PROJET SGBD
APPLICATION REFUGE ANIMALIER

Thomas Paquet
3ÈME BAC INFO

Table des matières

Introduction

L'application **Refuge Animalier Pro** est un logiciel lourd pour la gestion d'un refuge animalier. L'application est construite avec **Electron** et **Vue 3**, et elle permet de gérer les personnes, les animaux, les adoptions, les familles d'accueil et la santé des animaux.

Les pré-requis sont : une version de Node.js ≥ 18, npm ou un gestionnaire similaire et un serveur **MySQL ≥ 8.0**. Une base de données dédiée nommée par défaut refuge est nécessaire, et l'application utilise **Prisma** comme ORM pour générer le schéma SQL et interagir avec la base de données. La section *Installation* de la documentation explique comment installer les dépendances (npm install) et générer les fichiers Prisma (npm run prisma:generate).

Le dépôt contient un fichier .sql classique : la structure de la base est décrite dans prisma/schema.prisma, et Prisma génère automatiquement les scripts SQL à partir de ce fichier. La suite du rapport se base donc sur ce schéma. Mais j'ai aussi ajouté un script.sql qui contient la BDD.

Analyse de la base de données

Modèle Conceptuel des Données (MCD)

Le modèle conceptuel présente les entités principales, leurs attributs clés et les relations entre elles. Voici une synthèse textuelle du MCD déduit du fichier schema.prisma (principalement des modèles et des énumérations).

Entités principales

1. **Personne** – représente toute personne ayant une interaction avec le refuge.

- *Attributs :* id_personne (PK), nom, prenom, email, tel, date_naissance, jardin, rue, numero, code_postal, ville, pays, type_personne (enum : prospect, adoptant, fa, donateur, multiple).
- *Relations :* une personne peut être utilisateur, demander une adoption, adopter un animal, avoir des rendez-vous ou appartenir à une famille d'accueil.

2. **Utilisateur** – associe un compte logiciel à une personne.
   - *Attributs :* id_user (PK), id_personne (FK vers Personne), username (unique), password, role (enum : admin, agent, bénévole, vétérinaire externe).
   - *Relations :* chaque utilisateur se réfère à une et une seule personne. Un utilisateur (agent ou bénévole) peut réaliser des visites et gérer des rendez-vous.
3. **Espèce** et **Race** – l'espèce (chien, chat…) est le niveau de classification supérieur et la race (Berger Belge, Siamois…) dépend d'une espèce.
   - *Espèce :* id_espece (PK), nom.
   - *Race :* id_race (PK), nom, id_espece (FK vers Espèce).
   - *Relations :* 1 Espèce ⇨ n Races.
4. **Animal** – objet principal de la gestion (animaux recueillis).
   - *Attributs :* id_animal (PK), nom, id_espece (FK), date_naissance, sexe (enum M/F/Inconnu), statut (enum : arrive, quarantaine, soin, adoptable, reservé, adopté, en FA, transféré, décédé, indisponible).
   - *Relations :*
     - **Animal–Race** (animal_race) : relation n:m entre animal et race avec un pourcentage de race.
     - **Animal–Emplacement** (animal_emplacement) : relation n:m avec dates de début/fin pour suivre les emplacements occupés (box, quarantaine, etc.).
     - **Animal–Entrée** : une entrée représente la façon dont l'animal est arrivé (abandon, trouvé, saisie, transfert).
     - **Animal–Événement Médical** : un animal peut avoir plusieurs événements médicaux.
     - **Animal–Note Comportement** : un animal peut avoir plusieurs notes de comportement.
     - **Animal–Demande d'adoption** : via la table demande_animal, un animal peut être associé à plusieurs demandes, avec un ordre de priorité.
     - **Animal–Adoption** : relation 1:1 lorsqu'une adoption est finalisée.
     - **Animal–Réservation** et **Animal–Rendez-vous** : un animal peut avoir des réservations (bloquer l'animal pour un adoptant) et des rendez-vous.
     - **Animal–Famille d'accueil** : via placement_fa, un animal peut être placé en famille d'accueil.
5. **Emplacement** – représente un lieu physique du refuge (box, quarantaine, etc.).
   - *Attributs :* id_emplacement (PK), code (unique), type, capacite, actif (boolean).
   - *Relations :* 1 Emplacement ⇨ n Animal_Emplacement (un emplacement peut accueillir plusieurs animaux en différents temps).

