Documentation technique

Pour la réalisation de ce projet, j'ai décidé d'utiliser le framework Symfony en version 6.2 pour le backend, afin d'avoir un outil performant permettant d'implémenter aisément les éléments de sécurité (authentification, autorisation sur les routes, protection contre les attaques CSRF pour les formulaires), d'utiliser l'ORM Doctrine pour la gestion de la base de donnée, qui utilisera le système de gestion de base de données MySQL.

J'ai décidé de travailler de façon hybride concernant les controller : pour les pages principales du site, un controller avec une méthode pour chacune avec une page réalisée avec le moteur template TWIG. Soit l'utilisation par défaut et la plus simple du framework Symfony. L'idée étant d'avoir des pages rendues côté serveur, afin de pouvoir optimiser le référencement sur ces pages.

Pour les pages d'administration en revanche, je souhaitais que l'utilisateur (administrateur/client) puisse bénéficier d'une interface réactive, sans rechargement de page, et sans avoir à changer de page pour chaque opération de CRUD concernant les mêmes éléments.
J'ai donc choisi de construire un controller sous le modèle d'une API REST, avec une méthode pour chaque opération nécessaire aux échanges avec la base de donnée. Les pages d'administrations, bien que les routes sont générées via des controllers sous Symfony, et les vues rendues avec le moteur de templates TWIG, comportent toutes un composant React, implémenté grâce au bundle proposé par Symfony. Depuis ce composant, je réalise toutes les opérations CRUD nécessaires à l'administration de la base de donnée, en appelant les routes créées dans le controller API et en utilisant les verbes appropriés.

Procéder ainsi m'a permis de pouvoir utiliser, par exemple, le générateur de token de protection CSRF implémenté dans TWIG via le framework, de passer ce token dans les props du composant React, et ainsi de vérifier le token au niveau des méthodes du controller API lors des requêtes, dans le but de pouvoir sécuriser les échanges avec la base de donnée. J'ai pu aussi profiter de la protection des routes du framework, et pouvoir faire passer certaines variables nécessaires au bon fonctionnement des élément de page (affichage des horaires sur toutes les pages, récupération des infos sur l'utilisateur connecté…).

Pour les différentes étapes de la réalisation de ce projet j'ai utilisé les outils suivants :

- Gestion de projet : Trello
- Création du logo : Adobe Illustrator
- Réalisation des maquettes : Figma
- Palette de couleurs : coolors.co
- Palette de police : fontjoy.com
- logiciel de gestion de version : Git relié à un compte Github
- Editeur de code : Visual Studio Code avec les extensions suivantes :
  - PHP Debug
  - PHP DocBlocker
  - PHP Getters & Setters
  - PHP Intelephense
  - PHP Namespace Resolver

- Symfony code snippets
- Twig code snippets
- YAML
- Database Client
- ES7+ React/Redux/React-Native snippets

Le projet a été réalisé sous un environnement WSL 2 (Windows Subsystem For Linux) avec une distribution Ubuntu installée, nodejs, npm, composer, symfony-cli MySQL et PHP.

A. Spécifications techniques

Serveur :

- Système d' exploitation : Ubuntu v. 22.04.2
- Serveur web : Nginx v. 1.18.0
- Système de Gestion de base de donnée : MySQL v. 8.0
- PHP v. 8.2.2
- Node v. 19.6.0
- npm v. 9.4.0
- composer v. 2.5.1
- symfony-cli v. 5.4.21

