Documentation PPE

7 DECEMBRE

EPSI
Créé par : DAVID Rémi

Logo
Nom

Première Partie : Initialisation du site

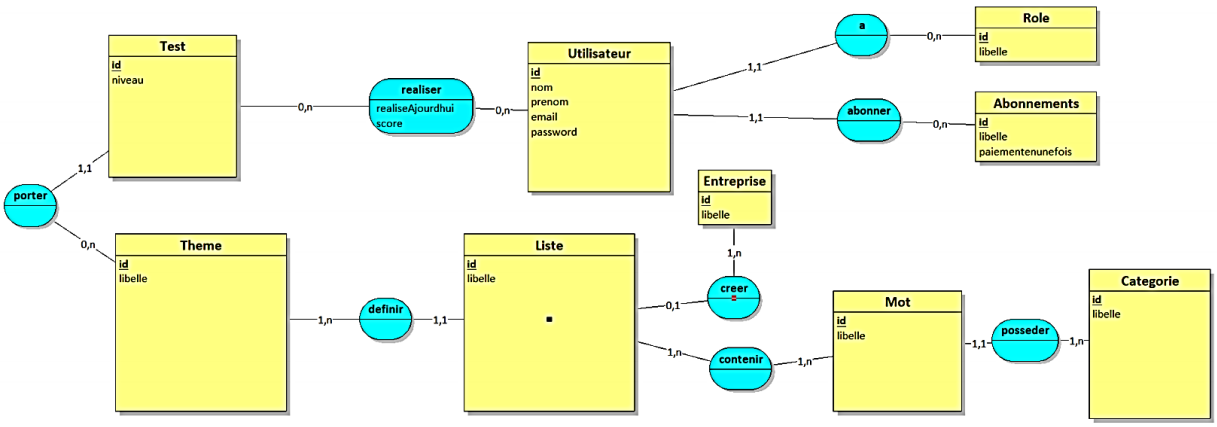
Réalisé par Théo Telliez

Cette partie donc le début du PPE a été réaliser par mon collègue Théo. Pour la liste des commandes qui ont permis de créer le repository sur GitHub et la construction du squelette du site je vous pris de bien vouloir consulter la documentation qu'il a réalisé.

Deuxième Partie : Création de la base de données (BDD)

Réalisé par Théo TELLIEZ et Rémi DAVID

Maintenant que le site a été réaliser nous allons procéder à la création de la base de données.
Pour ce faire nous allons nous aider du MCD que nous avons créé pour cela :



Pour créer la base de donnée, il faut commencer par la configurer dans le fichier .env.local et modifier la ligne suivante :

```
DATABASE_URL=mysql://login4058:vrevgEEejPTIFIC@127.0.0.1:3306/db_InnovAnglais?serverVersion=mariadb-10.3.27
```

Ici db_InnovAnglais correspond au nom que ma bd va porter.
Maintenant que ceci est fait nous pouvons créer notre BD avec la commande :

```
php bin/console doctrine:database:create
```

Avec ceci la Base de données est maintenant créée mais elle est vide pour le moment. Nous allons donc la remplir avec des tables dont voici un exemple de création de table.

Création d'une table :

Pour créer une table ou aussi appelé Entité sous Symfony, nous allons utiliser la commande suivante :

