Click here to enter text.

Progress Report 01

CSCI 363 Term Project

Team Rough Riders

Kenneth Jahnke, Max Forst

CSCI 363: User Interface Design
Fall 2024
October 27, 2024

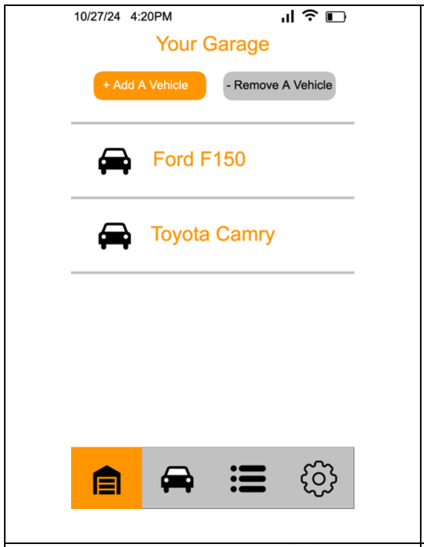
Table of Contents

Work Split

Kenneth Jahnke – 50%

Max Forst – 50%

Interface Drawings

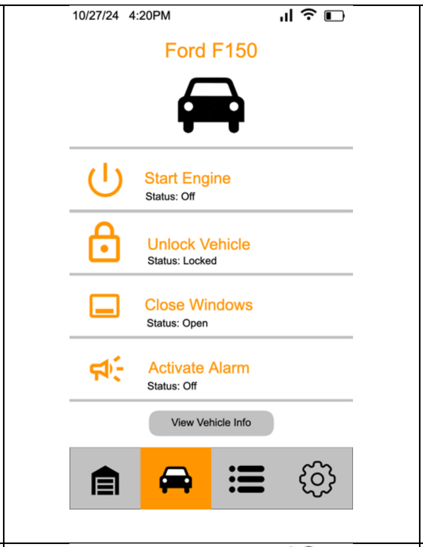

10/27/24 4:20PM

Your Garage

+ Add A Vehicle | - Remove A Vehicle

Ford F150

Toyota Camry


10/27/24 4:20PM

Ford F150

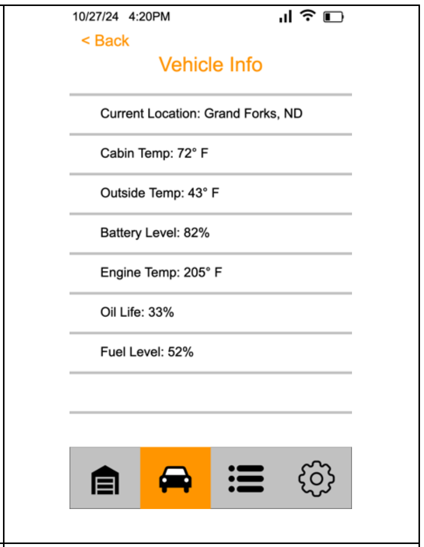
Start Engine
Status: Off

Unlock Vehicle
Status: Locked

Close Windows
Status: Open

Activate Alarm
Status: Off

View Vehicle Info


10/27/24 4:20PM

< Back

Vehicle Info

Current Location: Grand Forks, ND

Cabin Temp: 72° F

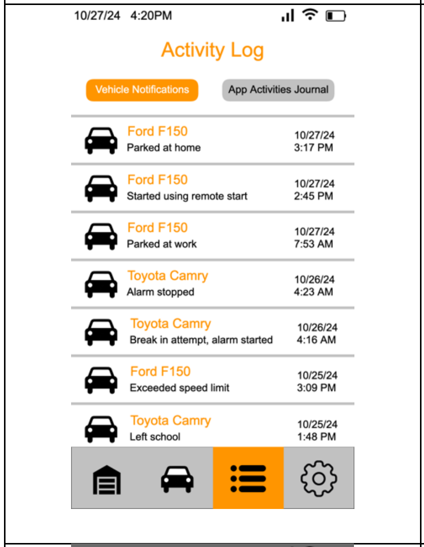
Outside Temp: 43° F

Battery Level: 82%

Engine Temp: 205° F

Oil Life: 33%

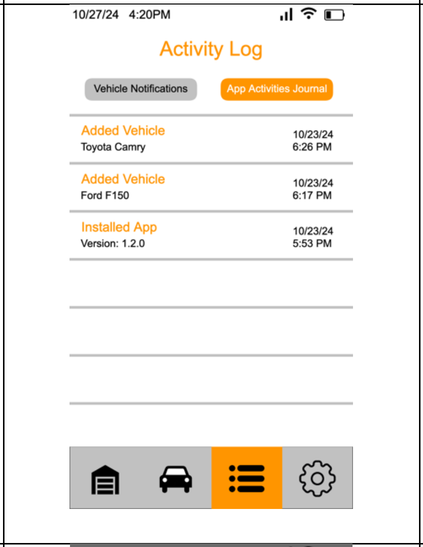
Fuel Level: 52%


