Documentation Technique de l'Application Zoo d'Arcadia

Réflexions Initiales Technologiques

L'application du zoo Arcadia est conçue pour offrir une expérience aux visiteurs dynamique et variée tout en fournissant également des outils de gestion efficaces au personnel et aux administrateurs du zoo. Les outils utilisés et les raisons de ces choix sont expliqués ci-dessous:

Framework Utilisé:

Symfony a été choisi car c'est un outil puissant et adaptable, ce qui le rend idéal pour créer des applications Web complexes. Les outils de génération de code et la configuration claire via les fichiers.env facilitent la mise en œuvre. La structure modulaire et la documentation complète facilitent la création et la gestion de fonctionnalités complexes. De plus, sa compatibilité avec la doctrine orm facilite la gestion efficace des bases de données relationnelles, intégrant les fonctionnalités avancées requises pour le projet.

Bases de Données :

MySQL a été choisi pour son intégration facile avec Symfony via doctrine orm ,sa fiabilité et sa facilité de gestions de données relationnelles.De plus MySQL est reconnu pour sa stabilité et ses performances. Enfin, son adoption généralisée garantit un soutien communautaire fort et des outils d'administration conviviaux.

Mongodb a été choisi pour sa facilité de mise en place, de configuration et de traitement des données non relationnelles, notamment des statistiques de consultation. La structure du document permet le stockage de diverses données sans schéma prescrit, ce qui en fait une option appropriée pour modifier les données. Le bundle mongodb qui intègre mongodb à symfony permet une évolutivité et des performances pouvant être ajustées à la croissance des données.

Gestion des Dépendances :

Pour la gestion des dépendances PHP, **Composer** a été choisi car il simplifie le processus d'installation et de mise à jour des bibliothèques requises pour PHP, tout en gérant les conflits potentiels de bibliothèques. Le développement est facilité par sa capacité à assurer la compatibilité avec Symfony et les dépendances.

Enfin **pour gérer les dépendances Javascript et et les outils de build tels que webpack,Npm a** quant à lui été choisi car il offre flexibilité et efficacité dans la gestion des assets front-end et la compilation du code.

Configuration de l'Environnement de Travail

Au début du projet, voici comment j'ai mis en place mon environnement de travail en utilisant les outils choisis :

1. **Vérification des Exigences avec Symfony**

Avant de commencer le développement, j'ai utilisé la commande suivante pour vérifier que l'environnement de développement satisfaisait les exigences minimales de Symfony:

symfony check:requirements

2. **Création du Projet Symfony**

La création du projet a été effectuée en utilisant Composer pour initialiser un nouveau projet Symfony :composer create-project symfony/website-skeleton zoo_d-arcadia

Cette commande configure un squelette de projet Symfony avec les paramètres de base nécessaires pour commencer le développement.

3. **Configuration des Variables d'Environnement**

Un fichier .env.local a été créé à la racine du projet pour configurer les variables d'environnement nécessaires:

DATABASE_URL=mysql://user:password@localhost:3306/nom_de_la_base_de_donnees

MONGODB_URL=mongodb://localhost:27017/nom_de_la_base_mongodb

MAILER_DSN=smtp://localhost:25

Ce fichier permet de définir les paramètres de connexion aux bases de données et au serveur de messagerie.

4. **Installation des Dépendances PHP**

Les dépendances PHP nécessaires au projet ont été installées en exécutant :

composer install

Cette commande télécharge et installe toutes les bibliothèques spécifiées dans le fichier composer.json.

5. **Installation des Dépendances JavaScript**

Les modules JavaScript et les outils de construction front-end ont été installés et configurés avec npm :

npm install

npm run dev

Cela permet de préparer les assets front-end et de configurer les outils nécessaires pour le développement.

6. **Configuration de la Base de Données**

La base de données MySQL a été configurée en créant la base de données spécifiée dans DATABASE_URL :

php bin/console doctrine:database:create

Ensuite, les migrations ont été générées et appliquées pour créer les tables nécessaires :

php bin/console make:migration

php bin/console doctrine:migrations:migrate

7. **Lancement du Serveur de Développement**

Le serveur de développement Symfony a été démarré pour vérifier le bon fonctionnement de

l'application :

php bin/console server:run

L'application est alors accessible à l'adresse http://localhost:8000, permettant de tester et de développer les fonctionnalités.

