SOFTWARE REQUIREMENTS SPECIFICATION

for

MiLog

Version 1.0

Prepared by
Shafayat Alam Fahim (2211645)
Computer Science and Engineering
School of Engineering, Technology and Science
Independent University, Bangladesh

Submitted to : Shanjita Akter Prome
Adjunct Faculty
Computer Science and Engineering
School of Engineering, Technology and Science
Independent University, Bangladesh

Contents

1 Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a detailed and comprehensive description of all requirements for the **Milog - Fuel Mileage and Expenses Tracker** web application. This document formally outlines the system's scope, primary functions, and constraints, and its structure is based on the IEEE Recommended Practice for Software Requirements Specifications [1]. It specifies both the functional requirements, detailing what the system must do, and the non-functional requirements, defining the system's operational qualities and standards. This SRS serves as the foundational agreement between the project developer and the course evaluator, ensuring a clear and mutual understanding of the project's objectives. It will act as the primary guiding reference throughout the design, development, implementation, and final testing phases of the software lifecycle.

1.2 Intended Audience and Reading Suggestions

This document is intended for:

- **Developers** – This document serves as the primary technical guide for the developer. It contains all the necessary specifications to implement, test, and maintain the application.
- **Course Evaluators** – The SRS provides a definitive baseline against which the final application will be assessed. It allows the evaluator to verify that all academic objectives and project requirements have been successfully met.
- **End Users** – Potential users of the application can read this document to gain a thorough understanding of the system's features, capabilities, and limitations before using it.

Readers are encouraged to focus on:

- For a general overview of the project's scope, purpose, and context, readers should focus on Chapter 1 (Introduction) and Chapter 2 (Overall Description).
- For a detailed, technical breakdown of every feature and function the system must perform, developers and evaluators should closely review Chapter 3 (System Features).

- For an understanding of the system's quality attributes, such as performance, security, and usability, all readers should refer to Chapter 4 (Non-Functional Requirements).

1.3 Project Scope

The scope of this project is the design, development, and implementation of the **MiLog - Fuel Mileage and Expenses Tracker**, a self-contained, multi-user web application. The system provides vehicle owners with a comprehensive platform to digitally log, manage, and analyze their fuel consumption and vehicle efficiency.

The primary objective is to deliver a secure, user-friendly application where registered users can manage profiles for multiple vehicles. For each vehicle, users can record detailed fuel entries, including odometer readings, fuel price, and volume. The system leverages this data to automatically calculate and display key performance metrics, such as lifetime average mileage and total fuel expenditure, on both a user-level and vehicle-specific dashboard.

The scope of the project includes the following key functions and deliverables:

- **Secure User Authentication:** A complete user registration and login system with session management and secure password hashing.
- **Multi-Vehicle Management:** Functionality for users to add multiple vehicle profiles to their account and delete vehicles they no longer own.
- **Detailed Fuel Logging:** A system for creating and storing fuel log entries for each vehicle.
- **Log Correction Feature:** The ability for users to edit or delete their most recent fuel log entry to correct any mistakes.
- **Dynamic User Dashboard:** A central dashboard for users that displays a performance summary for each vehicle and a live feed of recent fuel-logging activities.
- **Vehicle Analytics Dashboard:** A dedicated page for each vehicle that shows detailed statistics, including total fuel costs, lifetime average mileage, and a complete history of all its fuel logs.
- **Administrative Panel:** A secure, role-based admin panel accessible only to a designated administrator. The sole function of this panel is to provide user support by allowing the administrator to reset any user's password.

The application is developed using PHP for all server-side logic, MySQL for relational data persistence, and HTML/CSS for a responsive user interface. The entire system is designed to operate within a Docker environment for consistent deployment and operation. Features outside of this defined scope, such as real-time fuel price tracking or advanced graphical data visualization, are not included in this version of the project.

2 Overall Description

2.1 Product Perspective

The **MiLog - Fuel Mileage and Expenses Tracker** is a new, self-contained, and standalone web application. It is designed to be an independent system that does not require interaction with any other software to perform its core functions.

From a user's perspective, this product serves as a modern, digital replacement for manual tracking methods such as physical logbooks or personal spreadsheets. It is not an extension of a larger system nor does it replace a pre-existing automated system.

The application operates on a classic client-server model. The user interacts with the system through a graphical user interface (GUI) rendered in a standard web browser. The server-side application logic, written in PHP, interfaces directly and exclusively with a MySQL relational database for all data persistence, including user accounts, vehicle profiles, and fuel log entries.

2.2 User Classes and Characteristics

The **MiLog - Fuel Mileage and Expenses Tracker** system is designed to be used by two distinct classes of users, each with a specific set of characteristics, tasks, and system privileges.

1. **Regular User** - This class represents the primary and most common user of the application.
    - **Characteristics:**
        - A vehicle owner who wishes to track fuel consumption and expenses digitally.
        - Possesses basic computer and web browsing skills.
        - Does not require any specialized technical training to operate the application.
        - Motivated by the desire to monitor vehicle performance and manage personal finances related to fuel costs.
    - **Privileges:**
        - Can create a personal, password-protected account.

