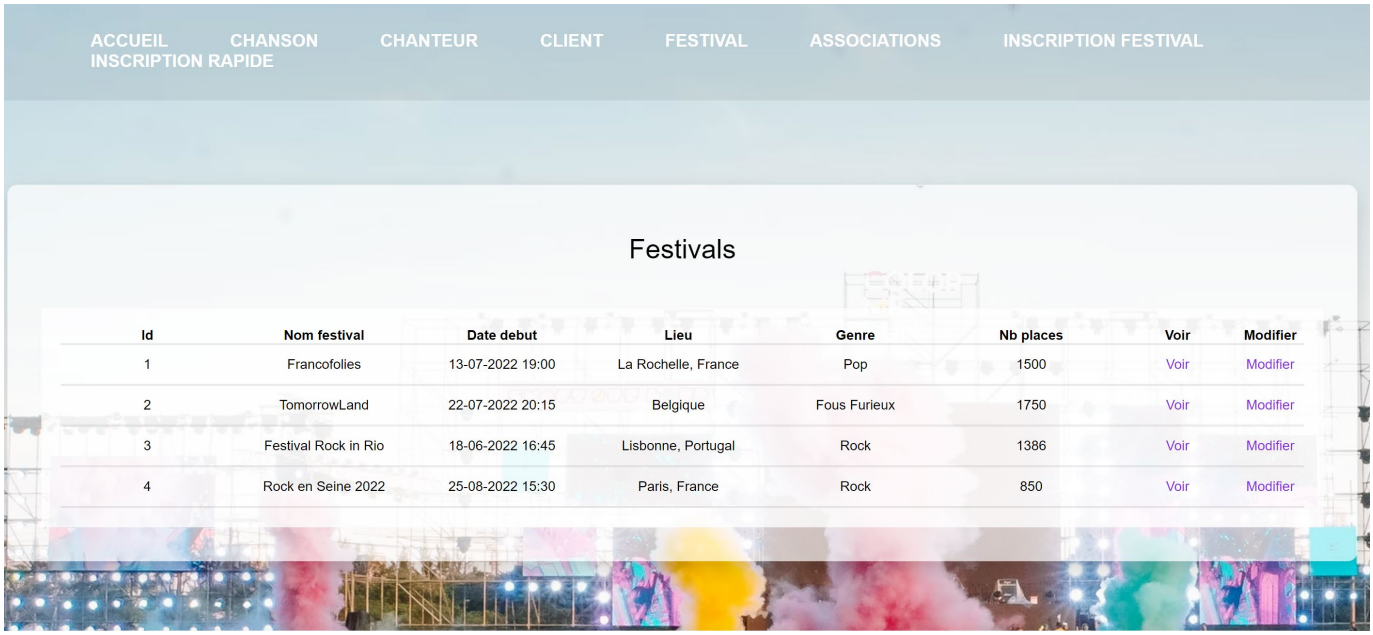
DOMAS-VASSEROT Jade
Option SLAM
Session 2022

Développement d'une application pour de la gestion d'évènements



BTS SERVICES INFORMATIQUES AUX ORGANISATIONS **SESSION 2022**

ANNEXE 7-1-B : Fiche descriptive de réalisation professionnelle (recto)

Épreuve E5 - Conception et développement d'applications (option SLAM) - Coefficient 4

DESCRIPTION D'UNE RÉALISATION PROFESSIONNELLE	**N° réalisation :**
Nom, prénom : DOMAS-VASSEROT Jade	**N° candidat : 02146788571**
Épreuve ponctuelle ☒ **Contrôle en cours de formation** ☐	**Date :** 15 / 04 / 2022

Contexte de la réalisation professionnelle
Une entreprise d'organisation d'évènements souhaite une solution afin de préparer leur gestion de festivals. Nous avons donc créé une application web permettant de gérer les festivals, les chansons, les chanteurs et les client avec le Framework Symfony

