École Studi **Bouzelmat Amar**

DOCUMENTATION TECHNIQUE DU PROJET ARKADIA ZOO

DIPLÔME DWWM **ANNÉE 2024-2025**

Sommaire

Introduction

Description du Projet

Le projet consiste en la création d'un site web pour Arkadia Zoo, un établissement zoologique dirigé par José, mettant l'accent sur l'écologie et le bien-être animal.

L'objectif principal est de développer une plateforme web interactive et transparente, offrant une expérience utilisateur immersive tout en promouvant les valeurs écologiques du zoo. Le site vise à informer, engager et attirer les visiteurs potentiels intéressés par une expérience zoologique éthique et écologique.

Le site web d'Arkadia Zoo se distinguera par sa transparence exceptionnelle, offrant aux visiteurs un accès détaillé aux informations sur l'état de santé et l'alimentation des animaux. Dès la page d'accueil, les utilisateurs pourront explorer visuellement les différents habitats et découvrir les animaux qui y résident.

Une attention particulière sera portée à l'expérience utilisateur, avec une navigation intuitive permettant aux visiteurs de consulter facilement l'ensemble des entités du zoo, ainsi que les différents services proposés. Chaque section sera enrichie de descriptions précises et d'images de haute qualité pour une immersion totale. Ces éléments seront rendus dynamique grâce a des technologies web moderne offrant une expérience interactive aux utilisateurs. Les visiteurs pourront ainsi explorer les habitats et les animaux de manière fluide, avec des transitions douces entres les différentes vues.

Le site intégrera également un système de retour d'expérience, permettant aux visiteurs de partager leurs avis, contribuant ainsi à l'amélioration continue des services du zoo. Un formulaire de contact sera mis à disposition pour faciliter la communication entre le public et l'établissement.

Pour répondre aux besoins de gestion interne, le site comportera trois interfaces utilisateur distinctes :

1. Une interface administrateur pour la gestion globale des utilisateurs, des entités et des services du zoo.

2.Une interface vétérinaire dédiée à la rédaction et au suivi des rapports de santé des animaux.

3.Une interface employé permettant la modération des avis visiteurs, l'enregistrement des rapports alimentaires pour chaque animal, et la mise à jour des informations sur les services du zoo.

Cette structure multi-niveaux assurera une gestion efficace et une mise à jour régulière des informations, garantissant ainsi la pertinence et l'actualité du contenu présenté aux visiteurs.

Objectif : Développer une plateforme web interactive et transparente, offrant une expérience utilisateur immersive tout en promouvant les valeurs écologiques du zoo.

Public cible : Visiteurs potentiels intéressés par une expérience zoologique éthique et écologique.

Principales fonctionnalités :

1. Interface utilisateur intuitive présentant les habitats et les animaux dès la page d'accueil.

2. Système de gestion de contenu (CMS) pour les informations détaillées sur les animaux, leur alimentation et leur état de santé.

3. Module d'avis et de retours d'expérience des visiteurs avec modération.

4. Présentation complète des services du zoo avec descriptions et images.

5.Formulaire de contact pour les visiteurs

6. Interfaces d'administration différenciées pour les gestionnaires, vétérinaires et employés du zoo.

Technologies clés :

1. Front-end : HTML, CSS, Bootstrap, JavaScript.

2. Back-end : PHP, SQL ; MongoDB Query Language (MQL)

3.Base de données : MYSQL et MongoDB (NoSQL)

Pourquoi avoir utilisé ces technologies ?

Front-end :

.HTML :
langage standard pour la structure des pages web, assurant une compatibilité universelle.

CSS :
utilisé pour le style et la mise en page, permettant une séparation claire entre le contenu et la présentation.

Bootstrap :
choisi pour sa capacité à créer rapidement des interface responsives et cohérentes. Il offre une collection de composants préconstruits, ce qui accélère considérablement le développement.

JavaScript :
L'utilisation de JavaScript permet de rendre les éléments dynamiques et interactifs, offrant une expérience immersive aux utilisateurs. Cela est particulièrement utile pour les fonctionnalités comme les galeries d'images d'animaux et des habitats, enrichissant l'exploration du zoo virtuel.

Back end

PHP :

PHP a été choisi pour sa polyvalence et sa capacité à gérer efficacement les différentes interfaces utilisateur (admin, vétérinaire, employé). Il permet de mettre en place le système de gestion de contenu (CMS) pour les informations détaillées sur les animaux, leur alimentation et leur état de santé.
Également, Sélectionné pour sa facilité d'intégration avec les bases de données.

SQL :

SQL est un langage éprouvée et largement utilisée pour la gestion des bases de données relationnelles. Il est Utilisé pour interagir efficacement avec la base de données relationnelle MySQL. Pour Arkadia zoo, cela garanti une gestion fiable et stable des données critiques, tels que les informations sur les animaux et leurs état sanitaire, les habitats, les services du zoo, et les utilisateurs.

MongoDB Query Language (MQL) :

Syntaxe JSON-like : Les requêtes MongoDB sont formulées en utilisant des documents JSON. Cela permet de structurer les requêtes de manière claire et lisible.
Le langage de MongoDB permet de manipuler des documents imbriqués et des listes, offrant une grande flexibilité pour les structures de données non structurées ou semi-structurées

Le choix des bases de données :

MYSQL :

MySQL gère la majorité des données du site, offrant une solution robuste et efficace pour stocker et récupérer les informations sur les animaux, les habitats, les services du zoo, et les utilisateurs
MYSQL est idéal pour gérer des données structurées avec des relations claires entre les différentes tables. Par exemple, les informations sur les animaux peuvent être liées a leurs habitats et à leurs régimes alimentaires, permettant une organisation et une récupération efficaces des données
MYSQL offre des fonctionnalités de sécurités robustes pour protéger les données sensibles. Cette base de donnée peut gérer un grand volume de requêtes rapidement. Offrant une indexation avancée, ainsi permettant des recherches rapides meme dans de grandes tables de données, chose essentiel pour un zoo qui peut contenir des centaines d ‘animaux.
Également, nous remarquons qu’il existe des période de pointe où le nombre de visite augmente considérablement(vacances, événements spéciaux…), pour ce faire MYSQL est capable de gérer des milliers de connexions simultanées, idéal pour ce type de période.

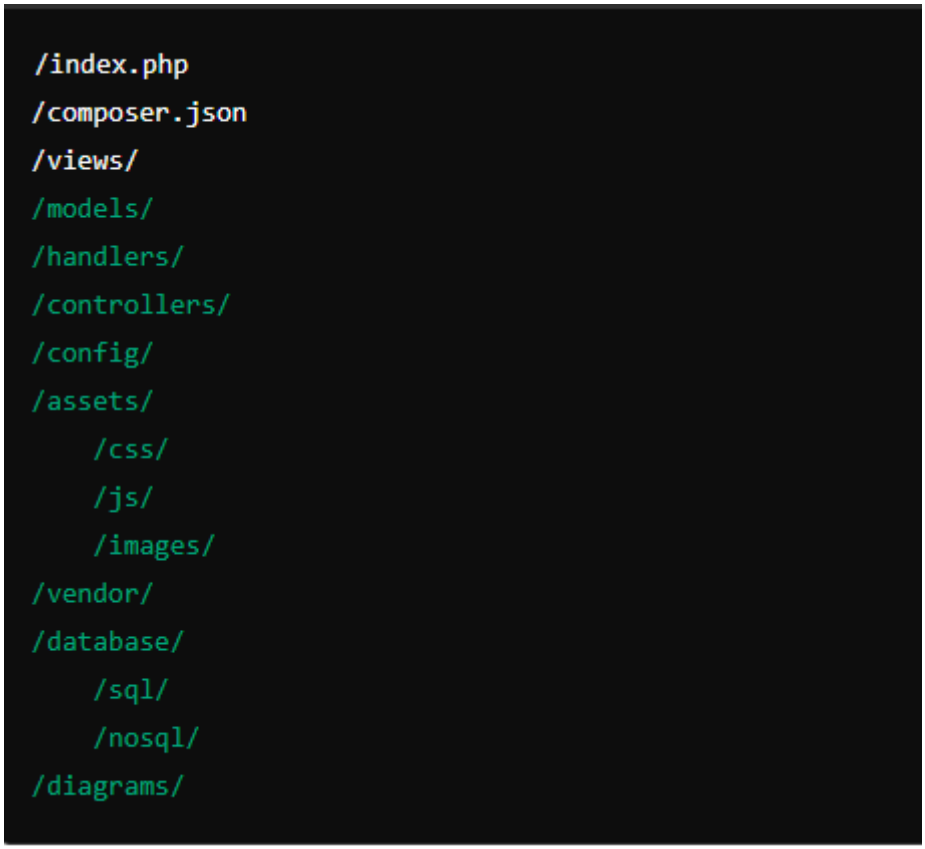
MongoDB :

Suivi des clics sur les images : Cette fonctionnalité utilise MongoDB pour sa capacité à gérer efficacement de grands volumes de données non structurées en temps réel. MongoDB offre des performances élevés pour l’écriture fréquentes de données, ce qui est crucial pour enregistrer chaque clic en temps réel. Les qualités de mongoDB permet de créer un dashboard dynamique visualisant l'engagement des utilisateurs avec le contenu visuel du site.
Gestion des événements : l’utilisation de MongoDB pour les opérations CRUD sur les événements du zoo est pertinent car le modèle de données flexibles de MongoDB permet d’adapter facilement la structure des événements sans nécessiter de modifications de schéma complexe. Les requêtes sur les documents JSON sont efficaces pour récupérer et manipuler les données d’événements. MongoDB offre une excellente performance pour les opérations de lecture et d’écriture, ce qui est important pour la gestion dynamique des événements par les employés.

En somme ces choix technologiques s'alignent parfaitement avec les objectifs du site d'Arkadia Zoo, offrant une plateforme performante interactive et transparente qui met en avant l'engagement écologique et le bien-être animal. La combinaison de ces technologies offre un équilibre entre performance, flexibilité et facilité de développement. L'utilisation de MySQL et MongoDB permet de tirer parti des avantages des bases de données relationnelles et NoSQL, adaptant le stockage des données aux besoins spécifiques de différentes parties de l’application. Cette approche hybride offre une grande souplesse pour gérer divers types de données et s'adapter aux futures évolutions du projet.

