### **1. Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Description** |
| 1.0 | Samarth Patel | 2025-03-22 | Initial Draft of High-Level Design Document |
| 1.1 | Mohammadsaffan Vahora | 2025-03-22 | Added Use Case Descriptions & ERD Diagram |
| 1.2 | Meet Ahalpara | 2025-03-22 | Finalized Application Architecture & DAO Logic |
| 1.3 | Krish Patel | 2025-03-22 | Reviewed and added Testing & Deployment sections. |
| 1.4 | Mohammadsaffan Vahora | 2025-04-06 | Final version |

### **About the System – Public Transit Fleet Management System (PTFMS)**

The **Public Transit Fleet Management System (PTFMS)** is a web-based application developed for public transit agencies to help them manage and monitor their fleet of vehicles such as **Diesel Buses**, **Electric Light Rails**, and **Diesel-Electric Trains**.

The system allows **Transit Managers** and **Operators** to:

* Register and log in with role-based access
* Add and manage vehicle information
* Track real-time vehicle locations using GPS
* Monitor fuel or energy consumption
* Receive predictive maintenance alerts
* View performance dashboards and reports

By using PTFMS, agencies can increase operational efficiency, reduce costs, and improve overall transit service reliability.

### **Purpose of This Document**

This document is the **High-Level Design (HLD)** for the PTFMS project. It acts like a **blueprint** that explains how the system is designed before the actual coding begins. It helps:

* Developers understand the system structure
* Architects visualize component interaction
* Testers plan test strategies
* Stakeholders review design goals and decisions

This document includes:

* Architecture diagrams (Application, Business, Data)
* UML Use Cases and Class Diagrams
* Security and Deployment Models
* Testing strategy and data models

## **3: TARGETED AUDIENCE**

This High-Level Design document is intended for all team members and stakeholders involved in the development, review, testing, and deployment of the **Public Transit Fleet Management System (PTFMS)**.

### **The targeted audience includes:**

1. **Software Developers**
   1. To understand how each module is designed and how different components interact (like Servlets, DAO, etc.)
2. **System Architects**
   1. To review the application’s 3-tier architecture, design patterns used, and how components are structured.
3. **Database Developers**
   1. To understand the Entity-Relationship Diagram (ERD), database design, and how tables relate to system modules.
4. **Test Engineers / QA Testers**
   1. To understand test points from the design and plan their test strategies accordingly.
5. **Project Managers / Product Owners**
   1. To track feature design against project requirements and ensure correct implementation of business logic.
6. **Transit Agency Stakeholders**
   1. To understand how the system supports transit vehicle management and decision-making through dashboards and alerts.

## **4: SCOPE**

### **Project Scope – Public Transit Fleet Management System (PTFMS)**

This document outlines the features and architecture of the **PTFMS**, a system designed for **Transit Managers** and **Operators** to manage, monitor, and optimize transit vehicles such as Diesel Buses, Electric Light Rails, and Diesel-Electric Trains.

### **In-Scope (What the system WILL do)**

1. **User Registration and Login**
   1. Role-based login for Transit Managers and Operators
2. **Vehicle Management**
   1. Add, edit, and assign transit vehicles
   2. Track details like vehicle type, energy type, and assigned route
3. **Real-Time GPS Tracking (Simulated)**
   1. Vehicles report their current location
   2. Operators can manually log out-of-service/break times
4. **Fuel/Energy Consumption Monitoring**
   1. Generate reports on vehicle-specific fuel or energy usage
   2. Alert managers for high consumption
5. **Predictive Maintenance Alerts**
   1. Monitor component usage (brakes, wheels, engine diagnostics)
   2. Notify managers when maintenance is needed
6. **Reporting and Dashboards**
   1. View maintenance schedules, fuel usage, operator performance
7. **Security Features**
   1. Role-based access, password hashing, SQL injection protection
8. **Design Patterns**
   1. Apply GoF design patterns like DAO, Builder, Strategy, etc.

