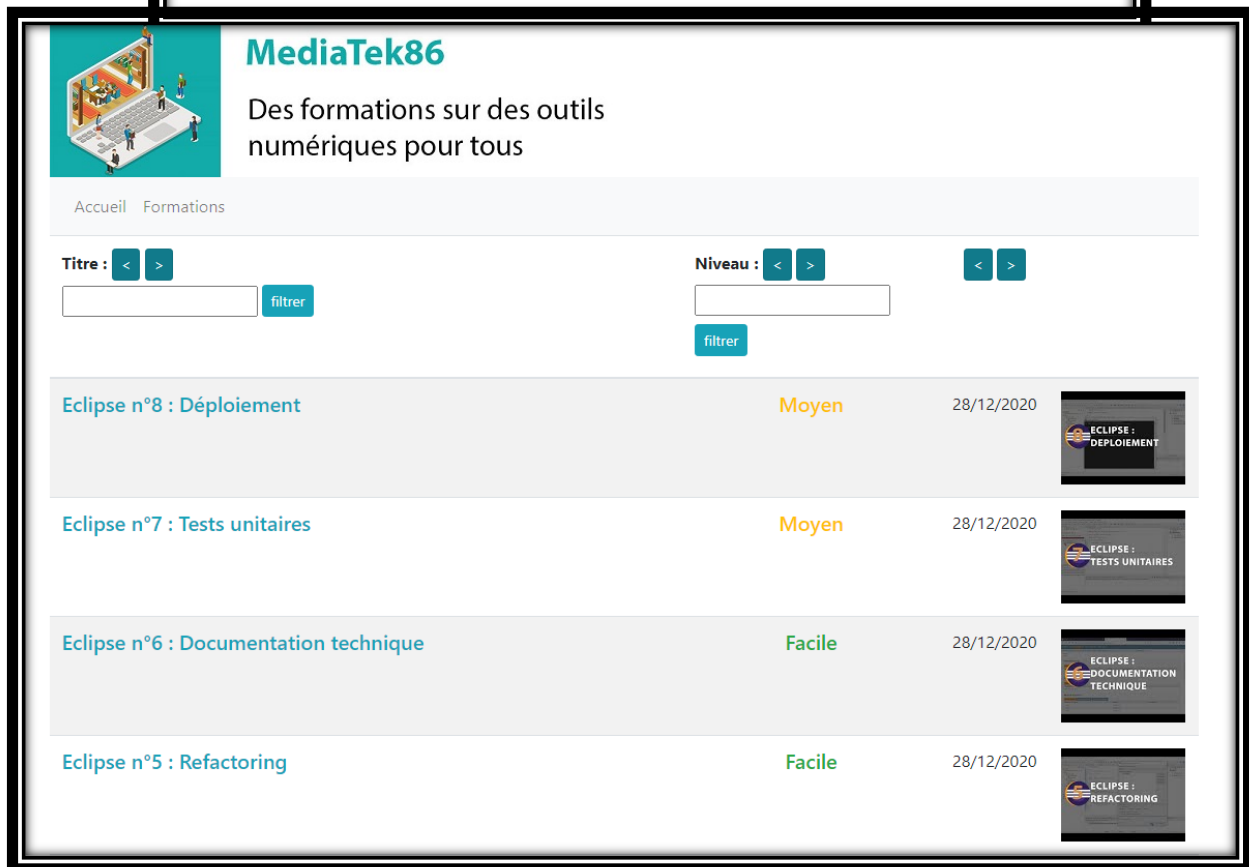


Axel STANKIEWICZ

BTS SIO 2nd année Option SLAM

Compte rendu de projet : Faire évoluer une application web (Symfony) exploitant une base de données relationnelle MySQL



Sommaire :

Rappel du contexte :	3
Rappel de la mission à effectuer :	3
Mission 0 : préparer l'environnement de travail :	4
Mission 1 : ajouter les niveaux :	6
Mission 2 : coder la partie back-office :	12
Mission 3 : créer une vidéo de démonstration d'utilisation du site :	22
Mission bilan : gérer le déploiement, rédiger le compte rendu, créer la page du portfolio dédiée à cet atelier :	22
Bilan final :	22
La liste des compétences officielles couvertes :	23



Rappel du contexte :

Afin de développer le projet dans un contexte réaliste d'entreprise, on nous propose le contexte MediaTek86, un réseau qui gère les médiathèques de la Vienne, et qui a pour rôle de fédérer les prêts de livres, DVD et CD et de développer la médiathèque numérique pour l'ensemble des médiathèques du département.

Nous allons travailler en tant que technicien développeur junior pour l'ESN InfoTech Services 86 qui vient de remporter le marché pour différentes interventions au sein du réseau MediaTek86, dont certaines dans le domaine du développement d'application.

