

# Atelier 1 – Compte rendu global

## Sommaire :

**Mission 1** : Nettoyer et optimiser le code existant (Page 2-8)

**Mission 2** : Coder la partie back-office (Page 3)

**Mission 3** : Tester et documenter (Page 4)

**Mission 4** : Déployer le site et gérer le déploiement continu (Page 5)

## Contexte de la réalisation professionnelle

Dans le cadre de ma formation en BTS SIO (option SLAM), j'ai participé à un projet de développement web dans un contexte simulé avec une fausse entreprise, qui se nomme **InfoTech Services 86 (ITS 86)**, une Entreprise de Services Numériques (ESN) implantée dans la Vienne. Cette société, fondée en 2002, elle est spécialisée dans le développement d'applications (web, mobiles, logiciels), l'infogérance, l'hébergement de sites, la gestion de parc informatique ainsi que l'ingénierie système et réseau.

## Objectif du projet

Mon projet consistait à faire évoluer une application Symfony destinée à la plateforme **MediatekFormation**, développée pour le réseau de médiathèques MediaTek86. Cette plateforme a pour vocation de mettre à disposition du public des vidéos d'auto-formation accessibles gratuitement en ligne.

L'application n'existait pas encore en ligne dans sa version finale. Une version de base du projet Symfony ainsi qu'une base de données MySQL étaient fournies en amont. Mon rôle était d'assurer l'intégration de nouvelles fonctionnalités, la correction de bugs, et l'optimisation du code, en veillant à respecter les bonnes pratiques de développement et la charte graphique fournie.

## Conditions de réalisation

- **Période** : du 1er mars au 15 mars 2025
- **Encadrement** : Projet réalisé individuellement
- **Organisation** : Dans le cadre de la formation, en collaboration avec les consignes techniques de l'ESN ITS 86
- **Ressources fournies** : Code Symfony existant, script de BDD MySQL, expression des besoins, consignes graphiques, description fonctionnelle complète

## Environnement technique utilisé

- **Langages & Frameworks** : PHP / Symfony, HTML, CSS, JavaScript
- **Base de données** : MySQL

- **IDE** : NetBeans
- **Outils de versioning** : Git & GitHub
- **Méthodologie de test** : Tests unitaires
- **Documentation** : Technique (en ligne) et utilisateur (sous forme vidéo)

Ce projet m'a permis de renforcer mes compétences en développement web back-end, de pratiquer la lecture et l'amélioration de code existant, et d'appliquer une démarche professionnelle complète : de la compréhension du besoin jusqu'à la documentation finale.

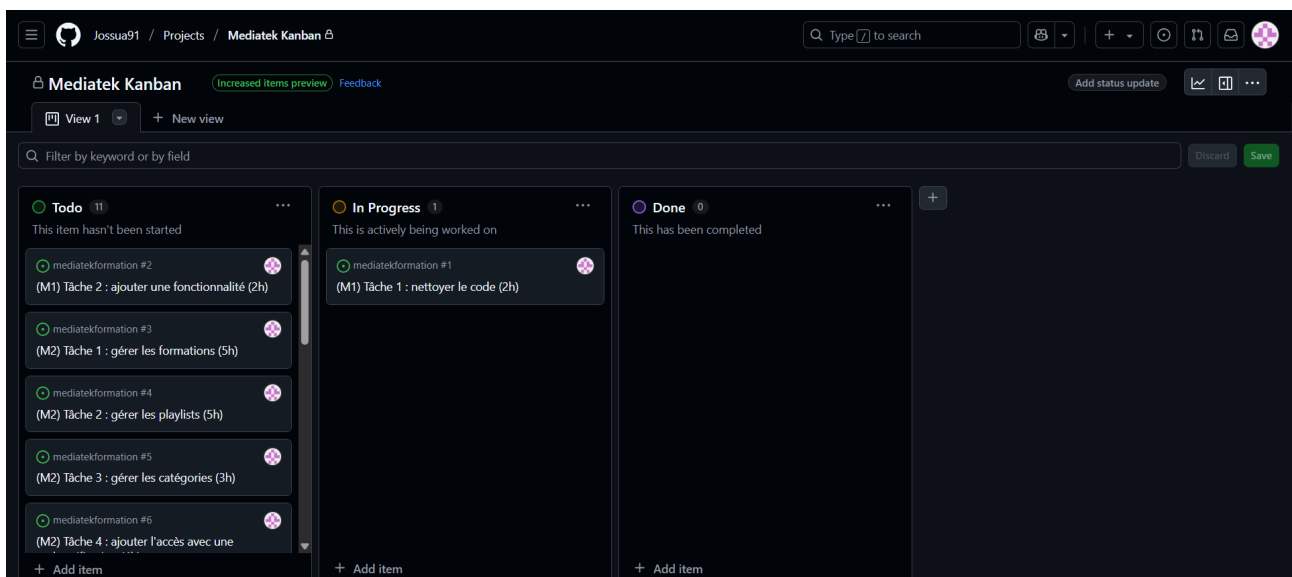
## Missions :

**Mission 1** : Nettoyer et optimiser le code existant

**Tâche 1** : Nettoyer le code

Demande : Nettoyer le code en suivant les indications de SonarLint.

Kanban :

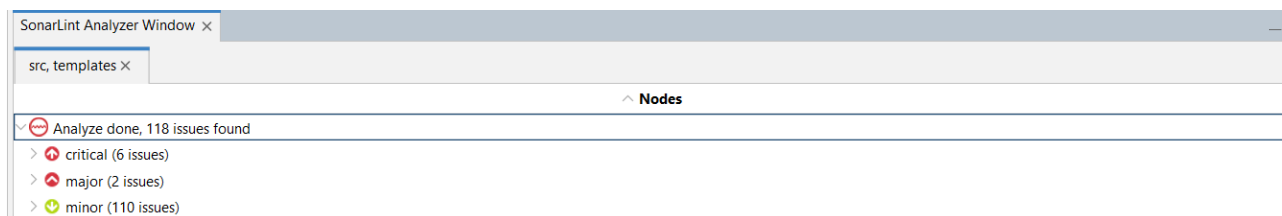


Temps estimé : 2 heures

Temps réel : 2 heures

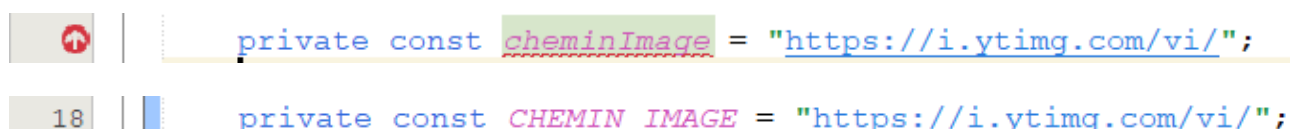
Problèmes :

On commence par lancer le scan de code à l'aide de SonarLint sur les dossiers src et templates, qui sont concernés par les modifications des développeurs. On obtient ce résultat :



[Critical] :

- php:S115 : Constant names should comply with a naming convention (1) | 18:18 : Formation.php



Le problème ici était que le nom de cette constante ne respectait pas les conventions de nommage en PHP. Les constantes doivent être écrites en majuscules et avec des underscores. Il a fallu également modifier 2 autres lignes plus bas dans le code pour y remplacer le nom de cette constante par le nouveau.

