

DOSSIER DE PROJET

Berbigier Thomas

Développeur Web et Web mobile

Studi

Référence : DPROJET1erdepot-TPDWWMGDW23295_382766_20240727173040

Application Web Zoo Arcadia



Sommaire

<u>Sommaire</u>	2
<u>Résumé du projet</u>	3
<u>Remerciements</u>	4
<u>Introduction</u>	4
1. Compétences du référentiel couvertes par le projet.....	4
a. Développer la partie front-end d'une application web ou web mobile sécurisée.....	4
b. Développer la partie back-end d'une application web ou web mobile sécurisée.....	5
2. Environnement humain et technique.....	5
a. Environnement humain.....	5
b. Environnement technique.....	5
<u>Développement</u>	6
1. Analyse.....	6
2. Développement de l'application.....	8
a. Installation.....	8
b. Les données.....	9
c. Fonctionnalités.....	10
i. Pages d'accueil.....	10
ii. L'authentification.....	11
iii. Vue globale des habitats.....	17
iv. Dashboard administrateur.....	22
v. CRUD Services.....	27
vi. La gestion des avis.....	31
vii. Système de mailing.....	33
viii. Rapports vétérinaire.....	35
ix. Déploiement en ligne.....	37
<u>Conclusion</u>	40
<u>Annexe</u>	41

Résumé du projet

Arcadia est un zoo situé près de la forêt de Brocéliande en Bretagne, France, fondé en 1960.

Le zoo est connu pour son engagement envers le bien-être animal et l'écologie.

José, le directeur d'Arcadia, souhaite moderniser le zoo en développant une application web afin d'améliorer l'expérience des visiteurs et de promouvoir les valeurs écologiques du zoo.

L'application doit offrir aux visiteurs une vue immersive du zoo, renforçant ainsi sa notoriété et son image de marque. En parallèle, elle doit intégrer un aspect fonctionnel avec un espace spécialement conçu pour les professionnels, leur permettant de gérer efficacement les opérations internes et de maintenir le bien-être des animaux.

L'application doit également refléter les valeurs écologiques du zoo, avec un design qui évoque l'indépendance énergétique et l'engagement envers l'environnement.

J'ai développé cette application web en utilisant les technologies suivantes :

- **Front-end : HTML, CSS, JavaScript, Bootstrap** pour un design réactif et attrayant.
- **Back-end : PHP** pour la logique de l'application.
- **Base de données : MongoDB** pour les statistiques, **MariaDB** avec **PDO** pour la gestion des interactions avec la base de données.

Le développement de cette application a été très formateur pour moi, car il s'agit d'un projet complet qui m'a permis de mettre en pratique toutes les compétences acquises lors de ma formation. Travailler sur ce projet m'a offert une expérience précieuse et enrichissante, consolidant ainsi mes connaissances et me préparant mieux pour de futurs projets.

J'ai travaillé seul sur ce projet, y consacrant environ 7 semaines à temps partiel. Le développement de l'application a suivi un cheminement méthodique, étape par étape, pour créer l'interface utilisateur, inclure chaque fonctionnalité, et rédiger la documentation technique du projet.

Remerciements

Je tiens à exprimer ma gratitude envers José, directeur, et Josette, assistante du zoo Arcadia, pour m'avoir offert l'opportunité de développer une application web pour leur établissement. Ce projet, particulièrement stimulant par sa complexité, m'a permis de me confronter à la réalité du métier de développeur et de mettre en pratique les connaissances acquises lors de ma formation, tout en en acquérant de nouvelles.

Je souhaite également remercier Studi et ses formateurs, dont les cours, les lives et les corrections m'ont grandement aidé à progresser dans ma formation. Leur soutien, leur disponibilité et leurs précieux conseils techniques ont été essentiels à mon développement professionnel durant ce projet.

Introduction

1. Compétences du référentiel couvertes par le projet

a. Développer la partie front-end d'une application web ou web mobile sécurisée

- ➔ Installer et configurer son environnement de travail en fonction du projet web ou web mobile
- ➔ Maquetter des interfaces utilisateur web ou web mobile
- ➔ Réaliser des interfaces utilisateur statiques web ou web mobile
- ➔ Développer la partie dynamique des interfaces utilisateur web ou web mobile

L'application Web doit être responsive et dynamique, d'où le choix de Bootstrap qui permet de développer en mobile first et qui inclut des composants interactifs. Des ajustements sont à prévoir côté CSS afin de proposer une expérience utilisateur similaire quelle que soit la taille de l'écran. J'ai donc fait en sorte que l'interface s'adapte aux différentes tailles d'écran.

b. Développer la partie back-end d'une application web ou web mobile sécurisée

- ➔ Mettre en place une base de données relationnelle
- ➔ Développer des composants d'accès aux données **SQL** et **NoSQL**
- ➔ Développer des composants métier côté serveur
- ➔ Documenter le déploiement d'une application dynamique web ou web mobile

La partie front renvoie les données d'une base de données relationnelle **MariaDB** et non relationnelle **MongoDB**. Les différents CRUD (Create, Read, Update, Delete) permettent de manipuler les données de l'application. La partie back est développée avec **PHP**, **PDO (PHP Data Object)** pour l'accès aux données **SQL** et Composer pour gérer les dépendances (**PHPMailer**, **MongoDB**). L'application est ensuite déployée en ligne.

2. Environnement humain et technique

a. Environnement humain

Je travaille seul sur ce projet dans le cadre de ma reconversion professionnelle.

Actuellement salarié, je suis cette formation à temps partiel, notamment les soirs et les weekends. Ce projet est préparé et réalisé selon un planning bien précis, me permettant de gérer efficacement mon temps.

b. Environnement technique

Le développement a été réalisé sous **Windows 11**, utilisant **Apache 2.4** via **XAMPP** comme serveur web. L'application sera déployée en ligne via **Heroku**.

Les maquettes ont été réalisées avec **Figma**.

Côté front-end, j'ai utilisé **HTML5** pour structurer les pages web, **CSS3** pour la présentation avec **Bootstrap 5. Bootstrap**, avec son développement mobile-first et ses composants interactifs, a permis de créer une interface responsive et dynamique, adaptable à toutes les tailles d'écran. J'ai également utilisé des media queries dans le **CSS** pour ajuster l'expérience utilisateur et offrir un design plus personnalisé et cohérent sur différents appareils.

JavaScript (ES6+) a été employé pour ajouter de la dynamique côté client.

Côté back-end, le choix de **PHP (8.2.12)** pour le développement back-end s'explique par le fait que c'est le premier langage back-end que j'ai appris et que je maîtrise le mieux. J'ai suivi la norme PSR pour écrire correctement les instructions. Pour gérer les dépendances **PHP**, j'ai utilisé Composer. Les bases de données utilisées sont **MariaDB 10.4.32** pour la gestion des données relationnelles et **MongoDB 7.0** pour les statistiques, chaque interaction étant facilitée respectivement par **PDO** et la **MongoDB PHP Library**, installée via **Composer**.

Pour le développement, j'ai utilisé **Visual Studio Code** comme environnement de développement intégré (IDE), agrémenté d'extensions telles que **PHP Intelephense**, **PHP Debug**, **Indent-rainbow** et **Prettier** pour améliorer l'efficacité et la qualité du code.

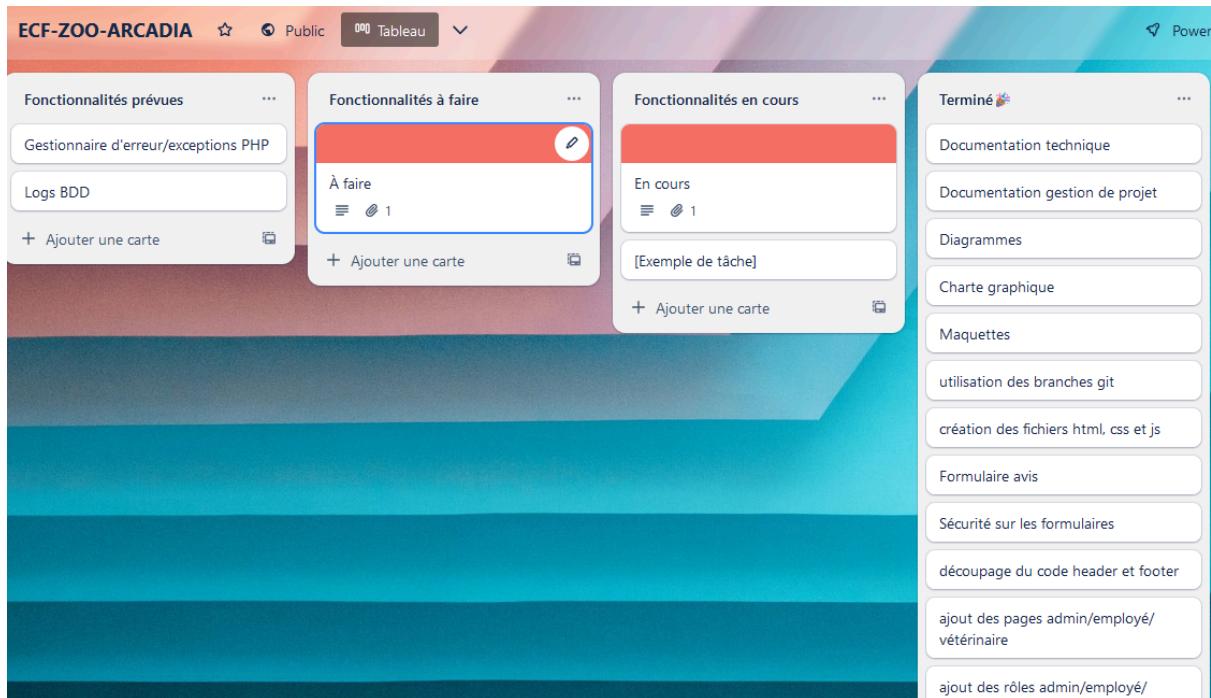
Le contrôle de version a été géré via **Git**, avec les dépôts hébergés sur **GitHub** pour un suivi des versions.

Développement

1. Analyse

Pour aborder le projet, j'ai commencé par une lecture approfondie de l'énoncé pour bien comprendre les besoins du client. Le client souhaitait une application au thème écologique, présentant le zoo, ses habitats, animaux, et services, avec un menu de navigation clair, une gestion des avis, un espace sécurisé pour les professionnels du zoo avec des fonctionnalités spécifiques, un formulaire de contact, et des statistiques sur la popularité des animaux. Une fois ces besoins non techniques clarifiés, j'ai décomposé les exigences en fonctionnalités spécifiques et défini leur mise en œuvre technique.

Cette phase de planification a été gérée sur Trello, où j'ai utilisé un tableau Kanban pour organiser les tâches et suivre l'avancement du projet.



La documentation technique a suivi, incluant une réflexion initiale sur les technologies appropriées et la configuration de l'environnement de travail.

J'ai élaboré des diagrammes pour affiner l'architecture et les interactions de l'application.

Le [**diagramme de classes**](#) (Figure 1 en Annexe) a été réalisé pour définir les structures de données et les relations entre les différentes entités, ce qui a permis de clarifier l'organisation des données et la logique de l'application.

Le [**diagramme de cas d'utilisation**](#) (Figure 2) a décrit les interactions entre les utilisateurs et le système, identifiant les fonctionnalités principales et les scénarios d'utilisation, ce qui a facilité la compréhension des besoins fonctionnels.

Enfin, le [**diagramme de séquence**](#) (Figure 3) a illustré les processus dynamiques au sein du système, montrant comment les différentes opérations et interactions se déroulent au fil du temps, ce qui a aidé à visualiser et planifier le flux des opérations.

Pour compléter l'analyse, j'ai établi la **charte graphique** (Figure 4) et conçu les maquettes (**wireframes** (Figures 5 à 8) et **mockups** (Figures 9 à 14)) pour les pages principales de l'application, comprenant la page d'accueil, les habitats, et les services, tant pour les versions bureautiques que mobiles. Ces maquettes ont servi de base pour le design et l'ergonomie de l'application, en s'assurant que l'interface soit intuitive et adaptée aux besoins des utilisateurs.

Dans le cadre de ce projet, chaque fichier a une responsabilité bien définie, j'ai séparé la logique métier du reste du code pour des raisons de maintenabilité et de clarté. D'une part, les fichiers contenant principalement du **HTML** (formulaires, header, footer, pages de contenu), et d'autre part les fichiers nécessaires pour manipuler les données, gérer les sessions, vérifier les authentifications et effectuer les opérations CRUD (Create, Read, Update, Delete) sur la base de données.

L'application sera développée en respectant les bonnes pratiques de programmation, telles que l'utilisation d'une indentation cohérente pour améliorer la lisibilité du code et faciliter la maintenance. Des commentaires clairs et explicites seront ajoutés tout au long du code pour expliquer et séparer les différentes sections. De plus, le code sera passé au Validator pour vérifier sa conformité aux standards du web, assurant ainsi une meilleure compatibilité et accessibilité sur différents navigateurs. En outre, l'application contiendra des balises HTML et des attributs optimisés pour le référencement SEO (Search Engine Optimization), tels que des balises **meta**, des titres structurés (**h1**, **h2**, etc.), et des descriptions pertinentes, afin d'améliorer la visibilité du site sur les moteurs de recherche et attirer davantage de visiteurs.

2. Développement de l'application

a. Installation

J'ai commencé par créer un dossier dans le répertoire htdocs de **XAMPP**, j'ai ouvert le terminal et navigué jusqu'au dossier créé dans htdocs. J'ai initialisé un dépôt **Git** en utilisant la commande **git init**, puis **git add .** pour ajouter les fichiers nécessaires au projet et enfin **git commit -m "Commit initial"** pour sauvegarder l'état initial du projet. Les bonnes pratiques de Git demandent une branche principale (**main**) et une branche développement

(**dev**), la branche par défaut étant déjà créée en tant que **main**, j'utilise alors la commande **git checkout dev** pour créer la branche développement. J'ai ensuite ouvert **Visual Studio Code** (VS Code), un IDE déjà installé sur mon poste, et j'ai ensuite ouvert le dossier créé dans htdocs. Pour ce faire, j'ai utilisé l'option "Open Folder" dans **VS Code** et sélectionné le dossier.

Dans le terminal intégré à **VS Code**, avec **Composer**, j'ai installé les dépendances nécessaires à mon projet.

