

# Refuge d'animaux – MVC, DAO et scénario d'ajout

## 1 Rappel rapide : MVC

L'architecture **MVC** (Model–View–Controller) sert à séparer clairement les responsabilités dans une application :

- **Model (M)** : représente les données et la logique métier. Dans notre projet, c'est par exemple `ModelAnimal.php` (classe qui représente un animal).
- **View (V)** : gère l'affichage et l'interface utilisateur. Ici : `admin_animaux.html` + `VuesAnimaux.js` (formulaire, tableau, messages).
- **Controller (C)** : reçoit les requêtes, décide quoi faire, appelle le Model/DAO et renvoie une réponse. Ici : `ControleurAnimal.php`.

Idée principale : **on évite le “spaghetti”** (HTML + SQL + logique mélangés) en séparant les couches.

Un bon petit rappel vidéo sur MVC (en anglais) :

<https://youtu.be/DUg2SWWK18I?si=5wF4RZeCXdyIsbK8>

## 2 Rappel rapide : DAO

Un **DAO** (Data Access Object) est une classe dont le rôle est de discuter avec la base de données :

- elle contient le code PDO ;
- elle exécute les requêtes SQL (`SELECT`, `INSERT`, `UPDATE`, `DELETE`) avec des requêtes préparées ;
- elle renvoie des `array` associatifs ou des objets Model.

Dans notre projet, `AnimalDAO.php` est le DAO qui gère la table `animal`.

Avantages :

- **centraliser** le SQL ;
- **réutiliser** les méthodes (`listierTous()`, `trouverParId()`, etc.) ;
- améliorer la **sécurité** (requêtes préparées, contre l'injection SQL).

## 3 Vue d'ensemble du projet

Architecture simplifiée du projet “Refuge d'animaux” :

- **Front-end (client)**
  - `admin_animaux.html` : page avec le formulaire et le tableau des animaux.
  - `VuesAnimaux.js` : gère le DOM (événements, affichage du tableau, messages).
  - `RequetesAnimaux.js` : gère les appels `fetch()` vers le serveur.
- **Back-end (serveur)**
  - `route.php` : front-controller qui lit `entite` et `action`, et appelle le bon contrôleur.
  - `ControleurAnimal.php` : reçoit les actions (`liste`, `ajouter`, `modifier`, `supprimer`) et appelle le DAO.
  - `ModelAnimal.php` : modèle qui représente un animal.
  - `AnimalDAO.php` : DAO qui exécute les requêtes SQL sur la table `animal`.
  - `Connexion.php` : fournit un objet PDO configuré (Singleton).

## 4 Scénario complet : ajout d'un animal

On détaille ici ce qui se passe quand on ajoute un animal via le formulaire.

### Étape 1 – Soumission du formulaire (Vue côté client)

L'utilisateur remplit le formulaire dans `admin_animaux.html` (nom, espèce, etc.) puis clique sur **Enregistrer**.

Dans `VuesAnimaux.js` :

- l'événement `submit` est intercepté avec `preventDefault()` pour éviter le rechargement complet de la page ;
- un objet `FormData` est construit à partir du formulaire (`new FormData(form)`) ;
- on vérifie si un `id` est présent :
  - si l'`id` est vide : c'est un **ajout** ;
  - si l'`id` est non vide : c'est une **modification**.
- on appelle la fonction `apiSauverAnimal(formData, isEdition)` définie dans `RequetesAnimaux.js`.

### Étape 2 – Appel à l'API (Requêtes côté client)

Dans `RequetesAnimaux.js` :

- on ajoute au `FormData` les champs `entite="animal"` et `action="ajouter"` (si ce n'est pas une édition) ;
- on envoie une requête `fetch()` vers `route.php` (POST avec le `FormData` en corps) ;
- on récupère la réponse JSON et on la retourne à `VuesAnimaux.js`.

### Étape 3 – route.php choisit le bon contrôleur

Dans `route.php` :

- on lit `entite` et `action` (dans `$_GET` ou `$_POST`);
- si `entite = "animal"`, on `require ControleurAnimal.php`;
- on appelle `ControleurAnimal::traiter($action)`.

### Étape 4 – ControleurAnimal traite l'action

Dans `ControleurAnimal::traiter()` :

- on utilise un `switch` sur `$action`;
- pour `"ajouter"`, on appelle la méthode privée `ajouter()`.

Dans `ajouter()` :

- on lit les valeurs du formulaire dans `$_POST` (`nom`, `espèce`, `description`, `statut`, `date_arrivee`);
- on valide les champs obligatoires (par ex. `nom` et `espèce`);
- on gère l'upload de la photo avec `$_FILES['photo']` :
  - vérifier l'extension (`jpg`, `png`, etc.);
  - générer un nom de fichier unique;
  - déplacer le fichier dans le dossier `serveur/Animal/photos/`.
- on construit un objet `ModelAnimal` avec les données validées;
- on crée un `AnimalDAO` et on appelle `insérer($animal)`;
- on renvoie un JSON du type : `{"succes": true, "message": "..."}.`

### Étape 5 – DAO insère dans la base de données

Dans `AnimalDAO::insérer()` :

- on écrit la requête SQL `INSERT INTO animal (...)` avec des paramètres nommés;
- on prépare la requête avec `$this->db->prepare($sql)`;
- on exécute la requête avec les valeurs de l'objet `ModelAnimal`;
- on retourne le booléen indiquant si l'exécution a réussi.

### Étape 6 – Réponse côté client et rafraîchissement de la liste

De retour dans `VuesAnimaux.js` :

- on récupère le JSON renvoyé par le serveur;
- si `succes = true` :
  - on réinitialise le formulaire;
  - on vide le champ caché `id`;
  - on rappelle `chargerAnimauxVue()` pour recharger la table avec la nouvelle liste.
- on affiche un message de succès ou d'erreur dans `#zone-message`.