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

Maintenant le terminal va nous demander plusieurs choses.
Premièrement, il va nous demander le nom de l'Entité puis il va nous demander les champs que l'on veut rajouter dans cette Entité (on peut aussi juste créer l'Entité et revenir faire les champs plus tard avec la commande précédente) et pour chaque champ il faudra préciser la nature du champ, sa taille (si besoin) ainsi que si le champ peut être nul ou non.
Voici un exemple pour l'Entité Categorie:

```
php bin/console make:entity
Categorie → nom de l'entité
libelle → nom du champ
string → nature du champ
255 → taille
no → nul ou non
```

```
login4059@symfony4-4059:~/public/InnovAnglais$ php bin/console make:entity

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

created: src/Entity/Categorie.php
created: src/Repository/CategorieRepository.php

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

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

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

Field length [255]:
>

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

updated: src/Entity/Categorie.php

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

Maintenant la table categorie est créée, donc on reprend le même fonctionnement pour les autres tables.
Après avoir créé toutes les entités, il faut maintenant entrer 2 lignes de commandes pour qu'elles soient visibles sur phpMyAdmin :

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

A présent, nous allons faire les liens entre les tables.

Troisième Partie : Créations des liens

Dans la base de données, il va exister 2 types de lien :

- Des ManyToOne ce qui correspond aux relations x,n → x,1 du mcd
- Des ManyToMany ce qui correspond aux relations x,n → x,n du mcd

Pour cela, nous allons entrer des commandes dans le terminal.

Relations ManyToOne :

Pour cela, il va falloir rentrer dans l'entité (celle du coté x,1 dans le mcd) via le terminal avec le make:entity, puis ajouter un nouveau champ (de préférence avec le nom de l'autre table concernée) et dans la nature de ce champ il faudra mettre ManyToOne
Puis il va nous demander de choisir l'entité à lier à celle-ci ainsi que si l'on veut aussi un champ dans l'autre entité et si ce dernier peut-être nul ou non.

Exemple de lien ManyToOne entre role et utilisateur :

php bin/console make:entity

Utilisateur → on rentre dans l'entité Utilisateur
role → on ajoute un champ role
ManyToOne → de type ManyToOne
Role → on le lie à l'entité Role
yes → on accepte que cette dernière puisse être nul
yes → on accepte de créer une variable dans l'autre table
utilisateur → on la nomme ici utilisateur

php bin/console make:migration
php bin/console doctrine:migrations:migrate

Maintenant le lien entre Role et Utilisateur est créé.

Exemple de relation ManyToMany entre liste et mot :

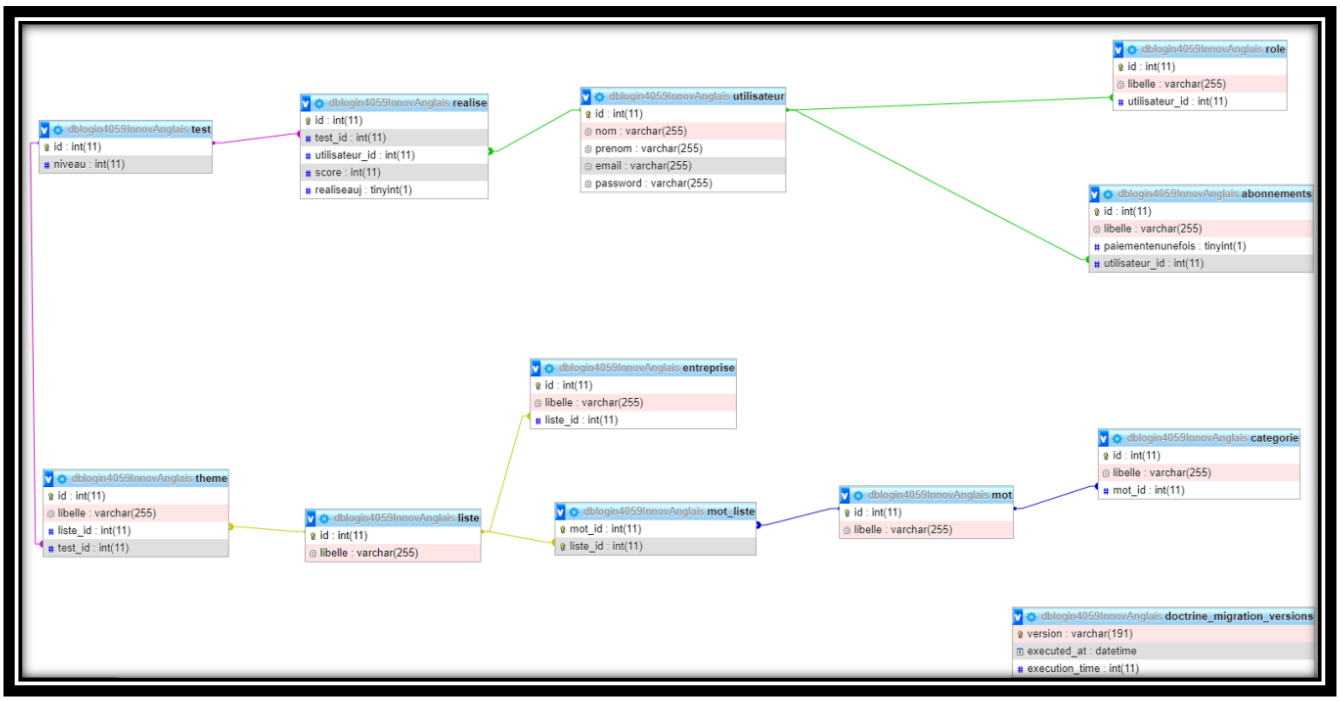
Pour cette relation le fonctionnement est le même sauf que l'on ManyToMany à la place de ManyToOne et cette relation va nous créer une table intermédiaire entre les deux tables concernées.

php bin/console make:entity

Liste
mot
ManyToMany
Mot
yes
liste

php bin/console make:migration
php bin/console doctrine:migrations:migrate

Maintenant les liens sont créés et cela donne cela :



Dernière partie : Création des formulaires thème et mot

Pour créer des formulaires, on va de nouveau passer par le terminal dans un premier temps puis nous passerons à du code « normal » mais avant toute chose nous devons créer les contrôleurs des entités.

Création des contrôleurs :

Pour cela nous allons entrer les commandes suivantes (exemple de thème) :

php bin/console make:controller
Theme → nom du contrôleur qui sera compléter par Controller
Theme → la classe qui est lié au controleur

Créer un contrôleur va aussi créer un dossier dans templates pour les vues (twig) concernant ce contrôleur.

Création des formulaires d'ajout/suppression/modification/liste :

Pour cela nous allons entrer les commandes suivantes (exemple de thème) :

php bin/console make:form
AjoutTheme → on entre le nom du formulaire qui sera compléter par Type
Theme → la classe qui est lié au formulaire

Maintenant le formulaire est créé et nous allons le modifier afin qu'il ait un bouton et cela donne ceci :