Pour installer la dépendance **MongoDB** : **composer require mongodb/mongodb**

Pour installer **PHPMailer** : **composer require phpmailer/phpmailer**

b.Les données

J'ai créé la base de données relationnelle locale via le SGBDR **MariaDB** en utilisant phpMyAdmin, que j'ai nommé "zoo_aradia". J'ai également créé un nouveau compte utilisateur avec un mot de passe solide respectant les normes CNIL (douze caractères, au moins un chiffre, une majuscule et un caractère spécial), en lui attribuant les privilèges nécessaires sur cette base de données. J'ai ensuite créé les tables et inséré les données via l'onglet **SQL** de phpMyAdmin.

```
GRANT USAGE ON *.* TO `administrateur_aradia`@`%`
IDENTIFIED BY PASSWORD
'*5F223EE94DACD2F15FC5F069D835E4ABE94EE511';

GRANT ALL PRIVILEGES ON `zoo\_aradia`.* TO
`administrateur_aradia`@`%`;

CREATE TABLE roles (
    id int(11) NOT NULL AUTO_INCREMENT,
    type varchar(255) NOT NULL,
    PRIMARY KEY (id)
);

INSERT INTO roles (id, type) VALUES
(1, 'administrateur'),
(2, 'employe'),
(3, 'veterinaire');

CREATE TABLE users (
    id int(11) NOT NULL AUTO_INCREMENT,
    email varchar(255) NOT NULL,
    password varchar(255) NOT NULL,
    role_id int(11) NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (role_id)
        REFERENCES roles (id)
);
```

Parallèlement, j'ai configuré la base de données non relationnelle via le SGBD **MongoDB** en utilisant l'interface utilisateur graphique **MongoDB Compass**, que j'ai également nommée "zoo_弧adia". J'y ai créé une collection "animals_clicks" pour stocker les clics des visiteurs sur chaque animal afin de suivre leur popularité.

The screenshot shows the MongoDB Compass interface. At the top, there's a header with a 'x' button and a '+' button. Below it, the database path is shown as 'localhost:27017 > zoo_弧adia > animals_clicks'. The main navigation bar has tabs for 'Documents' (which is selected, showing 10 documents), 'Aggregations', 'Schema', 'Indexes' (with 1 index), and 'Validation'. Below the navigation is a search bar with placeholder text 'Type a query: { field: 'value' } or Get' and buttons for 'Explain', 'Reset', 'Find', 'Options', and 'Export Data'. There are also buttons for 'ADD DATA' and 'EXPORT DATA'. The document list area shows one document with the following fields:

```
_id: ObjectId('66842c892c8744226b022032')
animal_id : 7
animal_name : "Koko"
click_count : 15
```

c. Fonctionnalités

i. Pages d'accueil

La [page d'accueil](#) est le point d'entrée principal du site web du zoo. Elle comprend une image attrayante représentant le zoo, qui capte immédiatement l'attention des visiteurs. Cette image est accompagnée d'une brève présentation du zoo, qui met en avant son histoire, sa mission de conservation, et son engagement écoresponsable.

La page propose également une vue d'ensemble des différents habitats et des animaux que les visiteurs peuvent découvrir, ainsi que des informations sur les services offerts, tels que les visites guidées, les événements spéciaux, et les installations pour les familles.

La présentation des animaux et des habitats dans l'application est réalisée à l'aide de cartes **Bootstrap**. Sur les écrans de bureau, ces cartes sont alignées côte à côte pour offrir une visualisation claire et ordonnée des informations. Sur les écrans mobiles, grâce à l'utilisation des colonnes flexibles du framework, les cartes sont empilées les unes sur les autres pour s'adapter aux dimensions réduites et offrir une navigation fluide. Chaque carte contient des

informations pertinentes sur l'animal ou l'habitat et est équipée d'un bouton **d'action** qui redirige le visiteur vers la section correspondante sur la page **habitats.php**, facilitant ainsi l'accès direct aux informations détaillées sur les habitats.

```
<!-- Début cards habitats/animaux -->
<div class="container accueil">
  <div class="container-card text-center mt-5">
    <div class="row">
      <div class="col-12 col-md-6 col-lg-4 mb-3">
        <div class="card h-100 card-savane">
          
          <div class="card-body">
            <h3 class="card-title text-center">Savane</h3>
            <p class="card-text">Bienvenue dans l'habitat savane de notre zoo, une vaste étendue qui capture l'essence des plaines africaines. Ici, vous pourrez découvrir diverses espèces de girafes et d'animaux de la savane. N'hésitez pas à faire un tour dans nos enclos et à prendre des photos !</p>
            <a href="habitats.php#heading1" class="btn btn-dark fs-5">Je découvre la savane !</a>
          </div>
        </div>
      </div>
    </div>
```

Un **effet de dégradé de couleur** a été appliqué à la seconde carte, ajoutant une touche de style et de dynamisme visuel pour rendre l'interface utilisateur plus attrayante et engageante.

```
/* cards habitats/animaux */
.accueil .card-savane {
  background-color: #rgb(67, 104, 80);
}
.accueil .card-marais {
  background-image: linear-gradient(to right, #436850, #365b46, #2a4f3d, #1e4333, #12372a);
}
.accueil .card-jungle {
  background-color: #rgb(18, 55, 42);
}
```

Enfin, une section dédiée aux avis des visiteurs permet aux utilisateurs de partager leurs expériences et de lire les retours d'autres visiteurs, contribuant ainsi à enrichir l'expérience en ligne et à encourager les visites en personne.

ii. L'authentification

Dans l'application, un visiteur ne peut pas créer de compte, seul l'administrateur, un vétérinaire ou un employé peut se connecter. Le système d'authentification doit alors déterminer le rôle du professionnel se connectant selon les identifiants fournis dans le formulaire de connexion.

Pour accéder aux données présentes dans la base de données relationnelle, j'ai créé un fichier **pdo.php** dans lequel sont définis les paramètres de connexion. Ensuite, j'ai créé un objet **PDO** avec ces paramètres pour établir la connexion à la base de données **MariaDB**. Ce fichier peut être inclus dans d'autres fichiers via un **require_once** afin d'éviter la duplication de code.

```

    // Local
    $username = "administrateur_arcadia";
    $password = "Mr7aF?nsozX4";
    $database = "zoo_arcadia";
    $hostname = "localhost";
}
try
{
    $pdo = new PDO("mysql:host=$hostname;dbname=$database", $username, $password);
    // Activation des erreurs PDO
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // mode de fetch par défaut : FETCH_ASSOC / FETCH_OBJ / FETCH_BOTH
    $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
}
catch (Exception $e)
{
    die('Erreur : ' . $e->getMessage());
}

```

Les mots de passe sont stockés de manière sécurisée dans la base de données en utilisant le hachage avec **password_hash()** lors de la création de nouveaux utilisateurs, garantissant ainsi leur sécurité. Cette fonction transforme le mot de passe pour ne pas le stocker en clair.

Les utilisateurs sont créés via un formulaire sur la page **administrateur.php**.

```
$password = password_hash($_POST['createPassword'], PASSWORD_BCRYPT);
```

```
INSERT INTO `users` (`id`, `email`, `password`, `role_id`) VALUES
(1, 'admin@arcadia.fr', '$2y$10$5PyxN9PEGSCRBhR2SKRto.lcK9bUs1GR3heL7DH94vkifi07jXqJi', 1),
(19, 'veterinaire@arcadia.fr', '$2y$10$sc2t0owJzuM1ZnY2gr/HVuDAqYX.pwPsCeroJAQdYMc5SgeSkGw9K', 3),
(20, 'employe@arcadia.fr', '$2y$10$Ai.xEJKllp2mtucFjklC5.Gp2ouTvq/S77V7pKYNmzxRS9yT7HPGe', 2);
```

Une fois le composant d'accès à la base de données **SQL** défini, j'ai créé un fichier **user.php** contenant une fonction qui compare l'email et le mot de passe entrés dans le formulaire de connexion avec ceux stockés dans la base de données. Cette fonction utilise **PDO** pour interagir de manière sécurisée avec la base de données, exécutant une requête préparée pour sélectionner les informations de l'utilisateur correspondant à l'email fourni. Les requêtes préparées sont sécurisées contre les injections **SQL**.

Si l'utilisateur est trouvé, la fonction utilise **password_verify()** pour comparer le mot de passe fourni avec le mot de passe haché stocké dans la base de données. Si la vérification est réussie, elle retourne les informations de l'utilisateur sous forme de tableau, sinon, elle retourne **false**.

```

function verifyUserLoginPassword(PDO $pdo, string $email, string $password):bool|array
{
    // Requête préparée = requête sécurisée
    $sql = "SELECT * FROM users WHERE email = :email";
    try {
        $query = $pdo->prepare($sql);
        $query->bindValue(':email', $email, PDO::PARAM_STR);
        $query->execute();
    }catch (Exception $e) {
        echo " Erreur ! " . $e->getMessage();
    }
    // fetch() récupère une seule ligne
    $user = $query->fetch(PDO::FETCH_ASSOC);

    // Authentification
    if($user && password_verify($password, $user['password'])) {
        // vérif ok
        return $user;
    } else {
        // email ou mdp incorrects : retourne false
        return false;
    }
}

```

Le bouton “Espace professionnel” permettant de se connecter est présent dans la barre de navigation, mais également dans le footer, présents dans toutes les pages de l’application. Au clic sur le bouton, le composant Bootstrap Offcanvas se déclenche et laisse apparaître le formulaire de connexion. (Figure 17).

```

<!-- Début Offcanvas pour espace professionnel -->
<section>
    <div class="offcanvas offcanvas-end" tabindex="-1" id="offcanvasRight" aria-labelledby="offcanvasRightLabel">
        <div class="offcanvas-header">
            <h2 class="offcanvas-title" id="offcanvasRightLabel">Espace professionnel</h2>
            <button type="button" class="btn-close" data-bs-dismiss="offcanvas" aria-label="Close"></button>
        </div>
        <div class="offcanvas-body">
            <div class="container">
                <div class="container-form">
                    <h2 class="mb-5">Connexion</h2>
                    <form action="login.php" method="POST">
                        <div class="form-group">
                            <label for="email">Email :</label>
                            <input type="email" id="email" name="email" placeholder="Ex : arcadia@zoo.fr" required>
                        </div>
                        <div class="form-group">
                            <label for="password">Mot de passe :</label>
                            <input type="password" id="password" name="password" placeholder="Entrez votre mot de passe" required>
                        </div>
                        <!-- Champ caché pour le token CSRF -->
                        <input type="hidden" name="csrf_token" value=<?= $_SESSION['csrf_token']; ?>>
                        <div class="form-group">
                            <input type="submit" name="loginUser" value="Se connecter">
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</section>
<!-- Fin Offcanvas -->

```

L'utilisation de la méthode **POST** dans le formulaire est cruciale pour la sécurité car elle permet de ne pas afficher les données sensibles (email, mot de passe) dans l'URL.

L'utilisation du token **CSRF** protège l'application en générant un jeton unique lors de la soumission du formulaire qui est ensuite vérifié côté serveur pour s'assurer de l'origine de la requête.

L'attribut **action** spécifie que lorsque l'utilisateur soumet le formulaire, les données saisies sont envoyées à cette URL pour traitement.

Les attributs **required** assurent que les champs soient remplis avant la soumission.

Le type d'entrée **email** permet de bénéficier d'une validation automatique dans les navigateurs compatibles et **password** masque les caractères saisis. L'attribut **placeholder** indique à l'utilisateur la nature du champ à entrer, sans compromettre la sécurité.

Pour traiter les informations de connexion, je crée une page [login.php](#) (Figure 15) où j'importe via des **require_once** le fichier **session.php**, qui configure et démarre une session PHP avec des paramètres de sécurité pour les cookies de session, et génère un token **CSRF** unique pour chaque session utilisateur.

```
session_set_cookie_params([
    'lifetime' => 3600,
    'path' => '/',
    'httponly' => true
]);
session_start();

// Génération du token CSRF
if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
```

Après l'import des fichiers nécessaires, je vérifie si la requête est bien de type **POST**, et que le champ **loginUser** est présent dans les données soumises. Cette vérification assure que le script traite une soumission de formulaire. Une condition vérifie que le token **CSRF** envoyé avec le formulaire correspond à celui stocké dans la session de l'utilisateur, protégeant ainsi contre les attaques **CSRF**.

Je stocke ensuite la fonction `verifyUserLoginPassword()` dans une variable `$user` en passant en argument la connexion à la base de données, l'email et le mot de passe récupérés depuis le formulaire de connexion grâce aux variables superglobales `$_POST['email']` et `$_POST['password']`.

Si l'utilisateur est authentifié, les informations de l'utilisateur sont stockées dans la session pour suivre son état de connexion et ses permissions. Une requête **SQL** récupère le type de rôle de l'utilisateur en fonction de l'ID du rôle. Si le rôle est reconnu, il est stocké dans la session et l'utilisateur est redirigé vers la page appropriée en fonction de son rôle.

Si, lors de la soumission du formulaire, l'identifiant et/ou le mot de passe sont incorrects, l'utilisateur est redirigé vers `login.php` où un message d'erreur affiche '**Email ou mot de passe incorrect.**', empêchant un potentiel attaquant de savoir si seulement l'email ou seulement le mot de passe est incorrect. Si les identifiants sont corrects, les utilisateurs sont redirigés vers leurs espaces respectifs

Un nouvel onglet apparaît alors dans le menu de navigation ainsi que dans le footer, "**Espace administrateur**" en cas de connexion par un administrateur, "**Espace employé**" en cas de connexion par un employé, "**Espace vétérinaire**" en cas de connexion par un vétérinaire. Il y a également un changement du bouton "**Espace professionnel**" en bouton "**Déconnexion**".

```
<?php if (isset($_SESSION['user'])) {  
    if ($_SESSION['role'] == 'administrateur') {?>  
        <a class="nav-link me-5 fs-4" href="administrateur.php">Espace  
        administrateur</a>  
    <?php } else if ($_SESSION['role'] == 'employe') { ?>  
        <a class="nav-link me-5 fs-4" href="employe.php">Espace employé</a>  
    <?php } else if ($_SESSION['role'] == 'veterinaire') { ?>  
        <a class="nav-link me-5 fs-4" href="veterinaire.php">Espace vétérinaire</a>  
    <?php } } ?>  
</div>  
<?php if (isset($_SESSION['user'])) { ?>  
    <a href="logout.php" class="btn btn-outline-light me-5 fs-4">Déconnexion</a>  
    <?php } else { ?>  
        <a href="#" class="btn btn-outline-light me-5 fs-4"  
        data-bs-toggle="offcanvas" data-bs-target="#offcanvasRight"  
        aria-controls="offcanvasRight">Espace professionnel</a>  
<?php } ?>
```

Une classe **navbar-custom** est définie dans le fichier **header.php** permettant de styliser le menu de navigation grâce au fichier **style.css**. Le code **CSS** positionne le menu de navigation en haut de la page, la rend semi-transparente, l'étend sur toute la largeur de la page, et le maintient au-dessus des autres éléments avec un z-index de 1000 car j'ai utilisé une image de fond.

```
/* navbar */
.navbar-custom {
    background-color: □rgb(18, 55, 42, 0.5);
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    z-index: 1000;
}
.nav-link {
    color: ■rgb(255, 255, 255);
```

L'image de fond est placée en arrière-plan grâce au z-index -1 avec une position fixe.

```
/* image de fond */
.background {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    z-index: -1; /* Mettre en arrière-plan */
}
/* Style pour l'image de fond "blobs" */
.blob-image {
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    opacity: 1;
    filter: brightness(0.5)
}
```

Pour le côté responsive de l'application, les éléments du menu de navigation sont sous forme de menu déroulant grâce à la classe **navbar-toggler**, l'icône du menu déroulant est modifiée avec le fichier **CSS**, le médiaquerie permet de ne plus rendre la couleur de fond du menu déroulé transparente afin d'apporter une meilleure visibilité.

```
.navbar-toggler {
    border-color: #rgba(255, 255, 255);
}
.navbar-toggler-icon {
    background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 30 30'%3E%3Cpath stroke='rgba%28255, 255, 255, 1%29' stroke-linecap='round' stroke-miterlimit='10' stroke-width='2' d='M4 7h22M4 15h22M4 23h22'%3E%3C/svg%3E");
}
```

```
/* media queries mobile */
@media only screen and (max-width: 600px) {
    .navbar-collapse {
        background-color: #rgb(18, 55, 42);
    }
}
```

Enfin, la fonctionnalité de déconnexion est également mise en place grâce au fichier **logout.php**, détruisant la session utilisateur pour sécuriser la fin de session.

```
<?php
require_once __DIR__ . "/lib/session.php";

// Prévient les attaques de fixation de session
session_regenerate_id(true);

// Supprime les données du serveur
session_destroy();

// Supprime les données du tableau $_SESSION
unset($_SESSION);

// Redirige vers la pages d'accueil
header('location:index.php');
```

iii. Vue globale des habitats

L'application comprend une page dédiée aux habitats, qui liste tous les habitats du zoo avec les animaux qui y sont associés. Lorsqu'un utilisateur clique sur un habitat, il peut voir les détails de l'habitat ainsi que les animaux qui y résident. En cliquant sur un animal, le visiteur accède aux détails de l'animal, incluant ses propriétés et l'avis du vétérinaire. Cette partie est très importante pour José, le directeur, dont la fierté repose sur le bien-être de ses animaux.

