

SOMMAIRE

ATELIER DE PROFESSIONNALISATION

Contexte	Pages 2/20
Mission 1-Nettoyer et optimiser le code existant	Page 3/20
Mission 2-Coder le back-office	Page 6/20
Mission 3-Tester et documenter	Page 14/20
Mission 4-Déployer le site et gérer le déploiement continu	Page 18/20
Bilan	Page 19/20

Contexte :

Mediatekformation est une application de gestion de cours en ligne en rapport avec le domaine du développement informatique.

Plusieurs pages sont accessibles dont la page des formations proposées, la page des playlists qui sont composées de plusieurs formations, une page d'accueil qui reprend les dernières publications et une page de catégories qui regroupe l'ensemble des catégories attribuées aux formations.

Dans le contexte de notre mission, nous devons ajouter des fonctionnalités qui permettront, après une authentification, d'ajouter, de modifier ou de supprimer les formations, les playlists et les catégories.

Afin d'implémenter ces fonctionnalités, il sera nécessaire d'ajouter de nouveaux liens sous forme de boutons, de nouveaux formulaires et de nouvelles tables dans la base de données.

La mise en place de ces nouveautés est reliée à plusieurs langages et technologies.

L'application est développée sur Symfony tandis que les données sont gérées sur une base de données SQL.

L'interface graphique est, par conséquent, faite en bootstrap tandis que le back-office est principalement construit en PHP à partir d'un schéma de modèle view controller.

Des extensions comme Sonarlint seront aussi utilisées afin de nettoyer le code et adapter notre code aux normes.

Enfin, après avoir ajouté toutes les fonctionnalités et réglé tous les problèmes liés à notre code, nous pourrons l'héberger en utilisant un site d'hébergement dans lequel nous enverrons nos fichiers grâce à FileZilla.

Mission 1 :

Tâche 1 - Nettoyer le code :

Temps estimé : 2h

Temps réel : 2h

Le nettoyage du code comprend la résolution des erreurs citées par l'outil SonarLint dont des erreurs d'itérations de chaînes, des tests inutiles, des mauvais noms de constantes et l'absence d'attribut description pour les tables et d'attribut alt pour les images de l'application traitée.

```
/**
 * Début de chemin vers les images
 */
private const CHEMIN_IMAGE = "https://i.ytimg.com/vi/";
```

```
const SELECT = 'p.id id';
const ADDNAME = 'p.name name';
const ADDSELECT = 'c.name categorienome';
```

```
->select(self::SELECT)
->addSelect(self::ADDNAME)
->addSelect(self::ADDSELECT)
```

```

```

```
<table class="table table-striped" description="description">
```

Tâche 2 - Respecter les bonnes pratiques du codage :

Temps estimé : 2h

Temps réel : 1h

Cette tâche consiste à respecter le principe de single responsibility en modifiant le code des fichiers FormationRepository et PlaylistRepository pour diviser les fonctions afin qu'elles soient divisées en fonction de leur utilité.

Des modifications doivent également être apportées afin que le projet s'adapte à ces modifications.

Sur playlistRepository, la fonction findByContainValue sera divisée en deux fonctions dont une qui prendra les cas de chaîne vide et l'autre dans le cas inverse.

Ainsi nous avons une fonction nommée findByContain et une autre nommée findByValue.

```
public function findByContain(): array{
    return $this->findAllOrderBy('name', 'ASC');
}

public function findByValue($champ, $valeur): array{
    return $this->createQueryBuilder('p')
        ->select(PlaylistRepository::SELECT)
        ->addSelect(PlaylistRepository::ADDNAME)
        ->addSelect(PlaylistRepository::ADDSELECT)
        ->leftjoin('p.formations', 'f')
        ->leftjoin('f.categories', 'c')
        ->where('c.'.$champ.' LIKE :valeur')
        ->setParameter('valeur', '%'.$valeur.'%')
        ->groupBy('p.id')
        ->addGroupBy('c.name')
        ->orderBy('p.name', 'ASC')
        ->addOrderBy('c.name')
        ->getQuery()
        ->getResult();
}
```

