FAKIRI YASSINE 2023

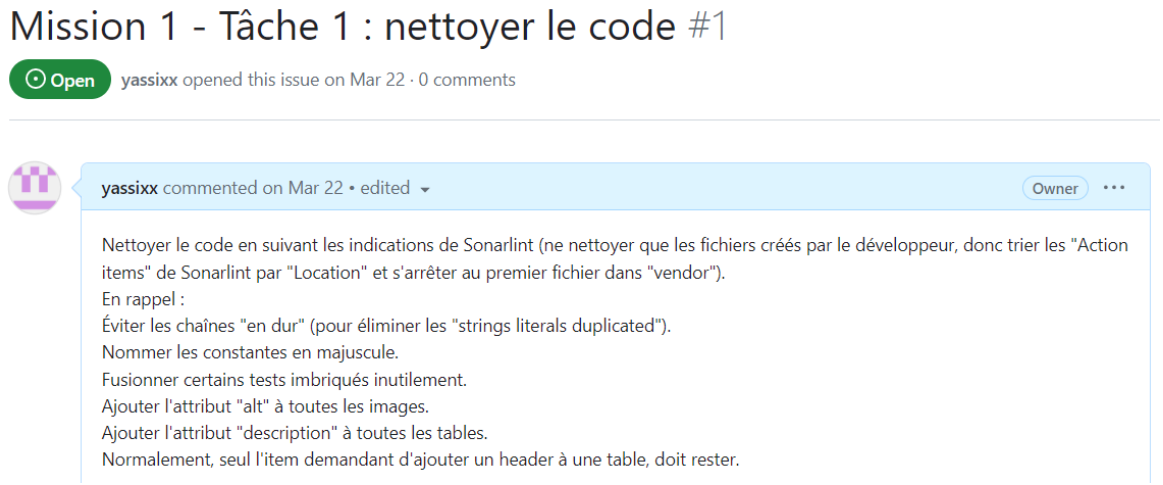
BTS SIO SLAM

ATELIER DE PROFESSIONNALISATION n°1

“MEDIATEK FORMATION”

Mission 1 :

Tache 1 – Nettoyer le code



Nettoyage du code en suivant les consignes de SonarLint. Pour chaque dossier du projet j'ai analysé le code avec SonarLint

Error : String String literals should not be duplicated

- "FormationsController"
- "PlaylistsController"
- "FormationRepository"
- "PlaylistRepository"

Cette erreur indique qu'il y a deux fois la même phrase dans le code. Cela peut arriver si un développeur copie et colle du code sans le modifier correctement ou s'il utilise par erreur la même phrase dans deux parties différentes du programme. Avoir deux phrases identiques peut causer des problèmes de lisibilité du code et des erreurs potentielles si une phrase est modifiée sans que l'autre ne le soit. Pour corriger cette erreur, il suffit de supprimer une des phrases identiques.

Exemple FormationController :

`pages/formations.html.twig -> FORMATIONS_PATH`

Création d'une constante PLAYLISTS_PATH dans laquelle j'affecte la chaine de caractères "pages/formations.html.twig" cela permettra d'éviter les redondances et je remplace les chaines "pages/formations.html.twig" par ma constante FORMATIONS_PATH

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

Exemple PlaylistsController :

`pages/playlists.html.twig -> PLAYLISTS_PATH`

Création d'une constante PLAYLISTS_PATH dans laquelle j'affecte la chaine de caractères "pages/playlists.html.twig" cela permettra d'éviter les redondances et je remplace les chaines "pages/playlists.html.twig" par ma constante PLAYLISTS_PATH

Exemple FormationRepository :

`f.publishedAt -> F_PUBLISHED_AT`

Création d'une constante PLAYLISTS_PATH dans laquelle j'affecte la chaine de caractères "f f.publishedAt" cela permettra d'éviter les redondances et je remplace les chaines "f.publishedAt" par ma constante F_PUBLISHED_AT

Error : Constant names should comply with a naming convention

- "Formation" dans dossier Entity :
  La constante "cheminImage" n'a pas été écrite correctement, les constantes doivent être en UPPERCASE

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

Error : Collapsible "If statements should be merged"

- "Playlists" dans dossier Entity
  Le deuxième if est inutile, il suffit de rajouter un &&… dans le premier if et effacer le deuxième :

```
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;
}
```

```php
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;
}
```

Error : and tags should be used

- “cgu” dossier templates > pages
- “basefront.html.twig” dossier templates

Cette erreur signifie que le code HTML utilise des balises qui ne sont pas conformes aux normes recommandées.
Exemple :
Remplecement des balises <i> par les balises dans “cgu” - Avant :

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

Après :

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

Error : image tags should have an "alt" attributes

- “accueil” dans templates > pages
- “formations” dans templates > pages
- “playlists” dans templates > pages
- “basefront” dans templates

Cette erreur signifie que les balises d'images ne contiennent pas l'attribut "alt" recommandé, qui fournit une description textuelle de l'image pour les utilisateurs qui ne peuvent pas la voir.

Exemple :
Ajout balise <alt> dans “accueil” - Avant :

```
<a href="{{ path('formations.showone', {id:formation.id}) }}">
<img src="{{ formation.picture }}"
    class="card-img-top" style="width:100%;height:auto;">
</a>
```

Après :

```
<img src="{{ formation.picture }}" alt="image logo de formation"
    class="card-img-top" style="width:100%;height:auto;">
</a>
```

Error : <table> tags should have a description

- “accueil” dans dossier templates > pages
- “formations” dans dossier templates > pages
- “playlists” dans dossier templates > pages

Cette erreur signifie que les balises de tableaux HTML ne contiennent pas de description appropriée pour améliorer l'accessibilité du contenu aux utilisateurs ayant des handicaps visuels ou cognitifs.

Exemple :
Ajout balise <caption> dans “accueil” - Avant :

```
<table class="table">
```

Après :

```
<table class="table">
    <caption></caption>
```

Tache 2 – Respecter les bonnes pratiques de codage

Mission 1 : Tâche 2 : respecter les bonnes pratiques de codage #2

Open yassixx opened this issue on Mar 22 · 0 comments

yassixx commented on Mar 22 — Owner

Dans le respect des bonnes pratiques de codage, en particulier SOLID (ici, le S : "Single responsability"), modifier les méthodes de FormationRepository et PlaylistRepository qui contiennent des tests sur $table : à chaque fois, 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.

Modification des méthodes du “FormationRepository”

Cela est nécessaire car la méthode contient un test sur le paramètre $table. Donc je supprime le test pour le réaliser dans une nouvelle méthode.

Je modifie “findAllOrderBy” pour qu’elle prend uniquement comme paramètre le $champ et l’ $ordre et pas la table et va permettre le tri dans la table “formation” :

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

Je crée une méthode “findAllOrderByTable”qui prend comme paramètre le $champ, l’ $ordre et la $table et va permettre le tri dans la table spécifié dans le paramètre :

```
public function findAllOrderByTable($champ, $ordre, $table = ""): array {
    return $this->createQueryBuilder('f')
                ->join('f.' . $table, 't')
                ->orderBy('t.' . $champ, $ordre)
                ->getQuery()
                ->getResult();
}
```

Selon le meme mode operatoire je vais modifier “findByContainValue” et créer “findByContainValueTable” :

```
public function findByContainValue($champ, $valeur): array {
    if ($valeur == "") {
        return $this->findAll();
    }
    return $this->createQueryBuilder('f')
                ->where('f.' . $champ . ' LIKE :valeur')
                ->orderBy(F_PUBLISHED_AT, 'DESC')
                ->setParameter('valeur', '%' . $valeur . '%')
                ->getQuery()
                ->getResult();
}
```

```
public function findByContainValueTable($champ, $valeur, $table = ""): array
    if ($valeur == "") {
        return $this->findAll();
    }
    return $this->createQueryBuilder('f')
                ->join('f.' . $table, 't')
                ->where('t.' . $champ . ' LIKE :valeur')
                ->orderBy(F_PUBLISHED_AT, 'DESC')
                ->setParameter('valeur', '%' . $valeur . '%')
                ->getQuery()
                ->getResult();
}
```

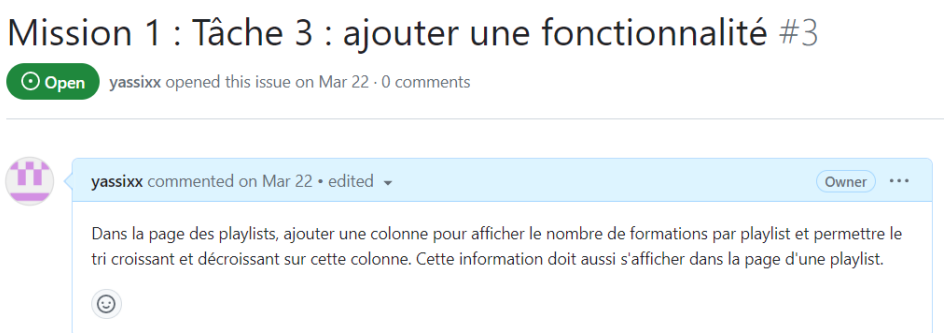
Les séparations en deux ont causé des erreurs de tris dans l’ordre croissant et décroissant. Cette erreur provient de la fonction "formations.sort" dans le fichier "FormationsController", car elle ne prend pas en compte la variable "$table" qui est supprimée et l'appel de la méthode "findAllOrderByTable". Pour résoudre ce problème, une condition "if" doit être instaurée pour appeler la fonction "findAllOrderByTable" si une table est affectée à $table et la méthode "findAllOrderBy" si la variable $table n'est pas assignée. De même, l'erreur de filtre sur les playlists provient de la fonction

"formations.findallcontain" car elle ne prend pas en compte la variable "$table". Pour résoudre ce problème, une condition doit être ajoutée pour appeler la méthode "findByContainValueTable" si une table est affectée à $table et la méthode "findByContainValue" si la variable $table n'est pas assignée.