### **Out-of-Scope (What the system WILL NOT do)**

1. **Live GPS Map Integration**
   1. No real GPS or Google Maps display; locations will be simulated or logged manually
2. **Mobile App Support**
   1. Web-based only; no Android/iOS apps included
3. **Integration with External APIs**
   1. No connection to government transit systems or external sensors
4. **Payment and Ticketing**
   1. No ticket sales, fare tracking, or payment gateway integration
5. **AI-based Route Prediction or Optimization**
   1. Routes are static and assigned manually by managers

## **5: APPLICATION ARCHITECTURE**

The **Public Transit Fleet Management System (PTFMS)** is designed using a **3-tier architecture**, where each layer has its own responsibility. This keeps the code clean, organized, and easier to maintain.

### **Subheading 1: 3-Tier Architecture Overview**

#### **1. Presentation Layer (Frontend/UI)**

* Technologies: HTML, CSS
* Role: Handles all user interactions.
* Users: Transit Manager and Operator
* Example Pages: Login, Vehicle Form, Dashboard

#### **2. Business Layer (Logic + Servlets)**

* Technologies: Java Servlets
* Role: Acts like the **brain** of the system.
* Handles user inputs, checks rules, and talks to the data layer.
* Example: LoginServlet, VehicleServlet, ReportServlet

#### **3. Data Layer (Database + DAO Pattern)**

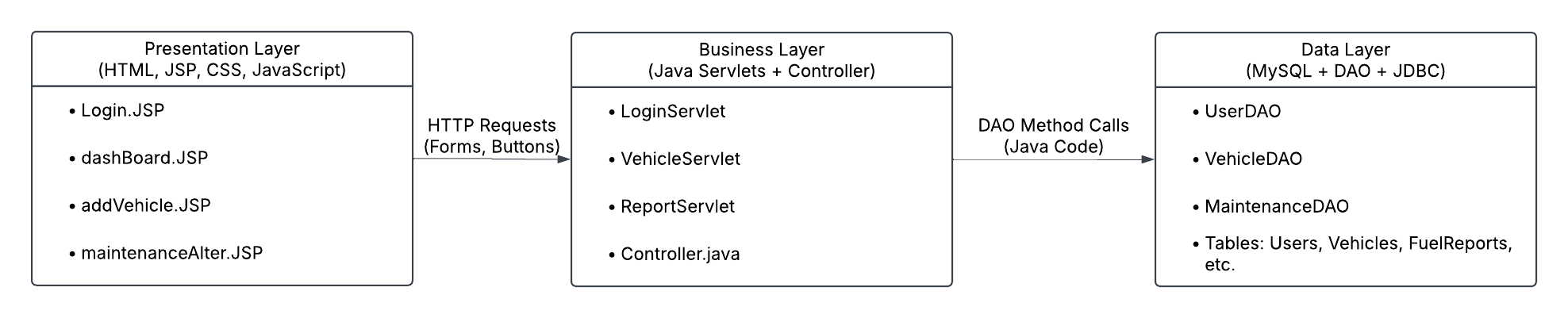
* Technologies: MySQL 8.0.40
* Role: Stores data (users, vehicles, fuel reports, etc.)
* Accessed using **DAO (Data Access Object)** classes
* Example: UserDAO, VehicleDAO

### **Subheading 2: Main Components and Their Roles**

|  |  |  |
| --- | --- | --- |
| **Component** | **Layer** | **Description** |
| LoginServlet | Business Layer | Authenticates users and sets session based on role |
| VehicleServlet | Business Layer | Adds/updates vehicle details |
| UserDAO, VehicleDAO | Data Layer | Handles all SQL queries (INSERT, SELECT, UPDATE) |
| login.jsp | Presentation | UI for user login |
| dashboard.jsp | Presentation | Manager dashboard to view reports |
| User / Vehicle Class | Shared (Model) | Java Beans or POJOs to store user/vehicle information temporarily |