Pour FormationRepository, le tri des données est effectué par la fonction findAll si aucun filtre n'est utilisé, dans le cas contraire la fonction findByValue prend en compte les valeurs entrées par l'utilisateur dans les différents champs disponibles.

```

/**
 * Tri de l'ensemble des données de manière décroissante
 * @return array
 */
public function findAll(): array
{
    return $this->createQueryBuilder('f')
        ->orderBy('f.publishedAt', 'DESC')
        ->getQuery()
        ->getResult();
}

```

```

/**
 * Tri des champs en fonction des entrées utilisateur
 * @param type $champ
 * @param type $valeur
 * @param type $table
 * @return array
 */
public function findByValue($champ, $valeur, $table = ''): array
{
    if ($valeur == '') {
        return $this->findAll();
    }

    $qb = $this->createQueryBuilder('f');
    if ($table != '') {
        $qb->join('f.' . $table, 't')
            ->where('t.' . $champ . ' LIKE :valeur')
            ->setParameter('valeur', '%' . $valeur . '%');
    } else {
        $qb->where('f.' . $champ . ' LIKE :valeur')
            ->setParameter('valeur', '%' . $valeur . '%');
    }

    return $qb->orderBy('f.publishedAt', 'DESC')
        ->getQuery()
        ->getResult();
}

```

Tâche 3 – Ajouter une fonctionnalité :

Temps estimé : 2h

Temps réel : 1h

Cette tâche consiste à sélectionner et trier les playlists disponibles afin d'afficher le nombre de formations qui y sont rattachées.

Mission 2 :

Tâche 1 – Gérer les formations :

Temps estimé : 5h

Temps réel : 5h

Le temps estimé pour cette tâche est de 3h, le temps effectif d'une heure.

Les formations peuvent être lus mais pas modifiées ni supprimer.

L'objectif de cette tâche est de permettre de trier, de supprimer, d'ajouter et de modifier les formations de l'application.

Il faudra donc créer un formulaire d'ajout, une option de suppression et une option de modification

```
<a href="{{ path('formation_delete', {'id':formation.id}) }}"
onclick="return confirm('Voulez vous supprimer la formation suivante : {{formation.title}} ?') "
class="btn btn-danger">Supprimer
</a>
```

```
/**
 * @Route("/formation/{id}/delete", name="formation delete")
 * @param Formation $formation
 * @return RedirectResponse
 */
public function suppr(Formation $formation): RedirectResponse
{
    $em = $this->getDoctrine()->getManager();
    $em->remove($formation);
    $em->flush();

    return $this->redirectToRoute("formations");
}
```

qui sera basé sur le formulaire, chacune de ces option sera liée à un bouton spécifique. Pour la suppression le code est le suivant :

Pour l'édition et l'ajout du code, il est nécessaire de créer un formulaire qu'on inclura sur des pages dédiées à ces possibilités afin de laisser l'utilisateur saisir les informations nécessaires et de les communiquer à la base de données.

Pour cela, nous ajoutons un fichier FormationType dont le contenu est le suivant :

```

public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
    ->add('publishedAt', DateTime::class, [
        'widget' => 'single_text',
        'required' => true, // champ obligatoire
        'empty_data' => new \DateTime(),
        'data' => isset($options['data']) ? $options['data']->getPublishedAt() : null,
        'label' => 'Date'
    ])
    ->add('title', null, [
        'required' => true // champ obligatoire
    ])
    ->add('description')
    ->add('videoId', null, [
        'required' => true // champ obligatoire
    ])
    ->add('categories', EntityType::class, [
        'label' => 'Categories',
        'class' => Categorie::class,
        'choice_label' => 'name',
        'multiple' => true,
        'expanded' => true,
        'required' => true // champ obligatoire
    ])
    ->add('playlist', EntityType::class, [

```

Pour faire fonctionner ce formulaire, il est aussi nécessaire de créer de nouvelles fonctions dans le contrôleur et de leur attribuer une route vers les fichiers twig précédemment créés.

```

/**
 * @Route ("/admin/ajout", name="formation_ajout")
 * @param Formation $formation
 * @return Response
 * @param Request $request
 */
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);
        return $this->redirectToRoute('formations');
    }
    return $this->render("admin/formationAjout.html.twig",[
        'formation'=>$formation,
        'formFormation' =>$formFormation->createView()
    ]);
}