Modification des méthodes du “FormationRepository”

Le contexte est similaire au précédent, il y a deux méthodes qui permettent de trier les enregistrements selon un champ et une valeur et qui peuvent prendre $table en option pour trier sur une table particulière. La première méthode "findByContainValue" ne doit pas prendre $table en paramètre et doit retourner tous les enregistrements triés par ordre ascendant. La seconde méthode "findByContainValueTable" prend $table en paramètre et permet de filtrer sur une table spécifique. Pour utiliser ces méthodes, il faut corriger la fonction "findAllContain" du "PlaylistsController", qui est appelée lors de la recherche de playlists ou de catégories de la page "playlists.html.twig". Une condition "if" est ajoutée pour appeler la méthode "findByContainValueTable" si $table est assigné, sinon la méthode "findByContainValue" est appelée.

Tache 3 – Ajouter une fonctionnalité



Création de la méthode “findAllOrderByNbFormatios” dans PlaylistRepository

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

Cette fonction "findAllOrderByNbFormations" retourne toutes les playlists triées en fonction du nombre de formations associées, dans un ordre spécifié par le paramètre $ordre, en utilisant la méthode QueryBuilder de Doctrine.

Création de la fonction getCategoriesPlaylist dans l’entité Playlist

```php
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;
}
```

Permet de parcourir la table des formations et d’obtenir les categories de chaque formation.

Dans ce contexte, les "add" et "select" ne sont plus nécessaires dans les fonctions du PlaylistRepository et peuvent être supprimés. La méthode "findAllOrderBy" est renommée en "findAllOrderByName" et les paramètres $champ sont supprimés de toutes les fonctions "findAllOrderBy".

Modification de la fonction sort de PlaylistsController

- Cela est nécessaire pour qu’elle puisse appeler les deux méthodes "findAllOrderByName" et "findAllOrderByNbFormations" du PlaylistRepository. Pour déterminer le champ à trier, un "switch case" est utilisé pour appeler la méthode appropriée en fonction du choix de l'utilisateur entre le tri selon le nom ou selon le nombre de formations :

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

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

Gestion de la vue

- La méthode "playlists.sort" doit être appelée dans la vue "playlists.html.twig" pour gérer le tri ascendant et descendant de la colonne du nombre de formations par playlist. Pour afficher le nombre de formations dans chaque playlist individuelle, la commande "playlistformations|length" est ajoutée dans la page "playlist.html.twig" avec un titre "nombre de formations".

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

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

- Dans la page "playlists.html.twig", on affiche un tableau avec les colonnes correspondant au titre des playlists, les catégories associées, le nombre de formations par playlist et le bouton "voir détail" pour chaque playlist. On modifie le code initial de la boucle "for" en utilisant les balises <td>, on teste ensuite que le nombre de formations s'affiche correctement et que le tri par ordre croissant ou décroissant fonctionne en exécutant l'application.

```
{% set ancplaylist = '' %}
{% for k in 0..playlists|length-1 %}
    <!-- affichage une fois chaque playlist -->
        <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 %}
                     {{ categories[c] }}
                {% endfor %}
            {% endif %}
        </td>
        <td class="text-center">
            {{ playlists[k].formations|length }}
        </td>
        <td class="text-center">
            <a href="{{ path('playlists.showone', {id:playlists[k].id}) }}" class="btn btn-secondary">Voir détail</a>
        </td>
    </tr>
```

playlist

filtrer

catégories

Nombre de formation

Détail

Bases de la programmation (C#)	C# POO	74	Voir détail
Compléments Android (programmation mobile)	Android	13	Voir détail
Cours Composant logiciel	Cours	2	Voir détail
Cours Curseurs	SQL Cours POO	2	Voir détail

Mission 2 - Coder la partie back-office :

Tache 1 – Gérer les formations

Mission 2 : Tâche 1 : gérer les formations #4

Open yassixx opened this issue on Mar 23 · 0 comments

yassixx commented on Mar 23 — Owner

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.

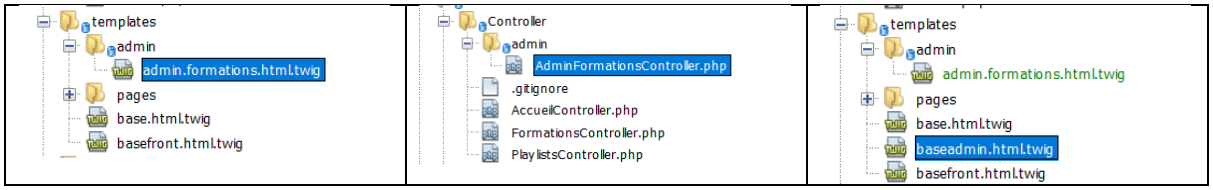
Les mêmes tris et filtres présents dans le front office doivent être présents dans le back office.

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.

Création de la page

Afin de créer une page de gestion des formations, je crée un dossier "admin" dans le dossier "templates" et une page "admin.formations.html.twig" ainsi qu'un dossier "admin" dans le dossier "controller" avec un fichier "AdminFormationsController" pour réaliser les routes vers la page. Ensuite, je crée un fichier "baseadmin.html.twig" dans le dossier "templates" qui contiendra la structure des pages d'administrations.



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

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

{% block title %}{% endblock %}
{% block stylesheets %}{% endblock %}
{% 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>
                    <li class="nav-item">
                        <a class="nav-link" href="#">Playlists</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">Catégories</a>
                    </li>
                </ul>
            </div>
        </nav>
    </div>
{% endblock %}
```

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

Je crée la "structure de base" de ma page d'administration des formations "admin.formations.html.twig", qui me permettra ultérieurement d'afficher un tableau des formations, playlists et catégories, et de les gérer en ajoutant, modifiant ou supprimant.

```
{% 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 %}
```

La classe "AdminFormationsController" du dossier "controller" est créée avec ses variables, son constructeur et l'index contenant la route vers la nouvelle page de l'application.

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

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

    /**
     * Création du constructeur
     * @param FormationRepository $formationRepository
     * @param CategorieRepository $categorieRepository
     */
    function __construct(FormationRepository $formationRepository, CategorieRepository $categorieRepository) {
        $this->formationRepository = $formationRepository;
        $this->categorieRepository = $categorieRepository;
    }

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

Affichage de la liste de formations

Dans la page "admin.formations.html.twig", j'affiche la liste des formations avec les informations suivantes : titre, playlists, catégories, date de création et image.

```twig
<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.categorie %}
                    {{ 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 reduite de formation">
                    </a>
                {% endif %}
            </td>
        </tr>
    {% endfor %}
{% endblock %}
```

Suppression et édition d'une formation

Pour permettre la suppression d'une formation, je créé une fonction "delete" dans "AdminFormationsController", qui utilise la méthode "remove" du "FormationsRepository", puis j'ajoute un bouton "supprimer" dans la page "admin.formations.html.twig" avec une demande de confirmation de suppression. Ensuite je vérifie le bon fonctionnement du bouton supprimer.

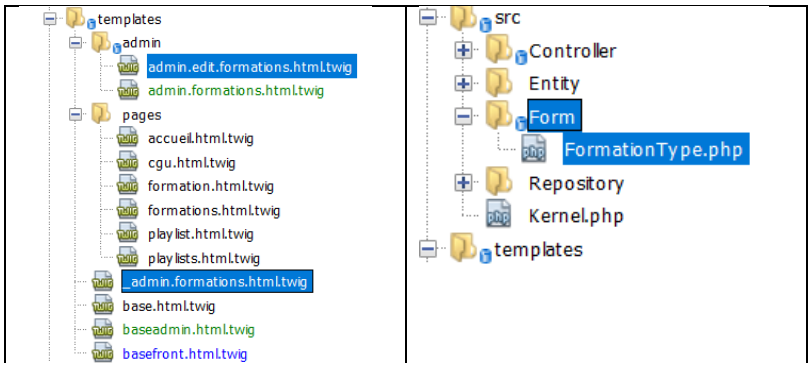
```php
/**
 * Suppression d'une formation
 * @Route("/admin/suppr.formation/{id}", name="admin.suppr.formation")
 * @param Formation $formations
 * @return Response
 */
public function suppr(Formation $formations): Response{
    $this->formationRepository->remove($formations, true);
    return $this->redirectToRoute('admin.formations');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

```twig
{% 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 reduite de formation">
                </a>
            {% endif %}
        </td>
        <td>
            <a href="{{ path('admin.edit.formations', {id:formation.id}) }}" class="btn btn-success"
            onclick="return confirm('Etes vous sur de vouloir supprimer {{ formation.title }}?')">Editer</a>
        </td>
        <td>
            <a href="{{ path('admin.suppr.formation', {id:formation.id}) }}" class="btn btn-danger"
            onclick="return confirm('Etes vous sur de vouloir supprimer {{ formation.title }}?')">Supprimer</a>
        </td>
    </tr>
{% endfor %}
```

Date de publication	Details	Action
09/07/2018		Supprimer
18/10/2018		Supprimer

Création du bouton d'édition d'une formation

Pour permettre la modification d'une formation, il faut créer un nouveau bouton "éditer" qui va rediriger vers une nouvelle page "_admin.formations.form.html.twig" contenant le formulaire d'édition, qui sera appelé dans la page "admin.edit.formation.html.twig". La création du formulaire nécessite également la création d'une nouvelle classe php "FormationType" dans le dossier "Form" du "src", qui va définir les champs du formulaire tels que le titre et la description de la formation, les catégories associées, la date de création, la playlist associée, et un bouton "submit" pour enregistrer les informations dans la base de données.




