

Atelier de professionnalisation 1

Application Web MediaTek86

Compte rendu



Carl Fremault - 2ième année BTS SIO - Mars 2022

Sommaire

1. Accès à l'application.....	3
2. Contexte.....	3
3. Présentation des missions.....	4
3.1 Mission 1 : ajouter les niveaux.....	4
3.2 Mission 2 : coder la partie back office.....	4
3.2.1 Formations.....	4
3.2.2 Niveaux.....	5
3.2.3 Sécurité.....	5
3.2.4 Tests unitaires.....	5
3.2.5 Accès avec authentification.....	5
3.2.6 Documentation technique.....	5
3.2.7 Scénario et test de compatibilité.....	6
3.3 Mission 3 : créer une vidéo de démonstration d'utilisation du site.....	6
4. Ressources logicielles utilisées.....	6
5. Mission 0 : Préparer l'environnement de travail.....	7
6. Mission 1 : Ajouter les niveaux.....	11
6.1 Ajouter une table dans la base de données pour mémoriser les niveaux, en créant en parallèle l'entity correspondante (utiliser 'composer' pour créer l'entity et la table)...	11
6.2 Remplir la table avec des exemples.....	13
6.3 Modifier la structure de la table 'formation' pour ajouter en clé étrangère l'id du niveau et modifier l'entity 'Formation'	13
6.4 Pour réaliser les tests, mettre un id de niveau aléatoire à toutes les formations existantes.....	15
6.5 Modifier la page d'affichage présentant les détails d'une formation en insérant le niveau.....	18
6.6 Modifier la page des formations pour insérer la colonne des niveaux ainsi que le filtrage.....	19
6.7 Bilan pour la mission 1 :	25
7. Mission 2 : Coder la partie back-office.....	25
7.1 Coder l'interface de la page d'administration sur le même format que le front office	26
7.2 Coder la page de gestion des formations avec les fonctionnalités demandées : lister les formations, boutons pour modifier, supprimer une formation,	27
7.3 Construire et gérer le formulaire qui permet de créer et modifier une formation.....	29
7.4 Coder la page de gestion des niveaux.....	40
7.5 Contrôler la sécurité.....	42
7.5.1 Partie Front Office :	42
7.5.2 Partie Back office :	45
7.6 Tests unitaires.....	49
7.7 Accès à la partie 'admin' avec authentification.....	49
7.8 Documentation technique.....	54
7.9 Créer un scénario sur la partie back-office pour tester la compatibilité des navigateurs.....	54
7.10 Bilan pour la mission 2 :	59
8. Mission 3 : Créer une vidéo de démonstration d'utilisation du site.....	60
9. Déploiement.....	60
10. Bilan final.....	62
11. Compétences couvertes.....	63

1. Accès à l'application

L'application web est disponible à l'adresse <https://mediatek86-formation.go.yj.fr/>.

L'accès au backoffice se fait par <https://mediatek86-formation.go.yj.fr/public/admin> avec utilisateur 'Mediadmin' et mot de passe 'M3di@dmin'.

Pour accéder à la base de données il faut se rendre sur <https://my.planethoster.com/> puis vous pouvez vous connecter avec les identifiants communiqués dans la fiche "Description d'une réalisation professionnelle". Dans l'onglet "Mes services" vous trouverez un lien "Panneau world" en bas à droite qui vous mène sur le cPanel de l'hébergeur.

Le script de la base de données est disponible dans le dépôt Github, dans le dossier 'script bdd'. Le script Selenium de test de compatibilité est disponible dans le dossier 'script selenium'.

Pour accéder au serveur KeyCloak l'adresse est la suivante :

<https://mediatekkc.francecentral.cloudapp.azure.com/>

Le nom d'utilisateur et le mot de passe sont spécifiés dans la fiche "Description d'une réalisation professionnelle" aussi.

2. Contexte

Pour enrichir ses services, MediaTek86, le réseau des médiathèques de la Vienne a décidé de proposer une application web qui propose des formations aux outils numériques et des autoformations en ligne, au format vidéo (YouTube).

Les premiers éléments de l'application ont été faits, avec une page d'accueil et une page qui propose une liste des formations actuellement disponibles. Pour chaque formation sont enregistrés : le titre, la date de parution, une description ainsi que deux URL pour des images (une image et une miniature) et un identifiant YouTube permettant de faire le lien avec ce plateforme.

Il est possible de visualiser une page qui présente les détails de chaque formation.

Pour ce projet j'ai été amené à faire évoluer cette application.

3. Présentation des missions

3.1 Mission 1 : ajouter les niveaux

Dans un premier temps il fallait implémenter la possibilité d'ajouter un niveau à chaque formation. Les différents niveaux doivent être enregistrés dans une table dans la base de données, et la table des formations doit intégrer un champ (avec clé étrangère) qui mémorise l'identifiant du niveau d'une formation.

Ce niveau doit apparaître pour les deux formations proposées sur la page d'accueil, sur la page qui présente les détails d'une formation ainsi que sur la page présentant la liste de tous les formations.

Sur la page qui présente la liste des formations il doit être possible de filtrer sur l'affichage pour un niveau choisi.

3.2 Mission 2 : coder la partie back office

La prochaine demande était de créer la partie back office du site. Elle doit permettre de gérer le contenu de la base de données. La présentation visuelle doit être la même que celle de la partie front office.

3.2.1 Formations

Une première page doit présenter une liste avec toutes les formations enregistrées, avec les mêmes fonctionnalités (recherche, tri, filtrage) que dans la partie front. Pour chaque formation, l'utilisateur doit disposer d'un bouton pour pouvoir la supprimer (après confirmation) et d'un bouton pour la modifier.

En cliquant sur ce bouton (modifier) l'utilisateur est redirigé vers un formulaire avec des champs pré-remplis qui reprennent les différentes données enregistrées concernant la formation. Il peut modifier les données puis les enregistrer. Les champs doivent être sécurisés et les saisies contrôlées.

Les champs 'titre', 'date de parution' et 'niveau' sont obligatoires. La date et le niveau doivent être sélectionnés et non pas saisis. Les champs 'miniature' et 'picture' contiennent des URL qui pointent vers des images. Les tailles des images doivent être contrôlés : 120x90 pixels pour les miniatures, et maximum 640x480 pixels pour les 'picture'.

Un bouton ajouté sur la page qui liste les formations permet d'enregistrer une nouvelle formation. L'utilisateur est amené vers le même formulaire qui évidemment n'est pas rempli cette fois.

3.2.2 Niveaux

Une deuxième page affiche les différents niveaux enregistrés dans la base de données. Pour chaque niveau, un bouton permet de le supprimer, à condition qu'il n'est pas utilisé pour une des formations. Il y a également un champ qui permet de saisir un libellé et ajouter un nouveau niveau.

3.2.3 Sécurité

L'application doit être entièrement sécurisé. Les requêtes à la base de données doivent être paramétrées pour éviter les injections SQL. Les formulaires doivent utiliser des token pour contrer les CSRF et les données saisies doivent être contrôlées.

3.2.4 Tests unitaires

L'application utilise une fonction qui retourne la date de parution d'une formation, dont l'enregistrement dans la base de données est fait avec le type 'datetime', formaté en chaîne de caractères, plus lisible pour les internautes. Il est demandé de créer un test unitaire pour cette fonction.

3.2.5 Accès avec authentification

La prochaine étape consiste à sécuriser l'accès au back office, en implémentant une fonctionnalité d'authentification avec nom d'utilisateur et mot de passe.

3.2.6 Documentation technique

Une documentation technique, couvrant la totalité du site (parties front et back) doit être générée.

3.2.7 Scénario et test de compatibilité

Finalement, pour cette deuxième mission, il fallait créer un scénario de test de compatibilité sur Selenium, et le jouer sur plusieurs navigateurs.

3.3 Mission 3 : créer une vidéo de démonstration d'utilisation du site

Pour la troisième, et dernière mission de ce projet, il fallait créer une vidéo de maximum cinq minutes, qui explique et montre clairement toutes les fonctionnalités de l'application web, pour les parties front et back.

4. Ressources logicielles utilisées

- L'application de départ qui nous a été fourni est écrite en PHP, en utilisant le framework Symfony.
- L'IDE que j'ai utilisé est NetBeans v12.0, avec installation des plugins Sonarlint4netbeans et GitHub Issues Support.
- L'application et la base de données tournent sur le serveur web WampServer.
- La base de données est définie en MySQL v5.7.36.
- Pour le versioning j'ai utilisé GitHub.
- Pour le suivi du projet j'ai utilisé GitHub Projects avec le template 'Automated Kanban', lié avec NetBeans par le plugin mentionné GitHub Issues Support. Cette configuration permet de gérer les différentes 'issues' du projet directement depuis l'IDE.
- L'authentification est gérée par KeyCloak v17.0.1 (Quarkus).
- Pour la génération de documentation technique j'ai utilisé phpDocumentor.
- La gestion de qualité du code et l'intégration continue ont été faits avec SonarQube et Jenkins.
- La vidéo de démonstration d'utilisation du site a été enregistré avec OBS.

5. Mission 0 : Préparer l'environnement de travail

Avant d'entamer les travaux il convient de vérifier que tous les outils nécessaires sont en place. Aussi je me suis familiarisé avec le contexte et les demandes pour ce projet. Voici les travaux réalisés pendant la préparation :

- Vérification de l'installation des outils de travail nécessaires (NetBeans IDE, WampServer)
- Familiarisation avec le dossier documentaire
- Récupération du code source du projet depuis le dépôt GituHb MediaTek86
- Création du projet 'mediatekformation' dans NetBeans
- Familiarisation avec le code du projet
- Récupération du script de création et remplissage de la base de données MySQL 'mediatekformations'
- Création des bases de données (une pour l'application, une pour Symfony) dans PhpMyAdmin depuis le script SQL fourni

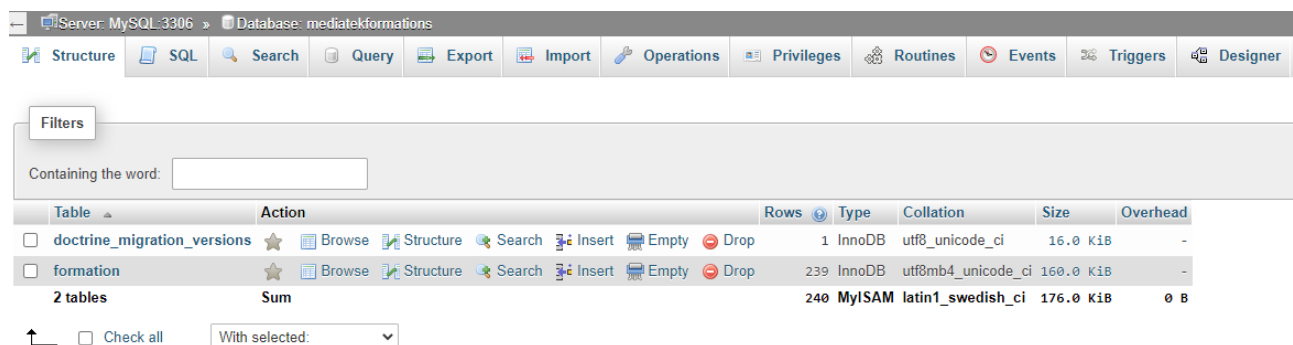


Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> doctrine_migration_versions	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8_unicode_ci	16.0 KiB	-
<input type="checkbox"/> formation	★ Browse Structure Search Insert Empty Drop	239	InnoDB	utf8mb4_unicode_ci	160.0 KiB	-
2 tables		Sum				
		240	MyISAM	latin1_swedish_ci	176.0 KiB	0 B

Figure 1: Les bases de données sont importées

- Test de l'application dans son état initial



Figure 2: Écran de bienvenue du site

- Création du dépôt GitHub 'mediatekformation', liaison du dépôt avec le projet local
- Création de projet GitHub 'mediatekformation', installation du plugin 'GitHub Issues Support' dans NetBeans et création de lien avec les 'issues' du projet.

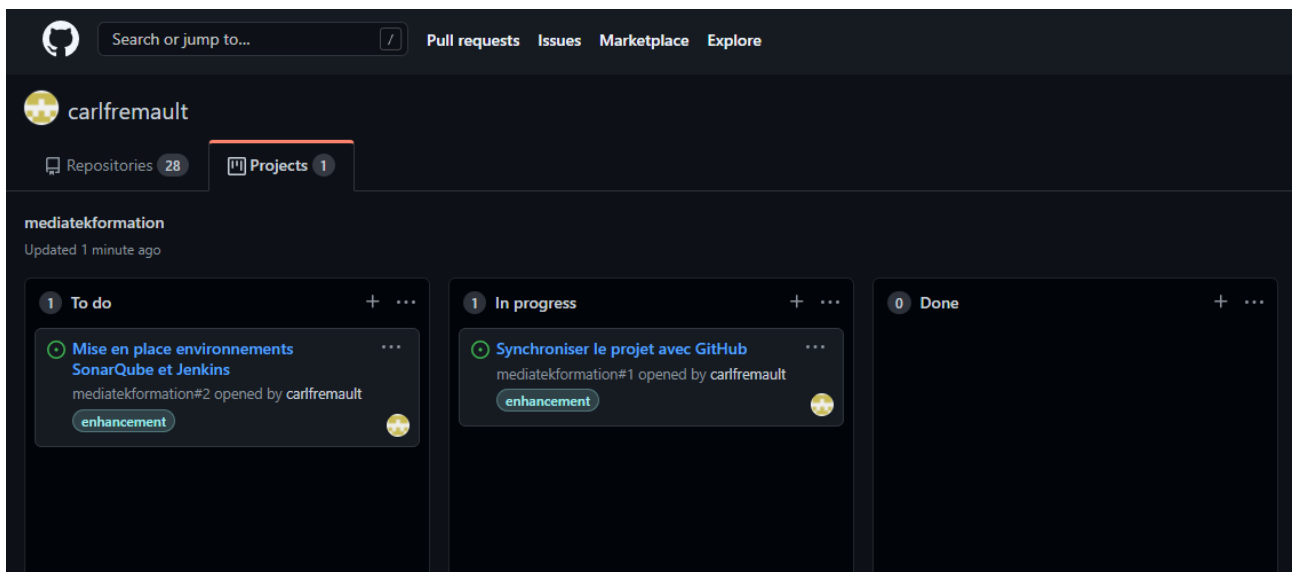


Figure 3: Le Storyboard sur GitHub

- Configuration SonarQube et Jenkins, installation plugin 'SonarLint4NetBeans' dans NetBeans

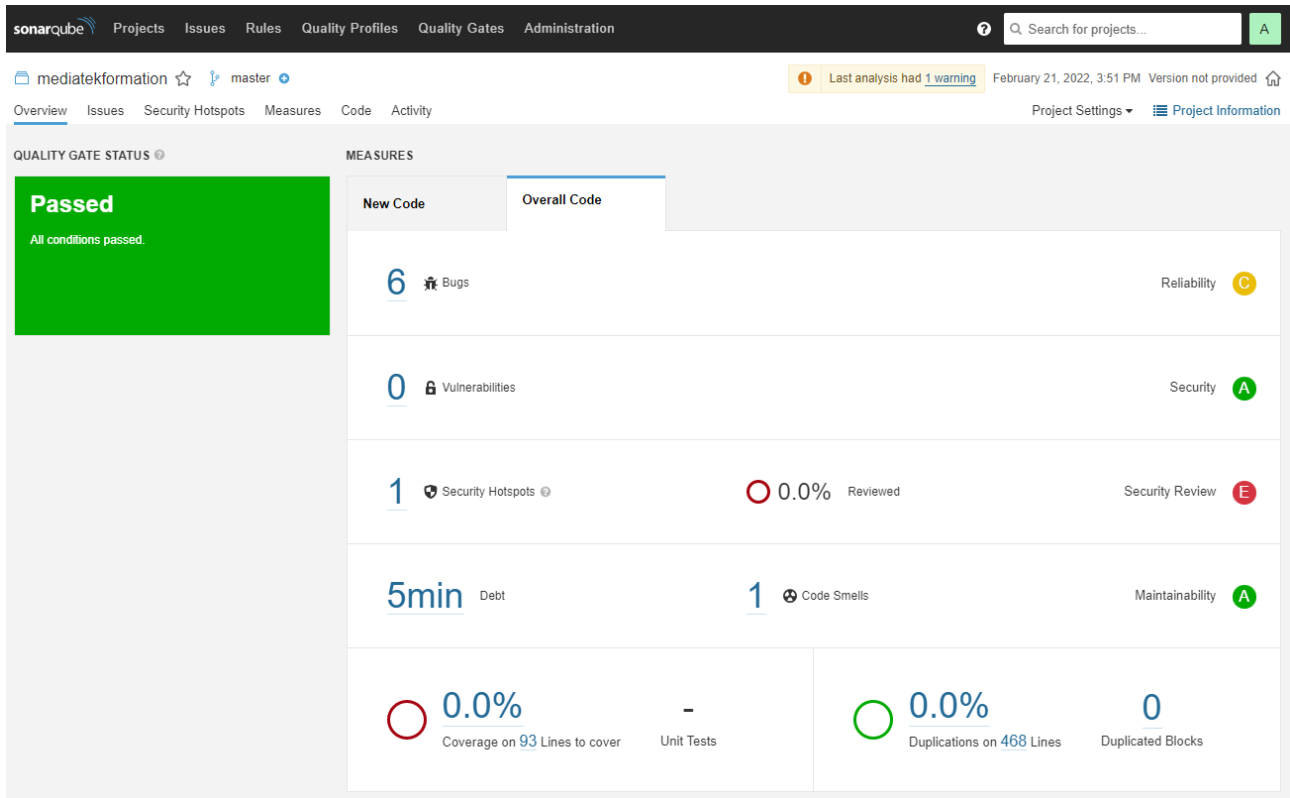


Figure 4: Première analyse SonarQube

- Création d'une branche 'dev' puis nettoyage initial du code suite aux recommandations SonarQube. (captions des tableaux (ignoré 1 fois), attributs 'alt' pour les images)
- Commit, push et merge avec la branche 'main'



