Documentation technique

Ce document présente la phase de « réflexion » du projet, ses diagrammes, et ses choix technologiques.

1. Réflexions initiale technologique

Back-end avec Symphony:

J'ai choisi de créer le back-end api avec symfony pour sa robustesse et sa modularité. La possibilité d'y ajouter des éléments comme l'authentification jwt rapidement et efficacement, ou ses mécanisme de sécurité intégrés, en font un atout majeur. Pour respecter les règles imposées par le projet, je n'ai pas utilisé Doctrine ORM et ai donc créé des entités (« model ») manuellement, en y intégrant les composants symfony nécessaires au fonctionnement de l'application. Enfin en installant webpack j'aurai la possibilité d'y « greffer » mon application React directement.

J'ai choisi « Firebase Storage » pour le stockage des images. La rapidité et la facilité de configuration d'un tel stockage ont été les principaux arguments pour ce choix.

Front-end avec React:

React est un framework reconnu pour sa gestion des états d'application et les mises à jour dynamique d'interface. Ses multiples composants ou encore son système de routage en font une librairie complète et performante pour de tels projets.

En résumé ces choix technologiques sont le résultat de considérations minutieuses visant à optimiser le développement, garantir la sécurité et fournir la meilleure expérience utilisateur possible.

2. Configuration de l'environnement de travail

Éditeur de Code : J'utilise Visual Studio Code (VS Code) en tant qu'éditeur principal en raison de sa flexibilité, et de son large éventail d'extensions qui supportent les technologies que j'utilise telles que React et Symfony.

Extensions VS Code :

PHP Intelephense : Pour l'auto-complétion php , et l'amélioration de la syntaxe.

ESLint et Prettier : Pour maintenir un style cohérent et identifier les erreurs de syntaxe dans le code.

Figma: Cette extension permet de récupérer rapidement et directement dans l'éditeur les propriétés des maquettes.

Serveurs de Développement Locaux :

Symfony Local Web Server : Utilisé pour servir le back-end de l'application. Ce serveur est intégré directement dans Symfony, offrant une solution facile à configurer et à utiliser pour le développement local. Il supporte automatiquement des configurations comme HTTPS et permet une intégration transparente avec Symfony CLI pour la gestion des routes, services, et autres.

