ARCADIA

ZOO DE BRETAGNE

Documentation Technique

Réalisé par Killian Marty, ECF Studi

Arcadia est un projet que j'ai eu l'opportunité de réaliser dans le cadre de ma formation Graduate Développeur Front-end Studi, dont le diplôme délivré à la fin est un Bac+2 reconnu par l'état au RNCP niveau 5.

Dès l'énoncé, les objectifs de l'évaluation sont clair :

- Réaliser un Site Web Responsive, réactif, avec une base de données relationnelle et non relationnelle
- Couvrir toutes les compétences requises par l'État pour l'obtention du diplôme en un seul projet
- Démontrer ses capacités à apprendre en pratiquant sur des deadlines courtes.

Pour ce faire, j'ai donc dû me mettre dans la peau d'un Développeur en situation d'Entreprise, avec pour objectif de le réaliser avant une date limite précise. La réalisation de ce projet exigeait rigueur et discipline dans mon travail, ainsi que beaucoup de réflexion concernant le style, la mise en branle des fonctionnalités de l'application, la représentation physique de ce qui m'était demandé, et la conception de ma base de données.

Ce document présent a donc pour but de :

- Vous présenter les technologies utilisées sur ce projet et comment je les ais implémentés
- Présenter la configuration de travail pour ce projet
- Expliquer mes démarches sur les décisions prises et leur importance pour le développement de l'application
- Comment j'ai déployé mon application
- La gestion de mon projet

1.Réflexions technologiques et configuration

1.1 Langages de programmation

À la genèse du projet, les questions fusent. “Vais-je réussir à tout finir à temps ? Qu’est-ce qu’une base de données ? Le projet me semble long et complexe à réaliser”

Ces questions, bien que naturellement très chargées, m’ont poussé directement à passer à la mise en pratique de ce que je savais déjà. À ce moment-là, je ne connaissais que HTML et CSS, et j'avais de bonnes bases en JavaScript, mais j’ai tout de suite compris que ce stack pour ce projet ne serait pas suffisant. La réalisation d’une base de données et la connexion avec un langage de programmation ne pourrait se faire avec un langage front-end. J’ai donc poussé ma réflexion pour regarder par quel langage commencer. et c’est sur PHP que s’est jeté mon dévolu, car bien qu’il soit le langage côté serveur le plus utilisé, il était extrêmement simple à implémenter, surtout pour son interaction avec html, là où JavaScript doit avoir recours à des librairies comme React afin d'interagir par des composants.

1.2 Logiciels de développement et gestion projet

Ensuite m’est venu l’idée de trouver un logiciel complet pour coder dans un environnement propre. En jetant un coup d'œil sur les IDE que proposait la formation via le GitHub Student Pack, je suis tombé sur la suite JetBrains, en licence gratuite le temps de ma formation. J’ai donc décidé de l’installer, et de tester un peu les fonctionnalités de l'environnement de Développement. J’ai très vite pris goût à ce logiciel très complet, qui m’aura beaucoup aidé pour déboguer lors de la réalisation de mon projet.

Concernant la Gestion de Projet, n’étant pas obligatoire pour une formation Graduate, je n’en ai pas réalisé, mais j’ai simplement découpé étapes par étapes les tâches de réalisation du site, sous forme de to-do-list sur Notion.

1.3 Configuration de l'environnement

La configuration de travail, aura été une tâche assez complexe à réaliser en tant que débutant, car beaucoup de paramètres étaient à prendre en compte :

- La version de Windows
- La version du langage installé
- La configuration Hardware de l'ordinateur sur lequel le projet doit être hébergé en local
- Les variables d'environnement de l'ordinateur

Paramètres qui m'ont finalement retardés sur ce projet, mais qui une fois installés m'ont permis de développer cette application de la manière la plus confortable et optimale possible. J'ai installé XAMPP, réputé pour sa qualité d'hébergement local pour PHP et SQL.
Enfin, pour le NOSQL, j'ai installé MongoDB et MongoSH, relativement simple d'utilisation, sa syntaxe étant similaire à celle d'un fichier JSON, ainsi que l'extension MongoDB pour php afin de pouvoir connecter php à la base de données non relationnelle.

Je me suis par la suite penché pour le déploiement sur Heroku, réputé pour s'exécuter rapidement pour des applications php sql.

Voici donc le Stack Technique d'Arcadia que j'ai décidé d'utiliser :

- FRONT : HTML, CSS, JS
- BACK : PHP avec utilisation de PDO
- SQL : MySQL - XAMPP
- NOSQL : MongoDB
- Déploiement : Heroku
- Librairies : PHP Mailer (SMTP)
- IDE : JetBrains - PHPStorm

2.Développement front-end

de US1 à US5

2.1 Site côté client

Comme dit plus haut, je possédais dors et déjà quelques bases en développement Front-end, chose qui m'aura fait gagner un temps considérable pour l'avancée de ce projet. J'ai donc pu prendre mon temps pour réaliser des maquettes très simples mais qui restent jolies et originales, en plusieurs résolutions responsive, comme présenté ci dessous :

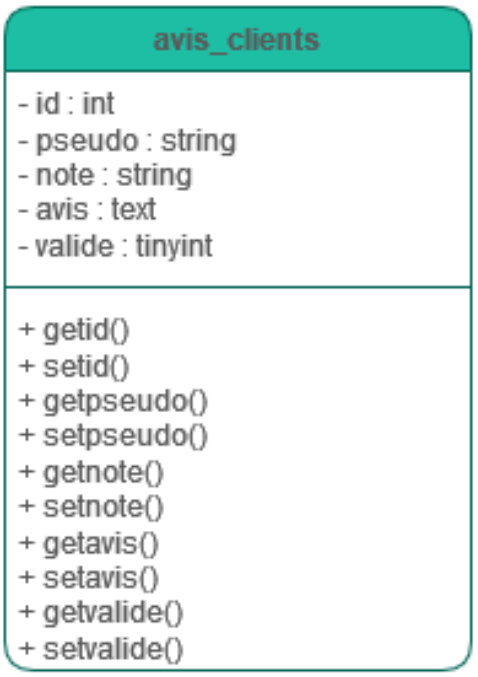
les différents Mockup et Wireframes sont présentés en format png et pdf dans Arcadia\projet\Maquettes

Ces maquettes dites responsives, s'adaptent parfaitement aux tailles d'écran de différents appareils, en passant d'un grand écran d'ordinateur de bureau, à celui d'un téléphone portable sans problème.
Lors de leur réalisation, j'ai fait en sorte de bien prendre en compte la présentation du zoo, avec des images, des avis en bas de page, un navbar qui change de position en fonction de la taille de l'écran, une vue globale des services et des animaux, en y ajoutant des images et des descriptions dès la page d'accueil, de sorte à ce que l'utilisation soit le plus simple possible : Dès qu'on arrive sur le site, on y voit tout ce que propose le Zoo, et c'est ce qui rend la visite de l'utilisateur lambda plus intuitive et rapide, pour comprendre directement ce que fait l'établissement.

