Final Project Report – Vehicle Maintenance & Service Management System

Courses: ITE5215 – Advanced Java (Frontend), ITE5220 – Oracle PL/SQL Programming (Backend)

Team Members: **Slade Sahoye, Homer Bernardino, Carlos Gabriel Cueto, Xuan Khai Dao**

Submission Date: April 10th, 2025

Instructor Name: Shahdad Shariatmadari

# 2. Executive Summary

This **Vehicle Maintenance System** is a desktop application developed by *Torque Titans* was designed to streamline vehicle service management for mechanics and administrators. Mechanics can log in to view and update their assigned tasks (e.g., marking a service as complete), while administrators can monitor overall service statistics, such as total services completed and the most requested service type. The system ensures secure user authentication, efficient task management, and a user-friendly interface for both roles.

**Technologies Used**:

* **Backend**: Oracle SQL Developer (database), PL/SQL (stored procedures), Java (DAO layer with JDBC).
* **Frontend**: JavaFX (UI framework), FXML (UI layout), CSS (styling).
* **Development Tools**: IntelliJ IDEA (IDE), Oracle SQL Developer (database management), JavaFX Scene Builder (UI design).

**Key Features**:

* Secure login for mechanics and admins using the UserMechanicMapping table.
* Mechanic Dashboard to view and update assigned tasks (pending/ongoing services).
* Admin Dashboard with statistics (e.g., total services completed, most requested service).
* Modular backend with stored procedures for database operations.
* Responsive and user-friendly frontend with input validation and tooltips.

# 3. Project Objectives

The main goals of the Vehicle Maintenance System project were to create an efficient and user-friendly application for vehicle service management. Below are the functional and non-functional requirements:

**Functional Requirements**:

* **User Authentication**: Allow mechanics and admins to log in securely using username and password.
* **Task Management**: Enable mechanics to view their assigned tasks (pending/ongoing) and mark them as completed.
* **Statistics Display**: Provide admins with statistics, such as total services completed and the most requested service type.
* **Data Management**: Store and manage vehicle service records, customer details, and mechanic information in a relational database.

**Non-Functional Requirements**:

* **Performance**: Ensure the system responds to user actions (e.g., marking a task as complete) within 2 seconds.
* **Security**: Use stored procedures to prevent SQL injection and securely store login credentials.
* **Usability**: Design an intuitive UI with clear labels, tooltips, and input validation.
* **Scalability**: Build a modular backend that can support additional features (e.g., new dashboards) in the future.

