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

Table des matières

[Installation et configuration de l’infra 3](#_Toc178318116)

[Objectif 3](#_Toc178318117)

[Configuration Générale 3](#_Toc178318118)

[Conteneur 1 : Serveur Web de la solution et de PhpMyAdmin 3](#_Toc178318119)

[Conteneur 2 : Base de données MariaDB 9](#_Toc178318120)

[**Connexion et Poussée des Modifications vers le Serveur** 10](#_Toc178318121)

[**1. Architecture de la Connexion** 10](#_Toc178318122)

[**Diagramme de Flux :** 11](#_Toc178318123)

[**2. Accès SSH et Configuration** 11](#_Toc178318124)

[**Prérequis :** 11](#_Toc178318125)

[**Étapes pour Configurer l'Accès SSH :** 11](#_Toc178318126)

[**Configuration du Fichier SSH Config (facultatif) :** 12](#_Toc178318127)

[**3. Étapes pour Pousser les Modifications sur le Serveur** 12](#_Toc178318128)

[**A. Valider et Pousser des Modifications avec Git :** 12](#_Toc178318129)

[**Script de Déploiement (.bat) pour Automatiser les Tâches** 12](#_Toc178318130)

[**Exemple du script .bat :** 13](#_Toc178318131)

[**Emplacement du script :** 13](#_Toc178318132)

[**4. Hook Git pour Déploiement Automatisé** 13](#_Toc178318133)

[**Création du Hook Git post-receive :** 13](#_Toc178318134)

[**Fonctionnement du Hook :** 14](#_Toc178318135)

[Configuration de la Box pour le Routage 14](#_Toc178318136)

[Vérification 15](#_Toc178318137)

[Conclusion 15](#_Toc178318138)

[Présentation code : 15](#_Toc178318139)

[**Documentation Technique : Front-End** 15](#_Toc178318140)

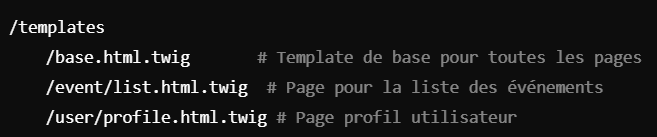
[**1. Technologies Utilisées** 15](#_Toc178318141)

[**Documentation Technique : Front-End** 16](#_Toc178318142)

[**1. Technologies Utilisées** 16](#_Toc178318143)

[**2. Structure du Front-End** 16](#_Toc178318144)

[**A. Fichiers HTML** 16](#_Toc178318145)

[B. Feuilles de style (CSS/SCSS) 17](#_Toc178318146)

[**C. Fichiers JavaScript** 17](#_Toc178318147)

[**D. Webpack Encore** 18](#_Toc178318148)

[3. Développement et Bonnes Pratiques 19](#_Toc178318149)

[**A. Développement du Front-End** 19](#_Toc178318150)

[**B. Bonnes Pratiques** 20](#_Toc178318151)

[**Documentation Technique : Back-End** 20](#_Toc178318152)

[**1. Technologies Utilisées** 20](#_Toc178318153)

[**2. Structure du Projet Symfony** 21](#_Toc178318154)

[**3. Configuration** 24](#_Toc178318155)

[**A. Configuration de la Base de Données** 24](#_Toc178318156)

[**4. Processus de Développement** 24](#_Toc178318157)

[**A. Installation des Dépendances** 24](#_Toc178318158)

[**B. Gestion des Entités et Migrations** 25](#_Toc178318159)

[**C. Débogage** 25](#_Toc178318160)

# 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.

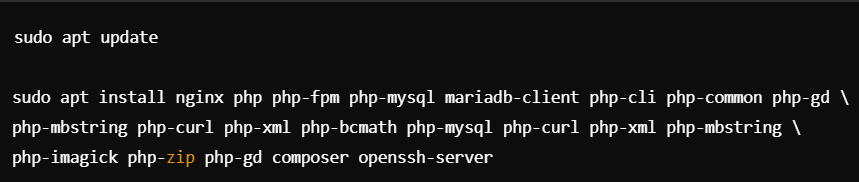
## Configuration Générale

* **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 :



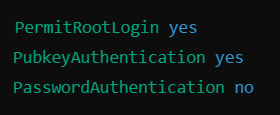
1. **Installer composer :**



1. **Configure ssh :**

Insertion de l’image...

