## 4.3 Choix techniques

4.3.1 Symfony5

Nous avons fait le choix d’utiliser un ensemble de composants PHP rassemblé dans un Framework. C’est un outil qui fournit des fonctionnalités modulables et adaptables, ce qui simplifie et accélère le développement d’une application web.

4.3.2 Gitea

Luence utilise cet outil collaboratif pour travailler à plusieurs sur les projets en développement. Il s’agit d’un service Git autohébergé simple à installer et à utiliser. Le but principal est de permettre à l’utilisateur de collaborer ou de versionner son travail sur le dépôt de son choix.

Pour plus d’informations sur Gitea, se rendre sur la page suivante, en anglais :

<https://gitea.io/en-us/>

4.3.3 le CSS avec Materialize

Le Framework Materialize CSS permet de reproduire des pages web dans le style du Material design, en utilisant des modules prêts à l’emploi. Il s’agit d’un Framework dont la visée est de respecter une charte graphique, en l’occurrence, celle du Material Design.

4.3.4 Java Script

C’est un langage de programmation permettant d’implémenter des mécanismes complexes sur une page web. Le choix de ce langage était requis pour initialiser les zones de listes de nos formulaires. Une exigence propre au Framework Materialize évoqué précédemment.

Note : pour échanger avec les membres de l’équipe, des outils de messageries ont été utilisés pour communiquer à propos du projet. Les échanges se font habituellement via Mattermost ou Slack en guise de messagerie interne. Quant aux réunions en visioconférence, elles se déroulent généralement sur l’outil Jitsi.

4.4 Installations requises

4.4.1 Wamp Server

Il s’agit d’une application disposant de tous les outils PHP et SQL nécessaires pour développer une application web. Il permet surtout de créer un serveur local pour tester le fonctionnement de l’application sans forcément passer par un hébergement web.

Note : Se servir du logiciel Mamp pour développer sur Mac Os.

4.4.2 Composer

Pour gérer les différentes bibliothèques appelées aussi dépendances, il convient d’installer Composer. Pour obtenir le logiciel, se rendre sur le site officiel de composer :

<https://getcomposer.org/download/>

## 4.5 **Traitement des données.**

### 4.5.1 L’ORM Doctrine

ORM pour Object Relations Mapper est un outil très puissant de Symfony. En plus de sa gestion des relations entre les différentes entités, il exécute également les requêtes SQL de récupération de données depuis la DB.

Pour en savoir plus sur cet outil, se rendre sur la page de la documentation de Symfony, contenu en anglais :

<https://symfony.com/doc/current/doctrine.html>

### 4.5.2 Le Repository

Il s’agit de classes PHP héritant de Doctrine\ORM\EntityRepository. Chacune est reliée à son entité.

Lors de la génération des entités en ligne de commandes, le Framework génèrera également le Repo correspondant. Plus explicitement, avec la commande « PHP bin/console make :entity », Symfony nous invitera à donner un nom à notre class. Pour l’exemple, elle s’appellera MyWonderfulEntity. Après avoir renseigné les champs souhaités, un Repo sera créé et s’appellera MyWonderfulEntityRepository.php.

Ce dernier comportera les méthodes natives pour récupérer les données à propos de l’entité créée.

Note : pour créer le Repository manuellement, il est primordial de respecter la convention de nommage. Le fichier devrait porter le nom de l’entité suivi du mot Repository. Ex : MyWonderfulEntityRepository.php.

4.5.2.1 Les fonctions natives :

Dans notre Repository, nous disposons déjà de fonctions toute faites permettant plusieurs possibilités :

* La fonction findAll() : elle permet de lister tous les enregistrements de notre entité
* La Fonction findBy(nom\_du\_champ) : elle permet de retrouver des enregistrements en fonction d’un ou de plusieurs critères.
* La Fonction findOneBy(Array $criteria) : Son comportement est similaire à celui de la fonction précédente. La différence est que celle-ci retourne un seul enregistrement.

4.5.2.2 les fonctions personnalisées

\*FindByCollaborator : elle permet de retrouver des projets par client via l’entreprise à laquelle celui-ci est rattachée.

//Repository/ProjectRepository.php

public function findByCollaborator($id)

{

return $this->createQueryBuilder('p')

->leftJoin('p.company', 'company')

->addSelect('company')

->leftJoin('company.collaborator', 'col')

->addSelect('col')

->andWhere('col.id = :id')

->setParameter('id', $id)

->getQuery()->getResult();

}

\*FindUserByRole : elle permet à l’administrateur d’afficher les administrateurs et les clients séparément.

/\*

Repository/UserRepository

Used to find user by role

\*/

public function findUserByRole(string $role)

{

return $this->createQueryBuilder('u')

->andWhere('u.roles LIKE :role')

->setParameter('role', "%\"$role\"%")

->getQuery()

->getResult()

;

}

### 4.5.3 Jeu de données

La génération de données est une étape cruciale. Tout dépend du type de l’application. Dans notre cas, Horizzon est essentiellement basé sur une interaction permanente entre clients et administrateurs du site. C’est pour cette raison que disposer de données fictives pour tester le comportement de l’application était nécessaire. Cela est susceptible de relever les éventuelles erreurs et de résoudre les différentes anomalies d’exécution.

Nous avons opté pour le DoctrineFixturesBundle, Un outil très efficace pour générer des données fictives proprement.

4.5.3.1 Installation

Pour utiliser ce Bundle, une procédure d’installation simplifiée est mise en place. Il suffit de se rendre sur le projet avec la console et introduire la commande suivante :

Composer require orm-fixtures –dev. Á la fin de l’installation, un dossier nommé DataFixtures est rajouter au dossier src de l’application. On y trouve la classe AppFixtures.php. Elle nous permettra d’insérer du code qui génèrera toutes les données dont on en aura besoin. Cela nous permettra à la fois de tester l’application et de faire des démonstrations.

4.5.3.2 La classe AppFixtures

Les fixtures nous permettront donc de générer :

* Des clients
* Des administrateurs
* Des phases\*
* Des entreprises
* Des projets
* Des phases de projets
* Des modules
* Des phases de modules

Le code intégral des fixtures figure dans l’annexe de ce rapport de stage.

Note : J’ai utilisé la bibliothèque Faker pour produire des données aléatoires vraisemblables. Cela serait pertinent pour générer des noms de personnes, de villes et pour insérer des lOrem également.

Cette bibliothèque PHP s’installe sur Symfony en saisissant la commande suivante :

Composer require –dev fzaninotto/faker

## 4.6 **Fonctionnalités pour les modules développés**

Nous avons pu listé tous les modules que comporte l’application Horizzon, voir le dossier fonctionnel à la section 3.3 de ce document. Dans cette rubrique, nous allons en parcourir les plus significatifs pour en exposer les principales fonctions qui définissent leur comportement et leur interaction avec les templates Twig.

4.6.1 Module User