6. **Entrée** – enregistrement de l'arrivée de l'animal.
    - *Attributs :* id_entree (PK), id_animal (FK), date_entree, type (enum : abandon, trouvé, saisie, transfert).
7. **Événement Médical** – décrit un acte vétérinaire ou un soin.
    - *Attributs :* id_evenement (PK), id_animal (FK), id_veterinaire (FK), date_evenement, type_evenement, commentaire.
8. **Note de comportement** – observations diverses sur le comportement de l'animal.
    - *Attributs :* id_note (PK), id_animal (FK), date_note, texte.
9. **Demande d'adoption** – dossier déposé par une personne pour adopter un ou plusieurs animaux.
    - *Attributs :* id_demande (PK), id_personne (FK), date_depot, statut (soumise, en étude, acceptée, refusée), type_logement, jardin, accord_proprio, enfants, autres_animaux, experience_animaux, preferences, commentaire.
    - *Relations :*
        - liée à une personne (le demandeur).
        - relation n:m avec Animal via demande_animal, avec un champ priorite pour indiquer le rang de préférence de l'animal.
        - peut conduire à une adoption (une adoption finalisée se réfère à une demande).
        - est associée à des visites à domicile.
10. **Visite à domicile** – inspection du logement de l'adoptant.
    - *Attributs :* id_visite (PK), id_demande (FK), id_user (utilisateur qui effectue la visite), date_visite, statut (planifiée, réalisée, annulée), notes.
11. **Rendez-vous** – rencontre planifiée entre une personne et un animal (séance d'observation ou rencontre adoption).
    - *Attributs :* id_rdv (PK), id_personne (FK), id_animal (FK), date_heure, type (enum : visite, adoption, vétérinaire), statut (planifié, effectué, annulé), notes.
12. **Réservation** – blocage temporaire d'un animal pour un adoptant potentiel.
    - *Attributs :* id_reservation (PK), id_animal (FK), id_personne (FK), date_reservation, statut (expirée, annulée, convertie en adoption), date_expiration.
13. **Adoption** – adoption d'un animal à la suite d'une demande.
    - *Attributs :* id_adoption (PK), id_animal (FK), id_personne (FK), id_demande (FK), date_adoption, statut (brouillon, finalisée, annulée, retour).
    - *Relations :* une adoption peut avoir plusieurs paiements et peut donner lieu à un retour post-adoption.
14. **Paiement** – enregistrement des paiements liés à l'adoption.
    - *Attributs :* id_paiement (PK), id_adoption (FK), montant, date_paiement, mode (espèces, carte, virement).
15. **Retour post-adoption** – retour d'un animal après adoption.
    - *Attributs :* id_retour (PK), id_adoption (FK), date_retour, suite_retour (repropose, transfert, décédé, autre), commentaire.
16. **Famille d'accueil** – famille qui héberge un animal temporairement.
    - *Attributs :* id_famille (PK), id_personne (FK), statut_famille (active, suspendue, terminée), date_debut, date_fin.
    - *Relations :* via placement_fa, une famille peut héberger plusieurs animaux.

17. **Placement_fa** – lien n:m entre famille d'accueil et animal, avec dates de début et fin du placement.
18. **Document** – pièce jointe (image, PDF…) associée à un animal, une demande ou un dossier médical.
    - *Attributs :* id_document (PK), id_animal/id_demande/id_personne selon le contexte, type_document, url, date_upload.

Ce MCD représente les grandes entités gérées par l'application. Les cardinalités (1:1, 1:n, n:m) sont décrites dans les relations ci-dessus.

Modèle Logique des Données (MLD)

Le MLD traduit le MCD en tables relationnelles. Chaque entité devient une table avec sa clé primaire, ses attributs et les clés étrangères. Ci-dessous, les principales tables et champs (types indiqués à titre informatif ; l'ORM utilise MySQL via Prisma).

