Réalisé par : BUCHELET Loris

Documentation technique (Arcadia)

Table des matières

1. Bref rappel du projet

Pour rappel, le projet Arcadia consiste en la réalisation d'un site web responsive pour le zoo Arcadia. Cette application requiert un design rendant compte de l'aspect écologique. De plus, l'application sert à la fois aux visiteurs pour une visée informative en amont d'une éventuelle visite des lieux mais aussi aux employés du zoo pour qui l'interface permettra, après connexion, la saisie des informations sur les animaux, les services et autres outils indispensables au bon fonctionnement du zoo.

La documentation suivante s'adresse aux développeurs afin d'accéder à une meilleure compréhension des outils utilisés pour développer l'application mais également comprendre les relations mises en place pour optimiser cette dernière.

2. Réflexions préliminaires

Dans cette dynamique, le projet a été pensé en amont tant sur le plan back-end que sur le plan front-end. En prenant en compte la complexité du projet, il est apparu clair que les langages incontournables pour la bonne réalisation de celui-ci sont :

- HTML et CSS pour le front-end

- JavaScript pour le front-end dans la mesure où celui-ci apporte un caractère dynamique au site.

- PHP pour le Back-end

- MySql pour la base de données relationnelle

Une fois les maquettes réalisées (voir la charte graphique), un premier jet a été effectué en HTML et CSS. Cette première étape a permis de mieux visualiser ce que rendaient visuellement les mock-ups une fois codés et adaptés au navigateur web. C'est d'ailleurs à cette occasion que Bootstrap a été ajouté pour mieux gérer le *responsive* (c'est-à-dire la faculté de l'application à mieux s'adapter à tout type d'écran).

L'étape suivante a consisté à intégrer peu à peu les bases de données. Étant donné que la plus grosse partie du site repose sur le modèle relationnel, la réalisation du diagramme de classe a pu apporter une réflexion quant à la définition des différentes tables et leurs interactions (voir la partie 4). Grâce à MAMP, la construction de la base de données 'arcadia' a pu se faire via *PHPMyAdmin*. A partir de ce moment, le serveur local Apache a pris le relais pour tester pas à pas les nouvelles fonctionnalités qui allaient être implantées.

L'utilisation de PHP a permis :

- d'intégrer de nombreuses boucles au sein du code HTML afin d'éviter les redondances. Par exemple, si le fragment de code des cartes habitats tel qu'elles sont visibles sur la page d'accueil requiert trois fois la mode structure en front-end pur, celui-ci a pu être drastiquement réduit grâce à une boucle *foreach*. Cette étape est par ailleurs cruciale car sans la communication avec la base de données, aucune édition ne sera possible de la part de l'administrateur.

- de mettre en place un système de routage. Jusqu'à présent, l'application reposait sur un unique renvoi d'une page à l'autre. Or, il est primordial de créer des chemins afin de restreindre les liens

utilisés par les visiteurs mais aussi rendre l'expérience de l'utilisateur plus cohérente. Des réflexions autour de menu, de la présence des boutons et des pages requises ont ainsi été effectuées (voir notamment la partie 6 avec les présentation d'un diagramme de séquence).

- d'établir un modèle MVC. Le Modèle / Vue / Controlleur s'est établi comme un moyen clair de séparer les différentes parties et fonctionnalités du site. Trois dossiers (models, views, controller) ont ainsi été ajoutés à l'arborescence.

→ Le dossier models comprend toutes les classes et la communication avec les bases de données,

→ Le dossier controllers comprend le router mais aussi toutes les instructions de redirection aux pages principales ainsi que les vérifications de l'authentification des utilisateurs pour accéder aux fonctionnalités requérant un compte.

→ Le dossier views comprend les instructions HTML finales telles qu'elles apparaissent sur le navigateur des utilisateurs.

