University of Arkansas – CSCE Department
Capstone I – Final Proposal – Fall 2022

General Motors - Vehicle to Grid Energy Buy Back

Julia Brixey, Benjamin Worthington, Chiyou Vang, David Hammons, Joseph Taylor

Abstract

General Motors has expressed a commitment to put drivers in electric vehicles on an unprecedented scale. This new prevalence of electric vehicles in our society has resulted in a record demand for energy and an increase in unused energy stored in these electric vehicles. However, there is no system in place to allow the sharing of electric vehicle energy with the grid. In order to solve this problem, we are designing and creating a mobile iOS application that will provide customers with the opportunity to sell electric vehicle energy back to the grid.

Our application will serve as an interface between electric vehicle owners and the grid. It will allow users to set up and manage their accounts, provide notification of grid needs, locate nearby charging stations, and will manage monetary transactions on the user's behalf.

1.0 Problem

With such a huge demand for electricity in our modern world, there is an unprecedented strain on global power grids. When demand is much greater than supply, the power grid may not be able to keep up. In worst-case scenarios, this can lead to brownouts and blackouts. Many states have already begun experiencing these problems, an example being the recent winter storm in Texas during the winter of 2021. In these cases, it is crucial to obtain access to energy as quickly as possible. In times of need, electric vehicle owners may be able to provide supplemental power to the grid by selling back their stored energy. This will help to alleviate the strain while also generating some income for the user.

Currently, General Motors has no system in place to facilitate the sharing of electric vehicle energy with the grid. This leaves electric vehicle owners with no way to sell their stored energy and regions in the grid with no way to tap into this great potential power source.

2.0 Objective

The objective of this project is to create an iOS application that will facilitate the transfer of energy between General Motors customers and the power grid. Our solution will allow users to set up accounts, update their information, locate charging stations, receive notifications about grid areas of need, and set up a payment system for energy transfers. In doing so, we will produce a platform that allows electric vehicle owners to sell their stored energy back to the grid, and provide regions in the grid with access to this potential power source.

3.0 Background

3.1 Key Concepts

The core technology involved with this problem is the electric vehicle itself. Standard all-electric vehicles (EVs) utilize large batteries to power an electric motor (as opposed to a gasoline engine). Some EVs have batteries that allow for bidirectional energy transfer, which is necessary to be able to transfer energy back to the grid. When fully charged, some of these cars can store enough energy to allow them to travel for more than 300 miles [1]. Benefits of electric vehicles, other than not paying for gasoline, include lower maintenance and reduced emissions. EVs can be charged at home or at public charging stations. These public stations can be free or charge drivers a slight cost for their use.

Some charging stations currently implement Vehicle-to-Grid (V2G) technology, which is essential to implementing a solution to our problem. The V2G platform communicates with the power grid to assess needs and encourages drivers to charge their vehicles when demand is “off-peak”. It also allows the car batteries of bidirectional EVs to transfer energy back to the grid in situations where the need is high [2]. General Motors (and several other companies) envision incentivizing drivers to use the V2G system by allowing them to “sell” their unused energy back to the grid.

This transaction of payment in exchange for energy is where our application will come in. Users with a compatible General Motors EV will be able to create an account on our application, and we will store all relevant user data in a devoted database. Our application will then communicate with the vehicle to receive information about its real-time charge levels and other essential metrics. Data will be gathered from location services in order to show users where nearby charging stations are located. We will also gather data on the charging stations themselves and show the user the type of charging available if the station is equipped with V2G technology, and current prices (for charging and vehicle-to-grid transactions). In both cases mentioned, we will utilize APIs to gather data. If a user decides to utilize the V2G functionality and transfer energy from their EV to the grid, we will connect with the payment service (also through APIs) in order to facilitate the necessary transactions.

3.2 Related Work

The idea of customers selling unused energy back to power grids is not a new one. For example, many homeowners with solar-powered homes will remain connected to the main power grid in case of emergencies. When their homes produce an excess of energy, these homeowners are often given the option by utility companies to sell their energy back to the grid. Most of these transactions happen automatically, and any excess energy produced by the home solar panels is immediately sent back to the grid without user interference [3]. The homeowners will simply receive a payment for all transactions within a certain period.

Recently, companies have considered equipping electric vehicles with the technology necessary to do the same energy transfer back to the grid. After the 2011 Tohoku Earthquake and Tsunami and Fukushima Daiichi nuclear disaster in Japan, there was a desperate and immediate need for energy. At that time, Nissan stepped up to provide an emergency source of energy using 66 of their "Leaf" model electric cars [4]. This was an early use of the V2G technology, and at the time, the Nissan Leaf was one of the only cars in production able to transfer energy in both directions.

From 2014 to 2018, the Los Angeles Air Force V2G Demonstration Project was carried out [5]. It was designed to test the feasibility of using a fleet of electric vehicles as an energy storage resource for the base buildings, known as vehicle-to-building. In the project, 42 vehicles were replaced with all-electric or plug-in hybrid electric vehicles. Twenty-nine of these electric vehicles were outfitted with bi-directional capability, meaning that they could be tapped as an energy resource when not being used for transportation. The fleet helped to maintain a stable system frequency, supporting grid reliability. The fleet also provided load shifting for time-of-use electric utility cost management, as well as demand response. In their report, they mention that "The full Los Angeles Air Force Base electric vehicle fleet could have provided emergency backup power to the base's emergency operations center for approximately 80 hours…" (Black, Douglas, Jason MacDonald, Nicholas DeForest, and Christoph Gehbauer, 2017, pg. 4). This effectively demonstrated that electric vehicles can be used as an energy storage resource for supplemental grid power. This project represented a significant step forward in the development of V2G technology and serves as a great example of how electric vehicles can be used to aid in an energy crisis.

