Akhmed Ozdamirov
BTS SIO option SLAM
2023-2025

Stage année 2ème année BTS SIO

Lovzysoft est une entreprise SAS dont le domaine d'activité est le service administratif combiné de bureau. L'entreprise est située au 220 rue Guy Arnaud, 30900 Nîmes.
L'entreprise propose des services de gestion de documents administratifs, elle aide notamment les auto-entrepreneurs à la création de leur entreprise. L'entreprise est constituée d'au moins 5 employés.
Le président de l'entreprise a aussi pour but de lancer une association dont l'objectif est l'insertion sociale des individus en mettant en liens les futurs employés avec les employeurs.

J'ai dû travailler seul durant mon stage car il n'y avait pas de développeur ou d'autres informaticiens dans l'entreprise. Pour mener à terme le projet, j'ai dû suivre plusieurs tutoriels en ligne car nous n'avions pas pu aborder le framework Symfony en classe.
J'ai utilisé Github pour gérer l'avancement de mon projet ainsi que l'éditeur de code Visual Studio Code.
Durant ce stage j'ai pu découvrir Symfony 6 grâce à l'aide en ligne.
J'ai apprécié ce stage puisque j'ai pu y découvrir Symfony 6 et voir quelque base simple qui me seront utile quand je voudrais plus approfondir mon apprentissage.

L'objectif principal de mon stage était de créer un site web pour une association. Ce site devait inclure plusieurs fonctionnalités essentielles :

- Une page d'accueil, d'articles, d'évènements, tableau de bord, page de connexion, de contact
- Un système de gestion d'articles et de catégories
- Un tableau de bord pour l'administrateur (EasyAdmin4)
- Un système d'authentification pour l'Admin
- Une page de contact

Afin de mener à bien ce projet j'ai utilisé les technologies suivantes :

- **Symfony 6**
- **EasyAdmin4 (Espace administrateur)**
- **Bootstrap 5**
- **Php 8.1 ou supérieur**
- **Wamp**
- **Git**
- **Composer**
- **NodeJS**

Lien du projet :
https://github.com/aozdamirov/association

- **Page d'accueil** : Présentation de l'association.
- **Système d'articles et de catégories** : Ajout, modification, suppression d'articles.
- **Tableau de bord (EasyAdmin4)** : Interface d'administration.
- **Système d'authentification** : inscription et connexion des utilisateurs.

Pour démarrer le serveur Symfony et voir notre site :
symfony serve

Symfony utilise Twig : pour gérer sa vue et afficher les pages HTML. Il permet d'écrire du code plus proprement en séparant la logique et l'affichage. On peut par exemple mettre des conditions d'affichage.

Exemple :

```
    {% if is_granted('IS_AUTHENTICATED_FULLY')%}
      <li class="nav-item">
       <a class="nav-link" href="{{ path('admin') }}">Tableau de bord</a>
      </li>
    {% endif %}
```

Le lien « tableau de bord » dans la navbar ne s'affiche que si l'utilisateur est connecté.
En Twig, les conditions s'écrivent entre {% %}.

Symfony utilise la Doctrine ORM : Doctrine permet de gérer les bases de données en définissant des entités comme des classes PHP, il génère ensuite automatiquement les requêtes SQL nécessaire. L'avantage d'une Doctrine est qu'il n'est pas nécessaire d'écrire du SQL manuellement, ce qui facilite le développement et améliore la sécurité en évitant les injections SQL. Un autre avantage est que Doctrine permet la migration automatique pour modifier la base de données.

Exemple :

Créer une base de donnée :

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

Créer une entité :