```

```

/**
 * @Route ("/formation/{id}/edit", name="formation_edit")
 * @param Formation $formation
 * @return Response
 * @param Request $request
 */
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);
        return $this->redirectToRoute('formations');
    }
    return $this->render("pages/ajoutFormation.html.twig",[
        'formation'=>$formation,
        'formFormation' =>$formFormation->createView()
    ]);
}

```


Tâche 2 - gérer les playlists

Temps estimé: 5h

Temps réel : 4h

La gestion des playlists se fait de la même manière que celle des formations à quelques exceptions près.

Les boutons d'ajout, de suppression et de modification sont bien ajoutés au fichier twig.

```
<td>
    <a href="{{ path('playlist_edit', {'id':playlists[k].id}) }}"
    class="btn btn-secondary">Editer
</a>

    <a href="{{ path('playlist_delete', {'id':playlists[k].id}) }}"
    onclick="return confirm('Voulez vous supprimer la playlist suivante : {{playlists[k].name}} ?'"
    class="btn btn-danger">Supprimer
</a>
</td>
{% endif %}
```

Ces liens sont reliés à des pages de formulaires qui permettent d'interagir avec la base de données à partir du contrôleur.

Les playlists peuvent ainsi être gérées depuis l'interface utilisateur.

```
{# empty Twig template #}
{{ form_start(formPlaylist) }}
    <div class="col">
        {{ form_row(formPlaylist.name) }}
        {{ form_row(formPlaylist.description) }}
    </div>
{{ form_end(formPlaylist) }}
```

```

<?php

namespace App\Form;

use App\Entity\Playlist;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class PlaylistType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name')
            ->add('description', null, [
                'required' => true // champ obligatoire
            ])
            ->add('submit', SubmitType::class, [
                'label' => 'Enregistrer'
            ]);
    }

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

```

Tâche 3- gérer les catégories

Temps estimé: 3h

Temps réel : 2h

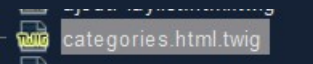
Pour cette tâche, nous devons créer une nouvelle page qui pourra accueillir les catégories stockées en base de données et ajouter un formulaire d'ajout à cette page afin de pouvoir en ajouter d'autres depuis l'interface.

Pour cela nous allons créer un nouveau lien dans la barre de navigation ainsi qu'un nouveau fichier twig.

```

<li class="nav-item">
  <a class="nav-link" href="{{ path('accueil') }}">Accueil</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="{{ path('formations') }}">Formations</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="{{ path('playlists') }}">Playlists</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="{{ path('categories') }}">Categories</a>
</li>

```



categories.html.twig

Il est aussi nécessaire d'ajouter un bouton de suppression de la même manière que pour les pages de playlists et de formations.

```

<td>
  <a href="{{ path('categorie_delete', {'id':categorie.id}) }}"
    onclick="return confirm('Voulez vous supprimer la categorie suivante : {{categorie.name}} ?')"
    class="btn btn-danger">Supprimer
  </a>
</td>

```

Un formulaire et un controller peuvent être créés afin d'apporter un événement lors de l'interaction avec les liens ajoutés à la page.

```

<form class="form-inline mt-1" method="POST" action="{{ path('categorie_ajout') }}">
<div class="form-group mr-1 mb-2">
  <input type="text" class="sm" name="nom">
  <button type="submit" class="btn btn-primary mb-2 btn-sm">Ajouter</button>
</div>
</form>