# 4. System Design

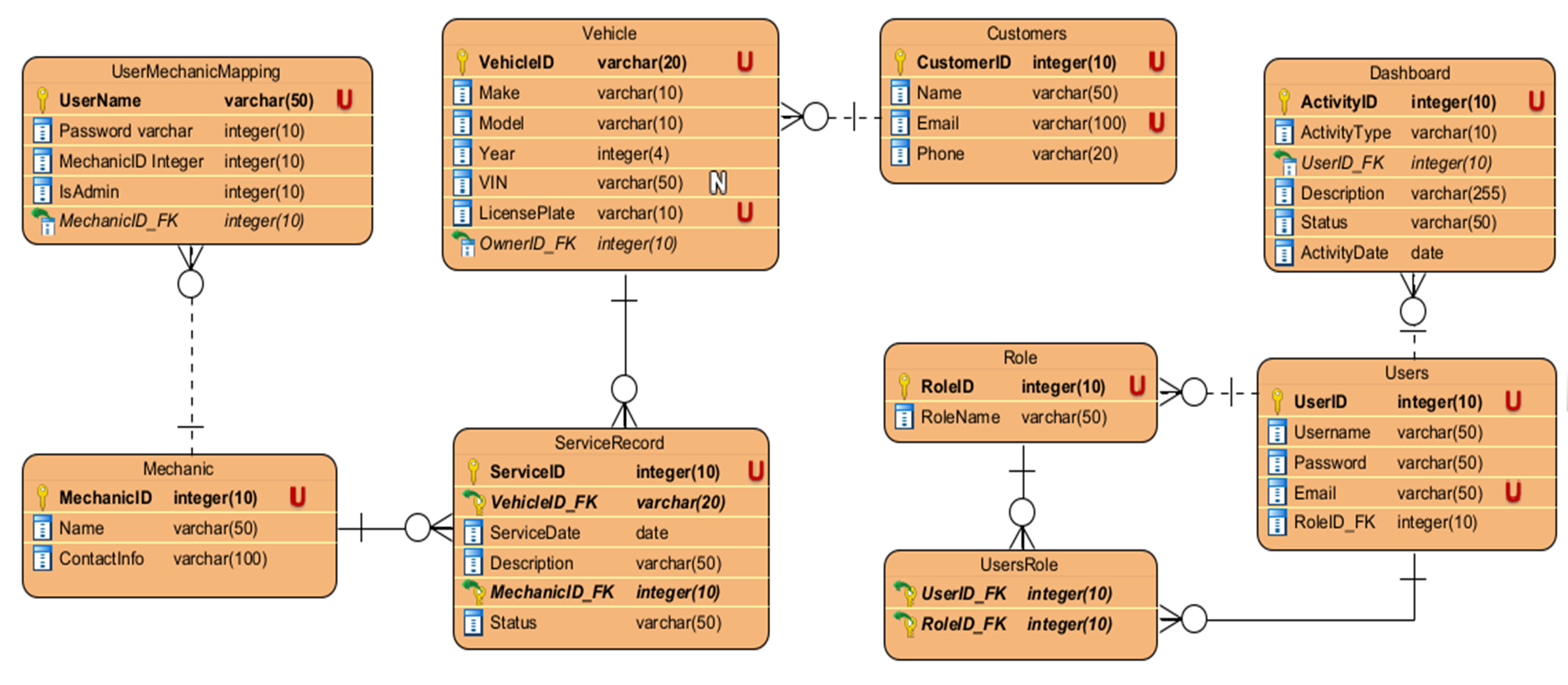
### System Architecture

The Vehicle Maintenance System follows a three-tier architecture:

* **Presentation Layer**: JavaFX frontend with FXML for UI design, CSS for styling, and controllers for user interaction.
* **Application Layer**: Java-based business logic, including controllers (LoginController, MechanicDashboardController) and DAOs (ServiceRecordDAO, UserMechanicMappingDAO) for database interaction.
* **Data Layer**: Oracle SQL Developer database with tables and PL/SQL stored procedures for secure and efficient data operations.

### Design Diagrams

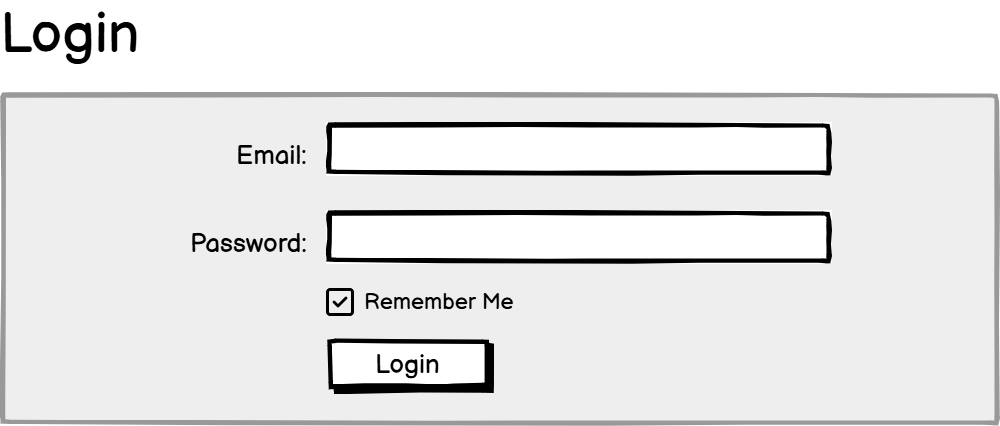
1. **ER Diagram**:



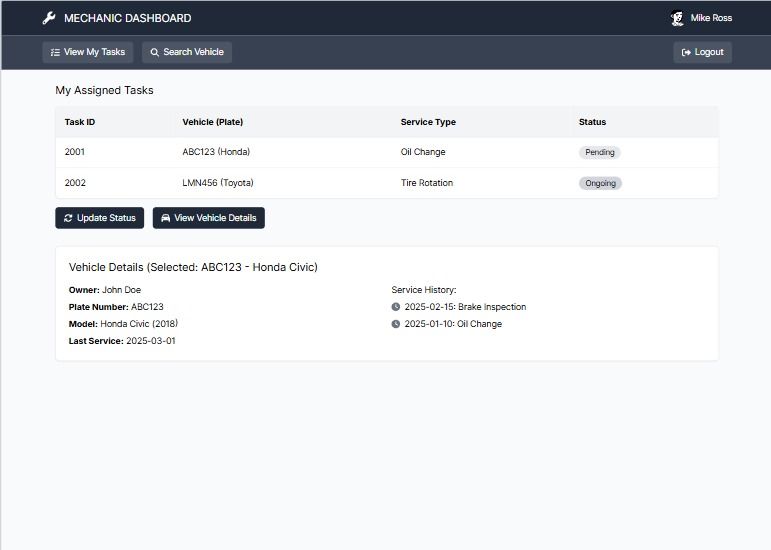
*Description*:

* 1. Entities and Relationships:
     1. Mechanic (MechanicID, FirstName, LastName)
     2. UserMechanicMapping (Username, Password, MechanicID, IsAdmin) → Foreign Key: MechanicID references Mechanic.
     3. Customers (CustomerID, FirstName, LastName, Email, Phone)
     4. Vehicle (VehicleID, CustomerID, Make, Model, Year, Plate) → Foreign Key: CustomerID references Customers.
     5. ServiceRecords (ServiceID, VehicleID, MechanicID, ServiceType, ServiceDate, Status, Description) → Foreign Keys: VehicleID references Vehicle, MechanicID references Mechanic.  
         *To Include*: Create this diagram using Oracle SQL Developer’s ERD tool or a tool like Lucidchart.

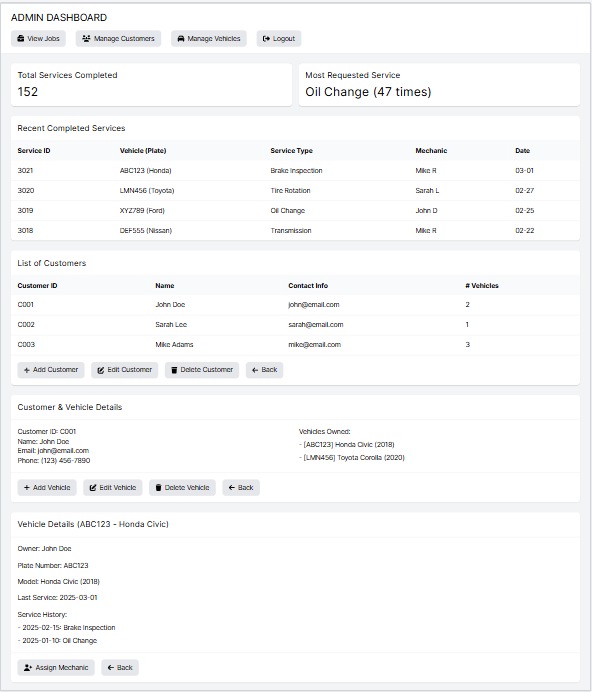
1. **GUI Mockups/Screenshots**:
   1. **Login Screen**: A grid layout with fields for username and password, a login button, and an error label



* 1. **Mechanic Dashboard**: A table view showing assigned tasks (columns: Service ID, Vehicle, Customer, Service Type, Status) and a "Mark as Complete" button



* 1. **Admin Dashboard**: A section displaying pending/completed jobs for all mechanics



1. **Component/System Architecture Diagram** (Placeholder):  
    *Description*:
   1. Same as the architecture diagram in the Executive Summary:
      1. **Client Layer**: JavaFX frontend.
      2. **Application Layer**: Java controllers and DAOs.
      3. **Database Layer**: Oracle SQL Developer with tables and stored procedures.  
          *To Include*: Reuse the diagram from the Executive Summary or create a more detailed version showing specific components (e.g., LoginController → UserMechanicMappingDAO → UserMechanicMapping table).

# 5. Implementation

### A. Backend (ITE5220)

The backend was implemented using Oracle SQL Developer for the database, PL/SQL for stored procedures, and Java for the data access layer.

**Tables**:

* Mechanic (MechanicID, FirstName, LastName): Stores mechanic details.
* UserMechanicMapping (Username, Password, MechanicID, IsAdmin): Stores login credentials and roles.
* Customers (CustomerID, FirstName, LastName, Email, Phone): Stores customer information.
* Vehicle (VehicleID, CustomerID, Make, Model, Year, Plate): Stores vehicle details.
* ServiceRecords (ServiceID, VehicleID, MechanicID, ServiceType, ServiceDate, Status, Description): Tracks service records.

**Stored Procedures**:

* GetAssignedTasks(p\_MechanicID, p\_Result): Fetches pending/ongoing tasks for a mechanic, returning a SYS\_REFCURSOR.
* UpdateServiceStatus(p\_ServiceID, p\_NewStatus): Updates the status of a service record in the ServiceRecords table.
* GetTotalServicesCompleted(p\_Total): Returns the total number of completed services.
* GetMostRequestedService(p\_ServiceName): Returns the most requested service type.

