CAHIER DES CHARGES TECHNIQUES :

Application JAVA cartes grises

Fait par Anatoli Mikoyan.

Sommaire

1. Contexte du projet
    1.1 Présentation
    1.2 Période et Modalité
2. Objectifs du projet
3. Besoins fonctionnels
    3.1 Fonctionnalités principales
    3.2 Sécurité
4. Architecture technique
    4.1 Ressources matérielles et logicielles
    4.2 Langages utilisés
5. Conception et Modélisation
    5.1 Modèle de données
    5.2 Diagrammes UML
6. Développement et Tests
    6.1 Gestion du projet
    6.2 Tests et validation
7. Déploiement et Maintenance
8. Livrables
9. Conclusion

1. Contexte du projet

1.1 Présentation

Ce projet a été réalisé dans le cadre **du BTS SIO – option SLAM** lors des séances d'Accompagnement Personnalisé **au Lycée La Tournelle de La Garenne-Colombes**. Il consiste au développement d'une **application Java** permettant de gérer différentes entités liées **aux cartes grises.**

L'application offre la possibilité de gérer les **marques**, les **modèles**, les **propriétaires**, les **véhicules** ainsi que les relations entre les **propriétaires** et leurs **véhicules**. Les utilisateurs peuvent **ainsi ajouter, modifier** et **supprimer** les informations relatives à ces entités.

1.2 Période et Modalité

- **Durée** : Du **06 novembre 2023 au 08 mars 2024**
- **Mode de réalisation** : **Travail individuel** avec un suivi pédagogique.

2. Objectifs du projet

L'objectif de ce projet est de développer une application Java de gestion des cartes grises. L'application doit permettre de :

- **Gérer les marques** : Ajouter, modifier et supprimer les informations relatives aux différentes marques de véhicules.
- **Gérer les modèles** : Associer un modèle à une marque et gérer les informations liées aux modèles de véhicules.
- **Gérer les propriétaires** : Enregistrer, consulter, modifier et supprimer les informations sur les propriétaires de véhicules.
- **Gérer les véhicules** : Suivre les informations des véhicules, notamment leur matricule et leurs caractéristiques.
- **Gérer les relations de propriété** : Associer un véhicule à un propriétaire en précisant la date de début et, si nécessaire, la date de fin de possession.

3. Besoins fonctionnels

3.1 Fonctionnalités principales

Gestion des marques :

- Ajouter, modifier et supprimer des marques de véhicules.
- Consulter la liste des marques existantes.

Gestion des modèles :

- Associer un modèle à une marque.
- Ajouter, modifier et supprimer des modèles de véhicules.
- Afficher la liste des modèles enregistrés.

Gestion des propriétaires :

- Enregistrer un nouveau propriétaire.
- Modifier ou supprimer les informations d’un propriétaire.
- Afficher la liste des propriétaires existants.

Gestion des véhicules :

- Ajouter un nouveau véhicule avec ses caractéristiques (matricule, modèle, etc.).
- Modifier ou supprimer les informations d’un véhicule.
- Afficher la liste des véhicules enregistrés.

Gestion des relations de propriété :

- Associer un propriétaire à un véhicule.
- Définir la période de possession (date de début et date de fin si applicable).
- Modifier ou supprimer une relation de propriété.

Navigation et interface utilisateur :

- Accéder aux différentes sections (marques, modèles, propriétaires, véhicules et relations) via une interface graphique **Swing**.
- Afficher des messages d'erreur en cas de problème lors des opérations.

Sécurité des données :

- Protéger l’application contre les **injections SQL** et les **attaques XSS**.
- Empêcher l’ajout de doublons lors de l'enregistrement de nouvelles données.

3.2 Sécurité

Protection contre les injections SQL

- Utiliser des **requêtes paramétrées (Prepared Statements)** au lieu des requêtes SQL classiques pour éviter l'injection de code malveillant dans les formulaires.
- Valider et filtrer les données d'entrée avant de les utiliser dans les requêtes SQL.

Protection contre les attaques XSS (Cross-Site Scripting)

- Échapper et **sanitiser** toutes les données affichées dans l’interface pour empêcher l’insertion de scripts malveillants.
- Vérifier et encoder les données provenant des utilisateurs avant de les afficher.