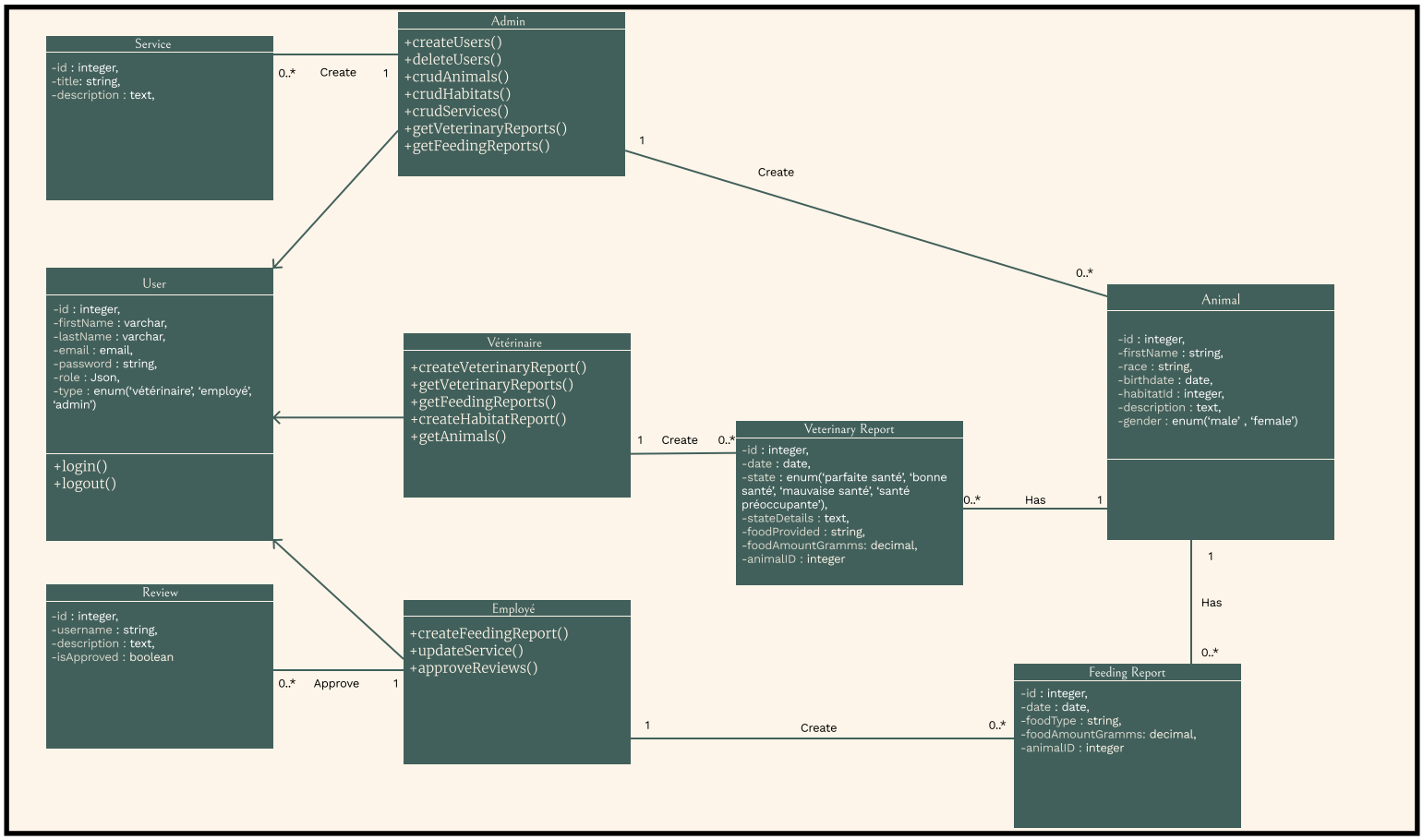
Webpack Dev Server : Utilisé pour le front-end React. Ce serveur permet le « Hot Module Replacement », signifiant que les modifications du code sont immédiatement visibles dans le navigateur sans nécessiter de rechargement complet.

Gestion des Dépendances :

Composer : Pour la gestion des dépendances PHP/Symfony, assurant que toutes les bibliothèques et frameworks côté serveur sont à jour.

