

Compte-rendu du projet d'atelier n°1

Sommaire

Mission 1 : tâche n°1.....	2
Mission 1 : tâche n°2.....	9
Mission 2 : tâche n°1.....	13
Mission 2 : tâche n°2.....	21
Mission 2 : tâche n°3.....	29
Mission 2 : tâche n°4.....	34
Mission 3 : tâche n°1.....	40
Mission 3 : tâche n°2.....	43
Mission 3 : tâche n°3.....	45
Mission 4 : tâche n°1.....	47
Mission 4 : tâche n°2.....	50
Mission 4 : tâche n°3.....	53

Contexte :

En tant que développeur pour le compte de la société Mediatek86, il m'est demandé de faire évoluer une application selon un cahier des charges. Cette application est développée en PHP pour la partie backend, avec le cadre applicatif Symfony et exploite une base de données MySQL. Le frontend est codé à l'aide du langage de template TWIG et utilise Bootstrap en tant que feuille de style.

Mediatekformation est une plateforme web permettant la mise en ligne et la consultation de vidéos d'auto-formations.

Cette application se distingue en 2 grandes parties :

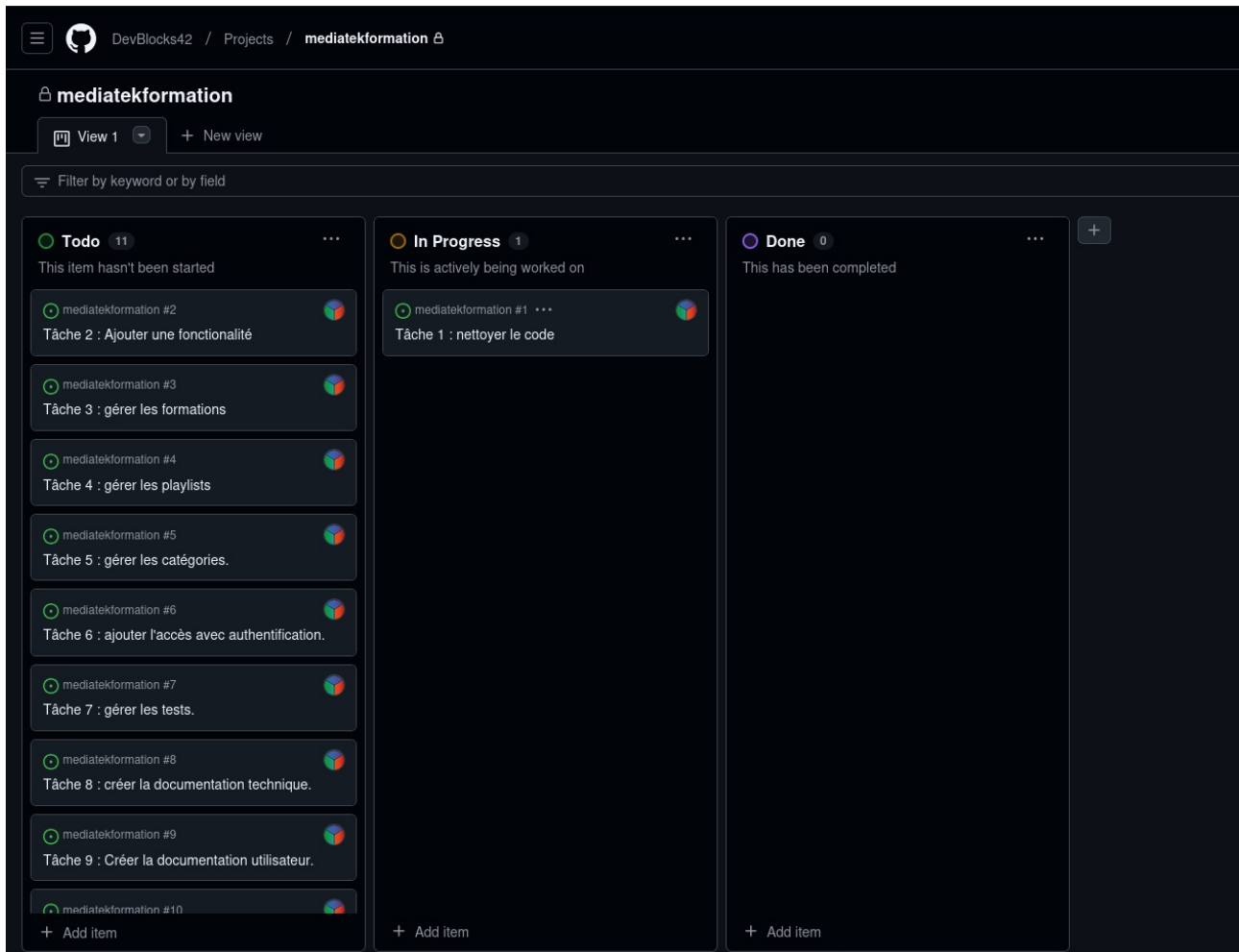
- la partie front-office destinée aux personnes souhaitant juste accéder aux formations.

- la partie back-office, destinée aux administrateurs et éditeurs de la plateforme pour gérer les formations

Mission 1 : tâche n°1

Intitulé : nettoyage du code

Description : Nettoyer le code en suivant les instructions de Sonarlint

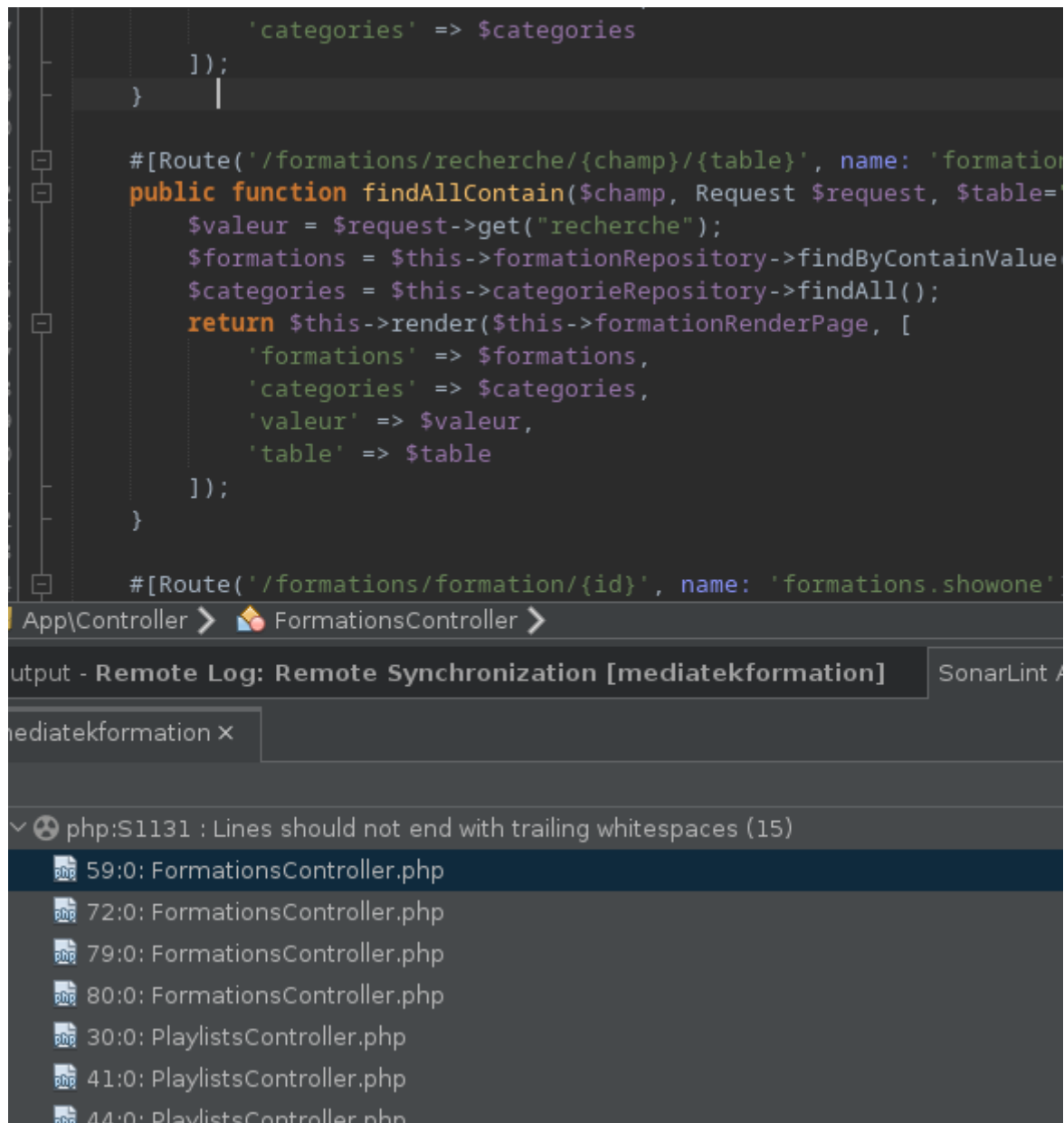


Aperçu du Kanban avec la tâche n°1 en cours de traitement

- Durée estimé : 2h

- Durée effective : 3h30

SonarLint : espaces inutiles



The screenshot shows a code editor with PHP code. The code includes a function `findAllContain` and a route definition. A SonarLint error message is displayed at the bottom: "php:S1131 : Lines should not end with trailing whitespaces (15)". The error is associated with several lines in the file `FormationsController.php` (lines 59, 72, 79, 80, 30, 41, and 44).

```
        'categories' => $categories
    });
}

#[Route('/formations/recherche/{champ}/{table}', name: 'formation
public function findAllContain($champ, Request $request, $table="")
{
    $valeur = $request->get("recherche");
    $formations = $this->formationRepository->findByContainValue($champ, $valeur);
    $categories = $this->categorieRepository->findAll();
    return $this->render($this->formationRenderPage, [
        'formations' => $formations,
        'categories' => $categories,
        'valeur' => $valeur,
        'table' => $table
    ]);
}

#[Route('/formations/formation/{id}', name: 'formations.showone')
App\Controller > FormationsController >
```

Output - Remote Log: Remote Synchronization [mediatekformation] SonarLint A

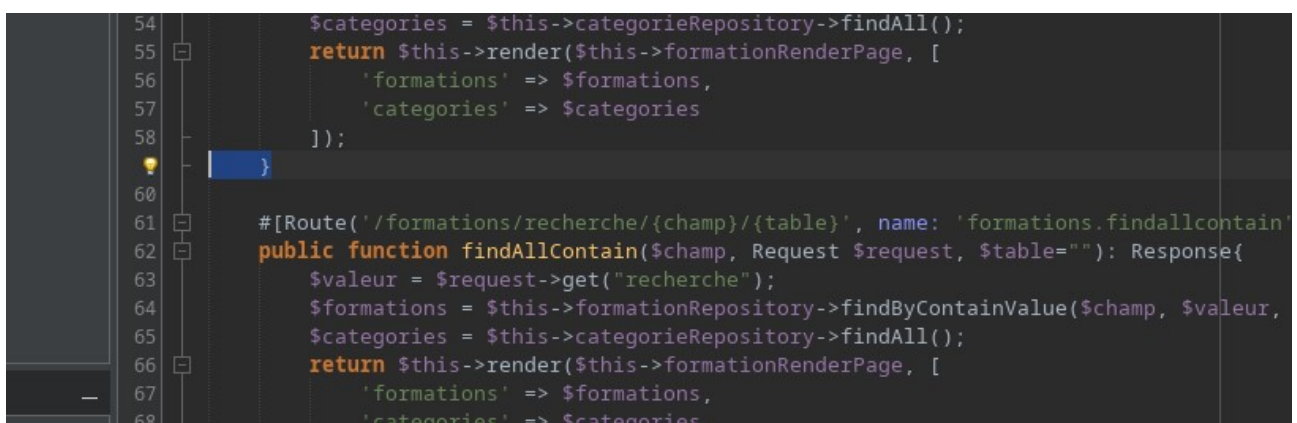
mediatekformation x

php:S1131 : Lines should not end with trailing whitespaces (15)

- 59:0: FormationsController.php
- 72:0: FormationsController.php
- 79:0: FormationsController.php
- 80:0: FormationsController.php
- 30:0: PlaylistsController.php
- 41:0: PlaylistsController.php
- 44:0: PlaylistsController.php

Aperçu du code avant modification

« Lines should not end with trailing whitespaces » indique qu'il y a des espaces inutiles en fin de ligne, il faut les supprimer. Cette règle n'est pas respectée à plusieurs endroits dans le code.



The screenshot shows a code editor with PHP code. The code includes a function `findAllContain` and a route definition. A SonarLint error message is displayed at the bottom: "php:S1131 : Lines should not end with trailing whitespaces (15)". The error is associated with several lines in the file `FormationsController.php` (lines 59, 72, 79, 80, 30, 41, and 44).

```
54     $categories = $this->categorieRepository->findAll();
55     return $this->render($this->formationRenderPage, [
56         'formations' => $formations,
57         'categories' => $categories
58     ]);
59 }
60
61 #[Route('/formations/recherche/{champ}/{table}', name: 'formations.findallcontain')
62 public function findAllContain($champ, Request $request, $table=""): Response{
63     $valeur = $request->get("recherche");
64     $formations = $this->formationRepository->findByContainValue($champ, $valeur);
65     $categories = $this->categorieRepository->findAll();
66     return $this->render($this->formationRenderPage, [
67         'formations' => $formations,
68         'categories' => $categories
```

Aperçu du code après suppression des espaces

Sonarlint : « String literals should not be duplicated »

Éviter la répétition de chaînes de caractères en dur dans le code. On préférera utiliser une variable contenant la chaîne, et dans l'idéal une constante.

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

Aperçu du code avant modification

```
private $formationRepository;
/**
 *
 * @var $formationRenderPage:String
 */
private $formationRenderPage;
```

```
function __construct(FormationRepository $formationRepository,
    $this->formationRenderPage = "pages/formations.html.twig";
    $this->formationRepository = $formationRepository;
    $this->categorieRepository = $categorieRepository;
}
```

```
1  #[Route('/formations', name: 'formations')]
2  public function index(): Response{
3      $formations = $this->formationRepository->findAll();
4      $categories = $this->categorieRepository->findAll();
5      return $this->render($this->formationRenderPage, [
6          'formations' => $formations,
7          'categories' => $categories
8      ]);
9  }
```

Aperçu du code après modification

On a remplacé (voir ci-dessus) toutes les occurrences de la chaîne par la variable qui la représente.

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

```
01 }
02 public function getMiniature(): ?string
03 {
04     return self::$cheminImage.$this->videoId."/default.jpg";
05 }
06 public function getPicture(): ?string
07 {
08     return self::$cheminImage.$this->videoId."/hqdefault.jpg";
09 }
```

On procède de manière analogue partout où cette règle n'est pas respectée. À noter que cette fois-ci, on a utilisé une variable statique. En effet, il s'agit d'une constante dont la valeur ne sera jamais modifiée à l'exécution du programme, et qui ne dépend pas d'une instance particulière de formation.

De cette façon, si jamais le chemin d'accès aux images doit être modifié (changement d'hébergeur d'image par exemple), il suffira de modifier la variable une seule fois pour que le nouveau chemin soit globalement pris en compte dans l'application.