J'ai ensuite commencé à developper la vue de l'application en reprenant les maquettes créees sur Figma afin d'avoir une vue d'ensemble de l'application et des endroits où implanter les différents fonctionnalités.

En suivant ces étapes, l'environnement de travail a été correctement configuré pour le développement de l'application **Zoo d'Arcadia**, assurant une base solide pour les fonctionnalités prévues et une gestion efficace des différentes parties du projet.

Modèle conceptuel de données:

En m'inspirant du MCD donné dans le sujet et au fil du développement, de l'ajout de tables, de modification et d'ajouts d'entrées dans certaines tables, voici mon MCD final représentant ma base de données:

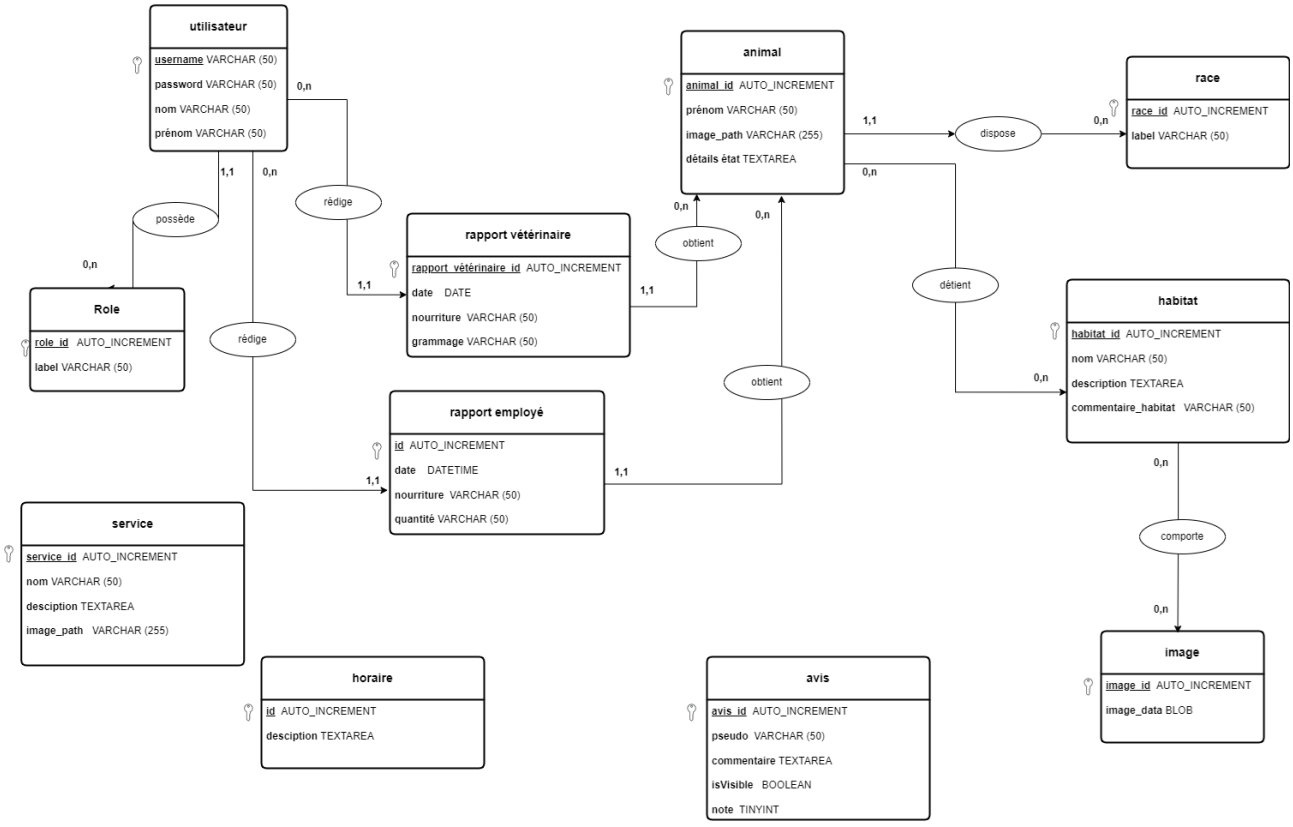


Diagramme d'utilisation :

