TP4 : Bases de données, ORM Doctrine et Opérations CRUD

Pour réaliser les différentes opération CRUD, on va utiliser dans cet atelier L'ORM Doctrine (ORM : Object Relational Mapping) qui va prendre en charge la correspondance entre les objets PHP et les tables de la BD MySql

1. Ouvrez le fichier .env pour configure les paramètres de la base de données :

DATABASE_URL=mysql://root:@127.0.0.1:3306/symfony?serverVersion=5.7

1. Créer la base de données en tapant dans le terminal :

php bin/console doctrine:database:create

2. Accéder à phpmyadmin et vérifier la création de la base de données en tapant : localhost/phpmyadmin/

Création de l'entité Article

3. Créer l'entité Article (une classe PHP dont les instances seront enregistrées dans la BD) de données en tapant dans le terminal :

php bin/console make:entity Article

Suivez l'assistant de la commande, l'entité aura deux champs (field) :

- Nom : string(255)
- Prix : decimal

Voici le code php de l'entité créée :

```php
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\ArticleRepository")
 */
class Article
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
```

```
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $nom;

    /**
     * @ORM\Column(type="decimal", precision=10, scale=0)
     */
    private $prix;

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

    public function getNom(): ?string
    {
        return $this->nom;
    }

    public function setNom(string $nom): self
    {
        $this->nom = $nom;

        return $this;
    }

    public function getPrix(): ?string
    {
        return $this->prix;
    }

    public function setPrix(string $prix): self
    {
        $this->prix = $prix;

        return $this;
    }
}
```

4. Créer la table Article qui correspond à l'entité Article, en tapant les deux commandes :

php bin/console doctrine:migrations:diff

puis

php bin/console doctrine:migrations:migrate

5. Vérifier la création de la table Article dans la base de données

Création d'une fonction pour ajouter des articles dans la BD

6. Ajouter le code suivant au fichier indexController.php

```
/**
      * @Route("/article/save")
      */
     public function save() {
       $entityManager = $this->getDoctrine()->getManager();

       $article = new Article();
       $article->setNom('Article 1');
       $article->setPrix(1000);

       $entityManager->persist($article);
       $entityManager->flush();

       return new Response('Article enregisté avec id    '.$art
icle->getId());
     }
```

7. Ajouter les use suivants au début du fichier :

```
use App\Entity\Article;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
```

8. Ajouter 3 articles en tapant l'url suivante (changer le nom et le prix de l'article dans le code à chaque fois) : http://127.0.0.1:8000/article/save
9. Vérifier l'ajout des 3 lignes dans la BD

Lire les articles de la BD et les transmettre à la vue

10. Pour lire les articles à partir de la BD et les transmettre comme modèle à la vue **index.html.twig**, modifier le code de la fonction **home**() comme suit :

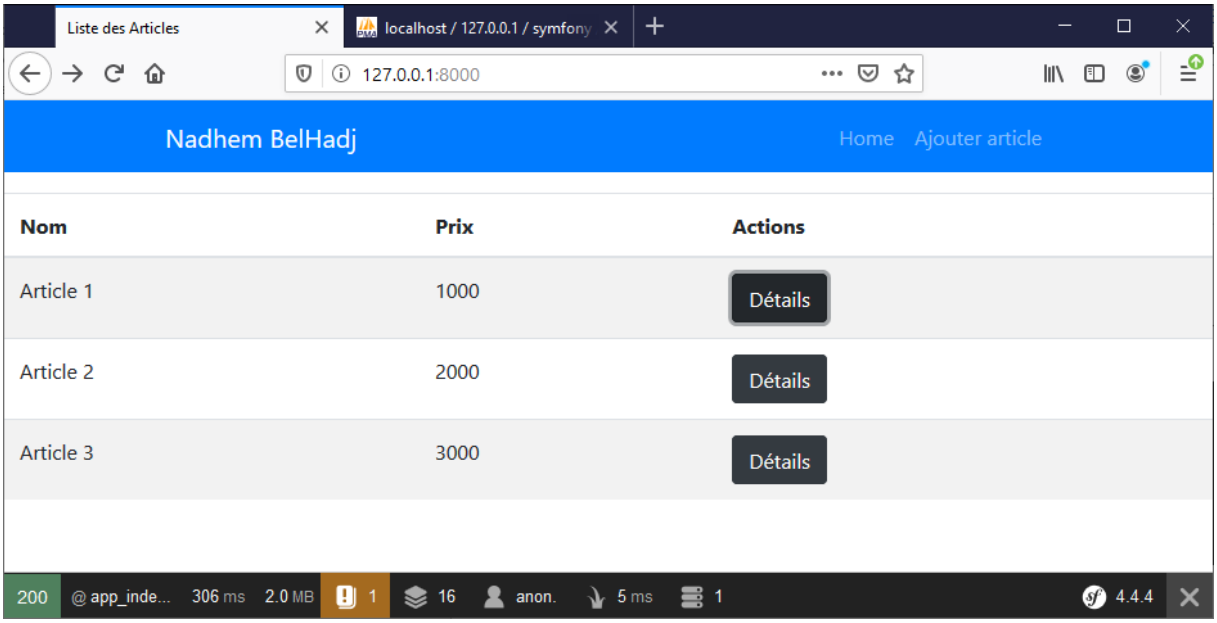
```
 /**
     *@Route("/",name="article_list")
     */
public function home()
  {
    //récupérer tous les articles de la table article de la BD //et les me
ttre dans le tableau $articles
$articles= $this->getDoctrine()->getRepository(Article::class)->findAll();
return $this->render('articles/index.html.twig',['articles'=> $articles]);
  }
```

