DOSSIER DE PROJET

Michel ALMONT

# Développeur Web et Web mobile Studi

## Application web SPORT-TRAINING



# Introduction

1. Compétences du référentiel couvertes par le projet
   1. Développer la partie front-end d’une application web ou web mobile en intégrant les recommandations de sécurité

* Maquetter une application.
* Réaliser une interface utilisateur web statique et adaptable.
* Développer une interface utilisateur web dynamique.
* Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce.

L’application doit être intuitive et responsive suivant les tailles d’écran.

Avec un design épuré et ergonomique et aller à l’essentiel compte tenu du temps qui nous ai imparti pour réaliser ce projet.

C’est avec Twig et Bootstrap que j’ai façonner l’interface utilisateur et d’administration.

* 1. Développer la partie back-end d’une application web ou web mobile en intégrant les recommandations de sécurité.
* Créer une base de données.
* Développer les composants d'accès aux données.
* Développer la partie back-end d'une application web ou web mobile.
* Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce.

La partie backend sera la partie administrative de l’application.

C’est-à-dire la façon dont sera géré et stocker les informations et les données de l’application.

Le Framework Symfony a été choisi pour ces nombreux modules et fonctionnalités, et il a été couplé à une base de données **POSTGRES** respectueux des normes **ANSI SQL.**

1. Résumé du projet

Ce projet a été réalisé Dans le cadre de ma formation à l’école STUDI, Développeur Web – Web Mobile qui a débuté le 1 février 2022.

L’application à développer est destinée à une grande marque de salle sport, qui propose déjà à ces partenaires franchisés, plusieurs services ayant pour but de faciliter la gestion d’une salle de sport au quotidien.

Ces services sont disponibles sur une plateforme déjà existante.

Voici quelques exemples de prestations disponibles :

* Faire son mailing
* Gérer le planning équipe
* Promotion de la salle
* Vendre des boissons

Chacune de ces activités peuvent être considérées comme des modules activables ou désactivables à volonté.

Ma mission est de créer un gestionnaire d'accès à ces modules disponibles sur une application tierce.

Un backend qui permet d’activer ou désactiver les services selon le package que le partenaire aura contracté.

L’application devra être intuitive, et aller l’essentiel.

Elle donnera la possibilité au staff de désactiver un partenaire par exemple qui n’aura pas renouveler son abonnement annuel ou encore de modifier son niveau de service.

Le développement de cette application m’a pris environ un mois.

C’est le duo Symfony/Postgress qui j’ai choisi pour la partie backend et Bootstrap et Twig pour la partie frontend.

Ce projet a été réaliser en total autonomie, J’ai pu acquérir durant cette expérience de nouvelle compétence, notamment sur la partie backend.

Ayant débuter mon apprentissage autodidacte dans le monde du développement mobile, J’ai été également agréablement surpris de la modernité du Framework Symfony.

1. Environnement technique ET HUMAIN

J’ai réalisé ce projet seul, et c’est sur un mac book pro que j’ai développé la totalité de ce projet.

Cela a été pour moi une expérience particulièrement enrichissante sur tous les plans.

En particulier la partie backend car je ne connaissais pas le Framework Symfony que j’ai trouvé particulièrement moderne.

L’environnement de Symfony m’a aussi permis d’asseoir ma compréhension du pattern MVC, et m’a permis d’améliorer ma connaissance du langage PHP.

Mon apprentissage dans le développement a été en premier orienté vers les technologies cross plateforme tel que Flutter et Xamarin Forms et la formation STUDI m’a apporté plus que ce que j’attendais, en particulier sur l’organisation de mon code et un rééquilibrage entre la pratique et la théorie.

Car étant autodidacte je souhaitais grâce à cette formation, acquérir le vocabulaire indispensable pour échanger avec d’autre développeur.

# LE PROJET DE STAGE

1. Le cahier des charges

**Fonctionnalité de l’application**

L’application devra disposer de plusieurs interfaces, une interface de gestion et une de visionnage des accès.

* 1. Interface de gestion

L’interface de gestion destinée à l’équipe d’administration et l’équite tech, devra permettre les fonctionnalités suivantes :

* Création et administration des partenaires franchisées
* Création et administrations des structures affiliés aux partenaires
* Possibilité au staff de désactiver un partenaire par exemple qui n’aura pas renouveler son abonnement annuel ou encore de modifier son niveau de service.
* Création et association de compte pour les responsables de partenaire et de structure.
* Envoi automatique d’un email lors de la création et de l’association d’un compte responsable avec un partenaire ou une structure.
* Envoi automatique d’un email lors des modifications de permissions, au responsable de partenaire ou de structure.
* Accès après authentification en lecture/écriture à la plateforme d’administration.
* A chaque modification ou suppression, affichage d'un message de confirmation pour valider l'action.
* L’application proposera une barre de recherche pour filtrer les recherches.
* Le premier filtre sera disponible en tapant les premières lettres du partenaire ou de la structure.
* Le deuxième sera paramétré par rapport aux éléments actifs ou non.
  1. Interface utilisateur en lecture seul

L’application proposera aussi une interface pour les responsables partenaires et une autre pour les responsables de structure.