- **personne** (id_personne BIGINT AUTO_INCREMENT, nom VARCHAR(255), prenom VARCHAR(255), email VARCHAR(255) UNIQUE, tel VARCHAR(100), date_naissance DATETIME, jardin TINYINT, rue VARCHAR(255), numero VARCHAR(50), code_postal VARCHAR(20), ville VARCHAR(255), pays VARCHAR(255), type_personne ENUM('prospect','adoptant','fa','donateur','multiple')).
- **utilisateur** (id_user BIGINT AUTO_INCREMENT, id_personne BIGINT REFERENCES personne(id_personne) ON DELETE RESTRICT, username VARCHAR(100) UNIQUE, password VARCHAR(255), role ENUM('admin','agent','benevole','veto_ext')).
- **espece** (id_espece BIGINT AUTO_INCREMENT, nom VARCHAR(255)).
- **race** (id_race BIGINT AUTO_INCREMENT, nom VARCHAR(255), id_espece BIGINT REFERENCES espece(id_espece) ON DELETE RESTRICT).
- **animal** (id_animal BIGINT AUTO_INCREMENT, nom VARCHAR(255), id_espece BIGINT REFERENCES espece(id_espece), date_naissance DATETIME, sexe ENUM('M','F','Inconnu'), statut ENUM('arrive','quarantaine','soin','adoptable','reserve','adopte','en_FA','transfere','decede','indisponible'), date_arrivee DATETIME, puce VARCHAR(255) UNIQUE, sterilise TINYINT, description TEXT, autres champs spécifiques).
- **animal_race** (id_animal BIGINT REFERENCES animal(id_animal) ON DELETE CASCADE, id_race BIGINT REFERENCES race(id_race) ON DELETE RESTRICT, pourcentage INT, PRIMARY KEY(id_animal,id_race)).
- **emplacement** (id_emplacement BIGINT AUTO_INCREMENT, code VARCHAR(50) UNIQUE, type VARCHAR(50), capacite INT, actif TINYINT).
- **animal_emplacement** (id_animal BIGINT REFERENCES animal(id_animal) ON DELETE CASCADE, id_emplacement BIGINT REFERENCES emplacement(id_emplacement) ON DELETE RESTRICT, date_debut DATETIME, date_fin DATETIME, PRIMARY KEY(id_animal,date_debut)).
- **entree** (id_entree BIGINT AUTO_INCREMENT, id_animal BIGINT REFERENCES animal(id_animal) ON DELETE CASCADE, date_entree DATE, type ENUM('abandon','trouve','saisie','transfert')).
- **veterinaire** (id_veterinaire BIGINT AUTO_INCREMENT, nom VARCHAR(255), telephone VARCHAR(100), email VARCHAR(255)).

