CENTRAL MICHIGAN UNIVERSITY

BIS 698 Information System Project

Vehicle Service & Inventory Management – Final Project Complexity Documentation

GROUP 6

Richitha Ananthoju

Tharun Dasari

Sindhu Kothuri

Sai Vineeth Neeli

Rohan Chandran Ravi

Submission Date: 4/29/2025

FINAL PROJECT COMPLEXITY

Introduction

Our project represents a strong and professional information system, developed through detailed planning, advanced database design, and structured system documentation. While basic concepts were introduced during the course, our team expanded the project's complexity by applying additional tools, designing richer functionalities, and integrating advanced features independently. This report presents the functions, features, and capabilities that demonstrate the technical depth and professional quality of our system.

Key Features and Complexities:

Role-Based Dashboards with Authentication Routing

A unified login system needed to differentiate between Admins, Customers, and Mechanics and redirect each user to the correct dashboard without manual selection.

Complexity:

• Developed centralized authentication logic that determines the role of the user and automatically routes to the appropriate dashboard.
• Created completely separate UI workflows for each role with unique access rights and restricted functionality.
• Implemented dynamic session tracking to maintain role-based state during user navigation.
• Prevented unauthorized access by validating role at every dashboard and form interaction level.

Multi-Table Transactional Updates

Completing or booking a service had to update three or more interrelated tables while maintaining referential integrity.

Complexity:

• Designed service workflows to update SERVICE_REQUEST, SERVICE_SELECTION, and PARTS_USED simultaneously using transaction-safe logic.
• Managed foreign key constraints to avoid data inconsistencies and orphan records.
• Implemented rollback strategies in case of partial update failures during service completion.
• Enabled coordinated UI inputs that trigger cascading updates to all affected tables.

Real-Time Service Status Tracking

Customers needed to see live updates of service request statuses as mechanics worked on them.

Complexity:

• Mechanic dashboards allow real-time updates to statuses via dropdowns (e.g., Pending → In Progress → Completed).
• Customer dashboards reflect status changes instantly by querying updated values on each login/session.
• Role-specific dashboards render only allowed status actions based on user type.
• Real-time feedback loop simulated using event-driven database reads to keep all roles in sync.

Mechanic Assignment and Feedback System

Admins must assign mechanics to jobs, and customers must provide and edit feedback interactively.

Complexity:

• Admin panel displays pending jobs and dynamically populates a dropdown with available mechanics from the database.

• Insert and update operations for feedback are conditionally handled to allow editing or new submission based on prior records.

• Created a star-rating interface using Tkinter widgets to capture visual feedback from users.

• Maintained data consistency by validating mechanic availability and service completion before feedback is accepted.

Data Import & Export Capabilities

Required the ability to populate and export data efficiently for setup, testing, and auditing.

Complexity:

• Designed scripts that insert large datasets into all primary tables in sequence (with foreign key integrity).

• Supported future export scenarios using SELECT INTO OUTFILE logic or Pandas-based data dumps.

• Used parameterized scripts to reuse insertion templates with dynamic data rows.

• Ensured no duplicate or inconsistent records were created during large-volume insertions.

External Library Integration

Native libraries were insufficient for GUI interactivity and advanced database functionality.

Complexity:

• Integrated MySQL Connector for seamless database transactions from Python.

• Used Tkcalendar to implement a calendar widget for date selection in service scheduling.

• Managed installation, import, and version compatibility of external Python packages.

• Built fallback mechanisms to handle library-specific errors during runtime.