- Can log in and out of the system securely.
- Can add, view, and delete their own vehicle profiles.
- Can create, view, and manage fuel log entries for their own vehicles.
- Can edit or delete the most recent fuel log entry to correct errors.
- Can view automatically calculated statistics and summaries on their personal and vehicle-specific dashboards.
- Crucially, a regular user has no access to any data—including accounts, vehicles, or logs—belonging to any other user.

2. **Administrator** - This class represents a single, privileged user responsible for basic system management and user support.
   - **Characteristics:**
     - A technically proficient individual, typically the system developer or a designated manager.
     - Possesses a complete understanding of the system's functionality.
     - Responsible for maintaining the integrity of user accounts and providing support.
   - **Privileges:**
     - Logs in through the same standard login page as a regular user.
     - Has access to a separate, secure `"Admin Panel"` that is not visible to regular users.
     - Can view a list of all registered users in the system.
     - Has the ability to set a new password for any regular user's account.
     - The administrator cannot view, create, edit, or delete any user's vehicle or fuel log data. Their privileges are strictly limited to user account support to maintain user privacy.

2.3 Product Functions

The **MiLog - Fuel Mileage and Expenses Tracker** provides a comprehensive suite of functions designed to meet the needs of both regular users and administrators. The core functions of the system are summarized below.

- **User Account Management:**
  - **Registration:** Allows new users to create a secure personal account using a nickname, a unique email address, and a password.
  - **Authentication:** Provides a secure login mechanism to authenticate users and grant access to their personal data. It also includes a secure logout function to terminate the user session.

  - **Session Persistence:** Manages user sessions to keep users logged in as they navigate between different pages of the application.

- **Vehicle Profile Management:**
  - **Add Vehicle:** Enables authenticated users to add one or more vehicle profiles to their account, capturing details such as vehicle name, model year, and tank capacity.
  - **View Vehicles:** Displays a consolidated list of all vehicles registered to a user's account.
  - **Delete Vehicle:** Allows users to permanently remove a vehicle and all of its associated data from their account.

- **Fuel Log Management:**
  - **Create Log Entry:** Provides a form for users to log new fuel entries for a specific vehicle, including details like odometer reading, date, fuel price, and either the total fuel volume or the total cost.
  - **Automatic Calculation:** Intelligently calculates the total fuel cost if the user provides the price and volume, or calculates the volume if the user provides the price and total cost.
  - **View Log History:** Displays a complete, chronologically ordered history of all fuel logs for a selected vehicle.
  - **Edit and Delete Last Entry:** Allows users to edit or delete the most recent fuel log to correct any data entry errors.

- **Dashboard and Analytics:**
  - **Main User Dashboard:** Presents a high-level summary of key statistics (Total Spent, Average Mileage, Last Refuel Date) for each of the user's vehicles. It also includes a dynamic `"Recent Activity"` feed showing the latest fuel entries.
  - **Vehicle-Specific Dashboard:** Offers a detailed analytics view for each individual vehicle, including lifetime performance metrics and a full, sortable log history.
  - **PDF Report Generation:** The system can generate and download a detailed PDF report containing the complete fuel log history for any specific vehicle.

- **Administrator Functions:**
  - **Secure Admin Panel:** Provides a role-based, secure administrative panel accessible only to a designated administrator.
  - **User Password Reset:** The sole function of the admin panel is to allow the administrator to view a list of all users and reset their passwords to provide

account recovery support.

2.4 Operating Environment

The Fuel Mileage Tracker is designed as a web application to be deployed within a containerized, server-based environment. The specific hardware and software components required for the system to operate successfully are detailed below.

- **Server-Side Environment:**
  - **Operating System:** The server can operate on any OS that supports Docker, such as Linux (e.g., Ubuntu, CentOS), Windows, or macOS.
  - **Web Server:** A standard web server such as Apache or Nginx is required to handle HTTP requests.
  - **PHP:** The application logic is built using PHP. It requires PHP version 7.4 or newer.
  - **Containerization:** The entire backend stack (PHP, Web Server, Database) is designed to run within a Docker environment, ensuring consistency and portability across different host machines.
- **Database Environment:**
  - **Database Management System:** The application requires a MySQL relational database server, version 5.7 or higher, to store all user, vehicle, and fuel log data.
- **Client-Side Environment:**
  - **Platform**: The application is platform-independent on the client side. It can be accessed from any device with a modern web browser and an internet connection, including desktops, laptops, tablets, and smartphones.
  - **Web Browser:** A modern web browser that supports HTML5, CSS3, and JavaScript is required. Recommended browsers include:
    - Google Chrome (latest version)
    - Mozilla Firefox (latest version)
    - Microsoft Edge (latest version)
    - Apple Safari (latest version)
    - Opera (latest version)

2.5 Design

The development of the **MiLog - Fuel Mileage and Expenses Tracker** was governed by a set of specific constraints to ensure security, maintainability, and adherence to modern web standards.