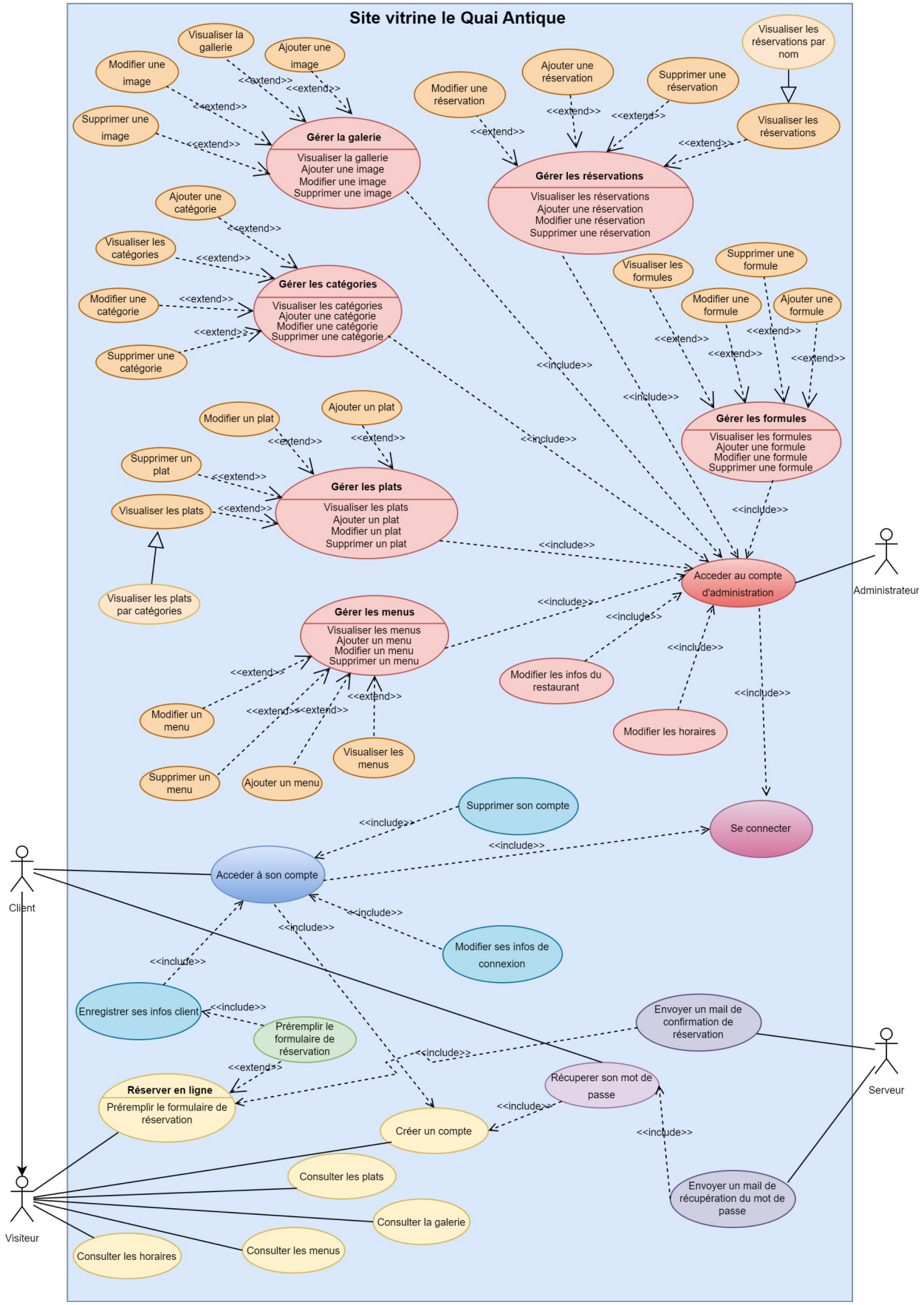
Pour le front :

- TWIG
- SASS
- JavaScript
- React v. 18

Pour le back :

