

COMPTE RENDU D'ACTIVITE D'ATELIER PROFESSIONNEL

Le cas MediaTekFormation

Symfony

**Tout le nécessaire pour se connecter aux différents outils sont fournis dans le fichier « LISEZ-MOI »
en racine du projet disponible sur GitHub**

SOMMAIRE

COMPTE RENDU D'ACTIVITE D'ATELIER PROFESSIONNEL	1
Contexte	3
Environnement de production	3
MISSION 1 :AJOUT DES NIVEAUX	4
Ajoute de l'entité et de la table Niveau	4
Modification des pages Formations.php et formation.php	5
MISSION 2 : CREATION DU BACK OFFICE	9
Création de la vue des formations	9
Création de la vue d'une modification/ajout	11
Création de la vue d'ajout/suppression d'un niveau	14
Sécurité par authentification du backend	16
Tests sur la méthode GetPublishedAtString	18
DEPLOIEMENT	19
La base de données	19
Le site web	20
CONCLUSION	20

Contexte

Créée en 2002, la société InfoTech Services 86 n'a cessé d'enrichir son équipe en nouvelles compétences et expertises afin de proposer à ses clients TPE et PME des solutions toujours plus innovantes.

Son application web proposant ses formations a un besoin d'évolution fourni dans le dossier documentaire. Ces besoins se divisent en 2 missions principales :

- Ajouter le système de niveau de difficulté au sein des formations sur la partie frontend de l'application et offrir la possibilité de faire un tri dessus
- Puis il faut créer la partie backend de l'application afin de pouvoir gérer toutes les données de la base données. Il faut pouvoir ajouter/supprimer et modifier les formations ainsi que les différents niveaux nouvellement créés, tout en sécurisant les accès

Environnement de production

Le développement se faisant sous Symfony, nous retrouverons les langages associés soit : HTML, CSS, PHP, JS et twigs

Pour le développement de l'application, l'IDE Netbeans sera utilisé. Il sera relié à un dépôt distant Github pour assurer le versionnage, régulièrement mit à jour via les commit.

La base de données est gérée sous MySQL avec le SGBDR phpMyAdmin

L'outil de suivis de projet Trello sera également utilisé pour structurer l'avancée du projet.

En ce qui concerne l'hébergement, les services AWS seront utilisés.

MISSION 1 :AJOUT DES NIVEAUX

Ajoute de l'entité et de la table Niveau

Avant toute chose, je crée la nouvelle table dans la base de données avec composer via la console, ce qui crée en même temps l'entité sous Symfony. Je rentre les différents paramètres de cette table, puis j'exécute les différentes étapes de la migration

```
Invite de commandes
C:\Users\quens>cd c:\wamp64\www\mediatekformation

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

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

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

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

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

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

Field length [255]:
> 20

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

updated: src/Entity/Niveau.php

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

Invite de commandes

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

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

Success!

Next: Review the new migration "migrations/Version20220307105544.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html

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

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

[notice] Migrating up to DoctrineMigrations\Version20220307105544
[notice] finished in 157.9ms, used 20M memory, 1 migrations executed, 2 sql queries

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

Ensuite je remplis la table nouvellement créée avec les trois niveaux de difficulté sous phpMyAdmin. À présent je modifie l'entité formation en ajoutant un nouveau champ nommé « idNiveau » qui sera ma clé étrangère des différents niveaux

```
Invite de commandes
c:\wamp64\www\mediatekformation>php bin/console make:entity formation

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

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

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

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

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

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

A new property will also be added to the Niveau class so that you can access the related Formation objects from it.

New field name inside Niveau [formations]:
>

updated: src/Entity/Formation.php
updated: src/Entity/Niveau.php
```

Je remplis la base de données avec les id de niveau aléatoirement pour les tests en créant une fonction stockée dans phpMyAdmin. Je fais un commit and push vers le dépôt distant dans une branche réservé nommé « développement » pour ne pas écraser le « master », tous en mettant à jour le Trello.

Modification des pages Formations.php et formation.php

Comme précisé dans la demande, la page « formations.php » doit être modifiée afin qu'elle affiche le niveau de difficulté entre le titre et la date de parution. Pour ce faire, dans la page formations.html.twig j'ajoute deux nouvelle balises <td> dans la partie <tbody> avec {{formation.idNiveau.niveaudifficulte}}

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

Après quoi, il faut pouvoir trier sur ces différents niveaux, le choix s'est porté sur trois boutons pour représenter les trois niveaux (débutant, confirmé, expert).


Dans le tableau (celui servant de zone pour les champs de recherche et les boutons) j'ajoute trois boutons qui vont chacun appeler la méthode de path : « formations.niveau » en envoyant le nom du niveau.

