

BUS MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

SHIVANESH.P

220701268

SHRIRAM SETHU.S

220701273

In partial fulfillment for the award of the degree of

BACHELOR OF

ENGINEERING IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105



2023 - 24

BONAFIDE CERTIFICATE

Certified that this project report “**BUS MANAGEMENT SYSTEM**” is the

bonafide work of “**SHIVANESH.P(220701268),**

SHRIRAM SETHU.S(220701273) ”

who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Dr.R.SABITHA
Professor and II Year Academic Head
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam, Chennai - 602 105

SIGNATURE

Ms.V.JANANEE
Assistant Professor (SG),
Computer Science and Engineering,
Rajalakshmi Engineering College,
(Autonomous),
Thandalam, Chennai - 602 105

INTERNAL EXAMINER

EXTERNAL EXAMINE

ABSTRACT

Bus management system is to offer a systematic method for managing the depot admin, drivers, and conductors. The Depot administrator can access all the data, add drivers and conductors, and allocate them trips thanks to the bus management system. The bus conductor can enter the revenue generated from each trip while also purchasing tickets for passengers. The driver can input the actual time of departure and arrival as well as the amount of fuel used. The benefits of implementing a BMS are numerous. It can improve operational efficiency, reduce costs, enhance passenger experience, and provide valuable data for informed decision-making. This abstract paves the way for a more detailed exploration of bus management systems, their functionalities, and their impact on the public transportation sector.

As a result, the Bus Management system makes it simple to operate and administer the bus transportation service.

Key words: database management system

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

1.2 OBJECTIVES

1.3 MODULES

1.4 SOFTWARE DESCRIPTION

1.5 LANGUAGES

2. SURVEY OF TECHNOLOGIES

2.1 SQL

2.2 JAVA

3. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

3.2 HARDWARE AND SOFTWARE
REQUIREMENTS

3.3 ARCHITECTURE DIAGRAM

3.4 ER DIAGRAM

3.5 NORMALIZATION

4.PROGRAM CODE

5. RESULTS AND DISCUSSION

6.CONCLUSION

7.REFERENCES

1. INTRODUCTION

1.1 introduction

Buses are a fundamental mode of transportation, serving millions of commuters daily. To ensure smooth and efficient operations, bus companies rely on bus management systems (BMS). A BMS is a software solution that acts as the central nervous system for bus operations, integrating various functions into a cohesive platform.

This introduction dives into the world of bus management systems. We'll explore the challenges faced by traditional, manual bus operations and how a BMS addresses them. We'll also highlight the key functionalities of a BMS, typically encompassing:

- **Fleet Management:** Gain real-time insights into bus locations, driver performance, and maintenance schedules.
- **Route Optimization:** Design efficient routes, manage bus stops strategically, and provide accurate arrival information to passengers.
- **Ticketing and Passenger Convenience:** Facilitate online ticket booking, manage fares effectively, and offer multiple payment options for a seamless experience.
- **Data-Driven Decision Making:** Generate comprehensive reports on ridership, revenue, and operational efficiency to guide strategic planning.

By implementing a BMS, bus companies can achieve significant improvements in several areas. We'll discuss the potential benefits of a BMS, including:

- **Enhanced Operational Efficiency:** Streamline workflows, optimize resource allocation, and reduce manual tasks.
- **Cost Reduction:** Save on fuel, maintenance costs, and manpower requirements.
- **Improved Passenger Experience:** Offer real-time information, convenient ticketing options, and a more reliable service.
- **Data-Driven Insights:** Gain valuable data to optimize routes, pricing strategies, and overall service delivery.

This introduction sets the stage for a deeper understanding of bus management systems. We'll delve into the specific functionalities, explore the benefits in detail, and showcase how BMS are transforming the public transportation landscape.

1.2 objectives

Developing a Bus Management System Application involves multiple objectives to ensure the application is user-friendly, secure, and engaging. Here are some key objectives to consider:

1. User Experience (UX) and Interface Design

Intuitive Design: Create a user-friendly interface that is easy to navigate for both novice and experienced users.

Responsive Design: Ensure the application works seamlessly across various devices (mobile, tablet, desktop).

Engaging Visuals: Use high-quality graphics and animations to enhance user engagement.

2. Functionality

Real-time Data: Provide live updates on Bus.

Account Management: Allow users to create and manage their accounts.

Notifications: Implement push notifications on Bus Updates.

3. Security

Data Protection: Ensure robust data encryption and secure user data storage to protect against breaches.