11. Modifier la vue index.html.twig comme suit :

```
{% extends 'base.html.twig' %}
{% block title%} Liste des Articles{% endblock %}

{% block body %}
 {% if articles %}
    <table id="articles" class="table table-striped">
      <thead>
        <tr>
          <th>Nom</th>
          <th>Prix</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {% for article in articles %}
          <tr>
            <td>{{ article.nom }}</td>
            <td>{{ article.prix }}</td>
            <td>
              <a href="/article/{{ article.id }}" class="btn btn-dark">Détails</a>
```

```
            </td>
          </tr>
        {% endfor %}
      </tbody>
    </table>
  {% else %}
    <p>Aucun articles</p>
  {% endif %}
{% endblock %}
```

12. Testez votre travail :



Ajouter un article à l'aide d'un formulaire

13. Ajouter, au contrôleur indexController.php, la route et la fonction qui permettent d'ajouter un nouvel article

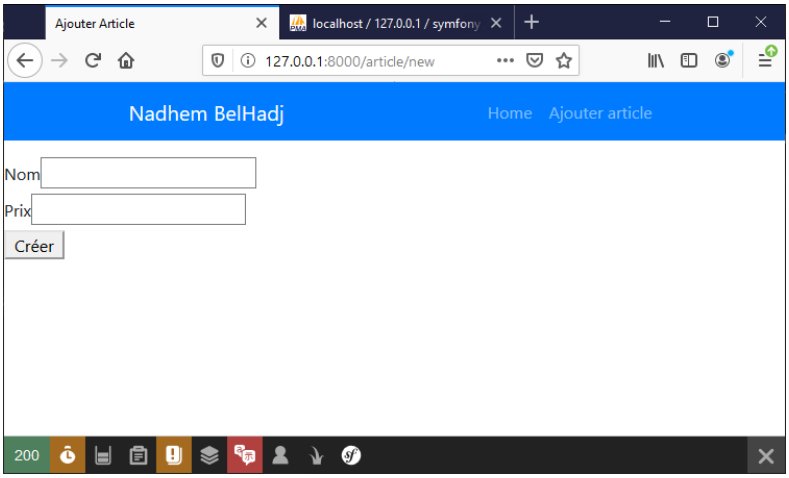
```
/**
     * @Route("/article/new", name="new_article")
     * Method({"GET", "POST"})
     */
    public function new(Request $request) {
        $article = new Article();
        $form = $this->createFormBuilder($article)
          ->add('nom', TextType::class)
          ->add('prix', TextType::class)
          ->add('save', SubmitType::class, array(
            'label' => 'Créer')
```

```
          )->getForm();


        $form->handleRequest($request);

        if($form->isSubmitted() && $form->isValid()) {
          $article = $form->getData();

          $entityManager = $this->getDoctrine()->getManager();
          $entityManager->persist($article);
          $entityManager->flush();

          return $this->redirectToRoute('article_list');
        }
        return $this->render('articles/new.html.twig',['form' => $form-
>createView()]);
    }
```

