# TP1 : Installation de Symfony

1. Installer XAMPP pour **Windows** https://www.apachefriends.org/fr/download.html

2. Ajouter le chemin de la commande php (D:\xampp\php ) ou (C:\xampp\php ) à votre variable d'environnement **PATH**,
3. Installer Composer (https://getcomposer.org/download/) qui est un gestionnaire de dépendances pour PHP,

4. Créer un projet symfony :
    a. Ouvrez une fenêtre « invite de commande » GIT,
    b. Puis placez-vous sur D:\xampp\htdocs ou C:\xampp\htdocs
    c. Puis tapez

D:\xampp\htdocs> **composer create-project symfony/website-skeleton sf5 ~5**

- **symfony/website-skeleton** : est le nom du package à installer
- **sf5** est le nom du dossier où le package sera installé, un dossier nommé sf5 sera créé dans le dossier courant, qui contiendra notre projet.
- **~5** la version de symfony

5. Installer le serveur intégré Symfony
    a. Ouvrez une fenêtre « invite de commande »,
    b. Puis placez-vous sur D:\xampp\htdocs\sf5 ou C:\xampp\htdocs
    c. Puis tapez :

    **composer require --dev symfony/web-server-bundle ^4.4.2**

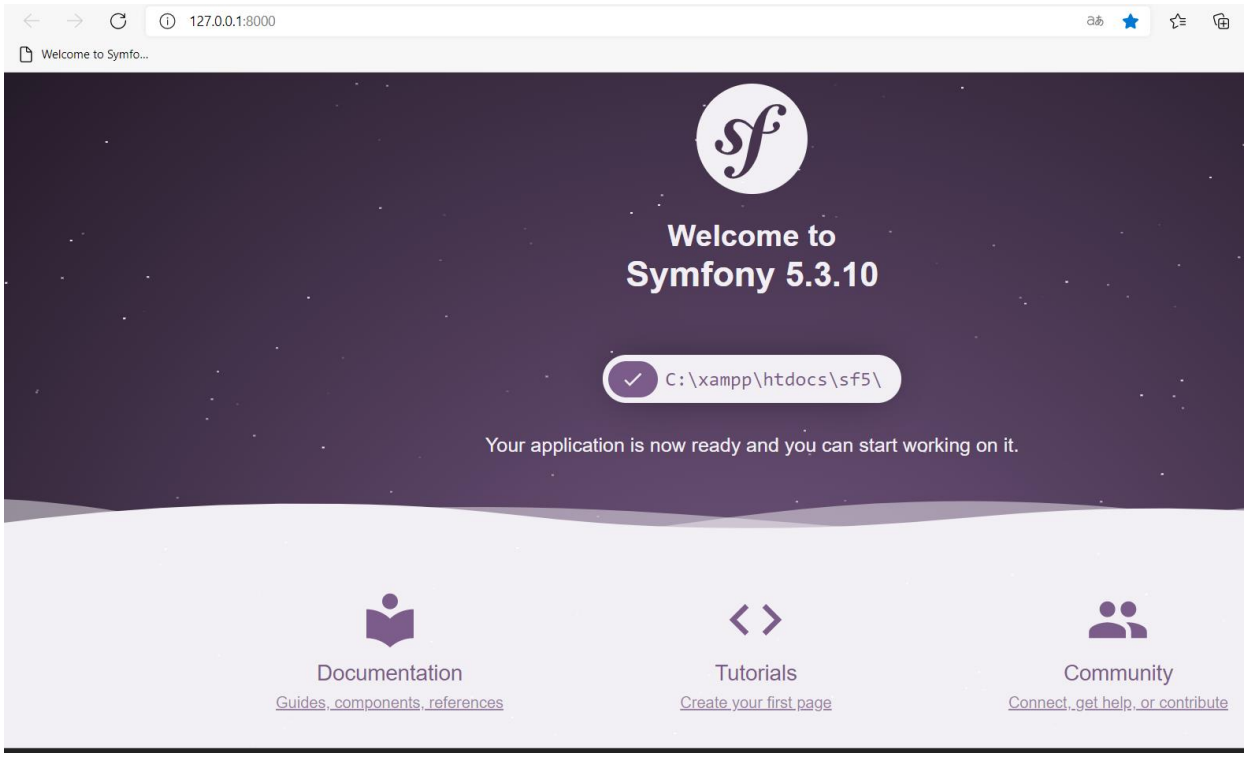
6. Lancer le serveur intégré Symfony
    a. Ouvrez une fenêtre « invite de commande »,
    b. Puis placez-vous sur D:\xampp\htdocs\sf4
    c. Puis tapez :