Intitulé de la réalisation professionnelle
Développement d'une application pour de la gestion d'évènements

Période de réalisation : 2eme année ..... **Lieu :** Lyon .....
Modalité : ☐ **Seul(e)** ☒ **En équipe**

Compétences travaillées
- ☒ Concevoir et développer une solution applicative
- ☐ Assurer la maintenance corrective ou évolutive d'une solution applicative
- ☒ Gérer les données

Conditions de réalisation[1] (ressources fournies, résultats attendus)
Ressources fournies :
- Sujet de projet, Cours de symfony, accès à la documentation internet

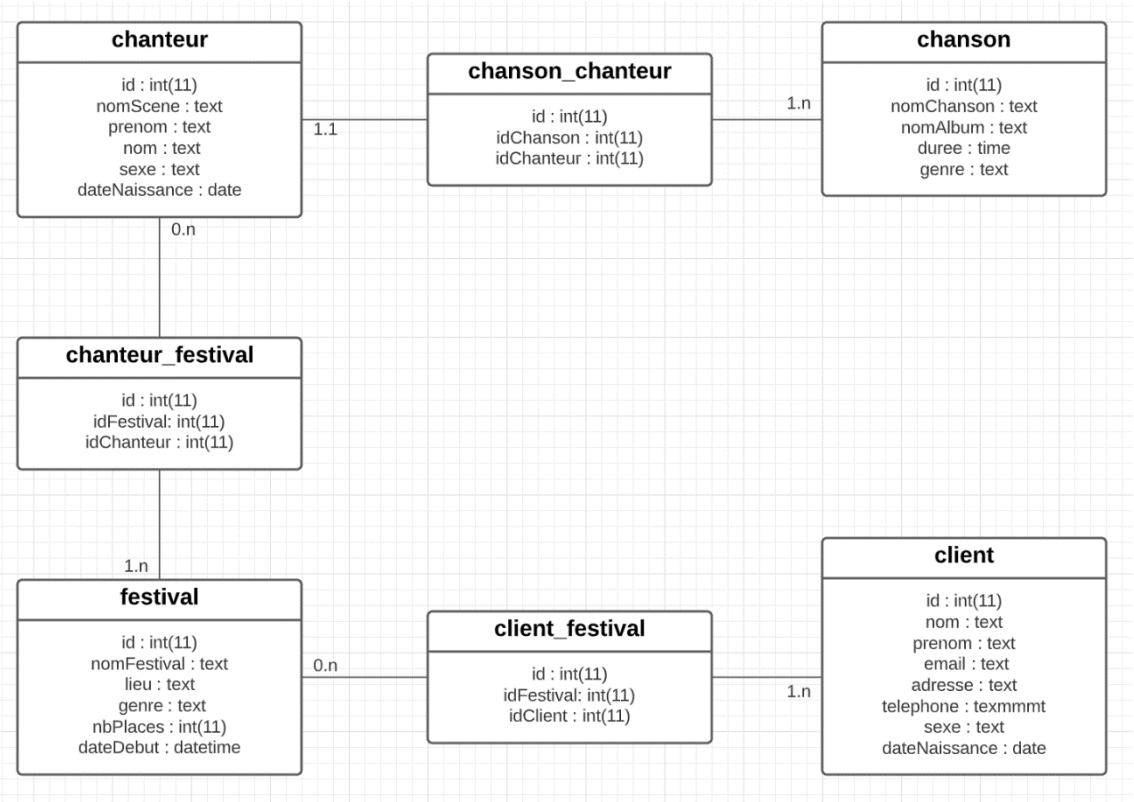
Résultats attendus :
- Conception d'une base de données
- Développement d'une application Symfony en MVC (CRUD)

Description des ressources documentaires, matérielles et logicielles utilisées [2]
Logiciels :
VS code
WampServer (PhpMyAdmin, Apache)
Accès internet
Navigateur Chrome
Installation de Composer
Langages :
Twig pour la vue
SQL
PHP avec le framework Symfony

