Documentation technique

-

ZOO ARCADIA

Table des matières

I - Réflexions initiales technologiques sur le sujet

Afin de réaliser mon application Web, je me suis servie des outils et langages suivants :

- Front-end :
  - **HTML5** : ce langage me permet de créer, structurer et organiser mes pages Web. Il me permet de réaliser le squelette des pages, de décrire le contenu textuel et les différents éléments que je souhaite afficher sur le navigateur ;
  - **CSS3** : ce langage me permet de styliser mon code HTML. Il me permet de contrôler la mise en page, la typographie, les couleurs, les bordures, les arrière-plans, … ;
  - **Bootstrap** : ce framework CSS me permet de gagner en rapidité sur la mise en place d'une solution responsive, plus cohérente et harmonieuse ;
  - **Javascript ES6.0** : ce langage me permet de dynamiser l'affichage de mon site. Il permet de changer l'affichage sans avoir à recharger le contenu de mes pages web si nécessaire ;

- Back-end :
  - **PHP 8.3.11** : ce langage a été spécialement conçu pour le développement d'applications Web et il s'intègre facilement au langage HTML. Il me permet de créer des pages web dynamiques grâce à sa capacité à se connecter à des bases de données et donc à personnaliser le contenu des pages en fonction de chaque accès ;
  - **Composer** : ce gestionnaire de dépendances permet de se connecter à la base de données non relationnelle via la **dépendance mongodb/mongodb**

- Bases de données :
  - **NoSQL** : Certaines données n'ont aucune relation les unes avec les autres (avis, horaires). J'ai donc décidé de les intégrer dans une base de données non relationnelle. Pour développer ma base de données non relationnelle, je me suis servie de l'outil en ligne **MongoDb Atlas**. Cela permet de faciliter le déploiement. La base de données étant déjà hébergée dans le cloud, je n'ai pas

besoin de la déployer avec mon application. Pour visualiser les actions sur mes données, j'ai utilisé le logiciel **Compass** de MongoDB.

  - SQL : **MariaDB version 10.11** : Pour la base de données relationnelle, j'ai suivi le même raisonnement que précédemment et j'ai utilisé le service en ligne **AlwaysData**. Pour visualiser les actions sur mes données, je me suis servie du logiciel **DataGrip**.

- Déploiement :
  - **Heroku** : J'ai choisi de déployer mon application sur Heroku car cette plateforme est simple d'utilisation et gratuite (avec le GitHub Student Developer Pack).

- IDE (Integrated Development Environment) : j'ai choisi de travailler sur l'IDE **PHP Storm** car il inclut déjà tout le nécessaire pour développer en PHP et Javascript. Pas besoin d'installer une multitude d'extensions. Le GitHubStudent Developer Pack m'a permis de tester ce logiciel gratuitement.

II - Configuration de mon environnement de travail

Afin de pouvoir développer mon application et vérifier le travail effectué au fur et à mesure de mon avancement, j'ai installé un environnement de travail me permettant de le faire :

- Installation de PHP Storm via le site de JetBrains et en ayant activé le GitHubStudent Developer Pack.
- Installation de PHP 8.3.11 avec Homebrew à travers la commande « brew install php » via le terminal de commandes de mon ordinateur
- Installation de Git avec Homebrew à travers la commande « brew install git »
- Installation de Composer à la racine de mon projet via le terminal de commandes de mon IDE : les commandes sont récupérées sur le site https://getcomposer.org et sont les suivantes (il est indispensable d'avoir installé PHP au préalable) :

```
- php -r "copy('https://getcomposer.org/installer',
  'composer-setup.php');"
- php -r "if (hash_file('sha384', 'composer-setup.php') ===
  'dac665fdc30fdd8ec78b38b9800061b4150413ff2e3b6f88543c636f
  7cd84f6db9189d43a81e5503cda447da73c7e5b6') { echo
  'Installer verified'; } else { echo 'Installer corrupt';
  unlink('composer-setup.php'); } echo PHP_EOL;"
- php composer-setup.php
- php -r "unlink('composer-setup.php');"
```

- Installation de la dépendance mongodb/mongodb à la racine de mon projet via le terminal de mon IDE à travers la commande : « php composer.phar require mongodb/mongodb »
- Création d'un compte sur le site AlwaysData puis j'ajoute une base de données MySQL