Secure Transactions: Implement secure payment

gateways for deposits and withdrawals.

Authentication: Use multi-factor authentication to enhance account security.

4. Performance and Reliability

Scalability: Design the application to handle high traffic volumes.

Low Latency: Ensure minimal delay in real-time updates and transactions.

Reliability: Maintain high uptime and quickly address any technical issues or outages.

5. Customer Support

Help Center: Provide a comprehensive help center with FAQs, tutorials, and guides.

Live Support: Offer live chat, email, and phone support to assist users with any issues.

1.3 MODULES

Bus Management System - Java & MySQL Modules

This document outlines the core modules of a Bus Management System (BMS) built with Java and MySQL. Each module focuses on a specific aspect of bus operations, providing a modular and well-organized system.

1. Bus Management Module

- **Functionalities:**
 - Add, edit, and delete bus information (registration number, model, capacity, etc.)
 - Track bus location (using GPS integration, if applicable)
 - Schedule bus maintenance (record maintenance history)
 - Manage driver assignments to buses
- **Java Classes:**
 - Bus.java
 - BusService.java (for interacting with Bus table in MySQL)
- **MySQL Tables:**
 - Bus (bus_id, registration_no, model, capacity, etc.)
 - Maintenance (maintenance_id, bus_id, date, description, etc.) (Optional - if tracking maintenance history)
 - DriverAssignment (assignment_id, bus_id, driver_id, date, etc.) (Optional - if managing driver assignments)

2. Route Management Module

- **Functionalities:**
 - Define bus routes (origin, destination, stops)
 - Manage bus schedules (departure and arrival times)
 - Optimize routes based on traffic patterns and passenger demand (optional - advanced functionality)

- **Java Classes:**

- Route.java
- Stop.java
- Schedule.java
- RouteService.java (for interacting with Route, Stop, and Schedule tables in MySQL)

- **MySQL Tables:**

- Route (route_id, origin, destination, etc.)
- Stop (stop_id, route_id, stop_name, location, etc.)
- Schedule (schedule_id, route_id, departure_time, arrival_time, day, etc.)

3. Ticketing and Passenger Management Module

- **Functionalities:**

- Online ticket booking (passengers can select route, date, seat)
- Manage fares for different routes and seat categories
- Issue tickets (digital or printed)
- Track passenger bookings and cancellations

- **Java Classes:**

- Ticket.java
- Passenger.java
- Booking.java
- TicketingService.java (for interacting with Ticket, Passenger, Booking, and Fare tables in MySQL)

- **MySQL Tables:**

- Passenger (passenger_id, name, contact_details, etc.)
- Ticket (ticket_id, booking_id, passenger_id, seat_no, fare, etc.)
- Booking (booking_id, passenger_id, route_id, schedule_id, date, etc.)
- Fare (fare_id, route_id, seat_category, fare_amount, etc.)

4. Reporting and Analytics Module

- **Functionalities:**

- Generate reports on ridership (passengers per route/schedule)
- Analyze revenue generated from ticket sales
- Track bus performance (on-time arrivals, maintenance costs)
- Provide insights for route optimization and pricing strategies
- **Java Classes:**
 - ReportGenerator.java (for generating reports based on data retrieved from MySQL)
- **This module utilizes data from all other modules**

5. User Management Module (Optional)

- **Functionalities:**
 - Manage user accounts (admin, staff, drivers)
 - Assign user roles and permissions
 - Secure user login and access control
- **Java Classes:**
 - User.java
 - Role.java
 - UserService.java (for interacting with User and Role tables in MySQL)
- **MySQL Tables:**
 - User (user_id, username, password, role_id, etc.)
 - Role (role_id, role_name, permissions, etc.)

This modular approach provides a well-organized structure for the Bus Management System. Each module encapsulates specific functionalities, making the system scalable and easier to maintain. By utilizing Java for application development and MySQL for database management, the system offers flexibility and robustness.

Note: This is a high-level overview. The actual implementation will involve additional classes, methods, and error handling for a fully functional system.

2. SURVEY OF TECHNOLOGIES

To create a Bus Management application using Java Swings and MySQL, you need to consider a few key technology components and design patterns. Here is a detailed survey of the technology stack and architecture:

1. Java Swings for GUI

Java Swings is a part of Java Foundation Classes (JFC) used to create window-based applications. It's suitable for building the user interface of Bus Management System application.

Advantages of Java Swings:

-Rich Set of Components:** Swings offer a wide range of GUI components like buttons, labels, text fields, tables, trees, etc.