Ces interfaces donneront accès en lecture seul, au niveau de service posséder par le partenaire ou la structure qui possède un abonnement à sport-training.

Les responsables qui auront accès à cette interface recevront leur identifiant par email ainsi que l’adresse de connexion à la plateforme.

1. Spécification technique
   1. Maquettage

* Pour la partie maquettage j’ai utilisé **Figma**.
* **Diagrams.net** est l’outil que j’ai utilisé pour réaliser le diagramme de classe ou, diagramme de cas d’utilisation et le diagramme de séquence
* **Canva.com** est l’outils que j’ai utilisé pour la documentation.
  1. Environnement de travail
* L’environnement de développement tournait sous **MacOs**
* Pour consulter, modifier et tester la base de données, j’ai utilisé l’IDE **Datagrip.**
* **Vscode** est mon éditeur de texte préféré.
  1. Technologie frontend
* Pour coder rapidement une interface responsive et efficace j’ai choisi l’outils **Bootstrap** couplé au thème **Bootswatch**.
* Le Template **Twig** est le moteur de gabarit que j’ai utilisé pour la gestion des vues.
* Le moteur de Template open source **Twig** facilite le développement, la sécurisation et la maintenance d'applications web PHP.
* JavaScript est un élément indispensable pour le bon fonctionnement de Bootstrap.
  1. Technologie backend
* Base de données

Le système de gestion de base de données est **POSTGRES.**

Ce système multi-plateforme est largement connu et réputé à travers le monde, notamment pour son comportement stable et pour être très respectueux des normes **ANSI SQL.**

* **Symfony**

Le Framework PHP Symfony est le partenaire que j’ai choisi pour interfacer le SGBD POSTGRESS.

Pourquoi Symfony ?

Il contient tout ce dont a besoin pour créer un site web professionnel et sécurisé :

* Un moteur de gabarit
* Un ORM
* Un client de test

Symfony est un Framework qu’on pourrait qualifier de modulaire.

Pour utiliser toute la puissance de ce Framework il faut installer et activer plusieurs modules indispensables pour réaliser un site web sécurisé et professionnel.

Les composants qui ont été installés et activer dans ce projet sont décrit succinctement dans les prochaines lignes.

* Doctrine

Doctrine est un ORM de Symfony

Un ORM est un ensemble de classes permettant de manipuler les tables d’une base de données relationnelle comme s’il s’agissait d’objets.

En base de données chaque élément ou objet d’une application est représenté par une table.

En programmation ces éléments sont des objets.

Un ORM est une couche d’abstraction d’accès à la base de données qui donne l’illusion de ne plus travailler avec des requêtes mais de manipuler des objets.

* Mail

Symfony possède son propre composant nous permettant d'envoyer des e-mails depuis la version 4.3. C’est le composant Mailer.

L’installation du composant Mailer se fait via composer.

* Forms

Les formulaires sont des éléments indispensables pour récupérer les informations de nos objets.

Symfony propose un système puissant et flexible qui permet d’unifier et de simplifier la génération et le traitement de formulaires.

A lui seul, il gère les éléments suivants :

* Rendu du formulaire dans la page
* Gestion de l'envoi du formulaire
* Validation des données
* Normalisation des données
* Protection CSRF

L’installation de ce module se fait par l’intermédiaire de composer.

* 1. GitHub
* Pour la gestion du travail collaboratif, du versionning et de l’hébergement des dépôts du projet le tandem GIT/GITHUB a été utilisé.
  1. Déploiement
* Le déploiement a été effectué sur un serveur VPS Ubuntu.
* Le serveur web utilisé est l’outils NGINX
  1. Sécurité
* SecurityBundle

Authentification avec le SecurityBundle de Symfony

Le contrôle de la sécurité sous Symfony est très avancé mais également très simple. Pour cela Symfony distingue :

• L’authentification,

• L’autorisation.

L'authentification, c'est le procédé qui permet de déterminer qui est votre visiteur. Il y a deux cas possibles :

• Le visiteur est anonyme car il ne s'est pas identifié,

• Le visiteur est membre de votre site car s'est identifié

Sous Symfony, c'est le firewall qui prend en charge l'authentification.

Régler les paramètres du firewall va vous permettre de sécuriser le site. En effet, vous pouvez restreindre l'accès à certaines parties du site uniquement aux visiteurs qui sont membres. Autrement dit, il faudra que le visiteur soit authentifié pour que le firewall l'autorise à passer.

L'autorisation intervient après l'authentification. Comme son nom l'indique, c'est la procédure qui va accorder les droits d'accès à un contenu. Sous Symfony, c'est l’Access control qui prend en charge l'autorisation.