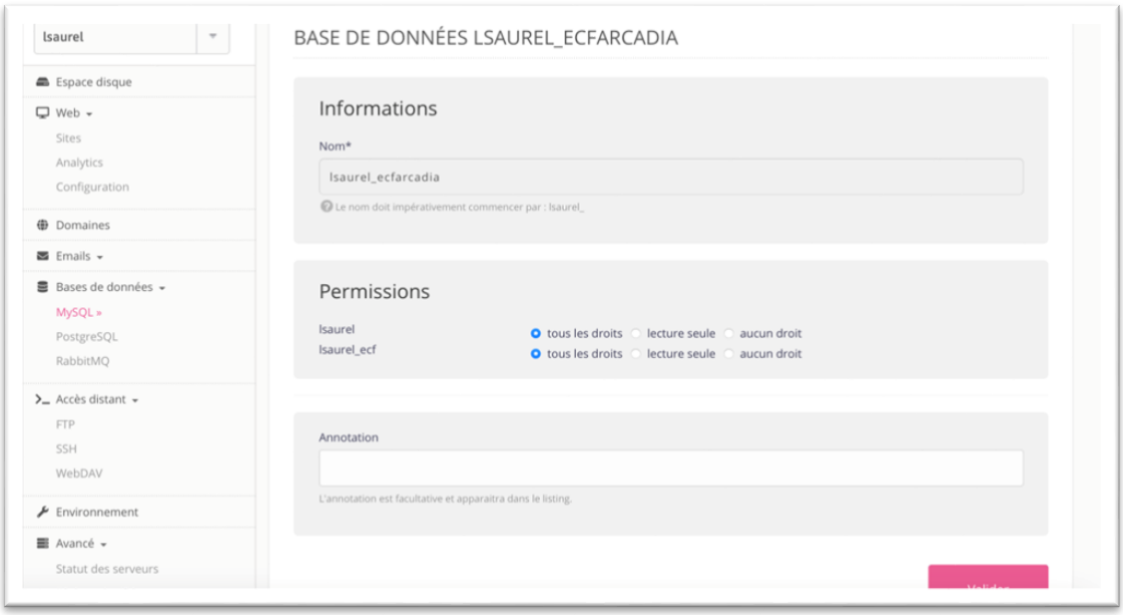

Figure 1 - Ajout d'une base de données dans AlwaysData

Figure 2 - Configuration de la base de données AlwaysData

- Création d'un utilisateur spécifique à cette base de données (lsaurel_ecf). C'est cet utilisateur que j'utiliserai dans mon code PHP pour accéder à la base de données SQL.

- Installation de Datagrip via le site de Jetbrains, en ayant activé le GitHubStudent Developer Pack puis configuration de la source de données pour me connecter à la base de données fournie par AlwaysData :

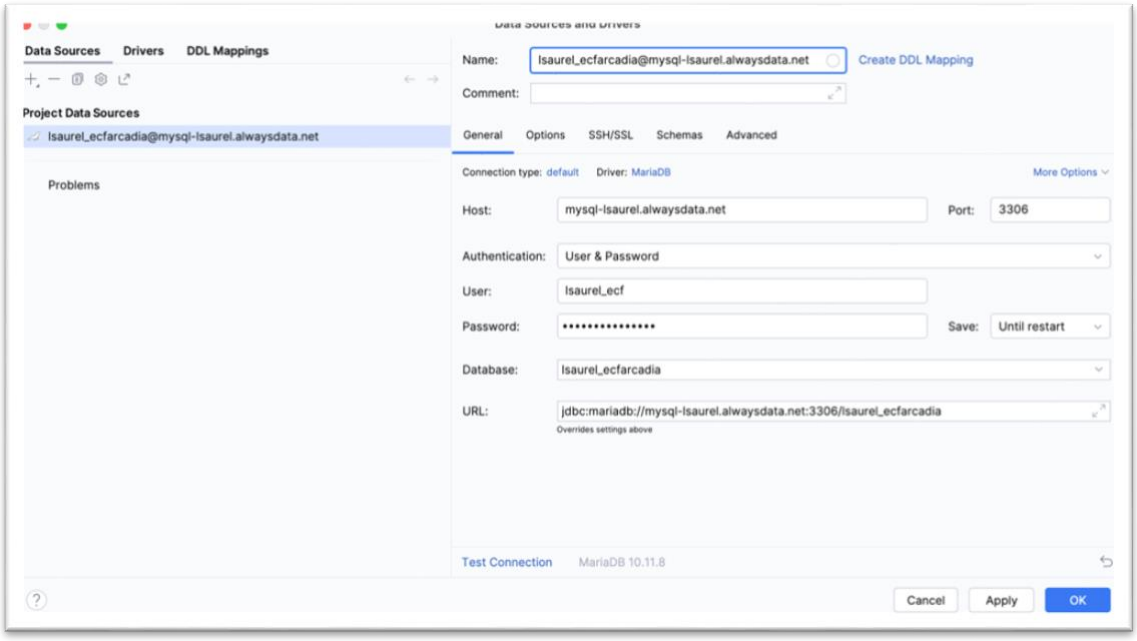

Figure 3 - Connexion à la source de données Datagrip

- Création du compte client MongoDB Atlas et création d'un cluster pour le zoo Arcadia