Customizable: Components can be easily customized to create a modern and attractive user interface.

Cross-Platform: Applications built with Swings are platform-independent and can run on any system with JVM.

Event Handling: Robust event-handling capabilities to manage user interactions.

Key Components:

JFrame: Main window of the application.

JPanel: Container to hold other components.

JButton, JLabel, JTextField: Basic interactive components.

JScrollPane: To handle scrolling within components.

2. MySQL for Database

MySQL is a popular relational database management system that is efficient for handling structured data and supports complex queries.

Advantages of MySQL:

-Reliability: Stable and reliable for managing large datasets.

Scalability: Can handle increased load by scaling vertically or horizontally.

Security: Provides robust security features to protect data.

Integration: Easily integrates with Java applications through JDBC.

Key Database Components:

User Table: Stores user information including account details and preferences.

3. JDBC for Database Connectivity

Java Database Connectivity (JDBC) is an API for connecting and executing queries on a database.

Key Concepts:

DriverManager: Manages a list of database drivers and establishes a connection to the database.

Connection: Interface that provides methods for interacting with the database.

Statement and PreparedStatement: Used to execute SQL queries and retrieve results.

ResultSet: Represents the result set of a query and allows traversal of the data.

4. MVC Architecture

Model-View-Controller (MVC) architecture is ideal for separating the application's logic, UI, and data management.

5. Additional Tools and Libraries

Apache Commons: Useful for various utility functions like String manipulation, IO operations, etc.

Log4j or SLF4J: For logging application events, useful for debugging and monitoring.

JUnit: For unit testing the application to ensure code quality and reliability.

SwingWorker: For handling background tasks and long-running operations without freezing the UI.

Example Application Flow

1. User Interface:

- Launch the application and display the main window (JFrame).
- Provide login and registration forms (JPanel with JTextField, JPasswordField, JButton).
- Main dashboard showing race schedules (JTable) and betting options (JComboBox, JButton).

2. Database Interaction:

- Establish a connection to the MySQL database using JDBC.
- Execute queries to fetch and display race information, odds, and user bets.
- Insert new bets and update user data based on interactions.

3. Event Handling:

- Handle user actions like placing a bet, checking results, and viewing transaction history.
- Use ActionListeners to manage button clicks and other user inputs.

Requirements and Analysis for Bus Management System

This document outlines the requirements and analysis for a Bus Management System (BMS) database, built using MySQL.

1. Introduction

The BMS aims to streamline bus operations for [Company Name]. It will provide a centralized platform to manage bus fleet, routes, ticketing, and passenger information. This document outlines the functional and non-functional requirements for the database.

2. Stakeholders

- **System Users:** Admins, staff (ticketing, operations), drivers (optional)
- **System Owner:** [Company Name] - IT Department

3. Functional Requirements

The database will support the following functionalities:

- **Bus Management:**
 - Store bus information (registration number, model, capacity, etc.)
 - Track bus location (if GPS integration exists)
 - Manage bus maintenance schedules and history (optional)
 - Manage driver assignments to buses (optional)
- **Route Management:**
 - Define bus routes with origin, destination, and stops
 - Store bus schedules with departure and arrival times for

each stop

- Allow for route optimization based on future needs (optional)

- **Ticketing and Passenger Management:**

- Manage passenger information (name, contact details)
- Facilitate online ticket booking with seat selection
- Store fare information for different routes and seat categories
- Issue tickets (digital or printable)
- Track passenger bookings and cancellations

- **Reporting and Analytics:**

- Generate reports on ridership (passengers per route/schedule)
- Analyze revenue generated from ticket sales
- Track bus performance (on-time arrivals, maintenance costs)

4. Non-Functional Requirements

- **Performance:** The system should handle a high volume of transactions efficiently, with minimal response time.
- **Scalability:** The database should accommodate future growth in the number of buses, routes, and passengers.
- **Security:** User access should be controlled with different permission levels. Data should be encrypted to ensure confidentiality and integrity.
- **Availability:** The database should be highly available with minimal downtime for maintenance or upgrades.

- **Backup and Recovery:** Regular backups should be performed to ensure data recovery in case of failures.