```
<th class="form-group mr-1 mb-2" scope="col">
  <a href="{{ path('formations.niveau', {valeur:'Débutant'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true">Débutant</a>
  <a href="{{ path('formations.niveau', {valeur:'Confirmé'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true">Confirmé</a>
  <a href="{{ path('formations.niveau', {valeur:'Expert'}) }}" class="btn btn-info btn-sm active" role="button" aria-pressed="true">Expert</a>
</th>
```

Cette méthode se trouve dans formationController. Elle commence par récupérer toutes les formations par le même principe que lors d'un tri croissant ou décroissant sur le titre, sauf que cette fois c'est valeur sont en « dur » puisque le traitement se concentre sur le test derrière. Un foreach va parcourir toutes les formations récupérées puis comparer la valeur de leur niveau avec le paramètre envoyé en paramètre pour les isoler dans un autre tableau qui sera retourné à la vue.

```
57  /**
58   * @Route("/formations/triNiveau/{valeur}", name="formations.niveau")
59   * @param type $valeur
60   * @return Response
61   */
62  public function sortNiveau($valeur): Response{
63      $formations = $this->repository->findAllOrderBy('title', 'DESC');
64      $formation = new Formation();
65      $lesFormations = array();
66      foreach($formations as $formation){
67          if($formation->getIdNiveau()->getNiveauDifficulte() == $valeur){
68              array_push($lesFormations, $formation);
69          };
70      };
71      return $this->render(self::PAGEFORMATIONS, [
72          'formations' => $lesFormations
73      ]);
74  }
75
```

Le résultat de la vue après les modifications



MediaTek86

Des formations sur des outils numériques pour tous

[Accueil](#) [Formations](#)

titre < >




filtrer

Débutant

Confirmé

Expert

< >

UML : Diagramme d'activité	Expert	24/09/2020	
TP Android n°7 : persistance par sérialisation	Expert	25/01/2018	
TP Android n°3 : construction de l'interface	Expert	16/01/2018	

Il faut afficher aussi le niveau dans le détail d'une formation entre le titre et la description. Dans la page formation.html.twig on ajouter la même ligne de code que précédemment.

