Document Technique : Installation de l'Infrastructure et présentation du code pour le Projet Workshop 404-hobby-found

Table des matières

```
/templates
    /base.html.twig        # Template de base pour toutes les pages
    /event/list.html.twig  # Page pour la liste des événements
    /user/profile.html.twig # Page profil utilisateur
```



```
const Encore = require('@symfony/webpack-encore');

Encore
    .setOutputPath('public/build/')
    .setPublicPath('/build')
    .addEntry('app', './assets/js/app.js')
    .addStyleEntry('styles', './assets/css/app.scss')
    .enableSassLoader()
    .enablePostCssLoader()
    .enableSourceMaps(!Encore.isProduction())
    .cleanupOutputBeforeBuild()
    .enableVersioning(Encore.isProduction());
```

Installation et configuration de l'infra

Objectif

Configurer deux conteneurs LXC Ubuntu sur Proxmox pour le projet Warshop, incluant un serveur web avec PHPMyAdmin et une base de données MariaDB.

Configuration Générale

- **Adresse IP Publique :** 86.214.56.98
- **Nombre de conteneurs :** 2
    - **Conteneur 1 :** Serveur Web de la solution et de phpmyadmin
    - **Conteneur 2 :** Base de données

Conteneur 1 : Serveur Web de la solution et de PhpMyAdmin

Étapes d'installation

1. **Créer le conteneur LXC :**
    - Ouvrir Proxmox et aller dans l'onglet "Create CT".
    - Remplir les détails du conteneur (Nom, ID, mot de passe, etc.).
    - Choisir l'image de l'OS Ubuntu.
2. **Configurer le réseau :**
    - Assigner une IP privée (par exemple : 192.168.0.2) au conteneur.
    - Assurez-vous que le réseau du conteneur est configuré pour pouvoir accéder à Internet.
3. **Installation des paquets nécessaires :**
    - Accéder au conteneur via SSH.

- Exécuter les commandes suivantes :

```
sudo apt update

sudo apt install nginx php php-fpm php-mysql mariadb-client php-cli php-common php-gd \
php-mbstring php-curl php-xml php-bcmath php-mysql php-curl php-xml php-mbstring \
php-imagick php-zip php-gd composer openssh-server
```

4. **Installer composer :**

```
sudo apt install composer
```

5. **Configure ssh :**

```
sudo nano /etc/ssh/sshd_config
```

- Décommenter et modifier :

```
PermitRootLogin yes
PubkeyAuthentication yes
PasswordAuthentication no
```

- Générer une clé ssh :

```
ssh-keygen -b 4096

cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

- transféré la clé priver dans le pc win :

```
curl ~/.ssh/id_rsa
```

copier/crée ficher sur win dans C://utilisateurs/nom/.ssh/id_rsa

- Ou si même réseau :

```
ssh-copy-id username@ip-adresse
```

6. **Configurer Nginx :**
    - Créer un fichier de configuration pour le site :

```
sudo nano /etc/nginx/sites-available/default
```

    - Ajouter la configuration suivante :

```
server {

    listen 80;

    server_name _;  # Remplacez par votre domaine ou adresse IP publique

    location / {
        try_files $uri $uri/ =404;
    }

    root /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.php index.html;
```

```
    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.3-fpm.sock;
    }

    listen 443 ssl;  # SSL

    ssl_certificate /etc/ssl/certs/ssl-cert-snakeoil.pem;  # Remplacez par votre certific
    ssl_certificate_key /etc/ssl/private/ssl-cert-snakeoil.key;  # Remplacez par votre cl

}
```

    - Activer la configuration et redémarrer Nginx :

```
ln -s /etc/nginx/sites-available/warshop /etc/nginx/sites-enabled/

sudo systemctl restart nginx
```

7. **Installer PHPMyAdmin :**
    - Installer PHPMyAdmin :

```
sudo apt install phpmyadmin
```

- Configurer Nginx pour PHPMyAdmin :

```
server {

    listen 8080;

    server_name _;

    root /usr/share/phpmyadmin;

    index index.php index.html index.htm;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

```
location ~ \.php$ {
    fastcgi_pass unix:/var/run/php/php-fpm.sock;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include fastcgi_params;
}
```

```
location ~ /\.ht {
    deny all;
}

listen 8443 ssl;  # SSL pour HTTPS

ssl_certificate /etc/ssl/certs/ssl-cert-snakeoil.pem;  # Remplacez par votre certificat
ssl_certificate_key /etc/ssl/private/ssl-cert-snakeoil.key;  # Remplacez par votre clé
```