Pour développer cette fonctionnalité, j'ai besoin d'une requête **SQL** pour récupérer les informations sur les habitats et les animaux associés, ainsi que l'état des animaux rapporté par les vétérinaires. J'ai commencé par créer la page [**habitats.php**](#). Après avoir inclus les fichiers nécessaires, j'ai récupéré les lignes des trois tables de ma base de données **SQL** et les ai combinées.

La requête **SQL** sélectionne les colonnes nécessaires et leur attribue un alias pour éviter tout conflit. Elle commence par la table **habitats**, effectue une jointure gauche avec la table **animals** en utilisant la colonne **name** de **habitats** et la colonne **habitat** de **animals** et une seconde jointure gauche entre **animals** et **reports** en utilisant la colonne **id** de **animals** et la colonne **animal_id** de **reports**. Ainsi, tous les habitats sont retournés, même ceux sans animaux associés, donc un habitat sans animal sera quand même affiché, et tous les animaux sont retournés, même ceux sans rapports associés. Les résultats sont triés par l'identifiant de l'habitat puis par celui de l'animal. La requête est préparée et exécutée en utilisant **PDO**, et en cas d'erreur, une exception est attrapée et un message d'erreur est

affiché.

```
$sql = "SELECT habitats.id AS habitat_id, habitats.name AS habitat_name, habitats.description AS habitat_description, habitats.picture AS habitat_picture, animals.id AS animal_id, animals.name AS animal_name, animals.race AS animal_race, animals.picture AS animal_picture, reports.state AS animal_state
FROM habitats
LEFT JOIN animals ON habitats.name = animals.habitat
LEFT JOIN reports ON animals.id = reports.animal_id
ORDER BY habitats.id, animals.id;";
try {
$stmt = $pdo->prepare($sql);
$stmt->execute();
} catch (Exception $e) {
echo " Erreur ! " . $e->getMessage();
}
$results = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Ensuite, j'ai initialisé un tableau vide **\$habitats** pour stocker les informations des habitats et des animaux. Une boucle **foreach** parcourt chaque ligne des résultats de la requête. Pour chaque ligne, l'identifiant de l'habitat est récupéré. Si l'habitat n'est pas encore dans le tableau **\$habitats**, il est ajouté avec ses informations de base (nom, description, image) ainsi qu'un tableau vide pour les animaux. Si une ligne contient des informations sur un animal, ces informations sont ajoutées dans le tableau **animals** de l'habitat correspondant.

```
$habitats = [];

foreach ($results as $row) {
    $habitat_id = $row['habitat_id'];
    // Données habitat, initialisation tableau vide pour animaux
    if (!isset($habitats[$habitat_id])) {

        $habitats[$habitat_id] = [
            'name' => $row['habitat_name'],
            'description' => $row['habitat_description'],
            'picture' => $row['habitat_picture'],
            'animals' => []
        ];
    }
    // Données animaux dans habitat correspondant
    if ($row['animal_id'] !== null) {

        $habitats[$habitat_id]['animals'][$row['animal_id']] = [
            'name' => $row['animal_name'],
            'race' => $row['animal_race'],
            'picture' => $row['animal_picture'],
            'state' => $row['animal_state']
        ];
    }
}
```

Pour la page **habitats.php**, j'ai utilisé des conteneurs pour la mise en page et des composants **Bootstrap** pour afficher les habitats, les animaux et leurs détails. Celà garantit un design responsive à l'application. Une boucle **foreach** parcourt le tableau **\$habitat** et crée une carte pour chaque habitat. Un composant **Bootstrap** collapse permet, au clic sur un habitat, de faire apparaître en dessous le nom de l'habitat ainsi que les animaux qui y résident.

```
<?php foreach ($habitats as $habitat_id => $habitat) { ?>
```

Pour sécuriser les données, j'ai utilisé la fonction **htmlspecialchars()**, qui convertit les caractères spéciaux en entités HTML, empêchant ainsi les attaques XSS (Cross-Site Scripting). Les variables PHP personnalisent le contenu de la page en fonction des données récupérées depuis la base de données.

```
<div class="col-12 my-4 d-flex justify-content-center" id="heading<?= $habitat_id ?>">
    <!-- Début cards -->
    <div class="card <?= $habitatClass ?>">
        " data-bs-toggle="collapse" data-bs-target="#collapse<?= $habitat_id ?>" role="button" aria-controls="collapse<?= $habitat_id ?>" tabindex="0">
        <div class="card-body ">
            <h2 class="card-title text-center"><?= htmlspecialchars($habitat['name']) ?></h2>
            <div class="collapse" id="collapse<?= $habitat_id ?>" aria-labelledby="heading<?= $habitat_id ?>" data-bs-parent="#accordionExample">
                <div class="collapse-habitats">
                    <div class="container">
                        <div class="fs-5">
                            <p class="habitats-text"><?= htmlspecialchars($habitat['description']) ?></p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

Pour les cartes représentant les animaux, une seconde boucle parcourt les animaux associés à chaque habitat et génère des cartes pour chaque animal. J'ai ajouté un composant **Bootstrap** accordéon permettant de visualiser les informations sur l'animal et son état de santé. **Bootstrap**, grâce à son système de grille flexible, permet d'adapter automatiquement

l'affichage des éléments en fonction de la taille de l'écran. Dans mon code, la classe col-12 garantit que l'élément occupe toute la largeur (12 colonnes) sur les petits écrans, tandis que col-md-6 et col-lg-4 ajustent la largeur respectivement à 6 colonnes sur les écrans moyens et à 4 colonnes sur les grands écrans. (Figure 18)

Grâce à cette requête SQL et à une structuration efficace des données, j'ai pu créer une vue d'ensemble des habitats, des animaux associés et des détails spécifiques, incluant l'avis du vétérinaire.

```
<div class="col-12 col-md-6 col-lg-4">
  <div class="card <?php echo $animalClass ?>">
    ">
    <div class="card-body">
      <!-- Début accordéon animaux -->
      <div class="accordion" id="<?= 'accordion'.htmlspecialchars($animal['name']) ?>">
        <div class="accordion-item">
          <div class="accordion-header">
            <button class="accordion-button fs-2 <?= $accordionClass ?>" data-animal-id="<?= $animal_id ?>" data-bs-toggle="collapse" data-bs-target="#<?= '#collapse'.htmlspecialchars($animal['name']) ?>" aria-expanded="true" aria-controls="collapseAnimals" type="button" tabindex="0">
              <?= htmlspecialchars($animal['name']) ?>
            </button>
          </div>
          <div id="<?= '#collapse'.htmlspecialchars($animal['name']) ?>" class="accordion-collapse collapse" data-bs-parent="#<?= '#accordion'.htmlspecialchars($animal['name']) ?>">
            <div class="accordion-body <?= $bodyClass ?>">
              <ul class="text-start">
                <li>
                  <p>Race : <?= htmlspecialchars($animal['race']) ?></p>
                </li>
                <li>
                  <p>Etat de l'animal : <?= htmlspecialchars($animal['state']) ?></p>
                </li>
              </ul>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
<?php foreach ($habitat['animals'] as $animal_id => $animal) { ?>
```

Pour donner un effet visuel indiquant à l'utilisateur que la carte est interactive, c'est-à-dire qu'il peut cliquer dessus, j'ai ajouté un léger zoom au survol de la carte. Pour cela j'ai utilisé du code **CSS** par le biais de classes ajoutées grâce à une condition **PHP**.

```
<?php if ($habitat['name'] == 'savane') {
    $habitatClass = 'habitat-savane';
} else if ($habitat['name'] == 'marais') {
    $habitatClass = 'habitat-marais';
} else if ($habitat['name'] == 'jungle') {
    $habitatClass = 'habitat-jungle';
} else {
    $habitatClass = 'habitat-marais';
} ?>
```

Le code **CSS** suivant applique un léger agrandissement de 5% de la carte au survol avec une transformation fluide de 0.3 seconde pour offrir une expérience utilisateur agréable.

```
.habitat-savane,  
.habitat-marais,  
.habitat-jungle {  
    max-width: 70%;  
    transition: transform 0.3s ease;  
    margin-top: 1%;  
}  
.habitat-savane:hover,  
.habitat-marais:hover,  
.habitat-jungle:hover {  
    transform: scale(1.05);  
}
```

iv. Dashboard administrateur

L'application comprend une section dans la page **administrateur.php** permettant à José, directeur et administrateur, de visualiser quels animaux plaisent le plus. La popularité des animaux est mesurée selon le nombre de clics que les visiteurs effectuent sur chaque animal. (Figure 19).

Pour stocker ces clics, j'ai commencé par créer une base de données non relationnelle **MongoDB** via **MongoDB Compass** ainsi qu'une collection dédiée. Dans la page **habitats.php**, où se trouvent les animaux, j'ai implémenté une fonction **JavaScript** qui sélectionne tous les éléments du **DOM** ayant la classe **accordion-button**. La méthode **foreach()** itère sur chaque boutons et ajoute un écouteur d'événement pour le clic. Lorsque le bouton est cliqué, une fonction anonyme est exécutée. Les constantes récupèrent respectivement l'**ID** de l'animal et son nom. La méthode **fetch()** envoie ensuite une requête **HTTP** utilisant un en-tête de contenu approprié et transmettant les données de manière sécurisée en **JSON** vers le fichier **register_click.php** qui contient la logique pour stocker les données en base de données. Le corps de la requête contient les données de l'animal (**ID** et nom), et une méthode traite la

réponse du serveur en la convertissant en **JSON**.

```
document.querySelectorAll('.accordion-button').forEach(button => {
    button.addEventListener('click', () => {
        const animalId = button.getAttribute('data-animal-id');
        const animalName = button.textContent.trim();

        // Envoi une requête HTTP à 'register_click.php'
        fetch('register_click.php', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            // Corps de la requête, conversion en chaîne JSON
            body: JSON.stringify({ animal_id: animalId, animal_name: animalName })
        })
        .then(response => response.json())
    });
});
```

Chaque fois qu'un utilisateur clique sur un animal dans l'interface, les informations sur cet animal sont envoyées à **register_click.php**, où elles sont enregistrées dans la base de données. Ce fichier charge tout d'abord la bibliothèque **MongoDB** gérée par **Composer**, et utilise la classe **Client** de cette bibliothèque pour établir la connexion avec la base de données.

```
require 'vendor/autoload.php';

use MongoDB\Client;
```

L'en-tête de la réponse **HTTP** est définie pour d'indiquer que son contenu est au format **JSON**. Les données envoyées par la requête précédente sont stockées dans une variable **\$data** puis validées pour vérifier que les champs **animal_id** et **animal_name** sont bien présents. En cas de données invalides, une erreur “Bad Request” est renvoyée.

```

// Réponse en JSON
header('Content-Type: application/json');

// Lit le corps brut de la requête et le décode en tableau associatif PHP
$data = json_decode(file_get_contents('php://input'), true);

if (!isset($data['animal_id']) || !is_numeric($data['animal_id']) || !isset($data['animal_name'])) {

    http_response_code(400);
    echo json_encode(['error' => 'Invalid input']);
    exit;
}

$animal_id = (int) $data['animal_id'];
$animal_name = $data['animal_name'];

```

J'ai ensuite configuré la connexion à **MongoDB**. En développement local, j'ai utilisé l'URL de connexion par défaut. J'ai initialisé une nouvelle connexion **MongoDB** et sélectionné la collection **animals_clicks** de la base de données **zoo_arctadia** créé au préalable.

```

// Local
$connect = "mongodb://localhost:27017";
}

try {
    $client = new Client($connect);
    $collection = $client->zoo_arctadia->animals_clicks;

```

La partie suivante traite de la vérification et de la mise à jour des clics. J'ai créé un filtre pour rechercher l'animal par son ID et vérifier si cet animal (document) existait dans la collection en utilisant la méthode **findOne()**. C'est une opération qui ne nécessite qu'un seul document, cette méthode est alors plus appropriée.

```

// Vérification si l'animal doit être enregistré
$filter = ['animal_id' => $animal_id];
$existingClick = $collection->findOne($filter);

```

Si l'animal n'existe pas dans la collection, il est ajouté avec la méthode **insertOne()**, et son compteur est initialisé à 1. Une réponse appropriée est renvoyée selon la réussite ou non de l'insertion.

```

if (!$existingClick) {

    // Animal non enregistré, enregistrement dans MongoDB
    $result = $collection->insertOne([
        'animal_id' => $animal_id,
        'animal_name' => $animal_name,
        'click_count' => 1,
    ]);

    if ($result->getInsertedCount() === 1) {
        echo json_encode(['status' => 'success', 'message' => 'Clic enregistré avec succès']);
    } else {
        http_response_code(500);
        echo json_encode(['error' => 'Erreur lors de l\'enregistrement du clic']);
    }
}

```

Si l'animal existe déjà dans la collection, son compteur est alors incrémenté de 1 en utilisant la méthode **updateOne()**.

```

} else {
    // Animal déjà enregistré, mise à jour du compteur de clics
    $result = $collection->updateOne(
        [
            '$inc' => ['click_count' => 1]
        ]
    );
}