Figure 5: Commit après nettoyage initial

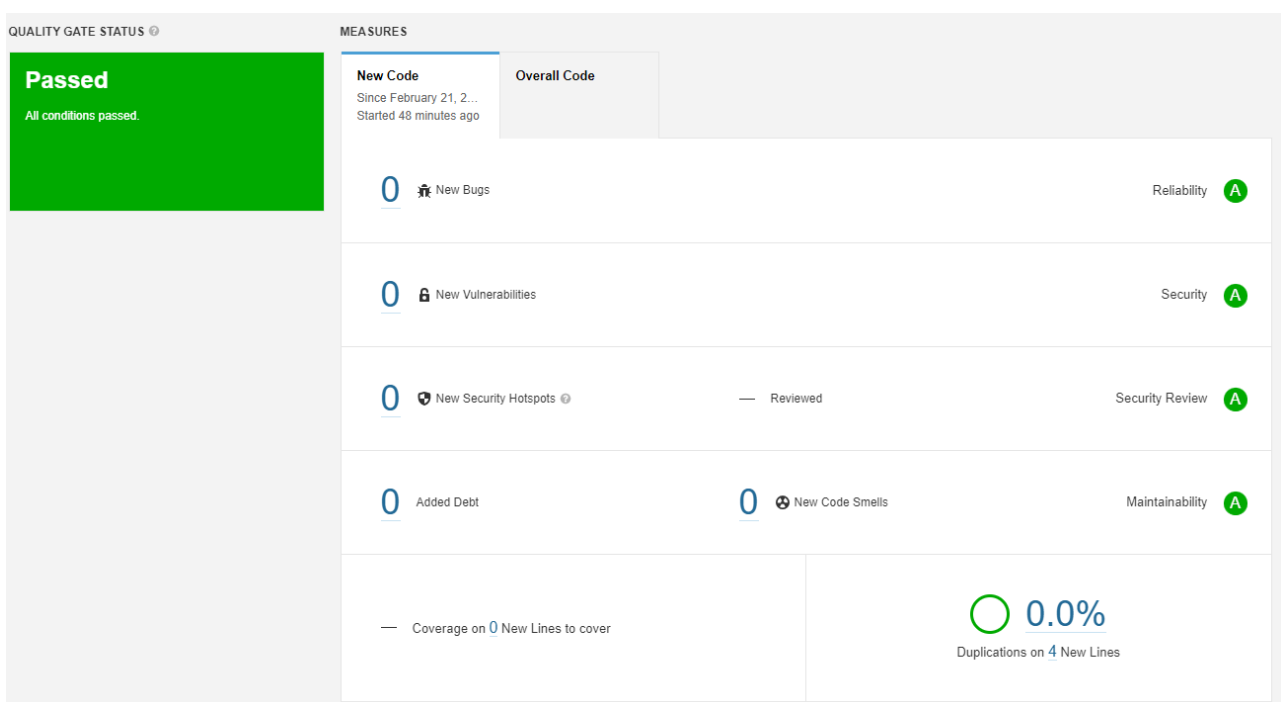


Figure 6: Analyse SonarQube après nettoyage initial

6. Mission 1 : Ajouter les niveaux

Plusieurs tâches étaient à réaliser pour cette première mission :

- Ajouter une table dans la base de données pour mémoriser les niveaux, en créant en parallèle l'entity correspondante (utiliser 'composer' pour créer l'entity et la table).
- Remplir la table avec des exemples.
- Modifier la structure de la table 'formation' pour ajouter en clé étrangère l'id du niveau et modifier l'entity 'Formation'.
- Pour réaliser les tests, mettre un id de niveau aléatoire à toutes les formations existantes.
- Modifier la page d'affichage présentant les détails d'une formation en insérant le niveau.
- Modifier la page des formations pour insérer la colonne des niveaux ainsi que le filtrage.

6.1 Ajouter une table dans la base de données pour mémoriser les niveaux, en créant en parallèle l'entity correspondante (utiliser 'composer' pour créer l'entity et la table)

J'ai commencé par créer la nouvelle table avec Symfony :

```
Administrator: Command Prompt

c:\wamp64\www\mediatekformation>php bin/console make:entity

Class name of the entity to create or update (e.g. GrumpyPuppy):
> niveau

created: src/Entity/Niveau.php
created: src/Repository/NiveauRepository.php

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

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

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

Field length [255]:
> 50

Can this field be null in the database (nullable) (yes/no) [no]:
> yes

updated: src/Entity/Niveau.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>

Success!

Next: When you're ready, create a migration with php bin/console make:migration

c:\wamp64\www\mediatekformation>
```

Figure 7: Création d'une Entity 'niveau'

```
Administrator: Command Prompt

c:\wamp64\www\mediatekformation>php bin/console make:migration

Success!

Next: Review the new migration "migrations/Version20220221162247.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html

c:\wamp64\www\mediatekformation>php bin/console doctrine:migrations:migrate

WARNING! You are about to execute a migration in database "mediatekformations" that could result in schema changes and
data loss. Are you sure you wish to continue? (yes/no) [yes]:
>

[notice] Migrating up to DoctrineMigrations\Version20220221162247
[notice] finished in 445.3ms, used 20M memory, 1 migrations executed, 2 sql queries

c:\wamp64\www\mediatekformation>
```

Figure 8: Migration réussie

6.2 Remplir la table avec des exemples

Pour l'insertion des valeurs, j'aurais pu créer une 'fixture' mais pour les quelques valeurs c'était bien plus efficace de le faire directement dans la base de données.

```
✓ 3 rows inserted.  
Inserted row id: 3 (Query took 0.0013 seconds.)  
  
INSERT INTO niveau(libelle) VALUES("débutant"), ("confirmé"),("expert");  
  
\[ Edit inline \] \[ Edit \] \[ Create PHP code \]
```

Figure 9: Insertion des niveaux dans la base de données

Puis j'ai fait un commit et push avant de modifier la table existante 'formation' :

```
Phase 3 : Création de table et entity 'niveau', création d'un champ '...  
_niveau' dans la table 'formations' (clé étrangère) puis création d'une 'fixture' pour remplir le nouveau champ avec une valeur aléatoire entre 1 et 3.  
  
Task #5 - Creation de table 'niveau'
```

Figure 10: Commit après la création de l'Entity

6.3 Modifier la structure de la table 'formation' pour ajouter en clé étrangère l'id du niveau et modifier l'entity 'Formation'

J'ai ajouté un champ contenant l'id du niveau dans la table 'formation', toujours depuis la ligne de commande :

```

c:\wamp64\www\mediatekformation>php bin/console make:entity formation

Your entity already exists! So let's add some new fields!

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

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

What class should this entity be related to?:
> niveau

What type of relationship is this?
-----
Type           Description
-----
ManyToOne      Each Formation relates to (has) one niveau.
                Each niveau can relate to (can have) many Formation objects
OneToMany      Each Formation can relate to (can have) many niveau objects.
                Each niveau relates to (has) one Formation
ManyToMany     Each Formation can relate to (can have) many niveau objects.
                Each niveau can also relate to (can also have) many Formation objects
OneToOne       Each Formation relates to (has) exactly one niveau.
                Each niveau also relates to (has) exactly one Formation.
-----

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToOne

Is the Formation.niveau property allowed to be null (nullable)? (yes/no) [yes]:
>

Do you want to add a new property to niveau so that you can access/update Formation objects from it - e.g. $niveau->get
Formations()? (yes/no) [yes]:
> no

updated: src/Entity/Formation.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>

Success!

```

Figure 11: Modification de l'entity 'Formation' pour la rélier à 'Niveau'

6.4 Pour réaliser les tests, mettre un id de niveau aléatoire à toutes les formations existantes

Afin d'automatiser cette tâche j'ai créé une 'fixture' :

```
c:\wamp64\www\mediatekformation>php bin/console make:fixture

The class name of the fixtures to create (e.g. AppFixtures):
> formationFixtures

created: src/DataFixtures/FormationFixtures.php

Success!

Next: Open your new fixtures class and start customizing it.
Load your fixtures by running: php bin/console doctrine:fixtures:load
Docs: https://symfony.com/doc/current/bundles/DoctrineFixturesBundle/index.html

c:\wamp64\www\mediatekformation>
```

Figure 12: Fixture pour automatiser la saisie des niveaux des formations

Puis j'ai inséré du code dans la méthode 'load' pour générer aléatoirement des valeurs pour l'attribution des niveaux ... :

```

1  <?php
2
3  namespace App\DataFixtures;
4
5  use App\Repository\FormationRepository;
6  use App\Repository\NiveauRepository;
7  use Doctrine\Bundle\FixturesBundle\Fixture;
8  use Doctrine\Persistence\ObjectManager;
9
10 class FormationFixtures extends Fixture
11 {
12     /**
13      *
14      * @var FormationRepository
15      */
16     private $formationRepository;
17
18     /**
19      *
20      * @var NiveauRepository
21      */
22     private $niveauRepository;
23
24     /**
25      * @param FormationRepository $formationRepository
26      * @param NiveauRepository $niveauRepository
27      */
28     function __construct(FormationRepository $formationRepository, NiveauRepository $niveauRepository) {
29         $this->formationRepository = $formationRepository;
30         $this->niveauRepository = $niveauRepository;
31     }
32
33
34     public function load(ObjectManager $manager): void
35     {
36         $formations = $this->formationRepository->findAll();
37         $niveaus = $this->niveauRepository->findAll();
38         foreach($formations as $formation) {
39             $id = rand(0, 2);
40             $unNiveau = $niveaus[$id];
41             $formation->setNiveau($unNiveau);
42             $manager->persist($formation);
43         }
44         $manager->flush();
45     }
46 }
47

```

Figure 13: Code de la fixture

... suivi de l'exécution du 'load' (NiveauFixtures était créé par erreur et était vide, et supprimé par la suite)

```
c:\wamp64\www\mediatekformation>php bin/console doctrine:fixtures:load --append

> loading App\DataFixtures\AppFixtures
> loading App\DataFixtures\FormationFixtures
> loading App\DataFixtures\NiveauFixtures

c:\wamp64\www\mediatekformation>
```

Figure 14: Chargement des fixtures

Toutes les formations avaient dorénavant un 'niveau_id' d'attribué :

			id	published_at	title	description	miniature	picture	video_id	niveau_id	
<input type="checkbox"/>	Edit	Copy	Delete	1	2021-11-22 00:00:00	Eclipse n°8 : Déploiement	Exécution de l'application en dehors de l'IDE, en ...	https://i.ytimg.com/vi/Z4yTTXka958/default.jpg	https://i.ytimg.com/vi/Z4yTTXka958/sddefault.jpg	Z4yTTXka958	1
<input checked="" type="checkbox"/>	Edit	Copy	Delete	2	2020-12-28 21:55:06	Eclipse n°7 : Tests unitaires	Intégration de JUnit dans l'application et créatio...	https://i.ytimg.com/vi/-nw42Xq6cYE/default.jpg	https://i.ytimg.com/vi/-nw42Xq6cYE/sddefault.jpg	-nw42Xq6cYE	2
<input type="checkbox"/>	Edit	Copy	Delete	3	2020-12-28 21:49:27	Eclipse n°6 : Documentation technique	Intégration des commentaires normalisés et générat...	https://i.ytimg.com/vi/PrK_P3TKc00/default.jpg	https://i.ytimg.com/vi/PrK_P3TKc00/sddefault.jpg	PrK_P3TKc00	3
<input checked="" type="checkbox"/>	Edit	Copy	Delete	4	2020-12-28 21:37:57	Eclipse n°5 : Refactoring	Utilisation des outils de refactoring et de généra...	https://i.ytimg.com/vi/1p_mKDDSMnQ/default.jpg	https://i.ytimg.com/vi/1p_mKDDSMnQ/sddefault.jpg	1p_mKDDSMnQ	1
<input type="checkbox"/>	Edit	Copy	Delete	5	2020-11-04 14:06:07	Eclipse n°4 : WindowBuilder	Intégration de l'outil WindowBuilder dans Eclipse ...	https://i.ytimg.com/vi/pQfbr3hpw04/default.jpg	https://i.ytimg.com/vi/pQfbr3hpw04/sddefault.jpg	pQfbr3hpw04	1
<input checked="" type="checkbox"/>	Edit	Copy	Delete	6	2020-11-03 17:39:37	Eclipse n°3 : GitHub et Eclipse	Créer un compte sur le site GitHub (site offrant u...	https://i.ytimg.com/vi/mIN7VvZkXtM/default.jpg	https://i.ytimg.com/vi/mIN7VvZkXtM/sddefault.jpg	mIN7VvZkXtM	2
<input type="checkbox"/>	Edit	Copy	Delete	7	2020-11-03 17:19:30	Eclipse n°2 : rétroconception avec ObjectAid	Utilisation de l'outil ObjectAid sous Eclipse pour...	https://i.ytimg.com/vi/9UBtVxHsnNk/default.jpg	https://i.ytimg.com/vi/9UBtVxHsnNk/sddefault.jpg	9UBtVxHsnNk	2

Figure 15: Les formations dans la base de données...

	video_id	niveau_id
pg	Z4yTTXka958	1
pg	-nw42Xq6cYE	2
pg	PrK_P3TKc00	3
lt.jpg	1p_mKDDSMnQ	1
ig	pQfbr3hpw04	1
pg	mIN7VvZkXtM	2
pg	9UBtVxHsnNk	2

Figure 16: ... avec un niveau attribué partout

6.5 Modifier la page d'affichage présentant les détails d'une formation en insérant le niveau

D'abord j'ai inséré l'affichage du niveau dans le template 'formation.html.twig' :



MediaTek86

Des formations sur des outils numériques pour tous

Accueil Formations



28/12/2020

Eclipse n°8 : Déploiement

niveau : expert

description :

Exécution de l'application en dehors de l'IDE, en invite de commande.
Création d'un fichier jar pour le déploiement de l'application.
00:20 : exécuter l'application à partir d'un invite de commandes
04:41 : créer un fichier jar auto exécutable
06:42 : exécuter un fichier jar directement
07:09 : exécuter un fichier jar dans l'invite de commande pour avoir les retours console

Figure 17: Modification de la vue des détails d'une formation pour afficher le niveau

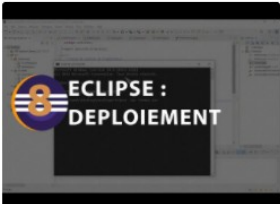
... et puis au niveau de la présentation des deux formations les plus récentes sur la page d'accueil :

Bienvenue sur le site de MediaTek86 consacré aux formations

Vous allez pouvoir vous former à différents outils numériques gratuitement et directement en ligne.

Dans la partie [Formations](#), vous trouverez la liste des formations proposées. Vous pouvez faire une recherche à partir d'un mot, trier les formations sur le titre ou la date de parution et, en cliquant sur la miniature, vous accéderez à la présentation plus détaillée de la formation ainsi que la vidéo correspondante.