Contrôle des accès et gestion des droits

- Restreindre l’accès aux différentes parties de l'application en fonction des rôles des utilisateurs (si applicable).
- Limiter les privilèges des utilisateurs à ce qui est strictement nécessaire (principe du **moindre privilège**).

Validation des données côté client et côté serveur

- Effectuer des **contrôles de validation** sur les champs de saisie (format, longueur, type de données) avant l'insertion en base.
- Empêcher la saisie de **données dupliquées** (par exemple, éviter l'ajout d'un propriétaire ou d'un véhicule déjà existant).

Gestion des erreurs sécurisée

- Afficher des **messages d’erreur génériques** aux utilisateurs pour éviter de divulguer des informations sensibles (ex. : "Erreur lors de l'ouverture de la vue" au lieu du message d'exception complète).
- Logger les erreurs détaillées en interne pour faciliter le diagnostic sans exposer ces informations aux utilisateurs.

Sécurisation des opérations CRUD (Create, Read, Update, Delete)

- Vérifier l’existence des enregistrements avant modification ou suppression pour éviter les erreurs ou les manipulations abusives.
- Empêcher les suppressions ou modifications non autorisées via un **contrôle d’intégrité**.

4. Architecture technique

4.1 Ressources matérielles et logicielles

Ressources matérielles :

- Ordinateur portable
- Connexion internet
- Clavier, souris et écran

Ressources logicielles :

- IDE : IntelliJ IDEA ou Eclipse (pour le développement en Java)
- Ensemble de solution logicielle : XAMPP ou MAMP (incluant le gestionnaire de base de données relationnelles MySQL)
- Outil de modélisation de base de données : Mocodo (pour la conception des modèles MERISE)
- Outil de modélisation UML : Visual Paradigm (pour les diagrammes UML : cas d'utilisation, activité, etc.)
- Gestion de projet : Trello (pour organiser les tâches et suivre l'avancement du projet)
- Outil de design : Figma (pour la création des wireframes et maquettes de l'interface utilisateur)
- Plateforme de développement collaborative : GitHub (pour versionner le code et partager les livrables du projet)

4.2 Langages utilisés

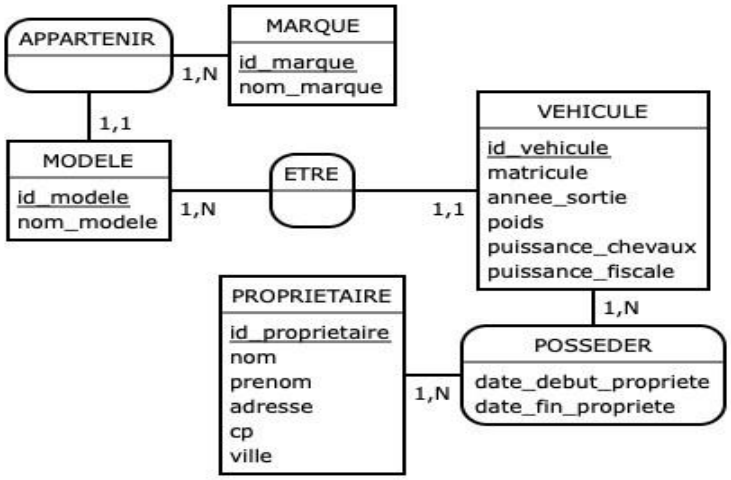
- **Back-end** : **Java** (JavaFX/Spring Boot)
- **Base de données** : **SQL**

5. Conception et Modélisation

5.1 Modèle de données

- **MCD , MLD & MPD** pour structurer les relations entre les tables

Modèle Conceptuel de données (MCD)