L'utilisation de MongoDB s'est enfin imposé pour utiliser une fonctionnalité de comptage des visiteurs des différents animaux. Il s'agit d'une des dernières fonctionnalités ajoutées.

Heroku a été choisi en amont afin de déployer l'application : démarche expliquée dans le dernier point.

3. Configuration de l'environnement de travail

3.1. Configuration des dépendances

a. PHP Composer (pour `mongodb/mongodb`)

Prérequis :
- PHP 7.1 ou supérieur
- Composer installé

Fichier de configuration `composer.json`:

```
json
{
        "require": {
        "mongodb/mongodb": "^1.18"
        "phpmailer/phpmailer": "^6.9"
        }
}
```

Instructions d'installation :

b. Node.js (pour `bootstrap` et `bootstrap-icons`)

Prérequis :
- Node.js et npm installés

Fichier de configuration package.json :

```
json
    {
      "name": "studi-arcadia",
      "version": "1.0.0",
      "lockfileVersion": 3,
      "requires": true,
      "dependencies": {
        "bootstrap": "^5.3.3",
        "bootstrap-icons": "^1.11.3"
      }
    }
```

Description :
- bootstrap : Framework CSS pour le développement front-end, version 5.3.3 ou supérieure.
- bootstrap-icons : Ensemble d'icônes compatibles avec Bootstrap, version 1.11.3 ou supérieure.

Instructions d'installation :

```
npm install
```

3.2. Paramètres de Configuration

a. Environnement de Développement

Variables d'environnement (.env) :

```
NODE_ENV=development
DATABASE_URL=mongodb://localhost:27017/studi-arcadia
```

Description :
- NODE_ENV : Définir l'environnement (development, production, etc.).
- DATABASE_URL : URL de connexion à la base de données MongoDB.

b. Bootstrap

- Fichier de configuration optionnel pour personnaliser Bootstrap (variables SASS/SCSS).

c. MongoDB

- Configuration de connexion (adresse, port, authentification, etc.).

3.3. Exécution des commandes

- **Composer**:

```
composer install
composer update
```

- **NPM**:

```
npm install
npm update
```

3.4. Gestion des Dépendances

- Mises à jour :
    - Composer :

```
composer update mongodb/mongodb
composer update phpmailer/phpmailer
```

    - NPM :