8. **Ajouter la base de donner externe :**

```php
<?php

$i = 0;
$i++;

$cfg['Servers'][$i]['host'] = '192.168.1.249'; // L'adresse IP de votre serveur MariaDB
$cfg['Servers'][$i]['port'] = '3306'; // Le port utilisé par MariaDB (par défaut 3306)
$cfg['Servers'][$i]['user'] = '404'; // Nom d'utilisateur MariaDB
$cfg['Servers'][$i]['password'] = 'j4BM&^R3hn8&'; // Mot de passe MariaDB
$cfg['Servers'][$i]['auth_type'] = 'cookie'; // Type d'authentification
```

- Redémarrer Nginx :

```
sudo systemctl restart nginx
```

9. **Configurer le https crée clé ssl**

```
sudo mkdir -p /etc/nginx/ssl && \
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout /etc/nginx/ssl/selfsigned.key -out /etc/nginx/ssl/selfsigned.crt
```

- Vérification de la création :

```
ls -l /etc/nginx/ssl/
```

- Remodifer :

```
sudo nano /etc/nginx/sites-available/default
```

```
server {

    listen 80 default_server;
    listen [::]:80 default_server;

    server_name _;

    # Redirection vers HTTPS
    return 301 https://$host$request_uri;

}
```

```
server {

    listen 443 ssl default_server;

    listen [::]:443 ssl default_server;


    # Si vous avez un nom de domaine ou une IP, remplacez ici

    server_name 86.214.56.98;  # Remplacez 'example.com' par votre nom de domaine ou IP


    # Emplacement du certificat et de la clé privée

    ssl_certificate /etc/nginx/ssl/selfsigned.crt;

    ssl_certificate_key /etc/nginx/ssl/selfsigned.key;
```

```
    # Protocoles et ciphers recommandés pour la sécurité

    ssl_protocols TLSv1.2 TLSv1.3;

    ssl_ciphers HIGH:!aNULL:!MD5;


    # Paramètres pour améliorer la sécurité HTTPS (facultatif)

    add_header Strict-Transport-Security "max-age=31536000" always;

    add_header X-Frame-Options DENY;

    add_header X-Content-Type-Options nosniff;
```

```
    # Chemin de la racine des fichiers

    root /var/www/html;

    index index.php index.html index.htm index.nginx-debian.html;


    # Fichier à servir par défaut

    location / {

        try_files $uri $uri/ =404;

    }
```

```
    # Traitement des fichiers PHP

    location ~ \.php$ {

        include snippets/fastcgi-php.conf;

        fastcgi_pass unix:/run/php/php8.3-fpm.sock;

    }


    # Désactiver l'accès aux fichiers .htaccess

    location ~ /\.ht {

        deny all;
```

```
    # Gérer les erreurs 404

    error_page 404 /404.html;

    location = /404.html {

        internal;

    }

    error_page 500 502 503 504 /50x.html;

    location = /50x.html {

        internal;

    }
```

- Puis :

```
sudo systemctl restart nginx
```

10. Configurer le https crée clé ssl pour PHPMyAdmin

- Remodifer :

```
sudo nano /etc/nginx/sites-available/phpmyadmin

server {

    listen 8080;

    listen [::]:8080;


    server_name 86.214.56.98;


    # Redirection vers HTTPS

    return 301 https://$server_name:8443$request_uri;

}
```

```
server {

    listen 8443 ssl;

    listen [::]:8443 ssl;


    # Si vous avez un nom de domaine ou une IP, remplacez ici

    server_name 86.214.56.98;  # Remplacez 'example.com' par votre nom de domaine ou IP
```

```
# Emplacement du certificat et de la clé privée

ssl_certificate /etc/nginx/ssl/selfsigned.crt;

ssl_certificate_key /etc/nginx/ssl/selfsigned.key;


# Protocoles et ciphers recommandés pour la sécurité

ssl_protocols TLSv1.2 TLSv1.3;

ssl_ciphers HIGH:!aNULL:!MD5;
```

```
# Paramètres pour améliorer la sécurité HTTPS (facultatif)

add_header Strict-Transport-Security "max-age=31536000" always;

add_header X-Frame-Options DENY;

add_header X-Content-Type-Options nosniff;


# Chemin de la racine des fichiers

root /usr/share/phpmyadmin;

index index.php index.html index.htm;


# Fichier à servir par défaut
```

```
# Fichier à servir par défaut

location / {

    try_files $uri $uri/ =404;

}
```

```
# Traitement des fichiers PHP

location ~ \.php$ {

    fastcgi_pass unix:/var/run/php/php-fpm.sock;

    fastcgi_index index.php;

    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;

    include fastcgi_params;

}


# Désactiver l'accès aux fichiers .htaccess

location ~ /\.ht {

    deny all;
```

```
    # Gérer les erreurs 404

    error_page 404 /404.html;

    location = /404.html {

        internal;

    }

    error_page 500 502 503 504 /50x.html;

    location = /50x.html {

        internal;

    }

}
```

