UNIVERSITY OF BUEA

REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

P.O. Box 63,
Buea, South West Region
CAMEROON
Tel : (237) 3332 21 34/3332 26 90
Fax: (237) 3332 22 72

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

DATABASE DESIGN AND IMPLEMENTATION

By:

BERINYUY CLETUS FE21A148
EFUETLATEH JOAN FE22A197
ETAPE NGABE FE22A210
ETIENDEM PEARL FE22A211
TATA GLEN FE22A309

2024/2025 Academic Year

Table of Contents

1. Data Elements

The CarCare app stores various data elements essential for enabling car diagnostics, mechanic discovery, and service tracking. These data elements are organized into entities, each representing a core component of the system.

- **User**
  - user_id – Unique identifier
  - password
  - phone_number
  - location
  - role – Either car_owner or mechanic

- **Vehicle**
  - vehicle_id
  - user_id – Linked to User
  - model

- **DashboardImage**
  - image_id
  - user_id
  - image_url
  - fault_detected
  - upload_date

- **EngineSoundFile**
  - audio_id
  - user_id
  - audio_url
  - diagnosis_result
  - recorded_at

- **Diagnostic**
  - diagnostic_id
  - user_id
  - vehicle_id
  - image_id
  - audio_id
  - summary
  - fault_code
  - recommendation
  - created_at

- **TutorialVideo**
  - video_id
  - title
  - url
  - fault_related
  - duration
  - source

2. Conceptual Design

This section outlines the high-level structure of the CarCare app's database using MongoDB. The system follows a flexible NoSQL schema optimized for scalability and fast querying.

2.1 Key Concepts:

- **User-centric design** with car owners and mechanics stored in the same collection but differentiated by role.
- **Media integration** through image and sound file storage for diagnostics.
- **Support for car tracking** via vehicle registration and service logs.
- **Mechanic rating system** using review and feedback collections.
- **Scalable content** through tutorial videos linked to fault types.

2.2 Main Relationships:

- One User owns multiple Vehicles, Diagnostics, DashboardImages, EngineSoundFiles, and can give Feedback or write Reviews.
- A MechanicProfile is a one-to-one extension of User.
- Each Diagnostic is linked to an optional Vehicle, an uploaded DashboardImage, and an EngineSoundFile.
- A TutorialVideo can be linked to fault codes in diagnostics but is generally standalone content.

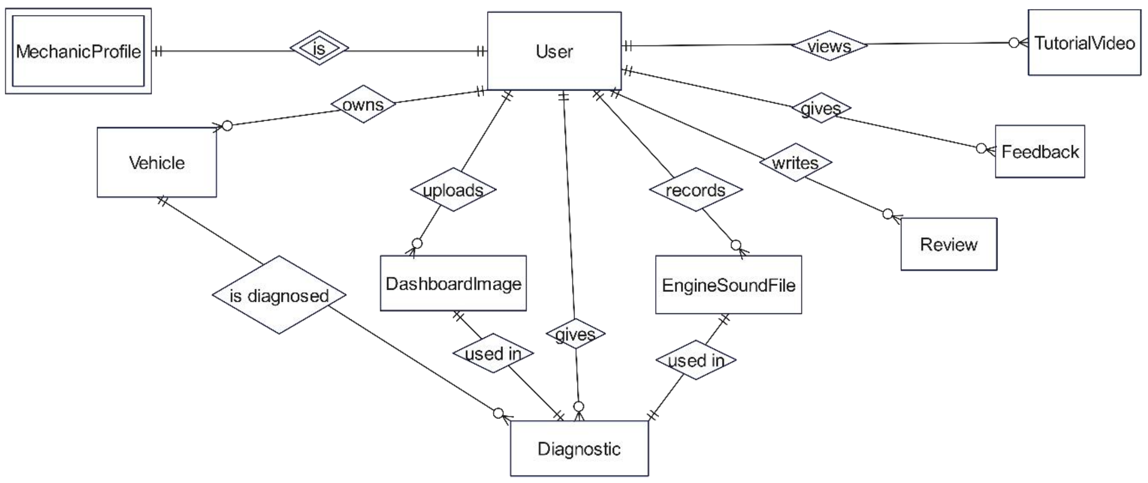
3. Entity-Relationship (ER) Diagram Summary