- **Core Technology Stack:** The application is constrained to a specific technology stack. All server-side logic must be implemented using PHP, and all data persistence must be handled by a MySQL relational database.
- **Password Security:** The system is prohibited from storing user passwords in plaintext. All passwords must be securely hashed using the **password_hash()** function with the **PASSWORD_DEFAULT** algorithm, which ensures a strong, industry-standard level of security [2].
- **SQL Injection Prevention:** To mitigate the risk of SQL injection attacks, all database queries that incorporate user-provided data must be executed using prepared statements with parameterized queries. Direct insertion of user input into SQL strings is strictly forbidden.
- **Responsive User Interface:** The front-end design is constrained to be fully responsive. The layout and user interface elements must adapt gracefully to provide an optimal viewing and interaction experience on various devices, including desktops, tablets, and smartphones.
- **Containerized Deployment:** The entire application stack, including the web server, PHP runtime, and MySQL database, must be configured to run within a Docker environment. This ensures a consistent and reproducible setup for both development and deployment.
- **Data Integrity:** The database schema design must enforce relational integrity through the use of foreign key constraints. Specifically, the **ON DELETE CASCADE** constraint is required between users and their vehicles, and between vehicles and their fuel logs, to ensure that deleting a parent record automatically removes all associated child records.

3 System Features

This chapter provides a detailed breakdown of the functional requirements for the **MiLog - Fuel Mileage and Expenses Tracker** system. Each system feature is described in detail to provide a clear and unambiguous understanding of its purpose and behavior.

The features are organized into logical groups based on their functionality to ensure clarity and readability. Each feature is assigned a unique identifier (e.g., SF-1.1) for easy reference and traceability throughout the project lifecycle. For each feature, a description is provided, followed by a list of specific functional requirements detailing the inputs, processing logic, and expected outputs.

The primary functional modules of the system are:

3.1 User Account Management

1. **User Registration**
   - **Feature ID:** SF-1.1
   - **Description:** This feature allows a new user to create a personal account in the system. The user must provide a nickname, a unique email address, and a password.
   - **Functional Requirements:**
     a) The system shall provide a web form with fields for Nickname, Email Address, and Password.
     b) Input: The user must enter a non-empty nickname, a valid and unique email address, and a password.
     c) Processing:
        – The system shall validate that all fields are filled.
        – The system shall check if the provided email address already exists in the database.
        – The system shall securely hash the user's password using the PASSWORD_DEFAULT algorithm.
        – Upon successful validation, the system shall create a new record in the users table with the provided details.
     d) Output:

- If registration is successful, the system shall display a success message and direct the user to the login page.
- If the email already exists, the system shall display an error message and will not create a new account.

2. **User Login**
   - **Feature ID:** SF-1.2
   - **Description:** This feature allows a registered user to securely access their account.
   - **Functional Requirements:**
     a) The system shall provide a web form with fields for Email Address and Password.
     b) Input: The user must enter their registered email and password.
     c) Processing:
        - The system shall retrieve the user record from the database that matches the provided email.
        - The system shall use the **password_verify()** function to compare the entered password against the stored hash.
        - Upon successful verification, the system shall create a new session and store the user's unique ID (**user_id**).
        - The system shall check the is_admin flag for the user.
     d) Output:
        - If authentication is successful and the user is a regular user, the system shall redirect them to the main user dashboard (**dashboard.php**).
        - If authentication is successful and the user is an administrator, the system shall redirect them to the admin dashboard (**admin_dashboard.php**).
        - If authentication fails, the system shall display an `"Invalid email or password"` error message.

3. **User Logout**
   - **Feature ID:** SF-1.3
   - **Description:** This feature allows a logged-in user to securely terminate their session.
   - **Functional Requirements:**
     a) Input: The user clicks the `"Sign Out"` link in the navigation bar.
     b) Processing: The system shall destroy the current session, removing all session variables.

c) Output: The system shall redirect the user to the login page (**login.php**).

3.2 Vehicle Management

This section details the features related to managing vehicle profiles.