10/27/24 4:20PM

Activity Log

Vehicle Notifications | App Activities Journal

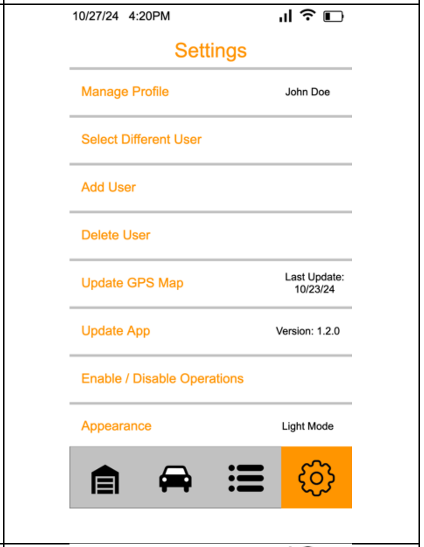
Ford F150 Parked at home	10/27/24 3:17 PM
Ford F150 Started using remote start	10/27/24 2:45 PM
Ford F150 Parked at work	10/27/24 7:53 AM
Toyota Camry Alarm stopped	10/26/24 4:23 AM
Toyota Camry Break in attempt, alarm started	10/26/24 4:16 AM
Ford F150 Exceeded speed limit	10/25/24 3:09 PM
Toyota Camry Left school	10/25/24 1:48 PM


10/27/24 4:20PM

Activity Log

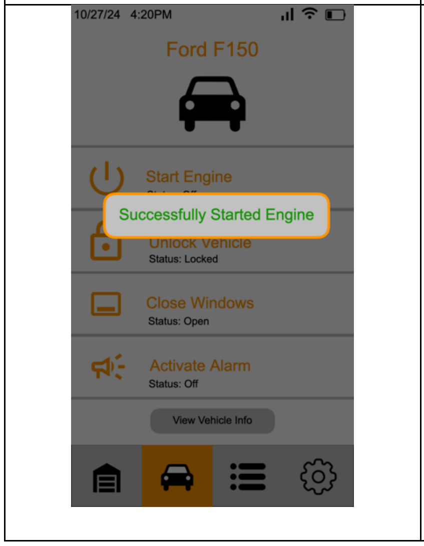
Vehicle Notifications | App Activities Journal

Added Vehicle Toyota Camry	10/23/24 6:26 PM
Added Vehicle Ford F150	10/23/24 6:17 PM
Installed App Version: 1.2.0	10/23/24 5:53 PM


10/27/24 4:20PM

Settings

Manage Profile	John Doe
Select Different User	
Add User	
Delete User	
Update GPS Map	Last Update: 10/23/24
Update App	Version: 1.2.0
Enable / Disable Operations	
Appearance	Light Mode


