## Cab Management System

MINI PROJECT REPORT

Submitted by

## Tanay Ghate[ RA2311050010032]

**Himanshu Kumar[RA2311050010004]**

Under the Guidance of

**Dr.D.Rajeswari**

**21CSC205P**– **Database Management Systems**

**DATA SCIENCE AND BUSINESS SYSTEMS**

****

**FACULTY OF ENGINEERING AND TECHNOLOGY SCHOOL OF COMPUTING SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR**

**February 2025**

**SRM INSTITUTION OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

**BONAFIDE CERTIFICATE**

Certified that the 21CSC205P Database Management System course project report titled **“Cab Management System”** is the Bonafide work done by **Tanay Ghate [**RA2311050010032**], Himanshu Kumar [**RA2311050010004**],** of **II**

**Year/IV Sem B.Tech(CSE-BCT)** who carried out the mini project under my supervision.

**Dr. D. Rajeswari**

**Associate Professor,**

Department of DSBS,

SRM Institute of Science and Technology Kattankulathur

**Dr. Kavitha V**

**Head of the Department**

Professor and Head

Department of DSBS, SRM Institute of Science and Technology Kattankulathur

# ABSTRACT

The Cab Management System is a comprehensive and user-friendly software application designed to streamline cab operations and provide efficient services to customers. Built with Java Swing for an intuitive graphical interface and JDBC for robust database connectivity, this project enables cab service providers to manage bookings, drivers, and vehicle information seamlessly.

The system offers essential features like customer registration, booking management, real-time cab tracking, fare calculation, and driver allocation. The Java Swing framework ensures a responsive and visually appealing user interface, making it easy for users to navigate and manage cab-related tasks. JDBC integration facilitates smooth and secure data handling, allowing for the efficient storage and retrieval of information such as ride history, payment details, and driver records.

In a time where fast, reliable transportation services are increasingly necessary, this Cab Management System provides a modern solution for cab companies to enhance operational efficiency, improve customer satisfaction, and maintain high service standards. This project showcases the practical application of Java and database management techniques to create a streamlined, responsive cab management platform.

# ACKNOWLEDGEMENT

I am grateful for the guidance, support, and encouragement I received throughout the development of this Cab Management System project. I would like to extend my heartfelt thanks to my mentor, **Dr. D. Rajeswari** , Associate Professor, for her invaluable guidance, expertise, and constructive feedback. Her continuous support and insightful suggestions have been instrumental in completing this project successfully.

I also wish to express my gratitude to **Dr. Kavitha V**, Head of the Department of DSBS, for her encouragement and for providing a conducive environment for learning and innovation. Her leadership and support have greatly motivated me in my academic journey.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

# INTRODUCTION

In today's fast-paced world, efficient and reliable transportation services have become an essential aspect of urban life. Managing a cab service effectively not only requires well-coordinated operations but also a user- centric approach to ensure customer satisfaction and seamless connectivity between drivers and riders. The Cab Management System, developed in Java with the use of Swing for graphical interface design and JDBC for database connectivity, is a solution crafted to meet these needs by optimizing cab operations, streamlining booking processes, and enhancing the overall service experience.

This project recognizes the increasing demand for structured and efficient management in the transportation sector, where timely and accurate data handling is crucial. By leveraging a Java-based GUI through Swing, this system provides a responsive and visually accessible interface that allows administrators and users to manage bookings, allocate drivers, track rides, and handle payments effortlessly. JDBC connectivity enables secure and effective database interactions, ensuring real-time access to essential information like cab availability, ride history, driver records, and customer data.

With a range of features, including customer registration, booking management, fare calculation, and driver assignment, the Cab Management System aims to simplify and improve the operational aspects of cab services. In a time where customer convenience and operational efficiency are paramount, this project stands as a modern, technology-driven solution for cab companies to streamline their services, maintain high standards, and foster customer loyalty. By integrating Java’s Swing and JDBC, this system demonstrates the potential of software development to transform increasingly connected world.

