



| **Révision** | **Rédacteurs** | **Date** | **Objet** |
| --- | --- | --- | --- |
| 1 | Paul RIPAULT Elisa PRUVOST  Aurélia PIC | 26/05/2025 | Création du document |
|  |  |  |  |
|  |  |  |  |

# INTRODUCTION

## Objet du document

Ce document a pour objectif de présenter l’essentiel des questions techniques liées à la mise en place de l’application VroomVroomCar.

Ce document présente :

* Le diagramme de classes
* Le modèle physique de données

# SOMMAIRE

### Table des matières

[**1 INTRODUCTION 2**](#_c5gljstc5sw8)

[1.1 Objet du document 2](#_pylorgyedach)

[**2 SOMMAIRE 3**](#_zbifqv785zdh)

[*2.1.1 Table des matières 3*](#_xr65eomxxnjk)

[**3 ARCHITECTURE LOGICIELLE 4**](#_gp0tfyd6mal2)

[*3.1 Produits et versions 4*](#_mi9688or5rlz)

[3.1.1 Logiciels installés sur le poste de travail 4](#_tjr65x2zrr9v)

[*3.1.2 Langages, frameworks et librairies spécifiques 4*](#_6y0umteik0ya)

[3.1.3 Serveur de base de données 4](#_ag9bp0j86l60)

[4 FOCUS TECHNIQUE 5](#_7es0awy6ivly)

[4.1 Diagramme de classes métier (ou MCD) 5](#_btnkab17avvk)

[4.2 Diagramme entités relations (ou MPD) 5](#_66j9dbpw0weo)

[4.3 Règles de développement coté back 6](#_we0yxz0wdhv)

[4.3.1 Découpage en couches 6](#_cs2b50wr5l3l)

[**5 TESTS ET INTÉGRATION 9**](#_uqz68ll549u6)

[5.1 Tests unitaires 9](#_2h0477cfun62)

[5.2 Tests fonctionnels 9](#_mwfpjl4lju7a)

[5.3 Indicateurs de qualité de code 9](#_og6od15zye21)

# ARCHITECTURE LOGICIELLE

## Produits et versions

### Logiciels installés sur le poste de travail

| **Nom** | **Version** |
| --- | --- |
| Spring Tool Suite ou IntelliJ | ? |
| MySQL | ? |
| Git bash | ? |
| Java Development Kit | ? |
|  |  |
|  |  |

