Atelier professionnel n°1 : Mediatekformation

Compte rendu de l'activité

FRANCART Jérémy
BTS SIO DEUXIEME ANNEE

Sommaire :

Contexte de l'atelier

Cet atelier prend place au sein de l'entreprise InfoTech Services 86 dans laquelle nous travaillons en tant que développeur junior. Cette entreprise a remporté des appels d'offres pour effectuer de multiples interventions pour le réseau de MediaTek86. MediaTek86 est un réseau qui gère les médiathèques de la Vienne, et qui a entre autres pour rôle développer la médiathèque numérique pour l'ensemble des médiathèques du département. C'est dans ce dernier but que MediaTek86 a fait recours aux services de InfoTech Services 86. Parmi ces interventions, nous avons été confiés la création d'un site internet pour exposer les formations en ligne de MediaTek86 aux utilisateurs

Mission globale

Notre mission consiste à développer et déployer un site internet utilisant le framework PHP Symfony et le SGBDR MySQL qui répond aux attentes de MediaTek86 : lister les formations, leurs playlists et leurs catégories pour tous les utilisateurs et gérer ces formations, playlists et catégories depuis un accès demandant une authentification. Il faudra aussi construire une documentation technique et utilisateur.

Un premier développeur à déjà commencé le travail, il a réalisé la majeure partie liée à l'affichage pour les utilisateurs (front-office), nous allons donc continuer ce qu'il a commencé.

Nous gérerons de multiples aspects qui vont être divisés en plusieurs étapes :

Étape 1 : Préparation de l'environnement de travail, de l'IDE (environnement de développement intégré), du site internet et de sa base de données

Étape 2 : Nettoyage et optimisation du code source existant

Étape 3 : Codage de la partie d'administration du site (back-office)

Étape 4 : Gestion des tests et de la documentation du site internet

Étape 5 : Déploiement du site internet en ligne et gestion du déploiement continu

Étape 6 : Construction du compte rendu et intégration du projet au portfolio

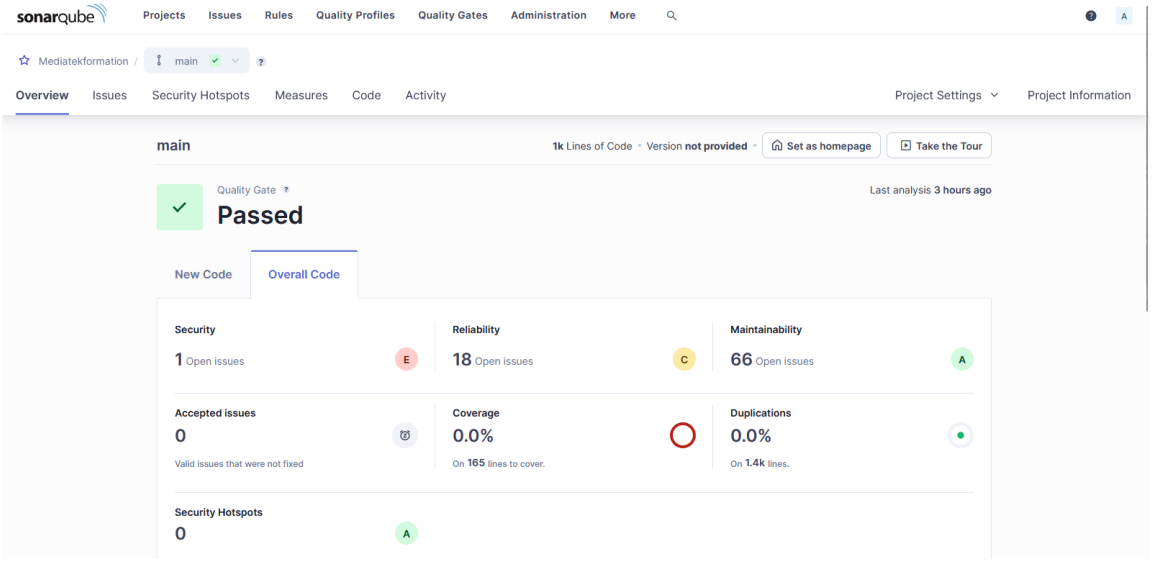
Réalisation de la mission

Étape 1 : Préparation de l'environnement de travail

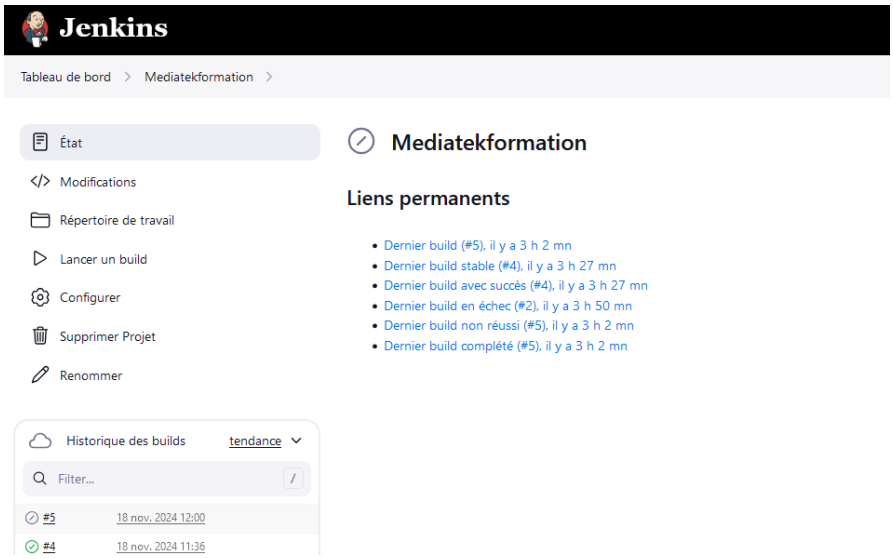
J'ai commencé par créer le dépôt GitHub de l'application d'où j'y ai ajouté les commits du premier développeur.

J'ai donc cloné le code source original que j'ai placé sous WampServer (une solution qui regroupe Windows, Apache, MySQL et PHP pour pouvoir tester le site en local).

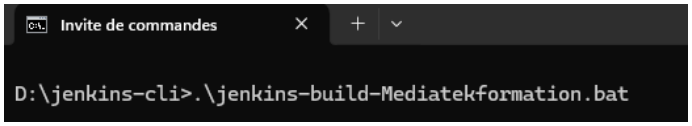
Afin de gérer la qualité de code, j'ai créé un projet pour le site internet sur SonarQube, une plateforme de qualité de code.



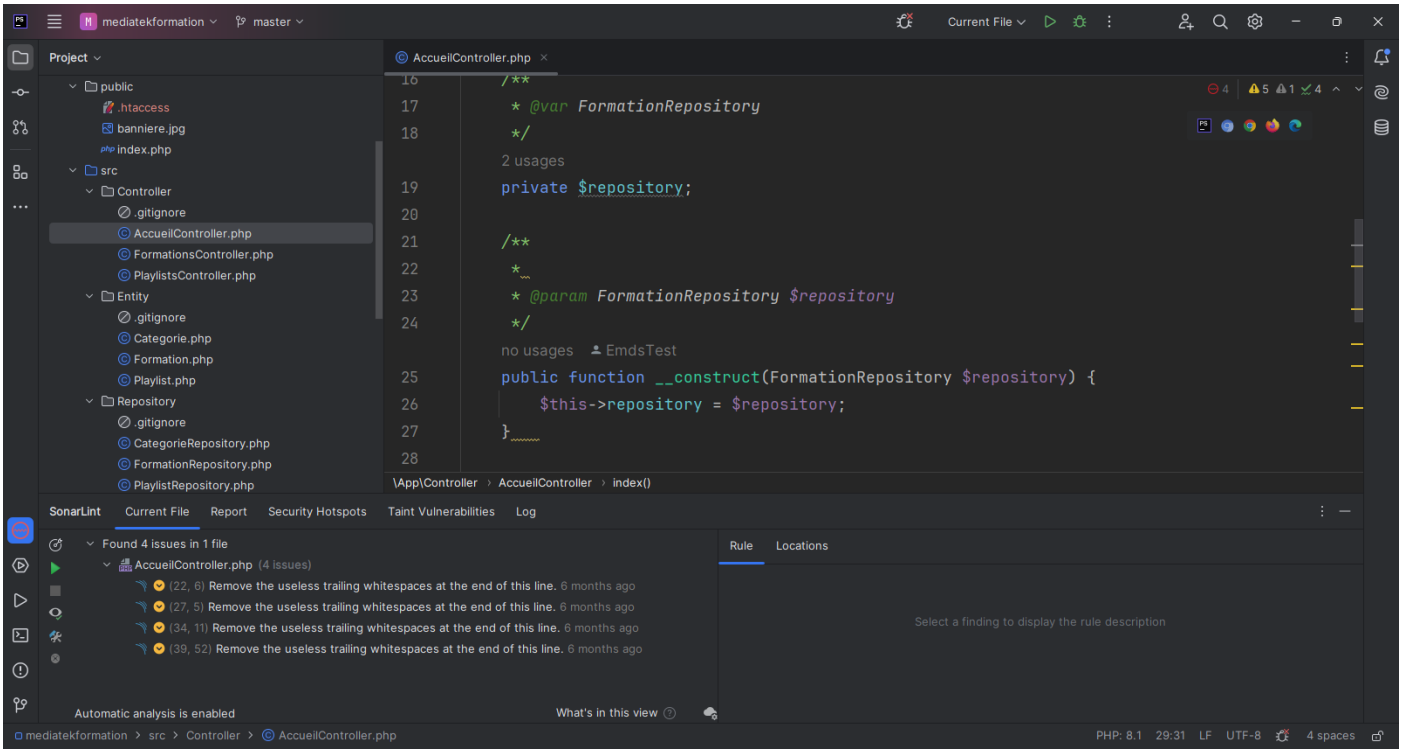
Puis, j'ai utilisé Jenkins afin de pouvoir semi-automatiser la mise à jour de SonarQube lorsque je réalisais un commit.



Il ne me restait plus qu'à exécuter ce script que j'ai créé pour lancer une analyse du code et la mettre à jour sous SonarQube.



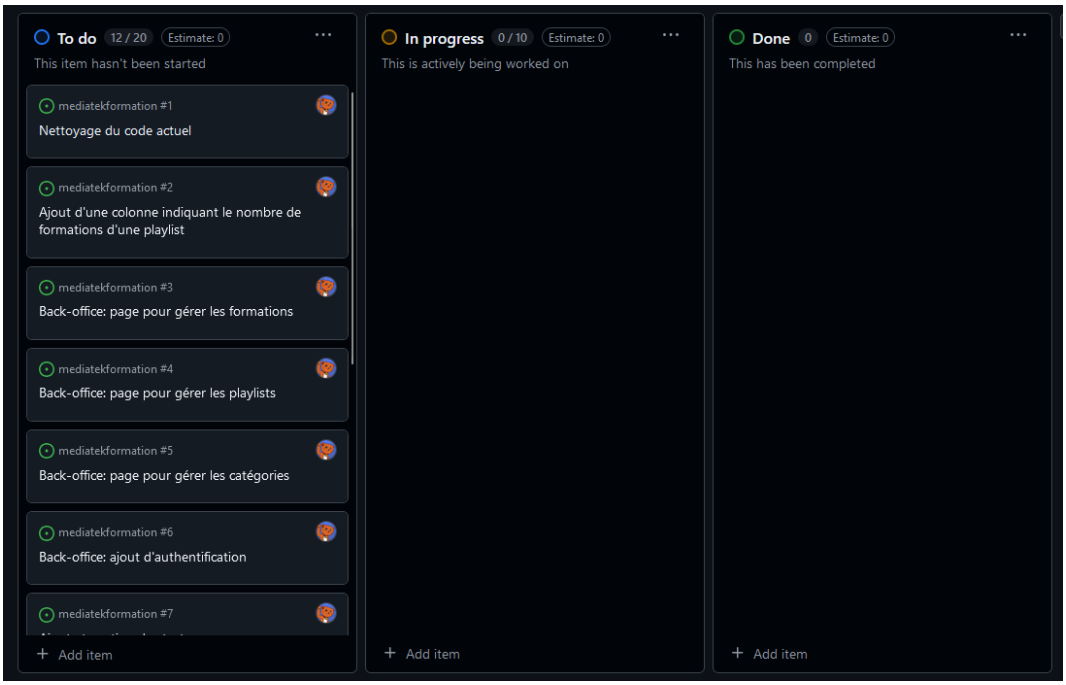
J'ai ensuite configuré PhpStorm, l'environnement de développement intégré choisi pour développer l'application :



Ensuite, j'ai ajouté et configuré la base de données existante du site sous phpMyAdmin.

Pour finaliser la configuration du site, j'ai exécuté `composer install` pour obtenir les bundles PHP nécessaires à l'application et j'y ai ajouté le bundle `symfony/apache-pack` pour avoir le fichier de redirection .htaccess et pour réparer la barre de débogage Symfony qui ne s'affichait pas.

