Travel Agency Management System - Architectural Document

Shreya M Hegde PES1UG22CS573

Shreya Mittal PES1UG22CS575

1. Introduction

1.1 Purpose

The purpose of the **Travel Agency Management System (TADMS)** is to automate the process of booking vehicles for road trips. The system facilitates seamless management of customer bookings, vehicle inventory, driver assignments, and payment processing. The system uses a MySQL database to securely and efficiently store customer, vehicle, and booking information, ensuring data accuracy and preventing double bookings.

1.2 Scope

The system serves as a stand-alone web-based platform for travel agencies to manage vehicle bookings and streamline trip logistics. It includes the following functionalities:

- **Customers** can register, browse available vehicles, and make bookings.
- **Administrators** manage vehicle inventory, driver assignments, and system configurations.
- **Drivers** are assigned to trips and can view their schedules through the system.

1.3 Definitions, Acronyms, and Abbreviations

- **CRUD:** Create, Read, Update, Delete – basic operations for managing data
- **API:** Application Programming Interface

1.4 References

- MySQL documentation for database architecture
- RESTful API design documentation

2. Architectural Representation

The TADMS architecture includes:

- **Frontend:** A web interface built using React.js for customer interactions (vehicle selection, booking, payment).
- **Backend (Logic):** Server-side business logic implemented using Node.js, which handles requests from the frontend and interacts with the database.
- **Database (MySQL):** A MySQL database that stores information about customers, vehicles, bookings, drivers, and payments.

3. Architectural Goals and Constraints

3.1 Goals

- Provide an intuitive and user-friendly interface for customers to book vehicles for road trips.
- Enable administrators to manage vehicle inventory, driver assignments, and bookings efficiently.
- Ensure data accuracy and security, particularly in managing vehicle availability and payment statuses.

3.2 Constraints

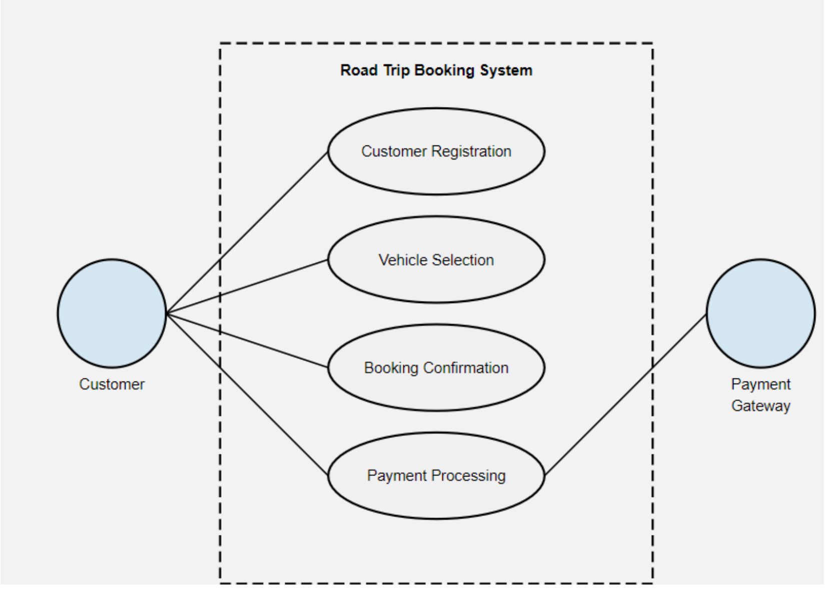
- The system must handle multiple concurrent users (up to 1,000).
- Booking data must be updated in real-time to prevent double bookings.
- The system must ensure secure data handling, including encrypted payment information.