Modalités d'accès aux productions [3] et à leur documentation [4]
GitHub: https://github.com/JadeDomasVasserot/festival

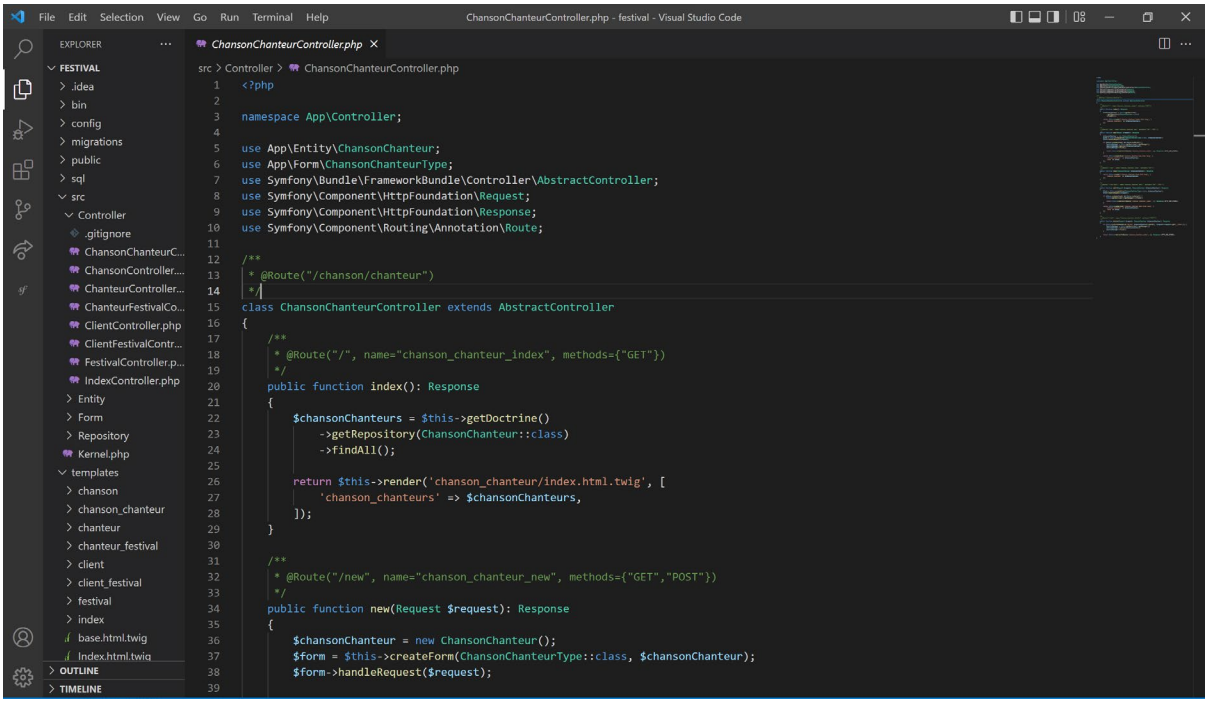
[1] En référence aux *conditions de réalisation et ressources nécessaires* du bloc « Conception et développement d'applications » prévues dans le référentiel de certification du BTS SIO.
[2] Les réalisations professionnelles sont élaborées dans un environnement technologique conforme à l'annexe II.E du référentiel du BTS SIO.
[3] Conformément au référentiel du BTS SIO « *Dans tous les cas, les candidats doivent se munir des outils et ressources techniques nécessaires au déroulement de l'épreuve. Ils sont seuls responsables de la disponibilité et de la mise en œuvre de ces outils et ressources. La circulaire nationale d'organisation précise les conditions matérielles de déroulement des interrogations et les pénalités à appliquer aux candidats qui ne se seraient pas munis des éléments nécessaires au déroulement de l'épreuve.* ». Les éléments peuvent être un identifiant, un mot de passe, une adresse réticulaire (URL) d'un espace de stockage et de la présentation de l'organisation du stockage.
[4] Lien vers la documentation complète, précisant et décrivant, si cela n'a été fait au verso de la fiche, la réalisation professionnelle, par exemple service fourni par la réalisation, interfaces utilisateurs, description des classes ou de la base de données.