5. Data Analysis

This section outlines the data entities and their relationships within the database:

- **Entities:**

- Bus
- Route
- Stop
- Schedule
- Passenger
- Ticket
- Booking (optional - links Passenger, Route, Schedule)
- Fare (optional - links Route, Seat category)
- Driver (optional)
- Maintenance (optional - links Bus)
- User (optional - for user authentication)
- Role (optional - defines user permissions)

- **Relationships:**

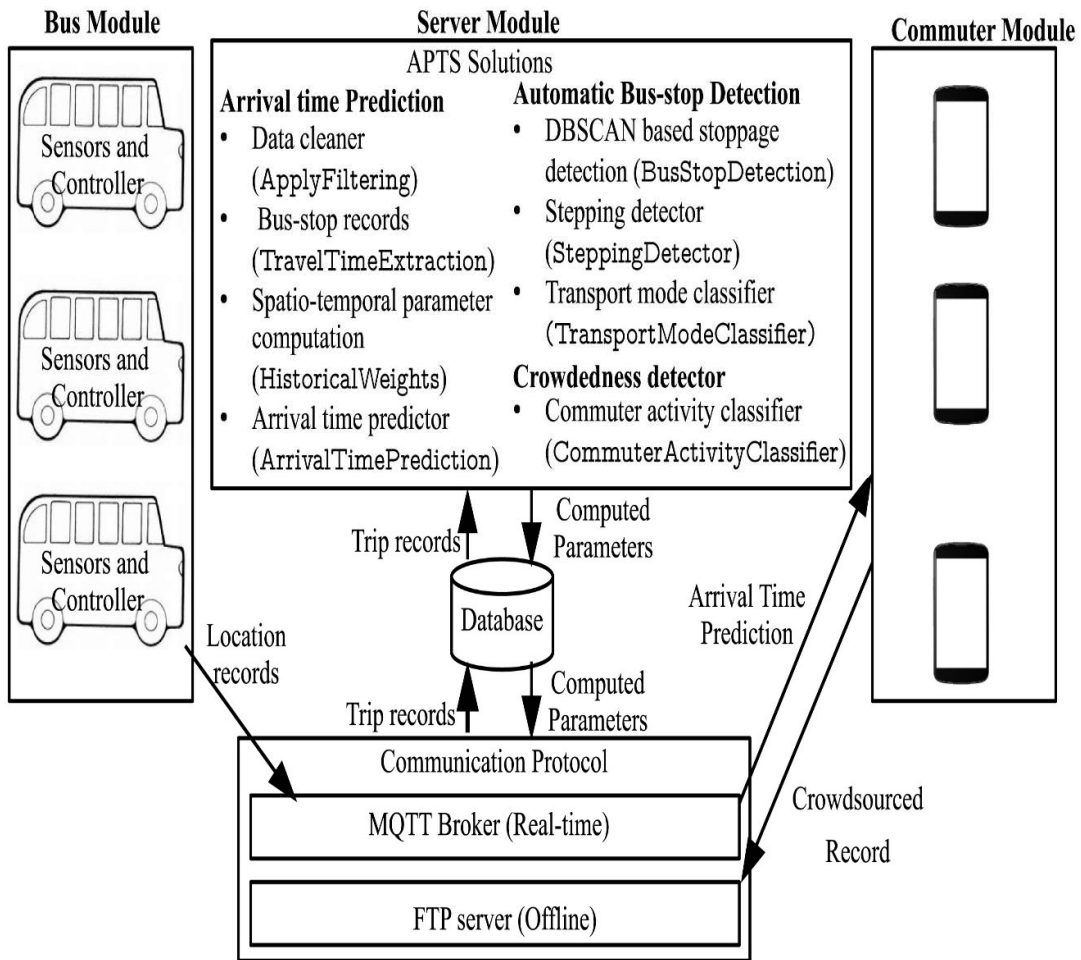
- A Bus can have many Schedules (one-to-many)
- A Route can have many Schedules (one-to-many)
- A Schedule belongs to one Route (many-to-one)
- A Schedule has many Stops (one-to-many)

- A Passenger can have many Bookings (one-to-many) (optional)
- A Booking belongs to one Passenger (many-to-one) (optional)
- A Booking is for one Schedule (many-to-one) (optional)
- A Ticket is for one Booking (one-to-one) (optional)
- A Fare applies to a Route and Seat category (many-to-many) (optional)
- A Driver can be assigned to many Buses (many-to-many) (optional)
- A Bus can have many Maintenance records (one-to-many) (optional)
- A User can have one Role (one-to-one) (optional)
- A Role can have many Users (one-to-many) (optional)