    **php bin/console server:run**

    **sinon utiliser la commande symfony server :start**

7. Tester votre projet

Ouvrez l'URL http://127.0.0.1:8000/ dans le navigateur de votre choix :


127.0.0.1:8000
Welcome to Symfo...
Welcome to
Symfony 5.3.10
C:\xampp\htdocs\sf5\
Your application is now ready and you can start working on it.
Documentation
Guides, components, references
Tutorials
Create your first page
Community
Connect, get help, or contribute

# TP2 Création de Controller symfony

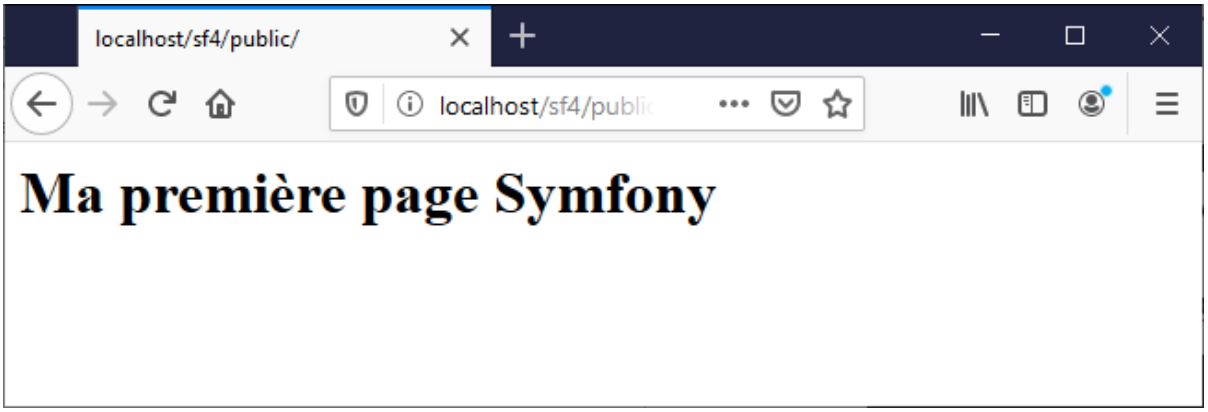
1. Ouvrez le projet avec l'IDE/éditeur de votre choix, ici on utilise Visual Code, (vous pouvez utiliser Sublime ou autre)
2. Créer, dans le dossier src/Controller, une classe PHP appelée : IndexController (créer le fichier IndexController.php) :

```php
<?php

namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;

class IndexController extends AbstractController
{
  public function home()
  {
     return new Response('<h1>Ma première page Symfony</h1>');

  }
}
```

3. Mettre à jour le fichier routes.yaml :

```yaml
homepage:
    path: /
    controller: App\Controller\IndexController::home
```

4. Testez votre travail :



## Utiliser les annotations pour le Routing

5. Dans un terminal taper la commande :
   composer require annotation
6. Vider le fichier routes.yaml puis modifier le code du controller comme suit :

```php
<?php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
Use Symfony\Component\Routing\Annotation\Route;

class IndexController extends AbstractController
{
    /**
     * @Route("/")
     */
  public function home()
  {
     return new Response('<h1>Ma première page Symfony</h1>');

  }
}
```

7. Testez votre travail

## Création d'une première vue

8. Créer, dans le dossier src/templates, le fichier index.html.twig, contenant le texte : `<h1>Mon premier fichier html.twig</h1>`
9. Modifier la méthode home() du controller comme suit :

```php
/**
    * @Route("/")
    */
 public function home()
 {
    return  $this->render('index.html.twig');

 }
```

