**Gathering Requirements and Conducting Analysis**

**(Transport Management System – Python & MySQL)**

-BY GETSY JACINTH

**2.1 Identify the Problem or Opportunity**

Manual transport management processes often cause booking conflicts, delays, and poor tracking of vehicles and drivers. Most small to medium-sized transport firms use spreadsheets or unconnected tools, which leads to inefficiency, miscommunication, and human errors.

There is a strong opportunity to develop a **centralized, automated system** that handles core transport operations through Python code and MySQL. This would not only streamline the workflow but also reduce the need for human intervention in repetitive tasks like booking, allocation, and tracking.

Additionally, this project allows students to simulate a real-world transport business while gaining hands-on experience with **Python programming**, **database design**, **exception handling**, and **software engineering best practices**.

**2.2 Define Goals and Objectives (SMART)**

* **Specific:** Develop a Python-based command-line application to manage vehicles, trips, routes, drivers, and passenger bookings efficiently.
* **Measurable:** The system should handle at least 100 records per table and return query results in under 2 seconds.
* **Achievable:** Implement using standard Python libraries (mysql-connector-python, unittest) and a MySQL database.
* **Realistic:** Designed to be completed within 3–4 weeks, with core functionalities delivered on time.
* **Time-bound:** Final project submission will follow the academic schedule and internal deadlines.

**2.3 Determine the Scope**

**Functional Scope – Included in the Project**

* Add, update, and delete vehicle records.
* Create and manage route entries with origin, destination, and distance.
* Schedule trips by assigning vehicles and setting departure/arrival times.
* Cancel trips by updating their status.
* Register passengers with details such as name, gender, age, email, and phone number.
* Book and cancel trips for passengers.
* Retrieve booking history by passenger ID or trip ID.
* Allocate and deallocate drivers for scheduled trips.
* List available drivers who are not currently assigned.
* Display a menu-driven user interface through the terminal.
* Handle all operations through a MySQL database.
* Use custom exceptions for controlled error handling.
* Test all core functions using Python’s **unittest** framework.

**Constraints**

* The application will be built using **Python 3.10+** and **MySQL 8.x**.
* All database credentials and configurations will be read from .ini or .properties files.
* Terminal-only execution; no graphical front-end.
* Dates must be entered in the format YYYY-MM-DD HH:MM:SS.
* A Driver table will be manually created for allocation purposes, even if not included in the original schema.

**2.4 Break Down Requirements**

**1. Vehicle Management**

* Add new vehicles with model, capacity, type, and status.
* Update existing vehicle details.
* Delete vehicles by their ID.

**2. Route Management**

* Create routes with start and end locations and distance.
* (Optional) View the list of available routes.

**3. Trip Management**

* Schedule trips by assigning a vehicle to a route with departure and arrival times.
* Cancel trips by updating their status.
* (Optional) Automatically update trip status (e.g., Scheduled → In Progress).

**4. Passenger and Booking Management**

* Register passengers with personal and contact information.
* Book trips for registered passengers.
* Cancel bookings using booking ID.
* Retrieve booking history by passenger or trip.

**5. Driver Allocation**

* Allocate drivers to trips.
* Deallocate drivers from trips.
* View all available (unassigned) drivers.

**6. Database and Exception Handling**

* Perform CRUD operations using SQL.
* Raise VehicleNotFoundException for invalid vehicle IDs.
* Raise BookingNotFoundException for invalid booking queries.
* Gracefully handle connection failures and query errors.

**7. Command-Line Interface (CLI)**

* Present a text-based menu to the user.
* Accept input using input() and trigger the corresponding action.

**8. Unit Testing**  
Use **unittest** to verify:

* Vehicle addition and update
* Trip booking and cancellation
* Driver assignment and removal
* Exception handling for invalid actions

**2.5 Prioritize Requirements**

**High Priority (Must-Have)**

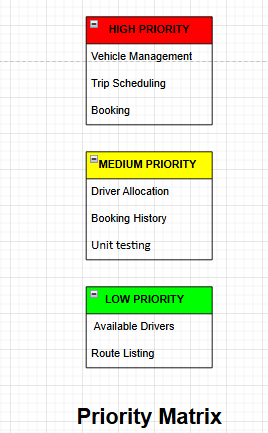
* Vehicle management
* Trip scheduling and cancellation
* Passenger registration and booking
* Exception handling
* Database connection
* Command-line menu interface

**Medium Priority (Important Added After Core)**

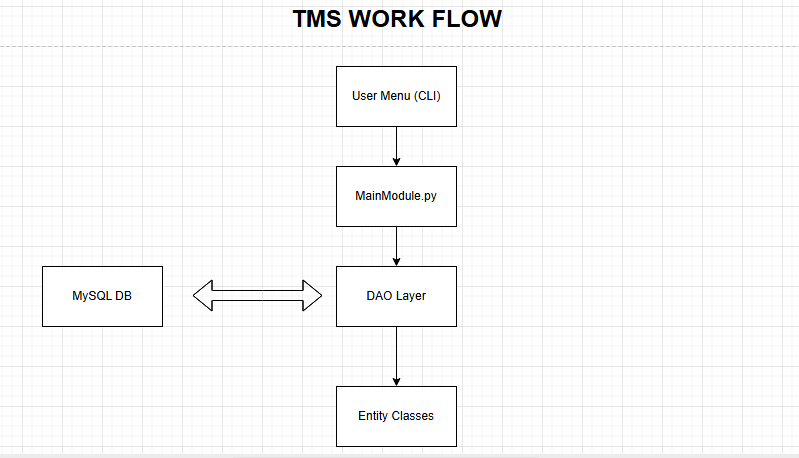
* Driver allocation and deallocation
* Booking retrieval by passenger/trip
* Unit testing coverage

**Low Priority (Optional Enhancements)**

* Displaying available drivers
* Viewing all routes or vehicles
* Auto transition of trip status



**2.6 Basic work Flow diagram**

****

**Summary**

This step defines **what the project aims to solve**, **what it will include**, and **how it will function** at a high level. By identifying the real-world problem, setting SMART objectives, and breaking down the requirements, we have ensured that the project has clear direction and boundaries.

This level of detail also helps prevent scope creep, guides your coding and testing process, and ensures that your submission aligns with academic and technical expectations.