11. Puis :
    sudo systemctl restart nginx

Conteneur 2 : Base de données MariaDB

Étapes d'installation

1. **Créer le conteneur LXC :**
    - Répéter la création du conteneur avec un ID différent (par exemple, 102).
    - Assigner une IP privée :
2. **Installation de MariaDB :**
    - Exécuter les commandes suivantes :

apt update

apt install mariadb-server mariadb-client

3. **Configurer MariaDB :**
    - Sécuriser l'installation :

mysql_secure_installation

    - Créer une base de données et un utilisateur :

CREATE DATABASE 404_hobby_found;

CREATE USER '404'@'%' IDENTIFIED BY 'password'; # Remplacez par vos valeurs

GRANT ALL PRIVILEGES ON 404_hobby_found.* TO '404'@'%';

FLUSH PRIVILEGES;

4. **Configurer l'aces depuis une autre machine réseau:**

Sudo nano / etc/mysql/my.cnf

Ajouter : bind-address = ip contenur 1 ou 0.0.0.0 pour autorizer tout le monde

Si nécessaire :

sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf

Modifiera :

bind-addres =

mysqlx-bind-address =

Puis :

sudo systemctl restart mysql

5. **Installation git sur conteneur 1 :**

Cette documentation couvrira les aspects suivants :

- Architecture de la connexion
- Accès SSH et configuration
- Étapes pour pousser des modifications vers le serveur
- Précautions et bonnes pratiques

Connexion et Poussée des Modifications vers le Serveur

1. Architecture de la Connexion

Les développeurs collaborent sur un projet qui est déployé sur un serveur distant. Chaque développeur est capable de pousser ses modifications locales vers le dépôt distant hébergé sur ce serveur, puis de déployer le projet à partir de ce serveur.

Diagramme de Flux :

12. **Local Development (Git)** : Les développeurs effectuent des modifications en local et les valident via Git.
13. **SSH Access** : Une connexion sécurisée via SSH est utilisée pour se connecter au serveur distant.
14. **Git Push to Remote** : Les modifications sont poussées depuis les machines locales vers le serveur distant hébergeant le dépôt Git.
15. **Deployment Process** : Après le push, un processus automatisé (ou manuel) permet de déployer les dernières modifications sur le serveur de production ou de staging.

2. Accès SSH et Configuration

Prérequis :

- **Accès SSH** au serveur doit être configuré pour chaque développeur. Cela permet de sécuriser les connexions et de garantir que seuls les développeurs autorisés peuvent pousser les modifications vers le serveur.

Étapes pour Configurer l'Accès SSH :

16. **Génération de Clés SSH** : Chaque développeur doit générer une paire de clés SSH (une clé publique et une clé privée) sur sa machine locale. Cela permet une connexion sécurisée sans mot de passe.

Commande pour générer une clé SSH (si elle n'existe pas déjà)

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Ajout de la Clé Publique sur le Serveur : Une fois la clé SSH générée, la clé publique (`~/.ssh/id_rsa.pub`) doit être copiée sur le serveur dans le fichier `~/.ssh/authorized_keys` du compte utilisateur utilisé pour les connexions SSH.

Commande pour copier la clé publique :

```
ssh-copy-id user@your-server-ip
```

Connexion via SSH : Une fois la clé ajoutée, chaque développeur peut se connecter au serveur via SSH sans avoir à saisir de mot de passe.

Commande pour se connecter :

```
ssh user@your-server-ip
```

Configuration du Fichier SSH Config (facultatif) :

Pour simplifier les connexions au serveur, un fichier de configuration SSH peut être utilisé (`~/.ssh/config`). Voici un exemple de configuration :

```
Host your-server
    HostName your-server-ip
    User your-username
    IdentityFile ~/.ssh/id_rsa
```

Cela permet de se connecter facilement au serveur avec la commande suivante :

```
ssh your-server
```

3. Étapes pour Pousser les Modifications sur le Serveur

A. Valider et Pousser des Modifications avec Git :

17. **Étape 1** : Sur la machine locale, les développeurs effectuent leurs modifications et les valident avec Git.

```
git add .
git commit -m "Description des modifications"
```

Étape 2 : Pousser les modifications vers le serveur distant. Assurez-vous d'être sur la bonne branche (par exemple `main` ou `production`).

```
git push origin main
```

Script de Déploiement (.bat) pour Automatiser les Tâches

Sur le serveur, nous avons mis en place un **script `.bat`** qui automatise les tâches suivantes :

- Récupération des modifications poussées sur le dépôt.
- Exécution des commandes nécessaires pour déployer le code.
- Optionnel : Effacement du cache, migration de la base de données, etc.