```php
<?php

namespace App\Form;

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

class AjoutThemeType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add( child: 'libelle', type: TextType::class)
            ->add( child: 'ajouter', type: SubmitType::class);
        ;
    }

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

Maintenant dans le contrôleur nous allons rajouter une fonction pour gérer cet ajout :

```php
/**
 * @Route("/ajout-theme", name="ajout-theme")
 */
public function ajoutTheme(Request $request)
{
    $theme = new Theme(); // Instanciation d'un objet Theme
    $form = $this->createForm( type: AjoutThemeType::class, $theme); // Création du formulaire pour ajouter un thème, en lui donnant l'instance .
    if ($request->isMethod( method: 'POST')) {
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $em = $this->getDoctrine()->getManager(); // On récupère le gestionnaire des entités
            $em->persist($theme); // Nous enregistrons notre nouveau thème
            $em->flush(); // Nous validons notre ajout
            $this->addFlash( type: 'notice', message: 'Thème inséré'); // Nous préparons le message à afficher à l'utilisateur sur la page où il se rendra
        }
        return $this->redirectToRoute( route: 'ajout-theme'); // Nous redirigeons l'utilisateur sur l'ajout d'un thème après l'insertion.
    }
    return $this->render( view: 'theme/ajout-theme', [
        'form' => $form->createView() // Nous passons le formulaire à la vue
    ]);
}
```

