Drive Smart Tracker

Software Smashers

Daisy Codenys, Sydney Nilles, Ty Gregory

CSci463 – Software Engineering

May 16, 2025 @ 12 noon

Index

Abstract

The Drive Smart Tracker is a remote motor vehicle application that can be used to view vehicle information and interact with selected vehicles. The main goal of this project was to create a visually appealing interface that simulates an actual motor vehicle remote starter application. Some of the features this application can simulate include: adding and deleting vehicles, adjusting for changing time zones, adding and deleting drivers, displaying vehicle information, and allowing for remote activation of the vehicle and its various functionalities. Throughout the project many diagrams, charts and drawings were created to facilitate the creation of the final draft of the application. In summary, the Software Smashers team have created an application that allows its users (Drivers and Owners) to remote control their vehicles.

1. Introduction

1.1 Description

The purpose of this project was to develop a remote vehicle starter application over the course of 4 months using an object-oriented model-driven methodology. This means that we studied and created numerous models and diagrams to design our system before and during software development. We were required to work as a team to ensure our application had the functions as specified in section 1.2. Our resulting application was an interface connected to a remote SQL database that will store and modify information as necessary.

1.2 Specifications

There were several functions that were requested to be included in the resulting application. These include:

- Visual feedback
- Services for an EV, hybrid, or gas vehicle:
    - Adding/deleting a driver
    - Adding/deleting a motor vehicle
    - Enabling/Disabling operation features
    - Adding/Deleting/Modifying user information
- System must have the following functionalities:
    - Stopping/Starting engine/motor
    - Locking/Unlocking doors
    - Opening/Closing windows
    - Displaying vehicle's internal and external temperatures
    - Displaying vehicle's battery strength or fuel level as applicable
    - Activating/Deactivating vehicle's alarm system
    - Displaying vehicle activities

1.3 Expected Deliverables

The resulting products from this project included a presentation covering the project, an accompanying demonstration, a functioning application-prototype capable of being opened and operated. As mentioned in section 1.1, numerous diagrams and models were also expected, as to illustrate the design process for the system along with assisting with development.

2. Background

2.1 Theoretical Topics

This project uses key software engineering principles similar to those applicable to safety critical systems and graphical user interface (GUI) development. Core software engineering principles such as modularity, abstraction, encapsulation, and separation of concerns help ensure that the

system is maintainable, scalable, and testable. For safety-critical systems, additional guidelines such as fault tolerance, formal verification, redundancy, and rigorous validation are emphasized to ensure reliability and minimize risks that could result in harm or failure.

In GUI design, principles like consistency, feedback, error prevention, and user control are crucial to creating interfaces that are both effective and user-friendly. These principles are guided by well-known and widely accepted design principles and aim to reduce cognitive load, improve navigation, and ensure accessibility for a broad range of users.

2.2 Models Used

In modeling the system effectively, various Unified Modeling Language (UML) diagrams have been utilized:

2.2.1 UML Class Diagram

Class Diagrams represent the static structure of the system, including classes, their attributes, methods, and relationships.

2.2.2 Use Case Diagram

Use Case Diagrams capture the system's functionality from a user's perspective. The diagram outlines the interactions between actors and the system.

2.2.3 Use Cases

Use Cases provide detailed scenarios that describe the way users interact with the system while achieving specific goals.

2.2.4 Activity Diagrams

Activity Diagrams describe the sequence of actions and decision-making involved in completing an activity. It illustrates the points in an activity that may require contingencies on the development side.

2.2.5 State Charts

State Charts show the various states an object can occupy and transitions triggered by events.

2.2.6 Sequence Diagrams

Sequence Diagrams depict the order of interactions between components or objects, emphasizing the time sequence of messages.

These models collectively work together to form a comprehensive understanding of system requirements, behaviors, and interactions.

2.3 Methodology Used

This project required the use of the “Object-Oriented Model-Driven Software Development” methodology. This was done by first creating models, like those mentioned above, to illustrate an outline with abstract imagery to represent the main components of our system, such as users, vehicles, and logs. This helped us visualize relationships that would inform necessary functions and internal interactions the system would require, allowing us to plan out features before implementing them.

The object-oriented portion was completed by separating out the major moving parts of the system into smaller modules that made the division of tasks amongst team members much easier, and will benefit long-term maintenance.

3. The Project

3.1 Methodology Applied

Over the course of development, each stage of progress resulted in a product such as a diagram or a version of the application. These products, usually the diagrams, would be presented to show our current status, and we would receive feedback. We would then go back, revise it accordingly, and submit it along with new diagrams to get further along into the process. This would repeat over the 4 months.

This methodology is called “Iterative Development”, and allowed for the correction of mistakes as we also continued development. We could continue at a good pace while also ensuring that the already established progress could keep up.

3.2 Models Developed

2.2.1 UML Class Diagram

We developed a class diagram that experienced multiple iterations, as visible in figures 6.2.1-3. It illustrates the various classes we planned to incorporate into our application. In the final version, it showed a User, Vehicle, Logs, Notification, and Settings. These would be implemented into a SQL database that would allow us to store and pull information as a driver needs. Though, in this diagram, it simply served as a foundation upon which we would build our application.

2.2.2 Use Case Diagram

The use case diagram (*Fig. 6.2.4*) allowed us to sort out the needs of the different drivers and what we would need to make sure was an accessible function in our application. We were then able to determine what functions were necessary.

2.2.3 Use Cases

After the class and use case diagrams, we created use case descriptions to outline the steps that would be taken in the event of, say, remotely starting or stopping a vehicle (*Fig. 6.2.5*) or viewing the information of a vehicle (*Fig. 6.2.6*). They show the steps taken to complete an activity outlined along with any conditions assumed to be present. Other actions to take should the main steps run into an issue are also included if necessary.

2.2.4 Activity Diagrams

Along with case descriptions, we created activity diagrams to show how certain vehicle features are carried out in sequence according to how a driver would experience them. For example, remotely starting or stopping a vehicle (Fig. 6.2.8) involves checking the vehicle's current state and either stopping it or attempting to start it, with retry logic if needed. Viewing vehicle information was also modeled in two versions (Figs. 6.2.9 and 6.2.10), first by displaying an unavailable message if data couldn't be retrieved, and later improved to better represent the actions a person would take while using our application.

2.2.5 State Charts

We developed state charts to show how the system responds based on different internal states and user inputs. In the vehicle registration chart (Fig. 6.2.12), the system either links an existing driver or creates a new one when a user adds a car. In the window control chart (Fig. 6.2.14), the system checks if the user has access to the car before allowing the windows to be opened or closed; otherwise, it shows an error. These diagrams help define how the system's behavior should change depending on access and input conditions.

2.2.6 Sequence Diagrams

For our sequence diagrams, our available examples are of starting a vehicle remotely (*Fig. 6.2.11*) and viewing its information (*Fig. 6.2.13*). They illustrate the user selecting their choice, which requests its information from the database that is then displayed. Then further into the process, the user will press the start/stop button. The system confirms that the vehicle started and an activity log is recorded in the database.

2.2.7 System Dictionary

In order to ensure our documentation is readable by non-programmers, we created a glossary of works and classes that are relevant to the applicable writing. It is visible in section 6.3 of the appendix.

3.3 User Interface

The user interface was made from scratch, rather than using a project from the last semester. We made this decision as we were all from different teams, and determined that we would prefer to have better control over the basis for our application. The interface was built to be easily read and visually simple to save on time and resources. The navigation would be done

through a menu on the left, as seen in one of our many interfaces figures in the appendix. Though, figure 6.4.2 serves as a fine example. Our interface would display vehicle information stored in the database (*Fig. 6.2.7*), and allow the driver to press buttons that communicate with the vehicle to perform functions such as starting the engine, opening windows, locking, setting the alarm, and the inverse. The buttons would change the icon above them to indicate the vehicle's status.

3.4 Issues and Resolutions

Throughout the project, our team faced many challenges including, but not limited to:

- Scheduling conflicts in the planning phase,
- Email miscommunication,
- Difficulty interpreting the project's requirements,
- Delays from too closely spaced report deadlines,
- And plans that proved to be too ambitious for our resources and time.

We addressed these issues by approaching the active issues during meetings, and discussing our course of action for reducing their impact on our workflow. These solutions for the issues mentioned above were:

- Setting a consistent meeting time that would work for each of us,
- Confirming email communication among members,
- Contacting the professor for clarification,
- The creation of a detailed tasks-to-complete schedule week by week,
- And the reductions of functions we planned to accomplish.

We had other, more technical, difficulties, like connecting the application to a database, and determining what the correct formatting was required by our API to send SQL queries. These were resolved through referencing the solutions of past projects along with trial and error testing.

Personal circumstances affected one team member's availability, so we adjusted our weekly tasks as necessary and stayed focused to accomplish what we needed to. Leadership responsibilities ended up being shared, with Daisy stepping in for Ty during periods of overload while still keeping communication focused through him.

3.5 Application Instructions

To open the project, you first need to either be on UND campus, or connected to UND's VPN.

Locate within our CSCI-FinalDeliverable-SoftwareSmashers folder a shortcut named "Drive Smart Tracker" with a star as the icon. Opening this shortcut will open the executable of our project.

When testing the application you may add a new account, add new vehicles or use one of our premade accounts from the presentation. The usernames and passwords are as follows:

- daisy@und.edu
    - codenys
- sydney@und.edu
    - yes
- ty@und.edu
    - imcool

3.6 Github Repository

For a detailed repository of all our code changes and version history our GitHub is linked here: https://github.com/tygregory26/Software-Smashers

4. Conclusion

4.1 What was Learned

One of the biggest lessons we learned from the project was the importance of breaking down large tasks into smaller, more manageable, weekly goals. This approach helped us to be more organized, and allowed us to better direct our focus during development. We also determined that it is important to communicate needs and expectations clearly with team members. Good communication plays a major role in keeping the project on track, and it also ensures that everyone is on the same page.

4.2 What Was and Was Not Useful

We found great use from figuring out how to connect Visual Studio to an SQL database, which allowed us to successfully integrate the backend and frontend components. The SQL database is a key functionality of our project, and without it, our project would not be able to display the same levels of function it does. Some early efforts to guess requirements without asking for clarification were not useful, which led to misunderstandings and extra work to correct them.

4.3 What Could be Improved

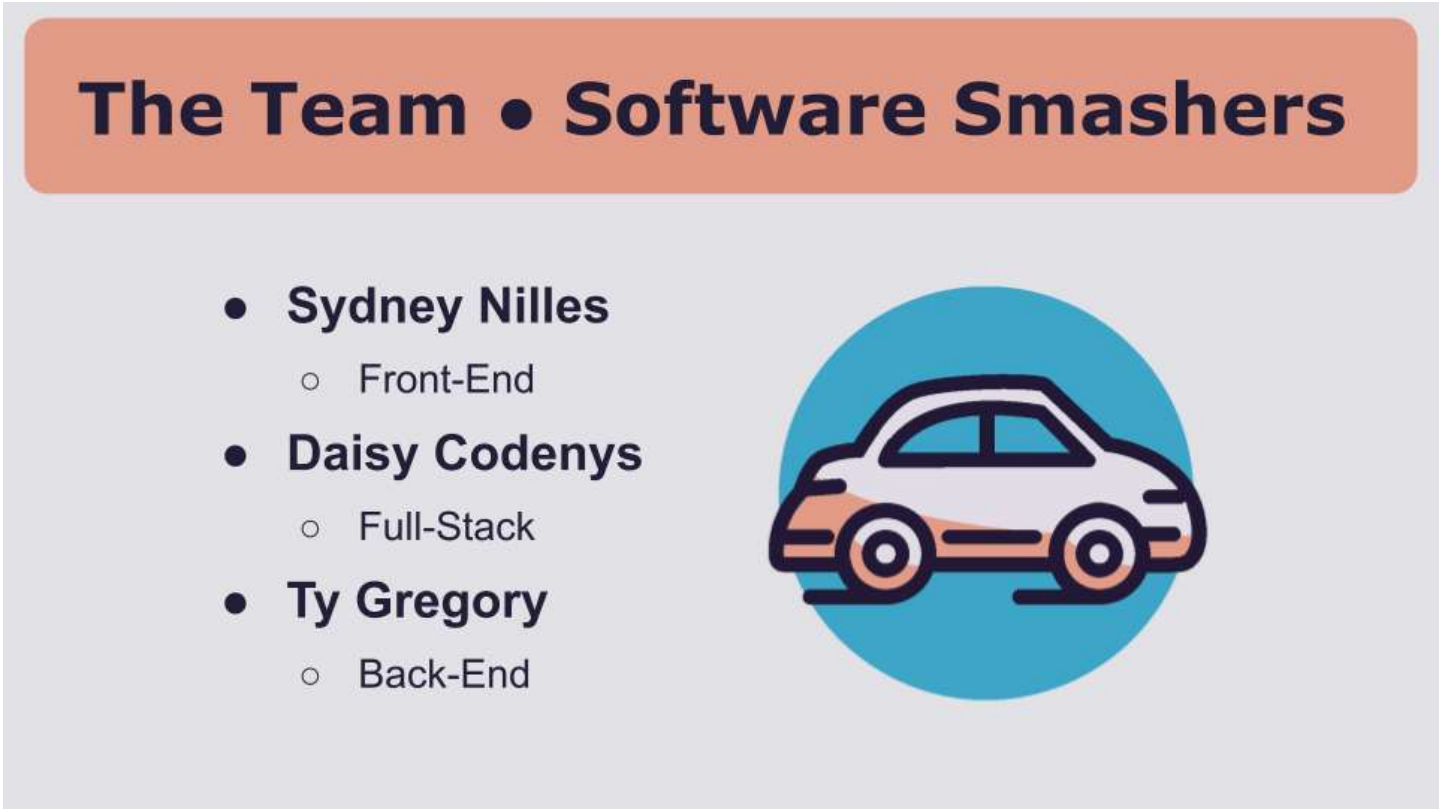
We acknowledge that there are many places in our application that could use improvement. In a scenario where we had access to the time and resources necessary, we would implement many of the functionality we had to cut out. These functions would be audio feedback, a dark visual theme for the UI, functional GPS tracking, better account organization and security, better accessibility features, and improved UI appearance overall. These, along with others that we might think of in the process of further development, would serve as the next step for our application to grow into something that could one day be used by the general public.