10. Testez votre travail

## Transmettre un modèle à la vue

11. Modifier le fichier IndexController.php comme suit :

```php
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractControll
er;
use Symfony\Component\HttpFoundation\Response;
Use Symfony\Component\Routing\Annotation\Route;

class IndexController extends AbstractController
{
      /**
     *@Route("/{name}")
     */
  public function home($name)
  {
  return $this->render('index.html.twig',['name' => $name]);
  }
}
```

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

```
<h1>Hello {{name}}</h1>
```

13. Testez votre travail en tapant l'url suivante :
    http://127.0.0.1:8000/votrenom

# TP3 Twig et Bootstrap symfony

1. Créer sous templates le dossier articles, puis déplacer vers le dossier articles le fichier index.html.twig enfin modifiez-le comme suit :

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

{% block body %}
<h1>Articles</h1>

{% endblock %}
```

2. Dans un terminal taper la commande suivante : composer require twig
3. Modifier le fichier IndexController.php comme suit :

```
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
Use Symfony\Component\Routing\Annotation\Route;

class IndexController extends AbstractController
{
      /**
     *@Route("/")
     */
  public function home()
  {
    return  $this->render('articles/index.html.twig');
  }
}
```

## Intégration de Bootstrap

4. Ouvrez le fichier base.html.twig
5. Aller sur le site : https://getbootstrap.com/docs/4.4/getting-started/introduction/

6. Copier et coller les balises CSS et JS comme suit :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Welcome!{% endblock %}</title>
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
        integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
        {% block stylesheets %}{% endblock %}
    </head>
    <body>

  <div class="container">

      {% block body %}{% endblock %}
  </div>

        <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>
        <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>
        {% block javascripts %}{% endblock %}
    </body>
</html>
```

### Création d'un navbar (menu)

7. Créer le dossier **inc** sous **templates**, puis créer dans inc le fichier navbar.html.twig :

```
<nav class="navbar navbar-expand-sm navbar-dark bg-primary mb-3">
<div class="container">
```

```
  <a href="/" class="navbar-brand">Nadhem BelHadj</a>
  <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#mobile-nav">
    <span class="navbar-toggle-icon"></span>
  </button>

  <div class="collapse navbar-collapse" id="mobile-nav">
    <ul class="navbar-nav ml-auto">
      <li class="nav-item">
        <a href="/" class="nav-link">Home</a>
      </li>
      <li class="nav-item">
  <a href="#" class="nav-link">Ajouter article</a>
      </li>
    </ul>
  </div>
</div>
</nav>
```

8. Inclure le fichier navbar.html.twig dans le fichier base.html.twig , pour afficher le menu (navbar) sur toutes les pages :

```
…
<body>
     {{ include('inc/navbar.html.twig')}}
        {% block body %}{% endblock %}
…
```

9. Testez votre travail :

10. On se propose, à présent de créer un tableau d'articles au niveau du controller et de l'envoyer en tant que modèle à la vue index.html.twig pour afficher les articles sous forme d'un tableau html, modifier le fichier IndexController.php comme suit :

```
 /**
     *@Route("/")
     */
  public function home()
  {
      $articles = ['Artcile1', 'Article 2','Article 3'];
    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>Article</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {% for article in articles %}
          <tr>
            <td>{{ article }}</td>
            <td>
              <a href="/article/1" class="btn btn-dark">Détails</a>

            </td>
          </tr>
        {% endfor %}
      </tbody>
    </table>
  {% else %}
```

```
    <p>Aucun articles</p>
  {% endif %}
{% endblock %}
```

# 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 configurer les paramètres de la base dedonnées :

DATABASE_URL=mysql://root:@localhost:3306/symfony

2. Dans un terminal taper la commande :
   composer require doctrine maker
3. Créer la base de données en tapant dans le terminal :

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

4. Accéder à phpmyadmin et vérifier la création de la base de donnéesen tapant : localhost/phpmyadmin/

## Création de l'entité Article

5. 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;
    }
```

```
}
```

6. Créer la table Article qui correspond à l'entité Article, en tapant lesdeux commandes :

   **php bin/console doctrine:migrations:diff**

   puis

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

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

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

8. 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());
     }
```