Exemple du script `.bat` :

Le fichier `.bat` (par exemple `deploy.bat`) pourrait ressembler à ceci :

```
@echo off
cd /path/to/your/project

:: Pull the latest changes from the repository
git pull origin main

:: Clear Symfony cache (if applicable)
php bin/console cache:clear --env=prod

:: Run migrations (if applicable)
php bin/console doctrine:migrations:migrate --no-interaction

:: Restart services (if necessary)
echo "Deployment complete!"
```

Ce script est exécuté automatiquement après chaque push grâce au **hook Git** que nous avons configuré.

Emplacement du script :

- Le fichier `.bat` est stocké dans un répertoire spécifique sur le serveur, par exemple dans le répertoire `/path/to/scripts/deploy.bat`.

4. Hook Git pour Déploiement Automatisé

Nous avons configuré un **hook Git `post-receive`** qui se déclenche automatiquement après qu'un développeur pousse des modifications vers le dépôt sur le serveur. Ce hook exécute le script `.bat` pour déployer automatiquement les modifications.

Création du Hook Git `post-receive` :

18. Sur le serveur, dans le répertoire du dépôt distant (généralement dans `/var/repo/mon_projet.git/hooks/`), créez ou modifiez le fichier `post-receive`.
19. Assurez-vous que le fichier est exécutable :

```
chmod +x post-receive
```

20. Le contenu du fichier `post-receive` doit ressembler à ceci :

```
#!/bin/bash
# Hook Git post-receive pour déployer automatiquement après chaque push

# Chemin vers le répertoire de travail du projet
WORK_DIR="/path/to/your/project"

# Naviguer vers le répertoire de travail
cd $WORK_DIR

# Exécuter le script de déploiement
/path/to/scripts/deploy.bat
```

Fonctionnement du Hook :

- Lorsqu'un `git push` est effectué par un développeur, Git déclenche automatiquement le hook `post-receive`.
- Le hook exécute le script `.bat`, qui s'occupe de tirer les dernières modifications du dépôt et d'effectuer les tâches nécessaires au déploiement.

6. **Config auto impot main lors du push sur serveur :**
   cd /var/repo/404-Hobbyfound.git/hooks
   touch post-receive
   chmod +x post-receive # Rendre le script exécutable
   #!/bin/bash
   GIT_WORK_TREE=/var/www/html git checkout -f main

   sudo chown -R www-data:www-data /var/www/html
   sudo chmod -R 755 /var/www/html

Configuration de la Box pour le Routage

1. **Accéder à l'interface de gestion de la box.**
2. **Configurer le routage :**
   - Ajouter des règles NAT pour rediriger les ports vers les adresses IP des conteneurs LXC sur la boxe.
   - Les règles ajouter :
     - **Port 80** → 192.168.1.248:80 (Conteneur 1)
     - **Port 443** → 192.168.248.248:443 (Conteneur 1)

- **Port 8080** → 192.168.1.248:8080 (Conteneur 1)
- **Port 8443** → 192.168.1.248:8443 (Conteneur 1)
- **Port 418** → 192.168.1.248:22 (Conteneur 1)
- **x** → 192.168.1.249:3306 (Conteneur 2, utilisable uniquement en interne)

Vérification

- Vérifier l'accès au serveur web via l'IP publique sur les ports http://86.214.56.98:80 et https://86.214.56.98:443.
- Accéder à PHPMyAdmin via http://86.214.56.98:8080 et https://86.214.56.98:8443.
- Tester la connexion à la base de données depuis le conteneur 1 avec :

mysql -u user -p -h 192.168.0.3

Conclusion

Pour ce connecter au gite pour pouch le code via git ou ce connecter en ssh sur la 1er machine via le port 418 il faut une clé ssh.

Pour y accéder :

Site 404 : http://86.214.56.98:80

Site PHPMyAdmin : https://86.214.56.98:8443

Présentation code :

Documentation Technique : Front-End

1. Technologies Utilisées

Le front-end de ce projet repose sur les technologies suivantes :

- **HTML5** : Structure de la page web.
- **CSS3 & SCSS (optionnel)** : Styles et mise en page, possibilité d'utiliser SCSS pour une gestion plus avancée des styles.
- **Bootstrap 4.5.2** : Framework CSS pour un design responsive et des composants pré-stylisés.
- **JavaScript (ES6)** : Pour les interactions dynamiques sur la page.
- **Webpack Encore (Symfony)** : Gestionnaire de bundling et de minification des fichiers JavaScript et CSS.

- **JQuery** : Bibliothèque JavaScript pour simplifier les manipulations du DOM.
- **Popper.js & Bootstrap JS** : Gestion des composants dynamiques Bootstrap (dropdowns, modals, tooltips, etc.).