```

```

public function suppr(Categorie $categorie): RedirectResponse
{
    $formations = $categorie->getFormations();
    foreach ($formations as $categorie) {
        if ($categorie->getCategories() !== null) {
            $this->addFlash('warning', 'La categorie est liée à une formation');
            return $this->redirectToRoute('categories');
        }
    }

    $this->repository->remove($categorie, true);
    $this->addFlash('success', 'La categorie a été supprimée avec succès');
    return $this->redirectToRoute('categories');
}

/**
 * @Route("/pages/categories/ajout", name="categorie ajout")
 * @param Request $request
 * @return Response
 */
public function ajout(Request $request): Response{
    $nomCategorie = $request->get("nom");
    $categorie=new Categorie;
    $categorie->setName($nomCategorie);
    $this->repository->add($categorie, true);
    return $this->redirectToRoute('categories');
}
}

```

Tâche 4 - ajouter l'accès avec authentification

Temps estimé : 4h

Temps réel : 2h

Pour cette tâche, nous devons ajouter un système d'authentification afin de limiter l'accès du back-end aux comptes présents dans la base de données.

Il est avant tout nécessaire de créer une nouvelle table 'user' qui pourra recevoir les informations des utilisateurs inscrits.

 Structure de table
  Vue relationnelle

	#	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	username 	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


☐ Tout cocher
 Avec la sélection :
  Parcourir
  Modifier
  Supprimer
  Primaire
  Unique
  Index
  Spatial
  Texte entier

Il est à présent possible de créer une page de formulaire en twig et de la lier au contrôleur auquel on peut ajouter une fonction de connexion.

```

<form action='{{ path('login') }}' method='post'>
  <h3>Authentifiez-vous</h3>
  <label for="username">Email :</label>
  <input type="text" id="username" name="_username" value="{{ last_username }}"
    class="form-control" required autofocus/>
  <label for="password">Password :</label>
  <input type="password" id="password" name="_password"
    class="form-control" required/>
  <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}"
  <br/>
  <button type="submit" class="btn btn-lg btn-primary">Se connecter</button>
</form>

```

Une fonction de déconnexion est aussi ajoutée afin d'enlever l'accès au back-end de l'application une fois que l'utilisateur le voudra.

```

/**
 * @Route("/login", name="login")
 */
public function index(AuthenticationUtils $authenticationUtils): Response
{
    $error=$authenticationUtils->getLastAuthenticationError();
    $lastUsername=$authenticationUtils->getLastUsername();
    return $this->render('login/index.html.twig', [
        'last_username'=>$lastUsername,
        'error'=>$error
    ]);
}

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

```

Mission 3 :

Tâche 1 - Gérer les tests

Temps estimé : 7h

Temps réel : 5h

Afin d'exécuter des tests unitaires, il est important d'installer au préalable les composants nécessaires en passant pas la fenêtre de commandes puis en entrant la commande "php bin/phpunit". Les composants seront alors installés par phpunit.

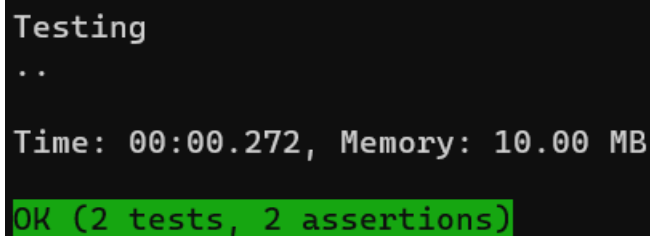
Le test unitaire se fait dans un fichier placé dans un dossier conçu pour ces tâches nommé "test", nous allons créer notre fichier de test au nom DataTest à cet endroit.

La classe sera équipée d'une fonction testGetPublishedAtString();

Afin d'effectuer le test, nous pouvons simplement relancer la même commande maintenant que les composants sont bien installés.

Si le test est concluant, la fenêtre de commande devrait renvoyer une ligne verte, dans le cas contraire il s'agira d'une ligne rouge avec un message d'erreur.

Dans notre cas, le test est valide.



```
Testing
..

Time: 00:00.272, Memory: 10.00 MB

OK (2 tests, 2 assertions)
```

Nous devons également procéder à des test d'intégration, notamment pour vérifier que la date d'une formation n'est pas postérieur à celle du jour de la publication ou de la modification.

Ensuite, nous allons utiliser la même méthode pour contrôler les méthodes des classes repository.