9. 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;
```

10. 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
11. Vérifier l'ajout des 3 lignes dans la BD

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

12. 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 :

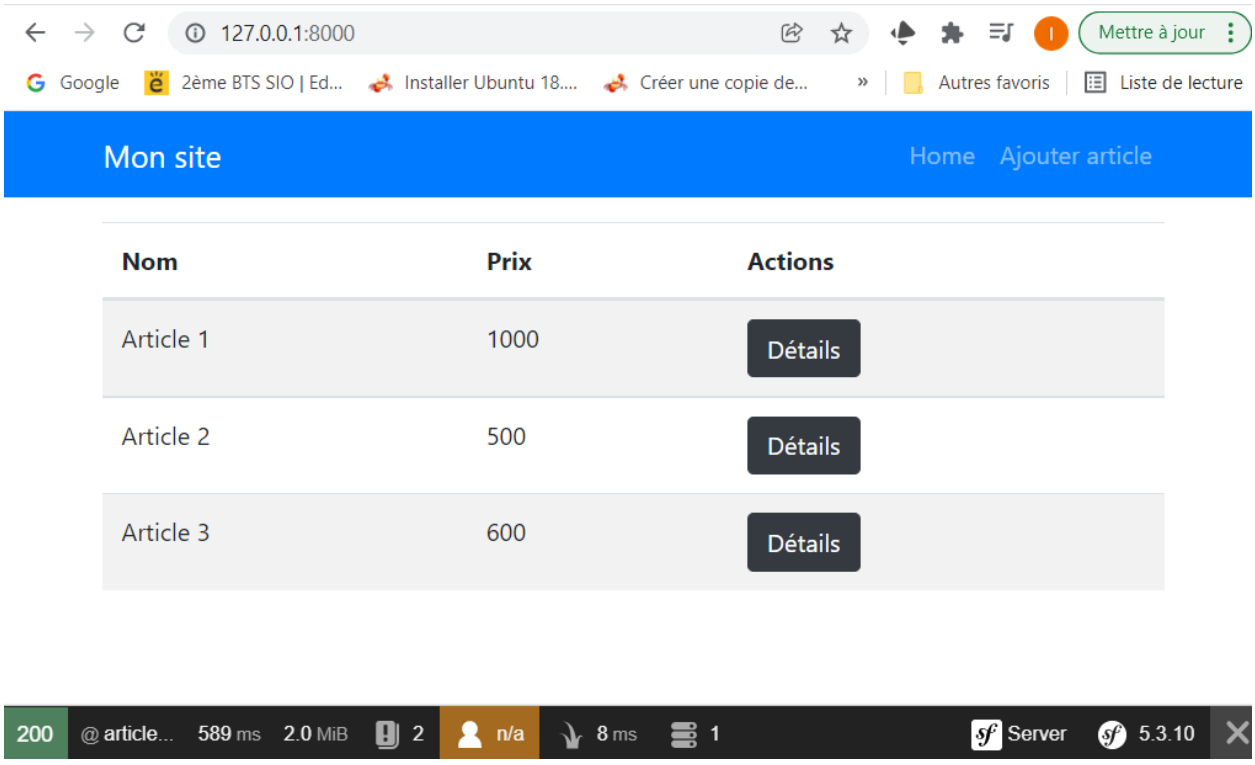
```
 /**
     *@Route("/",name="article_list")
     */
public function home()
  {
    //récupérer tous les articles de la table article de la BD
    // et les mettre dans le tableau $articles
$articles= $this->getDoctrine()->getRepository(Article::class)->findAll();
return $this->render('articles/index.html.twig',['articles'=>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