* Décommenter et modifier :



* Générer une clé ssh :



* transféré la clé priver dans le pc win :

Insertion de l’image...  
copier/crée ficher sur win dans C://utilisateurs/nom/.ssh/id\_rsa

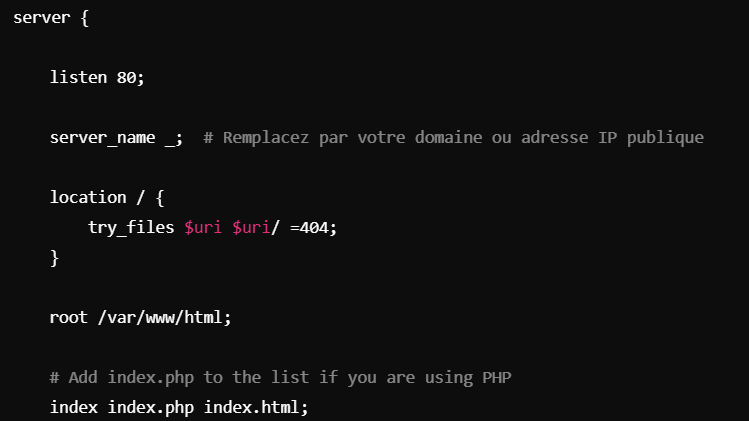
* Ou si même réseau :

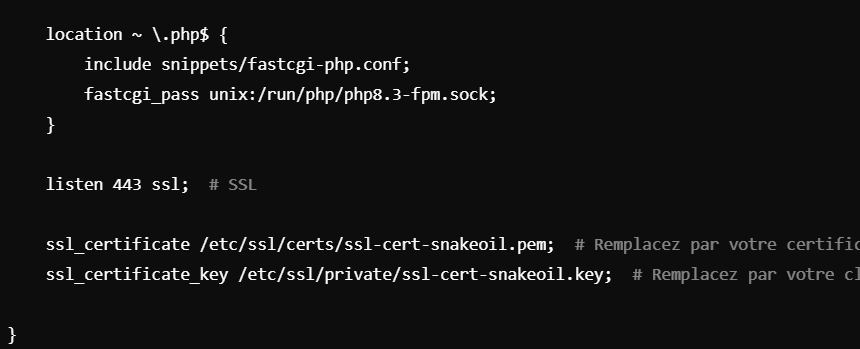
Insertion de l’image...

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

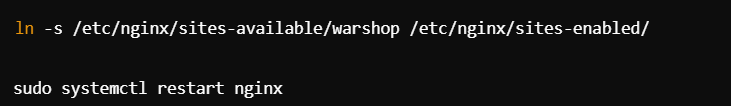


* + Ajouter la configuration suivante :





* + Activer la configuration et redémarrer Nginx :

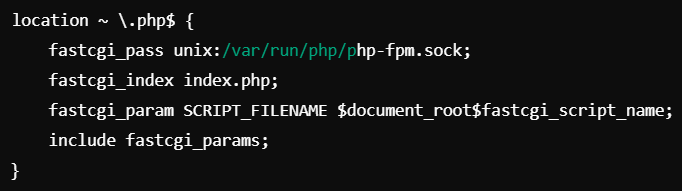


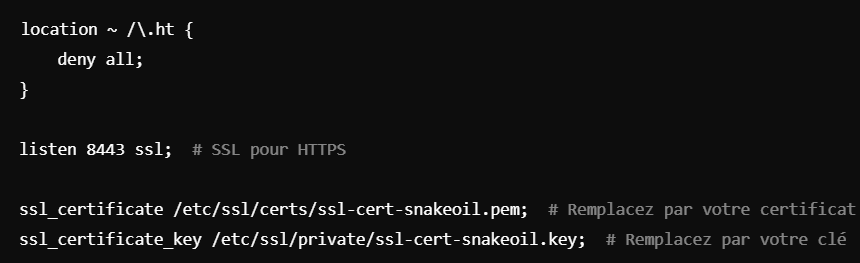
1. **Installer PHPMyAdmin :**
   * Installer PHPMyAdmin :

Insertion de l’image...