## Problem Understanding

The main goal of the Cab Management System is to **connect passengers with available cabs** while ensuring smooth management of drivers, vehicles, and trip data.

**Key Components of the System**

1. **User Management**
   * Passengers: Register, login, book cabs, make payments, view ride history.
   * Drivers: Register, accept/reject rides, update availability.
   * Admins: Manage users, monitor rides, generate reports.
2. **Cab Booking & Ride Management**
   * Passengers enter pickup and destination locations.
   * The system finds an available cab nearby.
   * Driver accepts the ride request.
   * Ride status updates (e.g., "Ongoing," "Completed").
3. **Payment & Fare Calculation**
   * Fare estimation based on distance, time, and type of cab.
   * Online payment integration (optional).
4. **Location Tracking & Maps Integration**
   * Displaying available cabs near the user.
   * Tracking ongoing rides.
5. **Database (SQL Integration)**
   * Storing user details, ride history, driver info, and payments.

.

# 2. REQUIREMENT ANALYSIS

The requirement analysis for a Cab Management System using Java with Swing for the GUI and JDBC for database connectivity outlines the essential features, constraints, and expected behaviors of the system. The goal is to ensure that the system can efficiently handle bookings, maintain driver and vehicle records, and provide a user-friendly experience.

**Functional Requirements:**

1. User Management:
   * User registration and login
   * User profile management
   * Customer details storage and retrieval
2. Driver and Cab Management:
   * Driver registration, profile updates, and assignment tracking
   * Cab registration, availability status, and maintenance tracking
   * Driver-cab assignment based on availability and location
3. Booking System:
   * Booking creation, cancellation, and modification
   * Real-time fare calculation based on distance and other parameters
   * Driver allocation based on nearest available cab
4. Payment System:
   * Payment gateway integration
   * Generation of ride receipts
   * Payment history and transaction tracking
5. Ride Tracking and Notifications:
   * Real-time tracking of active rides
   * Notification system for booking confirmation, ride start, and ride end
   * Updates for customers and drivers
6. Administrative Functions:
   * Admin dashboard for monitoring all bookings, drivers, and cabs
   * Data analysis and reporting features

**Non-Functional Requirements:**

1. Usability: The system should offer a simple and intuitive GUI with easy navigation and clear options for customers, drivers, and administrators.
2. Performance: The system should respond quickly to booking and database queries, maintaining efficient performance even during peak usage.
3. Scalability: The database design should allow for the addition of more users,

## 5. Identification of Entity and Relationships

1. **Entities in the System**

In a **Cab Management System**, entities represent real-world objects involved in the booking and management of cabs. The key entities are:

|  |  |
| --- | --- |
| **Entity Name** | **Description** |
| **Users** | Stores details of all users (Passengers, Drivers, Admins). |
| **Drivers** | Stores details of drivers (linked to Users). |
| **Cabs** | Stores cab details (linked to Drivers). |
| **Ride Requests** | Stores ride details (linked to Passengers and Drivers). |
| **Payments** | Stores payment transactions for completed rides. |
| **Ratings** | Stores feedback and rating details for rides. |

1. **Relationships Between Entities**

Entities are connected through relationships. Here’s how they interact:

|  |  |  |
| --- | --- | --- |
| **Entities Involved** | **Relationship**  **Type** | **Description** |
| **User - Ride Request** | **One-to-Many**  **(1:N)** | A user (passenger) can request multiple rides. |
| **Driver - Ride Request** | **One-to-Many**  **(1:N)** | A driver can be assigned multiple ride requests. |
| **Driver - Cab** | **One-to-One**  **(1:1)** | Each driver is assigned one cab, and each cab has one driver. |
| **Ride Request - Payment** | **One-to-One**  **(1:1)** | Each ride has one associated payment. |
| **Passenger - Ratings** | **One-to-Many**  **(1:N)** | A passenger can give multiple ratings (one per ride). |
| **Driver - Ratings** | **One-to-Many**  **(1:N)** | A driver can receive multiple ratings (one per ride). |