Pour répondre à la demande de José sur les couleurs de l'écologie, j'ai pensé à faire du vert et du marron, couleurs faisant penser à des lieux naturels tels que la jungle, et pour la police d'écriture, j'ai décidé de mettre Roboto Mono, puisque c'est une police assez linéaire, avec quelques bords arrondis, des formes laissant penser à quelque chose de plus naturel et sauvage.

2.2 Système des avis

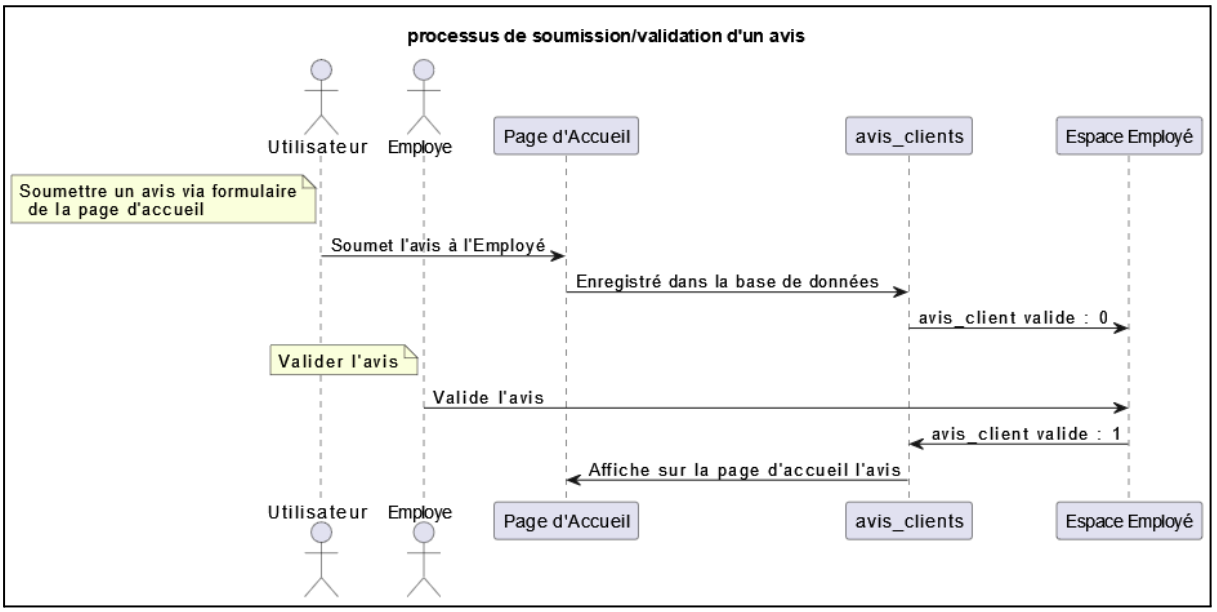
Concernant les avis, j'ai réalisé d'abord des maquettes de ce à quoi cela pourrait ressembler une fois soumis et validé sur la page d'accueil : J'ai opté pour une sorte de carte verte, avec le nom du visiteur, la note sur 5, ainsi que le commentaire que ce dernier souhaite laisser.
Afin de permettre que chaque avis puisse apparaître sur la page, j'ai dû donc créer une classe dans ma base de données SQL que j'ai nommé "avis_clients", comme présenté ci-dessous :



"avis_clients" contient les champs suivant :

- un identifiant par avis qui est un entier auto-incrémenté
- un pseudo en chaine de caractère
- la note sur 5 maximum en chaine de caractère
- l'avis écrit en texte
- La validation par l'employé qui est un nombre entier défini à 0 par défaut.

Le fonctionnement du système est simple, et pour appuyer mon propos, je vais utiliser un diagramme de séquence, décrivant les étapes essentielles de validation d'un avis en UML.



Ce diagramme décrit le processus de soumission/validation d'un avis de cette manière :

- L'Utilisateur réalise son avis via le formulaire, et une fois tous les champs rempli, le soumet à l'employé
- Les données saisies sont enregistrées dans la classe "avis_clients" avec pour occurrence 'valide' la valeur 0, ce qui signifie que l'avis n'est pas validé, et qu'il ne s'affiche pas sur la page, tant qu'il n'aura pas été incrémenté de 1 maximum.
- L'Employé se rend dans son espace, et voit qu'un nouvel avis a été soumis. Il décide donc de le valider, et automatiquement, l'occurrence 'valide' est incrémenté et devient 1.
- L'avis une fois validé disparaît de l'espace Employé,
- L'avis apparaît finalement en bas de page d'accueil.

```php
<?php
// Connexion à la base de données
$bdd = new PDO('mysql:host=localhost;dbname=espace_admin;', 'root', '');

if(isset($_POST['valider'])) {
    $id_avis = $_POST['id_avis'];

    // Mise à jour de la colonne valide dans la base de données
    $query = $bdd->prepare('UPDATE avis_clients SET valide = 1 WHERE id = ?');
    $query->execute([$id_avis]);

    echo "L'avis a été validé avec succès.";
?>
    <form action="espace-employe.php" method="post">
        <input type="submit" name="retour" value="retour espace employé">
    </form>
    <?php
} else {
    echo "L'avis a été rejeté.";
}
?>
```

Pour permettre le passage de 0 à 1 sur l'occurrence ‘valide’, j’ai dû faire usage d’une requête préparée d’insertion SQL pour permettre l’ajout d’un chiffre à chaque validation, comme présentée dans l’image ci-dessus.

