendre l'origine. Pour corriger cela, j'ai modifié la version de PHP (passant à PHP 7.4.33), restauré une sauvegarde vierge du code de la base de données (réalisée avant toute modification de Keycloak), puis supprimé le dossier "vendor" ainsi que le fichier "composer.lock".

Ensuite, dans l'invite de commande à la racine du projet, j'ai exécuté les commandes "composer update" et "composer install". J'ai ensuite suivi toutes les étapes de configuration avec Keycloak, y compris la création d'utilisateurs et les migrations. Tout s'est déroulé sans encombre.

J'ai également mis à jour la version de Keycloak (passant à la version 19.0.1). Ce projet m'a permis d'apprendre à configurer une machine virtuelle Keycloak pour permettre l'accès au serveur d'authentification en HTTPS, ce qui m'a ensuite permis de déployer le projet en ligne avec une authentification sécurisée.

Dans l'ensemble, je suis satisfaite du projet que j'ai réalisé.