```
<div class="col">
  <h5> {{ formation.publishedatstring }}</h5>
  <h3 class="text-info mt-5">{{ formation.title }}</h3>
  <p>{{ formation.idNiveau.niveaudifficulte }}</p>
  <p>&nbsp;</p>
  <p><strong>description :</strong></p>
  <p>
    {{ formation.description|nl2br }}
  </p>
</div>
```

Voici la vue après cet ajout



MediaTek86

Des formations sur des outils numériques pour tous

[Accueil](#) [Formations](#)



24/09/2020

UML : Diagramme d'activité

Expert

description :
Présentation du tableau descriptif d'un cas d'utilisation et la représentation graphique de son scénario avec un diagramme d'activité.

Encore une fois je mets à jour le Trello et je fais un commit and push vers le dépôt distant Github. La mission 1 est terminée.

The image shows two overlapping windows. On the left is a Trello board titled "Terminé et fonctionnel (première mission sur l'ajout du niveau)". It contains seven task cards, each with a green progress bar at the top and a description of a task. At the bottom of the board is a button that says "+ Ajouter une carte". On the right is a Git commit window titled "mediatekformation - developpement". It shows a commit message being typed: "Ajout de l'affichage du niveau dans le détail d'une formation, ainsi que cet affichage dans les listes des formations avec une fonction de tri par niveau". Below the message, the author and committer are set to "Quentin Stac <Quentin.stac@live.fr>". There is an unchecked checkbox for "Amend Last Commit". Below that, it says "Files to Commit:" followed by a table with columns "File", "Status", "Commit Action", and "Repository Path".

Terminé et fonctionnel (première mission sur l'ajout du niveau)

- créer la nouvelle entité "Niveau" ainsi que la table via composer
- modifier l'entité et la table "formation" via composer pour faire la relation ManyToOne
- remplir la table "niveau"
- affecter aléatoirement des id niveau à l'entité "formation"
- Sans le détail d'une formation, ajouter l'affiche du niveau entre le titre et la description
- dans la liste des formation ajouter l'affiche du niveau pour chaque formation
- Ajouter des bouton de tris pour les différents niveaux

+ Ajouter une carte

mediatekformation - developpement

Commit Message:

Ajout de l'affichage du niveau dans le détail d'une formation, ainsi que cet affichage dans les listes des formations avec une fonction de tri par niveau

Author: Quentin Stac <Quentin.stac@live.fr> Committer: Quentin Stac <Quentin.stac@live.fr>

☐ Amend Last Commit

Files to Commit:

File	Status	Commit Action	Repository Path
------	--------	---------------	-----------------

MISSION 2 : CREATION DU BACK OFFICE

Création de la vue des formations

Dans le dossier templates je crée un nouveau dossier « admin » qui sera réservé aux vues destinées à l'administration. Je crée un nouveau fichier nommé « admin.formations.html.twig ». Pour éviter la répétition de code je crée en parallèle un fichier « baseadmin.html.twig » qui aura la présentation des pages réservées à l'administration, présentation légèrement différente des pages « classique » puisqu'elle permettra cette fois de passer des formations aux différents niveaux.

baseadmin.html.twig

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

{% block title %}{% endblock %}
{% block stylesheets %}{% endblock %}
{% block top %}
    <div class="container">
        <!-- titre -->
        <div class="text-left">
            
        </div>
        <!-- menu -->
        <nav class="navbar navbar-expand-lg navbar-light bg-light">
            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <ul class="navbar-nav mr-auto">
                    <li class="nav-item">
                        <a class="nav-link" href="{{ path('admin.formations') }}">Formations</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ path('admin.niveaux') }}">Niveaux</a>
                    </li>
                </ul>
            </div>
        </nav>
    </div>
{% endblock %}
{% block body %}{% endblock %}
{% block javascripts %}{% endblock %}
```

Dans « admin.formations.html.twig » je récupère l'ensemble des éléments déjà présent des « formation.html.twig » où je change les path pour qu'il fasse référence aux nouvelles méthodes du nouveau controller que je vais créer juste après. Je rajoute également une ligne au tableau où j'insère deux boutons : Editer et Supprimer. Ces deux boutons permettront d'effectuer leurs actions sur la formation dans laquelle ils seront présents. J'ajuste également la structure de la page pour une meilleure harmonie.

```
<tbody>
    {% for formation in formations %}
        <tr>
            <td>
                <h5 class="text-info">
                    {{ formation.title }}
                </h5>
            </td>
            <td class="text-center">
                {{ formation.idNiveau.niveaudifficulte }}
            </td>
            <td class="text-center">
                {{ formation.publishedatstring }}
            </td>
            <td class="text-center">
                {% if formation.miniature %}
                    <a href="{{ path('formations.showone', {id:formation.id}) }}">
                        
                    </a>
                {% endif %}
            </td>
            <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"
                    onclick="return confirm('Etes vous sûr de vouloir supprimer la formation {{formation.title}} ?')">Supprimer</a>
            </td>
        </tr>
    {% endfor %}
</tbody>
</table>
{% endblock %}
```

À présent, dans le dossier controller, je crée un nouveau dossier « admin » qui accueillera l'ensemble des controller réservé à l'administration dans lequel je crée une nouvelle classe php « AdminFormationController.php ». Je récupère dans cette classe la quasi-entièreté du « FormationController » en changeant toutes les routes et les path des méthodes pour qu'elles correspondent aux path précédemment renseigné dans la vue. Afin de pouvoir gérer les différents traitements des boutons, je crée deux nouvelles méthodes, une pour la suppression et une autre pour la modification. Pour ces traitements, la création d'un objet de type EntityManagerInterface est nécessaire, ce dernier est valorisé directement dans le constructeur. Notez que dans la méthode « edit », un objet de type Request est attendu dans la signature, cet objet va permettre, en cas de validation du formulaire de modification (que l'on va créer juste après), d'apporter également les modifications dans la base de données.

```
/**
 * @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');
}

/**
 * @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()
    ]);
}
```