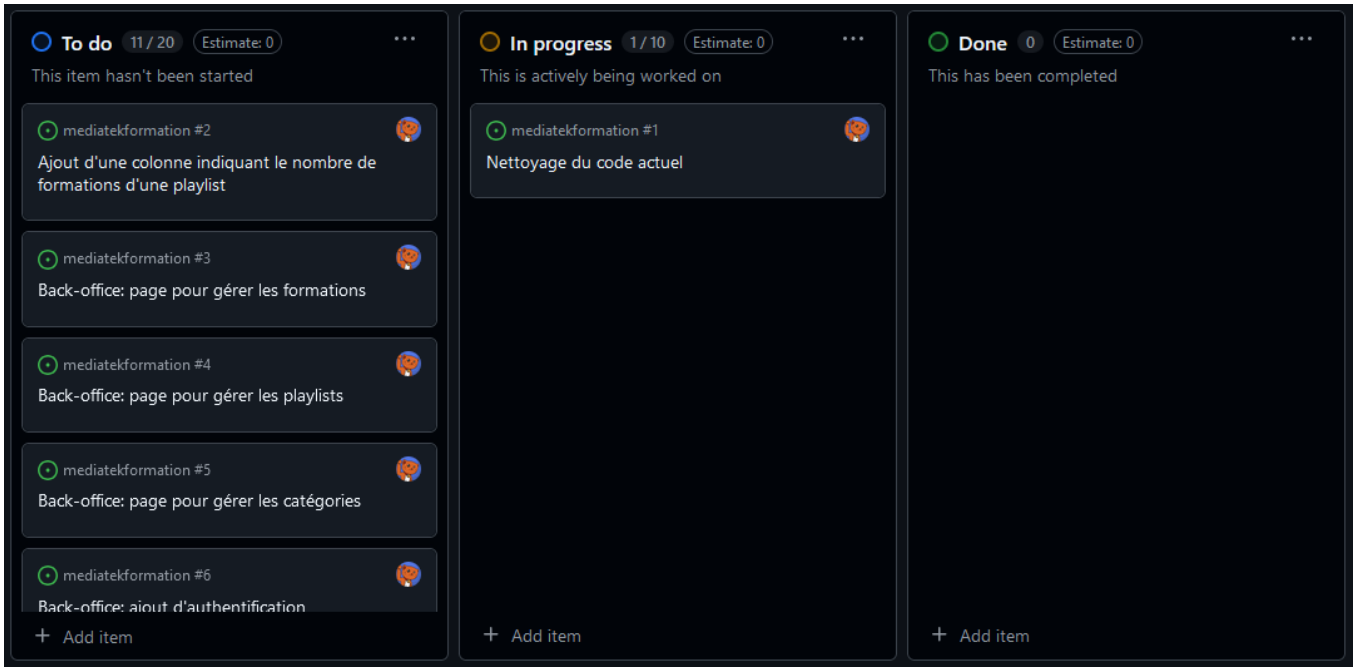
Enfin, j'ai mis en place un Kanban GitHub automatisé que j'ai rempli avec les tâches qui nous sommes confiés permettant la gestion du projet de manière organisée.



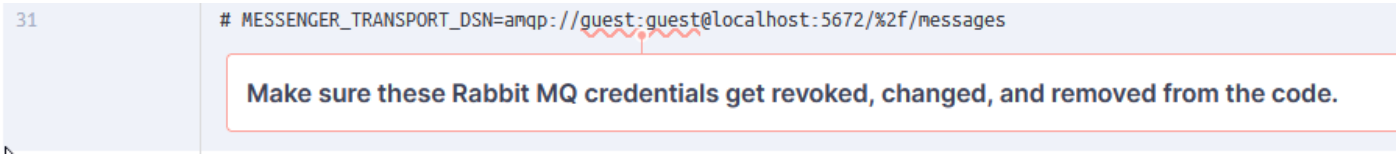
Et voici le site pour le moment :

Étape 2 : Nettoyer et optimiser le code existant

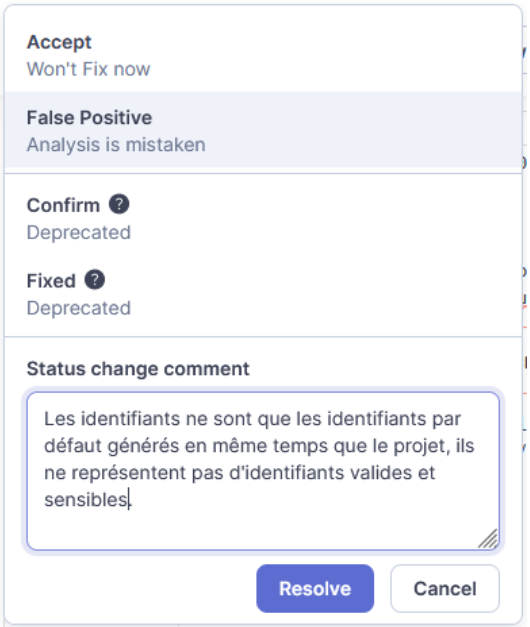
Tache 1 : Nettoyage du code actuel – Temps estimé 2 heures / temps réel 2 heures

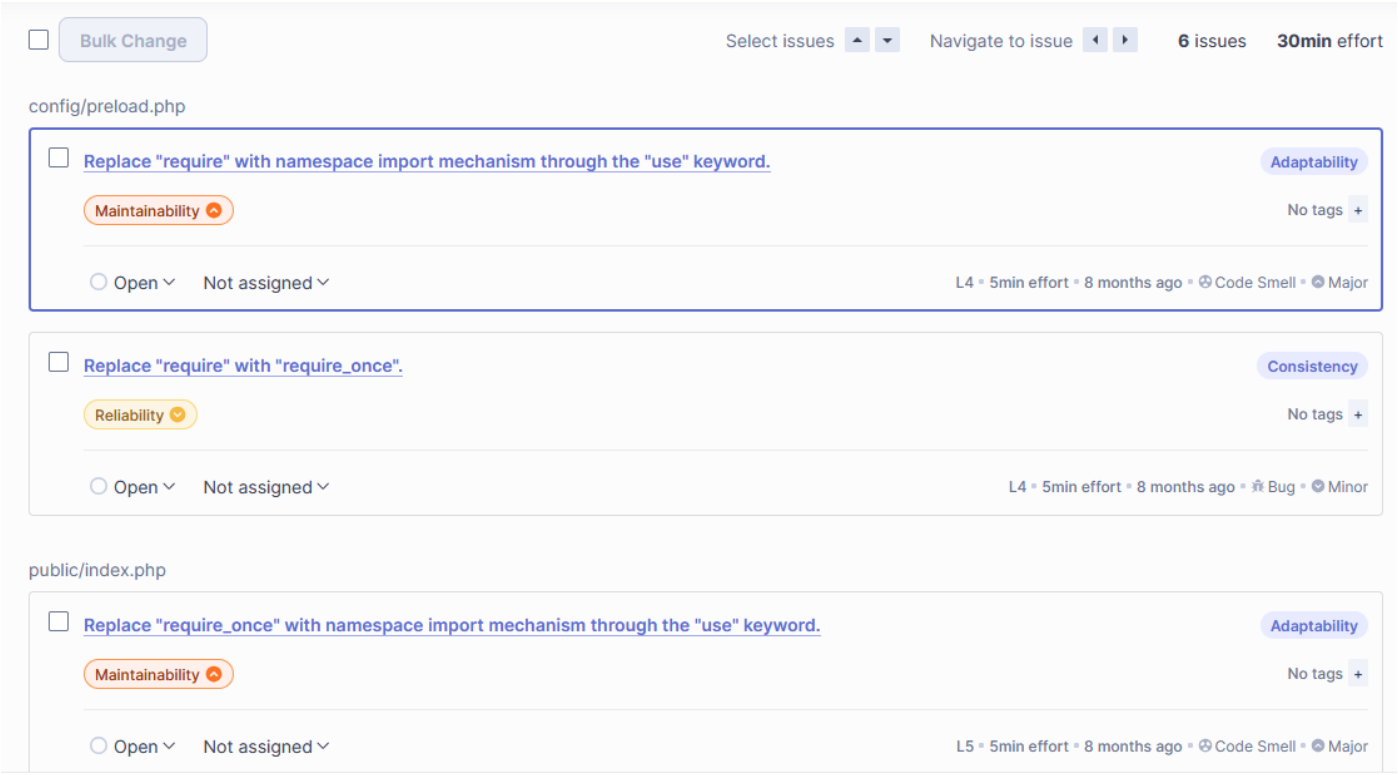


Nous allons utiliser SonarQube (et son intégration sous PhpStorm) afin de détecter et régler les problèmes de qualité de code. Nous allons voir les différents problèmes reportés par SonarQube et la (ou les) façon(s) de les régler.

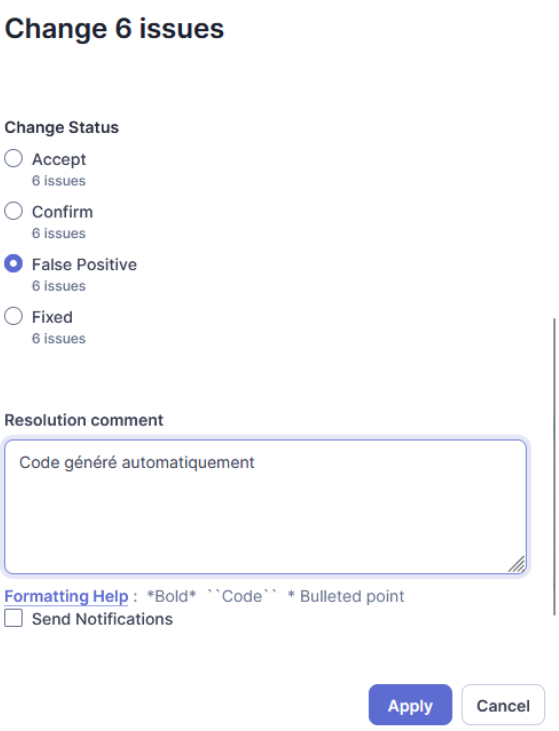


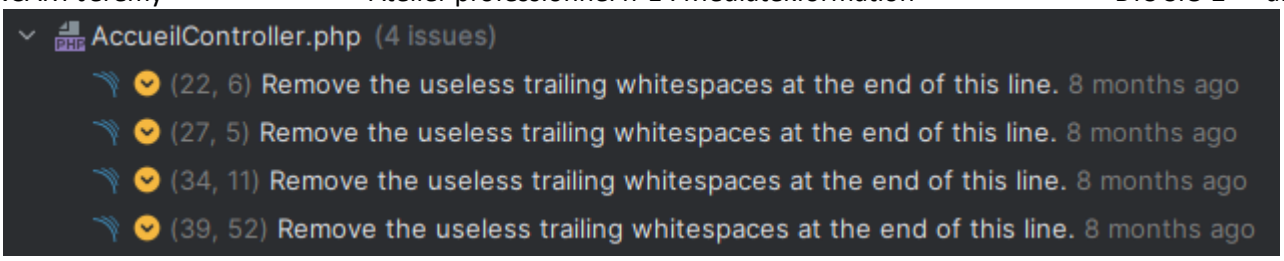
Ce premier problème apparrait dans le fichier .env du site où on y retrouve les secrets, identifiants et configurations du site. Ce problème est en fait un faux-positif pour les raisons évoquées dans le commentaire de résolution :





Ces problèmes concernent du code autogénéré par Symfony que l'on doit ignorer dans la recherche de problèmes.





Beaucoup de problèmes concernent juste des espaces en trop à la fin des lignes, on les retrouve dans plusieurs fichiers de l'application, on enlève juste ces espaces en trop pour régler ces problèmes.

Pour ce problème, il est indiqué que le nommage utilisé de cette constante ne respecte pas la convention de nommage pour les constantes, il faut renommer cette constante selon la convention de nommage « SCREAMING_SNAKE_CASE », on utilise les fonctionnalités de l'IDE afin de la renommer en plus de ces références.