14. Créer la vue articles/new.html.twig :

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

{% block title %}Ajouter Article{% endblock %}
{% block body %}
 {{ form_start(form) }}
 {{ form_widget(form) }}
 {{ form_end(form) }}
{% endblock %}
```

15. Testez votre travail :



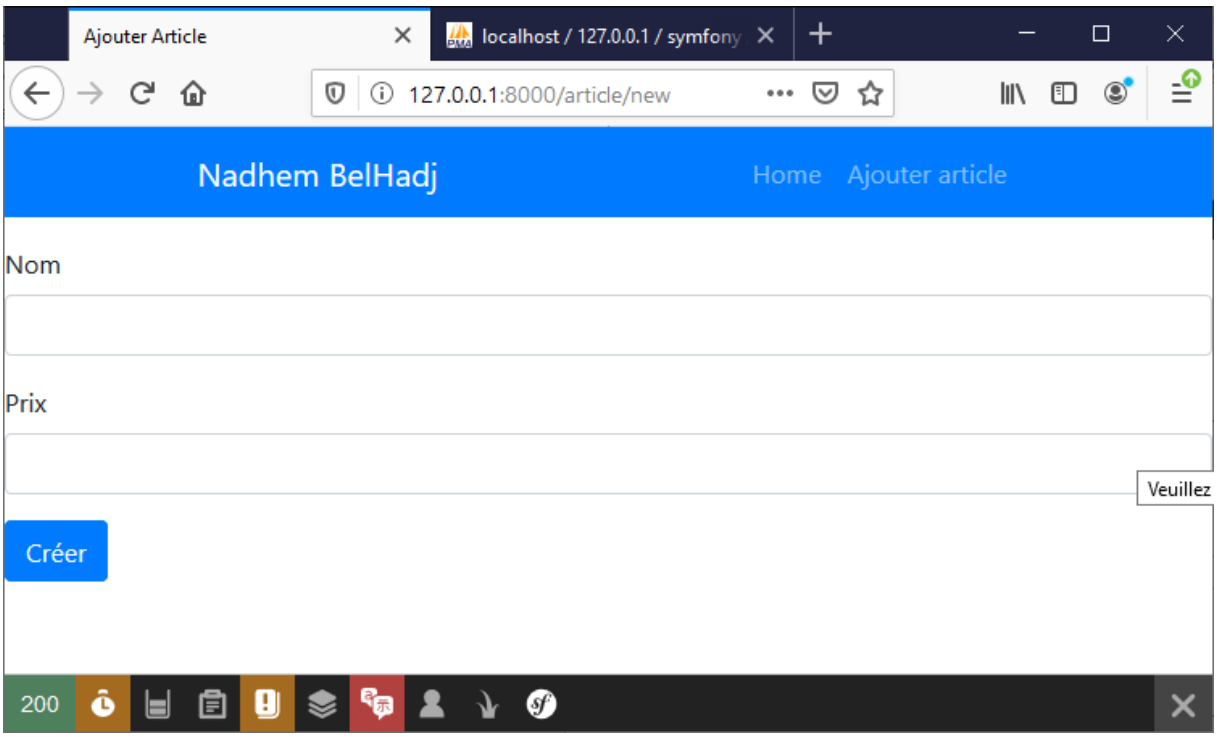
16. Pour que le générateur de template utilise automatiquement Bootstrap lors de la génération des formulaires, ouvrez le fichier config/packages/twig.yaml et ajoutez la ligne suivante :

twig:

…

form_themes: ['bootstrap_4_layout.html.twig']

17. Testez votre travail :



Afficher les détails d'un article

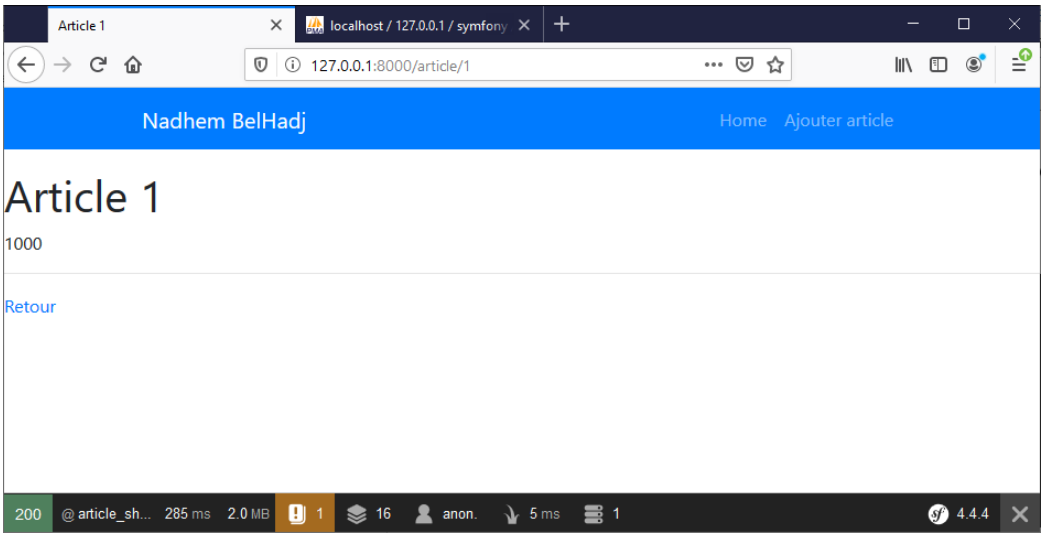
18. Ajouter, au contrôleur indexController.php, la route et la fonction qui permettent d'afficher les détails d'un article

