

# Compte-rendu de l'atelier de professionnalisation n°1

## Sommaire

### Contexte

**Mission 0 – Préparer l'environnement de travail (3/35)**

**Mission 1 – Nettoyer et optimiser le code existant (4/35)**

**Mission 2 – Coder le back-office (7/35)**

**Mission 3 – Tester et documenter (23/35)**

**Mission 4 – Déployer le site et gérer le déploiement continu (29/35)**

# Contexte

**MediaTek86 est une plateforme en ligne dédiée à la formation en ligne dans le domaine du développement informatique. La plateforme offre diverses pages, notamment une page présentant les formations disponibles, une page de playlists regroupant ces mêmes formations, une page d'accueil affichant les dernières publications, et une page de catégories qui organise les formations par thèmes.**

**Dans le cadre de notre projet, nous envisageons d'ajouter des fonctionnalités qui, après authentification, permettront aux utilisateurs d'ajouter, de modifier, ou de supprimer des formations, playlists et catégories. Pour ce faire, l'intégration de nouveaux boutons, formulaires, et tables dans la base de données est requise.**

**Ces améliorations nécessitent l'utilisation de divers langages et technologies. L'application est développée avec le framework Symfony, et les données sont stockées dans une base MySQL. L'interface utilisateur est conçue avec Bootstrap, tandis que la logique côté serveur est implémentée en PHP, suivant l'architecture modèle-vue-contrôleur (MVC). L'utilisation de l'extension SonarLint est prévue pour optimiser et standardiser le code selon les meilleures pratiques.**

**Une fois ces fonctionnalités intégrées et les éventuels problèmes de code résolus, nous procéderons au déploiement de l'application. Pour ce faire, nous utiliserons deux serveurs (l'un pour Keycloak, l'autre pour héberger l'application en elle-même).**

# Mission 0 – préparer l’environnement de travail

Il était initialement prévu de faire cet atelier avec Visual Studio Code, mais après quelques problèmes avec l’extension SonarLint, j’ai choisi Apache Netbeans IDE. Je récupère le projet mediatekformation sur github et je crée mon propre repo. J’initialise le projet avec la commande `composer install` dans le répertoire de mon projet et je crée la base de donnée en important mediatekformation.sql

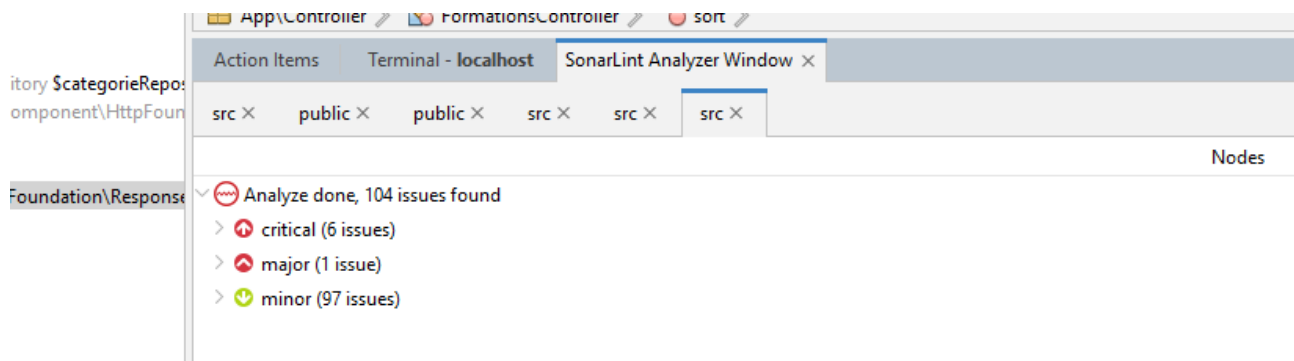
Je configure le fichier de configuration git sur netbeans en ajoutant mes credentials github :

```
[user]
  name = didlyn9
  email = ozgurbarkay39@gmail.com
```

et je génère ensuite le token que je vais utiliser tout au long de ce projet pour les commits.

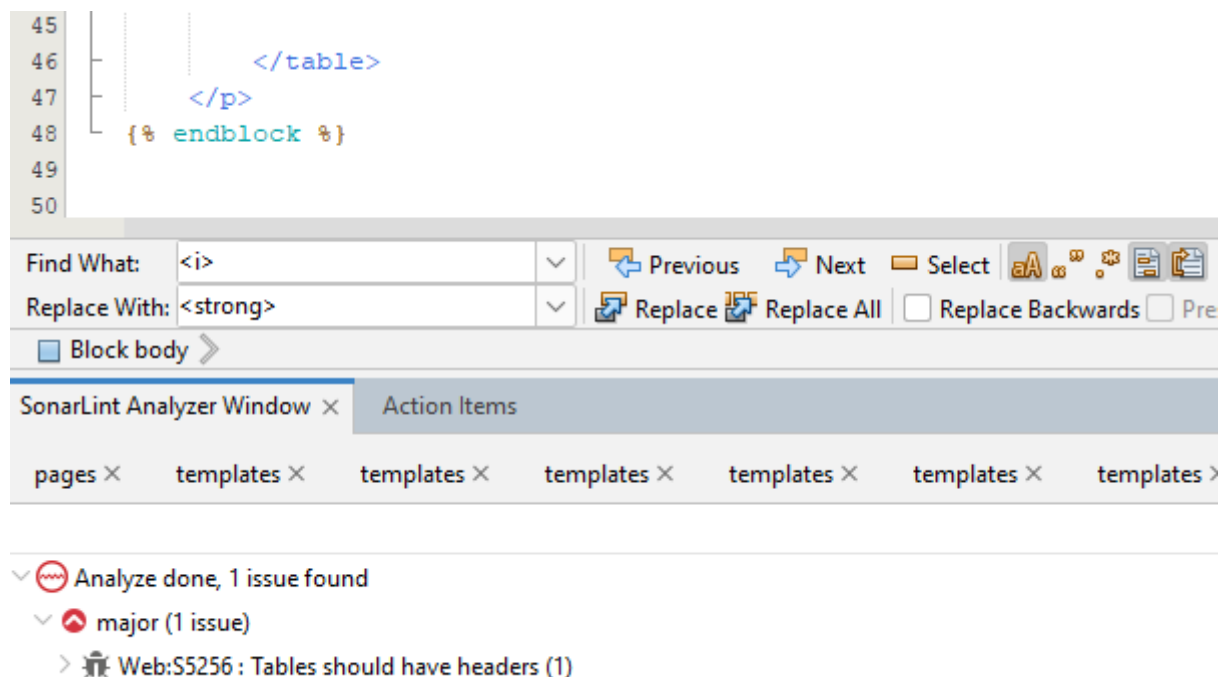
# Mission 1 – Nettoyer et optimiser le code existant

## Tâche 1 - nettoyer le code



Le nettoyage du code comprend la résolution d'erreurs (mineures, majeures jusqu'à critiques) signalées par SonarLint, dont le nettoyage des espaces vides/trailing whitespaces trouvables un peu partout, le formatage du code avec les conventions PSR-2, la correction des redondances avec des constantes (itérations de chaînes), des tests ou imbrications inutiles, l'absence d'attributs (description/caption/alt).