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

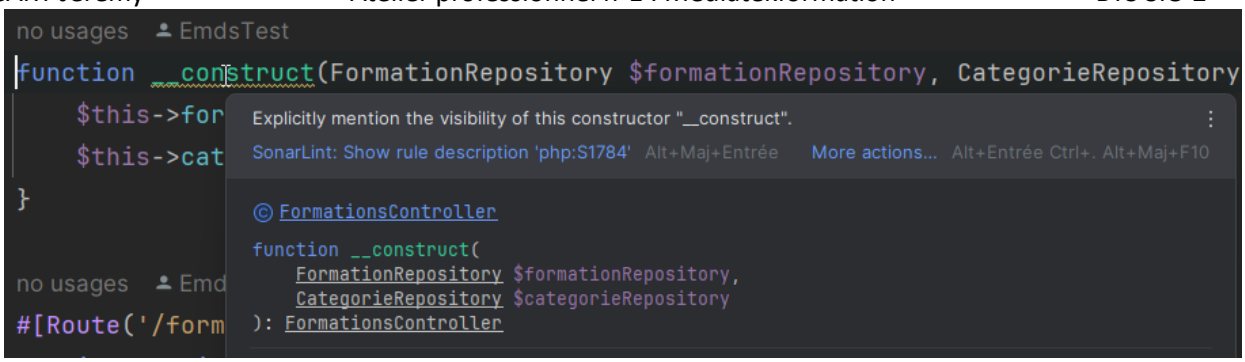
```
return $this->render( view: "pages/formations.html.twig", [
    'formations' => $formations,
    'categories' => $categories
]);
```

(22, 36) Define a constant instead of duplicating this literal "pages/formations.html.twig" 4 times. [+3 locations]

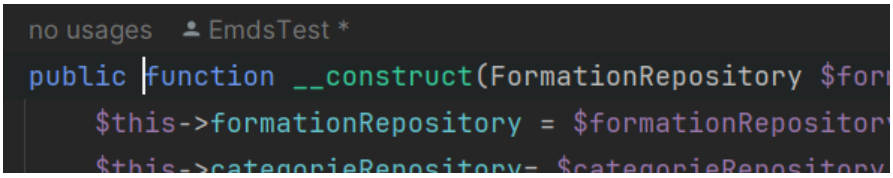
Ce problème indique que l'on répète une chaine en dur. Pour régler ce problème, on peut simplement créer une constante de classe que l'on va réutiliser plusieurs fois (ce type de problème apparait aussi dans d'autres fichiers, on fait la même chose pour le régler).

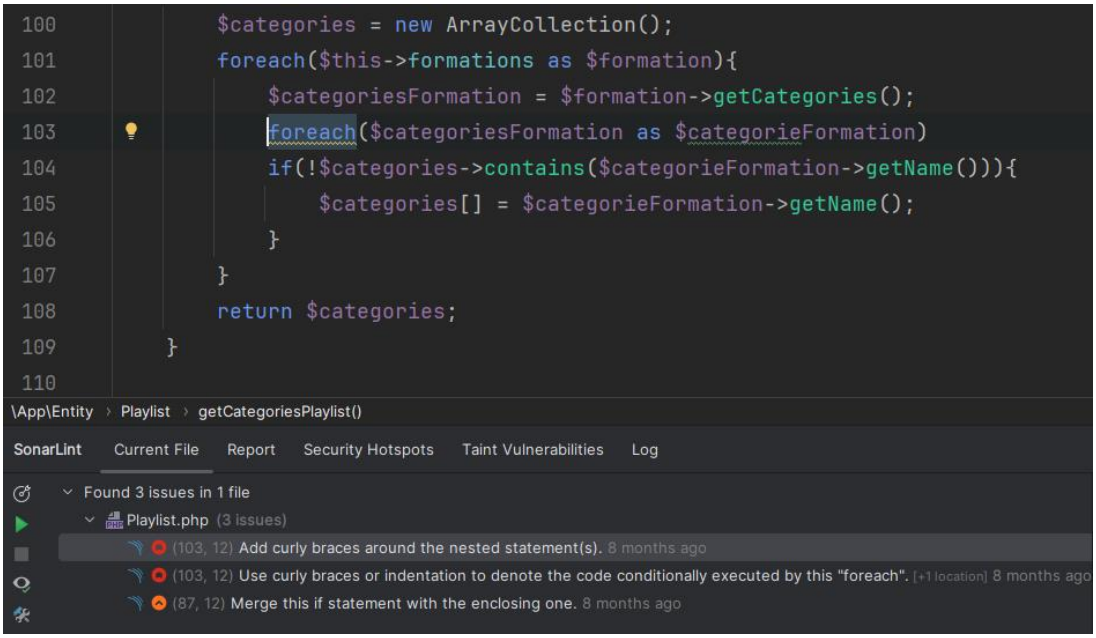
```
4 usages
private const PAGE_FORMATIONS = 'pages/formations.html.twig';
```

```
return $this->render( view: self::PAGE_FORMATIONS, [
    'formations' => $formations,
    'categories' => $categories
]);
```



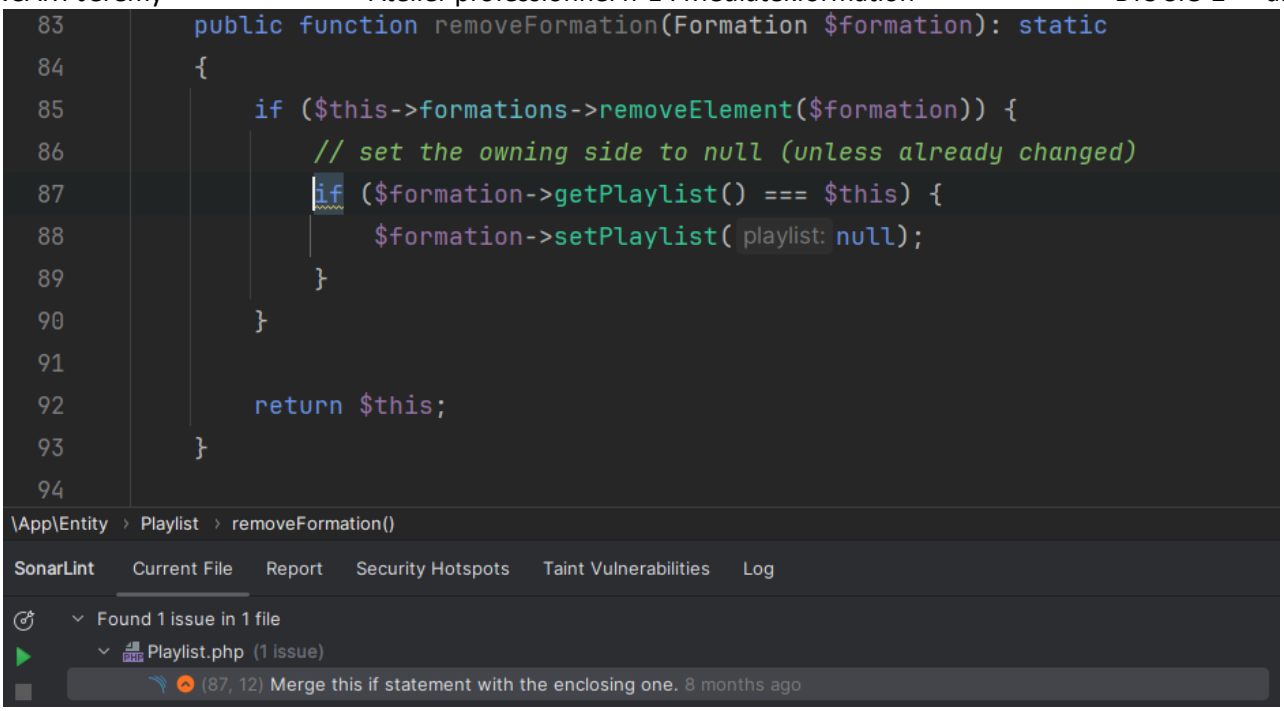
Ce constructeur doit pouvoir être appelé par Symfony donc il est de toute évidence public, on le marque comme tel pour régler le problème (le même problème apparait avec d'autres constructeurs).



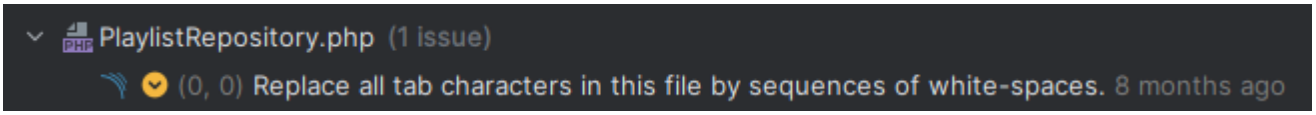


Il y a 2 problèmes ici : il manque des accolades pour dénoter la boucle « foreach » puis, le code au sein de la boucle « foreach » doit être indenté afin de rendre la lecture du code plus facile.

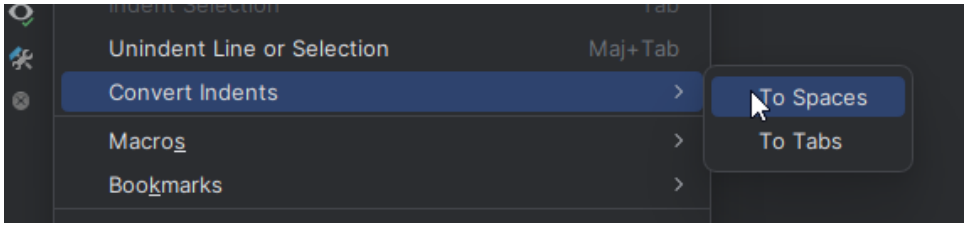
```
$categories = new ArrayCollection();
foreach($this->formations as $formation){
    $categoriesFormation = $formation->getCategories();
    foreach($categoriesFormation as $categorieFormation) {
        if(!$categories->contains($categorieFormation->getName())){
            $categories[] = $categorieFormation->getName();
        }
    }
}
return $categories;
```

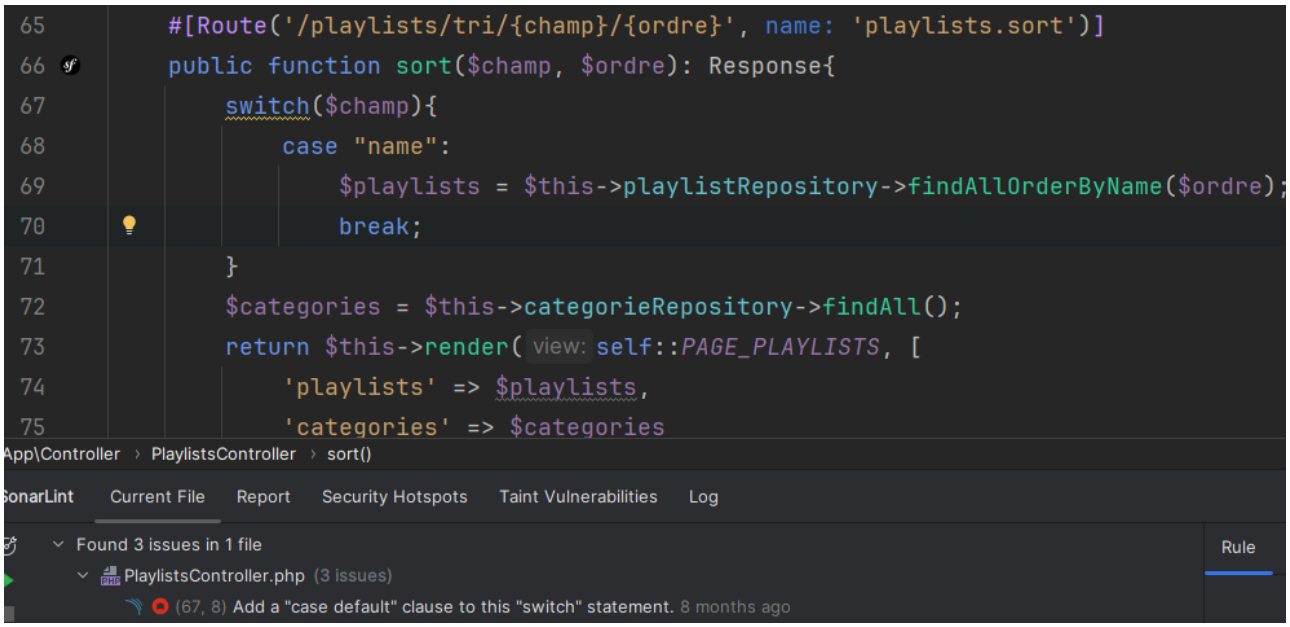


Cette méthode a été autogénérée donc on l'accepte comme elle l'est.



Ce fichier utilise des caractères de tabulation au lieu de remplacer les tabulations par des espaces. Pour régler ce problème, on peut utiliser l'IDE pour convertir les indentations en espaces.

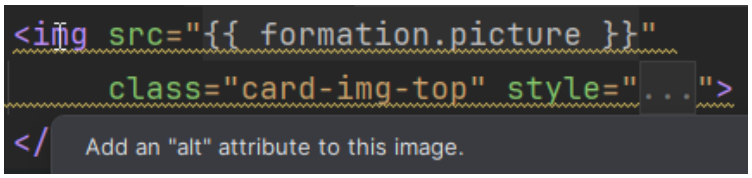




Ce problème nous indique qu'il faut ajouter un cas par défaut dans le switch dans le cas où champ ne correspond pas à "name".