```

Pour afficher les données à l'administrateur, je me suis dirigé vers la page **administrateur.php**, où j'ai récupéré les documents de ma collection **MongoDB** et les ai triés par ordre décroissant. La méthode **find()** est utilisée pour récupérer et trier plusieurs documents, avec un premier paramètre récupérant tous les documents et un second paramètre spécifiant le tri selon le champ **click_count** en ordre décroissant **-1**.

```

    // Local
    $connect = "mongodb://localhost:27017";
}

try {
    $client = new Client($connect);
    $collection = $client->zoo_arcadia->animals_clicks;
    // Récupère tous les documents de la collection, trie en ordre décroissant

    $cursor = $collection->find([], [
        'sort' => ['click_count' => -1]
    ]);
} catch (Exception $e) {
    $_SESSION['error'] = " Erreur ! " . $e->getMessage();
}

```

Les documents de la collection **animals_clicks** sont donc mis à jour selon les animaux présents dans la page **habitats.php**, si un animal est ajouté alors un nouveau document sera créé avec un compteur mis à jour.

```

_id: ObjectId('66842c892c8744226b022032')
animal_id : 7
animal_name : "Koko"
click_count : 15

}

_id: ObjectId('66842cba2c8744226b022033')
animal_id : 8
animal_name : "Goliath"
click_count : 21

}

_id: ObjectId('66842cc22c8744226b022034')
animal_id : 9
animal_name : "Talia"
click_count : 12

}

_id: ObjectId('66842cc92c8744226b022035')
animal_id : 6
animal_name : "Billy"
click_count : 12

}

_id: ObjectId('66842ccb2c8744226b022036')
animal_id : 5
animal_name : "Flora"
click_count : 11

```

Les données récupérées sont ensuite affichées dans un tableau Bootstrap pour garantir que l'application soit responsive. Une boucle **foreach** parcourt chaque document retourné par la

requête **MongoDB** précédent, et les données sont affichées dans des cellules de tableau en utilisant **htmlspecialchars()** pour éviter les failles XSS.

```
<!-- Début dashboard clics animaux -->
<section>
    <div class="container mt-5">
        <h2 class="text-center text-light">Tableau de bord des clics sur les animaux</h2>
        <div class="row d-flex justify-content-center">
            <div class="col-12 col-lg-6">
                <table class="table table-striped table-hover mt-3 ">
                    <thead>
                        <tr>
                            <th>Nom de l'animal</th>
                            <th>Nombre de clics</th>
                        </tr>
                    </thead>
                    <tbody>
                        <?php foreach ($cursor as $document) { ?>
                        <tr>
                            <td><?= htmlspecialchars($document['animal_name']) ?></td>
                            <td><?= htmlspecialchars($document['click_count']) ?></td>
                        </tr>
                        <?php } ?>
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</section>
<!-- Fin dashboard clics animaux -->
```

v. CRUD Services

L'application inclut plusieurs fonctionnalités **CRUD** (Create, Read, Update, Delete), notamment pour la gestion des services du zoo dans les interfaces réservées aux administrateurs et employés. Ces derniers peuvent, via des formulaires accessibles depuis leur espace dédié, créer, modifier et supprimer des services (Figure 22). En fonction des actions effectuées, les services seront ajoutés ou retirés de la page [services.php](#).

Pour la partie création, j'ai créé une page **services_form.php** contenant un formulaire **Bootstrap** recueillant un nom, une description, et une image associée via un téléchargement de fichier. Tous les champs sont obligatoires grâce à l'attribut **required**, ce qui empêche la soumission de formulaires incomplets.

L'attribut **enctype="multipart/form-data"** garantit l'intégrité des fichiers téléchargés, permettant ainsi de les traiter en toute sécurité.

De plus, un jeton CSRF est inclus pour prévenir les soumissions frauduleuses.

```
<h2>Ajouter un service</h2>
<!-- le formulaire contient un file upload -->
<form action="services_crud.php" method="POST" enctype="multipart/form-data">
    <div class="mt-2">
        <label for="add_name_service" class="form-label">Nom du Service :</label>
        <input type="text" id="add_name_service" name="add_name" class="form-control" required>
    </div>
    <div class="mt-2">
        <label for="add_description_service" class="form-label">Description :</label>
        <textarea name="add_description" id="add_description_service" class="form-control" required></textarea>
    </div>
    <div class="mt-2">
        <label for="add_picture_service" class="form-label">Image :</label>
        <input type="file" id="add_picture_service" name="add_picture" class="form-control" required>
    </div>
    <!-- Champ caché pour le token CSRF -->
    <input type="hidden" name="csrf_token" value=<?= $_SESSION['csrf_token']; ?>>
    <button type="submit" name="add_service" class="btn btn-outline-light mt-2 btn-lg">Ajouter le Service</button>
</form>
```

Lorsque le formulaire est soumis, les données sont envoyées pour validation côté serveur sur la page **services_crud.php**. Cette page vérifie que la requête reçue est bien une requête POST, confirmant que le formulaire a été soumis, et s'assure que le jeton CSRF est présent et valide.

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (isset($_POST['csrf_token']) && $_POST['csrf_token'] === $_SESSION['csrf_token']) {
        // CRUD SERVICE
        // Création d'un service
        if (isset($_POST['add_service'])) {
```

Les valeurs des champs **add_name** et **add_description** sont nettoyées pour supprimer les caractères spéciaux, protégeant ainsi contre les injections de code malveillant (attaques XSS). L'image téléchargée est également vérifiée pour s'assurer qu'elle est dans un format autorisé (JPEG, JPG, PNG, SVG, WEBP) et que sa taille ne dépasse pas 3 Mo.

```
// stockage données du formulaire
$name = filter_input(INPUT_POST, 'add_name', FILTER_SANITIZE_FULL_SPECIAL_CHARS);
$description = filter_input(INPUT_POST, 'add_description', FILTER_SANITIZE_FULL_SPECIAL_CHARS);
// Sécurité attaques XSS
$file_name = strip_tags($_FILES['add_picture']['name']);
$file_size = $_FILES['add_picture']['size'];
$file_tmp = $_FILES['add_picture']['tmp_name'];
$file_type = $_FILES['add_picture']['type'];

$file_ext = explode('.', $file_name);
$file_end = end($file_ext);
$file_end = strtolower($file_end);
$extensions = [ 'jpeg', 'jpg', 'png', 'svg', 'webp'];
```

Si l'une de ces conditions n'est pas respectée, un message d'erreur est renvoyé à l'utilisateur. Le nom du fichier est également nettoyé pour enlever tout caractère spécial susceptible de poser des problèmes de sécurité ou de compatibilité.

```
if(in_array($file_end, $extensions) === false) {
    $_SESSION['error'] = "Veuillez utiliser les extensions suivantes : JPEG, JPG , PNG , SVG, WEBP";
} elseif($file_size > 3000000) {

    $_SESSION['error'] = "Le fichier est trop volumineux";
} else {
    // Supprime les caractères spéciaux
    $file_name = preg_replace('/[^A-Za-z0-9\.-]/', ' ', $file_name);
    $file_bdd = "assets/main/services/".$file_name;
    // Déplacer l'image uploadée dans le répertoire souhaité
    move_uploaded_file($file_tmp, $file_bdd);
```

Si toutes les vérifications sont passées avec succès, le fichier est déplacé dans le répertoire approprié, et les données du service (nom, description, et chemin de l'image) sont insérées dans la base de données via une requête préparée, ce qui protège contre les injections **SQL**.

```
if (!empty($name) && !empty($description) && !empty($file_bdd)) {
    $sql = 'INSERT INTO services (name, description, picture) VALUES (?, ?, ?)';
    try {
        $stmt = $pdo->prepare($sql);
        $stmt->execute([$name, $description, $file_bdd]);
    } catch (Exception $e) {
        $_SESSION['error'] = "Erreur lors de l'ajout du service.". $e->getMessage();
    }
    $_SESSION['message'] = "Service ajouté avec succès.";
}
```

En cas d'erreur, un message d'alerte est affiché pour informer l'utilisateur. Sinon, un message de confirmation indique que le service a été ajouté avec succès.

Sur la page **services.php**, le nouveau service est affiché grâce à une boucle qui parcourt les services enregistrés dans la base de données. Ces services sont présentés sous forme de cards **Bootstrap**, stylisées avec le fichier **style.css**.

```
<!-- Début cards services-->
<div class="container">
    <div class="container-card text-center mt-5">
        <div class="row">
            <?php foreach ($services as $service) { ?>
                <div class="col-12 col-md-6 mb-3">
                    <div class="card h-100 card-services" style="width: 100%;">
                        
                        <div class="card-body">
                            <h2 class="text-center h3"><?= htmlspecialchars($service['name']) ?></h2>
                            <p><?= htmlspecialchars_decode($service['description'], ENT_QUOTES) ?></p>
                        </div>
                    </div>
                </div>
            </?php endforeach ?>
        </div>
    </div>
</div>

/* Page services */
.card-services {
    background-color: #rgb(251, 250, 218, 0.7);
    color: #rgba(0, 0, 0);
}
```

Pour résumer, dans le cadre de la création d'un service, les données en entrée incluent : le nom du service (texte), la description du service (texte), l'image du service (fichier) et le token **CSRF** (caché).

Durant le traitement, l'application compare le token **CSRF** avec celui stocké en session pour valider la requête, filtre les caractères spéciaux, nettoie le nom du fichier, vérifie le type et la taille du fichier, puis stocke le fichier et insère les données dans la base via une requête préparée si elles sont valides.

En sortie, un message de confirmation informe l'utilisateur du succès de l'opération, ou un message d'erreur lui signale tout problème survenu. Le service nouvellement créé est alors affiché sur la page **services.php**.

Pour la partie modification et suppression, j'ai conçu un formulaire commun avec deux boutons distincts : l'un pour modifier le service, et l'autre pour le supprimer. Une boucle parcourt et affiche toutes les données des services enregistrés dans la base de données, permettant ainsi de modifier directement le texte ou l'image associée à chaque service. L'ajout d'une nouvelle image lors de la modification d'un service est facultatif.

L'attribut **onclick="return confirm('Êtes-vous sûr de vouloir supprimer ce service ?');**"

déclenche l'exécution d'un script **JavaScript** lorsque l'utilisateur clique sur le bouton

Supprimer le Service. Ce script affiche une boîte de dialogue de confirmation, demandant à l'utilisateur de valider ou d'annuler l'action avant de procéder.

```
<h2 class="pt-5">Modifier / Supprimer un service</h2>
<?php foreach ($services as $service) { ?>
    <div class="card mb-3 mt-3">
        <img src=<?= htmlspecialchars($service['picture']) ?>" alt="Image du service" class="img-fluid">
        <div class="card-body">
            <form action="services_crud.php" method="post" enctype="multipart/form-data">
                <input type="hidden" name="id" value=<?= $service['id'] ?>">
                <div class="mb-3">
                    <label for=<?= htmlspecialchars($service['name']) ?>" class="form-label fs-5">Nom du Service :</label>
                    <input type="text" id=<?= htmlspecialchars($service['name']) ?>" name="ud_name" class="form-control" value=<?= htmlspecialchars($service['name']) ?>" required>
                </div>
                <div class="mb-3">
                    <label for=<?= htmlspecialchars($service['id']) ?>" class="form-label fs-5">Description :</label>
                    <textarea name="ud_description" id=<?= htmlspecialchars($service['id']) ?>" class="form-control" required><?= htmlspecialchars_decode($service['description'], ENT_QUOTES) ?></textarea>
                </div>
                <div class="mb-3">
                    <label for=<?= htmlspecialchars($service['picture']) ?>" class="form-label fs-5">Image (laisser vide pour conserver l'actuelle) :</label>
                    <input type="file" id=<?= htmlspecialchars($service['picture']) ?>" name="ud_picture" class="form-control">
                </div>
                <!-- Champ caché pour le token CSRF -->
                <input type="hidden" name="csrf_token" value=<?= $_SESSION['csrf_token'] ?>">
                <button type="submit" name="update_service" class="btn btn-warning">Mettre à jour le Service</button>
                <button type="submit" name="delete_service" class="btn btn-danger" onclick="return confirm('Êtes-vous sûr de vouloir supprimer ce service ?');">Supprimer le Service</button>
            </form>
        </div>
    </div>
<?php } ; ?>
```

Côté serveur, la gestion de la modification et de la suppression est similaire à celle de la création d'un service, à la différence qu'une ligne de code supplémentaire identifie le service à modifier ou à supprimer en fonction de son ID.

```
$id = filter_input(INPUT_POST, 'id', FILTER_VALIDATE_INT);
```

```
$sql = 'UPDATE services SET name = ?, description = ?, picture = ? WHERE id = ?';
```

```
$sql = 'DELETE FROM services WHERE id = ?';
```

vi. La gestion des avis

L'application possède un espace où les visiteurs peuvent laisser un avis, ainsi qu'un espace où les avis sont affichés (Figure 21). Pour laisser un avis, j'ai conçu un formulaire **Bootstrap** sur la page **index.php**. Celui-ci récupère le nom, l'avis du visiteur et un token CSRF caché. Les données sont envoyées au fichier **submit_review.php** pour être traitées.

```
<h3>Laisser un avis</h3>
</div>
<div class="card-body">
  <form action="submit_review.php" method="POST">
    <div class="mb-3">
      <label for="name" class="form-label form-label-bg fs-5">Nom</label>
      <input type="text" id="name" class="form-control" name="name" placeholder="Ecrivez votre pseudo ici" required>
    </div>
    <div class="mb-3">
      <label for="comment" class="form-label form-label-bg fs-5">Avis</label>
      <textarea class="form-control" id="comment" name="comment" rows="3" placeholder="Ecrivez votre texte ici" required></textarea>
    </div>
    <!-- Champ caché pour le token CSRF -->
    <input type="hidden" name="csrf_token" value=<?= $_SESSION['csrf_token']; ?>>
    <button type="submit" name="submit_review" class="btn btn-dark fs-5">Soumettre</button>
  </form>
```

Si les données sont jugées valides, l'avis est enregistré en base de données avec un statut "en attente de validation" (**validate = 0**).

```
$sql = "INSERT INTO reviews (name, comment, validate) VALUES (?, ?, 0)";
```

Par la suite, un employé peut valider ou invalider l'avis via son interface. Cette action met à jour le statut de l'avis dans la base de données en fonction de la décision prise.

```
$id = filter_input(INPUT_POST, 'id', FILTER_VALIDATE_INT);
$action = $_POST['action'];

if ($action === 'validate') {
    $status = 1;
} elseif ($action === 'invalidate') {
    $status = 2;
}

if (!empty($id)) {
    $sql = "UPDATE reviews SET validate = ? WHERE id = ?";
    try {
        $stmt = $pdo->prepare($sql);

        $stmt->execute([$status, $id]);
        $_SESSION['message'] = "L'avis a bien été traité";
    } catch(Exception $e){
        $_SESSION['error'] = "Erreur lors du traitement de l'avis.";
    }
}
```

Après soumission d'un avis, l'utilisateur reçoit un message de confirmation si l'opération a été réussie ou un message d'erreur en cas de problème. Les employés reçoivent également des messages de succès ou d'erreur après avoir validé ou invalidé un avis. Enfin, les avis validés sont affichés dynamiquement dans un carrousel sur la page web, où chaque avis est présenté dans une carte Bootstrap stylisée. Les utilisateurs peuvent parcourir les différents avis validés à l'aide de contrôles de navigation intégrés au carrousel.