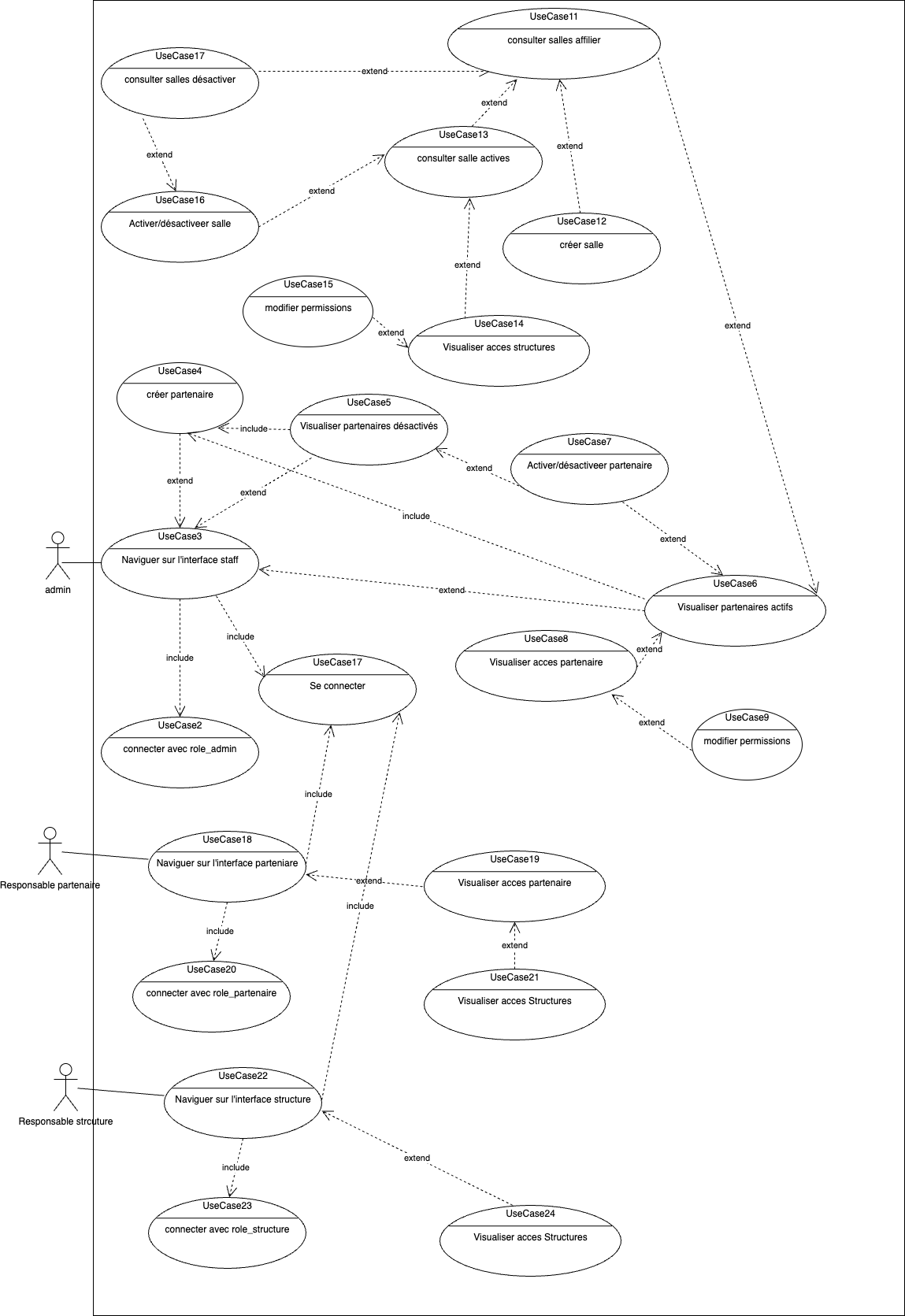
Prenons l'exemple de différentes catégories de membres. Tous les visiteurs authentifiés ont le droit de poster des messages sur le forum mais uniquement les membres administrateurs ont des droits de modération et peuvent les supprimer. C'est l’accès control qui permet de faire cela.

* 1. Sécurisation de l’environnement
* Concernant la sécurité du serveur, j’ai installé le tandem Fail2ban le firewall UFW pour filtrer les paquets et limiter l’ouverture des ports du serveur.
* Un pare-feu est essentiel lors de la configuration du VPS pour limiter le trafic indésirable sortant ou entrant dans votre VPS.
* L'outil fail2ban permet de surveiller l'activité des logs de certains services, tel que SSH ou Apache. Lors d'un trop grand nombre d'authentifications ratées fail2ban va générer une règle IPTables, cette règle aura pour but d'interdire pendant une durée déterminée les connexions depuis l'adresse IP susceptible d’être un attaquant.

# LA REALISATION DU PROJET

1. Conception de la base de données
   1. Diagramme de cas d’utilisation

J’ai construit un diagramme de cas d’utilisation pour établir précisément le comportement fonctionnel de l’application.