SonarLint : « "" and "" tags should be used. »

Il faut privilégier l'utilisation de balises sémantiques pour mettre en évidence une partie du texte. De cette façon, les moteurs de recherches peuvent savoir qu'il s'agit d'une information à mettre en relief.

```
{% block footer %}
<div class="container text-center">
  <footer>
    <hr>
    <p><small><i>
      Consultez nos <a class="link-secondary" href="{{ path('cgu') }}">Conditions Générales d'Utilisation</a>
    </i></small></p>
  </footer>
</div>
```

Aperçu du code avant modification de `<i></i>` en ``

```
30 {% block footer %}
31 <div class="container text-center">
32 <footer>
33 <hr>
34 <p><small><em>
35   Consultez nos <a class="link-secondary" href="{{ path('cgu') }}">Conditions Générales d'Utilisation</a>
36 </em></small></p>
37
38 </footer>
</div>
```

Aperçu du code après modification

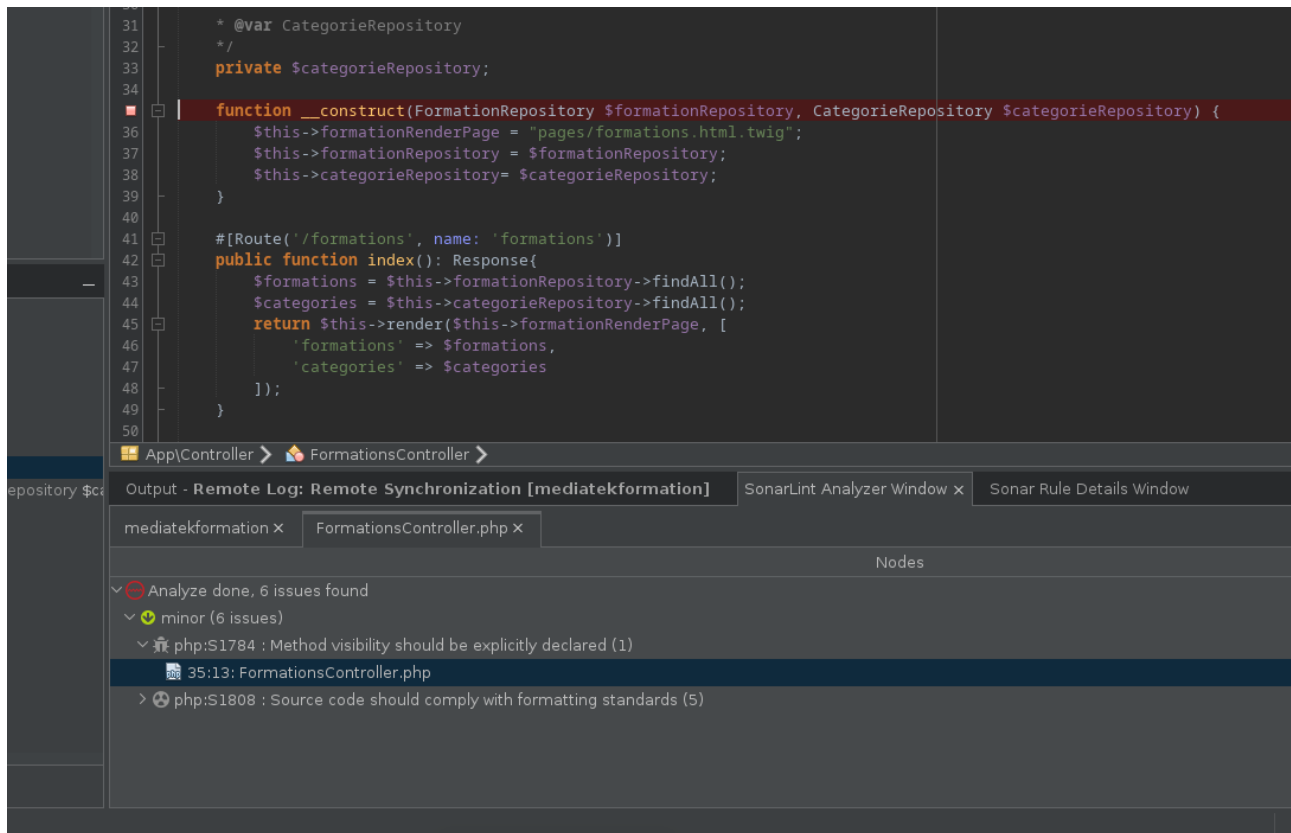
SonarLint : « Source code should comply with formatting standards. »

Certaines règles concernant le formatage du code doivent être respectées. Par exemple, il est obligatoire de mettre un espace avant la parenthèse ouvrante d'une condition et un espace (ou saut de ligne) avant une accolade ouvrante.

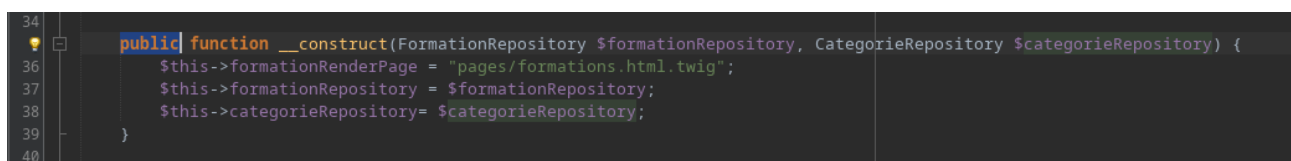
```
* @return array
*/
public function findAllForOnePlaylist($idPlaylist): array{
    return $this->createQueryBuilder('c')
        ->join('c.formations', 'f')
        ->join('f.playlist', 'p')
        ->where('p.id=:id')
        ->setParameter('id', $idPlaylist)
        ->orderBy('c.name', 'ASC')
        ->getQuery()
        ->getResult();
}
```

```
@return array
*/
public function findAllForOnePlaylist($idPlaylist): array
{
    return $this->createQueryBuilder('c')
        ->join('c.formations', 'f')
        ->join('f.playlist', 'p')
        ->where('p.id=:id')
        ->setParameter('id', $idPlaylist)
        ->orderBy('c.name', 'ASC')
        ->getQuery()
        ->getResult();
}
```

Sonarlint : « Method visibility should be explicitly declared »



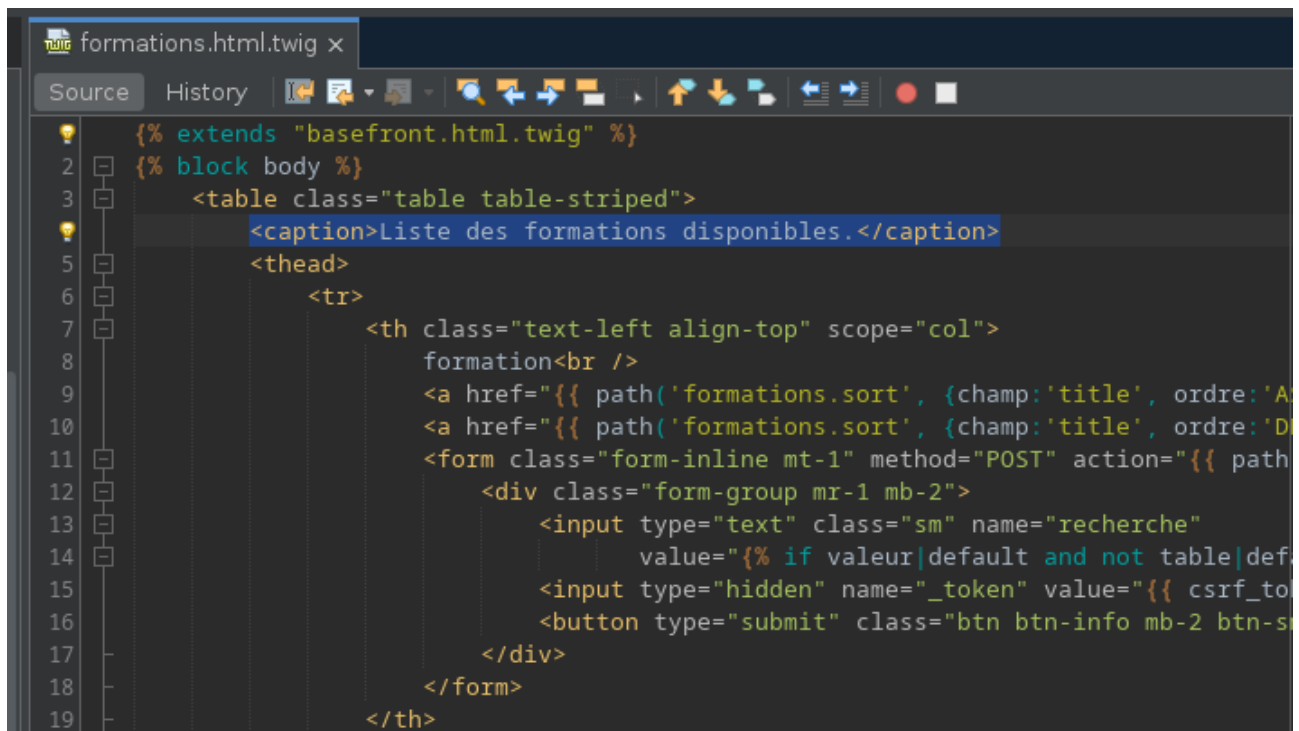
Il faut obligatoirement préciser la portée des méthodes et constructeurs.



SonarLint : « 'table' tags should have a description. »

Le plugin nous fait remarquer que les tables html de certains fichiers twig n'ont pas de description. En effet, la description joue un rôle dans la compréhension de la page par les moteurs de recherches.

Pour décrire une table, on ajoute la balise `<caption>` à l'intérieur de la balise `<table>`



```
1 {% extends "basefront.html.twig" %}
2 {% block body %}
3     <table class="table table-striped">
4         <caption>Liste des formations disponibles.</caption>
5     <thead>
6         <tr>
7             <th class="text-left align-top" scope="col">
8                 formation<br />
9                 <a href="{{ path('formations.sort', {champ:'title', ordre:'A'}) }}">Asc</a>
10                <a href="{{ path('formations.sort', {champ:'title', ordre:'D'}) }}">Desc</a>
11                <form class="form-inline mt-1" method="POST" action="{{ path('formations.search') }}">
12                    <div class="form-group mr-1 mb-2">
13                        <input type="text" class="sm" name="recherche"
14                            value="{{ if valeur|default and not table|default }}">
15                        <input type="hidden" name="_token" value="{{ csrf_token }}">
16                        <button type="submit" class="btn btn-info mb-2 btn-sm">Rechercher</button>
17                    </div>
18                </form>
19            </th>
```

À ce stade, les seules erreurs et avertissements rapportés par SonarLint concernent des fichiers automatiquement générés.

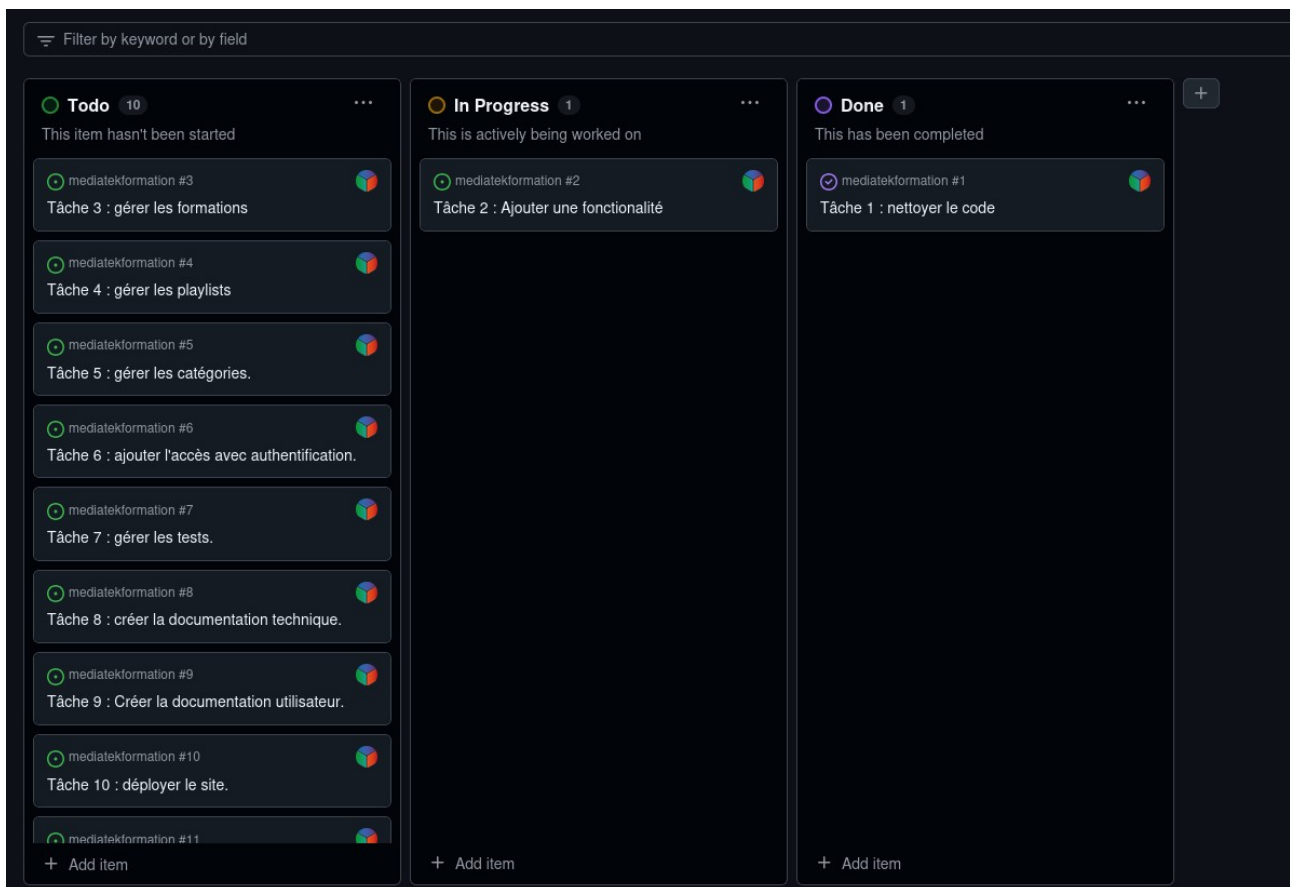
Mission 1 : tâche n°2

Intitulé : ajouter une fonctionnalité

Description : Dans la page des playlists, ajouter une colonne pour afficher le nombre de formations par playlist et permettre le tri croissant et décroissant sur cette colonne. Cette information doit aussi s'afficher dans la page d'une playlist.

