## ATELIER PROFESSIONNEL

# MEDIATEKFORMATION

COMPTE -RENDU


Réalisé par :

SOUMIA B.AZZA

BTS SIO-SLAM


## C O N T E X T E

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

Afin de donner plus d'attractivité aux médiathèques, MediaTek86 veut se développer en enrichissant ses services et en offrant aux adhérents des formations aux outils numériques, et des autoformations en ligne.

## MISSION

Un autre développeur a déjà créé l'application qui va présenter le site "mediatek formation" contenant les formations disponibles aux adhérents. Néanmoins, le chef de projet a constaté quelques manquements aux bonnes pratiques de codage. Il a également constaté l'oubli d'une fonctionnalité attendue dans le cahier des charges.

Dans un premier temps, il m'a chargé de corriger ces problèmes, puis plusieurs autres missions m'ont été confiées, en particulier le "back-office", pour finaliser et déployer le site.

## LANGAGE ET OUTILS

- L. programmation: PHP , TWIG.
- Serveur: WampServer: APACHE, PHP, MYSQL.
- Authentification: KEYCLOAK.
- FrameWork: SYMFONY.
- Versioning: GITHUB.
- IDE: NETBEANS.
- Hébergeur: HOSTINGER.
- VM: WINDOWS 10.

# SOMMAIRE

# MISSION 1: NETTOYER LE CODE EXISTANT ET AJOUTER UNE FONCTIONNALITÉ

## TÂCHE 1: NETTOYER LE CODE

La première tâche consiste à nettoyer le code en suivant les indications de Sonarlint. Précisément les fichiers créés par le développeur, et donc de trier les "Action items" de Sonarlint par "Location" et s'arrêter au premier fichier dans "vendor".

Le temps estimé pour réaliser cette tâche était de 2h, le temps que j'y ai passé a été de 3h.


Afin d'obtenir les erreurs à corriger, j'ai réalisé une "Analyze with Sonarlint" dans chacun des dossiers du projet contenant les fichiers que le développeur a initialement créés.

Une fois les erreurs obtenues, j'ai pu effectuer les corrections qui s'imposent, selon les intitulés des erreurs trouvées par Sonarlint.

### *Erreurs “STRING LITERALS SHOULD NOT BE DUPLICATED”*

L'intitulé des erreurs "String literals should not be duplicated", signifie que l'on peut éviter les redondances de certaines chaînes de caractères dans le code. Pour cela, il suffit d'affecter les chaînes de caractères qui apparaissent plusieurs fois dans le code, dans des constantes.

Les constantes seront ensuite utilisées pour remplacer les chaînes de caractères qui y sont

affectées, par le nom défini à chacune d'elles.

"FormationsController" (dossier "Controller")

### *Erreurs “CONSTANT NAMES SHOULD COMPLY WITH A NAMING CONVENTION”*

La constante "cheminImage" n'a pas été écrite correctement, en php les constantes doivent toujours s'écrire en lettres majuscules.

### *Erreurs “COLLAPSIBLE”“IF STATEMENTS SHOULD BE MERGED””*

Les "if" ont été imbriqués inutilement. L'intitulé de l'erreur "Collapsible "If statements should be merged"" signifie que les "If" présents dans la fonction doivent être fusionnés dans un seul "If".

Erreur "Collapsible "If statements should be merged"" - Fichier "Playlist" (dossier "Entity")

### *Erreurs “<STRONG>AND<EM> TAGS SHOULD BE USED”*

Les balises <i> ont été mises à la place des balises . On remplace les balises <i> par les balises . L'intitulé de l'erreur " and  tags should be used" signifie que les balises  et  doivent être utilisées à la place des balises déjà en place.

Première erreur " and  tags should be used" - Fichier "cgu" (dossier "templates > pages")

### *Erreurs “IMAGES SHOULD HAVE AN “ALT” ATTRIBUTES”*

La balise <img> ne contient pas de description "alt". On ajoute "alt" avec une description, dans la balise <img>. L'intitulé de l'erreur "image tags should have an "alt" attributes" signifie que les balises <img> doivent également contenir un "alt" qui va permettre d'ajouter une description à l'image.

Première erreur "image tags should have an "alt" attributes" - Fichier "accueil" (dossier "templates > pages"

### *Erreurs “<TABLE>TAGS SHOULD HAVE A DESCRIPTION”*

La balise <table> n'est pas succédée par une balise <caption> contenant la description. On ajoute une balise <caption> avec une description, après la balise <table>. L'intitulé de l'erreur "<table> tags should have a description" signifie que les balises <table> doivent être succédées par des balises <caption> qui vont permettre d'ajouter une description à la table.

Première erreur "<table> tags should have a description" - Fichier "accueil" (dossier "templates > pages")

# MISSION 1: NETTOYER LE CODE EXISTANT ET AJOUTER UNE FONCTIONNALITÉ

## TÂCHE 1: RESPECTER LES BONNES PRATIQUES DE CODAGE

Dans le respect des bonnes pratiques de codage, en particulier SOLID (ici, le S : "Single responsability"), je dois modifier les méthodes de FormationRepository et PlaylistRepository qui contiennent des tests sur  $table. A chaque fois, il faudra créer 2 méthodes plutôt qu'une, pour éviter ce test. Le reste du code de l'application doit être adapté pour exploiter ces nouvelles méthodes.

Le temps estimé pour réaliser cette tâche était de 2h, le temps que j'y ai passé a été 4h.


### *Modification des méthodes de "FORMATIONREPOSITORY"*

Le but est d'enlever le test de cette méthode, pour le réaliser dans une nouvelle méthode.

La première méthode à modifier dans le "FormationRepository", est la méthode "findAllOrderBy", car elle contient un test sur le paramètre $table.

J'ai réalisé une première méthode "findAllOrderBy" qui récupère tous les enregistrements de la table "formation", et qui va les trier selon le champ et l'ordre, spécifiés dans la page "formations.html.twig". Une fois le champ et l'ordre passés en paramètres de la fonction findAllOrderBy($champ, $ordre) , je conserve le premier "return" et son contenu, car il permet de faire la requête qui va trier les enregistrements de la table "formation", à partir du champ, et dans l'ordre choisi, grâce à la requête "orderBy('f.' .$champ, $ordre)".

La seconde méthode "findAllOrderByTable", permettant de réaliser le test sur  $table, si elle est spécifiée dans la page "formations.html.twig". Pour cela, je dois passer en paramètres ($champ, $ordre, $table="") pour récupérer tous les enregistrements. La requête "->join('f.'.$table, 't')" permet de récupérer la table spécifiée, et la requête "->orderBy('t.'.$champ, $ordre)" va servir à

effectuer le tri sur un champ et selon l'ordre "ascendant ou descendant", si le champ est dans une autre table.

La seconde méthode à modifier dans FormationRepository est la méthode "findByContainValue", car elle contient un test sur $table.

Afin de séparer le test sur $table, de la méthode, je dois supprimer le paramètre $table, de la fonction "findByContainValue", pour traiter le test dans une seconde méthode appelée "findByContainValueTable". Une fois le paramètre supprimé, je conserve la condition "if($valeur=="") et son contenu, car elle permet de retourner tous les enregistrements si la valeur est vide.En gardant le premier "return" dans cette fonction, car il permet de récupérer tous les enregistrements de la table "formation", où la valeur d'un champ est égale à la variable $valeur.

Avec ce tri, lorsque l'on va rechercher une formation avec le filtre, on aura la liste de seulement des formations qui ont la même chaîne de caractère que celle saisie dans la recherche.

Après la seconde méthode, permettant de réaliser le test sur $table. Si la valeur recherchée est une chaîne vide, la fonction renvoie tous les enregistrements de la table. La même condition "if" dans la fonction "findByContainValueTable", après avoir ajouté le troisième paramètre "$table" . La requête effectue ensuite une jointure de la $table si une table a été déclarée dans la page "formations.html.twig", et recherche si le champ contient la valeur dans la table spécifiée.

### *ADAPTATION DU CODE*

L'erreur de tri ascendant et descendant, provient de la fonction "formations.sort" du fichier "FormationsController", car il s'agit de la méthode qui est appelée dans la page "formations.html.twig" lors de l'utilisation des deux boutons "ascendant" et "descendant".

Si la variable $table a été assignée à une table, la fonction va appeler la méthode "findAllOrderByTable", et effectuer le tri de la table, par le $champ, dans l'ordre défini à la page "formations.html.twig".

Si la variable $table n'a pas été assignée, la fonction va appeler la méthode "findAllOrderBy", qui va retourner toutes les formations triées sur un champ, dans l'ordre défini. Je place donc un "else" pour réaliser l'appel de cette fonction.

Là , l'appel de la méthode "findByContainValue" est incorrecte, puisqu'avec les modifications, elle ne comporte plus que les paramètres $champ et $valeur. De plus, l'appel de la seconde méthode en cas d'assignation d'une table n'est pas réalisé dans cette fonction.

Je dois donc procéder de la même manière que précédemment et ajouter une condition, permettant d'obtenir le contenu qui aura été saisi et de le comparer avec la $table si elle a été assignée, grâce à l'appel de la fonction "findByContainValueTable($champ, $valeur, $table").

Si la table n'a pas été assignée, la fonction findByContainValue($champ, $valeur) sera appelée.

*Modification des méthodes de "PLAYLISTREPOSITORY"*

Comme pour le FormationRepository, on doit la séparer en deux méthodes.

La première méthode à réaliser, ne doit pas contenir le paramètre $table, et doit renvoyer tous les enregistrements, contenant la valeur assignée à la variable $valeur, selon un $champ défini et dans l'ordre "ascendant".

On reprend donc la fonction "findByContainValue" et on supprime le paramètre "$table=" " "car nous ferons le test $table dans la seconde méthode. On garde la condition "if" permettant de retourner la liste des enregistrements si la valeur est vide, triés par nom et dans l'ordre ascendant. Puis on récupère le premier return, qu'on ajoute à la première méthode après le "if".

La seconde méthode "findByContainValueTable" contient le paramètre $table, qui va permettre de réaliser les enregistrements lors du filtrage de recherche sur une autre table.( j'avais des probleme a ce niveau la du code et j'ai ajouter un test!).

Les corrections dans le "PlaylistsController", pour exploiter les deux nouvelles méthodes de "PlaylistRepository".

La fonction à corriger est la fonction "findAllContain" du PlaylistController, car c'est celle qui est appelée lors du filtre des playlists et des catégories de la page "playlists.html.twig".

On rajoute la condition "if" , qui va permettre d'appeler la méthode "findByContainValue" afin de gérer le test sur $table. Puis dans le else, on appelle la méthode "findByContainValueTable".

# MISSION 1: NETTOYER LE CODE EXISTANT ET AJOUTER UNE FONCTIONNALITÉ

## TÂCHE 3 : AJOUTER UNE FONCTIONNALITÉ

Dans cette tâche il faut ajouter une fonctionnalité dans la page des playlists, qui consiste à ajouter une colonne qui affiche le nombre de formations sur une playlist et aussi qui permettre le triage d'une façon croissante ou décroissante, et aussi cette information faut qu'elle soit afficher dans la page d'une playlist, j'ai fait cette tâche en suivant le guide dans le document présenté sur la zone de cette tâche.

Le temps estimé était de 2h, le temps mis a été 3h.


### *PLAYLIST-REPOSITORY "FindAllOrderByNbFormations"*

Afin de récupérer les données nécessaires dans la base de données, je crée la fonction "findAllOrderByNbFormations", en lui affectant les requêtes sql nécessaires.

J'ajoute une commande "->leftjoin(FORMATION, 'f')" dans ma requête CreateQueryBuilder, qui va permettre de récupérer les formations. Il faut ensuite regrouper les formations par la clé primaire de la table des playlists, grâce à la commande ->groupBy('p.id').

Les résultats sont ensuite classés selon le nombre de titres dans la table des formations, et l'ordre est déterminé par la valeur passée au paramètre $ordre, dans la page "playlists.html.twig".

Pour réaliser cela j'insère la commande "->orderBy('count(p.name)', $ordre)".

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

*Affichage de “nombre de formations”*

Dans la vue, je dois appeler ma méthode "playlists.sort" pour gérer le tri ascendant et descendant de la colonne du nombre de formations par playlists.