```
<!-- Présentation avis-->
<div id="reviewsCarousel" class="carousel carousel-dark review-carousel slide mt-5" data-bs-ride="carousel">
    <div class="carousel-inner" id="reviewsSection">
        <?php foreach ($reviews as $index => $review) { ?>
            <div class="carousel-item <?= $index === 0 ? 'active' : '' ?>">
                <div class="card review-card text-center">
                    <div class="card-header fs-5">
                        <h3 class="h5"><?= htmlspecialchars($review['name']) ?></h3>
                    </div>
                    <div class="card-body fs-5" id="reviewsComment">
                        <p><?= htmlspecialchars($review['comment']) ?></p>
                    </div>
                </div>
            </div>
        <?php } ?>
        <button class="carousel-control-prev" type="button" data-bs-target="#reviewsCarousel" data-bs-slide="prev">
            <span class="carousel-control-prev-icon" aria-hidden="true"></span>
            <span class="visually-hidden">Précédent</span>
        </button>
        <button class="carousel-control-next" type="button" data-bs-target="#reviewsCarousel" data-bs-slide="next">
            <span class="carousel-control-next-icon" aria-hidden="true"></span>
            <span class="visually-hidden">Suivant</span>
        </button>
    </div>
</div>
```

Le carrousel de présentation des avis et du formulaire permettant de laisser un avis ont été stylisés avec le fichier **style.css**.

```
/* Avis */
.review-carousel,
.form-card {
    max-width: 500px;
    margin: 0 auto;
    background-color: #rgb(173, 188, 159, 0.9);
    color: #rgba(0, 0, 0);
    border-radius: 5px;
}
.carousel-item {
    padding: 20px;
}
.review-card {
    background-color: #rgb(255, 255, 255, 0.5);
    color: #rgba(0, 0, 0);
}
```

vii. Système de mailing

L'application comprend deux fonctionnalités où le système de mailing est sollicité, lors de la création d'un compte utilisateur pour l'envoi d'un mail de confirmation, et lors de la soumission du formulaire de contact.

Pour la création de compte utilisateur, cela se passe depuis un formulaire sur l'espace administrateur. J'ai des données en entrée qui incluent l'email, le mot de passe (hashé pour la sécurité et la confidentialité), le rôle de l'utilisateur (employé ou vétérinaire), et un token CSRF caché. L'élément `<select>`, avec ses options fixes, contribue à sécuriser le formulaire en limitant les valeurs possibles envoyées au serveur, ce qui aide à prévenir les entrées non autorisées ou malveillantes.

```
<h2>Créer un compte utilisateur</h2>
<form action="administrateur.php" method="POST">
<div class="mt-2">
    <label for="createEmail" class="form-label">Email :</label>
    <input type="email" class="form-control" id="createEmail" name="createEmail" placeholder="Entrer l'email" required>
</div>
<div class="mt-2">
    <label for="createPassword" class="form-label">Mot de passe :</label>
    <input type="password" class="form-control" id="createPassword" name="createPassword" placeholder="Entrer le mot de passe" required>
</div>
<div class="mt-2">
    <label for="selectRole" class="form-label">Rôle :</label>
    <select class="form-select" id="selectRole" name="selectRole" aria-label="Default select example" required>
        <option value="2">Employé</option>
        <option value="3">Vétérinaire</option>
    </select>
</div>
<!-- Champ caché pour le token CSRF --&gt;
&lt;input type="hidden" name="csrf_token" value="<?= $_SESSION['csrf_token']; ?&gt;"&gt;
&lt;button type="submit" name="add_user" class="btn btn-outline-light mt-2 btn-lg"&gt;Créer le compte&lt;/button&gt;</pre>
```

Le système vérifie la validité du rôle sélectionné et s'assure que les champs ne sont pas vides. Si toutes les validations sont réussies, un nouvel enregistrement est créé dans la base de données avec l'email, le mot de passe haché, et le rôle de l'utilisateur.

Une fois l'insertion effectuée, un email de confirmation est envoyé à l'adresse de l'utilisateur via la bibliothèque PHPMailer, qui utilise des paramètres SMTP sécurisés pour assurer la confidentialité et l'intégrité du message depuis le fichier [send_email.php](#). Ce processus garantit que les données sont traitées de manière sécurisée et que l'utilisateur est correctement informé de la création de son compte.

En cas d'erreur lors de l'insertion en base de données ou de l'envoi de l'email, un message d'erreur est affiché. En sortie, un message de succès est présenté à l'administrateur, confirmant que le compte a été créé et que l'email de confirmation a été envoyé.

```

// Vérifier que le rôle sélectionné est valide (employé ou vétérinaire)
if (($role == 2 || $role == 3) && (!empty($email)) && (!empty($password))) {
    // Insérer le nouvel utilisateur
    $sql = 'INSERT INTO users (email, password, role_id) VALUES (?, ?, ?)';
    try {
        $stmt = $pdo->prepare($sql);
        $stmt->execute([$email, $password, $role]);
        $_SESSION['message'] = "Compte utilisateur créé avec succès. Mail de confirmation envoyé.";
        // Envoyer mail de confirmation
        sendEmail($email);
    }
}

```

Pour le formulaire de contact, les données en entrée comprennent le titre, la description, l'email de l'utilisateur, et un jeton CSRF pour la sécurité.

Ces données sont utilisées pour composer et envoyer un email via PHPMailer, avec les paramètres SMTP configurés pour Gmail. En cas de succès, un message de confirmation est affiché à l'utilisateur pour indiquer que le message a été envoyé correctement. Si une erreur survient, un message d'erreur est généré et affiché, fourniissant des informations sur le problème rencontré. La page est stylisée depuis le fichier **style.css**.

```

/* Page contact */
.form-style {
    padding-top: 7%;
}
.form-style .form-control {
    background-color: #rgb(255, 255, 255, 0.8);
}
.form-style {
    color: #rgb(255, 255, 255, 0.8);
}

```

viii. Rapports vétérinaire

L'application comprend un espace vétérinaire où celui-ci va donner les directives concernant la nourriture et la quantité à donner aux animaux (Figure 20). Ces rapports pourront ensuite être consultés par l'administrateur avec un système de filtre par animal et/ou par date.

Pour la création de comptes rendus vétérinaires, un formulaire **Bootstrap** permet au vétérinaire de soumettre des informations sur l'état d'un animal. Les données collectées incluent l'état général de l'animal via un **<select>**, la nourriture à lui donner, la quantité de nourriture, la date de passage, un éventuel détail sur l'état de l'animal, l'animal concerné via un **<select>**, et un jeton CSRF pour la sécurité.

Ensuite, les données sont filtrées et validées, telles que la nourriture (pour éviter les caractères spéciaux), la quantité de nourriture (pour vérifier qu'il s'agit d'un entier valide), et l'identifiant de l'animal.

Si toutes les informations sont valides, elles sont insérées dans la base de données pour créer un nouveau compte rendu. Selon le succès ou l'échec de l'opération, un message est affiché à l'utilisateur.

Pour l'affichage et le filtrage des comptes rendus, l'administrateur a accès depuis son interface aux comptes rendus soumis par les vétérinaires. Un formulaire de filtrage permet de trier les comptes rendus par animal et/ou par date de passage.

```
<label for="animal_id" class="form-label text-light">Filtrer par Animal :</label>
<select name="animal_id" id="animal_id" class="form-select">
    <option value="">Tous les animaux</option>
    <!-- Option pour chaque animal -->
    <?php foreach ($animals_filter as $animal_filter) { ?>
        <option value=<?= $animal_filter['id'] ?><?= ($_GET['animal_id'] ?? '') == $animal_filter['id'] ? 'selected' : '' ?>>
            <?= htmlspecialchars($animal_filter['name']) ?>
        </option>
    <?php } ?>
</select>
</div>
<div class="col-12 col-lg-3">
    <label for="report_date" class="form-label text-light">Filtrer par Date :</label>
    <input type="date" name="report_date" id="report_date" class="form-control" value=<?= $_GET['report_date'] ?? '' ?>>
</div>
<!-- Champ caché pour le token CSRF -->
<input type="hidden" name="csrf_token" value=<?= $_SESSION['csrf_token']; ?>>
<div class="col-3 align-self-end">
    <button type="submit" class="btn btn-primary">Filtrer</button>
</div>
```

Le code gère ces filtres en construisant dynamiquement des clauses SQL en fonction des choix de l'utilisateur.

```
// Initialisation variables pour filtres
$animalFilter = '';
$dateFilter = '';

// Ajoute une clause SQL en fonction de l'animal sélectionné
if(isset($_GET['animal_id']) && is_numeric($_GET['animal_id'])) {
    $animalFilter = 'AND reports.animal_id = :animal_id';
}

// Ajoute une clause SQL en fonction de la date de passage
if(isset($_GET['report_date']) && !empty($_GET['report_date'])) {
    $dateFilter = 'AND DATE(reports.passage) = :report_date';
}

$sql = "SELECT reports.id AS report_id, reports.state, reports.food, reports.food_weight, reports.passage, reports.detail,
        animals.name AS animal_name
    FROM reports
    JOIN animals ON reports.animal_id = animals.id
    WHERE 1=1
        $animalFilter
        $dateFilter
    ORDER BY reports.passage DESC";
```

Les comptes rendus correspondants sont récupérés depuis la base de données et affichés sous forme de tableau responsive **Bootstrap**.

```
<div class="table-responsive">
    <table class="table table-striped table-hover">
        <thead>
            <tr>
                <th>#</th>
                <th>Date de Passage</th>
                <th>Nom de l'Animal</th>
                <th>État</th>
                <th>Nourriture</th>
                <th>Poids de la Nourriture</th>
                <th>Détail</th>
            </tr>
        </thead>
        <tbody>
            <?php foreach ($reports as $index => $row) { ?>
                <tr>
                    <td><?= $index + 1 ?></td>
                    <td><?= htmlspecialchars($row['passage']) ?></td>
                    <td><?= htmlspecialchars($row['animal_name']) ?></td>
                    <td><?= htmlspecialchars($row['state']) ?></td>
                    <td><?= htmlspecialchars($row['food']) ?></td>
                    <td><?= htmlspecialchars($row['food_weight']) ?></td>
                    <td><?= htmlspecialchars_decode($row['detail'], ENT_QUOTES) ?></td>
                </tr>
            <?php } ?>
        </tbody>
    </table>
```

Chaque ligne du tableau contient des informations telles que la date de passage, le nom de l'animal, son état, la nourriture donnée, la quantité, et les détails supplémentaires éventuels.

ix. Déploiement en ligne

Pour le déploiement en ligne j'ai choisi la plateforme **Heroku** car j'ai bénéficié d'un cours sur celle-ci durant ma formation. J'ai commencé par créer un compte **Heroku**, configuré l'authentification à deux facteurs via l'application **Authenticator**, puis téléchargé et installé **Heroku CLI**. **Git** était déjà installé sur mon système, tout comme **MySQL**, **PHP** et **Composer**. J'ai ensuite utilisé le terminal et tapé la commande :

```
$ heroku login
heroku: Press any key to open up the browser to login or q to exit
  >  Warning: If browser does not open, visit
  >  https://cli-auth.heroku.com/auth/browser/***
heroku: Waiting for login...
Logging in... done
Logged in as me@example.com
```

Une page s'est ouverte et j'ai entré mes identifiants. J'ai ensuite navigué vers le répertoire de mon projet puis j'ai créé l'application avec la ligne de commande :

```
$ heroku create
Creating app... done, ⚡ sharp-rain-871
https://sharp-rain-871.herokuapp.com/ | https://git.heroku.com/sharp-rain-871.git
```

Cette commande configure un dépôt **Git** distant sur **Heroku**. Des **add-ons** sont nécessaires pour les bases de données, pour celà j'ai navigué dans la partie **Elements** puis **Add-ons** sur l'application web **Heroku**, j'ai installé **JawsDB Maria** pour ma base de données **SQL** et **ObjectRocket for MongoDB** pour ma base de données **NoSQL**. La commande suivante permet d'ajouter l'add-on **JawsDB Maria** à l'application :

```
$ heroku addons:create jawsdb-maria
----> Adding jawsdb-maria to sharp-mountain-4005... done, v18 (free)
```

J'ai ensuite migré les données présentes depuis mon fichier **zoo_arcadia.sql**, fichier importé préalablement depuis ma base de données **MariaDB** via **PhpMyAdmin**.

```
mysql -h hostname -u username -p password database < backup.sql
```

J'ai remplacé les parties **hostname**, **username**, **password**, **database** par les informations contenues dans le dashboard de l'add-on **JawsDBMaria** et la partie **backup.sql** par mon fichier **zoo_arcadia.sql**.

Property	Value
Host	q0h7yf5pynyqa54.cbetxkdyhwsb.us-east-1.rds.amazonaws.com
Username	fhxzv713j9wu956k
Password	kkf0mzq2v99wybqx
Port	3306
Database	nn6r8thfk7a2vil

Une condition a ensuite été ajoutée dans mon fichier **pdo.php** afin de pouvoir connecter ma base de données **SQL** en local ou en ligne avec l'ajout des variables de connexion.

```
// Heroku
if(getenv('JAWSDB_MARIA_URL') !== false) {
    $dbparts = parse_url(getenv('JAWSDB_MARIA_URL'));

    $hostname = $dbparts['host'];
    $username = $dbparts['user'];
    $password = $dbparts['pass'];
    $database = ltrim($dbparts['path'],'/');
} else {
    // Local
    $username = "administrateur_arctadia";
    $password = "Mr7aF?nsozX4";
    $database = "zoo_arctadia";
    $hostname = "localhost";
}
```

Pour l'add-on **ObjectRocket for MongoDB** j'ai utilisé la commande suivante :

```
$ heroku addons:create ormongo:5-mmap
Creating ormongo-infinite-92036... done, (free)
Adding ormongo-infinite-92036 to serene-brushlands-93817... done
Setting ORMONGO_URL and restarting serene-brushlands-93817... done, v13
Use `heroku addons:docs ormongo` to view documentation.
```

Puis cette commande pour ouvrir le dashboard:

```
$ heroku addons:open ormongo
Opening https://addons-sso.herokuapp.com/apps/serene-brushlands-93817/addons/fdb06610-23a3-4
```

Le dashboard ouvert, j'ai créé une base de données ainsi qu'un utilisateur et un mot de passe :

Add Database to test

Database Name

Username

Password

Add Database **Cancel**

Comme pour l'add-on précédent, j'ai ajouté une condition permettant de connecter ma base de données **NoSQL** en local ou en ligne avec l'ajout d'une variable de connexion :

```
if(getenv('ORMONGO_URL') !== false) {
    $connect = "mongodb://administrateur_arcadia:Mr7aF?nsozX4@iad2-c18-0.mongo.objectrocket.
com:52011,iad2-c18-1.mongo.objectrocket.com:52011,iad2-c18-2.mongo.objectrocket.com:52011/
zoo_arcadia?replicaSet=3ca8fb33ce9646b19289adf77e800551";
} else {
    // Local
    $connect = "mongodb://localhost:27017";
}
```

Un fichier **Procfile** a été ajouté à la racine du projet, spécifiant à Heroku d'utiliser PHP et Apache pour démarrer l'application :

```
Procfile
1 web: heroku-php-apache2
```

J'ai ensuite ajouté tous les fichiers du projet au dépôt, à l'exception du dossier **vendor**, mentionné dans un fichier **.gitignore**.