This year New York City launched its first V2G on its electricity grid. As of right now the rideshare network Nissan's LEAF is the only EV that is compatible with the V2G system. The three companies that are involved are Revel rideshare, Fermata energy, and Ninedot energy [6]. This brings us to our mission of helping General Motors create its vision of V2G via a user-friendly application, which will lay the groundwork for pushing forward the next generation of V2G technology.

Our application will fundamentally differ from these previous implementations of V2G technology in several key areas. Most notably, we will be allowing drivers to share energy back to the grid at their own convenience, and not at the need of the grid. Some users may decide to utilize the V2G technology frequently while some may never use it. We will also be providing payment to drivers for their shared energy, which is not something that has been previously done on a commercial scale. This system will be in place permanently, not specifically in cases of extreme grid need or emergency, and will ideally uniformly decrease grid need as a whole.

4.0 Approach/Design

4.1 Requirements and/or Use Cases and/or Design Goals

Our application will be designed to allow the user to

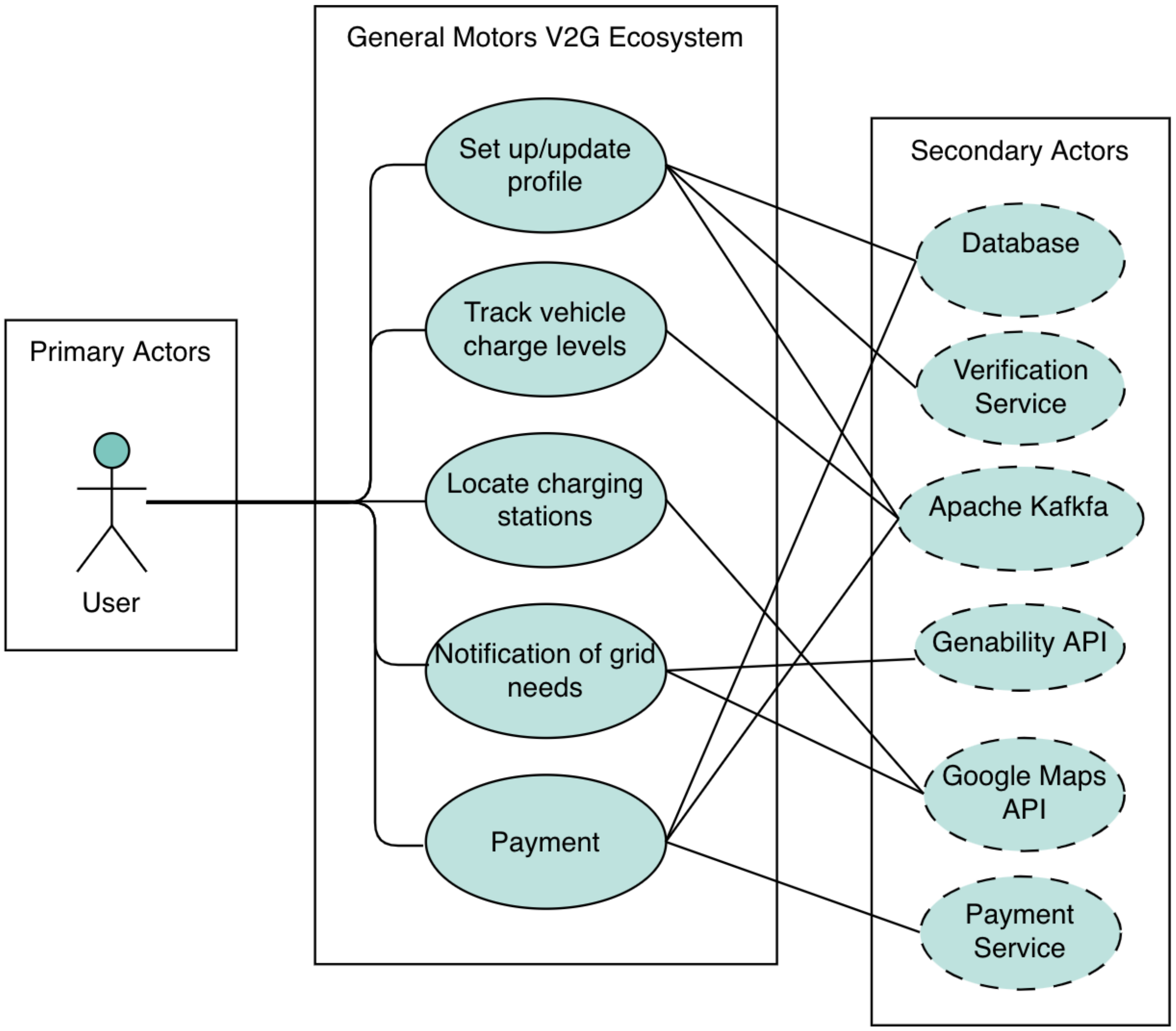
- Set up an account
- Update their information (including payment information)
- Locate charging stations
- Receive notifications about grid areas of need
- Set up a payment system for energy transfers
- Stretch Goal: Determine route based on fast charger location

To satisfy these design goals, we will implement several support actors to fulfill a variety of functions. These “secondary actors” will be essential in ensuring a smooth and user-friendly design. The supportive secondary actors are as follows:

- Database
  - User data
  - Vehicle data
  - Payment data
- Verification Service
  - Set up/store user authentication for accounts
  - Firebase
- Communication Service(s)
  - Communicate with the user’s vehicle
    - Via Apache Kafka client
  - Communicate with the energy grid/charging stations
    - Via Genability API
- Location Service
  - Show user geographical data for ease of locating charging stations
  - Google Maps API
- Payment Service
  - Establish a transfer of payment from the grid account (whoever will be purchasing the user’s energy) to the user’s account
  - Stripe API

Vehicle to Grid Energy Buy Back Application

<u>Use Case Diagram:</u>



Use Case: Set Up/Update Profile
Author: Julia Brixey
Primary Actor: User
Goal in context: Set up a user profile and populate it with relevant user information
Preconditions: Authentication service in place to allow users to create passwords or set up two-factor authentication, database in place so this authentication data and other information can be stored
Trigger: User "creates an account" or requests to edit existing account information

Scenario:

1. Prompt GUI: Loads prompts for inputting user information
2. User: Inputs the requested user information
3. User: Clicks the "save" or "update" button
4. Verification Service (in the case of new account creation):
    a. Verifies the user's passwords match AND
    b. Verifies two-factor authentication AND
    c. Verifies the user's input vehicle information corresponds with a valid General Motors electric vehicle
        i. Communication Service (Kafka): Establishes communication with this vehicle via Kafka client
5. Application: Sends the user information to the database
6. Database: Saves the information

Exceptions:

1. Remote database unavailable. The user will be alerted, and the data manager will be prompted to try to reconnect.
2. User does not have a General Motors electric vehicle. The user will be informed that they must have a valid vehicle to access vehicle information.

Priority: High
Channel to Actor: Graphical User Interface (GUI)
Usage Frequency: One-time (account creation) or as often as needed (updating profile information)
Secondary Actors: Database, Verification-Service, Communication Service

Channels to secondary actors:

Database:	Network
Verification Service (Firebase):	Network
Communication Service (Kafka):	Network

Open Issues:

1. Do users need to have the vehicle registered in their name? Will there be a database of vehicles and specific vehicle owners?
2. Should users be able to create an account without a General Motors vehicle?

Use Case: Track Vehicle Charge Levels
Author: Julia Brixey
Primary Actor: User
Goal in Context: Track the current level of charge in the user's vehicle in real-time
Preconditions: The user's profile is set up with a valid vehicle in the system. The vehicle's system was able to successfully connect to the application.
Trigger: The user looks to see their vehicle's charge level on the application.

Scenario:

1. User: Selects button on the application that indicates they want to see the charge levels of their vehicle / Logs in to the application and pulls up "home" page
2. Application: Pulls up the screen where vehicle charge data is to be displayed
3. Application: Requests vehicle charge data from communication service
4. Communication Service (Kafka): Communicates with vehicles using Kafka to determine the current level of charge. Returns this data to application
5. Application: Displays updated data

Exceptions:

1. The communication service is unable to communicate with the vehicle. It will return an error message. Upon receiving this error message, the application will alert the user of the issue. Communication service will continue to try to connect with vehicles.

Priority: High
Channel to Actor: Graphical User Interface (GUI)
Usage Frequency: As often as needed by user
Secondary Actors: Communication Service

Channels to secondary actors:

Communication Service (Kafka):	Network

Open Issues:

1. Should the active charge level be continuously updated in the application? Or should it only be updated upon prompting by the user?

Use Case: Locate Charging Stations
Author: Julia Brixey
Primary Actor: User
Goal in context: Utilize the application to quickly locate nearby charging stations compatible with their electric vehicle.
Preconditions: The user's profile is set up with a valid vehicle in the system. Location services are enabled on their application. The device in use can utilize location services.
Trigger: The user looks to see the location of compatible electric vehicle charging stations.

Scenario:

1. User: Selects button on the application that indicates that they want to view a map of compatible electric vehicle charging stations
2. Application: Loads map page and requests geographical data from location service
3. Location Service (Google Maps API): Loads geographical data based on current device location
4. Application: Requests charging station data from the API based on proximity to the user's current location (EX: all charging stations within 30 miles)
5. Location Service (Google Maps API): Returns charging station data to application
6. Application: Updates map with the addition of charging stations
7. GUI:
   a. Allows the user to select a charging station to see more info
   b. Allows the user to input a custom location or range to see charging stations within
8. Location Service (Google Maps API): Updates location information as necessary
9. Application: Requests charging station information from the API as necessary when location or range is updated

Exceptions:

1. Location services cannot be contacted. An error message is returned, and the application will alert the user of this issue. The application will continue to try to connect with the location service.

Priority: High
Channel to Actor: Graphical User Interface (GUI)
Usage Frequency: As often as needed by user
Secondary Actors: Location Service
Channels to secondary actors:
Location Service: Network

Open issues:

1. Will the application be able to recommend certain charging stations based on the current level of charge (optimal routes)?
2. Will Google Maps API be able to return all of the information about the charging stations we need to show the users?

Use Case: Notification of Grid Needs
Author: Julia Brixey
Primary Actor: User
Goal in context: Notify the user of active grid energy needs.
Preconditions: The user's profile is set up with a valid vehicle in the system. Location services are enabled on their application. The device in use can utilize location services.
Trigger: The application receives an alert about the energy needs of the grid.