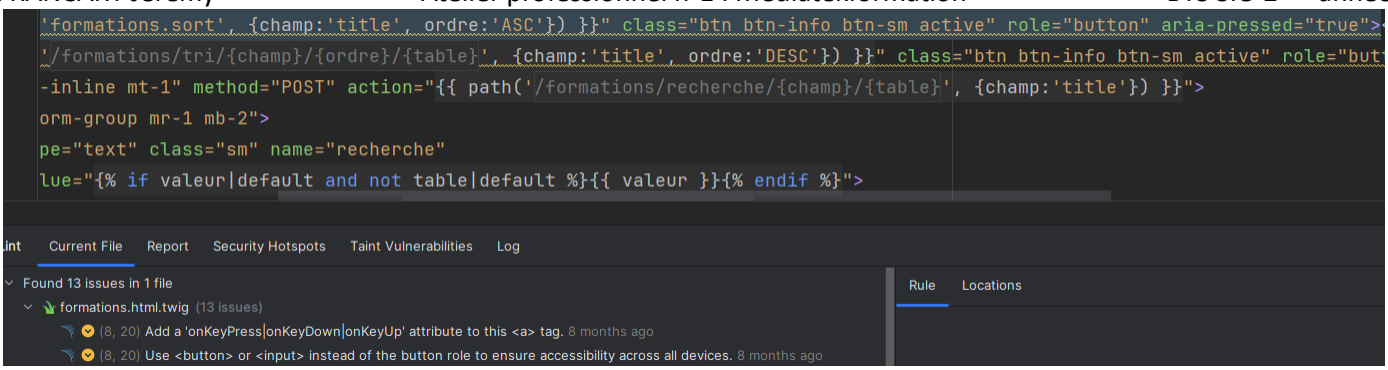
Comme un seul champ est filtré dans ce switch, on peut le changer par un bloc « if, else »

```
if ($champ === 'name') {
    $playlists = $this->playlistRepository->findAllOrderByName($ordre);
} else {
    $playlists = $this->playlistRepository->findAll();
}
```



Ce problème nous indique que pour des raisons d'accessibilité au site, il est important d'ajouter un attribut « alt » aux balises HTML « img ».

```
<img src="{{ formation.picture }}"
     alt="Miniature formation"
     class="card-img-top" style="...">
```

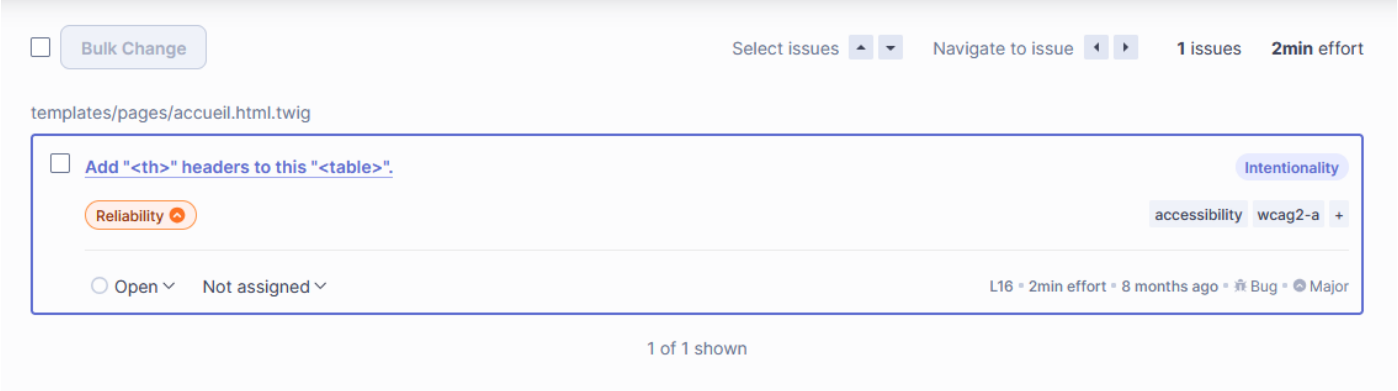


Ce problème provient du fait que l'on utilise les attributs « role » et « aria-pressed » sur des balises HTML « a ». Ces attributs sont censés être utilisés sur des boutons. On supprime donc ces attributs inutiles et on fait pareil pour tous les boutons.

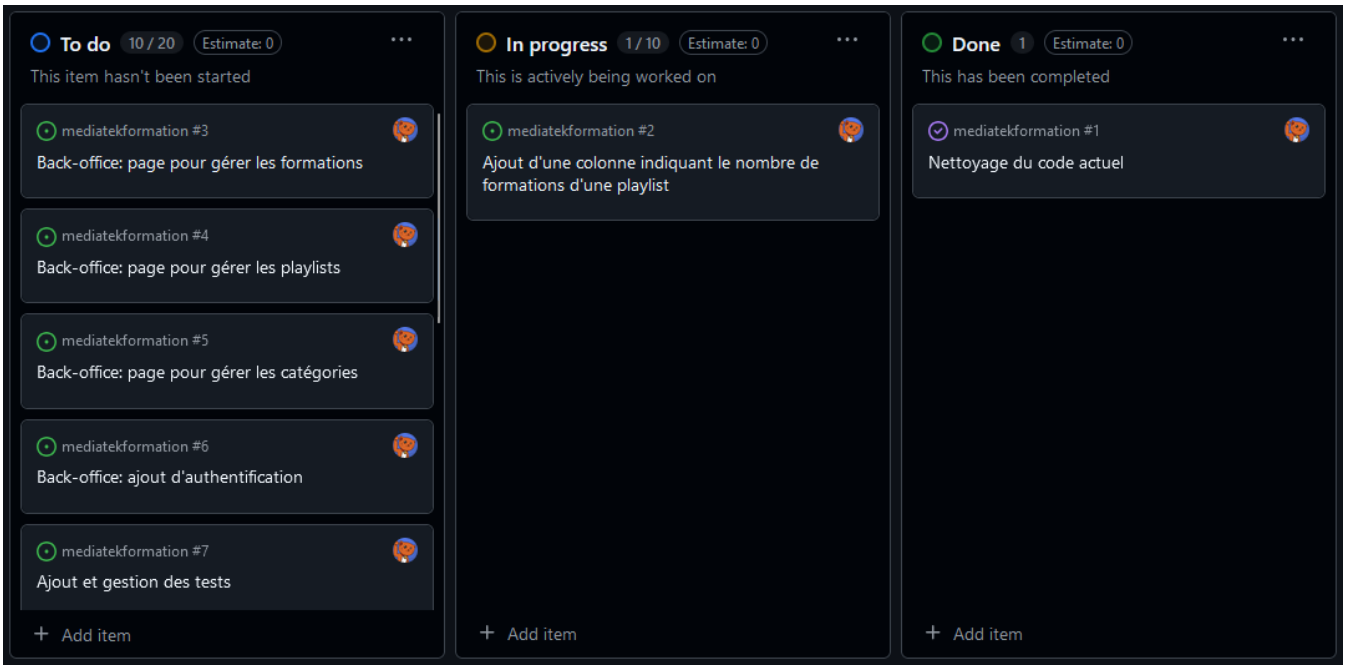
```
<a href="{{ path('formations.sort', {champ:'title', ordre:'ASC'}) }}" class="btn btn-info btn-sm active"><</a>
```

Maintenant que les différents problèmes ont été réglés, on peut effectuer un commit et utiliser le script Jenkins pour mettre à jour le projet sous SonarQube

Le seul problème restant concerne l'utilisation d'une table dans l'accueil pour disposer les 2 dernières formations dans l'accueil, notre responsable a indiqué que ce problème n'était pas à régler. Cependant, il pourrait facilement être réglé en remplaçant la table par un élément « div » disposant les éléments d'une manière particulière.



Tache 2 : Ajout d'une fonctionnalité – Temps estimé 2 heures / temps réel 2 heures



Dans cette tâche, il nous est demandé d'ajouter une colonne indiquant le nombre de formations par playlist dans la page listant les playlists. Cette colonne doit aussi permettre les tris comme les autres colonnes.

On commence par ajouter une fonction dans l'entité playlist pour obtenir le nombre de formations pour une playlist :

```php
/**
 * Retourne le nombre de formations contenues dans la playlist
 * @return int
 */
no usages  new *
public function getFormationsCount(): int {
    return count($this->formations);
}
```

Puis, on ajoute dans le contrôleur de tri le champ « count » pour qui va nous permettre de savoir que l'on doit trier sur le nombre de formations

```php
no usages  EmdsTest +1 *
#[Route('/playlists/tri/{champ}/{ordre}', name: 'playlists.sort')]
public function sort($champ, $ordre): Response{
    if ($champ === 'name') {
        $playlists = $this->playlistRepository->findAllOrderByName($ordre);
    } elseif ($champ === 'count') {
        $playlists = $this->playlistRepository->findAllOrderByCount($ordre);
    } else {
        $playlists = $this->playlistRepository->findAll();
    }
```

On crée la méthode findAllOrderByCount($ordre) utilisée dans le contrôleur afin de pouvoir récupérer les playlists triés par rapport à un ordre :

```php
/**
 * Retourne toutes les playlists triées sur le nombre de formations dans la playlist
 * @param string $ordre
 * @return Playlist[]
 */
1 usage  new *
public function findAllOrderByCount($ordre): array{
    return $this->createQueryBuilder( alias: 'p')
        ->leftjoin( join: 'p.formations',  alias: 'f')
        ->groupBy( ...groupBy: 'p.id')
        ->orderBy( sort: 'COUNT(f.id)', $ordre)
        ->getQuery()
        ->getResult();
}
```

Ensuite, du côté de l'affichage, on ajoute des boutons pour trier par nombre de formations et on affiche le nombre de formations dans une playlist pour chaque playlist en utilisant la méthode getFormationsCount de l'entité.

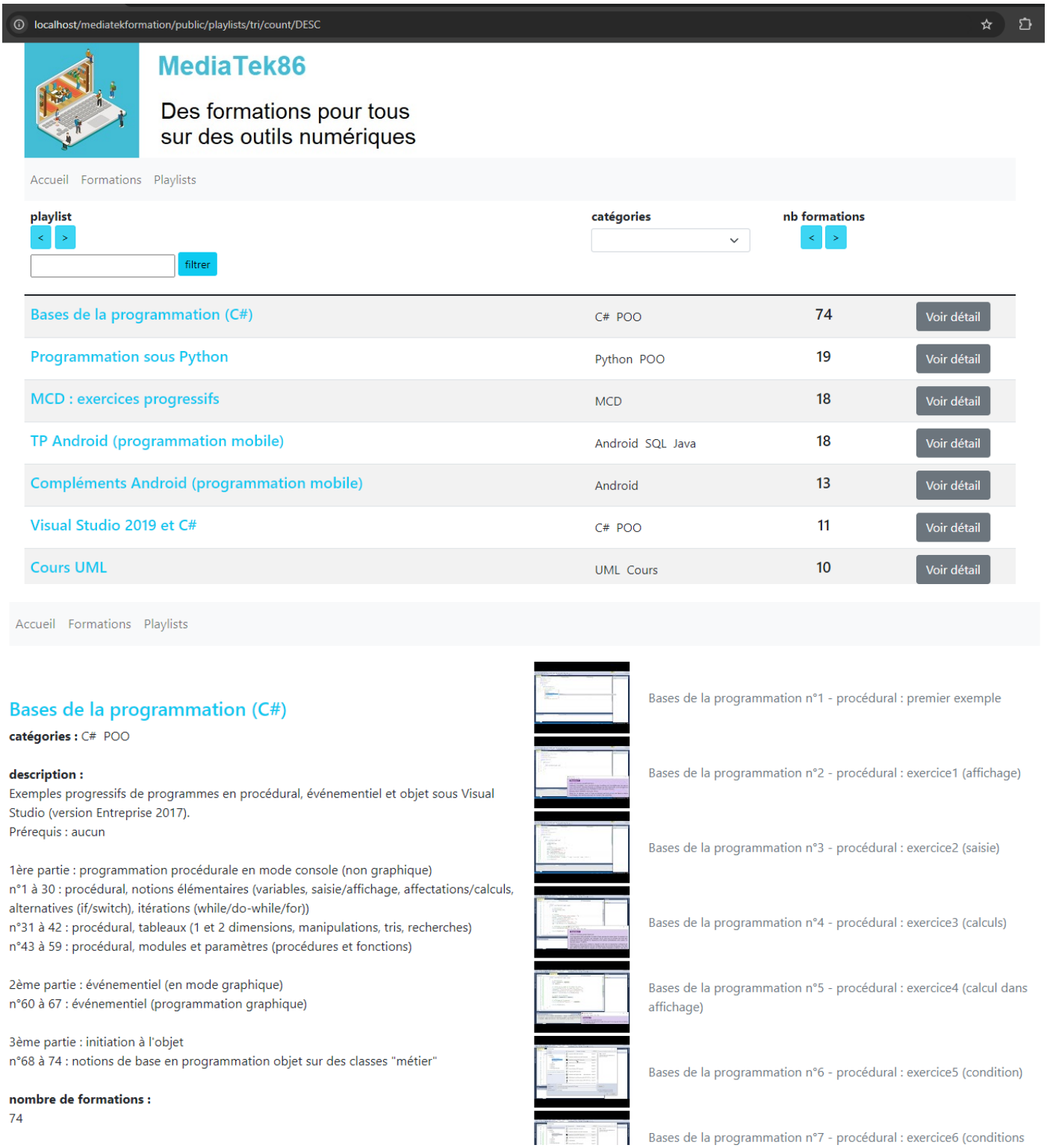
```twig
<th class="text-center align-top" scope="col">
    nb formations<br/>
    <a href="{{ path('playlists.sort', {champ:'count', ordre:'ASC'}) }}" class="btn btn-info btn-sm active"><</a>
    <a href="{{ path('playlists.sort', {champ:'count', ordre:'DESC'}) }}" class="btn btn-info btn-sm active">></a>
</th>
```

```twig
<td class="text-center">
    <h5>
        {{ playlists[k].formationsCount }}
    </h5>
</td>
<td class="text-center">
    <a href="{{ path('/playlists/playlist/{id
```