Ce diagramme permet donc d’établir les différentes interactions entre les utilisateurs et les différentes interfaces que proposes le logiciel.

Il y a certaine interface réservée aux administrateurs et d’autres aux responsables de partenaire et de structure.

Chaque interface peut donner accès à de nouvelle fonctionnalité en fonction du niveau d’accès utilisateur, ou de certaine condition.

Par exemple pour modifier un accès partenaire il faudra en premier accéder à l’interface d’administration, visualiser les partenaires et leurs accès puis accéder à une interface qui permettra de modifier les accès ou encore les données partenaires.

L’accès à l’administration de l’application implique d’abord que vous vous possédiez un utilisateur ayant les droits d’administration qui se trouve dans la base de données et que vous l’ayez connecté depuis l’interface de connexion de l’application.

* 1. Diagramme de classe

Le diagramme de classe permet de modéliser et de matérialiser les différents types d’objet de l’application, ainsi que le type relation qui existe entre eux.

Ces objets sont représentés par des tables contenant leurs caractéristiques ou encore appelé attributs.

Par exemple un objet user pourrait être représenté par les données suivantes rangé dans une table USER :

* Nom
* Prénom
* Email
* Téléphone
* Type

Les relations entre ces objets sont aussi modélisées par des liens et sont caractérisé par leur cardinalité.

La cardinalité d’un lien indique le nombre d'occurrences (aucune, une ou plusieurs) par rapport à leur relation qu’une entité peut avoir par rapport à une autre.

Par exemple le lien entre une voiture et un client, peut être défini de la sorte :

Le lien qui existe entre le client et la voiture est représenté par l’action « acheter une ou plusieurs voiture », cette action peut se répéter plusieurs fois sa cardinalité est donc représenté par une étoile.

Mais le lien entre ces 2 objets est bidirectionnel, donc le lien dans le sens inverse sera représenté par l’action :

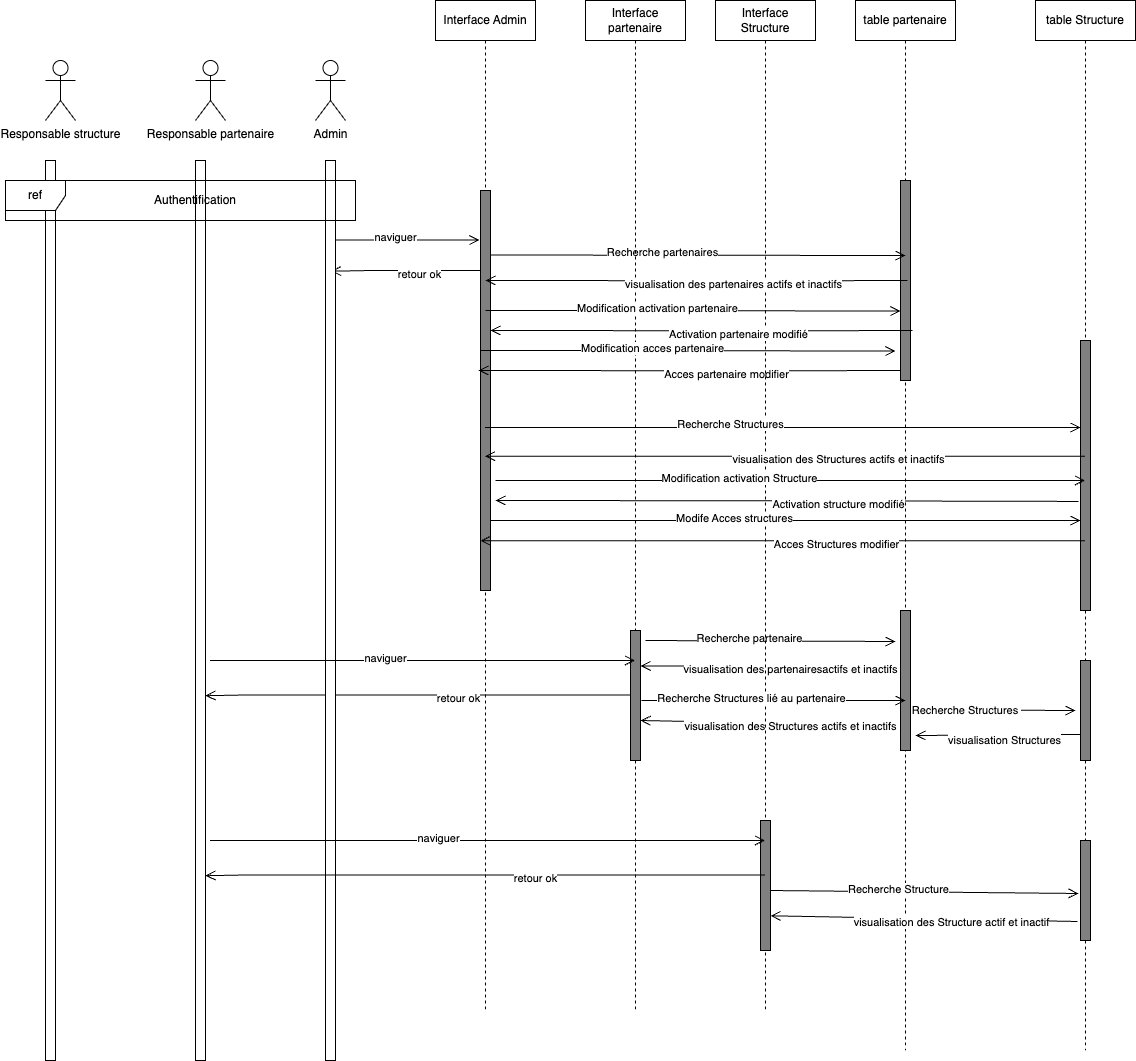
« La voiture peut être acheter par un seul client », le lien aura donc une cardinalité de 1 dans l’autre sens.