- php:S1192 : String literals should not be duplicated (2)

39:29 : FormationsController.php :



```

37 | #[Route('/formations', name: 'formations')]
38 | public function index(): Response{
39 |     $formations = $this->formationRepository->findAll();
40 |     $categories = $this->categorieRepository->findAll();
41 |     return $this->render(self::FORMATIONS_PATH, [
42 |         'formations' => $formations,
43 |         'categories' => $categories
44 |     ]);
45 | }
46 |
47 | #[Route('/formations/tri/{champ}/{ordre}/{table}', name: 'formations.sort')]
48 | public function sort($champ, $ordre, $table=""): Response{
49 |     $formations = $this->formationRepository->findAllOrderBy($champ, $ordre, $table);
50 |     $categories = $this->categorieRepository->findAll();
51 |     return $this->render(self::FORMATIONS_PATH, [
52 |         'formations' => $formations,
53 |         'categories' => $categories
54 |     ]);
55 | }

```

53:29 : PlaylistsController.php :

```

49 | #[Route('/playlists', name: 'playlists')]
50 | public function index(): Response{
51 |     $playlists = $this->playlistRepository->findAllOrderByName('ASC');
52 |     $categories = $this->categorieRepository->findAll();
53 |     return $this->render("pages/playlists.html.twig", [
54 |         'playlists' => $playlists,
55 |         'categories' => $categories
56 |     ]);
57 | }
58 |

```

```

37 | private const PLAYLISTS_PATH = "pages/playlists.html.twig";

```

```

51 | #[Route('/playlists', name: 'playlists')]
52 | public function index(): Response{
53 |     $playlists = $this->playlistRepository->findAllOrderByName('ASC');
54 |     $categories = $this->categorieRepository->findAll();
55 |     return $this->render(self::PLAYLISTS_PATH, [
56 |         'playlists' => $playlists,
57 |         'categories' => $categories
58 |     ]);
59 | }

```

Ici les problèmes étaient que le chemin était utilisé plusieurs fois, une manière optimisée est de le mettre dans une constante et de l'utiliser là où il y en a besoin et permet de modifier une seule fois le chemin si on a besoin.

- php:S121 : Control structures should use curly braces (1) & php:S3973 : A conditionally executed single line should be denoted by indentation (1) | 103:12 : Playlist.php

```

98 public function getCategoriesPlaylist() : Collection
99 {
100     $categories = new ArrayCollection();
101     foreach($this->formations as $formation){
102         $categoriesFormation = $formation->getCategories();
103         foreach($categoriesFormation as $categorieFormation)
104             if(!$categories->contains($categorieFormation->getName())){
105                 $categories[] = $categorieFormation->getName();
106             }
107     }
108     return $categories;
109 }

```

```

98 public function getCategoriesPlaylist() : Collection
99 {
100     $categories = new ArrayCollection();
101     foreach($this->formations as $formation){
102         $categoriesFormation = $formation->getCategories();
103         foreach($categoriesFormation as $categorieFormation) {
104             if(!$categories->contains($categorieFormation->getName())){
105                 $categories[] = $categorieFormation->getName();
106             }
107         }
108     }
109     return $categories;
110 }

```

Ici les problèmes étaient que le foreach n'avait pas ses accolades ouvrantes et fermantes et donc le if n'était pas pris en compte dans le foreach.

- phpS131 : « switch » statements should have « default » clauses (1) | 63:8 :  
PlaylistsController.php

```

61 #[Route('/playlists/tri/{champ}/{ordre}', name: 'playlists.sort')]
62 public function sort($champ, $ordre): Response{
63     switch($champ){
64         case "name":
65             $playlists = $this->playlistRepository->findAllOrderByName($ordre);
66             break;
67     }
68     $categories = $this->categorieRepository->findAll();
69     return $this->render(self::PLAYLISTS_PATH, [
70         'playlists' => $playlists,
71         'categories' => $categories
72     ]);
73 }

```

```

61      #[Route('/playlists/tri/{champ}/{ordre}', name: 'playlists.sort')]
62      public function sort($champ, $ordre): Response{
63          switch($champ){
64              case "name":
65                  $playlists = $this->playlistRepository->findAllOrderByName($ordre);
66                  break;
67              default:
68                  break;
69          }
70          $categories = $this->categorieRepository->findAll();
71          return $this->render(self::PLAYLISTS_PATH, [
72              'playlists' => $playlists,
73              'categories' => $categories
74          ]);
75      }

```

Ici le switch n'avait pas de valeur par default, ce qui est obligatoire pour cette fonctionnalité dans php.

[Major] :

- php:S1066 : Collapsible « if » statements should be merged (1) | 87:12 : Playlist.php

```

83      public function removeFormation(Formation $formation): static
84      {
85          if ($this->formations->removeElement($formation)) {
86              // set the owning side to null (unless already changed)
87              if ($formation->getPlaylist() === $this) {
88                  $formation->setPlaylist(null);
89              }
90          }
91
92          return $this;
93      }

```

```

83      public function removeFormation(Formation $formation): static
84      {
85          if ($this->formations->removeElement($formation) && $formation->getPlaylist() === $this) {
86              // set the owning side to null (unless already changed)
87              $formation->setPlaylist(null);
88          }
89
90          return $this;
91      }

```

Ici le problème était qu'il y avait deux « if » d'une ligne à l'autre, alors qu'ils auraient pu être fusionner en un seul.

[Minor] :

- Web:ImgWithoutAltCheck : Image, area and button with image tags should have an « alt » attribute (4) | 25:36 : accueil.html.twig & 9:12: basefront.html.twig & 80:32 : formations.html.twig & 23:32 : playlist.html.twig

```

25      

```

```
25 
27 </a>
```

Ici le problème était que les images n'avait l'attribut alt qui permet de décrire l'image.

- Web:TableWithoutCaptionCheck : « <table> » tags should have a description (3)




```

14  <p>
15  Voici les <strong>deux dernières formations</strong> ajoutées au catalogue :
16  <table class="table">
17    <tr>
18      {% for formation in formations %}
19        <td>
20          <div class="row">
21
22  <table class="table">
23    <caption>Liste des formations disponibles :</caption>

```

Ici le problème était qu'il fallait ajouter une description au tableau, avec caption.

Le reste des Minor comprend simplement des espaces, ainsi que d'autres choses qui ne sont que mineurs.