### Langages, frameworks et librairies spécifiques

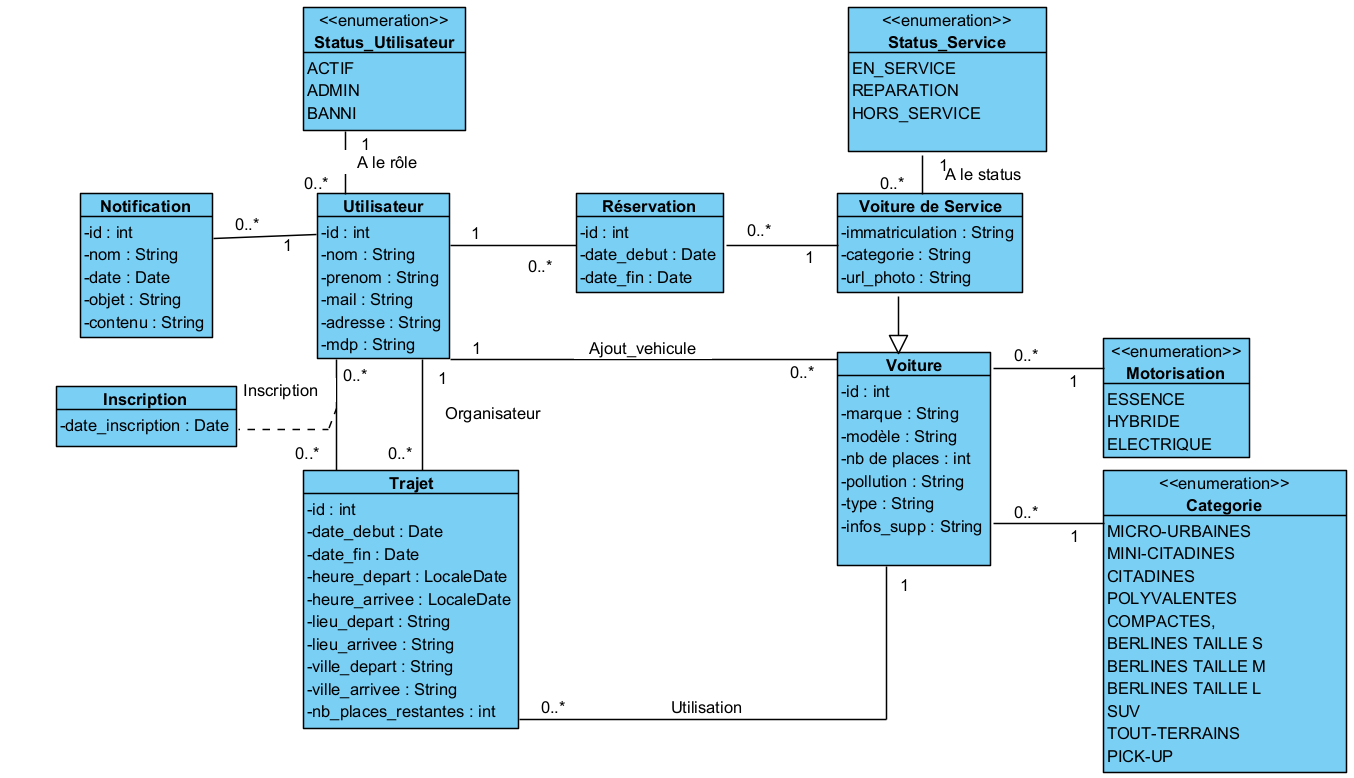
| **Nom** | **Version** |
| --- | --- |
| Spring Boot |  |
| JPA |  |
| Angular |  |
| iText (génération de PDF) |  |
| Apache POI (génération d’excel) |  |