Enfin nous pouvons tester la vue admin, voici le résultat




MediaTek86

Des formations sur des outils numériques pour tous

Formations Niveaux

titre < >
 filtrer

Débutant < >
Confirmé Expert

Titre	Niveau	Date	Miniature	Actions
UML : Diagramme de paquetages	Débutant	01/11/2020		Editer Supprimer
UML : Diagramme de classes	Confirmé	30/10/2020		Editer Supprimer
UML : Diagramme de cas d'utilisation	Confirmé	24/09/2020		Editer Supprimer

Création de la vue d'une modification/ajout

Pour gérer la modification, on va utiliser les formulaires de Symfony qui peuvent être créés sur une entité. Dans doctrine, je crée le formulaire « FormationType » sur l'entité Formation. Dans le fichier créé dans le dossier « Form », j'ajoute de quoi pouvoir envoyer le formulaire avec un submit et surtout je modifie le add sur idNiveau pour que celui-ci s'affiche correctement en proposant un combobox lors de la modification.

```
class FormationType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('publishedAt', null, ['label' => 'Date de publication'])
            ->add('title')
            ->add('description')
            ->add('miniature')
            ->add('picture')
            ->add('videoId')
            ->add('idNiveau', EntityType::class, [
                'class' => Niveau::class,
                'choice_label' => 'niveau_difficulte',
                'required' => true
            ])
            ->add('submit', SubmitType::class, ['label' => 'Enregistrer'])
    }
}
```

Maintenant, je crée une nouvelle vue « admin.formation.edit.html.twig » qui sera notre formulaire de modification. Dans cette vue, j'utilise les outils fournis par Symfony pour l'appel d'un formulaire. Afin d'avoir une jolie présentation, j'utilise le bootstrap déjà présent sous symfony : bootstrap_4_layout.html.twig que je place dans le fichier « twig.yaml ».