```
git add .
git commit -m "Initial commit"
```

Enfin, j'ai poussé l'application vers Heroku avec la commande appropriée, permettant à Heroku de construire et déployer automatiquement mon application en ligne :

```
$ git push heroku main
```

Conclusion

Le projet Zoo Arcadia a été un défi ambitieux visant à développer une application web qui répond aux besoins variés des visiteurs, administrateur, employés, et vétérinaires du zoo. En organisant méthodiquement les tâches à l'aide de Trello et en utilisant un ensemble d'outils technologiques adaptés, j'ai réussi à créer une application fonctionnelle.

Le projet a permis la mise en place de fonctionnalités essentielles telles qu'un système d'authentification et de gestion des rôles, une interface intuitive pour la gestion des animaux et des habitats, ainsi qu'un espace sécurisé pour les professionnels du zoo. La réalisation de ces objectifs a non seulement contribué à améliorer l'image de marque du zoo Arcadia, mais a également permis d'intégrer des valeurs d'écologie et de bien-être animal dans l'expérience utilisateur.

Tout au long de ce projet, j'ai acquis des compétences précieuses en gestion de temps et de stress, ainsi qu'une meilleure compréhension des concepts vus en cours. La création de cette application, bien plus complexe que mes précédents projets, m'a permis de réaliser l'importance de travailler sur des projets complets pour véritablement consolider mes connaissances.

Je suis satisfait du résultat final, le projet étant pleinement fonctionnel et conforme aux exigences initiales. Fort de cette expérience, je me sens prêt à relever de nouveaux défis, notamment en explorant d'autres langages de programmation pour continuer à progresser et à diversifier mes compétences techniques.

Annexe

Figure 1 Diagramme de classes

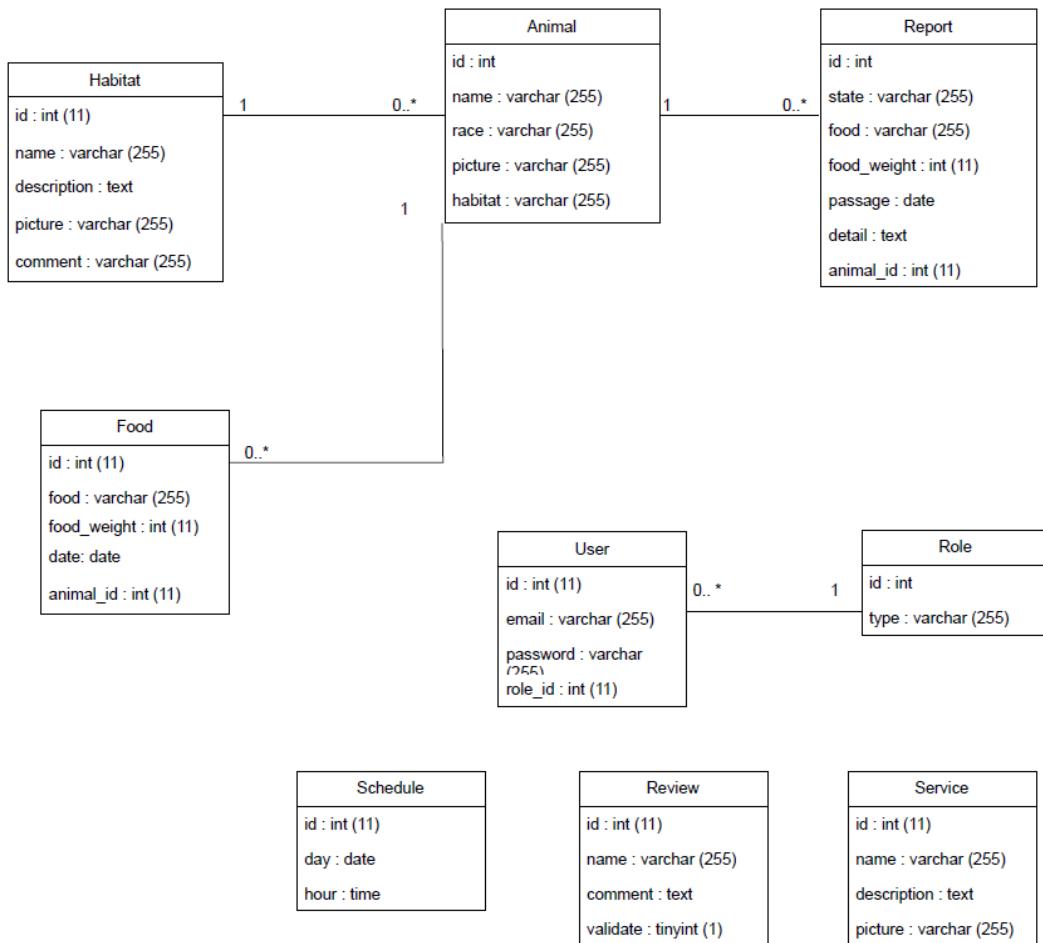


Figure 2 Diagramme de cas d'utilisation

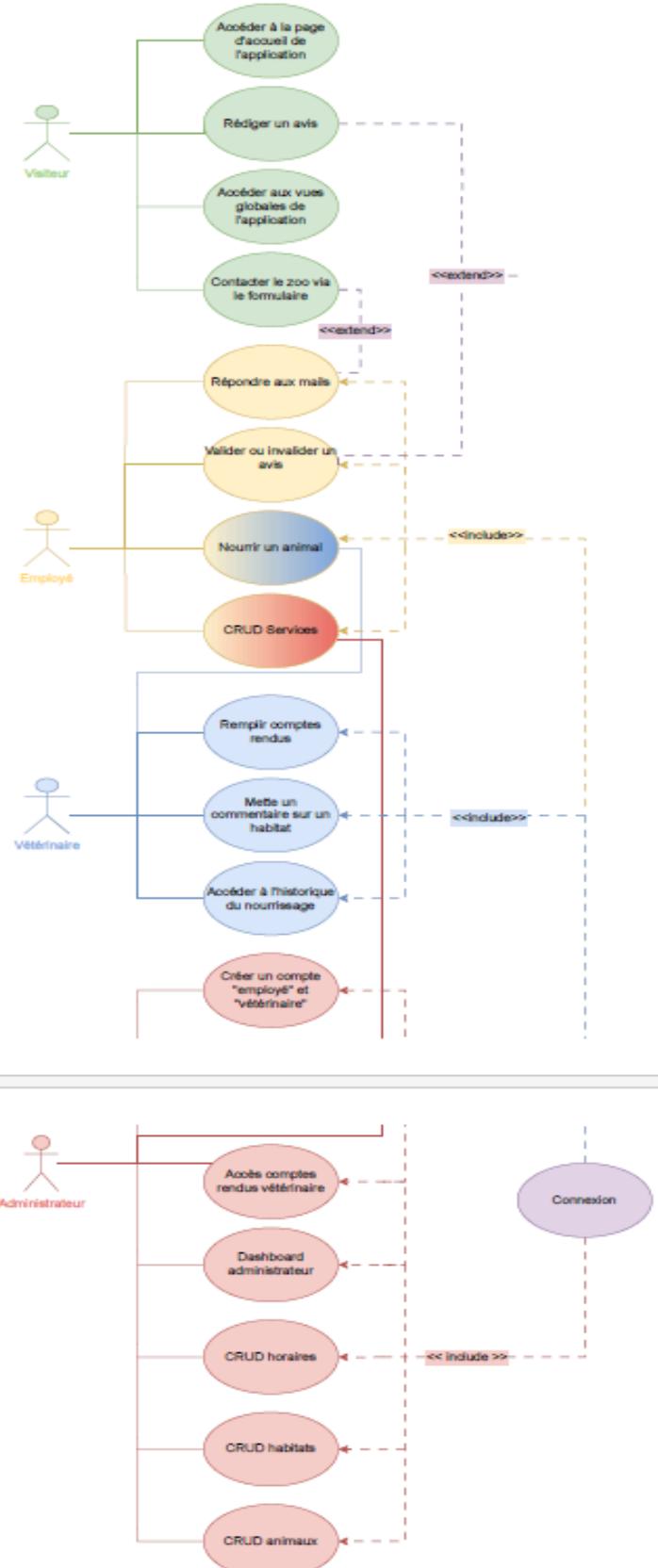


Figure 3 Diagramme de séquence

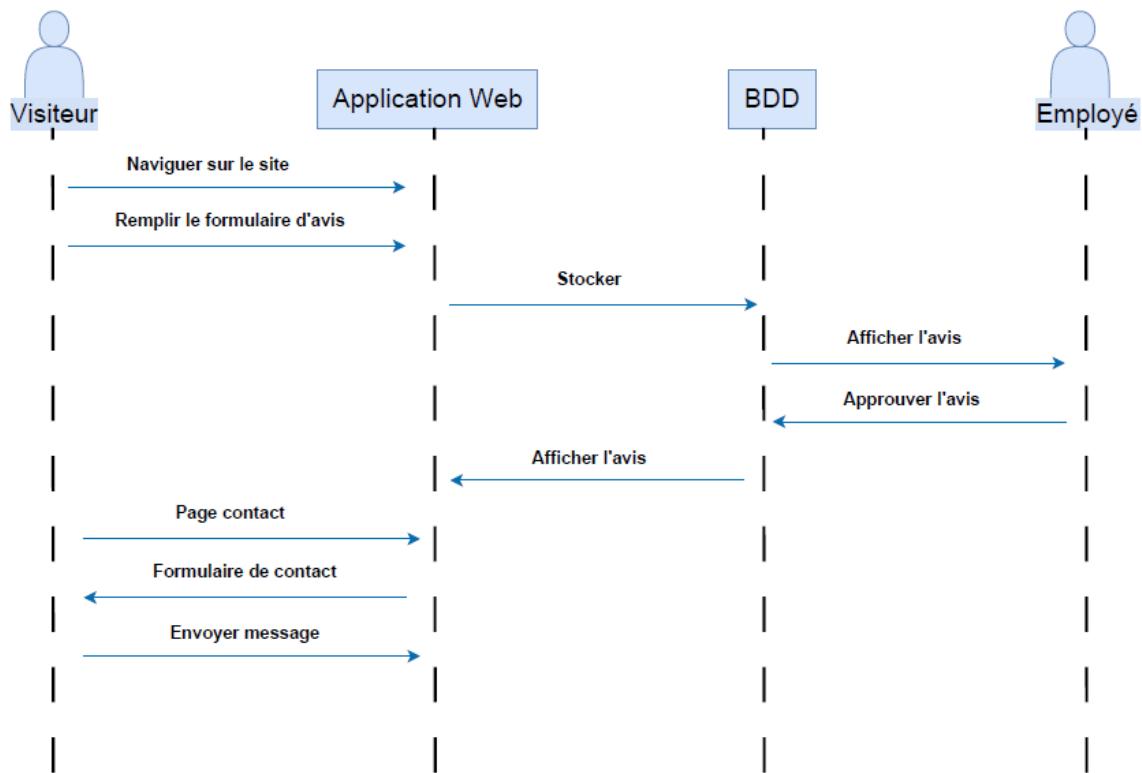


Figure 4 Charte graphique

ZOO ARCADIA - Charte graphique



Codes couleurs :



#12372A

rgb(18, 55, 42)



#436850

rgb(67, 104, 80)



#ADBC9F

rgb(173, 188, 159)



#FBFADA

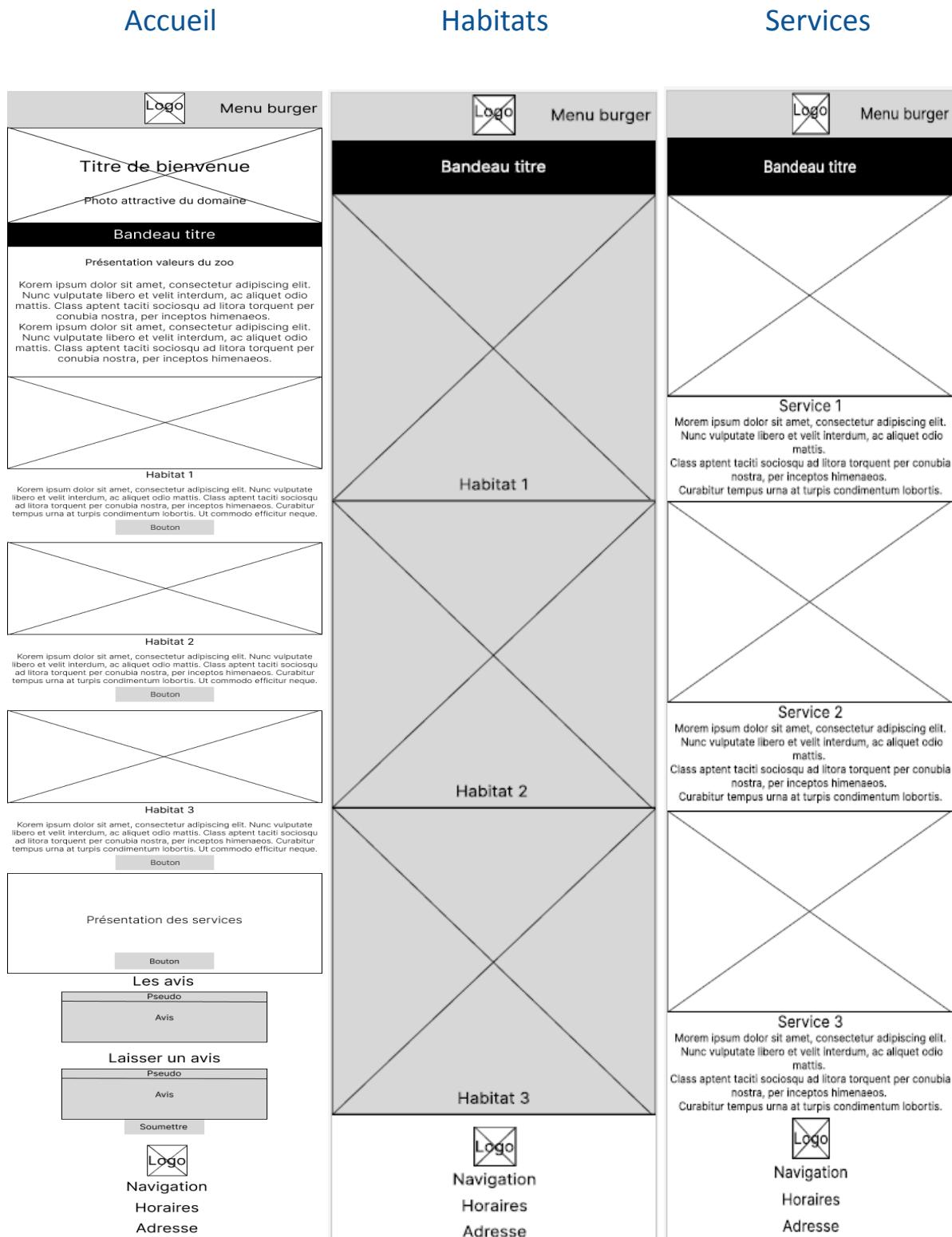
rgb(251, 250, 218)

Police d'écriture : *Arial, Helvetica, sans-serif;*

Logo :



Figure 5 Wireframes Mobiles



Wireframes Bureautique

Figure 6 Accueil

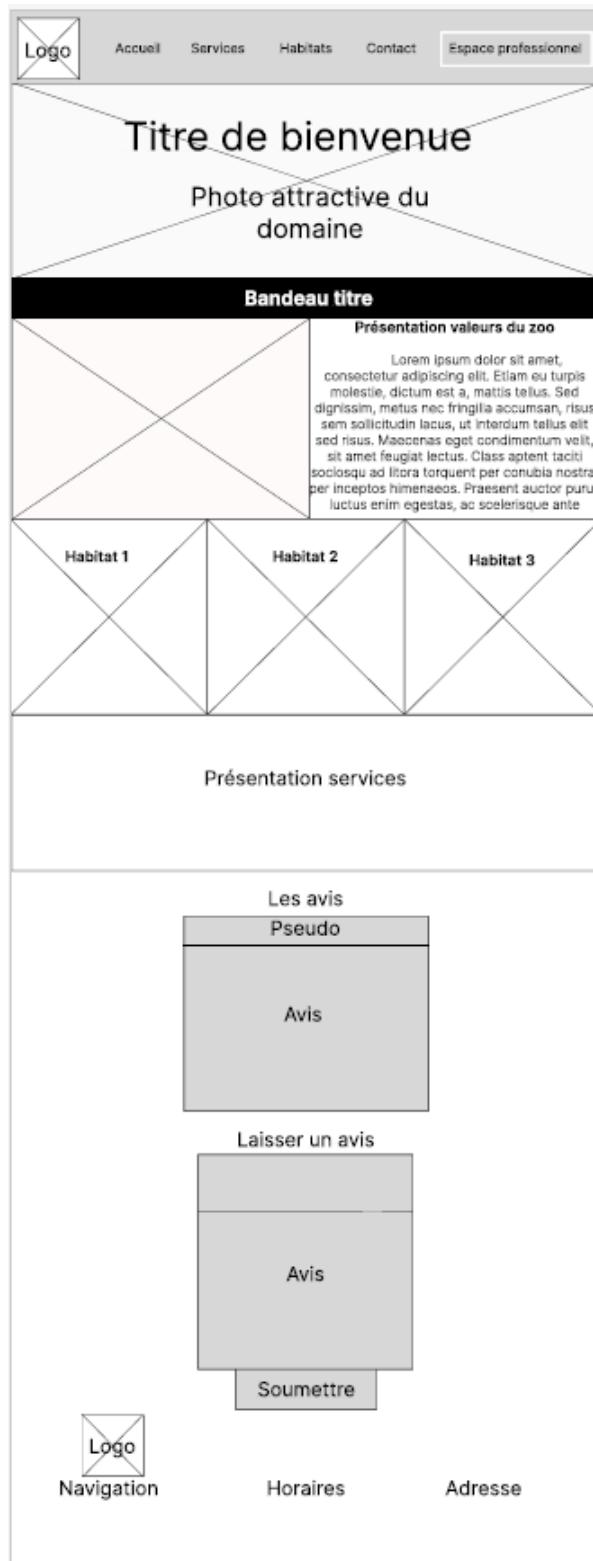


Figure 7 Habitats

The diagram illustrates three website layout variations (Habitat 1, Habitat 2, and Habitat 3) based on a template structure. The template includes a header with a logo, navigation links (Accueil, Services, Habitats, Contact, Espace professionnel), a black banner for the title, and a footer with a logo, navigation links (Navigation, Horaires, Adresse), and a copyright notice.

Habitat 1: A large rectangular container with a diagonal cross. Inside, the text "En savoir plus" is centered.

Habitat 2: A large rectangular container with a diagonal cross. Inside, the text "En savoir plus" is centered.

Habitat 3: A large rectangular container with a diagonal cross. Inside, the text "En savoir plus" is centered.

Header:

- Logo
- Accueil
- Services
- Habitats
- Contact
- Espace professionnel

Bandeau titre

Habitat 1

En savoir plus

Habitat 2

En savoir plus

Habitat 3

En savoir plus

Footer:

- Logo
- Navigation
- Horaires
- Adresse

Copyright © 2012

Figure 8 Services

The wireframe shows a website layout for 'Services'. At the top is a header bar with a logo icon and five menu items: Accueil, Services, Habitats, Contact, and Espace professionnel. Below the header are three large rectangular boxes, each containing a large 'X' and a title: 'Service 1', 'Service 2', and 'Service 3'. Each service box contains placeholder text. At the bottom is a footer bar with a logo icon and three links: Navigation, Horaires, and Adresse.

Logo

Accueil Services Habitats Contact Espace professionnel

Service 1

Dorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus.
Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus.
Maecenas eget condimentum velit, sit amet feugiat lectus.
Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

Service 2

Dorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus.
Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus.
Maecenas eget condimentum velit, sit amet feugiat lectus.
Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

Service 3

Dorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus.
Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus.
Maecenas eget condimentum velit, sit amet feugiat lectus.
Class aptent taciti sociosqu ad litora torquent per conubia nostra.

Logo

Navigation Horaires Adresse

Mockups mobiles

Figure 9 Accueil

ZOO ARCADIA

Bienvenue au Zoo Arcadia

Venez découvrir un lieu aux valeurs écologiques et humaines

Un zoo fort de 60 années d'expérience

Niché au cœur de la majestueuse forêt de Brocéliande en Bretagne depuis 1960, le zoo Arcadia est bien plus qu'un simple sanctuaire animalier. C'est un havre de paix où la faune prospère dans des habitats soigneusement reconstitués, de la savane à la jungle luxuriante en passant par le marais humide. Vous retrouverez dans ces habitats bon nombre de petits et gros pensionnaires dont le bien-être est notre priorité.

Savane

Bienvenue dans l'habitat savane de notre zoo, une vaste étendue qui capture l'essence des plaines africaines. Ici, les lions rognent en maîtres, les élégantes girafes se promènent gracieusement et les majestueux éléphants complètent ce tableau.

Je découvre la savane !

Marais

Bienvenue dans l'habitat marais de notre zoo, un écosystème fascinant où l'eau et la terre se rencontrent pour créer un refuge unique. Les imposants alligators, les flamboyants flamants roses et les robustes buffles peuplent cet habitat.

Je découvre le marais !

Jungle

Bienvenue dans l'habitat jungle de notre zoo, une oasis luxuriante où la nature s'étend dans toute sa splendeur. Ici, vous pouvez admirer les majestueux gorilles, plus loin, les adorables koalas et enfin, les puissants tigres.

Je découvre la jungle !

Retrouvez nos services

Notre zoo possède tous les services nécessaires pour une journée réussie, retrouvez entre autre :

- Restauration**
Venez goûter aux spécialités bretonnes dans l'un de nos trois restaurants !
- Visite guidée**
Un guide vous fera visiter gratuitement les différents habitats du parc !
- Tour du zoo en petit train**
Si vous aussi vous adorez les balades en petit train ne ratez pas le départ !

Voir tous les services

Marie Dupont