* + Configurer Nginx pour PHPMyAdmin :







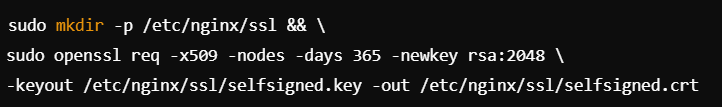
1. **Ajouter la base de donner externe :**



* + Redémarrer Nginx :

Insertion de l’image...

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

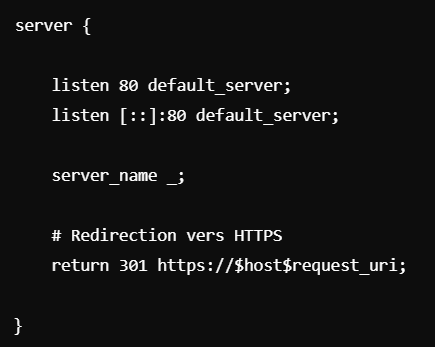


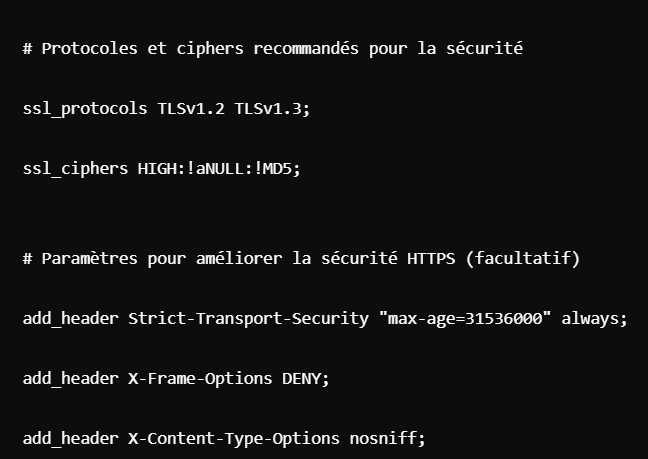
* Vérification de la création :