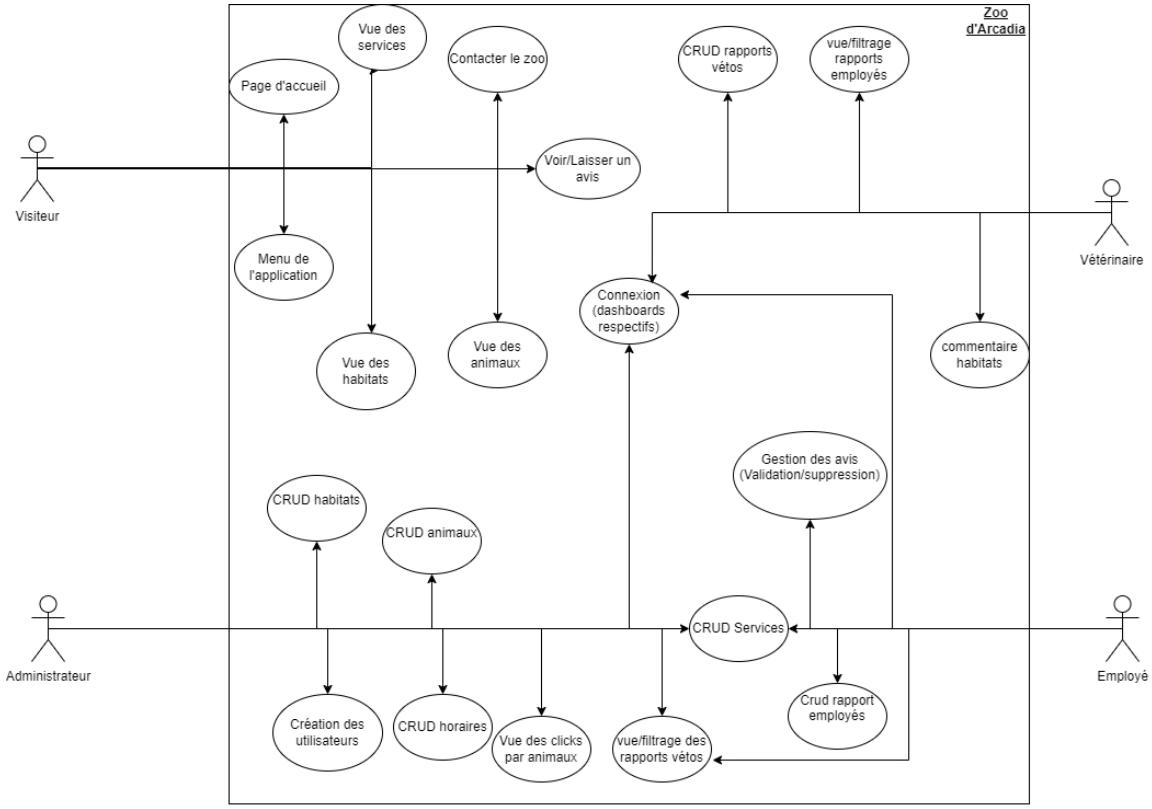

Zoo d'Arcadia
Vue des services
Contacter le zoo
CRUD rapports vétos
vue/filtrage rapports employés
Page d'accueil
Voir/Laisser un avis
Visiteur
Vétérinaire
Menu de l'application
Connexion (dashboards respectifs)
Vue des habitats
Vue des animaux
commentaire habitats
Gestion des avis (Validation/suppression)
CRUD habitats
CRUD animaux
Administrateur
CRUD Services
Employé
Crud rapport employés
Création des utilisateurs
CRUD horaires
Vue des clicks par animaux
vue/filtrage des rapports vétos

Diagramme de séquence visiteur:

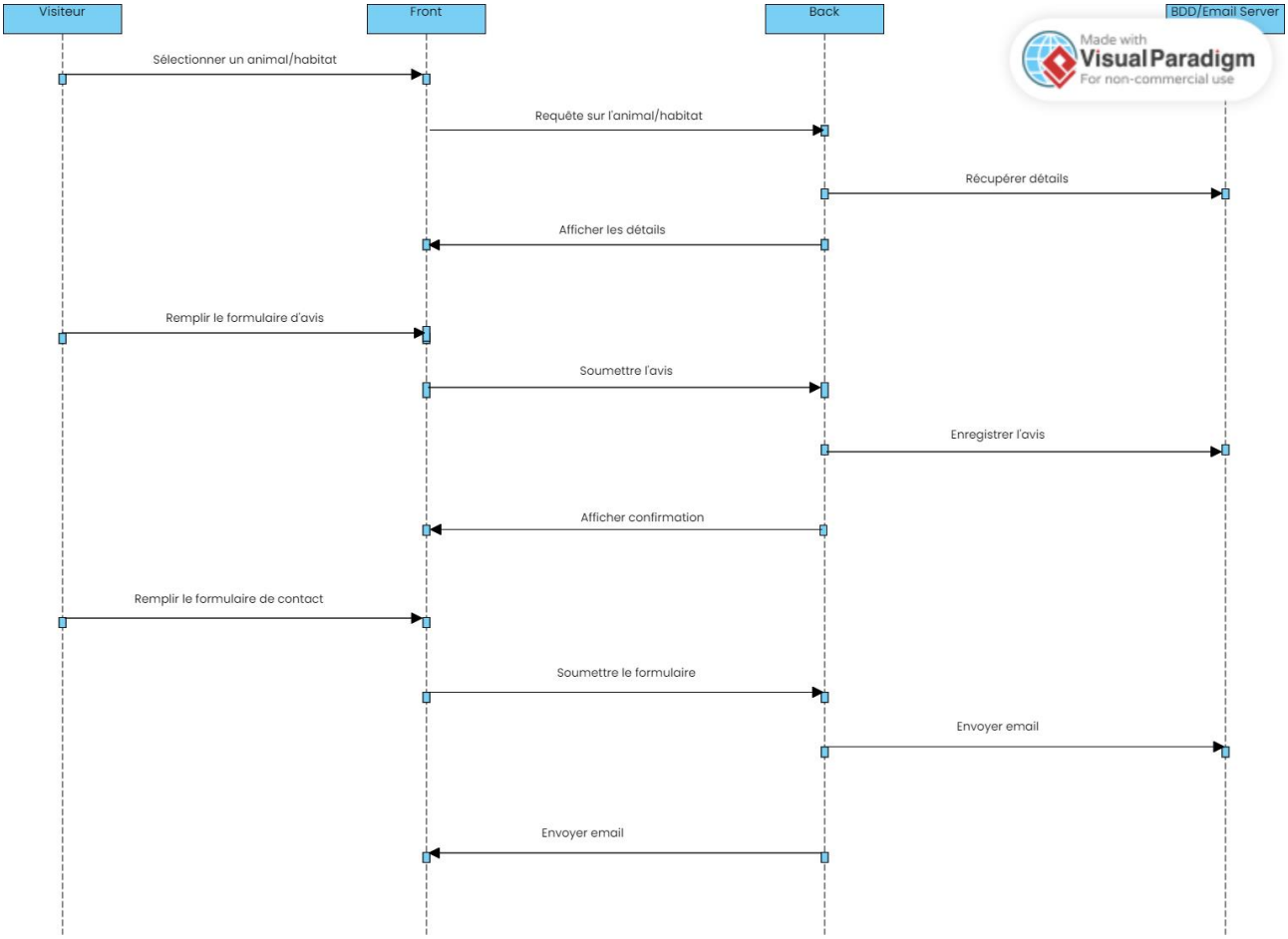


Diagramme de séquence administrateur:

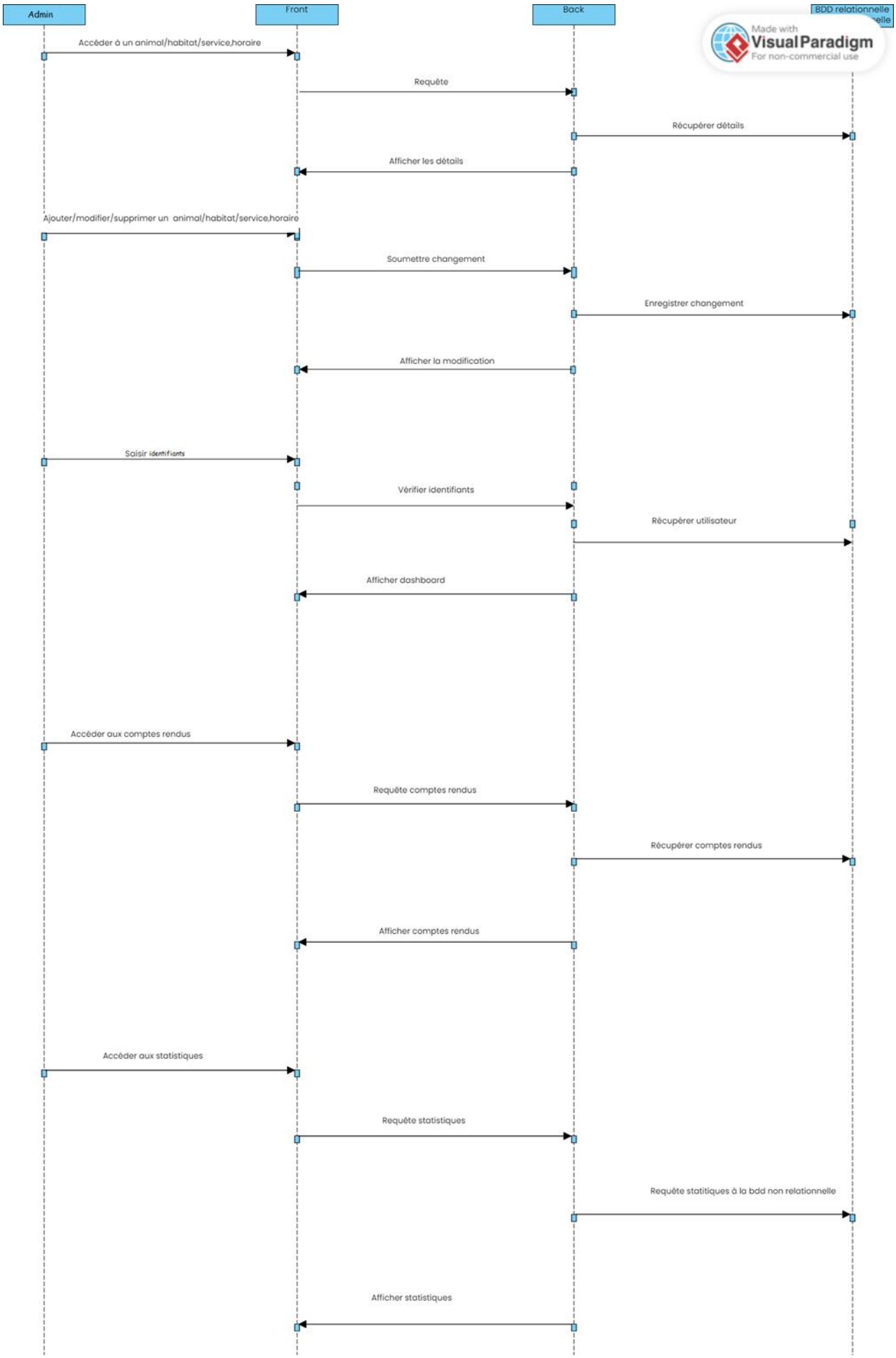

Admin
Front
Back
BDD relationnelle
Made with VisualParadigm For non-commercial use
Accéder à un animal/habitat/service,horaire
Requête
Récupérer détails
Afficher les détails
Ajouter/modifier/supprimer un animal/habitat/service,horaire
Soumettre changement
Enregistrer changement
Afficher la modification
Saisir identifiants
Vérifier identifiants
Récupérer utilisateur
Afficher dashboard
Accéder aux comptes rendus
Requête comptes rendus
Récupérer comptes rendus
Afficher comptes rendus
Accéder aux statistiques
Requête statistiques
Requête statitiques à la bdd non relationnelle
Afficher statistiques