- **evenement_medical** (id_evenement BIGINT AUTO_INCREMENT, id_animal BIGINT REFERENCES animal(id_animal), id_veterinaire BIGINT REFERENCES veterinaire(id_veterinaire), date_evenement DATETIME, type_evenement VARCHAR(100), commentaire TEXT).
- **note_comportement** (id_note BIGINT AUTO_INCREMENT, id_animal BIGINT REFERENCES animal(id_animal), date_note DATETIME, description TEXT).
- **demande_adoption** (id_demande BIGINT AUTO_INCREMENT, id_personne BIGINT REFERENCES personne(id_personne) ON DELETE RESTRICT, date_depot DATETIME DEFAULT NOW(), statut ENUM('soumise','en_etude','acceptee','refusee'), type_logement VARCHAR(255), jardin TINYINT, accord_proprio TINYINT, enfants TINYINT, autres_animaux VARCHAR(255), experience_animaux TEXT, preferences TEXT, commentaire TEXT).
- **demande_animal** (id_demande BIGINT REFERENCES demande_adoption(id_demande) ON DELETE CASCADE, id_animal BIGINT REFERENCES animal(id_animal) ON DELETE RESTRICT, priorite INT, PRIMARY KEY(id_demande,id_animal)).
- **visite_domicile** (id_visite BIGINT AUTO_INCREMENT, id_demande BIGINT REFERENCES demande_adoption(id_demande) ON DELETE CASCADE, id_user BIGINT REFERENCES utilisateur(id_user) ON DELETE RESTRICT, date_visite DATETIME, statut ENUM('planifie','realise','annule'), notes TEXT).
- **rendez_vous** (id_rdv BIGINT AUTO_INCREMENT, id_personne BIGINT REFERENCES personne(id_personne), id_animal BIGINT REFERENCES animal(id_animal), date_heure DATETIME, type ENUM('visite','adoption','veterinaire'), statut ENUM('planifie','effectue','annule'), notes TEXT).
- **reservation** (id_reservation BIGINT AUTO_INCREMENT, id_animal BIGINT REFERENCES animal(id_animal), id_personne BIGINT REFERENCES personne(id_personne), date_reservation DATETIME, statut ENUM('expiree','annulee','convertie'), date_expiration DATETIME).
- **adoption** (id_adoption BIGINT AUTO_INCREMENT, id_animal BIGINT REFERENCES animal(id_animal), id_personne BIGINT REFERENCES personne(id_personne), id_demande BIGINT REFERENCES demande_adoption(id_demande), date_adoption DATETIME, statut ENUM('brouillon','finalisee','annulee','retour')).
- **paiement** (id_paiement BIGINT AUTO_INCREMENT, id_adoption BIGINT REFERENCES adoption(id_adoption), montant DECIMAL(10,2), date_paiement DATETIME, mode ENUM('especes','carte','virement')).
- **retour_post_adoption** (id_retour BIGINT AUTO_INCREMENT, id_adoption BIGINT REFERENCES adoption(id_adoption), date_retour DATETIME, suite_retour ENUM('repropose','transfert','decede','autre'), commentaire TEXT).
- **famille_accueil** (id_famille BIGINT AUTO_INCREMENT, id_personne BIGINT REFERENCES personne(id_personne), statut_famille ENUM('active','suspendue','terminee'), date_debut DATETIME, date_fin DATETIME).
- **placement_fa** (id_placement BIGINT AUTO_INCREMENT, id_famille BIGINT REFERENCES famille_accueil(id_famille), id_animal BIGINT REFERENCES animal(id_animal), date_debut DATETIME, date_fin DATETIME).
- **document** (id_document BIGINT AUTO_INCREMENT, id_animal BIGINT NULL REFERENCES animal(id_animal), id_demande BIGINT NULL REFERENCES

demande_adoption(id_demande), id_personne BIGINT NULL REFERENCES personne(id_personne), type_document VARCHAR(50), url VARCHAR(255), date_upload DATETIME).

Ce MLD reflète la structure logique de la base MySQL générée par Prisma.

Exemples d'insertion SQL

Voici quelques exemples d'insertions SQL (en respectant le MLD ci-dessus) afin d'illustrer l'alimentation de la base :

```
-- Création d'une personne adoptante
INSERT INTO personne
  (nom, prenom, email, tel, date_naissance, jardin, rue, numero, code_postal,
   ville, pays, type_personne)
VALUES
  ('Dupont', 'Jean', 'jean.dupont@example.com', '0470/12.34.56',
   '1990-05-10', 1, 'Rue de la Paix', '123', '6000', 'Charleroi',
   'Belgique', 'adoptant');

-- Ajout d'un utilisateur agent lié à cette personne
INSERT INTO utilisateur
  (id_personne, username, password, role)
VALUES
  (LAST_INSERT_ID(), 'jdupont', 'motdepasse_hashé', 'agent');

-- Ajout d'espèces et de races
INSERT INTO espece (nom) VALUES ('Chien'), ('Chat');
INSERT INTO race (nom, id_espece) VALUES
  ('Berger Belge', 1),
  ('Siamois', 2);

-- Ajout d'un animal arrivé au refuge
INSERT INTO animal
  (nom, id_espece, date_naissance, sexe, statut, date_arrivee, puce, sterilise, description)
VALUES
  ('Rex', 1, '2022-01-15', 'M', 'arrive', '2024-09-01', 'BE123456789', 0, 'Chiot timide, vaccins à jour');

-- Lien entre l'animal et sa race
INSERT INTO animal_race (id_animal, id_race, pourcentage) VALUES (1, 1, 100);

-- Création d'un emplacement et association de l'animal à cet emplacement
INSERT INTO emplacement (code, type, capacite, actif) VALUES ('Q01', 'Quarantaine', 5, 1);
INSERT INTO animal_emplacement (id_animal, id_emplacement, date_debut, date_fin)
VALUES (1, 1, '2024-09-01', NULL);

-- Demande d'adoption pour deux animaux
INSERT INTO demande_adoption
  (id_personne, date_depot, statut, type_logement, jardin, accord_proprio,
   enfants, autres_animaux, experience_animaux, preferences, commentaire)
VALUES
  (1, '2024-10-15', 'soumise', 'Maison', 1, 1, 2,
   '1 chat', 'Expérience avec les chiens', 'Cherche chien calme', 'Aucune');

-- Ajout des animaux demandés dans la demande (exemple avec un seul animal)
INSERT INTO demande_animal (id_demande, id_animal, priorite) VALUES
  (LAST_INSERT_ID(), 1, 1);
```

```
-- Planification d'une visite à domicile
INSERT INTO visite_domicile (id_demande, id_user, date_visite, statut, notes)
VALUES (1, 1, '2024-10-20 15:00:00', 'planifie', 'Visite initiale');

-- Finalisation d'une adoption
INSERT INTO adoption (id_animal, id_personne, id_demande, date_adoption, statut)
VALUES (1, 1, 1, '2024-11-01', 'finalisee');

-- Enregistrement du paiement
INSERT INTO paiement (id_adoption, montant, date_paiement, mode)
VALUES (1, 150.00, '2024-11-01', 'carte');
```