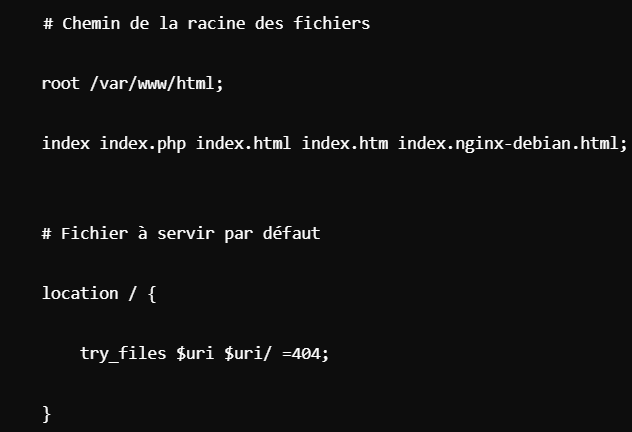


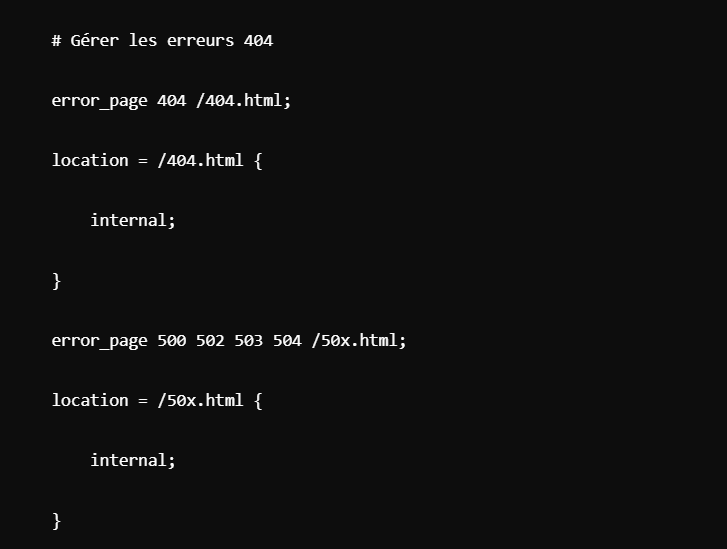
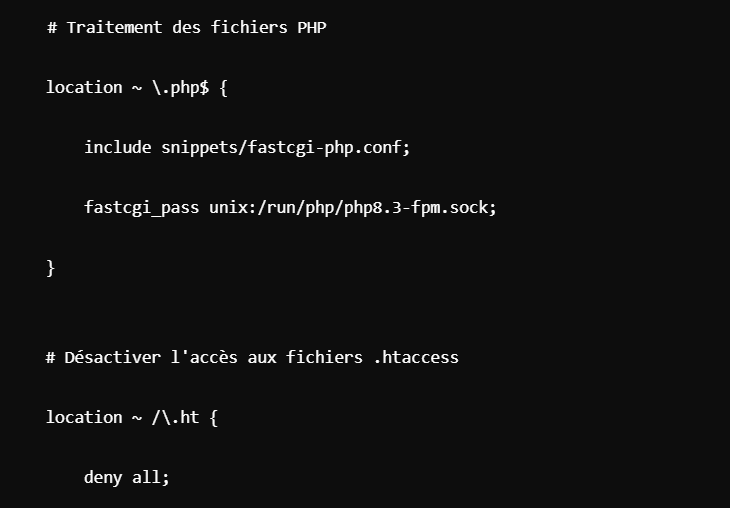
* Remodifer :











* Puis :



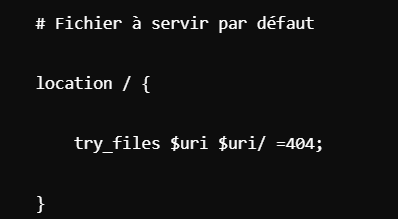
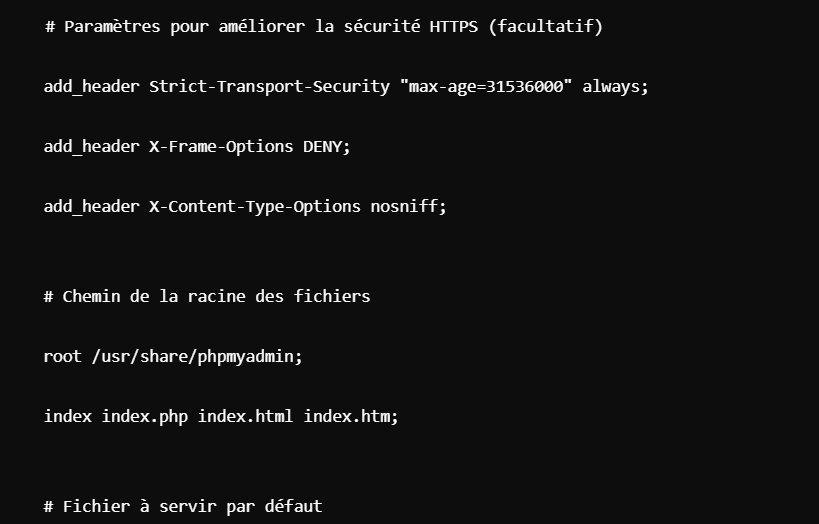
1. **Configurer le https crée clé ssl pour PHPMyAdmin**