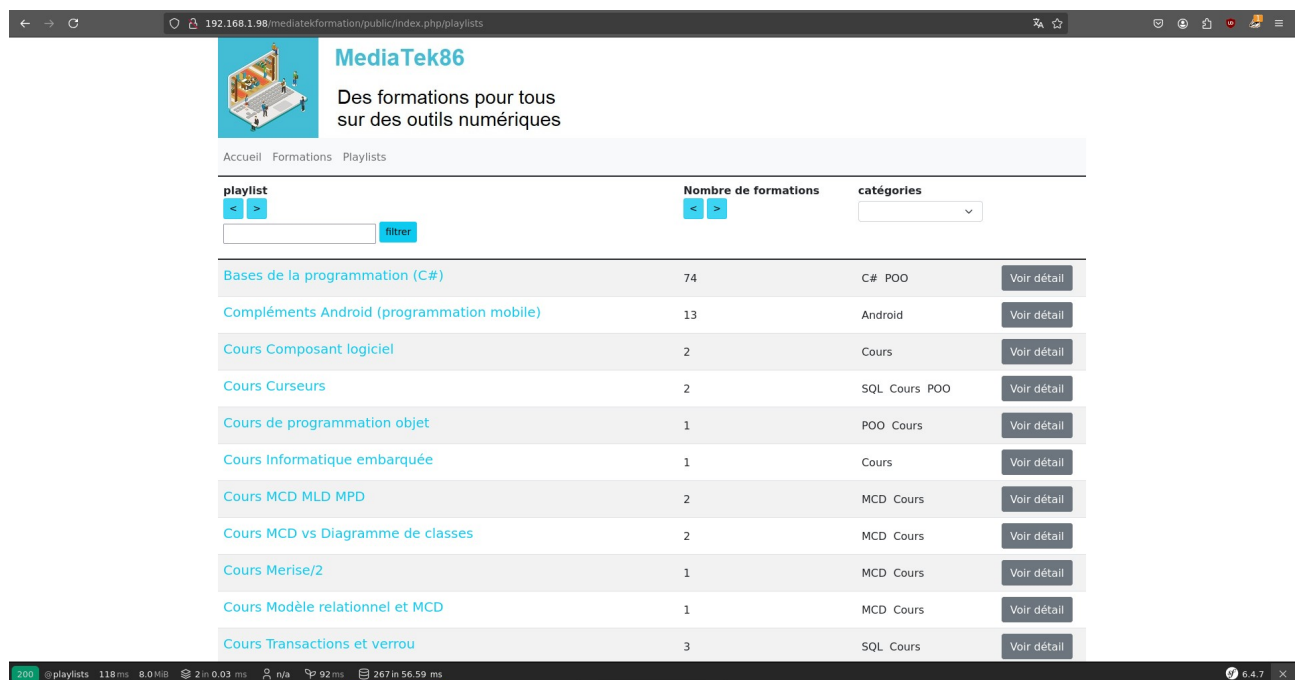
Durée estimée : 2h

Durée effective : 4h



Tâche en cours de réalisation dans le kanban du projet.

Captures d'écran montrant l'ajout de la nouvelle fonctionnalité :



The screenshot shows a web application for MediaTek86. The header includes the logo and the text 'Des formations pour tous sur des outils numériques'. Below the header, there are navigation links: 'Accueil', 'Formations', and 'Playlists'. The main content area displays a table of playlists. The table has three columns: 'playlist', 'Nombre de formations', and 'catégories'. The 'Nombre de formations' column is highlighted in blue. The table lists various playlists such as 'Bases de la programmation (C#)', 'Compléments Android (programmation mobile)', 'Cours Composant logiciel', etc. Each row has a 'Voir détail' button. The browser's developer tools are open at the bottom, showing the HTML structure of the table header.

playlist	Nombre de formations	catégories
Bases de la programmation (C#)	74	C# POO
Compléments Android (programmation mobile)	13	Android
Cours Composant logiciel	2	Cours
Cours Curseurs	2	SQL Cours POO
Cours de programmation objet	1	POO Cours
Cours Informatique embarquée	1	Cours
Cours MCD MLD MPD	2	MCD Cours
Cours MCD vs Diagramme de classes	2	MCD Cours
Cours Merise/2	1	MCD Cours
Cours Modèle relationnel et MCD	1	MCD Cours
Cours Transactions et verrou	3	SQL Cours

La capture précédente met en évidence la nouvelle colonne « Nombre de formations » avec la possibilité de trier l'affichage des playlist par nombre de formation croissant et décroissant.

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

(figure 2)

La figure 2 montre le code ajouté au niveau de l'en-tête de la table d'affichage des playlist. On a ajouté deux liens qui appellent la méthode 'sort' du contrôleur des Playlists sur le champ 'count' et en précisant l'ordre dans lequel trier les playlists. À noter que le champ 'count' n'existe pas explicitement au niveau de la base de données, c'est le résultat d'un traitement SQL comptant le nombre de formations par playlist.

```

<!-- boucle sur les playlists -->
{% if playlists|length > 0 %}
    {% for k in 0..playlists|length-1 %}
        <tr class="align-middle">
            <td>
                <h5 class="text-info">
                    {{ playlists[k].name }}
                </h5>
            </td>
            <td class="text-left">
                {{ playlists[k].formations|length }}
            </td>
        </tr>
    {% endfor %}
{% else %}
    <tr>
        <td colspan="2">Aucune playlist trouvée.</td>
    </tr>
{% endif %}

```

(figure 3)

L’affichage du nombre de formation dans une playlist se fait dans le corps de la table html, dans une boucle sur les playlists, pour chaque playlist, on affiche la taille de l’array ‘formations’ grâce à la fonction twig length, voir figure 3.

```

#[Route('/playlists/tri/{champ}/{ordre}', name: 'playlists.sort')]
public function sort($champ, $ordre): Response
{
    switch ($champ) {
        case "name":
            $playlists = $this->playlistRepository->findAllOrderByName($ordre);
            break;
        case "count":
            $playlists = $this->playlistRepository->findAllOrderByCount($ordre);
            break;
        default: break;
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render($this->playlistsRenderDocument, [
        'playlists' => $playlists,
        'categories' => $categories
    ]);
}

```

(figure 4)

La figure 4 montre l’ajout d’un cas dans le switch pour gérer le tri par nombre de formation croissant/décroissant. On appelle la méthode ‘findAllOrderByCount’ nouvellement créée dans PlaylistRepository.

```

/**
 * Retourne toutes les playlists triées sur le nombre de formation qu'elles contiennent
 * @return Playlist[]
 */
public function findAllOrderByCount($ordre): array
{
    return $this->createQueryBuilder('p')->addSelect('count(p.id) as HIDDEN cnt')->leftjoin('p.formations', 'f')
        ->groupBy('p.id')
        ->orderBy('cnt', $ordre)
        ->getQuery()
        ->getResult();
}

```

La méthode ci-dessus construit la requête permettant de trier les playlists par nombre de formation. On a utilisé « count(p.id) AS HIDDEN cnt » pour créer un alias caché du nombre de fois qu’une formation appartient à

une playlist, de cette façon, il est possible d'ordonner les playlist sur le nombre de formation qu'elle contient, sans changer le nombre de colonnes retournées par la requête.

La capture suivante montre le code affichant le nombre de formation d'une playlist (playlist.html.twig). On procède de la même manière que dans la table d'affichage de toutes les playlists, en affichant la taille de l'array des formations qui s'appelle 'playlistformations' passée en paramètre par le contrôleur des playlists dans la méthode showOne.

```
#[Route('/playlists/playlist/{id}', name: 'playlists.showone')]
public function showOne($id): Response
{
    $playlist = $this->playlistRepository->find($id);
    $playlistCategories = $this->categorieRepository->findAllForOnePlaylist($id);
    $playlistFormations = $this->formationRepository->findAllForOnePlaylist($id);
    return $this->render($this->playlistRenderDocument, [
        'playlist' => $playlist,
        'playlistcategories' => $playlistCategories,
        'playlistformations' => $playlistFormations
    ]);
}
```

Il ne reste plus qu'à afficher la taille de l'array :

```
16 <div class="col">
17 <!-- boucle sur l'affichage des formations -->
18 <em>Il y a {{ playlistformations.length }} formations dans cette playlist</em>
19 {% for formation in playlistformations %}
20 <div class="row mt-1">
21 <div class="col-md-auto">
22 {% if formation.miniature %}
23 <a href="{{ path('formations.showone', {id:formation.id}) }}">
24 
25 </a>
26 {% endif %}
27 </div>
28 <div class="col d-flex align-items-center">
29 <a href="{{ path('formations.showone', {id:formation.id}) }}"
30 class="link-secondary text-decoration-none">
31 {{ formation.title }}
32 </div>
33 </div>
34 </div>
35 </div>
```

Mission 2 : tâche n°1

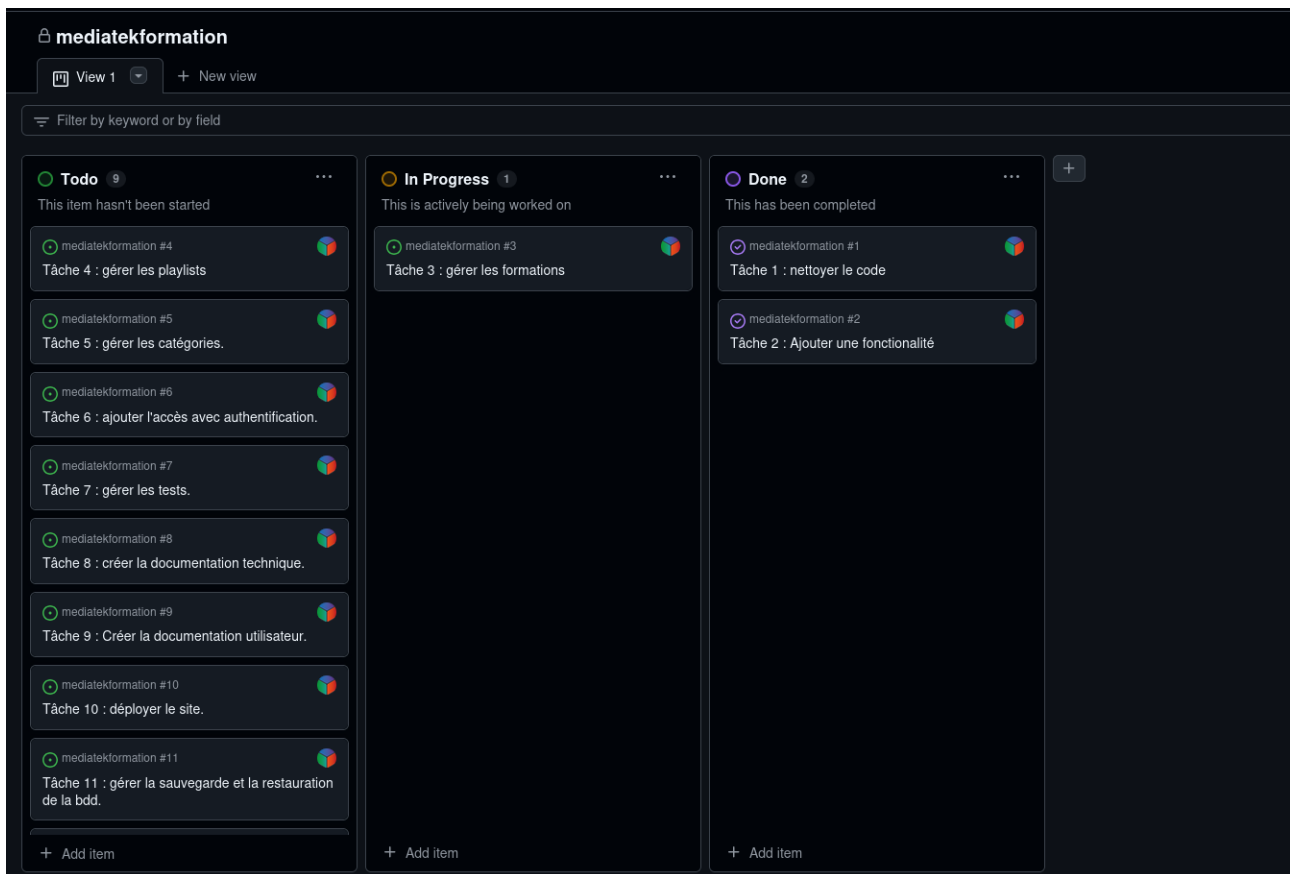
Intitulé : gérer les formations

Description :

- 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.

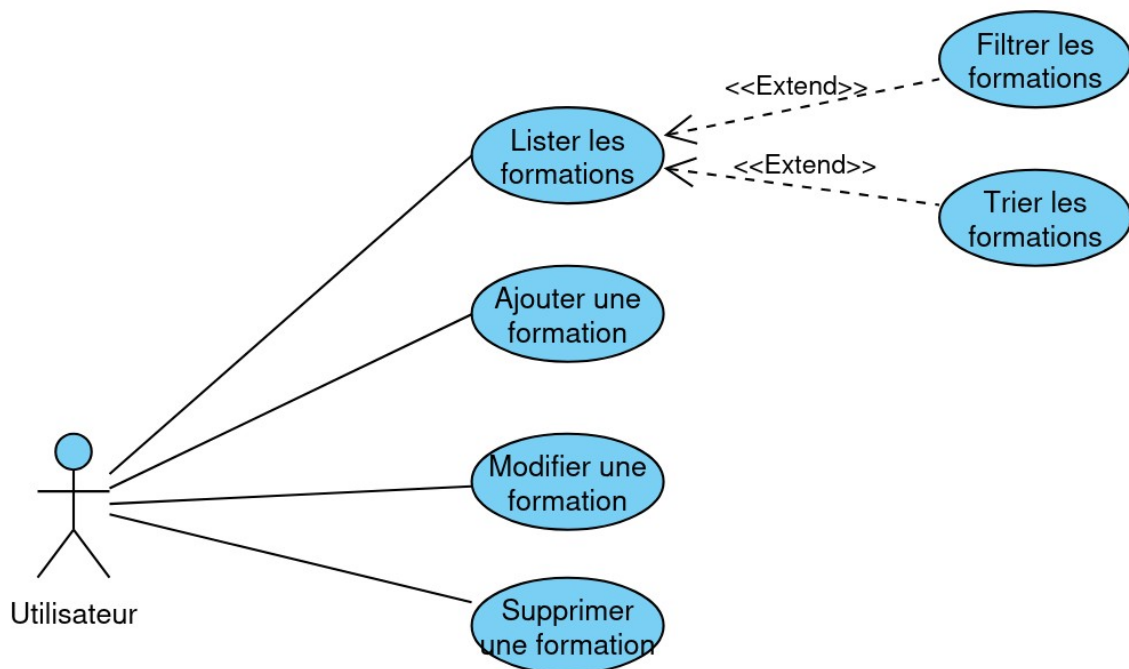
Durée estimée : 5h

Durée effective : 8h57



Aperçu du kanban lorsque la tâche est toujours en cours de traitement


Diagramme des cas du système « backoffice » (sans authentification)



Note : on aurait pu remplacer l'acteur par « Administrateur » mais comme

à cette étape du projet il n'y a pas encore d'authentification, on choisira de le nommer « Utilisateur ».

Maquette de la vue « backoffice » servant de page d'accueil pour l'espace de gestion du site :




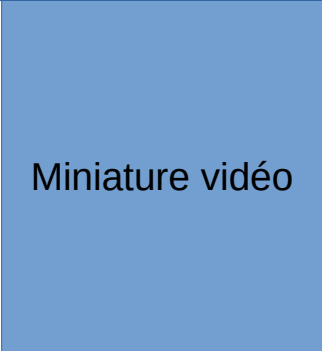
MediaTek86
Des formations pour tous
sur des outils numériques

Accueil Formations Playlists

Espace de gestion - mediatekformation

Depuis cette partie du site, vous pouvez gérer les formations du catalogue.

Voici les deux dernières formations ajoutées au catalogue :


	xx/xx/xxxx Titre de formation Playlist : ... Catégorie : ...		xx/xx/xxxx Titre de formation Playlist : ... Catégorie : ...
------------------------------------------------------------------------------------	-----------------------------------------------------------------------	-------------------------------------------------------------------------------------	-----------------------------------------------------------------------

Consultez nos [conditions générales d'utilisation](#)

Cette vue est un portail pour les administrateurs du site, les seules différences avec la page d'accueil du front office sont les chemins des routes contenues dans le menu de navigation.