J'ajoute mes deux boutons "<" et ">" , ainsi que mon chemin vers la méthode "playlists.sort" dans la page playlists.html.twig.

J’ajoute le nombre de formations dans chacune des playlists individuelles,donc dans la page "playlist.html.twig", j'ajoute la commande "playlistformations|length" ; qui permet d'afficher le nombre. Je mets un titre "nombre de formations".

Dans la page "playlists.html.twig", je modifie le code initial de la boucle "for" qui doit permettre d'afficher le tableau avec le titre des playlists, la catégorie et le nombre de formations associées à chaque playlist.

Les colonnes sont affichées grâce aux balises <td>.

La première colonne affiche donc le nom de la playlist grâce à "{{ playlists[k].name }}".

La seconde affiche les catégories associées à la playlist (le "if" permet de ne pas rechercher une catégorie qui n'existe pas).

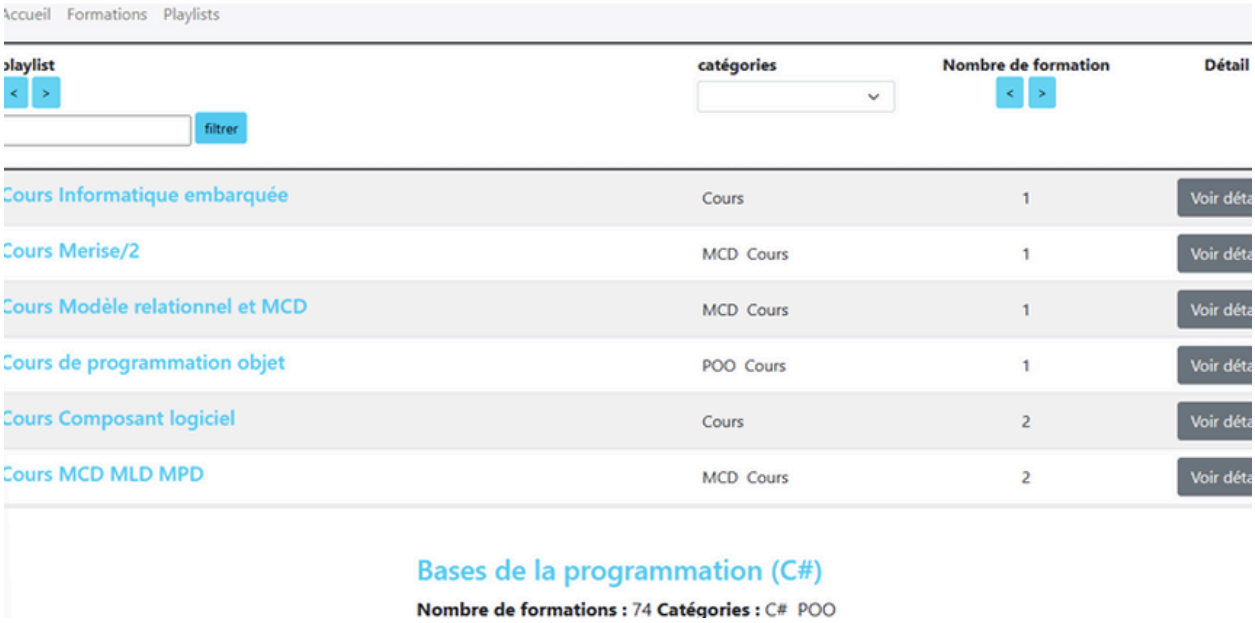
La colonne suivante affiche le nombre de formations par playlists, grâce à la commande {{ playlists[k].formations|length }} .

La dernière colonne affiche le bouton "voir détail", qui permet de voir le détail d'une playlist

Dans la page "playlist.html.twig", je dois également ajouter le nombre de formations. Pour cela, j'insère la commande {{ playlists[k].formations|length }}.

Je vérifie que le code fonctionne, en exécutant l'application sur la page "playlists". Le nombre de formations doit apparaître avec les boutons "<" et ">", et le clic sur les boutons doit permettre de trier par ordre croissant ou décroissant.

Lors du clic sur le bouton "Voir détail", le nombre de formations d'une playlist doit s'afficher dans la page de la playlist.




## MISSION 2: CODER LA PARTIE BACK-OFFICE

## TÂCHE 1: GÉRER LES FORMATIONS

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

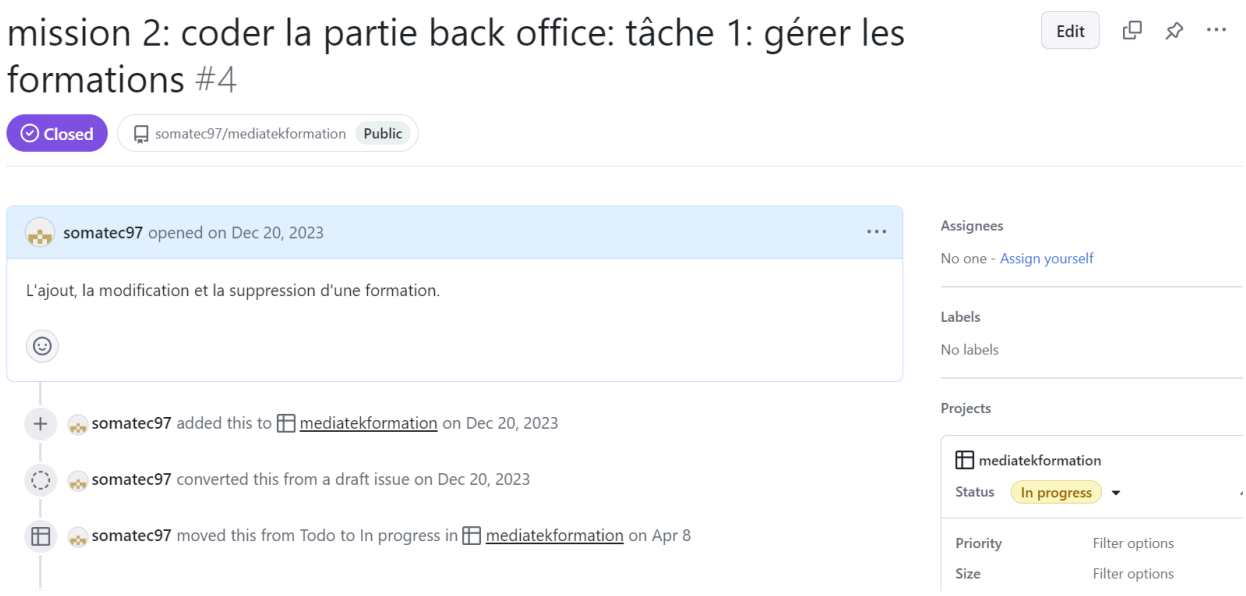
Si une formation est supprimée, il faut aussi l'enlever de la playlist où elle se trouvait.

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

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

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

Le temps estimé pour cette tâche était de 5h, le temps passé était d'environ 9h.



J commence par créer le dossier "admin" dans le dossier "controller", avec un fichier "AdminFormationsController", qui va me permettre de réaliser les routes vers la page "admin.formations.html.twig".Aussi créer le dossier "admin" dans le dossier "templates". Je crée ensuite une page "admin.formations.html.twig", qui va permettre de gérer les formations.

Puis créer un fichier "baseadmin.html.twig" dans le dossier "templates" , qui contiendra la structure des pages d'administrations.

### *Création de la page "Gestion de formations"*

Dans "baseadmin" , j'ajoute le chemin vers la page d'administration des formations dans la barre de navigation, et dans le fichier "basefront" j'ajoute un lien qui va rediriger vers la page de gestion des formations.

```
{% block top %}
    <div class="container">
        <!-- titre -->
        <div class="text-left">
            <img src="{{ app.request.getBasePath()~'/banniere.jpg' }}" alt="Image de la bannière">
        </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>
            </div>
        </nav>
    </div>
{% endblock %}
```

Je réalise le "squelette" de la page d'administration des formations, qui va permettre d'afficher les formations, leurs playlists et catégories, puis la possibilité d'ajout, modification ou la suppression de formation.