Architecture Générale

La structure des fichiers de ce projet est conçue pour optimiser la navigation, le développement et la maintenance. Voici une brève explication de l'organisation des principaux fichiers et répertoires.



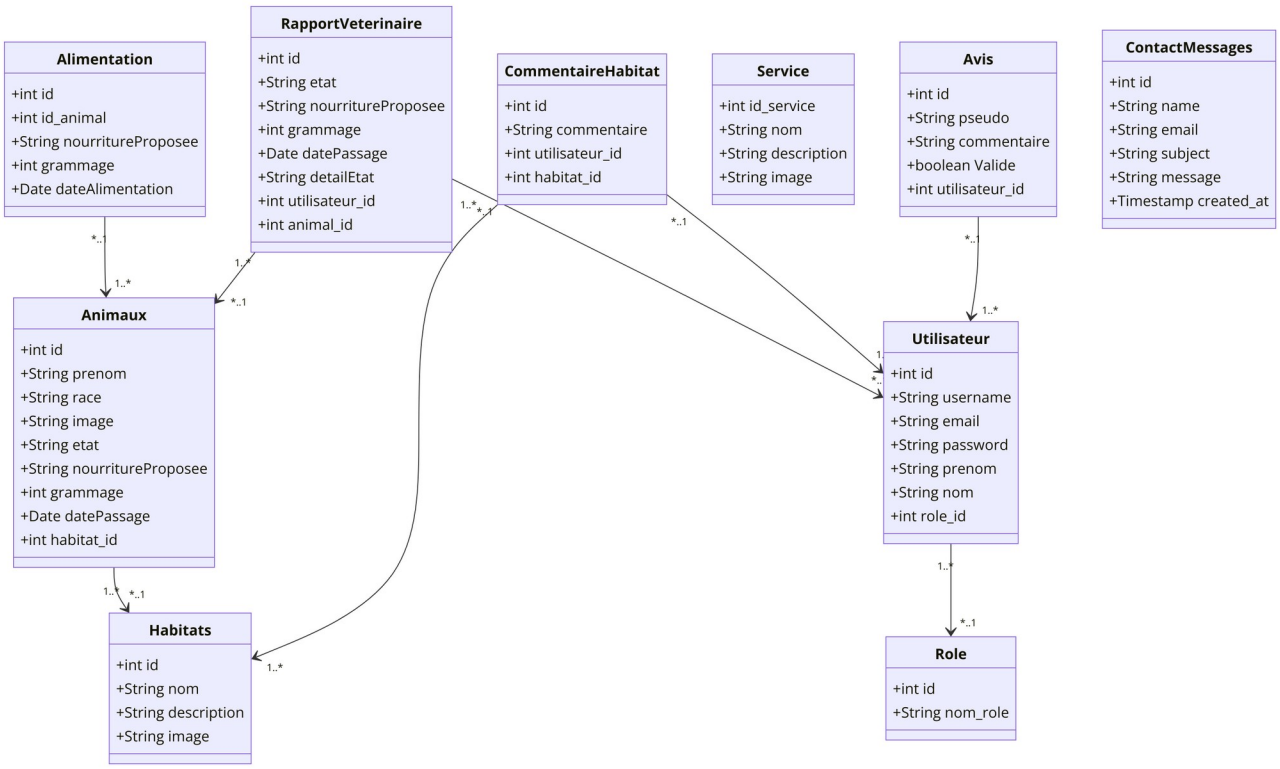
Explication des Répertoires

- **/index.php** : Point d'entrée de l'application.
- **/composer.json** : Fichier de gestion des dépendances pour Composer.
- **/views/** : Contient les fichiers de vue de l'application.
- **/models/** : Contient les modèles de données (les class)
- **/handlers/** : Contient les gestionnaires d'événements ou de requêtes.
- **/controllers/** : Contient les contrôleurs qui gèrent la logique de traitement des requêtes.
- **/config/** : Contient les fichiers de configuration (connexion à la bdd)
- **/assets/** : Contient les ressources statiques (CSS, JavaScript, images, etc.).
    - **/css/** : Feuilles de style.
    - **/js/** : Scripts JavaScript.
    - **/images/** : Images et autres fichiers multimédias.

- **/vendor/** : Contient les dépendances gérées par Composer.
- **/database/** : Contient les fichiers relatifs aux bases de données.
    - **/sql/** : Scripts et fichiers SQL.
    - **/nosql/** : Scripts et fichiers NoSQL.
- **/diagrams/** : Contient les diagrammes UML (diagrammes de classes, de séquence, de cas d'utilisation.).

Diagrammes

Diagramme de classe



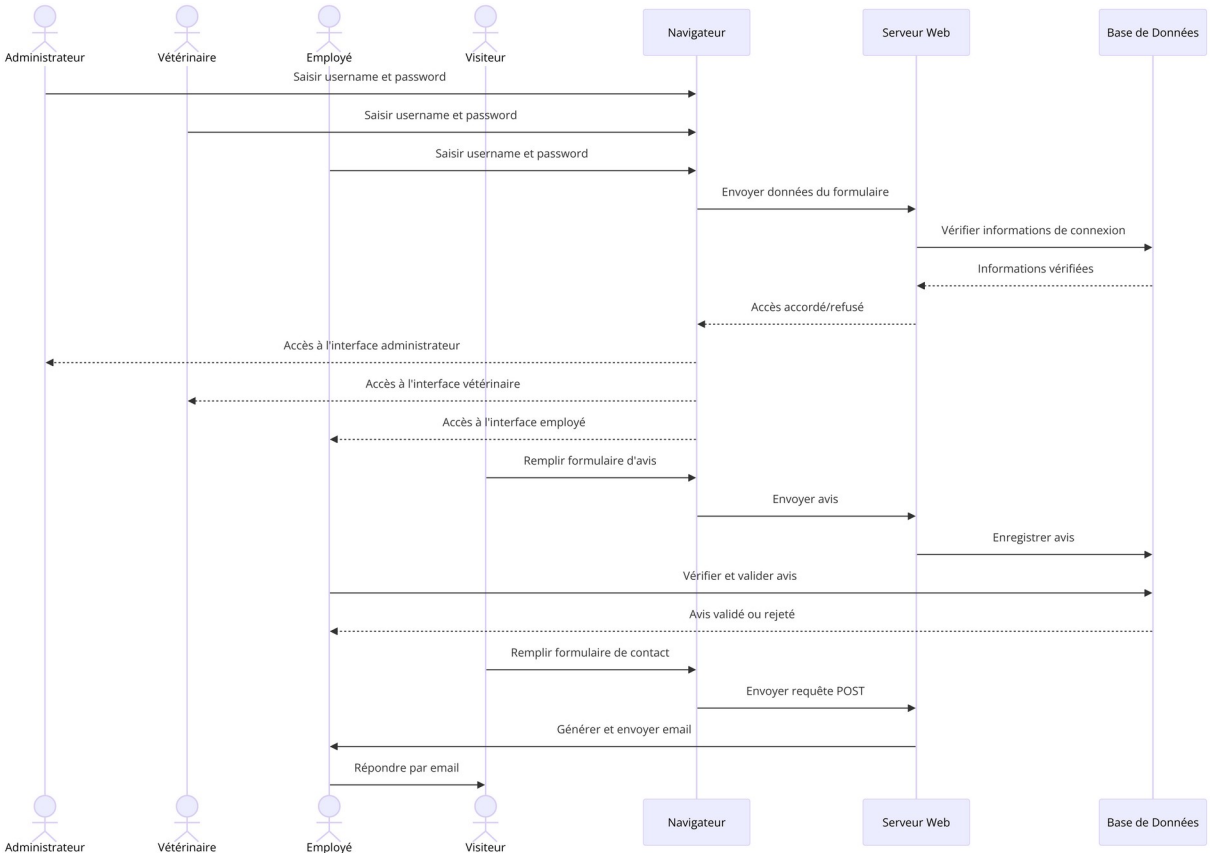
Commentaires sur le diagramme de manière générale :

1. La classe Utilisateur est centrale, avec des relations vers Role, RapportVeterinaire, CommentaireHabitat et Avis.
2. La classe Animaux est également importante, liée à Habitats, Alimentation et RapportVeterinaire.
3. Les classes Habitats et Animaux ont une relation bidirectionnelle.
4. La classe RapportVeterinaire lie les Utilisateurs (vétérinaires) aux Animaux.
5. La classe CommentaireHabitat relie les Utilisateurs aux Habitats.
6. Les classes Service et ContactMessages sont indépendantes, sans relations directes avec les autres classes.
7. La classe Avis a une relation optionnelle avec Utilisateur, permettant des avis anonymes ou associés à un utilisateur.

Voici les relations entre les classes avec leurs cardinalités :

1. Utilisateur - Role :
    - Un Utilisateur a exactement un (1) Role
    - Un Role peut être attribué à plusieurs (0..*) Utilisateurs*
      Cardinalité : 1 (Utilisateur) - 0.. (Role)
2. Animaux - Habitats :
    - Un Animal appartient à exactement un (1) Habitat
    - Un Habitat peut contenir plusieurs (0..*) Animaux*
      Cardinalité : 1 (Habitats) - 0.. (Animaux)
3. Animaux - Alimentation :
    - Un Animal peut avoir plusieurs (0..*) enregistrements d'Alimentation
    - Une Alimentation concerne exactement un (1) Animal
      Cardinalité : 1 (Animaux) - 0..* (Alimentation)
4. Animaux - RapportVeterinaire :
    - Un Animal peut avoir plusieurs (0..*) RapportVeterinaire
    - Un RapportVeterinaire concerne exactement un (1) Animal
      Cardinalité : 1 (Animaux) - 0..* (RapportVeterinaire)
5. Utilisateur - RapportVeterinaire :
    - Un Utilisateur (vétérinaire) peut rédiger plusieurs (0..*) RapportVeterinaire
    - Un RapportVeterinaire est rédigé par exactement un (1) Utilisateur
      Cardinalité : 1 (Utilisateur) - 0..* (RapportVeterinaire)
6. Utilisateur - CommentaireHabitat :
    - Un Utilisateur peut faire plusieurs (0..*) CommentaireHabitat
    - Un CommentaireHabitat est fait par exactement un (1) Utilisateur
      Cardinalité : 1 (Utilisateur) - 0..* (CommentaireHabitat)
7. Habitats - CommentaireHabitat :
    - Un Habitat peut avoir plusieurs (0..*) CommentaireHabitat
    - Un CommentaireHabitat concerne exactement un (1) Habitat
      Cardinalité : 1 (Habitats) - 0..* (CommentaireHabitat)
8. Utilisateur - Avis :
    - Un Utilisateur peut être associé à plusieurs (0..*) Avis
    - Un Avis peut être associé à au plus un (0..1) Utilisateur (car la relation est optionnelle)
      Cardinalité : 0..1 (Utilisateur) - 0..* (Avis)

Diagramme de séquence



Le diagramme de séquence représente les interactions entre les acteurs suivants : le visiteur, l'administrateur, le vétérinaire, l'employé, le serveur web et la base de données. Il illustre trois interactions principales.