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

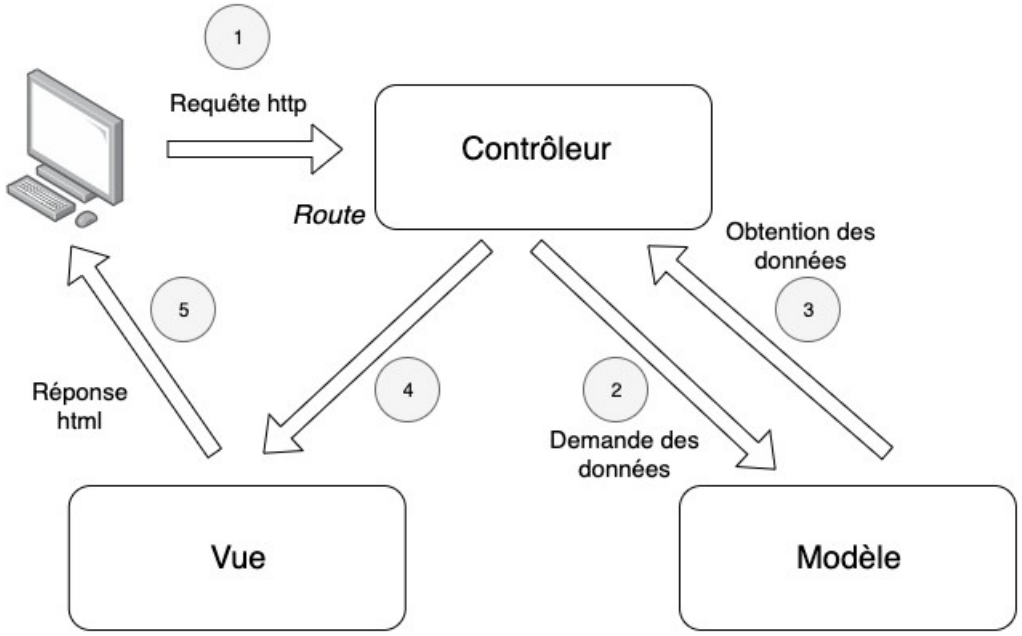
Faire la migration :

1. Générer le fichier de migration.
```
php bin/console make:migration
```

2. Exécuter le fichier de migration.
```
php bin/console doctrine:migrations:migrate
```

Symfony utilise le modèle vue controller (MVC). Elle consiste à séparer l'application en 3 parties qui ont chacun leur rôle :

- Les modèles gèrent l'accès aux données de la base de données.
- Les vues sont l'affichage présenté à l'utilisateur.
- Les controllers contiennent les méthodes/fonctions avec les actions à exécutés selon la requête de l'utilisateur.



Le **routeur** de l'application va appeler le bon controller en fonction de la route.

Le **controller** va dire les actions à effectuer.

Le **modèle** va valider et enregistrer les données dans la base de données.

La **vue** est l'affichage, elle contient la logique d'affichage des données qui ont été récupérées par le modèle.

Mon application suit l'architecture MVC (modèle, vue, contrôleur). Les requêtes de l'utilisateur sont envoyées aux contrôleurs qui récupèrent les données depuis les modèles et les affichent dans la vue.

Exemple : Système d'article

1. Route (config/routes.yaml)

```
controllers:
  resource: ../src/Controller/
  type: annotation
```

Ce code dit à Symfony d'analyser tous les fichiers dans le dossier /src/Controller pour récupérer les routes qui sont définies par #[Routes(...)]

2. Contrôleur (src/Controller/ArticleController.php)

```
#[Route('article/', name: 'app_article')]
  public function index(ArticleRepository $articleRepo, CategoryRepository
$categoryRepo): Response
  {
    return $this->render('article/index.html.twig', [
      'articles' => $articleRepo->findAll(),
      'categories' => $categoryRepo->findAll()
    ]);
  }
```

Cette méthode récupère tous les articles et catégories en base de données grace à la Doctrine ORM et les envoie à la vue article/index.html.twig.

3. Modèle (src/Entity/Article.php)

```
class Article implements TimestampedInterface
{
  #[ORM\Id]
  #[ORM\GeneratedValue]
  #[ORM\Column]
  private ?int $id = null;

  #[ORM\Column(length: 255)]
  private ?string $title = null;

  #[ORM\Column(length: 255)]
  private ?string $slug = null;

  #[ORM\Column(type: Types::TEXT, nullable: true)]
  private ?string $content = null;

  #[ORM\Column(length: 100, nullable: true)]
  private ?string $featuredText = null;

  #[ORM\Column(type: Types::DATETIME_MUTABLE)]
  private ?\DateTimeInterface $createdAt = null;

  #[ORM\Column(type: Types::DATETIME_MUTABLE, nullable: true)]
  private ?\DateTimeInterface $updatedAt = null;

  #[ORM\ManyToMany(targetEntity: Category::class, inversedBy: 'articles')]
  private Collection $categories;

  #[ORM\ManyToOne]
  private ?Media $featuredImage = null;
```

L’entité Article représente la table article dans la base de données. Doctrine la convertit en SQL automatiquement.