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

{% block body %}
    <table class="table table-striped">
        <caption></caption>
        <thead>
            <tr>
                <th class="text-left align-top" scope="col">
                    formation<br />
                </th>
            </tr>
        </thead>
    </table>
{% endblock %}
```

Dans la classe "AdminFormationsController", du dossier "controller", je créé les variables, le constructeur , et l'index contenant la route vers la nouvelle page admin.

## 2. AFFICHAGE DE LA LISTE DES FORMATIONS

Pour afficher la liste des formations, je vais dans la page "admin.formations.html.twig" , et je réalise une boucle "for", permettant de récupérer le titre des formations, et de les placer dans une première colonne.

J'affiche également les playlists, les catégories, la date de création, et l'image avec le détail de chaque formation, dans les autres colonnes.

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

### *Suppression d'une formation*

Maintenant que l'affichage des formations est fonctionnel dans la page de gestion, je vais créer les boutons de suppression et d'édition d'une formation.

Je vais créer une fonction dans "AdminFormationsController", qui récupère la méthode "remove" du "FormationsRepository", puis faire une redirection vers la route de la page "admin.formations.html.twig"

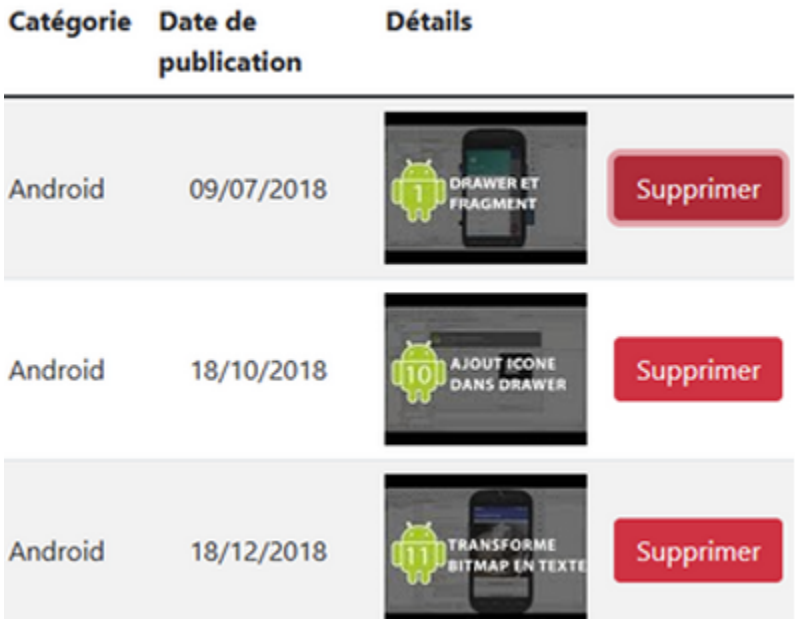
Je crée ensuite le bouton "supprimer" , dans la page "admin.formations.html.twig". Je dois pour cela, placer ma commande à la fin de ma boucle "for", pour ajouter la colonne qui affichera les boutons "supprimer" sur chacune des formations.

Il faut que je précise la route de ma fonction "admin.formation.suppr", créée dans AdminFormationsController, et qui permet la suppression des formations dans la base de données.

Je rajoute également un "onclick" permettant de demander la confirmation de suppression, lors du clic sur le bouton "supprimer".

Je vérifie que lebouton "supprimer" fonctionne, en exécutant l'application, et en affichant la page d'administration des formations.

Le bouton doit s'afficher, le clic dessus doit demander une confirmation, puis une fois la confirmation effectuée, la formation doit être supprimée.




*Edition d'une formation*

Pour modifier une formation, il faut créer un nouveau bouton "éditer", qui, lorsque l'on va cliquer dessus, va permettre d'accéder à un formulaire d'édition de la formation.

Pour créer ce formulaire, je dois réaliser une nouvelle page " _admin.formation.form.html.twig" dans la racine du dossier "templates", qui va contenir le formulaire d'édition.

Le formulaire que je vais exploiter dans d'autres pages. Je construis donc une page spéciale "admin.formation.edit.html.twig", dans mon dossier "templates/admin", qui va me permettre d'appeler la page du formulaire "_admin.formation.form.html.twig", lors du clic sur le bouton "éditer".

Dans "admin.formation.edit.html.twig", j'ajoute l'include qui va me permettre d'appeler le formulaire situé dans la page "_admin.formation.form.html.twig".

Pour la création du formulaire dans "_admin.formations.form.html.twig", il faut créer un nouveau dossier "Form", dans le "src", et que je conçoive une nouvelle classe php "FormationType". Cette classe va me permettre de 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, 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 formation.

Il doit également 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 va permettre de soumettre le formulaire, et d'enregistrer les informations dans la base de données.

Pour que builder puisse récupérer le champ avec le titre de la formation sélectionnée, et sa description, je dois ajouter au builder, un "->add title" de type "Text", pour ajouter le titre de la formation, on lui ajoute un "label => Formation", qui va afficher le texte "Formation" à côté du champ qui va contenir le titre de la formation. On précise "'required' => true" car il est obligatoire pour une formation d'avoir un titre.

Ensuite on ajoute un "->add description" de type "TextArea", avec un "'required' => false, la description n'est pas obligatoire (peut être vide), on lui donne également un "label =>Description"

Le formulaire doit également contenir le champ "playlist", permettant d'afficher une liste déroulante des playlists, et qui sélectionne la playlist de la formation pour laquelle on aura

cliqué sur le bouton d'édition. Il faut pouvoir sélectionner une nouvelle playlist dans cette liste si on veut 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, il va falloir ajouter la classe "Playlist" au builder.

La commande permettant de réaliser cela est la suivante "->add('playlist', EntityType::class" .On ajoute ensuite la classe "Playlist", on lui précise un label "playlist", puis pour réaliser la liste des choix de playlist possible, on ajoute un "choice_label => name" , afin de récupérer les noms des playlists. On ne doit pouvoir sélectionner qu'une seule playlist par formation, donc on fait un'"multiple' => false", puis comme il est obligatoire d'avoir une playlist on passe le "required" en "true".

Le formulaire doit aussi permettre l'ajout ou l'édition d'une catégorie, j'ajoute donc un "->add('categories)" de type Entity, car il s'agit là encore d'appeler une classe, la classe "Categorie".

Je lui donne le label "Catégorie", et en choice_label je précise "name", car je veux afficher le nom de chaque catégorie dans la liste. Il est ici possible d'en affecter plusieurs à une formation, aussi, je place le "multiple" à "true",on peut aussi choisir de n'affecter aucune catégories donc je place le 'required' à 'false', je créé un "->add('publishedAt')" de type "Date", qui va récupérer les informations de la classe(date de publication). Je crée un "widget -> single_text" , pour que la date soit présentée sous la forme d'un calendrier.

Enfin, j'ajoute un "add->submit", pour permettre l'enregistrement des données dans la base de données, grâce au clic sur un bouton "enregistrer". (avec l'import des "use" correspondant).

La fonction permet de créer le formulaire, en utilisant la classe "FormationType" contenant 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" qui va vérifier si le formulaire a été soumis et si il est valide. Si le formulaire est soumis et valide, il enregistre les informations du formulaire dans la base de

données, grâce à l'appel de la méthode "add" du FormationRepository. L'utilisateur est ensuite redirigé vers la page de gestion des formations.

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

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

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

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

Je dois ensuite réaliser l'affichage du formulaire, sur la page"_admin.formation.form.html.twig".

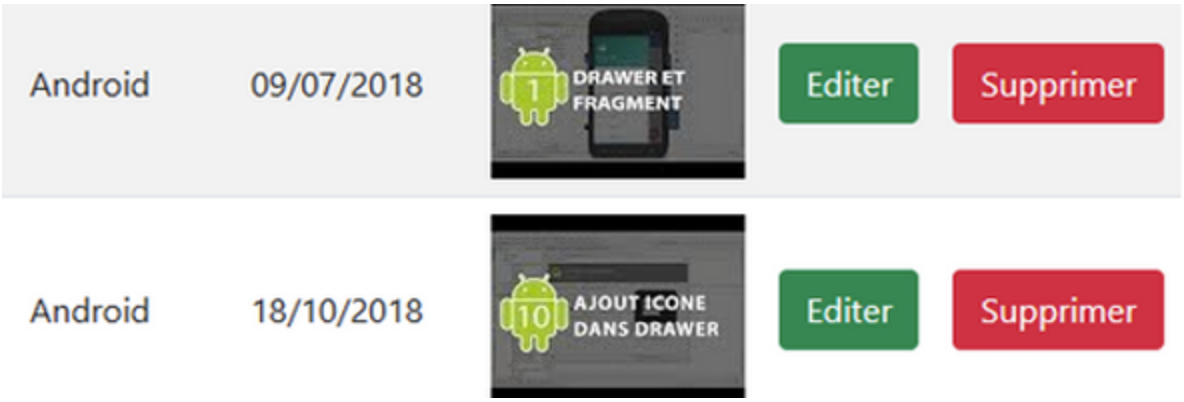
Pour cela, j'ajoute un "{{ form_start(formformation) }}" au tout début du code, pour signaler le démarrage du formulaire, et un ""{{ form_end(formformation) }}" en fin de code. Je crée après une colonne pour chacun des champs à afficher du "FormationType".

Les informations contenues dans les champs, sont affichées grâce à "{{ form_row(formformation.title) }}" . C'est à partir du formulaire créé dans le AdminFormationsController, qu'on récupère ces informations, donc je dois inscrire "formformation" devant chaque champ.