6. Conclusion

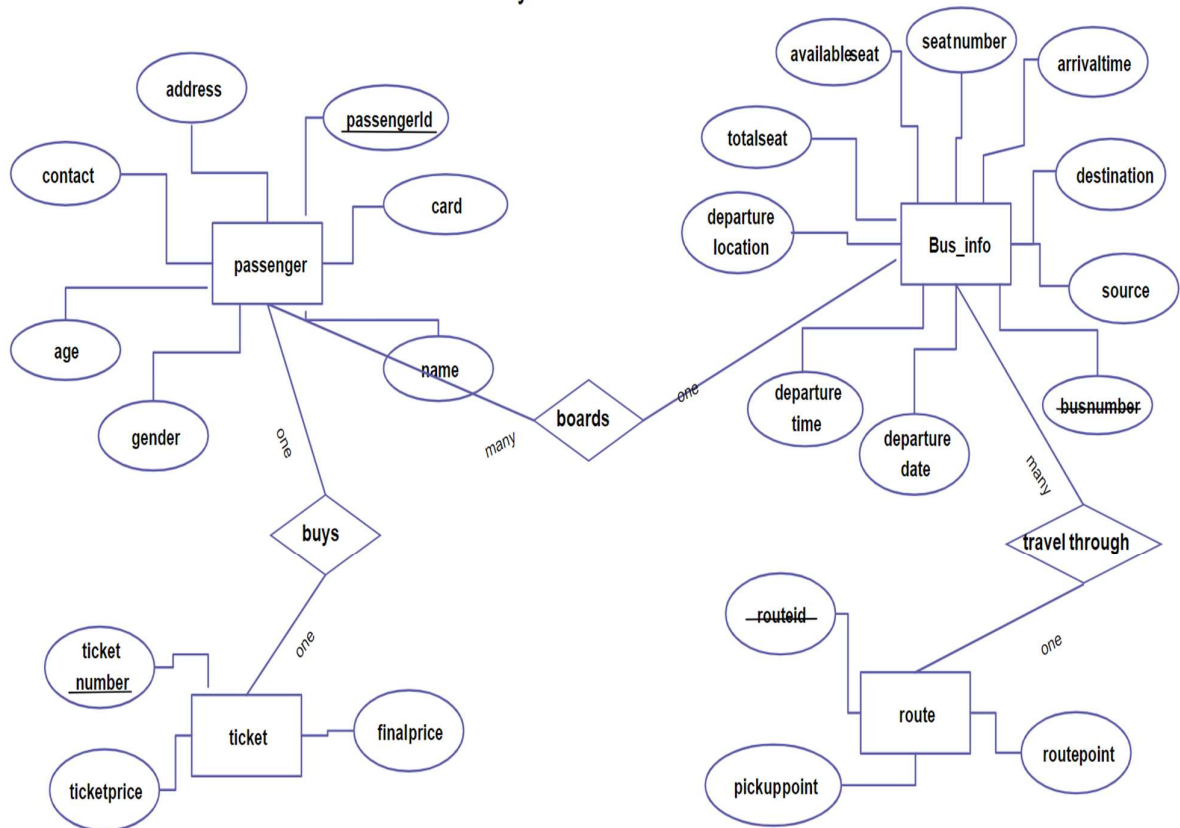
This document provides a comprehensive overview of the requirements and analysis for the Bus Management System database. The identified entities, relationships, and functional/non-functional requirements will serve as a blueprint for database design and implementation using MySQL.

Architecture Diagram



ER Diagram for Bus Management System

Here's an Entity-Relationship (ER) Diagram for Bus Management application:



PROGRAM CODE

Front End:

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Scanner;

public class BusManagementSystem {

    private static final String url =
        "jdbc:mysql://localhost:3306/bus_management_sys
        tem";

        private static final String username =
            "your_username";

            private static final String password =
                "your_password";


        public static Connection getConnection() throws
        SQLException {

            return DriverManager.getConnection(url,
            username, password);

        }

    }
```

```
public static void main(String[] args) throws
SQLException {

    Scanner scanner = new Scanner(System.in);

    int choice;

    do {

        System.out.println("\nBUS MANAGEMENT
SYSTEM");

        System.out.println("1. Bus Management");
        System.out.println("2. Route Management");
        System.out.println("3. Ticketing and
Passenger Management");

        System.out.println("4. Reporting and
Analytics");

        System.out.println("5. Exit");