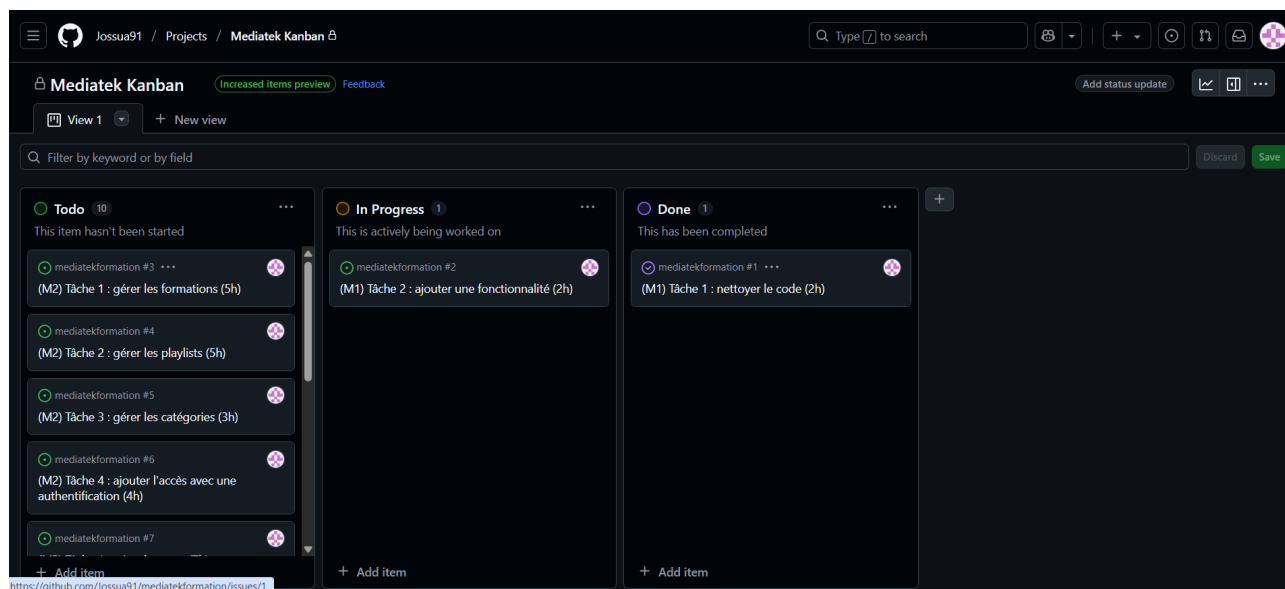
- ✓  Analyze done, 103 issues found
  - ✓  major (1 issue)
    - >  Web:S5256 : Tables should have headers (1)

On se retrouve donc avec plus aucuns Critical et plus qu'une major qui nous demande d'ajouter un header à une table.

## Tâche 2 : Ajouter une fonctionnalité

Demande : 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.

### Kanban :



Temps estimé : 2 heures  
Temps réel : 1 heure

Accueil Formations Playlists			
<b>Playlist</b> <div><div>&lt; &gt;</div><div><input type="text"/></div><div>filtrer</div></div>	<b>Nombre de formations</b> <div><div>&lt; &gt;</div></div>	<b>Catégories</b> <div><input type="text"/></div>	
Bases de la programmation (C#)	74	C# POO	Voir détail
Compléments Android (programmation mobile)	13	Android	Voir détail
Cours Composant logiciel	2	Cours	Voir détail
Cours Curseurs	2	SQL Cours POO	Voir détail
Cours de programmation objet	1	POO Cours	Voir détail
Cours Informatique embarquée	1	Cours	Voir détail
Cours MCD MLD MPD	2	MCD Cours	Voir détail
Cours MCD vs Diagramme de classes	2	MCD Cours	Voir détail



Playlist




filtrer

Nombre de formations



Catégories

playlist test	0		<a href="#">Voir détail</a>
Cours Informatique embarquée	1	Cours	<a href="#">Voir détail</a>
Cours Merise/2	1	MCD Cours	<a href="#">Voir détail</a>
Cours Modèle relationnel et MCD	1	MCD Cours	<a href="#">Voir détail</a>
Cours de programmation objet	1	POO Cours	<a href="#">Voir détail</a>
Cours Composant logiciel	2	Cours	<a href="#">Voir détail</a>
Cours MCD MLD MPD	2	MCD Cours	<a href="#">Voir détail</a>
Cours MCD vs Diagramme de classes	2	MCD Cours	<a href="#">Voir détail</a>

Ici j'ai ajouter une colonne dans le playlists.html.twig pour mettre le nombre de formations dans chaque playlists. Pour savoir combien il y avait de formations, j'ai ajouter une fonction `findAllOrderByNbFormations` dans le `PlaylistRepository` qui récupère le nombre de formations dans une playlists. Pour le tri j'ai simplement ajouter une case `nbFormations` qui permet de trier par nombre de formations.

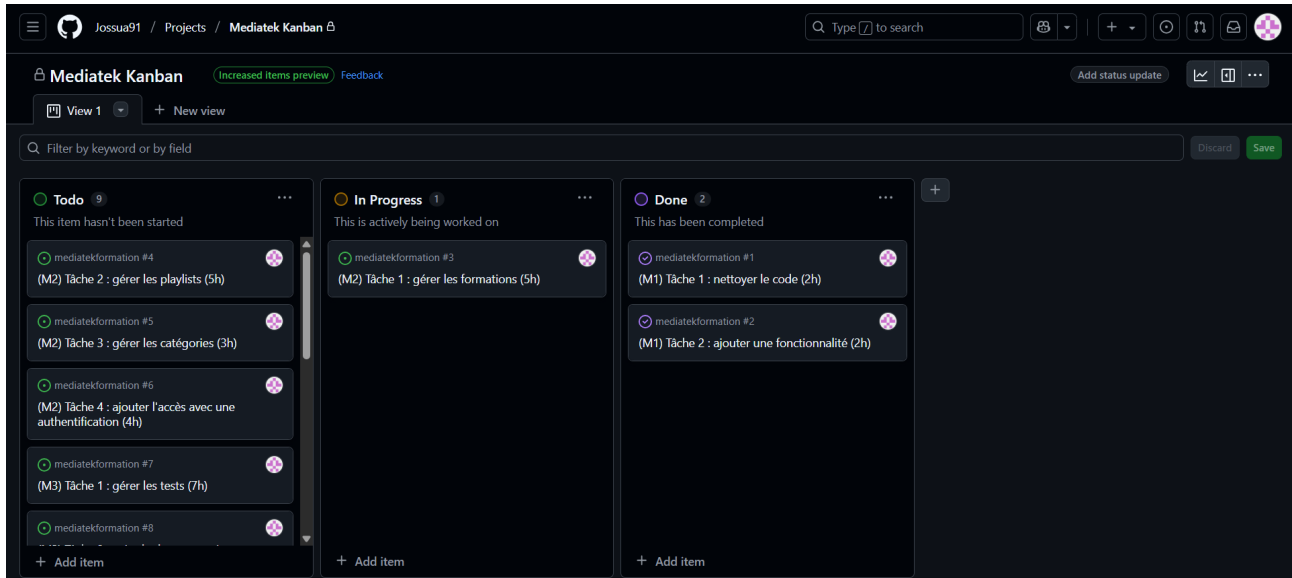
En ce qui concerne la page `playlist.html.twig` j'ai simplement ajouter une ligne pour le nombre de formations qui était déjà possible de récupérer sans modifications avec `playlistformations|length`, puisque `playlistformations` était exploiter en dessous pour afficher les formations.