Schéma de BDD :



Description de l'activité :

- Conception d'une BDD
- Développement d'une application Symfony



Contexte :

Avec l'arrivée de l'été et la digitalisation de plus en plus commune des applications, une association d'organisations d'évènements a voulu digitaliser son processus de gestion de festivals.

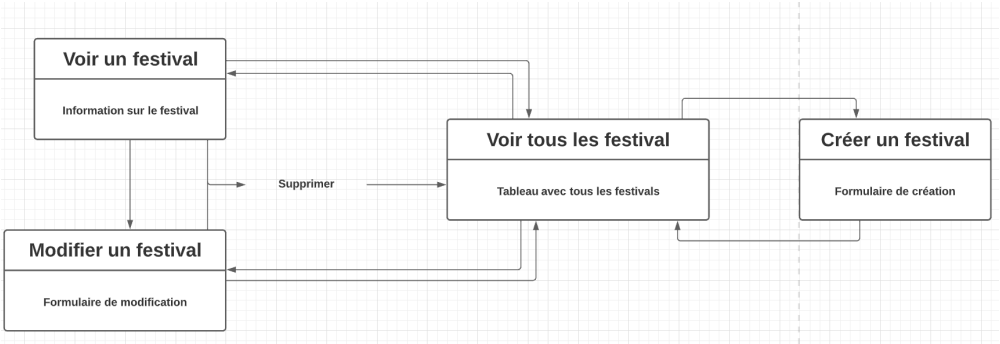
Il nous demande le fait de pouvoir ajouter/modifier/supprimer :

- Un festival
- Un chanteur
- Un client
- Une chanson

  Et de lier chaque information.

Réalisation :

Pour cela, il nous a fallu dans un premier temps concevoir le processus de l'application afin de comprendre les besoins de l'association.
L'association avait besoin de gérer les clients, chansons, festivals, chanteurs.
On a donc conçu la base de données (voir plus haut). Puis, nous avons décidé d'utiliser la méthode CRUD (Create Read Update Deleat) pour gérer les données.
Exemple de process avec le CRUD :