3.Développement back-end

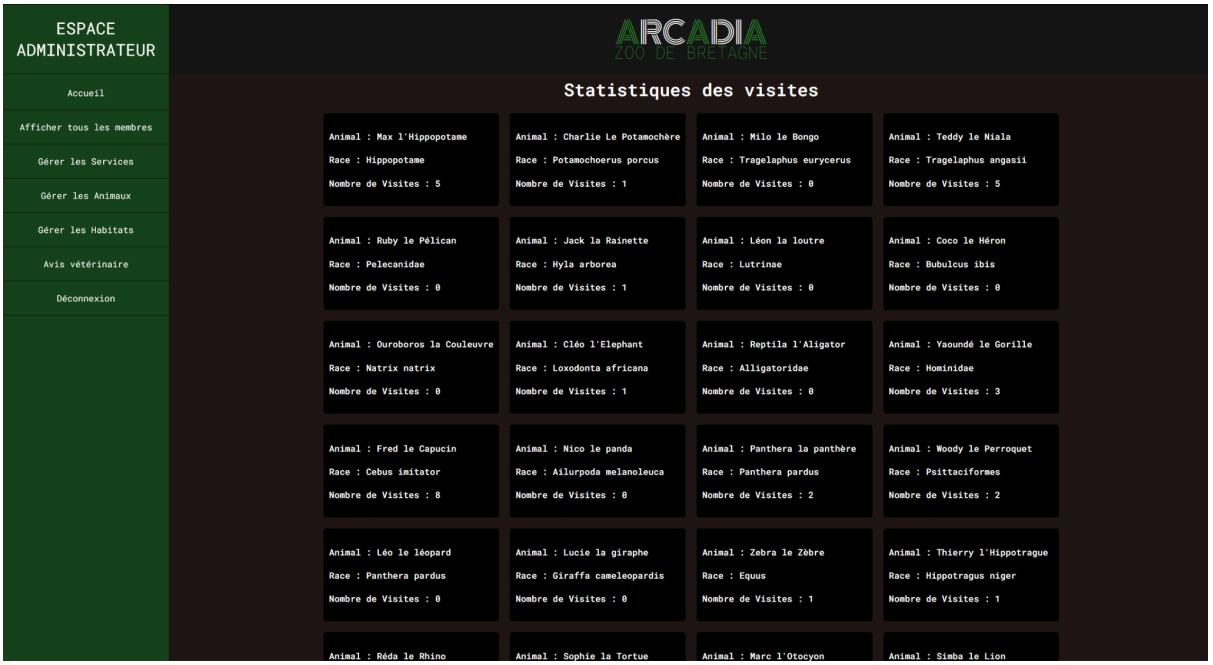
de US6 à US9

3.1 Site côté serveur

Afin de permettre la conception de cette partie du site, Je me suis lancé dans la création des pages de ces espaces avec HTML et CSS qui présente les différentes fonctionnalités, des CRUD, et des consultations de données, avec barre de recherche.

J’ai opté cette fois pour un style différent du site côté client, de sorte à ce qu’on remarque vraiment que la partie auquel on accède est restreinte, et non ouvert à tous.

Voici pour appuyer mon propos, une idée de ce à quoi peut ressembler un espace pour différents type de compte sur le site (en l'occurrence, il s'agit là de l'Espace Administrateur)



J'ai utilisé les mêmes palettes de couleur, la même police d'écriture, juste le style est les mises en pages sont différentes.

3.2 Base de données et classes

Par la suite, j'ai pu me consacrer à l'apprentissage de langages back-end comme PHP via les PDO, afin de connecter le langage avec MYSQL comme base de données, et pouvoir effectuer des requêtes d'insertion, de modification ou de suppression.

J'ai donc commencé simplement par réaliser une base de données SQL sur PHPMyAdmin, que j'ai nommée "espace_admin", et qui allait contenir toutes les données que les différents types de compte pourraient utiliser. J'ai donc réalisé un diagramme de classes, qui permet de classes, avec toutes les données qui allaient me servir pour la création de ces espaces.

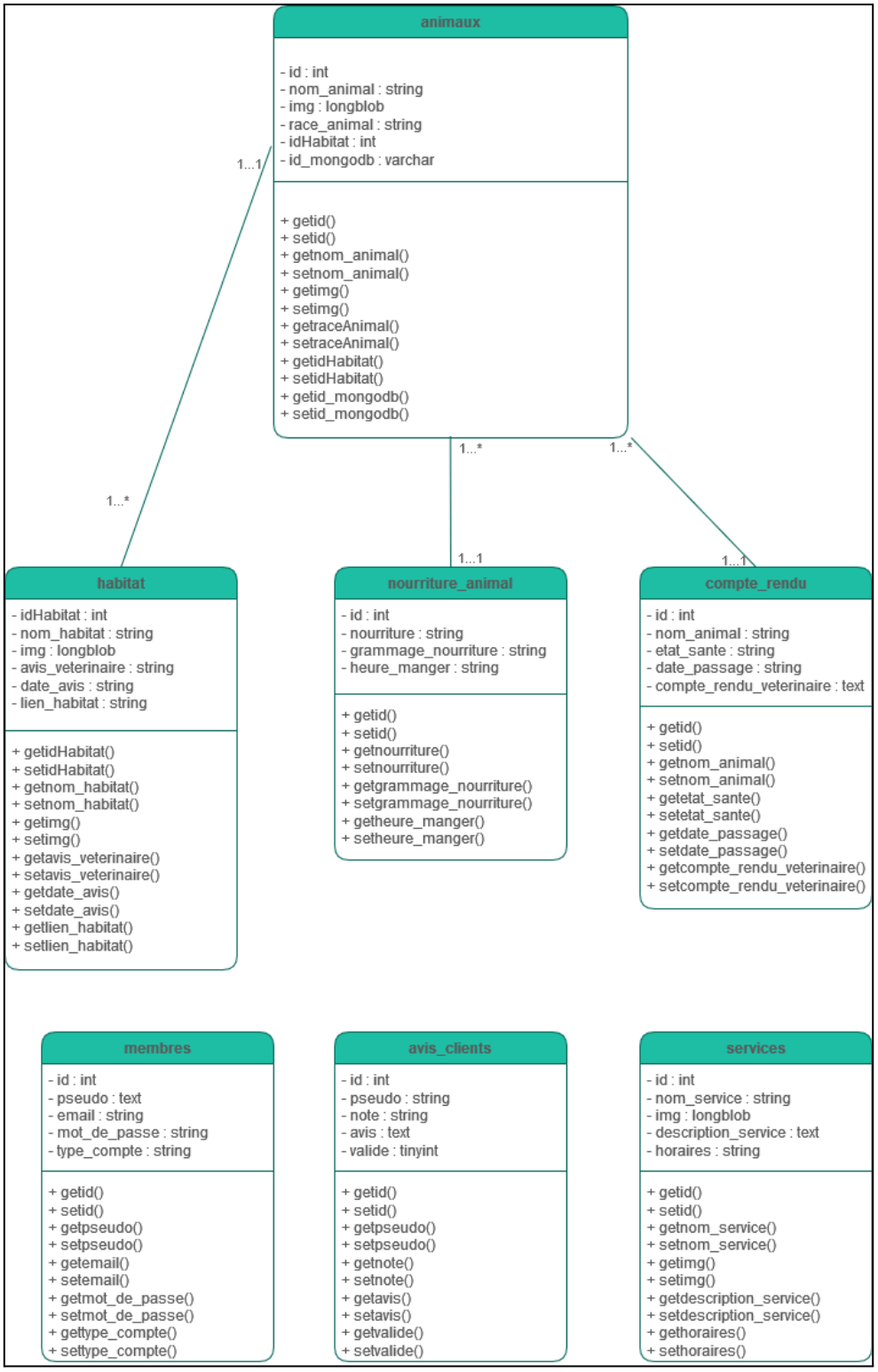
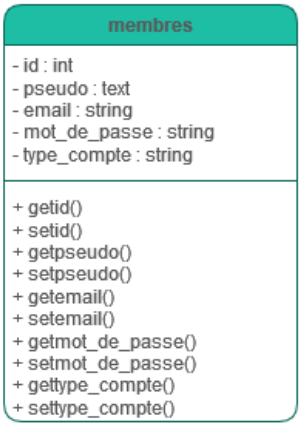