1. **Add New Vehicle**
   - **Feature ID:** SF-2.1
   - **Description:** This feature allows a logged-in user to add a new vehicle profile to their account.
   - **Functional Requirements:**
     a) The system shall provide a form with fields for Vehicle Name, Model Year, and Tank Capacity.
     b) Input: The user enters the required vehicle details.
     c) Processing:
        – The system shall validate that all inputs are valid.
        – The system shall create a new record in the vehicles table, associating it with the current user's user_id.
     d) Output: Upon successful creation, the system shall redirect the user back to the `"My Vehicles"` page (**vehicles.php**).
2. **View All Vehicles**
   - **Feature ID:** SF-2.2
   - **Description:** This feature allows a user to view a list of all vehicles associated with their account.
   - **Functional Requirements:**
     a) Input: The user navigates to the `"My Vehicles"` page.
     b) Processing: The system shall query the vehicles table for all records matching the current user's **user_id**.
     c) Output: The system shall display the list of vehicles in a card-based layout, showing each vehicle's name, model year, and tank capacity. Each card will also contain links to `"View Logs" and "Delete"`.
3. **Delete Vehicle**
   - **Feature ID:** SF-2.3

- **Description:** This feature allows a user to permanently delete a vehicle and all its associated data.
- **Functional Requirements:**
  a) Input: The user clicks the `"Delete"` button on a specific vehicle card.
  b) Processing:
     – The system shall display a custom confirmation modal to prevent accidental deletion.
     – Upon user confirmation, the system shall execute a DELETE query on the vehicles table for the specified vehicle ID.
     – The database's **ON DELETE CASCADE** constraint will automatically delete all associated fuel logs for that vehicle.
  c) Output: The system shall redirect the user back to the updated `"My Vehicles"` page.

4. **Generate PDF Report**
   - **Feature ID:** SF-2.4
   - **Description:** This feature allows a user to download a complete fuel log history for a specific vehicle as a professionally formatted PDF document. To achieve this, the system utilizes the open-source FPDF library [3].
   - **Functional Requirements:**
     a) Input: The user clicks the `"Download PDF"` button associated with a specific vehicle on the `"My Vehicles"` page.
     b) Processing:
        – The system shall perform a security check to ensure the requested vehicle belongs to the logged-in user.
        – The system shall query the database and retrieve all fuel logs associated with the specified **vehicle_id**, ordered by date.
        – The system shall dynamically generate a PDF document, including a header with the vehicle's name and model year, and a formatted table of all the fuel logs.
     c) Output: The system shall trigger a browser download of the generated PDF file, named appropriately (e.g., **fuel_logs_honda_civic.pdf**).

3.3 Fuel Log Management

This section details the features related to managing fuel log entries.

1. **Create Fuel Log Entry**
   - **Feature ID:** SF-3.1
   - **Description:** Allows a user to add a new fuel log for one of their vehicles.
   - **Functional Requirements:**
     a) The system shall provide a form with fields for Odometer, Date, Price per Liter, Fuel (Liters), and Total Cost.
     b) Input: The user enters the required log details.
     c) Processing:
        – The system shall perform an automatic calculation: if the user provides Price and Liters, it calculates the Total Cost; if the user provides Price and Total Cost, it calculates the Liters.
        – The system shall validate all inputs.
        – The system shall create a new record in the fuel_logs table, associating it with the correct vehicle_id.
     d) Output: Upon successful creation, the system shall redirect the user back to the vehicle-specific dashboard.
2. **View Fuel Log History**
   - **Feature ID:** SF-3.2
   - **Description:** Displays a complete history of all fuel logs for a specific vehicle.
   - **Functional Requirements:**
     a) Input: The user navigates to a vehicle's specific dashboard.
     b) Processing: The system shall query the fuel_logs table for all records associated with the selected vehicle_id.
     c) Output: The system shall display the logs in a table format, showing the Odometer, Liters, Price/L, and Total Cost for each entry.
3. **Edit Last Fuel Log**
   - **Feature ID:** SF-3.3
   - **Description:** Allows a user to edit their most recent fuel log entry to correct mistakes.
   - **Functional Requirements:**

a) Input: The user clicks the `"Edit"` link next to the most recent log in the history table.

b) Processing: The system shall display a new page with a form pre-filled with the existing data for that log.

c) Output: After the user submits the changes, the system shall update the corresponding record in the fuel_logs table and redirect the user back to the vehicle dashboard.

4. **Delete Last Fuel Log**
   - **Feature ID:** SF-3.4
   - **Description:** Allows a user to permanently delete their most recent fuel log entry.
   - **Functional Requirements:**
     a) Input: The user clicks the `"Delete"` link next to the most recent log.
     b) Processing:
        – The system shall display a custom confirmation modal.
        – Upon confirmation, the system shall delete the corresponding record from the **fuel_logs** table.
     c) Output: The system shall redirect the user back to the updated vehicle dashboard.

3.4 Dashboard and Analytics

This section details the features related to data presentation and calculation.

1. **Main User Dashboard Display**
   - **Feature ID:** SF-4.1
   - **Description:** Provides the user with a high-level summary of all their vehicles and recent activity upon logging in.
   - **Functional Requirements:**
     a) Processing:
        – The system shall calculate the lifetime statistics (Total Spent, Avg. Mileage, Last Refuel) for each of the user's vehicles.

- The system shall fetch the 4 most recent fuel logs from the database across all of the user's vehicles.

b) Output: The dashboard shall display two main cards:

- A `"My Vehicles"` card listing each vehicle and its calculated lifetime stats.
- A `"Recent Activity"` card showing a feed of the latest fuel log entries.