        System.out.print("Enter your choice: ");

        choice = scanner.nextInt();

        switch (choice) {

            case 1:

                busManagement();

                break;
```

```

        case 2:
            routeManagement();
            break;
        // ... similar cases for other options
        case 5:
            System.out.println("Exiting the
system...");
            break;
        default:
            System.out.println("Invalid choice!");
    }
} while (choice != 5);
}

public static void busManagement() throws
SQLException {
    Scanner scanner = new Scanner(System.in);
    int option;

    do {
        System.out.println("\nBus Management");
        System.out.println("1. Add Bus");
        System.out.println("2. View Buses");
    }
}

```



```
// ... other functionalities like edit, delete
System.out.println("9. Back to Main Menu");
System.out.print("Enter your choice: ");
option = scanner.nextInt();

switch (option) {
    case 1:
        addBus();
        break;
    case 2:
        viewBuses();
        break;
    // ... similar cases for other options
    case 9:
        System.out.println("Returning to Main
Menu...");
        break;
    default:
        System.out.println("Invalid choice!");
}
} while (option != 9);
```

```
}
```

```
public static void addBus() throws SQLException {
```

```
    Scanner scanner = new Scanner(System.in);
```

```
    System.out.print("Enter Registration Number: ");
```

```
    String regNo = scanner.nextLine();
```

```
    System.out.print("Enter Model: ");
```

```
    String model = scanner.nextLine();
```

```
    System.out.print("Enter Capacity: ");
```

```
    int capacity = scanner.nextInt();
```

```
    // Prepare SQL statement and insert data
```

```
    Connection connection = getConnection();
```

```
    String sql = "INSERT INTO Bus  
(registration_no, model, capacity) VALUES (?, ?,  
?)";
```

```
    PreparedStatement statement =  
connection.prepareStatement(sql);
```

```
    statement.setString(1, regNo);
```

```
statement.setString(2, model);  
statement.setInt(3, capacity);  
statement.executeUpdate();  
  
statement.close();  
connection.close();  
  
System.out.println("Bus added successfully!");  
}
```

Back End:

```
public class DatabaseConnection {

    private static final String url =
"jdbc:mysql://localhost:3306/bus_management_system";

    private static final String username = "your_username";

    private static final String password = "your_password";


    public static Connection getConnection() throws
SQLException {

        return DriverManager.getConnection(url, username,
password);

    }

}

public interface CrudRepository<T> {

    List<T> getAll() throws SQLException;

    T getById(int id) throws SQLException;

    void save(T entity) throws SQLException;

    void update(T entity) throws SQLException;

    void delete(int id) throws SQLException;

}

public class Bus {
```

```

private int busId;

private String registrationNo;

private String model;

private int capacity;


// Getters, setters, constructors (omitted for brevity)

}

public class BusService implements CrudRepository<Bus> {

    @Override

    public List<Bus> getAll() throws SQLException {

        List<Bus> buses = new ArrayList<>();

        Connection connection =
DatabaseConnection.getConnection();

        String sql = "SELECT * FROM Bus";

        PreparedStatement statement =
connection.prepareStatement(sql);

        ResultSet resultSet = statement.executeQuery();

        while (resultSet.next()) {

```

```

        Bus bus = new Bus(
            resultSet.getInt("bus_id"),
            resultSet.getString("registration_no"),
            resultSet.getString("model"),
            resultSet.getInt("capacity")
        );
        buses.add(bus);
    }

    resultSet.close();
    statement.close();
    connection.close();
    return buses;
}

@Override
public Bus getById(int id) throws SQLException {
    // Implement logic to fetch Bus by id
}

@Override

```

```
public void save(Bus bus) throws SQLException {  
    Connection connection =  
    DatabaseConnection.getConnection();  
  
    String sql = "INSERT INTO Bus (registration_no,  
model, capacity) VALUES (?, ?, ?)";  
  
    PreparedStatement statement =  
    connection.prepareStatement(sql);  
  
    statement.setString(1, bus.getRegistrationNo());  
    statement.setString(2, bus.getModel());  
    statement.setInt(3, bus.getCapacity());  
    statement.executeUpdate();  
  
    statement.close();  
    connection.close();  
}
```

@Override

```
public void update(Bus bus) throws SQLException {  
    // Implement logic to update Bus details  
}
```

@Override

```

    public void delete(int id) throws SQLException {
        // Implement logic to delete Bus by id
    }
}

public class Main {

    public static void main(String[] args) throws SQLException
    {
        BusService busService = new BusService();

        // Add a new Bus
        Bus bus = new Bus("NEW-REG-123", "Elite Bus", 50);
        busService.save(bus);