```
{{ form_start(formformation) }}
    <div class="row">
        <div class="col">
            <div class="row">
                <div class="col-auto">
                    {{ form_row(formformation.title) }}
                </div>
                <div class="col">
                    {{ form_row(formformation.description) }}
                </div>
                <div class="col">
                    {{ form_row(formformation.publishedAt) }}
                </div>
                <div class="col">
                    {{ form_row(formformation.playlist) }}
                </div>
                <div class="col">
                    {{ form_row(formformation.categories) }}
                </div>
                <div class="col">
                    {{ form_row(formformation.submit) }}
                </div>
            </div>
        </div>
    </div>
{{ form_end(formformation) }}
```

Après dans la page "admin.formations.html.twig", je rajoute le bouton "éditer", avec le chemin vers la fonction "admin.formation.edit". Lors du clic sur ce bouton, la méthode va ainsi pouvoir s'exécuter.

J'exécute l'application pour vérifier que le bouton "Editer" apparaît, et que le formulaire est fonctionnel lors du clic sur le bouton.




*Ajouter une formation*

Comme j'ai la fonctionnalité d'édition d'une formation, je n'ai pas à recréer de nouveau formulaire puisqu'il a déjà été construit. Je n'ai pas non plus besoin de faire de modifications dans mon builder.

Il ne reste donc qu'à créer la fonction d'ajout d'une formation dans AdminFormationsController, la page admin.formation.ajout.html.twig, qui va inclure le formulaire dans la page "_admin.formation.form.html.twig", puis ajouter le bouton et le chemin de la fonction d'ajout, dans la page admin.formations.html.twig, d'abord il faut créer la nouvelle page "admin.formation.ajout.html.twig", dans le dossier "templates>admin", avec l'include de la page "_admin.formation.form.html.twig" contenant le formulaire.

Je réalise ensuite la fonction dans le AdminFormationController le code est quasiment identique à celui de la fonction d'édition, puisque je récupère le même formulaire, je dois simplement affecter une nouvelle formation à la table des formations, j'ajoute donc la commande $formation=new Formation() ; juste avant de créer le formulaire. Je modifie les routes pour effectuer les redirections vers la page admin.formation.ajout.html.twig.

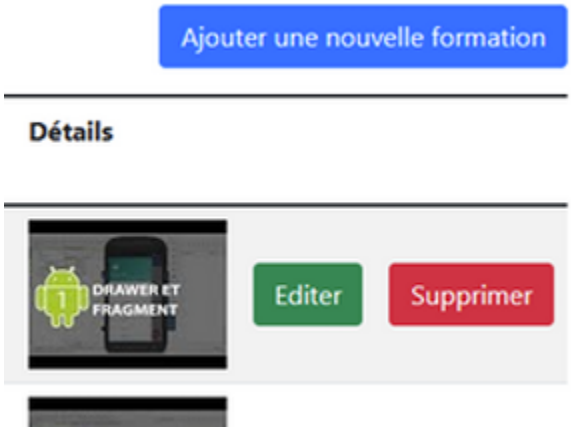
enfin j'insère le bouton « ajouter une nouvelle formation » à la page admin.formations.html.twig, en précisant la route vers la fonction admin.formation.ajout, pour que le click du bouton puisse exécuter la méthode.

Je place le bouton juste en dessous du block body, il se situe tout en haut à droite de la page.

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

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

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




### *Tris et filtre*

Je commence par le tri croissant et décroissant, de la liste des formations de la page des formations.

Pour cela, je dois créer une fonction "admin.formations.sort", qui sera appelée dans le fichier "admin.formations.html.twig", sur le clic des boutons "<" et ">".

Pour que la fonction réalise le tri "croissant et décroissant" des formations selon leur titre,je dois appeler les méthodes "findAllOrderBy" et "findAllOrderByTable" , du "FormationsRepository".

J'insère donc trois paramètres dans la fonction ($champ, $ordre, $table), si la variable $table possède une valeur, elle récupère la liste des formations grâce à l'appel de la méthode "findAllOrderByTable". Elle triera les enregistrements en prenant en compte les paramètres $champ, $valeur et $table, qui seront définis dans la page "admin.formations.html.twig". Si $table ne possède pas de valeur, la méthode qui sera appelée est "findAllOrderBy", la liste des formations est récupérée et sera triée selon le $champ et l'$ordre choisis dans la page "admin.formations.html.twig".

Ensuite la fonction va récupérer la liste des catégories, puis rediriger l'utilisateur vers la page des formations lorsqu'il cliquera sur les boutons "<" et ">"

Il faut ensuite appeler la méthode dans la page "admin.formations.html.twig", lors de la création des boutons "<" et ">".J'ajoute donc le chemin "admin.formations.sort" à chacun de mes deux boutons, et je définis le tri en insérant "ASC" au paramètre $ordre pour le bouton croissant, et "DESC" pour le bouton décroissant. Le $champ recherché sera "title" , car il s'agit du champ contenant le titre des formations, et je veux que le tri se fasse par le nom des formations

J'exécute ensuite l'application pour vérifier que dans la page de gestion des formations, les boutons de tris apparaissent et réalisent le tri correctement.

Je dois également réaliser un tri ascendant et descendant sur les playlists, et la date de publication.

Pour le tri sur les playlists, je réalise la même manipulation que pour le tri sur les formations, puis je modifie la valeur du $champ et de la $table, car il faut que le tri se fasse sur la table "playlist", et sur le nom de la playlist "name".

J'exécute le même procédé pour le tri par "date", en remplaçant le $champ par "publishedAt".

Maintenant le filtrage des formations, des playlists, et des catégories, grâce à un champ de recherche qui permettra de filtrer le tableau selon la valeur saisie dans la recherche.

Pour cela, je crée une fonction "findAllContain" dans le "AdminFormationsController", qui va permettre de récupérer le contenu de la recherche, et de le comparer avec les $champs des différentes $tables.

Je commence par chercher la valeur de la zone de recherche, avec une requête "get("recherche");" qui sera stockée dans la variable $valeur. Si une table est enregistrée en paramètre, la fonction va appeler la méthode "findByContainValueTable" du "FormationsRepository", pour retourner les enregistrements de la $table et du $champ qui correspondent à $valeur. Si aucune table n'est désignée, la fonction va appeler la méthode "findByContainValue" du "FormationsRepository", retournera les enregistrements du $champ correspondants à la $valeur. La fonction récupère également la liste de toutes les catégories.

Maintenant que la méthode permettant les enregistrements des saisies dans un formulaire de "recherche" est réalisée, je dois créer les formulaires de recherche dans la page "admin.formations.html.twig" et leur affecter un $champ et une $table si nécessaire.

Pour créer un formulaire de recherche, je dois utiliser les balises <form> et y insérer la méthode en "POST" , ainsi que le chemin vers la fonction à appeler, ici on va appeler la fonction "admin.formations.findallcontain".

Dans la balise , j'ajoute un input, qui va permettre la saisie d'une valeur dans la zone de recherche. J'ajoute un second input, pour créer un token "csrf_token" afin de masquer le champ.

J'utilise le même procédé pour le formulaire de recherche d'une playlist, mais je rajoute le paramètre $table en lui affectant "playlist" , et je modifie le $champ en lui affectant "name".

J'ajoute également "and table == 'playlist'" dans le "if" à droite du "table|default", et je remplace "filtre_title" par "filtre_name".

Je dois maintenant réaliser le filtre pour les catégories. Pour cela, je créé un formulaire d'une ligne, qui va contenir la liste des catégories, grâce à l'appel de la méthode "admin.formations.findallcontain", en remplaçant le $champ par "id" et en ajoutant le paramètre $table "categories".

J'ajoute une balise <select class> qui va permettre la sélection d'une catégorie dans la liste déroulante de recherche. Une fois qu'une catégorie a été sélectionnée, le formulaire va renvoyer les enregistrements qui contiennent cette catégorie.

Je rajoute la boucle "for" nécessaire au fonctionnement de la recherche par catégorie.

J'exécute l'application pour vérifier que les filtres de recherche s'affichent et fonctionnent correctement sur la page d'administration des formations.

# MISSION 2: CODER LA PARTIE BACK-OFFICE

## TÂCHE 2: GÉRER LES PLAYLISTS

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

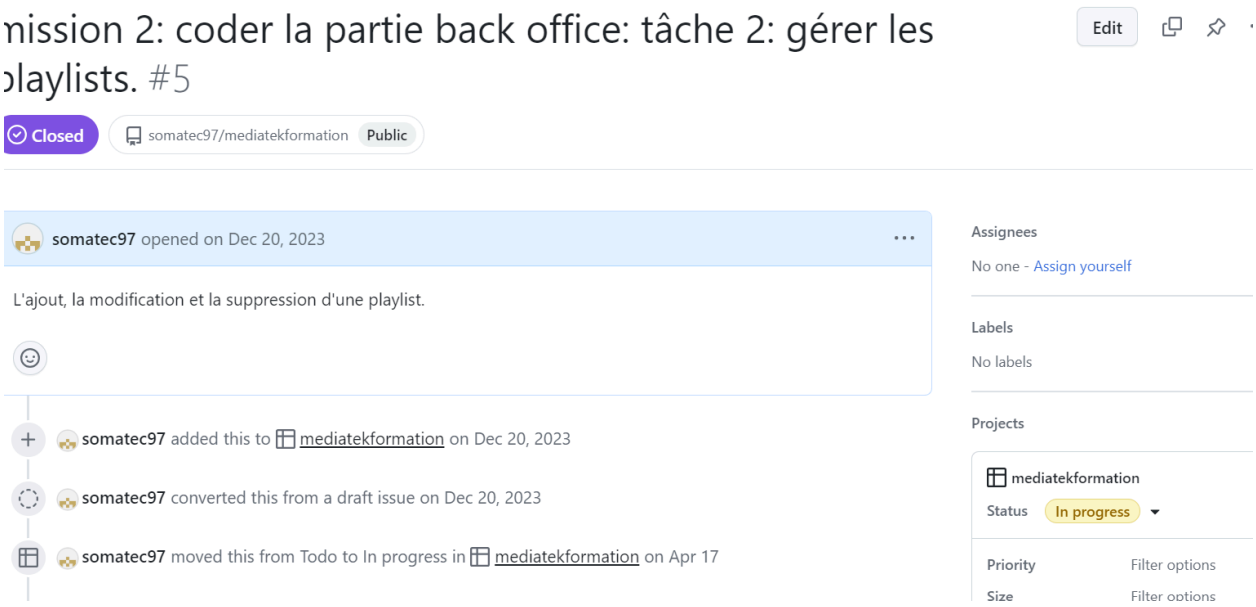
La suppression d'une playlist n'est possible que si aucune formation n'est rattachée à elle.

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

Un bouton doit permettre d'accéder au formulaire d'ajout d'une playlist. Les saisies doivent être contrôlées. L'ajout d'une playlist consiste juste à saisir son nom et sa description. Seul le champ name est obligatoire.

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

Temps estimé 5h - Temps mis : 3h (gain de temps grâce à la création de la page précédente).



### *Création de la page de gestion des playlists*

Afin de créer une page de gestion des playlists, je crée une page "admin.playlists.html.twig", qui va permettre de gérer les playlists, dans le dossier "templates>admin".