## Mission 2 : Coder la partie back-office

### Tâche 1 : Gérer les formations

Demande : Gérer les formations, pouvoir les supprimer, ajouter et modifier avec un formulaire.

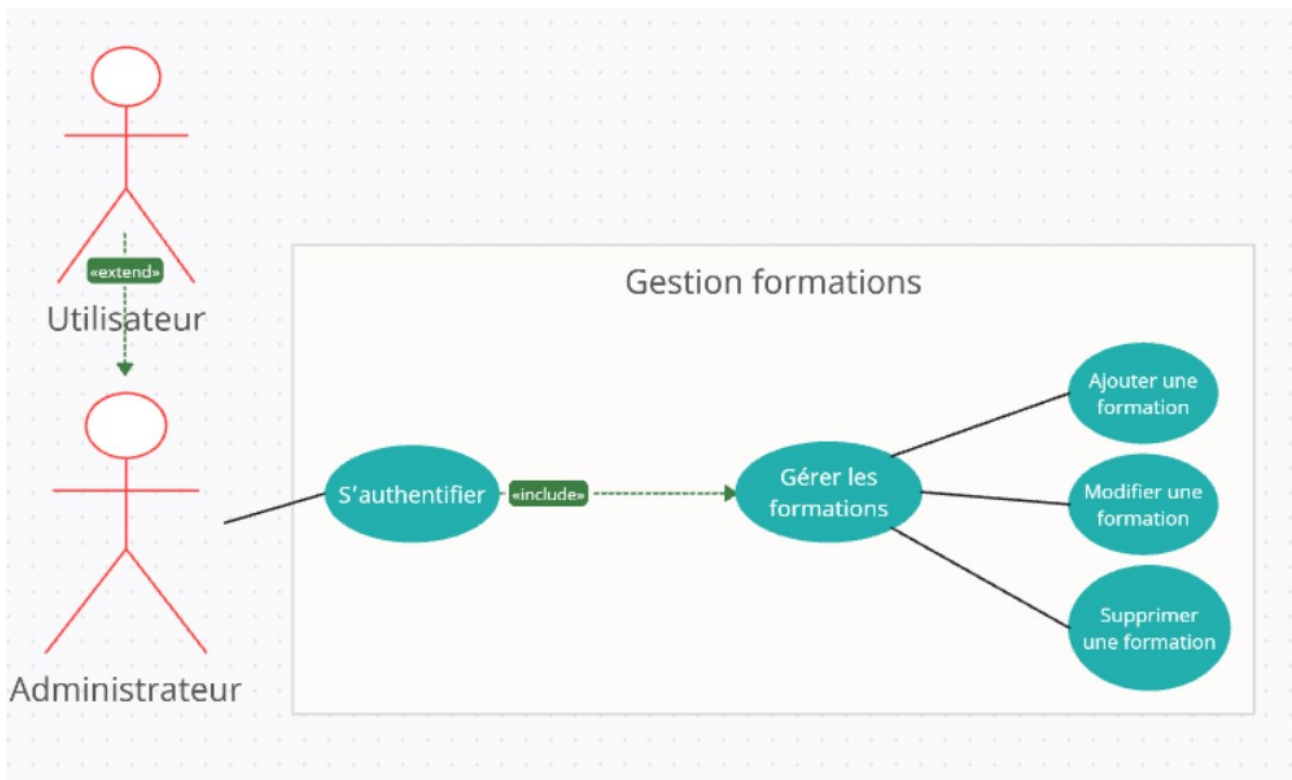
Kanban :



Temps estimé: 5 heures

Temps réel : 3 heures

Diagramme de cas d'utilisation :



## Maquettes et explications :

Maquette de la page d'administration des formations. Le titre principal est "Maquettes". À droite, il y a un bouton "Ajouter une formation". En dessous, il y a une table avec cinq colonnes : "Formation :", "Playlist :", "Catégories :", "Date :", et "Actions :". La colonne "Actions :" contient trois boutons : "Voir détail", "Modifier", et "Supprimer".

/pages/admin/formations :

- formations.html.twig : comporte toutes les formations et ses actions
- add.html.twig : correspond à la page d'ajout d'une formation
- edit.html.twig : correspond à la page de modification d'une formation
- form.html.twig : correspond au formulaire qui est appelé par add et edit

- Le bouton ajouter une formation envoie vers un formulaire pour remplir les informations de la formation
- Le bouton voir détail amène vers la page de la formation
- Le bouton modifier amène vers un formulaire pré rempli avec les informations de la formation
- Le bouton supprimer demande une confirmation puis supprime la formation

Il a fallu ici ajouter un contrôleur avec toutes les fonctions déjà présentes dans formations de base, plus les fonctions d'ajout, modification et suppression pour les formations.

