

Gestion des transports

réalisé par

**BURTE Julien**

**SAGOT Laurence**

**KORZHENEVSKIY Dmitry**

| **Révision** | **Rédacteurs** | **Date** | **Objet** |
| --- | --- | --- | --- |
| 1 | D. KORZHENEVSKIY | 01/09/2025 | 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 Gestion des transports.

Ce document présente:

* Le diagramme de classes
* Le modèle physique de données

# **SOMMAIRE**

### **Table des matières**

[**1 INTRODUCTION 2**](#_heading=)

[1.1 Objet du document 2](#_heading=h.hurt887cy8lx)

[**2 SOMMAIRE 3**](#_heading=)

[*2.1.1 Table des matières 3*](#_heading=h.n1nk116z1xmh)

[**3 ARCHITECTURE LOGICIELLE 4**](#_heading=)

[3.1 Produits et versions 4](#_heading=h.w6fa397b92qk)

[*3.1.1 Logiciels installés sur le poste de travail 4*](#_heading=h.4lqcx9ycch7r)

[*3.1.2 Langages, frameworks et librairies spécifiques 4*](#_heading=h.nlx2qiorfxdp)

[3.1.3 Serveur de base de données 4](#_heading=)

[**4 FOCUS TECHNIQUE 5**](#_heading=)

[4.1 Diagramme de classes 5](#_heading=h.97uiflf0yfon)

[4.2 Diagramme entités relations 6](#_heading=)

[4.3 Règles de développement coté back 7](#_heading=)

[**5 TESTS ET INTÉGRATION 8**](#_heading=)

[5.1 Tests unitaires 8](#_heading=h.50pkg76cnw5p)

[5.2 Tests fonctionnels 8](#_heading=)

[5.3 Indicateurs de qualité de code 8](#_heading=)

# **ARCHITECTURE LOGICIELLE**

## ***Produits et versions***

### **Logiciels installés sur le poste de travail**

| **Nom** | **Version** |
| --- | --- |
| Spring Tool Suite ou IntelliJ | IntelliJ IDEA 2025.2 (Ultimate Edition) |
| MySQL | MySQL Protocol version: 10 |
| Git bash | 2.49.0 |
| Java Development Kit | 21 |
|  |  |
|  |  |