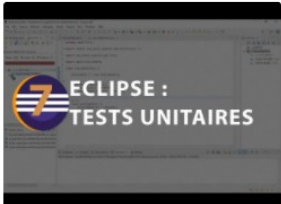
Voici les deux dernières formations ajoutées au catalogue :



22/11/2021

Eclipse n°8 : Déploiement

débutant



28/12/2020

Eclipse n°7 : Tests unitaires

confirmé

Figure 18: Modification de la page de bienvenue

J'ai fait un autre commit et push après ces changements :

```
Phase 4 : Insertion affichage des niveaux sur la page 'formations' et...
... sur les pages des détails d'une formations

+ ajout de lien vers une formation depuis le titre affiché dans la liste
Task #6 - Insertion affichage 'niveau' dans la page 'formations' et les pages des détails d'une formation
```

Figure 19: Commit après implémentation affichage niveaux

6.6 Modifier la page des formations pour insérer la colonne des niveaux ainsi que le filtrage

J'ai inséré l'affichage du niveau dans le template 'formations.html.twig' et créé une petite fonction dans le modèle 'Formation' pour pouvoir récupérer le libellé du niveau sous forme de chaîne de caractères (car la propriété 'niveau' dans Formation est de type 'niveau' et contient donc un objet de ce type) :

```
public function getNiveauString(): ?string
{
    return $this->niveau->getLibelle();
}
```

Figure 20: Getter pour le libellé d'une formation

```

  {% for formation in formations %}
    <tr>
      <td>
        <h5 class="text-info">
          <a href="{{ path('formations.showone', {id:formation.id}) }}">
            {{ formation.title }}
          </a>
        </h5>
      </td>
      <td class="text-center">
        {{ formation.niveaustring }}
      </td>
      <td class="text-center">
        {{ formation.publishedatstring }}
      </td>
      <td class="text-center">
        {% if formation.miniature %}
          <a href="{{ path('formations.showone', {id:formation.id}) }}">
            
          </a>
        {% endif %}
      </td>
    </tr>
  {% endfor %}
</tbody>

```

Figure 21: Affichage niveaux dans la vue liste de formations

Le résultat :



titre < >	niveau < >
<input type="text"/> <input type="button" value="filtrer"/>	
Eclipse n°8 : Déploiement	expert 28/12/2020 
Eclipse n°7 : Tests unitaires	expert 28/12/2020 

Figure 22: La vue après modification

Ensuite j'ai inséré le filtre pour les niveaux sur la page des formations (toujours dans le template 'formations.html.twig') :

```

<th class="text-center align-top" style="width: 15%" scope="col">
    niveau
    <div class="dropdown">
        <button class="btn btn-info btn-sm dropdown-toggle mt-2" type="button" id="
dropdownMenuButton" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
            filtrer
        </button>
        <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
            <a class="dropdown-item" href="{ path('formations.filter', {champ: 'niveau'}) }}">
                tous
            </a>
            {% for niveau in niveaux %}
                <a class="dropdown-item" href="{ path('formations.filter', {champ: 'niveau', valeur:
niveau.id}) }}">{{ niveau.libelle }}</a>
            {% endfor %}
        </div>
    </div>
</th>

```

Figure 23: Implémentation du filtre avec menu déroulant

Pour cela j'ai dû légèrement modifier le contrôleur, pour que le rendering d'une page prend un deuxième paramètre : un array avec les différents niveaux. J'ai créé une nouvelle propriété 'niveauRepository', initialisé dans le constructeur qui par la suite appelle la fonction 'findAll' sur ce repository pour remplir la propriété 'niveaux' qui contient ainsi un vecteur avec tous les niveaux présents dans la base de données au moment de la génération de la page.

En ajoutant ce vecteur en paramètre dans la méthode 'render' les différents niveaux sont disponibles pour 'remplir' le menu déroulant.

```

17 class FormationsController extends AbstractController {
18
19     private const PAGEFORMATIONS = "pages/formations.html.twig";
20
21     /**
22      * @var FormationRepository
23      */
24     private $formationRepository;
25
26     /**
27      * @var NiveauRepository
28      */
29     private $niveauRepository;
30
31     /**
32      *
33      * @var Niveaux[]
34      */
35     private $niveaux;
36
37     /**
38      *
39      * @param FormationRepository $formationRepository
40      */
41     function __construct(FormationRepository $formationRepository, NiveauRepository $niveauRepository) {
42         $this->formationRepository = $formationRepository;
43         $this->niveauRepository = $niveauRepository;
44         $this->niveaux = $this->niveauRepository->findAll();
45     }

```

Figure 24: Modification du contrôleur pour mémoriser les niveaux

```

/**
 * @Route("/formations", name="formations")
 * @return Response
 */
public function index(): Response {
    $formations = $this->formationRepository->findAllOrderBy('publishedAt', 'DESC');
    return $this->render(self::PAGEFORMATIONS, [
        'formations' => $formations,
        'niveaux' => $this->niveaux
    ]);
}

```

Figure 25: La méthode index utilise la nouvelle propriété

Le résultat :

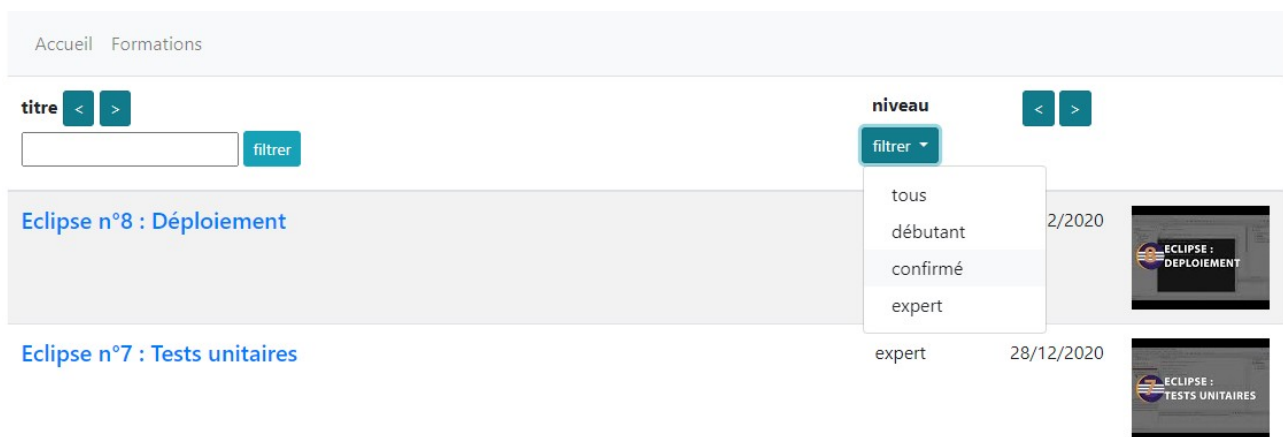


Figure 26: Filtrage sur les niveaux

Un autre commit et push ...

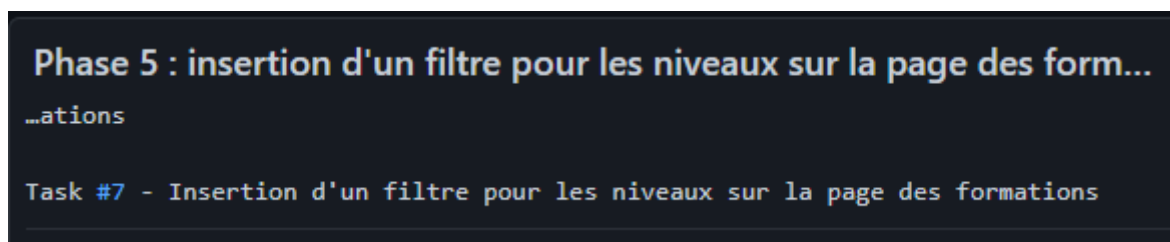


Figure 27: Commit phase 5

... puis un merge de la branche 'dev' avec 'main' à la fin de la mission 1, suivi d'un 'build' sur Jenkins. SonarQube avait quelques remarques :

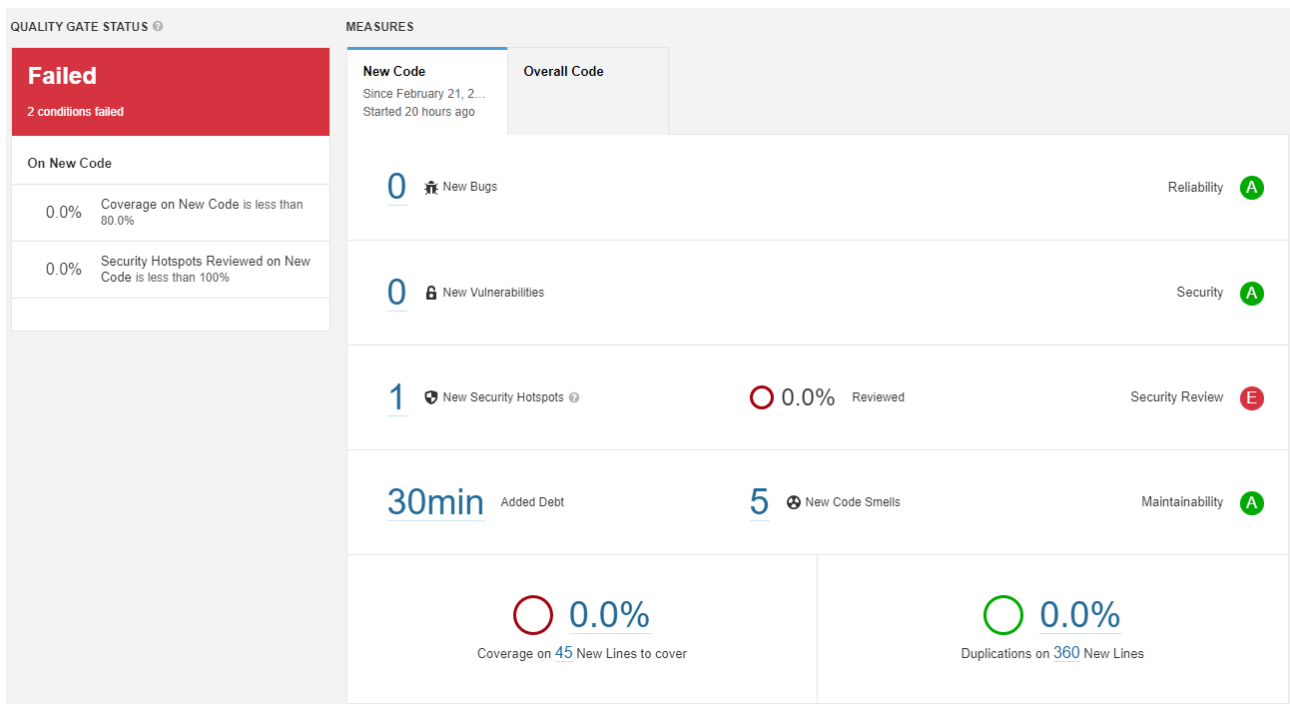


Figure 28: Résultat analyse SonarQube

Certaines des remarques pouvaient être ignorées cependant.

Make sure that using this pseudorandom number generator is safe here. [Add Comment](#) [Open in IDE](#) [Get Permalink](#)

Using pseudorandom number generators (PRNGs) is security-sensitive [php:S2245](#)

Category: **Weak Cryptography**

Review priority: **MEDIUM**

Assignee: **Not assigned**

Status: **To review**
This Security Hotspot needs to be reviewed to assess whether the code poses a risk.

[Change status](#)

[src/DataFixtures/FormationFixtures.php](#)

```

34 public function load(ObjectManager $manager)
35 {
36     $formations = $this->formationRepository
37     $niveaus = $this->niveauRepository->find
38     foreach($formations as $formation) {
39         $id = rand(0, 2);
40         $unNiveau = $niveaus[$id];
41         $formation->setNiveau($unNiveau);
42         $manager->persist($formation);
43     }
44     $manager->flush();

```

To review

☐ This Security Hotspot needs to be reviewed to assess whether the code poses a risk.

Fixed

☐ The code has been modified to follow recommended secure coding practices.

Safe

☒ The code is not at risk and doesn't need to be modified.

Add a comment (Optional)

aucun risque, génération d'un int aléatoire pour remplir table

Formatting Help: *Bold* ``Code`` * Bulleted point

[Change status](#)

What's the risk? Are you at risk? How can you fix it?

Figure 29: Une fausse alerte

Après avoir corrigé quelques points mineurs (notamment la suppression de code commenté ajouté automatiquement par Symfony lors de la création des fixtures) et ignoré d'autres, un commit et push s'impose :

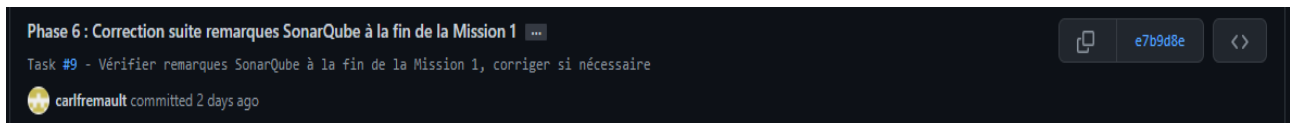


Figure 30: Commit après correction remarques SonarQube

Enfin (j'aurais dû le faire avant le merge de la fin de cette mission) j'ai créé des tests unitaires pour la fonction 'getPublishedAtString' qui transforme la date de création en chaîne de caractères pour une meilleure lisibilité, et la fonction 'getNiveauString' qui récupère le libellé du niveau attribué à une formation.

```
class FormationTest extends TestCase {  
  
    public function testGetPublishedAtString() {  
        $formation = new Formation();  
        $formation->setPublishedAt(new DateTime("2022-02-22"));  
        $this->assertEquals("22/02/2022", $formation->getPublishedAtString());  
    }  
  
    public function testGetNiveauString() {  
        $formation = new Formation();  
        $niveau = new Niveau();  
        $niveau->setLibelle("testNiveau");  
        $formation->setNiveau($niveau);  
        $this->assertEquals($niveau->getLibelle(), $formation->getNiveauString());  
    }  
}
```

Figure 31: Tests unitaires

Les tests passent :

```
c:\wamp64\www\mediatekformation>php bin/phpunit  
PHPUnit 9.5.14 by Sebastian Bergmann and contributors.  
  
Testing  
..  
Time: 00:00.044, Memory: 8.00 MB  
  
OK (2 tests, 2 assertions)  
  
c:\wamp64\www\mediatekformation>
```

Figure 32: Succès !

Il était temps de faire un dernier commit et push pour cette première mission, puis un merge des branches 'dev' et 'main' afin de partir sur de bonnes bases pour la prochaine mission.

6.7 Bilan pour la mission 1 :

Les différentes demandes ont été gérées sans difficultés particulières. Les niveaux sont affichés, aussi bien dans la base de données que sur les différentes pages de l'application. Le filtrage sur un niveau est fonctionnel.

7. Mission 2 : Coder la partie back-office

Voici les tâches qui m'ont été confiées pour cette deuxième mission :

- Coder l'interface de la page d'administration sur le même format que le front office
- Coder la page de gestion des formations avec les fonctionnalités demandées : lister les formations, boutons pour modifier, supprimer une formation, ...
- Construire et gérer le formulaire qui permet de créer et modifier une formation
- Coder la page de gestion des niveaux
- Contrôler la sécurité
- Tests unitaires
- Accès à la partie 'admin' avec authentification
- Documentation technique
- Créer un scénario sur la partie back-office pour tester la compatibilité des navigateurs

Ainsi j'ai commencé par mettre à jour le storyboard avec les tâches de cette mission :

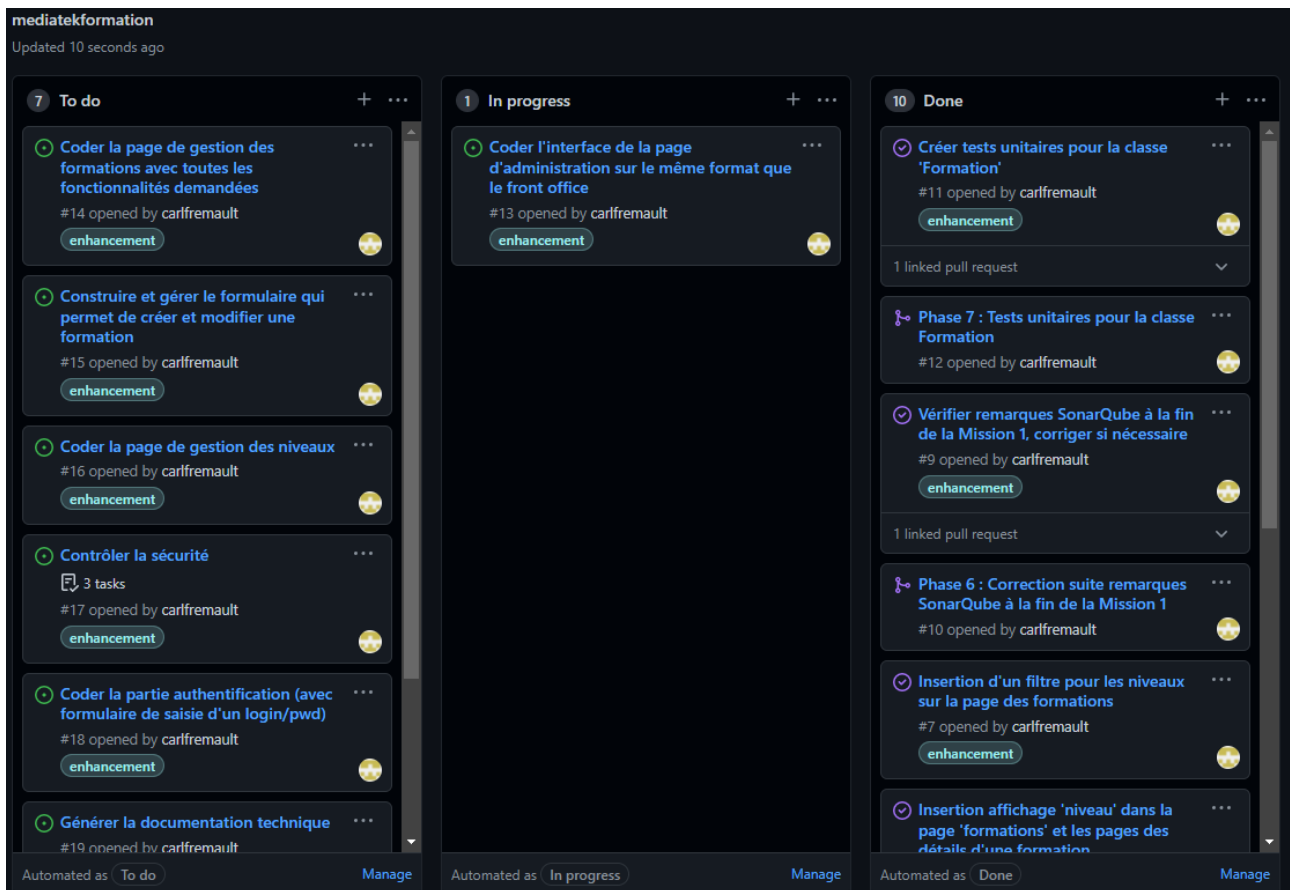


Figure 33: Storyboard en début de la deuxième mission

7.1 Coder l'interface de la page d'administration sur le même format que le front office

D'abord j'ai créé un template 'baseadmin.html.twig' sur les mêmes bases que 'basefront.html.twig'. Ensuite, un template 'admin.formationen.html.twig', qui est vide pour l'instant, et puis un contrôleur 'AdminFormationsController' qui gère la route '/admin' afin de rediriger vers les nouveaux templates créés (fonction 'index').



