```
const mongoose = require('mongoose');

const animalSchema = new mongoose.Schema({
  name: { type: String, required: true },
  habitat: { type: mongoose.Schema.Types.ObjectId, ref: 'Habitat' },
  race: { type: String, required: true },
  imagesUrl: [{ type: String }],
  vetReports: [{ type: mongoose.Schema.Types.ObjectId, ref: 'VetReport' }],
  foodRecords: [{ type: mongoose.Schema.Types.ObjectId, ref: 'FoodRecord' }],
  stats: { type: mongoose.Schema.Types.ObjectId, ref: 'Stat' },

});


module.exports = mongoose.model('Animal', animalSchema);
```

Le code crée un schéma pour un modèle Animal avec Mongoose. Ce schéma spécifie les différentes propriétés qu'un animal aura dans la base de données MongoDB, telles que :

- **name** (String) : Le nom de l'animal, obligatoire.
- **habitat** (Référence à un objet Habitat) : Un lien vers le modèle Habitat, qui représente l'habitat où vit l'animal.
- **race** (String) : La race de l'animal, obligatoire.
- **imagesUrl** (Tableau de chaînes) : Contient les URLs des images liées à l'animal.
- **vetReports** (Tableau d'ID de VetReport) : Référence à des rapports vétérinaires associés à l'animal.
- **foodRecords** (Tableau d'ID de FoodRecord) : Référence à des enregistrements alimentaires pour l'animal.
- **stats** (Référence à Stat) : Référence à un objet Stat contenant des statistiques liées à l'animal

```
const Animal = require('../models/Animal'); // MongoDB Animal model
const Habitat = require('../models/Habitat');
const VetReport = require('../models/VetReport'); // MongoDB VetReport model
const FoodRecord = require('../models/FoodRecord')
const Stat = require('../models/Stat')

// Get all animals
exports.getAllAnimals = async (req, res) => {
    try {
        const animals = await Animal.find()
            .populate('habitat')
            .populate('vetReports')
            .populate('foodRecords')
```

```
        res.status(200).json(animals);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};



exports.getAnimalById = async (req, res) => {
    try {
        // Find the animal by ID and populate references
        const animal = await Animal.findById(req.params.id)
            .populate('habitat')
            .populate('vetReports')
            .populate('foodRecords')
            .populate('stats'); // Include stats in population

        if (!animal) return res.status(404).json({ message: 'Animal not found'
});

        // Increment the views count in the associated stats
        if (animal.stats) {
            await Stat.findByIdAndUpdate(
                animal.stats._id,
                { $inc: { views: 1 } }, // Increment views by 1
                { new: true } // Return the updated document
            );
        } else {
            // Optionally create a stats document if none exists
            const newStat = await Stat.create({ animal: animal._id, views: 1
});
            animal.stats = newStat._id;
            await animal.save();
        }

        res.status(200).json(animal);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};

exports.createAnimal = async (req, res) => {
    const { name, race, habitat } = req.body;

    // Use Cloudinary URL if file is uploaded
    const imageUrl = req.file ? req.file.path : null;

    try {
        // Create the new animal
```

```
        const newAnimal = new Animal({
            name,
            race,
            habitat,
            imagesUrl: imageUrl ? [imageUrl] : [],
        });
        await newAnimal.save();

        // Update the corresponding habitat to include this animal
        if (habitat) {
            await Habitat.findByIdAndUpdate(
                habitat,
                { $push: { animals: newAnimal._id } },
                { new: true }
            );
        }

        res.status(201).json(newAnimal);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};


// Update animal details
exports.updateAnimal = async (req, res) => {
    try {
        const updatedAnimal = await Animal.findByIdAndUpdate(req.params.id,
req.body, { new: true });
        if (!updatedAnimal) return res.status(404).json({ message: 'Animal not
found' });
        res.status(200).json(updatedAnimal);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};

// Delete an animal
exports.deleteAnimal = async (req, res) => {
    try {
        const deletedAnimal = await Animal.findByIdAndDelete(req.params.id);
        if (!deletedAnimal) return res.status(404).json({ message: 'Animal not
found' });

        // Remove the associated
        await VetReport.deleteMany({ animal: req.params.id });
        await FoodRecord.deleteMany({ animal: req.params.id });
```

```
        res.status(200).json({ message: 'Animal and associated vet reports
deleted successfully' });
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};
```

getAllAnimals (Récupérer tous les animaux)

- Récupère tous les animaux de la base de données.
- Utilise .populate() pour inclure les **habitats, rapports vétérinaires et enregistrements alimentaires** liés.
- Retourne un tableau contenant tous les animaux avec leurs relations associées.

getAnimalById (Récupérer un animal par son ID)

- Cherche un animal spécifique via son ID et peuple ses références (habitat, vetReports, foodRecords, stats).
- Incrémente automatiquement le **compteur de vues** dans l'objet stats.
- Si l'animal n'a pas encore de statistiques, il en crée une nouvelle et l'associe à l'animal.

createAnimal (Créer un nouvel animal)

- Récupère les données du corps de la requête (name, race, habitat).
- Vérifie si une **image** a été téléchargée et l'ajoute à imagesUrl.
- Crée un **nouvel objet Animal** et l'enregistre dans la base de données.
- Met à jour l'**habitat** pour y ajouter l'ID du nouvel animal.

updateAnimal (Mettre à jour un animal)

- Recherche l'animal par ID et met à jour ses informations avec les nouvelles données fournies.
- Renvoie l'animal mis à jour ou un message d'erreur s'il n'existe pas.

deleteAnimal (Supprimer un animal)

- Supprime l'animal correspondant à l'ID fourni.

- Supprime également **tous les rapports vétérinaires et les enregistrements alimentaires** liés à cet animal.
- Retourne un message confirmant la suppression.

En résumé :

- getAllAnimals → Liste tous les animaux avec leurs relations.
- getAnimalById → Récupère un animal et met à jour son compteur de vues.
- createAnimal → Crée un animal et l'associe à un habitat.
- updateAnimal → Met à jour un animal existant.
- deleteAnimal → Supprime un animal ainsi que ses données associées.