1. Connexion des utilisateurs (administrateur, vétérinaire, employé)

- L'administrateur, le vétérinaire ou l'employé saisit son username et son password dans l'interface de connexion.
- Une flèche part de l'utilisateur vers le serveur web, représentant l'envoi des données du formulaire.
- Le serveur web interagit ensuite avec la base de données, symbolisé par une flèche bidirectionnelle, pour vérifier les informations de connexion.
- Selon le résultat, le serveur web renvoie l'interface personnalisée correspondant au type d'utilisateur (admin, vétérinaire ou employé).

2. Avis du visiteur

- Le visiteur remplit un formulaire d'avis sur le site web.
- Une flèche part du visiteur vers le serveur web, représentant l'envoi des données du formulaire.

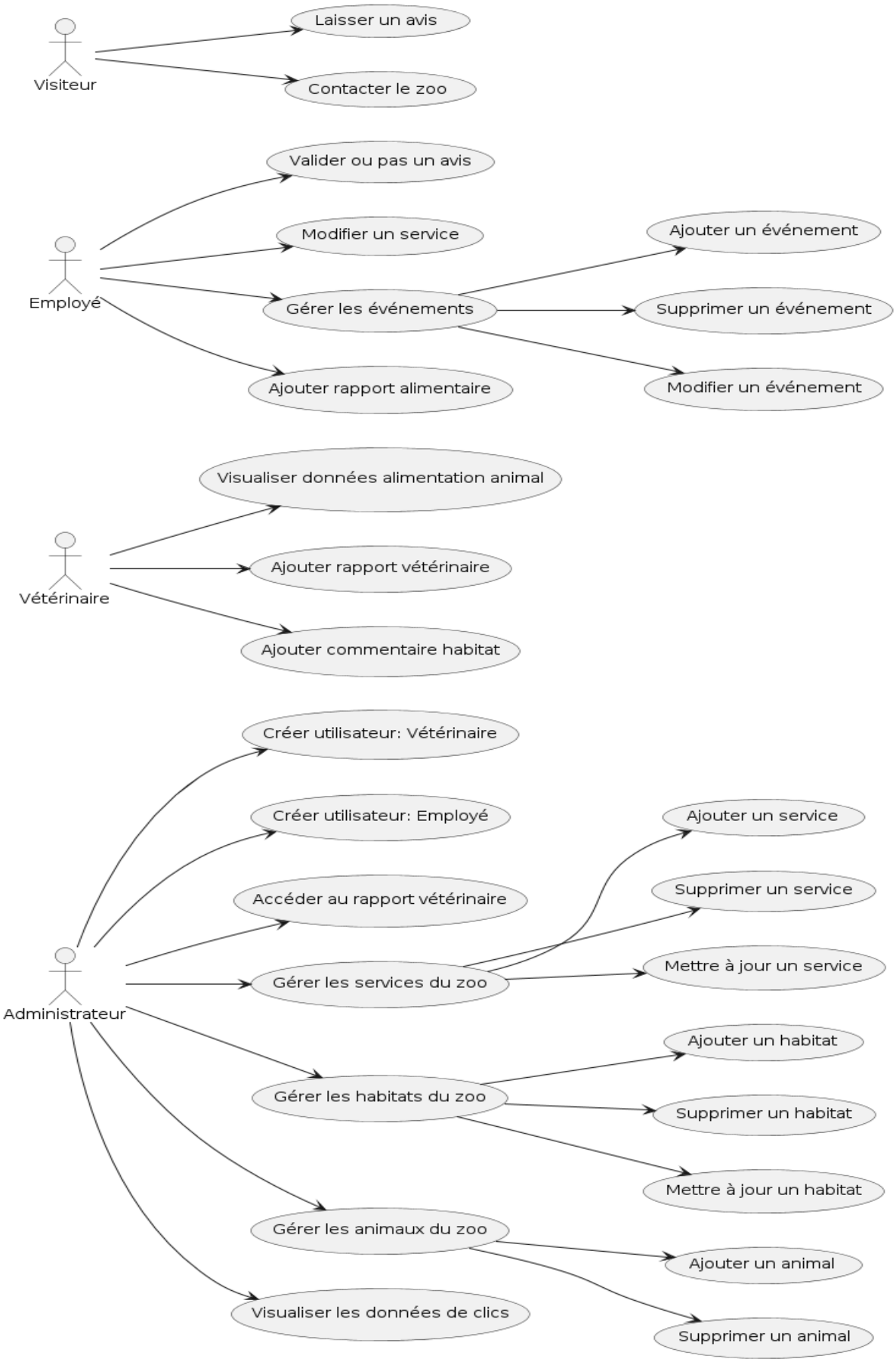
- Le serveur web interagit avec la base de données pour enregistrer l'avis, symbolisé par une flèche allant du serveur web à la base de données.
- Plus tard, une flèche part de l'employé vers le serveur web, représentant la validation ou non de l'avis.
- Le serveur web met à jour le statut de l'avis dans la base de données, symbolisé par une autre flèche.

3. Contact du visiteur

- Le visiteur remplit un formulaire de contact sur le site web.
- Une flèche part du visiteur vers le serveur web, représentant l'envoi des données du formulaire.
- Le serveur web traite les données et génère un e-mail.
- Une flèche pointillée représente l'envoi de l'e-mail du serveur web vers l'employé correspondant.
- Une autre flèche pointillée représente la réponse de l'employé au visiteur par e-mail.

Ce diagramme de séquence illustre clairement les flux d'interactions entre les différents acteurs et systèmes, mettant en évidence les processus de connexion, de gestion des avis et de communication avec les visiteurs.

Diagramme de cas d’utilisation



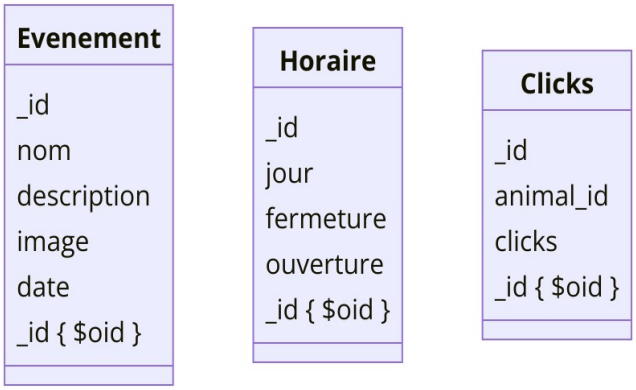
Voici une explication détaillée du diagramme de cas d'utilisation :

1. **Administrateur**
    - **Créer utilisateur: Vétérinaire** : L'administrateur peut ajouter un nouveau vétérinaire dans le système.
    - **Créer utilisateur: Employé** : L'administrateur peut ajouter un nouvel employé dans le système.
    - **Accéder au rapport vétérinaire** : L'administrateur a accès aux rapports créés par les vétérinaires.
    - **Gérer les services du zoo** : L'administrateur peut ajouter, supprimer et mettre à jour les services proposés par le zoo.
        - **Ajouter un service** : Ajout d'un nouveau service dans le zoo.
        - **Supprimer un service** : Suppression d'un service existant.
        - **Mettre à jour un service** : Mise à jour des informations sur un service existant.
    - **Gérer les habitats du zoo** : L'administrateur peut ajouter, supprimer et mettre à jour les habitats du zoo.
        - **Ajouter un habitat** : Ajout d'un nouvel habitat.
        - **Supprimer un habitat** : Suppression d'un habitat existant.
        - **Mettre à jour un habitat** : Mise à jour des informations sur un habitat existant.
    - **Gérer les animaux du zoo** : L'administrateur peut ajouter et supprimer des animaux.
        - **Ajouter un animal** : Ajout d'un nouvel animal.
        - **Supprimer un animal** : Suppression d'un animal existant.
    - **Visualiser les données de clics** : L'administrateur peut voir les données de clics pour analyser l'engagement des utilisateurs.
2. **Vétérinaire**
    - **Visualiser données alimentation animal** : Le vétérinaire peut consulter les informations sur l'alimentation des animaux.
    - **Ajouter rapport vétérinaire** : Le vétérinaire peut ajouter des rapports sur la santé des animaux.
    - **Ajouter commentaire habitat** : Le vétérinaire peut ajouter des commentaires sur les habitats des animaux.
3. **Employé**
    - **Valider ou pas un avis** : L'employé peut approuver ou rejeter des avis laissés par les visiteurs.
    - **Modifier un service** : L'employé peut modifier les informations sur les services proposés.
    - **Gérer les événements** : L'employé peut ajouter, supprimer et modifier des événements.
        - **Ajouter un événement** : Ajout d'un nouvel événement.
        - **Supprimer un événement** : Suppression d'un événement existant.

        - **Modifier un événement** : Mise à jour des informations sur un événement existant.
    - **Ajouter rapport alimentaire** : L'employé peut ajouter des rapports sur l'alimentation des animaux.
4. **Visiteur**
    - **Laisser un avis** : Les visiteurs peuvent laisser des avis sur leur expérience au zoo.
    - **Contacter le zoo** : Les visiteurs peuvent contacter le zoo pour des questions ou des commentaires.