Voici la fonction qui reprend les éléments de l'entité formation dont la fonction "setPublishedAt" afin de contrôler l'entrer la date dans le cas où celle-ci serait postérieur à la date du jour :



```
<?php

use App\Entity\Formation;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

class VerifDateTest extends KernelTestCase{

    public function testPublishedAtIsFutureDate()
    {
        $futureDate = new DateTime('+1 day');
        $formation = new Formation();
        $formation->setPublishedAt($futureDate);

        $this->assertTrue($formation->getPublishedAt() > new DateTime(), 'La date de publication doit être antérieur à la date actuelle.');
```

La commande "php bin/phpunit --filter VisiteValidationsTest" nous permet de vérifier la validité de ce test avec un code couleur similaire au test précédent.

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

Testing
.                                                                    1 / 1 (100%)

Time: 00:00.249, Memory: 10.00 MB

OK (1 test, 1 assertion)
```

Pour tester les repository, il faut également créer un dossier afin de le différencier du dossier dans lequel les autres test ont été faits.

Nous allons lier le test à notre base de données, pour débiter nous devons créer une nouvelle class qui reçoit une fonction qui va servir à vérifier le nombre de formations disponible dans la table et le comparer au chiffre entré.

```
class FormationRepositoryTest extends KernelTestCase {
    //put your code here
    public function recupRepository(): FormationRepository{
        self::bootKernel();
        $repository=self::getContainer()->get(FormationRepository::class);
        return $repository;
    }
    public function testNbFormations() {
        $repository=$this->recupRepository();
        $nbFormations=$repository->count([]);
        $this->assertEquals(127, $nbFormations);
    }
}
```

Pour lier le test à la base de données, il faut préciser le chemin de celle-ci dans le fichier env_test de la même manière que dans le fichier env à une différence près.

Nous allons copier la table afin d'en avoir une seconde avec la mention "test" pour la différencier et la rendre éligible au test.

```
Time: 00:00.543, Memory: 26.00 MB
```

```
OK (1 test, 1 assertion)
```

```
# define your env variables for the test env here
KERNEL_CLASS='App\Kernel'
APP_SECRET='$ecretf0rt3st'
SYMFONY_DEPRECATIONS_HELPER=999999
PANTHER_APP_ENV=panther
PANTHER_ERROR_SCREENSHOT_DIR=./var/error-screenshots
DATABASE_URL="mysql://root:@127.0.0.1:3306/mediatekformation"
```

Il est maintenant possible de procéder au test des méthodes du repository.
Il est par exemple possible d'ajouter des formations.

```
public function testAddFormation() {
    $repository=$this->recupRepository();
    $formation=$this->newFormation();
    $nbFormations=$repository->count([]);
    $repository->add($formation,true);
    $this->assertEquals($nbFormations+1,$repository->count([]),"erreur");
}
```

Les tests sont désormais liés à la base de données, plusieurs fonctions ont été ajoutées dans le but de vérifier les méthodes présentes dans le repository.

L'ajout, la sélection et la suppression des données sont testés dans la fenêtre de commande et renvoient un nombre de tests et d'insertion en réponse aux fonctions écrites prouvant leur bon fonctionnement.

```
public function testAddFormation() {
    $repository=$this->recupRepository();
    $formation=$this->newFormation();
    $nbFormations=$repository->count([]);
    $repository->add($formation,true);
    $this->assertEquals($nbFormations+1,$repository->count([]),"erreur");
}

public function testRemoveFormation() {
    $repository=$this->recupRepository();
    $formation=$this->newFormation();
    $repository->add($formation,true);
    $nbFormations=$repository->count([]);
    $repository->remove($formation,true);
    $this->assertEquals($nbFormations-1,$repository->count([]),"erreur");
}

public function testFindByEqualValue() {
    $repository=$this->recupRepository();
    $formation=$this->newFormation();
    $repository->add($formation,true);
    $formations=$repository->findByEqualValue("Title","Test");
    $nbFormations=count($formations);
    $this->assertEquals(1,$nbFormations);
    $this->assertEquals("Test",$formations[0]->getTitle());
}
```

```
Testing
...
3 / 3 (100%)

Time: 00:00.546, Memory: 26.00 MB

OK (3 tests, 3 assertions)
```


Il est aussi possible de procéder à des tests unitaire qui vérifient le bon fonctionnement du controller.

Pour cela nous allons ajouter un dossier et un fichier Class dédié au controller de la même manière que pour le repository.

Pour débiter nous pouvons faire un test d'accès sur une page de l'application, ici nous essayons avec la page "login" qui permet au client de l'application de se connecter pour avoir accès au services.