Intitulé	Route	Contrôleur
Accueil	/backoffice	BackofficeController
Formations	/backoffice/formations	BackofficeFormationsController
Playlists	Aucune à ce stade	Aucun à ce stade



Maquette de la vue « formations » dans le backoffice :



MediaTek86
Des formations pour tous
sur des outils numériques

Accueil Formations Playlists

+ ajouter une formation

formation	playlist	catégories	date		
< >	< >	<input type="text"/>	< >		
<input type="text"/>	<input type="text"/>			filtrer	filtrer
Eclipse n°1 : installation de l'IDE	Eclipse et Java	Java	3/11/2022	Miniature	 
//					
//					
//					
//					

Formations disponibles

Consultez nos [conditions générales d'utilisations](#)

Cette page liste toutes les formations disponibles et permet de trier ou filtrer l’affichage par formation, playlist, catégories et date en réutilisant les mêmes filtres présents dans FormationsController. À chaque ligne du tableau des formations, se trouvent deux liens-boutons dont les icônes sont explicites :

- le bouton gauche (poubelle) permet de supprimer la formation après confirmation par l'utilisateur.

- le bouton droit (crayon) permet d'accéder au formulaire de modification pré-rempli avec les informations de l'entité sélectionnée.

Au dessus du tableau, un grand bouton rouge permet d'accéder au formulaire d'ajout de formation.

Note : BackofficeFormationsController hérite de FormationController

Fonctionnalité	Route	Contrôleur	Méthode	Paramètre(s)
Trier par formation	/backoffice/formations/tri/	BackofficeFormationsController	backoffice_formations.sort (appelle la fonction formations.sort dans la méthode mère FormationsController)	- champ = title - ordre = ASC DESC - table = formation
Filtrer par formation	/backoffice/formations/recherche/	//	backoffice_formations.findallcontain	- champ = title - table = formation
Trier par playlist	/backoffice/formations/tri/	//	backoffice_formations.sort	- champ = name - ordre = ASC DESC - table = playlist
Filtrer par playlist	/backoffice/formations/recherche/	//	backoffice_formations.findallcontain	- champ = name - table = playlist
Filtrer par catégories	backoffice/formations/recherche/id/categories	//	backoffice_formations.findallcontain	- champ = id - table = categories
Trier par date	/backoffice/formations/tri/	//	backoffice_formations.sort	- champ = publishedAt - ordre = ASC DESC

Maquette de la vue « addformation » et « editformation » du backoffice :

Les formulaires pour l'ajout et la modification d'une formation sont dans le même fichier twig :

« templates/pages/backoffice/editformation.html.twig »

La seule différence entre les deux routes est le pré-remplissage du formulaire par le contrôleur avec les données de la formation sélectionnée.



MediaTek86
Des formations pour tous
sur des outils numériques

Accueil Formations Playlists

Date

Titre

Description

Identifiant vidéo

Catégories

Playlist

Eclipse et Java

Valider

Consultez nos [conditions générales d'utilisation](#)

Fonctionnalité	Route	Contrôleur	Méthode	Paramètre(s)
Ajouter une formation	/backoffice/formations/addformation	BackofficeFormationsController	backoffice_formation.addone	/
Modifier une formation	/backoffice/formations/editformation	//	backoffice_formation.edition	- id

Supprimer une formation	/backoffice/formations/delete	BackofficeFormationsController	backoffice_formation.deleteone	- id
-------------------------	-------------------------------	--------------------------------	--------------------------------	------

```

#[Route('/backoffice/formations/addformation', name: 'backoffice_formation.addone')]
public function addOne(Request $request)
{
    $formation = new Formation();
    $formationType = $this->createForm(FormationType::class, $formation, array(
        'categories' => $formation->getCategories(),
        'playlist' => $formation->getPlaylist()
    ));
    $formationType->handleRequest($request);
    if ($formationType->isSubmitted() && $formationType->isValid()) {
        $formation = $formationType->getData();
        $submitDate = $formation->getPublishedAt();
        //On s'assure que la date saisie est bien antérieure à la date actuelle
        if (time() - strtotime($submitDate->format('m/d/Y')) > 0) {
            $this->formationRepository->add($formation);
            $this->addFlash('formation_success', 'La formation a bien été ajoutée.');
```

Méthode addOne de BackofficeFormationsController

```

86 //
87 #[Route('/backoffice/formations/editformation/{id}', name: 'backoffice_formation.editone')]
88 public function editOne(Request $request): Response
89 {
90     $id = $request->get('id');
91     $formation = $this->formationRepository->findByContainValue("id", $id)[0];
92     $formationType = $this->createForm(FormationType::class, $formation, array(
93         'categories' => $formation->getCategories(),
94         'playlist' => $formation->getPlaylist()
95     ));
96     $formationType->handleRequest($request);
97     if ($formationType->isSubmitted() && $formationType->isValid()) {
98         $formationType->getData();
99         $submitDate = $formation->getPublishedAt();
100         //On s'assure que la date saisie est bien antérieure à la date actuelle
101         if (time() - strtotime($submitDate->format('m/d/Y')) > 0) {
102             $this->formationRepository->getEntityManager()->flush();
103             $this->addFlash('formation_success', 'La formation a bien été modifiée.');
```

Méthode editOne du contrôleur

```

#[Route('/backoffice/formations/delete/{id}', name: 'backoffice_formation.deleteone')]
public function deleteOne($id): Response
{
    $formation = $this->formationRepository->findByContainValue("id", $id)[0];
    $this->formationRepository->remove($formation);
    return parent::redirectToRoute("backoffice_formation");
}

```

Méthode deleteOne du contrôleur

Analyse de l'arborescence des fichiers ajoutés :

src

```

====> Controller
====> BackofficeController.php
====> BackofficeFormationsController.php
====> Form
====> FormationType.php

```

templates

```

====> pages
====> backoffice
====> accueil.html.twig
====> editformation.html.twig
====> formations.html.twig
====> playlists.html.twig
====> basefront_backoffice.html.twig

```

BackofficeFormationsController hérite de FormationsController de façon à pouvoir réutiliser les méthodes de tri et de filtrage déjà présentes dans la classe mère.

FormationType est une classe permettant de représenter le formulaire d'ajout/édition de formation. Il suffit de passer un objet de type formation lors de sa création dans le contrôleur pour qu'il pré-remplisse automatiquement le formulaire. Par ailleurs, il est possible de lui envoyer un objet formation « vide », nouvellement créé pour gérer l'ajout de nouvelles formations. Une fois soumis, le formulaire retourne la formation modifiée, ou une nouvelle formation remplie avec les données saisies dans le formulaire. Il ne reste plus qu'à persister ou mettre à jour l'entité dans la base de données, si la saisie est valide.

« basefront_backoffice.html.twig » est un template qu'hérite de « basefront.html.twig », mais remplace les liens du menu de navigation avec les nouvelles routes du backoffice.

Les pages twig présentent dans le dossier « templates/backoffice/ » héritent de « basefront_backoffice.html.twig » sauf « templates/backoffice/formations.html.twig », qui hérite directement de « templates/formations.html.twig » pour réutiliser les filtres et tris, seuls les chemins des routes changent.

Mission 2 : tâche n°2

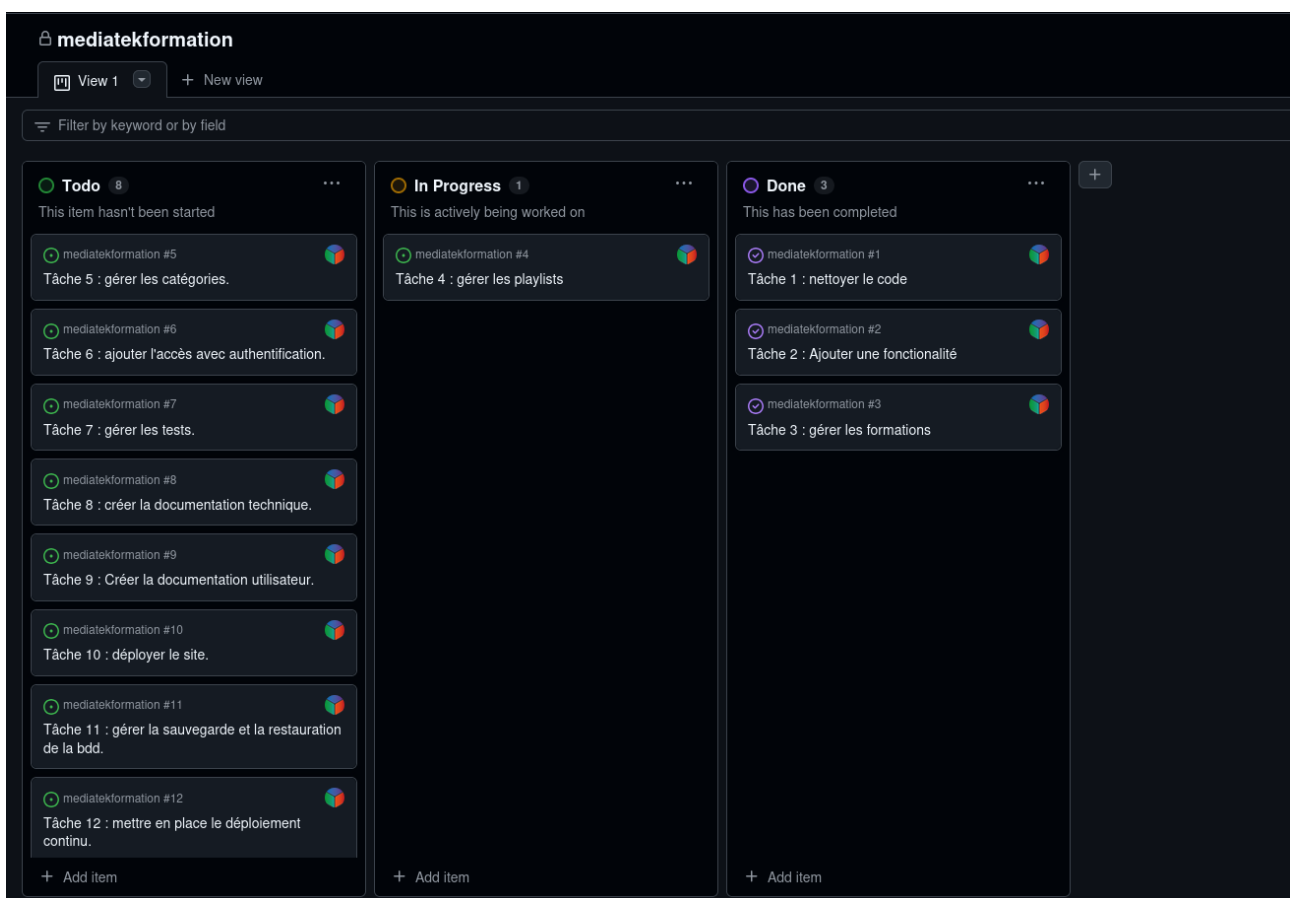
Intitulé : gérer les playlists

Description :

- 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.

Durée estimée : 5h

Durée effective : 5h20



Aperçu de la tâche en cours dans le kanban.

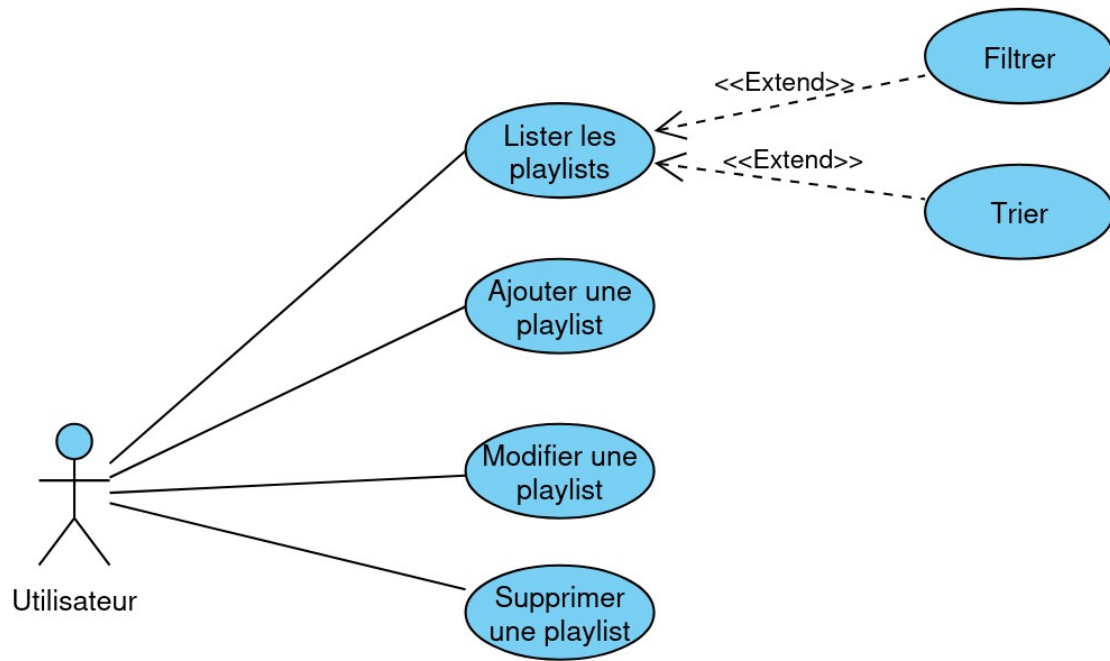


Diagramme des cas de la tâche

Maquette de la vue du gestionnaire de playlists «
templates/pages/backoffice/playlists.html.twig » :



MediaTek86

Des formations pour tous
sur des outils numériques

Accueil Formations Playlists

[+ ajouter une playlist](#)

playlist



filtrer

Nombre de
formation



catégories

Bases de la programmation
(C#)

74

C# POO



//

//

//

//

Playlists disponibles

Consultez nos [conditions générales d'utilisations](#)

Cette page permet de gérer les playlists et réutilise les tris et filtres déjà présents sur le front office. De la même façon que pour les formations, on fait hériter le contrôleur BackofficePlaylistsController de PlaylistsController, et on transfère les appels des méthodes de filtrage/tri vers la classe mère, mais à des routes différentes, on modifie également la valeur des variables 'protected' de la classe mère contenant les chemins aux fichiers twig, pour qu'elles affichent le bon template.

Maquette du formulaire d'ajout de playlist «
templates/pages/backoffice/editoraddplaylist.html.twig » :



MediaTek86

Des formations pour tous
sur des outils numériques

Accueil Formations Playlists

Nom

Description

Valider

Consultez nos [conditions générales d'utilisation](#)

Le formulaire ci-dessus est représenté par la classe PlaylistType, la construction de ce formulaire est établie selon deux cas :

- Ajout d'une nouvelle playlist : le formulaire se compose de deux champs, « Nom » et « Description ».

- Modification d'une playlist existante : comporte un champ supplémentaire en lecture seule listant la totalité des formations associées à la playlist à modifier.

À noter que l'implémentation réelle diffère légèrement au niveau du champ « Description » où une TextArea multi-ligne est utilisée au lieu d'un simple champ de saisie (input type='text').