Ce diagramme montre les interactions entre les différents acteurs (Administrateur, Vétérinaire, Employé, Visiteur) et les cas d'utilisation associés à chacun. Les administrateurs ont le plus grand nombre de responsabilités, incluant la gestion des utilisateurs, des services, des habitats et des animaux, tandis que les vétérinaires et les employés ont des responsabilités plus spécifiques. Les visiteurs ont des interactions limitées mais essentielles pour leur expérience au zoo.

Diagramme de modèle de données



1.Événements :

Cette collection contient les informations sur les différents événements organisés par le zoo, tels que le nom, la description, l'image et la date de l'événement.

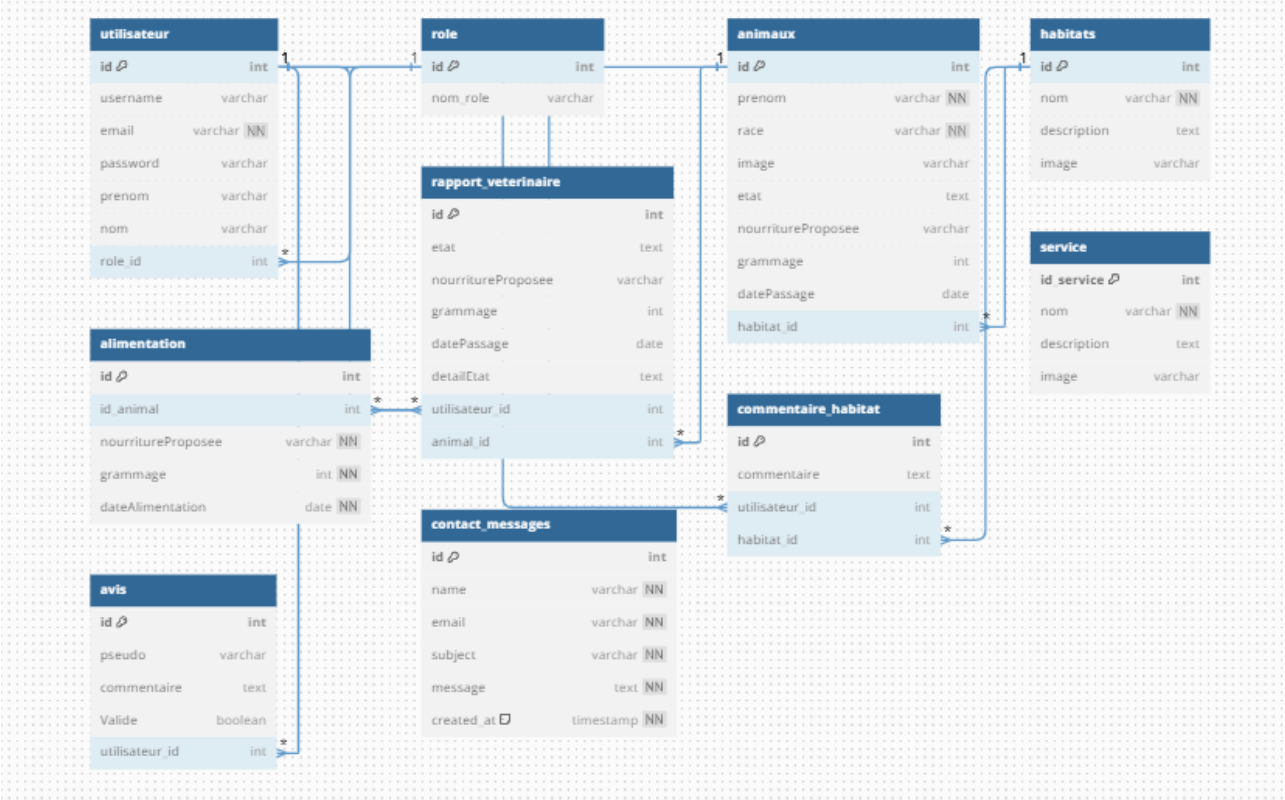
2.Horaires :

Cette collection stocke les informations sur les horaires d'ouverture et de fermeture du zoo pour chaque jour de la semaine.

3.Clics :

Cette collection enregistre le nombre de clics pour chaque animal du zoo.

Base de Données



Description :

utilisateur

- **id**: Identifiant unique de l'utilisateur (clé primaire).
- **username**: Nom d'utilisateur.
- **email**: Adresse e-mail de l'utilisateur (ne peut pas être nul).
- **password**: Mot de passe de l'utilisateur.
- **prenom**: Prénom de l'utilisateur.
- **nom**: Nom de famille de l'utilisateur.
- **role_id**: Identifiant du rôle associé à l'utilisateur (clé étrangère référencée par role.id).

- **role**

- **id**: Identifiant unique du rôle (clé primaire).
- **nom_role**: Nom du rôle.

- **animaux**

- **id**: Identifiant unique de l'animal (clé primaire).
- **prenom**: Prénom de l'animal (ne peut pas être nul).
- **race**: Race de l'animal (ne peut pas être nul).
- **image**: URL de l'image de l'animal.

- **etat**: État de l'animal.
- **nourritureProposee**: Nourriture proposée à l'animal.
- **grammage**: Quantité de nourriture en grammes.
- **datePassage**: Date du dernier passage.
- **habitat_id**: Identifiant de l'habitat de l'animal (clé étrangère référencée par habitats.id).

- **habitats**

- **id**: Identifiant unique de l'habitat (clé primaire).
- **nom**: Nom de l'habitat (ne peut pas être nul).
- **description**: Description de l'habitat.
- **image**: URL de l'image de l'habitat.

- **alimentation**

- **id**: Identifiant unique de l'entrée d'alimentation (clé primaire).
- **id_animal**: Identifiant de l'animal nourri (clé étrangère référencée par animaux.id).
- **nourritureProposee**: Nourriture donnée à l'animal (ne peut pas être nul).
- **grammage**: Quantité de nourriture en grammes (ne peut pas être nul).
- **dateAlimentation**: Date de l'alimentation (ne peut pas être nul).

- **rapport_veterinaire**

- **id**: Identifiant unique du rapport vétérinaire (clé primaire).
- **etat**: État général de l'animal.
- **nourritureProposee**: Nourriture recommandée.
- **grammage**: Quantité recommandée de nourriture en grammes.
- **datePassage**: Date de la visite vétérinaire.
- **detailEtat**: Détails supplémentaires sur l'état de l'animal.
- **utilisateur_id**: Identifiant de l'utilisateur qui a rédigé le rapport (clé étrangère référencée par utilisateur.id).
- **animal_id**: Identifiant de l'animal concerné par le rapport (clé étrangère référencée par animaux.id).

- **commentaire_habitat**

- **id**: Identifiant unique du commentaire sur l'habitat (clé primaire).
- **commentaire**: Texte du commentaire.
- **utilisateur_id**: Identifiant de l'utilisateur ayant fait le commentaire (clé étrangère référencée par utilisateur.id).
- **habitat_id**: Identifiant de l'habitat concerné par le commentaire (clé étrangère référencée par habitats.id).

- **service**

- **id_service**: Identifiant unique du service (clé primaire).
- **nom**: Nom du service (ne peut pas être nul).
- **description**: Description du service.
- **image**: URL de l'image du service.

- **avis**

- **id**: Identifiant unique de l'avis (clé primaire).
- **pseudo**: Pseudo de l'auteur de l'avis.
- **commentaire**: Texte de l'avis.
- **Valide**: Statut de validation de l'avis (0 ou 1).
- **utilisateur_id**: Identifiant de l'utilisateur ayant rédigé l'avis (clé étrangère référencée par utilisateur.id).

- **contact_messages**

- **id**: Identifiant unique du message de contact (clé primaire).
- **name**: Nom de l'expéditeur du message (ne peut pas être nul).
- **email**: Adresse e-mail de l'expéditeur (ne peut pas être nul).
- **subject**: Sujet du message (ne peut pas être nul).
- **message**: Contenu du message (ne peut pas être nul).
- **created_at**: Date et heure de création du message (défaut à l'heure actuelle).

Relations :

- **utilisateur** est lié à **role** par role_id.
- **animaux** est lié à **habitats** par habitat_id.
- **alimentation** est lié à **animaux** par id_animal.
- **rapport_veterinaire** est lié à **utilisateur** par utilisateur_id et à **animaux** par animal_id.
- **commentaire_habitat** est lié à **utilisateur** par utilisateur_id et à **habitats** par habitat_id.
- **avis** est lié à **utilisateur** par utilisateur_id.