3.1 Strong Entities:

- User
- Vehicle
- DashboardImage
- EngineSoundFile
- Diagnostic
- TutorialVideo

3.2 Weak Entity:

- MechanicProfile – relies on User (uses user_id as both PK and FK)



4. Database Implementation (Prisma + MongoDB)

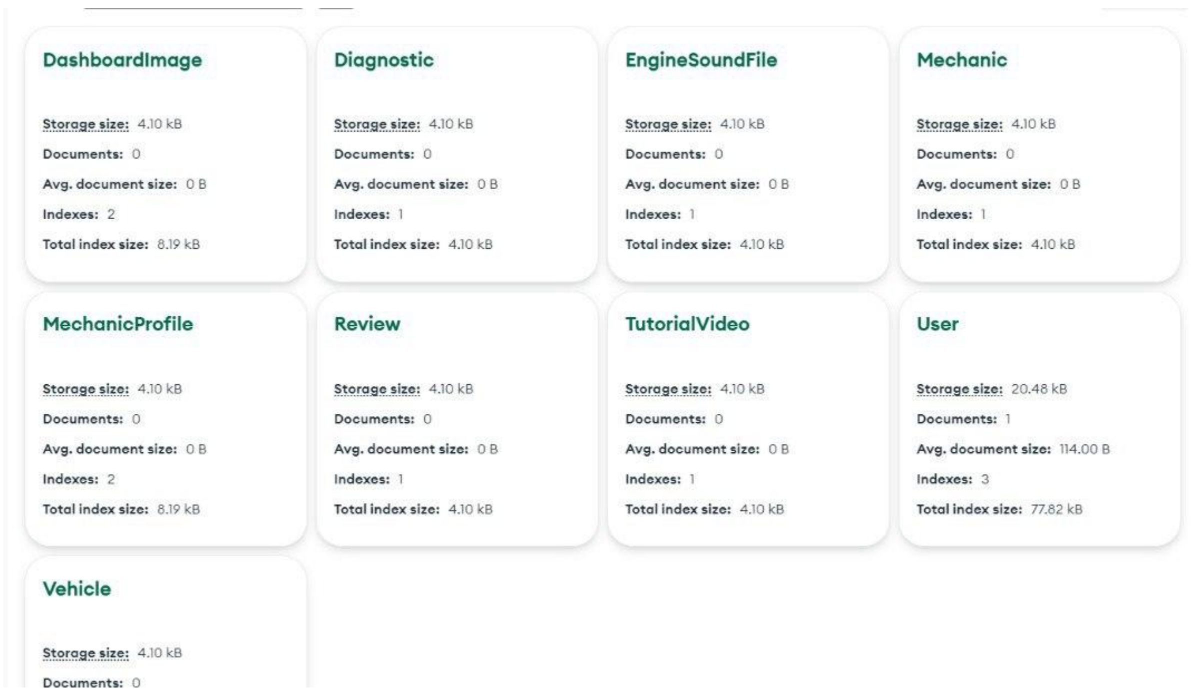
This project uses Prisma ORM with MongoDB as the database. The schema is defined in a schema prisma and models the relationships between users, users' vehicles, diagnostics, media files, reviews, and tutorial videos.

4.1 Key Concepts

- **User Roles:** There are two main user roles: `car_owner` and `mechanic`
- **User:** Stores user details, authentication info, and relations to vehicles, uploaded media, reviews, and mechanic profile.
- **MechanicProfile:** Contains additional info for users who are mechanics (experience, specialization, etc.).
- **Vehicle:** Represents a car owned by a user, with a relation to diagnostics performed on it.
- **DashboardImage & EngineSoundFile:** Media files uploaded by users for diagnostic purposes, each linked to the uploading user and related diagnostics.
- **Review:** Users can write reviews for diagnostics, with each review linked to a user and a diagnostic.
- **TutorialVideo:** Stores YouTube video url or other tutorial videos related to specific diagnostics and faults.
- **Diagnostic:** Central model representing a diagnostic event, linking to a vehicle, optional dashboard image and engine sound, reviews, and tutorial videos.