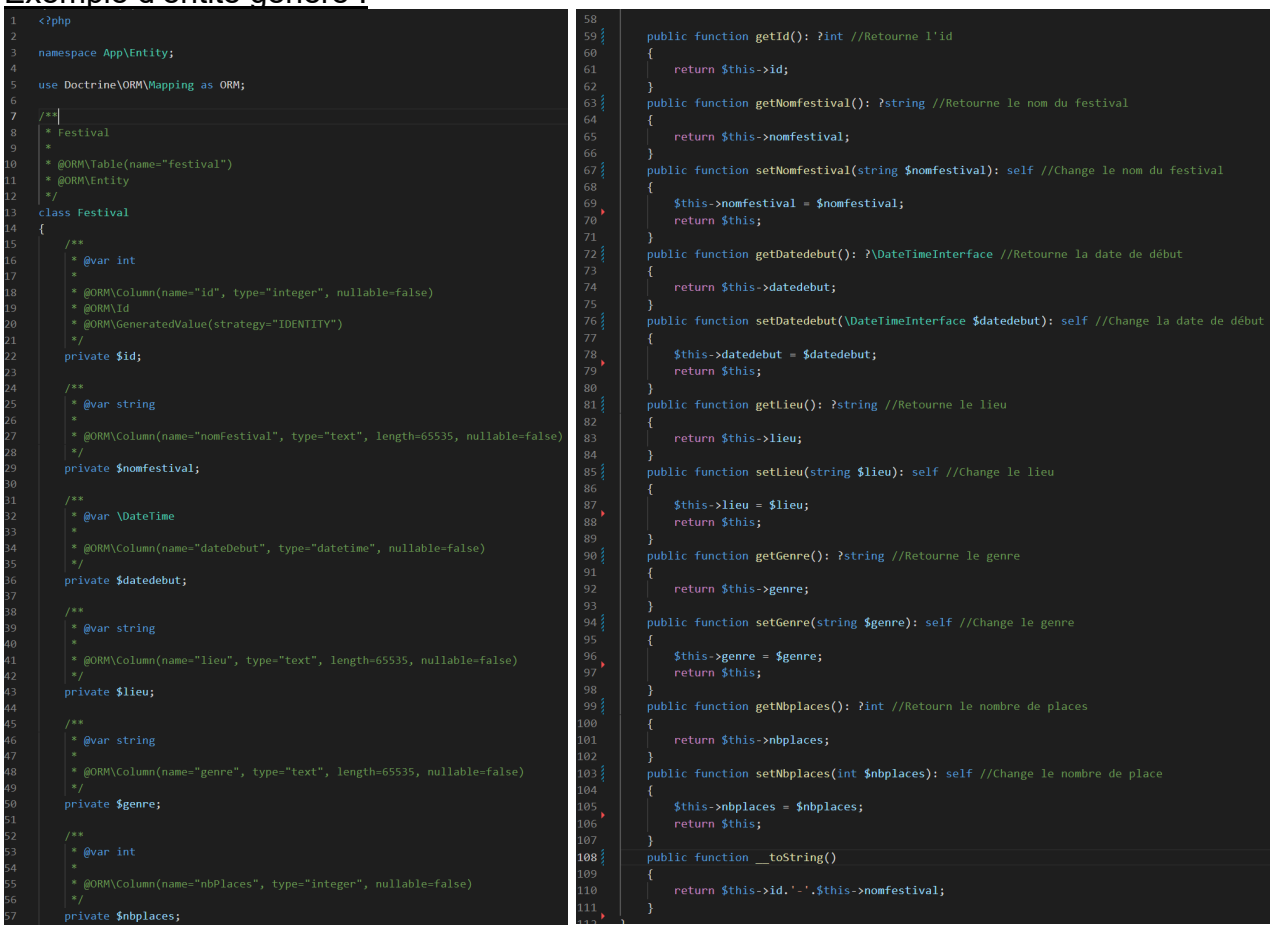
Pour chaque entité dans notre table (voir BDD).
Nous avons par la suite créé un projet Symfony.
composer create-project symfony/website-skeleton
Il a fallu connecter notre base de données dans .env qui a été créée au préalable.
La base de donnée a été créer avant.

.env

```
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
#
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://root:@127.0.0.1:3306/festival?serverVersion=5.7"
#DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=13&charset=utf8"
###< doctrine/doctrine-bundle ###

```

Pour créer les entités à partir de la BDD puis les getters/setters puis les contrôleurs.
Commande utilisée :
php bin/console doctrine:mapping:import "App\Entity" annotation --path=src/Entity
php bin/console make:entity --regenerate App
php bin/console make:controller

Exemple d'entité généré :