Il nous reste donc bien finalement une seule erreur :



## Tâche 2 – ajouter une fonctionnalité

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

Le contrôleur qui nous intéresse étant celui des playlists :

```
/**
 * @Route("/playlists", name="playlists")
 * @return Response
 */
public function index(): Response
{
    $playlists = $this->playlistRepository->findAllOrderByName('ASC');
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_PLAYLISTS, [
        'playlists' => $playlists,
        'categories' => $categories
    ]);
}
```

On rajoute un test dans la fonction sort de notre contrôleur :

```
/**
 * @Route("/playlists/tri/{champ}/{ordre}", name="playlists.sort")
 * @param type $champ
 * @param type $ordre
 * @return Response
 */
public function sort($champ, $ordre): Response
{
    if ($champ == "name") {
        $playlists = $this->playlistRepository->findAllOrderByName($ordre);
    }
    if ($champ == "amount") {
        $playlists = $this->playlistRepository->findAllOrderByAmount($ordre);
    }
    $categories = $this->categorieRepository->findAll();
    return $this->render(self::PAGE_PLAYLISTS, [
        'playlists' => $playlists,
        'categories' => $categories
    ]);
}
```

Et ensuite on code cette méthode dans PlaylistRepository.php

```

    /**
     * Retourne toutes les playlists triées sur la quantité de formation
     * @param type $ordre
     * @return Playlist[]
     */
    public function findAllOrderByAmount($ordre): array
    {
        return $this->createQueryBuilder('p')
            ->leftjoin('p.formations', 'f')
            ->groupBy('p.id')
            ->orderBy('count(f.id)', $ordre)
            ->getQuery()
            ->getResult();
    }

```

Il nous reste donc plus qu'à implémenter tout cela dans la vue (playlists.html.twig) :

```

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

<td>
    <h5 class="text-center">
        {{ playlists[k].formations|length }} {{ (playlists[k].formations|length) > 1 ? 'formations' : 'formation' }}
    </h5>
</td>

```

Ce qui conclue notre première mission.

# Mission 2 – Coder la partie back-office

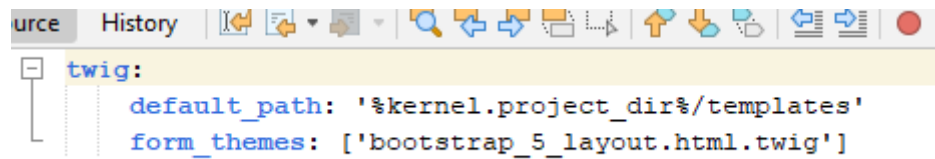
## Tâche 1 – gérer les formations

Je crée dans un premier temps le contrôleur AdminFormation pour le backoffice sur la base du contrôleur FormationsController, que je mets à jour, et je code la méthode suppr.

```
/**
 * @Route("/admin/formations/suppr/{id}", name="admin.formation.suppr")
 * @param Formation $formation
 * @return Response
 */
public function suppr(Formation $formation): Response{
    $this->formationRepository->remove($formation, true);
    $this->addFlash(
        'alert',
        'La formation ' . $formation->getTitle() . ' a été supprimé avec succès');
    return $this->redirectToRoute('admin.formations');
}
```

Pour la méthode edit et ajout, je crée un dossier Form pour y inclure les classes pour les formulaires, je crée FormationType et je le configure en fonction des variables dont nous allons avoir besoin pour ces deux méthodes.

J'inclue bootstrap5 dans le fichier twig.yaml pour pouvoir personnaliser le formulaire.



```
twig:
  default_path: '%kernel.project_dir%/templates'
  form_themes: ['bootstrap_5_layout.html.twig']
```

```
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
        ->add('publishedAt', DateType::class, [
            'widget' => 'single_text',
            'label' => 'Date'
        ])
        ->add('title')
        ->add('description')
        ->add('videoId')
        ->add('playlist', EntityType::class, [
            'class' => Playlist::class,
            'choice_label' => 'name',
            'required' => false
        ])
        ->add('categories', EntityType::class, [
            'class' => Categorie::class,
            'choice_label' => 'name',
            'multiple' => true,
            'required' => false
        ])
        ->add('submit', SubmitType::class, [
            'label' => 'Envoyer',
            'attr' => [
                'class' => 'btn btn-info'
            ]
        ])
    ];
}

public function configureOptions(OptionsResolver $resolver): void
{
    $resolver->setDefaults([
        'data_class' => Formation::class,
    ]);
}
```



Ensuite, je crée (pages/admin/) formation.html.twig et formations.html.twig et configure la vue :

```
History
{% extends "baseadmin.html.twig" %}
{% block body %}
    {{ include ('FormationForm.html.twig') }}
{% endblock %}
```

```
{{ form_start(formFormation) }}
<div class="row">
    <div class="col">
        <div class="row">
            <div class="col-auto">
                {{ form_row(formFormation.publishedAt) }}
            </div>
            <div class="col">
                {{ form_row(formFormation.title, {'label' : 'Titre'}) }}
            </div>
            <div class="col">
                {{ form_row(formFormation.videoId, {'label' : 'id de la vidéo'}) }}
            </div>
        </div>
        <br>
        {{ form_row(formFormation.description) }}
        <br>
        {{ form_row(formFormation.playlist) }}
        <br>
        {{ form_row(formFormation.categories, {'label' : 'Catégorie'}) }}
    </div>
</div>

<a href="{{ path('admin.formation.edit', {id:formation.id }) }}" class="btn btn-info">
    Modifier
</a>
```

Et après avoir implémenté le formulaire, je code la méthode edit.

```


/**
 * @Route("/admin/formation/edit/{id}", name="admin.formation.edit")
 * @param Formation $formation
 * @param Request $request
 * @return Response
 */
public function edit(Formation $formation, Request $request): Response{
    $formFormation = $this->createForm(FormationType::class, $formation);

    $formFormation->handleRequest($request);
    if($formFormation->isSubmitted() && $formFormation->isValid()){
        $this->formationRepository->add($formation, true);
        $this->addFlash(
            'success',
            'La formation ' . $formation->getTitle() . ' a été modifiée avec succès');
        return $this->redirectToRoute('admin.formations');
    }

    return $this->render(self::PAGE_FORMATION, [
        'formation' => $formation,
        'formFormation' => $formFormation->createView()
    ]);
}

```

Après quelques tests, nous sommes maintenant en mesure d'éditer les formations.