### Serveur de base de données

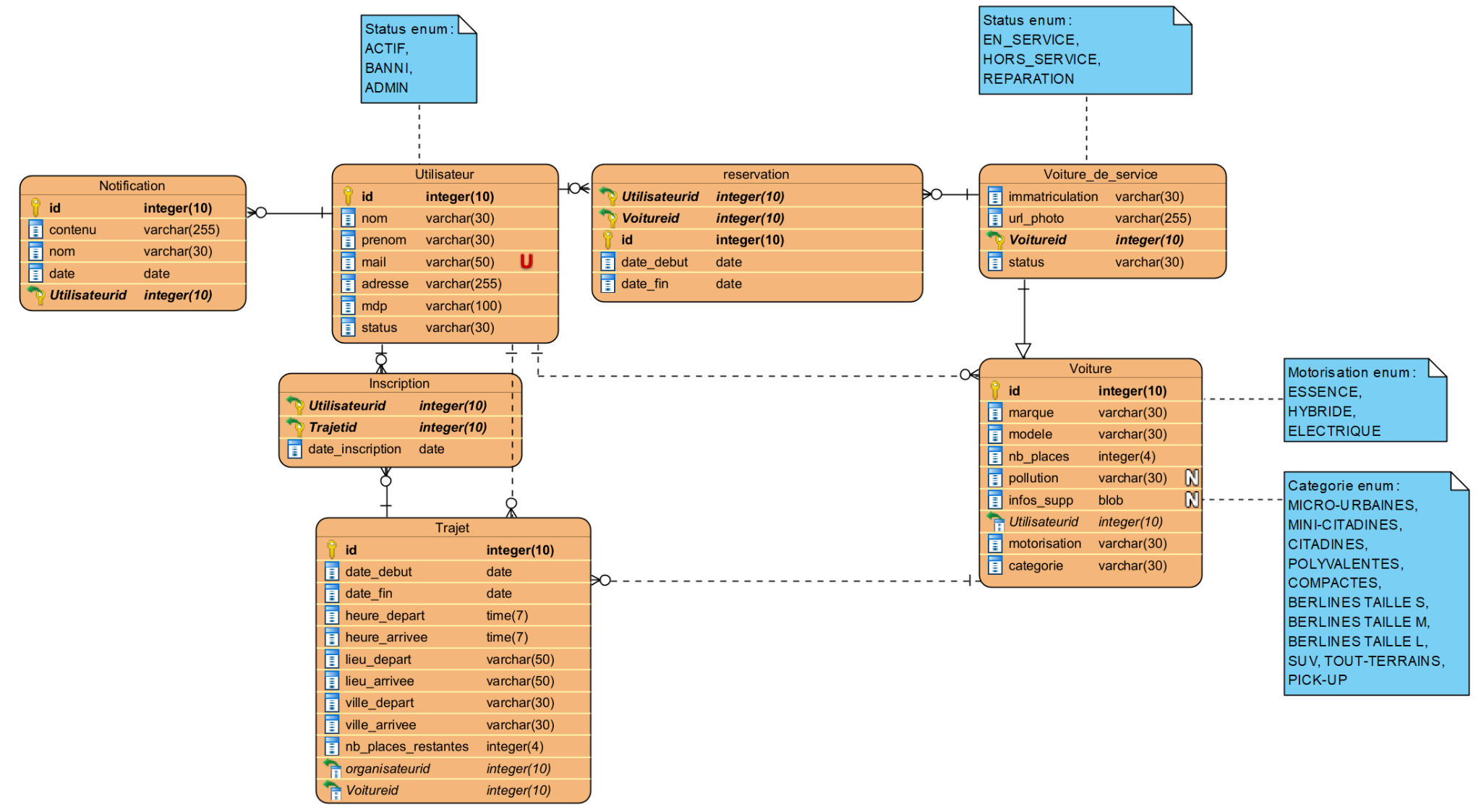
| **Nom** | **Version** |
| --- | --- |
| MySQL | 8 |
|  |  |

# FOCUS TECHNIQUE

## Diagramme de classes métier (ou MCD)



## Diagramme entités relations (ou MPD)



## Règles de développement coté back

Règles de développement :

* 100% de la Javadoc doit être renseignée
* Les règles de nommage respectent les conventions de nommage en usage en 2025.

### 4.3.1 Découpage en couches

L'architecture de l'application suit un découpage en couches respectant le flux suivant : **Contrôleurs > Services Application > Services Métier > Repository > Entités Métier + Classes Utilitaires + DTO et Mappers**

#### 4.3.1.1 Couche Contrôleur (Controllers)

**Rôle** : Point d'entrée de l'application, contient les routes d'accès au backend

* Gère les requêtes HTTP (GET, POST, PUT, DELETE)
* Validation des paramètres d'entrée
* Appel des services application
* Transformation des réponses en DTO pour le front-end

**Exemples** :

* CovoiturageController
* VehiculeServiceController
* UtilisateurController

#### 4.3.1.2. Couche Services Application (Application Services)

**Rôle** : Orchestration des cas d'utilisation complets, exécute un CU de bout en bout

* Coordonne plusieurs services métier si nécessaire
* Gère les transactions applicatives
* Assure la cohérence des cas d'utilisation

**Exemples** :

* CovoiturageApplicationService (orchestre création covoiturage + notifications)
* ReservationApplicationService (orchestre réservation + vérifications + notifications)