### **GoF Design Patterns Used**

|  |  |
| --- | --- |
| **Pattern** | **Use In PTFMS** |
| **DAO** | Separate database logic from business logic |
| **Builder** | To construct complex vehicle objects step by step |
| **Strategy** | For different fuel calculation strategies per vehicle type |
| **Observer** | To notify when fuel usage crosses threshold |
| **Command** | For scheduling or canceling maintenance tasks |
| **Simple Factory** | For creating Vehicle objects based on selected type (Bus, Train) |



## **6: BUSINESS ARCHITECTURE**

### **Subheading 1: UML Use Case Diagram**

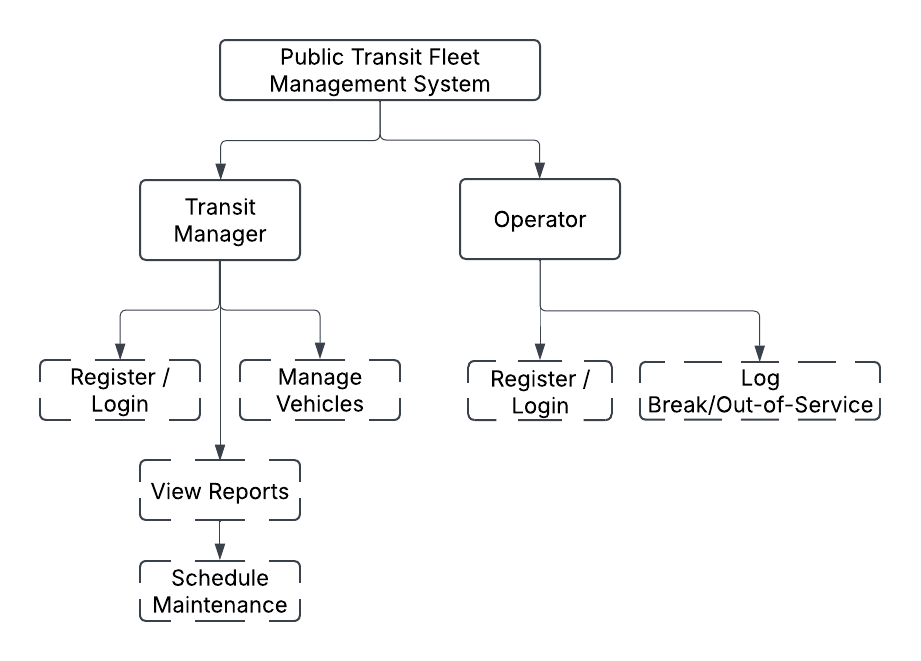
Let's first clearly list **Actors and their Roles**:

|  |  |
| --- | --- |
| **Actor** | **Description / Role** |
| **Transit Manager** | Person who manages vehicles, assigns routes, and checks reports |
| **Operator** | Person who drives vehicles and logs break/out-of-service times |

Now, clearly list **Main Use Cases**. (A **Use Case** is an action or activity done by actors):

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Actor(s)** | **Description** |
| **User Registration** | Manager, Operator | Register new users |
| **Login & Logout** | Manager, Operator | Login to the system securely |
| **Add/Manage Vehicles** | Manager | Add, edit, delete vehicles |
| **View GPS Tracking** | Manager | See real-time or logged vehicle location |
| **Log Break/Out-of-Service** | Operator | Manually log when taking breaks or vehicle not in use |
| **Monitor Fuel/Energy** | Manager | Check fuel/energy usage reports |
| **Get Maintenance Alerts** | Manager | Get alerts about vehicle maintenance |
| **Schedule Maintenance** | Manager | Schedule repairs |
| **View Reports & Dashboards** | Manager | See different dashboards like fuel usage, maintenance |

### **UML Use Case Diagram (Simplified Text Version)**



### **2: Detailed Use Case Descriptions (Example)**

Let’s take one example use case and describe it clearly in detail:

#### **Example Use Case: Add/Manage Vehicles**

* **Use Case Name:**  
   Add/Manage Vehicles
* **Actor:**  
   Transit Manager
* **Description:**  
   A manager can add new vehicles, edit details of existing vehicles, or remove vehicles.
* **Steps to Add Vehicle:**
  + Manager logs in.
  + Manager opens the "Vehicle Management" page.
  + Clicks "Add Vehicle" button.
  + Enters details (Vehicle Type, Vehicle Number, Fuel Type, Passengers limit, assigned route).
  + Clicks "Submit".
  + The system saves this information in the database.
  + System shows confirmation message "Vehicle added successfully".
* **Example Scenario:**  
   Manager wants to add a new **Diesel Bus** (Bus-120) to route **#55**:
  + Vehicle Type: Diesel Bus
  + Vehicle Number: Bus-120
  + Fuel Type: Diesel
  + Passengers: 50
  + Route: 55

The manager fills the form, clicks submit, and the vehicle is successfully registered and visible on the dashboard.

## **7: DETAILED DESIGN**

This section includes detailed diagrams explaining:

* **Classes** (objects that store data and behavior)
* **Components** (bigger parts/modules of our system)

Let’s start simply!

### **Subheading 1: UML Class Diagram**

Here’s a clear breakdown of important classes in our **PTFMS system**:

|  |  |
| --- | --- |
| **Class Name** | **Purpose/Description** |
| Analytics DAOImpl | Provides vehicle, maintenance, and fuel analytics from the database. |
| **Update Vehicle Status** | Updates vehicle status via POST request. |
| **MaintenanceAlert** | Alerts when vehicle needs fixing |
| **UserDAO** | Connects to DB, handles user data |
| **VehicleDAO** | Connects to DB, handles vehicle data |
| **LoginServlet** | Manages login/logout actions |
| **VehicleServlet** | Adds/edits vehicles |

### **Class Diagram (Textual UML)**

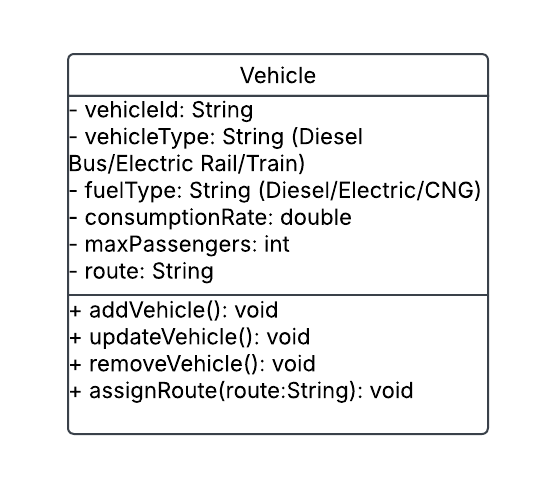
### **1. AnalyticsDAOImpl**

* Stores information about vehicle, maintenance and fuel analytics.

A white rectangular sign with black text

AI-generated content may be incorrect.

### **2. Update Vehicle status**

* Update vehicle status via POST request.  
  

### **3. MaintenanceServlet Class**

* Generates and stores maintenance alerts for vehicles.

A close-up of a computer code

AI-generated content may be incorrect.

### **4. DAO Classes**

* Handles interaction between Business Layer (Servlets) and Data Layer (MySQL).

A close-up of a computer code

AI-generated content may be incorrect.A close-up of a computer screen

AI-generated content may be incorrect.

