# **Transport Management System**

* **BY GETSY JACINTH**

**1. Project Overview**

The **Transport Management System (TMS)** is a modular, Python-based software solution designed to manage the core functions of a transport service provider. It automates and centralizes vehicle scheduling, route management, trip booking, driver allocation, and passenger interaction, all backed by a robust **MySQL** relational database.

Developed using a structured folder architecture and aligned with **object-oriented programming (OOP)** principles, the system demonstrates the application of key software engineering concepts including **database connectivity (via PyMySQL or MySQL Connector/Python)**, **custom exception handling**, and **unit testing using unittest**.

**2. Purpose of the Project**

The purpose of this project is to address inefficiencies in manual transport operations through a digital platform that is:

* Scalable and easy to maintain due to modular OOP design.
* Efficient in handling real-time trip and vehicle data.
* Accurate in recording, retrieving, and managing bookings and driver allocations.

This system aims to replicate operations found in small-to-medium logistics or passenger transport firms, enabling students to simulate real-world enterprise-level transport operations within an academic context.

By building this from scratch in Python, the project also highlights skills in:

* SQL schema design and foreign key constraints.
* Layered application architecture.
* File-based configuration management using .properties or .ini files.
* Exception handling via custom exception classes.
* CLI-based user interface for streamlined interaction.

**3. Project Scope**

**3.1 Functional Scope**

The system includes the following core features:

**Vehicle Management**

* Add, update, and remove vehicles.
* Maintain attributes like model, type (Truck/Van/Bus), capacity, and operational status (Available, On Trip, Under Maintenance).

**Route & Trip Management**

* Manage routes with origin, destination, and distance.
* Schedule or cancel trips with specified vehicles and routes, including departure/arrival timestamps, trip status (Scheduled, Completed, Cancelled), and passenger limits.

**Passenger and Booking Management**

* Register passengers with contact details.
* Book and cancel trips for passengers.
* Retrieve booking history by passenger or trip.

**Driver Allocation**

* Allocate and deallocate drivers to trips.
* Ensure that drivers are only assigned when available.
* Provide a function to list all available drivers.

**Data Persistence and Interaction**

* MySQL schema includes normalized tables for Vehicles, Routes, Trips, Passengers, and Bookings.
* CRUD operations are executed via Python's database connectivity layer using cursor-based SQL execution.

**Error Handling**

* Use of custom exceptions like VehicleNotFoundException, BookingNotFoundException for controlled failure scenarios.
* All exceptions are raised in the DAO layer and handled gracefully in the main application layer.

**CLI-Based UI**

* A menu-driven interface using the Python input() function simulates system interaction for non-technical users.

**3.2 Non-Functional Scope**

* **Performance:** Efficient SQL joins and indexing to handle high-volume data access.
* **Testability:** Unit tests ensure each function behaves as expected and validates edge cases and exceptions.
* **Security:** Configured SQL access using credentials stored in a secured .properties or .env file.
* **Maintainability:** Clear separation of logic using packages like:
  + entity → for data models.
  + dao → for database access and service logic.
  + exception → for user-defined exceptions.
  + util → for DB utilities and configuration loading.
  + app → for the CLI driver program.

**4. Architectural and Technical Constraints**

* **Language:** Python 3.10+
* **Database:** MySQL (preferably version 8.x)
* **Connection Library:** mysql-connector-python or PyMySQL
* **Testing Framework:** unittest or pytest
* **IDE:** VS Code
* **Version control:** GitHub
* **Folder Structure:**

/entity # Model classes

/dao # Database access & service layer

/exception # Custom exceptions

/util # DB connection/config utilities

/app # CLI interface (main menu)

* **Connectivity:** DB connection strings are dynamically loaded from .properties or .ini files using the configparser module.

**5. Assumptions and Limitations**

* System is terminal-based; no GUI or web interface included.
* Only basic validations are included (e.g., duplicate entries, missing references).
* No real-time tracking; all operations are static and manually triggered by user input.
* Admin role is assumed for all operations (no authentication/authorization logic included).
* No concurrency control or multi-user access implemented.

**5.Conclusion**

Defining the purpose and scope of the Transport Management System lays a solid foundation for aligning the system's objectives with its implementation. It ensures clarity of direction, effective resource allocation, and measurable deliverables. This documentation acts as the guiding framework for the remaining stages in the Software Development Life Cycle (SDLC)—from requirement analysis to implementation and testing—thus ensuring academic rigor and professional-quality submission.