Il faut également créer la classe "AdminPlaylistsController" dans le dossier "controller", qui va permettre de réaliser les routes vers la page "admin.playlists.html.twig".

Dans le fichier "baseadmin", j'ajoute le chemin vers ma page d'administration des playlists dans la barre de navigation "admin.playlists".

Je réalise le "squelette" de la page de la gestion des playlists, qui va permettre par la suite d'afficher le tableau des playlists et de leurs catégories, et de pouvoir en ajouter, modifier ou supprimer.

Dans la classe "AdminPlaylistsController", je crée les variables, le constructeur , et l'index contenant la route vers la nouvelle page de l'application.

Pour afficher la liste des playlists, je vais dans la page "admin.formations.html.twig" , et je réalise une boucle "for", permettant de récupérer le nom des playlists, et de les placer dans une première colonne. J'affiche également les catégories, le nombre de formations par playlists, et le bouton “détail", avec le détail de chaque playlist, dans les autres colonnes.

### *Suppression d’une playlist*

Pour effectuer la suppression d'une playlist, je dois créer une fonction dans "AdminPlaylistsController", qui récupère la méthode "remove" du "PlaylistsRepository", puis faire une redirection vers ma page "admin.playlists.html.twig". Je crée ensuite le bouton "supprimer" , dans la page "admin.playlists.html.twig", pour cela il faut placer la commande à la fin de ma boucle "for", pour ajouter la colonne qui affichera les boutons "supprimer" sur chacune des playlists.

Il faut que je précise la route de ma fonction "admin.playlist.suppr", créée dans le "AdminPlaylistsController", et qui permet la suppression des playlists dans la base de données.

Afin de permettre de ne supprimer que les playlists qui ne contiennent aucune formation j'ajoute une condition "if" dans la colonne de suppression.

Si les playlists ont un nombre de formations égal à 0, dans ce cas là le bouton "supprimer" s'activera sur la ligne des playlists que l'on peut supprimer.

Je rajoute un "onclick" permettant de demander la confirmation de suppression, lors du clic sur le bouton "supprimer".

### *Édition d'une playlist*

Pour pouvoir modifier une playlist, il faut créer un nouveau bouton "éditer", qui, lorsque l'on va cliquer dessus, va permettre d'accéder à un formulaire d'édition de la playlist.

Donc, je dois réaliser une nouvelle page "_admin.playlists.html.twig" dans la racine du dossier "templates", qui va contenir le formulaire d'édition d'une playlist.

"admin.playlist.edit.html.twig", dans mon dossier "templates/admin", qui va me permettre d'appeler la page du formulaire "_admin.playlists.html.twig", lors du clic sur le bouton "éditer".

Dans la page "admin.playlist.edit.html.twig", j'ajoute l'include pour appeler le formulaire situé dans la page "_admin.playlist.form.html.twig".

Pour la création du formulaire dans "_admin.playlist.form.html.twig", il faut une nouvelle classe php "PlaylistType", dans le dossier "src>Form". Cette classe va me permettre de définir les champs du formulaire d'ajout ou d'édition d'une playlist.

Le formulaire doit récupérer le champ "name", avec le nom et la description de la playlist.

Puis le champ "formations" qui va permettre d'afficher la liste des formations associées .

Enfin, le formulaire doit contenir un bouton "submit", qui va permettre d'enregistrer les informations dans la base de données.

Pour que mon builder puisse récupérer le champ avec le nom de la playlist sélectionnée, et sa description, je dois ajouter au builder, un "->add name" de type "Text", pour ajouter le nom de la

playlist, on lui ajoute un "label => Playlist", qui va afficher le texte "Playlist" à côté du champ qui va contenir le nom de la playlist. On précise "'required' => true" car il est obligatoire pour une playlist d'avoir un nom.

Ensuite on ajoute un "->add description" de type "TextArea", avec un "'required' => false, car la description n'est pas obligatoire, on lui donne également un "label => Description".

Le formulaire doit également contenir le champ "formations", permettant d'afficher une liste déroulante des formations, et qui sélectionne les formations associées à la playlist pour laquelle on aura cliqué sur le bouton d'édition.

Pour que le formulaire d'édition d'une playlist prenne en compte les informations de la classe "Formation", afin de les enregistrer dans la liste déroulante, il va falloir ajouter la classe "Formation" au builder. La commande permettant de réaliser cela est la suivante "->add('formation', EntityType::class," .

On ajoute ensuite la classe "Formation", en lui précisant un label "formation", puis pour récupérer la liste des formations associées, on ajoute un "choice_label => title" , afin de récupérer les titres des formations.

On peut sélectionner plusieurs formations par playlist, donc on fait un "'multiple' => true", puis comme il n'est pas obligatoire d'avoir une formation on passe le "required" en "false".

Enfin, je créé un "add->submit", pour permettre l'enregistrement des données dans la base de données, grâce au clic sur un bouton "enregistrer"

Le builder étant terminé, je peux passer à la création de ma fonction sur le bouton "éditer", dans la classe "AdminPlaylistsController".

La fonction permet de créer le formulaire, en utilisant la classe "PlaylistType" contenant la définition des champs du formulaire, et récupère les playlists. La méthode "$formPlaylist->handleRequest($request)" gère la soumission du formulaire.

J'ajoute une condition "if" qui va vérifier si le formulaire a été soumis et si il est valide. Si le formulaire est soumis et valide, il enregistre les informations du formulaire dans la base de

données, grâce à l'appel de la méthode "add" du PlaylistRepository. L'utilisateur est ensuite redirigé vers la page de gestion des playlists. Si le formulaire n'est pas soumis et n'est pas valide, la fonction maintient l'utilisateur sur la page d'édition du formulaire.

Enfin, je créé un "add->submit", pour permettre l'enregistrement des données dans la base de données, grâce au clic sur un bouton "enregistrer"(avec la vérification de l'import des "use" correspondants).

Je dois ensuite réaliser l'affichage du formulaire, dans la page"_admin.playlist.form.html.twig".

Pour cela, je crée un "{{ form_start(formplaylist) }}" au tout début du code, pour signaler le démarrage du formulaire, et un ""{{ form_end(formplaylist) }}" en fin de code. Je construis ensuite une colonne pour chacun des champs à afficher du "PlaylistType".

Les informations contenues dans les champs, sont affichées grâce à "{{form_row(formplaylist.name) }}" . C'est à partir du formulaire créé dans le AdminPlaylistsController, qu'on récupère ces informations, donc je dois inscrire"formplaylist" devant chaque champ.

Je me dirige ensuite dans la page "admin.playlists.html.twig", et j'ajoute le bouton "éditer", avec le chemin vers ma fonction "admin.playlist.edit". Lors du clic sur ce bouton, la méthode va ainsi pouvoir s'exécuter.

J'exécute l'application pour vérifier que le bouton "Editer" apparaît, et que le formulaire est fonctionnel lors du clic sur le bouton.

### *Ajouter une playlist*

Il faut créer la fonction d'ajout d'une playlist dans le AdminPlaylistsController, la page admin.playlists.ajout.html.twig, qui va inclure le formulaire dans la page "_admin.playlist.form.html.twig", puis ajouter le bouton et le chemin de la fonction d'ajout, dans la page admin.playlists.html.twig.

Tout d'abord je crée la nouvelle page "admin.playlist.ajout.html.twig", dans le dossier "templates>admin" avec l'include de la page "_admin.playlist.form.html.twig" contenant le formulaire.

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

Je dois simplement affecter une nouvelle playlist à la table des playlists.

J'ajoute la commande "$playlists = new Playlist();" juste avant de créer le formulaire.

Je modifie ensuite la route, pour effectuer les redirections vers ma page "admin.playlist.ajout.html.twig"

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

Je place mon bouton dans une <p class="text-end"> juste en dessous du block body, pour qu'il se situe tout en haut à droite de la page.

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

### *Tris et filtres*

Toutes les étapes sont identiques et les explications du code aussi.

Je dois créer une fonction "admin.playlists.sort", qui sera appelée dans le fichier "admin.playlists.html.twig", sur le clic des boutons "<" et ">".

J'exécute ensuite l'application pour vérifier que dans la page de gestion des playlists, les boutons de tris apparaissent et réalisent le tri correctement

FILTRES

Je dois créer une fonction "findAllContain" dans le "AdminPlaylistsController", qui va permettre de récupérer le contenu de la recherche, et de le comparer avec les $champs des différentes $tables.

J'exécute l'application afin de vérifier que les filtres de recherche s'affichent et fonctionnent correctement sur la page d'administration des playlists.