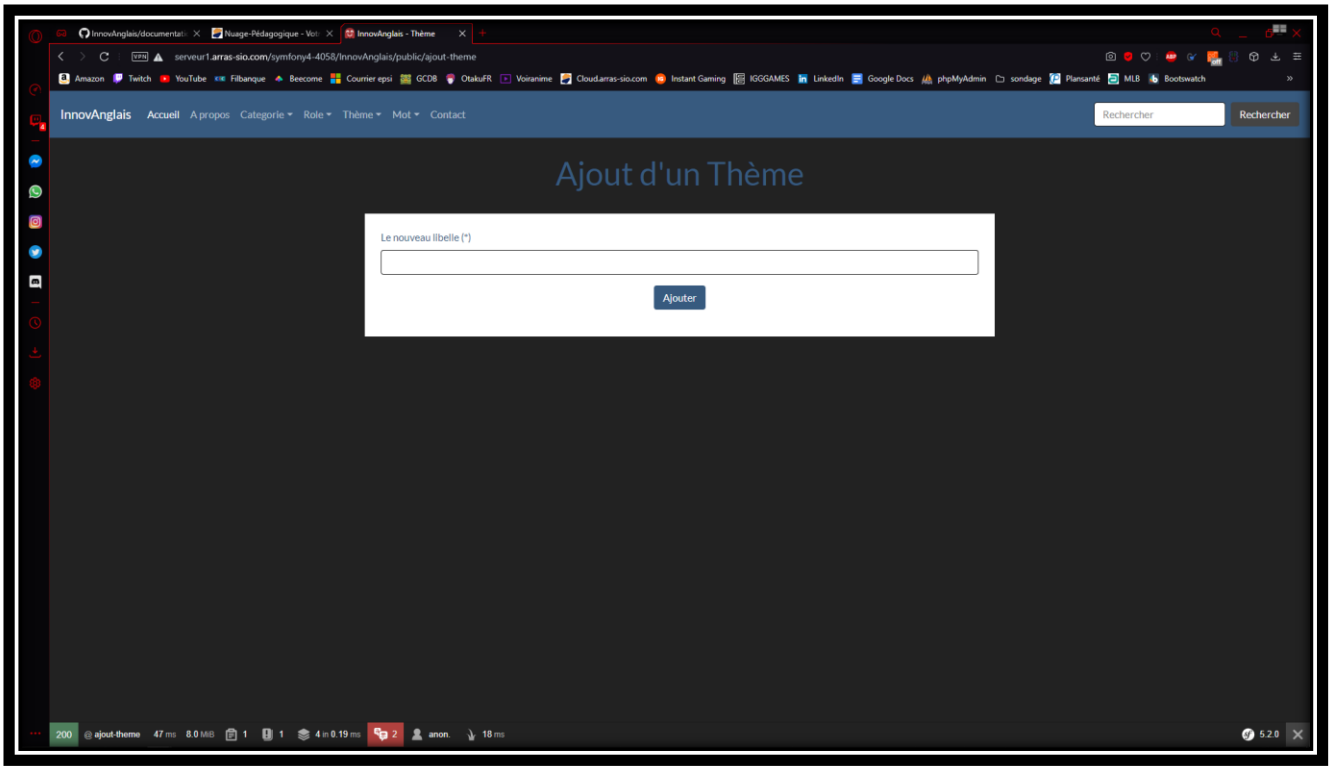
Puis nous allons passer à l'affichage du formulaire dans le twig :

```twig
{% extends 'base.html.twig' %}
{% block title %}{{ parent() }} - Thème {% endblock %}
{% block contenu %}
    <div class="container-fluid">
        <div class="row justify-content-center">
            <h1 class="text-center text-primary p-4">Ajout d'un Thème</h1>
        </div>
        <div class="row justify-content-center">
            <div class="col-6 bg-white p-4 m-0 text-primary">
                {{ form_start(form) }}

                <div class="form-group">
                    {{ form_label(form.libelle, 'Le nouveau libelle (*) ', {'label_attr': {'class': 'font-weightbold'}}) }}
                    {{ form_widget(form.libelle, {'attr': {'class': 'form-control'}}) }}
                </div>

                <div class="form-group mx-auto text-center">
                    {{ form_widget(form.ajouter, { 'label': "Ajouter", 'attr': {'class': 'btn font-weightbold bg-primary text-white mx-auto text-center'}}) }}
                </div>
                {{ form_end(form) }}
            </div>
        </div>
    </div>
{% endblock %}
```

Ce code permet l'affichage du formulaire comme ceci :