Modèle Logiques de données (MLD) format BTS

- MARQUE (id_marque, nom_marque)
  - Clé primaire : id_marque

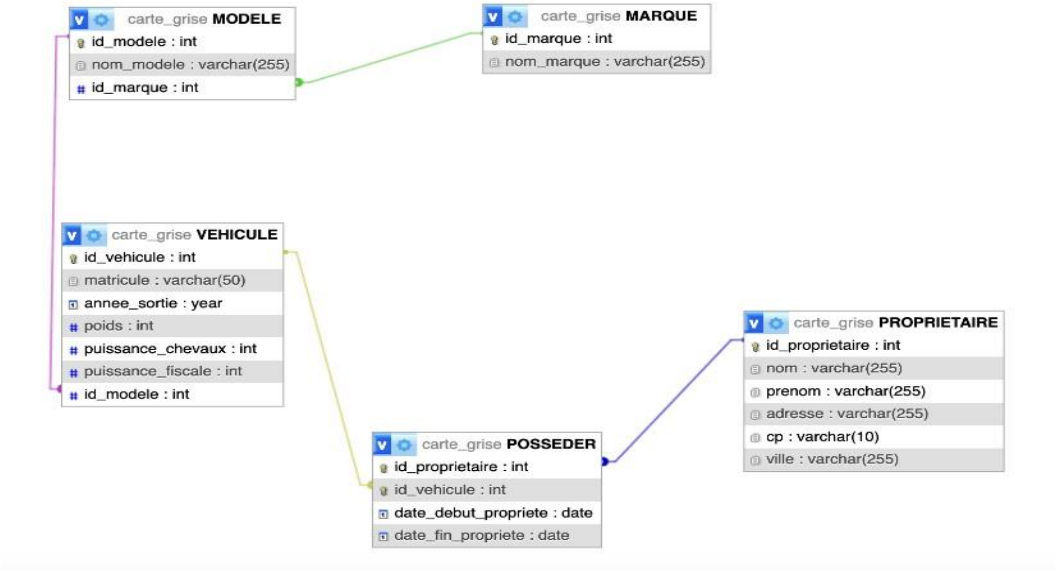
- MODELE (id_modele, nom_modele, id_marque)
  - Clé primaire : id_modele
  - Clé étrangère : id_marque → MARQUE (id_marque)

- VEHICULE (id_vehicule, matricule, annee_sortie, poids, puissance_chevaux, puissance_fiscale, id_modele)
  - Clé primaire : id_vehicule
  - Clé étrangère : id_modele → MODELE (id_modele)

- PROPRIETAIRE (id_proprietaire, nom, prenom, adresse, cp, ville)
  - Clé primaire : id_proprietaire

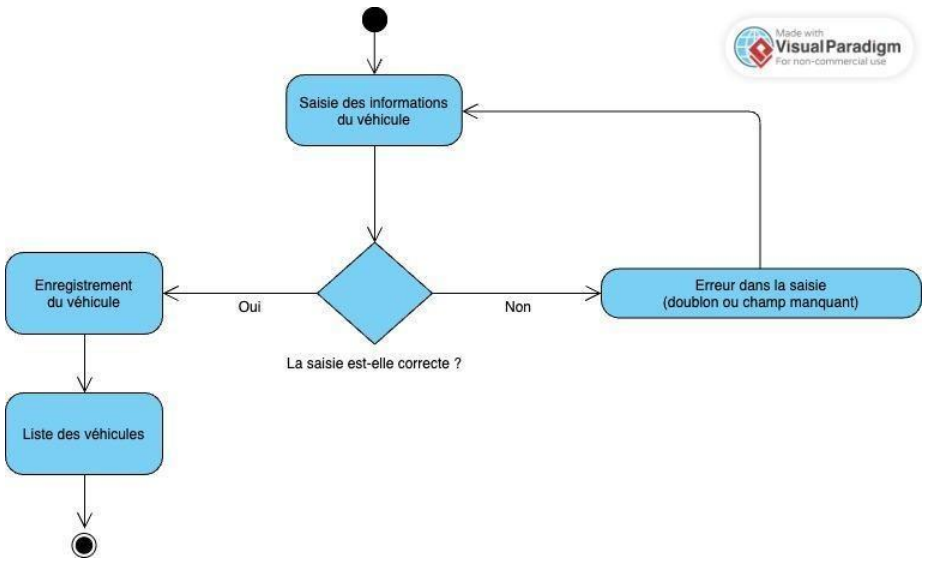
- POSSEDER (id_proprietaire, id_vehicule, date_debut_propriete, date_fin_propriete)
  - Clé primaire : id_proprietaire, id_vehicule, date_debut_propriete
  - Clé étrangère : id_proprietaire → PROPRIETAIRE (id_proprietaire)
  - Clé étrangère : id_vehicule → VEHICULE (id_vehicule)