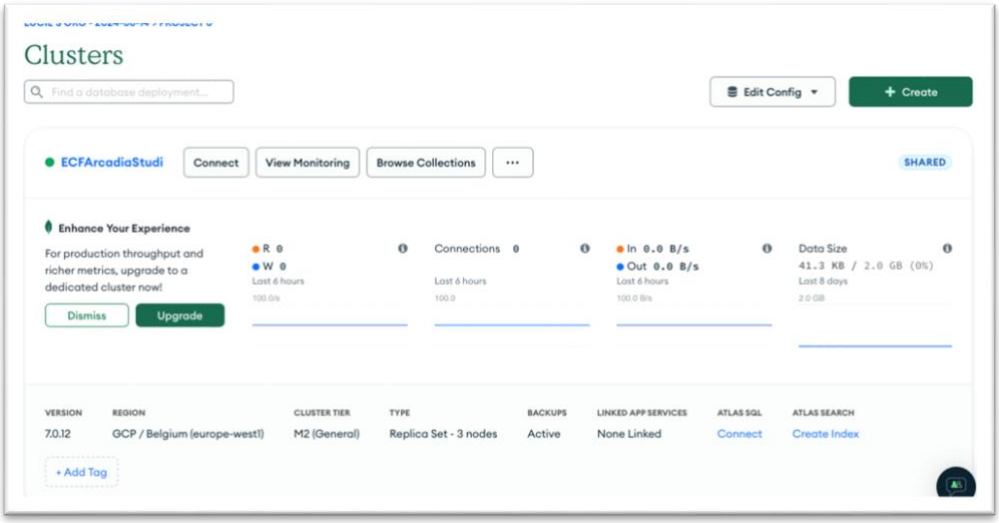

Figure 4 - Cluster ECFArcadia dans MongoDB Atlas

- Installation de MongoDB Compass en cliquant sur le bouton « Connect » du cluster. MongoDB Atlas indique également toute la marche à suivre pour se connecter à ce cluster :