Date	Titre	id de la vidéo
04/01/2021 	Eclipse n°8 : Déploiement	Z4yTTXka958

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.

Playlist

Eclipse et Java

Catégorie

- Java
- UML
- C#
- Python

Envoyer

Ensuite, je code la méthode ajout.

```

/**
 * @Route("/admin/formation/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()){
        $this->formationRepository->add($formation, true);
        $this->addFlash(
            'success',
            'La formation ' . $formation->getTitle() . ' a été ajoutée avec succès');
        return $this->redirectToRoute('admin.formations');
    }

    return $this->render(self::PAGE_FORMATION, [
        'formation' => $formation,
        'formFormation' => $formFormation->createView()
    ]);
}

```

Que implémente ensuite dans la vue (formations.html.twig).

```

<a href="{{ path('admin.formation.ajout') }}" class="btn btn-info">Ajouter une formation</a>

```

## Tâche 2 – gérer les playlists

Je crée dans un premiers temps le contrôleur AdminPlaylist sur la base de PlaylistController et je crée la vue dans la partie admin sur la base de playlist.html.twig/playlists.html.twig que je mets à jour.

Je crée ensuite les boutons modifier et supprimer dans la vue (que je mettrai aussi à jour après avoir créé le formulaire et codé les méthodes).

Je crée dans un premier temps le formulaire PlaylistType dont je configure les variables :

```
class PlaylistType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name', TextType::class, [
                'required' => true
            ])
            ->add('description')
            ->add('formations', EntityType::class, [
                'class' => Formation::class,
                'multiple' => true,
                'required' => false
            ])
            ->add('submit', SubmitType::class, [
                'label' => 'Envoyer',
                'attr' => [
                    'class' => 'btn btn-info'
                ]
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Playlist::class,
        ]);
    }
}
```

```

{{ form_start(formPlaylist) }}
<div class="row">
    {{ form_row(formPlaylist.name, {'label' : 'Nom de la playlist'}) }}
    <br>
    {{ form_row(formPlaylist.description) }}
    <br>
    {{ form_row(formPlaylist. formations, {'attr': {'size': 25}, 'label': 'Formation(s)'}) }}
    <br>
    {{ form_row(formPlaylist.submit, {'attr': {'onclick': 'return confirm("Veuillez confirmer")'}}) }}
</div>
{{ form_end(formPlaylist) }}

```

Une formation ne pouvant être associé qu'à une playlist je procède à quelques ajouts au niveau des entités :

Pour l'entité Playlist :

```

public function __toString()
{
    return $this->name;
}

```

Et pour les formations :

```

public function __toString()
{
    return $this->title . (($this->playlist == null) ? "" : " Playlist : ") . $this->playlist;
}

```

Et je code subséquemment les méthodes suppr, edit et ajout :

```

/**
 * @Route("/admin/playlist/suppr/{id}", name="admin.playlist.suppr")
 * @param Playlist $playlist
 * @return Response
 */
public function suppr(Playlist $playlist): Response
{
    if(count($playlist->getFormations()) > 0){
        $this->addFlash(
            'alert',
            'Impossible de supprimer la playlist ' . $playlist->getName() . ', elle n\'est pas vide'
        );
        return $this->redirectToRoute('admin.playlists');
    }

    $this->playlistRepository->remove($playlist, true);
    $this->addFlash(
        'alert',
        'La playlist ' . $playlist->getName() . ' a bien été supprimée'
    );
    return $this->redirectToRoute('admin.playlists');
}

```

```

* @Route("/admin/playlist/edit/{id}", name="admin.playlist.edit")
* @param Playlist $playlist
* @param Request $request
* @return Response
*/
public function edit(Playlist $playlist, Request $request): Response
{
    $formationsIni = $playlist->getFormations()->toArray();
    $formPlaylist = $this->createForm(PlaylistType::class, $playlist);
    $formPlaylist->handleRequest($request);
    if ($formPlaylist->isSubmitted() && $formPlaylist->isValid()) {
        $formations = $playlist->getFormations()->toArray();
        foreach($formations as $formation) {
            if(!in_array($formation, $formationsIni)) {
                $formation->setPlaylist($playlist);
                $this->formationRepository->add($formation, true);
            }
        }
        foreach($formationsIni as $formation) {
            if(!in_array($formation, $formations)) {
                $formation->setPlaylist(null);
                $this->formationRepository->add($formation, true);
            }
        }
        $this->playlistRepository->add($playlist, true);
        $this->addFlash(
            'success',
            'La playlist ' . $playlist->getName() . ' a bien été ajoutée'
        );
        return $this->redirectToRoute('admin.playlists');
    }

    return $this->render(self::PAGE_PLAYLIST, [
        'playlist' => $playlist,
        'formPlaylist' => $formPlaylist->createView()
    ]);
}

```

```

/**
 * @Route("/admin/playlist/ajout", name="admin.playlist.ajout")
 * @param Request $request
 * @return Response
 */
public function ajout(Request $request): Response
{
    $playlist = new Playlist();
    $formPlaylist = $this->createForm(PlaylistType::class, $playlist);

    $formPlaylist->handleRequest($request);
    if ($formPlaylist->isSubmitted() && $formPlaylist->isValid()) {
        $this->playlistRepository->add($playlist, true);
        $formations = $playlist->getFormations()->toArray();
        foreach($formations as $formation) {
            $formation->setPlaylist($playlist);
            $this->formationRepository->add($formation, true);
        }
        $this->addFlash(
            'success',
            'La playlist ' . $playlist->getName() . " a bien été ajoutée"
        );
        return $this->redirectToRoute('admin.playlists');
    }

    return $this->render(self::PAGE_PLAYLIST, [
        'playlist' => $playlist,
        'formPlaylist' => $formPlaylist->createView()
    ]);
}

```

Ce qui conclue cette tâche.