1. **Entity-Relationship (ER) Diagram Overview** A simple representation of relationships: mathematica

CopyEdit

Users (Passenger, Driver, Admin)

├── 1:N ───> Ride Requests

├── 1:N ───> Ratings Drivers

├── 1:1 ───> Cabs

├── 1:N ───> Ride Requests

├── 1:N ───> Ratings Ride Requests

├── 1:1 ───> Payments

├── N:1 ───> Users (Passengers)

├── N:1 ───> Drivers

1. **Primary and Foreign Keys in Relationships**

|  |  |  |
| --- | --- | --- |
| **Table** | **Primary Key**  **(PK)** | **Foreign Key (FK)** |
| **Users** | user\_id (PK) | None |
| **Drivers** | driver\_id (PK) | user\_id (FK) (from Users) |
| **Cabs** | cab\_id (PK) | driver\_id (FK) (from Drivers) |
| **Ride Requests** | ride\_id (PK) | passenger\_id (FK) (from Users), driver\_id |
| **Payments** | payment\_id  (PK) | ride\_id (FK) (from Ride Requests) |
| **Ratings** | rating\_id (PK) | ride\_id (FK) (from Ride Requests), passenger\_id (FK) (from Users), driver\_id (FK) (from Drivers) |

1. **Cardinality Explanation**
2. **One-to-Many (1:M)**
   * A **User** can request multiple rides, but each ride belongs to **one** user.
   * A **Driver** can complete multiple rides, but each ride is assigned to **one** driver.
3. **One-to-One (1:1)**
   * A **Driver** is assigned exactly **one** cab.
4. **Many-to-One (M:1)**
   * Multiple **Rides** are linked to **one** passenger and **one** driver.

# ARCHITECTURE AND DESIGN

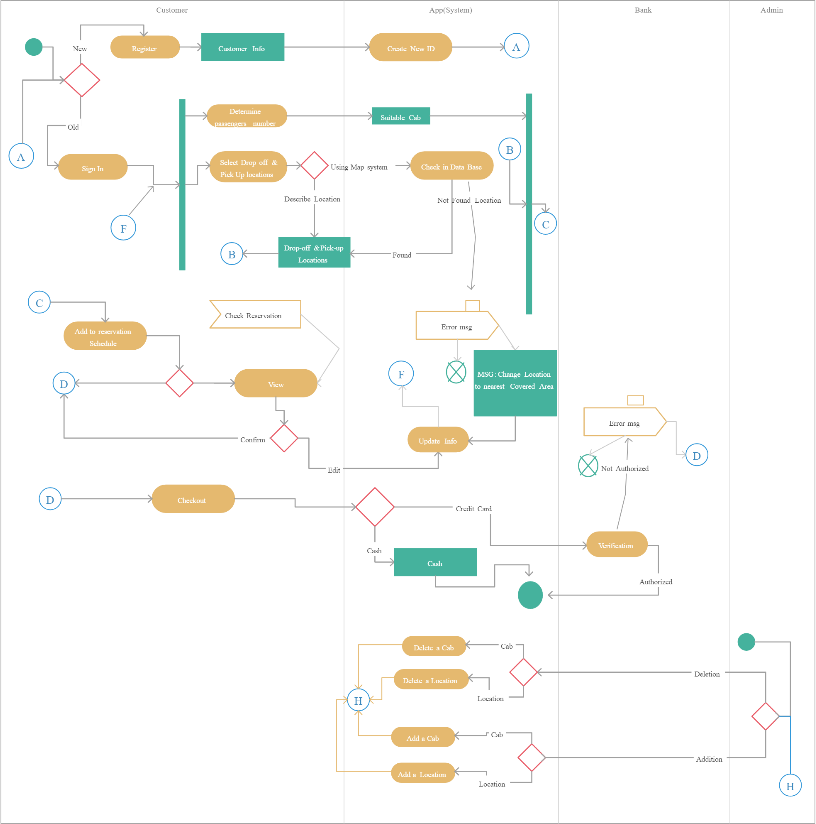
The architecture and design of a Cab Management System using Java Swing and JDBC focus on ensuring functionality, scalability, and a smooth user experience.