Scenario:

1. User: Indicates on the application that they would like to see grid areas of need OR
2. Application: Periodically requests grid need information (on a timer)
3. Communication Service (Genability API): Gathers information from the grid regarding the need for extra energy
4. Application: Alerts the user of areas where the need is high via notification if the user is within a certain range of the grid need location AND/OR commences updates on the geographical map on the application through a prompt to location services
5. Location Services: Loads geographical data within a range of grid needs and loads locations of energy "give-back" stations within the range of need. Returns data to application
6. Application: Updates map

Exceptions:

1. Location services cannot be contacted. An error message is returned, and the application will alert the user of this issue. The application will continue to try to connect with the location service.
2. Communication service is unable to communicate with the grid to alert the user of the need. Communication service will continue to try to connect with the grid, the user will be notified of disconnection.

Priority: High
Channel to Actor: Graphical User Interface (GUI)
Usage Frequency: As many times as needed by grid need or based on user notification preferences
Secondary Actors: Communication Service, Location Service
Channels to secondary actors:

Communication Service (Genability API):	Network
Location Service:	Network

Open issues:

1. Do we want to notify only users with excess energy in their vehicles?
2. Research the functionality of giving energy back to the grid, does it matter which charging/"give-back" station is used? Does it have to be within that grid range?
3. Will there be the opportunity for users to transfer energy from the vehicle to the grid when there is no need?

Use Case: Payment
Author: Julia Brixey
Primary Actor: Application
Goal in Context: Provide users with payment for their energy transfer.
Preconditions: The user's profile is set up with a valid vehicle in the system. The user was able to successfully transfer energy back to the grid at a valid station. User payment preferences are valid.
Trigger: The user has transferred energy from their electric vehicle back to the grid.

Scenario:

1. User: Drives their electric vehicle to the charging ("give-back") station, and transfers energy back to the grid
2. Communication Service (Kafka/Genability API): Receives notification from vehicle/grid of energy transfer and sends notice to application
3. Application: Displays payment page and shows the user the amount of the payment they are to receive for this transfer
4. Database: Accesses and provides payment service with the user's payment information
5. Payment Service (Stripe API): Facilitates the transfer of the correct amount of money from the grid account to the user-specified account
6. Communication Service: Receives alert of this transfer from grid account and sends the user a receipt of payment (via application or directly to user email)

Exceptions:

1. Grid payment account is not set up or is unable to be accessed. The user will be notified of this issue and payment will be delayed until the account is accessible.
2. User payment account is unable to be accessed. The user will be notified of this issue and payment will be delayed until the account is accessible.
3. Communication service does not receive notification of transfer. The user will have to manually request payment based on grid records.

Priority: High
Channel to Actor: Graphical User Interface (GUI)
Usage Frequency: As many times as needed based on user energy transfer activity
Secondary Actors: Payment Service, Communication Service, Database
Channels to secondary actors:

Communication Service (Kafka/Genability API):	Network
Payment Service (Stripe API):	Network
Database:	Network

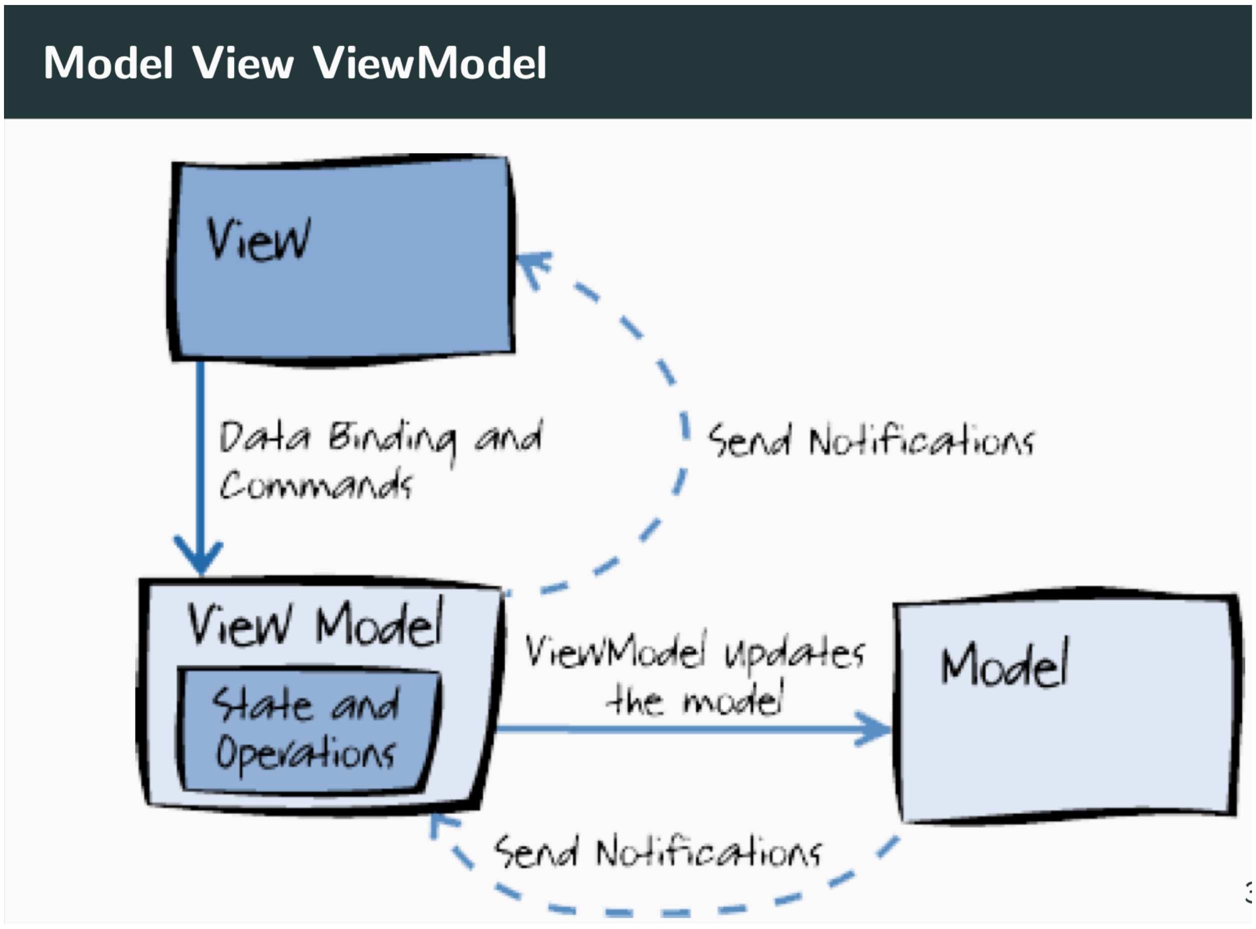
Open Issues:

1. Who is paying the user for their energy transfer? Will we need extra verification to set up the channel between the grid and the user?
2. How will we establish a secure transaction?

4.2 Detailed Architecture

Because we will be developing an iOS application, the architecture structure we are going with is Model View ViewModel (MVVM). This is more work upfront but for longevity, this will make it easier to scale, separate dependencies of different components, and allows for easier implementation of unit tests.

The idea of MVVM architecture is based around funneling communication between the Model and the View through a ViewModel object. When the model updates, the ViewModel is responsible for letting the View know. When someone interacts with the View, the ViewModel is in turn responsible for communicating those changes to the Model. This approach allows our objects that exist within the View to be built separately from the Model, as they will only need to communicate with the ViewModel. This allows us to use mock data to speed up UI development, regardless of whether the actual Model functionality is there yet. On the other side, this also allows the model to be developed and tested independently of the View. Ultimately, MVVM brings with it an organizational structure that will keep our application clean, testable, and adaptable to change.

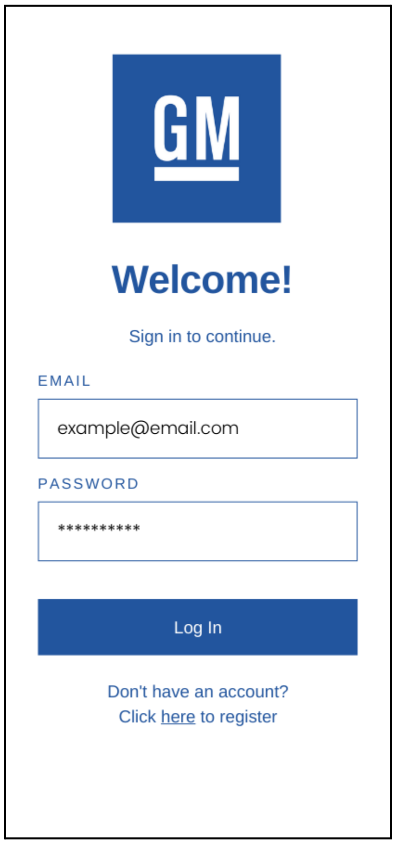


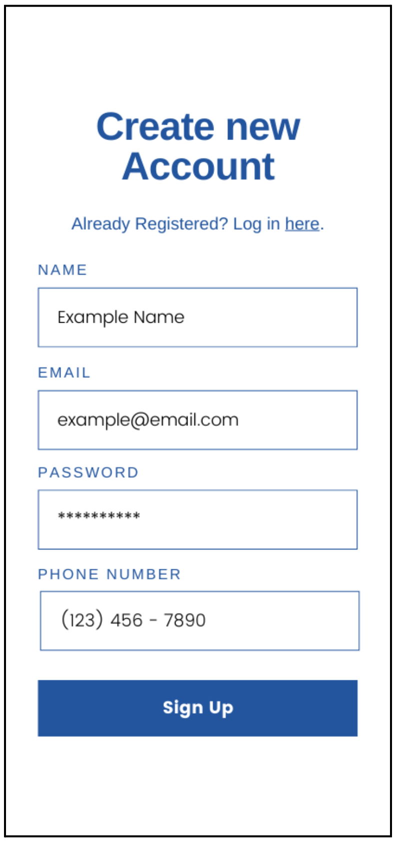
To support the functionality of our application we will utilize several APIs. To gather the location data necessary to show our users an interactive map, we will use the Google Maps API. We will also use the Google Maps API to show our users the location of nearby charging stations and stations that support vehicle-to-grid transactions, as well as relevant information about these stations. We will utilize the Genability API to gather information relating to the power grid, and use this to calculate real-time grid needs and historic trends of grid demand. The payments will all be facilitated through the Stripe API and we will also gather transaction history information through this API. Communication with the user's vehicle(s) will be established through Apache Kafka, and we will use information pulled from Kafka to determine the vehicle's current charge levels. The backend of our application will be built using XCode, Swift, and Firebase.

Login/Create Account

To utilize our app, users will have to first create an account. When the app is opened, users will be prompted to either log in to an existing account or create a new one. Authentication of each account will be completed through Firebase. Firebase itself handles the storage of user passwords and emails, which ensures that a breach of our application would not result in a breach of user passwords.