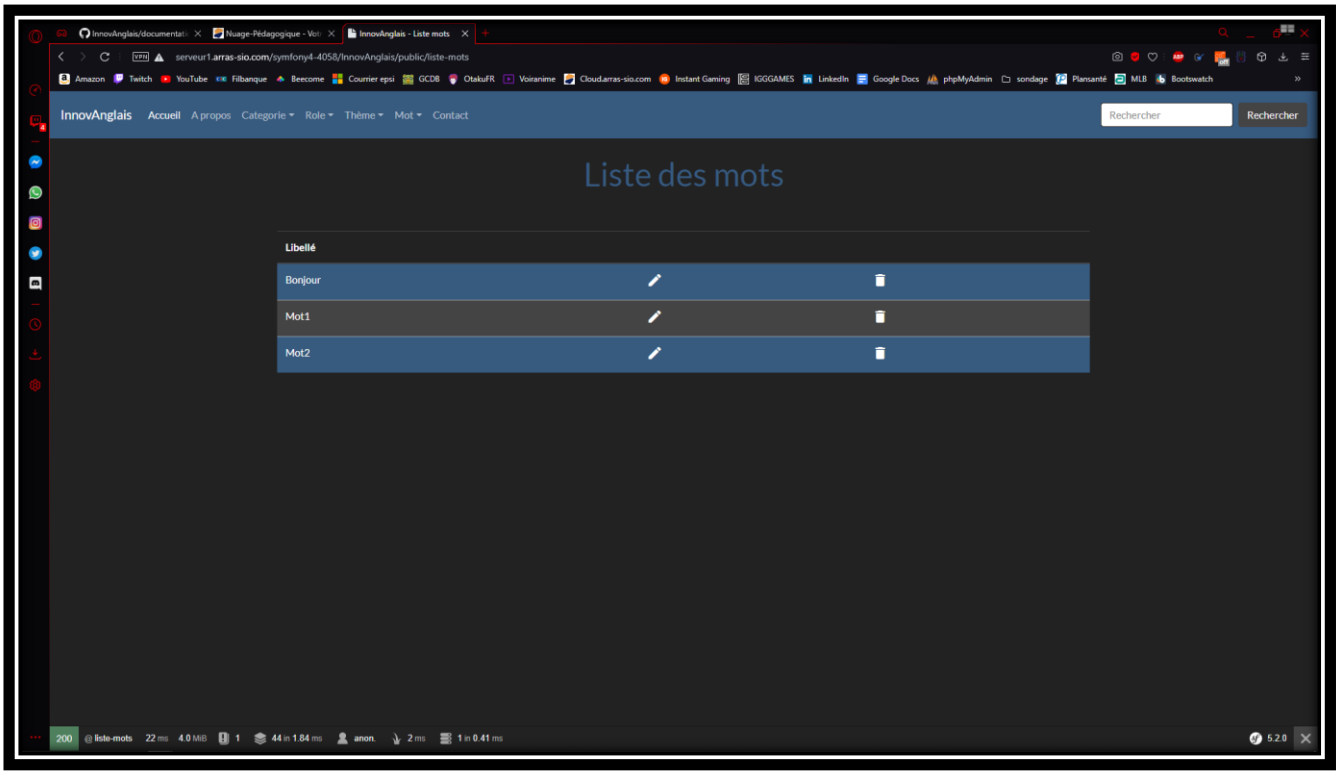
Maintenant, nous allons passer à l'affichage. Pour cela nous allons créer une nouvelle fonction dans le contrôleur qui se nomme listeThemes :

```php
/**
 * @Route("/liste-themes", name="liste-themes")
 */
public function listeThemes(Request $request)
{
    $em = $this->getDoctrine();
    $repoTheme = $em->getRepository(Theme::class);
    if ($request->get( key: 'supp')!=null){
        $theme = $repoTheme->find($request->get( key: 'supp'));
        if($theme!=null){
            $em->getManager()->remove($theme);
            $em->getManager()->flush();
            $this->addFlash( type: 'notice',  message: 'Thème supprimé');
        }
        return $this->redirectToRoute( route: 'liste-themes');
    }
    $themes = $repoTheme->findBy(array(), array('libelle' => 'ASC'));
    return $this->render( view: 'theme/liste-themes', [
        'themes' => $themes // Nous passons la liste des thèmes à la vue
    ]);
}
```