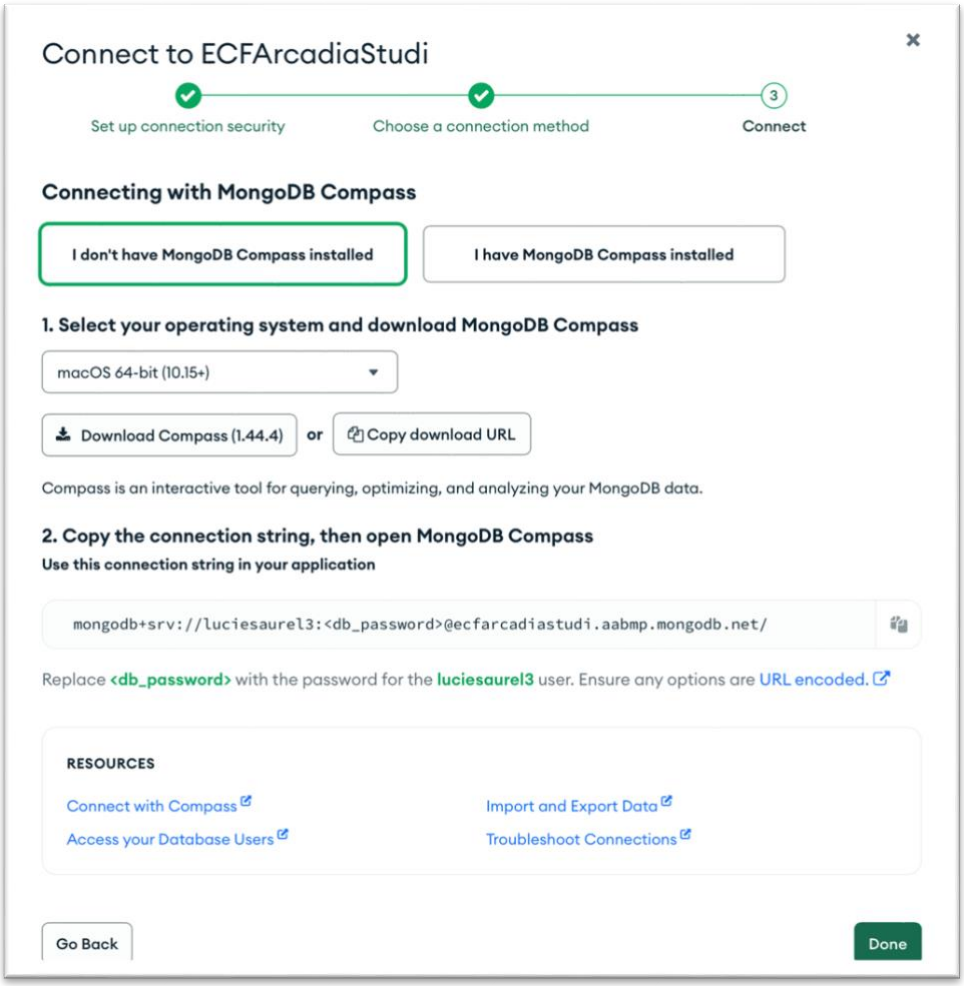

Figure 5 - Connexion au cluster ECFArcadiaStudi et installation de Compass

Ensuite, j'ouvre MongoDB Compass et je configure la source de données pour me connecter à la base de données fournie par MongoDB Atlas :

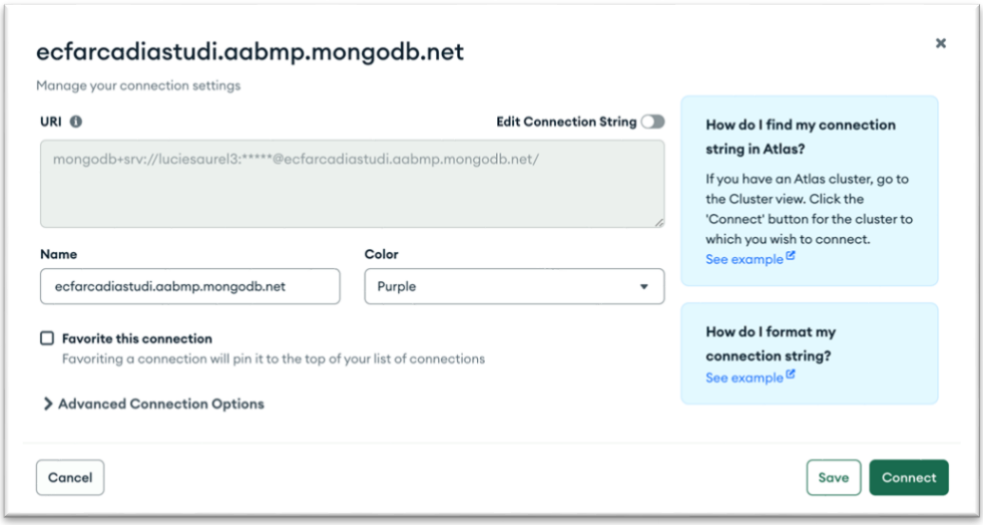

Figure 6 - Connexion à la base de données MongoDB Atlas

III - Modèle conceptuel de données (MCD)

Le modèle conceptuel de données est utilisé pour représenter l'ensemble des données de l'application et leur lien les unes avec les autres.

Je choisis de repenser le MCD en rapport avec ma compréhension du sujet.