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

{% block body %}

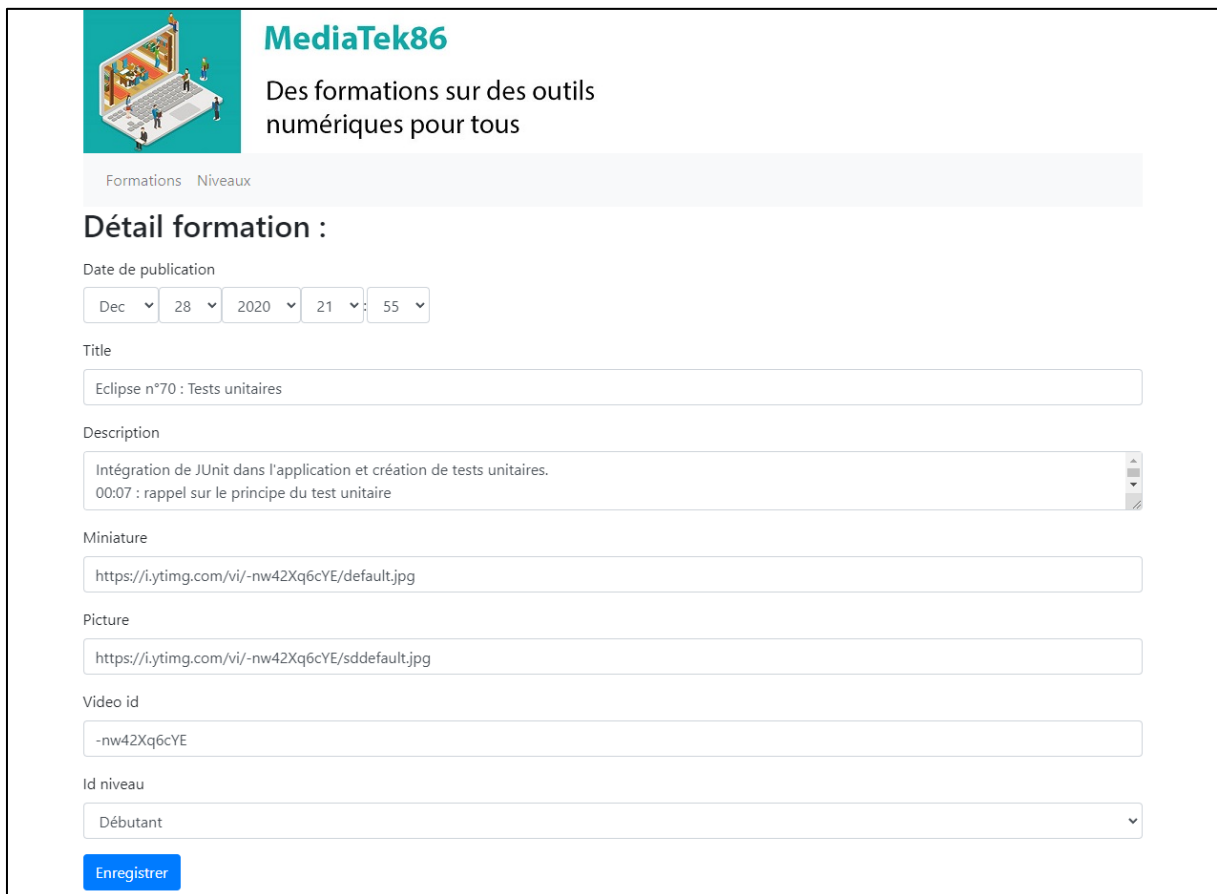
    <h2>Détail formation :</h2>

    {{ form_start(formformation) }}

    {{ form_end(formformation) }}

{% endblock %}
```

Voici le résultat de la vue



MediaTek86

Des formations sur des outils numériques pour tous

Formations Niveaux

Détail formation :

Date de publication

Dec 28 2020 21 : 55

Title

Eclipse n°70 : Tests unitaires

Description

Intégration de JUnit dans l'application et création de tests unitaires.
00:07 : rappel sur le principe du test unitaire

Miniature

https://i.ytimg.com/vi/-nw42Xq6cYE/default.jpg

Picture

https://i.ytimg.com/vi/-nw42Xq6cYE/sddefault.jpg

Video id

-nw42Xq6cYE

Id niveau

Débutant

Enregistrer

Pour enregistrer une nouvelle formationn je crée d'abord une nouvelle vue « admin.formation.ajout.html.twig » dans laquelle je mets le même code que pour la vue d'une modification. Puis, dans le controller « adminFormationController.php » je crée une nouvelle méthode « ajout » très proche de « edit » en terme de code. Dans cette nouvelle méthode, la méthode persiste est utilisé pour envoyer la nouvelle formation à la base de données.


```

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

    $formFormation->handleRequest($request);
    if($formFormation->isSubmitted() && $formFormation->isValid()){
        $this->om->persist($formation);
        $this->om->flush();
        return $this->redirectToRoute('admin.formations');
    }
    return $this->render("admin/admin.formation.ajout.html.twig", [
        'formation' => $formation,
        'formformation' => $formFormation->createView()
    ]);
}

```

Il ne reste plus qu'à ajouter dans « admin.formations.html.twig » un bouton pour ajouter une nouvelle formation. Voici le résultat sur la vue



MediaTek86

Des formations sur des outils numériques pour tous

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

titre

< >

filtrer


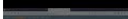
Débutant


Confirmé

Expert

< >

Ajouter une nouvelle visite

Titre	Niveau	Date	Miniature	Actions
Eclipse n°70 : Tests unitaires	Débutant	28/12/2020		<div>Editer</div> <div>Supprimer</div>
Eclipse n°6 : Documentation technique	Expert	28/12/2020		<div>Editer</div>



MediaTek86
Des formations sur des outils numériques pour tous

Formations Niveaux

Nouvelle formation :

Date de publication

▼

▼

▼

▼

▼

Title

Description

Miniature

Picture

Video id

Id niveau

Débutant▼

Enregistrer

Après vérification que tout fonctionne, je fais un commit and push vers Github.

Création de la vue d'ajout/suppression d'un niveau

Il faut commencer par créer le controller admin de la vue des niveaux « adminNiveauxController ». Ce controller est basé sur la même logique que celui des formations. Je crée tout de suite les deux méthodes qui me seront utiles par la suite, celle sur la suppression et celle sur l'ajout. Ces deux méthodes sont très similaire à celle de la formation.