Classes PHP

Class Database :

```php
<?php
namespace ecf_arkadia\Config;

class Database {
    private $host;
    private $port;
    private $db_name;
    private $username;
    private $password;
    public $conn;

    public function __construct() {
        // Charge les informations de connexion depuis le fichier de configuration
        $config = require __DIR__ . '/config.php';

        $this->host = $config['db_host'];
        $this->port = $config['db_port'];
        $this->db_name = $config['db_name'];
        $this->username = $config['db_username'];
        $this->password = $config['db_password'];
    }

    public function getConnection() {
    }

    // Méthode pour fermer la connexion
    public function closeConnection() {
        if ($this->conn) {
            $this->conn->close();
        }
    }

    // Méthode pour obtenir l'instance de connexion
    public function getConn() {
        return $this->conn;
    }

    // Méthode pour vérifier si la connexion est active
    public function isConnected() {
        return ($this->conn && $this->conn->ping());
    }
}
```

Class MongoDBconnection :

```php
<?php
namespace ecf_arkadia\Config;

use MongoDB\Client;

class MongoDBConnection {
    private $uri;
    private $client;

    public function __construct() {
        // Charger l'URI depuis un fichier de configuration
        $config = require __DIR__ . '/mongo_config.php';
        $this->uri = $config['mongo_uri'] ?? "mongodb://localhost:27017";
    }

    // Établit une connexion à MongoDB ou retourne une connexion existante.
    //Cette méthode utilise une connexion paresseuse (lazy loading).

    public function connect() {
        if ($this->client === null) {
            try {
                $this->client = new Client($this->uri);
                // Vérifier la connexion
                $this->client->listDatabases();
            } catch (\Exception $e) {
                // Journaliser l'erreur
                error_log("Erreur de connexion MongoDB : " . $e->getMessage());
                throw new \Exception("Erreur de connexion à MongoDB");
            }
        }
        return $this->client;
    }
}
```

Class Admin :

```php
<?php

namespace ecf_arkadia\Models;

require_once 'class_utilisateurs.php';


class Administrateur extends Utilisateur {
    public function __construct($db) {
        parent::__construct($db);
    }

    // methode permettant de créer deux types d'utilisateur : vétérinaire et employé
    public function creerCompte($username, $email, $password, $prenom, $nom,
$role_id) {

    }
}
```

Class Veterinaire :

```php
<?php
namespace ecf_arkadia\Models;

require_once 'class_utilisateurs.php';
require_once 'class_rapportVeterinaire.php';

class Veterinaire extends Utilisateur {
    protected $conn;
    protected $rapportVeterinaire;

    public function __construct($db) {
        parent::__construct($db);
        $this->conn = $db;
        $this->rapportVeterinaire = new RapportVeterinaire($this->conn);
    }

    // Utilisation de la méthode de la classe RapportVeterinaire pour créer un
rapport vétérinaire
    public function createRapportVeterinaire($etat, $nourritureProposee, $grammage,
$datePassage, $detailEtat, $utilisateur_id, $animal_id) {
    }
```

```
    // Utilisation de la méthode de la classe RapportVeterinaire pour lire tous les
rapports vétérinaires
    public function readRapportsVeterinaires() {
    }

    // Utilisation de la méthode de la classe RapportVeterinaire pour supprimer un
rapport vétérinaire
    public function deleteRapportVeterinaire($id) {
    }

    // Méthode pour lire les rapports vétérinaires filtrés
    public function readRapportsVeterinairesFiltres($filters = []) {
    }


// Méthode pour créer un commentaire sur un habitat
public function createCommentaireHabitat($commentaire, $utilisateur_id,
$habitat_id) {

}

// Méthode pour lire tous les commentaires sur les habitats
public function readCommentairesHabitat() {
}

// Méthode pour mettre à jour un commentaire sur un habitat
public function updateCommentaireHabitat($id, $commentaire) {
}

// Méthode pour supprimer un commentaire sur un habitat
public function deleteCommentaireHabitat($id) {
}

}
?>
```

class employe :

```
<?php
namespace ecf_arkadia\Models;

require_once 'class_utilisateurs.php';
require_once 'class_avis.php';

class Employe extends Utilisateur {
    private $avis;

    public function __construct($db) {
        parent::__construct($db);
```

```
        $this->avis = new Avis($db);
    }

    public function getAvisInstance() {
    }

    public function validerAvis($avis_id) {
    }

    public function supprimerAvis($avis_id) {
    }
}
?>
```

class animal :

```
<?php
namespace ecf_arkadia\Models;

class Animal {
    private $conn;
    private $table_name = "animaux";

    public $id;
    public $prenom;
    public $race;
    public $image;
    public $etat;
    public $habitat_id;
    public $utilisateur_id;

    public function __construct($db) {
        $this->conn = $db;
    }

    // Méthode pour créer un animal
    public function create($prenom, $race, $image, $etat, $habitat_id) {
    }

    // Méthode pour lire tous les animaux
    public function read() {
    }

    // Méthode pour supprimer un animal
    public function delete($id) {
    }
}
```

class alimentation :

```php
<?php
namespace ecf_arkadia\Models;

class Alimentation {
    private $conn;
    private $table_name = "alimentation";

    public $id;
    public $id_animal;
    public $nourritureProposee;
    public $grammage;
    public $dateAlimentation;

    public function __construct($db) {
        $this->conn = $db;
    }

    // Méthode pour créer un rapport alimentaire
    public function create() {
    }

    // Méthode pour lire toutes les rapports alimentaires
    public function read() {
    }

    // Méthode pour supprimer un rapport alimentaire par ID
    public function delete($id) {
    }

}
```

class avis :

```php
<?php

namespace ecf_arkadia\Models;

class Avis {
    private $conn;
    private $table_name = "avis";

    public $id;
    public $pseudo;
    public $commentaire;
```

```
    public $valide;
    public $utilisateur_id;

    public function __construct($db) {
        $this->conn = $db;
    }

    // méthode pour créer un avis
    public function create() {

    }

    // Méthode pour lire tous les avis non validés
    public function readNonValide() {
    }

    // Méthode pour valider un avis
    public function validerAvis($avis_id) {
    }

    // Méthode pour supprimer un avis
    public function supprimerAvis($avis_id) {
    }
}
```

class contact_messages

```
<?php


namespace ecf_arkadia\Models;

class ContactMessages {
    private $conn;

    public function __construct($db) {
        $this->conn = $db;
    }

    public function saveMessage($name, $email, $subject, $message) {
    }
    // ferme la connexion a la bdd lorsque l'objet est détruit
    public function __destruct() {
        if ($this->conn) {
            $this->conn->close();
        }
    }
}
```

Class Habitat :

```php
<?php

namespace ecf_arkadia\Models;
class Habitat {
    private $conn;
    private $table_name = "habitats";

    public $id;
    public $nom;
    public $description;
    public $image;

    public function __construct($db) {
        $this->conn = $db;
    }

    // Méthode pour créer un habitat
    public function create($nom, $description, $image) {
    }
    // Méthode pour lire tous les habitats
    public function read() {
    }
    // mise à jour d'un attribut de l'habitat
    public function update($id, $nom, $description, $image) {
    }
    // supprimer un habitat
    public function delete($id) {
    }
}
```

Class RapportVeterinaire :

```php
<?php

namespace ecf_arkadia\Models;
require_once 'class_utilisateurs.php';

class RapportVeterinaire {
    private $conn;
    private $table_name = "rapport_veterinaire";

    public $id;
    public $etat;
    public $nourritureProposee;
    public $grammage;
    public $datePassage;
```

```
    public $detailEtat;
    public $utilisateur_id;
    public $animal_id;

    public function __construct($db) {
        $this->conn = $db;
    }
    // créer un rapport vétérinaire
    public function create($etat, $nourritureProposee, $grammage, $datePassage,
$detailEtat, $utilisateur_id, $animal_id) {
    }
    // lire tous les rapports vétérinaires
    public function readAll() {
    }

    // supprimer un rapport vétérinaire
    public function delete($id) {
    }

    // Méthode pour lire les rapports vétérinaires filtrés
    public function readFiltered($filters = []) {
    }
```

Class Rôle :

```
<?php

namespace ecf_arkadia\Models;

class Role {
    private $conn;
    private $table_name = "role";

    public $id_role;
    public $nom_role;

    public function __construct($db) {
        $this->conn = $db;
    }

    // Méthode pour récupérer le nom du rôle par ID
    public function getRoleById($role_id) {
    }
}
```

Class Service :

```php
<?php

namespace ecf_arkadia\Models;

class Service {
    private $conn;
    private $table_name = "service";

    public $id_service;
    public $nom;
    public $description;
    public $image;

    public function __construct($db) {
        $this->conn = $db;
    }

    // Méthode pour créer un service
    public function create($nom, $description, $image) {
    }

    // Méthode pour lire tous les services  §
    public function read() {
    }

    // Méthode pour lire un seul service §
    public function read_one($id_service) {
    }

    // Méthode pour mettre à jour un service §
    public function update($service_id, $nom, $description, $image) {
    }

    // Méthode pour supprimer un service
    public function delete($service_id) {
    }
}
```

class evenement :

```php
<?php
namespace ecf_arkadia\Models;

use ecf_arkadia\Config\MongoDBConnection;
use MongoDB\BSON\ObjectId;

class Evenement {
    private $collection;

    public function __construct() {
        $mongo = new MongoDBConnection();
        $client = $mongo->connect();
        $this->collection = $client->ecf_arkadia->evenements;
    }
    // actions CRUD sur les évenements

    public function create($nom, $description, $image, $date) {
        $result = $this->collection->insertOne([
            'nom' => $nom,
            'description' => $description,
            'image' => $image,
            'date' => $date
        ]);
        return $result->getInsertedId();
    }

    public function readAll() {
        return $this->collection->find()->toArray();
    }

    public function update($id, $nom, $description, $image, $date) {

    }

    public function delete($id) {
        $this->collection->deleteOne(['_id' => new ObjectId($id)]);
    }
}
?>
```