* 1. Diagramme de séquence

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique

Cela permet de montrer comment interagi le système dans le temps, ainsi que l’ordre dans lequel ils se produisent lorsque l’utilisateur fait appel à une fonctionnalité.



1. Développement de la partie backend
   1. Installation et configuration
      1. Installation de la base de données

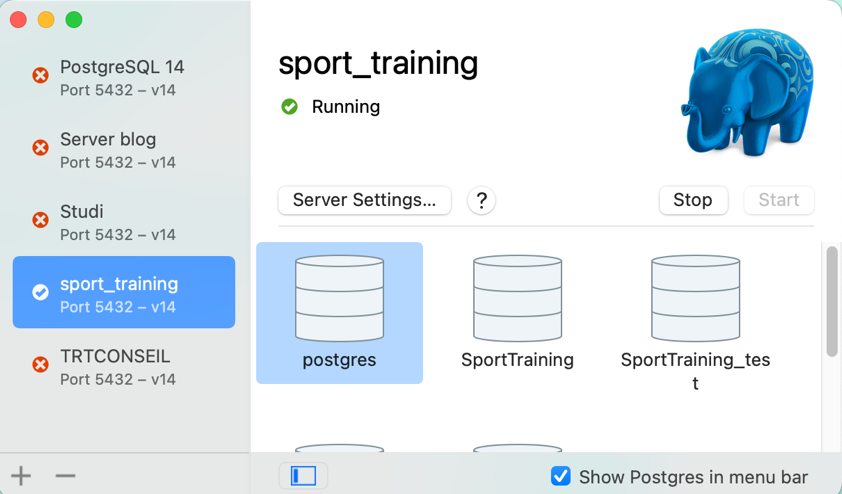
Les descriptions des procédures d'installation et de configuration sont destinées à l'OS MACOS, pour les autres systèmes d'exploitation vous pouvez vous référez à la documentation officielle.

J’ai téléchargé les paquets de postgress à l'adresse suivante :

Download PostGress](postgresql.org/download/macosx/

Puis je l’ai installé.

Post Gress met à disposition un petit utilitaire sur MacOs pour créer et paramétrer les bases de données.



J’ai créé un serveur dédié ainsi que la base de données pour le projet sport-training.

J’ai aussi créé un utilisateur et je lui ai assigné les accès sur la base de données sport-training.

Pour réaliser cela il faut cliquer sur la base de données par défaut du serveur PostgreSQL par exemple, cela ouvrira un terminal qui nous permettra de lancer les commandes suivantes pour créer notre utilisateur et notre base de données.

* Commande pour créer un utilisateur :

**CREATE USER studi WITH ENCRYPTED PASSWORD ‘Wipit2017’;**

**CREATE ROLE**

* Commande pour créer une base de données :

**CREATE DATABASE sportTraining;**

**CREATE DATABASE**

* Commande pour donner les accès à l’utilisateur studi à la base sportTrining :

**GRANT ALL PRIVILEGES ON DATABASE sportTraining TO studi;**

**GRANT**

* + 1. Installation de Symfony

Selon la Documentation de symfony officiel qui se trouve à l’adresse suivante :

« <https://symfony.com/doc/current/setup.html#technical-requirements> »,

avant de créer votre première application vous devez :

* Installer PHP 8.1

Installez PHP 8.1 ou supérieur et ces extensions avec la commande suivante :

$ brew install php

* Installer Composer

Install Composer, qui est utilisé pour installer les packages PHP avec la commande suivante :

$ brew install composer

$ brew link --overwrite composer

* Installer Symfony CLI

Il est recommandé aussi l'installation de Symfony CLI avec la commande suivante :

$ brew install symfony-cli/tap/symfony-cli

* + 1. Test de l’environnement

Le symfonybinaire fournit également un outil pour vérifier si votre ordinateur répond à toutes les exigences.