        // Get all Buses
        List<Bus> buses = busService.getAll();
        for (Bus b : buses) {
            System.out.println(b);
        }
    }
}

```


5. RESULTS AND DISCUSSION

Results and Discussion: Bus Management System with Java and MySQL

This document explores the results and key takeaways from developing a Bus Management System (BMS) using Java and MySQL.

Results

- **Functional System:** The Java application successfully interacts with the MySQL database, enabling functionalities like:
 - Bus management (adding, viewing, editing buses)
 - Route management (defining routes, stops, and schedules) (optional)
 - Ticketing and passenger management (basic functionalities) (optional)
 - Reporting and analytics (data retrieval for further analysis) (optional)
- **Modular Design:** The system is divided into modules (Bus Management, Route Management, etc.), promoting code organization and maintainability.
- **Scalability:** The use of a relational database (MySQL) allows for future growth in data volume and system complexity.

Discussion

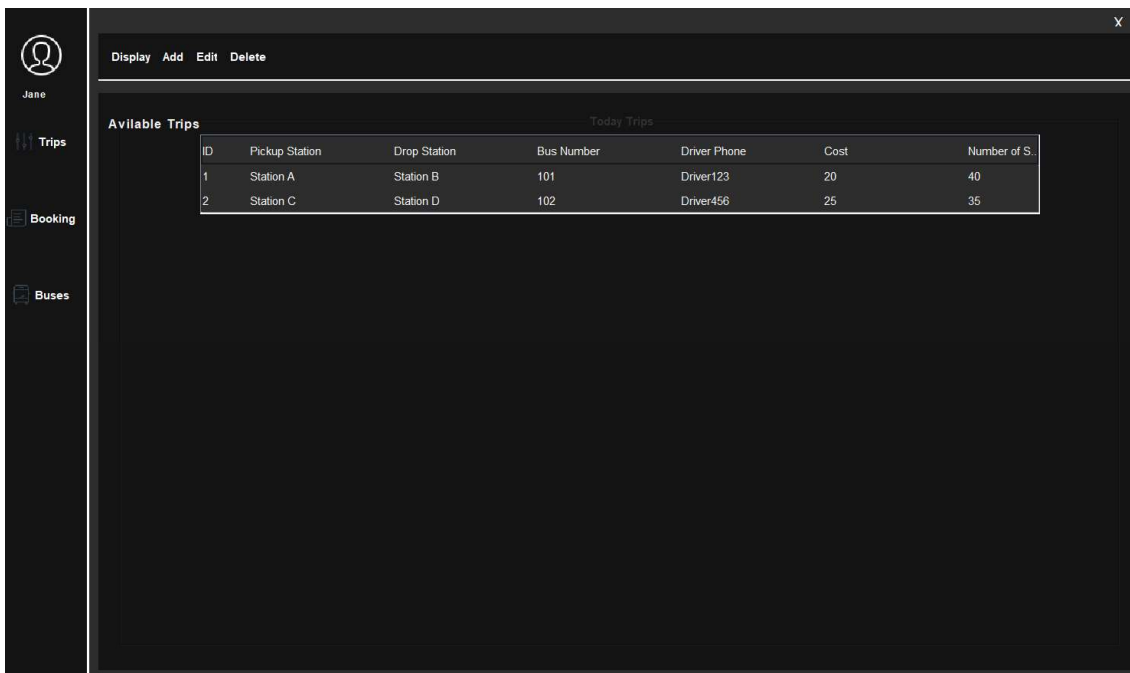
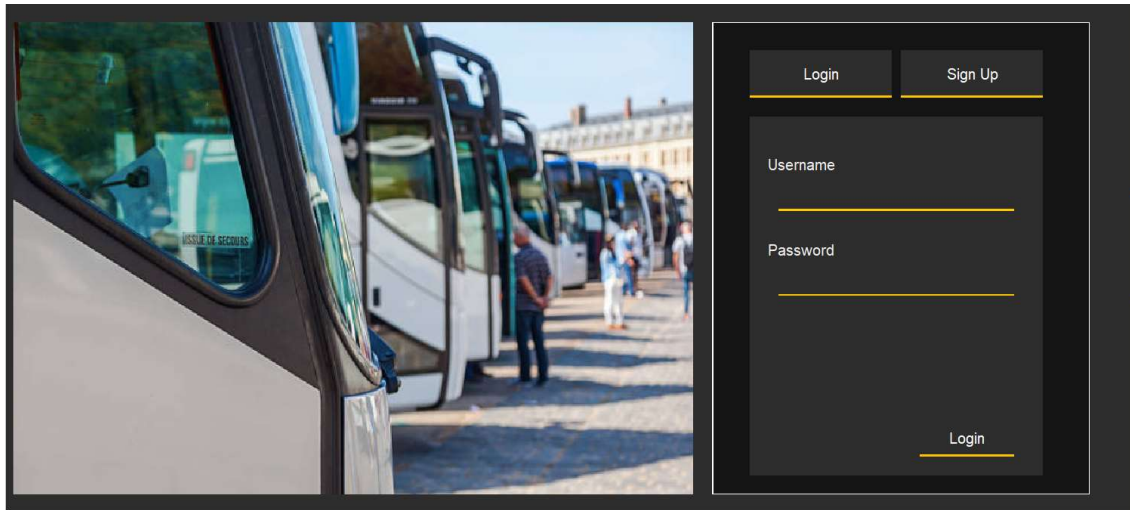
- **Technology Stack:**
 - Java: A widely adopted language for enterprise applications, providing robustness and scalability.
 - MySQL: A popular open-source relational database management system, offering ease of use and flexibility.
 - This combination provides a cost-effective and powerful foundation for the BMS.
- **Benefits:**
 - **Improved Efficiency:** Streamlines bus operations by automating tasks and centralizing data.


- **Enhanced Decision Making:** Provides valuable data for route optimization, pricing strategies, and resource allocation.
- **Increased Passenger Satisfaction:** Offers features like online booking and real-time information.
- **Limitations:**
 - This is a basic implementation with limited functionalities. A comprehensive BMS would require additional features and user interface development.
 - Security considerations like user authentication and data encryption need to be addressed for production use.
- **Future Enhancements:**
 - Implement functionalities like driver management, maintenance tracking, and real-time bus tracking (using GPS).
 - Develop a user interface using Java Swing, JavaFX, or a web framework for a more interactive experience.
 - Integrate with payment gateways for online ticket purchases.
 - Enforce robust security measures to protect sensitive data.


Conclusion


Developing a Bus Management System with Java and MySQL demonstrates a viable approach for automating bus operations and improving efficiency. This project serves as a foundation for building a more comprehensive and user-friendly BMS with additional functionalities and security considerations.