Construction du formulaire PlaylistType

Comme énoncé ci-dessus, le formulaire va légèrement différer selon le cas à gérer (ajout ou modification). Le contrôleur `BackofficePlaylistsController` appelle la méthode `buildForm` de `PlaylistType` et envoie la collection des formations de la playlist en paramètre. S'il s'agit d'une nouvelle playlist, la taille de la collection des formations associée à l'entité est vide, et il faut omettre l'affichage des formations. Si la collection n'est pas vide, on affiche un champ supplémentaire qui liste chacune des formations de cette playlist.

```
30 public function buildForm(FormBuilderInterface $formBuilder, array $options) : void
31 {
32     $formations = $options['formations'];
33     if(count($formations) > 0) {
34         $formBuilder
35             ->add('name', TextType::class, [
36                 'required' => true,
37                 'label' => 'Nom de la playlist'
38             ])
39             ->add('description', TextareaType::class, [
40                 'required' => false,
41                 'label' => "Description",
42                 'attr' => array('rows' => '15')
43             ])
44             ->add('formations', ChoiceType::class, [
45                 'label' => 'Formations dans cette playlist',
46                 'choices' => $formations,
47                 'choice_label' => 'title',
48                 'disabled' => true,
49                 'expanded' => true,
50                 'required' => false,
51                 'multiple' => false,
52                 'choice_attr' => [
53                     'checked' => true
54                 ]
55             ])
56             ->add('submit', SubmitType::class, [
57                 'label' => 'Valider'
58             ]);
59     } else {
```

La capture ci-dessus montre le cas d'édition d'une playlist pré-existante car la collection de formation associée à cette playlist n'est pas vide. Le champ formation, de type `ChoiceType` est ajouté et automatiquement rempli avec la propriété `'title'` de chaque entité de type `Formation` dans la collection `'$formation'`.

```

67         });
68     } else {
69         $formBuilder
70             ->add('name', TextType::class, [
71                 'required' => true,
72                 'label' => 'Nom de la playlist'
73             ])
74             ->add('description', TextareaType::class, [
75                 'required' => false,
76                 'label' => "Description",
77                 'attr' => array('rows' => '15')
78             ])
79             ->add('submit', SubmitType::class, [
80                 'label' => 'Valider'
81             ]);
82     }
83 }

```

Le deuxième cas est plus simple, comme la playlist vient d'être créée, il n'y a pas de formations à lister.

Ce sont les formations qui doivent être rattachées aux playlists et non l'inverse.

Traitement des formulaires

```

#[Route('/backoffice/playlists/addoreditplaylist/{id?}', name: 'backoffice_playlists.addoreditone')]
public function addOrEditOne(Request $request, $id=null): Response
{
    $playlist = null;
    //Il s'agit d'une modification d'entité car l'id n'est pas null.
    if ($id !== null) {
        $playlist = $this->playlistRepository->findByContainValue("id", $id)[0];
    } else {
        $playlist = new Playlist();
    }
    $playlistType = $this->createForm(PlaylistType::class, $playlist, array(
        'formations' => $playlist->getFormations()
    ));
    $playlistType->handleRequest($request);
    if ($playlistType->isSubmitted() && $playlistType->isValid()) {
        $playlist = $playlistType->getData();
        if ($id !== null) {
            $this->playlistRepository->getEntityManager()->flush();
            $this->addFlash('playlist_success', 'La playlist a été modifiée avec succès.');
```

Méthode addOrEditOne de BackofficePlaylistsController.

Cette fois-ci on gère l'ajout et l'édition de playlist dans une seule méthode. Si cette méthode est appelée avec un paramètre optionnel « id », alors il s'agit du cas d'édition d'une playlist existante, sinon si aucun id n'est spécifié (vaut null par défaut), il s'agit d'un ajout. Une fois soumis, le

formulaire est traité, selon les cas, il ajoutera une nouvelle entité ou modifiera l'entité playlist courante, puis redirigera le client vers la page d'accueil du gestionnaire de playlist, en affichant un message de succès si tout s'est bien passé.

```
#[Route('/backoffice/playlists/deleteplaylist/{id}', name: 'backoffice_playlists.deleteone')]
public function deleteOne($id): Response
{
    $playlist = $this->playlistRepository->findByContainValue("id", $id)[0];
    if (count($playlist->getFormations()) > 0) {
        $this->addFlash(
            'playlist_error',
            '[!] La playlist n\'a pas pu être supprimée car elle est rattachée à une ou plusieurs formations.'
        );
        return $this->redirectToRoute("backoffice_playlists");
    } else {
        $this->playlistRepository->remove($playlist);
        $this->addFlash('playlist_success', 'La playlist a bien été supprimée.');
```

Méthode de suppression de playlist dans BackofficePlaylistsController

La suppression de playlist peut se faire uniquement si elle n'est associée à aucune formation. Pour vérifier ça, on accède à la propriété de type collection 'formations' de la playlist courante, si la collection est vide, la playlist peut être supprimée, sinon, il faudra d'abord supprimer manuellement les formations qu'elle contient avant de supprimer la playlist. À la fin du traitement, on affiche un message d'erreur ou de succès après redirection vers la page d'accueil des playlists.

Mission 2 : tâche n°3

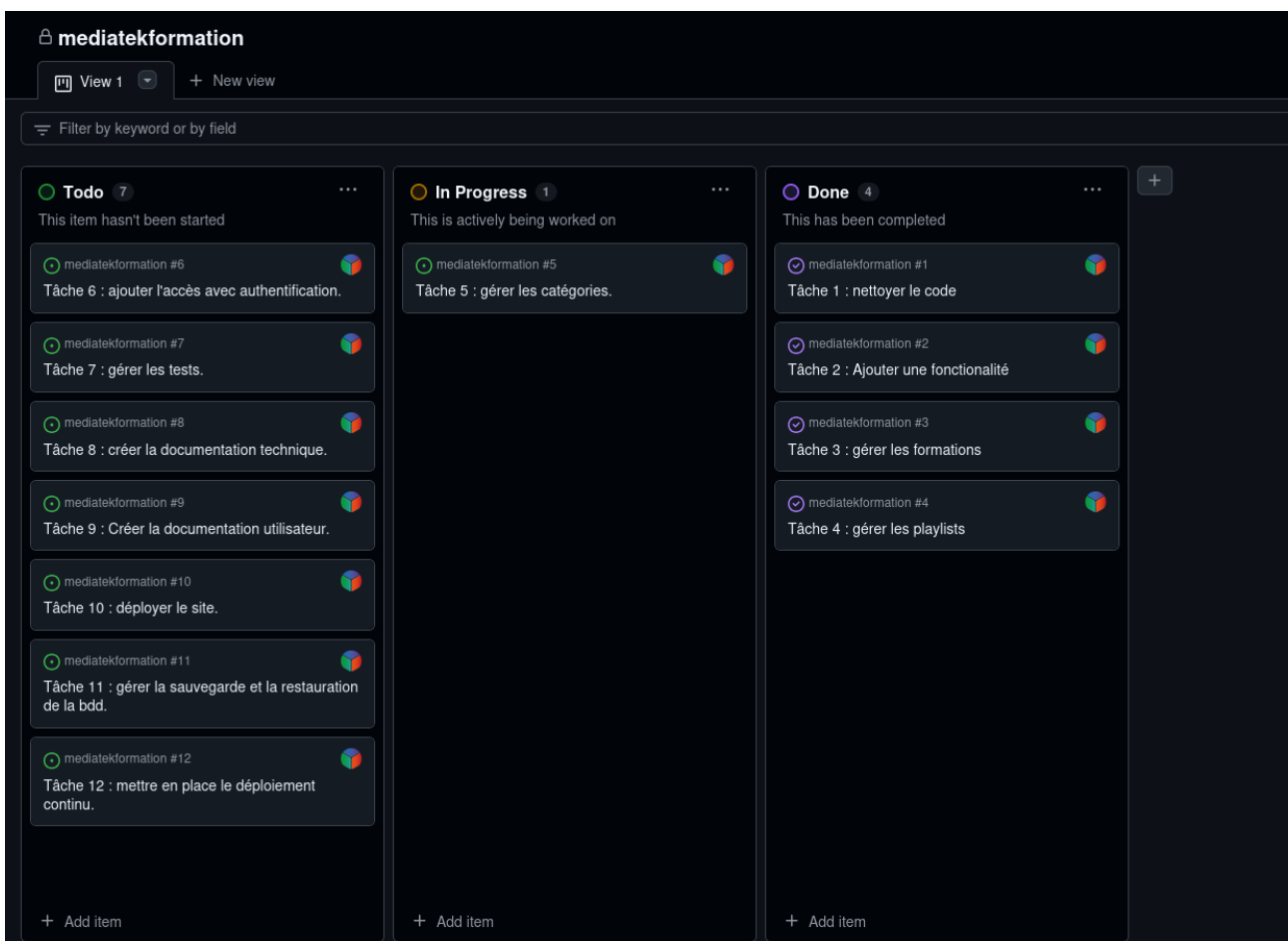
Intitulé : gérer les catégories

Description :

- 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.

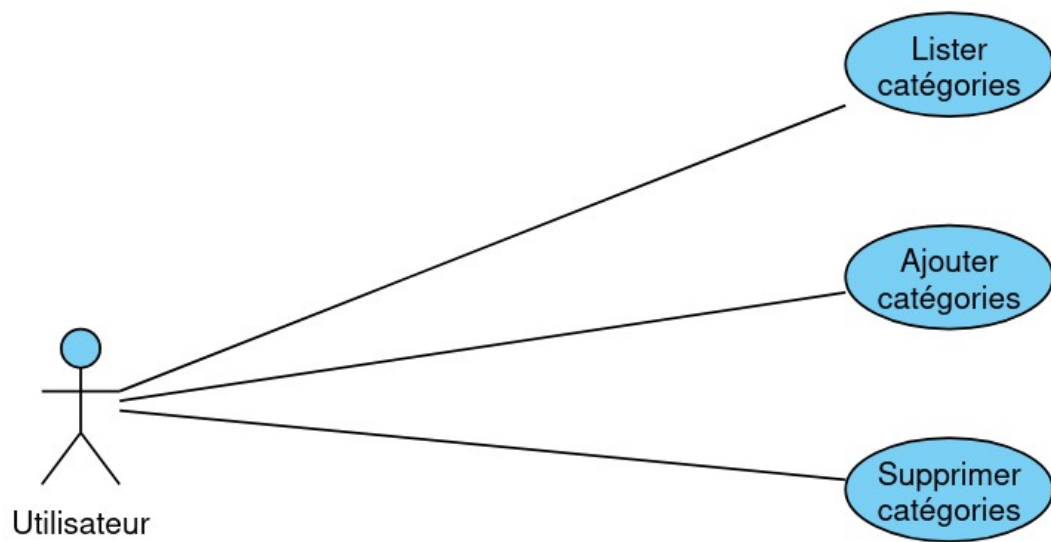
Durée estimée : 3h

Durée effective : 1h58



Aperçu du kanban de la tâche en cours

Diagramme des cas du gestionnaire de catégories



Maquette de la page de gestion des catégories



MediaTek86



Des formations pour tous
sur des outils numériques

Accueil Formations Playlists Catégories

Titre catégorie

Ajouter

Titre

Java	
UML	
//	
//	
//	
Catégories disponibles	

Consultez nos [conditions générales d'utilisations](#)

La maquette ci-dessus contient un formulaire permettant l'ajout d'une nouvelle catégorie en saisissant son nom et une table html chargée d'afficher le nom de chaque catégorie ainsi qu'un bouton permettant de la supprimer.

Formulaire d'ajout de catégorie

Une nouvelle classe `CategorieType` représente le formulaire d'ajout de catégorie.

```
10 use App\Entity\Categorie;
11
12 /**
13  * Description of CategorieType
14  *
15  * @author sysadmin
16  */
17 class CategorieType extends AbstractType
18 {
19     public function buildForm(FormBuilderInterface $formBuilder, array $options) : void
20     {
21         $formBuilder
22             ->add('name', TextType::class, [
23                 'label' => 'Nom'
24             ])
25             ->add('submit', SubmitType::class, [
26                 'label' => 'Ajouter'
27             ]);
28     }
29     public function configureOptions(OptionsResolver $resolver): void
30     {
31         $resolver->setDefaults([
32             'data_class' => Categorie::class
33         ]);
34     }
35 }
```

Ce formulaire est simple et ne contient que deux champs, l'un pour saisir le nom de la catégorie (requis par défaut), et un champ de type submit pour soumettre le formulaire.

La partie traitement se fait du côté du contrôleur nouvellement créé : « `BackofficeCategoriesController` »

Traitement et construction du formulaire

```
7  #[Route('/backoffice/categories', name: 'backoffice_categories')]
8  public function index(Request $request): Response
9  {
10     $categories = $this->categorieRepository->findAll();
11     $categorieType = $this->createForm(CategorieType::class, null, []);
12     $categorieType->handleRequest($request);
13     if ($categorieType->isSubmitted() && $categorieType->isValid()) {
14         $categorie = $categorieType->getData();
15         $found = $this->categorieRepository->findOneBy([
16             'name' => $categorie->getName()
17         ]);
18         if ($found === null) {
19             $this->categorieRepository->add($categorie);
20             $this->addFlash('categorie_success', 'La catégorie a bien été ajoutée.');
```

La méthode `index` de `BackofficeCategoriesController` se charge de l’affichage et du traitement du formulaire, ainsi que de l’affichage de la liste des catégories dans la table `html`.

Le nom de la catégorie à ajouter ne doit pas déjà exister dans la base de données, pour cela, après soumission du formulaire, on cherche toutes les occurrences de l’entité `Catégorie` dans la base dont le champ ‘`nom`’ est identique au nom saisi dans le formulaire. Si aucune entité n’est trouvée, cela signifie que la catégorie n’existe pas et peut donc être ajoutée de façon permanente dans la base. Sinon, il faut rediriger l’utilisateur en affichant un message d’erreur.

Suppression de catégories

```
64  #[Route('/backoffice/categories/delete/{id}', name: 'backoffice_categories.deleteone')]
65  public function deleteOne($id): Response
66  {
67      $cat = $this->categorieRepository->find($id);
68      if (count($cat->getFormations()) > 0) {
69          $this->addFlash(
70              'categorie_error',
71              "[!] La catégorie ne peut pas être supprimée si elle est présente dans des formations."
72          );
73      } else {
74          $this->categorieRepository->remove($cat);
75          $this->addFlash('categorie_success', "La catégorie a bien été supprimée.");
76      }
77      return $this->redirectToRoute('backoffice_categories');
78  }
```

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

La méthode `deleteOne` du contrôleur récupère la catégorie sélectionnée via son id envoyé en paramètre. L'entité `Catégorie` possède l'attribut `'formations'` de type `Collection` de formations, ce qui permet de facilement savoir si une catégorie est rattachée à d'autres formations. En effet, on compte le nombre d'éléments de cette collection grâce à la méthode `getFormations`, si cette collection est vide, cela veut dire que la catégorie peut être supprimée, sinon, on affiche un message d'erreur. Dans tous les cas, on redirige l'utilisateur vers le portail des catégories pour qu'il ait un aperçu des changements.