Ouvrez votre terminal de console et exécutez cette commande :

$ symfony check:requirements

Si votre environnement est optimal vous recevrez un retour positif dans votre terminal:

$ [OK]

Your system is ready to run Symfony projects

* 1. Initialisation du projet
     1. Création de l’application

Après avoir créer le dossier qui recevra tous les élements de votre appplication symfony,

vous pouvez l'ouvrir avec votre éditeur de texte préféré.

Ensuite vous ouvrez un terminal à la racine du dossier et vous tapez la commande suivante:

$ symfony sport-training –full

L'option --full est l'option pour préciser à symfony cli d'installer tous les packets nécéssaires

pour faire une application web complète.

* + 1. Démarrage de l’environnement

Pour démarrer l'environement il faut rentrer dans le projet et démarrer le server symfony

avec les commandes suivantes:

$ cd my-project/

$ symfony server:start

* 1. Création et structure des données
     1. Connexion de Symfony à la base de données

Pour connecter Symfony à la base de données PostgreSQL que nous avons créé.

Il nous rentrer les données suivantes de connexions dans le fichier ‘.env ‘qui se trouve à la racine du projet.

$DATABASE\_URL="postgresql://tintin:Wipit@2017@127.0.0.1:5432/SportTraining?

serverVersion=14&charset=utf8"

* + 1. Création de l’entité Partenaire

L'entité est un fichier PHP contenant des propriétés, des getters et setters. Ce fichier est la structure de notre table SQL.

Pour créer cette entité, Symfony fournit un bundle du nom de ‘DoctrineBundle’ qui est l’ORM qui va nous permettre de manipuler les données.

Un ORM Ou mapping objet-relationnel (en anglais object-relational mapping) est une couche d’abstraction à la base de données. Cette technique permet à l’utilisateur d’utiliser les tables d’une base de donnée comme des objets. Elle consiste à associer une ou plusieurs à chaque table , et un attribut de classe à chaque colonne de la table.

L’utilisation de doctrine requiert normalement son installation, mais il est déjà installé pour nous grâce à l’option de création –full que nous avons ajouté lors de la création du projet.

La commande pour l’installation est la suivante :

$ composer require --dev symfony/maker-bundle

J’ai créé l’entité Partenaire avec la commande suivante :

$ php bin/console make:entity Partenaire

Cette commande démarre un assistant qui vous aidera à créer votre entité et ces attributs.

Ci-dessous un résumé de ce que l’assistant demande :

$ Class name **of** the entity to create or update:

> Partenaires

New property name (press <**return**> to stop adding fields):

> name

Field type (enter ? to see all types) [string]:

> string

Field length [50]:

> 50

Can **this** field be null **in** the database (nullable) (yes/no) [no]:

> no

New property name (press <**return**> to stop adding fields):

> active

Field type (enter ? to see all types) [string]:

> boolean

Can **this** field be null **in** the database (nullable) (yes/no) [no]:

> no

Dans un premier temps, Symfony va créer deux fichiers au nom de votre entité. Le premier sera dans le dossier "src/Entity/Partenaire.php" et le second dans le dossier "src/Repository/PartenaireRepository.php".

Ces 2 fichiers sont reliés entre eux.

L'entité sera mise à jour à chaque nouvelle colonne.

Résumé du fichier de l’entité Partenaire :

<?php

namespace App\Entity;

use App\Repository\PartenaireRepository;

use Doctrine\Common\Collections\ArrayCollection;

use Doctrine\Common\Collections\Collection;

use Doctrine\DBAL\Types\Types;

use Doctrine\ORM\Mapping as ORM;

use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: PartenaireRepository::class)]

class Partenaire

{

#[ORM\Id]

#[ORM\GeneratedValue]

#[ORM\Column]

private ?int $id = null;

#[ORM\Column(length: 50)]

#[Assert\NotBlank()]

#[Assert\NotNull()]

#[Assert\Length(min:2, max:50)]

private ?string $name = null;

#[Assert\NotNull()]

#[ORM\Column]

private ?bool $active = null;

#[Assert\NotBlank()]

#[ORM\Column(length: 100, nullable: true)]

private ?string $shortDescription = null;

#[Assert\NotBlank()]

C’est ce fichier qui servira à créer les tables de notre base de données.

C’est aussi dans ce fichier que l’on va configurer la validation des données

Ces validations se font grâce aux contraintes qui se présente de la façon suivante :

* #[Assert\NotNull()] : interdit que la données soit nul.
* #[Assert\NotBlank()] interdit que la données soit vide.
* #[Assert\Length(min:2, max:50)] interdit à la données d’être plus petit que 2 et plus grand que 50.

Pour créer la table Partenaire j’ai d’abord généré un fichier de migration avec la commande :

symfony console make:migration

Ensuite pour générer les tables j’ai fait la commande suivante :

symfony console doctrine:migrations:migrate

Maintenant la table Partenaire a été créé.