2. **Vehicle-Specific Analytics Dashboard**
   - **Feature ID:** SF-4.2
   - **Description:** Provides a detailed analytical view for a single vehicle.
   - **Functional Requirements:**

     a) Processing:

     - The system shall calculate lifetime statistics (Total Fuel Expenses, Total Fuel Purchased, Average Mileage, Total Logs) for the selected vehicle.
     - The system shall fetch all fuel logs for the selected vehicle.

     b) Output: The page shall display:

     - A grid of `"stat cards"` showing the calculated analytics.
     - A form to add a new fuel log for that vehicle.
     - A complete, sortable table of the vehicle's entire fuel log history.

3.5 Administrator Panel

This section details the features exclusive to the administrator role.

1. **Admin Authentication**
   - **Feature ID:** SF-5.1
   - **Description:** Ensures that only designated administrators can access the admin panel.
   - **Functional Requirements:**

     a) Processing:

     - Upon any attempt to access an admin page (e.g., **admin_dashboard.php**), the system shall check if the user is logged in.

- The system shall then query the database to verify that the user's **is_admin** flag is set to TRUE.

b) Output:

- If the user is a valid admin, access is granted.
- If the user is not an admin, they are redirected to the regular user dashboard. If they are not logged in, they are redirected to the login page.

2. **User Password Reset**

- **Feature ID:** SF-5.2
- **Description:** Allows the administrator to reset the password for any regular user account.
- **Functional Requirements:**

a) Input: The administrator enters a new password for a user in the admin dashboard and clicks `"Reset"`.

b) Processing:

- The system shall securely hash the new password.
- The system shall update the **password_hash** in the users table for the specified user.

c) Output: The system shall redirect the administrator back to the admin dashboard with a success message.

4 Non-Functional Requirements

This chapter defines the non-functional requirements for the **MiLog - Fuel Mileage and Expenses Tracker** system. These requirements specify the quality attributes and constraints that the system must adhere to, focusing on how well the system performs its functions rather than the functions themselves.

4.1 Usability Requirements

Usability is a critical aspect of the application, ensuring that users can achieve their goals effectively, efficiently, and with satisfaction.

- **Ease of Learning:** The user interface shall be intuitive and self-explanatory. A new user with basic web literacy must be able to register, add a vehicle, and log their first fuel entry without requiring any external documentation or training.
- **Consistency:** The application shall maintain a consistent design and layout across all pages. Navigation menus, button styles, color schemes, and terminology shall be uniform throughout the user and admin interfaces to provide a predictable and comfortable user experience.
- **Efficiency of Use:** Data entry forms shall be designed for quick and efficient use. For example, the fuel log form provides automatic calculation, reducing the number of fields a user must manually complete.
- **Error Prevention:** The system shall guide users to prevent common errors. For instance, critical actions such as deleting a vehicle or a fuel log shall require user confirmation via a custom modal pop-up to prevent accidental data loss.
- **Feedback:** The system shall provide clear and immediate feedback for user actions. Successful operations, such as resetting a password in the admin panel, shall result in a visible success message. Invalid form submissions shall result in clear, user-friendly error messages.

4.2 Security Requirements

Security is paramount to protect user data and maintain the integrity of the system.

- **Authentication:** All user-specific data and functionality, excluding the login and registration pages, shall be accessible only to authenticated users. The system must enforce session management to prevent unauthorized access.

- **Password Management:** User passwords shall never be stored in plaintext. All passwords must be securely hashed using the **password_hash()** function with the PASSWORD_DEFAULT algorithm (BCrypt). The system must verify passwords using the **password_verify()** function.
- **Data Confidentiality:** A user must only be able to access and manipulate their own data. The system must enforce this by associating all vehicles and fuel logs with a specific user_id and including this ID in all database queries to filter results.
- **SQL Injection Prevention:** The application must be protected against SQL injection attacks. All database queries that involve user-supplied data must exclusively use prepared statements with parameter binding.
- **Role-Based Access Control (RBAC):** The system shall enforce a strict separation of privileges between regular users and administrators. Regular users shall have no access to the admin panel. The administrator's access to user data shall be limited strictly to password reset functionality to protect user privacy.

4.3 Performance Requirements

The application must be responsive and provide a smooth user experience.

- **Page Load Time:** All standard pages, including the user dashboard and vehicle dashboards, shall load completely within 3 seconds on a standard broadband internet connection.
- **Data Processing Time:** Server-side calculations for dashboard analytics (e.g., average mileage, total costs) shall be completed within 1 second after the page request is received.
- **Responsiveness:** The application must remain responsive under a normal load of concurrent users. Database queries shall be optimized to ensure they do not cause significant delays.

4.4 Reliability Requirements

The system must be stable and available for users when they need it.