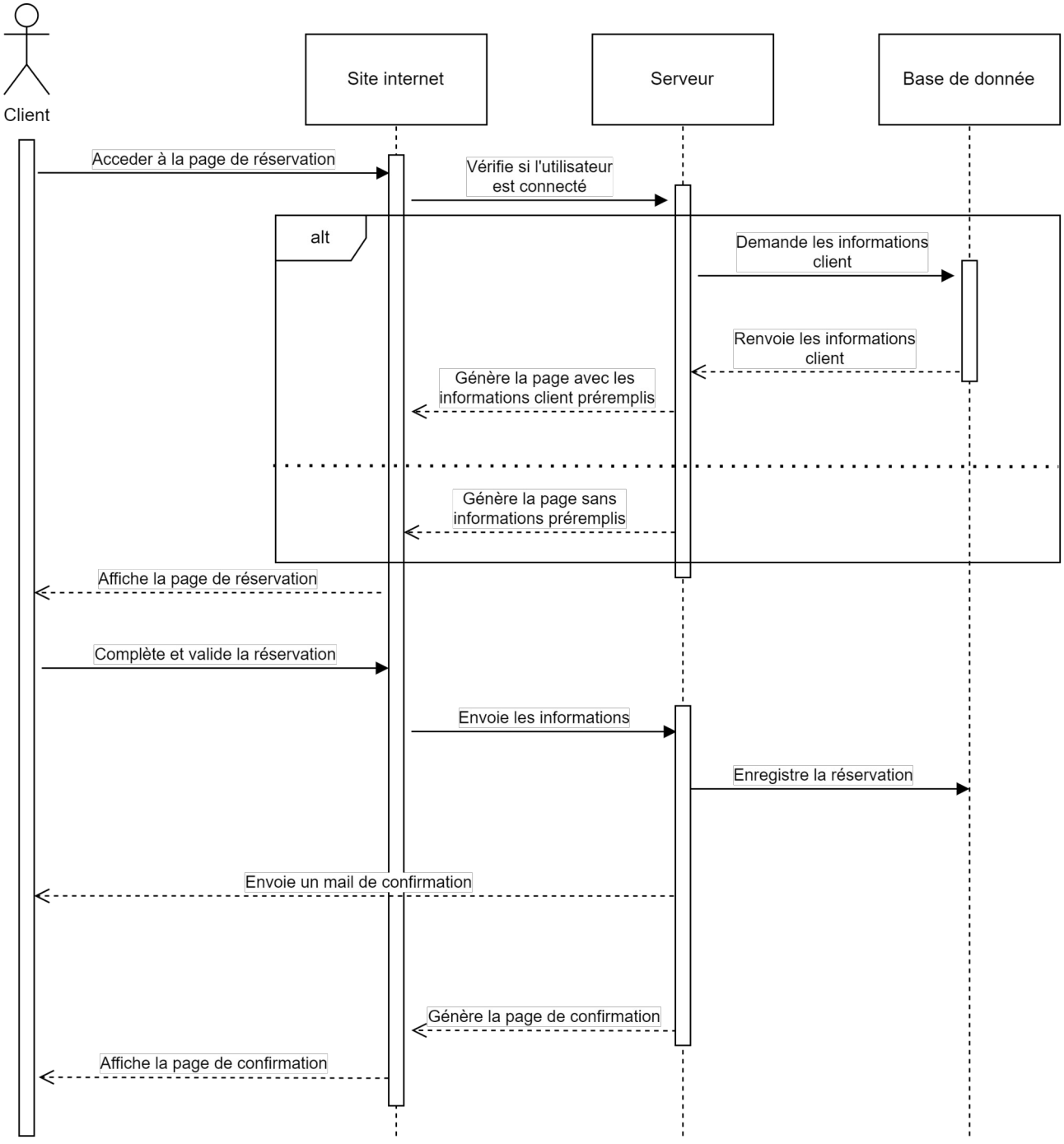
- PHP v. 8.2.2
- ORM Doctrine v. 2.14
- MySQL v 8.0
- Symfony 6.2

B. Diagramme de Cas d'utilisation



C. Diagrammes de séquence

Fonctionnalités « US6 – Réserver une table »



D. Diagramme de classe

Diagramme de classe