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

{% block body %}
    {{ include('_admin.formations.html.twig')}} }}
{% endblock %}
```

La création du formulaire "_admin.formations.form.html.twig" pour l'ajout ou l'édition d'une formation nécessite la création d'une nouvelle classe "FormationType" dans le dossier "Form", permettant de définir les champs du formulaire, tels que le titre et la description de la formation, les catégories associées, la date de création et la sélection d'une playlist.

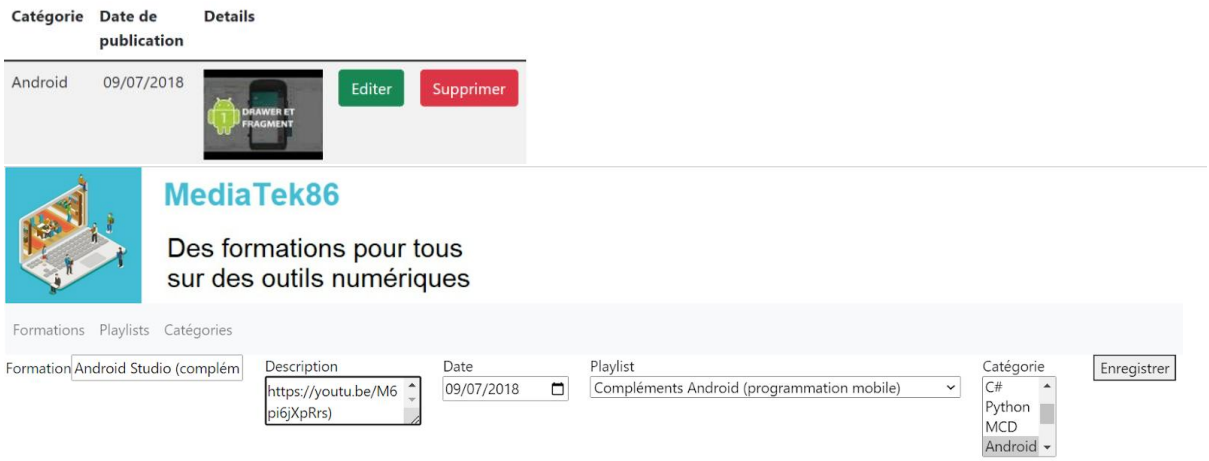
```php
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
            ->add('title', TextType::class, [
                'label' => 'Formation',
                'required' => true])
            ->add('description', TextareaType::class, [
                'label' => 'Description',
                'required' => false
            ])
            ->add('playlist', EntityType::class, [
                'class' => Playlist::class,
                'label' => 'Playlist',
                'choice_label' => 'name',
                'multiple' => false,
                'required' => true
            ])
            ->add('categories', EntityType::class, [
            'class' => Categorie::class,
            'label' => 'Catégorie',
            'choice_label' => 'name',
            'multiple' => true,
            'required' => false
            ])
            ->add('publishedAt', DateType::class, [
                'widget' => 'single_text',
                'data' => isset($options['data']) && $options['data']->getPublishedAt()
                    != null ? $options['data']->getPublishedAt() : new DateTime('now'),
                'label' => 'Date',
                'required' => true
            ])
            ->add('submit', SubmitType::class, [
            'label' => 'Enregistrer'
            ]);
}
```

La fonction "éditer" dans le fichier "AdminFormationsController" utilise la classe "FormationType" pour créer le formulaire d'édition de formation et récupère les informations de la formation. La méthode "$formFormation->handleRequest($request)" gère la soumission du formulaire et la condition "if" vérifie si le formulaire est soumis et valide, puis enregistre les informations dans la base de données, sinon maintient l'utilisateur sur la page d'édition.

```php
/**
 * Edition d'une formation
 * @Route("/admin/edit/{id}", name="admin.edit.formations")
 * @param Formation $formations
 * @param Request $request
 * @return Response
 */
public 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()
    ]);
}
```

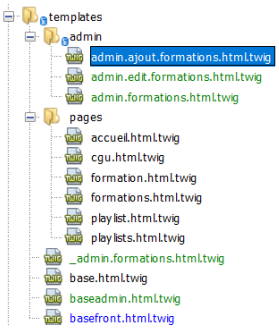
Pour permettre la modification d'une formation, j'ai créé un bouton "éditer" dans la page "admin.formations.html.twig", avec un lien vers une nouvelle page "admin.edit.formations.html.twig" contenant le formulaire d'édition. Ensuite, j'ai créé une fonction dans "AdminFormationsController" qui crée et gère ce formulaire, en utilisant la classe "FormationType", et qui enregistre les informations dans la base de données lors de la validation.

```
{{ 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) }}
```



Ajouter une formation

En utilisant la fonctionnalité d'édition de formation existante, je peux simplement créer une nouvelle fonction d'ajout dans AdminFormationsController et ajouter le formulaire dans la page admin.ajout.formations.html.twig, puis ajouter un bouton et un chemin d'accès pour cette fonctionnalité dans la page admin.formations.html.twig.



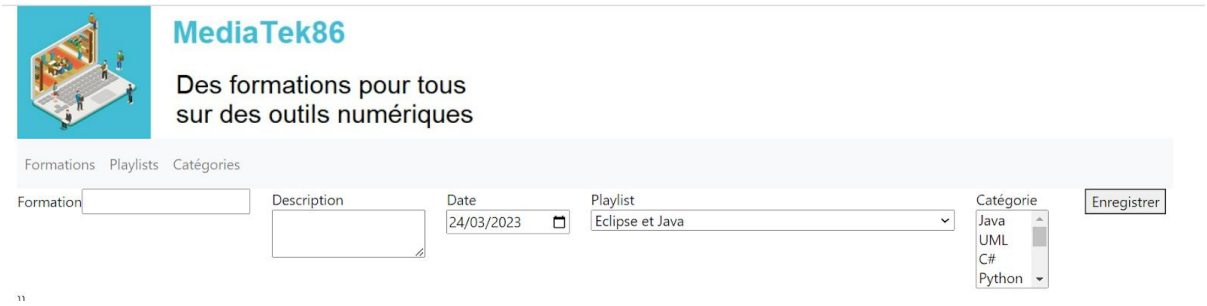
Dans le "AdminFormationsController", je crée une fonction d'ajout qui récupère le même formulaire que celui de la fonction d'édition, et j'ajoute la commande "$formations = new Formation();" pour

affecter une nouvelle formation à la table des formations, en modifiant également les routes pour effectuer les redirections vers la page "admin.ajout.formations.html.twig".

```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()
    ]);
}
```

Ajout du bouton "Ajouter une formation" avec la redirection vers la fonction correspondante dans la page "admin.formations.html.twig", positionné en haut à droite grâce à la balise <p class="text-end">, suivi de la vérification de son affichage et du bon fonctionnement de l'ajout d'une nouvelle formation.

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

{% block body %}
    <p class="text-end">
        <a href="{{ path('admin.ajout.formations') }}" class="btn btn-primary">
            Ajouter une formation
        </a>
    </p>
    <table class="table table-striped">
```



Tris ascendants et descendants

Pour trier les playlists en ordre croissant ou décroissant selon leur nom ou le nombre de formations, il est nécessaire de créer une fonction "admin.playlists.sort" appelée sur le clic des boutons de tri dans la page "admin.playlists.html.twig". Cette fonction doit appeler les méthodes "findAllOrderByName" et "findAllOrderByNbFormations" du "PlaylistRepository". Elle doit également recevoir deux paramètres ($champ et $ordre) pour trier le tableau de playlists. Si le $champ est égal à "name", la méthode triera les playlists en prenant en compte leur nom, sinon si $champ est égal à "nbformations", la méthode triera les playlists selon le nombre de formations. Finalement, la fonction récupère la liste des catégories et redirige l'utilisateur vers la page des playlists lorsqu'il clique sur les boutons de tri.

```php
/**
 * Retourne toutes les formations triées sur un champ
 * Et sur un champ si autre table
 * @Route("/admin/formations/tri/{champ}/{ordre}/{table}", name="admin.formations.sort")
 * @param type $champ
 * @param type $ordre
 * @param type $table
 * @return Response
 */
public function sort($champ, $ordre, $table=""): Response{
    if($table != ""){
        $formations = $this->formationRepository->findAllOrderByTable($champ, $ordre, $table);
    }else{
        $formations = $this->formationRepository->findAllOrderBy($champ, $ordre);
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render('admin/admin.formations.html.twig', [
        'formations' => $formations,
        'categories' => $categories
    ]);
}
```

Il faut appeler la méthode de tri dans la page "admin.formations.html.twig" en ajoutant le chemin "admin.formations.sort" aux boutons "<" et ">", et en définissant le tri en insérant "ASC" ou "DESC" au paramètre $ordre selon le bouton cliqué, et "title" comme $champ pour trier les formations par leur titre.

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

Je réalise un tri ascendant et descendant sur les playlists, ainsi que sur la date de publication, en modifiant le $champ et $table.

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

Je vérifie ensuite le bon fonctionnement des tris dans les pages correspondantes.