Rappel de la mission à effectuer :

Il nous a été confié l'évolution de l'application web (Symfony) exploitant une base de données relationnelle MySQL, pour mettre à disposition et gérer les auto-formations en ligne proposées par MediaTek86.

Ce projet se divise en 4 parties :

- Mission 0 : préparer l'environnement de travail
- Mission 1 : ajouter les niveaux
- Mission 2 : coder la partie back-office
- Mission 3 : créer une vidéo de démonstration d'utilisation du site

Mission 0 : préparer l'environnement de travail :

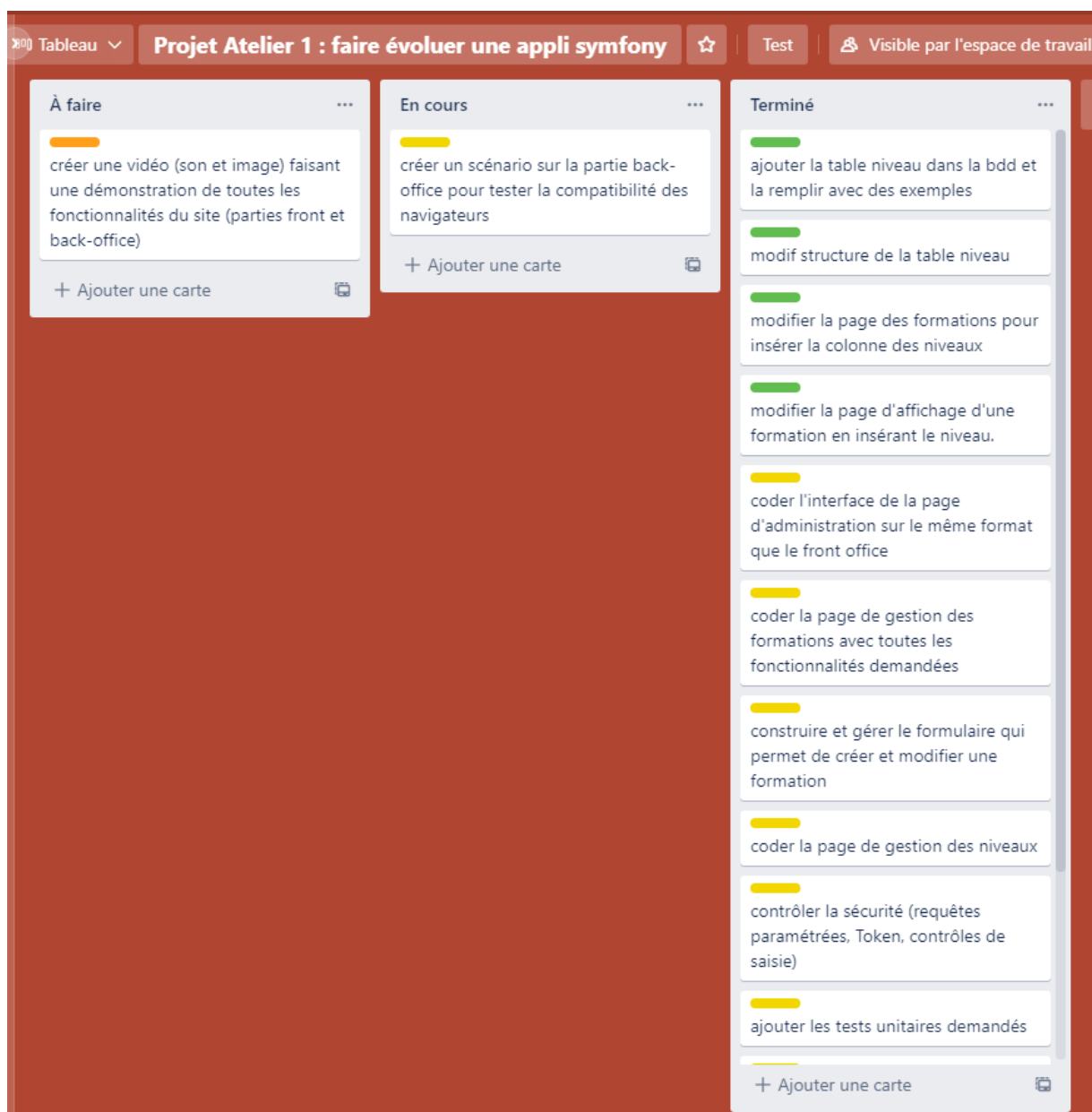
Nous allons travailler avec WampServer et NetBeans, après avoir [récupéré le dossier documentaire](#) qui va nous aider tout au long du projet, il a fallu récupérer le code source de l'application sur GitHub.

Ensuite on a dû s'occuper de la base de données 'mediatekformations' en [lançant le script SQL](#) fournis qui va la créer et la remplir.

