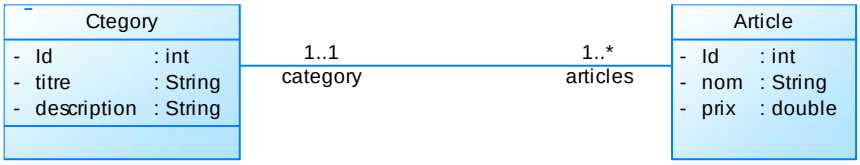
TP6 : LES ENTITES ET LEURS RELATIONS

On se propose dans cet atelier d'ajouter une entité « *Category* » et de créer une relation entre les entités *Article* et *Category*, comme le montre le diagramme de classes suivant :



1. Toujours dans le même projet symfony, créer une nouvelle entité *Category* :

php bin/console make:entity Category

Les propriétés de la classe Category sont :

- titre string(255)
- description text
- articles (relation/OneToMany)

L'entité Category générée aura le code suivant :

```php
<?php

namespace App\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\CategoryRepository")
 */
class Category
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;
```

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

    /**
     * @ORM\Column(type="text", nullable=true)
     */
    private $description;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\Article", mappedBy="category")
     */
    private $articles;

    public function __construct()
    {
        $this->articles = new ArrayCollection();
    }

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

    public function getTitre(): ?string
    {
        return $this->titre;
    }

    public function setTitre(string $titre): self
    {
        $this->titre = $titre;

        return $this;
    }

    public function getDescription(): ?string
    {
        return $this->description;
    }
```

```
    public function setDescription(?string $description): self
    {
        $this->description = $description;

        return $this;
    }

    /**
     * @return Collection|Article[]
     */
    public function getArticles(): Collection
    {
        return $this->articles;
    }

    public function addArticle(Article $article): self
    {
        if (!$this->articles->contains($article)) {
            $this->articles[] = $article;
            $article->setCategory($this);
        }

        return $this;
    }

    public function removeArticle(Article $article): self
    {
        if ($this->articles->contains($article)) {
            $this->articles->removeElement($article);
            // set the owning side to null (unless already changed)
            if ($article->getCategory() === $this) {
                $article->setCategory(null);
            }
        }

        return $this;
    }
}
```

La propriété article sera ajoutée à l'entité Article avec ses getter et setter:

```
/**
```

```
  * @ORM\ManyToOne(targetEntity="App\Entity\Category", inversedBy="articles")
    * @ORM\JoinColumn(nullable=false)
    */
   private $category;
```

```
public function getCategory(): ?Category
    {
        return $this->category;
    }

    public function setCategory(?Category $category): self
    {
        $this->category = $category;

        return $this;
    }
```

Génération de la table Category et ajout de la colonne category à la table Article

2. Supprimer avec MySql tous les anciens articles (car il n'ont pas de catégorie, puis que ce champ sera NOT NULL)

Pour générer de la table Category et ajouter la colonne category à la table Article, taper la commande suivante :

php bin/console make:migration

puis

php bin/console doctrine:migrations:migrate

3. Vérifier les modifications dans la BD

Ajouter un formulaire pour ajouter les catégories

4. Créer la classe formulaire correspondant à l'entité ***Category*** tapant la commande suivante :

php bin/console make:form

Suivez l'assistant en fournissant le nom de la classe : ***CategoryType*** et le nom de l'entité : ***Category***

5. Créer la fonction newCategory à la classe IndexController.php comme suit :

```
…
use App\Entity\Category;
use App\Form\CategoryType;
…
```

```
/**
     * @Route("/category/newCat", name="new_category")
     * Method({"GET", "POST"})
     */
    public function newCategory(Request $request) {
      $category = new Category();
      $form = $this->createForm(CategoryType::class,$category);
      $form->handleRequest($request);
      if($form->isSubmitted() && $form->isValid()) {
        $article = $form->getData();
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($category);
        $entityManager->flush();
      }
 return $this->render('articles/newCategory.html.twig',['form'=>
     $form->createView()]);
  }
```

6. Ajouter le lien Ajouter Catégorie eu menu (la navbar) en modifiant le fichier inc/navbar.html.twig comme suit :

```
<ul class="navbar-nav ml-auto">
     <li class="nav-item">
       <a href="/" class="nav-link">Home</a>
     </li>
     <li class="nav-item">
       <a href="{{ path('new_article')}}" class="nav-link">Ajouter article</a>
```

```
      </li>
      <li class="nav-item">
        <a href="{{ path('new_category')}}" class="nav-link">Ajouter catégorie</a>
      </li>
    </ul>
```

7. Créer la vue NewCategory.html.twig permettant d'ajouter une nouvelle catégorie :

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

{% block title %}Ajouter Category{% endblock %}
{% block body %}
 {{ form_start(form) }}

 {{ form_widget(form) }}

  <button type="submit" class="btn btn-success">Créer</button>
 {{ form_end(form) }}
{% endblock %}
```