## Rendu final :

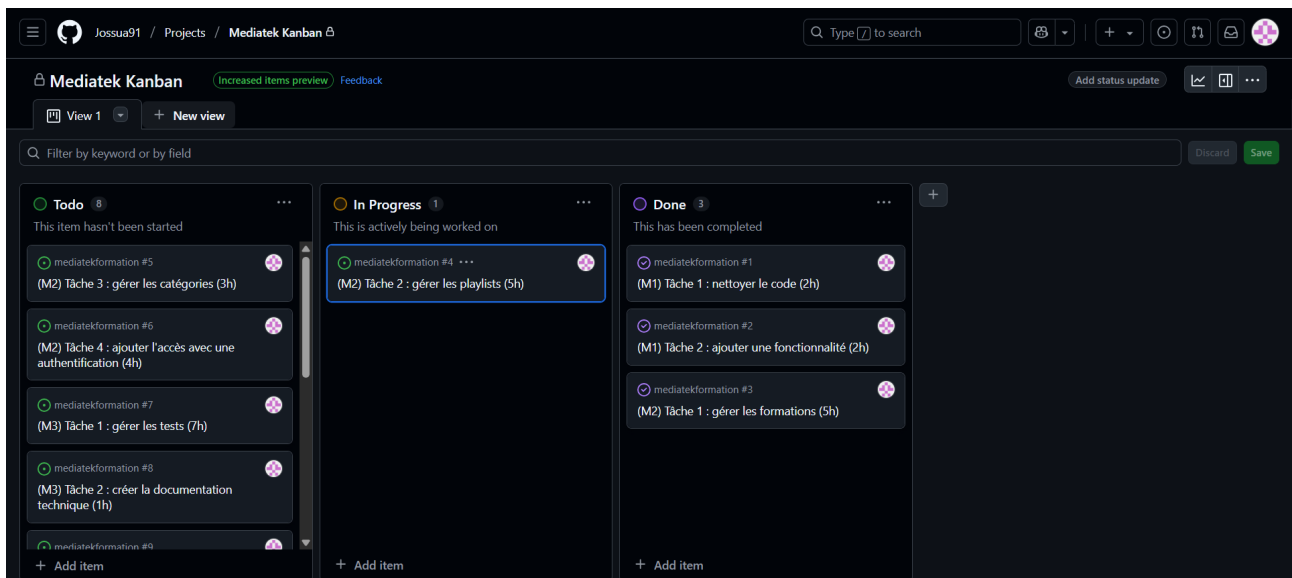
Rendu final de la page d'administration des formations. Le titre principal est "Maquettes". À droite, il y a un bouton "Ajouter une formation". En dessous, il y a une table avec cinq colonnes : "Formation", "Playlist", "Catégories", "Date", et "Actions". La colonne "Formation" contient des boutons "<" et ">" et un champ de recherche avec un bouton "filtrer". La colonne "Playlist" contient des boutons "<" et ">" et un champ de recherche avec un bouton "filtrer". La colonne "Catégories" contient un menu déroulant. La colonne "Date" contient des boutons "<" et ">". La colonne "Actions" contient trois boutons : "Voir détail", "Modifier", et "Supprimer".

Formation	Playlist	Catégories	Date	Actions
Eclipse n°6 : Refactoring	Eclipse et Java	Java UML	30/12/2020	Voir détail Modifier Supprimer

## Tâche 2 : Gérer les playlists

Demande : Gérer les playlists, pouvoir les supprimer, ajouter et modifier avec un formulaire.

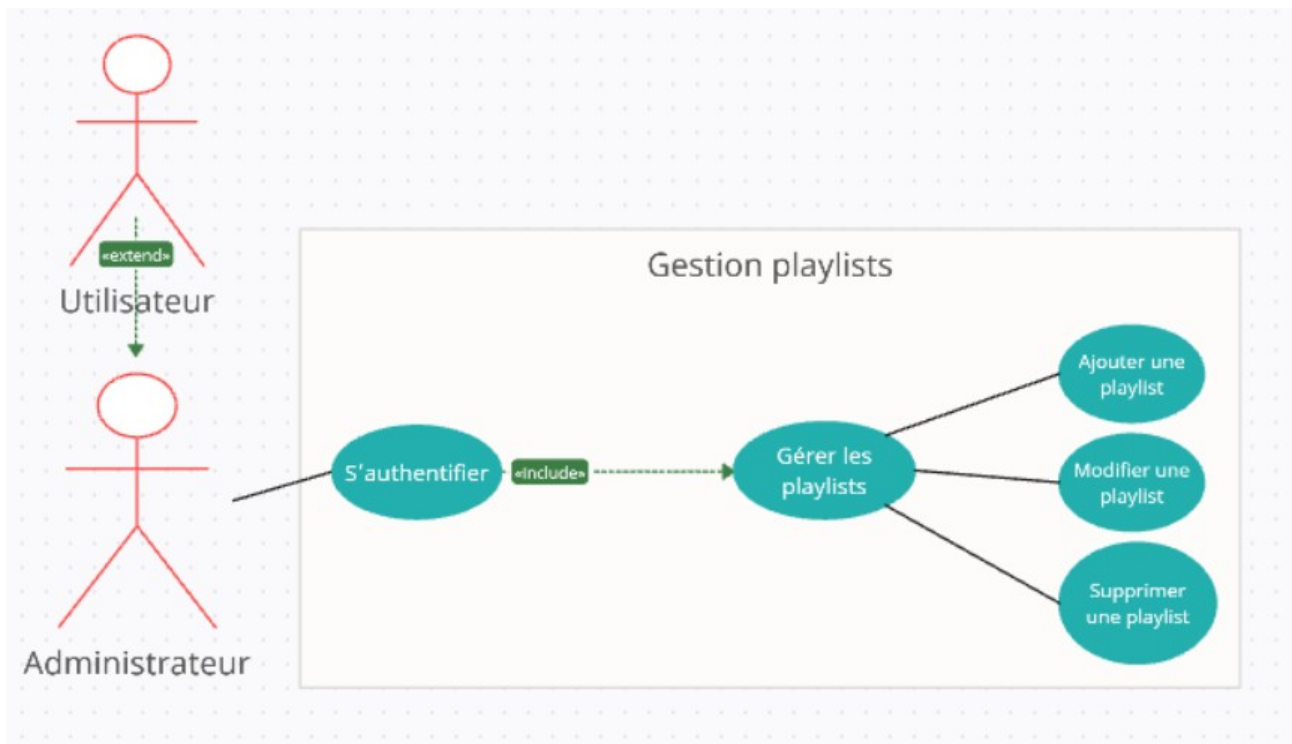
## Kanban :



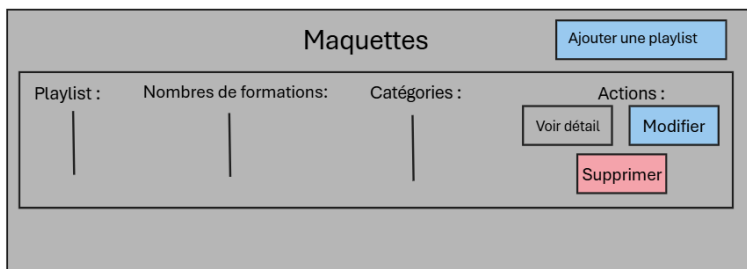
Temps estimé: 5 heures

Temps réel : 2 heures

Diagramme de cas d'utilisation :



## Maquettes et explications :



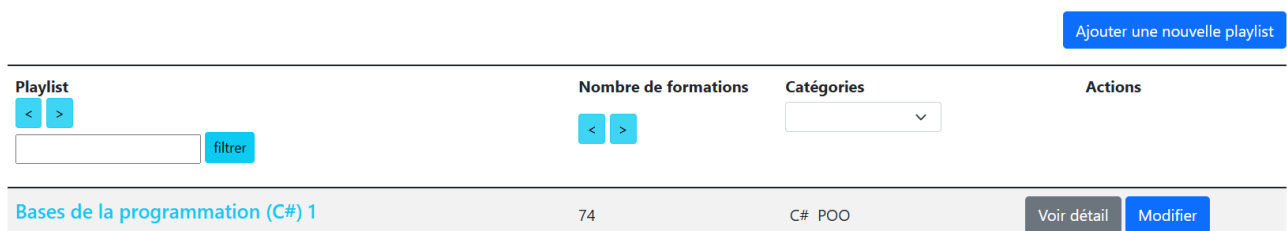
/pages/admin/playlists :