Filtres

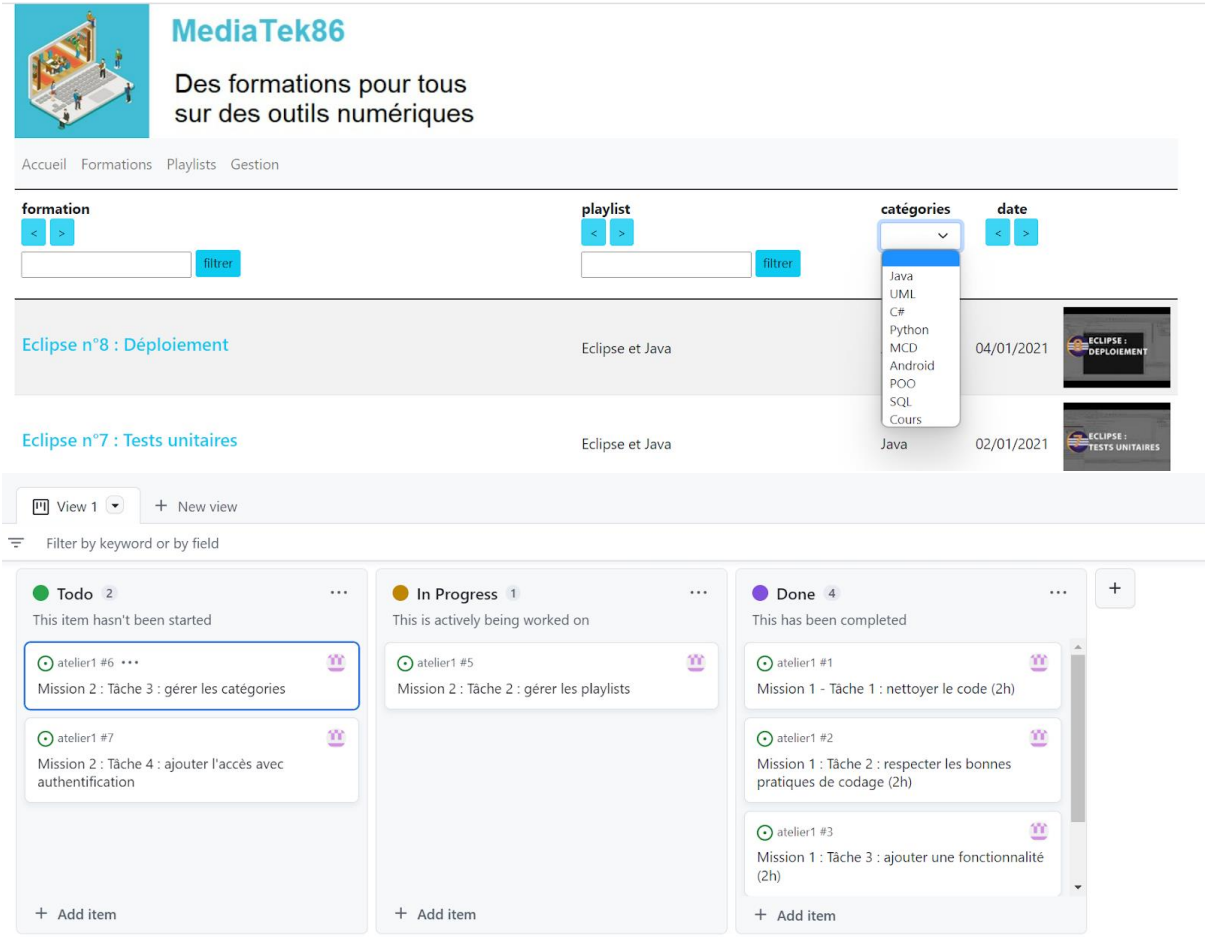
Je dois créer des formulaires de recherche dans la page "admin.playlists.html.twig" pour filtrer les playlists et les catégories selon une valeur saisie dans le formulaire. Cela nécessite la création d'une

fonction "findAllContain" dans le "AdminPlaylistsController" qui récupère la valeur de la zone de recherche, la compare aux $champs des différentes $tables, et appelle la méthode "findByContainValueTable" ou "findByContainValue" du "PlaylistRepository" en fonction des paramètres passés. La fonction récupère également la liste de toutes les catégories.

```php
/**
 * @Route("/admin/formations/recherche/{champ}/{table}", name="admin.formations.findallcontain")
 * @param type $champ
 * @param Request $request
 * @param type $table
 * @return Response
 */
public function findAllContain($champ, Request $request, $table=""): Response{
    $valeur = $request->get("recherche");
    if($table !=""){
        $formations = $this->formationRepository->findByContainValueTable($champ, $valeur, $table);
    }else{
        $formations = $this->formationRepository->findByContainValue($champ, $valeur);
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render('admin/admin.formations.html.twig', [
        'formations' => $formations,
        'categories' => $categories,
        'valeur' => $valeur,
        'table' => $table
    ]);
}
```

Pour permettre la recherche des enregistrements dans la page d'administration des formations, je dois créer un formulaire de recherche en utilisant les balises <form>, et y insérer la méthode "POST" ainsi que le chemin vers la fonction "admin.formations.findallcontain". Ensuite, j'ajoute un input qui permet la saisie de la valeur de recherche, et un second input pour créer un token "csrf_token" afin de masquer le champ. Enfin, je dois affecter un $champ et une $table si nécessaire pour que la fonction puisse rechercher les enregistrements correspondants.

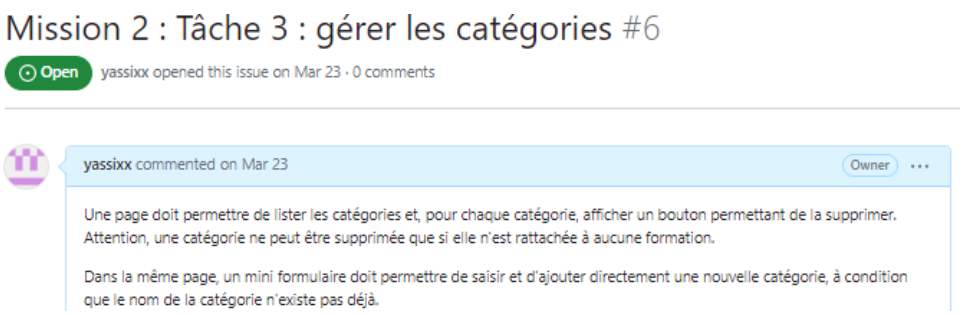
```twig
Formation<br />
<a href="{{ path('admin.formations.sort', {champ:'title', ordre:'ASC'}) }}"
   class="btn btn-info btn-sm active" role="button" aria-pressed="true"><</a>
<a href="{{ path('admin.formations.sort', {champ:'title', ordre:'DESC'}) }}"
   class="btn btn-info btn-sm active" role="button" aria-pressed="true">></a>
   <form class="form-inline mt-1" method="POST" action="{{ path('admin.formations.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>
Catégorie<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>
```



Tache 2 – Gérer les playlists

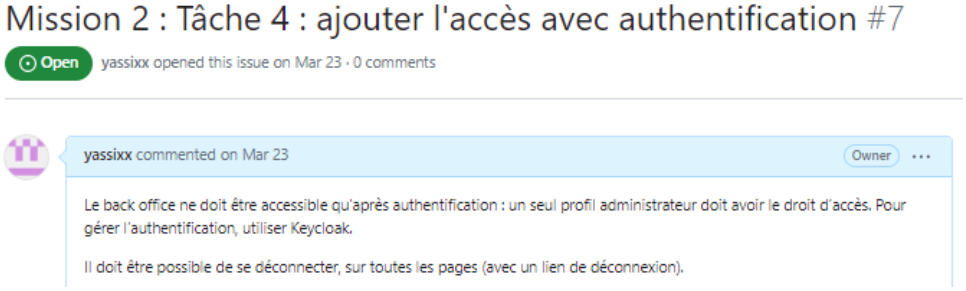
Pour cette tâche, j'ai suivi une approche similaire à celle de la tâche précédente qui consistait à gérer les formations.

Tache 3 – Gérer les catégories



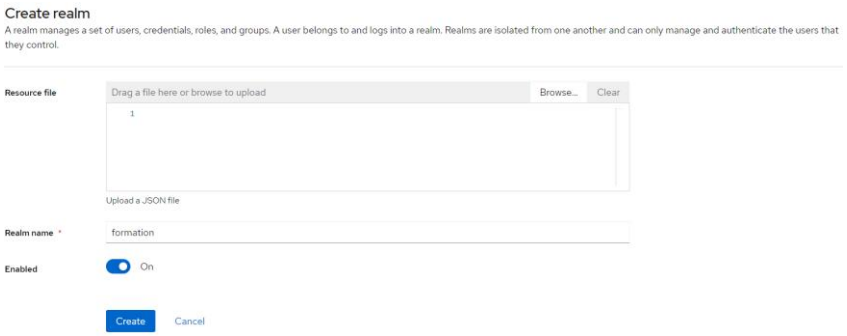
Pour cette tâche, j'ai suivi une approche similaire à celle de la tâche précédente qui consistait à gérer les formations.

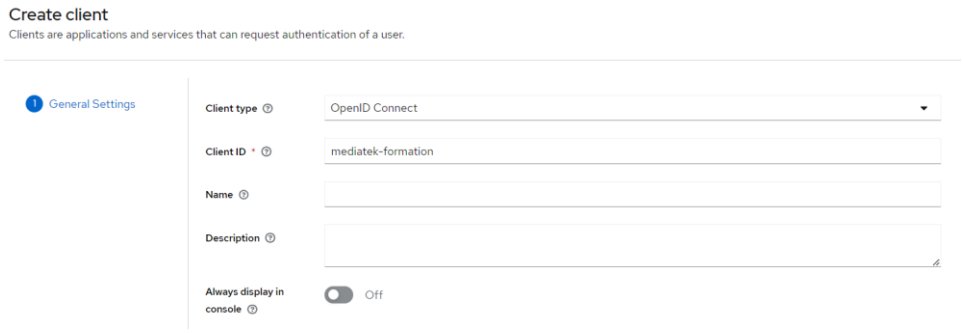
Tache 4 – Ajouter l'accès avec authentification