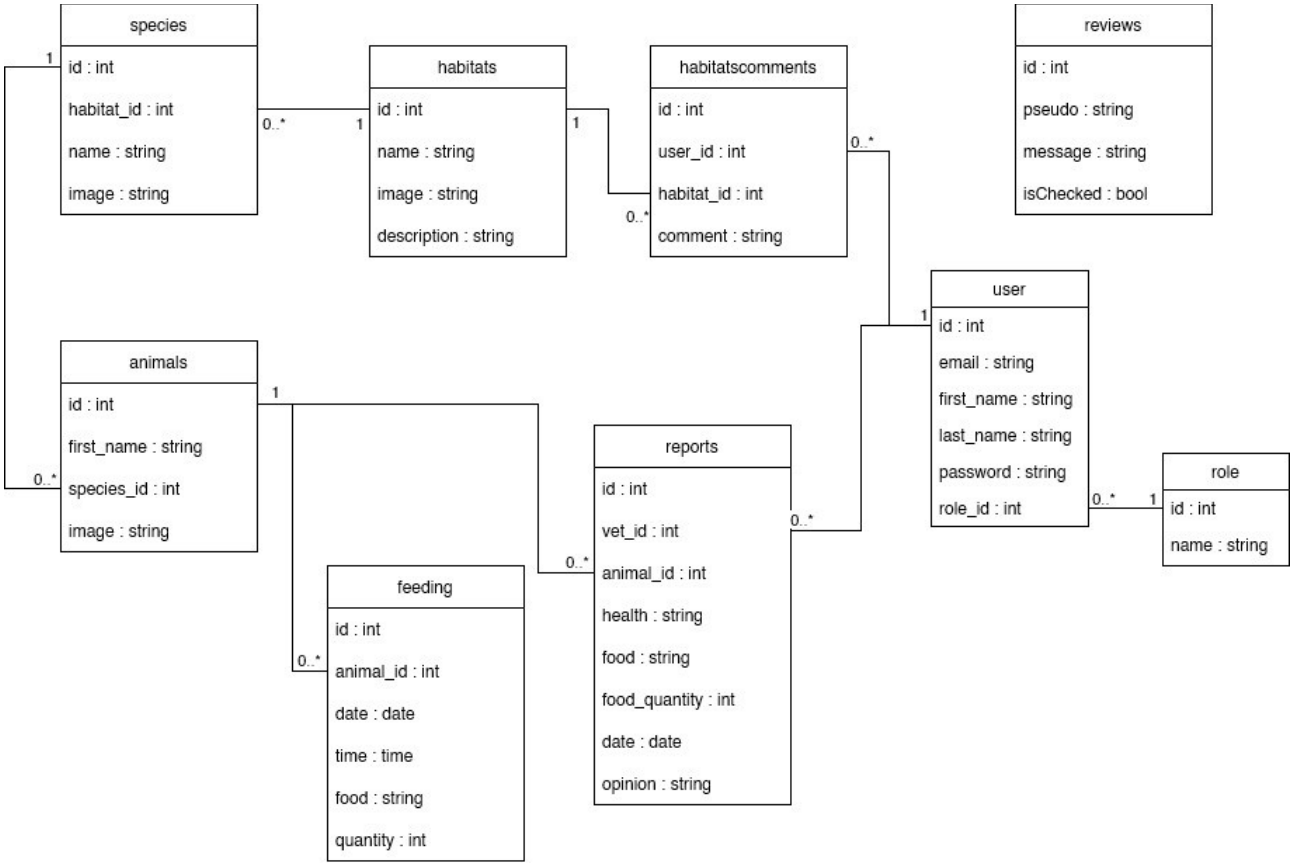
```
npm update bootstrap bootstrap-icons
```

4. Modèle conceptuel de données

Diagramme de classe

Le diagramme de classe qui suit a été réalisé en amont du développement de l'application. Il permet d'établir les connexions entre les différentes classes ainsi que leur composition. Sur ce diagramme, chaque classe correspond à une table de la base de données correspondante. Le modèle respecte la normalisation des BDD en prenant le soin d'établir pour chaque clé secondaire utilisée (nommée tableprimaire_id) une correspondance à la clé primaire de l'autre table.