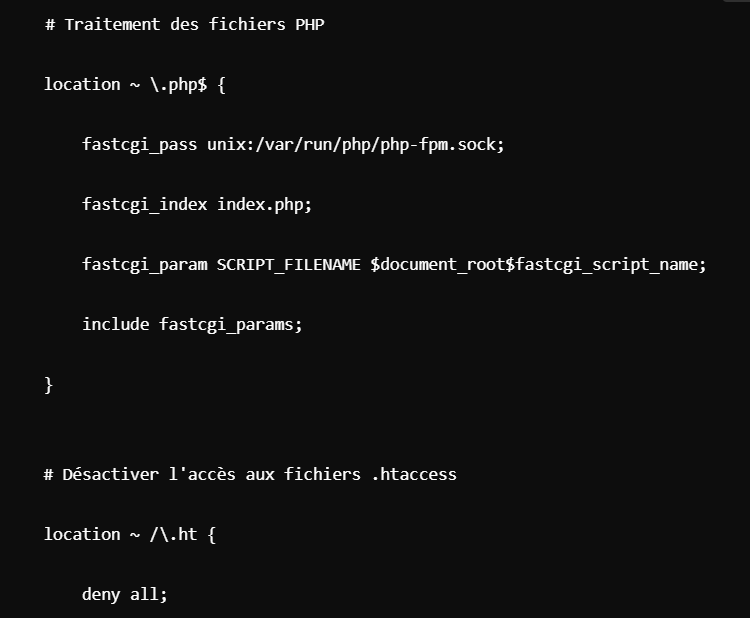
* Remodifer :

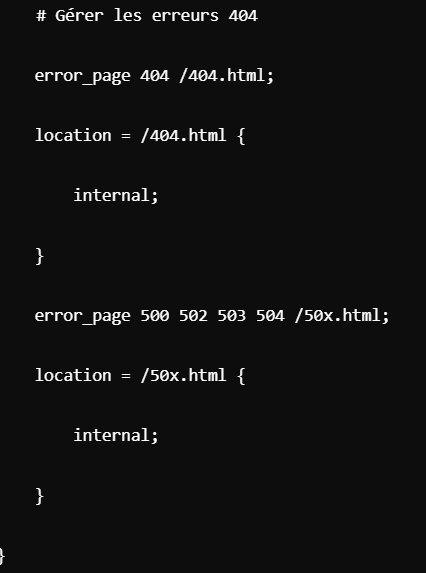












1. Puis :

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

1. **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;

1. **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

1. **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 :**

1. **Local Development (Git)** : Les développeurs effectuent des modifications en local et les valident via Git.
2. **SSH Access** : Une connexion sécurisée via SSH est utilisée pour se connecter au serveur distant.
3. **Git Push to Remote** : Les modifications sont poussées depuis les machines locales vers le serveur distant hébergeant le dépôt Git.
4. **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 :**

1. **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)



**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 :



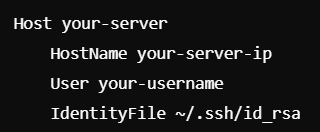
**Connexion via SSH** : Une fois la clé ajoutée, chaque développeur peut se connecter au serveur via SSH sans avoir à saisir de mot de passe.

Commande pour se connecter :



### **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 :



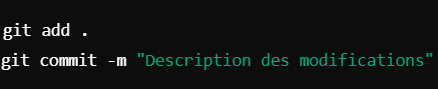
Cela permet de se connecter facilement au serveur avec la commande suivante :



## **3. Étapes pour Pousser les Modifications sur le Serveur**

### **A. Valider et Pousser des Modifications avec Git :**

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



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



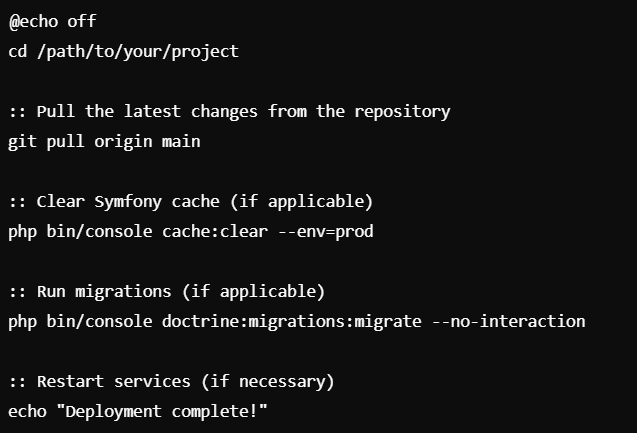
## **Script de Déploiement (.bat) pour Automatiser les Tâches**

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 :