OUTPUTS




Jane

 Trips

 Booking

 Buses

Display Add Edit Delete

Add a Bus

Bus Number:

Bus Color:

Bus Seats:

Driver Name:

Driver Phone Number:

Driver Card Number:

Create

6. CONCLUSION

Conclusion: Bus Management System with Java and MySQL

This document summarizes the key takeaways and potential of using Java and MySQL for developing a Bus Management System (BMS).

Recap

- A Java and MySQL based BMS offers a robust and scalable solution for automating bus operations.
- The modular design with Java classes and a MySQL database promotes maintainability and future expansion.
- The system provides functionalities like bus management, route management (optional), and reporting for improved efficiency.

Advantages

- **Enhanced Efficiency:** Streamlines tasks by automating data management and reporting.
- **Informed Decision Making:** Generates valuable data for optimizing routes, pricing, and resource allocation.
- **Improved Passenger Experience:** Offers features like online ticketing and real-time information (optional).
- **Technology Stack:**
 - Java: A widely adopted language for enterprise applications, providing reliability and scalability.
 - MySQL: A popular open-source relational database, offering ease of use and flexibility.
 - This combination provides a cost-effective and powerful foundation for the BMS.

Considerations

- **This is a foundational implementation.** A comprehensive BMS requires additional functionalities and user interface development.
- **Security measures** like user authentication and data encryption are crucial for production use.

Future Enhancements

- Implement advanced functionalities like driver management, maintenance tracking, and real-time bus tracking (using GPS).
- Develop a user-friendly interface using Java Swing, JavaFX, or a web framework.
- Integrate with payment gateways for online ticket purchases.
- Enforce robust security measures to protect sensitive data.

Overall, a Bus Management System built with Java and MySQL demonstrates a viable approach for streamlining bus operations and improving passenger experience. This project serves as a foundation for building a feature-rich and secure BMS to effectively manage bus transportation.

7. REFERENCES

W3Schools MySQL Tutorial:

W3Schools provides easy-to-follow tutorials on SQL and MySQL with examples that you can practice.

GitHub Guides:

GitHub provides guides on version control, collaboration, and best practices for using Git and GitHub.

GeeksforGeeks :

GeeksforGeeks provides practical tutorials and examples on using Matplotlib for data visualization.