- playlists.html.twig : comporte toutes les playlists et ses actions
- add.html.twig : correspond à la page d'ajout d'une playlist
- edit.html.twig : correspond à la page de modification d'une playlist
- form.html.twig : correspond au formulaire qui est appelé par add et edit

- Le bouton ajouter une playlist envoie vers un formulaire pour remplir les informations de la playlist
- Le bouton voir détail amène vers la page de la playlist
- Le bouton modifier amène vers un formulaire pré rempli avec les informations de la playlist
- Le bouton supprimer est affiché seulement si aucune formation n'est rattachée à elle, puis demande une confirmation puis supprime la playlist

Il a fallu ici ajouter un contrôleur avec toutes les fonctions déjà présentes dans les playlists de base, plus les fonctions d'ajout, modification et suppression pour les playlists.

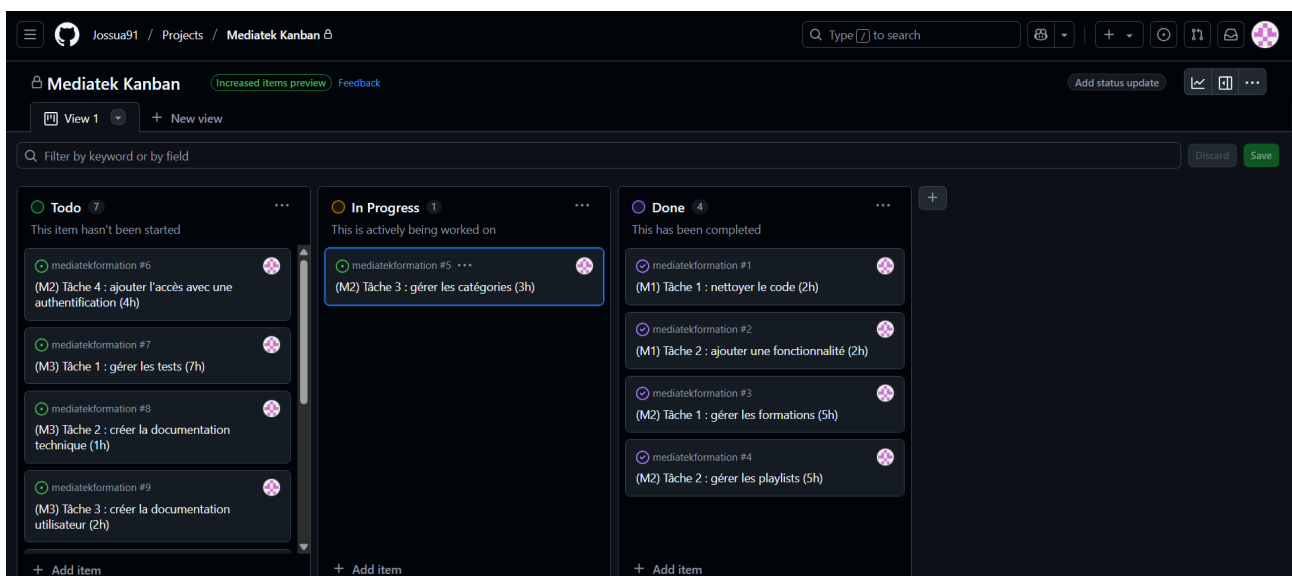
## Rendu final :



## Tâche 3 : Gérer les catégories :

Demande : Gérer les catégories, pouvoir les supprimer et en ajouter avec un formulaire.

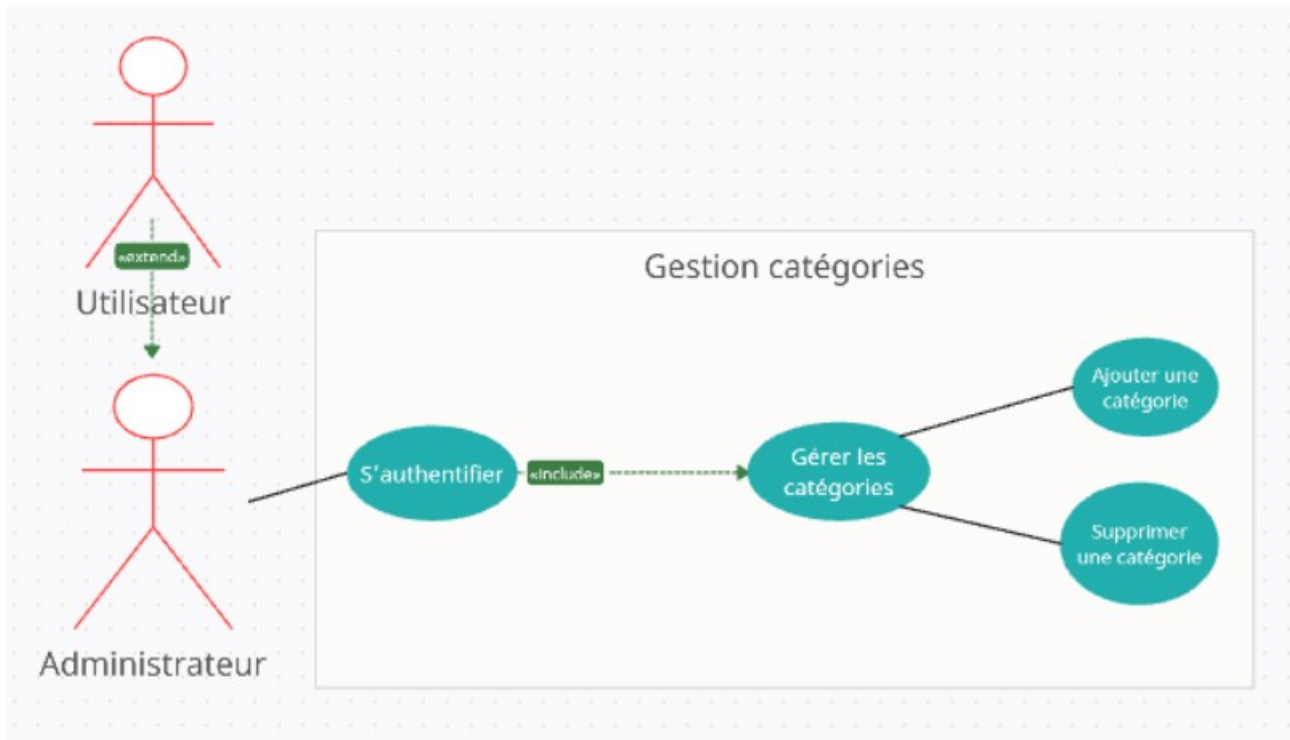
## Kanban :