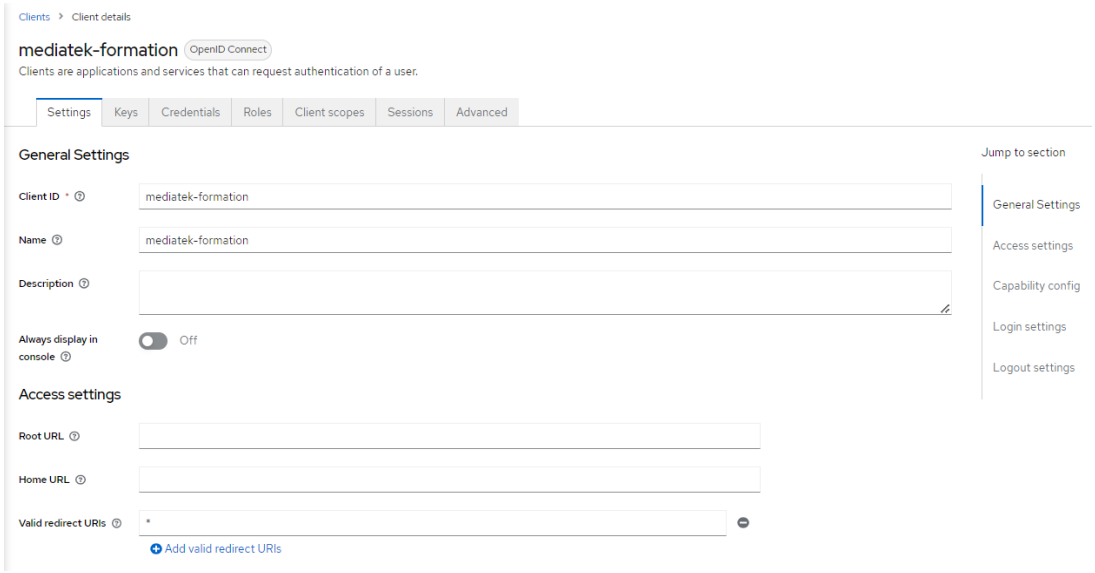


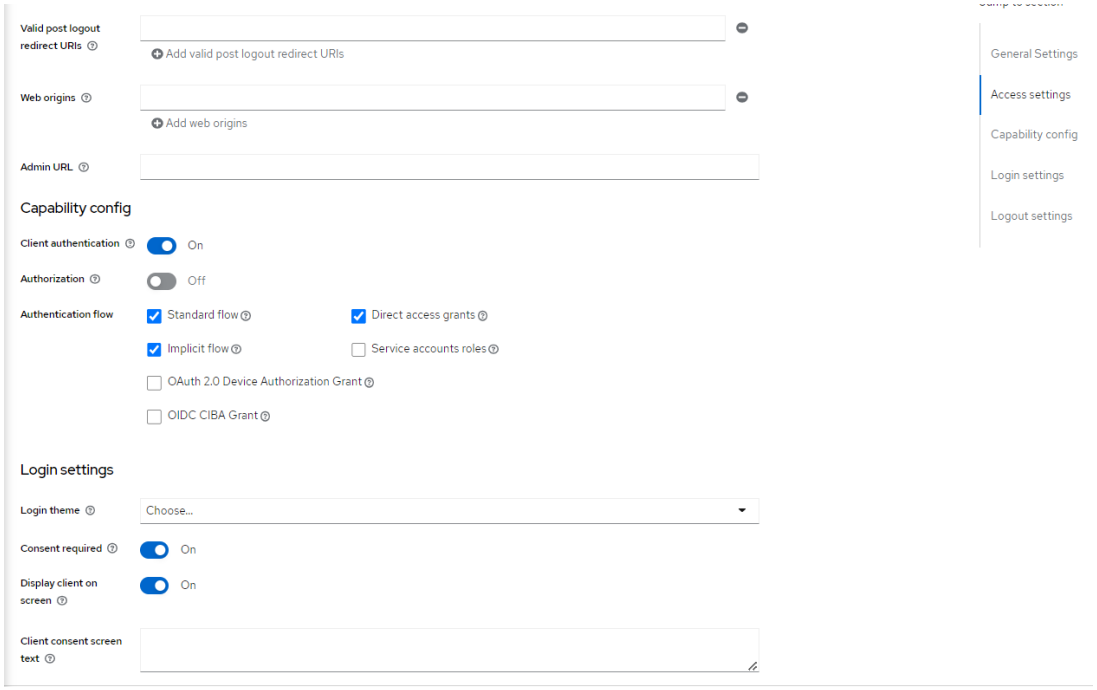
Configuration de Keycloak

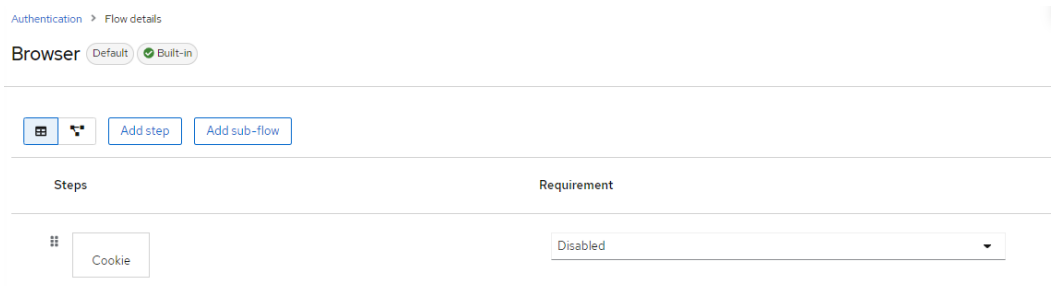
Je configure Keycloak en lançant le serveur à partir du dossier "C:/keycloak/bin" avec la commande "kc.bat start-dev", j'accède à la console d'administration du serveur via la page "http://localhost:8080", je crée un nouveau royaume nommé "formation", un client nommé "mediatek-formation" et je configure les paramètres dans l'onglet "settings", puis je désactive les cookies dans l'onglet "authenticate > browser > cookies".


Create realm
A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.
Resource file
Drag a file here or browse to upload
Browse...
Clear
Upload a JSON file
Realm name *
formation
Enabled
On
Create
Cancel

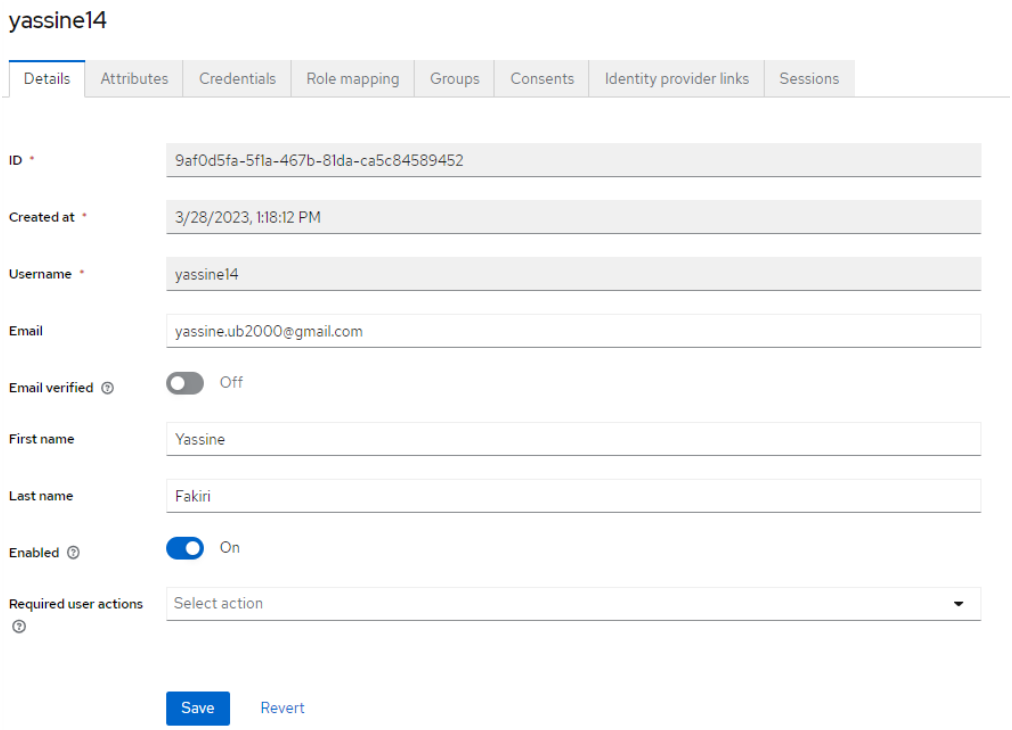

Create client
Clients are applications and services that can request authentication of a user.
1 General Settings
Client type
OpenID Connect
Client ID *
mediatek-formation
Name
Description
Always display in console
Off


Clients > Client details
mediatek-formation
OpenID Connect
Clients are applications and services that can request authentication of a user.
Settings
Keys
Credentials
Roles
Client scopes
Sessions
Advanced
General Settings
Client ID *
mediatek-formation
Name
mediatek-formation
Description
Always display in console
Off
Access settings
Root URL
Home URL
Valid redirect URIs
*
Add valid redirect URIs
Jump to section
General Settings
Access settings
Capability config
Login settings
Logout settings





Je récupère le client-secret dans l'onglet credentials et je poursuis en créent un user avec tous les droits.