Diagramme de classes représentant la base de données relationnelle “espace_admin”

Dans cette base de données, J'ai créé la table "membres" qui contient tous les comptes peu importe le type. La structure de cette table est la suivante :

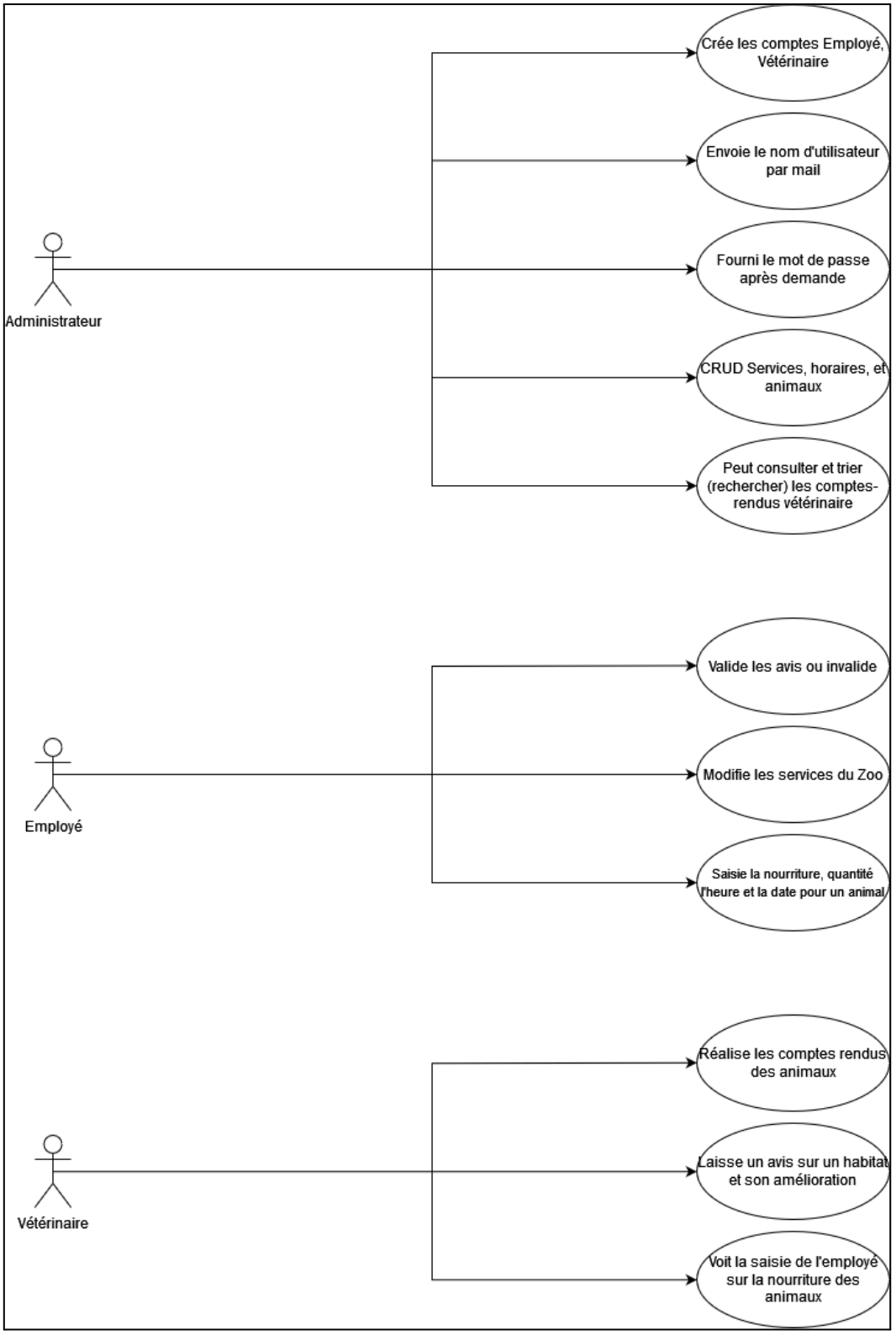
- L'identifiant du compte, en nombre entier
- Le pseudo en texte
- L'email, en chaîne de caractère
- Le mot de passe en chaîne de caractère
- Le type de compte, en chaîne de caractère



J'ai inséré directement 3 types de comptes pour permettre la connexion dans les différents espaces du site côté serveur, dont les informations sont disponibles dans le ***Manuel d'Utilisation.***

Les types de comptes dans ce projet ont une importance cruciale dans la mise en place des animaux, habitats, services, et avis, commentaires et données associées à ces éléments, car les rôles ne sont jamais les mêmes.

En suivant l'énoncé du sujet lors de la réalisation de mon Application, j'ai réalisé pour mieux me repérer dans les cas d'utilisation, un diagramme, qui explique clairement qui effectue quels types d'opérations sur leurs espaces.



Comme nous pouvons le constater dans le diagramme de cas d'utilisation ci-dessus, Il existe 3 types de compte ;

- L'Administrateur
- Le Vétérinaire
- L'Employé

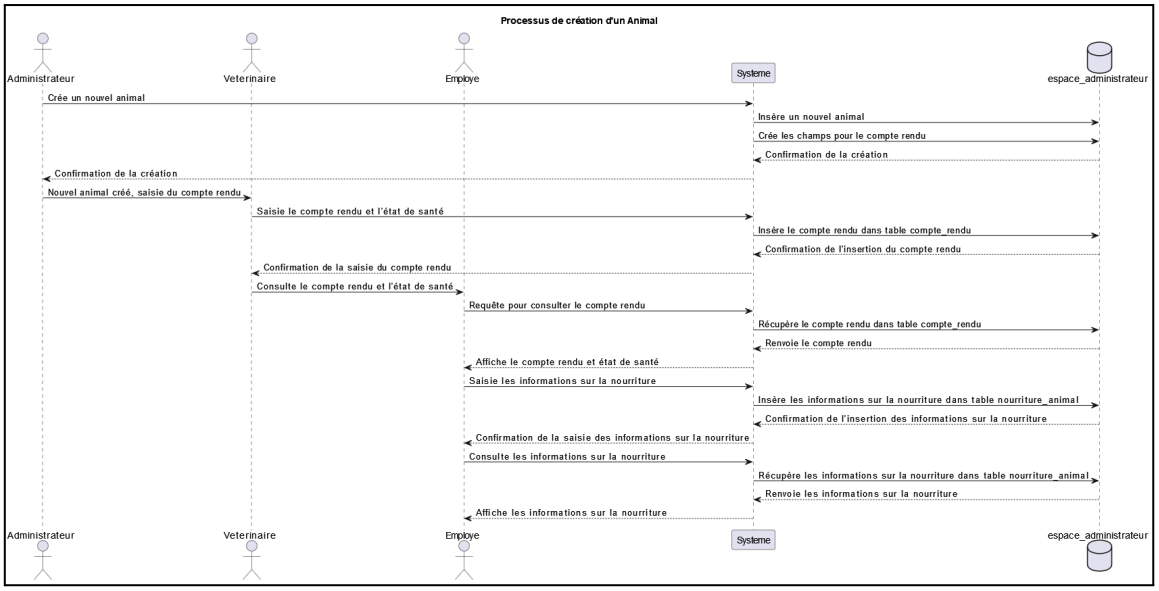
Le diagramme décrit les rôles suivant :