Diagramme de séquence employé:

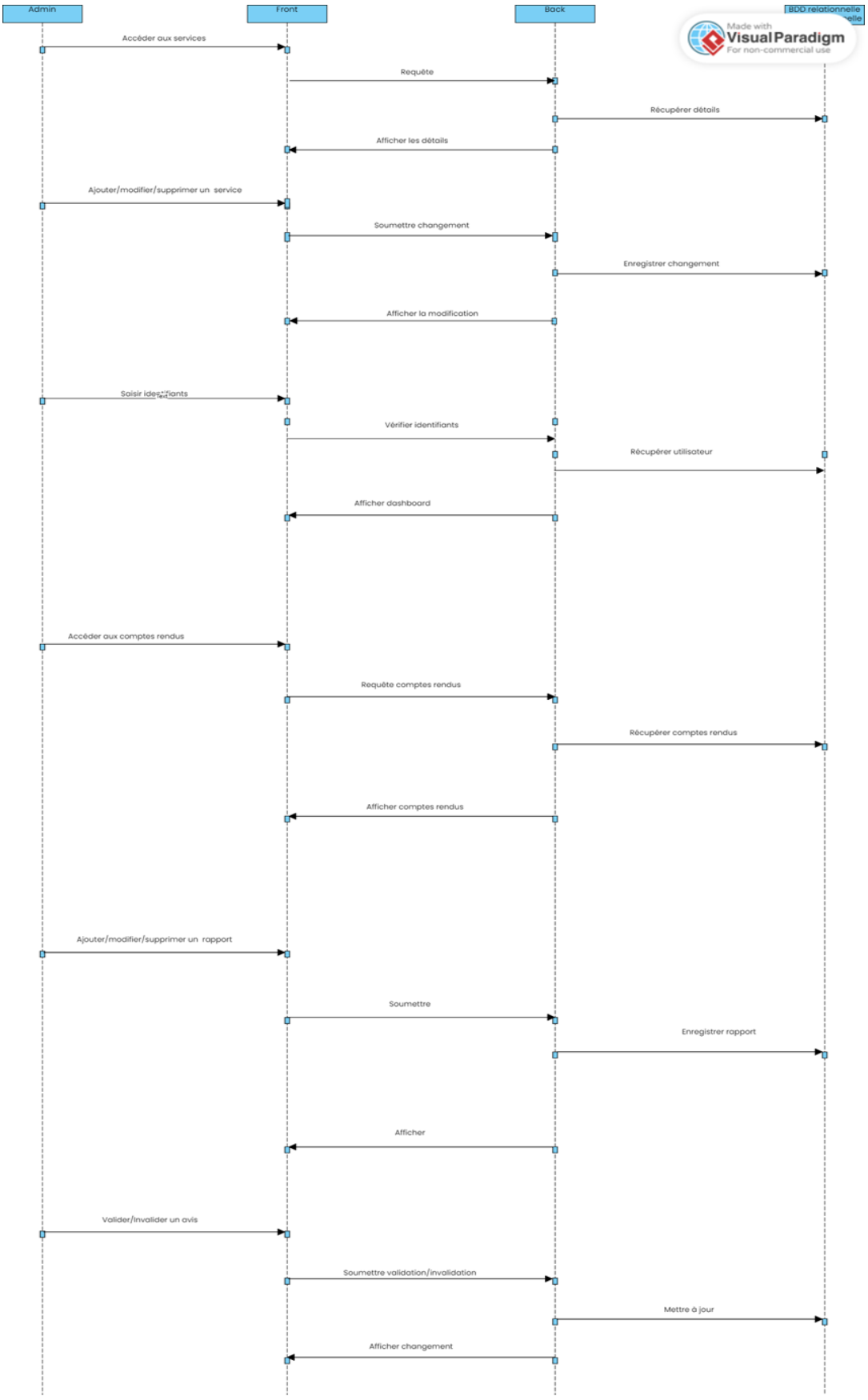
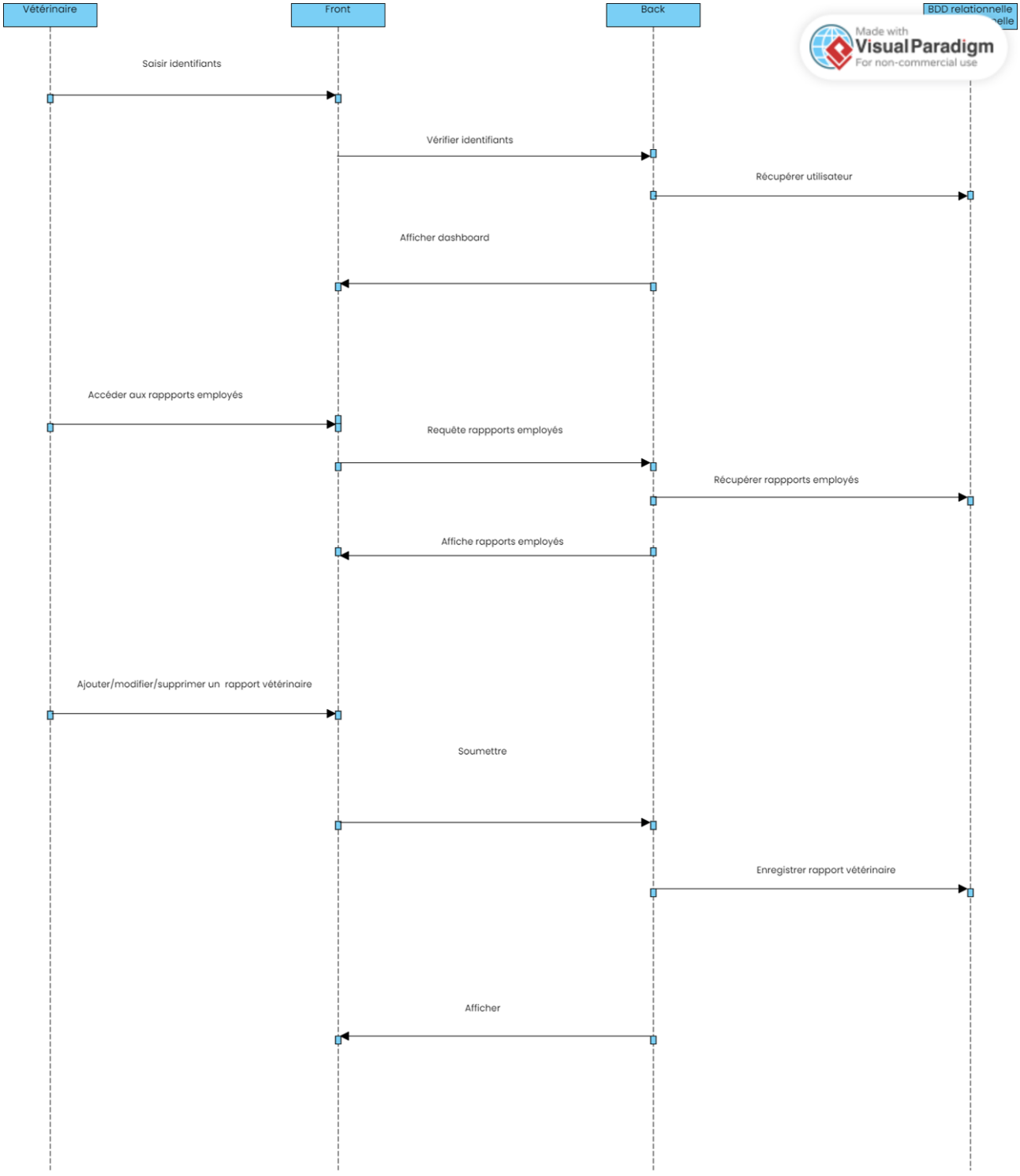


Diagramme de séquence vétérinaire :



Mis en place du Déploiement de l'Application Zoo d'Arcadia

Démarche de Déploiement sur Heroku

Pour déployer l'application Zoo d'Arcadia sur Heroku, j'ai suivi une série d'étapes pour assurer une transition fluide de l'environnement de développement à l'environnement de production. Voici la démarche que j'ai suivie ainsi que les problèmes rencontrés et leurs résolutions.