Ces exemples montrent comment créer des personnes, des utilisateurs, des animaux, des emplacements, des demandes et finaliser une adoption. Ils peuvent être adaptés pour les autres entités et usages.

Script complet (Prisma)

Le dépôt ne fournit pas de fichier .sql, mais le schéma est défini dans prisma/schema.prisma. Ce fichier comprend :

- Une section generator client (générateur Prisma Client) et datasource db qui indique que MySQL est utilisé et que la connexion est définie via la variable DATABASE_URL.
- Des énumérations pour les types de personnes, rôles utilisateurs, sexes, statuts des animaux, types d'entrées, statuts de demandes, types de rendez-vous, statuts d'adoption, modes de paiement, suites de retour, statuts de famille, etc..
- Les modèles (model) qui correspondent aux tables du MLD : personne, utilisateur, espece, race, animal, animal_race, emplacement, animal_emplacement, entree, evenement_medical, note_comportement, demande_adoption, demande_animal, visite_domicile, rendez_vous, reservation, adoption, paiement, retour_post_adoption, famille_accueil, placement_fa, document, etc.
- Chaque modèle définit la clé primaire via @id, les relations via @relation et les index/contraintes uniques via @unique ou @index.

Vous trouverez ces définitions en entier dans le fichier schema.prisma du dépôt. Par exemple, l'énumération StatutFamille y est déclarée ainsi :

```
enum StatutFamille {
  active
  suspendue
  terminee
}
```

et le modèle animal_emplacement est défini avec des relations en cascade.

Architecture et technologies

La documentation du dépôt fournit des indications claires sur l'architecture et l'utilisation des technologies. La structure du projet est orientée **Electron** avec une séparation nette entre le **processus principal** (main process) et le **processus de rendu** (renderer). Voici un aperçu :

- src/main/ : contient les modules du process principal, dont les handlers IPC et l'accès à Prisma. C'est ici que l'application démarre, crée les fenêtres et expose des API via ipcMain.
- src/preload/ : fichier de pré-chargement qui fait le pont (contextBridge) entre le process principal et le renderer. Il expose des fonctions sécurisées (API) que le front-end peut appeler.
- src/renderer/ : application **Vue 3** avec **Vite** comme bundler. On y trouve les composants, les vues, le routeur et les styles (style.css). Le README précise que les styles communs (page, card, bouton, etc.) sont centralisés pour garder une interface homogène.
- src/db/ : scripts pour initialiser la base de données, générer les seeds et installer Prisma (connexion MySQL). On y retrouve les fichiers de seed (seed-admin.ts, seed-sample.ts) et un script smoke-test.ts pour vérifier rapidement l'insertion et la suppression d'une demande d'adoption.
- src/ipc/ : modules métier exposés via IPC. Les dossiers mentionnés dans le README regroupent les fonctionnalités (adoption, familles, etc.).
- src/security/ : contient la configuration de sécurité (par exemple le durcissement des sessions ou contextes).
- src/shared/ : types et interfaces partagés entre le process principal et le renderer.
- .vite/ et out/ : dossiers générés par Vite et Electron Forge lors de la compilation/packaging.