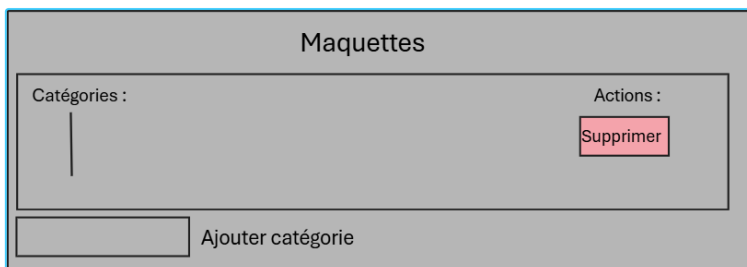
Temps estimé: 3 heures

Temps réel : 1 heure

Diagramme de cas d'utilisation :



Maquettes et explications :



/pages/admin/categories :

- categories.html.twig : comporte toutes les categories et ses actions, dont le mini formulaire pour ajouter une catégories avec son nom

- Le bouton supprimer demande une confirmation puis supprime la catégorie

Il a fallu ici ajouter un controller avec toutes les fonctions déjà présentes dans catégories de base, plus les fonctions d'ajout, suppression pour les catégories.

Rendu final :

Ajouter une catégorie :

Catégories

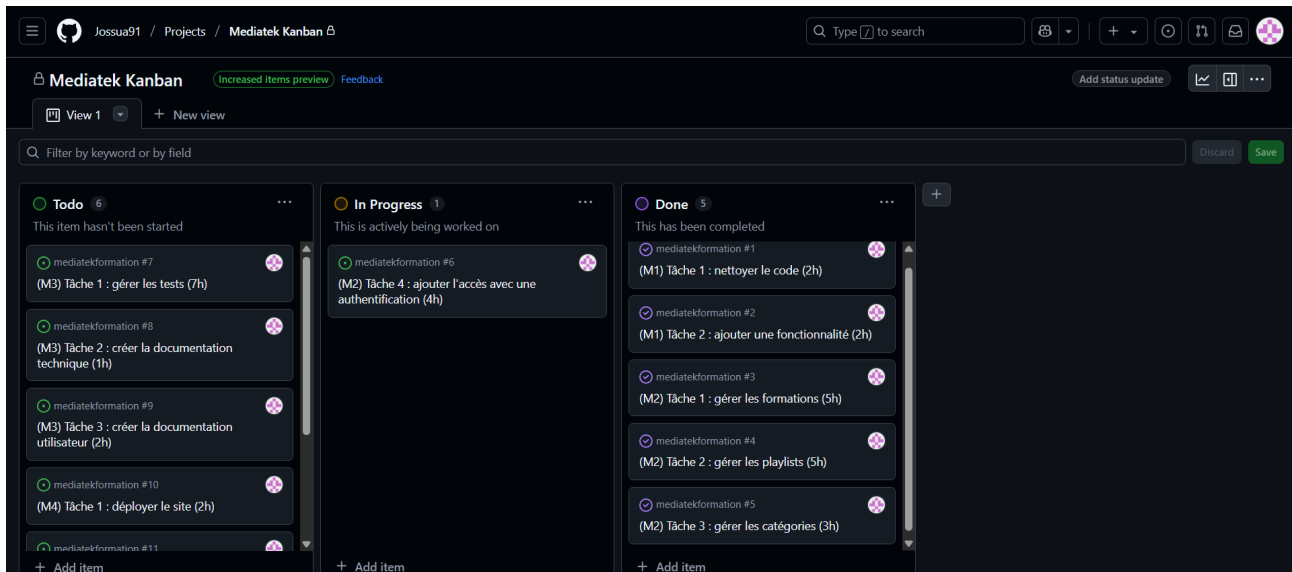
Actions

Java

## Tâche 4 : Ajouter l'accès avec authentification

Demande : Gérer les catégories, pouvoir les supprimer et en ajouter avec un formulaire.

Kanban :



Temps estimé: 4 heures

Temps réel : 3 heures

Rendu final :



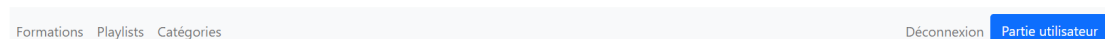
Une fois connecter :



Le bouton déconnecter nous déconnecte, le bouton gestion administrateur nous permet d'accéder à la page de gestion des formations, playlists et catégories.

Puis une fois sur la partie administration :

## Gestion administrateur

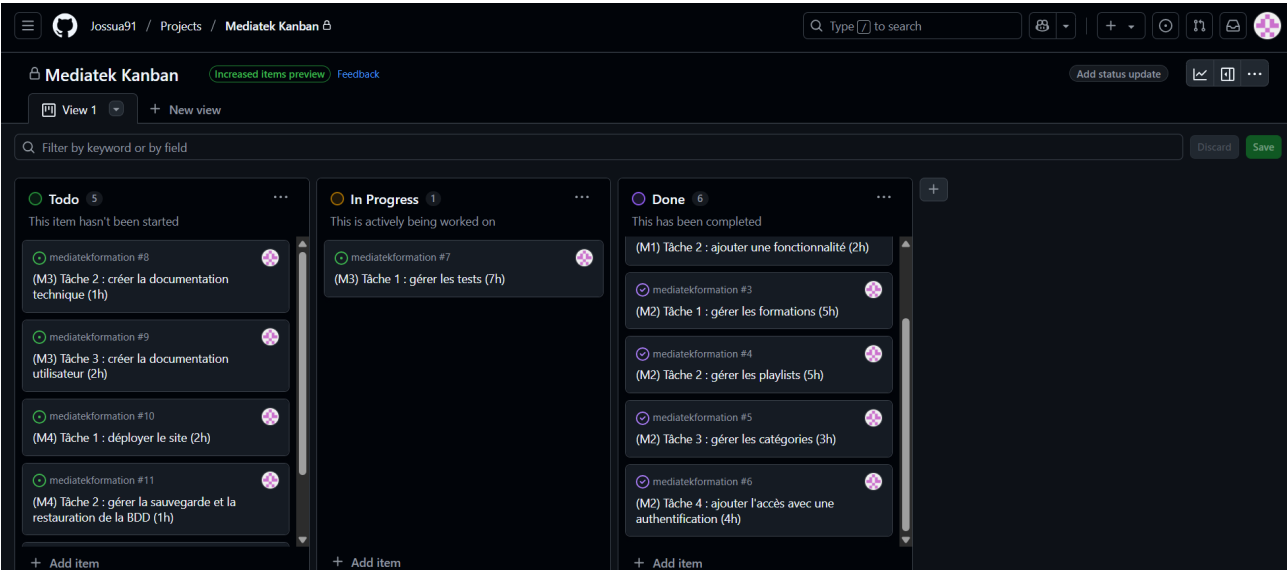


Mission 3 : Tester et documenter

Tâche 1 : Gérer les tests

Demande : Réaliser une série de tests en remplissant un plan de tests.