Pour vérifier que tout fonctionne bien il a fallu [tester le projet](#) en local.

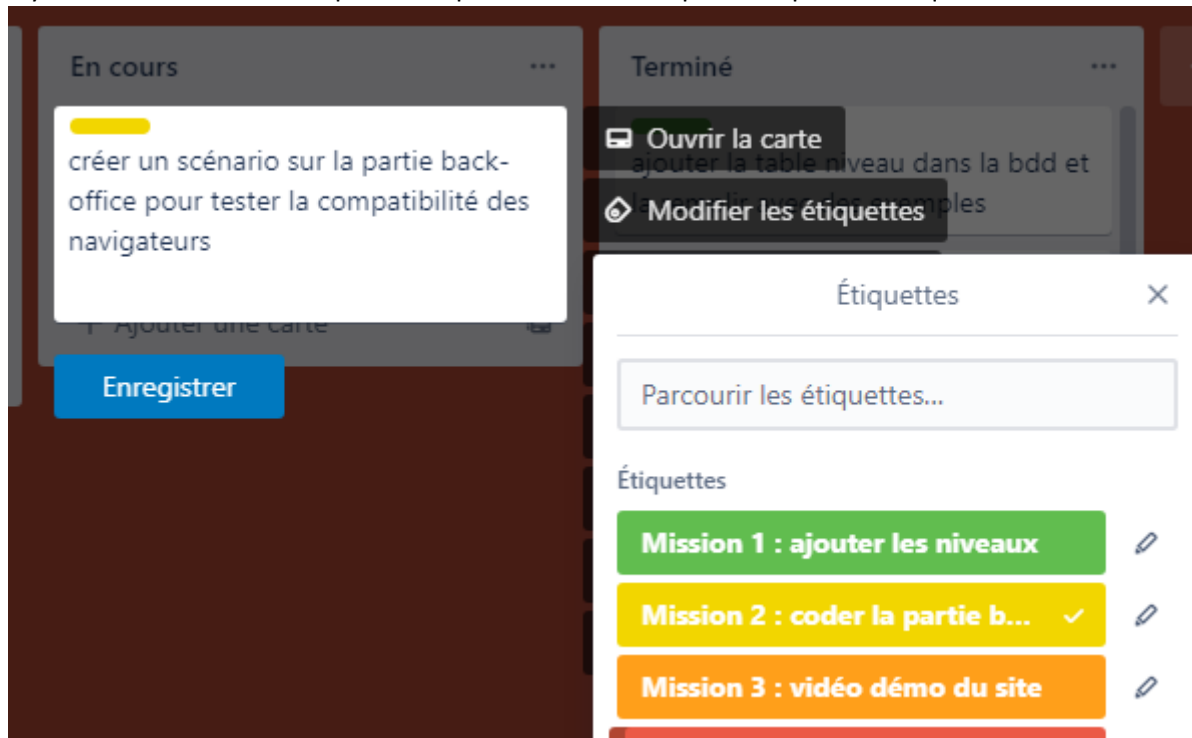
Le dépôt de ce projet est disponible sur ce lien [GitHub](#) :
<https://github.com/SHOOTSTV/mediatekformation>

Enfin, nous avons mis en place [un Trello](#) pour avoir un suivi de projet :



Il est facile à comprendre, avec 3 colonnes pour les tâches à faire, celles en cours et celles qui sont terminées.

Il y a aussi un code couleur pour se repérer à la mission qui correspond à chaque tâche.



A ce stade, tout est préparé pour commencer à travailler sur les évolutions de ce projet, avec un suivi et un dépôt configuré. La Mission 0 donc est complétée.

Mission 1 : ajouter les niveaux :

Tout d'abord je précise que cette évolution utilise le framework Symfony version 4.3.11 ainsi que la notion d'ORM (Object Relational Mapping = mapping entre une base de données et les classes métiers de l'application). Ensuite pour gérer les dépendances en PHP on à utiliser l'outil Composer.

Ce Composer a permis de créer la table ainsi que l'entity Niveau, dans l'invite de commande voici :

```
php bin/console make:entity
```

Voici ce qu'elle contient après avoir précisé sa nature :

```
/**
 * @ORM\Id
 * @ORM\GeneratedValue
 * @ORM\Column(type="integer")
 */
private $id;

/**
 * @ORM\Column(type="string", length=100)
 */
private $niveau;

public function getId(): ?int
{
    return $this->id;
}

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

public function setNiveau(string $niveau): self
{
    $this->niveau = $niveau;

    return $this;
}
```

Maintenant que l'entité est créée, il faut générer la table correspondante. Pour éviter des erreurs on utilise l'invite de commande encore une fois :

```
php bin/console make:migration
```

Cette commande va comparer les entity de l'application avec les tables existantes dans la BDD et créer un fichier de migration qui permet de créer les éventuelles tables manquantes.