**Functions/Triggers**:

* Auto-increment triggers for each table (e.g., CUSTOMERS\_BI uses Customers\_SEQ to generate CustomerID).

**Implementation Details**:

* The backend uses a Data Access Object (DAO) pattern to interact with the database. For example, ServiceRecordDAO contains methods like getAssignedTasks and updateServiceStatus, which call the corresponding stored procedures using JDBC.
* Stored procedures were used to ensure security (preventing SQL injection) and improve performance (precompiled queries).
* Exception handling was implemented in the DAO layer to catch and handle SQLExceptions, providing meaningful error messages to the frontend.

### B. Frontend (ITE5215)

The frontend was implemented using JavaFX, with FXML for UI design, CSS for styling, and controllers for handling user interactions.

**MVC Modules**:

* **Login Module**:
  + **Model**: UserMechanicMappingDAO (fetches user data for authentication).
  + **View**: Login.fxml (grid layout with username/password fields, login button, error label).
  + **Controller**: LoginController.java (handles login logic and validation).
* **Mechanic Dashboard Module**:
  + **Model**: ServiceRecordDAO (fetches and updates service records).
  + **View**: MechanicDashboard.fxml (table view for tasks, "Mark as Complete" button).
  + **Controller**: MechanicDashboardController.java (loads tasks, handles task updates).
* **Admin Dashboard Module** (partially implemented):
  + **Model**: ServiceRecordDAO (fetches statistics).
  + **View**: AdminDashboard.fxml (displays statistics).
  + **Controller**: AdminDashboardController.java (loads statistics).

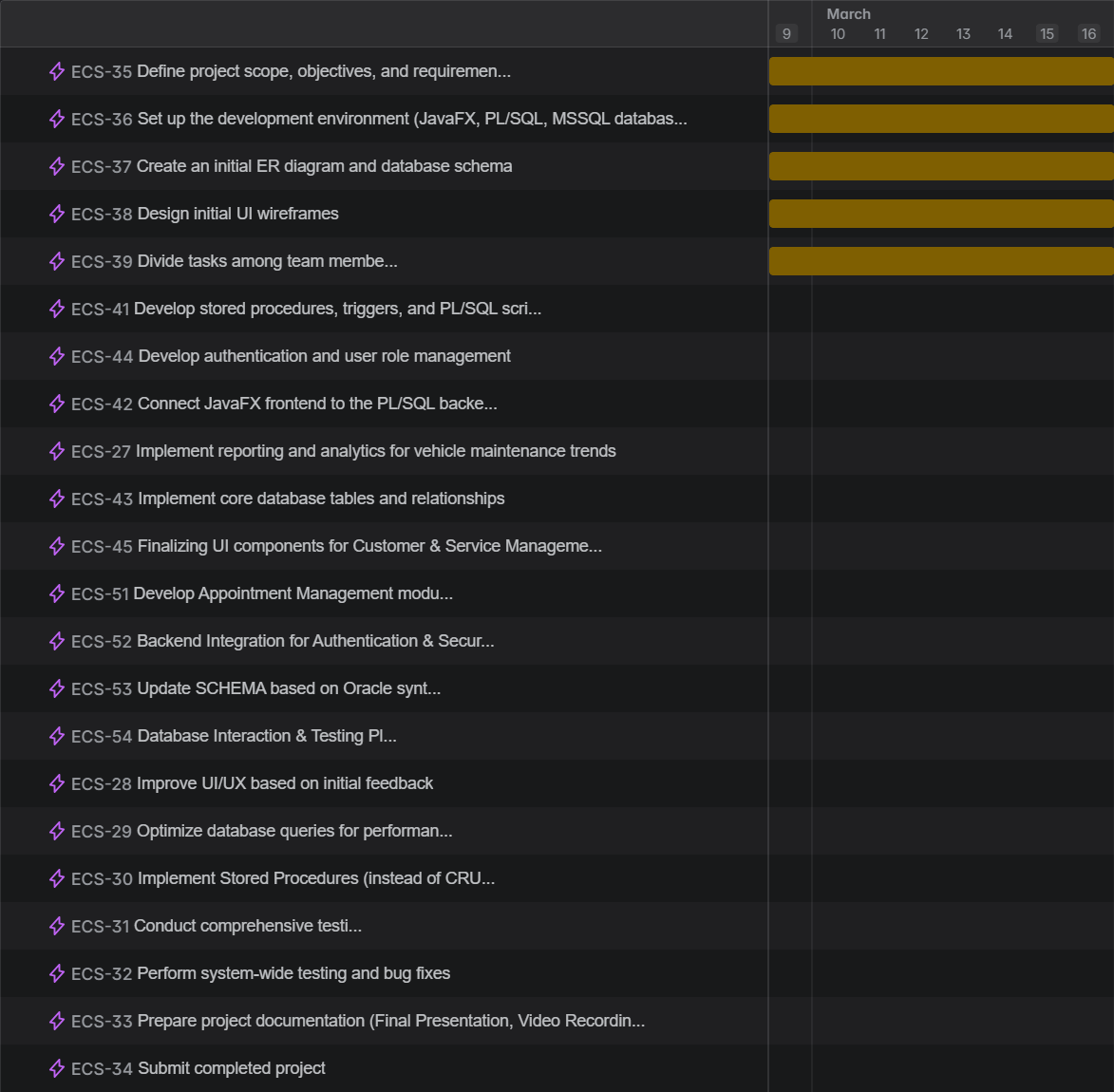
**Key Functionalities/Features**:

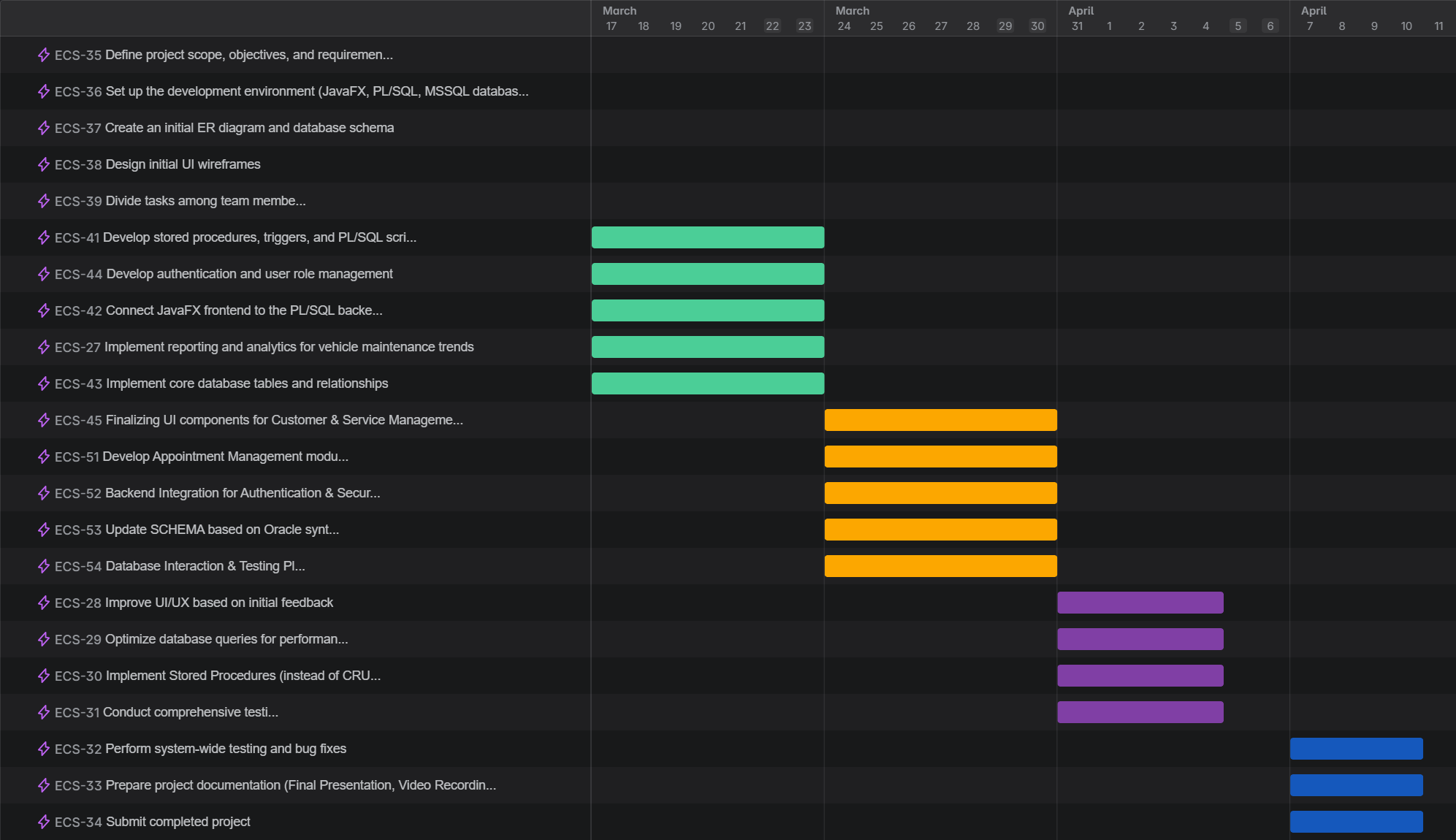
* **Secure Login**: Validates username and password, redirecting to the Mechanic or Admin Dashboard based on the user’s role.
* **Task Management**: Mechanics can view their assigned tasks in a table and mark them as completed, triggering a backend update.
* **Statistics Display**: Admins can view statistics (total services completed, most requested service) in the Admin Dashboard.
* **Input Validation**: Ensures fields (e.g., username, password) are not empty, disabling the login button if invalid.
* **User-Friendly Design**: Includes tooltips (e.g., on the "Mark as Complete" button) and consistent styling via CSS.