```
class FormationsControllerTest extends WebTestCase {  
    //put your code here  
  
    public function testAccesPage() {  
        $client=static::createClient();  
        $client->request('GET', '/login');  
        $this->assertResponseStatusCodeSame(Response::HTTP_OK);  
    }  
}
```

Pour tester les filtres présents sur une page, il est possible de créer une formation qui prend en compte le nom du champs, le contenu de celui-ci et la balise liée au résultat attendu.

Ainsi, nous pouvons créer une fonction testFiltreTitre() :

```
public function testFiltreTitre(){  
    $client = static::createClient();  
    $client->request('GET', '/formations');  
    $crawler=$client->submitForm('filtrer', [  
        'recherche'=>'cours'  
    ]);  
    $this->assertCount(1,$crawler->filter('h5'));  
    $this->assertSelectorTextContains('h5', 'cours');  
}
```

Dans ce cas précis, nous filtrons les formations avec le filtre de titre dans lequel le texte 'cours' est inséré.

Il est également possible de vérifier le bon fonctionnement d'un lien vers une autre page ainsi que du contenu de cette page après que le client ait appuyé dessus.

Dans notre exemple, nous prenons le cas du lien vers les formations depuis l'index de l'application.

```
public function testeLinkFormations() {  
    $client = static::createClient();  
    $client->request('GET', '/index.php');  
    $client->clickLink('Formations');  
    $response=$client->getResponse();  
    $this->assertEquals(Response::HTTP_OK,$response->getStatusCode());  
    $uri=$client->getRequest()->server->get("REQUEST_URI");  
    $this->assertEquals("/formations",$uri);  
}
```

Mission 4 :

Tâche 1 – Déployer le site

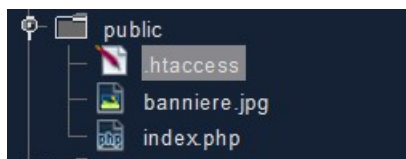
Temps estimé : 2h

Temps réel : 2h

Le déploiement du site requiert la modification des condition générales d'utilisation, le choix d'un hébergeur et la mise en place d'une base de données en ligne qui prend en charge les contraintes de l'application.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> categorie	★ Parcourir Structure Rechercher Insérer Vider Supprimer	9	InnoDB	utf8mb4_unicode_ci	16,0 kio	-
<input type="checkbox"/> doctrine_migration_versions	★ Parcourir Structure Rechercher Insérer Vider Supprimer	6	InnoDB	utf8_unicode_ci	16,0 kio	-
<input type="checkbox"/> formation	★ Parcourir Structure Rechercher Insérer Vider Supprimer	237	InnoDB	utf8mb4_unicode_ci	160,0 kio	-