## Tâche 3 – gérer les catégories

Dans un premier temps, je crée comme pour les autres tâches le contrôleur AdminCategorie de la même manière que pour les contrôleurs précédents, je crée le formulaire, je le configure, passe à la vue et code finalement les méthodes et je teste la vue.

```
class AdminCategorie extends AbstractController
{
    const PAGE_CATEGORIES = "pages/admin/categories.html.twig";

    const PAGE_CATEGORIE = "pages/admin/categorie.html.twig";

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

    public function __construct(CategorieRepository $categorieRepository) {
        $this->categorieRepository = $categorieRepository;
    }

    /**
     * @Route("/admin/categories", name="admin.categories")
     * @return Response
     */
    public function index(): Response{
        $categories = $this->categorieRepository->findAllOrderByName('ASC');
        return $this->render(self::PAGE_CATEGORIES, [
            'categories' => $categories
        ]);
    }
}
```



```

class CategorieType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name', TextType::class, [
                'required' => true
            ])
            ->add('formations', EntityType::class, [
                'class' => Formation::class,
                'multiple' => true,
                'required' => false,
            ])
            ->add('submit', SubmitType::class, [
                'label' => 'Envoyer',
                'attr' => [
                    'class' => 'btn btn-info'
                ]
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Categorie::class,
        ]);
    }
}

```

Ayant eu quelques problèmes avec doctrine et la configuration de ma bdd, je décide de rajouter deux méthodes dans CategorieRepository pour contourner ledit problème, et dont je me servirai dans mon contrôleur pour les méthodes qui m'ont posé problème :

```

/**
 * Ajoute la formation à la catégorie
 *
 * @param type $id_formation
 * @param type $id_categorie
 * @return void
 */
public function addFormationCategorie($id_formation, $id_categorie): void{
    $query = "INSERT INTO formation_categorie (formation_id, categorie_id)
    VALUE ($id_formation, $id_categorie);";
    $connexion = $this->getEntityManager()->getConnection();
    $connexion->executeQuery($query);
}

/**
 * Supprime la formation de la catégorie
 *
 * @param [type] $id_formation
 * @param [type] $id_categorie
 * @return void
 */
public function delFormationCategorie($id_formation, $id_categorie): void{
    $query = "
    DELETE FROM formation_categorie
    WHERE formation_id = $id_formation and categorie_id = $id_categorie;
    ";
    $connexion = $this->getEntityManager()->getConnection();
    $connexion->executeQuery($query);
}

```

```

/**
 * @Route("/admin/categorie/edit/{id}", name="admin.categorie.edit")
 * @param Categorie $categorie
 * @param Request $request
 * @return Response
 */
public function edit(Categorie $categorie, Request $request): Response
{
    $formationsIni = $categorie->getFormations()->toArray();
    $formCategorie = $this->createForm(CategorieType::class, $categorie);
    $formCategorie->handleRequest($request);
    if ($formCategorie->isSubmitted() && $formCategorie->isValid()){
        $this->categorieRepository->add($categorie, true);
        $formations = $categorie->getFormations()->toArray();
        foreach($formations as $formation){
            if(!in_array($formation, $formationsIni)){
                $this->categorieRepository->addFormationCategorie($formation->getId(), $categorie->getId());
            }
        }
        foreach($formationsIni as $formation){
            if(!in_array($formation, $formations)){
                $this->categorieRepository->delFormationCategorie($formation->getId(), $categorie->getId());
            }
        }
        $this->addFlash(
            'success',
            'La catégorie ' . $categorie->getName() . " a bien été modifiée"
        );
        return $this->redirectToRoute('admin.categories');
    }

    return $this->render(self::PAGE_CATEGORIE, [
        'categorie' => $categorie,
        'formCategorie' => $formCategorie->createView()
    ]);
}

```

pub  
{

```

/**
 * @Route("/admin/categorie/ajout", name="admin.categorie.ajout")
 * @param Request $request
 * @return Response
 */
public function ajout(Request $request): Response
{
    $categorie = new Categorie();
    $formCategorie = $this->createForm(CategorieType::class, $categorie);

    $formCategorie->handleRequest($request);
    if ($formCategorie->isSubmitted() && $formCategorie->isValid()){
        $this->categorieRepository->add($categorie, true);
        $formations = $categorie->getFormations()->toArray();
        foreach($formations as $formation) {
            $this->categorieRepository->addFormationCategorie($formation->getId(), $categorie->getId());
        }
        $this->addFlash(
            'success',
            'La catégorie ' . $categorie->getName() . " a bien été ajoutée"
        );
        return $this->redirectToRoute('admin.categories');
    }

    return $this->render(self::PAGE_CATEGORIE, [
        'categorie' => $categorie,
        'formCategorie' => $formCategorie->createView()
    ]);
}

```

Je fais quelques réajustements au niveau de la vue et je conclue cette tâche.

## Tâche 4 – ajouter l'accès avec authentification

Note : Pour éviter quelques confusions, la façon dont j'implémente Keycloak diffère en fonction de si l'environnement est local ou si c'est celui de production étant donné qu'en local j'utilise Windows (et en production Linux, plus précisément Debian 10). Je rapporterai les quelques déboires que j'ai eu avec (surtout pour php) dans la partie déploiement.

Je télécharge dans un premier temps Keycloak (19.0.1) que je lance avec

```
kc.bat start-dev
```













Je crée le realm « Sio » et je crée le client « mediatekformation » en OpenID Connect avec les options :

Standard flow  
Implicit flow  
Direct access grants

Et je récupère le client secret, que je rajoute dans mon fichier .env

Ensuite, je crée l'utilisateur admin dont je configure le mot de passe (sio:adminadmin) dans « get credentials », et pour l'utiliser, je crée une entité User (que je configure avec id, email, roles, password, keycloak\_id) et j'effectue la migration avec doctrine :

```
doctrine make:migration
```

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1	id 	int			Non	Aucun(e)		AUTO_INCREMENT	 Modifier  Supprimer Plus
<input type="checkbox"/>	2	email 	varchar(180)	utf8mb4_unicode_ci		Non	Aucun(e)			 Modifier  Supprimer Plus
<input type="checkbox"/>	3	roles	json			Non	Aucun(e)			 Modifier  Supprimer Plus
<input type="checkbox"/>	4	password	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			 Modifier  Supprimer Plus
<input type="checkbox"/>	5	keycloak_id	varchar(255)	utf8mb4_unicode_ci		Oui	NULL			 Modifier  Supprimer Plus

Une fois tout cela fait, j'installe les dépendances pour Keycloak avec les commandes :

```
composer require knpuniversity/oauth2-client-bundle 2.10
```

```
composer require stevenmaguire/oauth2-keycloak 3.1
```

Et que je configure

knpu\_oauth2\_client.yaml :

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

security.yaml :

```
security:
  enable_authenticator_manager: true
  # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
  password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
  # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
  providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
      entity:
        class: App\Entity\User
        property: email
  firewalls:
    dev:
      pattern: ^/(_(profiler|wdt)|css|images|js)/
      security: false
  main:
    lazy: true
    entry_point: form_login
    # provider: app_user_provider
    form_login:
      login_path: oauth_login
    custom_authenticators:
      - App\Security\KeycloakAuthenticator
    logout:
      path: logout

    # activate different ways to authenticate
    # https://symfony.com/doc/current/security.html#the-firewall

    # https://symfony.com/doc/current/security/impersonating user.html
    # switch_user: true

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

Je crée ensuite le contrôleur OAuthController, que je configure

OAuthController.php :