5. References

Software Engineering: Theory and Practices 4th ed., Shari L. Pfleeger Joanne M. Atlee, 2010.

6. Appendix

6.1 Presentation Slides



Drive Smart Tracker

Expected:

- A remote motor vehicle starter application capable of:
  - Viewing information
  - Interacting with the vehicle
  - And more…*

Resulting Products:

- Functional Application Prototype
- Project Code
- Project Report
- Application Demonstration

More items including:

- Visual feedback
- Services for an EV, hybrid, or gas vehicle:
  - Adding/deleting driver,
  - Adding/deleting motor vehicle,
  - Enabling/Disabling operation features,
  - Adding/Deleting/Modifying user information
- System must have the following functionalities
  - Stop/Start engine/motor,
  - Lock/Unlock doors,
  - Open/Close windows, including sunroof
  - Display vehicle's internal and external temperatures
  - Display vehicle's battery strength, or if not electric, fuel level.
  - Activate/Deactivate alarm system
  - Display application activities journal

Software Used

Canva

MySQL

Material UI

Discord

Google Drive

GitHub

Draw.io

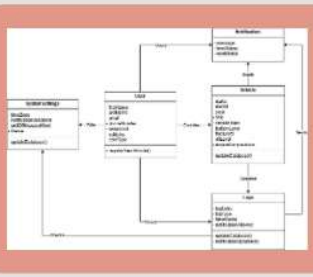
Microsoft Visual Studio

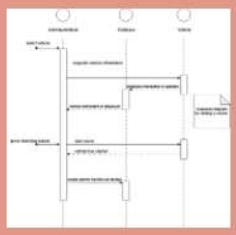
Medibang Paint Pro

Models & Diagrams

Many models and diagrams were used and modified over the course of development:

- UML Class Diagram
- Use Case Diagram
- Use Cases
- Activity Diagrams
- State Charts
- Sequence Diagrams





Examples (left to right):

- ➔ Class Diagram
- ➔ Activity Diagram
- ➔ Sequence Diagram

Methodology

Object-Oriented

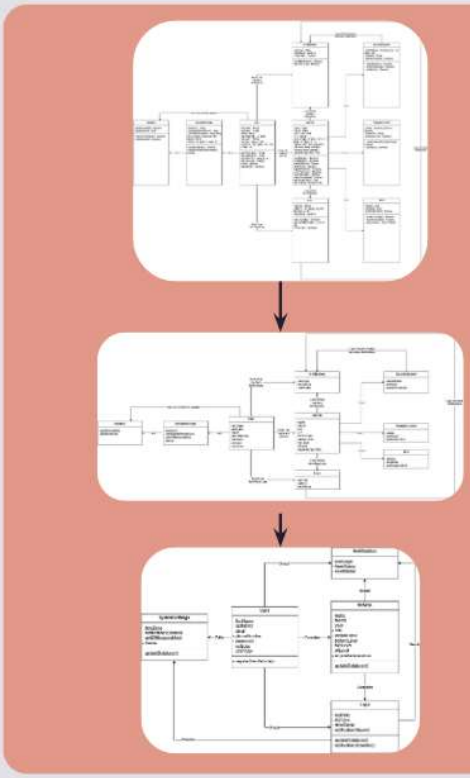
- Modular
- Reusable

Model-Driven

- Visual diagrams and models
- Organization of project parts and scope

Iterative Development (see right)

- Incremental updates
- Modification after feedback
- Multiple back and forth to get preferred outcome.



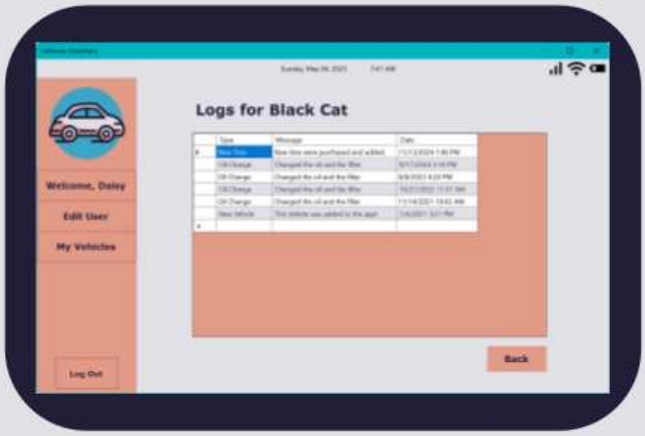
DST: What it Does

Owners

Drivers

DST: What it Does

See what the vehicle has been up to



Edit Vehicle Information

DST: Database Importance

- **Stores the users**
- **Stores the vehicles**
- **Stores logs about the vehicles**

DST: Database Queries

```
SELECT timeZone
FROM user
WHERE user = 1;
```

```
SELECT COUNT(*)
FROM drivers
WHERE driverLink = 1;
```

```
SELECT userID
FROM user
WHERE email = ty@und.edu;
```

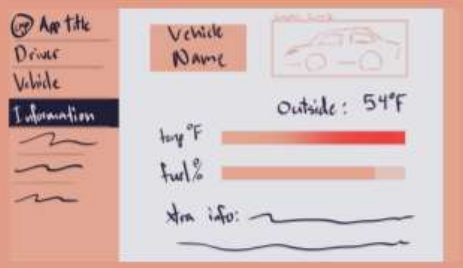
```
SELECT fuelLevel
FROM vehicle
WHERE carID = 1;
```

User Interface

We decided to have:

- A simple palette
- Large buttons for ease of reading
- Side menu panel for easy navigation
- Visual cues through changing icons
  - Shows when action is performed
- Customizable vehicle nicknames to determine them apart

Drawing 1



Refined Drawing

The Demo

Development Issues

- Reduction of planned functions
- UND Database server disconnections and unavailability
- Task management difficulties

If we had More Time

- Add availability to have multiple owners of a single car
- GPS and Vehicle Tracking
- Better Vehicle Selection UI

Reflection

What was Learned:

- Break down large tasks into smaller weekly goals
- Communicating needs with team members

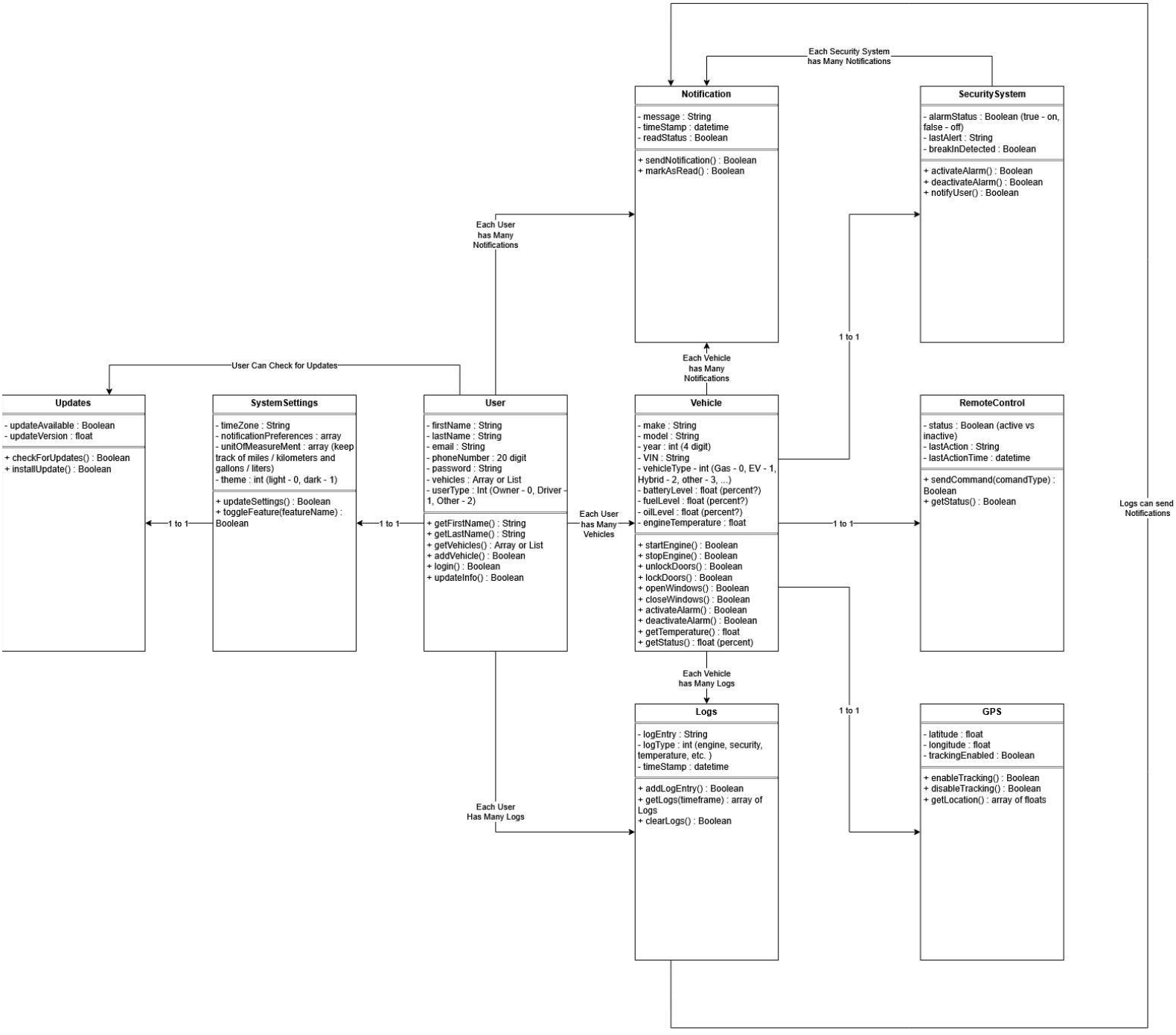
What was Useful:

- Determining how to connect Visual Studio to an SQL database

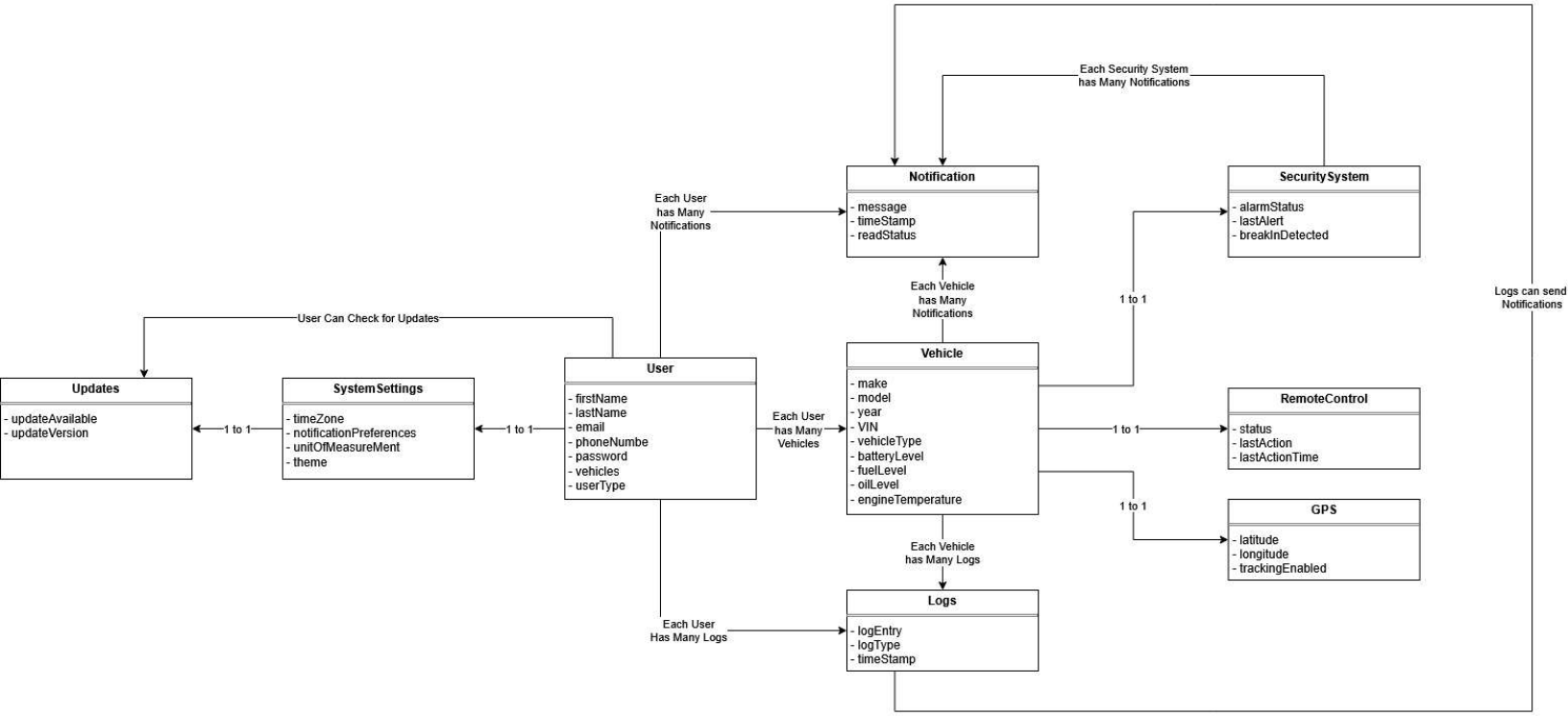
Improvements:

- Explicitness of requirements
- Distribution of tasks across time

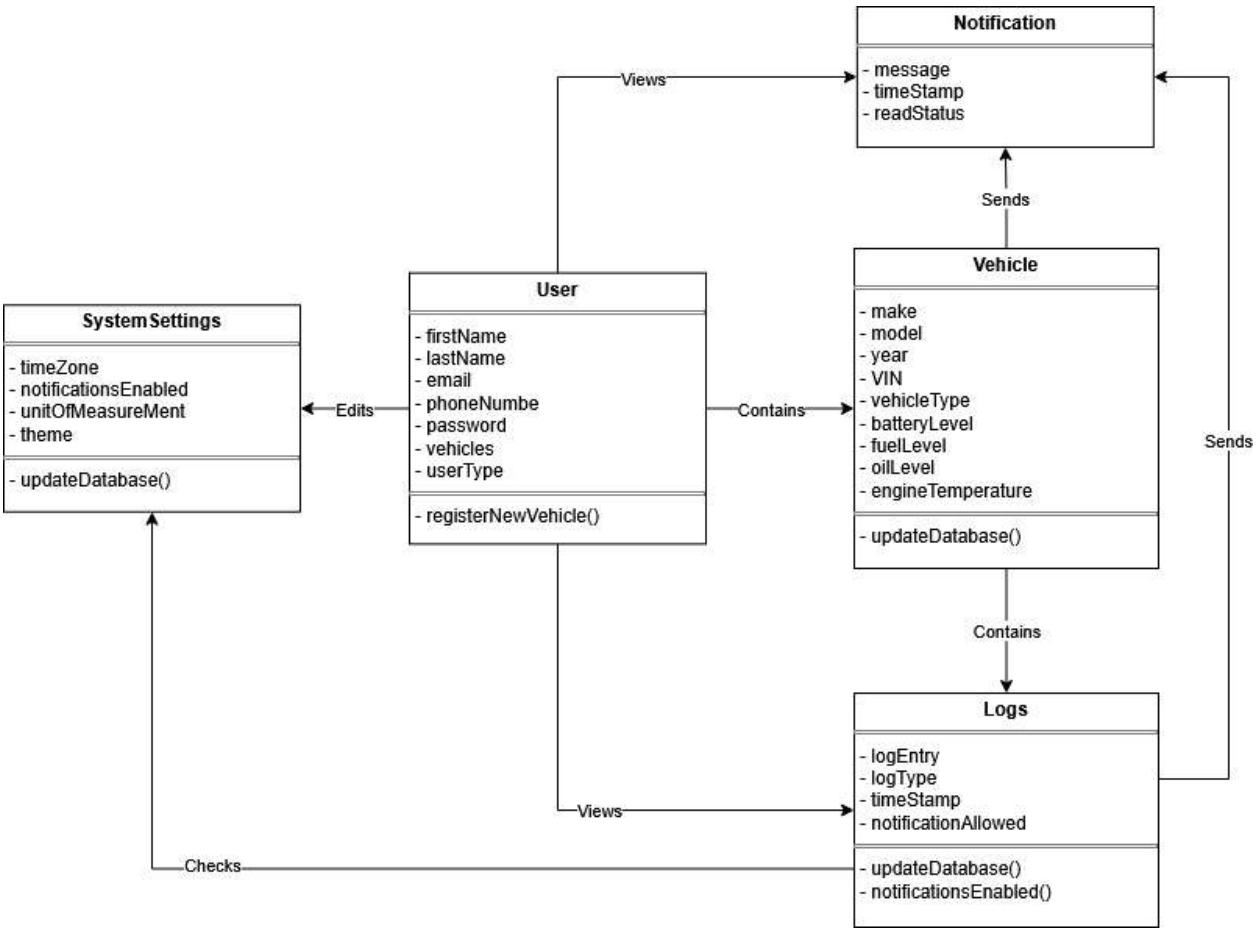
6.2. Models Developed



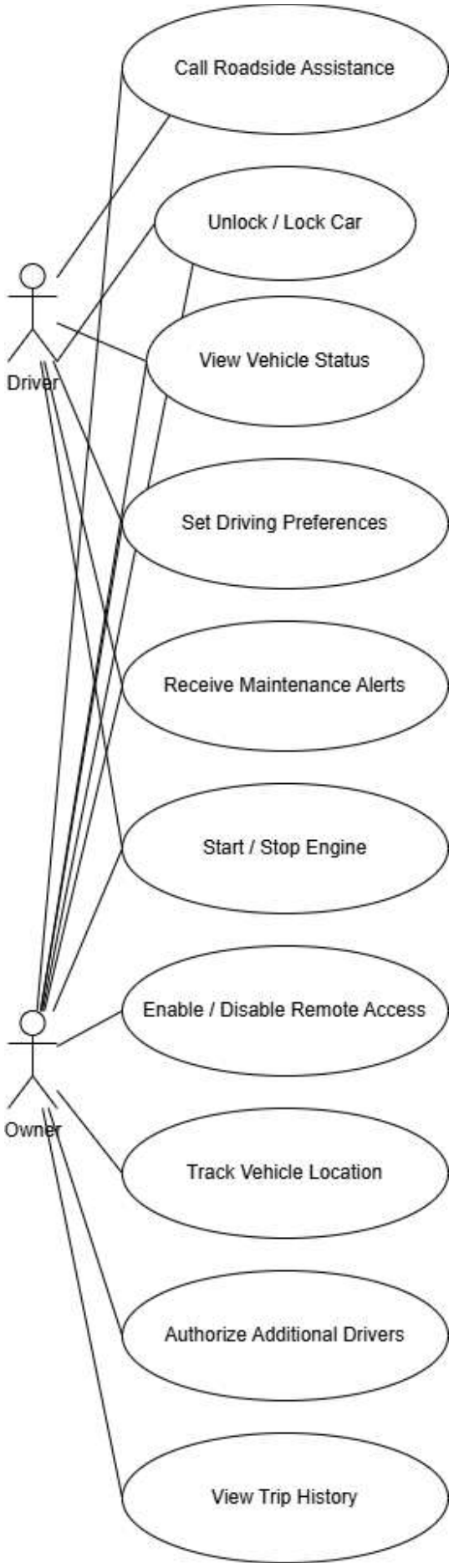
(Figure 6.2.1) UML Class Diagram First Draft



(Figure 6.2.2) UML Class Diagram Second Draft



(Figure 6.2.3) UML Class Diagram Final Version



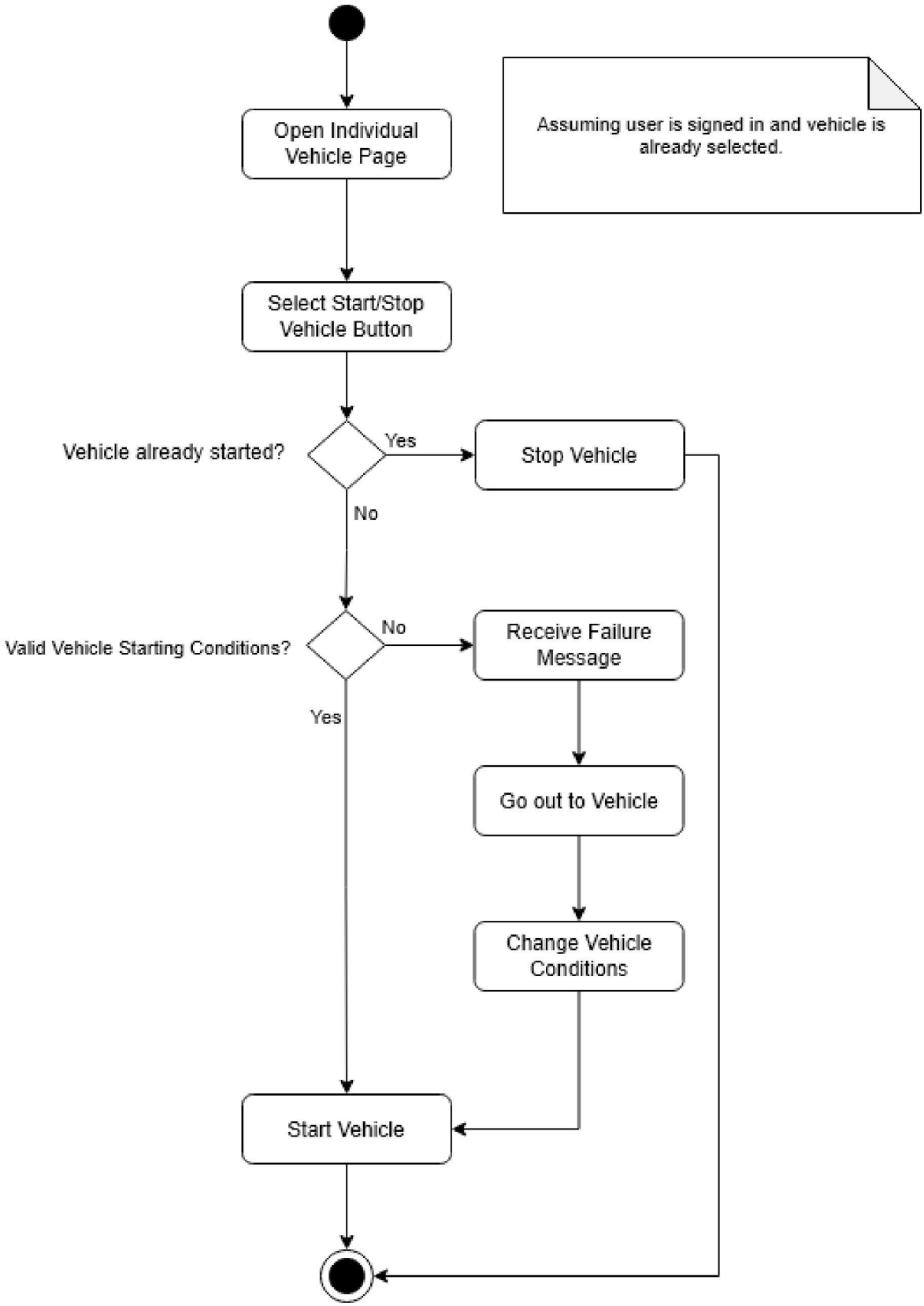
(Figure 6.2.4) Use Case Diagram

Use Case Name:	Remotely Start/Stop Vehicle	
Goal in context:	Allow the user to remotely start or stop the vehicle's engine using the app.	
Preconditions:	• The user is authenticated in the application. • The vehicle is in a state that allows remote start/stop. • The user has the necessary permissions.	
Description	1	User selects the "Start" or "Stop Engine" option in the app.
	2	The system verifies the user's access and sends the command to the vehicle.
	3	The vehicle starts or stops the engine and sends a confirmation notification.
Extensions	Step	Branching Action
	2a	If the vehicle is not in a valid state to start/stop, notify the user.
	3	If the command fails, display the message: "Failed to start/stop engine."