- **Availability:** The application is expected to be available to users 24/7, with an uptime goal of 99.5 percent, allowing for occasional maintenance.
- **Data Integrity:** The database schema shall enforce data integrity using foreign key constraints. The **ON DELETE CASCADE** constraint must be implemented to ensure that when a user or vehicle is deleted, all of their dependent records (vehicles and fuel logs) are also cleanly and completely removed.

- **Error Handling:** The application shall handle unexpected errors gracefully. In the event of a database connection failure or a critical error, the system shall display a user-friendly error message rather than crashing or exposing raw error output.

4.5 Maintainability Requirements

The codebase and project structure must be organized to facilitate future updates and modifications.

- **Modularity:** The code shall be organized into logical, reusable components. For example, the user interface header (**layout_header.php**), authentication logic (**auth.php**), and database connection (**db.php**) are separated into their own files to avoid code duplication.
- **Readability:** The PHP and SQL code shall be well-formatted and commented where necessary to explain complex logic, ensuring that another developer can understand and modify the code in the future.
- **Portability:** The use of Docker for containerizing the application environment ensures that the project can be set up and run consistently on any machine that supports Docker, simplifying development and deployment.

5 System Models

This chapter provides visual models of the **MiLog - Fuel Mileage and Expenses Tracker** system to supplement the textual descriptions provided in the previous chapters. These diagrams illustrate the system's functionality, data structure, and data flow from different perspectives, offering a clearer understanding of the system's architecture and behavior.

5.1 Use Case Diagram

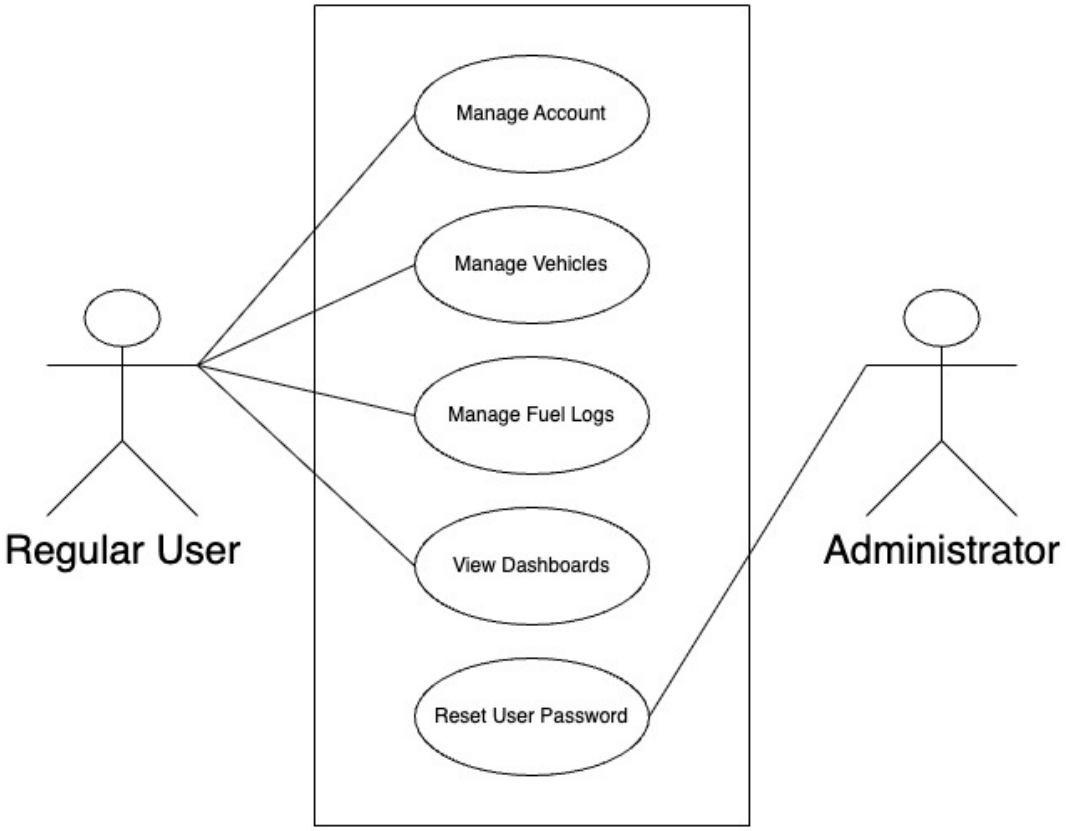

Figure 5.1: Use Case Diagram

The Use Case Diagram provides a high-level overview of the system's functionality and

illustrates the interactions between the primary actors (Regular User and Administrator) and the system itself. It visually summarizes the key features available to each user role.