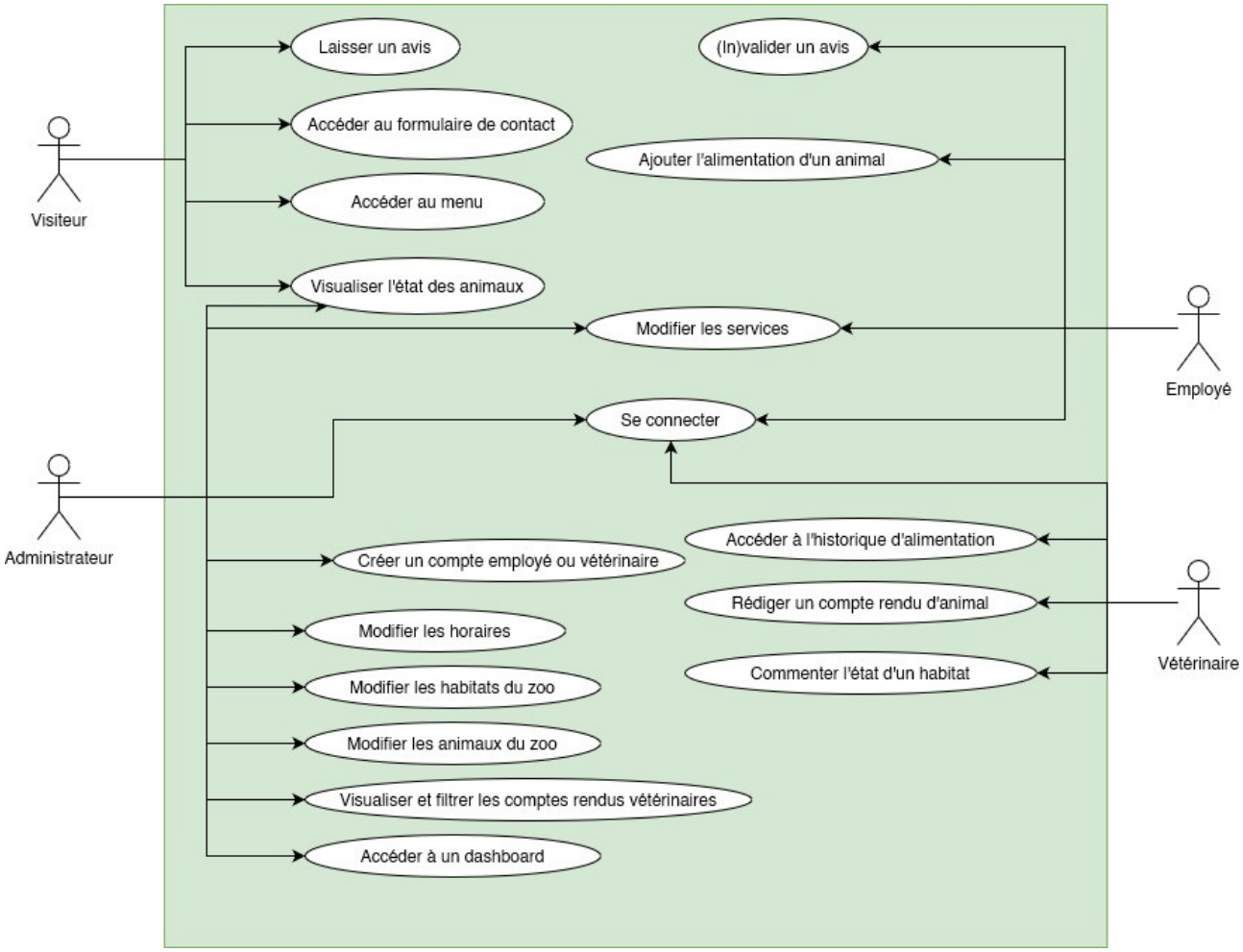
L'intégralité des connexions de cette table est marquée par une relation 1 à plusieurs. C'est ainsi que par exemple, la classe « habitatscomments » correspond à un seul habitat, mais qu'à l'inverse un habitat peut correspondre à plusieurs commentaires.