1. ***L'Administrateur :***
    - Crée les comptes, mais il ne peut y avoir qu'un seul Administrateur, alors il ne peut créer que des comptes Vétérinaires et Employés via son espace.
    - Lorsqu'un nouvel Employé ou Vétérinaire désire obtenir un compte, il doit se tourner vers l'Administrateur via le formulaire de mail (détaillé plus bas en US 10) pour obtenir son nom d'utilisateur et mot de passe.
    - Crée, consulte, modifie, ou supprime les Services, les Habitats et les Animaux du zoo s'il le souhaite
    - Consulte par la recherche des comptes rendus du Vétérinaire.

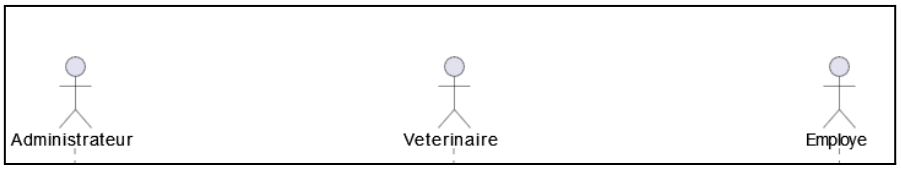
2. ***L'Employé :***
    - Valide un avis ou non d'un Utilisateur (dans le cas où ce dernier l'invalide, il est supprimé pour toujours)
    - Modifie les Services du zoo pour y incorporer une ou des images, une description,
    - Saisi la quantité de nourriture, la nourriture, la date pour un animal, tout comme le Vétérinaire

3. ***Le Vétérinaire :***
    - Réalise des comptes rendus pour chaque animal lorsqu'il est de passage
    - Laisse un avis sur l'état d'un habitat, son amélioration et sa qualité
    - Peut consulter les données de saisie de nourriture par l'Employé, et peut aussi les modifier.

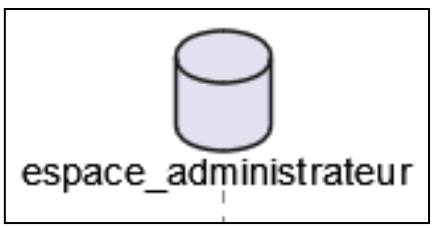
Les parcours étant différents, ils restent cohérents dans le but de créer des éléments sur le site, et de pouvoir les remplir chacun de son côté en fonction des privilèges accordés. C'est le cas pour la création d'un Animal, comme le démontre le Diagramme de Séquence dans la page suivante de cette documentation.

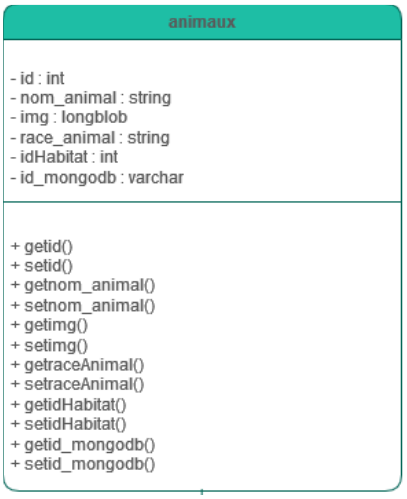


Ce diagramme de séquence décrit le ***"processus de création d'un Animal"*** (présent dans Arcadia\projet\fonctionnalités), du début à la fin, lorsque tous les acteurs ont réalisé leur tache, et que l'animal est affiché avec toutes ses informations sur le site côté client.

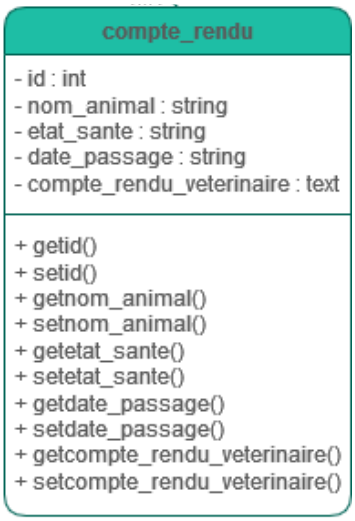


Voici nos 3 comptes créés sur le site. Pour procéder à la création d'un Animal, l'Administrateur va dans son espace saisir les données de l'animal qu'il souhaite créer. Une fois cela fait, il envoie toutes les données saisies dans la base de données du site, dans la table 'animaux' et permet la création d'un nouveau champ à saisir pour le compte rendu du vétérinaire. Il peut directement consulter ses données après création.

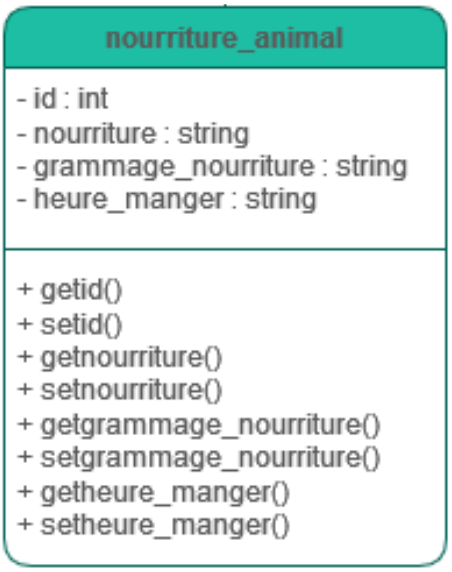




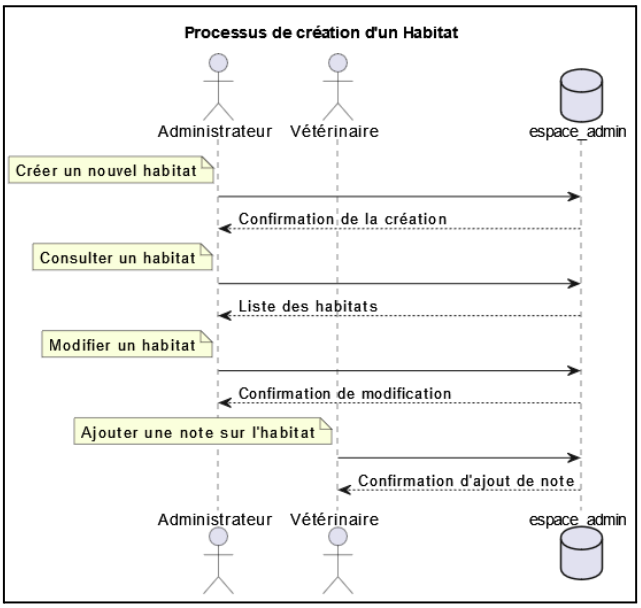
Par la suite, le vétérinaire va devoir saisir son nouveau compte rendu pour ce nouvel animal fraîchement arrivé, et les champs saisis vont s'insérer dans la base de données, dans la table 'compte_rendu'. Il peut les consulter à tout moment.