- **Actors:**
  - **Regular User:** An individual who uses the application to manage their vehicles and track fuel expenses.
  - **Administrator:** A privileged user who inherits all capabilities of a Regular User and also has access to administrative functions.
- **Use Cases (System Functions):**
  - **Manage Account:** Includes registering, logging in, and logging out.
  - **Manage Vehicles:** Includes adding, viewing, and deleting vehicle profiles.
  - **Manage Fuel Logs:** Includes creating new fuel entries and viewing/editing/deleting the last log.
  - **View Dashboards:** Includes viewing the main user dashboard and the vehicle-specific analytics.
  - **Reset User Password:** A function exclusive to the Administrator.
- **Relationships:**
  - A Generalization relationship exists from the Administrator to the Regular User, indicating the Administrator can perform all actions of a Regular User.
  - `<<Include>>` dependencies show that Manage Vehicles, Manage Fuel Logs, and View Dashboards all require the user to be authenticated via Manage Account.

5.2 Entity-Relationship Diagram (ERD)

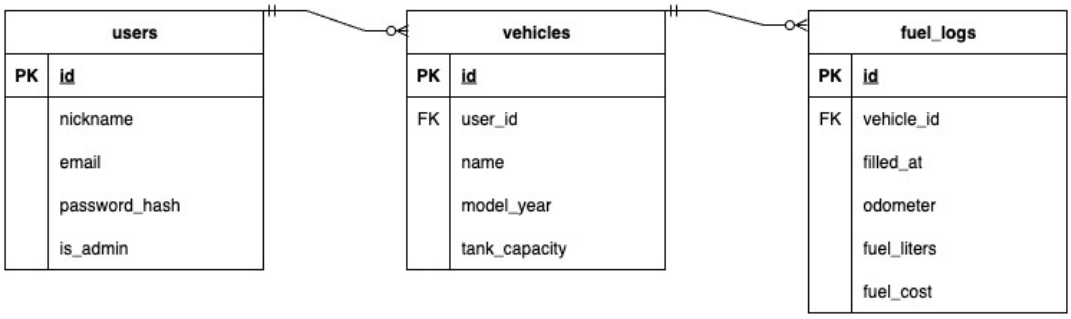

Figure 5.2: Entity Relation Diagram.

The Entity-Relationship Diagram (ERD) illustrates the logical structure of the system's database. It defines the main entities (tables), their attributes (columns), and the precise relationships between them.

- **Entities:**
  - **users:** Stores user account information.
  - **vehicles:** Stores vehicle profile information.
  - **fuel_logs:** Stores individual fuel entry data.
- **Attributes:**
  - **users table:** id (PK), nickname, email, password_hash, is_admin.
  - **vehicles table:** id (PK), user_id (FK), name, model_year, tank_capacity.
  - **fuel_logs table:** id (PK), vehicle_id (FK), filled_at, odometer, fuel_liters, fuel_cost.
- **Relationships:**
  1. users to vehicles: This is a **Mandatory One to Optional Many relationship**. A vehicle must belong to exactly one user, but a user can have zero, one, or many vehicles.
  2. vehicles to fuel_logs: This is a **Mandatory One to Optional Many relationshi**p. A fuel log must belong to exactly one vehicle, but a vehicle can have zero, one, or many fuel logs.