5. Diagramme d’utilisation

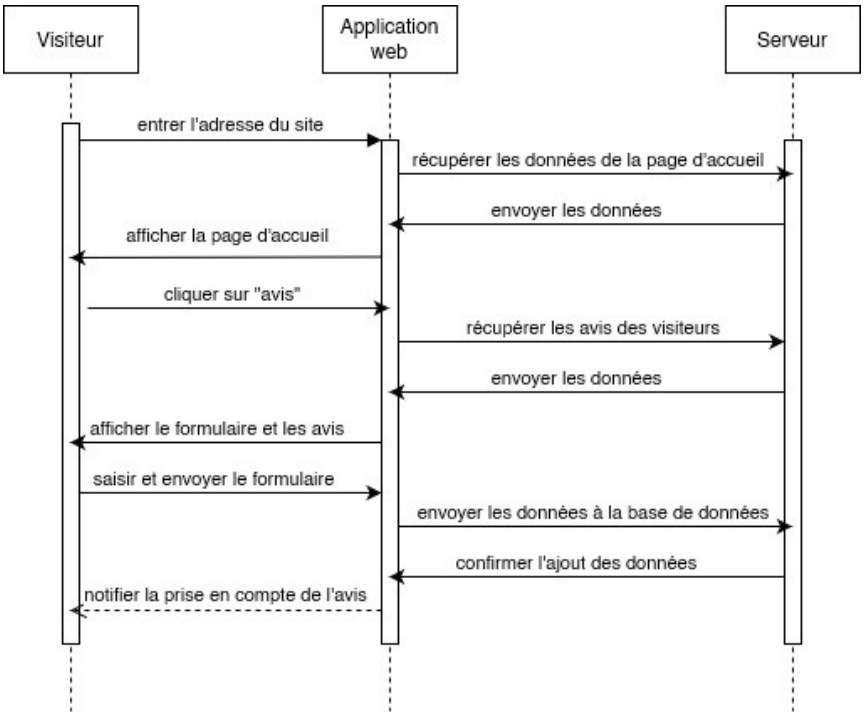
Le diagramme d’utilisation ci-contre rend compte des droits dont disposent chacun des utilisateurs de l’application web, à commencer par le simple visiteur non authentifié, en passant par les employés et vétérinaires authentifiés, pour finir avec l’administrateur qui disposent de la plupart des droits, une fois connecté.

L’élaboration de ce diagramme est survenue en amont du développement de l’application. Ce diagramme permet non seulement la synthèse des droits des utilisateurs tels qu’ils ont été émis par le client lors de la rédaction du cahier des charges, mais représente également un bon outil de repère afin de s’assurer du bon déroulement du développement et de la vérification des droits afin qu’aucun utilisateur ne puisse les outrepasser.



6. Diagramme de séquence

Le diagramme de séquence suivant montre le comportement de l’application et ses interactions entre le client (visiteur) et le serveur (accès à la base de données). L’exemple qui suit illustre la fonctionnalité d’ajout d’un avis sur le site du Zoo depuis l’arrivée du visiteur sur la page d’accueil jusqu’à la soumission de son avis à partir du formulaire.



7. Déploiement de l’application

1. **Préparation de l'application** :
    - Quelques pré-requis avant de commencer : la gestion des dépendances, la configuration des variables d'environnement, l’application fonctionne correctement en local.
2. **Configuration des bases de données** :
    - Pour MySQL : La base de données a été exportée depuis PhpMyAdmin.
    - Pour MongoDB : Un cluster avec une collection a été créée sur MongoDBAtlas afin de faciliter la communication avec Heroku.
3. **Création des bases de données sur Heroku** :
    - Connexion au compte Heroku sur le terminal grâce à la commande `heroku login`.
    - Création d’une nouvelle application Heroku via `heroku create`.
    - Ajout de l’add-on JAWS-DB : `heroku addons:create jawsdb`.
    - Ajout des variables d’environnement depuis l’espace en ligne heroku > Setting > Config Vars. Pour ce projet, les variables suivantes ont été ajoutées:
        - Pour les bases de données :
            - HEROKU_APP : true (permet d’indiquer au code le comportement à adopter face à une base de données via heroku)
            - JAWSDB_URL : (compléter avec l’url de la base de données importée sur JAWSDB)
            - MONGODB_URI : (compléter avec l’url mongoDB, ici atlas)
        - Pour la configuration SMTP (l’envoi d’e-mail) :
            - SMTP_HOST (ex. smtp.gmail.com)
            - SMTP_PORT (ex. par défaut 587)
            - SMTP_USER (l’adresse e-mail du zoo)
            - SMTP_PASS (le mot de passe d’accès à l’e-mail du zoo)
4. **Déploiement de l'application** :
    - Après l’ajout et le commit au référentiel Git, le déploiement sur Heroku se fait via la commande `git push heroku master`.
5. **Vérification du déploiement** :
    - Une fois le déploiement terminé, l’application peut être ouverte depuis la commande `heroku open`.