Configuration du projet symfony

Dans Netbeans, j'ouvre le fichier ".env" et j'ajoute à la fin 3 lignes pour définir les valeurs de mon client "mediatek-formation" créé sur le serveur Keycloak, puis j'enregistre le fichier.

```
KEYCLOAK_SECRET=ZUMqU
KEYCLOAK_CLIENTID=med
KEYCLOAK_APP_URL=http
```

Pour permettre à l'utilisateur keycloak d'être enregistré dans la base de données du site dès sa connexion, j'ai créé la classe "user" et la table correspondante en utilisant les commandes "php bin/console make:user" et "php bin/console make:entity User", puis en modifiant les paramètres. Ensuite, j'ai créé le fichier de migration en tapant "php bin/console make:migration", puis j'ai lancé la migration en tapant "php bin/console doctrine:migrations:migrate". Pour relier Symfony et Keycloak, j'ai installé les bundles "knpuniversity/oauth2-client-bundle 2.10" et "stevenmaguire/oauth2-keycloak 3.1 - with-all-dependencies", puis j'ai configuré le fichier "knpu_oauth2_client.yaml" avec les paramètres nécessaires. Des captures d'écran du terminal ont également été prises pour faciliter la compréhension de la procédure.

```
PS D:\wamp64\www\atelier1> php bin/console make:entity User

created: src/Entity/User.php
created: src/Repository/UserRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

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

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

Field length [255]:
> 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

PS D:\wamp64\www\atelier1> php bin/console make:migration


 Success!
```

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

Je configure le firewall en ajoutant la route de redirection "oauth_login" dans "security.yaml", ainsi que le chemin nécessitant une authentification et le type de rôle requis. Ensuite, j'utilise la commande "php bin/console make:controller OAuthController --no-template" pour créer un fichier de contrôleur qui gère les routes liées à l'authentification et contient les méthodes "index" et "connectCheckAction".

```php
<?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)
    {

    }
```

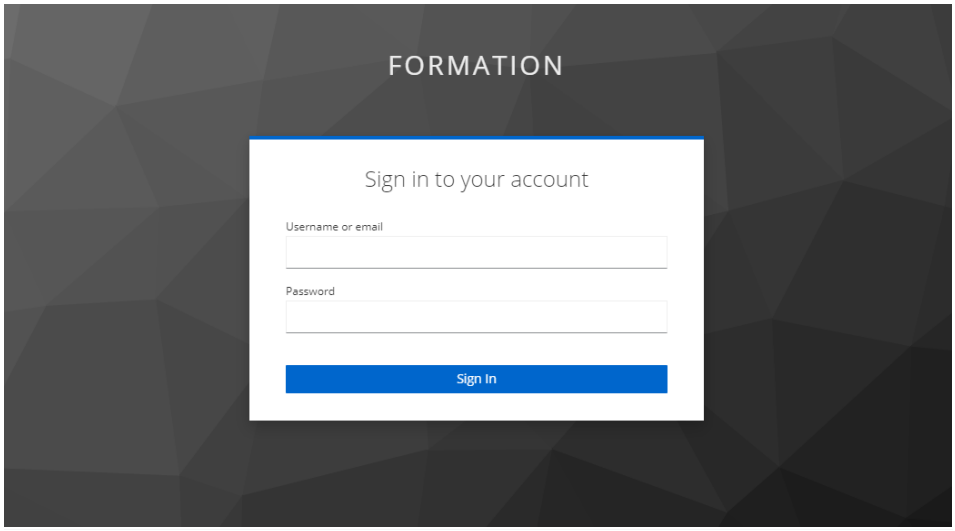
Pour gérer l'authentification de l'utilisateur, je crée une classe "KeycloakAuthenticator.php" qui implémente les méthodes nécessaires pour enregistrer l'utilisateur dans la BDD et gérer la connexion et la déconnexion. Ensuite, je configure le firewall dans "security.yaml" en ajoutant la route de redirection et le chemin nécessitant une authentification. Pour tester l'accès sécurisé, je crée un lien de déconnexion dans le fichier "baseadmin.html.twig", et j'ajoute la méthode "logout" dans le contrôleur "OAuthController". Enfin, je lance l'application, je m'authentifie et j'accède aux pages d'administration avec un lien de déconnexion.

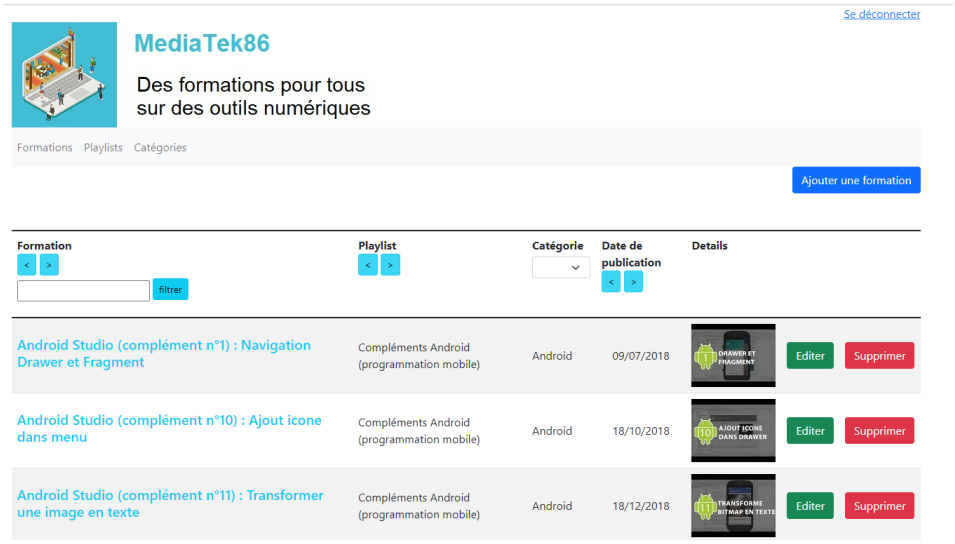
```php
/**
 * Description of KeycloakAuthenticator
 * gère les routes liées à l'authentification
 *
 * @author intad
 */
class KeycloakAuthenticator extends OAuth2Authenticator implements AuthenticationEntryPointInterface {

    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 start(Request $request, AuthenticationException $authException = null): Response {
    return new RedirectResponse(
            '/oauth/login',
            Response::HTTP_TEMPORARY_REDIRECT
    );
}
public function supports(Request $request): ?bool {
    return $request->attributes->get('_route') === 'oauth_check';
}
```

```php
public function authenticate(Request $request): Passport {
    $client = $this->clientRegistry->getClient('keycloak');
    $accessToken = $this->fetchAccessToken($client);
    return new SelfValidatingPassport(
            new UserBadge($accessToken->getToken(), function() use ($accessToken, $client) {
                /** @var KeycloakUser $keycloakUser */
                $keycloakUser = $client->fetchUserFromToken($accessToken);
                // 1) recherche du user dans la BDD à partir de son id Keycloak
                $existingUser = $this->entityManager
                        ->getRepository(User::class)
                        ->findOneBy(['keycloakId' => $keycloakUser->getId()]);
                if($existingUser){
                    return $existingUser;
                }
                // 2) Le user existe mais jamais connecté à Keycloak
                $email = $keycloakUser->getEmail();
                /** @var User $userInDatabase */
                $userInDatabase = $this->entityManager
                        ->getRepository(User::class)
                        ->findOneBy(['email' => $email]);
                if($userInDatabase){
                    $userInDatabase->setKeycloakId($keycloakUser->getId());
                    $this->entityManager->persist($userInDatabase);
                    $this->entityManager->flush();
                    return $userInDatabase;
                }
                // 3) Le user n'existe pas encore dans la DB
                $user = new User();
                $user->setKeycloakID(($keycloakUser->getId()));
                $user->setEmail($keycloakUser->getEmail());
                $user->setPassword("");
                $user->setRoles(['ROLE_ADMIN']);
                $this->entityManager->persist($user);
                $this->entityManager->flush();
                return $user;
            })
    );
```

```php
public function onAuthenticationFailure(Request $request, AuthenticationException $exception): ?Response {
    $message = strtr($exception->getMessageKey(), $exception->getMessageData());
    return new Response($message, Response::HTTP_FORBIDDEN);
}
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response {
    $targetUrl = $this->router->generate('admin.formations');
    return new RedirectResponse($targetUrl);
}
```

```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        lazy: true
        entry_point: form_login
        #provider: app_user_provider
        form_login:
            login_path: oauth_login
        custom_authenticators:
            - App\Security\KeycloakAuthenticator
        logout:
            path: logout
```





Mission 3 - Tester et documenter :

Tache 1 – Gérer les tests

Mission 3 : Tâche 1 : gérer les tests (7h) #8

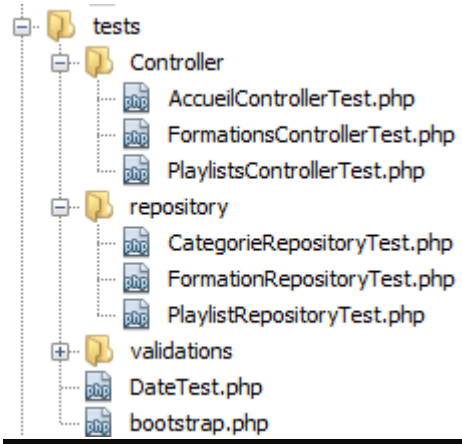
Open yassixx opened this issue on Mar 28 · 0 comments

yassixx commented on Mar 28 — Owner

Tests unitaires :
Contrôler le fonctionnement de la méthode qui retourne la date de parution au format string.
Tests d'intégration sur les règles de validation :
Lors de l'ajout ou de la modification d'une formation, contrôler que la date n'est pas postérieure à aujourd'hui.
Tests d'intégration sur les Repository :
Contrôler toutes les méthodes ajoutées dans les classes Repository (pour cela, créer une BDD de test).
Tests fonctionnels :
Contrôler que la page d'accueil est accessible.
Dans chaque page contenant des listes :
contrôler que les tris fonctionnent (en testant juste le résultat de la première ligne) ;
contrôler que les filtres fonctionnent (en testant le nombre de lignes obtenu et le résultat de la première ligne) ;
contrôler que le clic sur un lien (ou bouton) dans une liste permet d'accéder à la bonne page (en contrôlant l'accès à la page mais aussi le contenu d'un des éléments de la page).
Tests de compatibilité :
Créer un scénario avec Selenium, sur la partie front office, et le jouer sur plusieurs navigateurs pour tester la compatibilité du site.