### **5. Servlets (Controllers)**

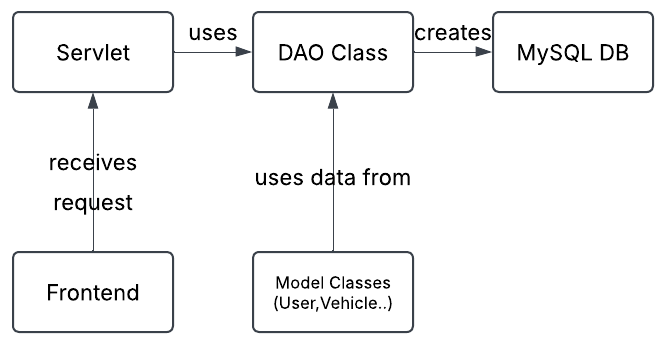
* Handle logic between Frontend (UI) and DAOs.

A close-up of a computer code

AI-generated content may be incorrect. A close-up of a vehicle service

AI-generated content may be incorrect.

Detailed UML Class Diagram (Simplified)



### **2: Component Diagram**

This clearly explains how different parts (components) of our system interact:

The system is divided into clear **modules/components**:

* **Frontend (Presentation)**
* **Controller (Servlets)**
* **DAO Layer (Data Access)**
* **MySQL Database**

**Manager (Browser)**

**↓ (login request)**

**Presentation Layer (JSP form)**

**↓ (POST request)**

**LoginServlet (checks credentials)**

**↓ (calls DAO method)**

**UserDAO.validateUser(email,password)**

**↓ (executes SQL Query)**

**MySQL Database [User table]**

**↓ (returns user details)**

**UserDAO (validates credentials)**

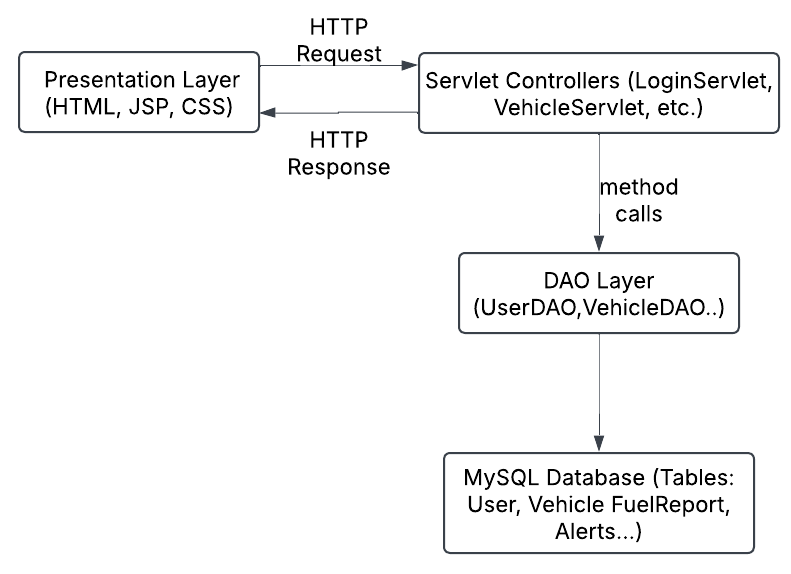
**↓ (true/false returned)**

**LoginServlet (decides login success/failure)**

**↓ (redirects)**

**Presentation Layer (Dashboard or Error page)**

#### **Component Diagram (Textual Version)**



## **8: DATA ARCHITECTURE**

### **Subheading 1: Database Structure**

The **PTFMS** database includes clearly defined tables:

* **Users**
* **Vehicles**
* **FuelReports**
* **MaintenanceAlerts**

Let’s clearly explain each table in detail:

### **1. Users Table**

Stores **Transit Managers** and **Operators** information:

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Description** |
| id | INT (PK) | Unique ID for each user (auto-increment) |
| name | VARCHAR(50) | Name of user |
| email | VARCHAR(100) | User email (unique) |
| password | VARCHAR(255) | Password (encrypted/hash) |
| role | VARCHAR(20) | Role ("Manager" or "Operator") |

### **2. Vehicles Table**

Stores details of each vehicle:

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Description** |
| vehicleId | VARCHAR(20)(PK) | Unique Vehicle ID (Bus-001, Train-01) |
| vehicleType | VARCHAR(30) | Diesel Bus/Electric Rail/Train |
| fuelType | VARCHAR(30) | Diesel, Electric, CNG |
| consumptionRate | DOUBLE | Fuel/Energy consumption rate |
| maxPassengers | INT | Maximum number of passengers |
| route | VARCHAR(50) | Assigned route number |
| userId | INT (FK) | Reference to Users (Manager) table |

### **3. FuelReports Table**

Stores details about fuel/energy consumption:

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Description** |
| reportId | INT (PK) | Unique Fuel Report ID (auto-increment) |
| vehicleId | VARCHAR(20)(FK) | Vehicle ID from Vehicles table |
| fuelUsed | DOUBLE | Amount of fuel or energy used |
| reportDate | DATE | Date of report generation |

### **4. MaintenanceAlerts Table**

Stores information about vehicle maintenance issues:

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Description** |
| alertId | INT (PK) | Unique Alert ID (auto-increment) |
| vehicleId | VARCHAR(20)(FK) | Vehicle ID (from Vehicles table) |
| issueType | VARCHAR(50) | Issue (e.g., brakes, tires, engine) |
| alertDate | DATE | Date alert created |

## **2: ERD Diagram (Entity Relationship Diagram)**

Let's clearly show relationships between these tables (Entities):

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## **Detailed Explanation of ERD Relationships:**

* **One Manager can manage many Vehicles**  
   *(Users table → Vehicles table)*
* **One Vehicle can have multiple Fuel Reports**  
   *(Vehicles table → FuelReports table)*
* **One Vehicle can have multiple Maintenance Alerts**  
   *(Vehicles table → MaintenanceAlerts table)*

## **Physical/Logical Data Model (Simplified)**

**Logical:** clearly describes tables, fields, data types (done above).

**Physical:** how this looks inside MySQL. Here's an example SQL structure clearly defined for one table:

### **Example SQL for Users table:**

CREATE TABLE Users (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(50) NOT NULL,  
 email VARCHAR(100) UNIQUE NOT NULL,  
 password VARCHAR(255) NOT NULL,  
 role VARCHAR(20) NOT NULL  
);

## **9: SECURITY ARCHITECTURE**

Security Architecture explains **how we will protect our system from unauthorized access and potential security threats**.

The PTFMS has two key security areas:

1. **User Authentication and Authorization**
2. **Data Protection and Database Security**

### **Subheading 1: User Authentication and Authorization**

This clearly explains how users safely log in and use the system:

* **Role-Based Access Control (RBAC)**
  + Users have clear roles (Manager or Operator).
  + Managers have access to all features.
  + Operators have limited access (e.g., logging breaks).

**Example clearly explained:**

* A Manager logs in and sees all vehicles.
* An Operator logs in and sees limited options.
* **Password Security**
  + Passwords securely stored (using **hashed**).
  + Clear example:
    - Original password: "mypassword"
    - Stored hashed password could be: "b109f3bbbc244eb82441917ed06d618b9008dd09..."

### **Subheading 2: Data Protection and Database Security**

Explains how the atabase stays safe:

* **Preventing SQL Injection**
  + Clearly use Java **Prepared Statements** to safely handle SQL queries.

**Example clearly explained:**

* Instead of unsafe queries like:

SELECT \* FROM Users WHERE email = '" + email + "'";

* Clearly use safe prepared queries:

SELECT \* FROM Users WHERE email = ?;

* **Database Access Security**
  + Clearly use secure connection strings.
  + Limit database access rights (only read/write permission).
* **Secure Session Management**
  + Clear use of secure sessions in Java Servlets.
  + Sessions timeout after inactivity (e.g., 15-30 minutes).

## **10: DEPLOYMENT ARCHITECTURE**

Deployment Architecture clearly describes the technical infrastructure and environment needed to host and run the system.