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**

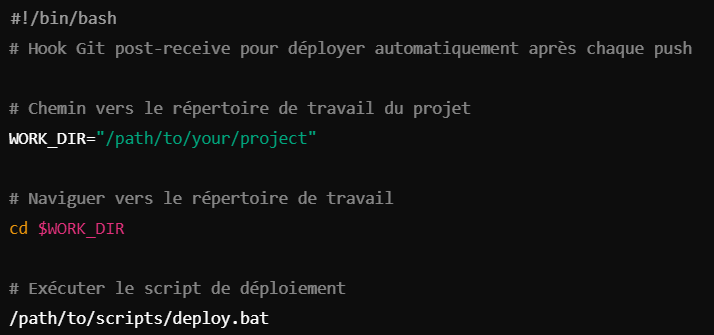
Nous avons configuré un **hook Git post-receive** qui se déclenche automatiquement après qu’un développeur pousse des modifications vers le dépôt sur le serveur. Ce hook exécute le script .bat pour déployer automatiquement les modifications.

### **Création du Hook Git post-receive :**

1. Sur le serveur, dans le répertoire du dépôt distant (généralement dans /var/repo/mon\_projet.git/hooks/), créez ou modifiez le fichier post-receive.
2. Assurez-vous que le fichier est exécutable :



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



### **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.

1. **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 :

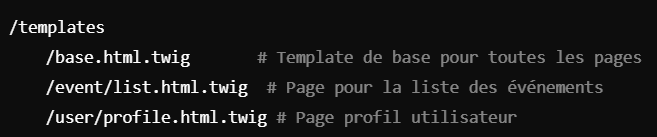
* **HTML5** : Structure de la page web.
* **CSS3 & SCSS (optionnel)** : Styles et mise en page, possibilité d'utiliser SCSS pour une gestion plus avancée des styles.
* **Bootstrap 4.5.2** : Framework CSS pour un design responsive et des composants pré-stylisés.
* **JavaScript (ES6)** : Pour les interactions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

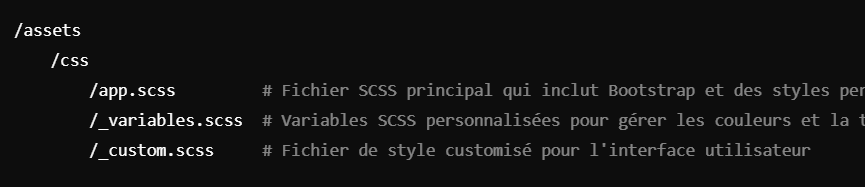
Les fichiers HTML sont gérés via des **templates Twig** (fichiers .html.twig) dans Symfony. Les composants récurrents (header, footer, navigation) sont séparés dans des templates pour la réutilisation.

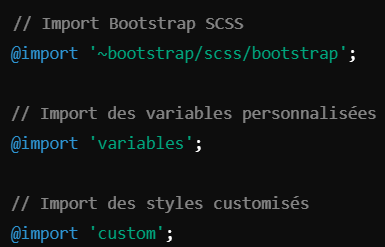
Exemple de structure :

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 :

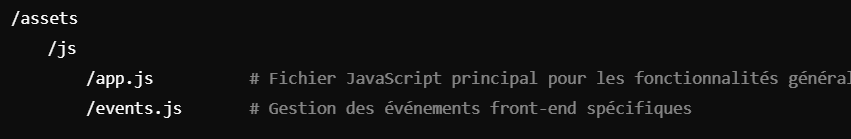
Le fichier app.scss inclut Bootstrap et d'autres styles personnalisés :

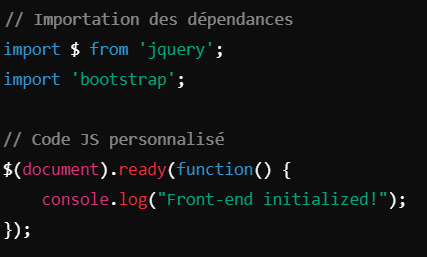


### **C. Fichiers JavaScript**

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

Exemple de structure :

Exemple de fichier app.js :



### **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 :

3. Développement et Bonnes Pratiques

### **A. Développement du Front-End**

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

Commandes à exécuter :