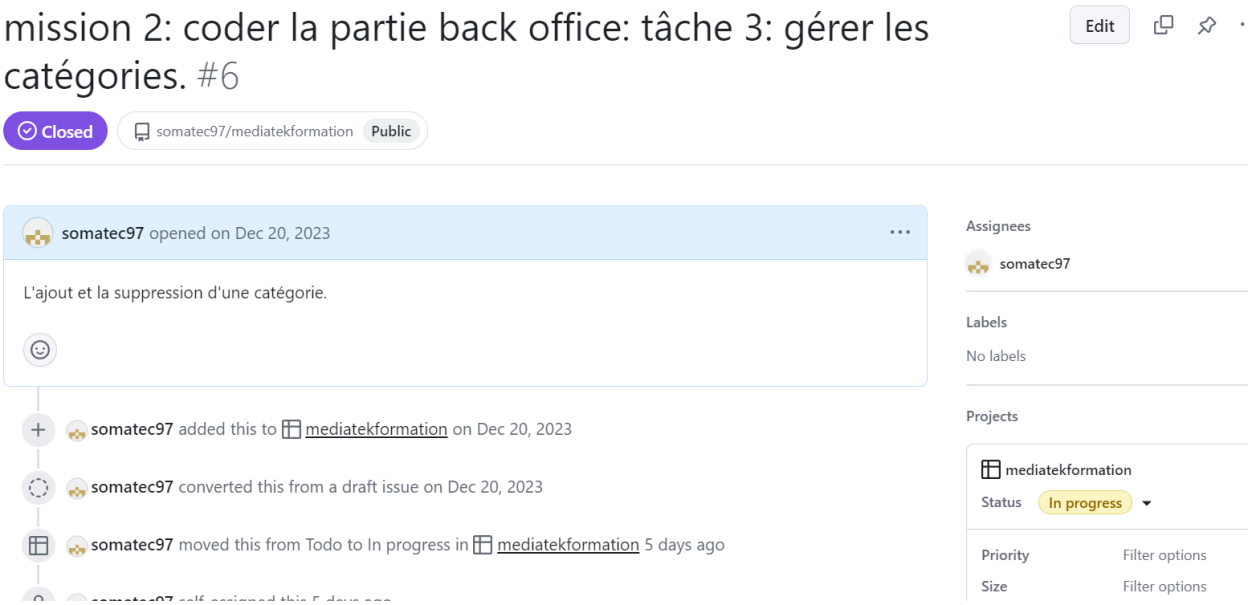
# MISSION 2: CODER LA PARTIE BACK-OFFICE

## TÂCHE 3: GÉRER LES CATÉGORIES

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.

Temps estimé 3h - Temps mis 2h.




Dans mon fichier "baseadmin" , j'ajoute le chemin vers la page de gestion des catégories dans la barre de navigation.

### *Création de la page gestion des catégories*

Je réalise le "squelette" de la page de gestion des catégories, qui va permettre par la suite d'afficher le tableau des catégories et de leurs formations, et de pouvoir en ajouter, modifier ou supprimer.

Dans la classe "AdminCategoriesController", je crée les variables, le constructeur , et l'index contenant la route vers la nouvelle page de l'application.

Pour afficher la liste des catégories, je vais dans la page "admin.categories.html.twig" , et je réalise une boucle "for", permettant de récupérer le nom des catégories, et de les placer dans une première colonne. J'affiche également les formations associées à chaque catégorie dans une seconde colonne.

### *Suppression d'une catégorie*

Pour la suppression d'une catégorie je dois créer une fonction dans"AdminCatégoriesController", qui récupère la méthode "remove" du"CategorieRepository", puis faire une redirection vers la route de la page "admin.categories.html.twig".

Je crée le bouton "supprimer" , dans la page "admin.categories.html.twig". Je dois pour cela, placer la commande à la fin de la boucle "for", pour ajouter la colonne qui affichera les boutons "supprimer" sur chacune des catégories.

Il faut que je précise la route de ma fonction "admin.categorie.suppr", créée dans le "AdminCategoriesController", et qui permet la suppression des catégories dans la base de données.

Afin de permettre de ne supprimer que les catégories qui ne contiennent aucune formation, j'ajoute une condition "if" dans la colonne de suppression.

Si les catégories ont un nombre de formations égal à 0, dans ce cas là le bouton "supprimer" s'activera sur la ligne des catégories que l'on peut supprimer.

J'ajoute aussi un "onclick" permettant de demander la confirmation de suppression lors du clic sur le bouton "supprimer".

### *Ajouter une catégorie*

Je crée la fonction d'ajout d'une catégorie "admin.categorie.ajout" dans la classe "AdminCategoriesController" qui consiste à comparer les noms des catégories, avec celui saisi dans le formulaire.

J'ajoute ensuite une condition "if" qui permet de créer la nouvelle catégorie si le nom de la catégorie n'est pas déjà présent dans la liste.

J'effectue ensuite la redirection vers la page d'administration des catégories.

Je dois ensuite construire le mini-formulaire, permettant d'effectuer l'ajout d'une catégorie dans la page  "admin.categories.html.twig".

# MISSION 2: CODER LA PARTIE BACK-OFFICE

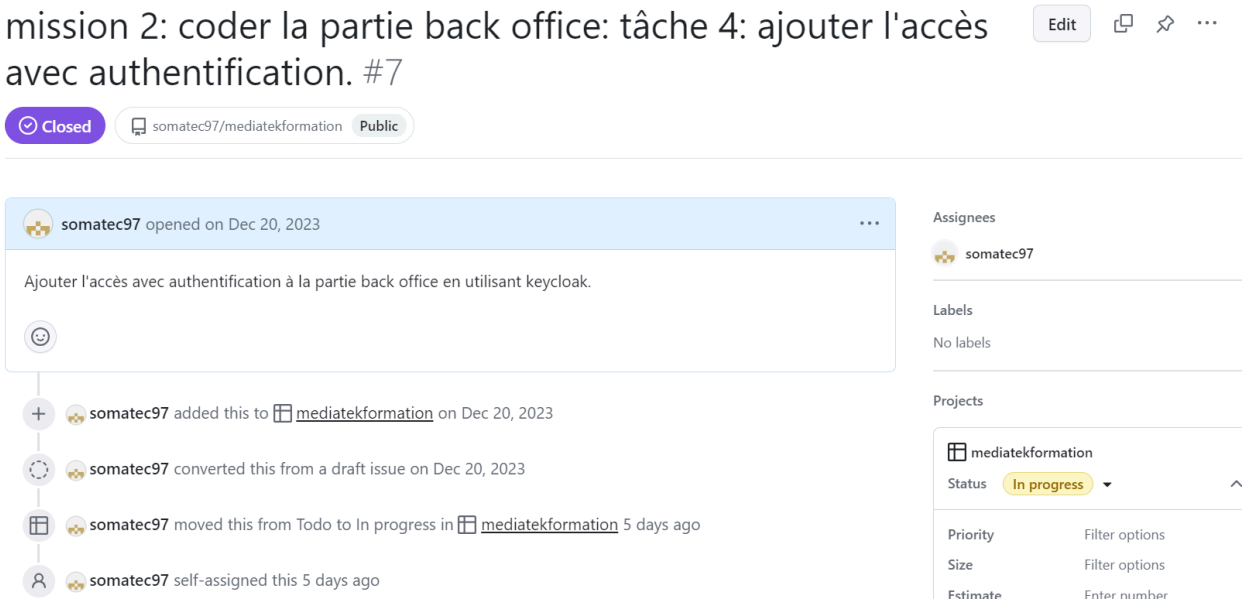
## TÂCHE 4: AJOUTER L'ACCÈS AVEC AUTHENTIFICATION

Le back office ne doit être accessible qu'après authentification : un seul profil administrateur doit avoir le droit d'accès.

Pour gérer l'authentification, utilisez Keycloak.

Il doit être possible de se déconnecter, sur toutes les pages (avec un lien de déconnexion).

Temps estimé 4h - Temps mis environ 3h.




### *Configuration de Keycloak*

je commence par configurer keycloak, pour le relier à l'application, et de permettre l'authentification de l'utilisateur pour accéder aux pages d'administration.

Il faut que je lance le serveur keycloak, en tapant la commande "kc.bat start-dev" , dans l'invite de commandes windows en mode Administration. Je dois pour cela être située dans le dossier "C:/keycloak/bin.

Une fois le serveur lancé, j'accède à la page "http://localhost:8080" dans mon navigateur, pour pouvoir aller dans la console d'administration de mon serveur keycloak.

Une fois dans le serveur, je créé un nouveau royaume que j'appelle "myapplis", et dans lequel je crée mon client "mediatekformation".

Avec ces paramètres dans l'onglet "settings" de mon client :

- Client type : OpenID Connect
- Always display in console : Off
- Client authentication : On
- Authorisation : Off
- Standard flow : On
- Implicit flow : On
- Direct access grants : On
- Service accounts roles : Off
- OAuth 2.0 Device Authorization Grant : Off
- OIDC CIBA Grant : Off

Je vérifie ensuite les paramètres suivants :

- Enabled : On
- Valid redirect URIs : *
- Consent required : On
- Display client on screen : On
- Front channel logout : Off
- Backchannel logout session required : On
- Backchannel logout revoke offline sessions : Off

Après il faut  désactiver les cookies, dans l'onglet "authentication > browser > cookies".

Je récupère ensuite le "client-secret" dans l'onglet "credentials" de mon client mediatekformation  et je le garde pour pouvoir l'utiliser plus tard dans l'application.

Je vais ensuite dans l'onglet "Users", et je créé un nouvel utilisateur, avec en paramètres :

- username : somaben
- email : soma.soumia.97@gmail.com
- Enabled : ON
- Email Vérified : OFF

Je lui affecte également tous les droits, et dans l'onglet "crendentials" je lui définit un mot de passe, avec "tempory" sur "OFF".

### *Configuration dans l'application*

Dans Netbeans, j'ouvre le fichier ".env" puis j'ajoute 3 lignes à la fin de mon fichier, dans lesquelles j'affecte les valeurs de mon client "mediatek-formation" créé sur le serveur keycloak.

L'utilisateur keycloak doit être enregistré dans la base de données du site dès qu'il s'est connecté, pour pouvoir gérer sa déconnexion sans dépendre de Keycloak.

Je crée donc la classe "user" et la table correspondante en tapant en lignes de commandes :

`php bin/console make:user` (dans mon dossier wamp/www/mediatekformation) .

Ensuite j'exécute la commande `php bin/console make:entity User` avec les paramètres suivants :

- name : keycloakId
- type : string
- length :255
- nullable : yes

Je crée ensuite le fichier de migration en tapant `php bin/console make:migration`, après je lance la migration en tapant `php bin/console doctrine:migrations:migrate` .

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

Deux bundles doivent être installés dans le projet.

Je tape la commande composer require knpuniversity/oauth2-client-bundle 2.10 qui me permet de créer le fichier "knpu_oauth2_client.yaml" dans le dossier "config/packages".

J'installe le second bundle en tapant composer require stevenmaguire/oauth2-keycloak 3.1 --with-all-dependencies

J'ouvre ensuite mon fichier "knpu_oauth2_client. yaml" et lui affecte les paramétrages.

- auth_server_url : url du serveur Keycloak, récupérée dans la variable décrite précédemment dans ".env"
- realm : le nom de notre royaume dans Keycloak
- client_id : client_id défini dans Keycloak (aussi récupéré dans ".env")
- client_secret : client_secret défini dans Keycloak (aussi récupéré dans ".env")
- redirect_route : nom de la route à appeler sur laquelle Keycloak redirigera aprèsl'authentification.

Je configure maintenant le "firewall", pour cela, j'ouvre mon fichier "security.yaml" dans "config>packages". J'y ajoute la route de redirection "oauth_login", et dans "access control", j'ajoute le chemin qui nécessite une authentification avec le type de rôle à avoir pour y accéder (path -> ^/admin et role -> ROLE_ADMIN).

Dans la fenêtre de commandes, je tape php bin/console make:controller OAuthController --no-template, la commande permet de créer le fichier "OAuthController.php" dans le dossier src > Controller. Ce contrôleur va gérer les routes liées à l'authentification.

Dans ce fichier, la méthode index doit recevoir un paramètre (de type ClientRegistry) et retourne un objet de type RedirectResponse. Cette méthode récupère le client 'keycloak' de l'objet reçu en

paramètre et appelle la méthode "redirect" sur cet objet, afin de rediriger la requête vers l'authentification gérée par Keycloak.

Il faut aussi une méthode pour prendre en charge la route de redirection du retour. Je rajoute donc la fonction "connectCheckAction".

Dans le dossier "src", je crée un dossier "Security", avec une nouvelle classe php"KeycloakAuthenticator.php", qui va permettre de gérer l'authentification. J'implémente les 5 méthodes de la classe, après avoir implémenté "AuthenticationEntryPointInterface" à la classe. J'ai juste à cliquer sur l'ampoule puis "implement all abstract method", puis je "fix imports".

En début de classe, j'ajoute les propriétés et le constructeur qui va les valoriser, puis je remplis mes 5 méthodes.

Explications :

La méthode "start" permet de spécifier comment démarrer une authentification. Elle envoie donc vers une route temporaire qui est celle définie dans le contrôleur précédent (OAuthController) et donc qui s'occupe de solliciter Keycloak.

La méthode "supports" doit renvoyer true si le système d'authentification doit se déclencher pour l'url donnée. Donc cette méthode est appelée à chaque fois qu'une requête est reçue (une url sollicitée). Il faut donc vérifier si l'url donnée correspond à 'oauth_check' : dans ce cas la méthode retourne true . Si cette méthode retourne 'true', alors la méthode "getCredentials" est automatiquement appelée.