13. 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 %}
```

14. Testez votre travail :



## Afficher les détails d'un article

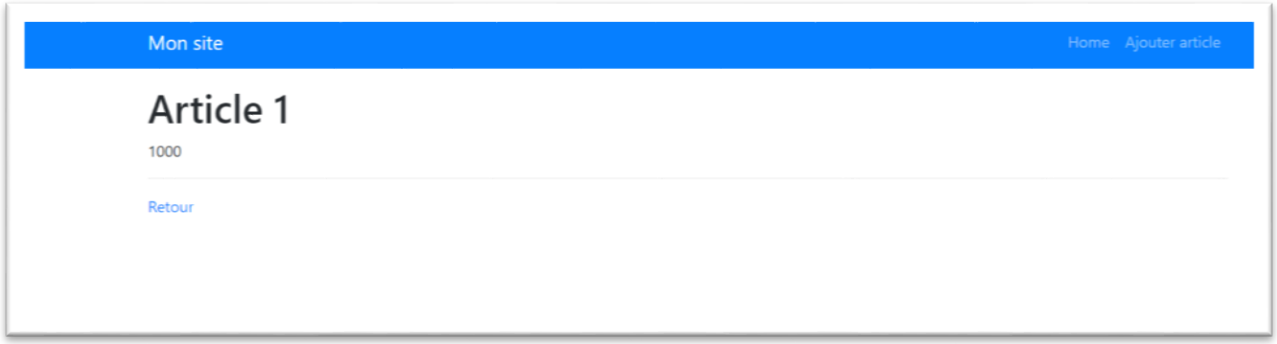
15. Ajouter, au contrôleur indexController.php, la route et la fonction qui permettent d'afficher les détails d'un article

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

16. créer la vue *articles/show.html.twig* qui va permettre d'afficher lesdétails 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 %}
```

17. Testez votre travail :



## Ajouter un article à l'aide d'un formulaire

18. 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()]);
    }
```

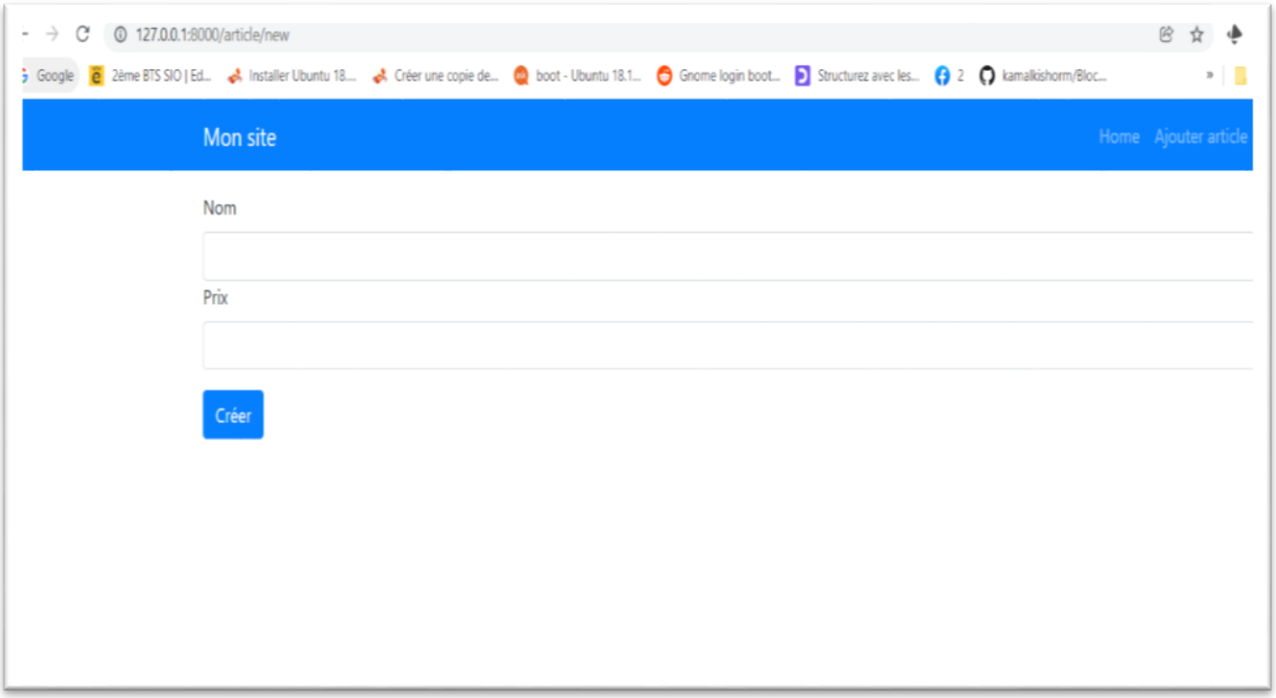
19. Créer la vue articles/new.html.twig :

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

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

20. Testez votre travail : avec l'URL 127.0.0.1/article/new

Si jamais vous avez une erreur concernant la route (no route found for get /) vider le cache avec la commande :

**php bin/console cache:clear**



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']
```