```

/**
 * @Route("/admin/niveau/suppr/{id}", name="admin.niveau.suppr")
 * @param Niveau $niveau
 * @return Response
 */
public function suppr(Niveau $niveau): Response{
    if($niveau->getFormations()->count() == 0){
        $this->om->remove($niveau);
        $this->om->flush();
    }
    return $this->redirectToRoute('admin.niveaux');
}

/**
 * @Route("/admin/niveaux/ajout", name="admin.niveau.ajout")
 * @param Request $request
 * @return Response
 */
public function ajoutNiveau(Request $request): Response{
    if($this->isCsrfTokenValid('token_ajouter', $request->get('_token'))){
        if($request != ""){
            $niveau = new Niveau();
            $valeur = $request->get("ajouter");
            $niveau->setNiveauDifficulte($valeur);
            $this->om->persist($niveau);
            $this->om->flush();
        }
    }
    return $this->redirectToRoute("admin.niveaux");
}

```

Je continue en créant la vue « admin.niveaux.html.twig » dans lequel je reprends le même principe que la vue des formations, mais où cette fois, je récupère et exploite les niveaux. J'intègre également un token pour le champ de saisie afin d'éviter les injections

```

<th class="text-left align-top" scope="col">
    Ajouter un nouveau niveau
    <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="ajouter">
        </div>
        <input type="hidden" name="_token" value="{{ csrf_token('token_ajouter') }}">
        <button type="submit" class="btn btn-info mb-2 btn-sm">Ajouter</button>
    </form>
    <br>
    Niveau
</th>
<th class="text-center">
    Actions
</th>

```

Voici le résultat sur la vue



MediaTek86

Des formations sur des outils numériques pour tous

Formations Niveaux

Ajouter un nouveau niveau

Ajouter

Niveau	Actions
Débutant	<button>Supprimer</button>
Confirmé	<button>Supprimer</button>
Expert	<button>Supprimer</button>

Sécurité par authentification du backend

Après avoir commit and push, je commence par retourner dans la console pour créer une classe user qui accueille l'administrateur.

```
Invite de commandes

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

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

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

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

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

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).
Does this app need to hash/check user passwords? (yes/no) [yes]:
>

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

Success!
```

Je continue en installant l'outil de Symfony Fixtures puis en créant la classe UserFixture. Je me base sur la documentation de Symfony pour remplir la méthode load du fichier UserFixture qui a été créé automatiquement.


```

namespace App\DataFixtures;

use App\Entity\User;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;

class UserFixture extends Fixture
{
    private $passwordEncoder;

    public function __construct(UserPasswordEncoderInterface $passwordEncoder) {
        $this->passwordEncoder = $passwordEncoder;
    }

    public function load(ObjectManager $manager)
    {
        $user = new User();
        $user->setUsername("admin");
        $user->setPassword($this->passwordEncoder->encodePassword($user, 'admin'));
        $user->setRoles(['ROLE_ADMIN']);
        $manager->persist($user);
        $manager->flush();
    }
}

```

J'exécute le load dans la console puis je contrôle que la table User dans la BDD a bien été remplie avec les informations de l'administrateur. Dans le fichier security.yaml, je décommente la ligne permettant de demander la connexion par administrateur pour l'accès à la partie /admin.

```

access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }

```

Pour créer le formulaire d'authentification, j'utilise les outils fournis par Symfony

```

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

What style of authentication do you want? [Empty authenticator]:
[0] Empty authenticator
[1] Login form authenticator
> 1

The class name of the authenticator to create (e.g. AppCustomAuthenticator):
> LoginFormAuthenticator

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
> SecurityController

Do you want to generate a '/logout' URL? (yes/no) [yes]:
>

created: src/Security/LoginFormAuthenticator.php
updated: config/packages/security.yaml
created: src/Controller/SecurityController.php
created: templates/security/login.html.twig

Success!

Next:
- Customize your new authenticator.

```

Symfony crée une série de fichier qui gère la création du formulaire ainsi que toutes les méthodes nécessaires à la connexion et déconnexion. Nous pouvons tester à nouveau le /admin, voici le résultat

Please sign in

Username

Password

Sign in

Une fois connecté avec les identifiants, on est bien en admin dans la barre de debug contre anonymous par défaut. Également, un /logout dans l'URL permet de se déconnecter sans devoir fermer les fenêtres.

Tests sur la méthode GetPublishedAtString

Comme demandé dans la mission 2, il faut réaliser un test sur cette méthode. Dans la console, j'exécute php bin/phpunit afin de lancer les installations nécessaires. Je crée ensuite dans le dossier « tests » une nouvelle classe « formationTest.php ». Depuis cette classe je crée une méthode « TestGetDateToString » dans laquelle je crée un scénario où je rentre une date au format MySQL, puis je compare la sortie avec la chaîne attendu.