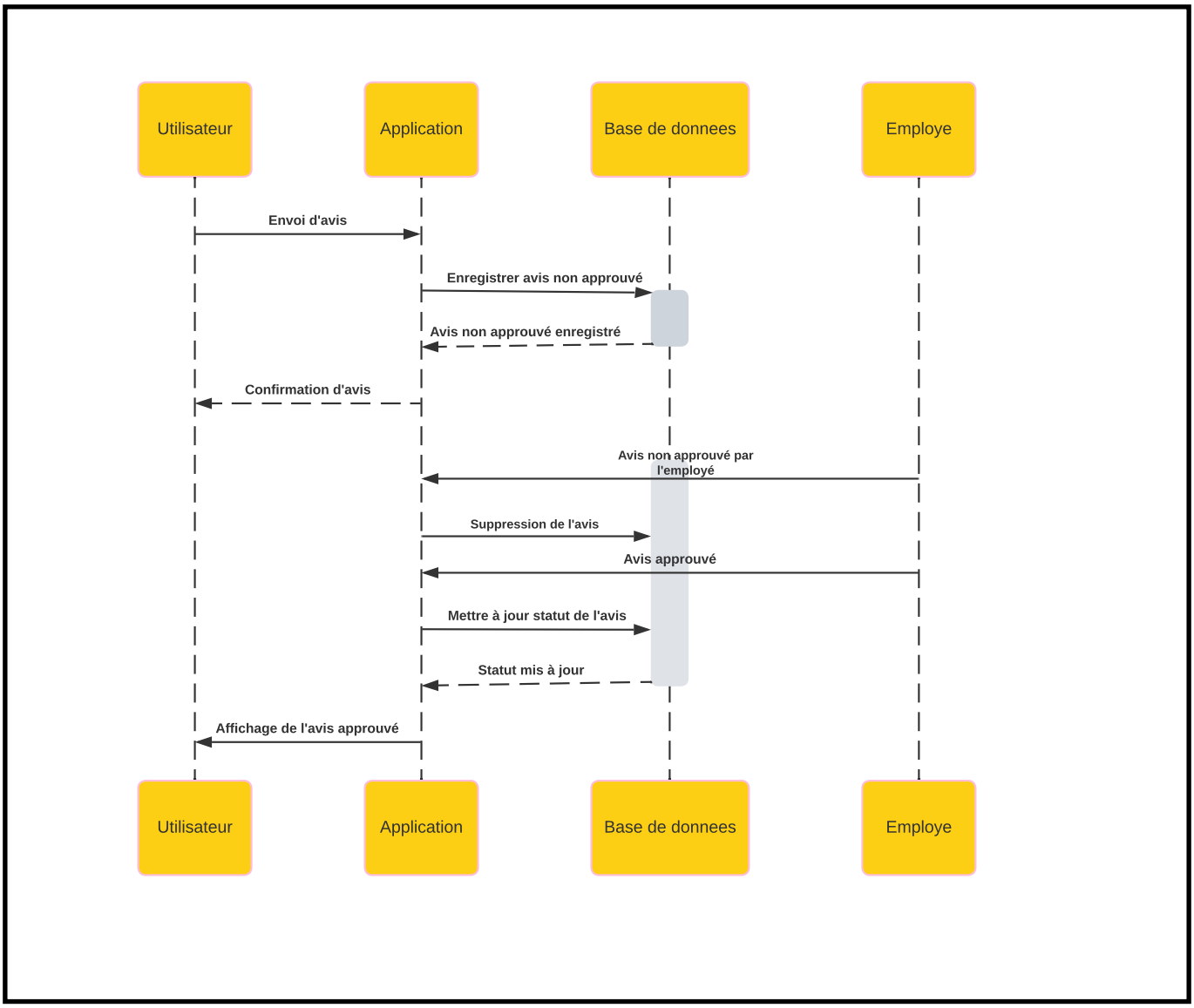
NPM/Yarn : Utilisé pour la gestion des dépendances nécessaires pour React.