Réalisation des tests unitaires, d'intégration, fonctionnels et de compatibilité :

J'ai utilisé Selenium, PHPUnit et ses classes TestCase, KernelTestCase, WebTestCase




```
D:\wamp64\www\atelier1>php bin/phpunit --filter CategorieRepositoryTest
PHPUnit 9.5.23 #StandWithUkraine

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

Time: 00:00.293, Memory: 28.00 MB

OK (5 tests, 7 assertions)
```

```
D:\wamp64\www\atelier1>php bin/phpunit --filter FormationRepositoryTest
PHPUnit 9.5.23 #StandWithUkraine

Testing
.........                                                           9 / 9
(100%)

Time: 00:00.353, Memory: 28.00 MB

OK (9 tests, 15 assertions)
```

```
D:\wamp64\www\atelier1>php bin/phpunit --filter PlaylistRepositoryTest
PHPUnit 9.5.23 #StandWithUkraine

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

Time: 00:00.309, Memory: 28.00 MB

OK (7 tests, 11 assertions)
```

```
D:\wamp64\www\atelier1>php bin/phpunit --filter AccueilControllerTest
PHPUnit 9.5.23 #StandWithUkraine

Testing
.                                                                   1 / 1
(100%)

Time: 00:00.499, Memory: 36.00 MB

OK (1 test, 1 assertion)
```

```
D:\wamp64\www\atelier1>php bin/phpunit --filter FormationsControllerTest
PHPUnit 9.5.23 #StandWithUkraine

Testing
........                                                            8 / 8
(100%)

Time: 00:01.297, Memory: 40.00 MB

OK (8 tests, 27 assertions)
```

```
C:\Windows\System32>selenium-side-runner -s http://10.7.178.49:4444 -c "browserName=chrome"
C:\Users\intad\Desktop\TestMediatekFormationdd.side
info: Running test Untitled
info: Building driver for chrome
info: Driver has been built for chrome
info: Finished test Untitled Success
 PASS  ../../Users/intad/AppData/Roaming/npm/node_modules/selenium-side-runner/dist/main.tes
t.js
  Running project TestMediatekFormation
    Running suite Default Suite
      √ Running test Untitled (1944 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        2.579 s, estimated 3 s
Ran all test suites within paths "C:\Users\intad\AppData\Roaming\npm\node_modules\selenium-s
ide-runner\dist\main.test.js".
```

Tache 2 – Créer la documentation technique

Mission 3 : Tâche 2 : créer la documentation technique #9

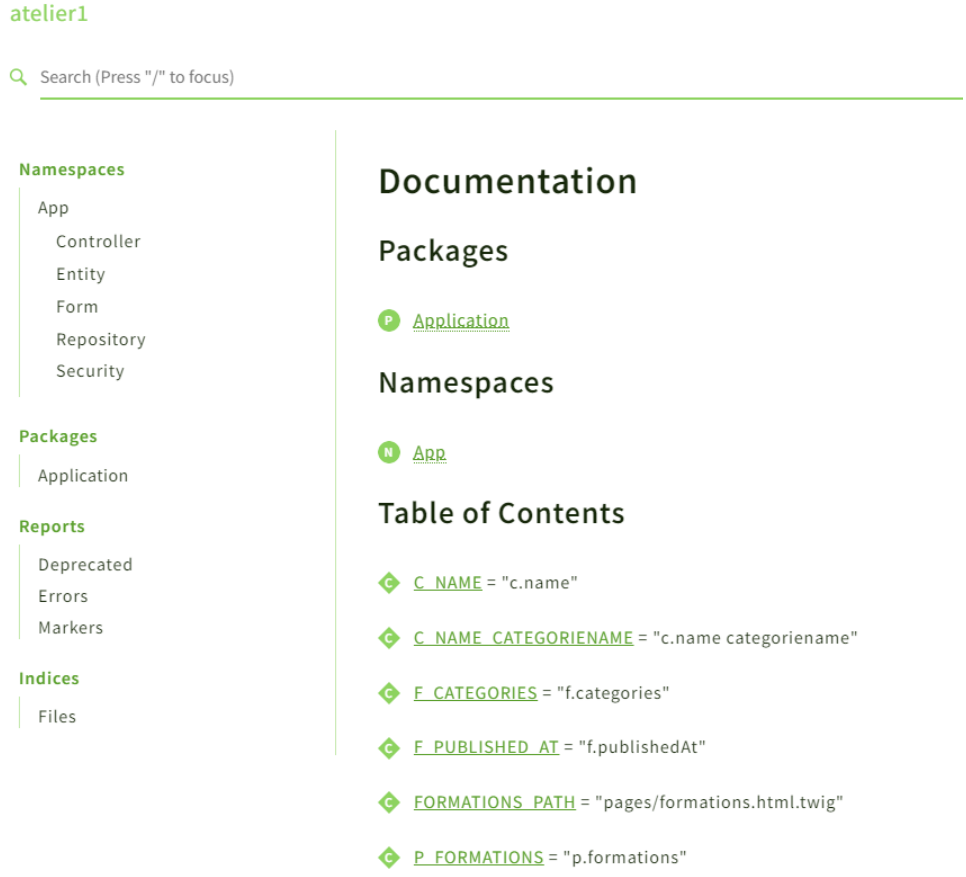
Open · yassixx opened this issue on Mar 28 · 0 comments

yassixx commented on Mar 28 — Owner

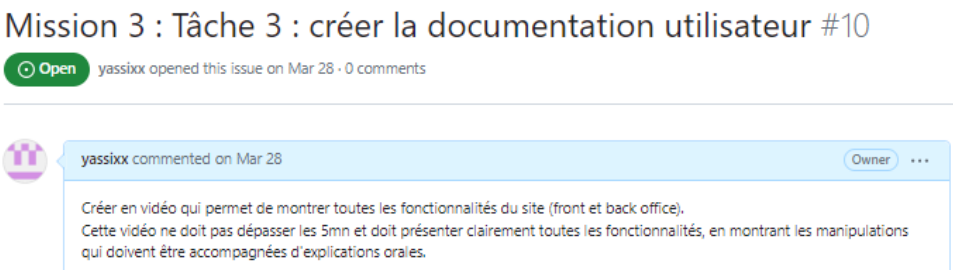
Contrôler que tous les commentaires normalisés nécessaires à la génération de la documentation technique ont été correctement insérés.
Générer la documentation technique du site complet : front et back office excluant le code automatiquement généré par Symfony (voir l'article "Génération de la documentation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

La documentation a été généré en utilisant le terminal en mode admin :

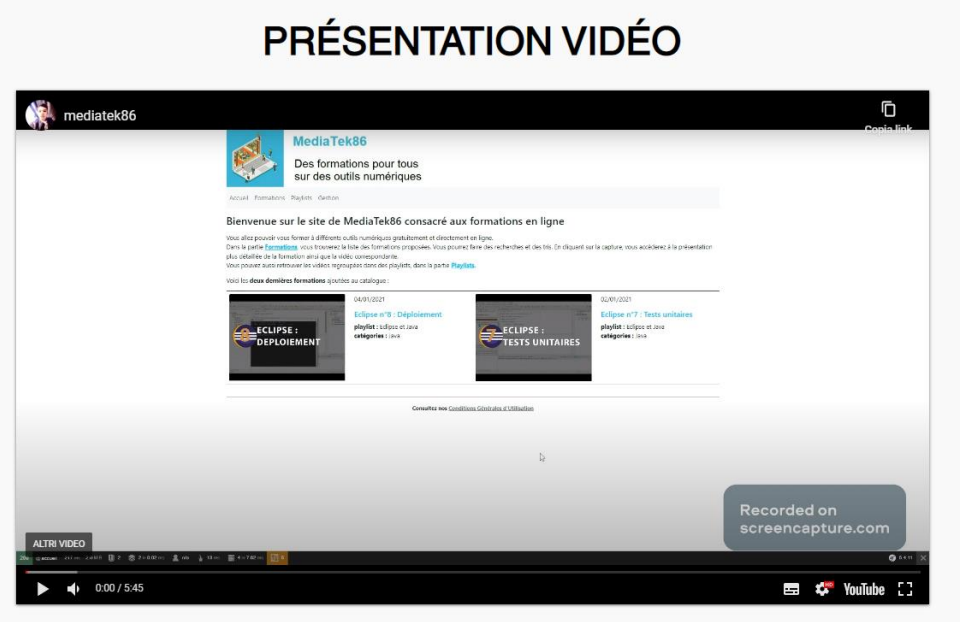
```
C:\Users\intad>"D:\wamp64\bin\php\php7.4.26\php.exe" "D:\wamp64\bin\php\php7.4.26\ext\phpDocumentor.phar" "run" "--ansi"
 "--directory" "D:\wamp64\www\atelier1\src" "--target" "D:\wamp64\www\atelier1_doc" --title "atelier1"
phpDocumentor v3.3.1

Parsing files
 19/19 [============================] 100%
Applying transformations (can take a while)
 21/21 [============================] 100%
```



Tache 3 – Créer la documentation utilisateur

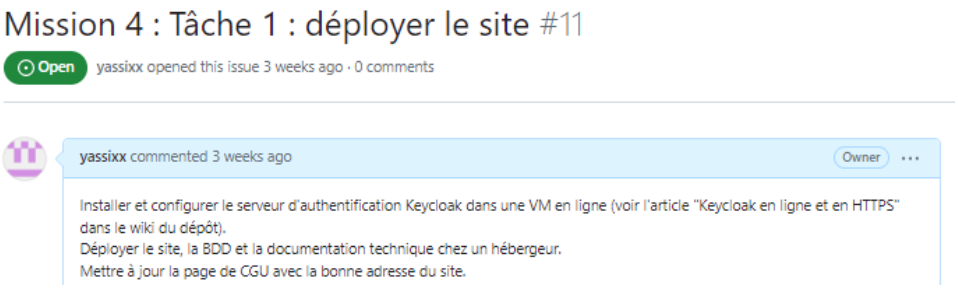


https://youtu.be/jUIsrXtZS6Y



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

Tache 1 – Déployer le site



Serveur Keycloak :

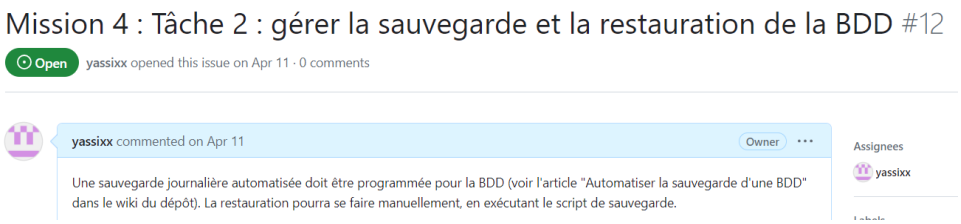
Pour mettre en ligne le serveur d'authentification Keycloak, j'ai créé une machine virtuelle Windows dans mon compte Azure et y ai installé Keycloak. J'ai également défini un nom DNS pour cette machine, ce qui m'a permis de me connecter à la VM depuis mon ordinateur. J'ai installé openJDK dans la VM, puis j'ai téléchargé Keycloak en version 19.0.1 depuis le site officiel. J'ai créé un compte admin, créé un royaume, configuré les paramètres et lancé la commande "kc.bat build". Ensuite, j'ai installé un certificat pour l'accessibilité de Keycloak en HTTPS en utilisant XAMPP et Certbot.