Figure 34: Entête de la page admin

J'ai fait un premier push comme les tâches suivantes s'annoncent assez conséquentes (comme ça j'aurai un point de repli si besoin).

```
Phase 8 : Coder l'interface de la page d'administration sur le même f...
...ormat que le front office

- templates 'baseadmin.html.twig', 'admin.formations.html.twig'
- controller 'AdminFormationsController' avec pour l'instant l'unique route '/admin'
Task #13 - Coder l'interface de la page d'administration sur le même format que le front office

main (#21)
```

Figure 35: Commit après premières modifications pour le back office

7.2 Coder la page de gestion des formations avec les fonctionnalités demandées : lister les formations, boutons pour modifier, supprimer une formation, ...

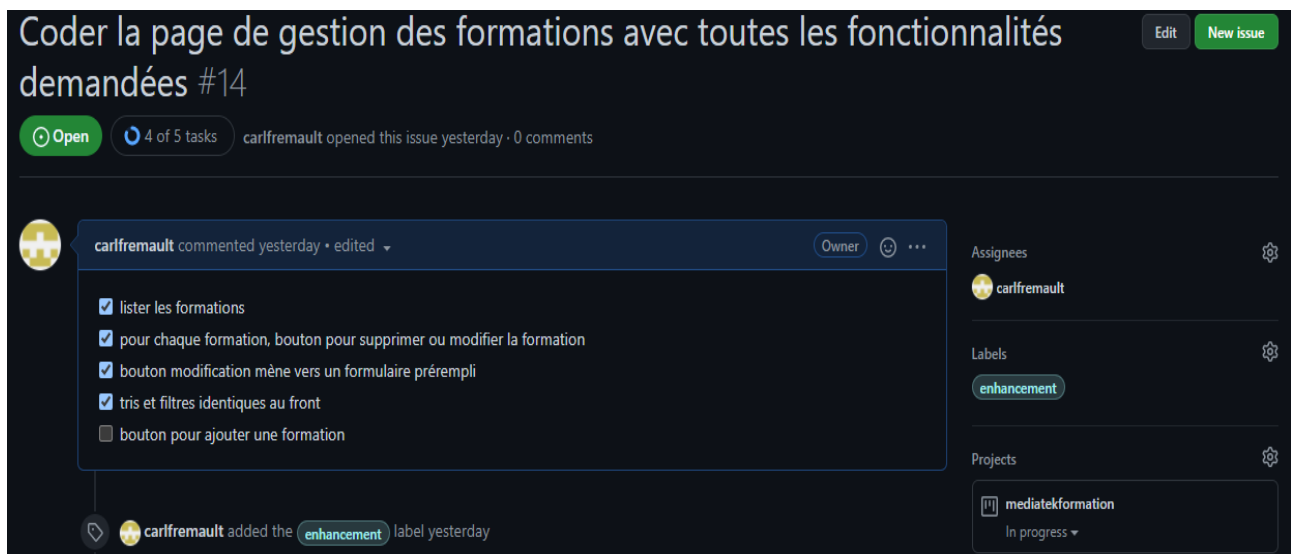


Figure 36: Kanban pour cette tâche

J'ai d'abord mis en place l'affichage de la liste des formations dans le template 'admin.formations.html.twig' (repris en grande partie depuis la page 'formations' du front) avec ajout des boutons pour éditer/supprimer. Après j'ai ajouté les fonctions pour trier/filtrer dans le contrôleur (repris depuis le contrôleur du front aussi).

titre	niveau	parution	actions
<input type="text"/> <input type="button" value="filtrer"/>	<input type="button" value="filtrer"/>	<input type="button" value="<"/> <input type="button" value=">"/>	
Eclipse n°8 : Déploiement	débutant	28/12/2020	 <input type="button" value="editer"/> <input type="button" value="supprimer"/>

Figure 37: Adaptation de la liste pour le back office

Ensuite : mise en place de la fonctionnalité 'suppression' en ajoutant la fonction (et la route) dans le contrôleur ...

```

/**
 * @Route ("/admin/suppr/{id}", name="admin.formations.suppr")
 * @param Formation $formation
 * @return Response
 */
public function suppr(Formation $formation): Response {
    $this->om->remove($formation);
    $this->om->flush();
    return $this->redirectToRoute('admin.formations');
}

```

Figure 38: Route de suppression d'une formation

... suivi de l'ajout de cette route dans le template, avec demande de confirmation de l'internaute :

```

72 | <td class="text-center">
73 |     <a href="{{ path('admin.formation.edit', {id: formation.id}) }}" class="btn btn-secondary my-1"
74 |     style="width: 100%">editer</a>
75 |     <a href="{{ path('admin.formation.suppr', {id: formation.id}) }}" class="btn btn-danger my-1"
76 |     onclick="return confirm('Etes-vous sûr de vouloir supprimer {{ formation.title }} ?');" style="width: 100%">supprimer</a>
77 | </td>

```

Figure 39: Modification de la vue pour implémenter la suppression

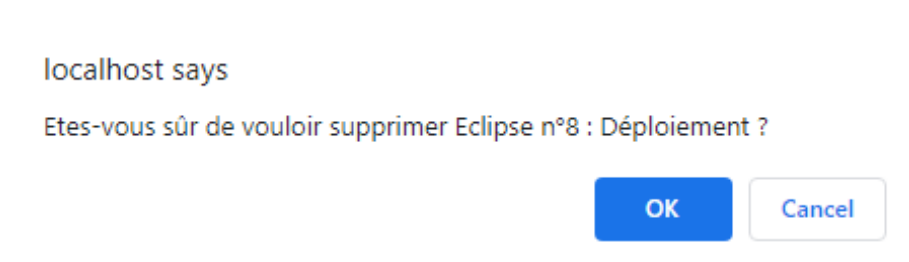


Figure 40: Demande de confirmation avant la suppression

7.3 Construire et gérer le formulaire qui permet de créer et modifier une formation

Ensuite, j'ai créé un formulaire pour pouvoir éditer/ajouter une formation. Cette manipulation crée un fichier 'FormationType' qui représente le formulaire :

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

The name of the form class (e.g. GentlePuppyType):
> FormationType

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

created: src/Form/FormationType.php

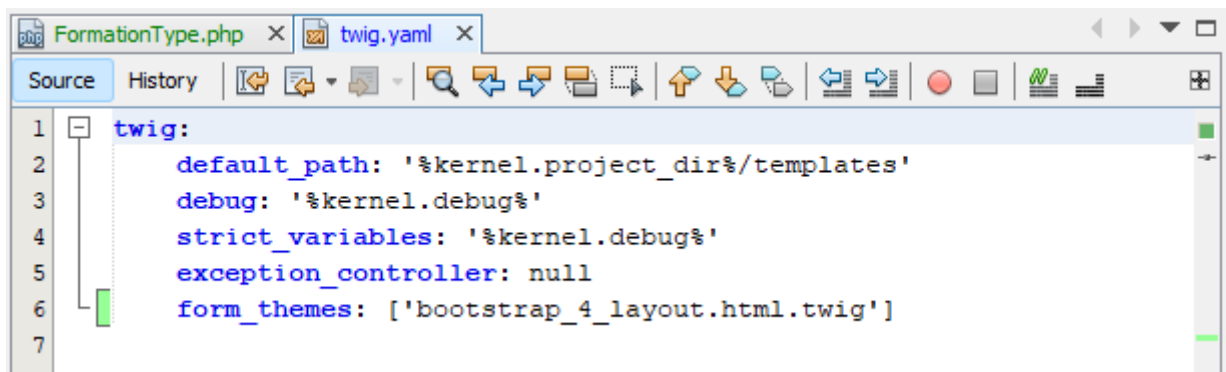
Success!

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

c:\wamp64\www\mediatekformation>
```

Figure 41: Création de formulaire avec l'aide de Symfony

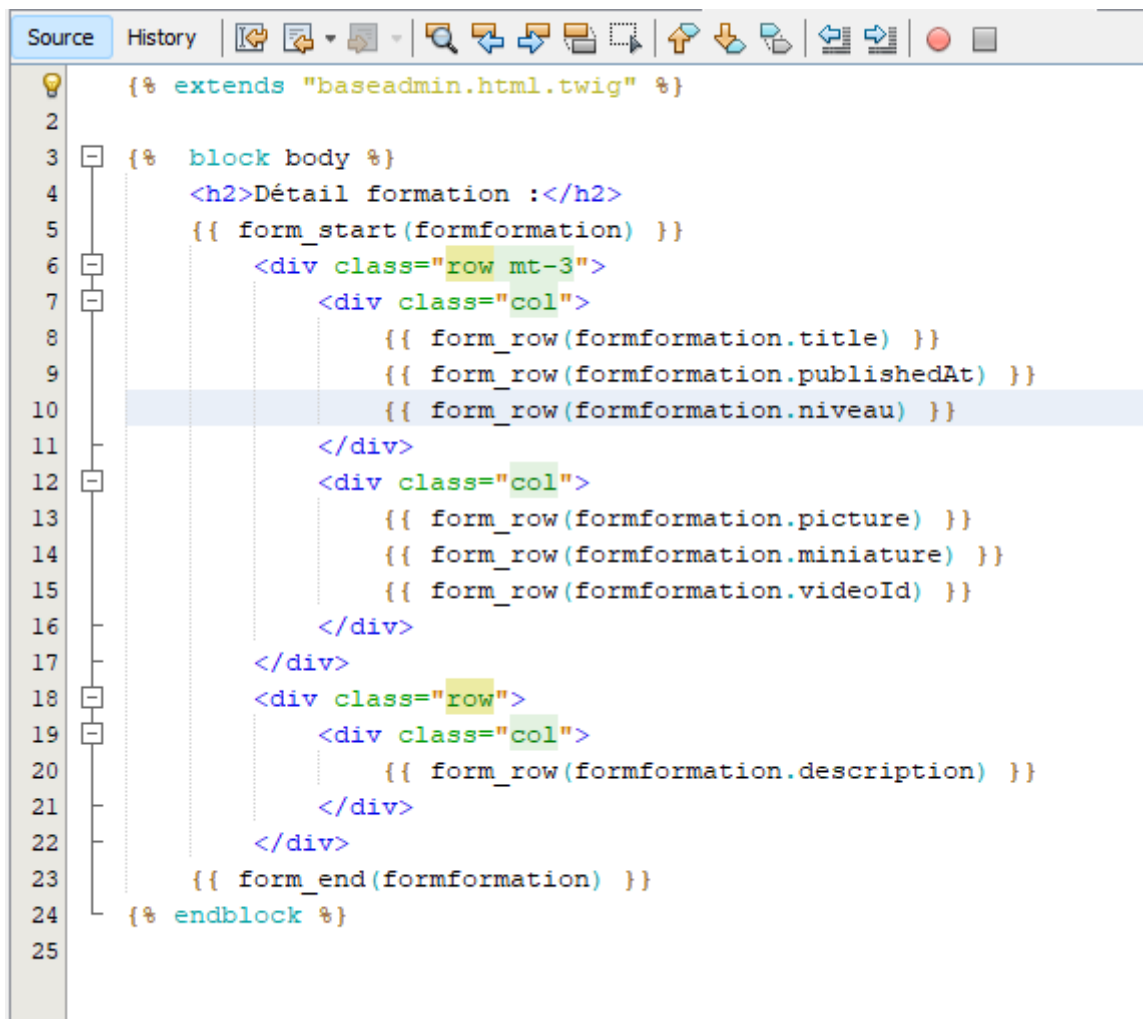
Puis j'ai précisé un thème dans le fichier de configuration de twig pour une meilleure présentation du formulaire :



```
1 twig:
2     default_path: '%kernel.project_dir%/templates'
3     debug: '%kernel.debug%'
4     strict_variables: '%kernel.debug%'
5     exception_controller: null
6     form_themes: ['bootstrap_4_layout.html.twig']
7
```

Figure 42: Configuration Twig

Par la suite j'ai créé un template pour gérer la présentation des différents champs :



```
1  {% extends "baseadmin.html.twig" %}
2
3  {% block body %}
4      <h2>Détail formation :</h2>
5      {{ form_start(formformation) }}
6          <div class="row mt-3">
7              <div class="col">
8                  {{ form_row(formformation.title) }}
9                  {{ form_row(formformation.publishedAt) }}
10                 {{ form_row(formformation.niveau) }}
11             </div>
12             <div class="col">
13                 {{ form_row(formformation.picture) }}
14                 {{ form_row(formformation.miniature) }}
15                 {{ form_row(formformation.videoId) }}
16             </div>
17         </div>
18         <div class="row">
19             <div class="col">
20                 {{ form_row(formformation.description) }}
21             </div>
22         </div>
23     {{ form_end(formformation) }}
24 {% endblock %}
25
```

Figure 43: Template pour le nouveau formulaire

Puis j'ai ajouté d'attributs nécessaires pour peaufiner la présentation et contrôler les saisies, dans le 'FormationType.php' créé auparavant.

```
16 class FormationType extends AbstractType {
17
18     public function buildForm(FormBuilderInterface $builder, array $options): void {
19         $builder
20             ->add('publishedAt', DateType::class, [
21                 'widget'=>'choice',
22                 'format' => 'dd-MMM-yyyy',
23                 'label'=>'Parution'
24             ])
25             ->add('title', TextType::class, [
26                 'required'=>true,
27                 'label'=>'Titre'
28             ])
29             ->add('description', TextareaType::class, [
30                 'attr' => array('style' => 'height: 20vh'),
31                 'required'=>false
32             ])
33             ->add('miniature', TextType::class, [
34                 'label' =>'Miniature (URL, taille 120x90 pixels)',
35                 'required'=>false
36             ])
37             ->add('picture', TextType::class, [
38                 'label'=>'Image (URL, taille maximale 640x480 pixels)',
39                 'required'=>false
40             ])
41             ->add('videoId', TextType::class, [
42                 'required'=>false
43             ])
44             ->add('niveau', EntityType::class, [
45                 'class' => Niveau::class,
46                 'choice_label' => 'libelle',
47                 'required'=>true
48             ])
49             ->add('submit', SubmitType::class)
50         ;
51     }
```


Figure 44: Définition des types

Il fallait aussi ajouter une route pour ouvrir et préremplir ce formulaire quand l'utilisateur clique sur le bouton 'éditer' :

```
128  /**
129  * @route ("/admin/edit/{id}", name="admin.formation.edit")
130  * @param Formation $formation
131  * @param Request $request
132  * @return Response
133  */
134  public function edit(Formation $formation, Request $request): Response {
135      $formFormation = $this->createForm(FormationType::class, $formation);
136      $formFormation->handleRequest($request);
137      if($formFormation->isSubmitted() && $formFormation->isValid()){
138          $this->om->flush();
139          return $this->redirectToRoute('admin.formations');
140      }
141      return $this->render("admin/admin.formation.edit.html.twig", [
142          'formation' => $formation,
143          'niveaux'=>$this->niveaux,
144          'formformation'=>$formFormation->createView()
145      ]);
146  }
147  }
```

Figure 45: Route pour la modification d'une formation

Voici le résultat pour une des formations :



MediaTek86

Des formations sur des outils numériques pour tous

Formations Niveaux

Détail formation :

Titre	Image (URL, taille maximale 640x480 pixels)
<input type="text" value="Eclipse n°8 : Déploiement"/>	<input type="text" value="https://i.ytimg.com/vi/Z4yTTXka958/sddefault.jpg"/>
Parution	Miniature (URL, taille 120x90 pixels)
<input type="text" value="22"/> <input type="text" value="Nov"/> <input type="text" value="2021"/>	<input type="text" value="https://i.ytimg.com/vi/Z4yTTXka958/default.jpg"/>
Niveau	Video id
<input type="text" value="débutant"/>	<input type="text" value="Z4yTTXka958"/>
Description	
<div>Exécution de l'application en dehors de l'IDE, en invite de commande. Création d'un fichier jar pour le déploiement de l'application. 00:20 : exécuter l'application à partir d'un invite de commandes 04:41 : créer un fichier jar auto exécutable 06:42 : exécuter un fichier jar directement 07:09 : exécuter un fichier jar dans l'invite de commande pour avoir les retours console</div>	

Enregistrer

Figure 46: Le formulaire fonctionne

Par rapport aux contraintes :

La saisie d'un titre est obligatoire: ceci est réglé en ajoutant l'attribut 'required'=>true au niveau du formulaire :

```
->add('title', TextType::class, [  
    'required'=>true,  
    'label'=>'Titre'  
])
```

Figure 47: Saisie obligatoire

Par ailleurs dans l'entité le champ est également défini de façon qu'il ne peut pas être vide (null ou 'blank' : une chaîne vide).