Architecture:

The system architecture will follow a client-server model, with Java Swing handling the front end and JDBC managing database interactions on the backend.

* Client Side (GUI): Developed using Java Swing, it includes screens for registration, login, booking, payment, and an admin dashboard.
* Server Side (Database): JDBC connects to a relational database (e.g., MySQL), which stores customer, driver, booking, and transaction data.

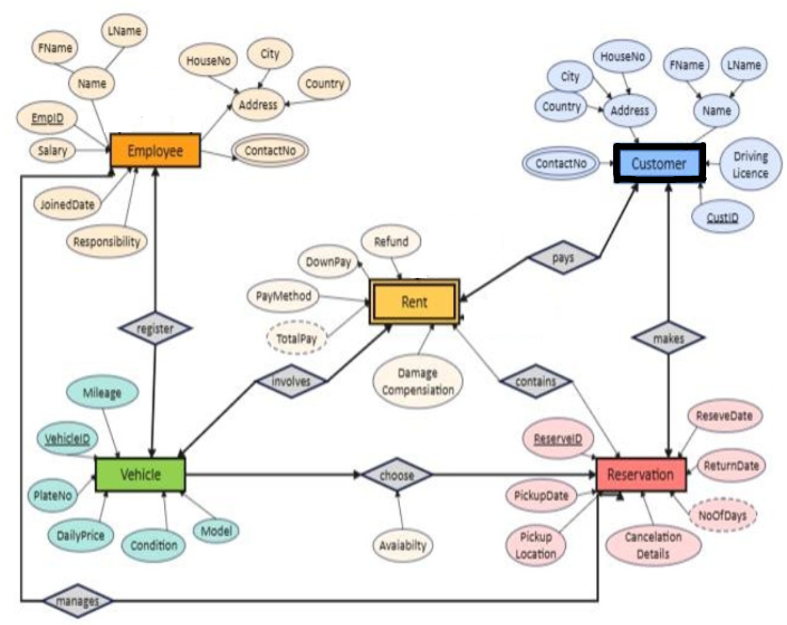
Architecture Diagram: An architectural diagram can represent the interactions between the client (Java Swing GUI), the application logic (Java code), and the database (via JDBC).



# IMPLEMENTATION

Implementing the Cab Management System involves coding the functionalities, designing the user interface with Java Swing, and setting up JDBC for database management.

1. **Set Up the Development Environment:**
   1. **Install Java:** Ensure the Java Development Kit (JDK) is installed.
   2. **Configure GUI:** Use Java Swing to create the GUI, as it provides an extensive toolkit for building a responsive interface.
   3. **Set up Database:** Use MySQL, PostgreSQL, or another relational database, and connect it using JDBC for data handling.
2. **Design the Database Schema:**
   1. **Define Database Tables:** Create tables for customers, drivers, cabs, bookings, and payments. Each table will store specific details relevant to its function (e.g., customer ID, driver availability).
   2. **Establish Relationships:** Define foreign key relationships to link booking data with customer and driver records, creating a seamless association between ride details and users.
3. **Create the User Interface:**
   1. **Design GUI Screens:** Using Java Swing, design the screens for: - **Registration and Login:** Allow customers and drivers to register and log in. - **Booking Interface:** Let customers book cabs, view details, and confirm bookings. - **Driver Dashboard:** Display driver availability and allow status updates. - **Admin Dashboard:** Provide monitoring and reporting tools for administrators.
   2. **Integrate Swing Components:** Develop navigation between screens and include real-time notifications, user input validation, and clear error messaging.
4. **Implement Key Functionalities:**
   1. **Booking Management:** Implement booking creation, modification, and cancellation features, ensuring proper updates in the database.
   2. **Payment Processing:** Integrate a payment system, allowing users to complete transactions and receive a receipt.
   3. **Ride Tracking and Notifications:** Provide live tracking of rides and automated notifications for key events like ride start, end, and booking confirmation
5. **E-R Diagram For Cab Management System**