4. Vue (templates/articles/index.html.twig)

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

{% block title %}Articles{% endblock %}

{% block body %}
    <main class="container">
        <div class="row">

            <div class="col-md-8">
                <h3 class="pb-4 mb-4 fst-italic border-bottom">Articles récents</h3>
                <div id="article-list">
                    {% include 'article/list.html.twig' with {articles} %}
                </div>
            </div>

            <div class="col-md-4">
                <div class="position-sticky" style="top: 2rem;">
                    <!--{% include 'widget/about.html.twig' %} -->
                    {% include 'widget/categories.html.twig' %}
                </div>
            </div>
        </div>
{% endblock %}
```

Hérite de base.html.twig (notre template de base du site).
Définit le tilte de la page.
Affiche le titre principal.
Charge les articles depuis article/list.html.twig.
Affiche un widget en sidebar qui montre les catégories.

Explication des dossiers Symfony :

/src (Source)

C'est le dossier principal du code **applicatif**. Il contient :

- **Controller/** → Contient les **contrôleurs** qui gèrent les requêtes et les réponses.
- **Entity/** → Contient les **entités** (modèles de base de données utilisés par Doctrine).
- **Repository/** → Contient les **repositories**, qui permettent d'effectuer des requêtes sur les entités.
- **Form/** → Contient les **formulaires** Symfony.
- **Security/** → Contient les classes liées à l'**authentification** et à la **gestion des utilisateurs**.

/templates (Vues)

Ce dossier contient les fichiers **Twig**, qui sont les **templates HTML** du projet.

Exemple :

- base.html.twig → Template de base utilisé par toutes les pages.
- home.html.twig → Page d'accueil.
- user/profile.html.twig → Page de profil utilisateur.

/config (Configuration)

Il contient tous les fichiers de **configuration** de Symfony et des services utilisés (base de données, sécurité, routes…).

- routes.yaml → Définit les **routes** de l'application.
- services.yaml → Configure les **services** Symfony.
- packages/ → Contient les configurations des **différents bundles** (Doctrine, Twig, Security, etc.).

/public (Fichiers accessibles publiquement)

Contient tout ce qui doit être **accessible par un navigateur** :

- index.php → Point d'entrée de l'application.
- assets/ → Images, CSS, JavaScript.