```

/**
 * @Assert\NotBlank
 * @ORM\Column(type="string", length=91, nullable=false)
 */
private $title;

```

Figure 48: Saisie obligatoire imposé par l'ORM

Le saisie de la date est obligatoire aussi : toutefois, comme la date est sélectionnée avec des menus déroulants (pour le jour, le mois, et l'année) elle ne peut effectivement pas être vide.

Parution

22 ▾

Nov ▾

2021 ▾

Figure 49: Champs sécurisés pour la date ...

Idem pour le niveau: il est sélectionné depuis un menu déroulant qui est rempli avec tous les éléments présents dans la table correspondante de la base de données et ne peut donc pas être vide.

Niveau

débutant ▾

Figure 50: ... et pour les niveaux

Pour vérifier et limiter les tailles des images c'était plus compliqué. Effectivement, comme les images ne sont pas hébergées sur le site, on doit aller vérifier par rapport à leur emplacement sur internet. Pour cela j'ai créé une Assertion de type 'Callback' dans l'entité Formation.

L'avantage de la démarche est que d'un coup on peut vérifier si l'URL saisie par l'utilisateur correspond bien à une image. Effectivement, si ce n'est pas le cas, la fonction 'getImageSize' génère une erreur. C'était donc nécessaire et très pratique d'appeler cette fonction dans un try/catch.

J'ai hésité par rapport à la présence des deux conditionnels 'if'. Cela ne correspond pas tout à fait à une démarche DRY (Don't Repeat Yourself). Toutefois, les deux parties sont assez différentes: les variables, les constantes, les conditions (taille unique pour les miniatures et donc comparaison '!=' avec les constantes, taille maximale pour les 'picture' et donc comparaison '>') et les messages d'erreur. Ainsi, pour extraire cette fonction il y

aura énormément de variables à gérer ce qui, à mon avis, n'améliorerait pas la lisibilité du code. J'ai donc décidé de faire une exception à la règle DRY.

```
public function validate(ExecutionContextInterface $context) {
    $miniatureUrl = $this->getMiniature();
    $miniatureWidth = 0;
    $miniatureHeight = 0;

    $pictureUrl = $this->getPicture();
    $pictureWidth = 0;
    $pictureHeight = 0;

    if ($miniatureUrl != "") {
        try {
            $miniatureUrlurl = 'http://'.$miniatureUrl;
            $info = getimagesize($miniatureUrlurl);
        } catch (\Exception $ex) {
            $context->buildViolation("Ceci n'est pas une image")
                ->atPath('miniature')
                ->addViolation();

            return;
        }

        if ($info) {
            $miniatureWidth = $info[0];
            $miniatureHeight = $info[1];
            if ($miniatureWidth != self::MINIATUREWIDTH || $miniatureHeight != self::MINIATUREHEIGHT) {
                $context->buildViolation("Les miniatures doivent être de taille " . self::MINIATUREWIDTH . "x" . self::MINIATUREHEIGHT . " pixels")
                    ->atPath('miniature')
                    ->addViolation();
            }
        } else {
            $context->buildViolation("Ceci n'est pas une image")
                ->atPath('miniature')
                ->addViolation();
        }
    }
}
```

Figure 51: Validation des images : exemple miniatures

De plus, j'ai dû implémenter la génération d'une 'buildViolation' deux fois. En fonction des URL saisies, même si une URL ne représente pas une image, la fonction getImageSize ne génère pas toujours une exception. J'avoue ne pas avoir bien compris pourquoi. Toutefois, de cette façon la solution est plus 'imperméable'.

Cette fonction vérifie tout d'abord si une URL insérée correspond bien à une image. Si ce n'est pas le cas l'utilisateur est prévenu :

Miniature (URL, taille 120x90 pixels)

ERROR Ceci n'est pas une image

<https://i.ytimg.com/vi/Z4yTTXka9>



Figure 52: Sécurisation des champs

Ensuite, la taille est vérifiée, l'utilisateur est prévenu si elle ne correspond pas au requis :

Image (URL, taille maximale 640x480 pixels)

ERROR La taille des images ne doit pas dépasser 640x480 pixels

https://upload.wikimedia.org/wikipedia/commons/thumb/1/10/Flag_c ✖

Miniature (URL, taille 120x90 pixels)

ERROR Les miniatures doivent être de taille 120x90 pixels

<https://i.ytimg.com/vi/Z4yTTXka958/sddefault.jpg> ✖

Figure 53: Restrictions sur les tailles

Finalement, pour respecter les contraintes de la base de données, il fallait également s'assurer que les chaînes des url (miniature et picture), du titre et de l'identifiant de la vidéo ne dépassent pas les longueurs définies.

	Browse	Structure	SQL	Search	Insert	Export	Import	Privileges	Operations	Triggers
	Table structure	Relation view								
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action	
<input type="checkbox"/> 1	id	int(11)			No	None		AUTO_INCREMENT	Change	Drop More
<input type="checkbox"/> 2	published_at	datetime			Yes	NULL			Change	Drop More
<input type="checkbox"/> 3	title	varchar(91)	utf8mb4_unicode_ci		Yes	NULL			Change	Drop More
<input type="checkbox"/> 4	description	longtext	utf8mb4_unicode_ci		Yes	NULL			Change	Drop More
<input type="checkbox"/> 5	miniature	varchar(46)	utf8mb4_unicode_ci		Yes	NULL			Change	Drop More
<input type="checkbox"/> 6	picture	varchar(48)	utf8mb4_unicode_ci		Yes	NULL			Change	Drop More
<input type="checkbox"/> 7	video_id	varchar(11)	utf8mb4_unicode_ci		Yes	NULL			Change	Drop More
<input type="checkbox"/> 8	niveau_id	int(11)			Yes	NULL			Change	Drop More

Figure 54: Structure de la table des formations

Ceci demandait quelques modifications dans l'entité Formation :

```
/**
 * @Assert\NotBlank
 * @Assert\Length(max=91,maxMessage = "Le titre ne peut pas excéder {{ limit }} caractères")
 * @ORM\Column(type="string", length=91, nullable=false)
 */
private $title;

/**
 * @ORM\Column(type="text", nullable=true)
 */
private $description;

/**
 * @Assert\Length(max=46,maxMessage = "L'url de la miniature ne peut pas excéder {{ limit }} caractères")
 * @ORM\Column(type="string", length=46, nullable=true)
 */
private $miniature;

/**
 * @Assert\Length(max=48,maxMessage = "L'url de l'image ne peut pas excéder {{ limit }} caractères")
 * @ORM\Column(type="string", length=48, nullable=true)
 */
private $picture;

/**
 * @Assert\Length(max=11,maxMessage = "L'identifiant de la vidéo ne peut pas excéder {{ limit }} caractères")
 * @ORM\Column(type="string", length=11, nullable=true)
 */
private $videoId;
```

Figure 55: Sécurisation des champs

Ainsi l'utilisateur est averti si la valeur saisie n'est pas valide :

Video id

ERROR L'identifiant de la vidéo ne peut pas excéder 11 caractères

unIdentifiantQuiEstTropLong ✖

Figure 56: Contrôle de longueur d'identifiant

Maintenant que le formulaire était en place, il était possible de gérer l'ajout de nouvelles formations, comme l'ajout va utiliser le même formulaire. Ainsi, j'ai commencé par extraire le contenu du template 'admin.formation.edit.html.twig' et je l'ai collé dans un nouveau fichier en racine '_admin.formation.form.html.twig'.

Le template pour éditer une formation (admin.formation.edit.html.twig) devait inclure ce fichier :

```

1  {% extends "baseadmin.html.twig" %}
2
3  {% block body %}
4      <h2 class="mt-3">Détail formation :</h2>
5      {{ include('_admin.formation.form.html.twig') }}
6  {% endblock %}

```

Figure 57: Inclusion du formulaire dans la vue

Ensuite j'ai fait une copie de 'admin.formation.edit.html.twig' vers un nouveau fichier 'admin.formation.ajout.html.twig' en modifiant le titre de la page.

Maintenant que les templates étaient en place j'ai créé une nouvelle fonction dans le contrôleur AdminFormationsController :

```

/**
 * @route ("/admin/ajout", name="admin.formation.ajout")
 * @param Request $request
 * @return Response
 */
public function ajout(Request $request): Response {
    $formation = new Formation();
    $formFormation = $this->createForm(FormationType::class, $formation);
    $formFormation->handleRequest($request);
    if ($formFormation->isSubmitted() && $formFormation->isValid()) {
        $this->om->persist($formation);
        $this->om->flush();
        return $this->redirectToRoute('admin.formations');
    }
    return $this->render("admin/admin.formation.ajout.html.twig", [
        'formation' => $formation,
        'niveaux' => $this->niveaux,
        'formformation' => $formFormation->createView()
    ]);
}

```

Figure 58: Route pour ajouter une formation

Puis j'ai rajouté un bouton dans le template 'admin.formations.html.twig' qui appelle cette fonction :



MediaTek86

Des formations sur des outils numériques pour tous

Formations Niveaux

ajouter une nouvelle formation


titre	niveau	parution	actions
<input type="text"/> <input type="button" value="filtrer"/>	<input type="button" value="filtrer"/>	<input type="button" value=""/> <input type="button" value=""/>	
Eclipse n°8 : Déploiement	débutant	22/11/2021	 <input type="button" value="editer"/> <input type="button" value="supprimer"/>
Eclipse n°7 : Tests unitaires	confirmé	28/12/2020	 <input type="button" value="editer"/> <input type="button" value="supprimer"/>

Figure 59: Vue avec bouton d'ajout de formation

Du coup deux des 'issues' du tableau Kanban étaient faites (j'ai choisi de gérer les deux en même temps car la gestion d'ajout d'une formation était étroitement lié avec la gestion de modification). Ainsi j'ai fait un commit et push avant de continuer avec cette mission.

Phase 9 : Intégration de la liste des formations sur la page admin, a...

avec les mêmes options (tri, filtrage, recherche) que sur la page front, ainsi que les possibilités de modifier, supprimer et ajouter des formations

- contrôleur AdminFormationsController: ajout des fonctions sort, filter, findAllContain
- contrôleur AdminFormationsController: ajout des fonctions suppr, edit, ajout
- modification de l'entité Formation pour permettre de vérifier la taille d'une image dont l'url est saisie
- modification de l'entité Formation pour contrôler les saisies (titre ne peut pas être vide, longueur des chaînes)
- modification du fichier de configuration twig.yaml pour utiliser le thème bootstrap_4_layout.html.twig pour les formulaires
- création de formulaire FormationType pour pouvoir modifier ou ajouter une formations
- création d'un template _admin.formation.form.html.twig pour intégrer le formulaire FormationType
- création des templates admin.formation.ajout.html.twig et admin.formation.edit.html.twig qui intègrent le template _admin.formation.form.html.twig
- modification du template admin.formations.html.twig pour ajouter un bouton qui permet d'ajouter une nouvelle formation

Task #15 - Construire et gérer le formulaire qui permet de créer et modifier une formation

Figure 60: Commit après intégration de la liste et fonctionnalités d'ajout et modification

7.4 Coder la page de gestion des niveaux

D'abord j'ai créé un nouveau contrôleur 'AdminNiveauxController', avec création des routes, récupération et adaptation du constructeur et certaines des fonctions présentes dans AdminFormationController :

Une fonction 'index' pour lister tous les niveaux :

```
/**
 * @Route("/admin/niveaux", name="admin.niveaux")
 * @return Response
 */
public function index(): Response {
    $niveaux = $this->niveauRepository->findAll();
    return $this->render(self::PAGEADMINNIVEAUX, [
        'niveaux' => $niveaux
    ]);
}
```

Figure 61: Route pour afficher les niveaux

Une fonction 'ajout' pour ajouter un niveau :

```
/**
 * @Route("/admin/niveau/ajout", name="admin.niveau.ajout")
 * @param Request $request
 * @return Response
 */
public function ajout(Request $request) : Response {
    $libelleNiveau = $request->get("libelle");
    $niveau = new Niveau();
    $niveau->setLibelle($libelleNiveau);
    $this->om->persist($niveau);
    $this->om->flush();
    return $this->redirectToRoute('admin.niveaux');
}
```

Figure 62: Ajout d'un niveau

Et enfin une fonction 'suppr' pour supprimer un niveau. Cette fonction vérifie d'abord si le niveau qu'on souhaite supprimer n'est pas utilisé pour une des formations. Si ce n'est pas le cas le niveau est supprimé. S'il est utilisé le niveau n'est pas supprimé et l'utilisateur est averti. Pour cela j'ai utilisé la fonctionnalité des messages 'flash' de Symfony. Le principe est qu'un message peut être ajouté à une session de navigation. Un message flash n'est affiché qu'une seule fois, ensuite il est supprimé de la mémoire.


```

/**
 * @Route ("/admin/niveau/suppr/{id}", name="admin.niveau.suppr")
 * @param Niveau $niveau
 * @return Response
 */
public function suppr(Niveau $niveau, Request $request): Response {
    $formations = $this->formationRepository->findByEqualValue('niveau', $niveau->getId());
    if (count($formations) == 0) {
        $this->om->remove($niveau);
        $this->om->flush();
    } else {
        $this->addFlash(
            'notice',
            'Impossible de supprimer ce niveau. Il est utilisé pour (au moins) une des formations.'
        );
    }
    return $this->redirectToRoute('admin.niveaux');
}

```

Figure 63: Insertion de message flash pour avertir l'utilisateur

Après génération du message il faut incorporer la possibilité de l'afficher (au cas où un message 'flash' existe) dans le template voulu (admin.niveaux.html.twig) :

```

{% for message in app.flashes('notice') %}
    <div class="alert alert-danger" role="alert">
        {{ message }}
    </div>
{% endfor %}

```

Figure 64: Affichage du message flash

Voici le résultat, après avoir essayé de supprimer un des niveaux utilisés :

[Formations](#)
[Niveaux](#)

Impossible de supprimer ce niveau. Il est utilisé pour (au moins) une des formations.

libellé	actions
débutant	<input type="button" value="supprimer"/>
confirmé	<input type="button" value="supprimer"/>
expert	<input type="button" value="supprimer"/>

Figure 65: Message affiché dans la vue

La page de gestion des niveaux est prête, je peux faire un autre commit et push.

Phase 10 : Coder la page de gestion des niveaux :

- ajout de contrôleur AdminNiveauxController.php avec constructeur et les fonctions et routes index, ajout, suppr
 - ajout de template admin.niveaux.html.twig
 - la fonction suppr vérifie si un niveau n'est pas utilisé avant de le supprimer
 - affichage de message flash (incorporé dans le template) si le niveau ne peut pas être supprimé
- + correction oubli: affichage du niveau dans la partie front, sur la page 'accueil' pour les deux formations les plus récentes
- Task #16 - Coder la page de gestion des niveaux

Figure 66: Commit après implémentation de la gestion de niveaux

7.5 Contrôler la sécurité

Afin de m'assurer que l'application ne présentait pas de vulnérabilités j'ai fait un tour sur tous les endroits où l'utilisateur peut saisir des données, voire intercepter et modifier des requêtes faites à la base de données.

7.5.1 Partie Front Office :

Aucune saisie n'est présente sur la page 'accueil' ni sur les pages qui détaillent les formations

Pour les liens (vers les détails d'une formation ou vers la page qui liste les formations) toutes les routes sont prédéfinies, et toutes les demandes à la BDD sont des requêtes paramétrées avec l'ORM Doctrine qui assure une protection avancée contre les injections SQL.

Sur la page qui présente la liste des formations nous avons tout d'abord le champ texte où l'utilisateur peut faire une recherche par rapport à une chaîne de caractères.

Le 'submit' du formulaire utilise bien un token CSRF

```
<form class="form-inline mt-1" method="POST" action="{{ path('formations.findallcontain', {champ:'title'}) }}">
  <div class="form-group mr-1 mb-2">
    <input type="text" class="sm" name="recherche">
  </div>
  <input type="hidden" name="_token" value="{{ csrf_token('filtre_title') }}">
  <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
</form>
```

Figure 67: Implémentation de token CSRF

Le contrôleur vérifie le token avant de transmettre la demande au repository :

```
/**
 * @Route("/admin/recherche/{champ}", name="admin.formationen.findallcontain")
 * @param type $champ
 * @param Request $request
 * @return Response
 */
public function findAllContain($champ, Request $request): Response {
    if ($this->isCsrfTokenValid('filtre_' . $champ, $request->get('_token')))) {
        $valeur = htmlentities($request->get("recherche"));
        $formations = $this->formationRepository->findByContainValue($champ, $valeur);
        return $this->render(self::PAGEADMININFORMATIONS, [
            'formations' => $formations,
            'niveaux' => $this->niveaux
        ]);
    }
    return $this->redirectToRoute("admin");
}
```

Figure 68: Exemple de vérification du token CSRF

J'ai également passé le paramètre 'valeur' dans la fonction 'htmlentities' qui 'nettoie' la chaîne pour convertir tous les caractères éligibles en entités HTML.

Et enfin, la requête SQL est protégée en utilisant une requête paramétrée :

```
/**
 * Enregistrements dont un champ contient une valeur
 * ou tous les enregistrements si la valeur est vide
 * @param type $champ
 * @param type $valeur
 * @return Formation[]
 */
public function findByContainValue($champ, $valeur): array {
    if ($valeur == "") {
        return $this->createQueryBuilder('f')
            ->orderBy('f.' . $champ, 'ASC')
            ->getQuery()
            ->getResult();
    } else {
        return $this->createQueryBuilder('f')
            ->where('f.' . $champ . ' LIKE :valeur')
            ->setParameter('valeur', $valeur)
            ->orderBy('f.publishedAt', 'DESC')
            ->setParameter('valeur', '%' . $valeur . '%')
            ->getQuery()
            ->getResult();
    }
}
```

Figure 69: Requête paramétrée

Pour le filtrage il n'y avait pas de risque d'injection comme j'ai utilisé un menu déroulant avec des valeurs fixes (uniquement les niveaux présents dans la base de données). L'utilisateur ne peut donc pas saisir une valeur lui-même.

Pour éviter une manipulation en forgeant une requête j'ai à nouveau passé le paramètre 'valeur' dans la fonction 'htmlentities'.