****

1. **Construction of Database Using ER Model for Cab Management System**

**To construct a Database (DB) using the ER Model, we need to:**

1. **Design an Entity-Relationship (ER) Diagram with entities, attributes, and relationships.**
2. **Convert the ER Model into a Relational Schema with Primary and Foreign Keys.**
3. **Write SQL Queries to create the tables and establish relationships.**
4. **ER Diagram Design (Entities and Relationships) The main entities in the system:**
   * **Users (Passengers, Drivers, Admins)**
   * **Drivers (Linked to Users)**
   * **Cabs (Linked to Drivers)**
   * **Ride Requests (Linked to Users and Drivers)**
   * **Payments (Linked to Rides and Users)**
   * **Ratings (Feedback from Passengers to Drivers) ER Model Notation:**
   * **Entities: Rectangles**
   * **Relationships: Diamonds**
   * **Attributes: Ovals**
   * **Primary Keys: Underlined**
   * **Foreign Keys: Arrow pointing to referenced entity Here’s the ER Model representation:**

**scss**

**]**

1. **Relational Schema from ER Model**

|  |  |  |
| --- | --- | --- |
| **Entity** | **Primary Key**  **(PK)** | **Foreign Key (FK)** |
| **Users** | **user\_id** | **None** |
| **Drivers** | **driver\_id** | **user\_id (from Users)** |
| **Cabs** | **cab\_id** | **driver\_id (from Drivers)** |
| **Ride** | **ride\_id** | **passenger\_id (from Users), driver\_id (from** |

|  |  |  |
| --- | --- | --- |
| **Entity** | **Primary Key**  **(PK)** | **Foreign Key (FK)** |
| **Requests** |  | **Drivers)** |
| **Payments** | **payment\_id** | **ride\_id (from Ride Requests), passenger\_id (from Users)** |
| **Ratings** | **rating\_id** | **ride\_id (from Ride Requests), passenger\_id (from Users), driver\_id (from Drivers)** |

1. **SQL Queries to Construct the Database**
2. **Creating Users Table sql**

**CopyEdit**

**CREATE TABLE Users (**

**user\_id INT PRIMARY KEY AUTO\_INCREMENT, name VARCHAR(100) NOT NULL,**

**email VARCHAR(100) UNIQUE NOT NULL, phone VARCHAR(15) UNIQUE NOT NULL, password VARCHAR(255) NOT NULL,**

**role ENUM('Passenger', 'Driver', 'Admin') NOT NULL, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP**

**);**

1. **Creating Drivers Table sql**

**CopyEdit**

**CREATE TABLE Drivers (**

**driver\_id INT PRIMARY KEY AUTO\_INCREMENT, user\_id INT UNIQUE NOT NULL,**

**license\_number VARCHAR(50) UNIQUE NOT NULL, cab\_id INT,**

**availability ENUM('Available', 'On Ride') DEFAULT 'Available', FOREIGN KEY (user\_id) REFERENCES Users(user\_id) ON DELETE**