Voici la documentation technique pour le **front-end** de ton projet. Cette documentation couvre la structure du front-end, les technologies utilisées, le processus de développement, les bonnes pratiques, et le déploiement des fichiers front-end.

Documentation Technique : Front-End

1. Technologies Utilisées

Le front-end de ce projet repose sur les technologies suivantes :

- **HTML5** : Structure de la page web.
- **CSS3 & SCSS (optionnel)** : Styles et mise en page, possibilité d'utiliser SCSS pour une gestion plus avancée des styles.
- **Bootstrap 4.5.2** : Framework CSS pour un design responsive et des composants pré-stylisés.
- **JavaScript (ES6)** : Pour les interactions dynamiques sur la page.
- **Webpack Encore (Symfony)** : Gestionnaire de bundling et de minification des fichiers JavaScript et CSS.
- **JQuery** : Bibliothèque JavaScript pour simplifier les manipulations du DOM.
- **Popper.js & Bootstrap JS** : Gestion des composants dynamiques Bootstrap (dropdowns, modals, tooltips, etc.).

2. Structure du Front-End

A. Fichiers HTML

Les fichiers HTML sont gérés via des **templates Twig** (fichiers `.html.twig`) dans Symfony. Les composants récurrents (header, footer, navigation) sont séparés dans des templates pour la réutilisation.

Exemple de structure :

```
/templates
    /base.html.twig       # Template de base pour toutes les pages
    /event/list.html.twig  # Page pour la liste des événements
    /user/profile.html.twig # Page profil utilisateur
```

B. Feuilles de style (CSS/SCSS)

Les styles sont organisés et gérés par **Bootstrap** et des **fichiers CSS/SCSS personnalisés**. Nous utilisons **Webpack Encore** pour gérer et compiler ces fichiers.

Exemple de structure :

```
/assets
    /css
        /app.scss         # Fichier SCSS principal qui inclut Bootstrap et des styles per
        /_variables.scss  # Variables SCSS personnalisées pour gérer les couleurs et la t
        /_custom.scss     # Fichier de style customisé pour l'interface utilisateur
```

Le fichier `app.scss` inclut Bootstrap et d'autres styles personnalisés :

```
// Import Bootstrap SCSS
@import '~bootstrap/scss/bootstrap';

// Import des variables personnalisées
@import 'variables';

// Import des styles customisés
@import 'custom';
```

C. Fichiers JavaScript

Les fichiers JavaScript sont également gérés via **Webpack Encore**. Nous utilisons principalement **JQuery** et les scripts Bootstrap pour les interactions dynamiques.

Exemple de structure :

```
/assets
    /js
        /app.js          # Fichier JavaScript principal pour les fonctionnalités général
        /events.js       # Gestion des événements front-end spécifiques
```

Exemple de fichier `app.js` :

```
// Importation des dépendances
import $ from 'jquery';
import 'bootstrap';

// Code JS personnalisé
$(document).ready(function() {
    console.log("Front-end initialized!");
});
```

D. Webpack Encore

Webpack Encore est utilisé pour compiler les fichiers SCSS et JavaScript en bundles optimisés pour la production.

- Fichiers de configuration :
    - `webpack.config.js` : Configuration principale de Webpack Encore.

Exemple de configuration `webpack.config.js` :

```
const Encore = require('@symfony/webpack-encore');

Encore
    .setOutputPath('public/build/')
    .setPublicPath('/build')
    .addEntry('app', './assets/js/app.js')
    .addStyleEntry('styles', './assets/css/app.scss')
    .enableSassLoader()
    .enablePostCssLoader()
    .enableSourceMaps(!Encore.isProduction())
    .cleanupOutputBeforeBuild()
    .enableVersioning(Encore.isProduction());
```

3. Développement et Bonnes Pratiques

A. Développement du Front-End

21. **Installation des Dépendances** : Avant de commencer le développement front-end, les dépendances nécessaires (Bootstrap, JQuery, etc.) doivent être installées avec **npm**.

Commandes à exécuter :

```
npm install
```

Compilation des Fichiers : Webpack Encore est utilisé pour compiler les fichiers SCSS et JavaScript.

Pour **compiler les fichiers** en mode développement (avec les sourcemaps activés), exécutez :

```
npm run dev
```

Pour **compiler en mode production** (avec minification et versioning), utilisez :

```
npm run build
```

22. **Structure Modulaire** :

- Utiliser des composants réutilisables : Lorsque vous travaillez sur des fonctionnalités front-end, essayez de découper les styles et le JavaScript en composants réutilisables (ex : cards, modals).
- Modularisation des styles : Utilisez des **partiels SCSS** (fichiers commençant par un _) pour gérer des segments spécifiques du style (ex : `_buttons.scss` pour les styles de boutons, `_cards.scss` pour les cartes).