4.2 Relationships

- **User ↔ Vehicle:** One-to-many (a user can own multiple vehicles).
- **User ↔ DashboardImage/EngineSoundFile:** One-to-many (a user can upload multiple images/sounds).
- **User ↔ Review:** One-to-many (a user can write multiple reviews).
- **User ↔ MechanicProfile:** One-to-one (if the user is a mechanic).
- **Vehicle ↔ Diagnostic:** One-to-many (a vehicle can have multiple diagnostics).
- **DashboardImage/EngineSoundFile ↔ Diagnostic:** One-to-many (a media file can be used in multiple diagnostics).
- **Diagnostic ↔ Review/TutorialVideo:** One-to-many (a diagnostic can have multiple reviews and tutorial videos).


DashboardImage
Storage size: 4.10 kB
Documents: 0
Avg. document size: 0 B
Indexes: 2
Total index size: 8.19 kB

Diagnostic
Storage size: 4.10 kB
Documents: 0
Avg. document size: 0 B
Indexes: 1
Total index size: 4.10 kB

EngineSoundFile
Storage size: 4.10 kB
Documents: 0
Avg. document size: 0 B
Indexes: 1
Total index size: 4.10 kB

Mechanic
Storage size: 4.10 kB
Documents: 0
Avg. document size: 0 B
Indexes: 1
Total index size: 4.10 kB

MechanicProfile
Storage size: 4.10 kB
Documents: 0
Avg. document size: 0 B
Indexes: 2
Total index size: 8.19 kB

Review
Storage size: 4.10 kB
Documents: 0
Avg. document size: 0 B
Indexes: 1
Total index size: 4.10 kB

TutorialVideo
Storage size: 4.10 kB
Documents: 0
Avg. document size: 0 B
Indexes: 1
Total index size: 4.10 kB

User
Storage size: 20.48 kB
Documents: 1
Avg. document size: 114.00 B
Indexes: 3
Total index size: 77.82 kB

Vehicle
Storage size: 4.10 kB
Documents: 0

4.3 Implementation Notes

- All models use MongoDB's ObjectId as the primary key.
- Relations are explicitly defined using Prisma's `@relation` attribute.
- The schema supports advanced queries and population of related data via Prisma Client.
- The design supports extensibility for new media types, diagnostic methods, or user roles.

4.4 File Reference

- **Prisma schema:** `prisma/schema.prisma`
- **Generated Prisma Client:** Used in `src/routes.ts` and controllers for all DB operations.

4.5 Example Entity Flow

1. A user (car owner) uploads a dashboard image or engine sound.
2. The system creates a `DashboardImage` or `EngineSoundFile` record linked to the user.
3. A `Diagnostic` is created, referencing the vehicle, uploaded media, and storing the diagnosis result.
4. The user (or mechanic) can add a `Review` for the diagnostic.
5. The system can link relevant `TutorialVideo` records to the diagnostic for user guidance.