```
class OAuthController extends AbstractController
{
    /**
     * @Route("/oauth/login", name="oauth_login")
     */
    public function index(ClientRegistry $clientRegistry): RedirectResponse{
        return $clientRegistry->getClient('keycloak')->redirect();
    }

    /**
     * @Route("/oauth/callback", name="oauth_check")
     */
    public function connectCheckAction(Request $request, ClientRegistry $clientRegistry){

    }

    /**
     * @Route("/logout", name="logout")
     */
    public function logout(){

    }
}
```

KeycloakAuthenticator.php :

```
public function authenticate(Request $request): Passport {
    $client = $this->clientRegistry->getClient('keycloak');
    $accessToken = $this->fetchAccessToken($client);
    return new SelfValidatingPassport(
        new UserBadge($accessToken->getToken(), function() use ($accessToken, $client){
            /** @var KeycloakUser $keycloakUser */
            $keycloakUser = $client->fetchUserFromToken($accessToken);
            $existingUser = $this->entityManager
                ->getRepository(User::class)
                ->findOneBy(['keycloakId' => $keycloakUser->getId()]);
            if($existingUser){
                return $existingUser;
            }
            $email = $keycloakUser->getEmail();
            /** @var User $userInDatabase */
            $userInDatabase = $this->entityManager
                ->getRepository(User::class)
                ->findOneBy(['email' => $email]);
            if($userInDatabase){
                $userInDatabase->setKeycloakId($keycloakUser->getId());
                $this->entityManager->persist($userInDatabase);
                $this->entityManager->flush();
                return $userInDatabase;
            }
            $user = new User();
            $user->setKeycloakId($keycloakUser->getId());
            $user->setEmail($keycloakUser->getEmail());
            $user->setPassword("");
            $user->setRoles(['ROLE_ADMIN']);
            $this->entityManager->persist($user);
            $this->entityManager->flush();
            return $user;
        })
    );
}
```

Tout fonctionne, pour accéder au backoffice et se connecter avec keycloak, il nous suffit simplement de rajouter admin dans l'accueil :

`http://localhost/mediatekformation/public/index.php/admin`

Et j'ajoute finalement au niveau de la vue le bouton qui nous permet de nous déconnecter :

```
<a href="{{ path('logout') }}" class="text-decoration-none text-info">Se déconnecter&nbsp;&nbsp;&nbsp;</a>
```

# Mission 3 – Tester et documenter

## Tâche 1 – gérer les tests

Je crée une bdd de test (mediatekformation\_test).

### Tests unitaires

Contrôler le fonctionnement de la méthode qui retourne la date de parution au format string : DateTest.php

```
C:\wamp64\www\mediatekformation>php bin/phpunit --filter DateTest
PHPUnit 9.5.23 #StandWithUkraine

Testing
...
3 / 3 (100%)

Time: 00:00.041, Memory: 10.00 MB

OK (3 tests, 3 assertions)
```

### Tests d'intégration sur les règles de validation

Lors de l'ajout ou de la modification d'une formation, contrôler que la date n'est pas postérieure à aujourd'hui : VerifDateTest.php

```
C:\wamp64\www\mediatekformation>php bin/phpunit --filter VerifDateTest
PHPUnit 9.5.23 #StandWithUkraine

Testing
.
1 / 1 (100%)

Time: 00:00.245, Memory: 10.00 MB

OK (1 test, 1 assertion)
```

## Tests d'intégration sur les Repository

Contrôler toutes les méthodes ajoutées dans les classes Repository :

CategorieRepositoryTest.php

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

Testing
...
3 / 3 (100%)

Time: 00:14.462, Memory: 62.00 MB

OK (3 tests, 3 assertions)
```

FormationRepositoryTest.php

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

Testing
...
3 / 3 (100%)

Time: 00:00.412, Memory: 28.00 MB

OK (3 tests, 3 assertions)
```

PlaylistRepositoryTest.php

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

Testing
...
3 / 3 (100%)

Time: 00:00.313, Memory: 28.00 MB

OK (3 tests, 3 assertions)
```



## Tests fonctionnels

Contrôler que la page d'accueil est accessible.

Dans chaque page contenant des listes :

- contrôler que les tris fonctionnent (en testant juste le résultat de la première ligne) ;
- contrôler que les filtres fonctionnent (en testant le nombre de lignes obtenu et le résultat de la première ligne) ;
- contrôler que le clic sur un lien (ou bouton) dans une liste permet d'accéder à la bonne page (en contrôlant l'accès à la page mais aussi le contenu d'un des éléments de la page).

Pour mener ces tests, nous allons utiliser le crawler fourni par Symfony.

AccueilControllerTest.php

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

Testing
...
3 / 3 (100%)

Time: 00:02.731, Memory: 40.00 MB

OK (3 tests, 8 assertions)
```

FormationControllerTest.php

```
C:\wamp64\www\mediatekformation>php bin/phpunit --filter FormationControllerTest
PHPUnit 9.5.23 #StandWithUkraine

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

Time: 00:01.496, Memory: 42.00 MB

OK (5 tests, 15 assertions)
```

## PlaylistControllerTest.php

```
C:\wamp64\www\mediatekformation>php bin/phpunit --filter PlaylistControllerTest
PHPUnit 9.5.23 #StandWithUkraine