"Nous avons passé une super journée au zoo ! Les habitats des animaux sont spacieux et bien entretenus, ce qui est très rassurant. Mes enfants ont adoré observer les lions et les éléphants, et nous avons particulièrement apprécié le spectacle des oiseaux. Le personnel était très accueillant et informatif. La propreté du parc et la qualité des installations étaient impressionnantes. Nous reviendrons certainement pour une prochaine visite !"

Laisser un avis

Nom
Ecrivez votre pseudo ici

Avis
Ecrivez votre texte ici

Soumettre

ZOO ARCADIA

Accueil **Contact**
Habitats **Laisser un avis**
Services **Espace pro**

HORAIRES D'OUVERTURE
Du lundi au dimanche de 10h à 18h

ADRESSE
1 chemin du roi Saint-Judicaël
35380 Palmpont
Bretagne, France

© 2024 Zoo Arcadia - Tous droits réservés.
Mentions légales | Politique de confidentialité
Conditions d'utilisation

Figure 10 Habitats



Figure 11 Services

Restauration

Notre zoo vous propose trois délicieux restaurants pour satisfaire toutes vos envies culinaires. Au Savanna Grill, dégustez des spécialités africaines authentiques en admirant la vue sur notre habitat savane. Le Jungle Café vous invite à découvrir des saveurs exotiques tout en étant entouré de la végétation luxuriante de notre habitat jungle. Pour une pause rafraîchissante, rendez-vous au Swamp Bistro, où vous pourrez savourer des plats inspirés des marais, avec une vue imprenable sur nos flamants roses et nos alligators. Chaque restaurant offre une expérience gastronomique unique, en parfaite harmonie avec l'atmosphère naturelle de nos habitats.

Visite guidée

Profitez de notre service exclusif de visite guidée gratuite des habitats pour découvrir les merveilles de notre zoo. Nos guides experts, véritablement passionnés d'animaux, vous accompagneront à travers les différents environnements, offrant des anecdotes fascinantes et des informations enrichissantes sur les lions majestueux de la savane, les koalas adorables de la jungle, et bien plus encore. Cette expérience immersive vous permettra d'apprécier pleinement la diversité de notre faune et de comprendre l'importance de la conservation des espèces, qui est un des principaux objectifs de notre parc. Ne manquez pas cette opportunité unique d'enrichir votre visite avec des connaissances précieuses et des souvenirs inoubliables.

Tour du Zoo en petit train

Découvrez le zoo de manière confortable et accessible à tous grâce à notre petit train spécialement conçu et équipé pour accueillir les personnes à mobilité réduite (PMR) afin de garantir une plein expérience pour tout le monde. Ce service pratique vous permet de parcourir l'ensemble du parc sans effort, en profitant des commentaires instructifs et amusants de notre guide. Montez à bord et laissez-vous transporter à travers nos habitats variés, en observant les majestueux paysages de la savane, les luxuriantes forêts tropicales, et les mystérieux marais. Le tour en petit train est une manière idéale pour tous nos visiteurs, y compris les familles et les personnes âgées, de vivre une expérience enrichissante et relaxante au cœur de notre zoo.

ZOO ARCADIA

[Accueil](#) [Contact](#)
[Habitats](#) [Laisser un avis](#)
[Services](#) [Espace pro](#)

HORAIRES D'OUVERTURE

Du lundi au dimanche de 10h à 18h

ADRESSE

1 chemin du roi Saint-Judicaël
35380 Paimpont
Bretagne, France

© 2024 Zoo Arcadia - Tous droits réservés.
[Mentions légales](#) | [Politique de confidentialité](#)
[Conditions d'utilisation](#)

Mockups bureautique

Figure 12 Accueil

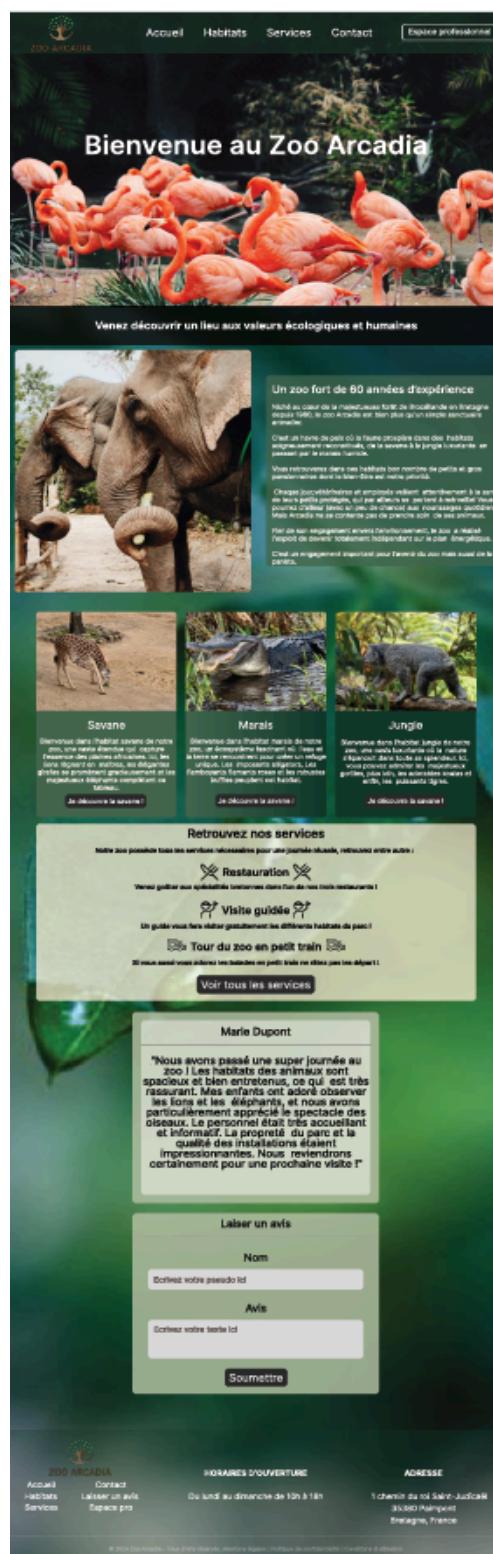


Figure 13 Habitats

ZOO ARCADIA

Accueil Habitats Services Contact Espace professionnel

Découvrez les habitats et leurs habitants

savane

marais

jungle

ZOO ARCADIA

Accueil Habitats Services

Contact Laisser un avis Espace pro

HORAIRES D'OUVERTURE

Du lundi au dimanche de 10h à 18h

ADRESSE

1 chemin du roi Saint-Judicaël
35380 Palmpont
Bretagne, France

© 2018 Zoo Arcadia - Tous droits réservés. Mentions légales | Politique de confidentialité | Conditions d'utilisation

Figure 14 Services



ZOO ARCADIA

Accueil Habitats Services Contact Espace professionnel

Découvrez tous les services proposés



Restauration

Notre zoo vous propose trois délicieux restaurants pour satisfaire toutes vos envies culinaires. Au Savanna Grill, dégustez des spécialités africaines authentiques en admirant la vue sur notre habitat savane. Le Jungle Café vous invite à découvrir des saveurs exotiques tout en étant entouré de la végétation luxuriante de notre habitat jungle. Pour une pause rafraîchissante, rendez-vous au Swamp Bistro, où vous pourrez savourer des plats inspirés des marais, avec une vue imprenable sur nos flamants roses et nos alligators. Chaque restaurant offre une expérience gastronomique unique, en parfaite harmonie avec l'atmosphère naturelle de nos habitats.



Visite guidée

Profitez de notre service exclusif de visite guidée gratuite des habitats pour découvrir les merveilles de notre zoo. Nos guides experts, véritables passionnés d'animaux, vous accompagneront à travers les différents environnements, offrant des anecdotes fascinantes et des informations enrichissantes sur les lions majestueux de la savane, les koalas adorables de la jungle, et bien plus encore. Cette expérience immersive vous permettra d'apprécier pleinement la diversité de notre faune et de comprendre l'importance de la conservation des espèces, qui est un des principaux objectifs de notre parc. Ne manquez pas cette opportunité unique d'enrichir votre visite avec des connaissances précieuses et des souvenirs inoubliables.



Tour du Zoo en petit train

Découvrez le zoo de manière confortable et accessible à tous grâce à notre petit train spécialement conçu et équipé pour accueillir les personnes à mobilité réduite (PMR) afin de garantir une pleine expérience pour tout le monde. Ce service pratique vous permet de parcourir l'ensemble du parc sans effort, en profitant des commentaires instructifs et amusants de notre guide. Montez à bord et laissez-vous transporter à travers nos habitats variés, en observant les majestueux paysages de la savane, les luxuriantes forêts tropicales, et les mystérieux marais. Le tour en petit train est une manière idéale pour tous nos visiteurs, y compris les familles et les personnes âgées, de vivre une expérience enrichissante et relaxante au cœur de notre zoo.



HORAIRES D'OUVERTURE

Du lundi au dimanche de 10h à 18h

ADRESSE

1 chemin du roi Saint-Judicaël
35380 Paimpont
Bretagne, France

© 2024 Zoo Arcadia - Tous droits réservés. Mentions légales | Politique de confidentialité | Conditions d'utilisation

Figure 15 Login.php