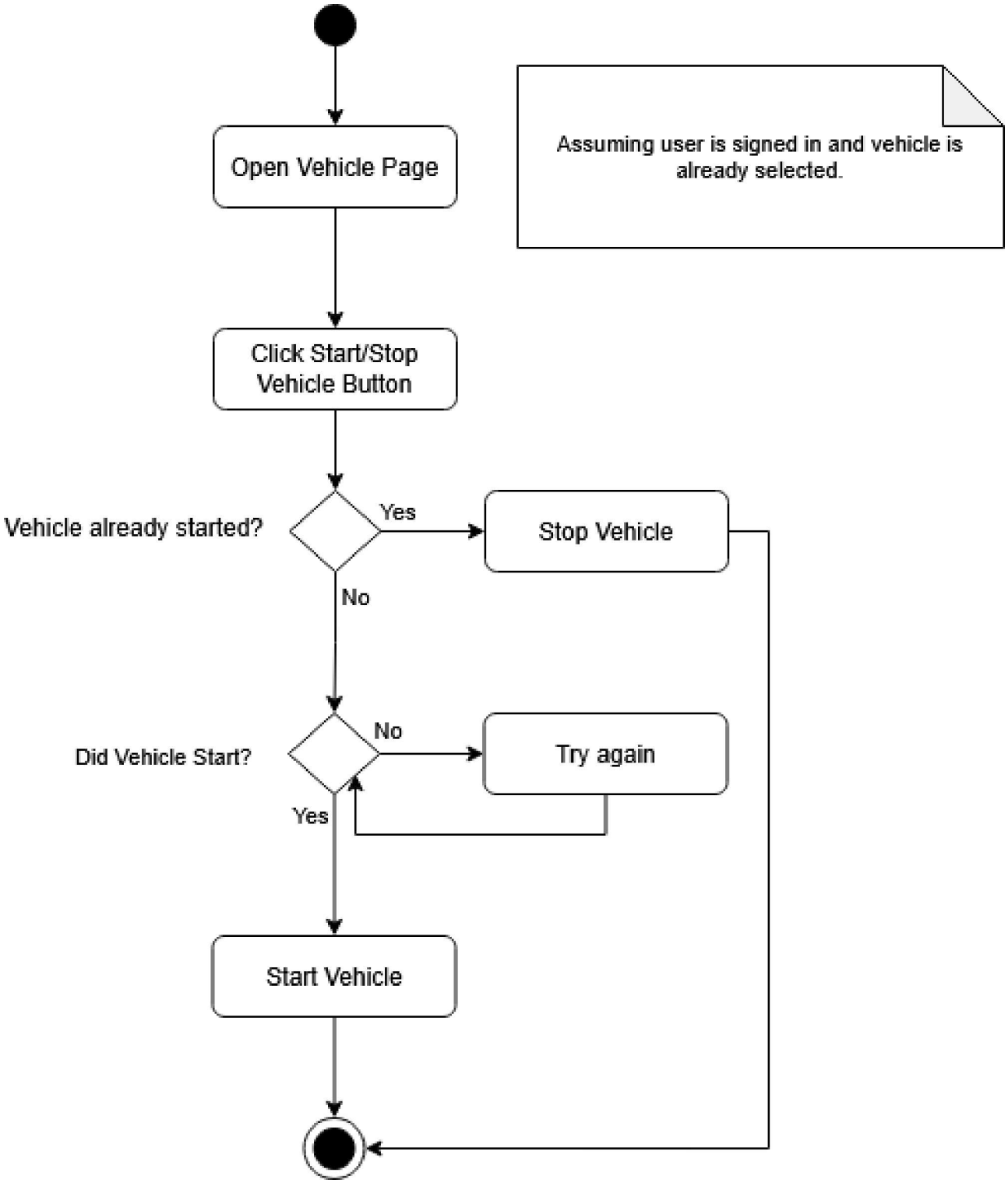
(Figure 6.2.5) Use case description for starting/stopping a vehicle.

Use Case Name:	View Vehicle Information	
Goal in context:	Allow the user to remotely view key vehicle status information like battery level, fuel level, tire pressure, and engine health from the app.	
Preconditions:	• The user is authenticated in the application. • The vehicle is online and connected to the network. • The vehicle supports transmitting status information to the app.	
Description	1	User selects the "View Vehicle Status" option in the app.
	2	The system sends a data request to the vehicle for status information.
	3	The vehicle responds with its current status data and the app displays the retrieved status information to the user.
Extensions	Step	Branching Action
	2a	If the vehicle is offline or unable to send data, display: "Unable to retrieve vehicle status. Please check the connection."
	3a	If specific data is unavailable, display available data and show: "Some status information is currently unavailable."

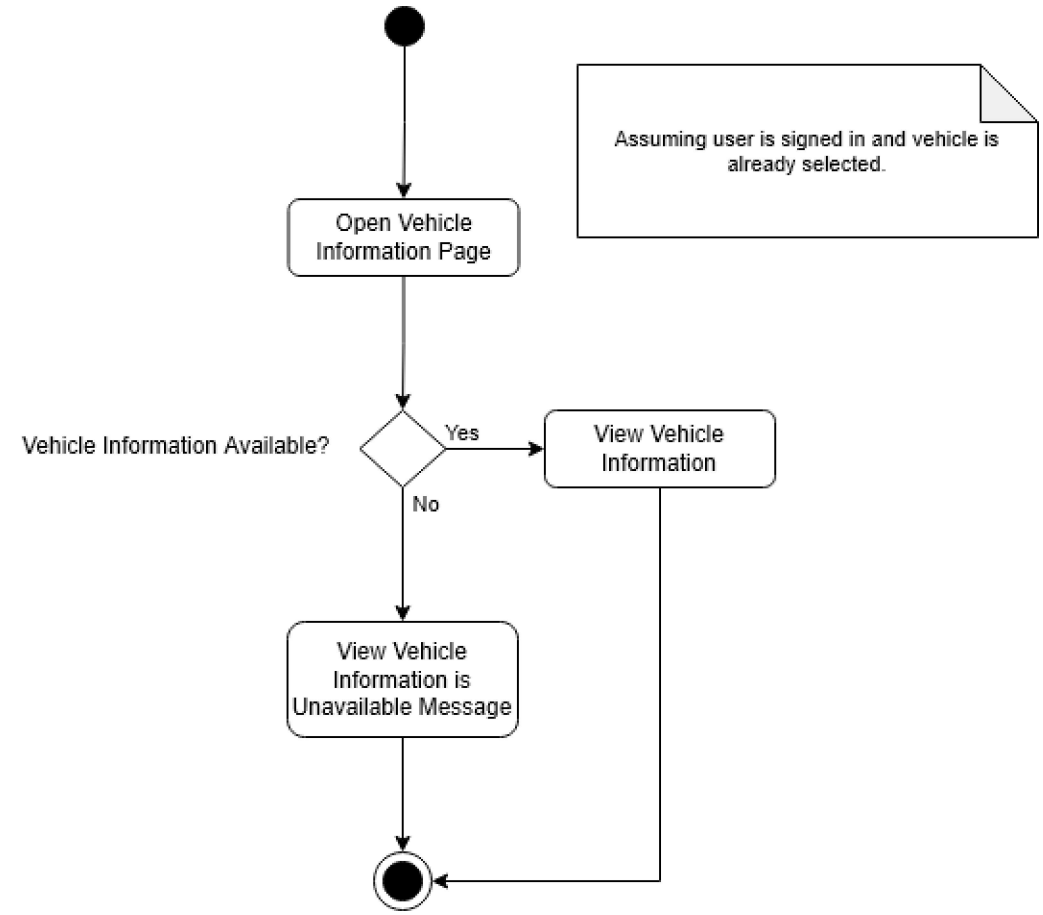
(Figure 6.2.6) Use case description for viewing a vehicle's information.



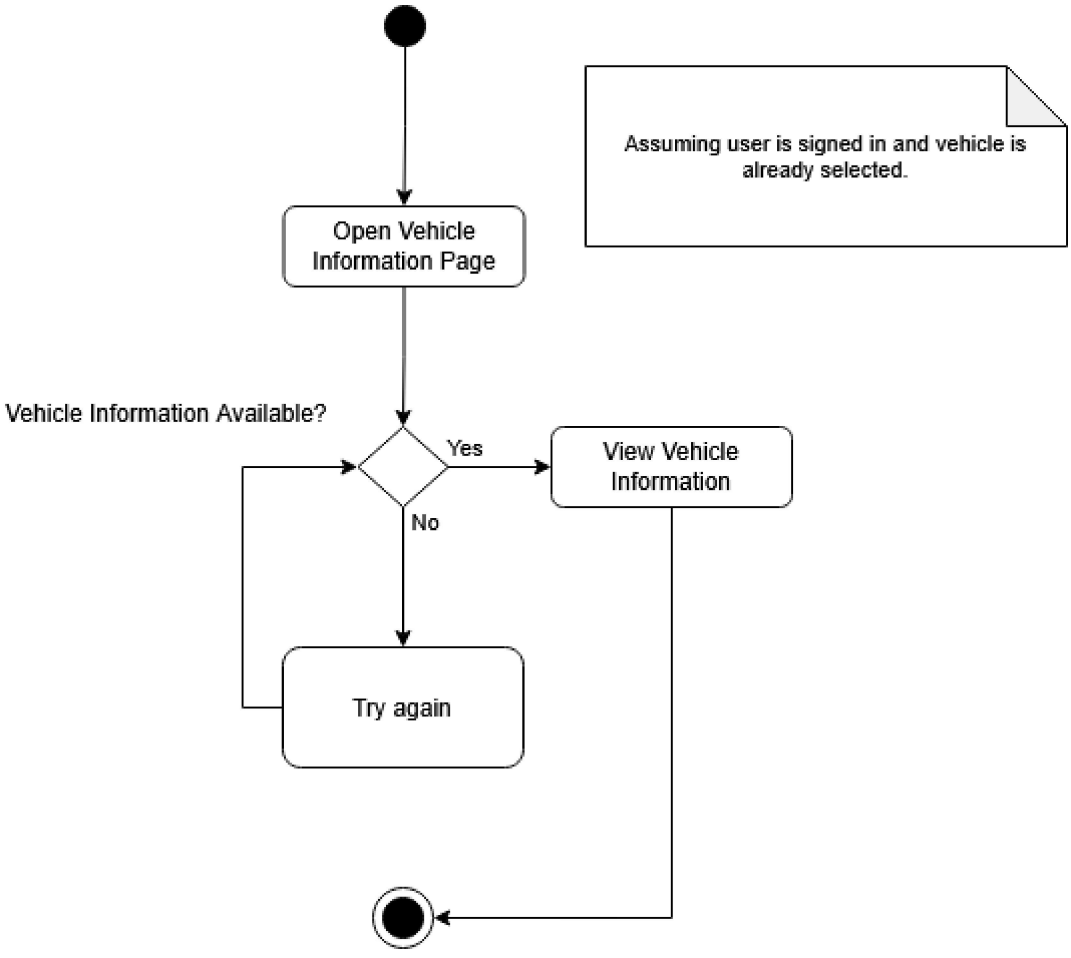
(Figure 6.2.7) First draft activity diagram for starting/stopping a vehicle.