**Implementation Details**:

* JavaFX was used to create a responsive UI with FXML for layout design and CSS for styling.
* The MVC pattern was followed to separate concerns: models handle data, views define the UI, and controllers manage user interactions.
* The frontend integrates with the backend via DAOs. For example, MechanicDashboardController calls ServiceRecordDAO.getAssignedTasks to populate the task table and ServiceRecordDAO.updateServiceStatus to mark tasks as complete.

# 6. Project Timeline





# 7. Team Collaboration & Roles

**Team Members and Responsibilities**:

* **Team Member 1 (Backend Lead)**: Designed the database schema, implemented stored procedures, and developed the DAO layer (ServiceRecordDAO, UserMechanicMappingDAO).
* **Team Member 2 (Frontend Lead)**: Designed the UI using JavaFX Scene Builder, implemented FXML files (Login.fxml, MechanicDashboard.fxml), and developed controllers (LoginController, MechanicDashboardController).
* **Team Member 3 (Integration & Testing)**: Integrated the frontend and backend, tested the application (e.g., login, task updates, statistics display), and prepared screenshots for the presentation.
* **Team Member 4 (Documentation & Presentation)**: Prepared the project report, created diagrams (e.g., ER diagram, architecture diagram), and coordinated the presentation slides.

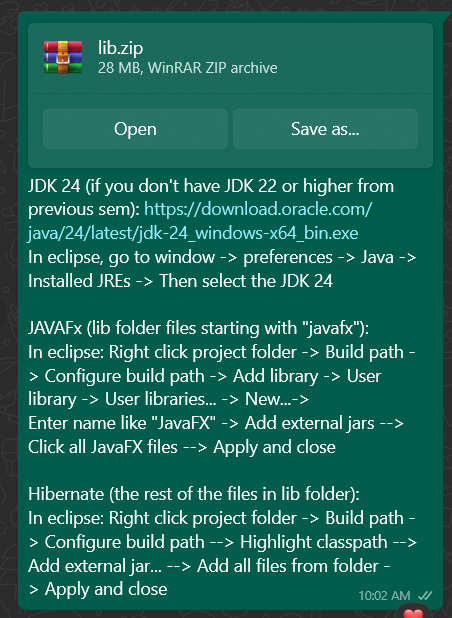
**Task Division**:

* Tasks were divided based on expertise: backend team members focused on database and DAO implementation, while frontend team members focused on UI design and user interaction.
* Weekly meetings were held to track progress, resolve issues, and ensure integration between frontend and backend.

# 8. Challenges & Solutions

**Challenge 1: Oracle SQL Developer Setup**

* **Issue**: Some team members faced issues setting up Oracle SQL Developer and connecting it to the Java application via JDBC.
* **Solution**: We followed Oracle’s official documentation to install the correct JDBC driver (ojdbc8.jar) and updated the connection URL in ServiceRecordDAO and UserMechanicMappingDAO. Slade also shared a step-by-step setup guide among the team:



**Challenge 2: Frontend-Backend Integration**

* **Issue**: The MechanicDashboardController initially failed to load tasks due to a mismatch in the ServiceRecord model and the data returned by GetAssignedTasks.
* **Solution**: We updated the ServiceRecord model to match the stored procedure’s output (e.g., adding serviceType as a String) and added proper exception handling in the DAO layer to debug issues.

**Challenge 3: Time Management**

* **Issue**: The team underestimated the time required for UI design and integration, leading to delays in testing.
* **Solution**: We simplified the UI (e.g., removed unnecessary features like a logout button) and focused on core functionality. We also allocated extra time in Week 5 for testing and bug fixing.