B. Bonnes Pratiques

23. **Responsive Design** : Toujours tester les composants sur différentes tailles d'écran (ordinateurs, tablettes, mobiles) en utilisant les classes Bootstrap (`col-md-*`, `col-lg-*`, `d-none`, etc.).
24. **Variables SCSS** : Utilisez les variables SCSS pour centraliser la gestion des couleurs, des espacements et des tailles de police. Cela permet de changer facilement le thème du site sans modifier chaque fichier.

Exemple de variables SCSS personnalisées :

```scss
$primary-color: #007bff;
$secondary-color: #6c757d;
$font-family-base: 'Roboto', sans-serif;
```

Documentation Technique : Back-End

1. Technologies Utilisées

Le back-end de ce projet repose sur le **framework Symfony** et est conçu pour gérer la logique métier, la communication avec la base de données, ainsi que l'API ou les routes HTTP. Voici les technologies principales utilisées :

- **PHP 8.x** : Langage de programmation utilisé par Symfony pour gérer les requêtes serveur.
- **Symfony 5.x/6.x** : Framework PHP utilisé pour structurer l'application et gérer le routage, la logique métier, et les services.
- **Doctrine ORM** : Système de gestion de base de données utilisé pour interagir avec la base de données relationnelle.
- **MySQL** : Base de données relationnelle pour stocker les informations.
- **Twig** : Moteur de templates utilisé pour générer les vues côté serveur.

2. Structure du Projet Symfony

Voici la structure typique d'un projet Symfony, avec un focus sur les dossiers et fichiers importants pour le développement back-end :

```
/src
    /Controller      # Contient les contrôleurs qui gèrent les requêtes HTTP
    /Entity          # Contient les entités (modèles) liées à la base de données
    /Repository      # Contient les classes de repository pour les requêtes personnalisé
    /Service         # Contient les services pour gérer la logique métier
/config              # Contient la configuration de l'application (routing, services, ba
/migrations          # Contient les fichiers de migration de la base de données
```

A. Contrôleurs (/src/Controller)

Les contrôleurs sont responsables de la gestion des requêtes HTTP et du renvoi des réponses appropriées (pages HTML, JSON, redirections, etc.). Chaque méthode du contrôleur correspond à une action, comme afficher une page ou traiter un formulaire.

Exemple de contrôleur :

```php
namespace App\Controller;

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

class EventController extends AbstractController
{
    /**
     * @Route("/events", name="event_list")
     */
    public function list(): Response
    {
        $events = $this->getDoctrine()->getRepository(Event::class)->findAll();
        return $this->render('event/list.html.twig', ['events' => $events]);
    }
}
```

B. Entités (/src/Entity)

Les entités représentent les modèles de données qui correspondent aux tables dans la base de données. Chaque entité est une classe PHP avec des propriétés qui correspondent aux colonnes de la table.

Exemple d'entité Event :

```php
use Doctrine\ORM\Mapping as ORM;

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

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

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

C. Repository (/src/Repository)

Les repositories contiennent la logique pour interagir avec la base de données. Symfony génère automatiquement des repositories pour chaque entité, mais il est possible d'ajouter des méthodes personnalisées pour effectuer des requêtes plus complexes.

Exemple d'une méthode personnalisée dans un repository :

```php
use App\Entity\Event;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

class EventRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Event::class);
    }

    public function findUpcomingEvents()
    {
        return $this->createQueryBuilder('e')
            ->andWhere('e.date > :today')
            ->setParameter('today', new \DateTime())
            ->orderBy('e.date', 'ASC')
            ->getQuery()
            ->getResult();
    }
```

D. Services (/src/Service)

Les services contiennent la logique métier qui ne doit pas nécessairement être liée aux contrôleurs ou aux entités. Ils permettent de séparer les responsabilités et de faciliter les tests unitaires.

Exemple de service :

```php
namespace App\Service;

class EventNotifier
{
    public function sendNotification($event)
    {
        // Logique pour envoyer une notification concernant un événement
    }
}
```

3. Configuration

La configuration du projet Symfony se fait principalement dans le dossier `/config`. Ce dossier contient des fichiers YAML qui définissent les routes, les services, et la connexion à la base de données.

A. Configuration de la Base de Données

Le fichier de configuration principal pour la base de données se trouve dans `/config/packages/doctrine.yaml`. Voici un exemple de configuration pour une base de données MySQL :