…

## 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');
        }
```

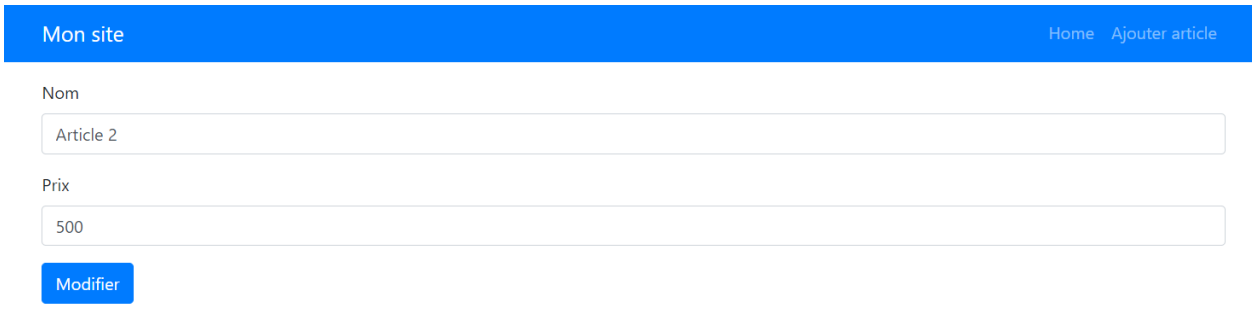
```
        return $this->render('articles/edit.html.twig', ['form' => $form-
>createView()]);
      }
```

22. Créer la vue *articles/edit.html.twig* qui va permettre de modifier unarticle :

```
{% 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.html.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 :

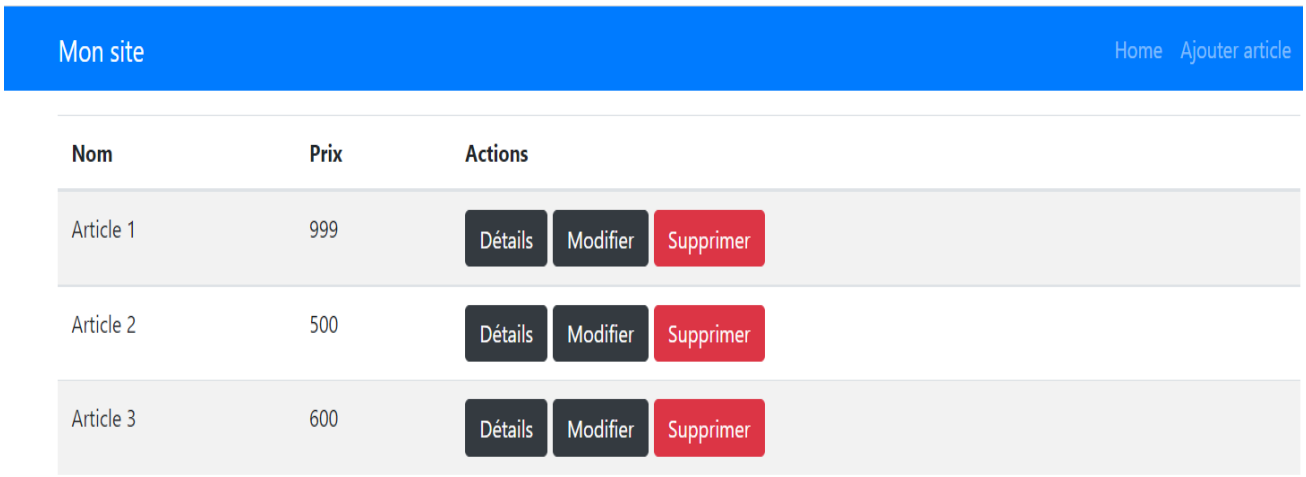
```
<td>
 <a href="/article/{{ article.id }}" class="btn btn-dark">Détails</a>
 <a href="/article/edit/{{ article.id }}" class="btn btn-
dark">Modifier</a>
 <a href="/article/delete/{{ article.id }}" class="btn btn-danger"
 onclick="return confirm('Etes-
vous sûr de supprimer cet article?');">Supprimer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



# TP5 : Formulaires et Validation

## Création de Formulaires

1. Ouvrez un terminal sous votre projet Sf5, puis créer la classe formulaire correspondant à l'entité ***Article*** tapant la commande suivante :

   **php bin/console make:form**

Suivez l'assistant en fournissant le nom de la classe : ***ArticleType*** et le nom de l'entité : ***Article***

2. La classe form/ArticleType sera créée :