Enfin dans la page de détails d'une playlist, on affiche le nombre de formations de cette playlist.

```twig
<br /><br />
<strong>nombre de formations :</strong><br />
    {{ playlist.formationsCount }}
```

Voici le résultat :



Étape 3 : Coder la partie back-office

Dans cette étape, on va coder la partie d'administration du site (back-office). Pour ce faire, on va créer des routes sous /admin/ qui redirigeront vers l'administration du site

Tache 1 : Gestion des formations – Temps estimé 5 heures / temps réel 5 heures

Tache 2 : Gestion des playlists – Temps estimé 5 heures / temps réel 5 heures

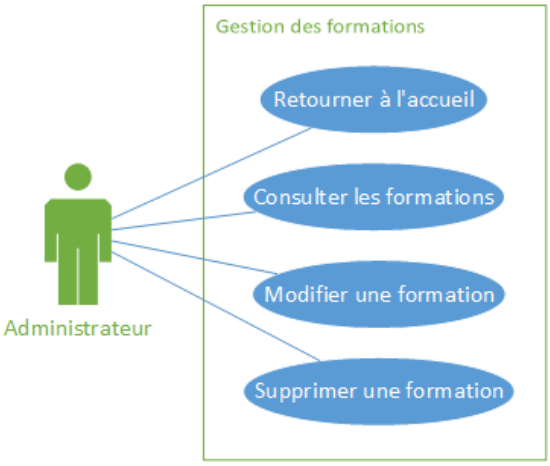
Tache 3 : Gérer les catégories – Temps estimé 3 heures / temps réel 1 heure 30 minutes

Ces 3 tâches sont regroupées car elles sont similaires. Il faut en fait que ces pages listent les formations / playlists / catégories de la même manière que sur la partie front-office avec les tris et filtres présents. La différence majeure est qu'une colonne actions doit être ajoutée pour pouvoir créer, modifier et supprimer les formations / playlists (pour les catégories on supportera seulement la création et suppression). Le clic sur les boutons ajouter et modifier doivent amener vers un formulaire d'addition / modification permettant d'ajouter / modifier l'entité. Le clic sur le bouton de suppression devra entrainer la suppression de l'entité après confirmation.

J'ai commencé par créer des maquettes pour les différentes pages ainsi que les diagrammes de cas d'utilisation.

Voici l'une des maquettes pour la gestion des formations :

Voici un des diagrammes de cas d'utilisation pour la gestion des formations :



Le reste des maquettes et diagrammes est disponible en téléchargement sur la page dédiée au projet sur mon portfolio

Une fois les maquettes et diagrammes créés, j'ai créé les contrôleurs et les templates Twig pour les différents onglets d'administration du site. J'ai pu réutiliser du code déjà présent dans la partie front-office pour pouvoir afficher, trier et filtrer les listes, j'y ai rajouté une dernière colonnes actions contenant les boutons créer, modifier, supprimer. Un clic sur un bouton ajouter / modifier emmène vers un formulaire Symfony créé grâce à la commande `make:form` des outils en ligne de commande de Symfony. J'ai appliqué la template de formulaires Bootstrap 5 afin d'améliorer le visuel des formulaires.

Un exemple de la commande pour créer un formulaire :

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

The name of the form class (e.g. AgreeableChefType):
> ExempleType

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
> User

created: src/Form/ExempleType.php

Success!

Next: Add fields to your form and start using it.
Find the documentation at https://symfony.com/doc/current/forms.html
```

La route d'édition du contrôleur d'administration des formations où l'on affiche le formulaire ou on gère la modification :

```php
     * @param int $id L'identifiant de la formation à modifier
     * @param Request $request La requête actuelle (injecté par Symfony)
     * @return Response
     */
    no usages  Refragg
    #[Route('/admin/formations/edit?{id}', name: 'admin.formations.edit')]
    public function edit(int $id, Request $request): Response {
        $formation = $this->formationRepository->find($id);
        $formFormations = $this->createForm( type: FormationsType::class, $formation);

        $formFormations->handleRequest($request);
        if ($formFormations->isSubmitted() && $formFormations->isValid()) {
            $this->formationRepository->addOrEdit($formation);
            return $this->redirectToRoute( route: 'admin.formations');
        }

        return $this->render( view: 'pages/admin/admin.formations.edit.html.twig', [
            'formation' => $formation,
            'formformations' => $formFormations->createView()
        ]);
    }
```

L'utilisation de la template Bootstrap 5 dans les formulaires (fichier de configuration de twig):

```
twig:
    file_name_pattern: '*.twig'
+   form_themes: ['bootstrap_5_layout.html.twig']
```

L'objet « formformations » envoyé depuis le contrôleur peut ensuite être utilisé dans la template twig afin de personnaliser comment le formulaire va s'afficher

```twig
{{ form_start(formformations) }}
<div>
    <div class="row">
        <div class="col">
            {{ form_row(formformations.title) }}
        </div>
        <div class="col">
            {{ form_row(formformations.description) }}
        </div>
        <div class="col">
            {{ form_row(formformations.publishedAt) }}
        </div>
    </div>
    <div class="row" style="..."...>
</div>
<div>
    <div class="align-content-center justify-content-center" style="...">
        {{ form_row(formformations.submit) }}
        <div class="mb-3" style="..."...>
    </div>
</div>
{{ form_end(formformations) }}
```

Enfin, on inclus cette template de formulaire dans la template d'édition / addition :

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

{% block body %}
    <h2>Détails formation :</h2>
    {% include 'pages/admin/admin.formations.form.html.twig' %}
{% endblock %}

{% block title %}
    Édition d'une formation
{% endblock %}
```

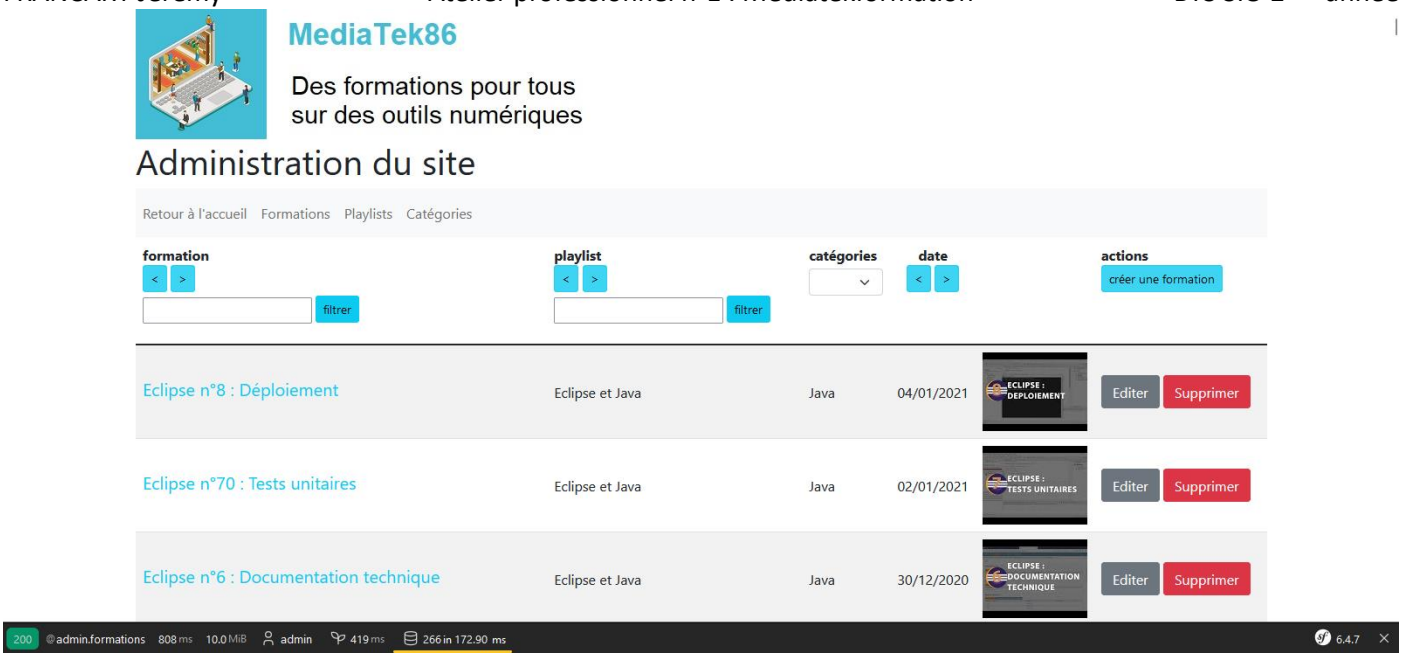
Pour la validation du formulaire, il est possible d'ajouter des contraintes directement dans l'entité et les erreurs vont être gérées automatiquement par Symfony :

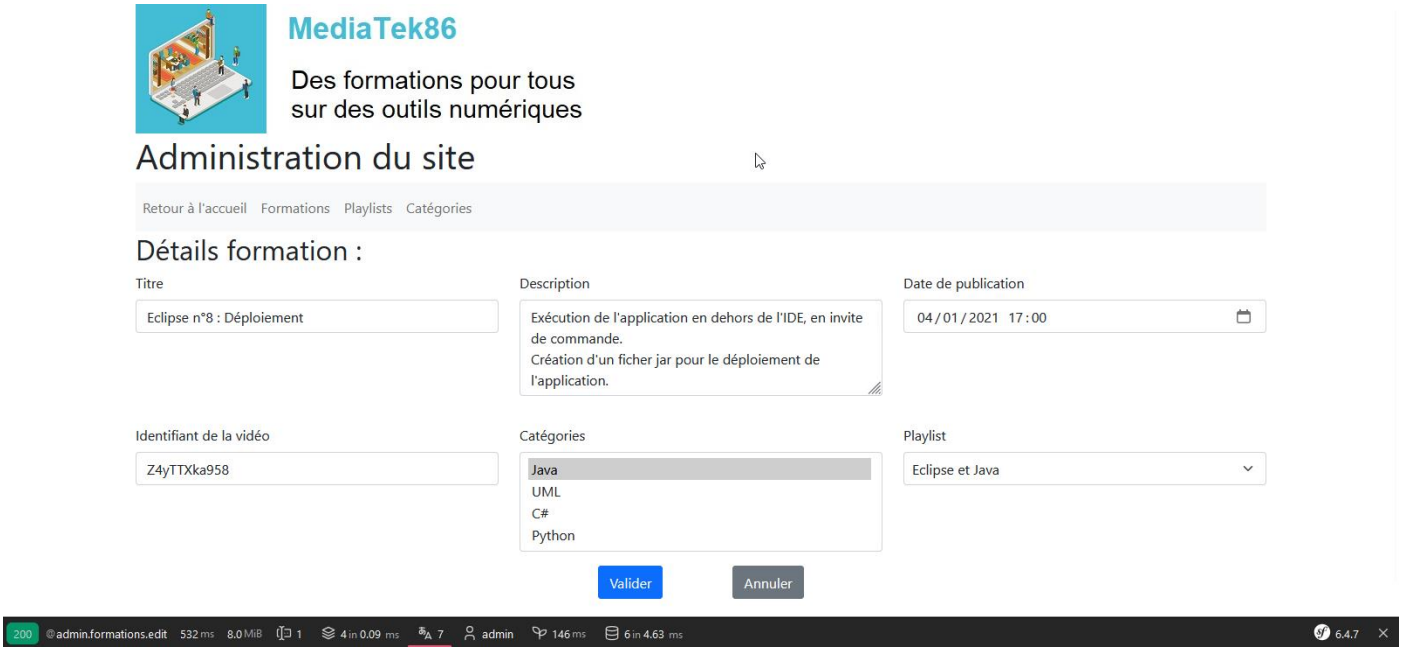
```php
4 usages
#[ORM\Column(type: Types::DATETIME_MUTABLE, nullable: true)]
#[Assert\LessThanOrEqual(['value' => 'now'])]
private ?\DateTimeInterface $publishedAt = null;
```

(cette contrainte valide que la date et heure n'est pas postérieure à la date et heure actuelle)

Aussi, afin de pouvoir ajouter, supprimer et modifier les entités, certaines méthodes ont dû être rajoutées dans le repository des entités.