```
public function filter($champ, $valeur = null): Response {
    $valeur = htmlentities($valeur);
    $formations = $this->formationRepository->findByEqualValue($champ, $valeur);
    return $this->render(self::PAGEADMININFORMATIONS, [
        'formations' => $formations,
        'niveaux' => $this->niveaux
    ]);
}
```

Figure 70: Assainissement des entrées utilisateur

Puis la requête à la BDD est paramétrée :

```
/**
 * Enregistrements dont un champ est égal à une valeur
 * ou tous les enregistrements si la valeur est vide
 * @param type $champ
 * @param type $valeur
 * @return Formation[]
 */
public function findByEqualValue($champ, $valeur): array {
    if ($valeur == "") {
        return $this->createQueryBuilder('f')
            ->orderBy('f.publishedAt', 'DESC')
            ->getQuery()
            ->getResult();
    } else {
        return $this->createQueryBuilder('f')
            ->where('f.' . $champ . ' = :valeur')
            ->setParameter('valeur', $valeur)
            ->orderBy('f.publishedAt', 'DESC')
            ->getQuery()
            ->getResult();
    }
}
```

Figure 71: Requête paramétrée

Il en est de même pour les routes qui gèrent le tri (pour le titre et pour la date de parution).

7.5.2 Partie Back office :

Pour le backoffice, les fonctions et templates utilisées pour l'affichage des formations sont faites de façon identique et présentent donc les mêmes protections. Deux possibles failles se sont introduites cependant: le formulaire pour modifier ou ajouter une nouvelle formation, et le champ texte pour ajouter un niveau.

Formulaire pour ajouter / modifier une formation

L'utilisation de formulaires créés automatiquement par Symfony assure une première protection. L'utilisation de token CSRF et de cookies de session est incorporé par défaut.

Pour l'insertion de nouveaux tuples dans la base de données la sécurité est assurée par l'utilisation de l'ORM Doctrine. Les différents champs sont typés ce qui limite la possibilité d'injections, et lors de la soumission d'un formulaire sa validité est contrôlée avant l'enregistrement.

```
public function ajout(Request $request): Response {
    $formation = new Formation();
    $formFormation = $this->createForm(FormationType::class, $formation);
    $formFormation->handleRequest($request);
    if ($formFormation->isSubmitted() && $formFormation->isValid()) {
        $this->om->persist($formation);
        $this->om->flush();
        return $this->redirectToRoute('admin.formations');
    }
    return $this->render("admin/admin.formation.ajout.html.twig", [
        'formation' => $formation,
        'niveaux' => $this->niveaux,
        'formformation' => $formFormation->createView()
    ]);
}
```

Figure 72: Contrôle de validité du formulaire soumis

En plus, non seulement le formulaire prend en compte le typage des différentes propriétés de l'entité correspondante, pour une double sécurité j'ai spécifié les types des champs là aussi.

```

public function buildForm(FormBuilderInterface $builder, array $options): void {
    $builder
        ->add('publishedAt', DateType::class, [
            'widget'=>'choice',
            'format' => 'dd-MMM-yyyy',
            'label'=>'Parution'
        ])
        ->add('title', TextType::class, [
            'required'=>true,
            'label'=>'Titre'
        ])
        ->add('description', TextareaType::class, [
            'attr' => array('style' => 'height: 20vh'),
            'required'=>false
        ])
        ->add('miniature', TextType::class, [
            'label' =>'Miniature (URL, taille 120x90 pixels)',
            'required'=>false
        ])
        ->add('picture', TextType::class, [
            'label'=>'Image (URL, taille maximale 640x480 pixels)',
            'required'=>false
        ])
        ->add('videoId', TextType::class, [
            'required'=>false
        ])
        ->add('niveau', EntityType::class, [
            'class' => Niveau::class,
            'choice_label' => 'libelle',
            'required'=>true
        ])
        ->add('Enregistrer', SubmitType::class)
    ;
}

```

Figure 73: Champs typés

En ce qui concerne les liens correspondant aux images, comme évoqué auparavant ceux-ci sont contrôlés avant soumission pour vérifier s'il s'agit bien d'une image.

Miniature (URL, taille 120x90 pixels)

ERROR Ceci n'est pas une image

✖

Figure 74: Contrôle si les liens saisis correspondent bien à une image

Champ texte pour ajouter un niveau

Côté base de données, l'insertion de nouveaux niveaux est protégé suite à l'utilisation de Doctrine, avec le typage des champs.

Pour le formulaire de soumission cependant, celui-ci a été créé manuellement et ne bénéficie donc pas des protections mises en place par les formulaires Symfony. Il convient donc de rajouter l'utilisation d'un token au niveau du template :

```
<form class="form-inline mt-3" method="POST" action="{{ path('admin.niveau.ajout', {champ: 'niveau'}) }}">
    <div class="form-group mr-1 mb-2">
        <input type="text" class="sm" name="libelle">
    </div>
    <input type="hidden" name="_token" value="{{ csrf_token('ajout_niveau') }}">
    <button type="submit" class="btn btn-primary mb-2 btn-sm">ajouter un niveau</button>
</form>
```

Figure 75: CSRF token dans un champ 'hidden'

Ensuite le contrôleur doit vérifier le token avant de soumettre l'enregistrement :

```
/**
 * @Route("/admin/niveau/ajout/{champ}", name="admin.niveau.ajout")
 * @param type $champ
 * @param Request $request
 * @return Response
 */
public function ajout($champ, Request $request) : Response {
    if ($this->isCsrfTokenValid('ajout_'.$champ, $request->get('_token'))) {
        $libelleNiveau = htmlentities($request->get("libelle"));
        $niveau = new Niveau();
        $niveau->setLibelle($libelleNiveau);
        $this->om->persist($niveau);
        $this->om->flush();
    }
    return $this->redirectToRoute('admin.niveaux');
}
```

Figure 76: Contrôle du token lors d'un ajout de niveau

La tâche étant finie et la sécurité assurée, j'ai fait un commit et push.

mediatekformation - dev

Commit Message:

Phase 11 : Contrôler la sécurité
 - vérification de la mise en place de typage de champs et propriétés des entités : Formation Niveau

Author: Carl Fremault <carlfremault@yahoo.com> Commiter: Carl Fremault <carlfremault@yahoo.com>

☐ Amend Last Commit

Files to Commit:

...	File	Status	Commit Action	Repository Path ▲
<input checked="" type="checkbox"/>	AdminFormationsController.	-/Modified	Commit	...er\admin\AdminFormationsController.php
<input checked="" type="checkbox"/>	AdminNiveauxController.php	-/Modified	Commit	...roller\admin\AdminNiveauxController.php

By right-clicking on a row you may specify some additional Actions.

Update Task

Task Repository: carlfremault/mediatekformation

Task: 17 - Contrôler la sécurité

(Choose a recent task from the list or type a summary or ID)

☒ Resolve as FIXED ☐ After Commit

☒ Add details to commit message and task (modify...) ☒ After Push

Figure 77: Commit avec mise à jour de tâche de suivi

```

Phase 11: Contrôler la sécurité ...
- vérification de la mise en place de typage de champs et propriétés des entités : Formation, Niveau
- vérification du typage des champs du formulaire FormationType
- vérification de la mise en place de token CSRF où nécessaire : formulaire de filtrage sur le titre des formations
- ajout de gestion de token CSRF pour la fonctionnalité d'ajout d'un niveau (template et contrôleur : fonction 'ajout')
- utilisation de la fonction htmlentities() pour nettoyer les chaînes saisies : fonctions filtrer, findAllContain du contrôleur
AdminFormationsController, fonction ajout du contrôleur AdminNiveauxController

Task #17 - Contrôler la sécurité
carlfremault committed 6 minutes ago
  
```

Figure 78: Commit de gestion de sécurisation

7.6 Tests unitaires

Il était demandé d'ajouter un test unitaire pour vérifier le bon fonctionnement de la méthode qui retourne la date de parution en format string. J'ai été prévoyant cependant, ce test a déjà été intégré lors de la première mission, pendant que je mettais en place le test pour la fonction que j'avais créé pour récupérer le libellé d'un niveau.

7.7 Accès à la partie 'admin' avec authentification

Initialement j'avais implémenté l'authentification avec les fonctionnalités prévues par le framework Symfony. Il s'agit du commit 'Phase 12". Toutefois, entre-temps nos cours du bloc 3 "Cybersécurité" ont traité le sujet du "Single sign-on" avec utilisation de la solution OAuth Keycloak. Ainsi je suis revenu sur cette partie pour utiliser cette technologie intéressante.

J'ai commencé par créer une machine virtuelle sur Microsoft Azure puis j'ai installé Keycloak Server dessus.

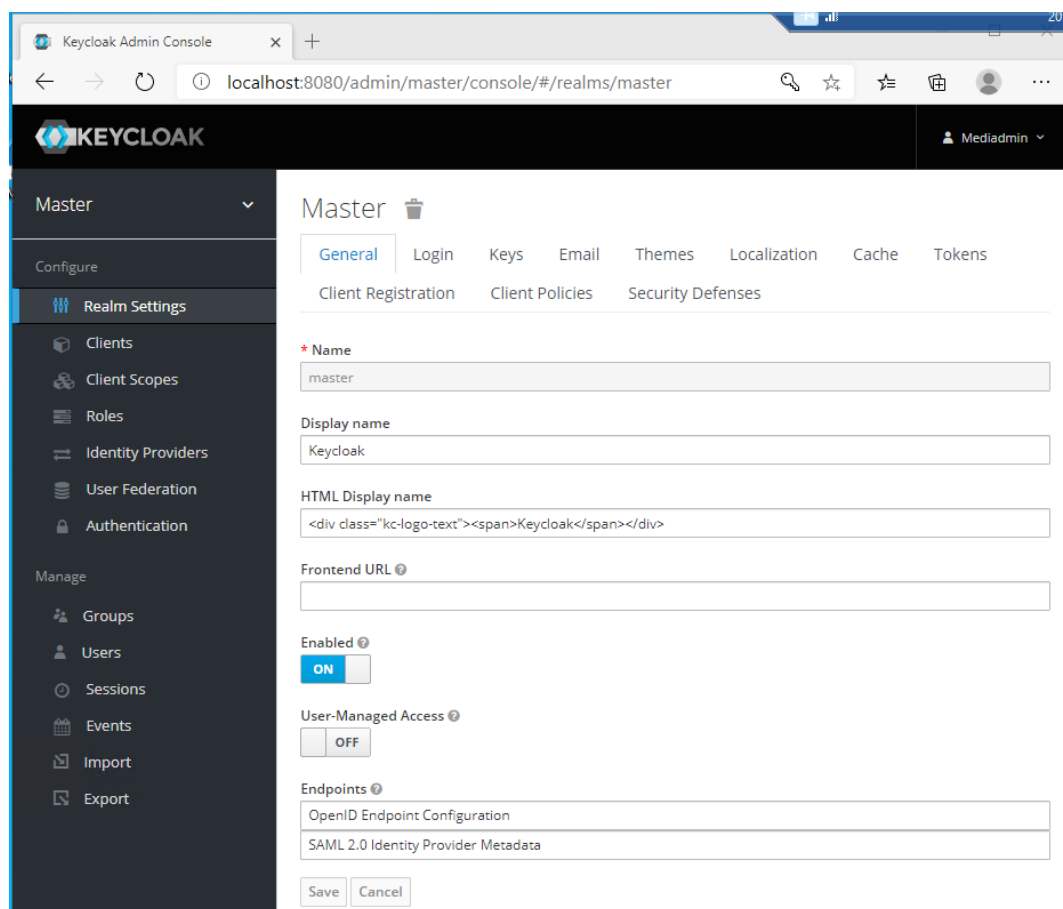


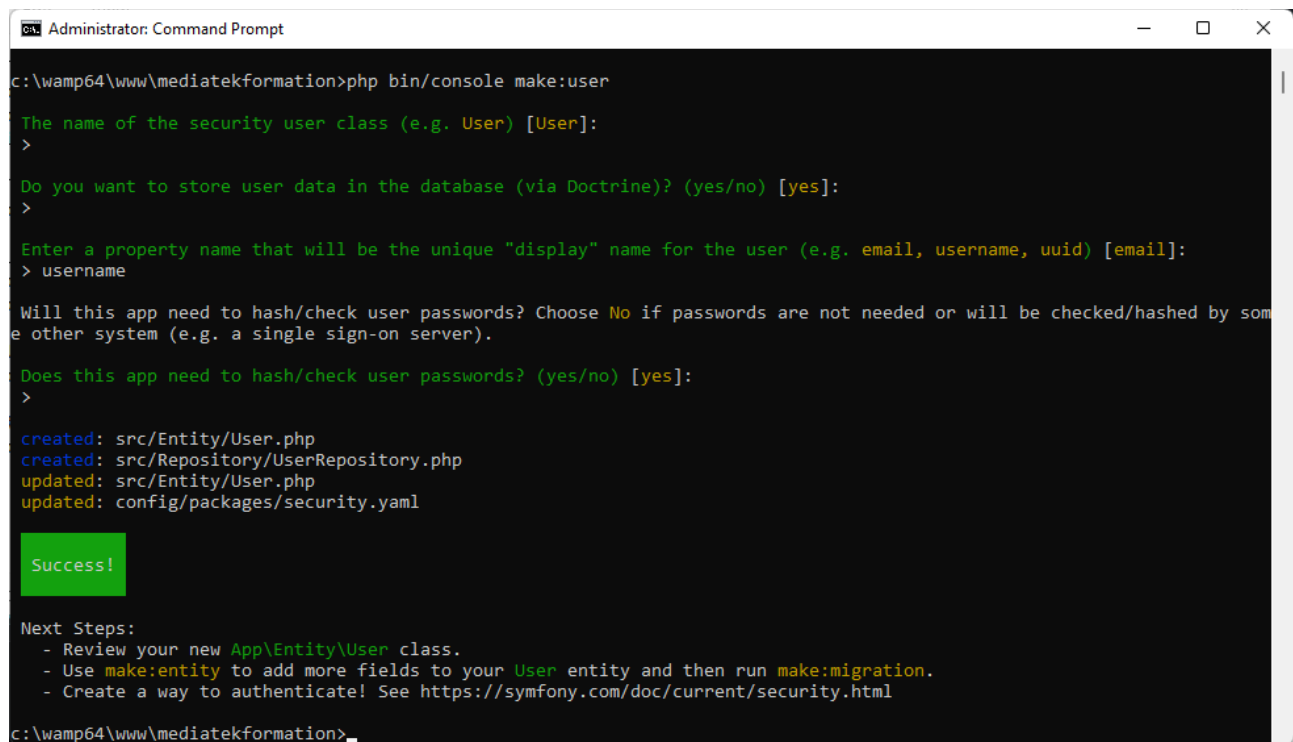
Figure 79: Configuration du serveur Keycloak

Une fois le serveur en route j'ai procédé à faire les modifications nécessaires afin de l'exploiter dans l'application. J'ai commencé par l'insertion des données Keycloak dans le fichier de configuration ".env" :

```
KEYCLOAK_SECRET=Dn8uSN05jikrim0lTaJrrC3y9cOJC7aC
KEYCLOAK_CLIENTID=mediatekformations
KEYCLOAK_APP_URL=https://https://mediatekkc.francecentral.cloudapp.azure.com/auth
```

Figure 80: Ajout de paramètres dans le fichier '.env'

Ensuite j'ai créé une nouvelle entité 'user'. Comme c'est un nom d'entité 'réservé' Symfony comprend tout de suite qu'il s'agit d'une entité 'utilisateur' et propose de choisir quel champ sera utilisé comme identifiant (j'ai choisi 'email', choix par défaut).