La méthode "authenticate" s'occupe de récupérer le client correspondant, dans Keycloak, et le token, à partir des informations données.Une fois ces informations obtenues, il est possible de récupérer l'utilisateur : le but est de l'enregistrer dans la base de données.

Donc, il y a 3 possibilités :

- soit l'utilisateur existe déjà dans la BDD et s'est déjà connecté avec Keycloak
- soit l'utilisateur existe dans la BDD mais ne s'est jamais connecté avec Keycloak
- soit l'utilisateur n'existe pas encore dans la BDD et il faut alors le créer.

Dans un premier temps, on tente de récupérer dans la BDD, un utilisateur dont le champ 'keycloakId' correspond à l'id récupéré avec Keycloak. Si un utilisateur est trouvé, alors il est directement retourné.

Dans le second cas, lors de la création de la classe User, c'est la propriété 'email' qui a été définie comme unique. Il suffit de récupérer l'email de l'utilisateur récupéré avec Keycloak, puis de rechercher dans la BDD si un utilisateur existe avec cet email. Si c'est le cas, alors on valorise sa propriété 'keycloakId' avec l'id récupéré avec Keycloak, puis on enregistre les modifications dans la BDD et on retourne l'utilisateur.

Dans le troisième cas, l'utilisateur est créé, les différentes propriétés renseignées, avant d'être enregistré dans la BDD puis retourné.

La méthode "onAuthenticationFailure" s'exécute en cas de problème (exception déclenchée) dans une autre méthode.

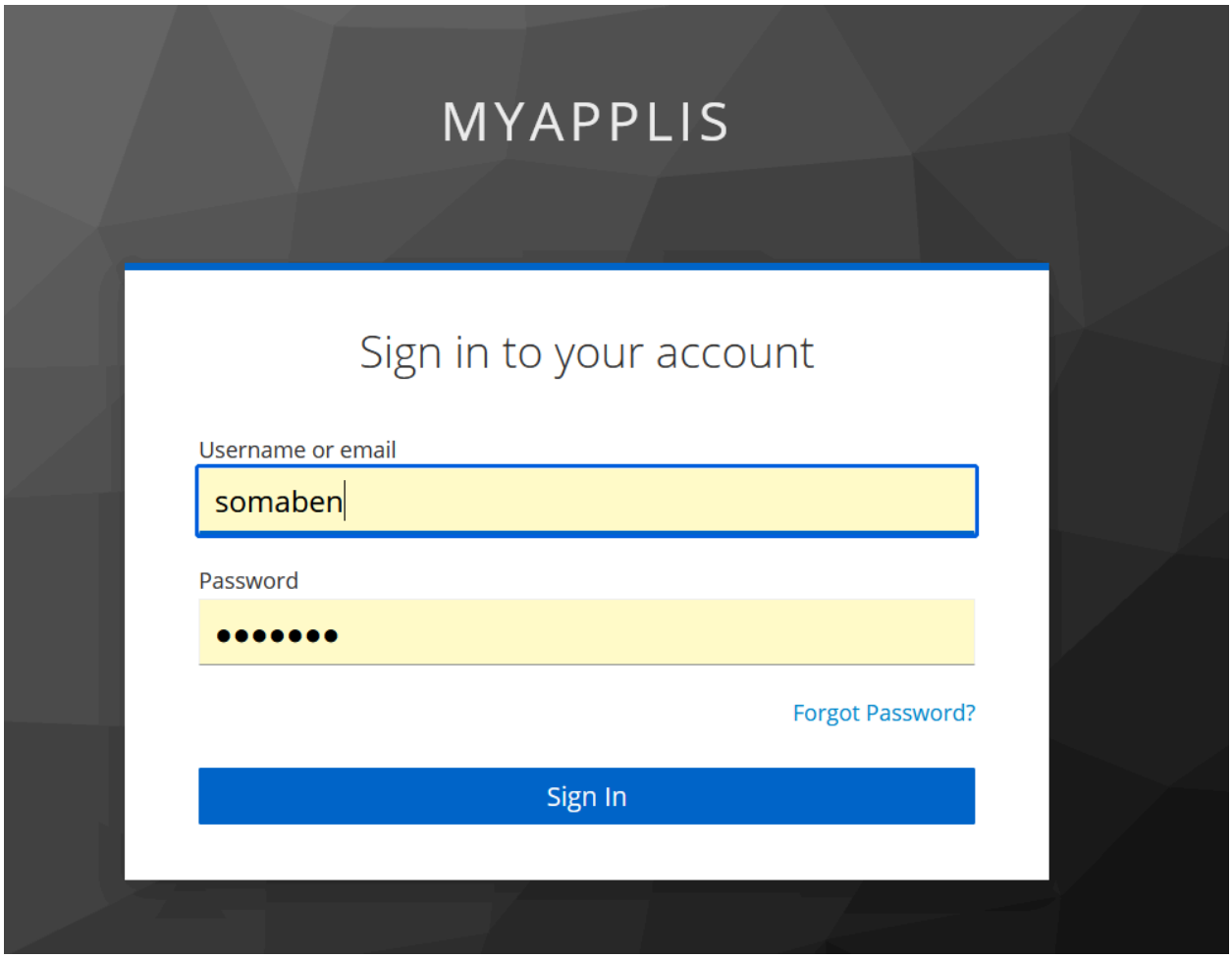
La méthode "onAuthenticationSuccess" s'exécute quand tout s'est bien passé. On peut alors rediriger vers la route voulue au départ, donc la partie 'admin' du site.

Il reste à ajouter le chemin de la classe précédente dans le firewall (fichier security.yaml qui se trouve dans "config > packages") et à préciser le point d'entrée.

### *Tester l'accès sécurisé*

Je dois lancer l'application, qui me dirige vers la 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".

Si les informations saisies sont incorrectes, je vais rester sur la même fenêtre avec un message d'information, en attendant une saisie correcte. Si je saisi les bonnes informations de l'utilisateur créé dans Keycloak, "somaben" avec le password que j'ai défini, alors je vais pouvoir consulter les pages d'administration.




Je vais maintenant m'occuper de configurer la déconnexion de l'utilisateur.

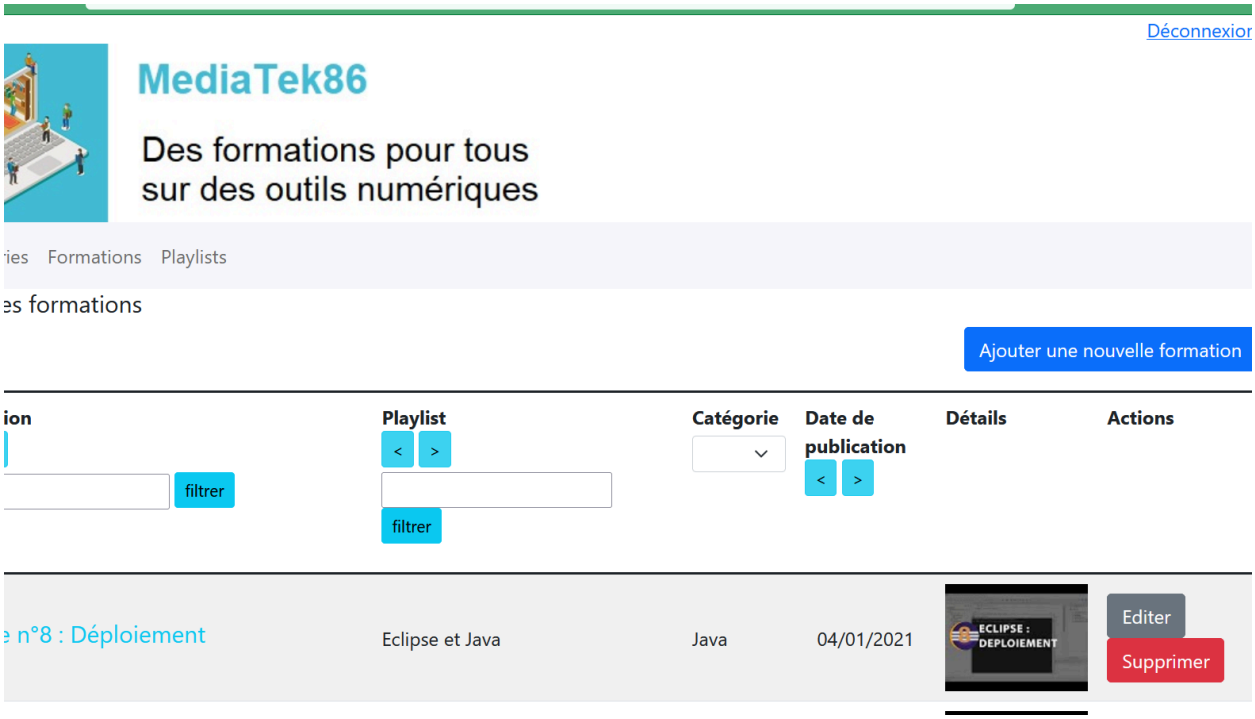
Le but est d'ajouter un lien dans la partie "admin" pour pouvoir se déconnecter. Dans le dossier "templates", j'ouvre le fichier "baseadmin.html.twig". Dans ce fichier, au-dessus du titre, j'ajoute un lien "déconnexion" qui redirige vers la route 'logout' non encore créée.

Dans le contrôleur "OAuthController", dossier "src > Controller", j'ajoute la méthode qui sera appelée avec la route 'logout', la méthode est vide car c'est le firewall qui va prendrel a relève.

Dans security.yaml (qui se trouve dans "config > packages"), dans la partie main du firewall, je peux ajouter le logout.

Je lance l'application, et je me dirige dans la page d'administration, le serveur keycloak après la demande de m'authentifier.

Je peux désormais accéder à mes pages d'administrations, et j'ai également le lien du bouton "déconnexion" tout en haut, qui me permet de me déconnecter lorsque je clique dessus.




# MISSION 3: TESTER ET DOCUMENTER

## TÂCHE 1: CRÉER LES TESTS

Tests unitaires:

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

Tests d'intégration sur les règles de validation:

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

Tests d'intégration sur les Repository:

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

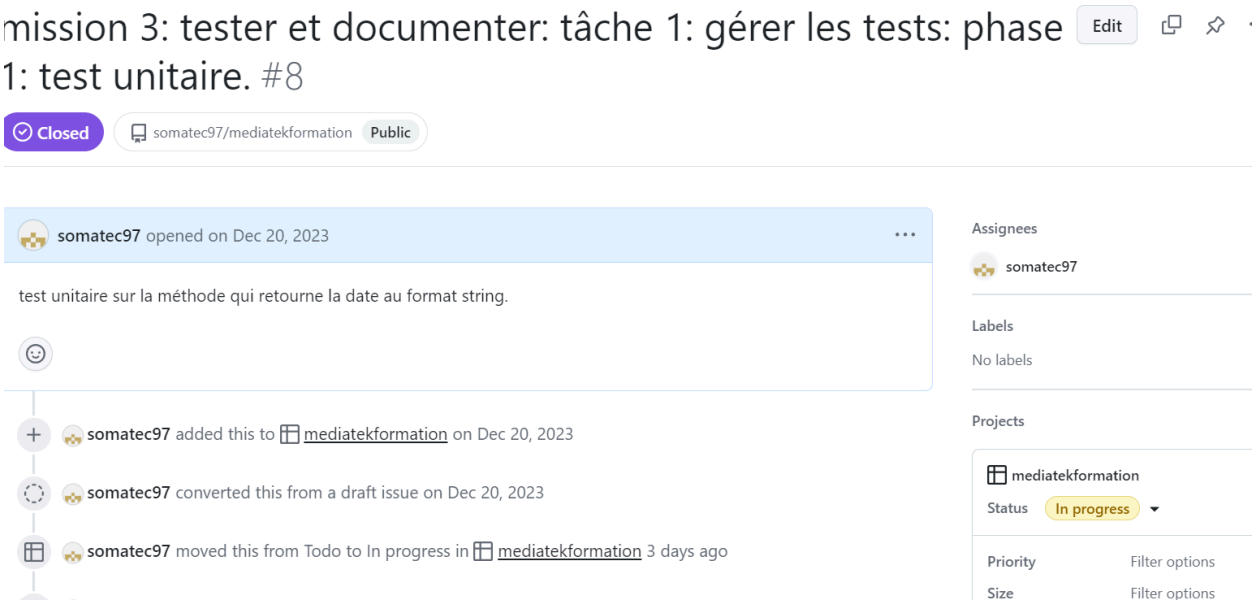
Tests fonctionnels :

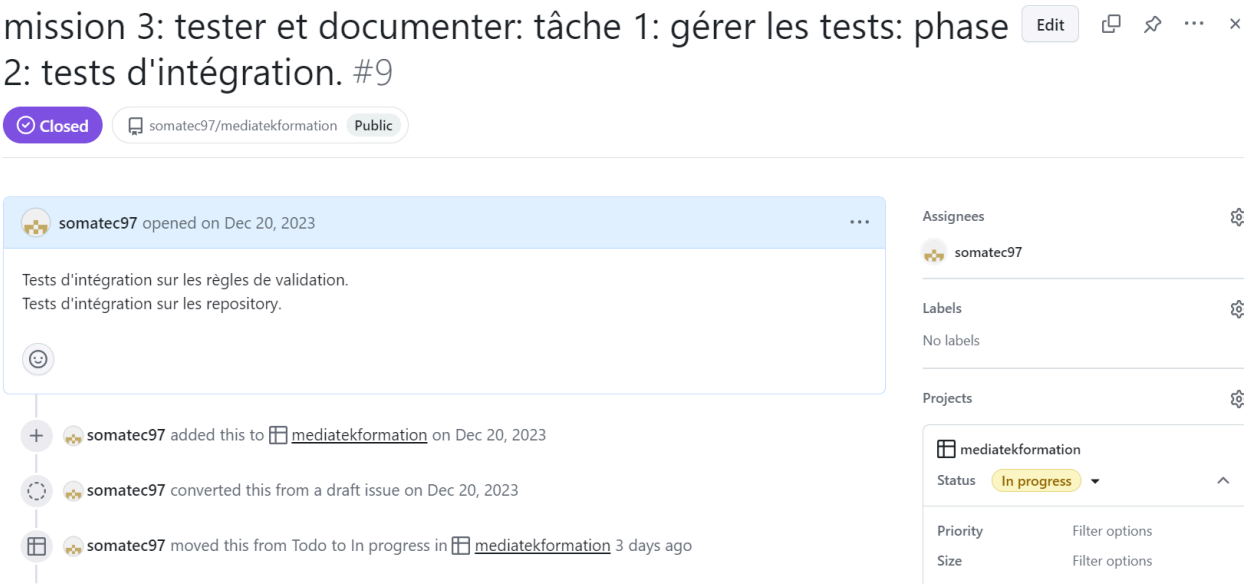
- Contrôler que la page d'accueil est accessible.
- Dans chaque page contenant des listes :
    - contrôler que les tris fonctionnent (en testant juste le résultat de la première ligne).
    - contrôler que les filtres fonctionnent (en testant le nombre de lignes obtenues et le résultat de la première ligne).
    - contrôler que le clic sur un lien (ou bouton) dans une liste permet d'accéder à la bonne page (en contrôlant l'accès à la page mais aussi le contenu d'un des éléments de la page).

Tests de compatibilité :

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

Temps estimé 7h - Temps mis environ 15h.


mission 3: tester et documenter: tâche 1: gérer les tests: phase 1: test unitaire. #8
Edit
Closed
somatec97/mediatekformation
Public
somatec97 opened on Dec 20, 2023
test unitaire sur la méthode qui retourne la date au format string.
somatec97 added this to mediatekformation on Dec 20, 2023
somatec97 converted this from a draft issue on Dec 20, 2023
somatec97 moved this from Todo to In progress in mediatekformation 3 days ago
Assignees
somatec97
Labels
No labels
Projects
mediatekformation
Status
In progress
Priority
Filter options
Size
Filter options


mission 3: tester et documenter: tâche 1: gérer les tests: phase 2: tests d'intégration. #9
Edit
Closed
somatec97/mediatekformation
Public
somatec97 opened on Dec 20, 2023
Tests d'intégration sur les règles de validation.
Tests d'intégration sur les repository.
somatec97 added this to mediatekformation on Dec 20, 2023
somatec97 converted this from a draft issue on Dec 20, 2023
somatec97 moved this from Todo to In progress in mediatekformation 3 days ago
Assignees
somatec97
Labels
No labels
Projects
mediatekformation
Status
In progress
Priority
Filter options
Size
Filter options

# mission 3: tester et documenter: tâche 1: gérer les tests: phase 3: tests fonctionnels. #10

Edit

Closed | somatec97/mediatekformation Public

somatec97 opened on Dec 20, 2023

Tests fonctionnels (accès à l'accueil, tris, filtres...).

somatec97 added this to mediatekformation on Dec 20, 2023

somatec97 converted this from a draft issue on Dec 20, 2023

somatec97 moved this from Todo to In progress in mediatekformation 3 days ago

somatec97 self-assigned this 3 days ago

Assignees: somatec97

Labels: No labels

Projects: mediatekformation

Status: In progress

Priority: Filter options

Size: Filter options

# mission 3: tester et documenter: tâche 1: gérer les tests: phase 4: tests de compatibilité (Selenium) #11

Edit

Closed | somatec97/mediatekformation Public

somatec97 opened on Dec 20, 2023

Tests de compatibilité de navigateurs (avec Selenium).

somatec97 added this to mediatekformation on Dec 20, 2023

somatec97 converted this from a draft issue on Dec 20, 2023

somatec97 moved this from Todo to In progress in mediatekformation 3 days ago

somatec97 self-assigned this 3 days ago

Assignees: somatec97

Labels: No labels

Projects: mediatekformation

Status: In progress

Priority: Filter options

Size: Filter options

# Plan de tests

## Tests unitaires

| But du test | Action de contrôle | Résultat attendu | Bilan |
|---|---|---|---|
| Contrôler la méthode getPublishedAtString() de la classe Formation pour voir si elle retourne la bonne date au bon format. | Test unitaire lancé avec la date : 2021-01-04 17:00:12 | 04/01/2021 | OK |

*Tests d'intégration*

## Tests d'intégration

| But du test | Action de contrôle | Résultat attendu | Bilan |
|---|---|---|---|
| Lors de l'ajout ou de la modification d'une formation, contrôler que la date n'est pas postérieure à aujourd'hui. | Tst d'intégration sur les règles de validation | Test passé | ok |
| Contrôler la méthode add() de «FormationRepository» pour voir si elle ajoute une formation. | Test d'intégration avec count[] | $nbformation+1==>test passé | ok |
| Contrôletr la méthode remove() de «FormationRepository» pour voir si elle supprime une formation. | Test d'intégration avec count[] | $nbformation-1==>test passé | ok |
| Contrôler la méthode findByContainValue() de «FormationRepository» pour voir si elle filtre une formation dont un champ contient une valeur. | Test d'intégration est lancé le filtre sur le champ «title» et la valeur «UML» | UML : Diagramme de paquetages, (la premier formation). | ok |
| Contrôler la méthode add() de «PlaylistRepository» pour voir si elle ajoute une playlist. | Test d'intégration avec count[] | $nbformation+1==>test passé | ok |
| Contrôler la méthode remove() de «PlaylistRepository» pour voir si elle supprime une playlist.. | Test d'intégration avec count[] | $nbformation-1==>test passé | ok |
| Contrôler la méthode add() de «CategorieRepository» pour voir si elle supprime une formation. | Test d'intégration avec count[] | $nbformation+1==>test passé | ok |
| Contrôler la méthode remove() de «CategorieRepository» pour voir si elle supprime une catégorie. | Test d'intégration avec count[] | $nbformation-1==>test passé | ok |
| | | | |
| | | | |

*Tests fonctionnels*

## Tests fonctionnels

| But du test | Action de contrôle | Résultat attendu | Bilan |
|---|---|---|---|