3. Modèle conceptuel de données



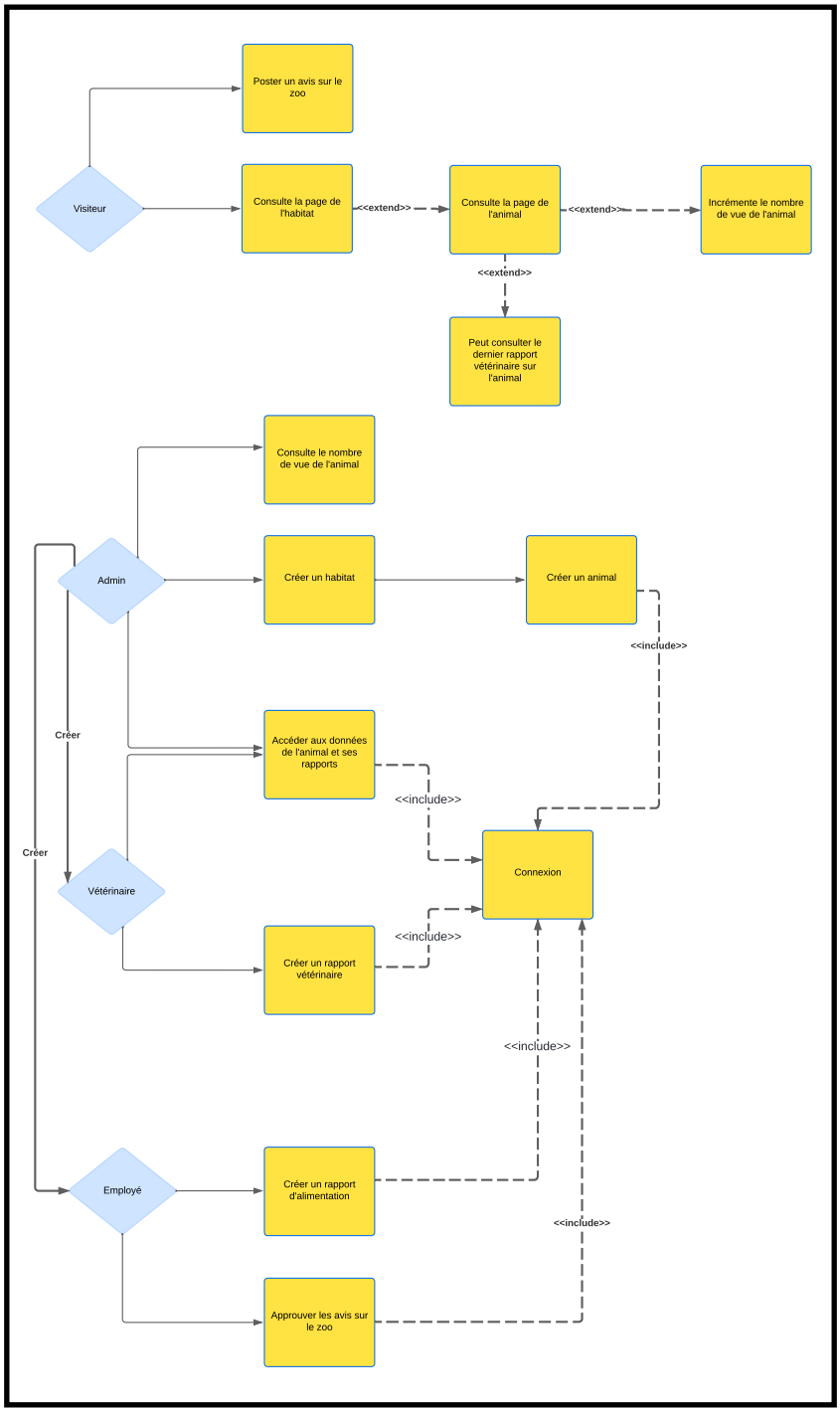
4. Diagramme de séquence

Pour le diagramme de séquence j'ai choisi d'illustrer la phase de traitement d'un avis posté par le visiteur sur l'application. L'avis est soumis à approbation avant d'être affiché au grand public sur la page d'accueil de l'application.



5. Diagramme de cas d'utilisation

Le cas d'utilisation traité par ce diagramme décrit la relation entre les différentes entités d'utilisateur autour de la gestion des animaux du zoo . Le diagramme n'est pas exhaustif de la totalité des utilisations possibles pour chaque entité, mais en présente la majorité.



6. Déploiement de l’application

1. Pré-requis

AVANT DE DÉMARRER LE PROCESSUS DE DÉPLOIEMENT DE MON APPLICATION, JE M’ASSURE QUE TOUS LES ENVIRONNEMENTS CONCERNÉS — DÉVELOPPEMENT, TEST, ET PRODUCTION — DISPOSENT DES INSTALLATIONS ET CONFIGURATIONS NÉCESSAIRES. VOICI LES PRÉREQUIS DÉTAILLÉS QUE JE VÉRIFIE :

Environnement Serveur

Serveur Web : J'utilise Apache 2.4 ou Nginx 1.18, conformément aux exigences de Symfony et React.