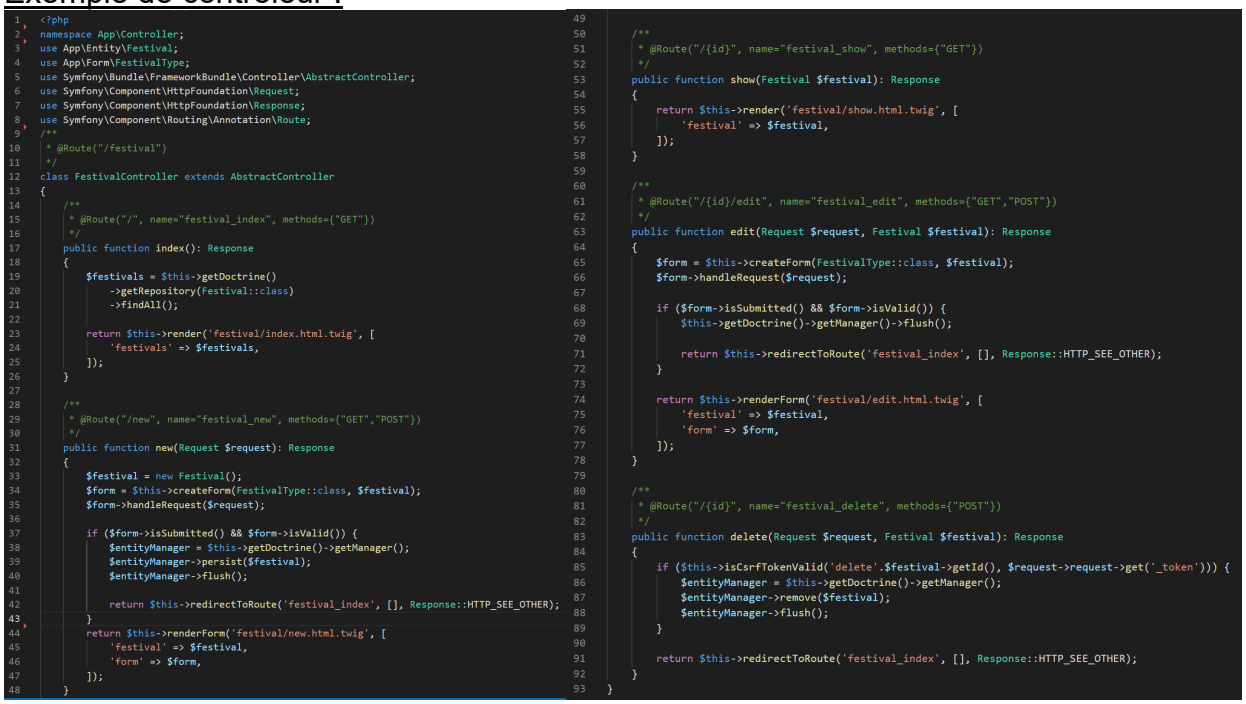


```php
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Festival
 *
 * @ORM\Table(name="festival")
 * @ORM\Entity
 */
class Festival
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="nomFestival", type="text", length=65535, nullable=false)
     */
    private $nomfestival;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="dateDebut", type="datetime", nullable=false)
     */
    private $datedebut;

    /**
     * @var string
     *
     * @ORM\Column(name="lieu", type="text", length=65535, nullable=false)
     */
    private $lieu;

    /**
     * @var string
     *
     * @ORM\Column(name="genre", type="text", length=65535, nullable=false)
     */
    private $genre;

    /**
     * @var int
     *
     * @ORM\Column(name="nbPlaces", type="integer", nullable=false)
     */
    private $nbplaces;

    public function getId(): ?int //Retourne l'id
    {
        return $this->id;
    }
    public function getNomfestival(): ?string //Retourne le nom du festival
    {
        return $this->nomfestival;
    }
    public function setNomfestival(string $nomfestival): self //Change le nom du festival
    {
        $this->nomfestival = $nomfestival;
        return $this;
    }
    public function getDatedebut(): ?\DateTimeInterface //Retourne la date de début
    {
        return $this->datedebut;
    }
    public function setDatedebut(\DateTimeInterface $datedebut): self //Change la date de début
    {
        $this->datedebut = $datedebut;
        return $this;
    }
    public function getLieu(): ?string //Retourne le lieu
    {
        return $this->lieu;
    }
    public function setLieu(string $lieu): self //Change le lieu
    {
        $this->lieu = $lieu;
        return $this;
    }
    public function getGenre(): ?string //Retourne le genre
    {
        return $this->genre;
    }
    public function setGenre(string $genre): self //Change le genre
    {
        $this->genre = $genre;
        return $this;
    }
    public function getNbplaces(): ?int //Retourn le nombre de places
    {
        return $this->nbplaces;
    }
    public function setNbplaces(int $nbplaces): self //Change le nombre de place
    {
        $this->nbplaces = $nbplaces;
        return $this;
    }
    public function __toString()
    {
        return $this->id.'-'.$this->nomfestival;
    }
}
```