```
Administrator: Command Prompt
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 some
other system (e.g. a single sign-on server).

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

c:\wamp64\www\mediatekformation>
```

Figure 81: Création d'entité 'User'

J'ai également créé une propriété 'keycloakId' dans cette table, afin de pouvoir mémoriser l'identifiant Keycloak. Il n'y aura pas besoin de mémoriser le mot de passe de l'utilisateur comme cette partie est gérée par Keycloak.

```

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

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

Field length [255]:
>

Can this field be null in the database (nullable) (yes/no) [no]:
> yes

updated: src/Entity/User.php

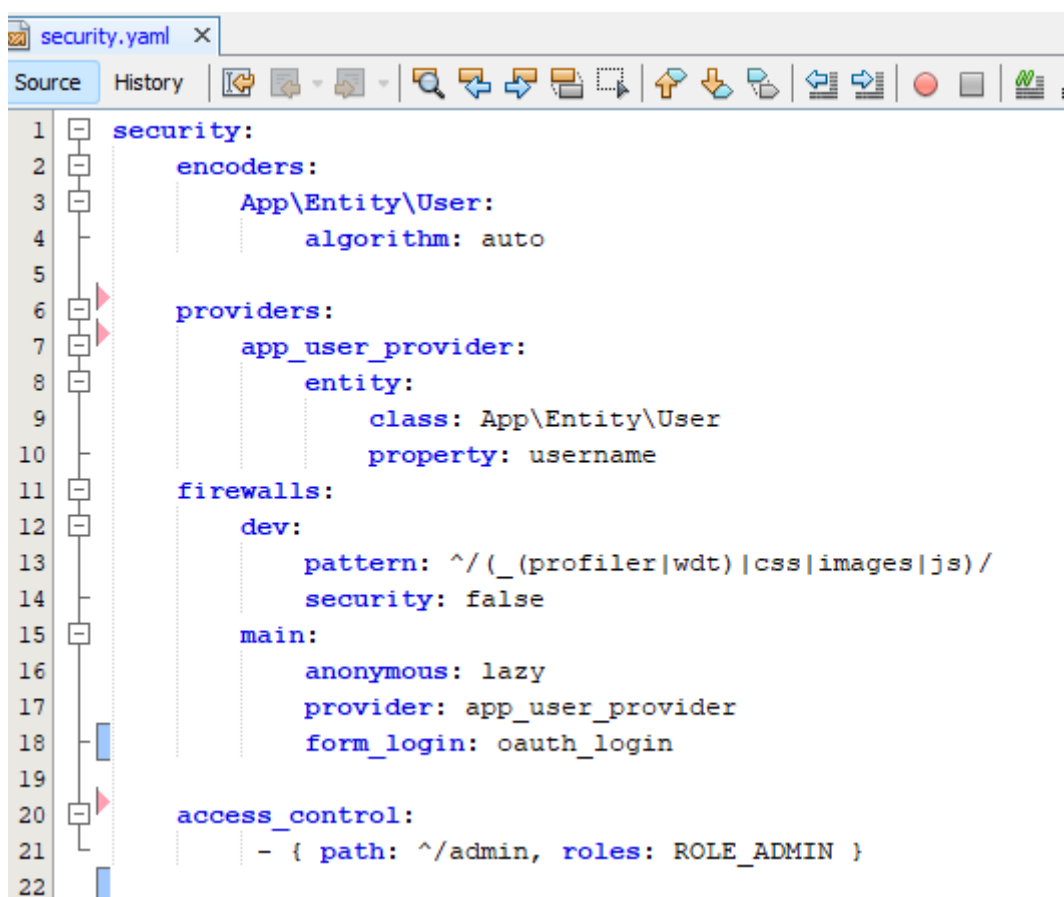
Add another property? Enter the property name (or press <return> to stop adding fields):
>

Success!

```

Figure 82: Ajout de champ dans l'Entity

Par la suite j'ai installé deux packages nécessaires pour la communication entre l'application et le serveur Keycloak, puis j'ai paramétré les fichiers de configuration venus avec, ainsi que le firewall, dans le fichier "security.yaml" :



```

1 security:
2   encoders:
3     App\Entity\User:
4       algorithm: auto
5
6   providers:
7     app_user_provider:
8       entity:
9         class: App\Entity\User
10        property: username
11
12   firewalls:
13     dev:
14       pattern: ^/(_(profiler|wdt)|css|images|js)/
15       security: false
16
17   main:
18     anonymous: lazy
19     provider: app_user_provider
20     form_login: oauth_login
21
22   access_control:
23     - { path: ^/admin, roles: ROLE_ADMIN }

```

Figure 83: Configuration du firewall

Après il fallait générer le contrôleur pour la route qui s'occupe de l'authentification, ainsi que la classe 'KeycloakAuthenticator.php' qui va s'occuper du processus d'authentification. Un utilisateur connecté va être enregistré dans la base de données afin de faciliter sa déconnexion ultérieure.

```
public function getUser($credentials, UserProviderInterface $userProvider) {
    $keycloakUser = $this->getKeycloakClient()->fetchUserFromToken($credentials);

    // Le user existe et s'est déjà connecté avec Keycloak
    $existingUser = $this
        ->em
        ->getRepository(User::class)
        ->findOneBy(['keycloakId' => $keycloakUser->getId()]);
    if($existingUser) {
        return $existingUser;
    }

    // Le user existe mais ne s'est pas encore connecté avec Keycloak
    $email = $keycloakUser->getEmail();
    $userInDatabase = $this->em->getRepository(User::class)->findOneBy(['email' => $email]);
    if($userInDatabase) {
        $userInDatabase->setKeycloakId($keycloakUser->getId());
        $this->em->persist($userInDatabase);
        $this->em->flush();
        return $userInDatabase;
    }

    // Le user n'existe pas encore dans la bdd, il est ajouté
    $user = new User();
    $user->setKeycloakId($keycloakUser->getId());
    $user->setEmail($keycloakUser->getEmail());
    $user->setPassword("");
    $user->setRoles(['ROLE_ADMIN']);
    $this->em->persist($user);
    $this->em->flush();
    return $user;
}
```

Figure 84: Recherche et/ou enregistrement de l'utilisateur dans la base de données

Il ne reste plus qu'à gérer la déconnexion, dans le template 'base.html.twig'. De cette façon il sera visible partout dans le back office quand un utilisateur est connecté :

```

<body>
  <div class="container text-right">
    {% if app.user %}
      <a href="{{ path('app_logout') }}" class="btn btn-sm btn-primary mt-1">déconnexion</a>
    {% endif %}
  </div>
  <!-- entête -->
  {% block top %}{% endblock %}
  <!-- corps -->
  <div class="container">
    {% block body %}{% endblock %}
  </div>

```

Figure 85: Le bouton de déconnexion ...



MediaTek86

Des formations sur des outils numériques pour tous

Formations Niveaux

ajouter une nouvelle formation

déconnexion

Figure 86: ... apparaît lorsqu'on est connecté

La route 'logout' devait également être créée dans le contrôleur. La fonction est vide car les fonctionnalités sont prises en charge par le firewall (dans le fichier 'security.yaml') :