Au final, voici les fichiers modifiés lors de l'ajout de l'administration du site et voici les résultats :

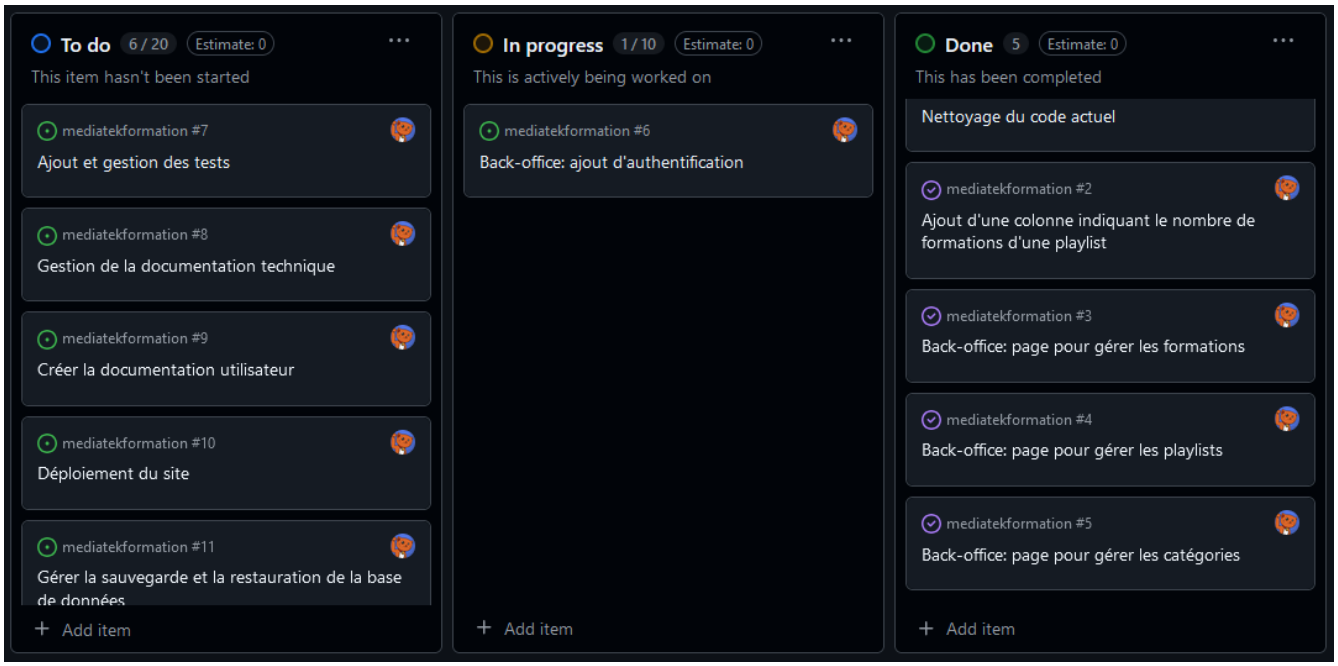

MediaTek86
Des formations pour tous
sur des outils numériques
Administration du site
Retour à l'accueil Formations Playlists Catégories
formation
playlist
catégories
date
actions
filtrer
filtrer
créer une formation
Eclipse n°8 : Déploiement
Eclipse et Java
Java
04/01/2021
Editer
Supprimer
Eclipse n°70 : Tests unitaires
Eclipse et Java
Java
02/01/2021
Editer
Supprimer
Eclipse n°6 : Documentation technique
Eclipse et Java
Java
30/12/2020
Editer
Supprimer
200 @admin.formations 808 ms 10.0 MiB admin 419 ms 266 in 172.90 ms
6.4.7


MediaTek86
Des formations pour tous
sur des outils numériques
Administration du site
Retour à l'accueil Formations Playlists Catégories
Détails formation :
Titre
Eclipse n°8 : Déploiement
Description
Exécution de l'application en dehors de l'IDE, en invite de commande.
Création d'un ficher jar pour le déploiement de l'application.
Date de publication
04/01/2021 17:00
Identifiant de la vidéo
Z4yTTXka958
Catégories
Java
UML
C#
Python
Playlist
Eclipse et Java
Valider
Annuler
200 @admin.formations.edit 532 ms 8.0 MiB 1 4 in 0.09 ms 7 admin 146 ms 6 in 4.63 ms
6.4.7


sur des outils numériques
istration du site
ueil Formations Playlists Catégories
playlist
catégories
date
filtrer
localhost
Êtes-vous sûr de vouloir supprimer Eclipse n°8 : Déploiement ?
OK
Annuler
: Déploiement
Eclipse et Java
Java
04/01/2021
0 : Tests unitaires
Eclipse et Java
Java
02/01/2021
: Documentation technique
Eclipse et Java
Java
30/12/2020

Tache 4 : Ajouter l'accès avec authentification – Temps estimé 4 heures / temps réel 1 heure

A travers cette tâche, il nous est demandé de gérer l'authentification pour la partie back-office afin que tout le monde ne puisse pas accéder au panel administratif.



Pour commencer, la maquette et le diagramme de cas d'utilisation ont été créés (disponible en téléchargement sur la page dédiée au projet dans le portfolio)

Maintenant nous allons créer une entité « User » avec la commande Symfony `make:user` :

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

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

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

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

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by

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

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

 Success!

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

Une fois l'entité l'utilisateur créé, on utilise la commande `make:migration` pour générer un fichier de migration de base de données que l'on va ensuite exécuter avec `doctrine:migrations:migrate`.

Ensuite, on va initialiser un utilisateur admin dans la base de données. Pour ce faire, on va utiliser le bundle orm-fixtures, ce bundle va nous permettre d'ajouter un utilisateur utilisant les interfaces de programmation de Symfony.

On peut maintenant définir la méthode load dans un fichier Fixtures spécifique pour initialiser un utilisateur :

```php
no usages  Refragg
public function load(ObjectManager $manager): void
{
    $user = new User();
    $user->setUsername( username: 'admin');
    $password = 'admin';
    $hashedPassword = $this->passwordHasher->hashPassword($user, $password);
    $user->setPassword($hashedPassword);
    $user->setRoles(['ROLE_ADMIN']);
    $manager->persist($user);
    $manager->flush();
}
```

On exécute cette Fixture avec `doctrine:fixtures:load`. Notre base de données possède maintenant un compte administrateur.

Maintenant, on va paramétrer l'authentification dans le fichier de configuration security.yaml pour rediriger vers la route d'un contrôleur.

```yaml
main:
    lazy: true
    provider: app_user_provider
    form_login:
        login_path: app_login
        check_path: app_login
        enable_csrf: true
    logout:
        path: logout
```

On crée ensuite le contrôleur Login pour gérer l'authentification.

```php
no usages
class LoginController extends AbstractController
{
    #[Route('/login', name: 'app_login')]
    public function index(AuthenticationUtils $authenticationUtils): Response
    {
        // Récupération éventuelle de l'erreur
        $error = $authenticationUtils->getLastAuthenticationError();
        // Récupération éventuelle du dernier nom d'utilisateur utilisé
        $lastUsername = $authenticationUtils->getLastUsername();

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

    no usages
    #[Route('/logout', name: 'logout')]
    public function logout() {
        // Géré par le bundle security de Symfony
    }
}
```

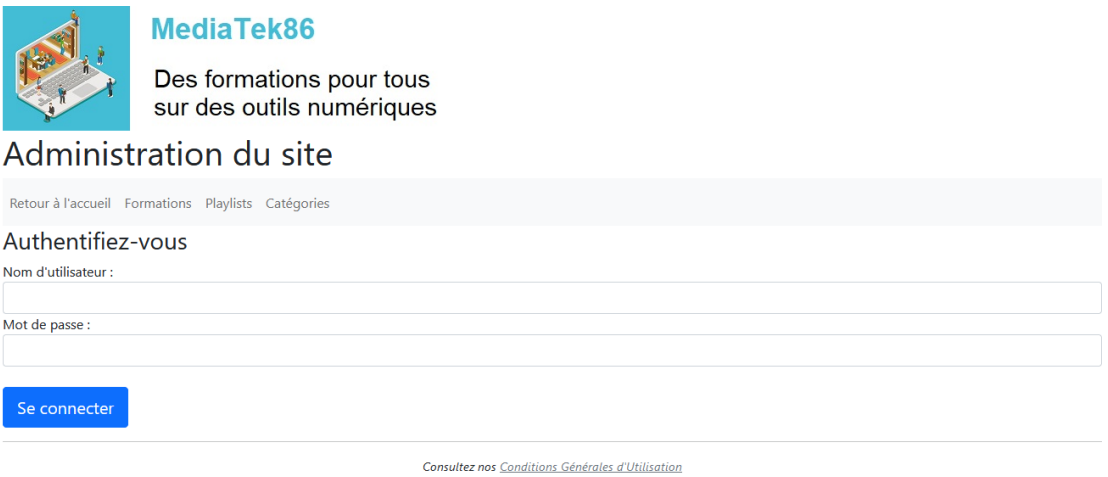
Il faut maintenant créer le formulaire de login puis, on définit dans security.yaml qu'il faut le rôle administrateur pour accéder aux routes sous /admin/. Cela va déclencher le processus d'authentification si aucun utilisateur n'est connecté

```
34      access_control:
35          - { path: ^/admin, roles: ROLE_ADMIN }
```

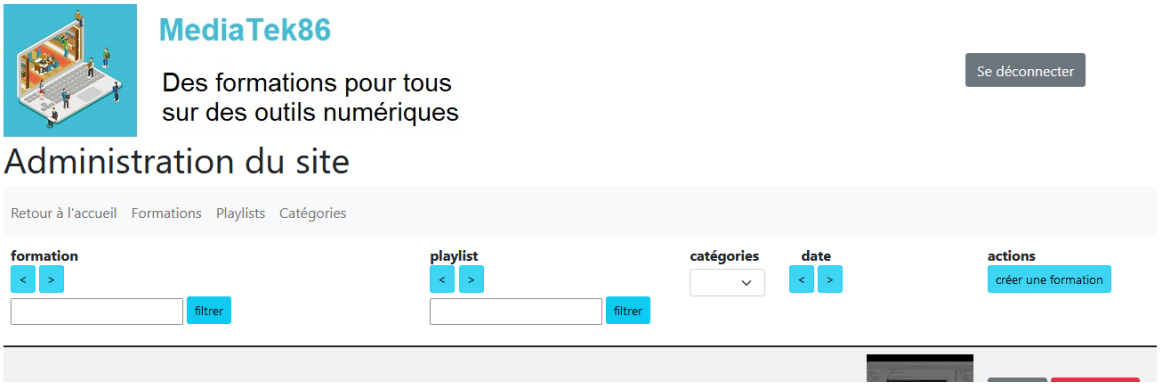
Enfin, on ajoute un bouton se déconnecter dans les templates baseadmin et basefront.

```
<div class="text-left" style="...">
    <img src="{{ app.request.getBasePath()~'/banniere.jpg' }}" alt="Bannière" >
    {% if app.user %}
        <a href="{{ path('logout') }}" class="btn btn-secondary" style="...">Se déconnecter</a>
    {% endif %}
</div>
```

Voilà le résultat :



Puis, le bouton « Se déconnecter » apparait quand on est connecté :



Étape 4 : Tester et documenter

Tâche 1 : Gestion des tests – Temps estimé 7 heures / temps réel 6 heures

Pour cette tâche, il nous est demandés de créer plusieurs tests au travers du site internet : des tests unitaires, d'intégrations, fonctionnels et de compatibilité.

Les tests doivent respecter le cahier des charges et le PV de recette (disponible depuis le portfolio).

Pour commencer, on réalise des tests unitaires sur la méthode formation.getPublishedAtString()

```php
class FormationTest extends TestCase
{
    public function testGetPublishedAtStringDateValide() {
        $date = new \DateTime( datetime: '2025-01-04 17:00:12');

        $formation = new Formation();
        $formation->setPublishedAt($date);

        $this->assertEquals( expected: '04/01/2025', $formation->getPublishedAtString());
    }

    public function testGetPublishedAtStringDateVide() {
        $formation = new Formation();

        $this->assertEquals( expected: '', $formation->getPublishedAtString());
    }
}
```

Ensuite, on réalise des tests d'intégration sur les règles de validation d'une formation

```php
class FormationValidationsTest extends KernelTestCase
{
    3 usages
    private function getFormation() {
        $formation = new Formation();
        $formation->setTitle( title: "Formation de test");
        $formation->setDescription( description: "Description de test");
        $formation->setVideoId( videoId: "rUnuYTjaBoU");
        return $formation;
    }
    public function testPublishedAtPosterieurAMaintenant() {
        $datePosterieure = (new \DateTime( datetime: 'now'))->add(new \DateInterval( duration: 'PT2H'));
        $formation = $this->getFormation()->setPublishedAt($datePosterieure);

        self::bootKernel();
        $validateur = self::getContainer()->get(ValidatorInterface::class);
        $erreurs = $validateur->validate($formation);
        $this->assertCount( expectedCount: 1, $erreurs);
    }
    public function testPublishedAtAnterieurAMaintenant() {...}
    public function testPublishedAtEgalAMaintenant() {...}
}
```

Ensuite, il faut des tests d'intégration sur les méthodes ajoutées dans les repositories, pour cela, j'ai créé une base de données de test afin de ne pas modifier la base de données de production.