Après la création des entitées nous avons fait les contrôleurs avec la méthode du CRUD.
Commande utilisée :

php bin/console make:crud

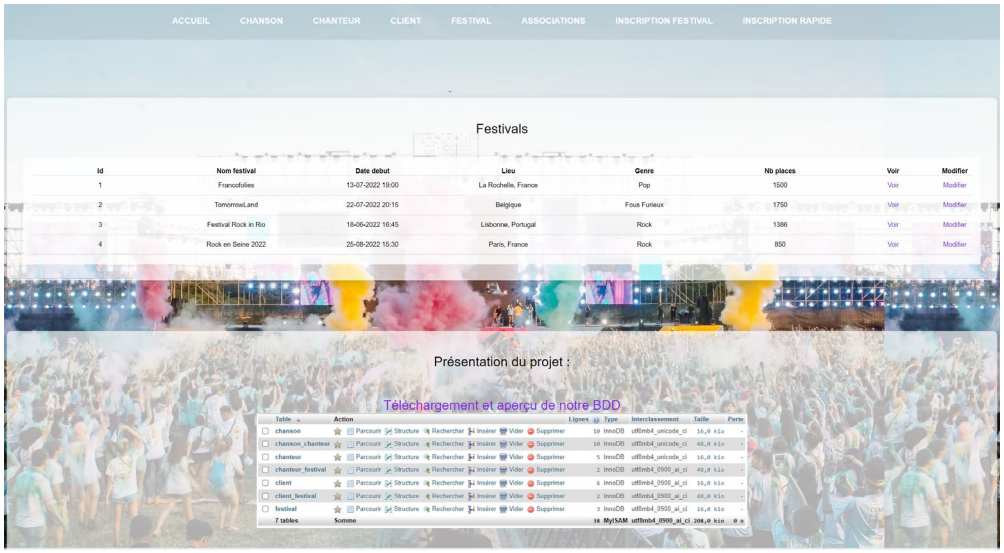
Exemple de contrôleur :



```php
<?php
namespace App\Controller;
use App\Entity\Festival;
use App\Form\FestivalType;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
/**
 * @Route("/festival")
 */
class FestivalController extends AbstractController
{
    /**
     * @Route("/", name="festival_index", methods={"GET"})
     */
    public function index(): Response
    {
        $festivals = $this->getDoctrine()
            ->getRepository(Festival::class)
            ->findAll();

        return $this->render('festival/index.html.twig', [
            'festivals' => $festivals,
        ]);
    }

    /**
     * @Route("/new", name="festival_new", methods={"GET","POST"})
     */
    public function new(Request $request): Response
    {
        $festival = new Festival();
        $form = $this->createForm(FestivalType::class, $festival);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $entityManager = $this->getDoctrine()->getManager();
            $entityManager->persist($festival);
            $entityManager->flush();

            return $this->redirectToRoute('festival_index', [], Response::HTTP_SEE_OTHER);
        }
        return $this->renderForm('festival/new.html.twig', [
            'festival' => $festival,
            'form' => $form,
        ]);
    }

    /**
     * @Route("/{id}", name="festival_show", methods={"GET"})
     */
    public function show(Festival $festival): Response
    {
        return $this->render('festival/show.html.twig', [
            'festival' => $festival,
        ]);
    }

    /**
     * @Route("/{id}/edit", name="festival_edit", methods={"GET","POST"})
     */
    public function edit(Request $request, Festival $festival): Response
    {
        $form = $this->createForm(FestivalType::class, $festival);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $this->getDoctrine()->getManager()->flush();

            return $this->redirectToRoute('festival_index', [], Response::HTTP_SEE_OTHER);
        }

        return $this->renderForm('festival/edit.html.twig', [
            'festival' => $festival,
            'form' => $form,
        ]);
    }

    /**
     * @Route("/{id}", name="festival_delete", methods={"POST"})
     */
    public function delete(Request $request, Festival $festival): Response
    {
        if ($this->isCsrfTokenValid('delete'.$festival->getId(), $request->request->get('_token'))) {
            $entityManager = $this->getDoctrine()->getManager();
            $entityManager->remove($festival);
            $entityManager->flush();
        }

        return $this->redirectToRoute('festival_index', [], Response::HTTP_SEE_OTHER);
    }
}
```