# 9. Future Improvements

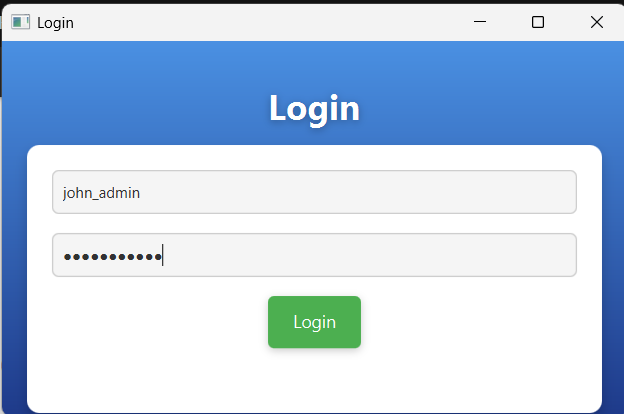
* **Real-Time Updates**: Add a background thread to the Mechanic Dashboard to auto-refresh the task table every 30 seconds, ensuring mechanics see the latest tasks without manual refresh.
* **Customer Management**: Implement a feature for admins to add/edit customers, with corresponding backend stored procedures (AddCustomer, EditCustomer) and a new frontend module.
* **Service History**: Add a feature for admins to view a vehicle’s service history, fetching all past ServiceRecords for a given VehicleID.
* **Enhanced Security**: Hash passwords in the UserMechanicMapping table using a library like BCrypt and implement session management for logged-in users.
* **Mobile Support**: Port the application to a mobile-friendly framework (e.g., JavaFX Mobile or a web-based frontend using React) to support mechanics on the go.

# 10. Appendix

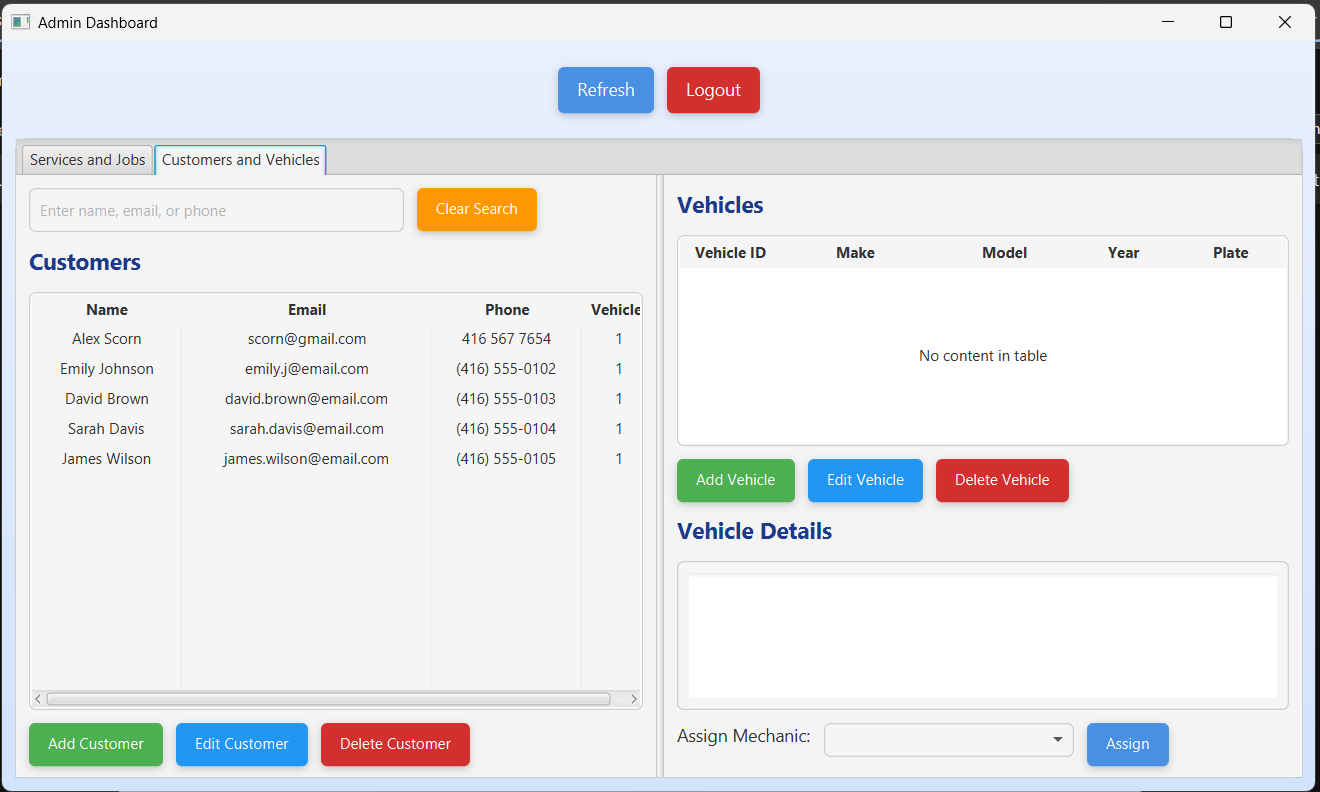
**GitHub:** <https://github.com/SladePunch/torquetitansapp>