| Contrôler que la page d'accueil est accessible | Test fonctionnel avec : assertResponseStatusCodeSame(Response::HTTP_OK) | HTTP_OK | ok |
| Contrôler que la page des formations est accessible | Test fonctionnel avec : assertResponseStatusCodeSame(Response::HTTP_OK) | HTTP_OK | ok |
| Contrôler que la page playlists est accessible | Test fonctionnel avec : assertResponseStatusCodeSame(Response::HTTP_OK) | HTTP_OK | ok |
| | | | |
| | | | |

*Tests de compatibilité*

## Tests de compatibilité

| But du test | Action de contrôle | Résultat attendu | Bilan |
|---|---|---|---|
| Créer un scénario avec Selenium, sur la partie front office, et le jouer sur plusieurs navigateurs pour tester la compatibilité du site | Test de compatibilité est lancé pour tester toutes les fonctionnalitées de site. | Aucune erreur | ok |
| | | | |
| | | | |

### TÂCHE 2: DOCUMENTATION TECHNIQUE

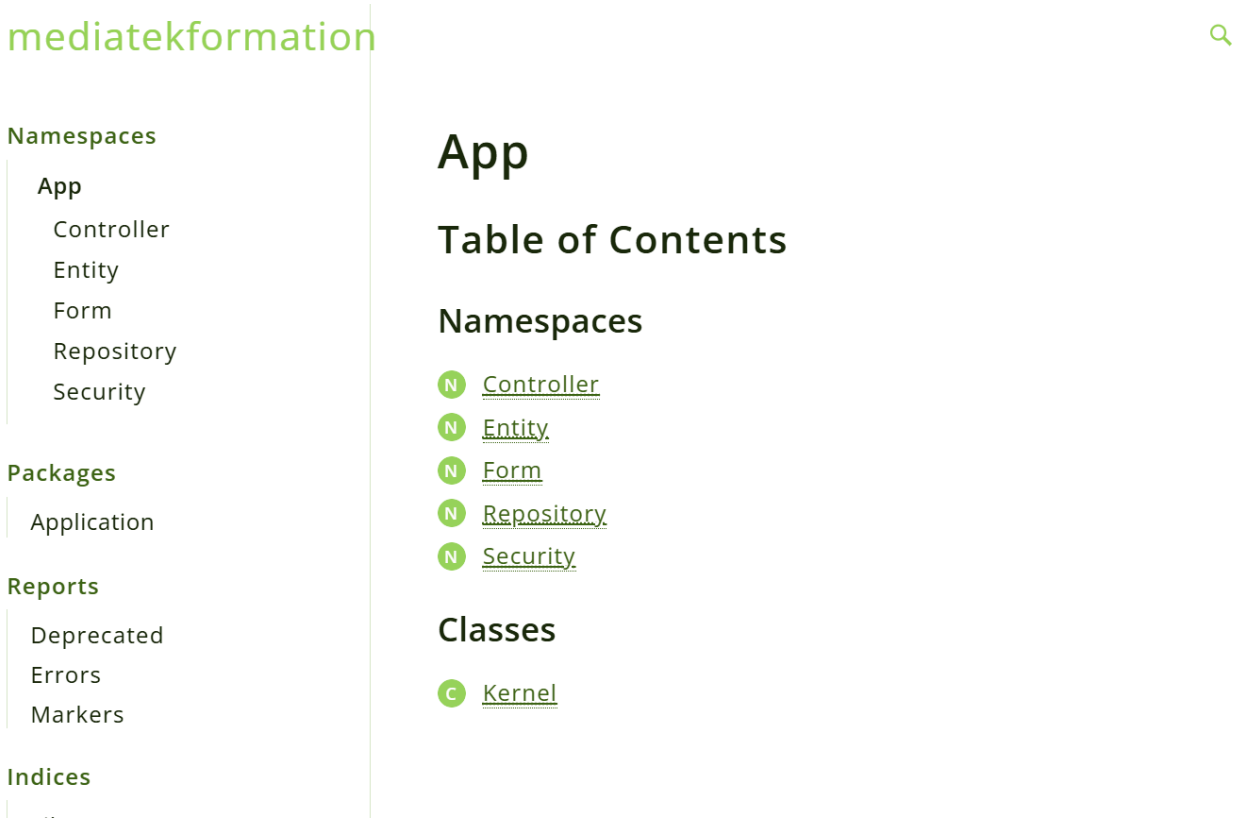
Contrôler que tous les commentaires normalisés nécessaires à la génération de la documentation technique ont été correctement insérés.

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 vais dans mon invite de commandes en mode administrateur et j'insère la commande suivante.

"C:\wamp\bin\php\php8.1.2\php.exe""C:\wamp\bin\php\php8.1.2\ext\phpDocumentor.phar" "run" "--ansi" "--directory" "C:/wamp/www/mediatekformation/src" "--target" "C:/wamp/www/mediatekformation_doc" "--title" "mediatekformation"

je vais ensuite dans le dossier où la documentation a été enregistrée, puis je clique sur le fichier "index" > "ouvrir avec google chrome" , et j'accède ainsi à la première page de la documentation technique.



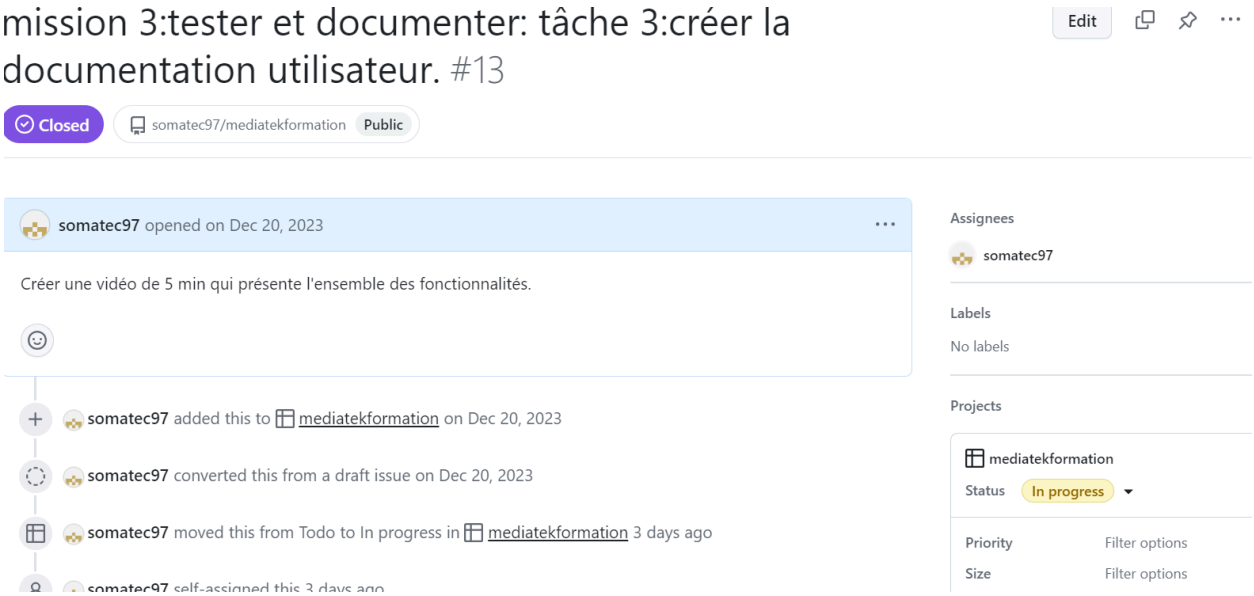

# MISSION 3: TESTER ET DOCUMENTER

## TÂCHE 3: DOCUMENTATION UTILISATEUR

Créer une vidéo qui permet de montrer toutes les fonctionnalités du site (front et back-office).

Cette vidéo ne doit pas dépasser les 5 min et doit présenter clairement toutes les fonctionnalités, en montrant les manipulations qui doivent être accompagnées d'explications orales.

Temps estimé 2h - Temps mis 2h.




# MISSION 4: DÉPLOYER LE SITE ET GÉRER LE DÉPLOIEMENT CONTINU

## TÂCHE 1: DÉPLOYER LE SITE

Installer et configurer le serveur d'authentification Keycloak dans une VM en ligne.

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.

Temps estimé 2h - Temps mis 2h.

### *Configuration Keycloak HTTPS*

Afin de procéder à la mise en ligne de mon serveur d'authentification keycloak, j'ai dû procéder à différentes étapes.

Tout d'abord, il a fallu que je crée une machine virtuelle windows, dans mon compte Azure, dans laquelle il a ensuite été nécessaire que j'installe le serveur keycloak.

J'ai défini un nom DNS à ma machine virtuelle "somakeycloak", qui va m'être utile durant la suite des étapes.

Grâce à la connexion bureau à distance de windows, je peux accéder à ma VM à partir de mon ordinateur, et procéder à l'installation du serveur keycloak.

Dans la machine virtuelle, il faut installer "openJDK" et créer la variable d'environnement "JAVA_HOME", puis mettre le chemin vers mon installation jdk.

Je vais ensuite sur le site keycloak, pour le télécharger en version 19.0.1, je lance ensuite keycloak en tapant la commande "kc.bat start-dev" . Je me rends ensuite à l'adresse "localhost:8080" dans le navigateur de la VM, après il faut créer un compte admin, j'accède à la console d'administration après m'être authentifiée avec ce compte.

Je crée après le royaume "myapplis", et lui affecte les mêmes paramètres que lors de la configuration de mon projet symfony en localhost.

J'ajoute un nouvel utilisateur et lui affecte tous les droits nécessaires.

Je finalise la configuration en tapant dans la fenêtre de commandes "kc.bat build" .

Il faut ensuite que j'installe le certificat nécessaire à l'accessibilité de keycloak en HTTPS, pour cela, je commence par installer "xampp", une fois l'installation terminée, je clique sur le "start" de "apache" dans mon serveur xampp.

Je dois ensuite installer "Certbot" pour obtenir un certificat, je choisis "Apache" on "Windows", une fois téléchargé, je lance l'installation, puis j'ouvre ma fenêtre de commandes en mode admin et je tape "certbot certonly --webroot".

Je rentre un email valide, je valide les termes, je rentre mon nom de domaine "DNS" → somakeycloak.francecentral.cloudapp.azure.com, et je rentre le web root "C:\xampp\htdocs" . Je stoppe Apache, je ferme et désinstalle Xampp qui n'est plus utile.

Dans une fenêtre de commandes lancée en mode admin, dossier C:\keycloak\bin, je lance la commande suivante, dans ma machine virtuelle :

kc.bat.start--hostname=somakeycloak.francecentral.cloudapp.azure.com--https-certificate-file=C:\Certbot\live\somakeycloak.francecentral.cloudapp.azure.com\fullchain.pem --https-certificate-key-file=C:\Certbot\live\somakeycloak.francecentral.cloudapp.azure.com\privkey.pem --https-port=443

Pour accéder au serveur keycloak en dehors de la VM, je me rends à l'adresse "somakeycloak.francecentral.cloudapp.azure.com" .

Dans l'application "mediatekformation", il faut donner les bonnes valeurs aux 3 variables : KEYCLOAK du fichier .env :

- ➔ KEYCLOAK_APP_URL: https://somakeycloak.francecentral.cloudapp.azure.com
- ➔ KEYCLOAK_SECRET : qQBW720kwKFvXHC5ZqsDMnlsGTF8EAQw
- ➔ KEYCLOAK_CLIENTID : mediatekformation.

*Déployer la base de données*

J'ai choisi l'hébergeur "hostinger", pour toute la partie de déploiement de l'application.

Afin de déployer la bdd dans mon hébergeur, je fais une exportation de la bdd de l'application, et je copie le contenu de mon fichier.

Je vais ensuite dans l'onglet "bases de données > phpmyadmin" de mon hébergeur, je crée ma base de données, et un utilisateur associé.

Ma base de données aura en paramètres :

- ➔ nom base de données : u891929832_mediatekformat
- ➔ utilisateur : u891929832_somaben
- ➔ mot de passe : Somaben91@

J'ouvre ensuite la base de données, et j'importe le fichier sql, je clique ensuite sur "importer", et ma base de données apparaît avec les tables correctement implantées.

J'attribue également un site web à ma base de données "mediatekformation-soma.com" , il s'agit du site dans lequel l'application sera implantée.

Je vais ensuite dans Netbeans, pour modifier les informations d'accès à la base données dans le fichier ".env”

*Déployer le site*

Le déploiement de l'application nécessite quelques configurations dans Netbeans. Je vais dans la fenêtre de commandes et je tape la commande : composer require symfony/apache-pack , dans le dossier du projet. Cette commande permet de créer le fichier ".htaccess", dans le dossier "public". Je modifie également la variable "APP_ENV : dev", en "APP_ENV : prod" , dans mon fichier ".env".

Pour déployer le site sur mon hébergeur hostinger, je vais dans l'onglet "hébergement", puis je clique sur le bouton "gérer" qui est présent à droite de mon nom de domaine.

Je vais ensuite dans le "gestionnaire de fichier", et j'y insère le dossier du projet, préalablement envoyé dans un dossier zip compte tenu du temps que cela prend. Mon projet est déposé dans le fichier "public_html". J'ai extrait les fichiers de mon dossier .zip .

# MISSION 4 : DÉPLOYER LE SITE ET GÉRER LE DÉPLOIEMENT CONTINU

## TÂCHE 2: GÉRER LA SAUVEGARDE ET LA RESTAURATION DE LA BDD

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

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

Temps estimé 1h.


*Automatisation de la sauvegarde*

*Restauration manuelle de la BDD*

# MISSION 4: DÉPLOYER LE SITE ET GÉRER LE DÉPLOIEMENT CONTINU

## TÂCHE 3: METTRE EN PLACE ET LE DÉPLOIEMENT CONTINU

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

Temps estimé 1h

# BILAN

Lors de la réalisation de ce projet, j'ai atteint les objectifs concernant le nettoyage et l'optimisation du code, et l'ajout de plusieurs fonctionnalités.

Par contre, j'ai rencontré des difficultés au début du projet, comme le code ayant déjà été créé par un autre développeur.

J'ai également réussi à créer les pages d'administrations des formations, des playlists et des catégories, puis à créer une authentification keycloak sur ces pages. Je n'ai rencontré aucun problèmes particuliers lors de cette étape.

Lors de la phase de création des tests de mes fonctions, j'ai éprouvé quelques difficultés. Je n'ai pas bien compris comment remplir le "plan de tests", et j'ai rencontré des bugs qui m'ont fait échouer plusieurs  tests (je n'ai fait que certains tests!).

Le projet m'a permis d'apprendre à configurer une machine virtuelle keycloak, permettant l'accès au serveur d'authentification en HTTPS, et j'ai ainsi pu réaliser le déploiement en ligne avec authentification sécurisée.

En conclusion, c'était une expérience  satisfaisante, malgré ses difficultés sur tout le départ et la mission des tests.

J'ai dû revoir la totalité du TP réalisé en cours afin de réussir toutes les missions.

Page de la mission: https://soumia-ben-portfolio.com/mediatekformation/

Dépôt distant: https://github.com/somatec97/mediatekformation.git

Lien du site en ligne: https://mediatekformation-soma.com