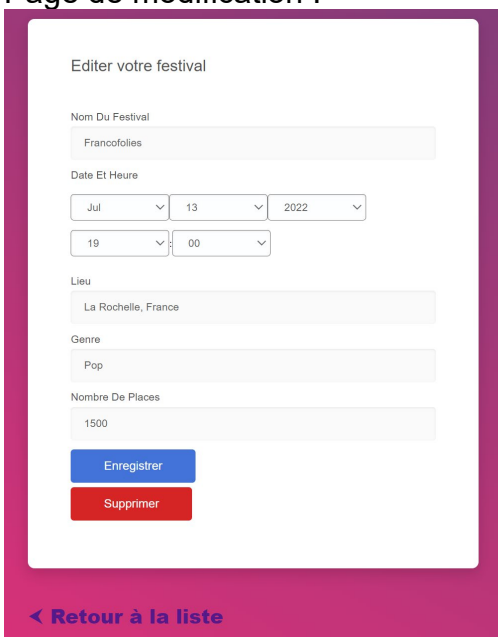
Nous avons donc pu partir de cette base afin de faire les fonctionnalités voulues en modifiant les contrôleurs, routes, entités.
Comme la page d'index où l'on affiche tous les festivals.
En parallèle, nous avons fait les pages twig pour le visuel avec de l'HTML CSS.

Page d'index :

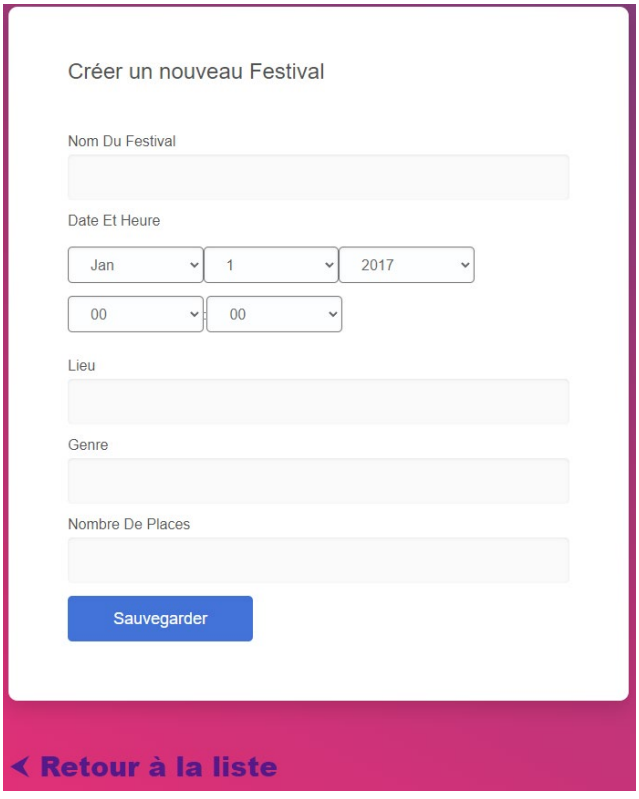


Page index d'un contrôleur :

Page de modification :



Page de création :



Page pour voir une entité :