```

/**
 * Route pour la déconnexion du back office. Fonction vide car le firewall s'occupe des fonctionnalités
 * @Route("/logout", name="logout")
 */
public function logout()
{
}

```

Figure 87: Ajout de route dans le contrôleur

L'authentification fonctionne, l'utilisateur est redirigé vers le serveur Keycloak :

Figure 88: Fenêtre d'authentification Keycloak

Puis je termine cette partie avec un autre commit & push.



7.8 Documentation technique

Avant de générer la document technique j'ai fait un tour sur l'ensemble du projet pour vérifier si les commentaires normalisés sont bien insérés partout, et que les types de variables sont bien définis.

Après avoir créé les commentaires manquantes, et complété les commentaire incomplets, il était temps de faire un autre push et commit puis je peux lancer la génération de la documentation technique avec phpDocumentor.

7.9 Créer un scénario sur la partie back-office pour tester la compatibilité des navigateurs

Pour la dernière tâche de cette mission il était demandé de créer un scénario de test de charge et de compatibilité avec Selenium. J'ai donc enregistré deux scénarios qui passent par les différentes fonctionnalités du site (un pour la partie front, un pour le back office), à l'aide de l'extension Chrome Selenium IDE.

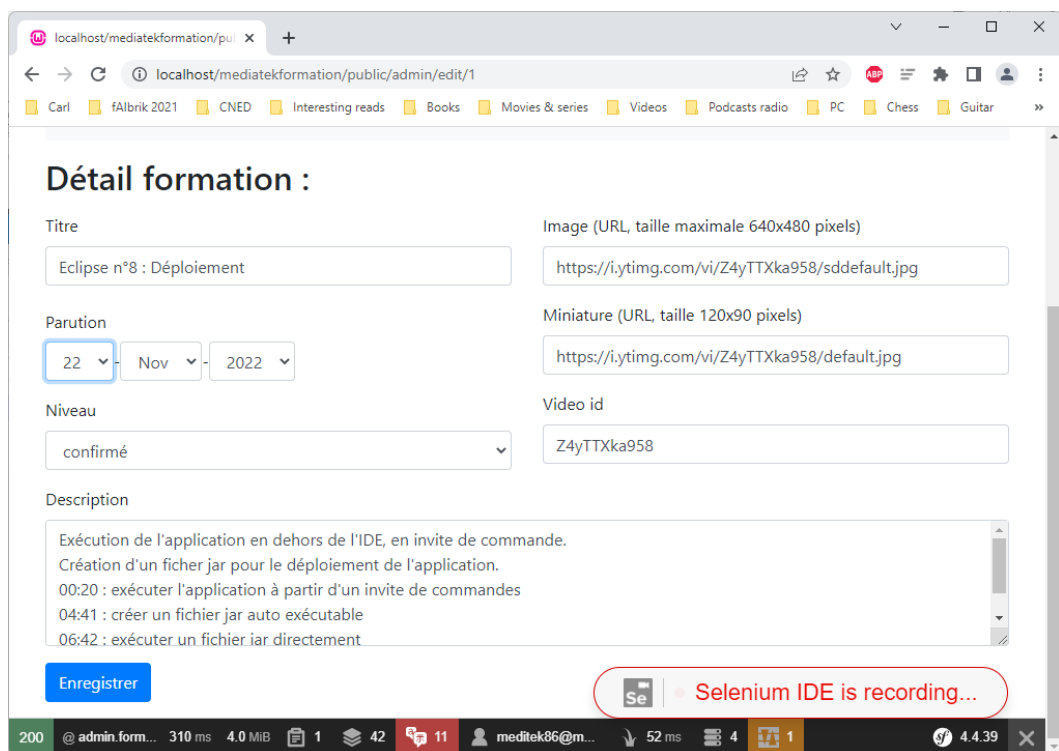


Figure 89: Enregistrement de test

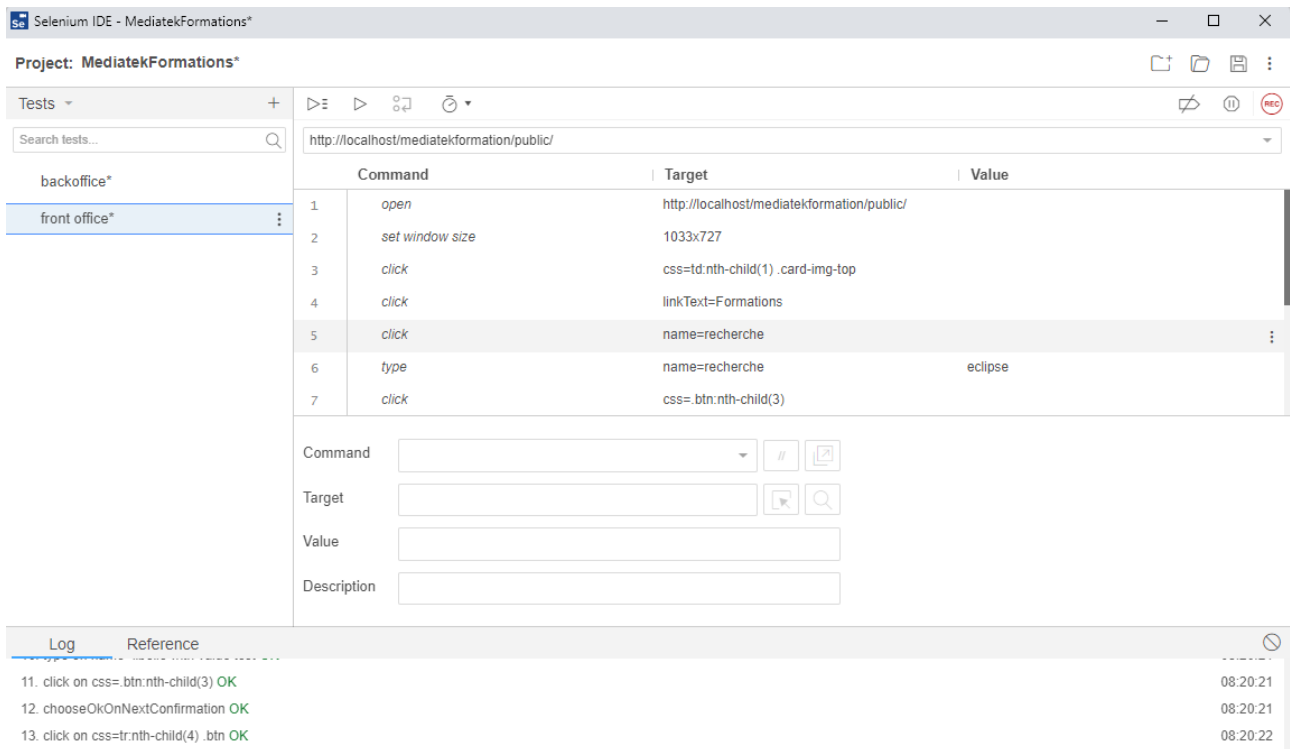
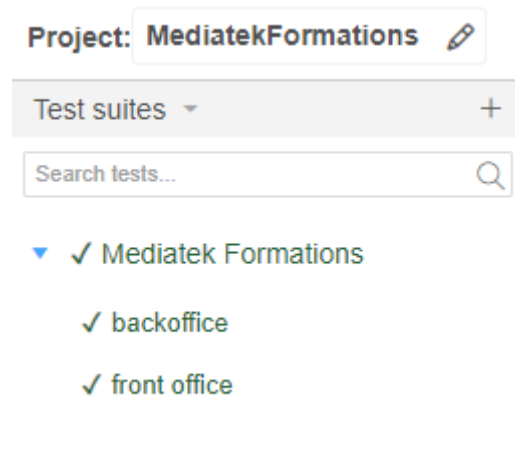


Figure 90: Les tests dans Selenium IDE

Ensuite j'ai ajouté les deux scénarios dans une 'suite' pour pouvoir les rejouer en parallèle.

Les tests passent :



Maintenant que les scénarios de test étaient enregistrés, j'ai pu les rejouer sur des clients différents. À part Chrome, où j'ai généré et vérifié les scénarios, j'ai exécuté les tests sur Firefox et Edge. Tout est fonctionnel à souhait.

A la fin de la deuxième mission il y a eu beaucoup de changements dans le code. Il était donc temps de repasser une vérification avec SonarQube. Ainsi j'ai fait un pull request puis un merge des branches 'dev' et 'main'. Après avoir fait un 'build' avec Jenkins, effectivement SonarQube trouve 2 'bugs' et 5 'code smells' :

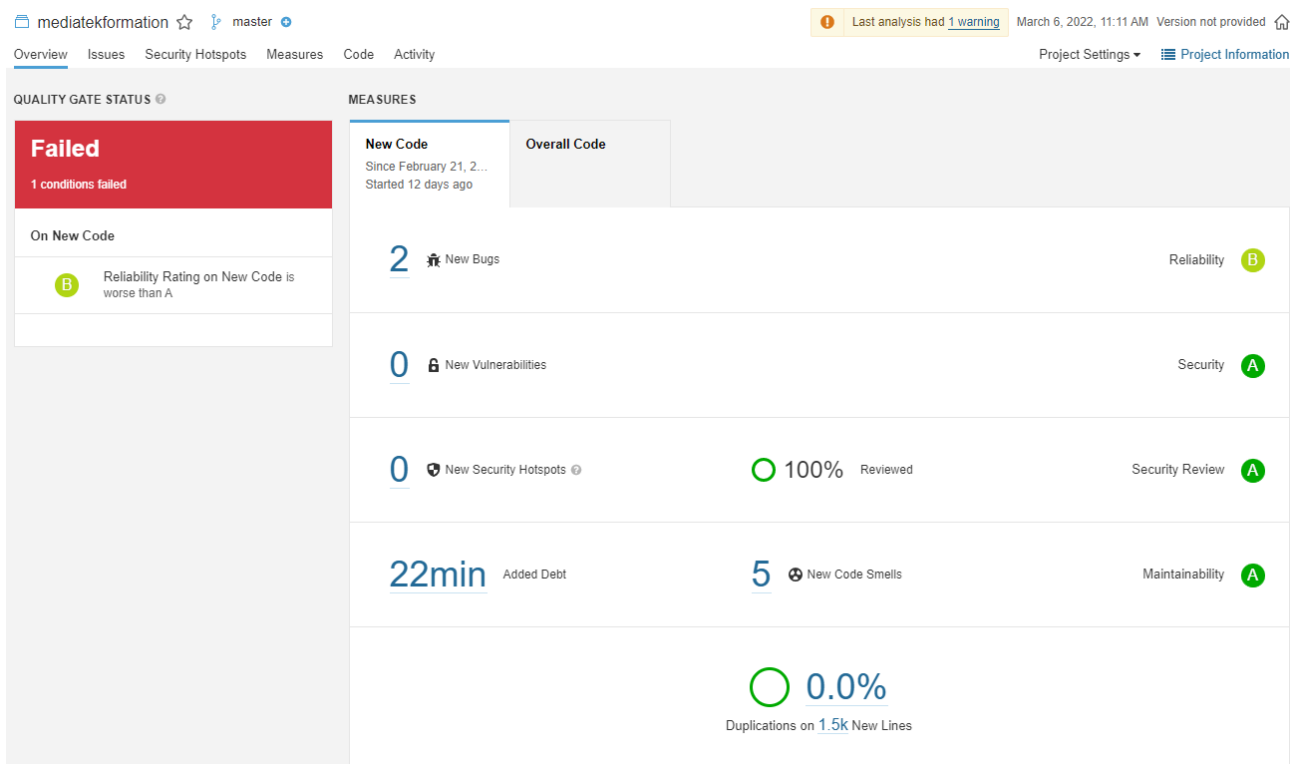


Figure 91: Analyse SonarQube

Les 'bugs' étaient vite résolus en ajoutant des attributs 'aria-label' :



Figure 92: Les 'bugs' rapportés par SonarQube

Et puis ils y avaient quelques corrections à faire :

src/Controller/admin/AdminFormationsController.php	Define a constant instead of duplicating this literal "admin.formations" 3 times. Why is this an issue?	10 days ago	L133	🔗	🔽
	🐞 Code Smell	🔴 Critical	🔵 Open	Not assigned	8min effort
					Comment
					🔗 design
src/Entity/Formation.php	Remove the unused function parameter "\$payload". Why is this an issue?	9 minutes ago	L195	🔗	🔽
	🐞 Code Smell	🔴 Major	🔵 Open	Not assigned	5min effort
					Comment
					🔗 unused
	Remove this empty statement. Why is this an issue?	9 minutes ago	L214	🔗	🔽
	🐞 Code Smell	🟢 Minor	🔵 Open	Not assigned	2min effort
					Comment
					🔗 unused
	Remove this empty statement. Why is this an issue?	9 minutes ago	L233	🔗	🔽
	🐞 Code Smell	🟢 Minor	🔵 Open	Not assigned	2min effort
					Comment
					🔗 unused
src/Entity/User.php	Rename "\$roles" which has the same name as the field declared at line 33. Why is this an issue?	9 days ago	L57	🔗	🔽
	🐞 Code Smell	🔴 Major	🔵 Open	Not assigned	5min effort
					Comment
					🔗 pitfall, suspicious

5 of 5 shown

Figure 93: Les 'code smells' trouvés par SonarQube

Après ces corrections, un nouveau commit, push, pull request et merge :

Phase 14 : Corrections suite analyse SonarQube #22

Open carlfremault wants to merge 1 commit into **main** from **dev**

Conversation 0 | Commits 1 | Checks 0 | Files changed 5,000+

carlfremault commented now

- ajout de 'aria-label' dans les tables des vues admin.formations.html.twig et admin.niveaux.html.twig
- définition d'une constante pour la route 'admin.formations' utilisée trois fois dans AdminFormationsController
- suppression d'un paramètre non-utilisé dans la classe métier Formation
- suppression de deux point-virgules non nécessaires après des blocs 'if'
- modification du nom de la variable locale \$roles dans la fonction getRoles de la classe métier User, car utilisé comme propriété aussi.

Phase 14 : Corrections suite analyse SonarQube **fef7461**

Add more commits by pushing to the **dev** branch on **carlfremault/mediatekformation**.

Continuous integration has not been set up
[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.

✓ This branch has no conflicts with the base branch
 Merging can be performed automatically.

Merge pull request You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Figure 94: Commit après corrections

SonarQube était satisfait :

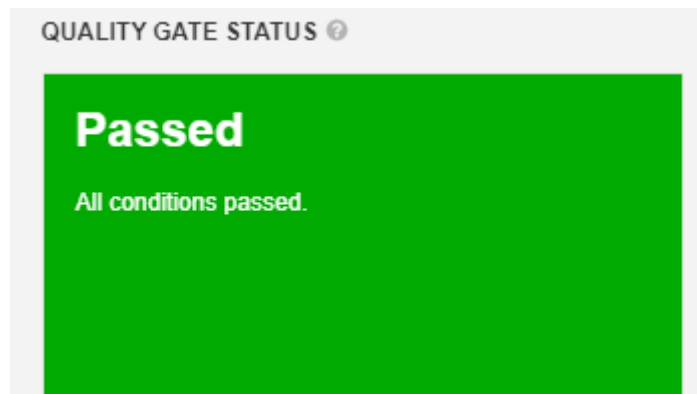


Figure 95: Qualité de code approuvée

Enfin, comme j'ai inséré une nouvelle constante (avec commentaire explicatif) j'ai dû régénérer la documentation technique.

7.10 Bilan pour la mission 2 :

Les différents objectifs ont été atteints. La page 'back office' est créée, elle présente les différentes formations de la même façon que la page front, avec les mêmes fonctionnalités de tri, filtrage et recherche. Il est possible d'ajouter, modifier ou supprimer une formation par intermédiaire d'un formulaire (pré-rempli dans le cas d'une modification). Les différents champs sont typés et sécurisés correctement.

La page de gestion des niveaux est également créée, avec possibilité de créer ou supprimer des niveaux. Lors d'une demande de suppression, d'abord l'application vérifie si le niveau concerné n'est pas utilisé pour une des formations, auquel cas la suppression n'est pas faite et l'utilisateur averti.

J'ai vérifié et modifié où nécessaire pour assurer la sécurité de l'application : typage des champs, présence de token CSRF, validation des saisies. Un test unitaire a été inséré.

Ensuite, j'ai intégré une fonctionnalité d'authentification SSO par intermédiaire d'un serveur Keycloak pour pouvoir accéder au back office, pour finir la mission avec l'enregistrement de tests de compatibilité de navigateur en utilisant Selenium.

Au cours de cette mission j'ai notamment dû faire des recherches pour deux fonctionnalités : la vérification de la taille d'une image 'remote', ainsi que l'affichage d'un message d'erreur dans une vue suite à une condition (non) remplie dans un contrôleur. Je pense que les solutions trouvées sont efficaces et suffisamment élégantes.

Quelques difficultés au niveau de la validation d'image se sont ajoutées après déploiement de l'application. Effectivement, le pare-feu de l'hébergeur n'apprécie pas la

saisie d'URL dans les champs. J'ai réussi à contourner le problème en ajoutant une fonction événementielle JavaScript dans le template de base de l'application'. Lorsqu'une entrée contenant la chaîne '/' est détectée, c'est uniquement la partie après cette chaîne qui est retournée. Par la suite, j'ai ajouté un événement 'Post_submit' dans le formulaire qui rajoute à nouveau 'http:/' devant l'entrée, avant enregistrement dans la base de données. Quelques captures d'écran sont ajoutées ci -dessous dans la partie '9. Déploiement'.

8. Mission 3 : Créer une vidéo de démonstration d'utilisation du site

Pour créer la vidéo de démonstration j'ai utilisé OBS Studio. La vidéo est disponible sur mon portfolio.

9. Déploiement

Pour le déploiement j'ai choisi de profiter de l'offre gratuite 'World Lite' de Planet Hoster. Cet offre permet la mise en ligne d'un site PHP/Symfony avec hébergement de base de données. La configuration est la mise en ligne ne sont pas compliquées avec un simple téléchargement FTP.

Surprise cependant, il s'avère que le pare-feu de l'hébergeur n'aime pas la saisie des URL pour les miniatures :

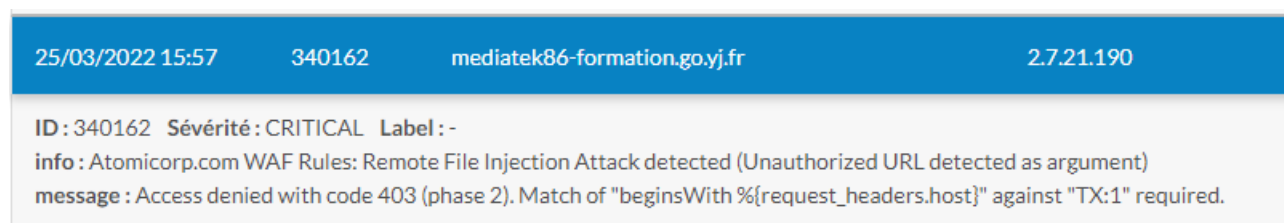


Figure 96: Détection de 'Remote File Injection Attack' lors de la saisie d'un URL dans le formulaire d'ajout de formation

Afin de contourner ce problème (qui serait résolu en souscrivant à l'offre d'hébergement payant qui permet de configurer les règles du pare-feu) il fallait un peu de gymnastique.

D'abord j'ai ajouté une fonction événementielle dans le template de base de l'application :

```

{% block javascripts %}
    <script>
        function validationForm() {
            var miniature = document.getElementById('formation_miniature');
            var miniaturevalue = miniature.value;
            if(miniaturevalue.search("/") >= 0){
                miniature.value = miniaturevalue.split("/") [1];
            }
            var picture = document.getElementById('formation_picture');
            var picturevalue = picture.value;
            if(picturevalue.search("/") >= 0){
                picture.value = picturevalue.split("/") [1];
            }
            return true;
        }
    </script>
{% endblock %}

```

Figure 97: Fonction événementielle appelée lors de la validation d'un formulaire

Cette fonction vérifie si la chaîne '/' est présente dans la saisie, et si oui, envoie uniquement la partie qui suit.

Pour que l'entité puisse vérifier s'il s'agit réellement d'un URL d'image, il fallait modifier la fonction 'validate' aussi :

```

if ($miniatureUrl != "") {
    try {
        $miniatureUrlurl = 'http://'.$miniatureUrl;
        $info = getimagesize($miniatureUrlurl);
    }
}

```

Figure 98: Ajout de 'http' devant l'URL reçu

Pourquoi http et non pas https ? Car au moins cela fonctionnerait pour tous les images (l'inverse ne serait pas vrai).

Quand le formulaire est accepté toutefois il faut rajouter la chaîne enlevé avant enregistrement dans la base de données. Pour ce faire j'ai ajouté un event 'post submit' dans le formulaire :

```

->addEventListener(FormEvents::POST_SUBMIT, function (FormEvent $event) {
    $infos = $event->getData();
    $picture = $infos->getPicture();
    if (strlen($picture) > 0 && strpos($picture, "//") === false) {
        $infos->setPicture(self::HTTP . $picture);
    }
    $miniature = $infos->getMiniature();
    if (strlen($miniature) > 0 && strpos($miniature, "//") === false) {
        $infos->setMiniature(self::HTTP . $miniature);
    }
    $event->setData($infos);
})

```

Figure 99: Ajout de 'http://' devant les url qui en sont dépourvus

Ainsi le pare-feu ne détecte plus d'attaques d'injection.

À vrai dire j'aurais préféré implémenter une solution à l'aide des fonctions 'encodeURIComponent' du côté client (JavaScript) puis 'html_entity_decode' du côté PHP. Ces fonctions assainissent et re-encodent respectivement les chaînes contenant des caractères 'spéciaux' HTML en les 'traduisant' : par exemple '/' devient '%20'. Toutefois cette solution n'arrive pas à contourner le pare-feu, j'ignore pourquoi.

10. Bilan final

A la fin de ce projet j'ai fait l'évolution demandée du site MediaTek86 Formations, selon les besoins formulés. J'ai ajouté la gestion des niveaux des formations sur la partie front en ajoutant une table dans la base de données puis j'ai implémenté les changements nécessaires au niveau de la vue.

Le travail le plus important était au niveau du back office, qui n'existait pas encore. J'ai donc créé les vues et les contrôleurs nécessaires, ainsi qu'un formulaire permettant d'ajouter ou de modifier une formation, en tenant compte des contraintes spécifiées : obligation de saisir titre, date et niveau, et un contrôle sur les tailles des images. Il est également possible d'ajouter et de supprimer un niveau, la suppression n'étant possible uniquement si le niveau en question n'est pas attribué à une des formations.

La totalité du back office est sécurisé avec des requêtes paramétrées, des token CSRF où cela est nécessaire, et le nettoyage des champs des saisies utilisateur. L'accès au back office passe dorénavant par une phase d'authentification avec Keycloak, avec fonctionnalité SSO.

J'ai inséré le test unitaire demandé, généré la documentation technique, et j'ai fait un scénario de test avec Selenium pour tester la compatibilité sur des différents navigateurs, ainsi que la résistance aux charges de l'application.

Finalement, pour optimiser la qualité du code j'ai utilisé l'intégration continue avec Jenkins et SonarQube. Cette partie n'était pas demandée mais j'ai voulu profiter de cet atelier pour mettre en pratique ces connaissances acquises pendant les cours.

Hormis quelques difficultés rencontrées en route, qui étaient vite résolues cependant, à l'aide des documentations officielles PHP et Symfony, ainsi que StackOverflow, l'ensemble du projet s'est passé de façon efficace. La principale difficulté ne s'est présentée qu'après déploiement : le pare-feu de l'hébergeur n'accepte pas de champ avec saisie d'URL. La solution trouvée contourne le problème. Ce n'est peut être pas la solution la plus élégante, toutefois dans une situation professionnelle l'hébergement ne sera pas fait avec une offre gratuite, et il y aura la possibilité de configurer le pare-feu à souhait.

L'ensemble du projet est hébergé dans un dépôt GitHub, et le suivi a été fait à l'aide d'un "GitHub Project", en mode "kanban automatisé" qui permet de gérer les tâches depuis NetBeans : chaque commit correspond ainsi à une tâche complétée.

11. Compétences couvertes

Bloc 1 :

- Répondre aux incidents et aux demandes d'assistance et d'évolution
 - Collecter, suivre et orienter des demandes
 - Traiter des demandes concernant les applications
- Développer la présence en ligne de l'organisation
 - Participer à l'évolution d'un site Web exploitant les données de l'organisation
- Travailler en mode projet
 - Analyser les objectifs et les modalités d'organisation d'un projet
 - Planifier les activités
- Mettre à disposition des utilisateurs un service informatique
 - Réaliser les tests d'intégration et d'acceptation d'un service
 - Déployer un service
 - Accompagner les utilisateurs dans la mise en place d'un service

Bloc 2 :

- Concevoir et développer une solution applicative
 - Participer à la conception de l'architecture d'une solution applicative
 - Exploiter les ressources du cadre applicatif (framework)
 - Identifier, développer, utiliser ou adapter des composants logiciels
 - Exploiter les technologies Web pour mettre en œuvre les échanges entre applications, y compris de mobilité
 - Utiliser des composants d'accès aux données
 - Intégrer en continu les versions d'une solution applicative
 - Réaliser les tests nécessaires à la validation ou à la mise en production d'éléments adaptés ou développés
 - Rédiger des documentations techniques et d'utilisation d'une solution applicative
 - Exploiter les fonctionnalités d'un environnement de développement et de tests
- Assurer la maintenance corrective ou évolutive d'une solution applicative
 - Recueillir, analyser et mettre à jour les informations sur une version d'une solution applicative
 - Évaluer la qualité d'une solution applicative
 - Analyser et corriger un dysfonctionnement
 - Mettre à jour des documentations techniques et d'utilisation d'une solution applicative
 - Élaborer et réaliser les tests des éléments mis à jour
- Gérer les données
 - Exploiter des données à l'aide d'un langage de requêtes
 - Concevoir ou adapter une base de données
 - Administrer et déployer une base de données

Bloc 3 :

- Sécuriser les équipements et les usages des utilisateurs
 - Identifier les menaces et mettre en œuvre les défenses appropriées
 - Gérer les accès et les privilèges appropriés

- Vérifier l'efficacité de la protection
- Assurer la cybersécurité d'une solution applicative et de son développement
 - Prendre en compte la sécurité dans un projet de développement d'une solution applicative
 - Prévenir les attaques