Après avoir regarder si le contenu du fichier était bon, on valide cette migration avec cette commande :

```
php bin/console doctrine:migrations:migrate
```

Après cela on vérifie bien que la nouvelle table est apparue dans phpMyAdmin.

Ensuite on a rempli cette table avec la **création des niveaux**, nous avons choisi 3 types de difficultés : Facile ; Moyen et Difficile.

Voici la table niveau après injection des niveaux :

Nous avons modifié la structure de la table 'formation' pour ajouter en clé étrangère l'id du niveau à l'aide du composer :

published_at	title	description	miniature	picture	video_id	niveau_id
2020-12-28 22:00:00	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	2

Pour cela il a fallu refaire un make:entity mais cette fois avec le nom de la table à modifier donc « Formation » en ajoutant un champ « niveau » de type relation.

La relation entre les deux tables est une relation de type ManyToOne car une formation ne peut avoir qu'un niveau mais un niveau peut être dans plusieurs formations

Voici les modifications dans l'entity 'Formation' :

```
/**
 * @ORM\ManyToOne(targetEntity=Niveau::class)
 */
private $niveau;
```

De même voici le getter et le setter créé par le composer dans l'entity Formation :

```
public function getNiveau(): ?Niveau
{
    return $this->niveau;
}

public function setNiveau(?Niveau $niveau): self
{
    $this->niveau = $niveau;

    return $this;
}
```

Ensuite il a fallu donner un niveau de difficulté pour chaque formation, voici un extrait :

niveau_id

2

2

1

1


1

3

2

3

Dans la logique de cet ajout, il faut afficher la colonne des niveaux sur la page des formations, voici le résultat sur le site :



MediaTek86

Des formations sur des outils numériques pour tous

Accueil Formations

Titre :

<

>

filtrer

Niveau :






<

>

filtrer

<

>

Eclipse n°8 : Déploiement	Moyen	28/12/2020	
Eclipse n°7 : Tests unitaires	Moyen	28/12/2020	
Eclipse n°6 : Documentation technique	Facile	28/12/2020	
Eclipse n°5 : Refactoring	Facile	28/12/2020	
Eclipse n°4 : WindowBuilder	Facile	04/11/2020	

Pour afficher les niveaux, dans le fichier formations.html.twig, dans la boucle for j'ai ajouté ce code :

```

<h5 class="text-center">
    {% if formation.niveau.niveau == "Difficile" %}
        <div class="text-danger">
            {{ formation.niveau.niveau }}
        </div>
    {% elseif formation.niveau.niveau == "Moyen" %}
        <div class="text-warning">
            {{ formation.niveau.niveau }}
        </div>
    {% else %}
        <div class="text-success">
            {{ formation.niveau.niveau }}
        </div>
    {% endif %}
</h5>

```

Noter une boucle en if pour utiliser le code couleur pour chaque niveau (facile = vert pour exemple) ce qui peut améliorer la compréhension sur le niveau sans avoir à lire chaque ligne.

Pour le **filtrage par un nom de niveau** voici les modifications que nous avons fait :

Dans FormationRepository voici la requête SQL faite

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

Ce qui équivaut à faire par exemple pour une recherche « Facile », cette requête SQL :

```
SELECT * FROM formation f LEFT JOIN niveau n ON f.niveau_id=n.id WHERE niveau = "Facile"
```

Il a fallu un LEFT JOIN pour joindre les deux tables ensemble.

Ensuite dans FormationsController voici ce que nous avons ajouté :

```
/**
 * @Route("/formations/recherche/niveau/{champ}", name="formations.findallbyniveau")
 * @param type $champ
 * @param Request $request
 * @return Response
 */
public function findAllbyNiveau($champ, Request $request): Response{
    if($this->isCsrfTokenValid('filtre_'.$champ, $request->get('_token'))){
        $valeur = $request->get("recherche");
        $formations = $this->repository->findByNiveau($champ, $valeur);
        return $this->render("pages/formations.html.twig", [
            'formations' => $formations
        ]);
    }
    return $this->redirectToRoute("formations");
}
```

Noter la présence d'un **token** dans une optique de sécurisation. On utilise la fonction "csrf_token" de Twig pour générer une clé dans la vue. Ensuite dans le contrôleur on ajoute une vérification si le token est valide ou non avec la méthode "isCsrfTokenValid", elle reçoit en paramètre en premier le nom envoyé en à la méthode "csrf_token" (ici filtre_niveau) et en second la valeur du champ (la clé générée dans la vue présente dans la vue). Si le test échoue alors on nous renvoie sur la page des formations.


Le Contrôleur va envoyer la valeur écrite dans la barre de recherche dans la requête SQL.