```php
<?php

namespace App\Form;

use App\Entity\Article;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ArticleType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $option
s)
    {
        $builder
            ->add('nom')
            ->add('prix')
        ;
    }

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

3. Modifier la fonction new de la classe IndexController.php comme suit :

```
/**
     * @Route("/article/new", name="new_article")
     * Method({"GET", "POST"})
     */
    public function new(Request $request) {
        $article = new Article();
        $form = $this->createForm(ArticleType::class,$article);
        $form->handleRequest($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()]);
    }
```

4. Ajouter le *use* le classe ArticleType au début du fichier :

```
use App\Form\ArticleType;
```

5. Ajouter le Bouton « Créer » à la vue new.htm.twig :

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

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

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

6. Modifier le fichier inc/navbar.htm.twig pour ajouter la route new_article:

```
<a href="{{ path('new_article')}}" class="nav-link">Ajouter article</a>
```

7. Testez votre travail

8. Faire la même chose pour la fonction **edit** :

```
 /**
     * @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 = $this->createForm(ArticleType::class,$article);

        $form->handleRequest($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()]);
      }
```

9. Ajouter le Bouton « Modifier » à la vue edit.htm.twig :

```
{{ form_widget(form) }}
  <button type="submit" class="btn btn-success">Modifier</button>
{{ form_end(form) }}
```

10. Testez votre travail

## Validation des données

11. Pour créer des contraintes de validation sur les données entrées dans les formulaires, symfony offre des annotations de validation qui seront créées au niveau de l'entité, pour plus de détails vous pouvez consulter la documentation symfony sur la page :
https://symfony.com/doc/current/validation.html

12. On voudrait ajouter les contraintes :

    - Le nom d'un article doit comporter au moins 5 caractères et au maximum 50
    - Le prix d'un article ne doit pas être égal à 0

Pour ce faire modifier l'entité Article comme suit :

```php
<?php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;
/**
 * @ORM\Entity(repositoryClass="App\Repository\ArticleRepository")
 */
class Article
{    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;
    /**
     * @ORM\Column(type="string", length=255)
     * @Assert\Length(
     *      min = 5,
     *      max = 50,
     * minMessage = "Le nom d'un article doit comporter au moins {{ limit }} caractères",
     * maxMessage = "Le nom d'un article doit comporter au plus {{ limit }} caractères"
     * )
     */
    private $nom;

    /**
```

```
     * @ORM\Column(type="decimal", precision=10, scale=0)
     * @Assert\NotEqualTo(
     *     value = 0,
     *     message = "Le prix d'un article ne doit pas être égal à  0 "
     * )
     */
    private $prix;
...
```

13. Pour desactiver la validation HTML 5 coté client au niveau du navigateur modifier la fonction form_start au niveau du fichier new.html.twig comme suit :

```
{{ form_start(form, {'attr': {'novalidate': 'novalidate'}}) }}
```

14. Tester votre travail