**Adding Customer Scenario:**

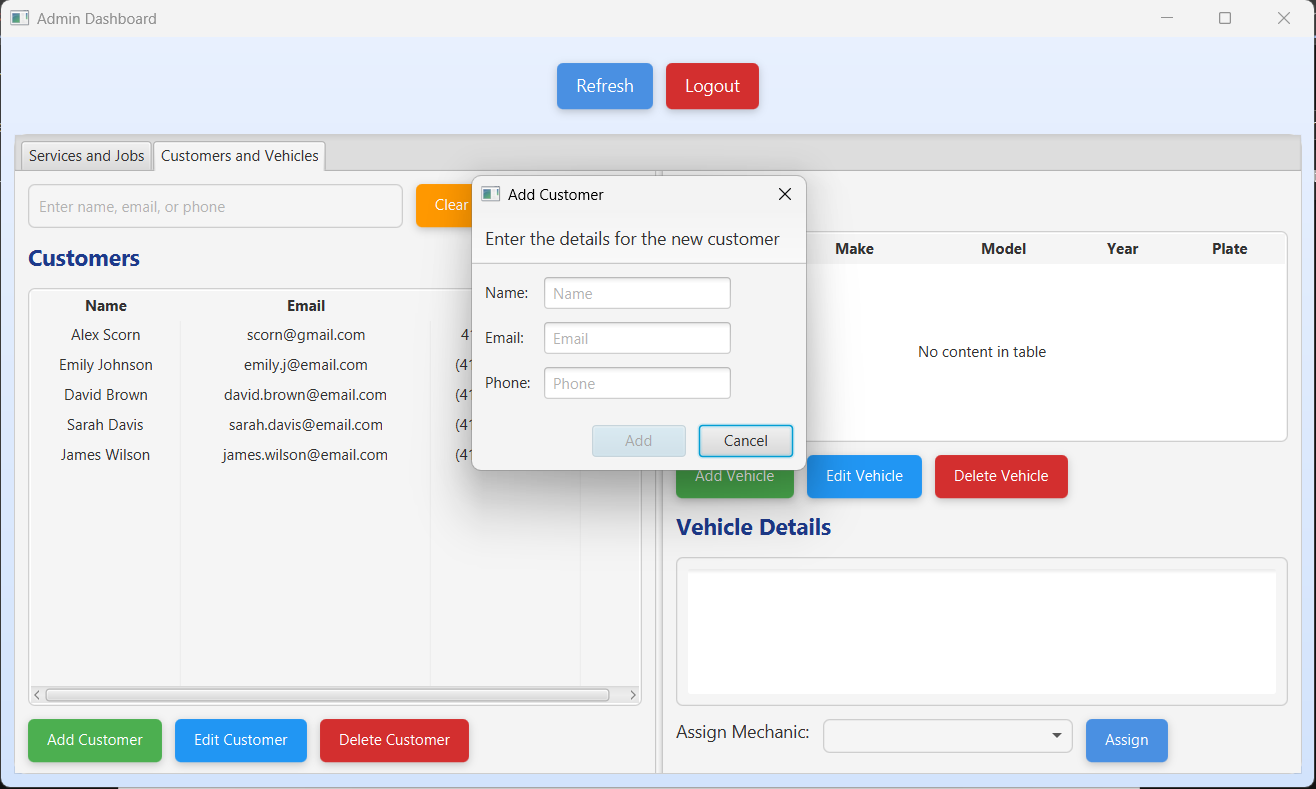
Step 1: Login as Administrator

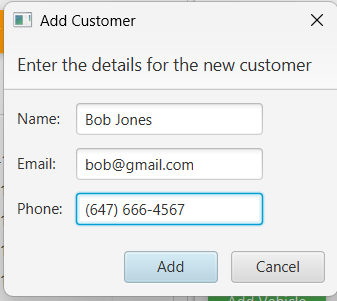


Step 2: Navigate to “Customers and Vehicles” Tab



Step 3: Fill in pop-up dialog box





Step 5: Customer has been successfully added!