Finalement, l'Employé, va saisir la nourriture, la quantité, la date et heure pour ce nouvel animal, et une fois ces informations validées insérées dans la table 'nourriture_animal', elles seront consultables et sur les espaces, et sur la fiche animale du site.



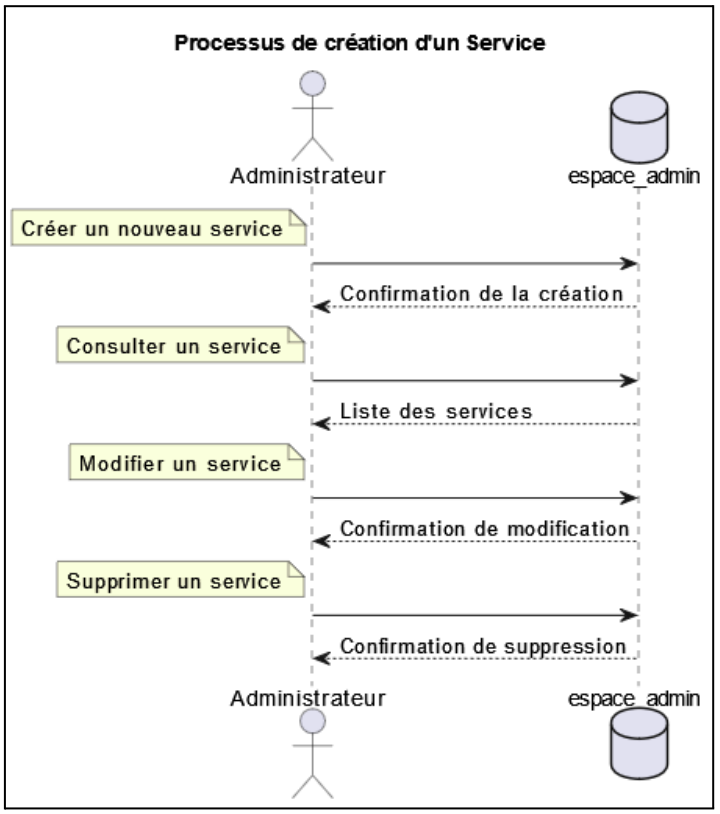
Pour un Habitat, il s'agit du même principe, à la différence que les autres acteurs ne peuvent pas forcément agir pour les compléter :



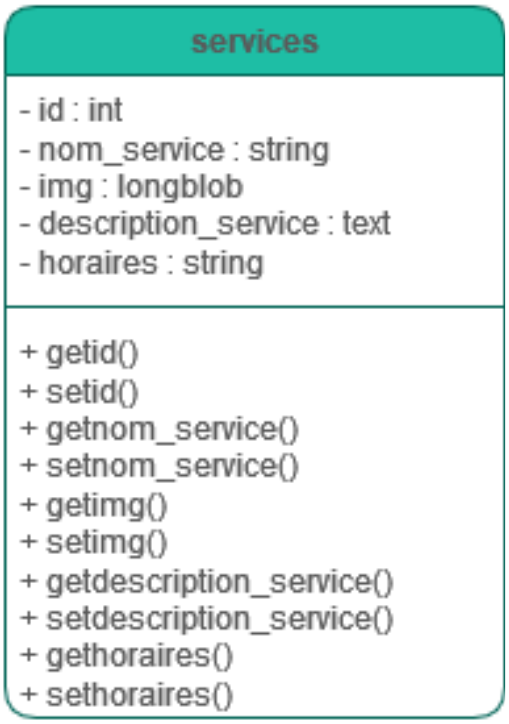
- L'Administrateur crée le nouvel habitat, mais peu aussi le modifier, le supprimer (CRUD)
- Il peut ensuite consulter à tout moment les données saisies
- Le Vétérinaire modifie et consulte l'habitat
- Le Vétérinaire peut ajouter un avis sur l'habitat
  les données d'un habitat sont enregistrées dans la table 'Habitat'.

les données d'un service sont enregistrés dans la table 'Habitat', ci-dessous.

La création d'un service et le CRUD, elle, n'est réalisée que par l'Administrateur, comme le démontre ce diagramme de séquence ci-dessous.



Les données d'un service sont enregistrés dans la table 'services'.



4.Connexion et déconnexion

Pour US9

4.1 Connexion

La seule difficulté dans un système de connexion est la récupération correcte des informations saisies par l'utilisateur pour l'exécuter. Pour ce faire, il m'a d'abord fallu afin de récupérer les données et de les comparer, connecter PHP avec ma base de données SQL, via les PDO, tout comme toutes les pages de mon site qui interagissent avec la base de données, comme présenté dans l'image ci-dessous :

```php
<?php
// démarre une session de connexion
session_start();
// connexion à la base de données
$bdd = new PDO('mysql:host=localhost;dbname=espace_admin;', 'root', '');

```

Par la suite, j'ai récupéré via les variables POST les données saisies, et les ai fait comparer avec la base de données via une requête préparée SQL. Dans le cas où toutes les informations matchent, il faudra ensuite vérifier si le type de compte est Administrateur, Vétérinaire ou Employé.

```php
$recupUser = $bdd->prepare('SELECT * FROM membres WHERE pseudo = ? AND mot_de_passe = ? AND type_compte = ?');
        $recupUser->execute(array($pseudo, $mot_de_passe, $type_compte));
```