Modèle Physique de données (MPD)

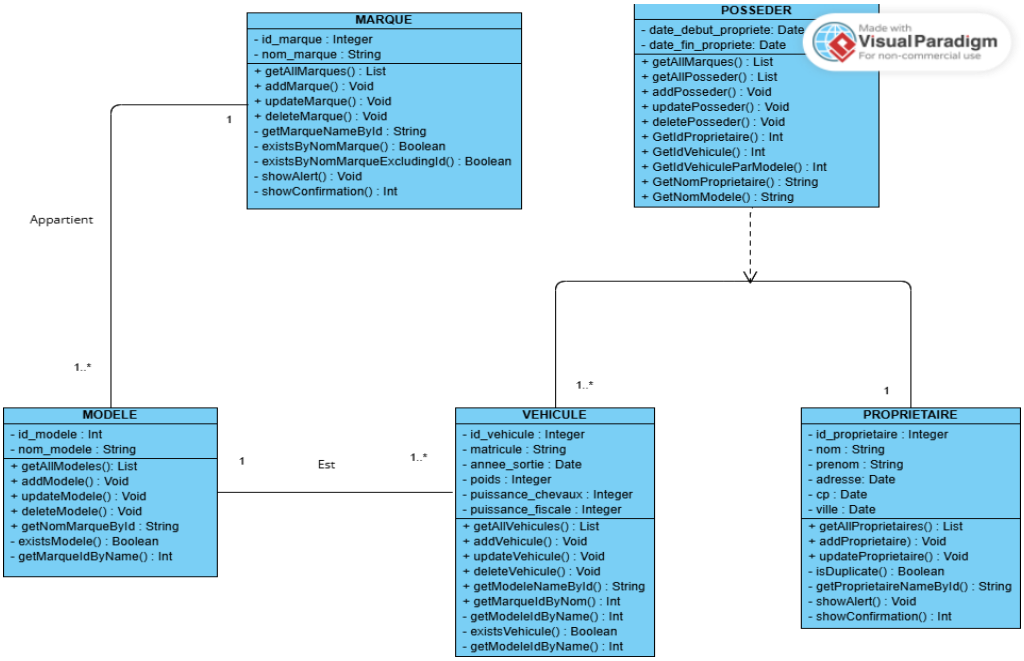


5.2 Diagrammes UML

- **Diagramme activité** pour définir les interactions



- **Diagramme de classe** pour modéliser le fonctionnement



6. Développement et Tests

6.1 Gestion du projet

- **Méthodologie Classique** en cycle V
- **Utilisation de GitHub** pour la gestion du code source

6.2 Tests et validation

- **Tests unitaires en JUnit** pour valider chaque fonctionnalité
- **Tests d'intégration** pour assurer le bon fonctionnement global

7. Déploiement et Maintenance

- **Hébergement sur un serveur local via MAMP**
- **Documentation technique** et **guide utilisateur** fournis
- **Maintenance évolutive** pour mises à jour futures

8. Livrables

- Code source du projet sur **GitHub**
- Base de données SQL exportable
- Documentation technique complète
- Rapport de test détaillé

9. Conclusion

Ce projet propose une application Java robuste et conviviale dédiée à la gestion des informations des cartes grises pour divers modèles de véhicules. Grâce à l'implémentation de fonctionnalités complètes, telles que la gestion des marques, des modèles, des propriétaires, des véhicules et des relations de propriété, il permet une administration fluide et structurée des données. L’application intègre également des mécanismes de sécurité avancés, incluant la protection contre les injections SQL et les attaques XSS, garantissant ainsi une gestion sécurisée des données sensibles.

Ce projet met en avant des compétences approfondies en conception et développement d’applications, ainsi qu’en gestion de données dans un environnement professionnel. L'utilisation d'une interface graphique Swing offre une navigation intuitive, tandis que des pratiques sécuritaires rigoureuses assurent l'intégrité des informations et la confidentialité des utilisateurs. En somme, cette application démontre une approche méthodique et professionnelle, alliant performance, sécurité et ergonomie pour répondre efficacement aux besoins de gestion des cartes grises.