DEPLOIEMENT DE L'APPLICATION ARCADIA

1. Backend

- **Déploiement sur Render :**

    1. Le backend a été déployé avec succès sur Render.
    2. Les routes principales ont été configurées, incluant /auth, /animals, /habitats, etc.

- **Configuration des variables d'environnement :**

    1. API_URL a été défini avec la valeur /api/v1 pour structurer les routes de l'API.
    2. JWT_SECRET a été configuré avec une clé secrète sécurisée pour la génération et la vérification des tokens JWT.

- **Résolution des erreurs :**

    1. Une erreur 500 a été détectée lors de la connexion des utilisateurs. Après investigation :
        - Les logs ont révélé que soit le token JWT était absent, soit les identifiants étaient incorrects.
        - Les console.log ont permis de vérifier :
            - La présence d'un token valide dans la requête.
            - La comparaison des mots de passe via bcrypt.compare.
        - L'erreur venait principalement de l'absence de variables d'environnement correctement configurées sur Render.

2. Frontend

- **Déploiement sur Vercel :**

    1. Le frontend a été déployé avec succès sur Vercel.
    2. Le fichier .env du frontend a été configuré pour pointer vers l'URL de l'API déployée sur Render.
    3. L'intégration entre le frontend et le backend a été testée.

- **Gestion de CORS :**

    1. Des problèmes liés aux restrictions de CORS ont été rencontrés.
    2. Une solution a été mise en place dans le backend pour permettre les requêtes venant des domaines autorisés (frontend déployé sur Vercel).

3. Tests et vérifications

Tests réalisés :

- **Connexion utilisateur :**

    1. Les tests ont confirmé que les utilisateurs pouvaient se connecter avec des identifiants valides.
    2. Les tokens JWT sont maintenant correctement générés et envoyés au frontend.

- **Accès sécurisé :**

    1. Les routes nécessitant une authentification (middleware authMiddleware) fonctionnent correctement.
    2. Les permissions sont vérifiées selon le rôle de l'utilisateur.

- **Tests des fonctionnalités principales :**

    1. Création et récupération des données (par exemple : animaux, habitats, services).
    2. Envoi et affichage des images (serveur statique configuré avec /images).

3/Déploiement des fichiers

1. **Configuration de Cloudinary** :

    - Intégration de la bibliothèque cloudinary pour héberger les images.
    - Ajout des identifiants nécessaires dans les variables d'environnement (API_KEY, API_SECRET, cloud_name).

2. **Configuration de Multer avec Cloudinary** :

    - Utilisation de CloudinaryStorage pour spécifier le dossier Cloudinary (zoo_images) et les formats de fichiers autorisés (jpg, jpeg, png).
    - Création et exportation d'une instance Multer avec un stockage basé sur Cloudinary.

3. **Création de l'API d'upload** :

    - Mise en place d'une route /upload pour gérer les envois d'images.
    - Vérification de la présence d'un fichier avant d'envoyer une réponse au client.

- Retour d'une URL sécurisée de Cloudinary en cas de succès.

4. **Gestion des modèles liés** :

- Modification du modèle Animal pour inclure une liste de liens d'images hébergées sur Cloudinary.
- Mise à jour de la méthode createAnimal pour prendre en compte les images envoyées via Multer et enregistrer leurs URL dans MongoDB.