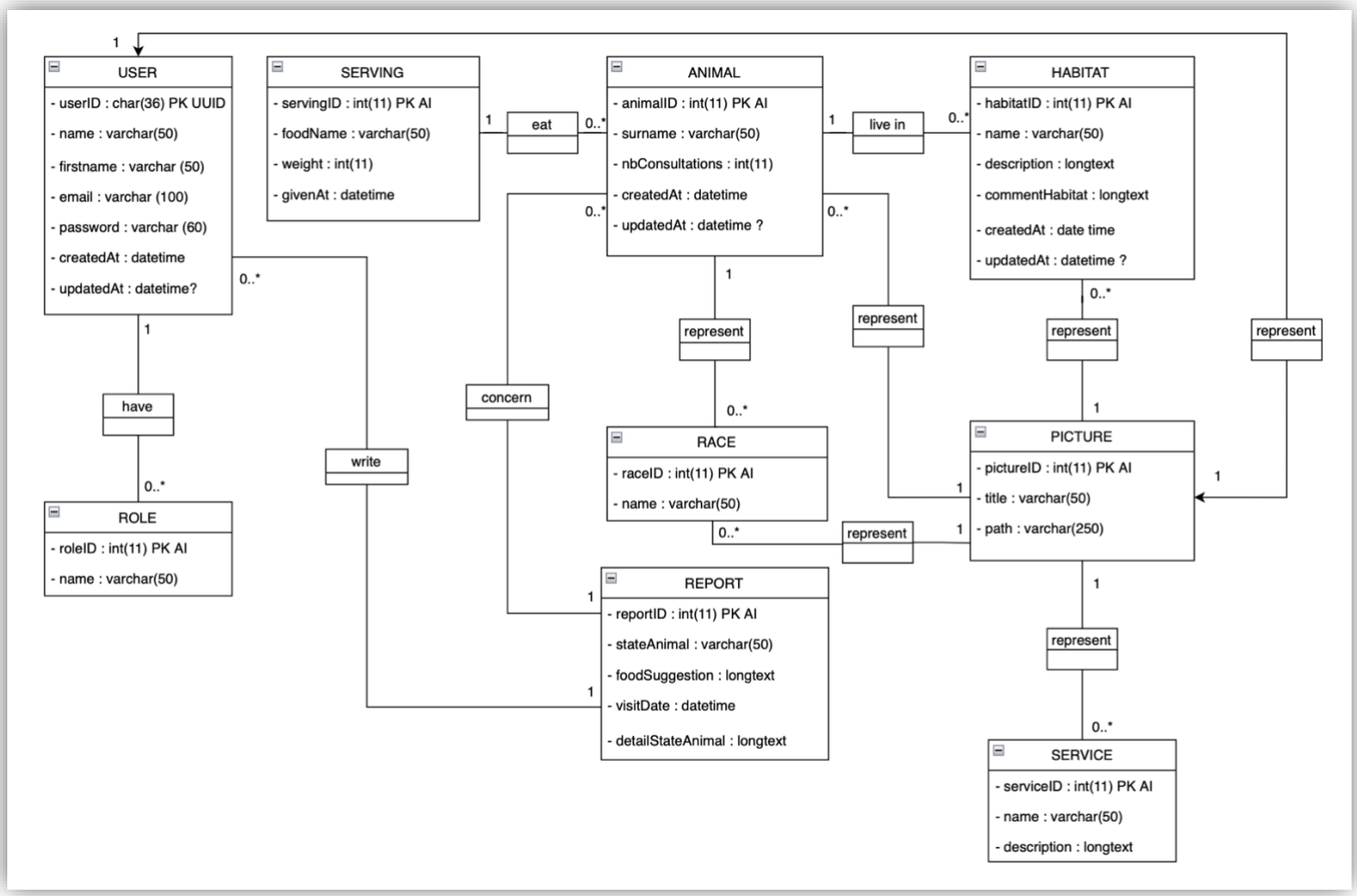

Figure 7 - Diagramme MCD

Voici le détail explicatif des relations entre les différentes tables :

- Relation One-to-Many Animal/Habitat:
  - Un animal ne vit que dans un habitat.
  - Un habitat peut accueillir plusieurs animaux.
- Relation One-to-Many Animal/Race:
  - Un animal est caractérisé par une seule race.
  - Une race peut être représentée par plusieurs animaux.

- Relation One-to-Many Animal/Picture:
    - Un animal peut être représenté sur plusieurs photos.
    - Une photo ne représente qu'un seul animal.
- Relation One-to-Many Habitat/Picture:
    - Un habitat peut avoir plusieurs photos.
    - Une photo ne représente qu'un seul habitat.
- Relation One-to-Many Utilisateur/Role :
    - Un utilisateur ne peut avoir qu'un rôle.
    - Un rôle peut être possédé par plusieurs utilisateurs.
- Relation One-to-Many Rapport Vétérinaire / Animal :
    - Un rapport vétérinaire ne concerne qu'un seul animal.
    - Un animal peut être concerné par plusieurs rapports vétérinaires.
- Relation One-to-Many Nourriture / Animal :
    - Une portion de nourriture est dédiée à un animal.
    - Un animal peut avoir plusieurs portions de nourriture.
- Relation One-to-Many Service/Picture :
    - Un service peut avoir plusieurs photos.
    - Une photo ne correspond qu'à un seul service.
- Relation One-to-Many Race/Picture :
    - Une race peut avoir plusieurs photos.
    - Une photo ne correspond qu'à une seule race.
- Relation One-to-One User/Picture :
    - Un utilisateur peut avoir une seule photo.
    - Une photo ne correspond qu'à un seul utilisateur.