5.3 Data Flow Diagram (DFD)

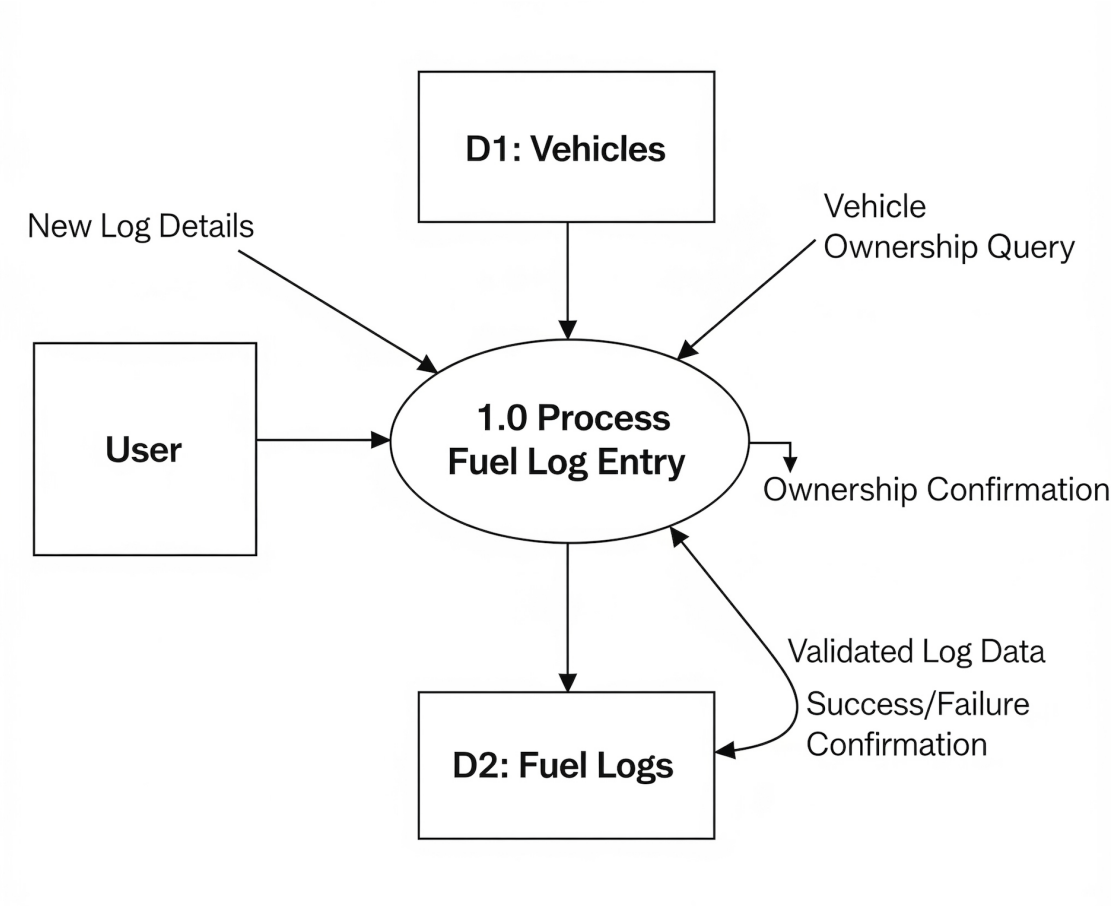

Figure 5.3: Data Flow Diagram

The **Data Flow Diagram (DFD)** illustrates how data moves through the system for the `"Add Fuel Log"` process, showing inputs from the user, internal processing, and final data storage.

- **External Entity:**
  - **User:** The person who initiates the process by submitting fuel log information.
- **Process:**
  - **1.0 Process Fuel Log Entry:** The central function that receives, validates, and stores the fuel log data.
- **Data Stores:**

- **D1: Vehicles:** Represents the **vehicles** database table.
- **D2: Fuel Logs:** Represents the **fuel_logs** database table.

- **Data Flows (The movement of data):**
  1. **New Log Details:** The user submits the fuel entry data to the process.
  2. **Vehicle Ownership Query:** The process reads from the Vehicles data store to confirm the vehicle belongs to the user.
  3. **Ownership Confirmation:** The Vehicles data store returns a confirmation to the process.
  4. **Validated Log Data:** The process writes the clean, final data to the Fuel Logs data store.
  5. **Success/Failure Confirmation:** The process sends a final status back to the user (e.g., a page redirect or an error message).

6 Conclusion

This chapter concludes the Software Requirements Specification for the **MiLog - Fuel Mileage and Expenses Tracker** system. It summarizes the project's objectives and scope and suggests potential avenues for future development.

6.1 Summary

This document has provided a comprehensive set of requirements for the **MiLog - Fuel Mileage and Expenses Tracker**, a web-based application designed to offer a modern, efficient solution for vehicle fuel management. The system provides a secure, multi-user environment where individuals can manage profiles for multiple vehicles and maintain detailed logs of their fuel consumption.// //

The core functionalities, including user authentication, vehicle and fuel log management (CRUD operations), and a dynamic dashboard with automatically calculated analytics, have been fully specified. Both the functional requirements that define the system's behavior and the non-functional requirements that dictate its quality attributes—such as usability, security, and performance—have been detailed. The system models, including the Use Case, Entity-Relationship, and Data Flow diagrams, provide a clear visual representation of the application's architecture.// //

In its current state, the project successfully meets all the defined objectives, delivering a complete and robust application that is ready for deployment.

6.2 Future Scope

While the current version of the **MiLog - Fuel Mileage and Expenses Tracker** is a complete product, several features could be implemented in the future to further enhance its value and user experience. Potential areas for future development include:

- **Graphical Data Visualization:** Implementing charts and graphs to visually represent mileage trends, monthly fuel costs, and fuel efficiency over time. This would make it easier for users to gain insights from their data at a glance.
- **Maintenance Logging:** Expanding the application's scope to include a module for logging vehicle maintenance activities, such as oil changes, tire rotations, and other services. This would transform the app into a more comprehensive vehicle management tool.

- **Data Export/Import:** Adding functionality to allow users to export their data to common formats like CSV or PDF for personal record-keeping, or to import data from existing spreadsheets.
- **Multi-Currency and Unit Support:** Introducing options for users to select their local currency and preferred units of measurement (e.g., gallons instead of liters, miles instead of kilometers).
- **Mobile Application:** Developing a dedicated mobile application (for iOS or Android) to provide a more convenient and streamlined experience for users on the go.

Bibliography

[1] Institute of Electrical and Electronics Engineers, "Ieee recommended practice for software requirements specifications," Standard 830-1998, IEEE, New York, NY, 1998.

[2] The PHP Group, "password_hash." `https://www.php.net/manual/en/function.password-hash.php`, 2025. Accessed: 2025-08-20.

[3] FPDF library, "Fpdf." `http://www.fpdf.org/`, 2024. Accessed: 2024-08-20.