10/27/24 4:20PM

Ford F150

Start Engine

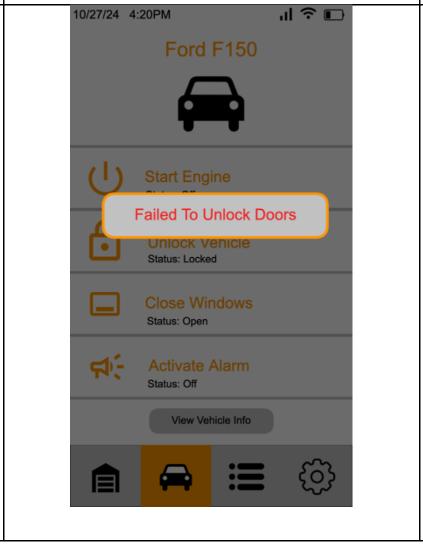
Successfully Started Engine

Unlock Vehicle
Status: Locked

Close Windows
Status: Open

Activate Alarm
Status: Off

View Vehicle Info


10/27/24 4:20PM

Ford F150

Start Engine

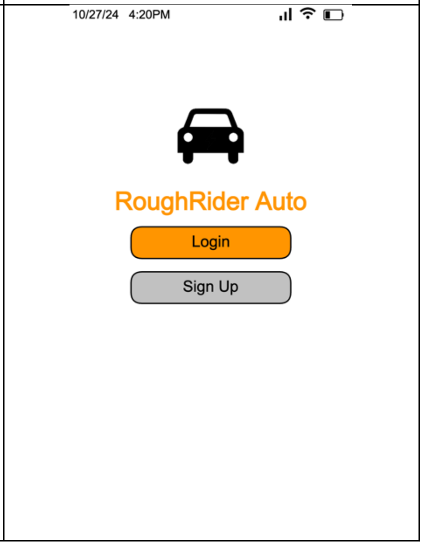
Failed To Unlock Doors

Unlock Vehicle
Status: Locked

Close Windows
Status: Open

Activate Alarm
Status: Off

View Vehicle Info


10/27/24 4:20PM

RoughRider Auto

Login

Sign Up

For the interface drawing designs we opted to go with tabs along the bottom of the screen with images that clearly identify which aspect of the app a user would need to go to if they wanted to accomplish a specific task. We have made sure to make selected tabs highlighted in the app's accent color, so the user will know exactly what page or tab that they are on. We have also highlighted more important items like page titles and menu item titles in this accent color as well to help draw the user's attention to them.

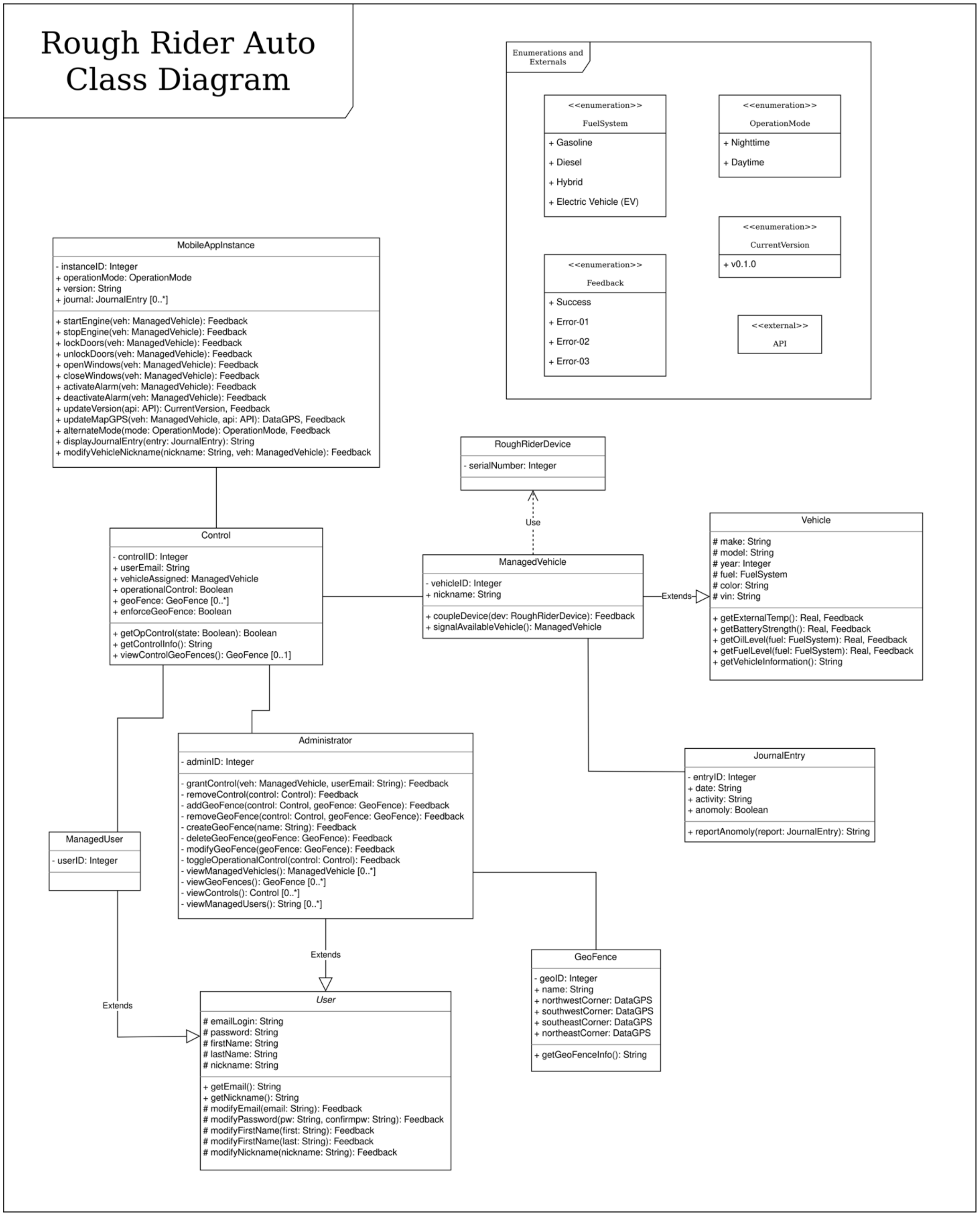
On each page, we opted for a vertical layout of list-like items so that it is clear to the user that they can scroll down on the screen if they need to see more items. This also contrasts with the horizontal separated layout of the menu bar at the bottom, helping distinguish the two as separate entities.