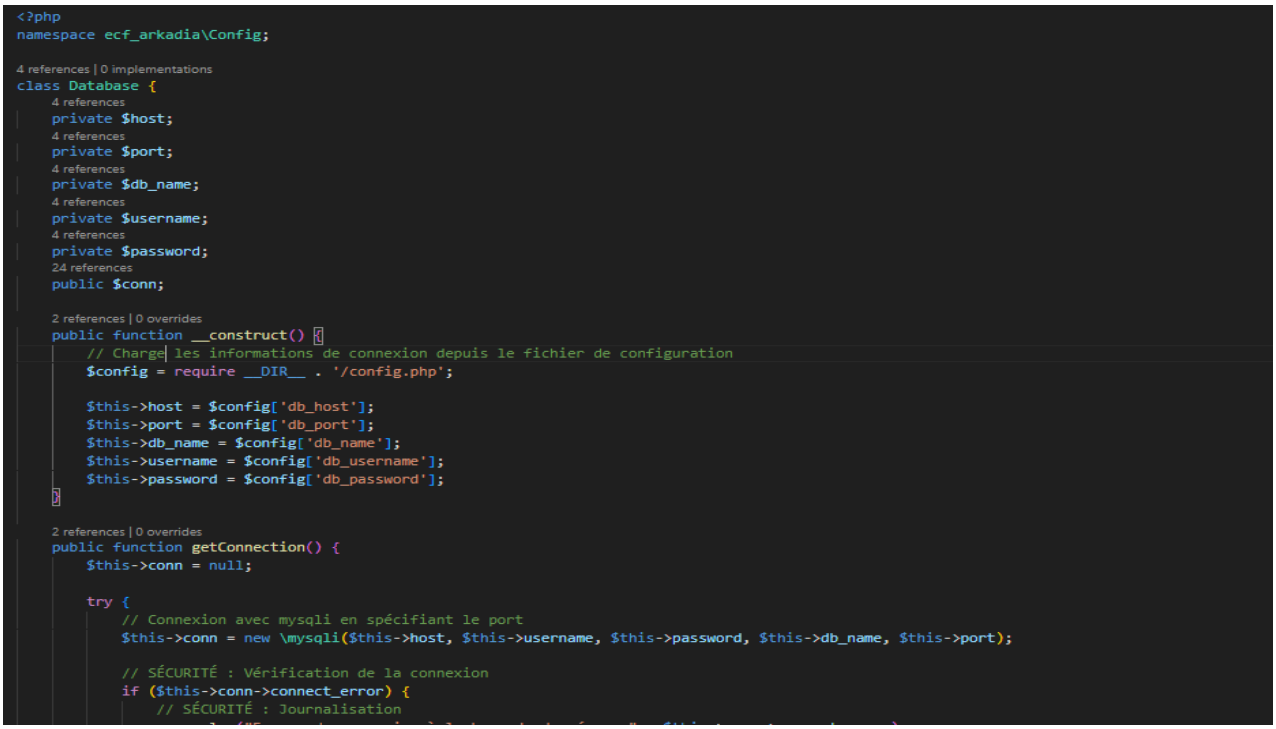
class horaireManager :

```php
<?php

namespace ecf_arkadia\Models;

use ecf_arkadia\Config\MongoDBConnection;

class HoraireManager {
  private $collection;

  public function __construct() {
    $mongo = new MongoDBConnection();
    $client = $mongo->connect();
    $this->collection = $client->ecf_arkadia->horaires;
  }

  public function obtenirHoraires() {
    try {
      $horaires = $this->collection->find()->toArray();

      // Définir l'ordre des jours
      $ordre = [
        'Lundi Mardi' => 1,
        'Mercredi' => 2,
        'Jeudi Vendredi' => 3,
        'Samedi' => 4,
        'Dimanche' => 5
      ];

// Trier les horaires selon l'ordre défini
      usort($horaires, function($a, $b) use ($ordre) {
  }

  public function modifierHoraires($horaires) {
  }

  public function nettoyerDoublons() {
}
```

Stratégies de Sécurité

Plusieurs mesures de sécurités sont mise en place dans ce projet. Pour illustrer ces mesures nous allons cité de nombre cas tiré du code de l’application.

A. le fichier de traitement de la connexion a la base de données :



```php
<?php
namespace ecf_arkadia\Config;

4 references | 0 implementations
class Database {
    4 references
    private $host;
    4 references
    private $port;
    4 references
    private $db_name;
    4 references
    private $username;
    4 references
    private $password;
    24 references
    public $conn;

    2 references | 0 overrides
    public function __construct() {
        // Charge les informations de connexion depuis le fichier de configuration
        $config = require __DIR__ . '/config.php';

        $this->host = $config['db_host'];
        $this->port = $config['db_port'];
        $this->db_name = $config['db_name'];
        $this->username = $config['db_username'];
        $this->password = $config['db_password'];
    }

    2 references | 0 overrides
    public function getConnection() {
        $this->conn = null;

        try {
            // Connexion avec mysqli en spécifiant le port
            $this->conn = new \mysqli($this->host, $this->username, $this->password, $this->db_name, $this->port);

            // SÉCURITÉ : Vérification de la connexion
            if ($this->conn->connect_error) {
                // SÉCURITÉ : Journalisation
```

```php
                error_log("Erreur de connexion à la base de données : " . $this->conn->connect_error);
                throw new \Exception("Erreur de connexion à la base de données");
            }

            // SÉCURITÉ : Configuration de la connexion
            $this->conn->set_charset('utf8mb4');
            $this->conn->options(MYSQLI_OPT_INT_AND_FLOAT_NATIVE, 1);

            return $this->conn;
        } catch (\Exception $e) {
            // SÉCURITÉ : Journalisation
            error_log("Erreur lors de la connexion à la base de données : " . $e->getMessage());
            throw $e;
        }
    }

    // Méthode pour fermer la connexion
    0 references | 0 overrides
    public function closeConnection() {
        if ($this->conn) {
            $this->conn->close();
        }
    }

    // Méthode pour obtenir l'instance de connexion
    0 references | 0 overrides
    public function getConn() {
        return $this->conn;
    }

    // Méthode pour vérifier si la connexion est active
    0 references | 0 overrides
    public function isConnected() {
        return ($this->conn && $this->conn->ping());
    }
```

1.Utilisation d'un fichier de configuration séparé : Le chargement des informations de connexion depuis un fichier externe (config.php) est une bonne pratique. Cela permet de séparer les données sensibles du code source.

2.Gestion des exceptions : L'utilisation de blocs try-catch pour gérer les erreurs de connexion.

3.Journalisation des erreurs : L'utilisation de error_log() pour enregistrer les erreurs de connexion pour le débogage et la surveillance.

3.Configuration de la connexion : L'utilisation de set_charset('utf8mb4') et MYSQLI_OPT_INT_AND_FLOAT_NATIVE sont utilisés pour renforcer la sécurité et la cohérence des données.

4.Méthode de fermeture de connexion : La présence d'une méthode closeConnection() permet de gérer proprement les ressources.

5.Vérification de l'état de la connexion : La méthode isConnected() est utile pour vérifier l'état de la connexion avant de l'utiliser.

B. Class admin :

```php
<?php

namespace ecf_arkadia\Models;

require_once 'class_utilisateurs.php';


1 reference | 0 implementations
class Administrateur extends Utilisateur {
    3 references | 0 overrides | prototype
    public function __construct($db) {
        parent::__construct($db);
    }

    1 reference | 0 overrides
    public function creerCompte($username, $email, $password, $prenom, $nom, $role_id) {
        // Vérifier si le nom d'utilisateur ou l'email existe déjà
        $query = "SELECT id FROM utilisateur WHERE username = ? OR email = ?";
        $stmt = $this->conn->prepare($query);
        $stmt->bind_param("ss", $username, $email);
        $stmt->execute();
        $stmt->store_result();

        if ($stmt->num_rows > 0) {
            // Nom d'utilisateur ou email déjà utilisé
            return false;
        }

        // Insérer l'utilisateur
        $query = "INSERT INTO utilisateur (username, email, password, prenom, nom, role_id) VALUES (?, ?, ?, ?, ?, ?)";
        $stmt = $this->conn->prepare($query);
        $hashed_password = password_hash($password, PASSWORD_DEFAULT);
        $stmt->bind_param("sssssi", $username, $email, $hashed_password, $prenom, $nom, $role_id);

        return $stmt->execute();
    }
}
```

1.Héritage sécurisé : La classe Administrateur hérite de la classe Utilisateur, ce qui permet de réutiliser de manière sûre les fonctionnalités de base de la gestion des utilisateurs

2.Requêtes préparées : L'utilisation de requêtes préparées avec $stmt = $this->conn->prepare($query) et $stmt->bind_param() protège contre les injections SQL.

3.Vérification de l'unicité : Avant de créer un nouveau compte, le code vérifie si le nom d'utilisateur ou l'email existe déjà dans la base de données. Cela empêche la création de comptes en double et renforce la sécurité de l'authentification.

4.Hachage du mot de passe : Le mot de passe est haché avant d'être stocké dans la base de données (password_hash($password, PASSWORD_DEFAULT)). Cela protège les mots de passe des utilisateurs en cas de compromission de la base de données.

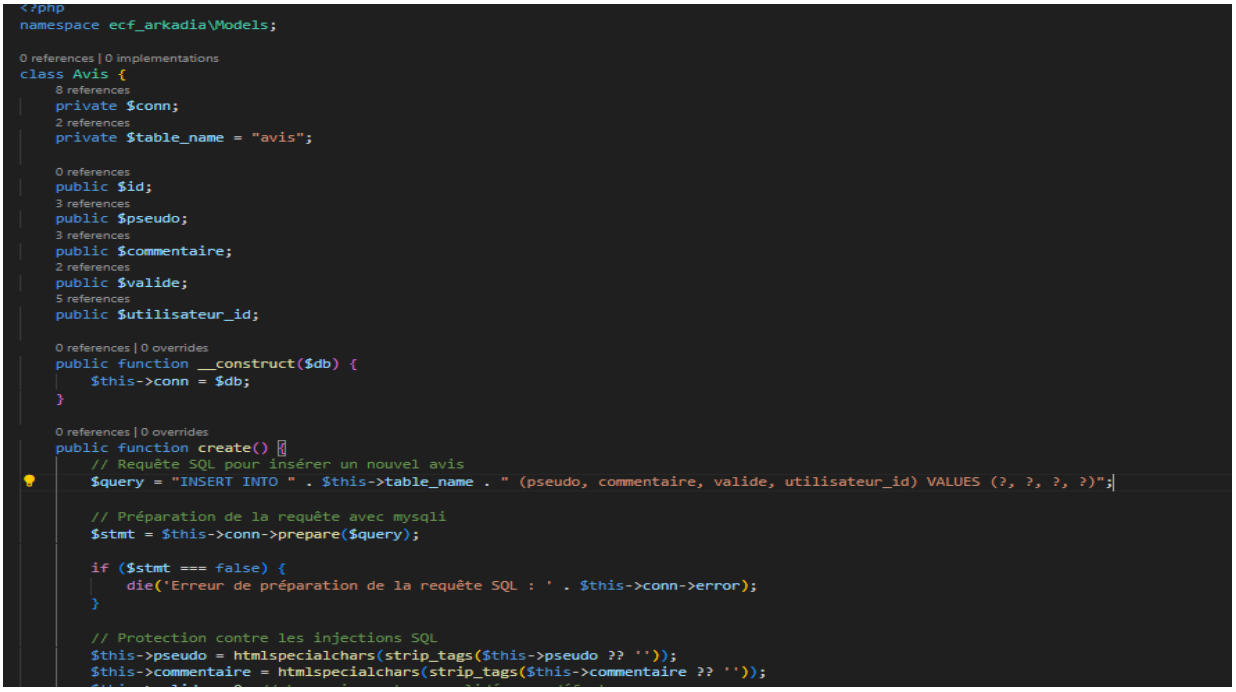
5.Utilisation de types de données spécifiques : Dans les appels à bind_param(), les types de données sont spécifiés ("ss" pour string, "i" pour integer), ce qui aide à prévenir les injections et assure que les données sont du type attendu.