Mission 2 : tâche n°4

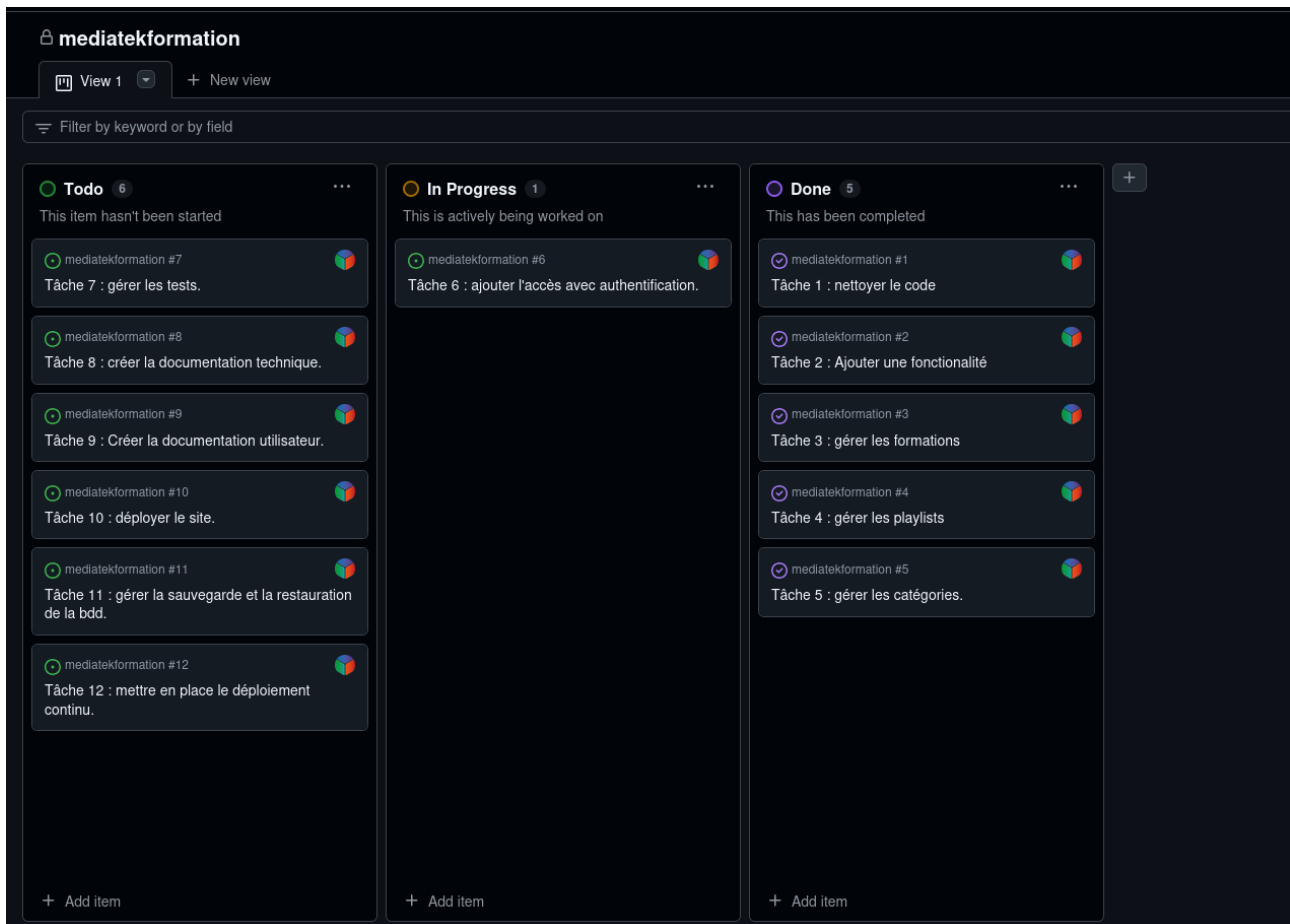
Intitulé : ajouter l'accès avec authentification

Description :

- Le back office ne doit être accessible qu'après authentification : un seul profil administrateur doit avoir le droit d'accès.
- Il doit être possible de se déconnecter, sur toutes les pages (avec un lien de déconnexion).

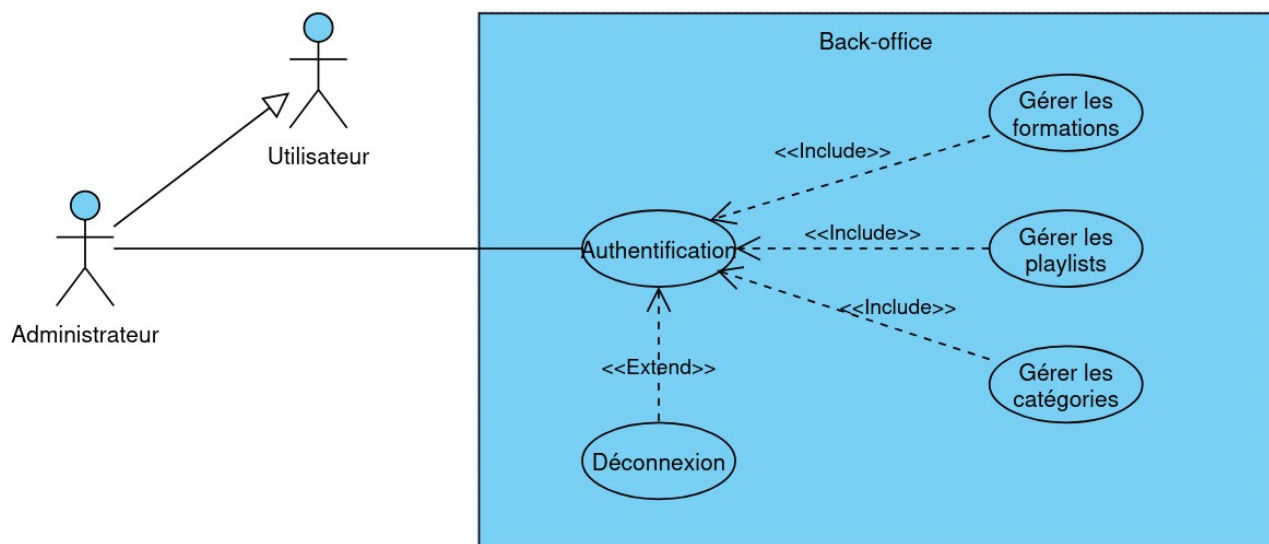
Durée estimée : 4h

Durée effective : 1h30



Aperçu du kanban de la tâche en cours

Diagramme de cas de la tâche



Maquette du formulaire d'authentification



MediaTek86

Des formations pour tous
sur des outils numériques

Accueil Formations Playlists Connexion

Nom d'utilisateur

Mot de passe

Connexion

Consultez nos [conditions générales d'utilisations](#)

Création de l'authentification avec symfony CLI

Symfony possède un utilitaire en ligne de commande pour générer du code personnalisé. Pour sécuriser le backoffice, il faut générer une nouvelle entité Admin.

```
symfony console make:user Admin
```

La commande précédente génère automatiquement l'entité Admin avec son repository et un contrôleur, puis ajoute également une table 'admin' dans la base de données.

Note : make:user permet de spécifier qu'il s'agit d'une entité de type user, et qu'elle sera utilisée pour un système d'authentification.

À ce stade, les modifications ne sont pas encore appliquées au projet, il faut migrer les changements au niveau de symfony et de doctrine :

```
symfony console make:migration  
symfony console doctrine:migrations:migrate -n
```

On doit maintenant ajouter un nouvel administrateur dans la base. Pour des raisons de sécurité, on ne stocke jamais le mot de passe en clair, mais une représentation hashée du mot de passe, de cette façon, même dans le cas d'un accès non autorisé à la base de données, les mots de passes ne peuvent pas directement être récupérés par un attaquant.

À noter que le hashage seul n'est pas suffisant, il faut également 'saler' les hashes pour éviter les attaques par force brute ou apparentées.

Pour générer un hash, on utilise l'utilitaire symfony :

```
symfony console security:hash-password
```

On saisit le mot de passe en clair que l'on souhaite hasher, puis l'utilitaire nous retourne le hash du mot de passe. Il ne reste plus qu'à ajouter une entrée dans la nouvelle table 'admin' avec les informations d'authentification souhaitées (dont le hash du mot de passe).

Pour ce faire, on peut utiliser phpmyadmin pour exécuter la requête d'insertion suivante :

```
INSERT INTO admin (id, username, `password`, roles) VALUES(1,
'admin', 'HASH_DU_MOT_DE_PASSE', '[\ «ROLE_ADMIN»\]');
```

On a ajouté un utilisateur avec le bon mot de passe et le rôle d'administrateur.

Formulaire d'authentification

Le formulaire d'authentification a été généré par l'utilitaire symfony :

```
symfony console make:security:form-login
```

Un nouveau template twig apparaît, dans lequel se trouve le formulaire d'authentification sécurisé (contient un champ caché pour le token CSRF).

La route vers l'authentification et la dés-authentification est gérée par le contrôleur qui a été généré précédemment.

```
class AdminLoginController extends AbstractController
{
    #[Route(path: '/login', name: 'app_login')]
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        // get the login error if there is one
        $error = $authenticationUtils->getLastAuthenticationError();

        // last username entered by the user
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render('adminlogin/login.html.twig', [
            'last_username' => $lastUsername,
            'error' => $error,
        ]);
    }

    #[Route(path: '/logout', name: 'app_logout')]
    public function logout(): void
    {
        throw new \LogicException('This method can be blank - it will be intercepted by the logout key on your firewall.');
```

La comparaison des hash est gérée de façon interne par le framework.

Restriction d'accès au backoffice

Le but de cette authentification est de protéger l'espace de gestion du site afin de ne permettre son accès qu'à l'administrateur.

Autrement dit, il faut spécifier la ou les routes ayant besoin d'un accès protégé par authentification dans config/packages/security.yaml

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

Ici, toutes les routes du backoffice seront protégées.

Si l'utilisateur tente d'y accéder sans s'être authentifié préalablement, il sera redirigé vers le formulaire d'authentification.

Afficher les liens de connexion ou déconnexion

Cette partie est concentrée dans le front-end, il faut décider selon si l'utilisateur est connecté ou non, d'afficher, soit un lien de connexion, soit un lien de déconnexion.

```
13 <div class="collapse navbar-collapse" id="navbarSupportedContent">
14   <ul class="navbar-nav mr-auto">
15     <li class="nav-item">
16       <a class="nav-link" href="{{ path('accueil') }}">Accueil</a>
17     </li>
18     <li class="nav-item">
19       <a class="nav-link" href="{{ path('formations') }}">Formations</a>
20     </li>
21     <li class="nav-item">
22       <a class="nav-link" href="{{ path('playlists') }}">Playlists</a>
23     </li>
24     {% if app.user %}
25     <li class="nav-item">
26       <a class="nav-link" href="{{ path('app_logout') }}">Déconnexion</a>
27     </li>
28     {% else %}
29     <li class="nav-item">
30       <a class="nav-link" href="{{ path('app_login') }}">Connexion</a>
31     </li>
32     {% endif %}
33   </ul>
34 </div>
35 </nav>
36 </div>
```

On gère ces deux cas dans une condition vérifiant si un objet 'user' existe pour cette session. Si c'est le cas, on affiche le lien de déconnexion, sinon, le lien de connexion.

Mission 3 : tâche n°1

Intitulé : gérer les tests.

Description :

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.

Durée estimée : 7h

Durée effective : 6h10 (abandon des derniers tests)

Filter by keyword or by field

Todo 5

This item hasn't been started

- mediatekformation #8
Tâche 8 : créer la documentation technique.
- mediatekformation #9
Tâche 9 : Créer la documentation utilisateur.
- mediatekformation #10
Tâche 10 : déployer le site.
- mediatekformation #11
Tâche 11 : gérer la sauvegarde et la restauration de la bdd.
- mediatekformation #12
Tâche 12 : mettre en place le déploiement continu.

+ Add item

In Progress 1

This is actively being worked on

- mediatekformation #7
Tâche 7 : gérer les tests.

+ Add item

Done 6

This has been completed

- mediatekformation #1
Tâche 1 : nettoyer le code
- mediatekformation #2
Tâche 2 : Ajouter une fonctionnalité
- mediatekformation #3
Tâche 3 : gérer les formations
- mediatekformation #4
Tâche 4 : gérer les playlists
- mediatekformation #5
Tâche 5 : gérer les catégories.
- mediatekformation #6
Tâche 6 : ajouter l'accès avec authentification.

+ Add item

Contexte : MediaTek86
Situation professionnelle : Symfony
Application : mediatekformation (site de mise à disposition des auto-formations).

Plan de tests

Tests unitaires

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

Tests d'intégration

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler que la méthode d'ajout de formations du repository n'ajoute pas une formation lorsque sa date de publication est postérieure à la date d'aujourd'hui.	Test d'intégration lancé avec les paramètres suivants : - <code>publishedAt = '2025-01-01 00:00:00'</code>	L'entité n'est pas ajoutée à la base de données. Le nombre de ligne est identique avant et après le test.	Ok
Contrôler que la méthode de modification de formation ne conserve pas les changements de date dans la base de données si la date est postérieure à la date d'aujourd'hui.	Test d'intégration lancé avec la date suivante : - <code>id = 2</code> (bdd de test) - <code>publishedAt = '2025-01-01 00:00:00'</code>	La date n'est pas sauvegardée dans la base de données lorsqu'elle est postérieure à la date actuelle.	Ok
Contrôler les méthodes ajoutées dans les classes repository.	Pour chaque méthode de chaque repository, on test que les méthodes retournent bien une entité ou une collection d'entité.	Toutes les assertions des tests sont vraies.	Ok

Tests fonctionnels

But du test	Action de contrôle	Résultat attendu	Bilan
Vérifier le tri par ordre alphabétique sur les titres, des formations, le nombre de formation et les catégories de formations.	/	Le tableau des formations est classée selon la relation d'ordre spécifiée en paramètre.	Internal Server Error (HTTP 500) (fonction

			ne de façon attendue en dehors de phpunit/ WebTest Case

Tests de compatibilité

But du test	Action de contrôle	Résultat attendu	Bilan
/	/	/	Runtime Exception: Could not start chrome. Exit code: 126

Mission 3 : tâche n°2

Intitulé : créer la documentation technique

Description :

- 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).

Durée estimée : 1h

Durée effective : 1h30

🔒 mediatekformation

📄 View 1

+ New view

☰ Filter by keyword or by field

🟢 Todo 4

⋮

This item hasn't been started

🟢 mediatekformation #9

Tâche 9 : Créer la documentation utilisateur.

🟢 mediatekformation #10

Tâche 10 : déployer le site.

🟢 mediatekformation #11

Tâche 11 : gérer la sauvegarde et la restauration de la bdd.

🟢 mediatekformation #12

Tâche 12 : mettre en place le déploiement continu.

+ Add item

🟡 In Progress 1

⋮

This is actively being worked on

🟢 mediatekformation #8

Tâche 8 : créer la documentation technique.

+ Add item

🟣 Done 6

⋮

This has been completed

🟣 mediatekformation #1

Tâche 1 : nettoyer le code

🟣 mediatekformation #2

Tâche 2 : Ajouter une fonctionnalité

🟣 mediatekformation #3

Tâche 3 : gérer les formations

🟣 mediatekformation #4

Tâche 4 : gérer les playlists

🟣 mediatekformation #5

Tâche 5 : gérer les catégories.