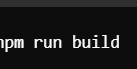


**Compilation des Fichiers** : Webpack Encore est utilisé pour compiler les fichiers SCSS et JavaScript.

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



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

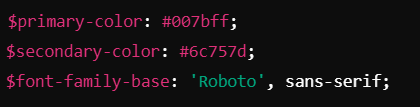


1. **Structure Modulaire** :
   * Utiliser des composants réutilisables : Lorsque vous travaillez sur des fonctionnalités front-end, essayez de découper les styles et le JavaScript en composants réutilisables (ex : cards, modals).
   * Modularisation des styles : Utilisez des **partiels SCSS** (fichiers 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**

1. **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.).
2. **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 :



# **Documentation Technique : Back-End**

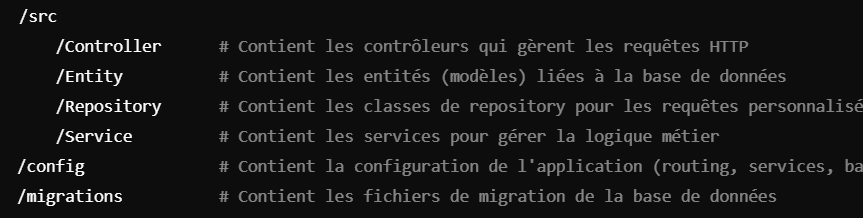
## **1. Technologies Utilisées**

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 :

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 :

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 :

C. Repository (/src/Repository)

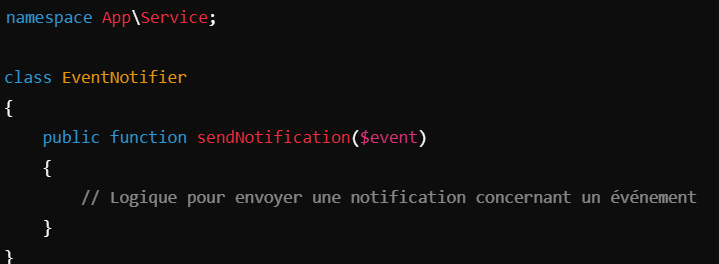
Les repositories contiennent la logique pour interagir avec la base de données. Symfony génère automatiquement des repositories pour chaque entité, mais il est possible d'ajouter des méthodes personnalisées pour effectuer des requêtes plus complexes.

Exemple d'une méthode personnalisée dans un repository :

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 :

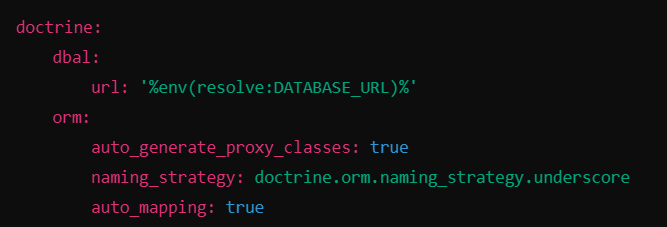


## **3. Configuration**

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**

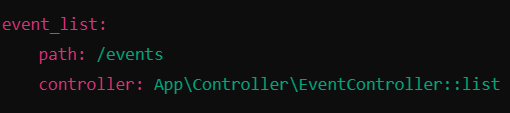
Le fichier de configuration principal pour la base de données se trouve dans /config/packages/doctrine.yaml. Voici un exemple de configuration pour une base de données MySQL :

La variable DATABASE\_URL est définie dans le fichier .env du projet :

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 :