The PTFMS deployment clearly includes:

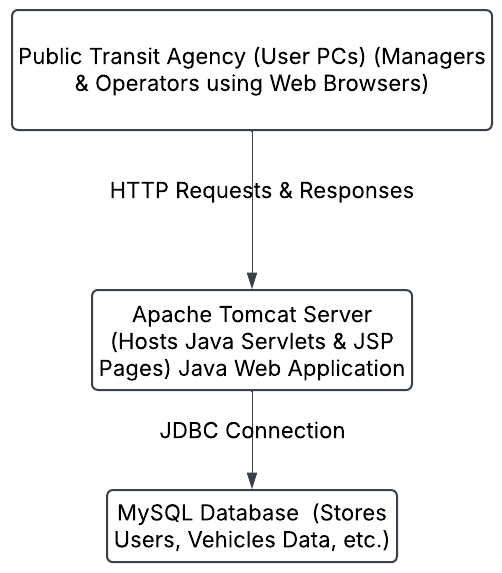
* **Infrastructure Components**
* **Deployment Steps**

### **Subheading 1: Infrastructure Components**

This describes clearly the technologies used:

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology/Software** | **Version** |
| Server | Apache Tomcat | 9.0.0 |
| Database | MySQL | 8.0.0 |
| Programming Language | Java | Java 21 |
| IDE | NetBeans (with Maven) | Latest Version |
| Operating System | Windows/Linux | Any supported |
| Web Browser | Chrome, Firefox, Edge | Latest Version |

### **Clear Visual Diagram of Deployment:**



### **2: Deployment Steps Clearly Explained**

Let's clearly explain how we deploy (set up) our project:

**Step 1: Prepare Application**

* Build the Java web application (WAR file) using **NetBeans + Maven**.

**Step 2: Server Setup (Apache Tomcat)**

* Install Apache Tomcat 9.0.0
* Ensure Tomcat is running properly on port 8080.

**Step 3: Database Setup (MySQL)**

* Install MySQL Server 8.0.0
* Create database called ptfms
* Run provided .sql script clearly to set up tables:

CREATE DATABASE ptfms;

USE ptfms;

CREATE TABLE Users (...);

CREATE TABLE Vehicles (...);

CREATE TABLE FuelReports (...);

CREATE TABLE MaintenanceAlerts (...);

**Step 4: Configure Database Connection**

* Create .properties file clearly:

jdbc.url=jdbc:mysql://localhost:3306/ptfms  
jdbc.username=cst8288  
jdbc.password=cst8288

**Step 5: Deploy Application to Tomcat**

* Copy project.war to Tomcat’s webapps folder.
* Tomcat auto-deploys the WAR file, and the application becomes available at:

[http://localhost:8080/project](http://localhost:8080/exampleProject)

**Step 6: Test Deployment**

* Clearly test accessing the app via browser  
   (e.g., login, add vehicles, view dashboards).

## **11: TESTING MODEL**

**Testing Model** describes clearly **how we verify the application is working correctly**.

Testing includes two clear parts:

1. **Unit Testing (Automatic Testing)**
2. **Manual Testing**

### **Subheading 1: Unit Testing (Automated)**

Clearly explained purpose:

* Tests smaller pieces (units) of code automatically.
* Uses Java's **JUnit** framework to clearly test DAO classes and Servlet methods.

#### **Clear Example of Unit Testing DAO (UserDAO):**

* **Scenario:** Testing user login functionality.

**Components clearly tested with Unit Testing:**

|  |  |  |
| --- | --- | --- |
| **Component** | **Test Example** | **Tools Used** |
| DAO Classes | UserDAO (user validation), VehicleDAO | JUnit |
| Java Servlets | LoginServlet, VehicleServlet | Mockito |
| Database Operations | Insert, Update, Delete operations | JUnit |

### **2: Manual Testing (Functional/UI Testing)**

Manual Testing clearly ensures that the application behaves correctly from a user's viewpoint.