IV - Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation permet de représenter les fonctionnalités de mon application telles qu'elles seront perçues par chaque utilisateur.

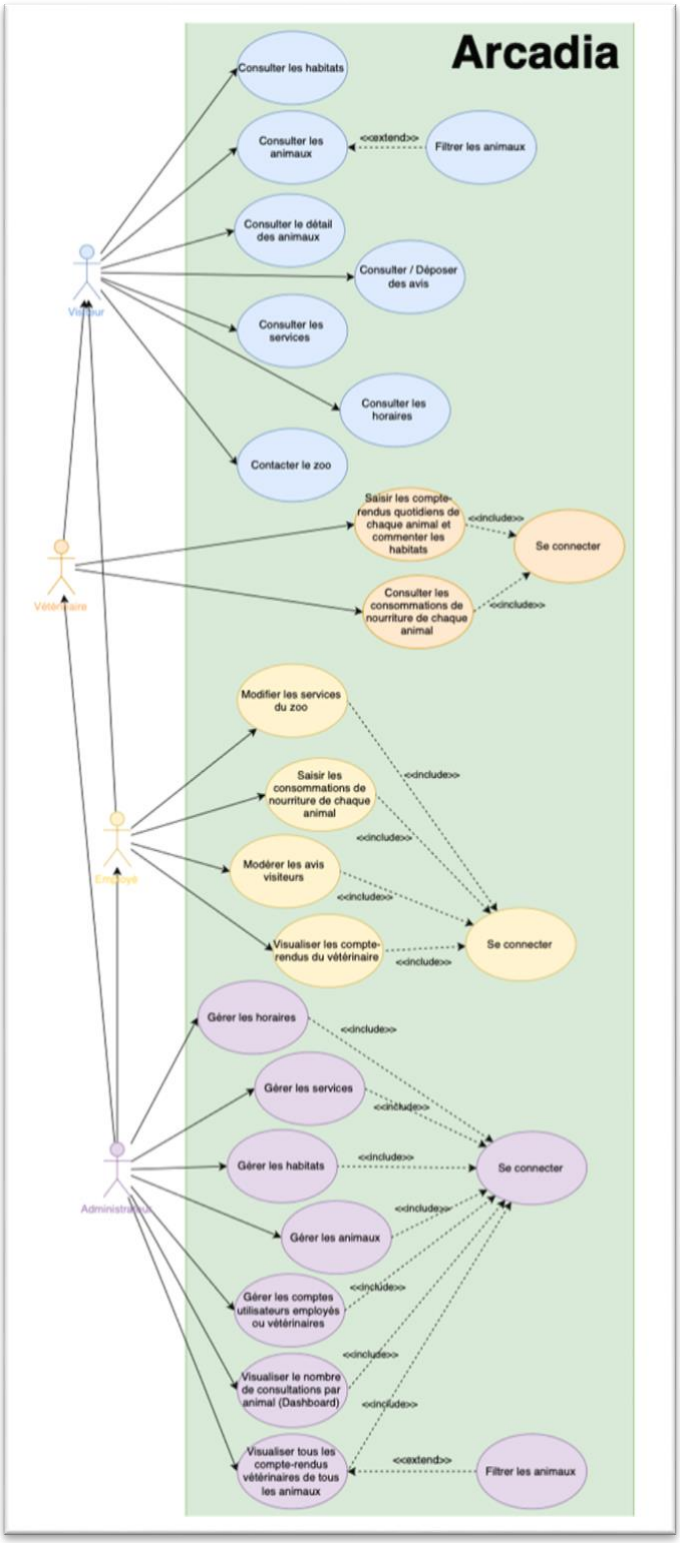

Figure 8 - Diagramme de cas d'utilisation

V - Diagrammes de séquences

Le diagramme de séquence est utilisé pour détailler chaque cas d'utilisation listé dans le diagramme précédent. Il permet d'illustrer la manière dont les objets interagissent les uns avec les autres.