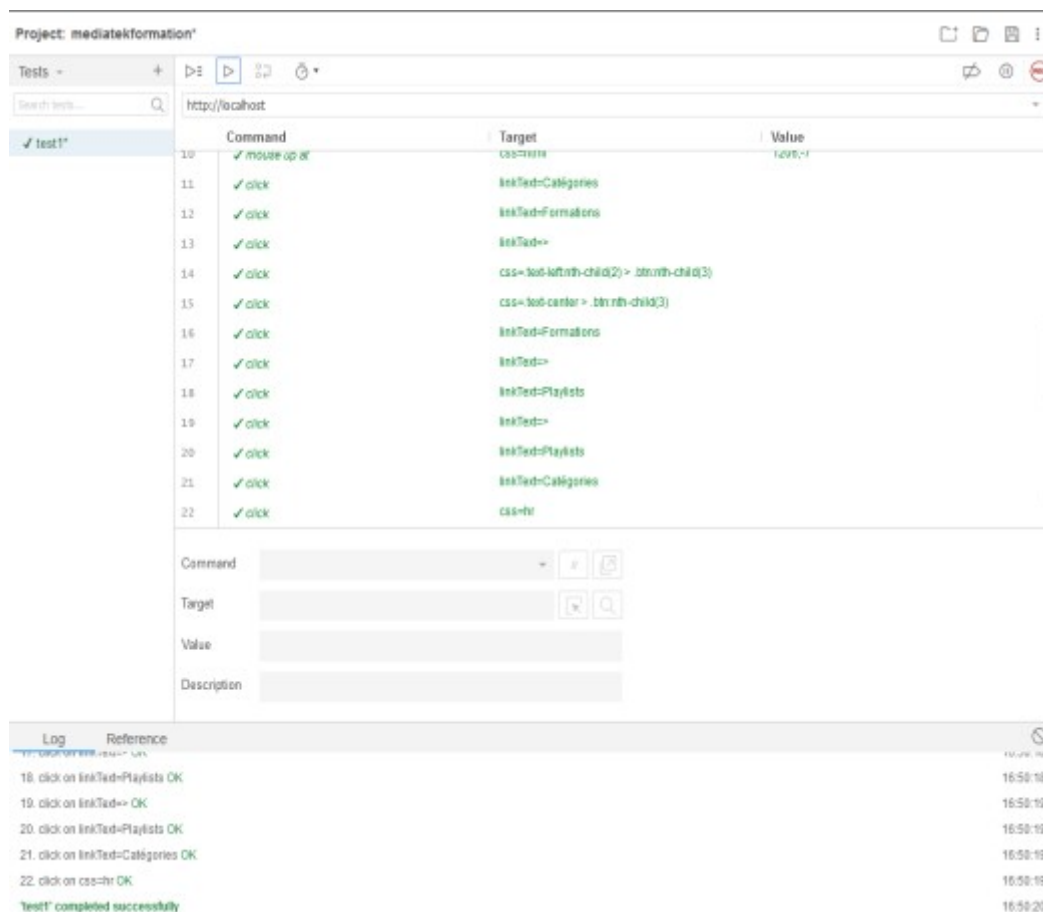
Testing
.....

Time: 00:01.244, Memory: 40.00 MB

OK (5 tests, 15 assertions)
```

## Tests de compatibilité

On enregistre un scénario avec l'extension sous Chrome, et on l'exporte pour tester avec Firefox :



Command	Target	Value
mouse up at	css=html	1200,1
click	linkText=Catégories	
click	linkText=Formations	
click	linkText=>	
click	css=tbody:nth-child(2) > tbody-child(3)	
click	css=tbody-center > tbody-child(3)	
click	linkText=Formations	
click	linkText=>	
click	linkText=Playlists	
click	linkText=>	
click	linkText=Playlists	
click	linkText=Catégories	
click	css=hr	

Log Reference

Log	Reference
18. click on linkText=Playlists OK	16:50:18
19. click on linkText=> OK	16:50:19
20. click on linkText=Playlists OK	16:50:19
21. click on linkText=Catégories OK	16:50:19
22. click on css=hr OK	16:50:19
test1* completed successfully	16:50:20

Sous firefox :

The screenshot displays the Selenium IDE interface for a project named 'mediatekformation'. The test suite 'test1' is selected and contains four 'click' commands. The background shows a web application with a sidebar menu and a main content area.

## MediaTek86

Accueil For...

- Java
- UML
- C#
- Python
- MCD
- Android
- POO
- SQL
- Cours

tableau des ca...

**Project: mediatekformation**

Tests ▾ + ▶ ▶ ⌂ ⌚ ⌂

Search tests... Run current test Ctrl+R

	Command	Target	Value
✓ test1			
19	✓ click	linkText=>	
20	✓ click	linkText=Playlists	
21	✓ click	linkText=Catégories	
22	✓ click	css=hr	

Command: [dropdown] [//] [↗]

Target: [input] [↗] [🔍]

Value: [input]

Description: [input]

Log	Reference	
17. click on linkText=> OK		16:55:52
18. click on linkText=Playlists OK		16:55:53
19. click on linkText=> OK		16:55:54
20. click on linkText=Playlists OK		16:55:55
21. click on linkText=Catégories OK		16:55:56
22. click on css=hr OK		16:55:57
*test1* completed successfully		16:55:58

100 @ categories 33 ms 6.0 MB 2 n/a 3 ms 1 in 0.44 ms

## **Tâche 2 – créer la documentation technique**

Pour créer la documentation technique, j'ai eu quelques soucis de compatibilité avec la version de php et le phpDocumentor.phar fourni dans le wiki. La solution que j'ai trouvée était de changer la version de php utilisé dans mes options Netbeans et Wamp. On exécute la commande :