## **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.



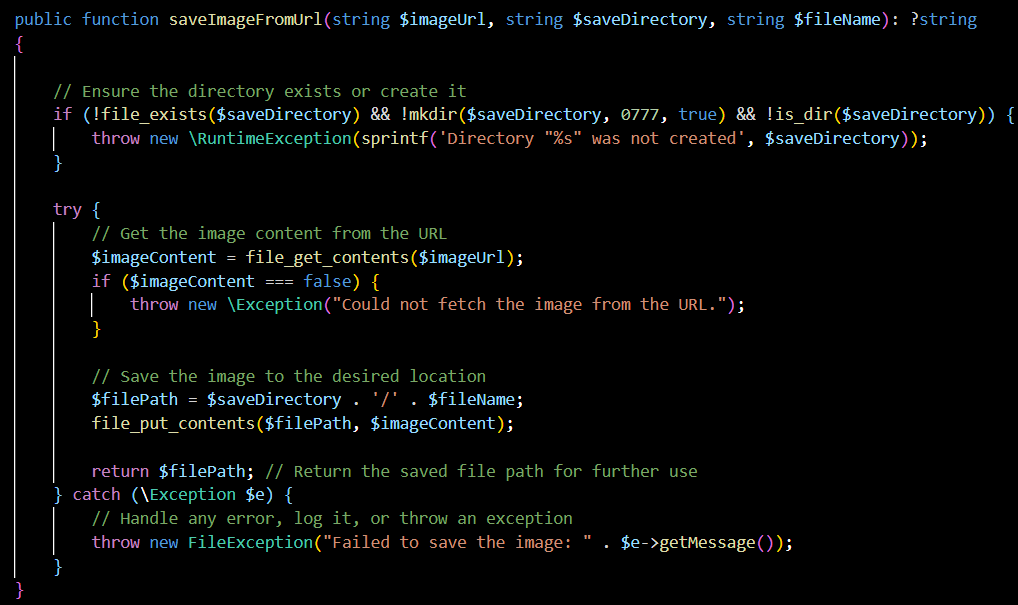
* Voici la partie concernant la connexion à l'API DALL-E de ChatGPT. Nous allons configurer l'API en utilisant notre clé API de ChatGPT.



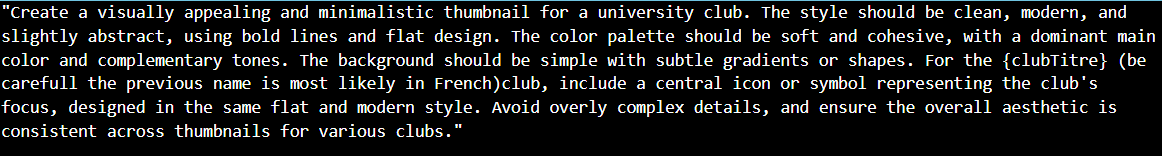
* L'image sera créée à partir du prompt que l'on fournira à DALL-E. Le contrôleur enregistrera ensuite l'URL générée dans un dossier à l'aide de la fonction saveImageFromUrl, puis enregistrera cette URL dans la base de données avec la méthode setClubImg.



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



* Voici le prompt pour chat Gpt .

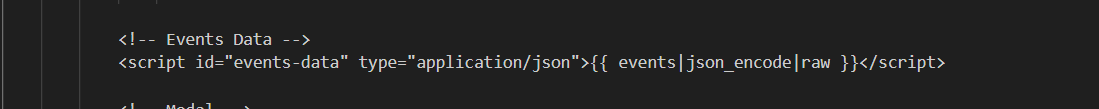


#### Utilisation de FullCalendar

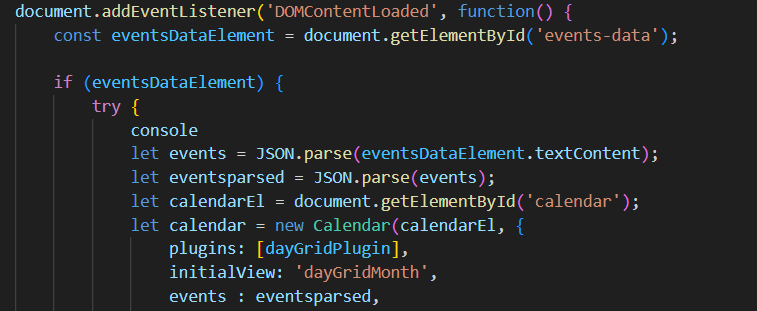
Nous avons créé une page permettant d’afficher l’ensemble des évènements entrés en bdd sur un calendrier. Pour se faire, on commence par récupérer les évènements depuis notre contrôleur. Ensuite, on formate un doc JSON qui réunit l’ensemble des informations que l’on souhaitera exploiter dans notre calendrier (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 :



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



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.



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.