Avec cette fonction nous sommes en mesure de récupérer les données de la table theme de la bdd mais pour les afficher nous avons besoin d'un twig :

```twig
{% extends 'base.html.twig' %}
{% block title %}{{parent()}} - Liste thèmes{% endblock %}
{% block contenu %}
    {{parent()}}
    <div class="container-fluid">
        <div class="row justify-content-center">
            <h1 class="text-center text-primary p-4">Liste des thèmes</h1>
        </div>
        <div class="row justify-content-center">
            <div class="col-8 p-4 m-0 text-primary">
                <div class="table-responsive">
                    <table class="table table-hover">
                        <thead>
                        <tr>
                            <th scope="col">Libellé</th>
                            <th scope="col"></th>
                            <th scope="col"></th>
                        </tr>
                        </thead>
                        <tbody>
                        {% for theme in themes %}
                            <tr class="{{ cycle(['table-primary', 'table-secondary'], loop.index0) }}">
                                <td>{{theme.libelle |capitalize }}</td>
                                <td><a href="{{path('modif-theme',{'id':theme.id})}}"class="text-white">
                                        <span class="material-icons" title="Modifier un theme">create</span></a></td>
                                <td><a href="{{path('liste-themes',{'supp':theme.id})}}" class="text-white">
                                        <span class="material-icons" title="Supprimer le theme">delete</span></a></td>
                            </tr>
                        {% endfor %}
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
{% endblock %}
```

Grâce à cela les données s’affiche sous forme de tableau comme ceci :



Maintenant, nous allons passer à la modifcation.
Pour cela nous allons créer un nouveau formulaire comme précedemment sauf qu’ici il s’appelera ModifThemeType et qui ressemble à cela :

```php
<?php

namespace App\Form;

use ...

class ModifThemeType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add( child: 'libelle')
            ->add( child: 'modifier', type: SubmitType::class)
        ;
    }

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

Maintenant il nous faut créer la fonction et la page de modification :

Fonction :

```php
/**
 * @Route("/modif-theme/{id}", name="modif-theme", requirements={"id"="\d+"})
 */
public function modifTheme(int $id, Request $request)
{
    $em = $this->getDoctrine();
    $repoTheme = $em->getRepository(Theme::class);
    $theme = $repoTheme->find($id);
    if ($theme == null) {
        $this->addFlash( type: 'notice',  message: "Ce thème n'existe pas");
        return $this->redirectToRoute( route: 'liste-themes');
    }
    $form = $this->createForm( type: ModifThemeType::class, $theme);
    if ($request->isMethod( method: 'POST')) {
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $em = $this->getDoctrine()->getManager();
            $em->persist($theme);
            $em->flush();
            $this->addFlash( type: 'notice',  message: 'Thème modifié');
        }
        return $this->redirectToRoute( route: 'liste-themes');
    }
    return $this->render( view: 'theme/modif-theme', [
        'form' => $form->createView()
    ]);
}
```

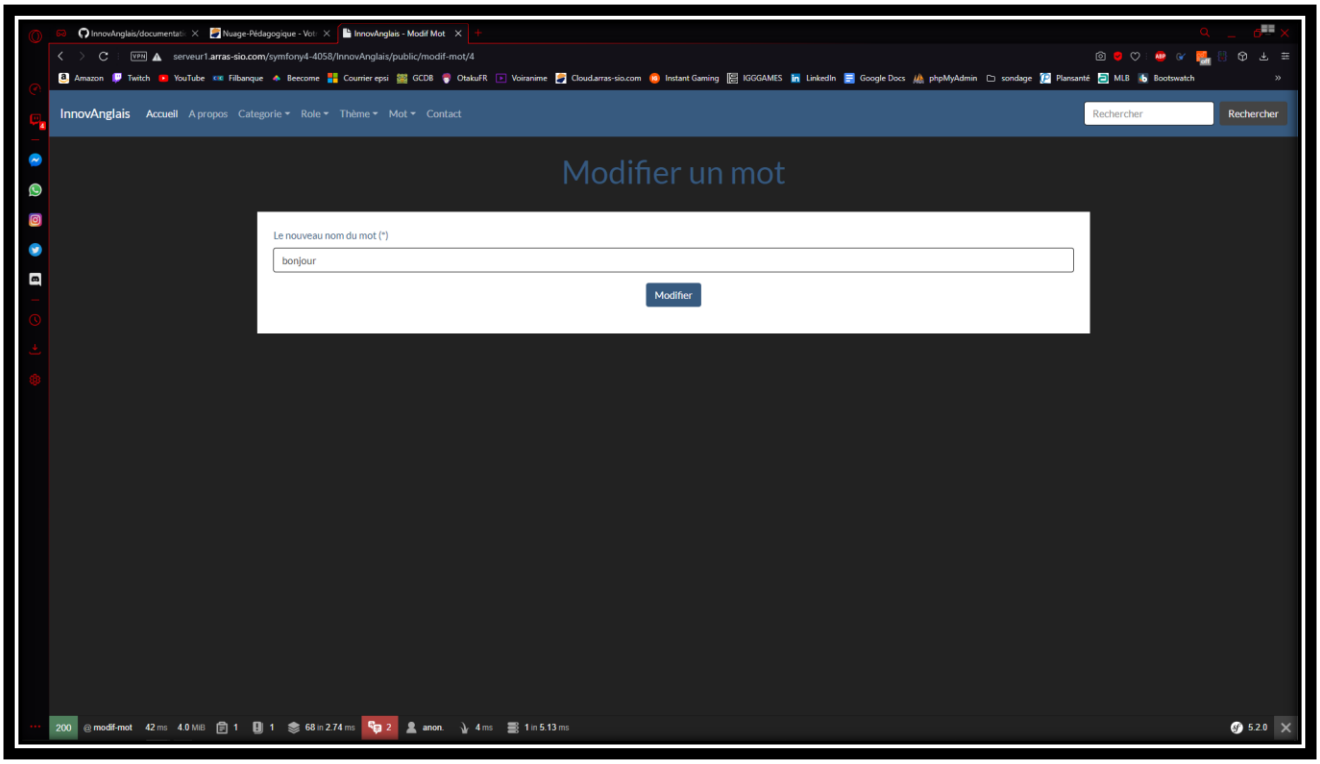
Twig :