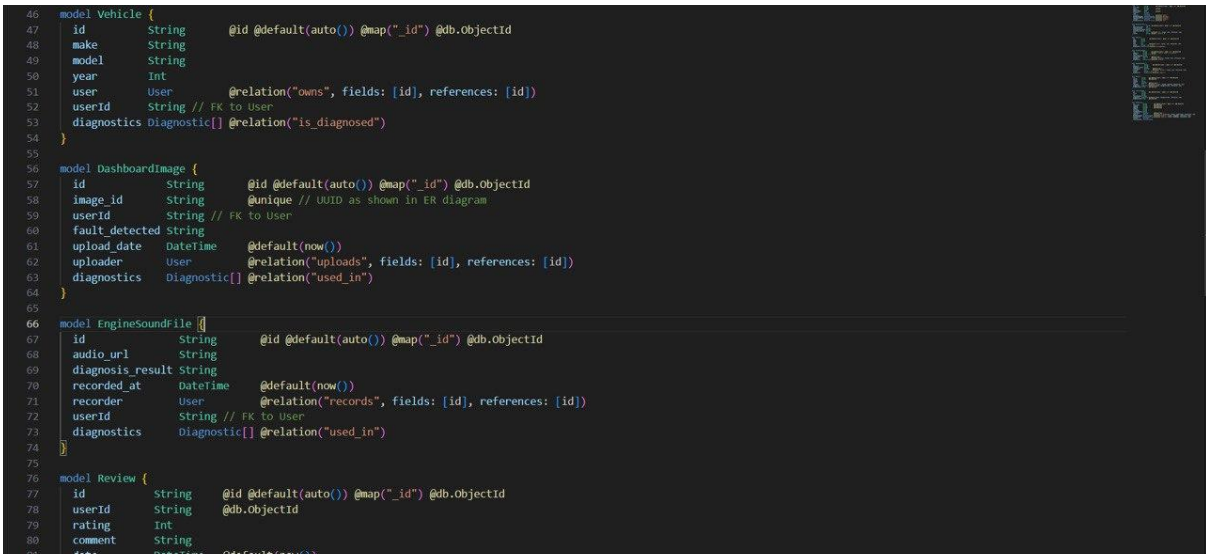
5. Backend Implementation

This backend is built with Node.js, Express, TypeScript, and uses Prisma ORM to interact with a MongoDB database. It powers the CarCare car fault diagnosis system, providing RESTful API endpoints for user management, vehicle and diagnostic data, media uploads, reviews, and AI-powered car fault diagnosis.

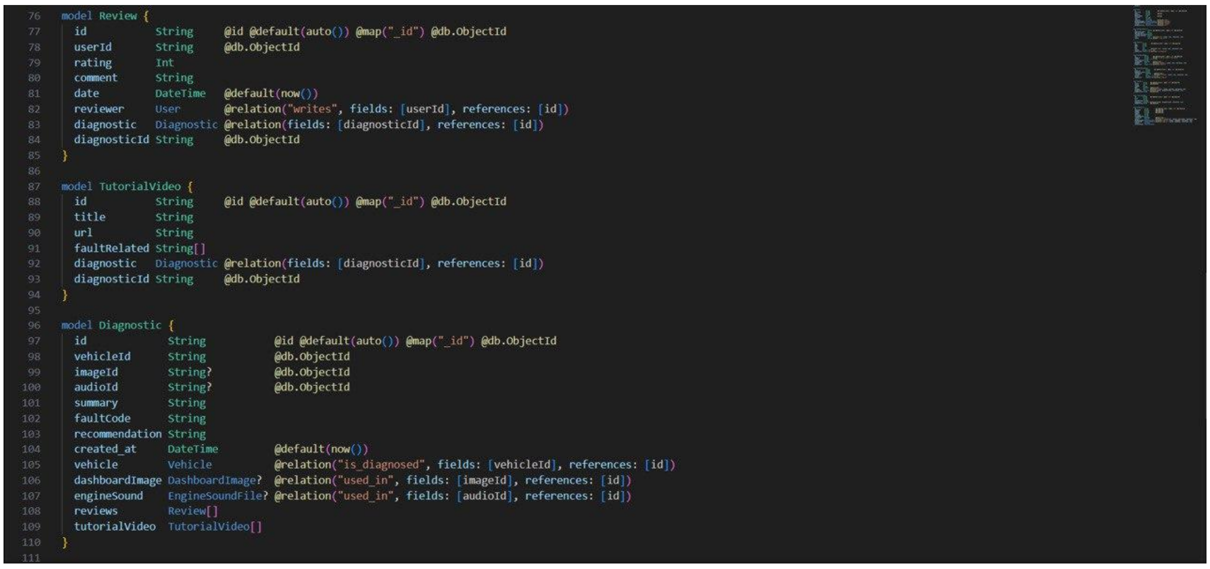
5.1 Key Technologies

- **Node.js & Express:** For building scalable REST APIs.
- **TypeScript:** For type safety and maintainability.
- **Prisma ORM:** For database modeling and queries (MongoDB as the database).