```yaml
doctrine:
    dbal:
        url: '%env(resolve:DATABASE_URL)%'
    orm:
        auto_generate_proxy_classes: true
        naming_strategy: doctrine.orm.naming_strategy.underscore
        auto_mapping: true
```

La variable `DATABASE_URL` est définie dans le fichier `.env` du projet :

```
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name"
```

B. Configuration des Routes

Les routes sont définies dans `/config/routes.yaml` ou directement via les annotations dans les contrôleurs.

Exemple de définition d'une route dans `routes.yaml` :

```yaml
event_list:
    path: /events
    controller: App\Controller\EventController::list
```

4. Processus de Développement

A. Installation des Dépendances

Avant de commencer à développer, assurez-vous que toutes les dépendances PHP sont installées via Composer.

B. Gestion des Entités et Migrations

1. Création d'une nouvelle entité :

Lorsque vous devez créer une nouvelle entité (par exemple, pour une nouvelle table dans la base de données), utilisez la commande `make:entity` de Symfony.

Cela vous guide pour définir les propriétés de l'entité, qui seront automatiquement ajoutées sous forme de colonnes dans la base de données.

2. Migrations :

Après avoir créé ou modifié une entité, vous devez générer une migration pour mettre à jour la base de données :

Puis, appliquez la migration à la base de données :

C. Débogage

Pour déboguer les problèmes, Symfony propose un **profiler** et une **barre de débogage** disponibles lorsque vous accédez à l'application en mode développement.

- Pour visualiser le profiler, accédez à une page web et cliquez sur la barre de débogage en bas de la page.
- Utilisez également les logs générés dans le fichier `/var/log/dev.log` pour investiguer les problèmes.

5. Code Critique :

1. Img Chat GPT:

- Lors de la création d'un club, il est possible d'utiliser ChatGPT pour générer une image servant de logo, en appelant le contrôleur qui gère l'intégration de ChatGPT.

```
    //generer img?
    $imgGen = false;
    if ($form->get('genererImg')->getData()) {
        // Perform the action for generating the image
        $imgGen = true;
        // e.g., calling another service or generating an image
    }
    // Set additional properties for the club
    $club->setCreatedAt(new \DateTimeImmutable());
    $club->setLastModifiedAt(new \DateTimeImmutable());
    // Persist the new club entity to the database
    $em->persist($club);
    $em->flush();
    $this->addFlash('success', 'Club créé!');
    // Check if the checkbox was checked
    if ($imgGen) {
        return $this->redirectToRoute('app_gpt', ['id' => $club->getId()]);
    }
    // Redirect to the homepage (or your desired route)
    return $this->redirectToRoute('accueil');
}
// Render the form if it is not submitted or not valid
return $this->render('clubs/addClub.html.twig', [
    'clubForm' => $form->createView(),
]);
```

- Voici la partie concernant la connexion à l'API DALL-E de ChatGPT. Nous allons configurer l'API en utilisant notre clé API de ChatGPT.

```php
class GptController extends AbstractController
{
    #[Route('/gpt/{id}', name: 'app_gpt')]
    public function index(
        EntityManagerInterface $em,
        ClubRepository $clubRepository,
        $id,


    ): Response
    {

        try {
            $club = $clubRepository->find($id);
            $msg = file_get_contents(__DIR__ . '/prompt.txt');
            $prompt = null;
            if ($club) {

                $values = [
                    '{clubTitre}' => $club->getName()
                ];
                $prompt = strtr($msg, $values);
                ////////////// A PARTIR DICI ON UTILISE LE MOTEUR DE GPT

                // Clés d'API de ChatGPT

                $apiKey = $this->getParameter('api_key');
                // Configurer le client
                $client = OpenAI::client($apiKey);
```

- L'image sera créée à partir du prompt que l'on fournira à DALL-E. Le contrôleur enregistrera ensuite l'URL générée dans un dossier à l'aide de la fonction `saveImageFromUrl`, puis enregistrera cette URL dans la base de données avec la méthode `setClubImg`.

```php
    //Crée une image à partir d'un prompt
    $response = $client->images()->create([
        'model' => 'dall-e-2',
        'prompt' => $prompt,
        'n' => 1,
        'size' => '1024x1024',
        'response_format' => 'url',
    ]);

    $response->created;
    $url = null;

    foreach ($response->data as $data) {
        $url = $data->url; // 'https://oaidalleapiprodscus.blob.core.windows.net/private/.
        $data->b64_json; // null
    }
    ////////////////////////////////////////////////////////////////////////


    $saveDirectory = $this->getParameter('kernel.project_dir') . '/public/uploads/clubs';
    $fileName = $club->getId() . '_club_image.jpg';

    $this->saveImageFromUrl($url,$saveDirectory,$fileName);

    $club->setclubImg($fileName);
    $em->persist($club);
    $em->flush();
```

- Cette fonction nous permet d'enregistrer l'image générée.