PHP : J'installe la version 8.0 ou supérieure et m'assure que toutes les bibliothèques PHP requises sont activées.

Composer : Je l'utilise pour gérer les dépendances PHP et je vérifie qu'il est installé correctement sur les serveurs.

Node.js et npm/yarn : Nécessaires pour le traitement des dépendances et des scripts JavaScript pour React, je vérifie leur présence et leur bonne configuration.

Base de données

Système de gestion de base de données : J'opte pour une base de donnée mySQL.

Base de donnée non-relationnelle: Firebase assurera le stockage des images et la partie non-relationnelle.

Outils de développement

Git : Indispensable pour la gestion des versions du code source.

IDE : Utilisation de Visual Studio Code, qui supporte les langages utilisés dans le projet via ses extensions.

2. Tests en local

Une fois la version mobile terminée, et avant mise en production, les tests suivants ont été effectués en local :

- Toutes les routes de l'api via Postman api.
- Le système de route privée empêchant l'accès à des pages réservée à l'équipe du parc via un test Jest.
- Le système d'authentification JWT via un test php unit.
- Les différents parcours possible, manuellement grâce au serveur de développement.
- Un test de sécurité global avec OWASP ZAP.

Une fois ces tests concluants, la branche main a été « rebasé » sur la branche « develop ».

3. Configuration et déploiement sur Heroku

Après avoir créé un compte sur Heroku et installé le CLI de Heroku sur ma machine, la configuration de l'environnement de production a pu commencer.

Configuration des buildpacks :

La première étape fut la configuration des buildpacks nécessaires à mon application. Pour cela, j'ai ajouté le buildpack PHP pour gérer l'environnement Symfony et le buildpack Node.js pour compiler les assets React avec Webpack. J'ai utilisé les commandes suivantes pour ajouter ces buildpacks à mon projet Heroku :

$heroku buildpacks:add heroku/php —app arcadia-webapp

$heroku buildpacks:add heroku/nodejs --app arcadia-webapp

Configuration des variables d'environnement :

J'ai défini les variables d'environnement nécessaires à travers le tableau de bord Heroku, y compris les paramètres de la base de données, les clés API et d'autres configurations essentielles.

Configuration de la base de donnée :

Après définition des variables d'environnement, j'ai créé l' « add-on » heroku permettant d'initialiser une base de donnée mySQL JawsDb. Une fois créé, j'ai pu lancé les commandes de création de la base de donnée , des tables et de l'administrateur de mon projet.
Pour se faire , une fois connecté au projet heroku dans le terminal, il suffit de lancer les commandes de création décrites dans le fichier ReadMe du projet.

Déploiement de l'application :

Pour déployer l'application, j'ai poussé le code depuis mon dépôt Git local vers le dépôt Heroku en utilisant Git. La commande suivante permet de déployer l'application :

$git push heroku main

Heroku détecte automatiquement les changements et lance les processus de build et de déploiement en utilisant les buildpacks configurés.

Vérification du déploiement :

Après le déploiement, j'ai vérifié que l'application fonctionne correctement en accédant à l'URL fournie par Heroku. J'ai également utilisé le CLI de Heroku pour vérifier l'état de l'application et consulter les logs en cas d'erreurs.

Automatisation des déploiements :

Pour les déploiements futurs j'ai choisi d'instaurer une intégration continue depuis le dépôt git afin d'automatiser le déploiement sur Heroku à chaque push sur la branche main.

Par manque de temps, je n'ai malheureusement pas pu intégrer les tests automatisés au « workflow » mais il sera bénéfique de les intégrer ultérieurement.

Cette approche me permet de bénéficier de la simplicité et de la puissance de Heroku pour gérer l'infrastructure, sans avoir à configurer et maintenir un serveur personnellement.