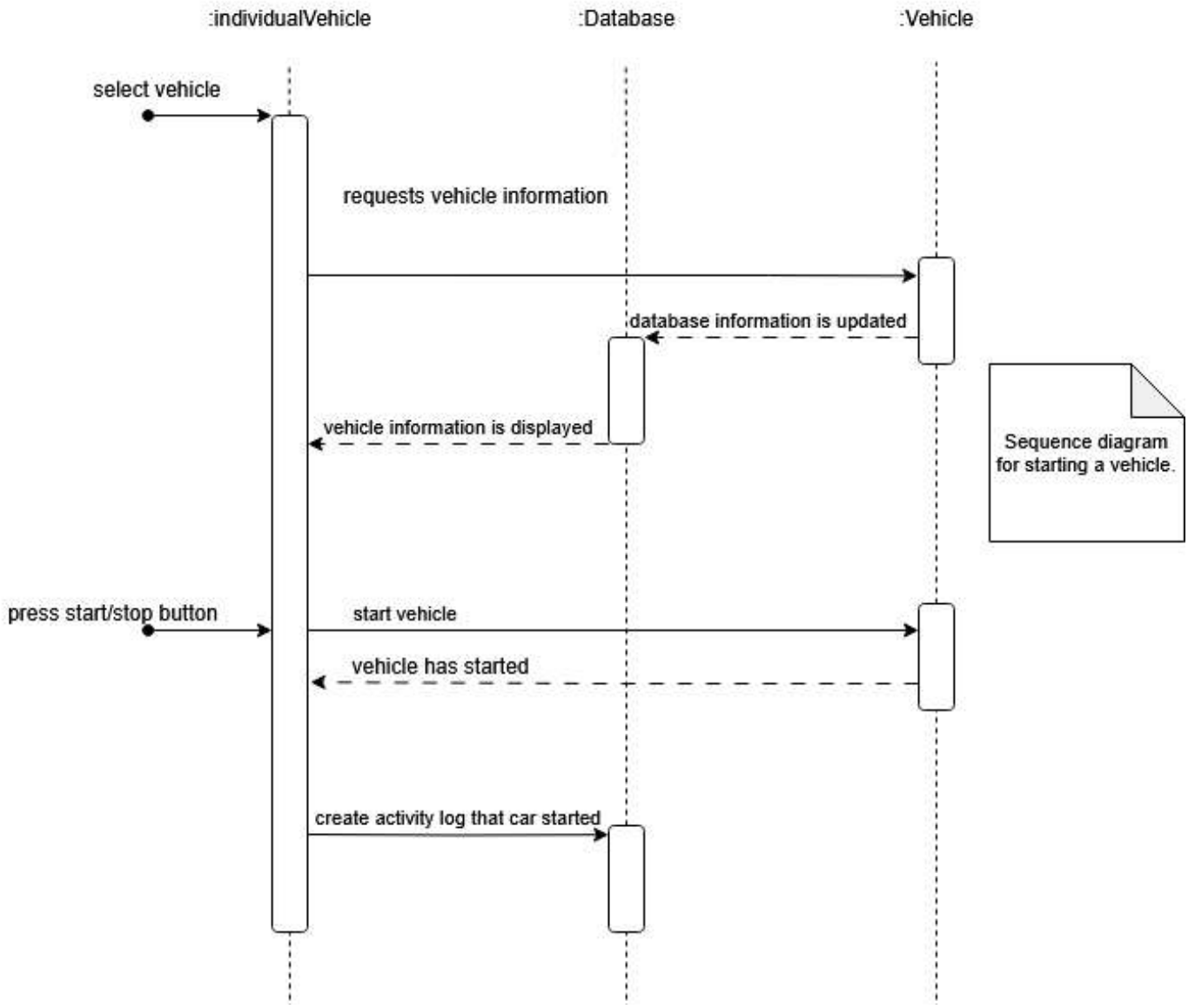
(Figure 6.2.8) Final draft activity diagram for starting/stopping a vehicle



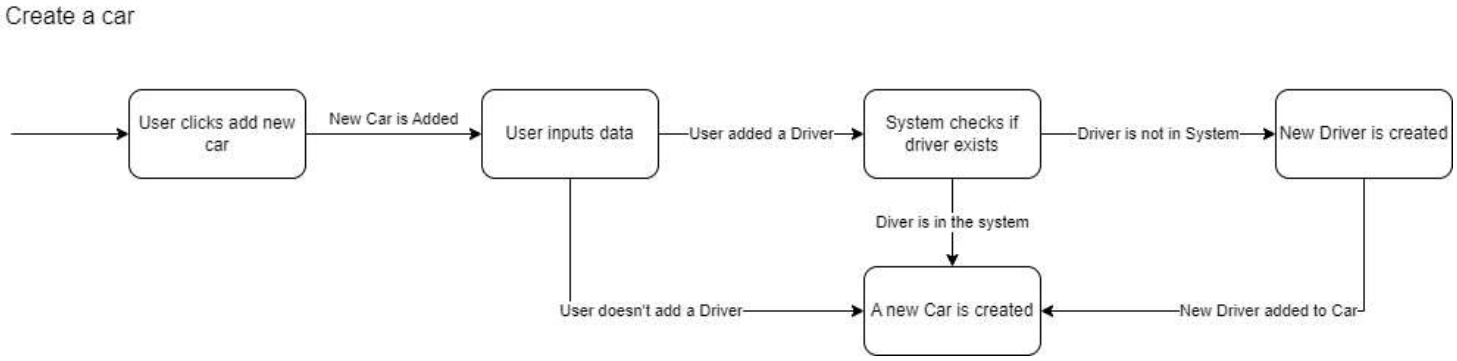
(Figure 6.2.9) First draft activity diagram for viewing vehicle information.



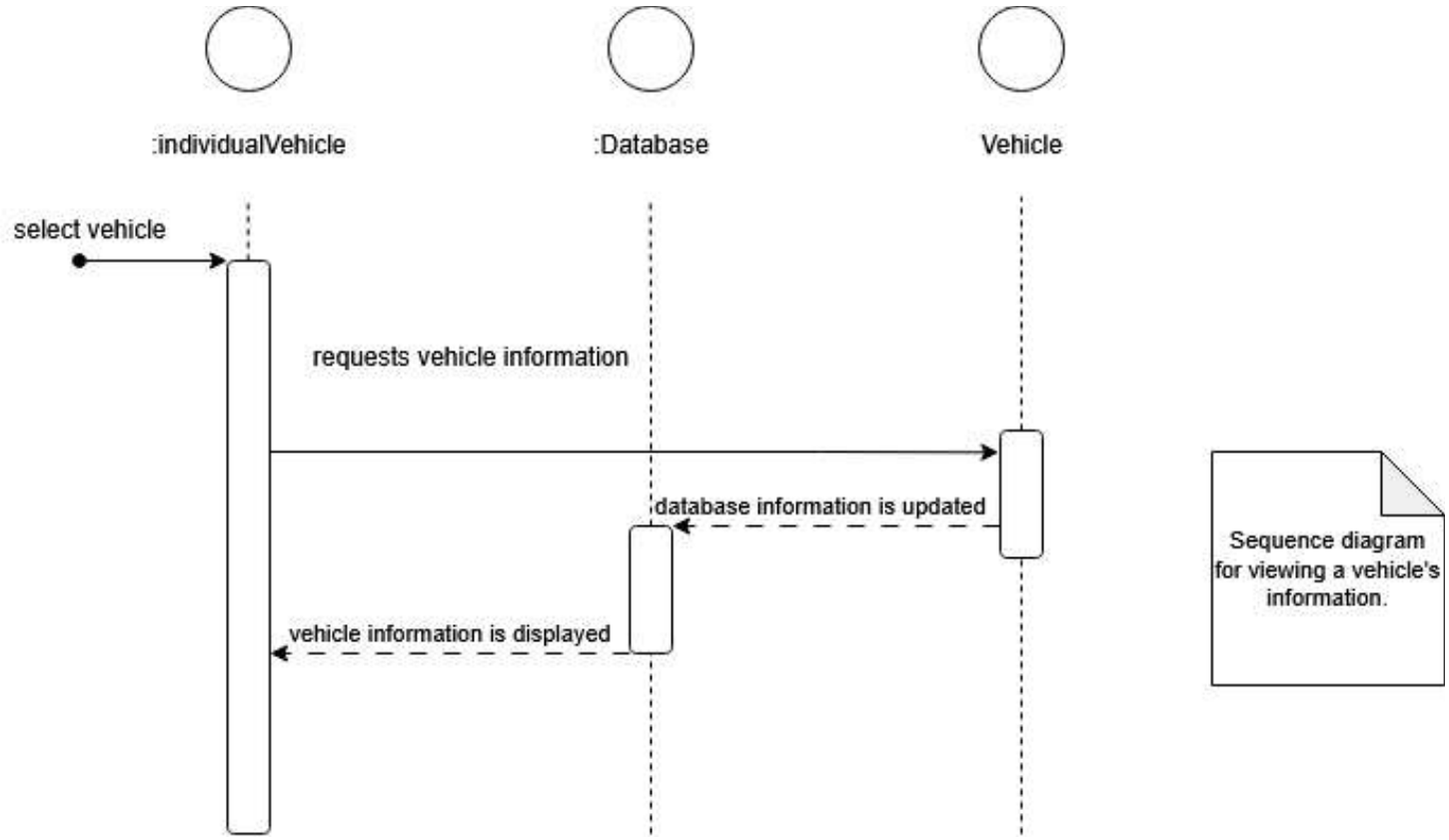
(Figure 6.2.10) Final draft activity diagram for viewing vehicle information.



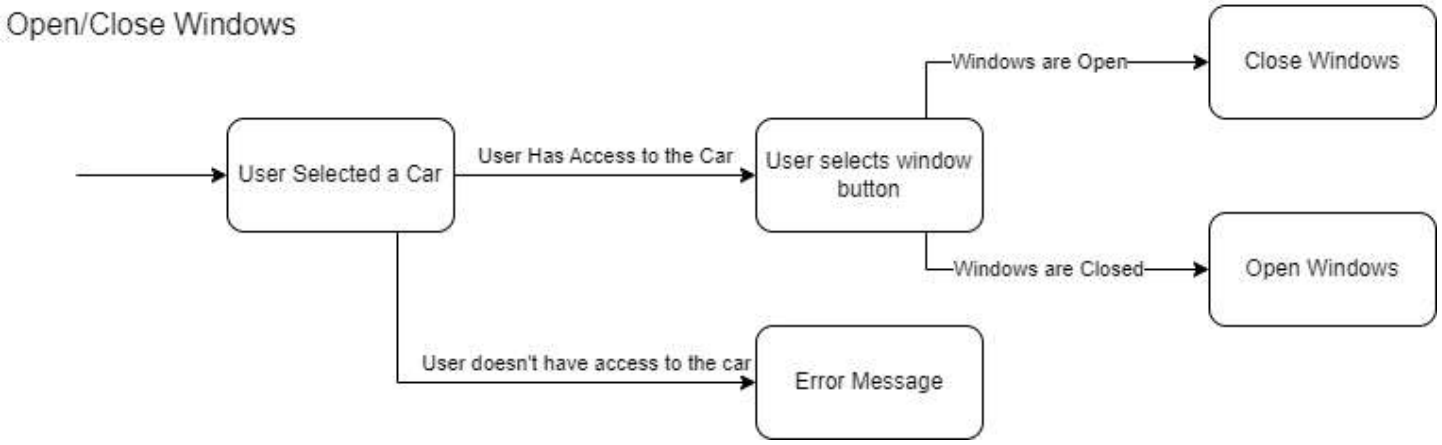
(Figure 6.2.11) Sequence diagram for starting a vehicle.



(Figure 6.2.12) State chart for registering a vehicle.



(Figure 6.2.13) Sequence diagram for viewing vehicle information.

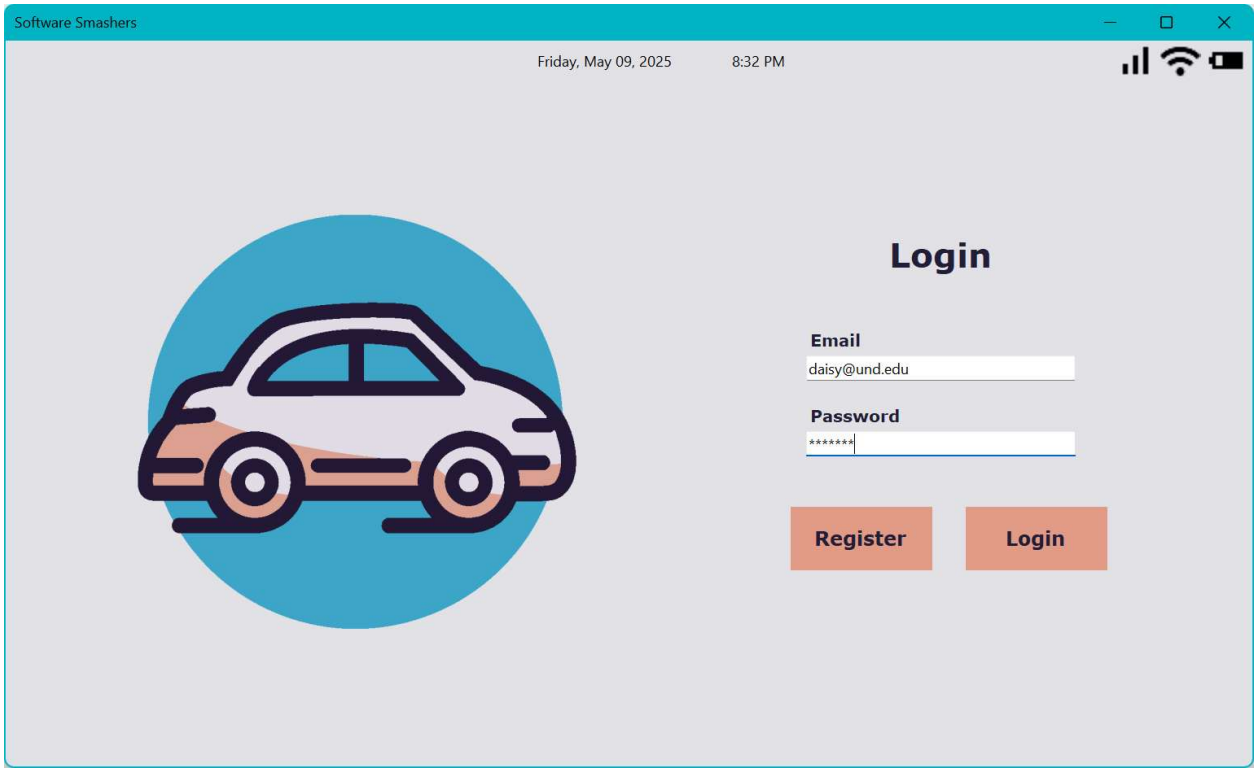


(Figure 6.2.14) State chart for opening/closing vehicle windows.