```php
public function saveImageFromUrl(string $imageUrl, string $saveDirectory, string $fileName): ?string
{

    // Ensure the directory exists or create it
    if (!file_exists($saveDirectory) && !mkdir($saveDirectory, 0777, true) && !is_dir($saveDirectory)) {
        throw new \RuntimeException(sprintf('Directory "%s" was not created', $saveDirectory));
    }

    try {
        // Get the image content from the URL
        $imageContent = file_get_contents($imageUrl);
        if ($imageContent === false) {
            throw new \Exception("Could not fetch the image from the URL.");
        }

        // Save the image to the desired location
        $filePath = $saveDirectory . '/' . $fileName;
        file_put_contents($filePath, $imageContent);

        return $filePath; // Return the saved file path for further use
    } catch (\Exception $e) {
        // Handle any error, log it, or throw an exception
        throw new FileException("Failed to save the image: " . $e->getMessage());
    }
}
```

- Voici le prompt pour chat Gpt .

```
"Create a visually appealing and minimalistic thumbnail for a university club. The style should be clean, modern, and slightly abstract, using bold lines and flat design. The color palette should be soft and cohesive, with a dominant main color and complementary tones. The background should be simple with subtle gradients or shapes. For the {clubTitre} (be carefull the previous name is most likely in French)club, include a central icon or symbol representing the club's focus, designed in the same flat and modern style. Avoid overly complex details, and ensure the overall aesthetic is consistent across thumbnails for various clubs."
```

Utilisation de FullCalendar

Nous avons créé une page permettant d'afficher l'ensemble des évènements entrés en bdd sur un calendrier. Pour se faire, on commence par récupérer les évènements depuis notre contrôleur. Ensuite, on formate un doc JSON qui réunit l'ensemble des informations que l'on souhaitera exploiter dans notre calendrier (l'id de l'event, le nom du club associé, le titre de l'event etc ...). Enfin, on envoit notre JSON en parametre à notre TWIG et on génère la page :

```php
    #[Route('/events', name: 'events')]
public function events(EventRepository $eventRepository): Response
{
    $events = $eventRepository->findAll();

    $eventData = [];
    foreach ($events as $event) {
        $eventData[] = [
            'id'=> $event->getId(),
            'title' => $event->getTitle(),
            'start' => $event->getDate()->format('Y-m-d\TH:i:s'),
            'description' => $event->getDescription(),
            'club' => $event->getClub()->getName(),
        ];
    }

    return $this->render('event/calendar.html.twig', [
        'events' => json_encode($eventData),
    ]);
}
```

On execute ensuite un script dans notre twig qui va déclencher l'utilisation de notre librairie FullCalendar:

```
<!-- Events Data -->
<script id="events-data" type="application/json">{{ events|json_encode|raw }}</script>
```

On commence par instancier le calendar dans l'élément du DOM ciblé. Ensuite, on récupère les données de notre JSON en sous forme de string. Nous somme ici obligés de les parser 2 fois consécutives afin de les retransformer en JSON. Enfin, au moment de l'instantiation de notre calendar, nous lui ajoutons ces évènements. Il seront donc utilisables à souhait.

```
document.addEventListener('DOMContentLoaded', function() {
    const eventsDataElement = document.getElementById('events-data');

    if (eventsDataElement) {
        try {
            console
            let events = JSON.parse(eventsDataElement.textContent);
            let eventsparsed = JSON.parse(events);
            let calendarEl = document.getElementById('calendar');
            let calendar = new Calendar(calendarEl, {
                plugins: [dayGridPlugin],
                initialView: 'dayGridMonth',
                events : eventsparsed,
```

Pour finir, nous ajoutons un eventListener intégré à la librairie : evenCLick. Cela permettra de rendre clicable les évènements affichés sur notre calendrier. On crée ensuite un modal avec bootstrap pour générer un modal au clic.

```
    eventClick : function(info) {

        // Format the date in a user-friendly way
        let options = {
            weekday: 'long',
            year: 'numeric',
            month: 'long',
            day: 'numeric',
            hour: '2-digit',
            minute: '2-digit',
            hour12: true
        };
        let formattedDate = new Intl.DateTimeFormat('fr-FR', options).format(info.event.start);
        console.log(formattedDate);

        // Populate modal with event details
        document.getElementById('modalTitle').innerText = info.event.title;
        document.getElementById('modalDescription').innerText = info.event.extendedProps.description;
        document.getElementById('modalDate').innerText = formattedDate;
        document.getElementById('modalClub').innerText = info.event.extendedProps.club;

        let modal = new bootstrap.Modal(document.getElementById('eventModal'));
        modal.show();
    }

});
```