#### 4.3.1.3. Couche Services Métier (Domain Services)

**Rôle** : Contient la logique métier pure et les règles de gestion

* Implémente les règles métier spécifiques au domaine
* Effectue les contrôles et validations métier
* Indépendante de la technologie

**Exemples** :

* CovoiturageService (règles de création, modification, annulation)
* VehiculeService (règles de disponibilité, statuts)
* ReservationService (règles de réservation, conflits)
* NotificationService (règles d'envoi de notifications)

#### 4.3.1.4. Couche Repository

**Rôle** : Accès aux données via Spring Data JPA

* Interface entre les services métier et la base de données
* Requêtes personnalisées si nécessaire
* Gestion de la persistance

**Exemples** :

* CovoiturageRepository extends JpaRepository<Covoiturage, Long>
* VehiculeServiceRepository extends JpaRepository<VehiculeService, Long>
* UtilisateurRepository extends JpaRepository<Utilisateur, Long>

#### 4.3.1.5. Couche Entités Métier

**Rôle** : Modèle de données avec annotations JPA

* Représentation des objets métier
* Relations entre entités
* Contraintes de base de données

**Exemples** :

* Covoiturage (avec @Entity, @Table, relations @ManyToOne, @OneToMany)
* VehiculeService
* Utilisateur
* Reservation

#### 4.3.1.6. Couche DTO (Data Transfer Objects)

**Rôle** : Objets de transfert entre le contrôleur et le front-end

* Séparation entre modèle interne et exposition externe
* Contrôle des données exposées
* Adaptation du format pour le front-end

**Exemples** :

* CovoiturageDTO
* CovoiturageCreationDTO
* VehiculeServiceDTO
* UtilisateurDTO

#### 4.3.1.7. Couche Mappers

**Rôle** : Transformation entre entités et DTO

* Conversion Entity vers DTO et vice-versa
* Gestion des transformations de données
* Utilisation de MapStruct recommandée

**Exemples** :

* CovoiturageMapper
* VehiculeServiceMapper
* UtilisateurMapper

#### 4.3.1.8. Classes Utilitaires

**Rôle** : Fonctions transverses réutilisables

* Utilitaires techniques
* Formatage et transformation de données
* Helpers métier

**Exemples** :

* DateUtils (traitement des dates, calculs de durée)
* AdresseUtils (formatage d'adresses, validations)
* EmailUtils (formatage et validation des emails)
* DistanceUtils (calculs de distance et durée de trajet)

#### 4.3.1.9. Flux d'exécution type

1. **Requête HTTP** → Contrôleur
2. **Contrôleur** → Service Application
3. **Service Application** → Services Métier (un ou plusieurs)
4. **Services Métier** → Repository
5. **Repository** → Entités Métier
6. **Retour** : Entités → DTO (via Mappers) → Contrôleur → Front-end

#### 4.3.1.10. Avantages de cette architecture

1. **Séparation des responsabilités :** chaque couche a un rôle précis
2. **Testabilité :** chaque couche peut être testée indépendamment
3. **Maintenabilité :** modifications localisées selon la nature du changement
4. **Réutilisabilité :** services métier réutilisables dans différents contextes
5. **Évolutivité :** ajout de nouvelles fonctionnalités facilité

# TESTS ET INTÉGRATION

## Tests unitaires

Il y aura des tests unitaires à développer pour tester le code de toutes les classes de services.

Mettre ici le tableau de couverture des tests unitaires : lorsque vous aurez eu les 3 jours de projet sur l’intégration continue.

## Tests fonctionnels

Des campagnes de tests seront réalisées pour tester les différents cas d’utilisation.

Mettre ici le résultat de votre campagne de tests : lorsque vous serez sur la phase de finalisation du projet.

## Indicateurs de qualité de code

Renseigner à 100% la javadoc.

Règles de nommage des classes, des packages : conventions standards à respecter

Mettre ici le tableau SonarQube de qualité de code : lorsque vous aurez eu les 3 jours de projet sur l’intégration continue.