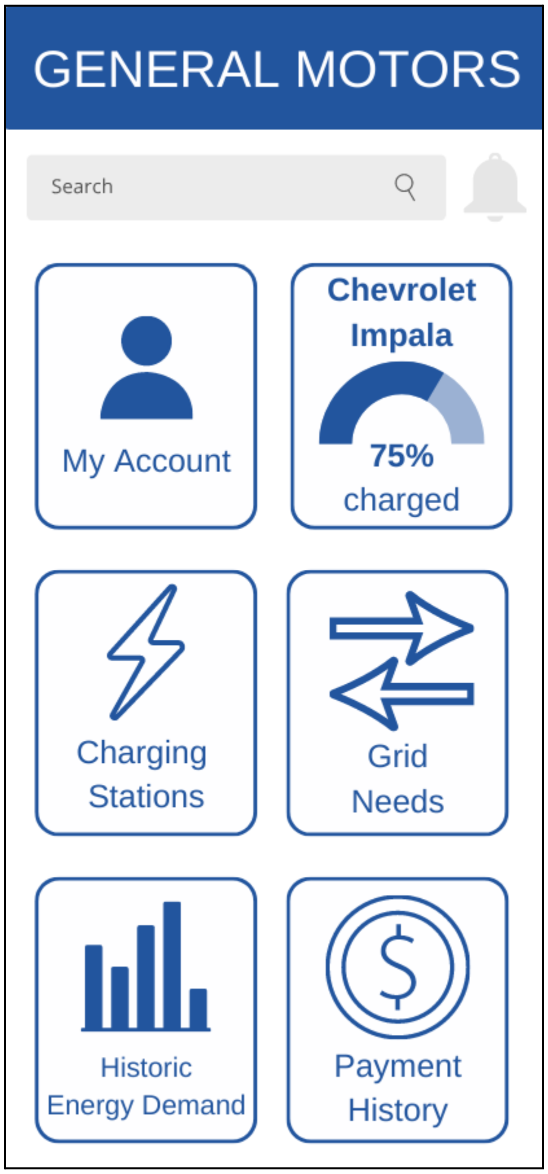
Our MongoDB database will store the user's username, phone number, and id. This id will be associated with the unique id (UUID) of a user stored in the Firebase database, allowing us to interact with the user and their information without having to ever touch their password or email in our application.





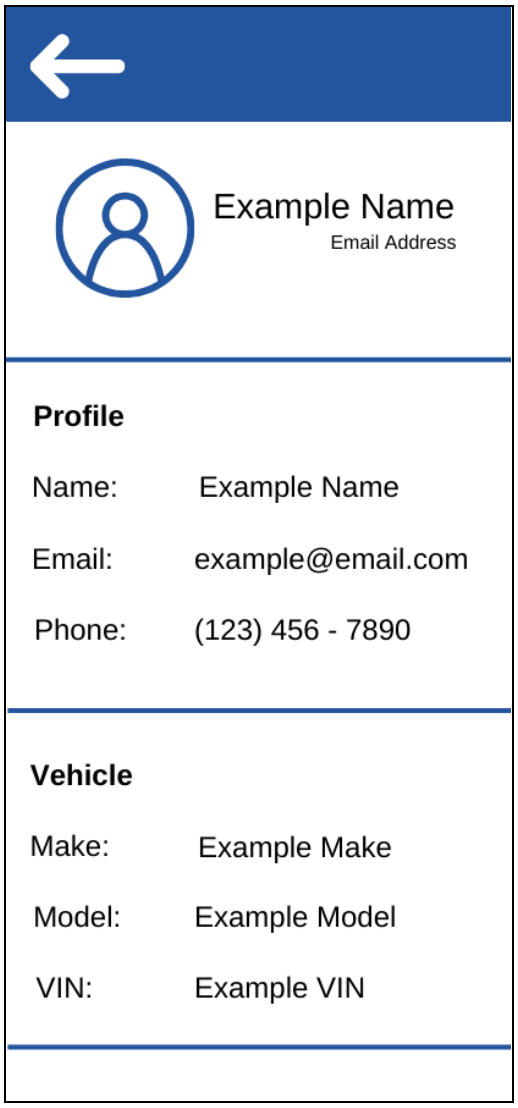
Home

Once the user is fully logged in they will have access to the 'Home' screen. From here, all of the application's functionality is accessible. They will be able to view and modify account details, view their vehicle information and charge level, view a map of nearby charging stations and their corresponding information, view a map of grid needs, view historic energy demand, and access their payment history. Notifications will also be visible to the user in this section as well, accessible through the bell icon in the upper right-hand corner. In this implementation, we intend for historic energy demand to be periodically updated on the home screen to provide users with a helpful metric in visualizing when the best time may be to sell their energy. The data necessary to create this visualization will be gathered through the Genability API.



My Account

Below is a mockup of the ‘My Account’ page. The name, email, and phone number associated with the account will all be visible here, along with any associated vehicle(s) information. Upon scrolling down, the user will also be presented with historic payment information. We will collect the user’s vehicle(s) make, model, VIN, the name of their bank, the routing number, and receiving account number. User vehicle and payment information will be stored in our MongoDB database. There will also exist two buttons that allow the user to (a) update any information on the page, or (b) log out of the application.