<input type="checkbox"/> formation_categorie	★ Parcourir Structure Rechercher Insérer Vider Supprimer	277	InnoDB	utf8mb4_unicode_ci	48,0 kio	-
<input type="checkbox"/> messenger_messages	★ Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_unicode_ci	64,0 kio	-
<input type="checkbox"/> playlist	★ Parcourir Structure Rechercher Insérer Vider Supprimer	27	InnoDB	utf8mb4_unicode_ci	16,0 kio	-
<input type="checkbox"/> user	★ Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_unicode_ci	32,0 kio	-
7 tables	Somme	557	MyISAM	latin1_swedish_ci	352,0 kio	0 o

⬅ ☐ Tout cocher Avec la sélection : ▼



Tâche 2 - gérer la sauvegarde et la restauration de la BDD

Temps estimé : 1h

Temps réel : 1h

Pour protéger les données de l'application, il est possible de faire appel à une sauvegarde de la base de données.

La sauvegarde de la base de données sera donc fait de manière journalière,

Tâche 3 – Mettre en place le déploiement continu

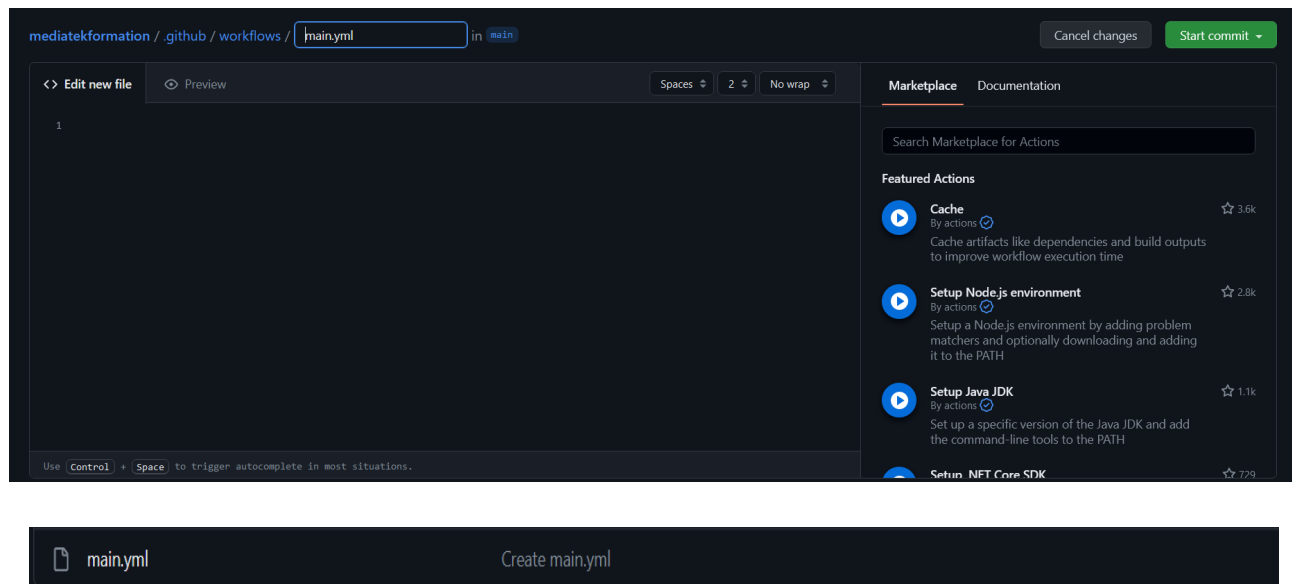
Temps estimé : 1h

Temps réel : 1h

Maintenant que le site est déployé, nous pouvons configurer le GitHub afin que les modifications effectuées soient apportées au site hébergé.

Pour cela, quelques manipulations sont nécessaires sur le compte GitHub sur lequel le projet se trouve.

Une fois les paramètres entrés, le fichier main.yml est disponible.



Ensuite, le fichier doit être récupéré en local à partir d'un Git pull.

La clé secrète ftp doit être générée sur la page d'action de GitHub afin de faire le lien entre les différentes sources du site web.

Une fois la clé créée, le déploiement continu peut se faire à partir de chaque push et le site peut rester à jour de la même manière que pour l'application locale.

Bilan :

Finalement, ces missions ont permis de mettre en place de nouvelles fonctionnalités qui élargissent les possibilités offertes par l'application Mediatekformation avec de nouveaux ajouts que ce soit sur l'interface utilisateur ou sur le back-office.

Ces ajouts comportent des modifications sur tous les aspects de l'application, que ce soit sur le model, la vue, le controller ou même les fichiers de paramétrages qui servent également lors de la liaison à la base de données ou à la publication du site en ligne.

L'apprentissage nécessaire pour cette activité peut être considéré comme complet dans le sens où il reprend tous les aspects de la mise en ligne d'une application Symfony depuis la modification de l'interface jusqu'à l'hébergement en ligne.