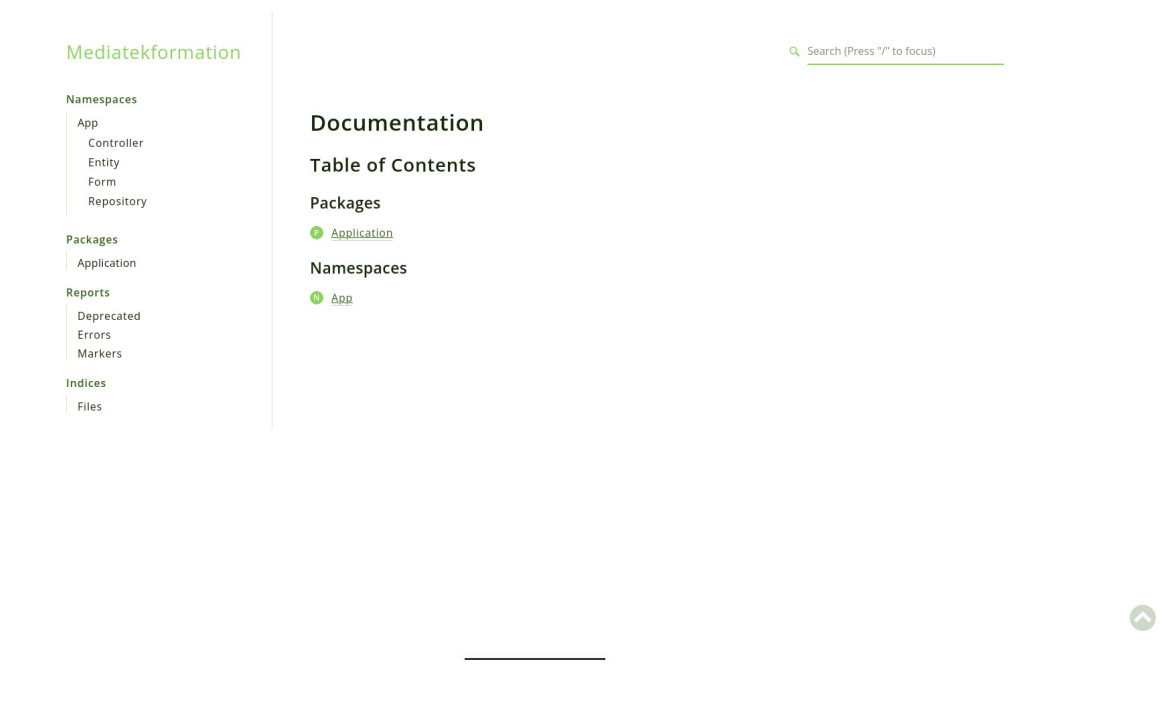
🟣 mediatekformation #6

Tâche 6 : ajouter l'accès avec authentification.

+ Add item

Kanban montrant la tâche en cours de réalisation

Aperçu de la page d'accueil de la documentation



Note : uniquement le code source ajouté par le développeur est documenté.

Mission 3 : tâche n°3

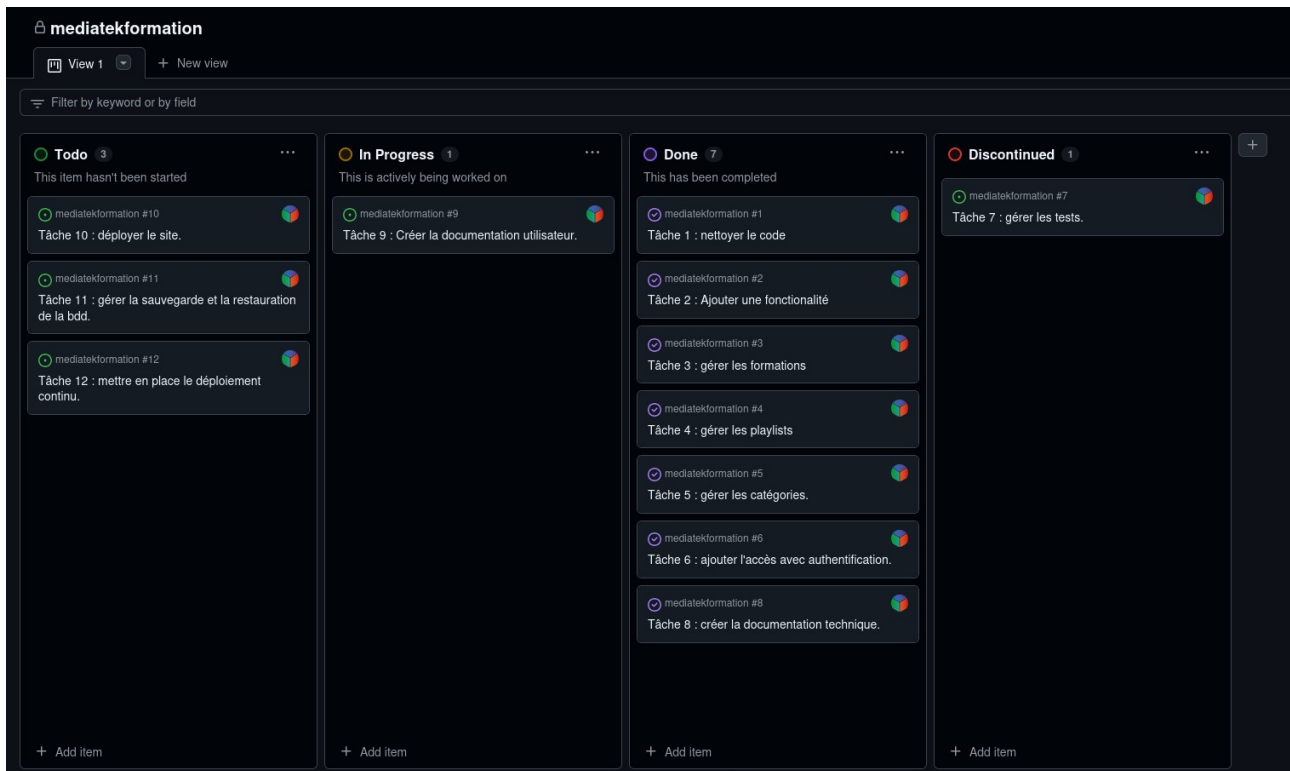
Intitulé : créer la documentation utilisateur

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

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

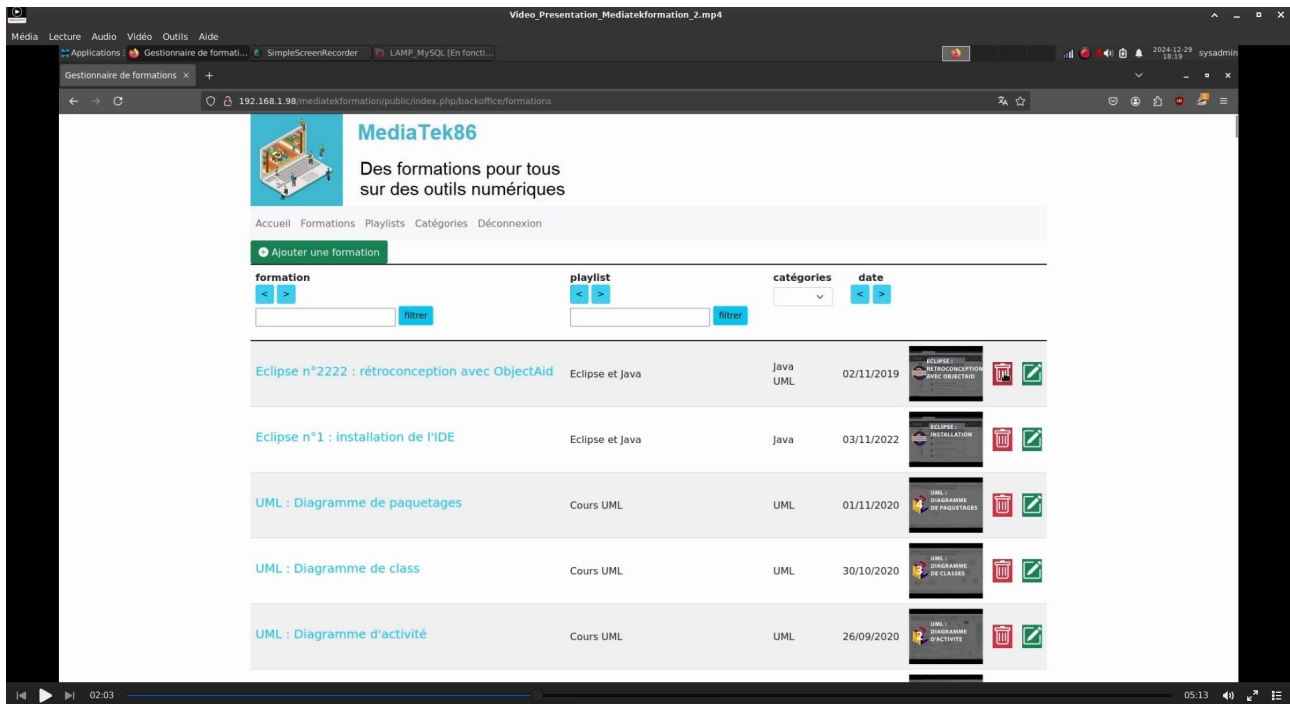
Durée estimée : 2h

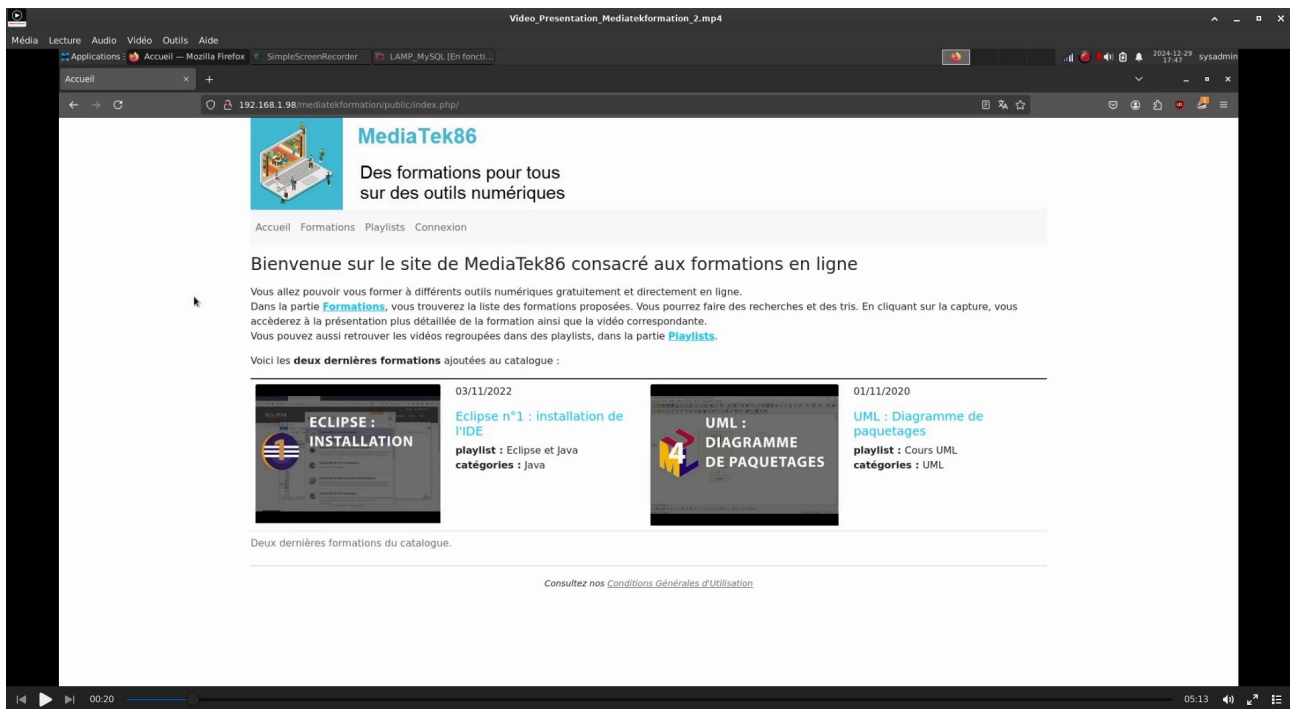
Durée effective : 1h45



Aperçu de la tâche en cours dans le kanban

Aperçus de la documentation utilisateur vidéo





Mission 4 : tâche n°1

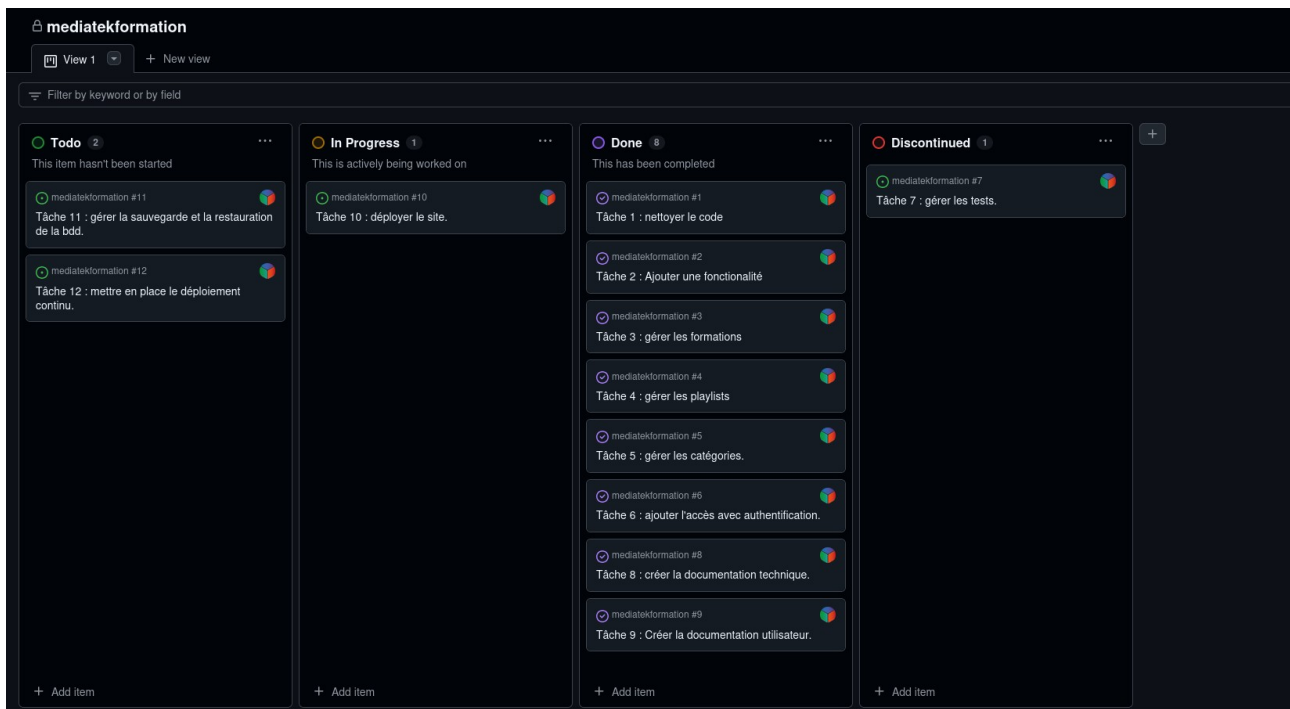
Intitulé : déployer le site

Description :

- 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.

Durée estimée : 2h

Durée effective : 2h30



Aperçu de la tâche en cours dans le kanban

Étapes de déploiement du site

1) Trouver l'hébergeur

Choix d'Infinityfree pour les critères suivants :

- Gratuit
- Sous-domaine gratuit
- Php
- MySQL
- Url rewriting

Note : Planethoster ne semble plus proposer d'offre gratuite (world lite).

2) Création de la base de données

Export de la base de données locale avec phpmyadmin puis import du fichier sql généré dans la base de données distante.