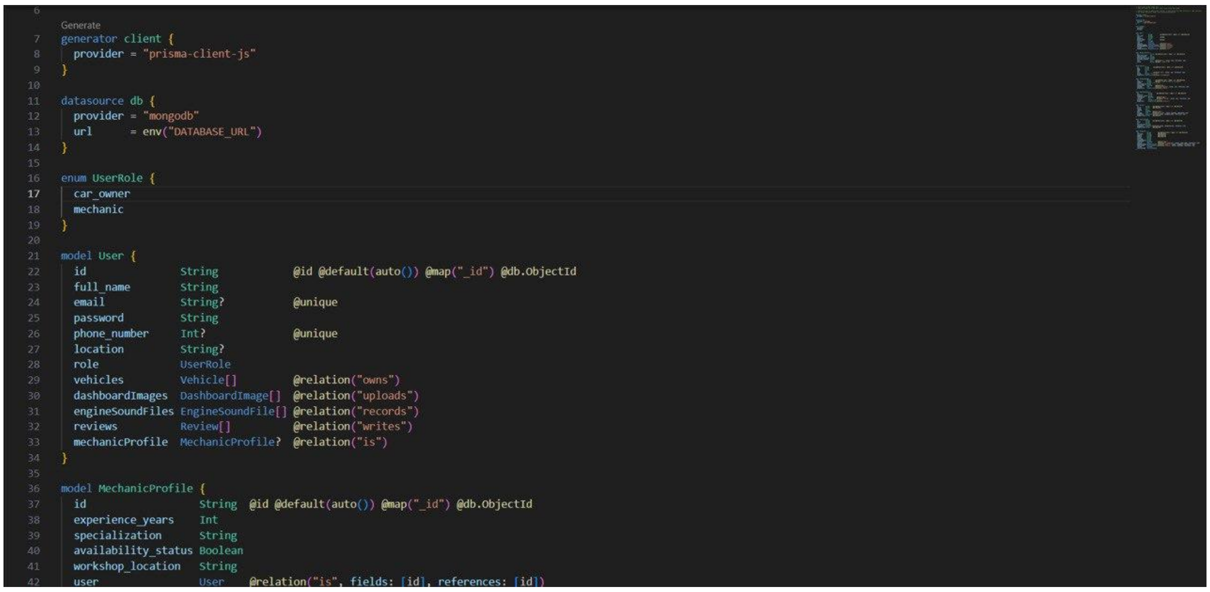
- **Mongoose (optional):** Alternative routes provided for direct MongoDB access.
- **Swagger:** For API documentation and testing.
- **Multer:** For handling file uploads (dashboard images, engine sounds).
- **JWT & bcryptjs:** For authentication and password security.
- **Axios:** For calling external APIs (AI diagnosis, YouTube search).



```
model Vehicle {
  id          String      @id @default(auto()) @map("_id") @db.ObjectId
  make        String
  model       String
  year        Int
  user        User        @relation("owns", fields: [id], references: [id])
  userId      String // FK to User
  diagnostics Diagnostic[] @relation("is_diagnosed")
}

model DashboardImage {
  id             String      @id @default(auto()) @map("_id") @db.ObjectId
  image_id       String      @unique // UUID as shown in ER diagram
  userId         String // FK to User
  fault_detected String
  upload_date    DateTime    @default(now())
  uploader       User        @relation("uploads", fields: [id], references: [id])
  diagnostics    Diagnostic[] @relation("used_in")
}

model EngineSoundFile {
  id               String      @id @default(auto()) @map("_id") @db.ObjectId
  audio_url        String
  diagnosis_result String
  recorded_at      DateTime    @default(now())
  recorder         User        @relation("records", fields: [id], references: [id])
  userId           String // FK to User
  diagnostics      Diagnostic[] @relation("used_in")
}

model Review {
  id           String     @id @default(auto()) @map("_id") @db.ObjectId
  userId       String     @db.ObjectId
  rating       Int
  comment      String
```



```
model Review {
  id           String     @id @default(auto()) @map("_id") @db.ObjectId
  userId       String     @db.ObjectId
  rating       Int
  comment      String
  date         DateTime   @default(now())
  reviewer     User       @relation("writes", fields: [userId], references: [id])
  diagnostic   Diagnostic @relation(fields: [diagnosticId], references: [id])
  diagnosticId String     @db.ObjectId
}

model TutorialVideo {
  id           String     @id @default(auto()) @map("_id") @db.ObjectId
  title        String
  url          String
  faultRelated String[]
  diagnostic   Diagnostic @relation(fields: [diagnosticId], references: [id])
  diagnosticId String     @db.ObjectId
}

model Diagnostic {
  id             String           @id @default(auto()) @map("_id") @db.ObjectId
  vehicleId      String           @db.ObjectId
  imageId        String?          @db.ObjectId
  audioId        String?          @db.ObjectId
  summary        String
  faultCode      String
  recommendation String
  created_at     DateTime         @default(now())
  vehicle        Vehicle          @relation("is_diagnosed", fields: [vehicleId], references: [id])
  dashboardImage DashboardImage?  @relation("used_in", fields: [imageId], references: [id])
  engineSound    EngineSoundFile? @relation("used_in", fields: [audioId], references: [id])
  reviews        Review[]
  tutorialVideo  TutorialVideo[]
}
```



```
Generate
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mongodb"
  url      = env("DATABASE_URL")
}

enum UserRole {
  car_owner
  mechanic
}

model User {
  id               String           @id @default(auto()) @map("_id") @db.ObjectId
  full_name        String
  email            String?          @unique
  password         String
  phone_number     Int?             @unique
  location         String?
  role             UserRole
  vehicles         Vehicle[]        @relation("owns")
  dashboardImages  DashboardImage[] @relation("uploads")
  engineSoundFiles EngineSoundFile[] @relation("records")
  reviews          Review[]         @relation("writes")
  mechanicProfile  MechanicProfile? @relation("is")
}

model MechanicProfile {
  id                  String  @id @default(auto()) @map("_id") @db.ObjectId
  experience_years    Int
  specialization      String
  availability_status Boolean
  workshop_location   String
  user                User    @relation("is", fields: [id], references: [id])
```