```
C:\wamp64\bin\php\php8.2.13\php.exe" "C:\wamp64\bin\php\php8.2.13\ext\
phpDocumentor.phar" "run" "--ansi" "--directory"
"C:/wamp64/www/mediatekformation/src" "--target"
"C:/wamp64/www/mediatekformation_doc" "--title" "mediatekformation
```

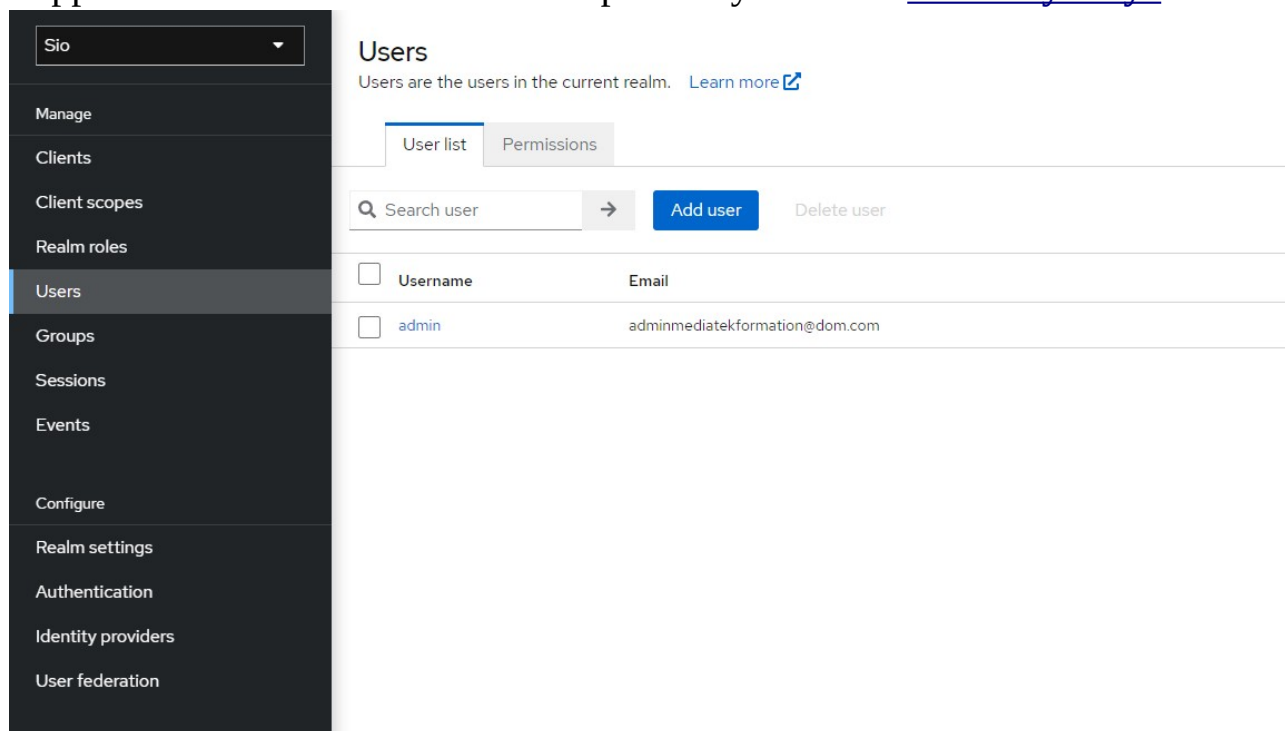
## **Tâche 3 – créer la documentation utilisateur**

L'enregistrement de la documentation utilisateur a été fait via OBS Studio et la vidéo a été compressée avec Handbrake.

# Mission 4 – déployer le site et gérer le déploiement continu

## Tâche 1 – déployer le site

L'application a été déployée via 2 serveurs sous Linux, l'un pour l'application en elle-même et l'autre pour Keycloak sur [www.neyer.xyz](http://www.neyer.xyz)



J'ai eu quelques problèmes pour utiliser composer à cause de la version de php mais en réinitialisant le vps ovh sous debian 10, j'ai pu utiliser une version compatible de php pour pouvoir installer les dépendances avec `composer install`

Après avoir configuré les entrées DNS, j'ai déployé le certificat SSL via certbot, aussi bien pour keycloak (lekeycloak.francecentral.cloudapp.azure.com) que pour neyer.

En ce qui concerne la documentation technique de l'application, vous pouvez la retrouver sur mon portfolio [www.ozgurbarkay.fr](http://www.ozgurbarkay.fr) dans l'onglet « portfolio », le nom de domaine a été acheté sur ovh, et hébergé via github pages.

J'ai rencontré notamment un problème au niveau des redirections http et https qui étaient dues au fait que je n'arrivais pas à éditer et sauvegarder le fichier

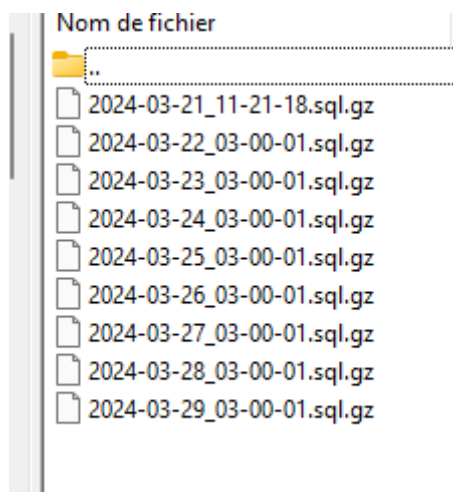
000-default-le-ssl.conf et que j'ai pu résoudre en supprimant et transférant depuis filezilla. Ce problème m'empêchait notamment de pouvoir accéder à phpmyadmin depuis neyer.xyz ce que je voulais absolument pouvoir faire (avec le SSL).

## **Tâche 2 – gérer la sauvegarde et la restauration de la BDD**

Pour ce qui est de la sauvegarde de la bdd, j'ai eu quelques soucis dans un premier temps avec ovh. En n'ayant acheté qu'un serveur et n'ayant souscrit à aucune offre d'hébergement, les tâches cron m'étaient indisponibles. J'ai donc installé mysql sur la vm de mon serveur et j'ai paramétré apache pour pouvoir accéder via l'url.

En ce qui concerne la sauvegarde la bdd, j'ai tout simplement créé un script pour le backup dans /home/scripts/ et j'ai créé une tâche cron pour tous les jours à 3h du matin dans ce répertoire.

J'ai utilisé dos2unix convertir le script (en raison des fins de lignes qui après transfert ne sont pas au bon format).



```
#!/bin/bash

source /home/debian/scripts/.env

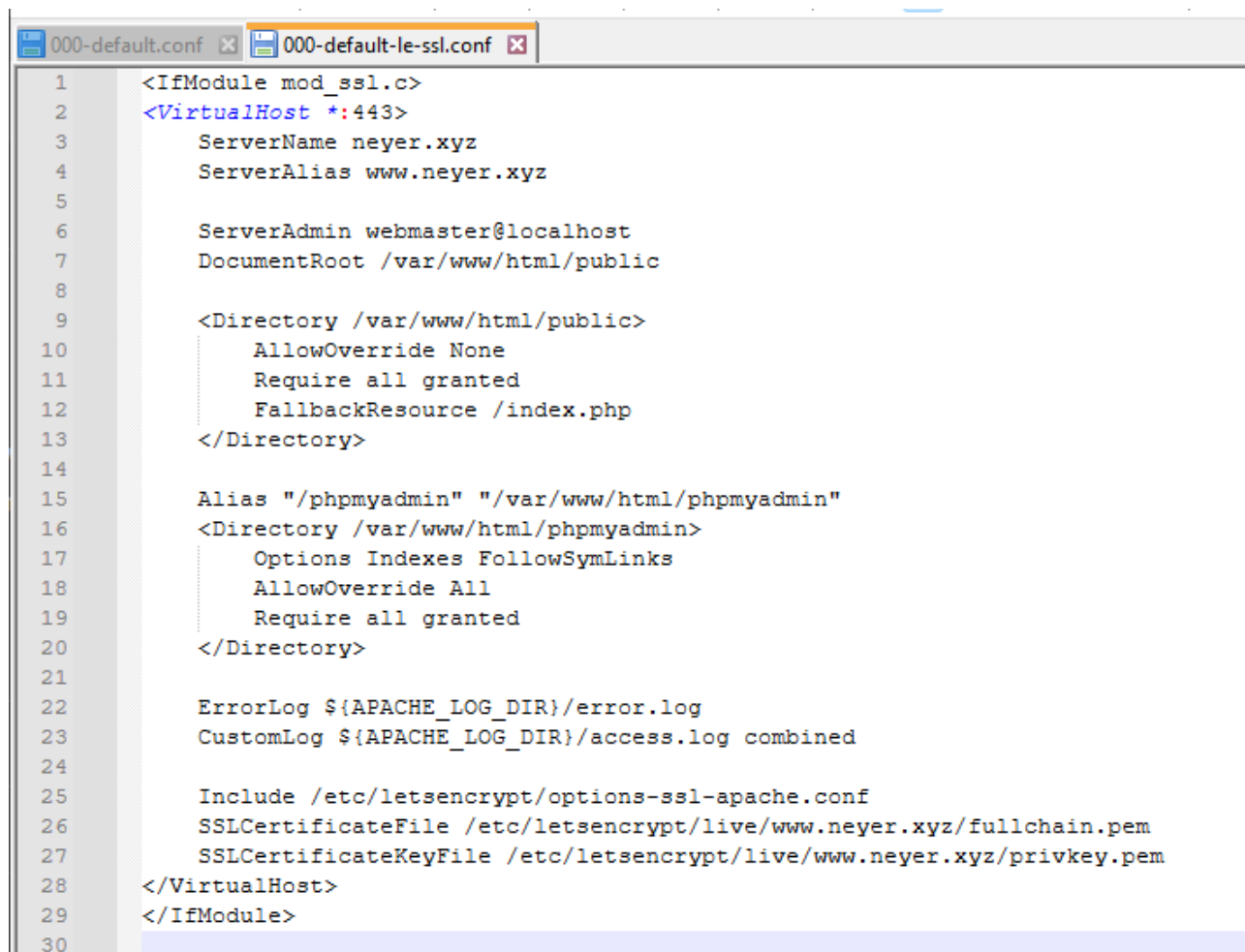
BACKUP_DIR="/home/debian/scripts/bddsave/"
DATE=$(date +%Y-%m-%d_%H-%M-%S)
BACKUP_FILE="$BACKUP_DIR/$DB_NAME_$DATE.sql"