Technologies utilisées :

- **Electron + Vite** : pour créer une application bureau multiplateforme. Electron Forge est utilisé pour gérer la compilation et la distribution (création d'installateurs Squirrel/ZIP/DEB/RPM).
- **Vue 3** : framework front-end pour le renderer.
- **Prisma** : ORM TypeScript qui génère un client pour MySQL à partir du fichier schema.prisma.
- **TypeScript** : le code est typé et compilé.
- **MySQL 8** : base de données relationnelle utilisée via Prisma.
- **Node.js 20** et **npm** : environnement d'exécution du back-end, gestion des dépendances et scripts.
- **ESLint** et *scripts utilitaires* pour le linting, la génération de build, le packaging et les tests.

Cette architecture est modulaire : le front-end Vue communique avec le process principal via un pont sécurisé (contextBridge). Le process principal gère l'accès à la base via Prisma et isole la logique métier. Cela respecte les bonnes pratiques d'Electron (séparation des privilèges, isolement du DOM).

Axes d'amélioration

1. **Couverture de tests** – Actuellement, un seul test *smoke* existe pour vérifier la création/suppression d'une demande d'adoption. Il serait bénéfique d'ajouter :
    - des tests unitaires pour les modules Prisma (validation des requêtes, gestion des relations et contraintes),
    - des tests d'intégration pour vérifier les flux d'adoption, de réservation et de placement,
    - des tests end-to-end (E2E) avec des outils comme Playwright ou Spectron pour valider l'UI et le comportement d'Electron.
2. **Sécurité et conformité RGPD** – Étant donné que l'application gère des données personnelles (adoptants, bénévoles) et des données sensibles (adoptions d'animaux), il serait pertinent :
    - d'implémenter un système d'authentification solide (par exemple, stockage des mots de passe avec argon2 et gestion de sessions avec expiration),
    - de chiffrer les champs sensibles en base (numéro de puce, notes confidentielles),
    - de mettre en place un système de rôles et permissions fines (interface d'administration distincte).
3. **Expérience utilisateur** – Quelques pistes :
    - améliorer la gestion des états (chargement, erreurs) via un store (Pinia) et ajouter des notifications claires,
    - proposer un thème responsive et accessible (contraste élevé, navigation clavier),
    - intégrer une recherche et des filtres avancés pour les animaux (statut, race, âge, santé).
4. **Synchronisation et mobilité** – L'application est un client lourd ; envisager une version web progressive (PWA) ou une synchronisation cloud pour permettre l'accès aux données depuis plusieurs postes.
    - Utiliser **Prisma Data Proxy** ou un service API pour exposer des endpoints sécurisés.
    - Mettre en place un mode hors-ligne avec synchronisation différée (IndexedDB pour stocker localement).
5. **Analytique et tableaux de bord** – Ajouter des tableaux de bord dynamiques (nombre d'animaux par statut, délais moyens d'adoption, ressources par race/espèce) afin d'aider les gestionnaires du refuge à prendre des décisions.
6. **Automatisation administrative** –
    - Génération automatique de contrats d'adoption en PDF avec signature électronique.
    - Envoi d'emails automatisés aux adoptants pour le suivi post-adoption, rappels de vaccination, etc.
    - Intégration avec un service de paiement pour sécuriser les transactions (Stripe, Mollie…).
7. **Modularité et ouverture** – Faciliter l'ajout de nouveaux modules (par ex. gestion de dons, parrainage, boutique solidaire) grâce à une architecture plugin ou un framework modulaire.

8. **Internationalisation** – Prévoir la traduction des libellés et des messages pour une adoption dans d'autres refuges ou régions linguistiques (français/néerlandais/anglais), en utilisant les outils d'i18n de Vue.

Conclusion

Le projet **Refuge Animalier Pro** propose une solution complète pour la gestion d'un refuge d'animaux avec une architecture moderne (Electron + Vue 3 + Prisma + MySQL) et une base de données riche en entités. L'utilisation de Prisma simplifie la gestion du schéma et assure la cohérence des données.

Ce rapport a détaillé le modèle conceptuel et logique de données, fourni des exemples d'insertion SQL et décrit l'architecture du code. Des pistes d'amélioration ont été proposées pour renforcer la sécurité, améliorer l'expérience utilisateur, faciliter l'administration et ouvrir l'application à de nouvelles fonctionnalités.