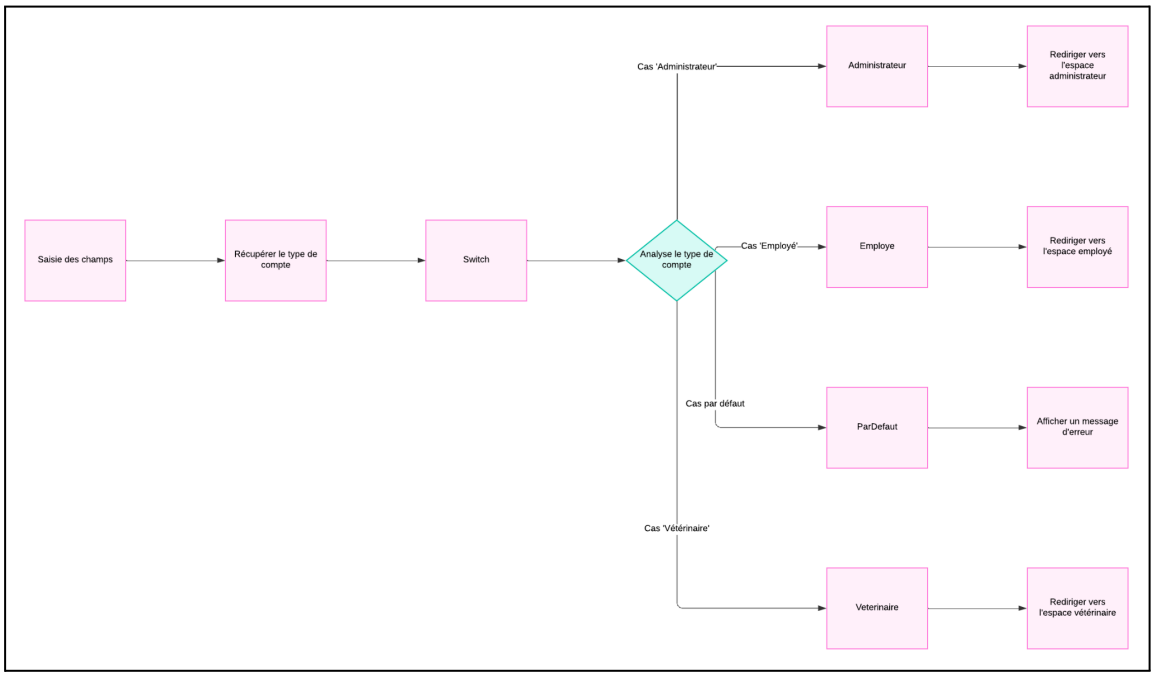
Cette vérification est cruciale, parce qu'elle va nous permettre d'être redirigé sur l'espace qui concerne le compte en fonction de son type, sans cela, tous les comptes vont se rediriger vers l'Espace Administrateur.

.

Afin d'implémenter cette fonctionnalité, j'ai voulu une première fois utiliser des conditions if elseif else, mais le code étant beaucoup trop étiré et par soucis de lisibilité, j'ai finalement opté pour un switch case, très adapté dans ce genre de situation.

```php
// Rediriger en fonction du type de compte
            switch ($type_compte) {
                case 'Administrateur':
                    header('Location: ../../backend/espace-admin.php');
                    exit();
                    break;
                case 'Vétérinaire':
                    header('Location: ../../backend/espace-veterinaire.php');
                    exit();
                    break;
                case 'Employé':
                    header('Location: ../../backend/espace-employe.php');
                    exit();
                    break;
                default:
                    echo "<script>alert('Type de compte non reconnu.');</script>";
            }
        } else {
            echo "<script>alert('Votre mot de passe ou pseudo est incorrect.');</script>";
        }
```

Afin d'implémenter cette fonctionnalité, j'ai voulu une première fois utiliser des conditions if elseif else, mais le code étant beaucoup trop étiré et par soucis de lisibilité, j'ai finalement opté pour un switch case, très adapté dans ce genre de situation.



Pour illustrer mon propos, j'ai réalisé ce diagramme de flux, que vous pouvez retrouver dans le fichier ***Arcadia\projet\fonctionnalités*** du projet et qui explique de manière plus claire la condition de connexion en fonction des types de comptes :

- L'Utilisateur saisi les champs de connexion et soumet le formulaire
- Le formulaire via la méthode POST va récupérer le type de compte
- Il va finalement le comparer, et dans le cas où il matche avec un des 3 disponibles, il va rediriger l'utilisateur vers la page associée.
- Dans le cas par défaut, il envoie un message d'erreur.

Cette manière de procéder est particulièrement utile lors de cas d'utilisation nombreux, et m'aura appris à me servir des conditions de manière plus optimisée en programmation.

4.2 Déconnexion

Afin que lorsqu'un Utilisateur souhaite se déconnecter, il puisse le faire sans problème, j'ai simplement créé un bouton de déconnexion dans les différents espaces du site, puis ait importé par la soumission le script PHP suivant :

```php
<?php
// Déconnecte la session
    session_start();
    $_SESSION = array();
    session_destroy();
    header('Location: ../frontend/Client/connexion.php');
?>
```

Après chaque fin de session, l'Utilisateur est redirigé vers la page de connexion du site.

5.PHPMailer et SMTP

Pour US10

5.1 Contact

La conception de ce formulaire de contact a été relativement rapide, car j'ai dans un premier temps réalisé le formulaire avec HTML et CSS, puis dans un second temps, je suis passé au côté back-end pour réaliser l'envoie de messages via SMTP. Au départ, j'avais opté pour utiliser la fonction mail() de php afin d'envoyer le message. cependant cela exigeait d'être hébergé sur un serveur local SMTP comme Mailjet, et prendrait plus de temps que prévu à configurer. En faisant mes recherches, notamment sur les forums tels que StackOverflow, j'ai pu apercevoir une multitude de manières de procéder à la mise en place d'envoi de mail, et celle qui semblait la plus fiable était la librairie PHPMailer, qui permet de se connecter automatiquement à un serveur SMTP, de sorte à ce que l'on ait seulement à réaliser la structure du mail à envoyer.

```php
<?php
// Importation de la librairie PHPMailer
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

// fichiers requis
require '../libraries/phpmailer/src/Exception.php';
require '../libraries/phpmailer/src/PHPMailer.php';
require '../libraries/phpmailer/src/SMTP.php';
```

J'ai donc commencé par importer la librairie dans mon script php qui me servirait à fournir l'envoi de mail via SMTP après avoir soumis les champs saisis sur la page de contact, puis j'ai ensuite placé une condition qui stipule que si j'ai cliqué sur le bouton soumettre, je puisse connecter PHP au SMTP via PHPMailer.

```php
if (isset($_POST["submit"])) {
    $mail = new PHPMailer(true);
```

Pour détecter les erreurs, j'ai mis en place la condition try-catch, qui envoie un message d'erreur spécifié si le mail ne s'est pas envoyé correctement. Vous retrouverez les détails concernant le formulaire de contact dans le ***Manuel d'Utilisation***.

6. NoSQL avec MongoDB

Pour US11

6.1 Statistique sur la consultation des habitats

Cette partie fut la dernière que j'eut à réaliser dans ce projet, puisqu'elle exigeait que j'apprenne à utiliser MongoDB avec PHP, et user de Javascript pour réaliser une API.
J'ai donc créé une base de données non-relationnelle, sur base de la syntaxe des objets JSON, comme sur la photo d'en dessous.

```
[{
  "_id": {
    "$oid": "6602c629fb0a7ea799d14a13"
  },
  "nom_animal": "Max l'Hippopotame",
  "race_animal": "Hippopotame",
  "nombre_visites": 2
},
{
  "_id": {
    "$oid": "6602f277fb0a7ea799d14a14"
  },
  "nom_animal": "Charlie Le Potamochère",
  "race_animal": "Potamochoerus porcus",
  "nombre_visites": 0
},
```