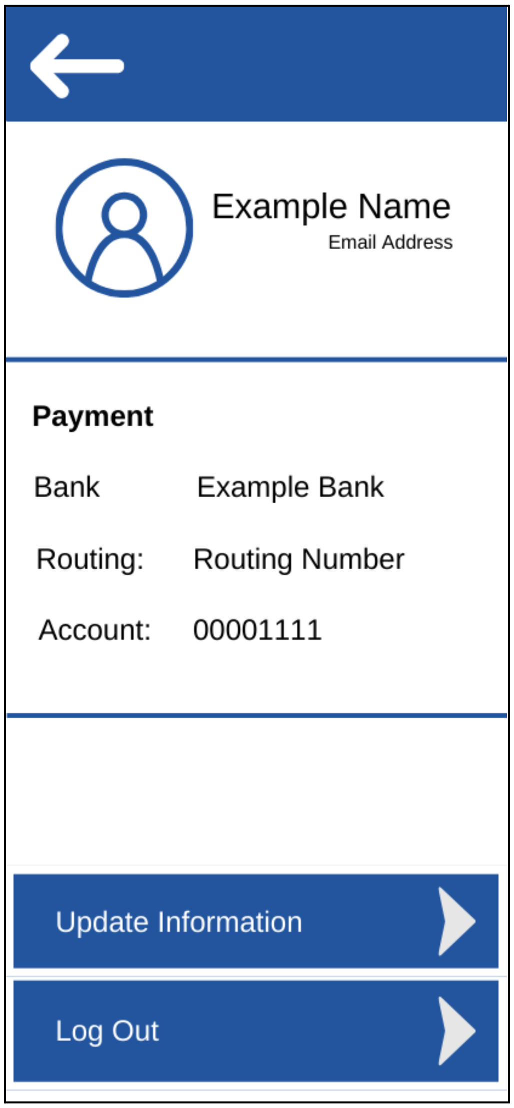

Example Name
Email Address

Profile

Name:	Example Name
Email:	example@email.com
Phone:	(123) 456 - 7890

Vehicle

Make:	Example Make
Model:	Example Model
VIN:	Example VIN

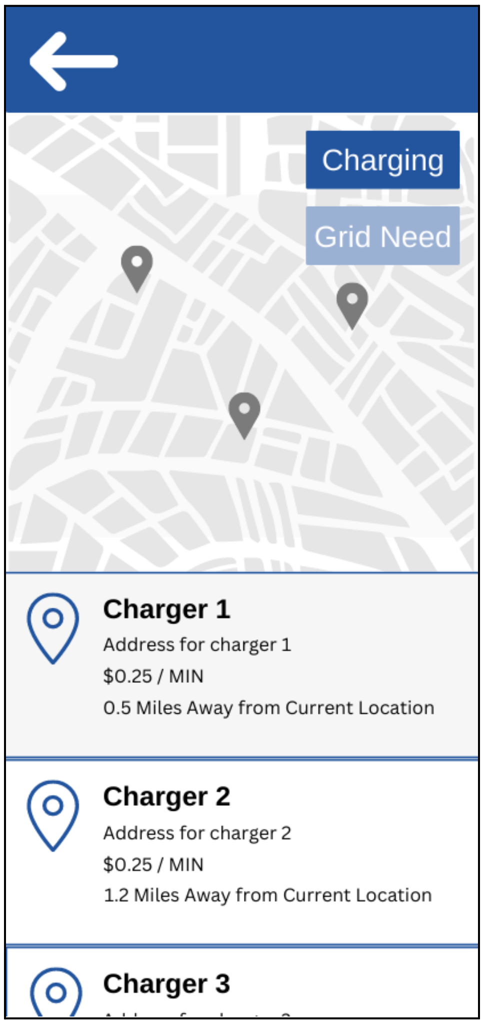

Example Name
Email Address

Payment

Bank	Example Bank
Routing:	Routing Number
Account:	00001111

Charging Stations

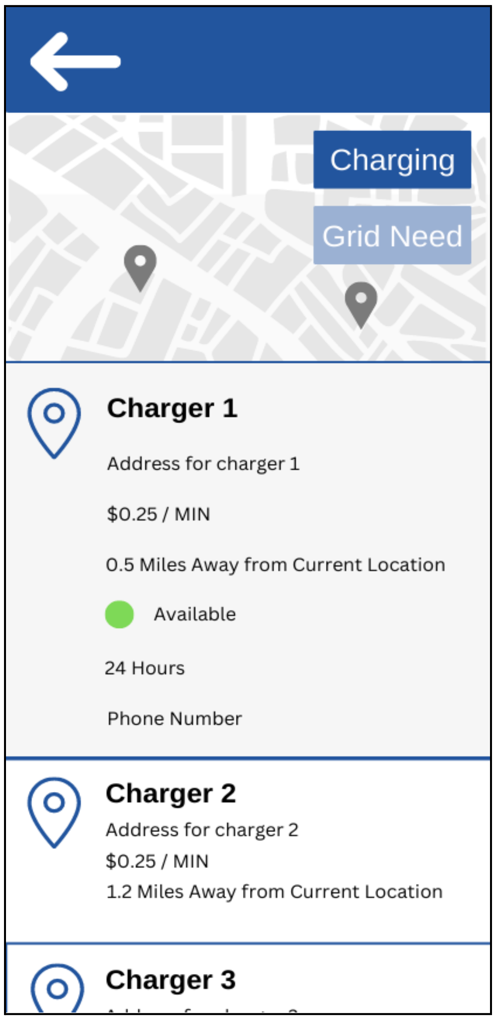
Next, we see the ‘Charging Stations’ screen. This screen shows a map of all nearby chargers, along with their distance from the user and the price at that location. Upon selecting a station on the map by tapping on the marker, more detailed information is displayed. This detailed information will provide the availability, hours, and phone number associated with the charging station. All information displayed on the map and related to the charging stations will be gathered using the Google Maps API.