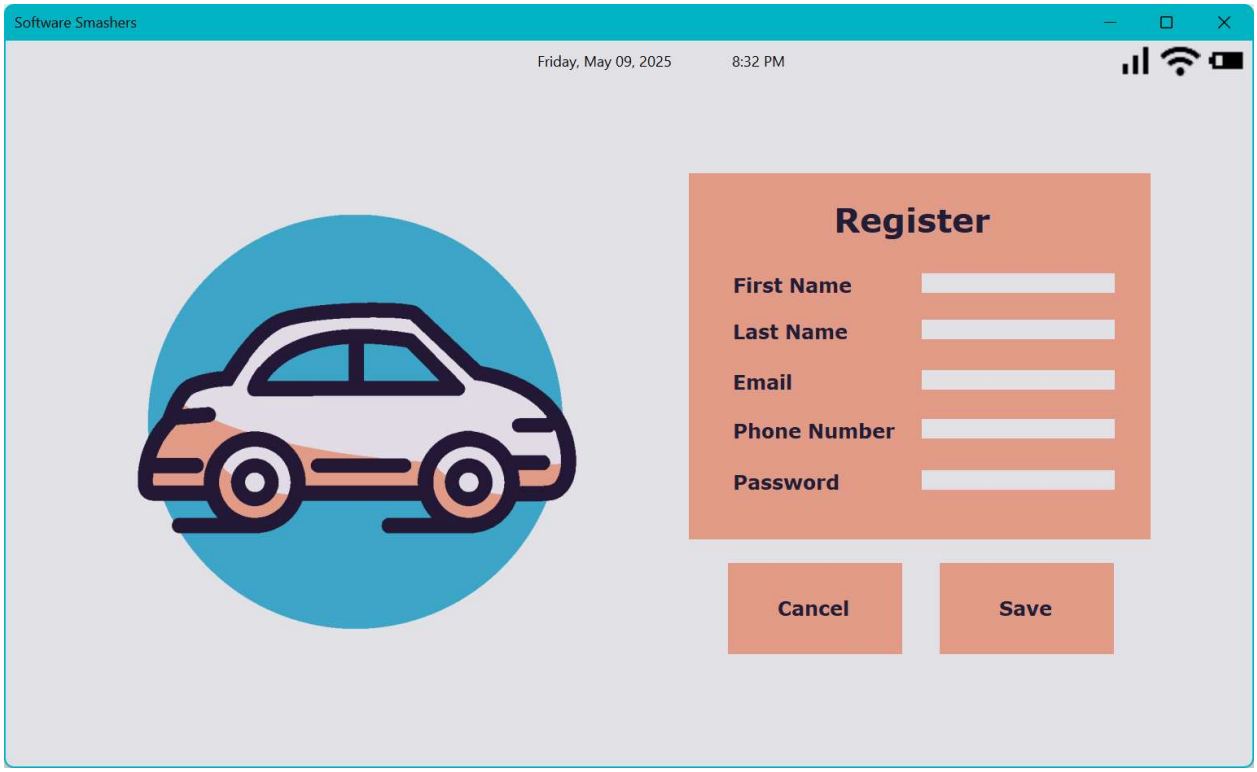
6.3 System Dictionary.

Class Diagram	A diagram showing how we've broken up our application into easier to complete modules.
Logs	A class that will serve as information storage when actions are performed, allowing the application to send notifications if necessary, and if notifications are enabled using "notificationsEnabled()". It will also be capable of updating the database with information.
Notification	A class that will take information from the relevant Log and use it to send a notification to the user about activity.
Sequence Diagram	A diagram to show how our system communicates with itself to perform desired actions.
SQL	Structured Query Language
SQL Query	A request to a database written in SQL to perform a specific action.
SQL Database	A collection of organized data structures that contain data and its relations to other data.
SystemSettings	A class that contains a user's preferences for their system settings, allowing the application to display accordingly.
	Includes a function "updateDatabase" that will store information in the database.
User	A class that will contain user information, and allow them to do things like "registerNewVehicle()", which would create a new Vehicle object with the given information.
UserID	A unique identification number that allows us to identify specific users and their associated information.
Vehicle	A class that contains information for a given vehicle, and will send commands to the real vehicle, while also being able to update its information in the database.
VehicleID	A unique identification number that allows us to identify specific vehicles and their associated information.
View Vehicle Information	The act of a driver selecting a vehicle in our application, bringing up the vehicle's associated information to be displayed on the next page.

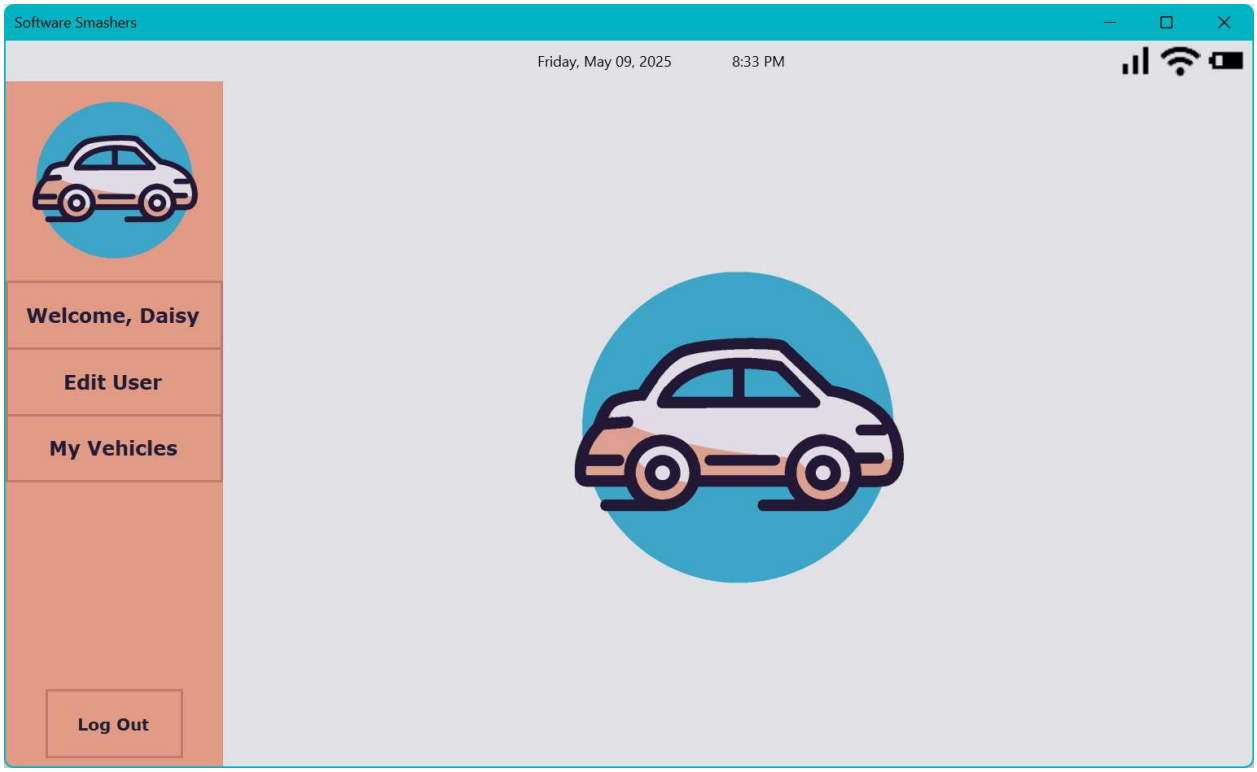
6.4 All interface screens.



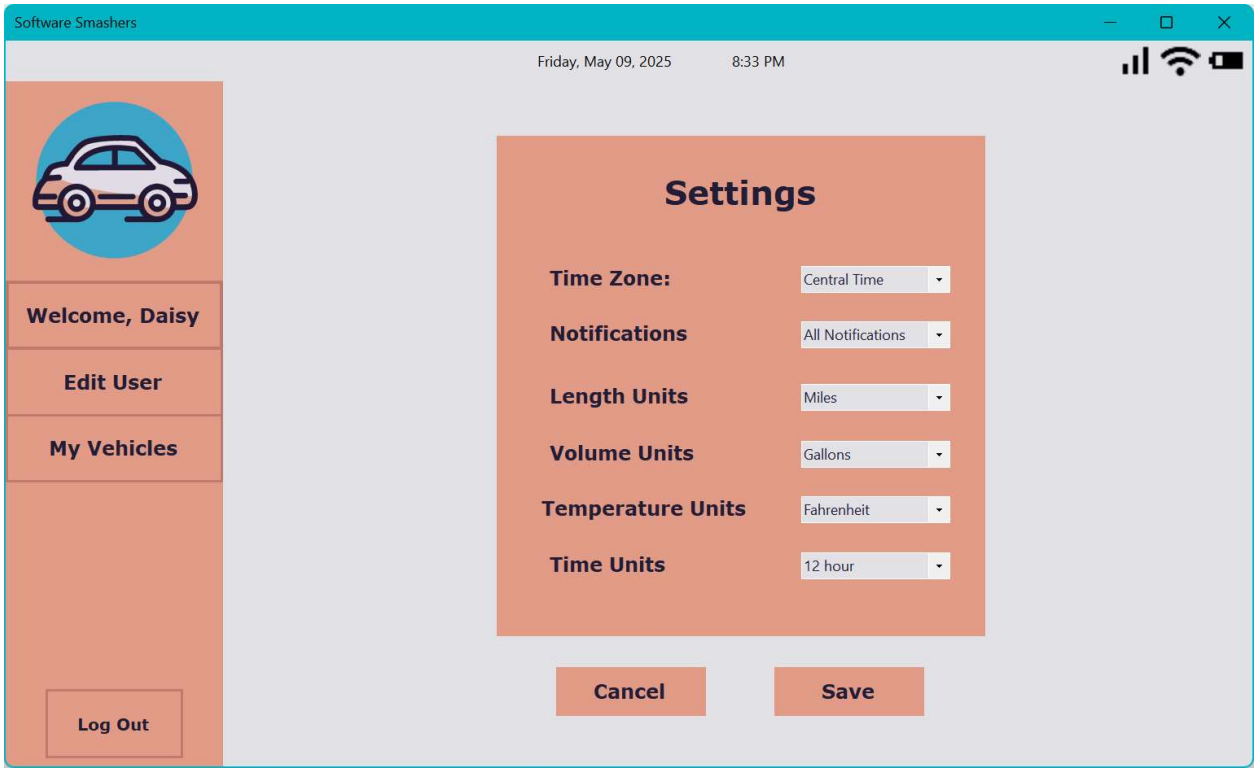
(Figure 6.4.1) Login Page with login information added to show the password being hidden.