1. Installation de l'outil Heroku CLI pour pouvoir intéragir directement avec la plateforme depuis le terminal puis connexion à Heroku via la commande *heroku login.*

2. Création du projet avec la commande *heroku create*

3. Configuration du projet en plusieurs étapes:

- Ajout du fichier Procfile :

J'ai crée un fichier nommé Procfile à la racine de mon projet pour indiquer à Heroku comment exécuter l'application dont voici le contenu:

'web: heroku-php-apache2 public/'

- Configuration des Buildpacks :

J'ai ajouté les buildpacks nécessaires pour PHP avec la commande heroku buildpacks:set heroku/php

- Configuration du Dépôt Distant Heroku :

Ensuite, il a fallu configurer Heroku comme dépôt distant avec la commande *git remote add heroku <git-de-mon-projet>* (donné lors de la création du projet)

4. Il a ensuite fallu configurer les variables d'environnement avec la commande *heroku config:set APP_ENV=prod APP_SECRET=Wh4t*

5. Le projet a ainsi pu être déployé avec les commandes:

git add .

git commit -m "Initial files"

git push heroku main

6. Je me suis rendu compte à ce moment là que mon site n'était pas fonctionnel pour une raison simple: je n'avais pas configuré mes variables d'environnement de base de données. Il a donc fallu configurer heroku avec MySQL et MongoDB.

MySQL n'étant pas proposé nativement sur heroku, il a donc fallu utiliser l'addon ClearDB et migrer les données de MySQL à ClearDB.

La commande *heroku addons:create cleardb:punch8* m'a permis de rajouter l'addon a mon projet.

Après l'ajout de l'addon, il a fallu récupérer l'URL de la base de données et mettre à jour les

variables d'environnement avec les commandes:

heroku config CLEARDB_DATABASE_URL

heroku config:set DATABASE_URL='mysql://user:password@us-cdbr-east-05.cleardb.net/heroku_dbname'

7. Pour la base de données non relationnelle, j'ai utilisé MongoDB Atlas. Après avoir configuré un cluster MongoDB sur MongoDB Atlas. J'ai récupéré l'URI de connexion fourni par MongoDB Atlas, qui ressemble à ceci :

mongodb+srv://<username>:<password>@cluster0.mongodb.net/<dbname>?retryWrites=true&w=majority.

Il a fallu aussi configurer cette URI en tant que variable d'environnement sur Heroku avec la commande:

heroku config:set MONGODB_URI='mongodb+srv://<username>:<password>@cluster0.mongodb.net/<dbname>?retryWrites=true&w=majority'

Aussi, afin d'assurer une bonne connexion aux bases de données après le déploiement, il a fallu modifier certains fichiers de configuration, le services.yaml pour que la configuration de MongoDB utilise l'URI de connexion plutôt qu'une URL fixe. Cela permet de s'adapter à l'URI fourni par MongoDB Atlas et aux variables d'environnement définies sur Heroku et le fichier MongoDBService.php pour faire appel à l'URI de connexion MongoDB, ce qui assure que l'application se connecte correctement à MongoDB Atlas. L'ancienne configuration, qui utilisait une URL de connexion statique, a été remplacée par une configuration dynamique qui utilise l'URI stocké dans les variables d'environnement.

7. La commande *heroku logs –tail*, m'a aidé tout au long du déploiement afin de savoir les erreurs et comprendre où le problème se situait.

8. Une fois le déploiement terminé, j'ai accédé à mon application via la commande :

heroku open

Ainsi,cette démarche m'a permis de déployer mon application Symfony sur Heroku avec succès, en veillant à la configuration correcte des bases de données, des variables d'environnement et des buildpacks.