```
<?php

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

namespace App\Tests;

use App\Entity\Formation;
use PHPUnit\Framework\TestCase;

/**
 * Description of FormationTest
 *
 * @author quens
 */
class FormationTest extends TestCase{
    public function TestGetDateToString(){
        $formation = new Formation();
        $formation->setPublishedAt(new \DateTime("2022-03-09"));
        $this->assertEquals("09/03/2022", $formation->getPublishedAtString());
    }
}
```

Dans la console je réexécute la précédente commande qui me donne le résultat du test. Ici il fonctionne comme prévu, la méthode pour retourner la date en string fonctionne correctement.

```
c:\wamp64\www\mediatekformation>php bin/phpunit
PHPUnit 9.5.10 by Sebastian Bergmann and contributors.

Testing
.
1 / 1 (100%)

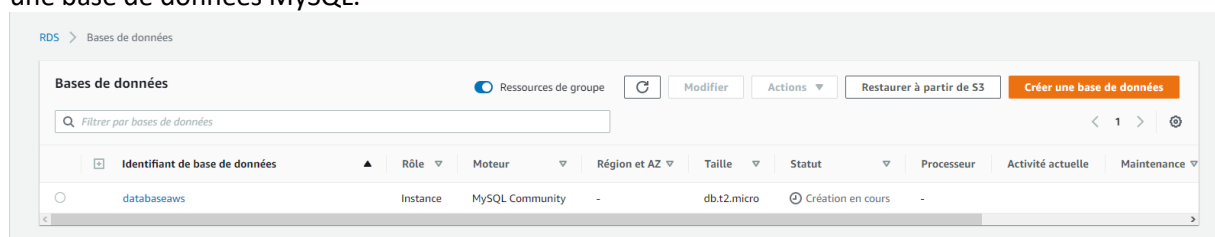
Time: 00:00.016, Memory: 8.00 MB

OK (1 test, 1 assertion)
```

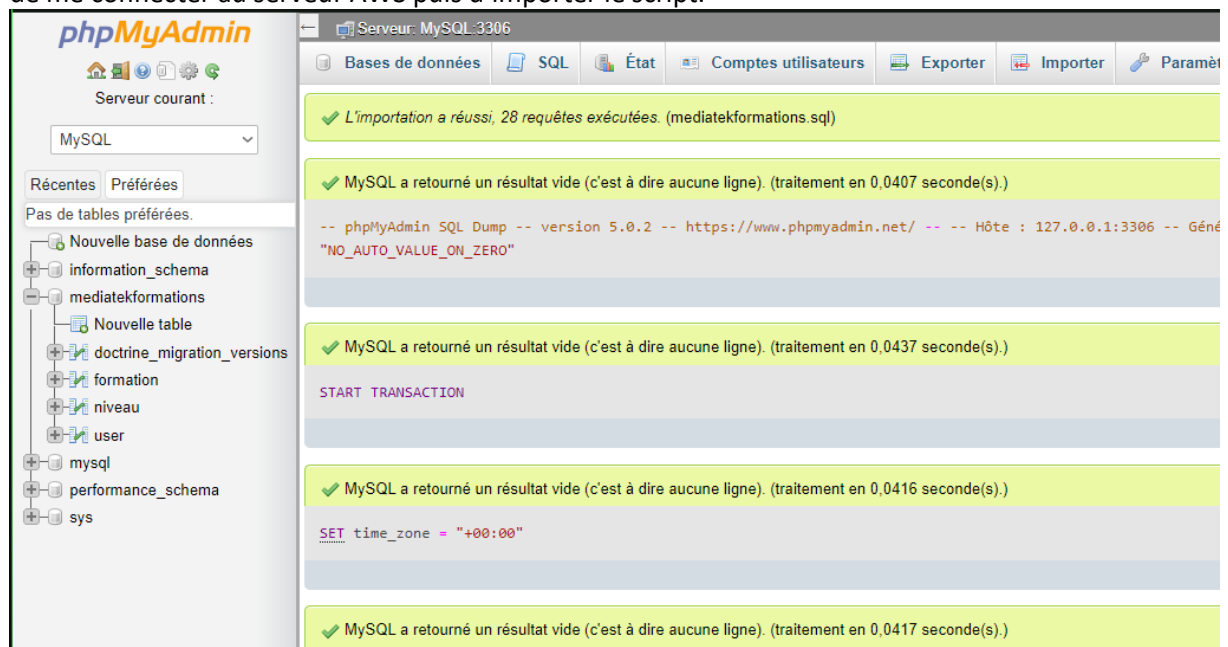
DEPLOIEMENT

La base de données

Pour déployer le site, je vais utiliser les offres d'hébergements AWS d'Amazon. Je commence par créer une base de données MySQL.



Une fois le serveur créé, je récupère le script de la bdd depuis phpMyAdmin (du serveur local) avant de me connecter au serveur AWS puis d'importer le script.



Reste plus qu'à tester la nouvelle connexion à partir du site en modifiant le fichier .env

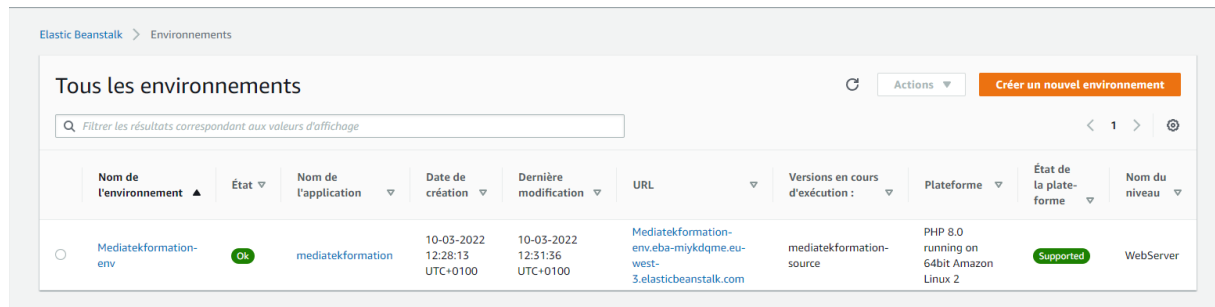
```
#DATABASE_URL="mysql://root:W4F0I9Us0DJy@127.0.0.1:3306/mediatekformations"
DATABASE_URL="mysql://admin:adminaws530@bddawsfree.cil0i3jrrmqo.eu-west-3.rds.amazonaws.com:3306/mediatekformations"
# DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
```

La connexion est réussie.

Le site web

Je commence par créer le fichier .htaccess qui permettra d'avoir une URL plus pratique à l'utilisation et surtout plus jolie. J'utilise les outils fournis par Symfony via la console pour le générer automatiquement. Également je passe la variable APP_env en prod pour ne plus avoir la barre de débogage.