6.Séparation des rôles : L'utilisation d'un role_id suggère une séparation des rôles utilisateurs, ce qui renforce bonne le contrôle d'accès.

7.Namespace : L'utilisation d'un namespace (ecf_arkadia\Models) aide à organiser le code et à éviter les conflits de noms.

Class avis :

en quelque mots cette class traite les avis laissé par les visiteurs sur la page d’accueil, et permet par la suite à l’employé de valider ou d’invalider l’avis sur son interface. Remarquez que l’avis laissé par le visiteur quelque soit son état (valide ou invalide) sera stocké dans la BDD, voila pourquoi les mesures de sécurité doivent êtres d’avantage rigoureuses dans ce type d’opération.



```php
<?php
namespace ecf_arkadia\Models;

0 references | 0 implementations
class Avis {
    8 references
    private $conn;
    2 references
    private $table_name = "avis";

    0 references
    public $id;
    3 references
    public $pseudo;
    3 references
    public $commentaire;
    2 references
    public $valide;
    5 references
    public $utilisateur_id;

    0 references | 0 overrides
    public function __construct($db) {
        $this->conn = $db;
    }

    0 references | 0 overrides
    public function create() {
        // Requête SQL pour insérer un nouvel avis
        $query = "INSERT INTO " . $this->table_name . " (pseudo, commentaire, valide, utilisateur_id) VALUES (?, ?, ?, ?)";

        // Préparation de la requête avec mysqli
        $stmt = $this->conn->prepare($query);

        if ($stmt === false) {
            die('Erreur de préparation de la requête SQL : ' . $this->conn->error);
        }

        // Protection contre les injections SQL
        $this->pseudo = htmlspecialchars(strip_tags($this->pseudo ?? ''));
        $this->commentaire = htmlspecialchars(strip_tags($this->commentaire ?? ''));
```

```php
        $this->valide = 0; // Les avis sont non validés par défaut
        $this->utilisateur_id = htmlspecialchars(strip_tags($this->utilisateur_id ?? ''));

        // Si `utilisateur_id` est vide, le définir à NULL pour respecter la contrainte de clé étrangère
        if (empty($this->utilisateur_id)) {
            $this->utilisateur_id = NULL;
        }

        // Liaison des paramètres
        $stmt->bind_param("ssii", $this->pseudo, $this->commentaire, $this->valide, $this->utilisateur_id);

        // Exécution de la requête
        if ($stmt->execute()) {
            return true;
        } else {
            die('Erreur lors de l\'exécution de la requête : ' . $stmt->error);
        }
    }

    // Méthode pour lire tous les avis non validés
    0 references | 0 overrides
    public function readNonValide() {
        $query = "SELECT * FROM " . $this->table_name . " WHERE valide = 0";
        $stmt = $this->conn->prepare($query);
        $stmt->execute();
        $result = $stmt->get_result();
        $stmt->close(); // Fermez le statement après avoir récupéré le résultat
        return $result;
    }

    // Méthode pour valider un avis
    0 references | 0 overrides
    public function validerAvis($avis_id) {
        $query = "UPDATE avis SET valide = 1 WHERE id = ?";
        $stmt = $this->conn->prepare($query);

        if ($stmt === false) {
            die('Erreur de préparation de la requête SQL : ' . $this->conn->error);
```

```php
        $stmt->bind_param("i", $avis_id);

        if ($stmt->execute()) {
            return true;
        } else {
            die('Erreur lors de l\'exécution de la requête : ' . $stmt->error);
        }
    }

    // Méthode pour supprimer un avis
    0 references | 0 overrides
    public function supprimerAvis($avis_id) {
        $query = "DELETE FROM avis WHERE id = ?";
        $stmt = $this->conn->prepare($query);

        if ($stmt === false) {
            die('Erreur de préparation de la requête SQL : ' . $this->conn->error);
        }

        $stmt->bind_param("i", $avis_id);

        if ($stmt->execute()) {
            return true;
        } else {
            die('Erreur lors de l\'exécution de la requête : ' . $stmt->error);
        }
    }
}
?>
```

voici une lisete de mesures de sécurité mise en place dans la class avis :

Utilisation de Namespace :

1.Le code utilise un namespace (ecf_arkadia\Models), ce qui aide à éviter les conflits de noms et améliore l'organisation du code.

2.Requêtes préparées : Toutes les requêtes SQL utilisent des requêtes préparées ($stmt = $this->conn->prepare($query)), ce qui est une protection efficace contre les injections SQL.

3.Liaison de paramètres : Les valeurs sont liées aux requêtes préparées avec bind_param(), ce qui renforce la protection contre les injections SQL.

4.Nettoyage des entrées : Les données d'entrée sont nettoyées avec htmlspecialchars() et strip_tags(), ce qui aide à prévenir les attaques XSS (Cross-Site Scripting).

5.Validation par défaut : Les nouveaux avis sont marqués comme non validés par défaut ($this->valide = 0), ce qui permet un contrôle avant publication.

6.Gestion des valeurs NULL : Le code gère correctement les valeurs NULL pour utilisateur_id, respectant ainsi les contraintes de clé étrangère.

7.Vérification des erreurs : Le code vérifie les erreurs lors de la préparation et de l'exécution des requêtes, ce qui aide à détecter et à gérer les problèmes.

8.Fermeture des statements : Dans la méthode readNonValide(), le statement est fermé après utilisation ($stmt->close()), ce qui est une bonne pratique pour la gestion des ressources.

9. Contrôle d'accès implicite : La méthode readNonValide() ne retourne que les avis non validés, ce qui implique un certain niveau de contrôle d'accès.

→ Nous avons également renforcé le formulaire d'envoi des avis, voici le code HTML. Quant au fichier de traitement du formulaire, il sera immédiatement présenté après ce code :

```php
<?php
session_start();

if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}

require_once '../handlers/data_page_accueil.php'; ?>

<?php include 'header.php'; ?>

// le reste du code de la page

    <div class="container py-5">
        <h2 class="text-center text-success">Laisser un avis</h2>
        <form action="../handlers/data_formulaire_avis.php" method="post" class="mt-4">
            <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>">
            <div class="mb-3">
                <label for="pseudo" class="form-label">Pseudo</label>
                <input type="text" class="form-control" id="pseudo" name="pseudo" required maxlength="50">
            </div>
            <div class="mb-3">
                <label for="commentaire" class="form-label">Commentaire</label>
                <textarea class="form-control" id="commentaire" name="commentaire" rows="3" required maxlength="500"></textarea>
            </div>
            <button type="submit" class="btn btn-primary">Envoyer</button>
        </form>
    </div>
```

Dans ce fichier, plusieurs mesures de sécurité importantes sont mises en place :

1.Protection CSRF (Cross-Site Request Forgery) : Un token CSRF est généré et stocké dans la session : $_SESSION['csrf_token'] = bin2hex(random_bytes(32)). Ce token est inclus dans le formulaire comme un champ caché : <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>"> Cette mesure protège contre les attaques CSRF en s'assurant que la requête provient bien de votre site.

2.Limitation de la longueur des entrées : Pour le champ "pseudo" : maxlength="50" . Pour le champ "commentaire" : maxlength="500". Cela aide à prévenir les attaques par dépassement de buffer et limite la quantité de données que l'utilisateur peut soumettre.

3.Champs obligatoires : L'attribut required est utilisé pour les deux champs, assurant qu'ils ne peuvent pas être soumis vides.

4.Méthode POST : Le formulaire utilise la méthode POST (method="post"), ce qui est plus sécurisé que GET pour l'envoi de données sensibles, car les données ne sont pas visibles dans l'URL.

5.Séparation des préoccupations : Le traitement des données est effectué dans un fichier séparé (data_formulaire_avis.php), ce qui est une bonne pratique de sécurité et d'organisation du code.