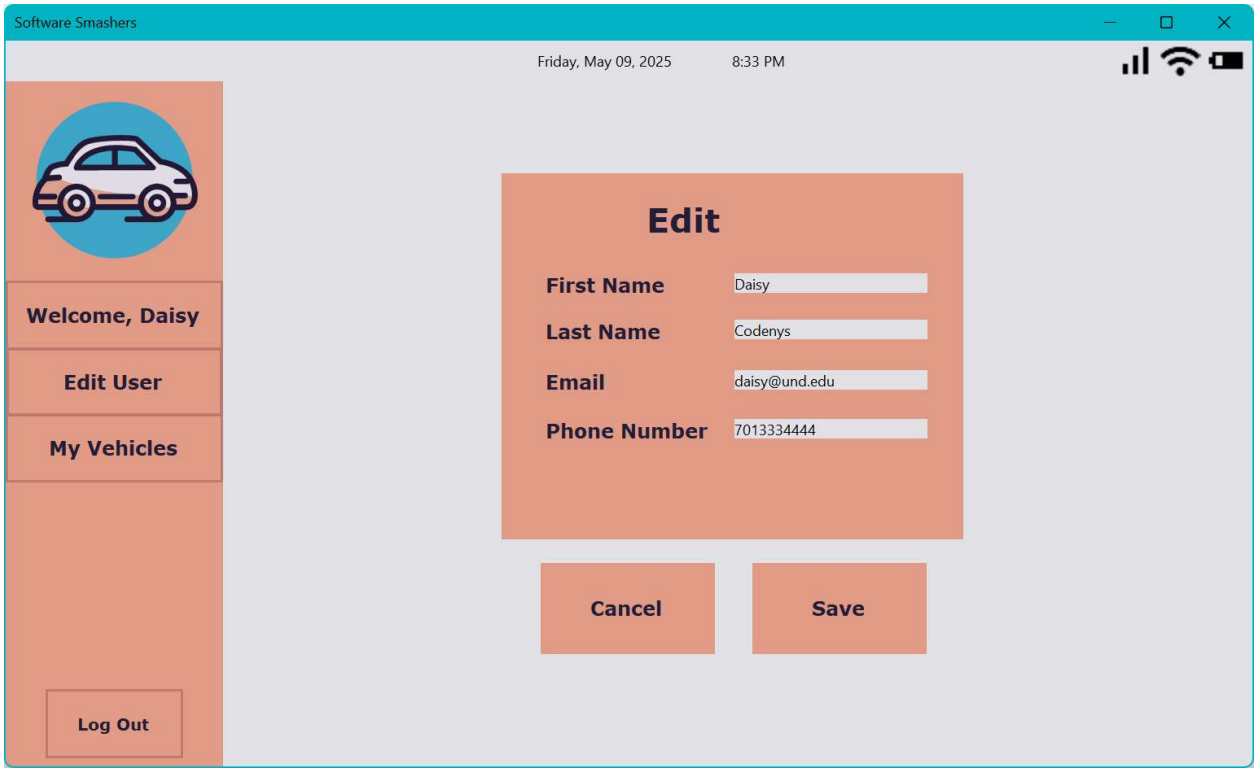
(Figure 6.4.2) Registration or New User Page.



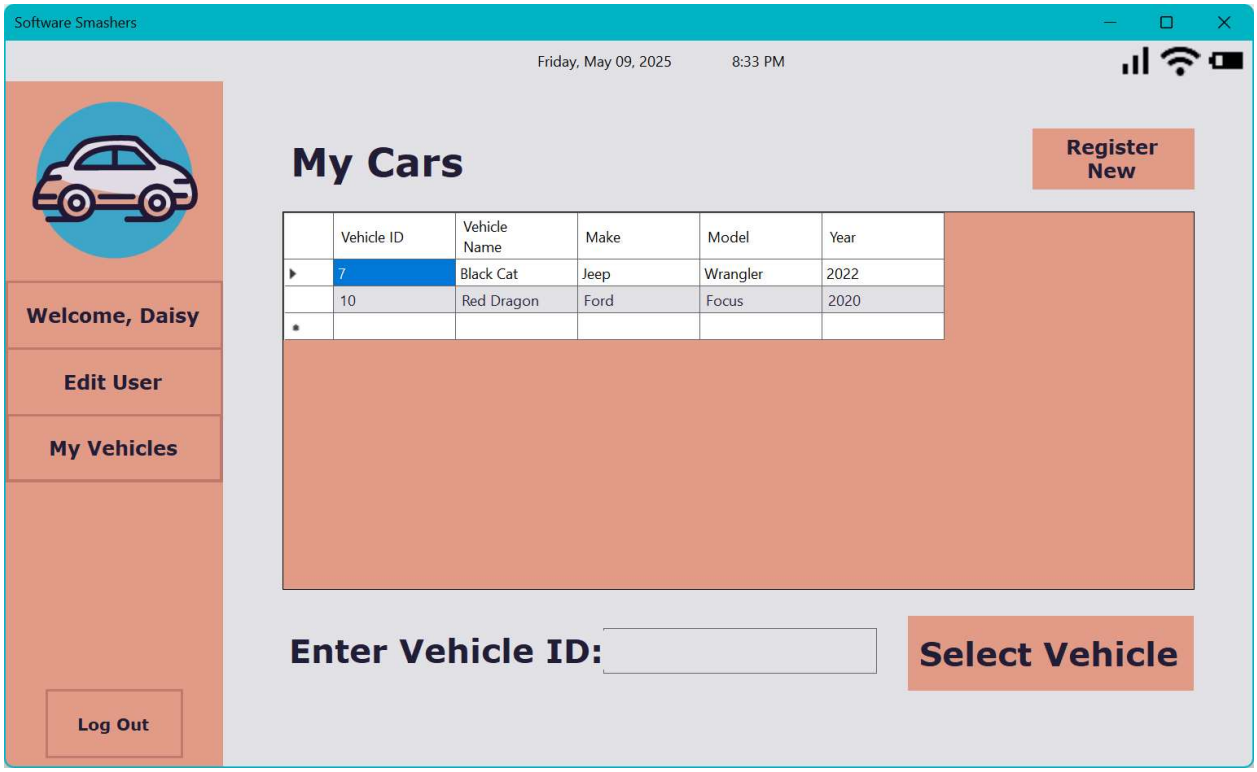
(Figure 6.4.3) Home Page. Notice the name in “Welcome, Daisy” will change with different user logins.



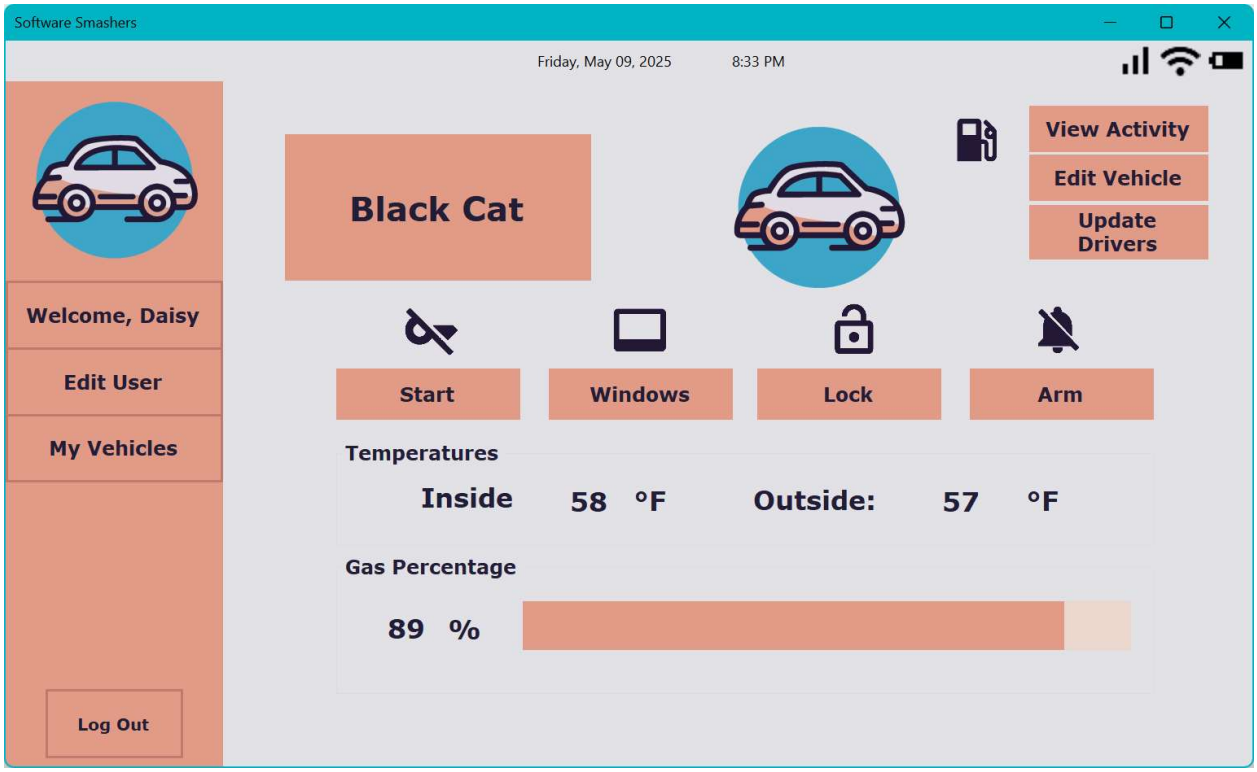
(Figure 6.4.4) Settings Page. This is used to change how the application shows information.



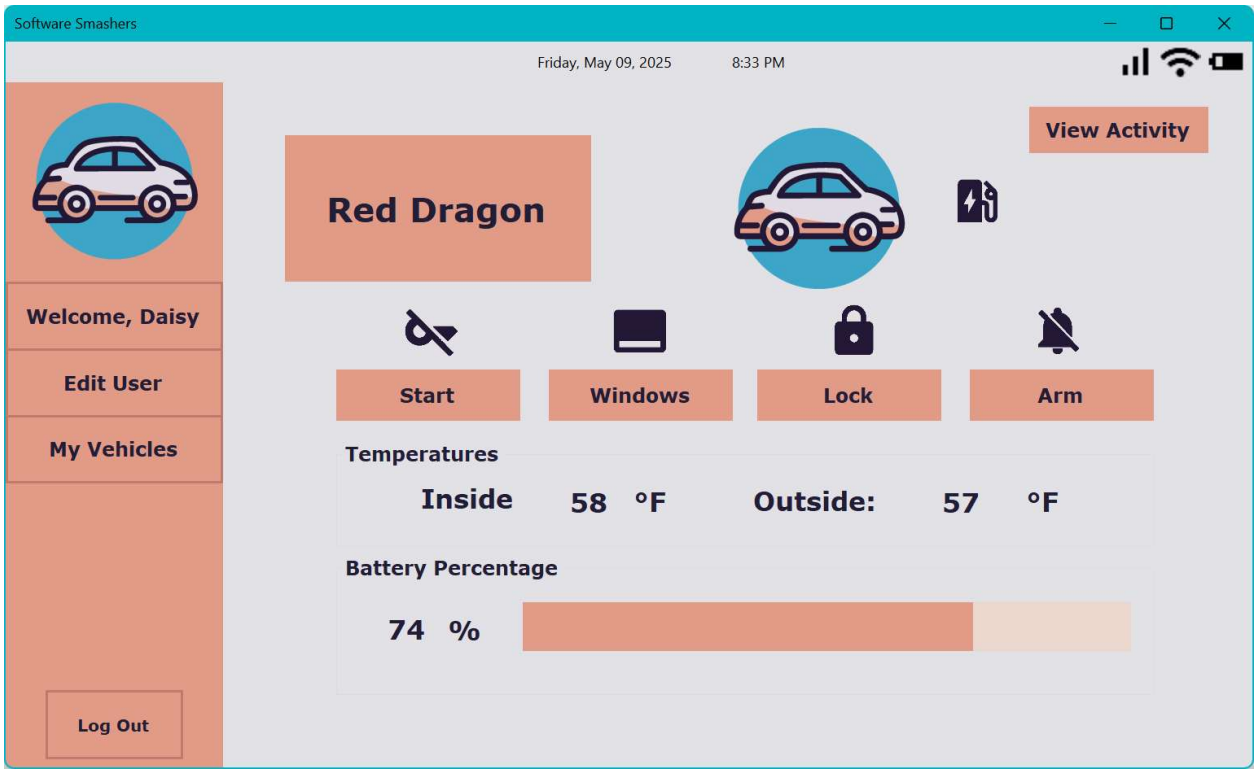
(Figure 6.4.5) Edit User Page. Similar to the Registration Page, you can update login information.