Encore une fois je vais utiliser les offres d'AWS pour l'hébergement. AWS propose un service « Elastic Beanstalk » qui permet d'ajuster automatiquement les ressources en fonction du code source importé. Je zip le site local pour l'importer. L'environnement est très assisté et se crée en quelques minutes avec le l'URL d'accès



The screenshot shows the AWS Elastic Beanstalk console interface. At the top, there's a breadcrumb 'Elastic Beanstalk > Environnements'. Below it, a section titled 'Tous les environnements' contains a search bar and a 'Créer un nouvel environnement' button. A table lists the environments. The table has columns for: Nom de l'environnement, État, Nom de l'application, Date de création, Dernière modification, URL, Versions en cours d'exécution, Plateforme, État de la plateforme, and Nom du niveau. One environment is listed: 'Mediatekformation-env' with a status of 'Ok', application 'mediatekformation', creation date '10-03-2022 12:28:13 UTC+0100', last modification '10-03-2022 12:31:36 UTC+0100', URL 'Mediatekformation-env.eba-miykdqme.eu-west-3.elasticbeanstalk.com', version 'mediatekformation-source', platform 'PHP 8.0 running on 64bit Amazon Linux 2', and level 'WebServer'.

Nom de l'environnement ▲	État ▼	Nom de l'application ▼	Date de création ▼	Dernière modification ▼	URL ▼	Versions en cours d'exécution : ▼	Plateforme ▼	État de la plateforme ▼	Nom du niveau ▼
Mediatekformation-env	Ok	mediatekformation	10-03-2022 12:28:13 UTC+0100	10-03-2022 12:31:36 UTC+0100	Mediatekformation-env.eba-miykdqme.eu-west-3.elasticbeanstalk.com	mediatekformation-source	PHP 8.0 running on 64bit Amazon Linux 2	Supported	WebServer

CONCLUSION

Le site web a bien été modifié selon les critères d'évolution puis mis en ligne avec sa base de données. Le nom de domaine n'est pas encore actif, il le sera dans quelques jours.