```
<?php

require_once __DIR__. "/lib/session.php";
require_once __DIR__. "/lib/pdo.php";
require_once __DIR__. "/lib/user.php";

// Initialisation tableau d'erreurs
$errors = [];

if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['loginUser'])) {
    // Vérification du token CSRF
    if (isset($_POST['csrf_token']) && $_POST['csrf_token'] === $_SESSION['csrf_token']) {
        $user = verifyUserLoginPassword($pdo, $_POST['email'], $_POST['password']);

        if($user) {
            // connexion => session
            $_SESSION['user'] = $user;
            $_SESSION['user_id'] = $user['id'];
            $_SESSION['role_id'] = $user['role_id'];

            // récupérer le type de rôle
            $sql = 'SELECT type FROM roles WHERE id = ?';
            try {
                $stmt = $pdo->prepare($sql);
                $stmt->execute([$user['role_id']]);
                $role = $stmt->fetch(PDO::FETCH_ASSOC);

                if($role) {
                    $_SESSION['role'] = $role['type'];

                    // Rediriger l'utilisateur en fonction de son rôle
                    if ($role['type'] == 'administrateur') {
                        header('Location: administrateur.php');
                        exit();
                    } else if ($role['type'] == 'employe') {
                        header('Location: employe.php');
                        exit();
                    } else if ($role['type'] == 'veterinaire') {
                        header('Location: veterinaire.php');
                        exit();
                    } else {
                        // Rôle non reconnu
                        $errors[] = "Rôle utilisateur non reconnu.";
                    }
                }
            } catch (Exception $e) {
                $errors[] = " Erreur ! " . $e->getMessage();
            }
        } else {
            // affiche une erreur
            $errors[] = 'Email ou mot de passe incorrect.';
        }
    } else {
        // Token CSRF invalide
        die("Token CSRF invalide.");
    }
}

require_once __DIR__. "/templates/header.php";
```

Figure 16 send_email.php

```
<?php
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

require 'vendor/autoload.php';

function sendEmail($recipientEmail) {
    $mail = new PHPMailer(true);
    try {
        // Paramètres du serveur
        $mail->isSMTP();
        $mail->Host      = 'smtp.gmail.com';
        $mail->SMTPAuth  = true;
        $mail->Username   = 'jose555.arcadia@gmail.com';
        $mail->Password   = 'qypa zmrk nbog ytzm'; // Mot de passe d'application
        $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
        $mail->Port       = 587;

        // Expéditeur / destinataire
        $mail->setFrom('jose555.arcadia@gmail.com', 'Zoo Arcadia');
        $mail->addAddress($recipientEmail);

        // Contenu de l'email
        $mail->isHTML(true);
        $mail->Subject = 'Bienvenue chez Zoo Arcadia';
        $mail->Body    = '<h1>Bienvenue!</h1><p>Votre compte a été créé avec succès.</p><p>Votre nom d\'utilisateur pour vous connecter est votre mail. Pour le mot de passe, veuillez vous rapprocher de José afin qu'il vous le communique.<br>Cordialement,<br> Zoo Arcadia.</p>';
        $mail->AltBody = 'Votre compte a été créé avec succès.';

        $mail->send();
    } catch (Exception $e) {
        echo "L'email n'a pas pu être envoyé. Erreur de PHPMailer:
        {$mail->ErrorInfo}";
    }
}
?>
```

Figure 17

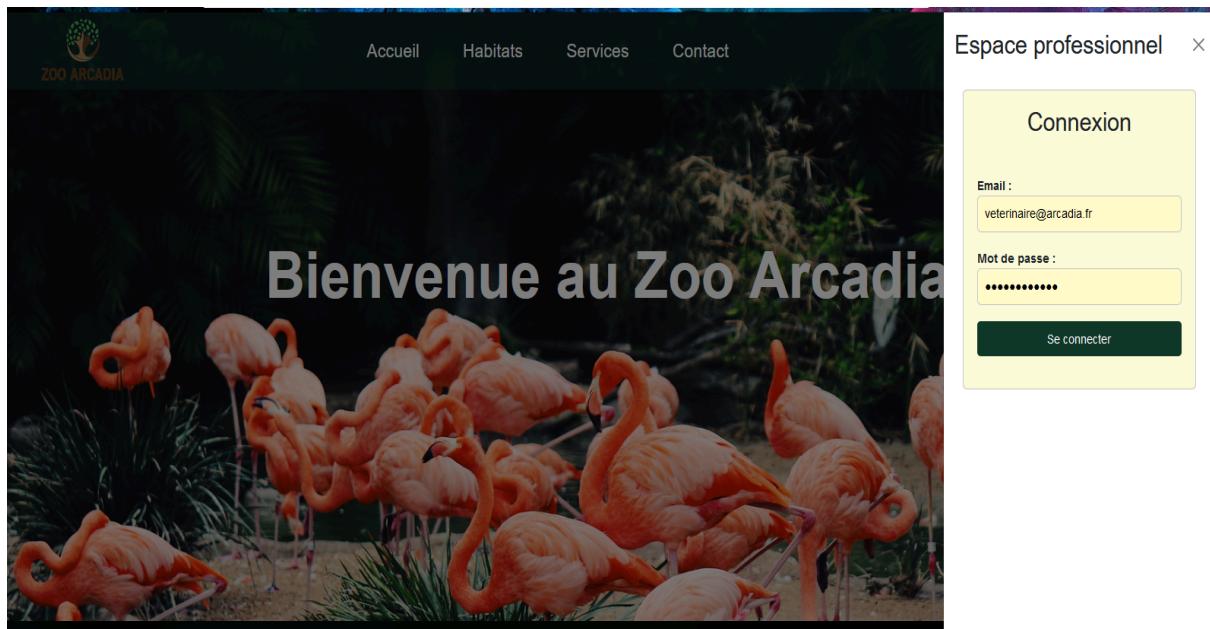


Figure 18

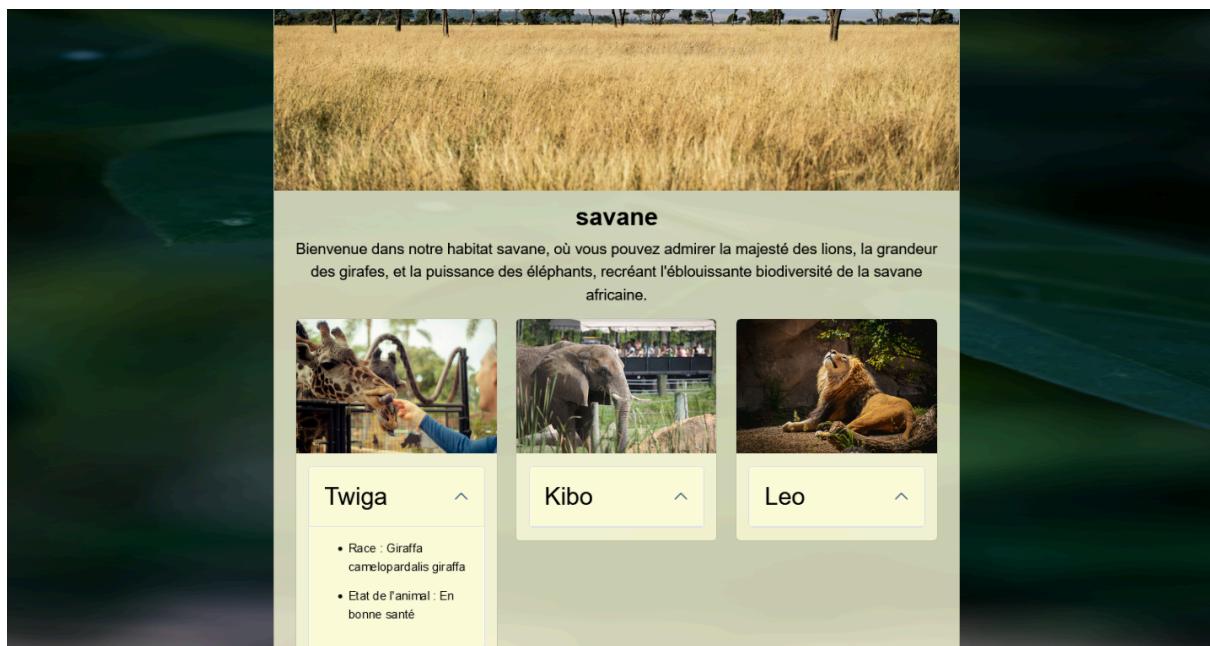


Figure 19

Tableau de bord des clics sur les animaux	
Nom de l'animal	Nombre de clics
Leo	53
Kibo	25
Goliath	21
Twiga	19
Koko	15
Gator	15
Talia	12
Billy	12
Flora	11
test	2

Figure 20

The screenshot shows a web-based application for managing animal health records. At the top, there is a navigation bar with links for Accueil, Habitats, Services, Contact, Espace vétérinaire, and Déconnexion. The main content area is titled "Suivi des animaux". It contains several input fields and dropdown menus:

- Etat général : dropdown menu showing "En bonne santé".
- Nourriture à donner : dropdown menu showing "Entrer la nourriture".
- Quantité à donner (en kilogrammes) : input field with placeholder "Entrer la quantité".
- Date de passage : date input field showing "19/06/2024".
- Animal : dropdown menu showing "Twiga".
- Détail de l'état de l'animal (facultatif) : input field showing "Facultatif".

At the bottom of the form is a button labeled "Poster compte rendu".

Figure 21

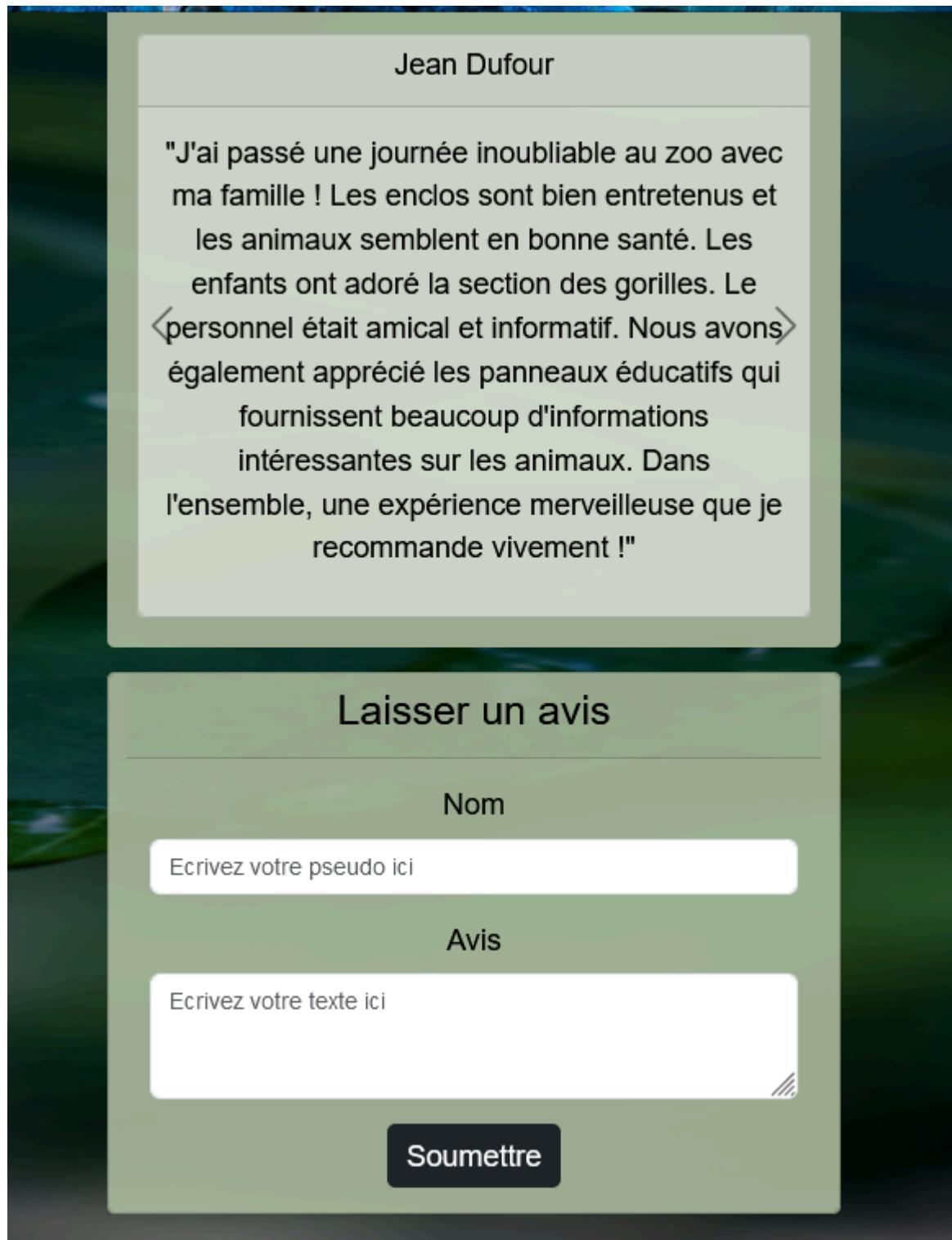


Figure 22

Ajouter un service

Nom du Service :

Description :

Image :

Parcourir... Aucun fichier sélectionné.

Ajouter le Service

Modifer / Supprimer un service



Nom du Service :

Description :

Image (laisser vide pour conserver l'actuelle) :

Parcourir... Aucun fichier sélectionné.

Mettre à jour le Service **Supprimer le Service**