**CASCADE,**

**FOREIGN KEY (cab\_id) REFERENCES Cabs(cab\_id) ON DELETE SET NULL**

**);**

1. **Creating Cabs Table sql**

**CopyEdit**

**CREATE TABLE Cabs (**

**cab\_id INT PRIMARY KEY AUTO\_INCREMENT,**

**driver\_id INT UNIQUE,**

**model VARCHAR(50) NOT NULL,**

**plate\_number VARCHAR(20) UNIQUE NOT NULL, capacity INT NOT NULL,**

**cab\_type ENUM('Sedan', 'SUV', 'Mini', 'Auto') NOT NULL, status ENUM('Active', 'Inactive') DEFAULT 'Active',**

**FOREIGN KEY (driver\_id) REFERENCES Drivers(driver\_id) ON DELETE SET NULL**

**);**

1. **Creating Ride Requests Table sql**

**CopyEdit**

**CREATE TABLE RideRequests (**

**ride\_id INT PRIMARY KEY AUTO\_INCREMENT,**

**passenger\_id INT NOT NULL, driver\_id INT,**

**pickup\_location VARCHAR(255) NOT NULL, drop\_location VARCHAR(255) NOT NULL,**

**ride\_status ENUM('Pending', 'Accepted', 'Ongoing', 'Completed', 'Cancelled') DEFAULT 'Pending',**

**fare DECIMAL(10,2),**

**ride\_start\_time DATETIME, ride\_end\_time DATETIME,**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**FOREIGN KEY (passenger\_id) REFERENCES Users(user\_id) ON DELETE CASCADE,**

**FOREIGN KEY (driver\_id) REFERENCES Drivers(driver\_id) ON DELETE SET NULL**

**);**

1. **Creating Payments Table sql**

**CopyEdit**

**CREATE TABLE Payments (**

**payment\_id INT PRIMARY KEY AUTO\_INCREMENT, ride\_id INT UNIQUE NOT NULL,**

**passenger\_id INT NOT NULL, amount DECIMAL(10,2) NOT NULL,**

**payment\_status ENUM('Pending', 'Completed', 'Failed') DEFAULT**

**'Pending',**

**payment\_method ENUM('Cash', 'Card', 'UPI') NOT NULL, transaction\_id VARCHAR(50) UNIQUE,**

**payment\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**FOREIGN KEY (ride\_id) REFERENCES RideRequests(ride\_id) ON DELETE CASCADE,**

**FOREIGN KEY (passenger\_id) REFERENCES Users(user\_id) ON DELETE CASCADE**

**);**

1. **Creating Ratings Table sql**

**CopyEdit**

**CREATE TABLE Ratings (**

**rating\_id INT PRIMARY KEY AUTO\_INCREMENT, ride\_id INT UNIQUE NOT NULL,**

**passenger\_id INT NOT NULL, driver\_id INT NOT NULL,**

**rating INT CHECK (rating BETWEEN 1 AND 5), review TEXT,**

**created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**FOREIGN KEY (ride\_id) REFERENCES RideRequests(ride\_id) ON DELETE CASCADE,**

**FOREIGN KEY (passenger\_id) REFERENCES Users(user\_id) ON DELETE CASCADE,**

**FOREIGN KEY (driver\_id) REFERENCES Drivers(driver\_id) ON DELETE CASCADE**

**);**

1. **Query Examples for Database Operations**
2. **Insert a New Passenger sql**

**CopyEdit**

**INSERT INTO Users (name, email, phone, password, role) VALUES ('John Doe',** [**'john@example.com',**](mailto:%27john@example.com) **'9876543210', 'hashed\_password', 'Passenger');**

1. **Insert a New Driver sql**

**CopyEdit**

**INSERT INTO Users (name, email, phone, password, role)**

**VALUES ('Alice Smith',** [**'alice@example.com',**](mailto:%27alice@example.com) **'9876543211', 'hashed\_password', 'Driver');**

**INSERT INTO Drivers (user\_id, license\_number, availability) VALUES (LAST\_INSERT\_ID(), 'DL12345XYZ', 'Available');**

1. **Assign a Cab to a Driver sql**

**CopyEdit**

**INSERT INTO Cabs (model, plate\_number, capacity, cab\_type, status) VALUES ('Toyota Innova', 'KA-01-AB-1234', 4, 'SUV', 'Active');**

**UPDATE Drivers**

**SET cab\_id = LAST\_INSERT\_ID() WHERE driver\_id = 1;**

1. **Fetch Available Cabs sql**

**CopyEdit**

**SELECT Cabs.cab\_id, model, plate\_number, cab\_type FROM Cabs**

**JOIN Drivers ON Cabs.driver\_id = Drivers.driver\_id**

**WHERE Drivers.availability = 'Available' AND Cabs.status = 'Active';**