a) Connexion d'un utilisateur

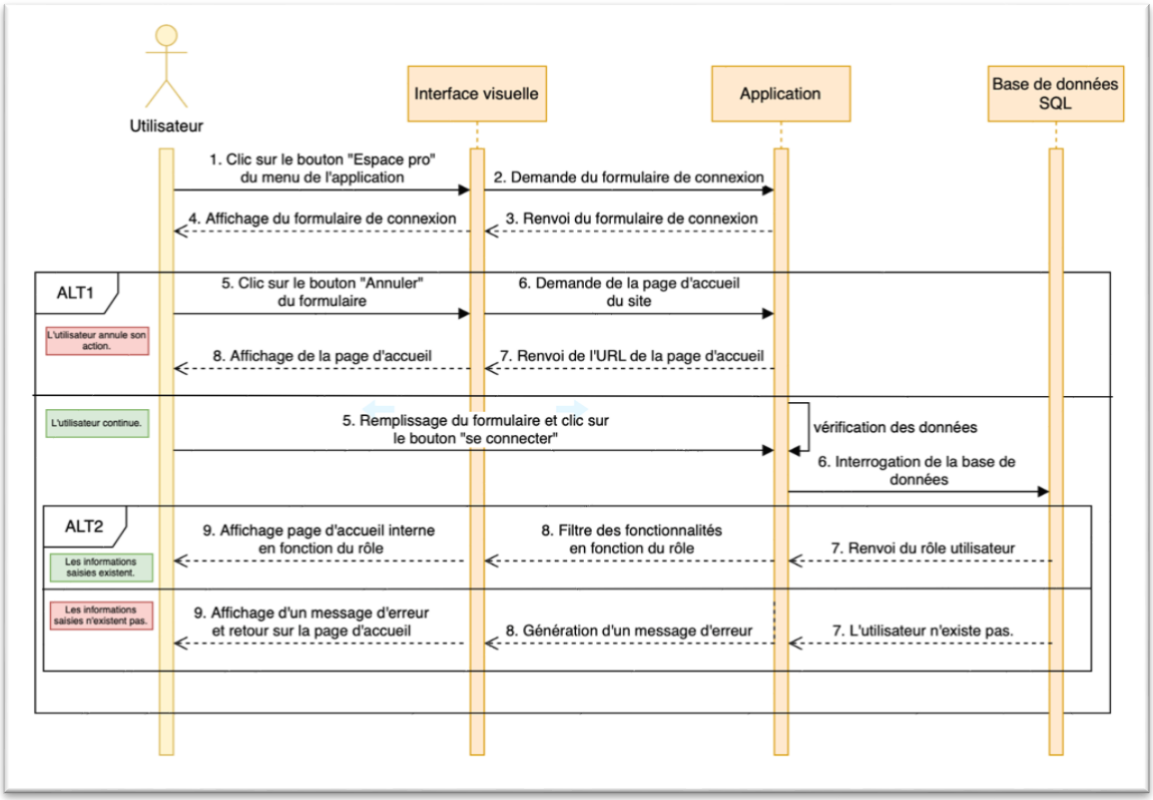

Figure 9 - Diagramme de séquence - Connexion d'un utilisateur

b) Consulter les animaux

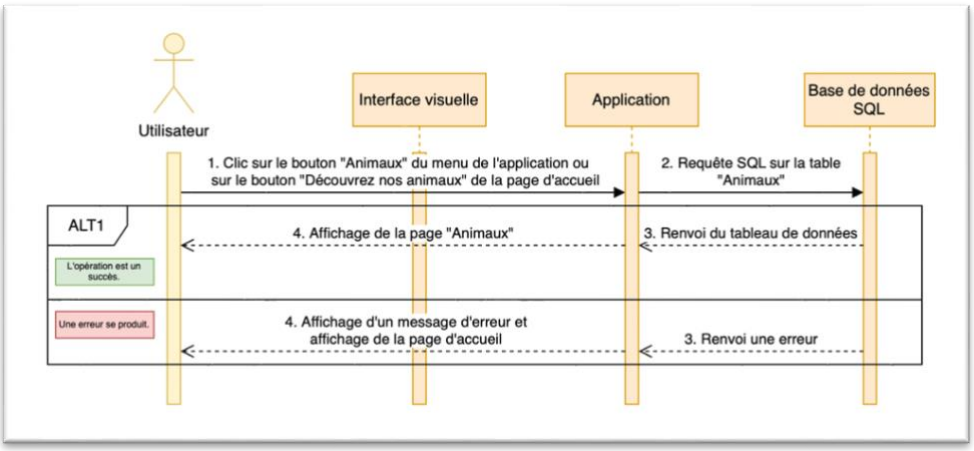

Figure 10 - Diagramme de séquence - Consulter les animaux

c) Déposer un avis

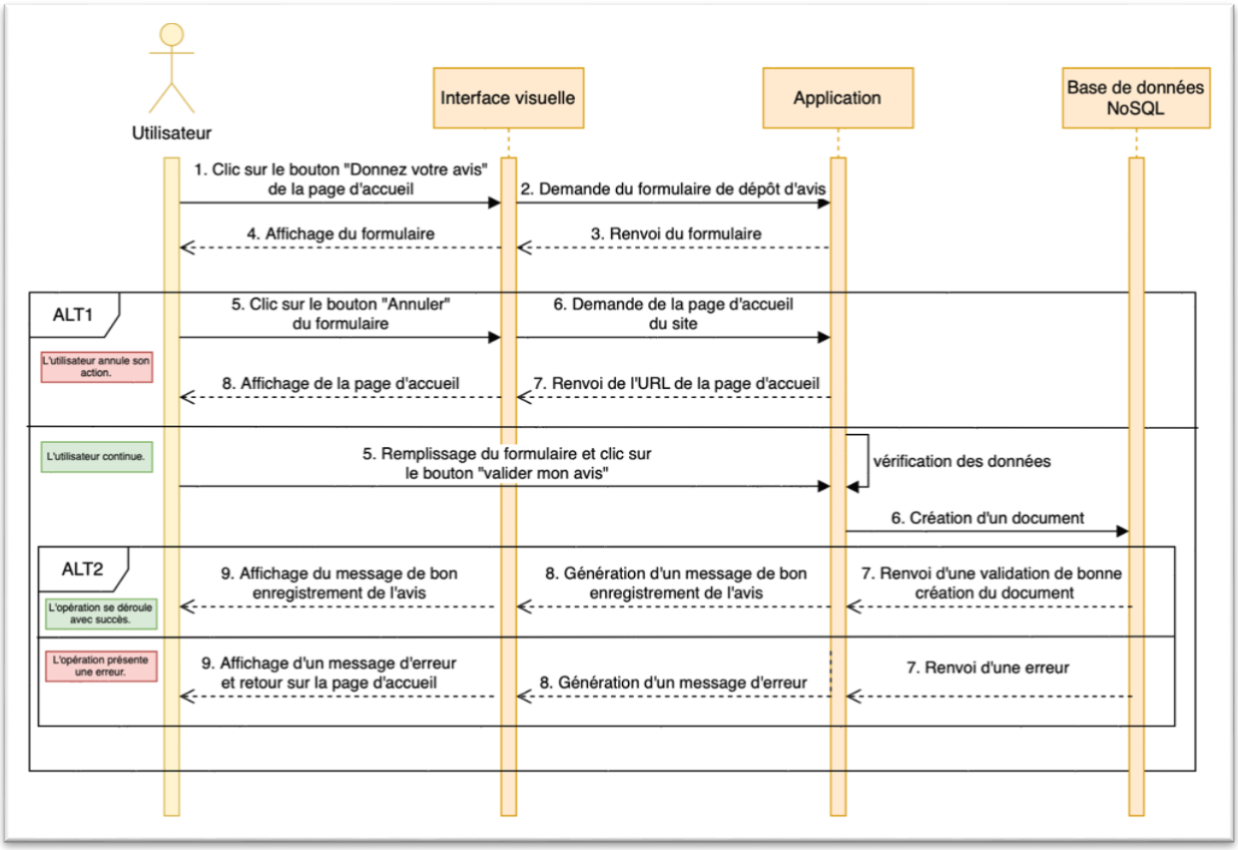

Figure 11 - Diagramme de séquence - Déposer un avis

VI - Documentation du déploiement

Au préalable de mon déploiement, je me suis assurée que mon projet est bien initialisé dans Git et que ma branche main de Git ne présente pas d'erreurs. C'est cette branche main que je vais déployer.

Afin de déployer mon application, je choisis d'utiliser la plateforme Heroku. Cette plateforme est simple d'utilisation et gratuite (avec le GitHub Student Developer Pack).

Voici les différentes étapes que j'ai réalisées afin de déployer mon application :

- Installation de Heroku : création d'un compte client puis installation d'Heroku CLI via l'invite de commandes mac : « brew tap heroku/brew && brew install heroku »
- Authentification sur la plateforme Heroku via la commande « heroku login » depuis mon invite de commandes
- Ouverture du projet ECF_Arcadia dans mon IDE PHP Storm

- Ouverture d'un terminal en vérifiant que nous sommes bien à la racine du projet
- Création d'une application (ou dyno) sur Heroku via la commande « heroku create » dans le terminal ouvert
- Déploiement du projet via la commande « git push heroku main » dans le terminal ouvert
- Modification du nom de l'URL sur le site de Heroku, dans les settings de l'application. Cela permet également de vérifier si le nom est disponible ou pas
- Modification du fichier caché config dans le dossier .git de mon projet ECF_Arcadia.