Et enfin voici le code dans le fichier formations.html.twig :

```
<th class="text-left align-top" scope="col">
  Niveau :
  <a href="{{ path('formations.sort', {champ:'niveau', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"><</a>
  <a href="{{ path('formations.sort', {champ:'niveau', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true">></a>
  <form class="form-inline mt-1" method="POST" action="{{ path('formations.findallbyniveau', {champ:'niveau'}) }}">
    <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_niveau') }}">
    <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
  </form>
</th>
```

Il va permettre d'afficher la barre de recherche avec deux boutons de tris. Ici les deux boutons de tris étaient déconseillés mais dans une logique de visiteur débutant nous avons pensé que le fait de trier de manière croissante ou décroissante les formations pourrait être utile, afin de voir le nombre de formations par niveaux de manière globale et non uniquement par recherche.

Enfin, voici **l'affichage d'un niveau** sur la page d'une formation :



28/12/2020

Eclipse n°8 : Déploiement

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

Niveau :

Moyen

A ce stade la base de données a été modifiée en rajoutant les niveaux et l'application aussi en affichant les niveaux dans la liste des formations et sur leur page respective.

Ceci clos la Mission 1.

Mission 2 : coder la partie back-office :

Tout d'abord il a fallu **coder la page d'administration** qui est sur le même type de format que le front office.

Voici le résultat :

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

{% block body %}
<p class="text-right">
  <a href="{{ path('admin.formation.ajout') }}" class="btn btn-primary">
    Ajouter une nouvelle formation
  </a>
</p>
<table class="table table-striped">
  <thead>
    <tr>
      <th>
        Formation
        <a href="{{ path('formations.sort', {champ:'title', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"></a>
        <a href="{{ path('formations.sort', {champ:'title', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"></a>
        <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>
      </th>
      <th>
        Niveau
        <a href="{{ path('formations.sort', {champ:'niveau', ordre:'DESC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"></a>
        <a href="{{ path('formations.sort', {champ:'niveau', ordre:'ASC'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true"></a>
        <form class="form-inline mt-1" method="POST" action="{{ path('formations.findallbyniveau', {champ:'niveau'}) }}">
          <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_niveau') }}">
          <button type="submit" class="btn btn-info mb-2 btn-sm">filtrer</button>
        </form>
      </th>
      <th>
        Date
      </th>
      <th>
        Actions
      </th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>
        <h5 class="text-primary">
          {{ formation.title }}
        </h5>
      </td>
      <td>
        {{ formation.niveau.niveau }}
      </td>
      <td>
        {{ formation.publishedatstring }}
      </td>
      <td>
        <a href="{{ path('admin.formation.edit', {id:formation.id}) }}" class="btn btn-secondary">Editer</a>
        <a href="{{ path('admin.formation.suppr', {id:formation.id}) }}" class="btn btn-danger" class="btn btn-danger" onclick="return confirm('Etes-vous sûr de vouloir supprim
      </td>
    </tr>
  </tbody>
</table>
{% endblock %}
```

[se déconnecter](#)

Gestion des formations

Formations Niveaux				Ajouter une nouvelle formation
Formation	Niveau			
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
Formation	Niveau	Date	Actions	
Eclipse n°8 : Déploiement	Moyen	28/12/2020	Editer	Supprimer
Eclipse n°7 : Tests unitaires	Moyen	28/12/2020	Editer	Supprimer
Eclipse n°6 : Documentation technique	Facile	28/12/2020	Editer	Supprimer
Eclipse n°5 : Refactoring	Facile	28/12/2020	Editer	Supprimer
Eclipse n°4 : WindowBuilder	Facile	04/11/2020	Editer	Supprimer

Il y a aussi la présence des même filtrages et recherches, le code est pratiquement le même :

Pour trier :

```
/**
 * @Route("admin/admin.formationen/tri/{champ}/{ordre}", name="admininformations.sort")
 * @param type $champ
 * @param type $ordre
 * @return Response
 */
public function sortAdmin($champ, $ordre): Response{
    $formations = $this->repository->findAllOrderBy($champ, $ordre);
    return $this->render("admin/admin.formationen.html.twig", [
        'formations' => $formations
    ]);
}
```

Pour la recherche de formations :

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

Pour la recherche de niveaux :

```
/**
 * @Route("admin/admin.formationen/recherche/niveau/{champ}", name="admininformations.findallbyniveau")
 * @param type $champ
 * @param Request $request
 * @return Response
 */
public function findAllbyNiveauAdmin($champ, Request $request): Response{
    if($this->isCsrfTokenValid('filtre_'.$champ, $request->get('_token'))){
        $valeur = $request->get("recherche");
        $formations = $this->repository->findByNiveau($champ, $valeur);
        return $this->render("admin/admin.formationen.html.twig", [
            'formations' => $formations
        ]);
    }
    return $this->redirectToRoute("formations");
}
```

Le code html dans les templates twig sont identiques sauf le nom et le répertoire qui changera.

Nous avons créé un fichier `_admin.formation.form.html.twig` pour afficher et enregistrer les données présentes dans chaque champ du form concernant les formations, et il va être utilisé dans la page édition et ajout :