J'ai utilisé GitHub pour voir l'historique et les méthodes que j'ai ajoutées dans les repositories puis, j'ai créé des classes de tests pour les différents repositories

```php
class PlaylistRepositoryTest extends KernelTestCase
{
    2 usages
    private function getRepository() {
        self::bootKernel();
        return self::getContainer()->get(PlaylistRepository::class);
    }

    public function testFindAllOrderByCountAscendant() {
        $repository = $this->getRepository();
        $playlists = $repository->findAllOrderByCount('ASC');

        $this->assertEquals( expected: 'playlist test', $playlists[0]->getName());
        $this->assertEquals( expected: 'Cours Informatique embarquée', $playlists[1]->getName());
        $this->assertEquals( expected: 'Cours Merise/2', $playlists[2]->getName());
    }

    public function testFindAllOrderByCountDescendant() {...}
}
```

Ensuite, des tests fonctionnels ont été construits, ici, on va tester les contrôleurs. Pour ce faire, on effectue des requêtes HTTP avec un client et on vérifie le résultat

Par exemple pour l'accueil :

```php
class AccueilControllerTest extends WebTestCase
{
    public function testIndex() {
        $client = self::createClient();
        $client->request( method: 'GET', uri: '/');
        $this->assertResponseIsSuccessful();
    }
}
```

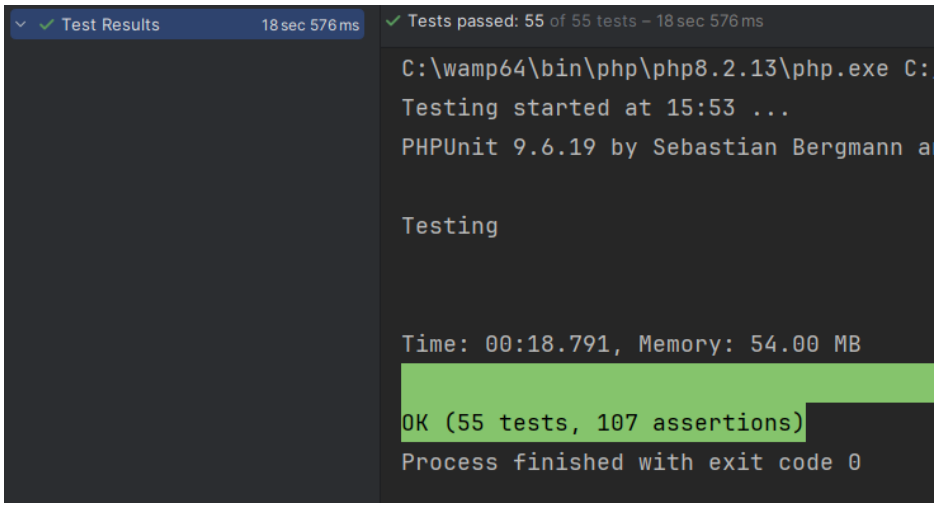
On teste aussi sur les pages avec les listes les tris, les recherches puis on teste un des liens pour voir si le contenu est celui attendu :

```php
public function testLienPlaylistFonctionne() {
    $client = self::createClient();
    $client->request( method: 'GET', uri: '/playlists');
    $client->clickLink( linkText: 'Voir détail');
    $response = $client->getResponse()->getContent();
    self::assertStringContainsString( needle: '<h4 class="text-info mt-5">Bases de la programmation (C#)</h4>', $response);
}

public function testSortPlaylistAscendant() {
    $client = self::createClient();
    $client->request( method: 'GET', uri: '/playlists/tri/name/ASC');
    $this->assertSelectorTextContains( selector: 'h5.text-info', text: 'Bases de la programmation (C#)');
}

public function testSortPlaylistDescendant() {...}

public function testRecherchePlaylist() {
    $client = self::createClient();
    $client->request( method: 'POST', uri: '/playlists/recherche/name', ['recherche' => 'test']);
    $this->assertSelectorCount( expectedCount: 1, selector: 'h5.text-info');
    $this->assertSelectorTextSame( selector: 'h5.text-info', text: 'playlist test');
}
```

Pour pouvoir tester les contrôleurs admin, on commence par s'identifier avec un utilisateur ayant le rôle admin puis on peut effectuer les mêmes types de vérifications :

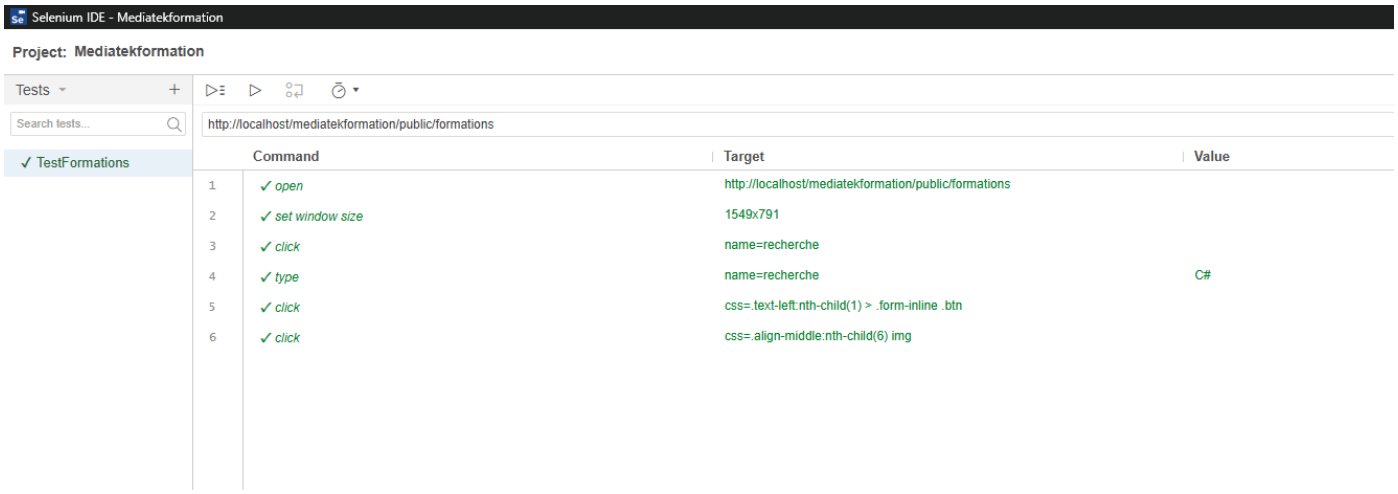
```php
12      private function loginClientAsAdmin($client) {
13          $userRepository = self::getContainer()->get(UserRepository::class);
14          $adminUser = $userRepository->findOneBy(['id' => 1]);
15          return $client->loginUser($adminUser);
16      }
17
18      public function testIndexAdminFormations() {
19          $client = self::createClient();
20          $this->loginClientAsAdmin($client);
21          $client->request( method: 'GET', uri: '/admin/formations');
22
23          $this->assertResponseIsSuccessful();
24      }
```

Voilà le résultat de tous les tests qui passent :



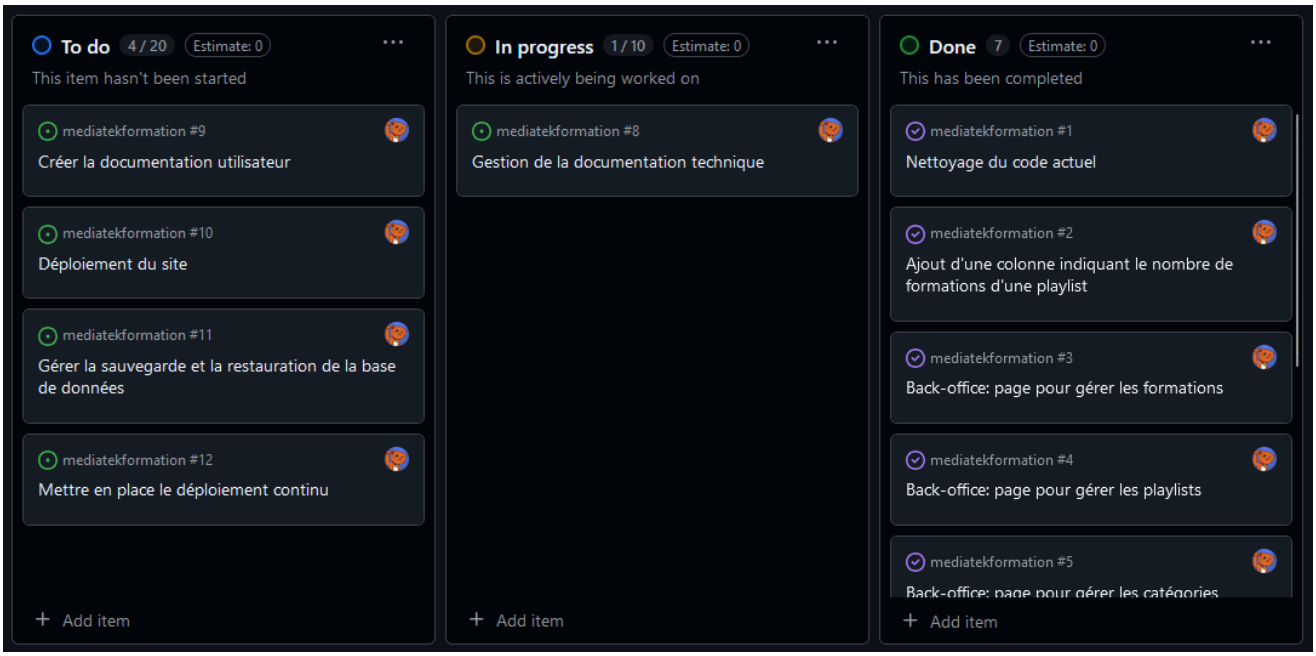
Enfin, pour les tests de compatibilité, on utilise Selenium pour simuler un navigateur et on teste un scénario pour voir s'il se rejoue avec succès.

On peut exporter ce test et l'exécuter sur différents navigateurs pour voir si cela fonctionne partout.

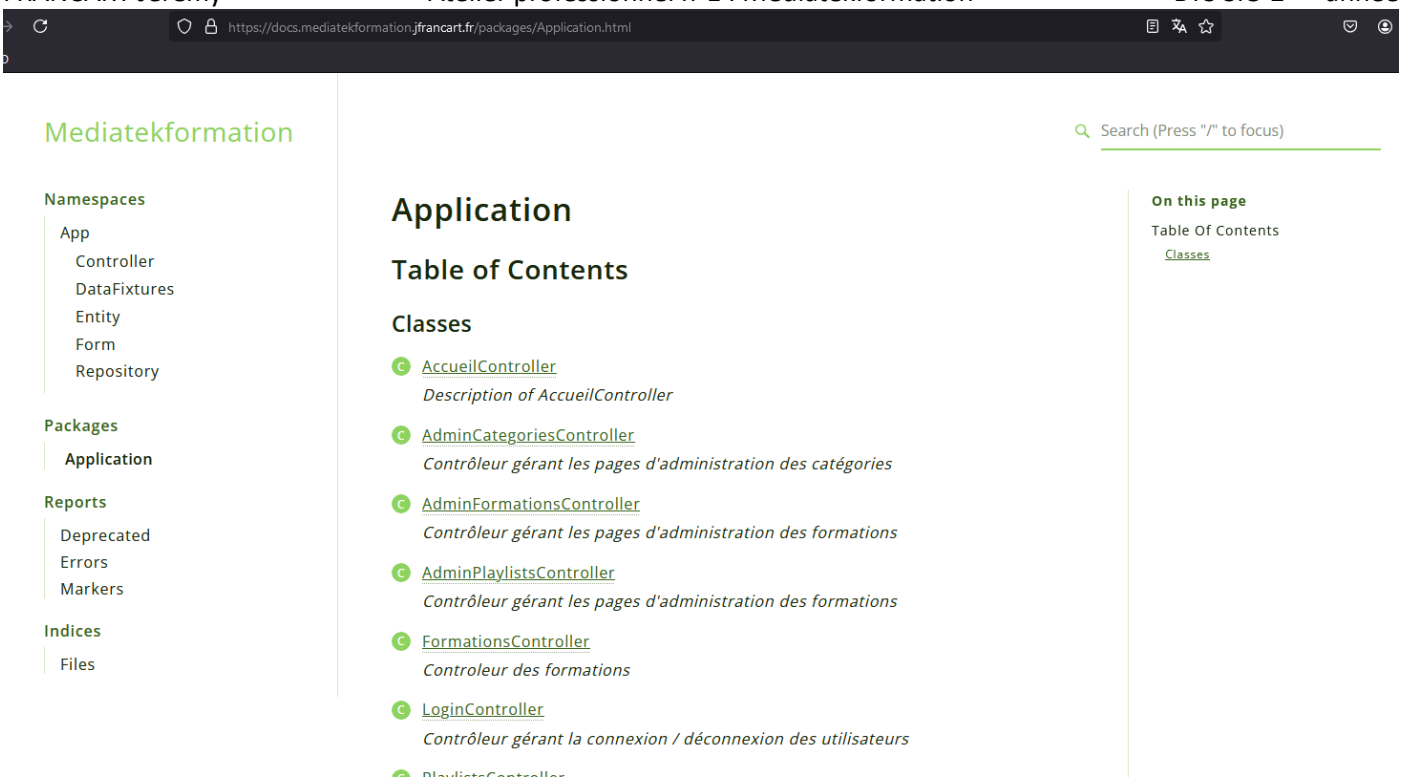


Tâche 2 : Créer la documentation technique – Temps estimé 1 heure / temps réel 1 heure

Pour cette tâche, il nous est demandé d'ajouter des commentaires normalisés à travers tout le code créé afin de pouvoir générer une documentation technique du site complète.