We have made sure to add images next to each vehicle and menu item on several screens to help users more easily identify which operation they want to accomplish more quickly without needing to read the options next to each image.

We have made sure to show enough information on each page to make the user feel well informed about aspects of their vehicles and account, without showing too much information, which could cause the user to feel overwhelmed from an information overload.

These are just basic designs and may change in the future. Additionally, they are not complete as there are missing screens that would provide functionality such as typing the name of the vehicle, or signing up or logging into an account. Several settings options also do not have a drawing associated with them here. These are things that we aim to accomplish with the final design, but for now we just have the basic concepts of design down.

Class Diagram


Rough Rider Auto Class Diagram
Enumerations and Externals
<<enumeration>> FuelSystem
+ Gasoline
+ Diesel
+ Hybrid
+ Electric Vehicle (EV)
<<enumeration>> OperationMode
+ Nighttime
+ Daytime
<<enumeration>> CurrentVersion
+ v0.1.0
<<enumeration>> Feedback
+ Success
+ Error-01
+ Error-02
+ Error-03
<<external>> API
MobileAppInstance
- instanceID: Integer
+ operationMode: OperationMode
+ version: String
+ journal: JournalEntry [0..*]
+ startEngine(veh: ManagedVehicle): Feedback
+ stopEngine(veh: ManagedVehicle): Feedback
+ lockDoors(veh: ManagedVehicle): Feedback
+ unlockDoors(veh: ManagedVehicle): Feedback
+ openWindows(veh: ManagedVehicle): Feedback
+ closeWindows(veh: ManagedVehicle): Feedback
+ activateAlarm(veh: ManagedVehicle): Feedback
+ deactivateAlarm(veh: ManagedVehicle): Feedback
+ updateVersion(api: API): CurrentVersion, Feedback
+ updateMapGPS(veh: ManagedVehicle, api: API): DataGPS, Feedback
+ alternateMode(mode: OperationMode): OperationMode, Feedback
+ displayJournalEntry(entry: JournalEntry): String
+ modifyVehicleNickname(nickname: String, veh: ManagedVehicle): Feedback
RoughRiderDevice
- serialNumber: Integer
Use
Control
- controlID: Integer
+ userEmail: String
+ vehicleAssigned: ManagedVehicle
+ operationalControl: Boolean
+ geoFence: GeoFence [0..*]
+ enforceGeoFence: Boolean
+ getOpControl(state: Boolean): Boolean
+ getControlInfo(): String
+ viewControlGeoFences(): GeoFence [0..1]
ManagedVehicle
- vehicleID: Integer
+ nickname: String
+ coupleDevice(dev: RoughRiderDevice): Feedback
+ signalAvailableVehicle(): ManagedVehicle
Extends
Vehicle
# make: String
# model: String
# year: Integer
# fuel: FuelSystem
# color: String
# vin: String
+ getExternalTemp(): Real, Feedback
+ getBatteryStrength(): Real, Feedback
+ getOilLevel(fuel: FuelSystem): Real, Feedback
+ getFuelLevel(fuel: FuelSystem): Real, Feedback
+ getVehicleInformation(): String
Administrator
- adminID: Integer
- grantControl(veh: ManagedVehicle, userEmail: String): Feedback
- removeControl(control: Control): Feedback
- addGeoFence(control: Control, geoFence: GeoFence): Feedback
- removeGeoFence(control: Control, geoFence: GeoFence): Feedback
- createGeoFence(name: String): Feedback
- deleteGeoFence(geoFence: GeoFence): Feedback
- modifyGeoFence(geoFence: GeoFence): Feedback
- toggleOperationalControl(control: Control): Feedback
- viewManagedVehicles(): ManagedVehicle [0..*]
- viewGeoFences(): GeoFence [0..*]
- viewControls(): Control [0..*]
- viewManagedUsers(): String [0..*]
JournalEntry
- entryID: Integer
+ date: String
+ activity: String
+ anomoly: Boolean
+ reportAnomoly(report: JournalEntry): String
ManagedUser
- userID: Integer
Extends
Extends
User
# emailLogin: String
# password: String
# firstName: String
# lastName: String
# nickname: String
+ getEmail(): String
+ getNickname(): String
# modifyEmail(email: String): Feedback
# modifyPassword(pw: String, confirmpw: String): Feedback
# modifyFirstName(first: String): Feedback
# modifyFirstName(last: String): Feedback
# modifyNickname(nickname: String): Feedback
GeoFence
- geoID: Integer
+ name: String
+ northwestCorner: DataGPS
+ southwestCorner: DataGPS
+ southeastCorner: DataGPS
+ northeastCorner: DataGPS
+ getGeoFenceInfo(): String