Déploiement de la BDD :

Pour déployer la base de données dans mon hébergeur Hostinger, j'ai exporté la BDD de mon application et copié le contenu du fichier. J'ai créé ma base de données et un utilisateur associé dans l'onglet "bases de données > phpmyadmin" de mon hébergeur, puis j'ai collé le contenu de mon fichier SQL dans l'onglet "sql" de phpmyadmin. Ensuite, j'ai attribué un site web à ma base de données.

Déploiement du site :

Le déploiement de l'application nécessite quelques configurations dans Netbeans. J'ai tapé la commande "composer require symfony/apache-pack" dans la fenêtre de commandes pour créer le fichier ".htaccess" dans le dossier "public". J'ai également modifié la variable "APP_ENV : dev" en "APP_ENV : prod" dans mon fichier ".env". Pour déployer le site sur mon hébergeur Hostinger, j'ai inséré le dossier du projet dans le "gestionnaire de fichier" de l'onglet "hébergement". J'ai extrait les fichiers de mon dossier .zip et lancé le site.



Automatisation de la sauvegarde :

Dans le gestionnaire de fichiers FTP de Hostinger, créez un dossier "savebdd" à la racine.

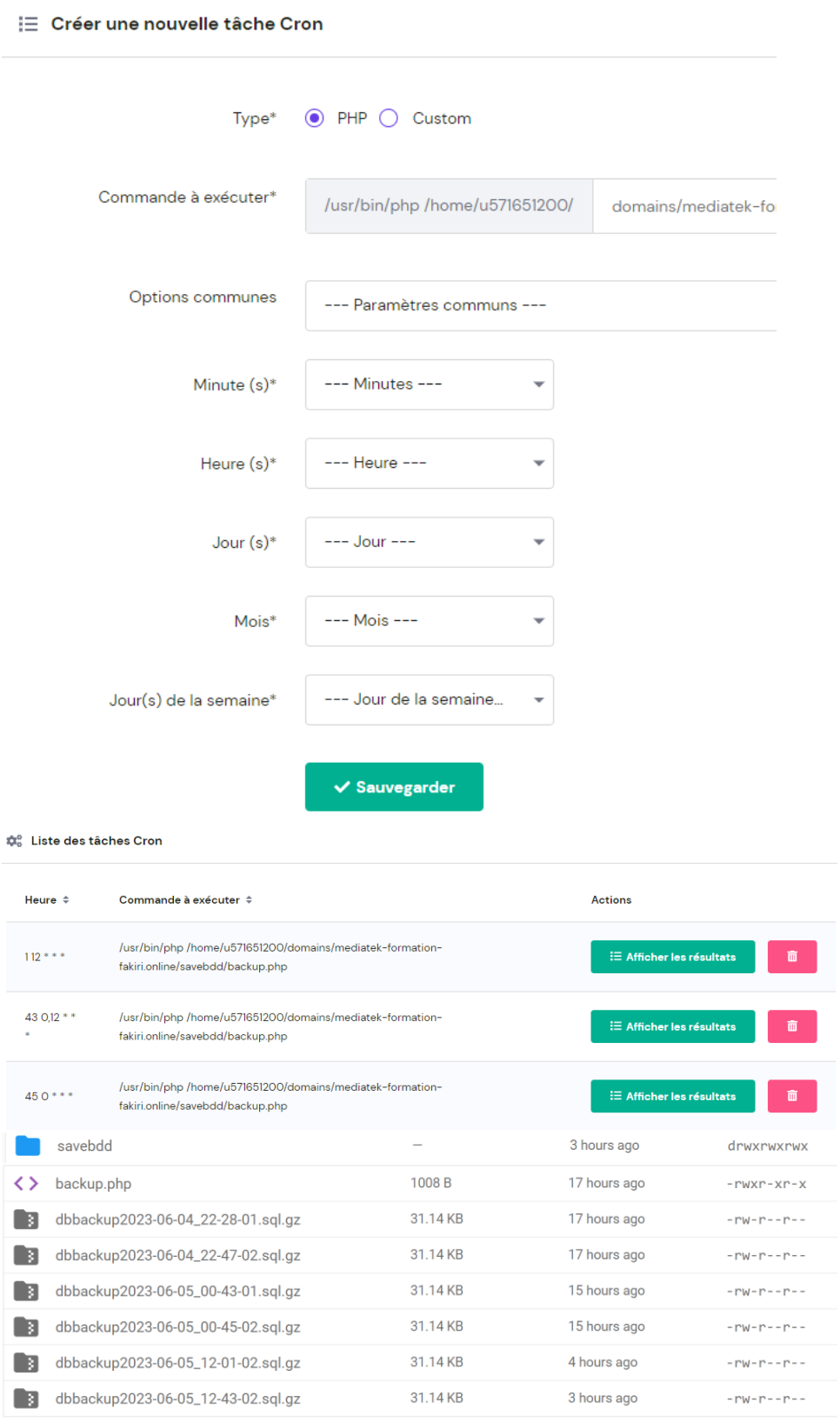
Dans ce dossier, créez un script "backup.php" contenant les instructions pour sauvegarder la base de données.

Ces instructions sauvegarderont la base de données dans un nouveau fichier chaque jour.

Dans l'onglet "tâches cron" de Hostinger, ajoutez une nouvelle tâche avec la commande /home/u571651200 /savebdd/backup.sh pour exécuter le script de sauvegarde.

Configurez cette tâche pour qu'elle s'exécute quotidiennement.

La sauvegarde sera enregistrée dans le dossier "savebdd" sous "public_html".



Restauration manuelle de la BDD :

Pour récupérer ma base de données, je dois me rendre dans mon dossier "savebdd" et télécharger le fichier "bddbackup_.sql.gz". Après avoir extrait les fichiers, j'obtiens la base de données. Si j'ai besoin de restaurer cette dernière, je peux simplement importer le script depuis mon fichier "bddbackup_.sql.gz" après avoir préalablement vidé ma base de données.

Cependant, concernant la restauration manuelle de la base de données, j'ai rencontré des problèmes avec Hostinger. Certaines actions requises pour la restauration nécessitent un privilège "SUPER", que je ne peux pas obtenir malgré tous les privilèges que j'ai activés sur Hostinger pour mon compte utilisateur. Je ne suis donc pas en mesure de récupérer la sélection des tables depuis mon fichier. Hostinger m'a proposé de souscrire à l'un de leurs plans pour résoudre ce problème, mais j'ai décidé de ne pas le faire.

DÉPLOYER LE SITE ET GÉRER LE DÉPLOIEMENT CONTINU :

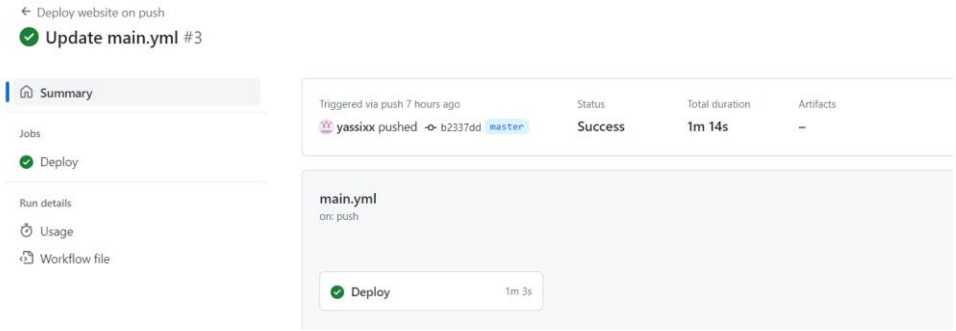
Pour déployer automatiquement l'application sur Github, j'implémente une série d'étapes visant à synchroniser les mises à jour du site avec chaque push sur GitHub. Cette synchronisation est possible en configurant GitHub pour diriger les fichiers vers le bon FTP.

La première étape consiste à créer un fichier d'automatisation "yml" dans le dépôt Github. Pour cela, je sélectionne "actions", puis je clique sur "set up a workflow yourself". Ensuite, je remplace le contenu par défaut par le script d'automatisation adapté, qui déclenche le déploiement du site web lors de chaque push.

Après avoir confirmé la création du fichier d'automatisation en cliquant sur "start" et "commit new file", un nouveau dossier ".github/workflows" est ajouté au dépôt. Ce dossier contient le fichier yml de déploiement que je viens de créer.

Dans Netbeans, j'actualise le dépôt local en utilisant l'option "Git > Remote > Pull > Next > Finish" pour récupérer le nouveau dossier. Ensuite, je stocke le mot de passe du FTP dans le dépôt Github en utilisant l'option "Settings > Secrets > Actions > New repository secret" à la racine du dépôt.

Avec le déploiement continu en place, je teste le système vérifier si le déploiement a fonctionné, je consulte l'onglet "actions" de mon dépôt Github.



Nom d'hôte actif	Répertoire	Nom d'utilisateur	
ftp.mediatek-mediatheque-fakiri.online	/home/u571651200/domains/mediatek-mediatheque-fakiri.online/public_html	u571651200.yassine	Changer le mot de passe

yassixx now (edited) Edit

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.
2h - 4h