### **Langages, frameworks et librairies spécifiques**

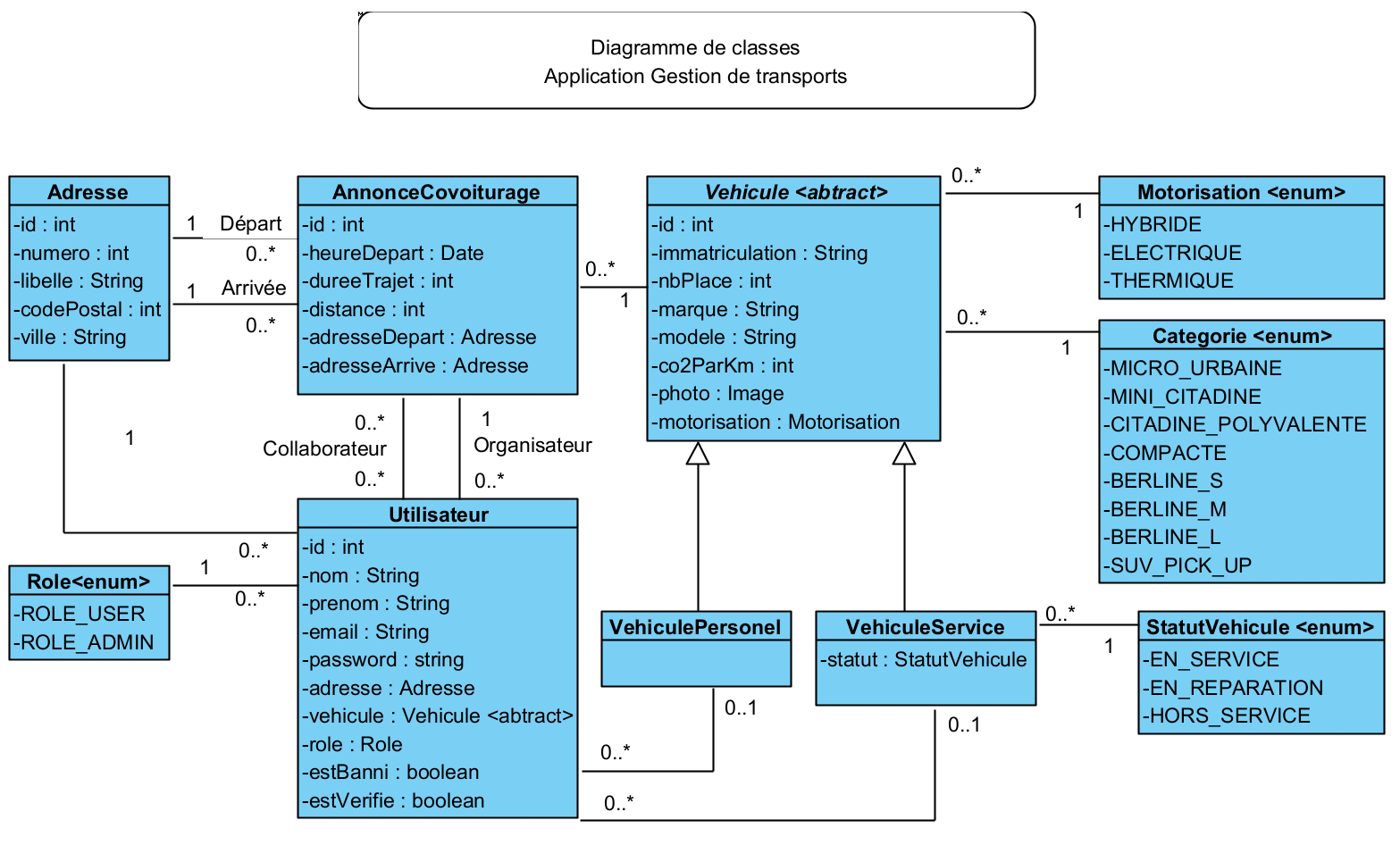
| **Nom** | **Version** |
| --- | --- |
| Spring Boot | 3.5.5 |
| JPA | 3.5.5 |
| Angular | 20 |
|  |  |
|  |  |