Kanban :



Temps estimé: 7 heures

Temps réel : 5 heures

Plans de test :

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 : 2025-01-04	04/01/2025	OK

Tests d'intégration

But du test	Action de contrôle	Résultat attendu	Bilan
Lors de l'ajout ou de la modification d'une formation, contrôler que la date n'est pas postérieure à aujourd'hui.	Test d'intégration lancé	Aucunes erreurs	OK
Contrôler les méthodes ajoutées les classes <code>Repository</code> ( <code>Formations</code> , <code>Catégories</code> , <code>Playlists</code> )	Test d'intégration lancé	Aucunes erreurs	OK

Tests fonctionnels

But du test	Action de contrôle	Résultat attendu	Bilan
Contrôler que la page d'accueil est accessible.	Tests fonctionnels lancé	Aucunes erreurs	OK
Contrôler que les tris et filtres fonctionnent.	Tests fonctionnels lancé	Aucunes erreurs	OK
Contrôler que le clic sur un lien dans une liste permet d'accéder à la bonne page.	Tests fonctionnels lancé	Aucunes erreurs	OK

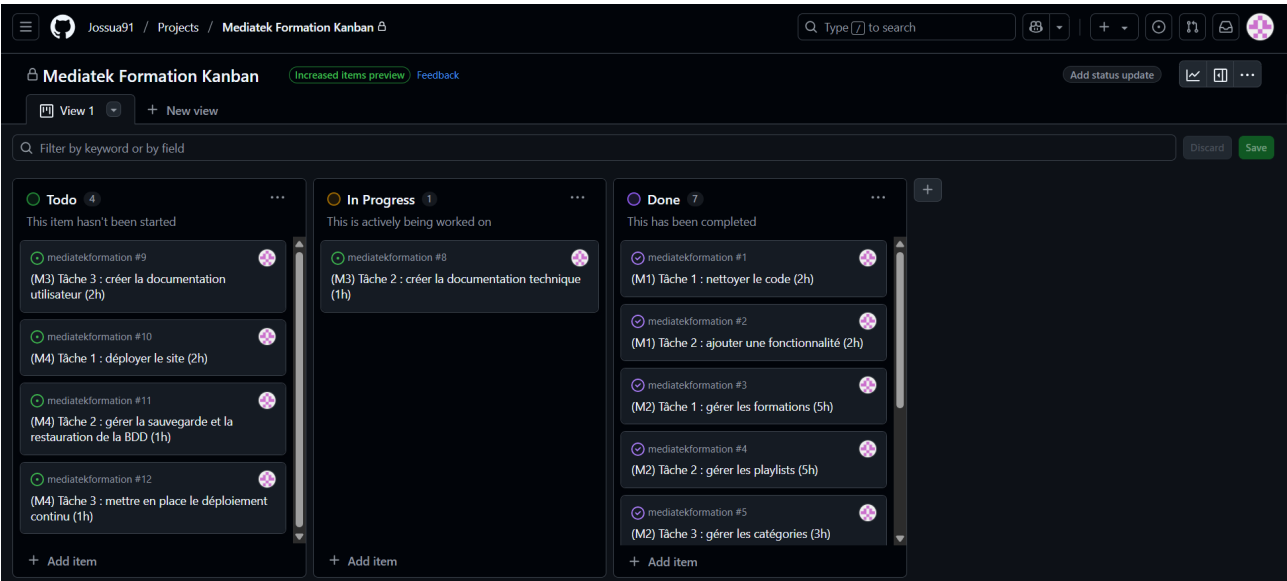


## Tests de compatibilité

But du test	Action de contrôle	Résultat attendu	Bilan
Tester le site sur plusieurs navigateurs pour contrôler la compatibilité.	Tests fait sur plusieurs navigateurs (Chrome, Edge, Firefox)	Aucuns problèmes	OK

### Tâche 2 : Créer la documentation technique

Demande : Création d’une documentation technique portant sur l’ensemble de l’application.



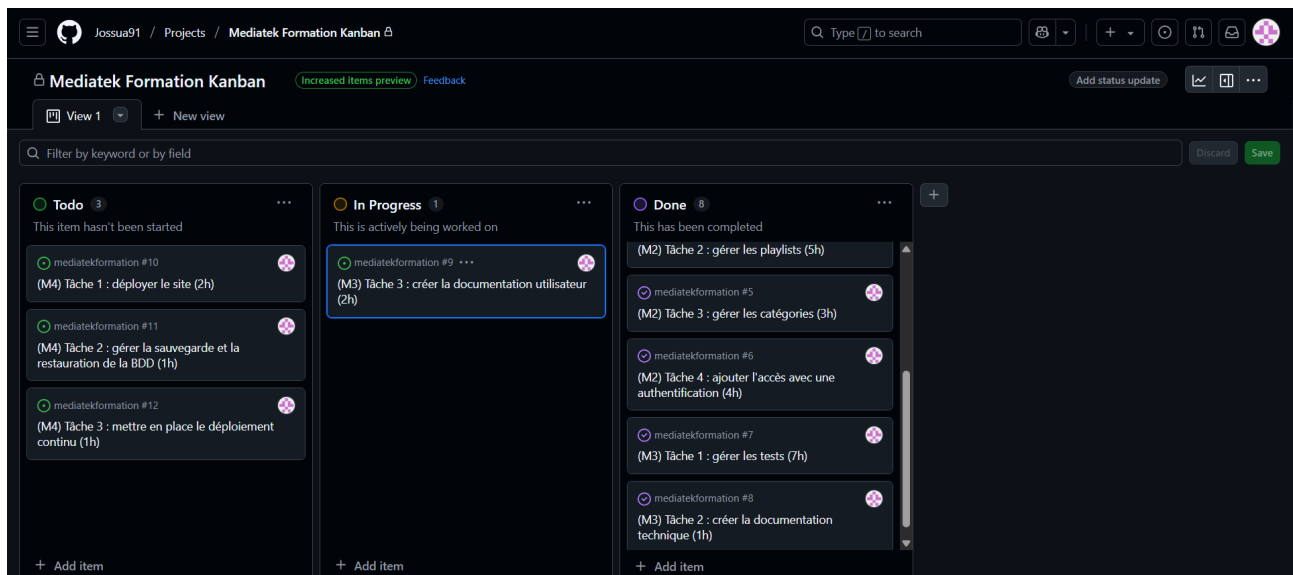
Temps estimé: 1 heure

Temps réel : 1 heure

Première page de la documentation technique :



Demande : Création d’une vidéo qui permet de montrer toutes les fonctionnalités du site.



Temps estimé : 2 heures

Temps réel : 1 heure

[Capture d'écran de la vidéo]

Diagramme de cas d'utilisation de l'application Symfony Mediatek Formation :