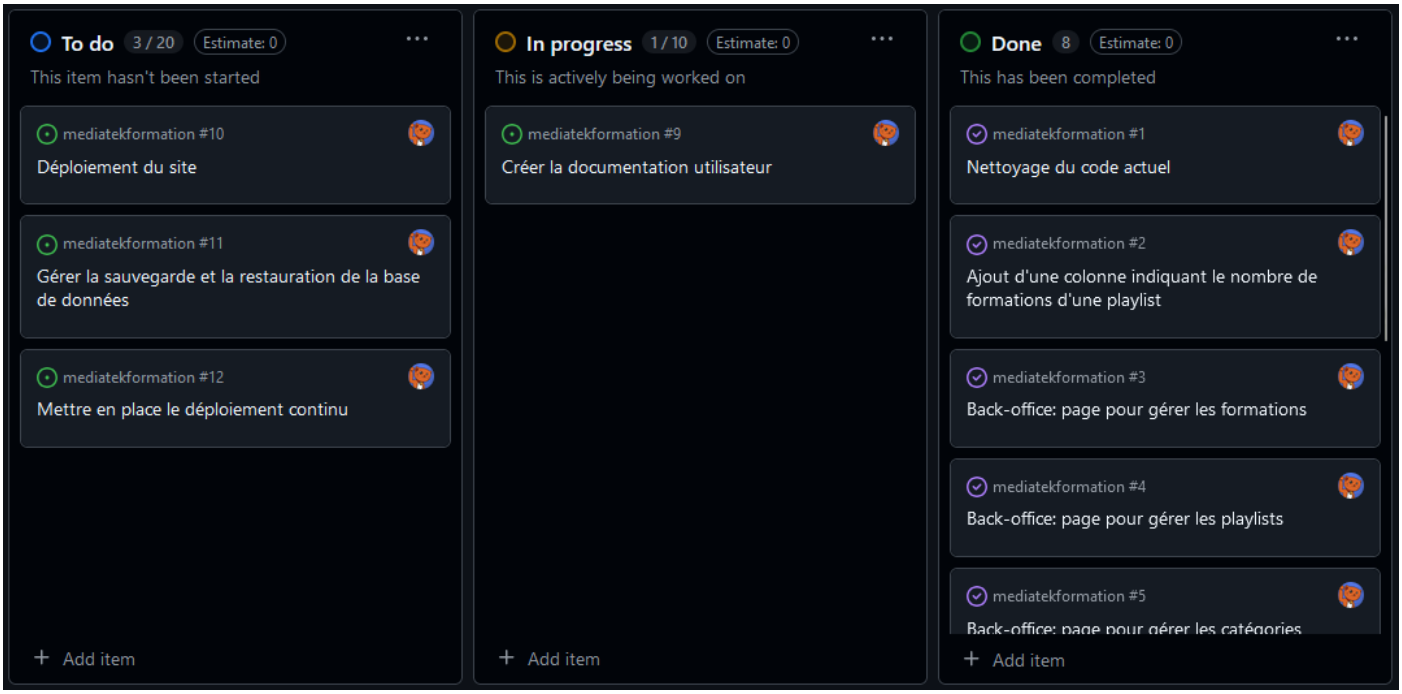


J'ai donc commencé par ajouter des commentaires normalisés dans tout le code qui a été créé par les développeurs. Ensuite j'ai testé la génération en local avec phpDocumentor. Puis, j'ai mis en place la génération et publication automatique vers GitHub Pages grâce à une GitHub Action. La documentation est maintenant accessible depuis https://docs.mediatekformation.jfrancart.fr



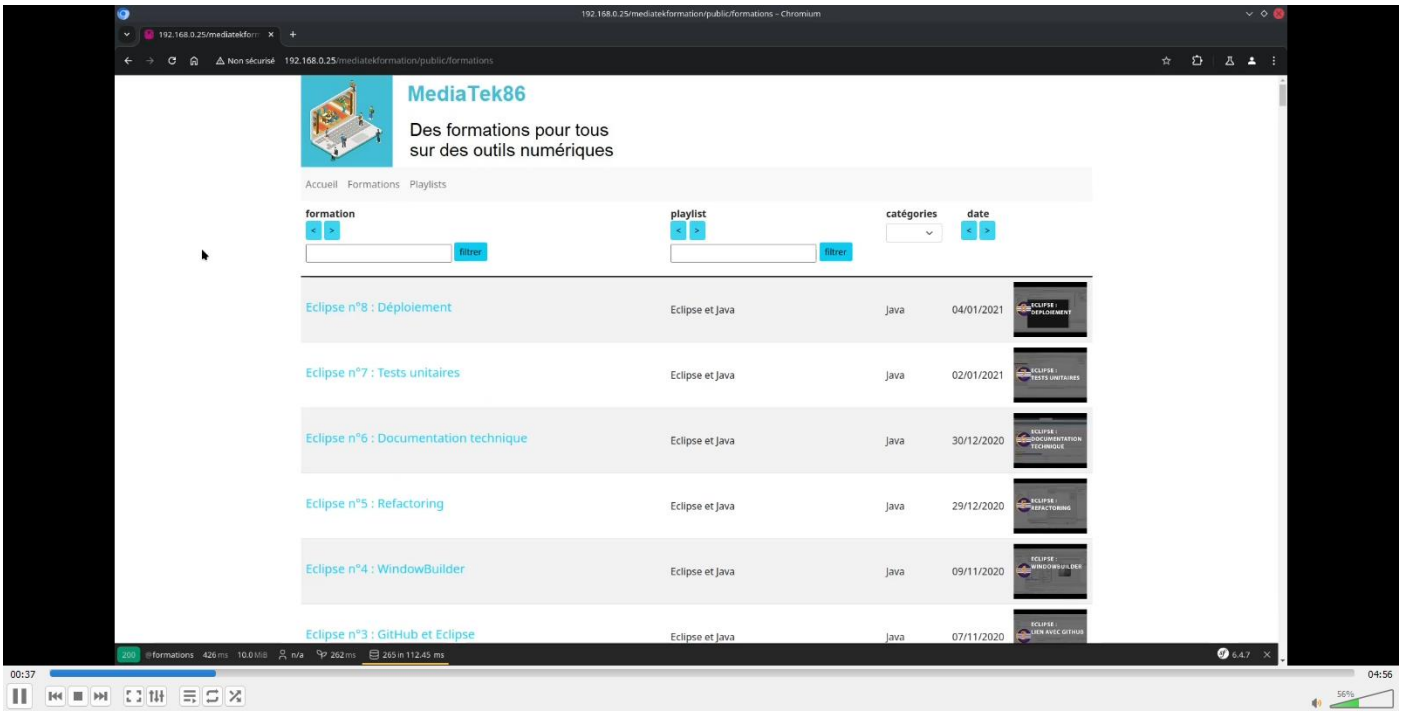
Tâche 3 : Créer la documentation utilisateur - Temps estimé 2 heures / temps réel 2 heures

Pour cette tâche, il nous est demandé de créer une vidéo montrant toutes les fonctionnalités du site internet avec explication orales et faisant moins de 5 minutes.



J'ai donc enregistré une vidéo après avoir préparé les outils et le script. J'ai ensuite coupé quelques parties de la vidéo afin de la rendre plus fluide.

Capture d'écran de la vidéo :



Étape 5 : Déployer le site et gérer le déploiement continu

Tâche 1 : déployer le site – Temps estimé 2 heures / temps réel 2 heures

Dans cette tâche, il faut déployer le site chez un hébergeur en mettant à jour notamment les conditions générales d'utilisation et la configuration pour rendre le site fonctionnel sur l'infrastructure de l'hébergeur.

J'ai pour ma part déployé le site sur un VPS hébergé chez IONOS en utilisant Docker.

Après connexion sur le VPS, j'ai commencé par cloner et paramétrer un projet utilisant docker-compose-lamp. Ce dépôt contient une base pour déployer un stack LAMP facilement avec Docker compose.

Pour déployer le site correctement, plusieurs étapes ont été réalisées :

- Clonage du code source de Mediatekformation
- Configuration DNS du domaine mediatekformation.jfrancart.fr
- Paramétrage de la base docker-compose-lamp pour l'adapter au projet
- Ajout de la base de données depuis le panel phpMyAdmin
- Configuration du proxy inversé de test utilisé (permet la gestion centralisée de plusieurs sites sur un serveur)
- Modification des paramètres d'accès à la base de données dans le fichier .env.local
- Installation des bundles utilisés dans l'application avec `composer install`
- Une fois que la configuration fonctionne sur le proxy inversé de test, déploiement sur celui de production
- Changement du mot de passe administrateur pour la partie back-office
- Création de services systemd pour automatiser le démarrage du site lors du redémarrage du serveur

Une fois ces étapes réalisées, le site est déployé et accessible sur https://mediatekformation.jfrancart.fr

Tâche 2 : Gérer la sauvegarde et la restauration de la base de données – Temps estimé 1 heure / temps réel 1 heure

Pour cette tâche, il nous est demandé d'automatiser la sauvegarde et de permettre la restauration de la base de données.

Pour la sauvegarde automatique, on peut créer un simple script qui va exécuter mysqldump et sauvegarder le résultat dans un dossier :

```
#!/bin/sh
DATE=`date -I`

# Suppression de l'ancien fichier de sauvegarde de BDD
#find /root/mediatekformation-bdd-bkps/bdd* -mtime -1 -exec rm {} \;

# Sauvegarde de la BDD dans /root/mediatekformation-bdd-bkps/
docker exec mediatekformation-mariadb106 /usr/bin/mysqldump -u root --password=tiger --databases mediatekformation --single-transaction | gzip > /root/mediatekformation-bdd-bkps/bddbackup_${DATE}.sql.gz
```

On teste le script et on observe le fichier résultant :

```
root@saesr-box:~# ./mediatekformation-bdd-bkp
root@saesr-box:~# ls -lah mediatekformation-bdd-bkps/
total 40K
drwxr-xr-x  2 root root 4.0K Feb  5 10:12 .
drwx------ 15 root root 4.0K Feb  5 10:40 ..
-rw-r--r--  1 root root  31K Feb  5 10:41 bddbackup_2025-02-05.sql.gz
```

Ensuite, on peut modifier le « crontab » pour lui dire d'exécuter le script tous les jours à 12:00 :

```
# m h  dom mon dow   command
00 12 * * * /root/mediatekformation-bdd-bkp
```

Tâche 3 : mettre en place le déploiement continu (estimé 1h réel 1h)

Pour cette dernière tâche, il nous est demandé de mettre en place le déploiement en continu, c'est-à-dire le déploiement automatique lors de changements dans le code source de l'application.

On va utiliser une action GitHub qui synchronise les fichiers avec le serveur à chaque fois qu'un commit est « push » dans le dépôt :

mediatekformation / .github / workflows / **deploy.yml**

Refragg Mise en place du déploiement continu

Code Blame 18 lines (17 loc) · 549 Bytes

```yaml
on: push
name: Déploiement du site après push
jobs:
  web-deploy:
    name: Deploy
    runs-on: ubuntu-latest
    steps:
      - name: Récupération du code source
        uses: actions/checkout@v2

      - name: Synchronisation des fichiers
        uses: SamKirkland/FTP-Deploy-Action@4.3.0
        with:
          server: ${{ secrets.ftp_server }}
          server-dir: ${{ secrets.ftp_server_dir }}
          port: ${{ secrets.ftp_port }}
          username: ${{ secrets.ftp_username }}
          password: ${{ secrets.ftp_password }}
```

Après configuration d'un serveur FTP sur le VPS et remplissage des secrets nécessaires dans le dépôt GitHub, on peut tester le déploiement continu en faisant une modification sur le site pour voir si elle est prise en compte automatiquement :

templates/pages/accueil.html.twig +1 -1

```
@@ -2,7 +2,7 @@
 {% block body %}
     <p class="mt-3">
-        <h3>Bienvenue sur le site de MediaTek86 consacré aux formations en ligne</h3>
+        <h3>Bienvenue sur le site de MediaTek86 consacré aux formations en ligne !</h3>
     </p>
     <p class="mt-3">
         Vous allez pouvoir vous former à différents outils numériques gratuitement et directement en ligne.<br />
```

Nous pouvons observer que le site s'est mis à jour automatiquement

Bilan

Le développement et le déploiement de ce site internet m'a permis d'élargir mes compétences notamment sur les langages et frameworks utilisés, la gestion des données d'un site, l'authentification, la gestion de projet pour réaliser les tâches à temps et autres.

Les objectifs de développement, documentation et déploiement ont pu être menés selon moi à bien malgré le fait que certaines libertés ont été prises ainsi ce qui a mené à certaines adaptations nécessaires.