Voici le fichier de traitement du formulaire :

```php
<?php
// SÉCURITÉ : Démarrage de la session
if (session_status() == PHP_SESSION_NONE) {
    session_start();
}

// SÉCURITÉ : Ajout de headers de sécurité
header("Content-Security-Policy: default-src 'self'; style-src 'self' https://stackpath.bootstrapcdn.com;");
header("X-Frame-Options: DENY");
header("X-XSS-Protection: 1; mode=block");
header("X-Content-Type-Options: nosniff");

// les fichiers nécessaires
require_once '../config/Database.php';
require_once '../models/class_avis.php';
require_once '../models/class_employe.php';

// Connexion à la base de données
$database = new \ecf_arkadia\Config\Database();
$conn = $database->getConnection();

$message = '';
$messageType = '';

// SÉCURITÉ : Vérification de la méthode de requête et du token CSRF
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // SÉCURITÉ : Vérification du token CSRF
    $csrf_valid = isset($_POST['csrf_token']) && isset($_SESSION['csrf_token']) && $_POST['csrf_token'] === $_SESSION['csrf_token'];
    if (!$csrf_valid) {
        // Log l'erreur
        error_log("Tentative de soumission avec un token CSRF invalide");
        $message = "Une erreur de sécurité est survenue. Veuillez réessayer.";
        $messageType = "danger";
    } else {
        // SÉCURITÉ : Validation et nettoyage des entrées
        $pseudo = filter_input(INPUT_POST, 'pseudo', FILTER_SANITIZE_SPECIAL_CHARS);
        $commentaire = filter_input(INPUT_POST, 'commentaire', FILTER_SANITIZE_SPECIAL_CHARS);

        // SÉCURITÉ : Vérification des champs obligatoires et de leur longueur
        if (empty($pseudo) || empty($commentaire) || strlen($pseudo) > 50 || strlen($commentaire) > 500) {
            $message = "Erreur de validation. Vérifiez vos entrées.";
            $messageType = "danger";
        } else {
            try {
                // Création d'une nouvelle instance de la classe Employe
                $employe = new \ecf_arkadia\Models\Employe($conn);

                // Utilisation du getter pour accéder à l'instance de Avis
                $avis = $employe->getAvisInstance();

                // Attribution des valeurs
                $avis->pseudo = $pseudo;
                $avis->commentaire = $commentaire;
                $avis->valide = 0; // Par défaut, l'avis n'est pas validé

                // SÉCURITÉ : Vérification de la session et de l'ID utilisateur
                $avis->utilisateur_id = isset($_SESSION['user_id']) ? intval($_SESSION['user_id']) : NULL;

                // Appel de la méthode create pour insérer l'avis dans la base de données
                if ($avis->create()) {
                    $message = "Votre avis a été soumis et est en attente de validation";
                    $messageType = "success";
                    // SÉCURITÉ : Journalisation
                    error_log("Nouvel avis soumis par : $pseudo");
                } else {
                    $message = "Erreur lors de la soumission de votre avis. Veuillez réessayer.";
                    $messageType = "danger";
                    // SÉCURITÉ : Journalisation
                    error_log("Échec de la soumission d'avis pour : $pseudo");
                }
            } catch (Exception $e) {
                $message = "Une erreur est survenue. Veuillez réessayer plus tard.";
                $messageType = "danger";
                // SÉCURITÉ : Journalisation
                error_log("Erreur lors de la soumission d'avis : " . $e->getMessage());
            }
        }
    }
```

```
    }
}

// Génération d'un nouveau token CSRF pour la prochaine requête
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));
?>

<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Soumettre un avis</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <div class="container mt-5">
        <?php if (!empty($message)): ?>
            <div class="alert alert-<?php echo $messageType; ?>" role="alert">
                <?php echo $message; ?>
            </div>
        <?php endif; ?>

        <?php if ($messageType !== 'success'): ?>
            <a href="../views/zoo_arkadia_accueil.php" class="btn btn-primary">Retour au formulaire</a>
        <?php endif; ?>

        <a href="../views/zoo_arkadia_accueil.php" class="btn btn-secondary">Retour à l'accueil</a>
    </div>
</body>
</html>
```

Ce fichier de traitement du formulaire contient plusieurs mesures de sécurité importantes. Voici une analyse détaillée :

1.Démarrage de session sécurisé : La session est démarrée de manière sécurisée au début du script.

2.En-têtes de sécurité HTTP : Content Security Policy (CSP) : Limite les sources de contenu autorisées. X-Frame-Options : Empêche le clickjacking en interdisant l'intégration de la page dans des iframes. X-XSS-Protection : Active la protection contre les attaques XSS dans les navigateurs. X-Content-Type-Options : Empêche le MIME type sniffing.

3.Vérification de la méthode de requête : S'assure que la requête est bien une méthode POST.

4.Protection CSRF : Vérifie la validité du token CSRF soumis avec le formulaire.

5.Validation et nettoyage des entrées : Utilise filter_input() avec FILTER_SANITIZE_SPECIAL_CHARS pour nettoyer les entrées utilisateur.

6.Vérification de la longueur des entrées : Vérifie que le pseudo et le commentaire respectent les limites de longueur définies.

7.Gestion sécurisée des erreurs : Utilise des messages d'erreur génériques pour éviter de divulguer des informations sensibles.

8.Journalisation des erreurs : Enregistre les erreurs et les tentatives suspectes dans les logs du serveur.

9.Gestion sécurisée de l'ID utilisateur : Vérifie la présence de l'ID utilisateur dans la session et le convertit en entier.

10.Régénération du token CSRF :Génère un nouveau token CSRF après chaque requête pour une sécurité accrue.

11.Utilisation de requêtes préparées : Implicitement utilisé dans la méthode create() de la classe Avis (non visible dans ce fichier, mais important à noter).

12.Gestion des exceptions : Utilise un bloc try-catch pour gérer les erreurs de manière sécurisée.

13.Affichage sécurisé des messages : Utilise des variables intermédiaires pour les messages, évitant l'injection directe de contenu dans la page.

Installation et Configuration

Prérequis Matériel

Ordinateur avec au moins 4 Go de RAM , Espace disque : minimum 2 Go disponible

Logiciel

Système d'exploitation : Windows 10/11 (testé sur ces versions) , XAMPP version 3.3.0 ou supérieure (inclut Apache, PHP, MySQL)

1.PHP version 8.2

2.MongoDB (dernière version stable)

3.Composer (gestionnaire de dépendances PHP)

4.Git (pour le clonage du dépôt)

Extensions PHP requises

php_mongodb.dll (version 1.15.1 pour PHP 8.2 TS x64). Autres extensions standard de PHP (incluses dans XAMPP)

Instructions d'Installation

1. Installation de XAMPP

1.1. Téléchargez XAMPP 3.3.0 ou version supérieure depuis apachefriends.org.

1.2. Installez XAMPP en suivant les instructions du programme d'installation.

2. Configuration de MySQL dans XAMPP

2.1. Ouvrez le fichier my.ini dans le dossier d'installation de XAMPP (généralement C:\xampp\mysql\bin\my.ini).

2.2. Modifiez la ligne du port par défaut pour MySQL :

port=3307

2.3. Sauvegardez le fichier et redémarrez le service MySQL via le panneau de contrôle XAMPP.

3. Installation de MongoDB

3.1. Téléchargez et installez MongoDB depuis le site officiel de MongoDB.

3.2. Suivez les instructions d'installation par défaut.

4. Installation de l'extension PHP MongoDB

4.1. Téléchargez le fichier php_mongodb-1.15.1-8.2-ts-vs16-x64.zip.

4.2. Extrayez php_mongodb.dll et placez-le dans le dossier C:\xampp\php\ext\.

4.3. Ajoutez la ligne suivante à votre fichier C:\xampp\php\php.ini :

extension=php_mongodb.dll

4.4. Redémarrez le service Apache via le panneau de contrôle XAMPP.

5. Installation de Composer

5.1. Téléchargez et installez Composer depuis getcomposer.org.

6. Clonage et Configuration du Projet

6.1. Clonez ou téléchargez le projet dans le dossier C:\xampp\htdocs\.

6.2. Ouvrez un terminal et naviguez vers le dossier du projet.

6.3. Installez les dépendances du projet :

composer install

7. Configuration des Bases de Données

7.1. Pour MySQL :

Lancez phpMyAdmin depuis le panneau de contrôle XAMPP.

Créez une nouvelle base de données nommée arkadia_zoo.

Importez le fichier SQL fourni dans le dossier database du projet.

7.2. Pour MongoDB :

Ouvrez un terminal et exécutez mongod pour démarrer le serveur MongoDB.

Dans un autre terminal, exécutez mongosh pour vous connecter à MongoDB.

Utilisez MongoDB Compass pour importer les données fournies dans le dossier database du projet.

8. Configuration du Projet

8.1. Assurez-vous que les fichiers de configuration dans le dossier config sont correctement remplis avec les informations de connexion aux bases de données.

9. Lancement de l'Application

9.1. Démarrez les services Apache et MySQL depuis le panneau de contrôle XAMPP.

9.2. Ouvrez un navigateur et accédez à http://localhost/[nom_du_dossier_projet]/index.php pour vérifier l'état des connexions aux bases de données.

9.3. Pour accéder à l'application, il existe un fichier utilisé comme point d'entrée de l'application, ils se trouve à la racine du projet, voici le lien d'accès : http://localhost/arkadia_zoo/index.php.

Accès à l'Application

Utilisez les identifiants suivants pour accéder aux différentes interfaces :

Admin : username: userPhilipe, password: passwordPhilipe

Vétérinaire : username: userMathieu, password: passwordMathieu

Employé : username: userLisa, password: passwordLisa

Dépannage

Si vous rencontrez des erreurs liées à l'extension MongoDB, vérifiez que le fichier DLL est correctement placé et que l'extension est activée dans php.ini.

Pour les problèmes de connexion à MySQL, vérifiez que le port 3307 est bien configuré et que le service est en cours d'exécution.

En cas de problème avec MongoDB, assurez-vous que le service est démarré et que vous pouvez vous connecter via mongosh.

Conclusion

Le développement du site web pour Arkadia Zoo a été un projet ambitieux et enrichissant, mettant en œuvre une variété de technologies modernes pour créer une plateforme interactive et transparente. Ce projet a permis de relever plusieurs défis techniques et conceptuels, tout en restant fidèle à la vision écologique et éthique du zoo.

L'utilisation combinée de MySQL et MongoDB comme bases de données a offert une flexibilité remarquable dans la gestion des différents types de données, permettant une performance optimale et une scalabilité pour les futures évolutions du site. L'architecture du projet, structurée autour de PHP en backend et d'une interface utilisateur responsive, a démontré l'efficacité d'une approche hybride dans le développement web moderne.

Les fonctionnalités mises en place, telles que le système de gestion de contenu pour les informations sur les animaux, le module d'avis des visiteurs, et les interfaces d'administration différenciées, répondent pleinement aux objectifs initiaux du projet. Elles offrent une expérience utilisateur immersive tout en facilitant la gestion quotidienne du zoo par le personnel.

Un accent particulier a été mis sur la sécurité, avec l'implémentation de diverses stratégies pour protéger les données sensibles et assurer une utilisation sûre du site tant pour les visiteurs que pour les administrateurs.

Ce projet a non seulement abouti à la création d'un site web fonctionnel et attrayant pour Arkadia Zoo, mais a également posé les bases pour de futures améliorations et extensions. Il illustre comment la technologie peut être mise au service de la sensibilisation écologique et du bien-être animal, tout en offrant une expérience numérique engageante pour les utilisateurs.

En conclusion, ce projet représente une étape importante dans la modernisation de la présence en ligne d'Arkadia Zoo, alignant parfaitement les objectifs de communication du zoo avec les attentes des visiteurs d'aujourd'hui en matière d'information et d'interaction numérique.