```
/**
     * @Route("/article/{id}", name="article_show")
     */
    public function show($id) {
$article = $this->getDoctrine()->getRepository(Article::class)
                ->find($id);
return $this->render('articles/show.html.twig',
array('article' => $article));
    }
```

19. créer la vue *articles/show.html.twig* qui va permettre d'afficher les détails d'un article :

```
{% extends 'base.html.twig' %}
{% block title %}{{ article.nom }}{% endblock %}
{% block body %}
 <h1>{{ article.nom }}</h1>
 <p>{{ article.prix }}</p>
 <hr>
 <a href="/">Retour</a>
{% endblock %}
```

20. Testez votre travail :



Modifier un article

21. Ajouter, au contrôleur indexController.php, la route et la fonction qui permettent de modifier un article :

```
 /**
     * @Route("/article/edit/{id}", name="edit_article")
     * Method({"GET", "POST"})
     */
    public function edit(Request $request, $id) {
        $article = new Article();
        $article = $this->getDoctrine()->getRepository(Article::class)->find($id);

        $form = $this->createFormBuilder($article)
          ->add('nom', TextType::class)
          ->add('prix', TextType::class)
          ->add('save', SubmitType::class, array(
            'label' => 'Modifier'
          ))->getForm();

        $form->handleRequest($request);
        if($form->isSubmitted() && $form->isValid()) {

          $entityManager = $this->getDoctrine()->getManager();
          $entityManager->flush();

          return $this->redirectToRoute('article_list');
        }

        return $this->render('articles/edit.html.twig', ['form' => $form-
>createView()]);
      }
```

22. créer la vue *articles/edit.html.twig* qui va permettre de modifier un article :

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

{% block title %}ModifierArticle{% endblock %}

{% block body %}
 {{ form_start(form) }}
 {{ form_widget(form) }}
 {{ form_end(form) }}
```

```
{% endblock %}
```

23. Modifier le fichier index.htm.twig pour ajouter le bouton Modifier :

```
<td>
<a href="/article/{{ article.id }}" class="btn btn-dark">Détails</a>
<a href="/article/edit/{{ article.id }}" class="btn btn-dark">Modifier</a>
</td>
```

Supprimer un article

24. Modifier le fichier index.htm.twig pour ajouter le bouton Supprimer :

```
<td>
 <a href="/article/{{ article.id }}" class="btn btn-dark">Détails</a>
 <a href="/article/edit/{{ article.id }}" class="btn btn-
dark">Modifier</a>
 <a href="/article/delete/{{ article.id }}" class="btn btn-danger"
 onclick="return confirm('Etes-
vous sûr de supprimer cet article?');">Supprimer</a>
</td>
```

25. Ajouter, au contrôleur indexController.php, la route et la fonction qui permettent de supprimer un article :

```
/**
     * @Route("/article/delete/{id}",name="delete_article")
     * @Method({"DELETE"})
     */
    public function delete(Request $request, $id) {
        $article = $this->getDoctrine()->getRepository(Article::class)-
>find($id);

        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->remove($article);
        $entityManager->flush();

        $response = new Response();
        $response->send();

        return $this->redirectToRoute('article_list');
      }
```

26. Testez votre travail