**Database Final Project Report**

**CS631853 Weiyu Li & Jin Cheng**

**1. Introduction**

**Goal and Structure of the System**

This project is a simple car dealership user management system that follows a modular and layered architecture to ensure maintainability and scalability. It integrates three core applications, designed to manage a car dealership's data and streamline operations.

* **The Frontend Application** provides an interactive user interface for managing customers, vehicles, and sales data.
* **The Backend Application** acts as a REST API layer, implementing business logic and ensuring secure interaction with the database.
* **The Database Validation Application** automates validation processes, ensuring data consistency and integrity.

Currently, the system is using Docker Compose for deployment, initializing the MySQL database, the backend server, and the frontend application. This setup is intended to act as a test platform for software engineers to manage the car dealership data efficiently, incorporating direct and indirect validation mechanisms to enforce data constraints and maintain entity integrity.

**Project Scope and Assumptions**

The project’s database design is rooted in an ER-Diagram and a provided data schema (with detailed references available in the [README.md](https://github.com/Livia-1212/cs631_final/blob/master/README.md) file). Key assumptions include default configurations for MySQL, such as port, username, and password settings. Users deploy the system using Docker, while database management is facilitated via MySQL Workbench. Data validation is prioritized, with checks embedded in both the database schema and the backend logic to ensure consistency and compliance with defined constraints. Additionally, the Database Validation Application provides an added layer of automation, verifying updates and modifications. This design ensures ease of use, scalability, and a consistent environment for testing and development.

**2. System Requirements**



**Functional Requirements: Key Features**

* **REST API**
  + Facilitates interactions between the frontend and backend applications, supporting CRUD operations for dealership data (e.g., customers, vehicles, sales).
  + Endpoints include operations such as creating, reading, updating, and deleting entities like Customer, Vehicle, and Sales.
* **CORS Support**
  + Implements Cross-Origin Resource Sharing (CORS) to enable secure communication between the frontend application (hosted locally or remotely) and the backend server.
  + Configurable settings allow for flexible control over allowed origins, HTTP methods, and credentials.
* **Docker-Based Deployment**
  + Uses **Docker Compose** to streamline system setup, initializing the database, backend, and frontend as services.
  + Ensures consistent environments across different development and production setups.
* **Frontend Application**
  + Interactive interface for managing dealership data, built using modern JavaScript frameworks.
  + Includes components for managing **Customer**, **Vehicle**, and **Sales** records with seamless user experience.
  + API integration to fetch, display, and manipulate data dynamically.
* **Backend Business Logic**
  + Spring Boot-based backend implementing validation and processing logic for data received from the frontend.
  + Provides robust error handling and ensures adherence to defined data constraints (e.g., no duplicate VINs, valid sales records).
* **Database Validation Application**
  + Automates data validation by running scripts that ensure schema compliance, entity integrity, and referential integrity for MySQL data.
  + Includes checks for foreign key dependencies and business rules (e.g., ensuring all sales records link to valid customers and vehicles).
* **MySQL Database**
  + Schema includes tables for Customer, Vehicle, Sales, and related entities, ensuring normalization and integrity.
  + Includes pre-seeded data for testing, and supports automated creation and validation scripts.
* **Data Validation Mechanisms**
  + Embedded validation in backend services for API data.
  + Database-level constraints to ensure valid, consistent, and complete data (e.g., unique keys, not-null constraints).
  + External Python-based automation script to validate data updates and ensure adherence to integrity rules.
* **Customizable Configuration**
  + Application settings for database connectivity, CORS, and environment variables can be adjusted via configuration files.
  + Includes support for multi-environment deployment (e.g., development, staging, production).
* **Health Checks**
  + Docker services include health checks for database readiness (e.g., mysqladmin ping).
  + Backend service ensures API availability and database connectivity.
* **Search and Filtering Capabilities**
  + Allows for advanced search (e.g., searching vehicles by VIN, customers by name) and filtering of data through API endpoints.
* **Scalability**
  + Modular architecture supports adding new modules or features, such as integrating service appointments or parts inventory management.
* **Additions or Changes**: Note any modifications to the original requirements.

**3. Database Design**

**3.1 ER Diagram**

This project schema is not directly based on the ER-Diagram reference provided in the link, but it is a similar representation. [ER-Diagram](https://drive.google.com/file/d/183xLD1TsGAuvct3AC0KwaiUeueA9yWRK/view?usp=drive_link)

This is the [ER-Diagram](https://drive.google.com/file/d/1MiAQvyQXmPOjU8QzwXUQuKGdaeWfPnKd/view?usp=drive_link) our team created for the final schema.

**3.2 Relational Schema**

* Summarize the schema with primary keys, foreign keys, and a few key tables:

Customer(Customer\_ID (PK), FirstName, LastName, Contact\_number, email, is\_returning\_customer)

Vehicle(Vehicle\_ID (PK), make, model, vin, price, cost\_price, sold\_status)

Salesperson(Salesperson\_ID (PK), FirstName, LastName, Contact\_number)

Sales(Sale\_ID (PK), Customer\_ID (FK), Vehicle\_ID (FK), sold\_price, sale\_date, Salesperson\_ID(FK))

Servicepackage (PackageID (PK), PackageName, CarAge, LaborCost)

ServiceAppointment (Appointment\_ID (PK), Customer\_ID (FK), Vehicle\_ID (FK), ScheduledTime, Package\_ID (FK), EstimatedTime, AppointmentStatus)

ServiceDetail (Service\_ID (PK), Appointment\_ID(FK), ArrivalTime, PickupTime, ServicePerformed, PartsUsed, LaborHours, TotalCost)

Parts (PartID (PK), PartName, Price, StockQuantity)

**3.3 Design Decisions**

* Steps-by-Steps Mapping by [report](https://drive.google.com/file/d/1qodh2vqWYaIu1mbUI1j6o-MFaOYQgMgZ/view?usp=drive_link)

A computer screen shot of a black screen

Description automatically generated**3.4 Screenshot of the Application**

**4. User Manual**

1. Setup: Use Docker Compose to initialize the application. Ensure Docker is installed, and run docker-compose up to start the backend, frontend, and MySQL services.
2. Database Management: Access MySQL Workbench to manage the dealership database. Use provided SQL scripts (create\_table.sql and seed.sql) to create and populate the database.
3. Frontend: Open the application in a browser at http://localhost:5173 to interact with the system.
4. Features: Add customers, vehicles, sales, and appointments. View and manage dealership data via a user-friendly interface.
5. Validation: All inputs are validated to ensure data integrity.

**6. Appendix**

A computer screen with text and numbers

Description automatically generated The seed.sql provides dummy mysql data to populate the database. For example:

**7. Conclusion**

The project successfully developed a modular car dealership management system, integrating a robust MySQL database with frontend and backend applications. Core functionalities include customer and vehicle management, sales tracking, and service appointments, ensuring data consistency through validation mechanisms. Docker streamlines deployment, while REST APIs facilitate seamless interactions.

For future improvements, extending functionalities to include advanced analytics, real-time inventory updates, and user access control would enhance usability. Additionally, automating service notifications and integrating payment systems would provide a comprehensive solution. Refining the user interface for better accessibility and optimizing database queries for scalability remain areas for further enhancement.