Concept of Operations

Most of the basic functionalities specified by the customer (Grant) are available within as methods with *MobileAppInstance*. Basic functionalities are those that are made possible by the Rough Rider Device ™ (RRD), such as remote starting/stopping the engine, remote activating/deactivating the alarm system, etc. A few customer requirements are inherent to the vehicle and such data/readings would still be available in the absence of the RRD, such as determining the external temperature, the batter level, etc. Hence, this data is accessible via public methods within the vehicle (which a *MobileAppInstance* could call).

A *Vehicle* becomes a *ManagedVehicle* when it is equipped with (a depedency) and coupled to (*coupleDevice* method) an RRD. An *Administrator* can then view the *ManagedVehicle* as available within the motor pool via its *signalAvailableVehicle* method (part of initial setup).

A user (*ManagedUser* or *Administrator*) can exercise the functionalities made available by the RRD via a *MobileAppInstance* when he/she is assigned to a vehicle (*ManagedVehicle)* by way of a *Control* node with operational control enabled. A *Control* node outlines which user is assigned to what vehicle, what instances of GeoFence the node is bound to (if any), and whether the user has operational control over the vehicle (i.e., can exercise the functionalities).

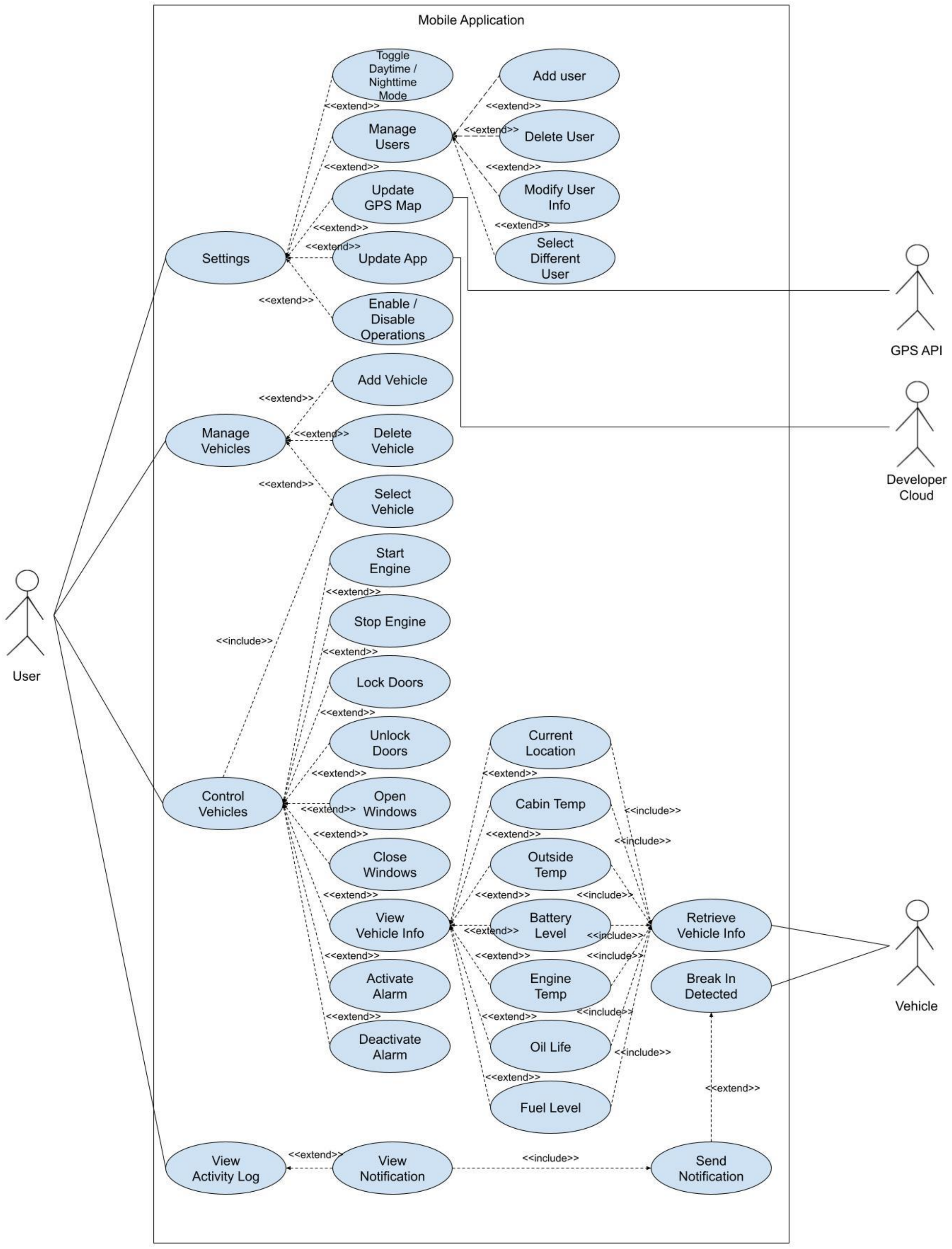
An *Administrator* "does the heavy lifting" within the local system by incorporating instances of *ManagedVehicle*; creating *Control* nodes, which assigns operational control of instances of *ManagedVehicle* to subclasses of *User* (*ManagedUser* or *Administrator*); managing operational control; deleting *Control* nodes; creating and deleting instances of *GeoFence*; applying and removing instances of *GeoFence* to *Control* nodes; and dictating whether instances of *GeoFence* are enforced within *Control* nodes. While multiple instances of *Administrator* are possible, it is not recommended.

A number of enumerations were declared to specify a range of values a class can take on for an attribute when applicable:

1. *FuelSystem* – This fulfills the customer requirement to support (1) petro-fuel (gas or diesel) vehicles, (2) hybrid vehicles, and (3) electric vehicles.
2. *OperationMode* – This fulfills the customer requirement to have a day- and night-time operational mode.
3. *CurrentVersion* – When the application updates, it is assumed there is at any time only one supported version for distribution, which we be the only value available for this enumeration.
4. *Feedback* – This is a mock-up class to demonstrate that after most actions, the user will be given a feedback message. The “Success” message can be tailored to match the context of individual successful actions. The list of error codes can be expanded to describe known exceptions, each with a message to the user on what went wrong and how to fix it.

The external data type *API* is a mock for all data services the system can call upon to retrieve external data, such as GPS data. These data services can be specified to fit customer needs. (Note: The *DataGPS* data type found about the diagram implies retrieved data is likely not of a primitive type or one that can be defined within the system, but one that a GPS API will issue “as-is”.)

Use Case Diagram



This use case diagram shows how a user would interact with our application on their mobile device. It shows the connections between the main functions that a user can execute as well as sub-functions that extend the main function. Such functions include starting the vehicle under the control vehicle function for example.