```twig
{% extends 'base.html.twig' %}
{% block title %}{{ parent() }} - Modif Thème{% endblock %}
{% block contenu %}
    {{ parent() }}
    <div class="container-fluid">
        <div class="row justify-content-center">
            <h1 class="text-center text-primary p-4">Modifier un thème</h1>
        </div>
        <div class="row justify-content-center">
            <div class="col-8 bg-white p-4 m-0 text-primary">
                {{ form_start(form) }}

                <div class="form-group">
                    {{ form_label(form.libelle, 'Le nouveau nom du thème (*) ', {'label_attr': {'class': 'font-weightbold'}}) }}
                    {{ form_widget(form.libelle, {'attr': {'class': 'form-control'}}) }}
                </div>

                <div class="form-group mx-auto text-center">
                    {{ form_widget(form.modifier, { 'label': "Modifier", 'attr': {'class': 'btn font-weightbold bg-primary text-white mx-auto text-center'}}) }}
                </div>
                {{ form_end(form) }}
            </div>
        </div>
    </div>
{% endblock %}
```

Résultat :



Maintenant que cela est fait nous pouvons passer à la suppression. Cela est plus simple que le reste car nous avons « juste » à modifier la fonction listeThemes dans le controleur que voici :

```php
/**
 * @Route("/liste-themes", name="liste-themes")
 */
public function listeThemes(Request $request)
{
    $em = $this->getDoctrine();
    $repoTheme = $em->getRepository(Theme::class);
    if ($request->get( key: 'supp')!=null){
        $theme = $repoTheme->find($request->get( key: 'supp'));
        if($theme!=null){
            $em->getManager()->remove($theme);
            $em->getManager()->flush();
            $this->addFlash( type: 'notice',  message: 'Thème supprimé');
        }
        return $this->redirectToRoute( route: 'liste-themes');
    }
    $themes = $repoTheme->findBy(array(), array('libelle' => 'ASC'));
    return $this->render( view: 'theme/liste-themes', [
        'themes' => $themes // Nous passons la liste des thèmes à la vue
    ]);
}
```

La partie entouré en rouge correpond à la fonction supprimé.

Liens Utiles

Le projet est disponible sur GitHub à l'adresse suivante :
https://github.com/TheoTelliez/InnovAnglais

Le projet est disponible en ligne sur le serveur de Théo à l'adresse suivante :
http://serveur1.arras-sio.com/symfony4-4059/InnovAnglais/public/accueil