mysqldump -u $DB_USER -p"$DB_PASSWORD" $DB_NAME > $BACKUP_FILE

gzip $BACKUP_FILE

echo "Sauvegarde de la base de données effectuée avec succès."
```

Après les dernières paramétrages au niveau des DNS pour le nom de domaine OVH et les fichiers de configuration apache dans sites-avaible, j'en ai terminé pour le déploiement de l'application.



```
000-default.conf x 000-default-le-ssl.conf x
1 <IfModule mod_ssl.c>
2 <VirtualHost *:443>
3     ServerName neyer.xyz
4     ServerAlias www.neyer.xyz
5
6     ServerAdmin webmaster@localhost
7     DocumentRoot /var/www/html/public
8
9     <Directory /var/www/html/public>
10         AllowOverride None
11         Require all granted
12         FallbackResource /index.php
13     </Directory>
14
15     Alias "/phpmyadmin" "/var/www/html/phpmyadmin"
16     <Directory /var/www/html/phpmyadmin>
17         Options Indexes FollowSymLinks
18         AllowOverride All
19         Require all granted
20     </Directory>
21
22     ErrorLog ${APACHE_LOG_DIR}/error.log
23     CustomLog ${APACHE_LOG_DIR}/access.log combined
24
25     Include /etc/letsencrypt/options-ssl-apache.conf
26     SSLCertificateFile /etc/letsencrypt/live/www.neyer.xyz/fullchain.pem
27     SSLCertificateKeyFile /etc/letsencrypt/live/www.neyer.xyz/privkey.pem
28 </VirtualHost>
29 </IfModule>
30
```

```

ServerName neyer.xyz
ServerAlias www.neyer.xyz

ServerAdmin webmaster@localhost
DocumentRoot /var/www/html/public
    <Directory /var/www/html/public>
        AllowOverride None
        Require all granted
        FallbackResource /index.php
    </Directory>
# Configuration de phpMyAdmin
Alias "/phpmyadmin" "/var/www/html/phpmyadmin"
    <Directory /var/www/html/phpmyadmin>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# For most configuration files from conf-available/, which are
# enabled or disabled at a global level, it is possible to
# include a line for only one particular virtual host. For example the
# following line enables the CGI configuration for this host only
# after it has been globally disabled with "a2disconf".
#Include conf-available/serve-cgi-bin.conf
RewriteEngine on
RewriteCond %{SERVER_NAME} =www.neyer.xyz
RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

```



J'ai volontairement compressé pour pouvoir me mettre en situation dans le cas où l'on gérerait une base de donnée plus conséquente (qu'on a juste à décompresser) puis réimporter.

### Tâche 3 – mettre en place le déploiement continu

J'ai eu quelques soucis pour le déploiement continu, mais après quelques recherches et tests pour finalement utiliser un paquet d'actions de déploiement compatible avec le sftp, ça a fini par marcher (branche prod-v2).

```
on: [push]
name: Deploy website on push
jobs:
  web-deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Deploy file
        uses: wlixcc/SFTP-Deploy-Action@v1.2.4
        with:
          username: debian
          server: 51.75.26.163
          port: 22
          local_path: './*'
          remote_path: '/var/www/html/'
          sftp_only: true
          password: ${ secrets.FTP_PASSWORD }
```

test déploiement continu

didlyn9 committed last week · ✓ 1 / 1

Update main.yml

didlyn9 committed last week · ✓ 1 / 1

Nom de fichier	Taille de
var	
vendor	
.env	2
.env.test	
.gitignore	
composer.json	
composer.lock	417
mediatekformation.sql	126
phpunit.xml.dist	1
README.md	8
symfony.lock	8
test.txt	
test2.txt	

Verified 557aebb

Verified a921d36



# Bilan

Ainsi, au cours de ce projet, j'ai fait face à plusieurs défis et commis des erreurs qui m'ont permis d'apprendre beaucoup. Malgré ces difficultés, j'ai réussi à mettre en place de nouvelles fonctionnalités dans l'application Mediatekformation, touchant à tous les aspects de l'application, de l'interface utilisateur au back-office. Ces modifications ont nécessité des ajustements complexes et parfois délicats.

Mes erreurs ont souvent découlé d'une méconnaissance initiale de certaines parties de Symfony et de ses bonnes pratiques, ainsi que d'une sous-estimation de la complexité de certaines tâches, tandis que je sur-estimais la complexité d'autres tâches alors que les corrections à apporter étaient parfois toutes bêtes...

En fin de compte, malgré les difficultés rencontrées, cette expérience a été très enrichissante. Elle m'a permis d'améliorer mes compétences en développement web (dont le déploiement) avec Symfony, et de pouvoir mieux appréhender les défis futurs.