There are also functions that need to include other functions in order to fully operate. An example of a function like this would be how the control vehicle function needs the user to select a vehicle to control first, so it includes the select vehicle function, which itself is extending the manage vehicles function.

There are also other users / outside entities that will need access to the app in order to make it function. These other entities include the vehicle itself, so that way the app can actually control and receive information about the vehicle. The GPS API, which will allow the user to tell where each of their vehicles is at, as well as enable geofencing capabilities. Also, there is the developer cloud, which is how updates to the app would be pushed to the user if we were to actually publish this project as an actual application.

Issues and Resolutions

1. Coordination.

   Issue – Both Max and Jahnke are remote students, so they needed to identify a way to collaborate on work and communicate.

   Resolution – Both submitted documents to OneDrive and shared them with the other so different areas of work could be done concurrently. The two formed a dedicated Discord server to hold meetings if needed, coordinate basic tasks, and send updates.

2. System conceptualization.

   Issue – The concept is simple. The system implementation is not so simple, as evidenced by the questions such an endeavor raises. How do you gain control over a vehicle via an after-market device? Does the user issue commands to the device or does the app issue commands to the device? How does data move through the system and from where does it originate? These questions and more were asked.

   Resolution – The resolution was a white board and sanity checks. Each of the classes went through several iterations in terms of attributes, methods, and intention. Some were scrapped entirely. For instance, a *MotorPool* class once existed to indicate the collection of vehicles at the disposal of an *Administrator*. This was scrapped in favor of the *signalAvailableVehicle* and *viewManagedVehicle* methods within *ManagedVehicle* and *Administrator*, respectively, thus eliminating an additional relationship.

3. Visibility.

   Issue – Public, private, and protected attributes and methods – what do they mean? Certain elements should be accessible across relationship chains, such as user's mobile app needing to know which vehicle to control. Others should not be accessible, such as user's account passwords.

Resolution – Internet research on the meanings of visibility with UML. The “rule-of-thumb" was attributes and methods should be public unless there was specific reason to make them otherwise.

4. Relationships.

Issue – In class, we were discouraged from using any relationship other than association. But given the problem at hand, it was deemed unlikely that associations alone would provide an accurate enough concept of how the classes are related.

Resolution – Internet research on the meanings of UML class relationships. Given that a vehicle without an RRD could not be controlled, it was determined that the <u>*RoughRiderDevice*</u> was a dependency. Generalizations were introduced where it could be said with high confidence that they fit criteria for commonality between classes or that certain attributes should not be accessible outside of the system-facing nodes.

5. Security.

Issue – The current version relies merely on a username (email address) and password to become a user for the service. This creates potential security concerns. For instance, if a ne’er-do-well were to hack a user’s account, not only would the account be compromised but potentially the vehicle as well.

Resolution – At this stage of development, the team chose to focus on creating a minimum viable product. The explicit goal was to develop a system that allows users remote control over certain functions of vehicles. The project instructions do not state this must be done securely. However, security considerations can be made during future phases of development (i.e., once the primary concept has been approved and if security considerations should even be incorporated into this project). Some of possible resolutions:

1. Enforcing two-factor authentication for log-ins.

2. The administrator issues a code when creating a control node, which must be verified by the gaining user.
3. The administrator issues an NFC signal from his/her phone, which must be received by the gaining user’s phone, promoting physical proximity to known persons.

6. Interface Drawings

Issue – Coming up with a design for a brand-new app was challenging, especially as there were so many options and requirements that needed to be incorporated.

Resolution – Looking at how other mobile apps were organized for inspiration made coming up with a design easier as we saw how different aspects could be separated into different categories or subtasks within the app.

Future Goal Statements

The first future goal would be to listen to feedback about our project and to implement changes that are recommended. These changes could help improve our UI physically by making it more appealing or easier for users to understand. The changes could also refer to our workflow and logic of how the app should operate too.

The next goal would be to start coding each of the screens that are shown in the interface drawings. Just coding how they will appear without any functionality will help bring forward other ideas or concerns that we had not thought about before.

After that, the next goal would be to actually implementing some functionality into the app. For example, showing the functionality of adding or removing a vehicle, or switching pages between the ‘garage’ page and the ‘vehicle’ page would be a good start.

We also will need to implement additional functionalities that were not fully displayed without user interface drawings, such as GPS functionality, geofencing, etc.