- uploads/ → Dossier pour stocker les fichiers uploadés.

/var (Données temporaires)

Ce dossier contient les fichiers **temporaires** générés par Symfony :

- cache/ → Contient le cache pour optimiser les performances.
- log/ → Contient les **logs** (erreurs, requêtes…).

/vendor (Dépendances)

C'est le dossier qui contient **toutes les bibliothèques installées via Composer**. Tu ne dois jamais modifier ce dossier directement.

/migrations (Gestion des bases de données)

Ce dossier contient les fichiers de **migration Doctrine** qui permettent de gérer les modifications de la base de données.

ANNEXE

Cours apprendre Symfony : https://www.youtube.com/watch?v=UTusmVpwJXo

Doc Symfony : https://symfony.com/doc/current/index.html

Comprendre la structure d'un projet Symfony : https://www.youtube.com/watch?v=hDMtkcFljKU&t

Créer blog : https://www.youtube.com/watch?v=1BbmGc6J7qA&list=PLkHw7J3J2iaoIowz7LIImIFHkMJzO_84F

Image du site :

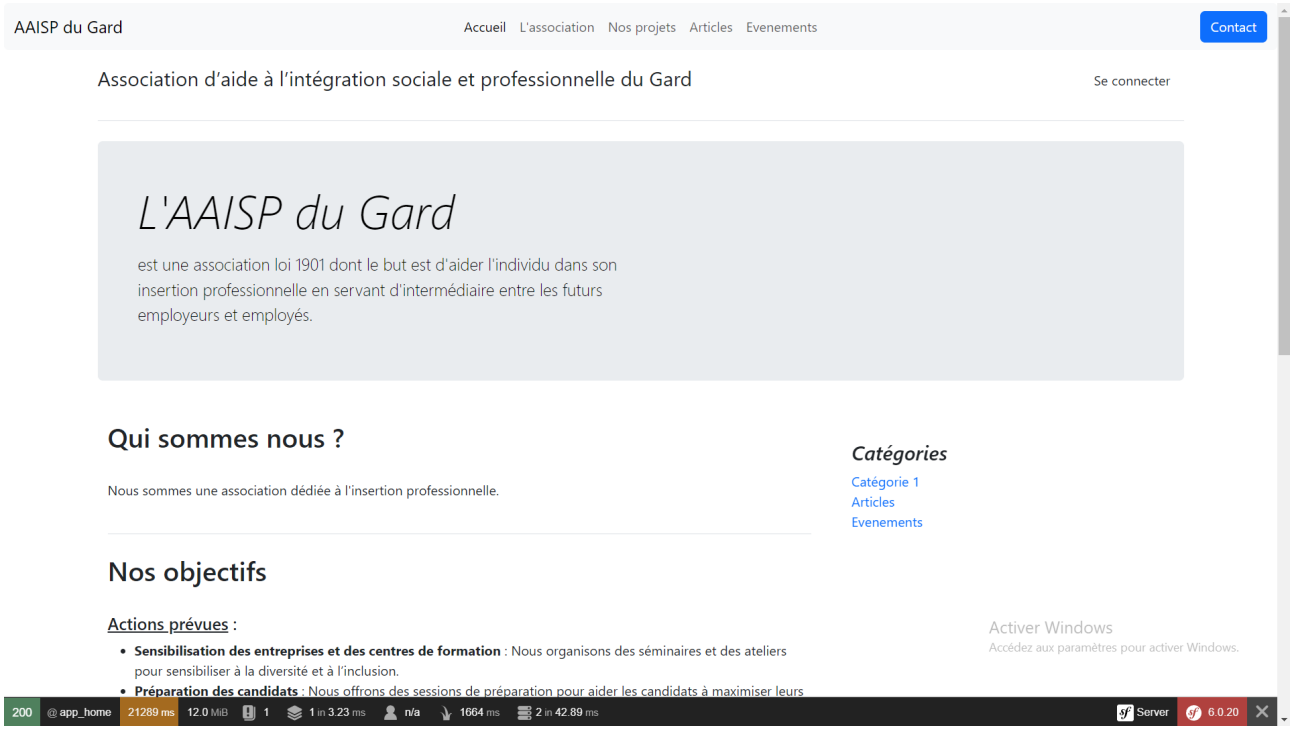
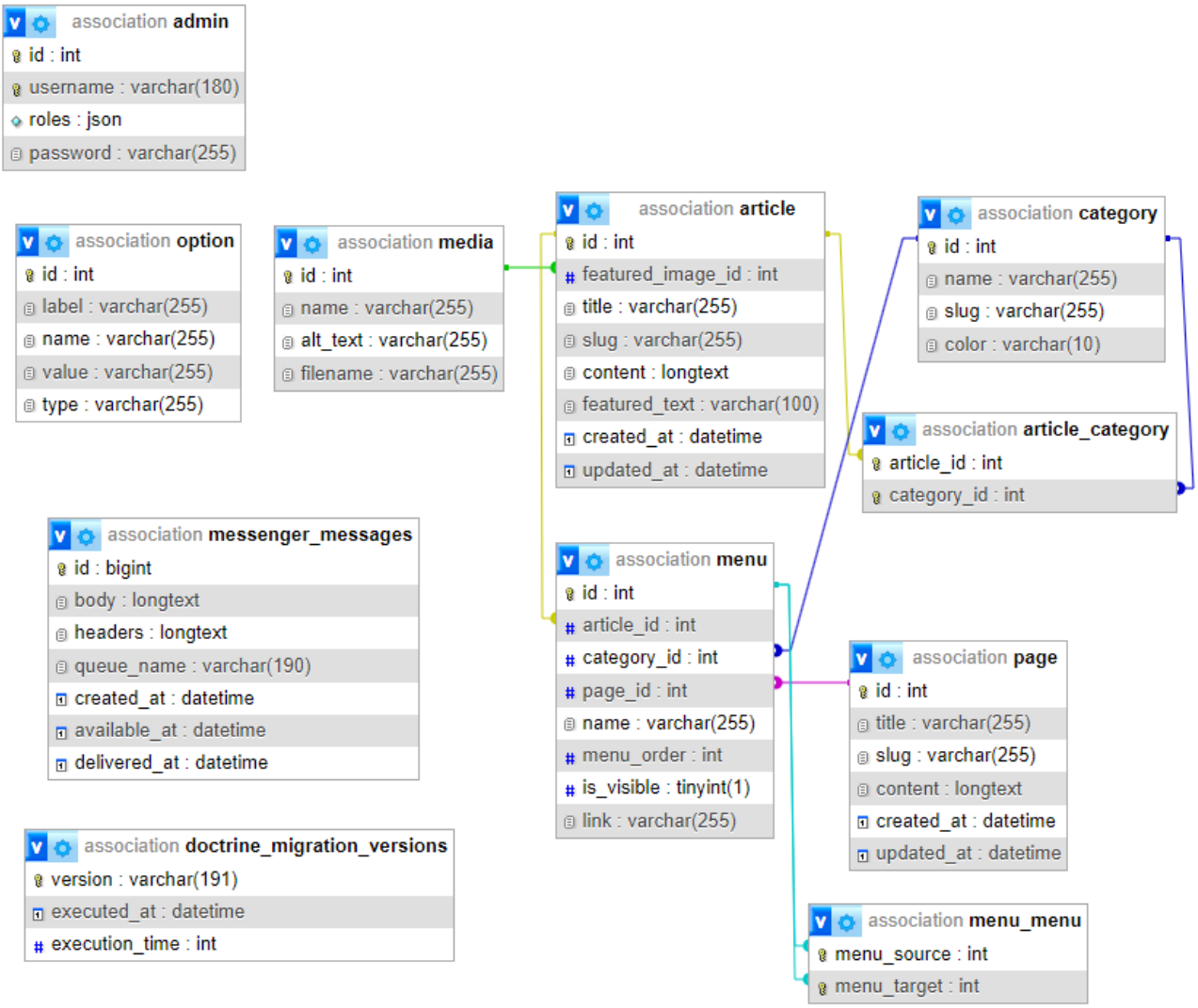
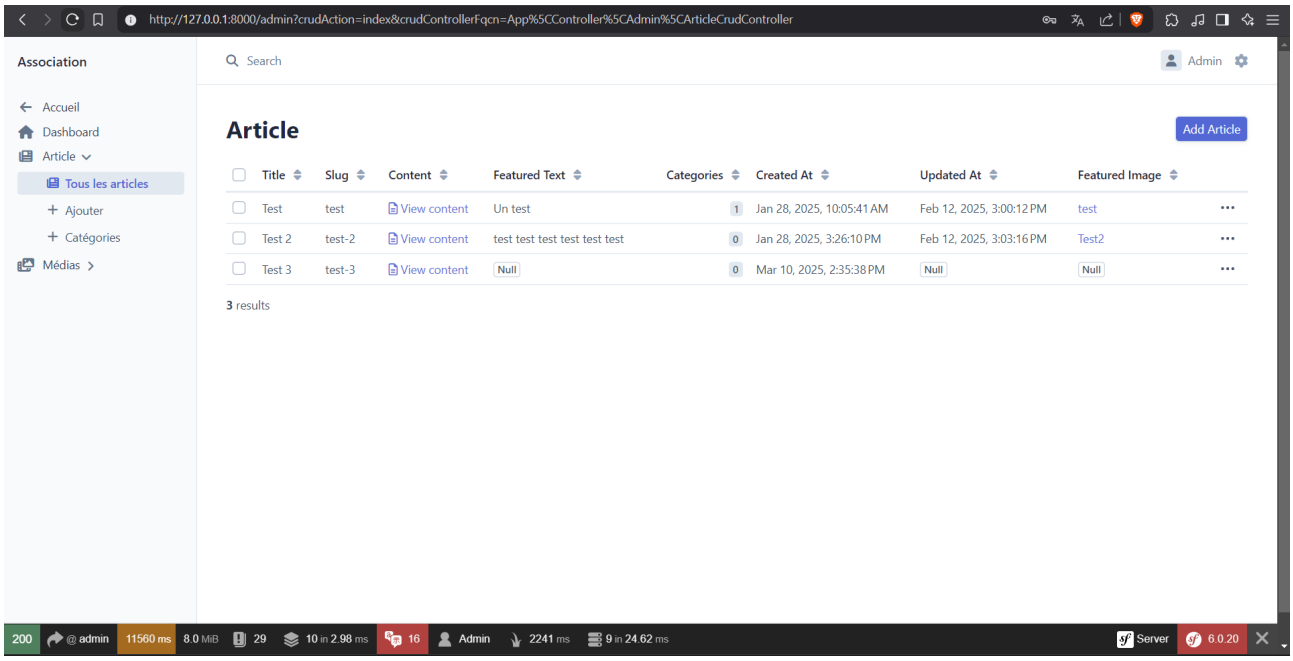


Diagramme de base de données :



Dashboard / Tableau de bord création d'article



Structure d'un projet symfony :