8. Ajoutez 3 catégorie de votre choix

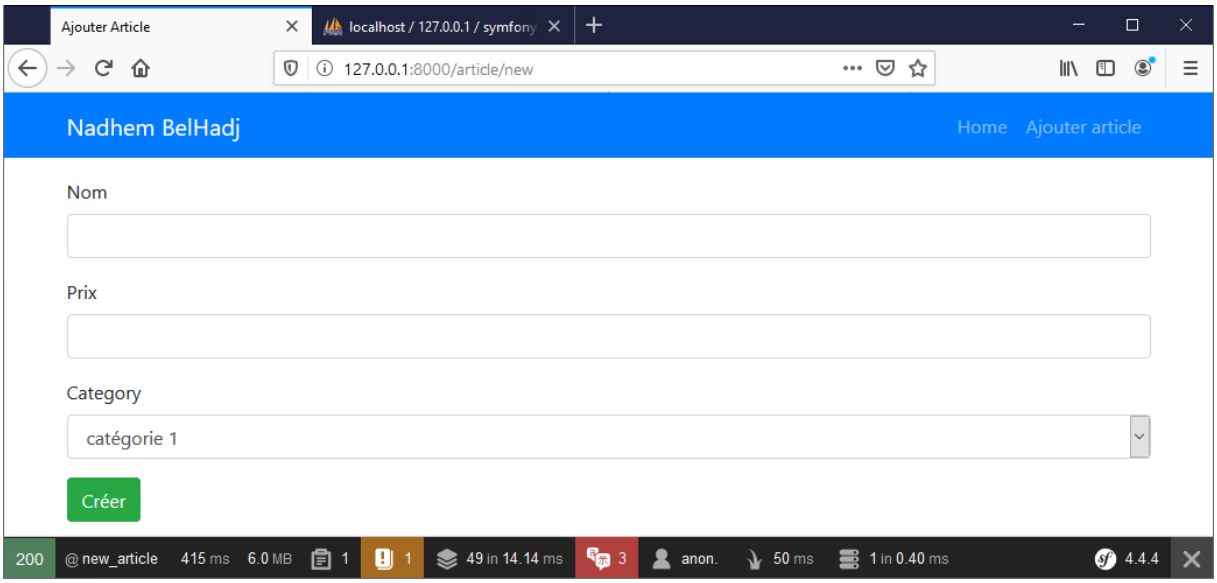
Modifier le formulaire de création d'un article

9. Pour ajouter une liste déroulante afin d'affecter une catégorie à un article, modifier la fonction buildForm de la classe Form/ArticleType.php comme suit :

```
…
use App\Entity\Category;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
…
```

```
public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('nom')
            ->add('prix')
            ->add('category',EntityType::class,['class' => Category::class,
                                        'choice_label' => 'titre',
                                         'label' => 'Catégorie']);}
```

10. Testez votre travail :



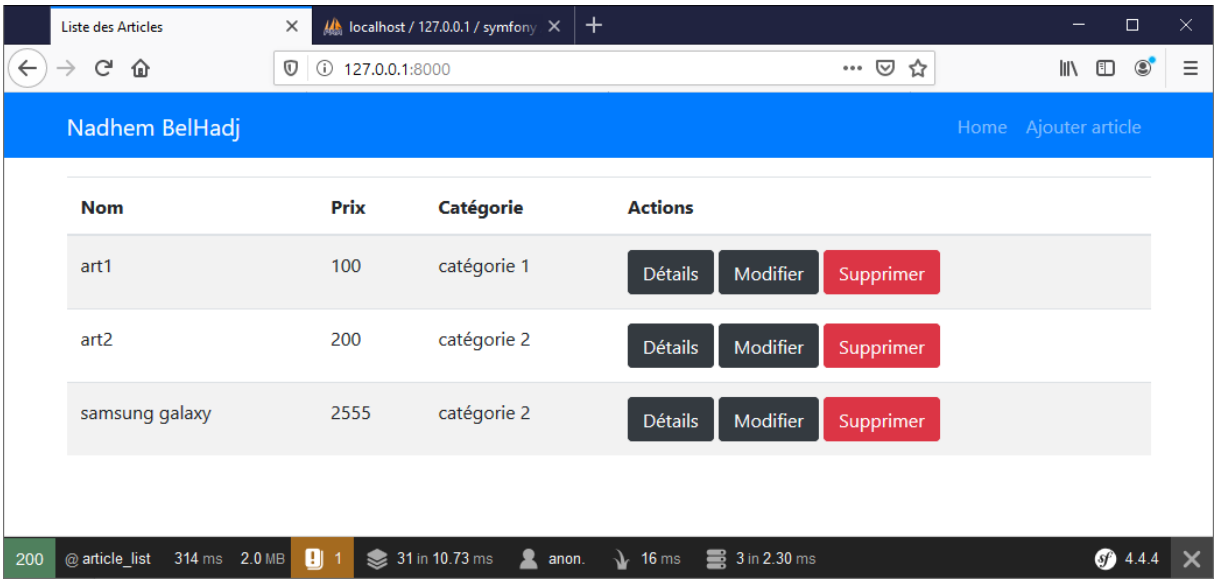
Afficher la catégorie des articles dans la liste

11. Pour afficher la catégorie des articles dans la liste des articles, modifier le code de la vue index.html.twig comme suit :

```
<table id="articles" class="table table-striped">
      <thead>
        <tr>
          <th>Nom</th>
          <th>Prix</th>
          <th>Catégorie</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {% for article in articles %}
          <tr>
            <td>{{ article.nom }}</td>
            <td>{{ article.prix }}</td>
             <td>{{ article.category.titre }}</td>
           <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>
        </tr>
      {% endfor %}
    </tbody>
  </table>
```

12. Testez votre travail



13. Afficher la catégorie dans la page show.html.twig, en ajoutant la ligne suivante :

```
<p>{{ article.category.titre }}</p>
```