```
{{ form_start(formformation) }}
<div class="row mt-3">
  <div class="col">
    <div class="row">
      <div class="col">
        {{ form_row(formformation.publishedAt) }}
        {{ form_row(formformation.title) }}
        {{ form_row(formformation.niveau) }}
      </div>
      <div class="col">
        {{ form_row(formformation.miniature) }}
        {{ form_row(formformation.picture) }}
        {{ form_row(formformation.videoId) }}
      </div>
    </div>
  </div>
</div>
<div class="row">
  <div class="col">
    {{ form_row(formformation.description) }}
  </div>
</div>
{{ form_end(formformation) }}
```

Aussi voici le code dans `FormationType` qui contient les différents champs du form

```
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
        ->add('publishedAt', null, [
            'label' => 'Date de création :',
            'required' => true
        ])
        ->add('title', null, [
            'label' => 'Titre :',
            'required' => true
        ])
        ->add('description', null, [
            'label' => 'Description :'
        ])
        ->add('miniature')
        ->add('picture', null, [
            'label' => 'Photo'
        ])
        ->add('videoId')
        ->add('niveau', EntityType::class, [
            'class' => Niveau::class,
            'choice_label' => 'niveau',
            'multiple' => false,
            'required' => true,
            'label' => 'Niveau :'
        ])
        ->add('submit', SubmitType::class, [
            'label' => 'Enregistrer'
        ])
    ];
}

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

On notifie que les champs « `publishedAt` » « `title` » et « `niveau` » sont requis pour obliger de les renseigner avant d'enregistrer la saisie.

Pour la fonctionnalité « ajout » voici le code :

Dans admin.formation.ajout.html.twig pour la partie visuelle on a:

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

{% block body %}
    <h2>Nouvelle Formation :</h2>
    {{ include ('_admin.formation.form.html.twig') }}
{% endblock %}
```

Remarquez l'include du fichier _admin.formation.form.html.twig (précisé ci-dessus) qui va afficher directement le form sans avoir besoin de coder plusieurs form pour les parties ajout et modification.

Ensuite voyons le code dans le Controller AdminMediatekformation.php qui va créer la formation

```
/**
 * @Route("/admin/ajout", name="admin.formation.ajout")
 * @param Formation $formation
 * @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,
        'formformation' => $formFormation->createView()
    ]);
}
```

Voici le résultat dans le site de l'option ajout :

Gestion des formations

Formations Niveaux

Nouvelle Formation :

Date de création :

Jan▼1▼2017▼00▼00▼

Miniature

Titre :

Photo

Niveau :

Facile▼

Video id

Description :

Enregistrer

Pour la fonctionnalité « **modification** » voici le code :

Dans la partie visuelle :

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

{% block body %}
    <h2>Détail visite :</h2>
    {{ include ('_admin.formation.form.html.twig') }}
{% endblock %}
```

On voit clairement la ressemblance avec la structure du fichier qui sert à l'ajout d'une formation.

Dans le Controller nous retrouvons cette fonction qui permet d'enregistrer les modifications :

```
/**
 * @Route("/admin/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->om->flush();
        return $this->redirectToRoute('admin.formations');
    }

    return $this->render("admin/admin.formation.edit.html.twig", [
        'formation' => $formation,
        'formformation' => $formFormation->createView()
    ]);
}
```

Voici le résultat dans le site de l'option modification :

SECTION DES FORMATIONS

Formations Niveaux

Détail visite :

Date de création :	Miniature
<div>Dec 28 2020 22 00</div>	<div>https://i.ytimg.com/vi/Z4yTTXka958/default.jpg</div>
Titre :	Photo
<div>Eclipse n°8 : Déploiement</div>	<div>https://i.ytimg.com/vi/Z4yTTXka958/sddefault.jpg</div>
Niveau :	Video id
<div>Moyen</div>	<div>Z4yTTXka958</div>
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.</div>	
<div>Enregistrer</div>	

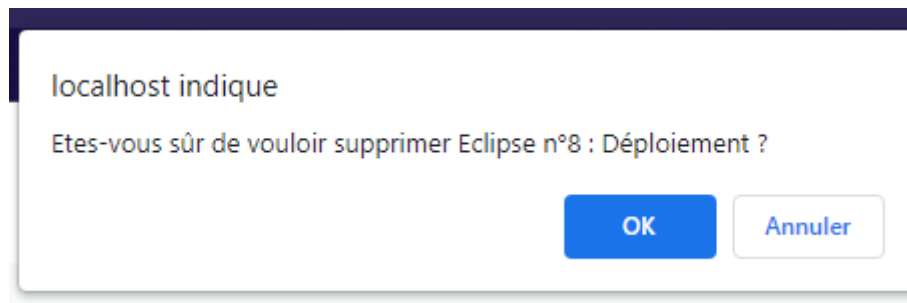
Ici nous avons cliqué sur modifier la formation « Eclipse n°8 » et on voit bien tous les champs préremplis en rapport avec les données de la formation.