#### **Clear Manual Testing Scenarios:**

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Steps Clearly Defined** | **Expected Result Clearly Defined** |
| User Login | Enter valid email/password, click login. | Redirects to dashboard page |
| Vehicle Addition | Manager adds new vehicle with correct details. | Vehicle added, confirmation message appears |
| Fuel/Energy Consumption | Generate report clearly for vehicle. | Accurate fuel usage report displayed |
| Maintenance Alerts | Trigger maintenance alert scenario manually. | Alert clearly generated and shown in UI |
| Role-based Access (Operator) | Login as operator, test restricted functionality. | Operator sees limited features |

### **Detailed Test Plan Clearly Explained Example:**

**Scenario:** Add new Diesel Bus to system

* **Steps Clearly:**
  + Login as **Manager**
  + Click "**Add Vehicle**"
  + Fill in details:
    - Vehicle Type: Diesel Bus
    - Vehicle ID: BUS-101
    - Fuel Type: Diesel
    - Max Passengers: 45
    - Route: 15
  + Click "**Submit**"
* **Expected Result:**
  + Vehicle BUS-101 added
  + Clearly see confirmation message "**Vehicle added successfully**"
  + Verify vehicle is in database clearly using MySQL

## **12: REFERENCES**

Clearly list the important references you used to design the **Public Transit Fleet Management System (PTFMS)**.

# References

|  |  |
| --- | --- |
| [1] | ORACLE, "Oracle Java Documentation," Oracle, [Online]. Available: https://docs.oracle.com/javaee/7/tutorial/servlets.htm. |
| [2] | MySQL, "MySQL," [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/. |
| [3] | A. T. 9, "Apache Tomcat 9," Apache, [Online]. Available: https://tomcat.apache.org/tomcat-9.0-doc/index.html. |

<https://tomcat.apache.org/tomcat-9.0-doc/index.html>

## **13: ACRONYMS & ABBREVIATIONS**

Clearly explain the short forms (acronyms) used throughout the document:

### **Clear Acronyms/Abbreviations Table:**

|  |  |
| --- | --- |
| **Acronym** | **Meaning Clearly Defined** |
| **PTFMS** | Public Transit Fleet Management System |
| **DAO** | Data Access Object |
| **JDBC** | Java Database Connectivity |
| **ERD** | Entity Relationship Diagram |
| **GPS** | Global Positioning System |
| **HTML** | HyperText Markup Language |
| **CSS** | Cascading Style Sheets |
| **SQL** | Structured Query Language |
| **RBAC** | Role-Based Access Control |
| **IDE** | Integrated Development Environment |
| **PK** | Primary Key |
| **FK** | Foreign Key |
| **JUnit** | Java Unit Testing Framework |
| **WAR** | Web Application Archive |
| **SHA-256** | Secure Hash Algorithm (used for passwords) |
| **UI** | User Interface |
| **Maven** | Java build and dependency management tool |

List of Figures

[Figure 1: 3-Tier Application Architecture](#Bookmark1)

[Figure 2: UML Use Case Diagram](#Bookmark2)

[Figure 3: Analystic DAOImpl L](#Bookmark3)

[Figure 4: Update Vehicle Status L](#Bookmark4)

[Figure 5: MaintenanceAlert Class UML](#Bookmark6)

[Figure 6: User DAO Class UML](#UserDAO_UML)

[Figure 7: Vehicle DAO Class UML](#Bookmark8)

[Figure 8: Login Servlet UML](#Bookmark9)

[Figure 9: Vehicle Servlet UML](#Bookmark10)

[Figure 10: Detailed UML Class Diagram (Simplified)](#Bookmark12)

[Figure 11: Component Diagram](#Bookmark13)

[Figure 12: E-R Diagram](#Bookmark14)

[Figure 13: Clear Visual Diagram of Deployment](#Bookmark15)