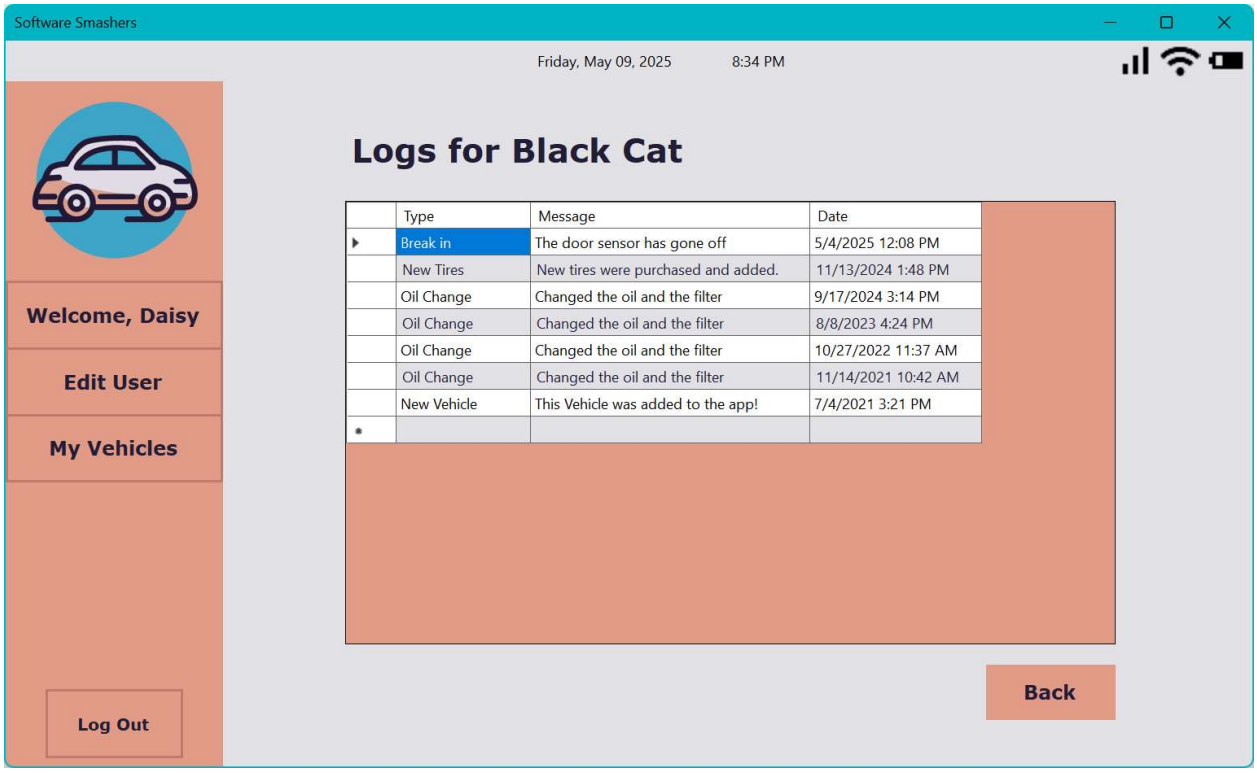
(Figure 6.4.6) My Cars Page. A view of all vehicles that you are an owner or a diver of.



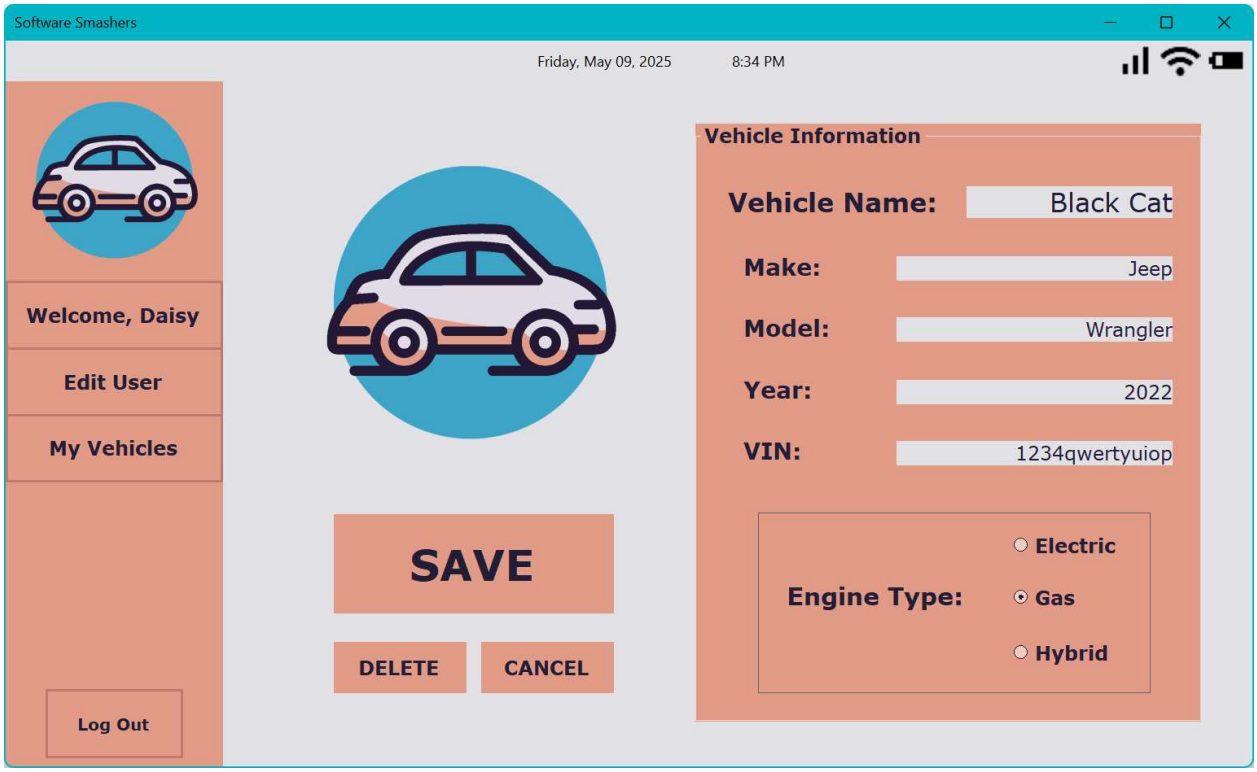
(Figure 6.4.7) My Car Page. This vehicle is named “Black Cat” and is a gas vehicle.



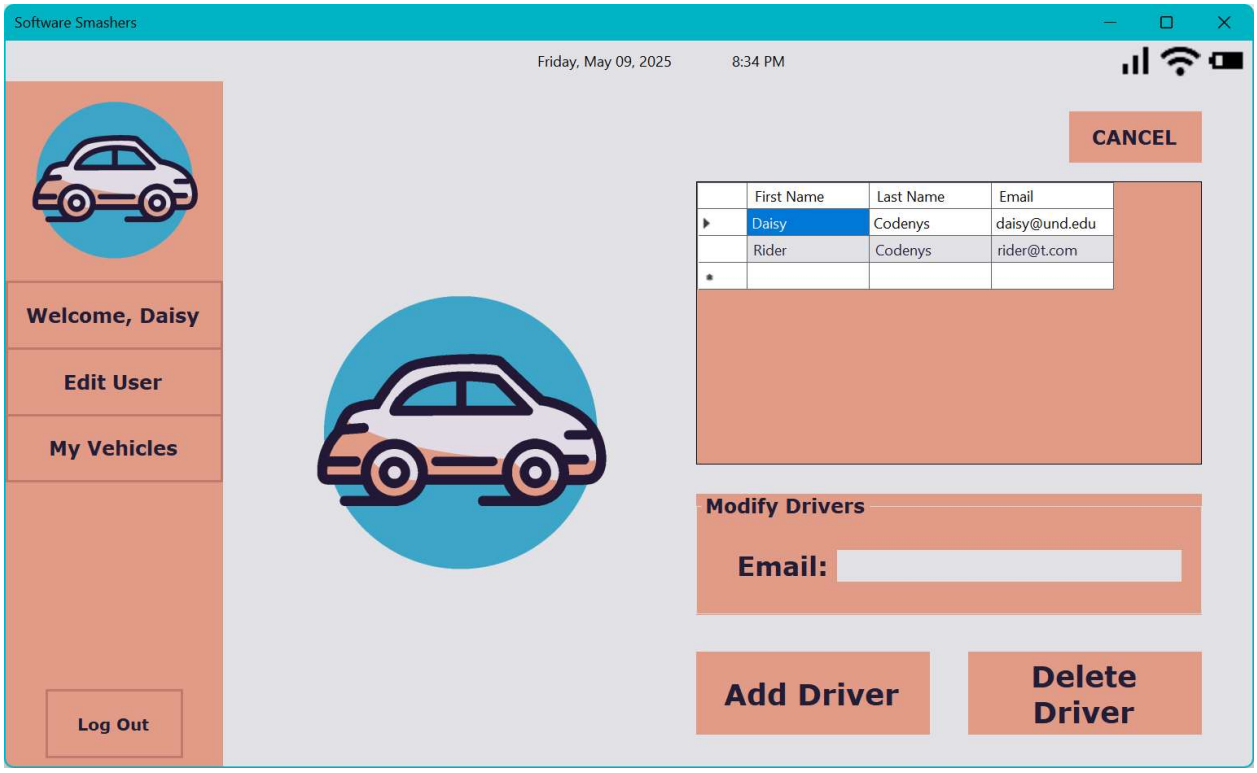
(Figure 6.4.8) My Car Page. This vehicle is named “Red Dragon” and is an electric vehicle.



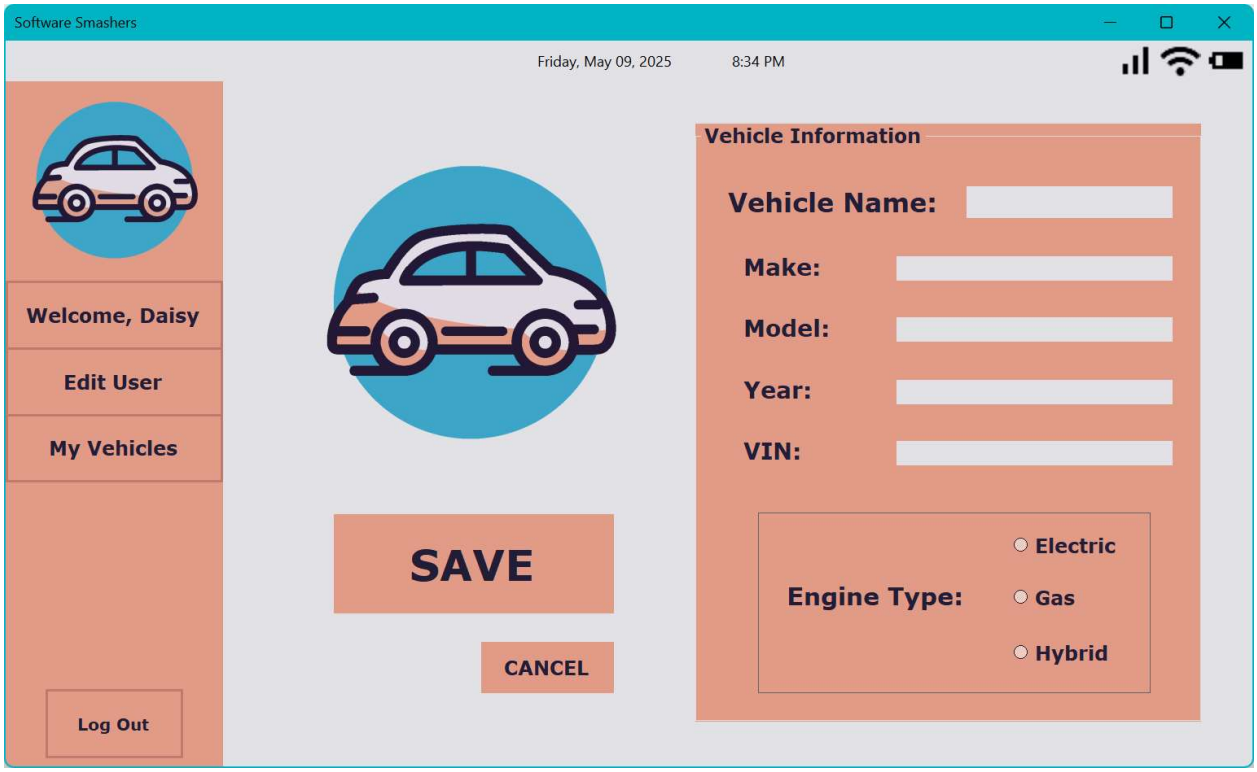
(Figure 6.4.9) Logs Page. This page will display a list of messages that the vehicle has.



(Figure 6.4.10) Edit Vehicle Page. Similar to the New Vehicle Page, you can update vehicle information.



(Figure 6.4.11) Driver Page. Add and remove drivers of a specified vehicle.



(Figure 6.4.12) New Vehicle Page. Create and register a vehicle with your account as the owner.

6.5 Invoice

The Software Smashers team worked well together and created a balanced project that required specialties from all team members. From creating UI drawings to SQL database queries our team worked together in voice calls to make sure that every part of this project included each of us.

Invoice

Daisy Codenys	100%
Sydney Nilles	100%
Ty Gregory	100%