Pour la fonctionnalité « **suppression** » voici le code :

Dans le Controller nous récoltons l'id de la formation sélectionnée puis nous la supprimons dans la BDD.

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

Si nous cliquons sur le bouton supprimer un pop-up nous demande de confirmer ou non la suppression de la formation sélectionnée



La partie **gestion des niveaux** reste la même chose sauf qu'ici nous n'allons pas faire de partie édition puisqu'il n'existe qu'un champ donc cela ne serait pas très pertinent.

Voici la partie html.twig :

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

{% block body %}
<form class="form-inline mt-1" method="POST" action="{{ path('admin.niveau.ajout') }}">
<div class="form-group mr-1 mb-2">
<input type="text" class="sm" name="nom">
</div>
<button type="submit" class="btn btn-primary mb-2 btn-sm">Ajouter</button>
</form>
<table class="table table-striped">
<thead>
<tr>
<th>
Niveau
</th>
<th>
Action
</th>
</tr>
</thead>
<tbody>
{% for niveau in niveaux %}
<tr>
<td>
{{ niveau.niveau }}
</td>
<td>
<a href="{{ path('admin.niveau.suppr', {id:niveau.id}) }}"
class="btn btn-danger"
onclick="return confirm('Etes-vous sûr de vouloir supprimer {{ niveau.niveau }} ?')">
Supprimer
</a>
</td>
</tr>
{% endfor %}
</tbody>
</table>
{% endblock %}
```

Voici la vue sur le site :

Gestion des formations

Formations
Niveaux

Ajouter

Niveau	Action
Facile	Supprimer
Moyen	Supprimer
Difficile	Supprimer

Pour la fonctionnalité « ajout » des niveaux voici le code :

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

Pour la fonctionnalité « **suppression** » des niveaux voici le code :

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

Pour la **partie sécurité**, tout d'abord comme précisé plus tôt, nous avons utilisé un token pour sécuriser notre site.

Avec comme précisions dans le html twig sur la partie filtre :

```
<input type="hidden" name="_token" value="{ csrf_token('filtre_niveau') } }">
```

Ainsi que dans le Controller :

```
if($this->isCsrfTokenValid('filtre_'.$champ, $request->get('_token'))){
```

Ensuite en termes de sécurité il est obligatoire de **coder un login pour accéder à la partie admin**.

Après avoir créé le formulaire d'authentification plusieurs fichiers ont été créés par le composer

```
php bin/console make:auth
```

Voici le détail des 4 fichiers :

- SecurityController.php (Controller qui possède deux méthodes login et logout)
- login.html.twig (contient le code du formulaire d'authentification)
- LoginFormAuthenticator (c'est la classe qui va gérer la soumission du formulaire d'authentification)
- security.yaml qui est lui modifié avec l'option « guard » qui précise l'utilisation de LoginFormAuthenticator pour se connecter et se déconnecter .

Voici la vue de la page login quand on essaie d'entrer dans la partie admin :

Veuillez vous connecter

Nom d'utilisateur

Mot de passe

Se connecter

Pour accéder à cette page administrateur il suffit de rajouter un '/admin' à la fin de l'url de la page d'accueil : "mediatek86.go.yj.fr/public/admin".

Pour se déconnecter nous avons ajouter un logout en haut à droite de toute les pages ce qui permet de se déconnecter de n'importe où.

```
<!-- entête -->
<div class="container text-right">
    {% if app.user %}
        <a href="{{ path('app_logout') }}">se déconnecter</a>
    {% endif %}
</div>
{# block top area endblock #}
```

[se déconnecter](#)

Gestion des formations

[Ajouter une nouvelle formation](#)

Ensuite nous avons coder un **test unitaire** qui va tester le fonctionnement de la méthode date de parution au format string :

Dans FormationTest voici le code de testgetPublishedAtString :

```
class FormationTest extends TestCase {

    public function testgetPublishedAtString() {
        $formation = new Formation();
        $formation->setPublishedAt(new DateTime("2022-02-26"));
        $this->assertEquals("26/02/2022", $formation->getPublishedAtString());
    }
}
```

Après le test dans le cmd nous voyons que le test est validé :

```
C:\wamp64\www\mediatekformation>php bin/phpunit
PHPUnit 9.5.16 by Sebastian Bergmann and contributors.