* + 1. Création de l’entité Structure

J’ai créé l’entité Structure de la même façon que l’entité Partenaire avec la commande suivante :

$ php bin/console make:entity Structure

Cette commande démarre un assistant qui vous aidera à créer votre entité et ces attributs.

Pour créer la table Structure j’ai d’abord généré un fichier de migration avec la commande :

symfony console make:migration

Ensuite pour générer les tables j’ai fait la commande suivante :

symfony console doctrine:migrations:migrate

Maintenant la table Partenaire a été créé.

* + 1. Création de l’entité User

La table utilisateur dans Symfony, est différente des autres, car elle est liée à l’authentification de Symfony.

Et c’est le bundle Security qui gère cette partie, ce bundle est responsable de toute la partie sécurité d’une application Symfony.

Comment est géré la sécurité dans Symfony ?

La sécurité dans symfony implique plusieurs éléments :

* Le firewall: qui est la porte d'entrée pour le système d'authentification, on définit différents firewall (au minimum 1 seul) qui va permettre de mettre en place le bon système de connexion pour l'url spécifiée via un pattern.
* Le provider : qui permet au firewall d'interroger une collection d'utilisateurs/mot de passe . C'est une sorte de base de tous les utilisateurs avec les mots de passe.
* Un encoder : qui permet de générer des hashs/d'encoder des mots de passe ; le plus connu étant MD5 mais vous pouvez utiliser d'autres encoders tels que : sha1, bcrypt ou plaintext (qui n'encode rien c'est le mot de passe en clair) <http://symfony.com/doc/current/security/named_encoders.html>
* Les rôles : qui permettent de définir le niveau d'accès des utilisateurs connectés (authentifiés) et de configurer le firewall en fonction de ces rôles. Les rôles peuvent être hierarchisées afin d'expliquer par exemple qu'un administrateur (ROLE\_ADMIN par exemple) et avant tout un utilisateur (ROLE\_USER).

Pour fonctionner, il est nécessaire d'ajouter le composant security à votre application symfony.

Ce composant est déjà installé dans notre environnement.

Pour créer l’entité USER j’ai utilisé la commande suivante :

bin/console make:user

Cette commande démarre un assistant qui vous aidera à créer votre entité et ces attributs.

Pour créer la table USER j’ai d’abord généré un fichier de migration avec la commande :

symfony console make:migration

Ensuite pour générer les tables j’ai fait la commande suivante :

symfony console doctrine:migrations:migrate

Maintenant la table USER a été créé.

* + 1. Création des relations
       1. Relation Partenaire-structure OneToMany

Les liens entre les tables sont représentés dans Symfony par des relations entre les entités.

Il existe plusieurs types de relation :

* One-To-One : 1 entité est liée à 1 entité ;
* Many-To-One : plusieurs entités liées à 1 entité (liée à une OneTo-Many) ;
* One-To-Many : une entité liée à plusieurs ;
* Many-To-Many : plusieurs entités liées à plusieurs.

La relations entre l’entité Partenaire et l’entité Structure est de type OneToMany

Et la procédure pour créer cette relation est la suivante.

Il faut modifier la structure de l’entité partenaire avec la commande suivante :

$ php bin/console make:entity Partenaire

L’assistant propose de modifier l’entité partenaire et de lui ajouter un nouvel attribut.

J’ai choisi un attribut de type relation.

Ensuite, j’ai précisé l'entité liée, ainsi que le type de relation

Enfin, j’ai indiqué le type de relation.

$ Field type (enter ? to see all types) [string]:

> relation

What class should this entity be related to?:

> Structure

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:

> OneToMany

Cela à ajouter la clé externe de la table Partenaire à la table structure.

Une image contenant texte

Description générée automatiquement

* + - 1. Relation User-structure OneToOne

Pour la relation entre l’entité User et l’entité Structure, ça été exactement la même procédure, sauf que le type de relation a été OneToOne

* + - 1. Relation User - Partenaire OneToOne

Il en a été de même pour la relation entre l’entité User et Structure.

Cela à ajouter la clé externe de la table Partenaire et la table structure à la table User.

* 1. Remplissage de la base de données

Pour générer rapidement et facilement des fausses données aléatoirement, Symfony propose le bundle Fixture qui s’installe avec la commande suivante :

$ composer require –dev orm-fixture

Ce bundle sera couple à Faker php qui permet de générer de faux nom, de fausse adresse email, etc.

$ composer require –fakerphp/faker –dev

Le bundle fixture créer un fichier de configuration appelé AppFixture.php dans le dossier Datafixture.

Ce fichier a été configurer pour générer une dizaine de partenaire et de structure, ainsi qu’une vingtaine d’utilisateur.

1. Développement de la partie contrôleurs
   1. Définition

Symfony est un Framework PHP basé sur le modèle MVC, les chapitres précédent traite succinctement de la partie modèle, ici nous allons développer la partie Controller.