3) Préparation du fichier .env

Saisit des informations de connexion vers la base de données distante dans le fichier .env (modification de la chaîne de connexion). Note : pour des raisons de sécurité, le fichier .env a été copié et déplacé hors du projet, il ne sera jamais inclus dans les commit github.

4) Téléversement des fichiers du site via FTP(S)

L'hébergeur permet d'accéder aux fichiers du serveur web via FTP sécurisé. L'arborescence du serveur est la suivante :

```
/
=> htdocs
    => Fichiers du site ici
=> mail
    => ...
.htaccess
.override
```

Il faut donc mettre les fichiers dans le dossier htdocs.

L'hébergeur ne permettant pas d'utiliser Composer directement, il faudra également copier/coller le dossier vendor. On téléverse le site (et sa documentation) dans le dossier htdocs du serveur web.

5) Test de fonctionnement du site

Le site est opérationnel, néanmoins la connexion n'étant pas encore sécurisée, on attend de mettre en place le certificat SSL avant de saisir les informations de connexion au backoffice.

6) Installation du certificat SSL

InfinityFree permet d'installer un certificat émis par Google Trust, on génère le certificat, puis on l'installe sur le serveur web. L'hébergeur propose de le faire automatiquement via un clic dans le panel de gestion.

7) Redirection vers /public/index.php

Lorsqu'un visiteur arrive sur le site, il est automatiquement redirigé dans le dossier racine du serveur (htdocs/). Il faut donc le rediriger automatiquement vers `htdocs/public/index.php`. Pour cela, on crée le fichier `index.php` à la racine du site (`htdocs/index.php`) et on redirige le navigateur vers `htdocs/public/index.php`.

```
<?php
    header("location: https://mediatek-formations.42web.io/public/index.php");
?>
```

8) Modification des CGU

On remplace les informations des CGU par les véritables informations de l'hébergeur, on remplace également le lien vers le site.

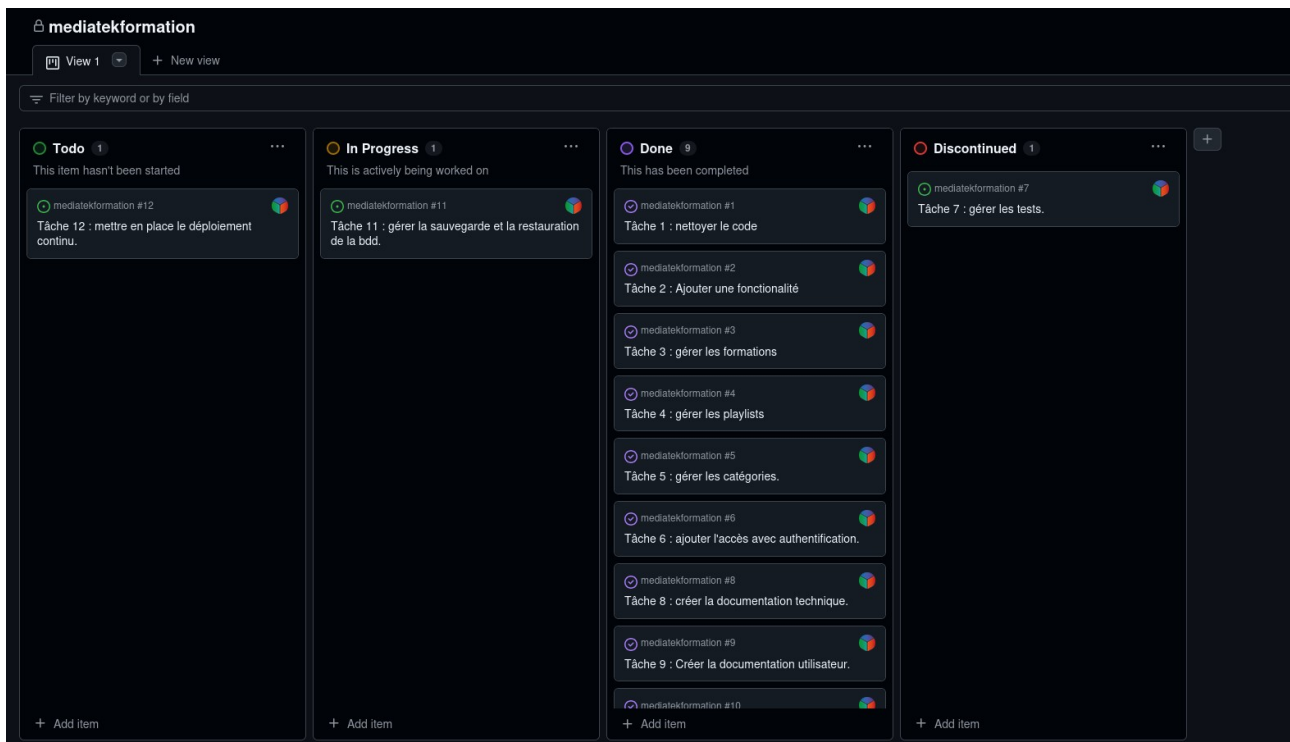
Le site est prêt à être utilisé.

Mission 4 : tâche n°2

Intitulé : gérer la sauvegarde et la restauration de la BDD

Description :

- Une sauvegarde journalière automatisée doit être programmée pour la BDD.
- La restauration pourra se faire manuellement, en exécutant le script de sauvegarde.



Aperçu du kanban de la tâche en cours

Durée estimée : 1h

Durée effective : /

La création de tâches CRON n'est plus acceptée par l'hébergeur depuis le 9 août 2023, source : <https://forum.infinityfree.com/t/cron-job-feature-is-now-disabled/80495>

Il serait toujours possible de mettre en place une sauvegarde automatisée de la base de données en utilisant le PHP.

- On créer une page php sous authentification .htaccess qui se charge de sauvegarder le contenu de la base de données.
- On paramètre un script externe pour envoyer une requête vers cette page chaque jour.

Néanmoins, il est déconseillé de déléguer la sauvegarde au serveur web pour des raisons de performance et de sécurité (possibilité de flood de sauvegarde en cas de contournement de l'authentification htaccess).

On privilégierais donc un script bash exécuté par le système, chaque jours via une tâche CRON.

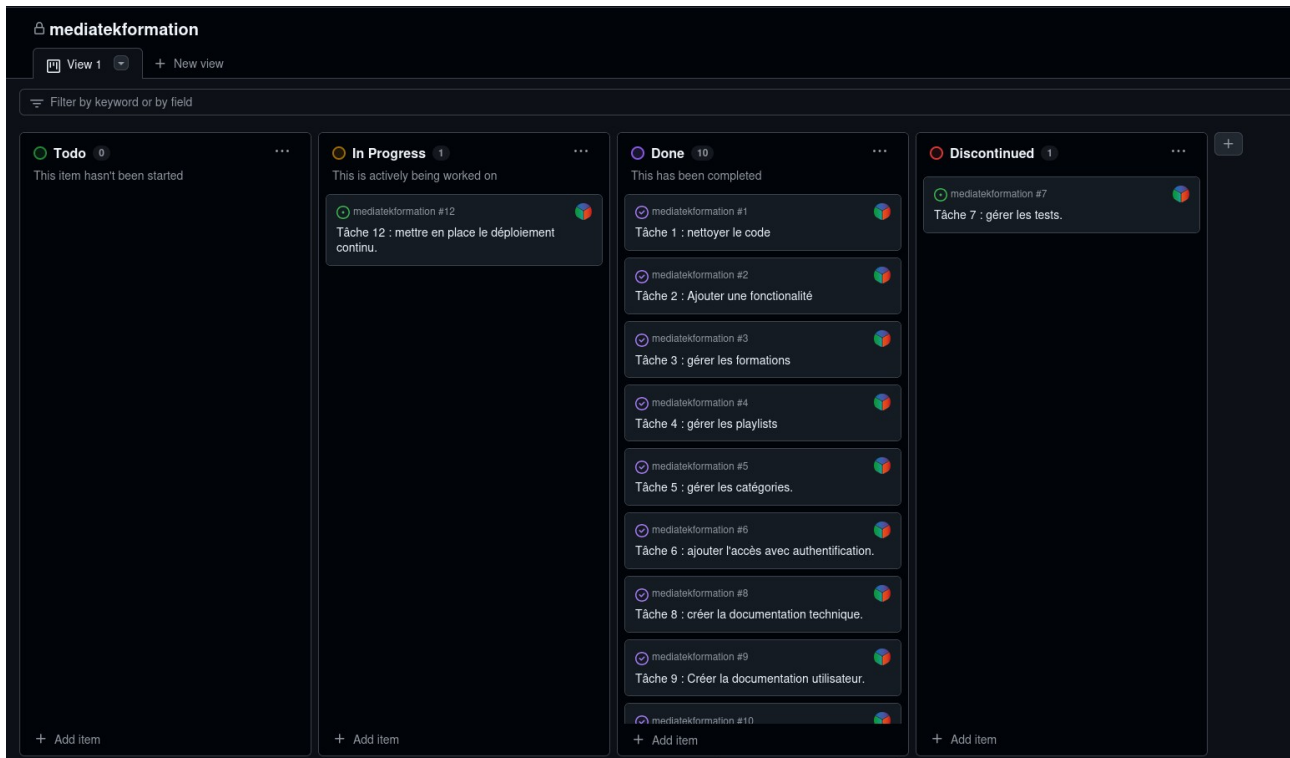
```
#!/bin/sh
DATE=`date -I`
find home/vol8_7/infinityfree.com/if0_38010723/backup/bdd* -
mtime -1 -exec rm {} \;
mysqldump -u UTILISATEUR -p MDP --databases
if0_38010723_mediatekformation --single-transaction | gzip >
home/vol8_7/infinityfree.com/if0_38010723/backup/bddbbackup_${
DATE}.sql.gz
```

Pour rétablir la base de données, il suffira de dézipper l'archive .sql.gz la plus récente, d'extraire le fichier sql, puis de l'importer via phpmyadmin du côté de l'hébergeur.

Mission 4 : tâche n°3

Intitulé : mettre en place le déploiement continu

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



Aperçu de la tâche en cours

Durée estimée : 1h

Durée effective : 40 minutes

Démarche suivie pour le déploiement continu :

1) Configuration du dépôt github distant avec la configuration yaml suivante :

```
on: push
name: Deploy website on push
jobs:
  web-deploy:
    name: Deploy
    runs-on: ubuntu-latest
    steps:
      - name: Get latest code
        uses: actions/checkout@v2
      - name: Sync files
        uses: SamKirkland/FTP-Deploy-Action@4.3.0
        with:
          server: ftpupload.net
          server-dir: /htdocs/
          username: if0_38010723
          password: ${ secrets.ftp_password }
```

2) On fait un pull sur le serveur web local pour prendre en compte l'ajout de la configuration pour le déploiement continu.

3) On modifie un fichier localement pour effectuer un push vers le dépôt distant. Désormais les fichiers du projet sont téléversés à chaque push via le protocole FTP sur le serveur web distant.

4) Test du site en ligne

- Une erreur Symfony nous indique un problème lors de la connexion à la base de données. En fait, le fichier .env a été remplacé avec les informations locales. Il suffit de téléverser le fichier et le remplacer manuellement avec la bonne chaîne de connexion.

- Plus d'erreur, le site fonctionne de façon attendue.

Aperçu de la phase de déploiement vers le site distant

Summary

Jobs

Deploy

Run details

Usage

Workflow file

Deploy

Started 41s ago

Search logs

39s

274 uploading "config/packages/security.yaml"

275 uploading "config/packages/translation.yaml"

276 uploading "config/packages/web.yaml"

277 uploading "config/packages/validator.yaml"

278 uploading "config/packages/web_profiler.yaml"

279 uploading "config/preload.php"

280 uploading "config/routes.yaml"

281 uploading "config/routes/framework.yaml"

282 uploading "config/routes/security.yaml"

283 uploading "config/routes/web_profiler.yaml"

284 uploading "config/secrets/test/test.AKISMET_KEY.ca01fb.php"

285 uploading "config/secrets/test/test.decrypt.private.php"

286 uploading "config/secrets/test/test.decrypt.public.php"

287 uploading "config/secrets/test/test.list.php"

288 uploading "config/services.yaml"

289 uploading "docs/classes/App-Controller-AccueilController.html"

290 uploading "docs/classes/App-Controller-AdminLoginController.html"

291 uploading "docs/classes/App-Controller-BackofficeCategoriesController.html"

292 uploading "docs/classes/App-Controller-BackofficeFormationsController.html"

293 uploading "docs/classes/App-Controller-BackofficePlaylistsController.html"

294 uploading "docs/classes/App-Controller-FormationsController.html"

295 uploading "docs/classes/App-Controller-FormationsController.html"

296 uploading "docs/classes/App-Controller-PlaylistsController.html"

297 uploading "docs/classes/App-Entity-Admin.html"

298 uploading "docs/classes/App-Entity-Categorie.html"

299 uploading "docs/classes/App-Entity-Formation.html"

300 uploading "docs/classes/App-Entity-Playlist.html"

301 uploading "docs/classes/App-Form-CategorieType.html"

302 uploading "docs/classes/App-Form-FormationType.html"

303 uploading "docs/classes/App-Form-PlaylistType.html"

304 uploading "docs/classes/App-Kernel.html"

305 uploading "docs/classes/App-Repository-AdminRepository.html"

306 uploading "docs/classes/App-Repository-CategorieRepository.html"

307 uploading "docs/classes/App-Repository-FormationRepository.html"

308 uploading "docs/classes/App-Repository-PlaylistRepository.html"

309 uploading "docs/css/base.css"

310 uploading "docs/css/normalize.css"

311 uploading "docs/css/template.css"

Post Get latest code

Deploy website on push

Delete src/build/api directory #2

Re-run all jobs

Summary

Jobs

Deploy

Run details

Usage

Workflow file

Triggered via push 2 minutes ago

Status

Total duration

Billable time

Artifacts

DevBlocks42 pushed 9890e6 main

Success

1m 30s

2m

-

main.yml

on: push

Deploy 1m 21s

Annotations

1 warning

Deploy

ubuntu-latest pipelines will use ubuntu-24.04 soon. For more details, see <https://github.com/actions/runner-images/issues/18636>

Bilan

Ce projet a permis de mettre en pratique les connaissances du framework Symfony, de PHP, du patron de conception MVC, l'utilisation de différents outils et logiciels concernant le développement web, notamment Apache2, Git, Netbeans, et MySQL, mais aussi des logiciels de modélisation pour le maquettage des vues ou encore pour faire des diagrammes de cas des tâches du projet. Des difficultés notables ont été rencontrées lors des tests d'intégrations et de compatibilité, le crawler produisait une longue pile d'exceptions et une erreur http 500 (internal server error) qui empêchait de lire la table html et de tester le résultat attendu (pour les filtres). Par ailleurs, les tests de compatibilité n'ont pas non plus été possibles car ils produisaient une erreur lors du lancement des drivers de navigateurs (peut-être car le développement a été fait sur une machine virtuelle locale utilisant debian sans interface graphique). D'autre part, la mise en place de la sauvegarde quotidienne de la base de données n'a pas pu être réalisée à cause de limites de l'hébergeur concernant les tâches CRON.

À la fin, on obtient une application web avec toutes les fonctionnalités attendues, disponible en ligne via l'url suivante : <https://mediatek-formations.42web.io>

Le déploiement continu permet de modifier le site distant directement après un push depuis le dépôt local, facilitant ainsi l'évolution de l'application dans le futur.