Pour donner un peu de descriptif à l'animal, j'ai rajouté la race et le nom de l'animal pour le reconnaître, puis un nombre de visites établi par défaut à 0, et que l'on incrémente à chaque visualisation de la fiche sur le site côté client.
Pour que SQL reconnaisse les données de la base de données NOSQL, il aura fallu que dans la table animaux, je rajoute l'id de MongoDB de chaque animal de sorte à

ce que lorsque je vais réaliser une action pour MongoDB, il se serve des données de SQL pour s'exécuter, comme par exemple pour la récupération de nom, ou pour la suppression d'un animal.
Lorsque l'on clique sur l'image d'un animal dans la page des habitats, on réalise par derrière une requête AJAX comme ci-dessous :

```
if (isDetailsVisible) {
            // Envoie une requête AJAX au serveur PHP
            const xhr = new XMLHttpRequest();
            xhr.open('POST', 'incremente-visites.php', true);
            xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    // Affiche la réponse du serveur dans la console
                    console.log(xhr.responseText);
                }
            };
            // Envoie les données du formulaire (nom de l'animal)
            xhr.send('nom_animal=' + nomAnimal);
        }
```

Cette requête va permettre de récupérer les données de l'animal, et de les envoyer vers un fichier PHP qui incrémente le nombre de visites, via une requête MongoDB

```

// Vérifie si le formulaire a été soumis et récupère le nom de l'animal
if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['nom_animal'])) {
    $nom_animal = $_POST['nom_animal'];

    // Mise à jour des visites dans MongoDB
    $updateResult = $collection->updateOne(
        ['nom_animal' => $nom_animal],
        ['$inc' => ['nombre_visites' => 1]]
    );

    // Répond avec un message indiquant le succès ou l'échec de la mise à jour
    if ($updateResult->getModifiedCount() > 0) {
        echo "Nombre de visites mis à jour avec succès pour $nom_animal.";
    } else {
        echo "Erreur lors de la mise à jour du nombre de visites pour $nom_animal.";
    }
}
```

Le résultat une fois mis à jour, est consultable par l'Administrateur, afin qu'il puisse garder un œil sur quel animal est le plus populaire du zoo, et qui l'est beaucoup moins.

7. Heroku et AtlasDB

7.1 Déploiement de l'Application et démarches

Le déploiement ne fut pas une tâche aisée, car il m'a fallu comprendre comment installer un environnement, exporter et importer correctement des bases de données sur les serveurs, donner des autorisations d'accès à des adresses IP.

Lors de mon déploiement, j'ai changé plusieurs fois de site, en passant par Fly.io, AWS EC2 et RDS, puis Heroku, qui au final m'aura été le plus simple.
J'ai donc procédé ainsi en décomposant plusieurs tâches :

7.2 MongoDB avec AtlasDB

- J'ai créé un Cluster dans Atlas, avec un nom par défaut 

- J'ai cliqué sur le bouton "connect" qui donne toutes les instructions pour connecter ce cluster sur son application MongoDB Compass, qu'il faut installer impérativement afin de permettre son utilisation.
- Une fois l'installation réalisée, un URI de connexion est fourni par le cluster, et est à renseigner dans la connexion de l'application Compass, en remplaçant le champ <password> par un mot de passe qu'on a défini lors de la création du cluster.
- Il ne me reste plus qu'à me connecter, récupérer ma collection au niveau local, et l'importer dans une nouvelle base de données créée sur le Cluster, puis dans le code de mon application changer le lien de la base de données MongoDB.

```
try {
    $mongoClient = new MongoDB\Client("mongodb://localhost:27017/");
```

Lien BDD avant le déploiement

```
try {
    $mongoClient = new MongoDB\Client("mongodb+srv://killiancodes:Test1234@cluster0.ho8h1io.mongodb.net/");
```

Lien BDD après le déploiement

7.3 Déployer sa Base de données SQL et NoSQL sur Heroku

La base de données pour être utilisable, doit être hébergée sur un serveur ClearDB sur Heroku.
Tout d'abord, il m'aura fallu créer une application sur Heroku. Une fois créée, je devais renseigner de nouvelles ressources dans 'add-ons' et y installer ClearDB.

Je me suis par la suite rendu dans les config vars et j'ai créé deux configurations :

- Une qui existait déjà via ClearDB pour phpmyadmin
- Une qui prendra le lien du Cluster AtlasDB pour Mongo

J'ai récupéré les données de configuration de la base de données SQL, et je les ai renseignées dans le fichier ***xampp\phpMyAdmin\config.inc*** afin que lorsque je me rend dans PHPMyAdmin, je puisse avoir accès à ce serveur. Une fois connecté, j'ai créé une base de données, j'ai exporté les tables que j'avais en local et les ai importées sur le serveur, puis j'ai changé dans le code de mon application tous les liens vers la base de données SQL.

Finalement, afin que le site puisse avoir accès à la base de données MongoDB via Atlas, j'ai autorisé sur Atlas la connection de l'IP du site déployé vers le Cluster, de sorte à ce que la page qui contient du MongoDB sur le site ne puisse pas finir vierge une fois déployée.

7.4 Déployer son application avec Heroku

Pour ce qui est du déploiement de l'application, j'ai pu profiter du GITHUB Student Pack qui m'a offert 2 ans gratuit d'utilisation de Heroku, chose extrêmement bénéfique pour une personne qui n'a pas les moyens de déployer en utilisant son argent.

Cette tâche exigeait que j'installe des dépendances, que j'ai dû renseigner dans un fichier composer.json afin de faire en sorte que la base de données mongoDB soit renseignée.

```
{
        "require": {
            "mongodb/mongodb": "^1.9",
            "ext-mongodb": "*"
    }

}

```

Une fois cela fait, j'ai installé le CLI de Heroku, me suis connecté avec mon compte, puis j'ai entré la commande '**- composer install**' afin d'installer le fichier composer.lock ainsi que le dossier *vendor/autoload.php*, (normalement présent dans le dossier composer de XAMPP) que je devais remplacer dans mon code pour faire marcher ma base de données NoSQL.

Par la suite, j'ai réalisé un dépôt Git avec les commandes '**-git add .**' et '**-git commit -m**' puis j'ai déployé le site sur Heroku avec la commande '**-git push heroku master**' qui envoie le dépôt git vers heroku pour être hébergé.

En conclusion

La réalisation de ce site aura été une véritable épreuve, très bénéfique pour mon avancée vers les métiers de l'informatique, un milieu qui exige des délais très court pour la réalisation de tâches complexes, et j'en suis très fier du résultat, même si j'ai quelques regrets concernant le style, et certaines fonctionnalitées via la base de données qui ne sont à mon sens pas assez complets. Désormais je sais comment m'y prendre pour gagner énormément de temps, être autonome et savoir me débrouiller avec les ressources que j'ai à ma disposition, afin de réaliser un travail très qualitatif. Bien que la programmation ait encore beaucoup de secrets que je ne connais pas, le fait d'avoir pu réussir à concevoir cette application démontre mes capacités d'analyse et de gestion des erreurs.
Mes choix technologiques résultent donc des erreurs que j'ai commises, des différentes démarches algorithmiques que j'ai conceptualisées pour cette application, et des difficultés qu'elles engagent.