4. Use-Case View

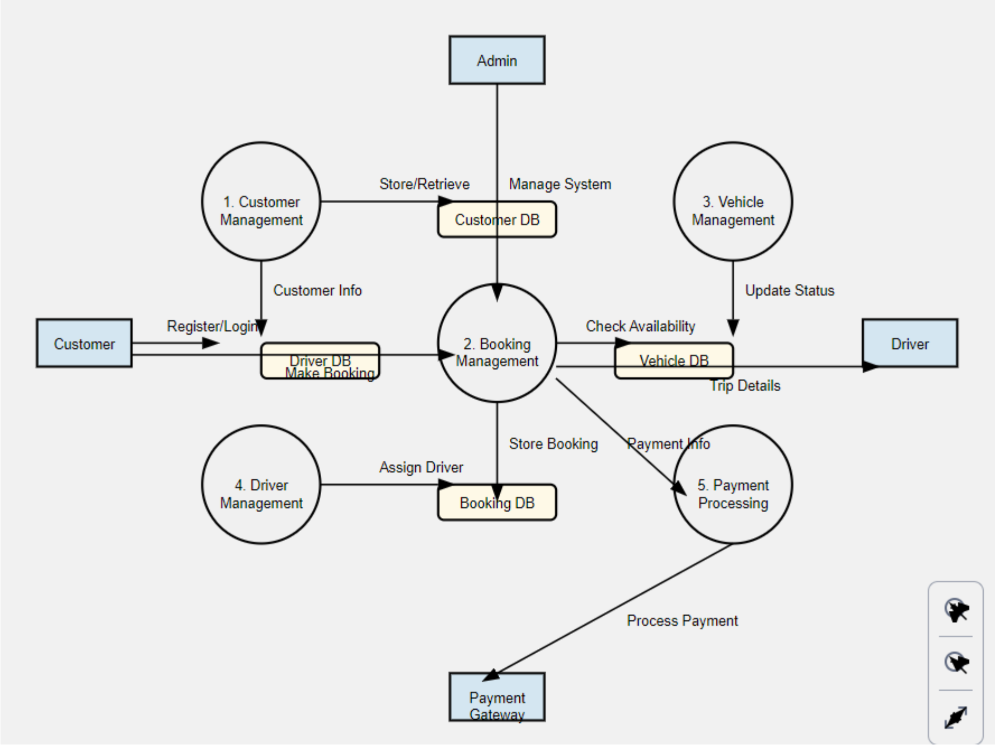
4.1 Architecturally Significant Use Cases

- **Customer Registration:** Customers sign up and provide details, which are stored in the MySQL database.
- **Vehicle Selection:** Customers select available vehicles for road trips, and the system verifies availability through the database.
- **Booking Confirmation:** Once a vehicle is booked, the system confirms the booking, assigns a driver, and updates the vehicle status.
- **Payment Processing:** Customers complete payments via an integrated payment gateway, and the system tracks payment statuses.

Use case diagram



DFD



5. Logical View

The TADMS follows a client-server model:

- **Frontend (React):**
  - Customers can register, browse vehicles, make bookings, and view payment status.
  - Administrators can manage vehicles, bookings, and driver assignments.

- **Backend (Node.js API):**
    - Handles customer registration, vehicle bookings, and driver assignments.
    - Interacts with the MySQL database to manage and update vehicle availability and payment statuses.
- **Database (MySQL):**
    - Stores customer data, vehicle inventory, booking records, driver details, and payment information.
    - Ensures secure data storage and retrieval.

6. Process View

6.1 Processes

- **Customer Booking Process:**
    1. The customer selects a vehicle and a trip date via the web interface.
    2. The frontend sends a request to the backend API to check vehicle availability.
    3. The backend queries the database and returns the result.
    4. If available, the system confirms the booking, updates the database, and assigns a driver.

6.2 Frontend-Backend Interaction

- When a customer books a vehicle, the request flows as follows:
    1. The frontend sends a request to the backend (Node.js).
    2. The backend checks availability by querying the MySQL database.
    3. The frontend receives a response and displays the booking confirmation to the customer.

6.3 Process Model to Design

- Each process is associated with user interface elements (forms, buttons) on the frontend, API endpoints on the backend, and database queries for managing data.

7. Performance

- The system is designed for scalability, ensuring it can handle up to 1,000 concurrent users.
- Optimized database queries and asynchronous API calls improve performance and reduce response time for booking requests.
- The system ensures a response time of less than 2 seconds for booking requests.

8. Quality.

- Data security is ensured through encrypted storage of customer and payment data.
- Error-handling mechanisms ensure smooth operation even during system failures.
- The system will undergo regular maintenance and updates to ensure optimal performance.