Le Controller, est l’élément qui contient la logique métier. C’est ici que sont la plupart des algorithmes, calculs, etc.

C’est aussi l’intermédiaire principal entre la vue et le modèle.

Nous pourrons effectuer les opération CRUD sur la base de données.

CRUD est un acronyme des noms des quatre opérations de base de la gestion de la persistance des données et applications :

* Create (créer)
* Read ou Retrieve (lire)
* Update (mettre à jour)
* Delete ou Destroy (supprimer)

Plus simplement, le terme CRUD résume les fonctions qu’un utilisateur a besoin d’utiliser pour créer et gérer des données

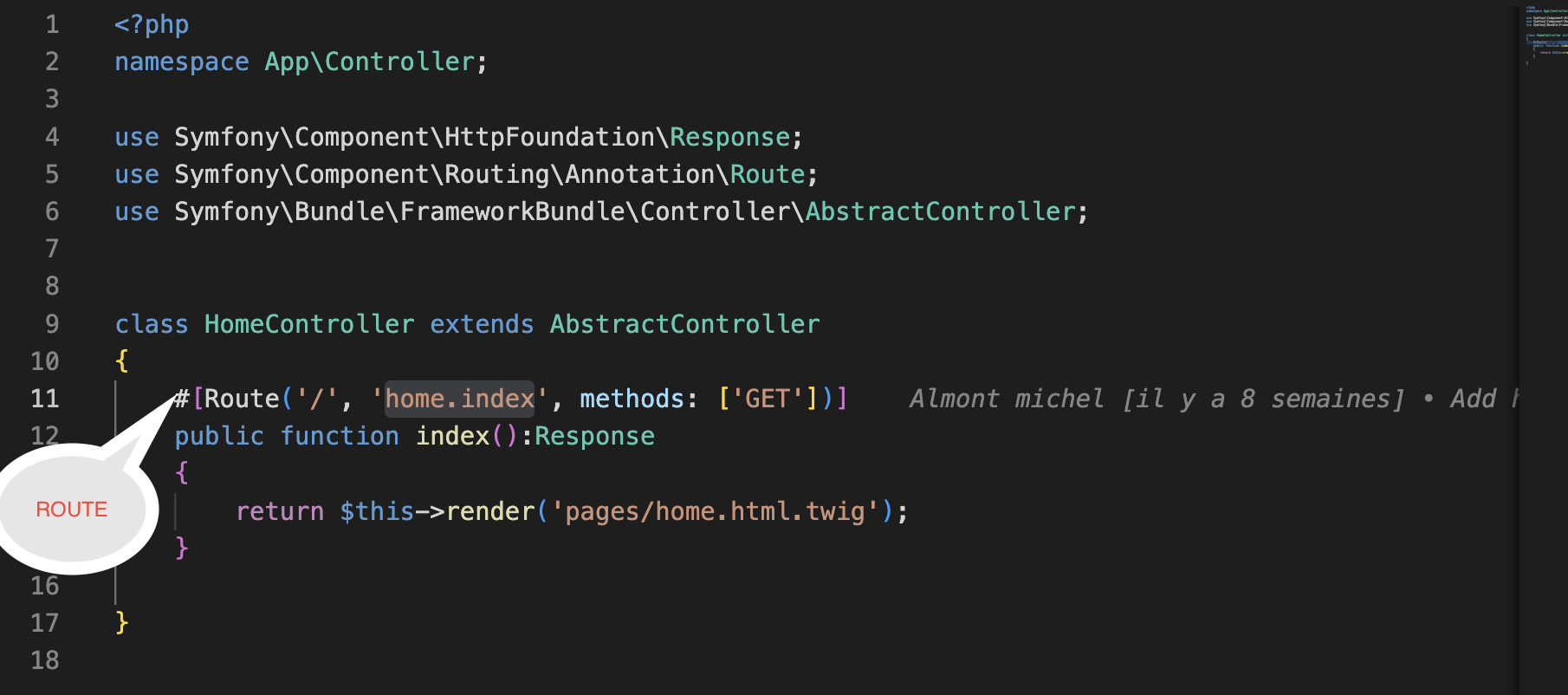
Pour créer un Controller, symfony-cli met à notre disposition une commande pour faire cela.

$ php bin/console make:controller

* 1. HomeController
     1. Présentation

Le HomeController est la partie logique qui gère la page de garde, la première page de présentation quand on arrive sur l’adresse de l’application.

* + 1. Décomposition



La classe d’un Controller doit hériter d’une classe ‘AbstractController’.

Cette classe dispose d’une méthode qui permet de générer le rendu d’un Template de TWIG.

Nous allons définir plus en détails le moteur de gabarit TWIG dans le volet Font-end.

Cette classe dispose aussi d’une route qui représente l’adresse ou sera afficher la page.

* 1. PartenaireController
  2. StructureController
  3. SecurityController
  4. UserController
  5. EspaceControllerPartenaire
  6. EspaceControllerStructure

1. Développement de la partie frontend