Testing
.
1 / 1 (100%)

Time: 00:00.010, Memory: 8.00 MB

OK (1 test, 1 assertion)
C:\wamp64\www\mediatekformation>
```

Après avoir fait tout cela, nous avons eu à **générer la documentation technique** que voici :

mediatekformation

Search (Press "/" to focus)

Namespaces

DoctrineMigrations
App
Controller
DataFixtures
Entity
Form
Repository
Security
Tests

Container4dnfyqx
ContainerEoj0ixQ
Proxies
__CG__

Composer
Autoload

PackageVersions
Doctrine
Common
DBAL
StaticAnalysis
Deprecations
Foo
Bundle
Inflector

Documentation

Packages

[Application](#)

Namespaces

[DoctrineMigrations](#)
[App](#)
[Container4dnfyqx](#)
[ContainerEoj0ixQ](#)
[Proxies](#)
[Composer](#)
[PackageVersions](#)
[Doctrine](#)
[DeprecationTests](#)
[Egulias](#)
[ProxyManager](#)
[Laminas](#)
[Monolog](#)

Enfin pour finir cette partie nous avons **créé un scénario sur la partie back-office** pour tester la compatibilité des navigateurs :

Project: mediatekformation

Tests +

Search tests...

http://localhost/mediatekformation/public/admin

testbackoffice	Command	Target	Value
1	open	http://localhost/mediatekformation/public/admin	
2	set window size	1806x1020	
3	click	linkText=Ajouter une nouvelle formation	
4	click	id=formation_title	
5	type	id=formation_title	test
6	click	css=body	
7	click	id=formation_submit	
8	click	name=recherche	
9	type	name=recherche	test
10	click	css=tr:nth-child(1) > .form-inline > .btn	
11	click	css=tr:nth-child(6) .btn-secondary	
12	click	css=body	
13	type	id=formation_title	test45
14	click	id=formation_submit	
15	click	name=recherche	
16	type	name=recherche	test
17	choose ok on next confirmation		

Command: open

Target: http://localhost/mediatekformation/public/admin

Value:

Description:

Log Reference

Ce qui clos cette 2^{ème} partie, cette partie majeure vient apporter toute la partie back-office avec toutes les fonctionnalités demandées (ajout modification et suppression pour les formations par exemple) pour les formations et les niveaux. Ainsi que de la sécurité d'application avec le contrôle de saisies, le login et les tokens. Cela m'a permis de développer mes connaissances dans la programmation web sans aucun doute.

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

Une vidéo démonstration des différentes fonctionnalités est disponible à cette adresse :

<https://youtu.be/CTJyGv8YnJc>

L'étape permet d'informer rapidement un utilisateur du fonctionnement de l'application.

Mission bilan : gérer le déploiement, rédiger le compte rendu, créer la page du portfolio dédiée à cet atelier :

Le site est disponible à l'adresse suivante :

<https://mediatek86.go.vj.fr/public/>

Le portfolio avec la page du projet est disponible ici :

<https://axelstzportfolio.000webhostapp.com/index.html>

Il suffit de cliquer sur le bouton « détail du projet » pour avoir tous les liens demandés.

Bilan final :

J'ai beaucoup aimé ce projet parce que j'ai opté pour me concentrer sur le métier de développeur web. Ce qui m'a permis d'en apprendre beaucoup plus sur ce sujet dont sur le backend. Comme évolution possible j'aurais changé l'apparence du site pour le rendre plus moderne.

La liste des compétences officielles couvertes :

B1 :

B1.2 Répondre aux incidents et aux demandes d'assistance et d'évolution

- Traiter des demandes concernant les applications

B1.3 Développer la présence en ligne de l'organisation

- Participer à l'évolution d'un site Web exploitant les données de l'organisation
- Référencer les services en ligne de l'organisation et mesurer leur visibilité

B1.4 : Travailler en mode projet

- Planifier les activités (Trello)

B1.5 Mettre à disposition des utilisateurs un service informatique

- Déployer un service
- Réaliser les tests d'intégration et d'acceptation d'un service

B2 :

B2.1 Concevoir et développer une solution applicative

- Exploiter les ressources du cadre applicatif (framework)
- Identifier, développer, utiliser ou adapter des composants logiciels
- Utiliser des composants d'accès aux données
- Réaliser les tests nécessaires à la validation ou à la mise en production d'éléments adaptés ou développés

- Intégrer en continu les versions d'une solution applicative
- Exploiter les fonctionnalités d'un environnement de développement et de tests

B2.2 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.

B2.3 Gérer les données

- Exploiter des données à l'aide d'un langage de requêtes.
- Concevoir ou adapter une base de données.

B3 :

B3.3 Sécuriser les équipements et les usages des utilisateurs

- Gérer les accès et les privilèges appropriés.

B3.5 SLAM - 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