5.2 Project Structure

- `src/index.ts` — Main server entry point, Express app setup, Swagger docs, and DB connection.
- `src/routes.ts` — Main API routes using Prisma.
- `src/routes.mongoose.ts` — Alternative API routes using Mongoose.
- `src/swagger.ts` — Swagger documentation setup.
- `prisma/schema.prisma` — Prisma schema defining all models and relationships.
- `DATABASE.md` — Database design and relationship documentation.

6. Database Connection

6.1 Prisma:

- The connection string is set in `.env` as `DATABASE_URL` (must be a valid MongoDB URI).

- Prisma Client is initialized in `src/index.ts` and used in all controllers/routes for DB operations.

6.2 Main Features

- **User Authentication:** Signup/login with JWT, password hashing with bcryptjs.

- **Role-based Users:** Car owners and mechanics, with mechanic profiles.

- **Vehicle Management:** CRUD for vehicles owned by users.

- **Media Uploads:** Dashboard images and engine sound files, linked to users and diagnostics.

- **Diagnostics:** Central model linking vehicles, media, reviews, and tutorial videos.

- **AI Diagnosis:** `/diagnose` endpoint accepts image/sound, calls external AI APIs, and returns a diagnosis.

- **YouTube Integration:** After diagnosis, the system searches YouTube for a relevant tutorial and returns the video URL.

- **Reviews & Tutorials:** Users can review diagnostics and view related tutorial videos.

- **Swagger Docs:** All endpoints are documented and testable at `/api-docs`.

- All database models and relationships are defined in `prisma/schema.prisma`.

- The backend is modular and can be extended with new features or endpoints as needed.

7. Conclusion

In this task, we have successfully designed the foundational elements of the CarCare app's database, focusing on data elements, conceptual design, and the ER diagram. The system's data structure is designed to manage key entities such as Users, MechanicProfiles, Vehicles, Diagnostics, and Reviews. These entities are interconnected to facilitate seamless interactions between car owners and mechanics.

The conceptual design follows a flexible and scalable approach, leveraging MongoDB for efficient data storage and querying. The relationships between entities, including ownership, diagnostic data, and media files (such as images and audio), have been clearly mapped out.

The ER diagram provides a visual representation of the relationships between entities, indicating which are strong or weak entities, as well as identifying multivalued and derived attributes. With this foundational work completed, the next steps will involve moving into database implementation, backend development, and connecting the database to the application's backend.

The successful design of the database schema lays the groundwork for the effective functioning of the CarCare app, ensuring that data management and retrieval will support all future functionality efficiently.