### **Serveur de base de données**

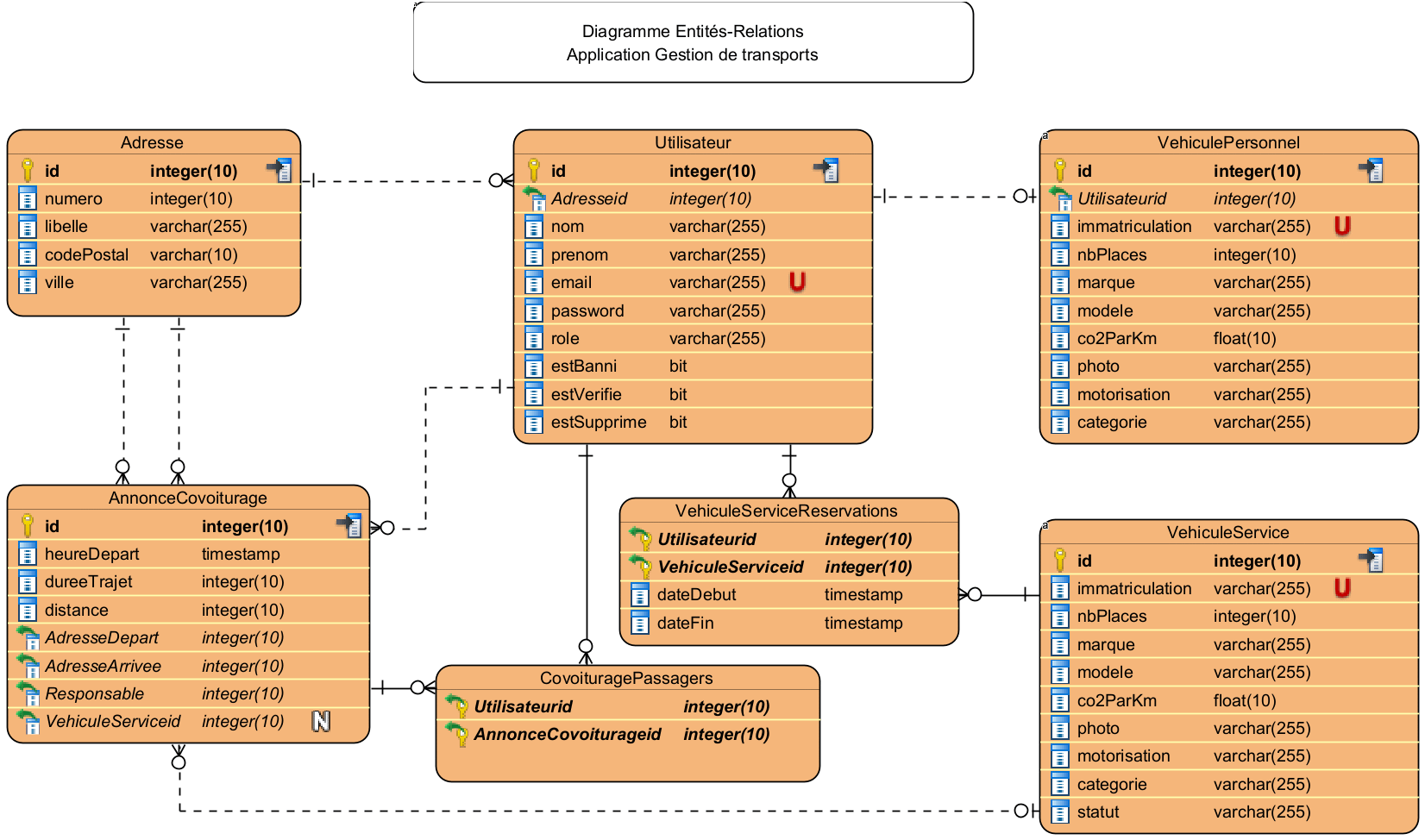
| **Nom** | **Version** |
| --- | --- |
| MySQL | 8.0.43 - MySQL Community Server |
|  |  |

# **FOCUS TECHNIQUE**

## ***Diagramme de classes***



## ***Diagramme entités relations***



## ***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.

**Découpage en couches** :

* Couche contrôleur

La couche contrôleur constitue le point d'entrée de l'application et gère les requêtes HTTP provenant du frontend Angular. Elle contient les classes annotées @RestController qui définissent les endpoints REST, valident les paramètres d'entrée, appellent les services métier appropriés et retournent les réponses JSON au client.

* Couche service

La couche service contient la logique métier de l'application et implémente les règles de gestion du cahier des charges. Les classes annotées @Service orchestrent les opérations complexes, gèrent les contrôles métier, les validations et les transactions pour réaliser chaque cas d'utilisation complet.

* Couche repository

La couche repository assure la persistance des données vers la base de données MySQL en utilisant Spring Data JPA. Les interfaces qui étendent JpaRepository fournissent les opérations CRUD et permettent d'exécuter des requêtes personnalisées, garantissant l'indépendance entre la logique métier et la technologie de persistance.

* Couche DTO

La couche DTO (Data Transfer Object) contient les objets de transfert de données utilisés pour les échanges entre le backend et le frontend Angular. Les contrôleurs renvoient exclusivement des instances de DTO vers le client plutôt que les entités JPA directement, permettant ainsi de contrôler précisément les données exposées et d'éviter les problèmes de sérialisation liés aux relations JPA.

* Couche entités

La couche entités regroupe les classes du modèle de données annotées avec les annotations JPA (@Entity, @Table, @Id, @OneToMany, etc.). Ces entités représentent la structure de la base de données et définissent les relations entre les différentes tables comme Utilisateur, AnnonceCovoiturage, VehiculeService et Adresse. Elles sont exclusivement utilisées par la couche repository pour la persistance

* Classes utilitaires

Les classes utilitaires contiennent des méthodes statiques réutilisables qui fournissent des fonctionnalités transversales à l'ensemble de l'application. Elles regroupent des traitements communs comme DateUtils pour la manipulation des dates, EmailUtils pour l'envoi d'emails, ou ValidationUtils pour les contrôles de format, évitant ainsi la duplication de code et centralisant ces fonctionnalités.

# **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.