Charger 1
Address for charger 1
$0.25 / MIN
0.5 Miles Away from Current Location

Charger 2
Address for charger 2
$0.25 / MIN
1.2 Miles Away from Current Location

Charger 3



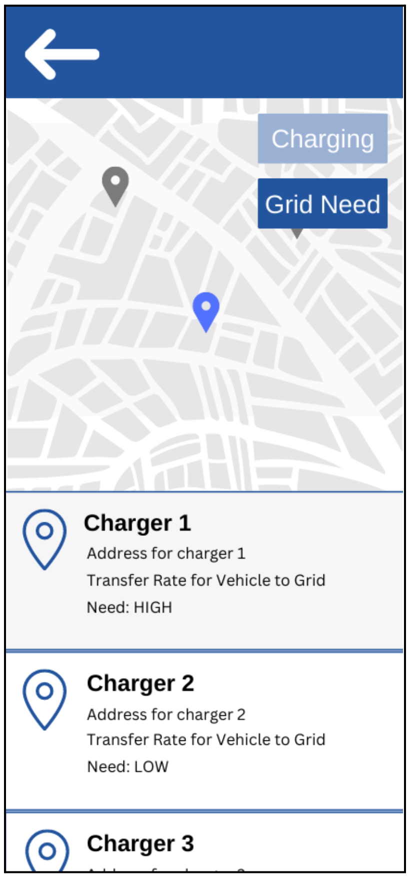
Charger 1
Address for charger 1
$0.25 / MIN
0.5 Miles Away from Current Location
● Available
24 Hours
Phone Number

Charger 2
Address for charger 2
$0.25 / MIN
1.2 Miles Away from Current Location

Charger 3

Grid Need

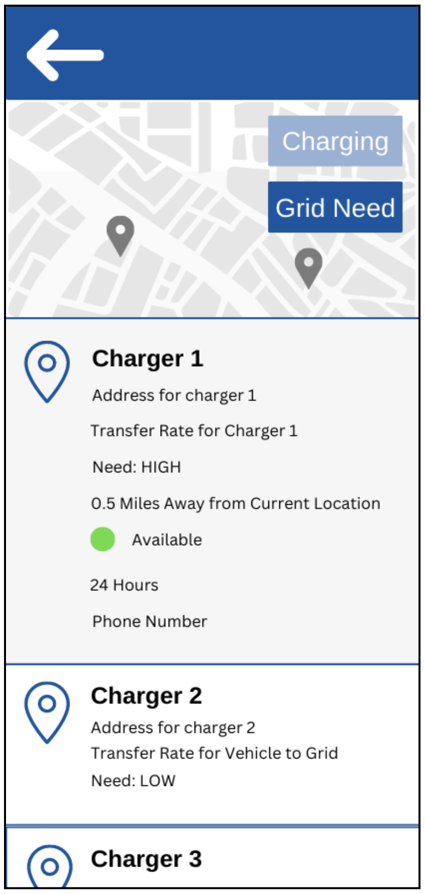
The next screen is that of ‘Grid Need’, which is very similar (and can be accessed from) the ‘Charging Stations’ screen. This screen also consists primarily of a map, but instead of displaying charging stations where the vehicle may be charged, the ‘Grid Need’ screen is responsible for marking stations where power may be resold by the user. This screen will also show the level of need at these stations, along with the same detailed information as in the ‘Charging Stations’ screen. The level of need at each station will be determined by utilizing data gathered through the Genability API. Like the ‘Charging Stations’ screen, all information on this page will be gathered from the Google Maps API.



Charger 1
Address for charger 1
Transfer Rate for Vehicle to Grid
Need: HIGH

Charger 2
Address for charger 2
Transfer Rate for Vehicle to Grid
Need: LOW

Charger 3



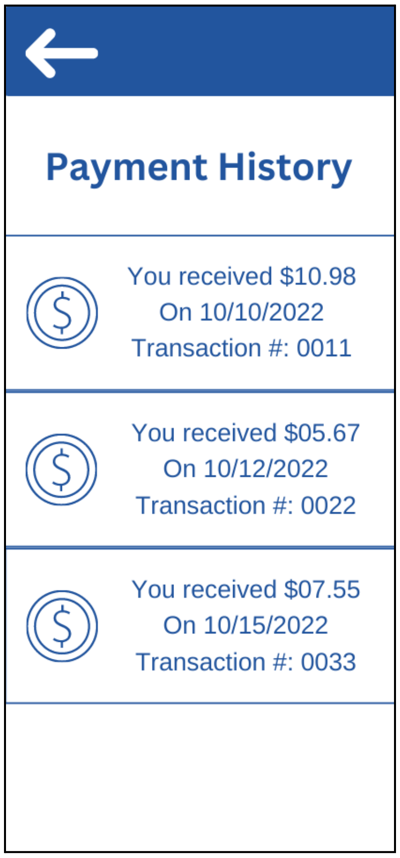
Charger 1
Address for charger 1
Transfer Rate for Charger 1
Need: HIGH
0.5 Miles Away from Current Location
● Available
24 Hours
Phone Number

Charger 2
Address for charger 2
Transfer Rate for Vehicle to Grid
Need: LOW

Charger 3

Payment History

Finally, we have the "Payment History" screen. This is a very simple screen and is simply responsible for displaying a scrolling list of payments received by the user through our app. Each payment will show the total received amount, the date of the payment, and the transaction number. Payment transaction history will be accessed through the utility of the Stripe payment API. This is also how transactions will be facilitated.



Payment History

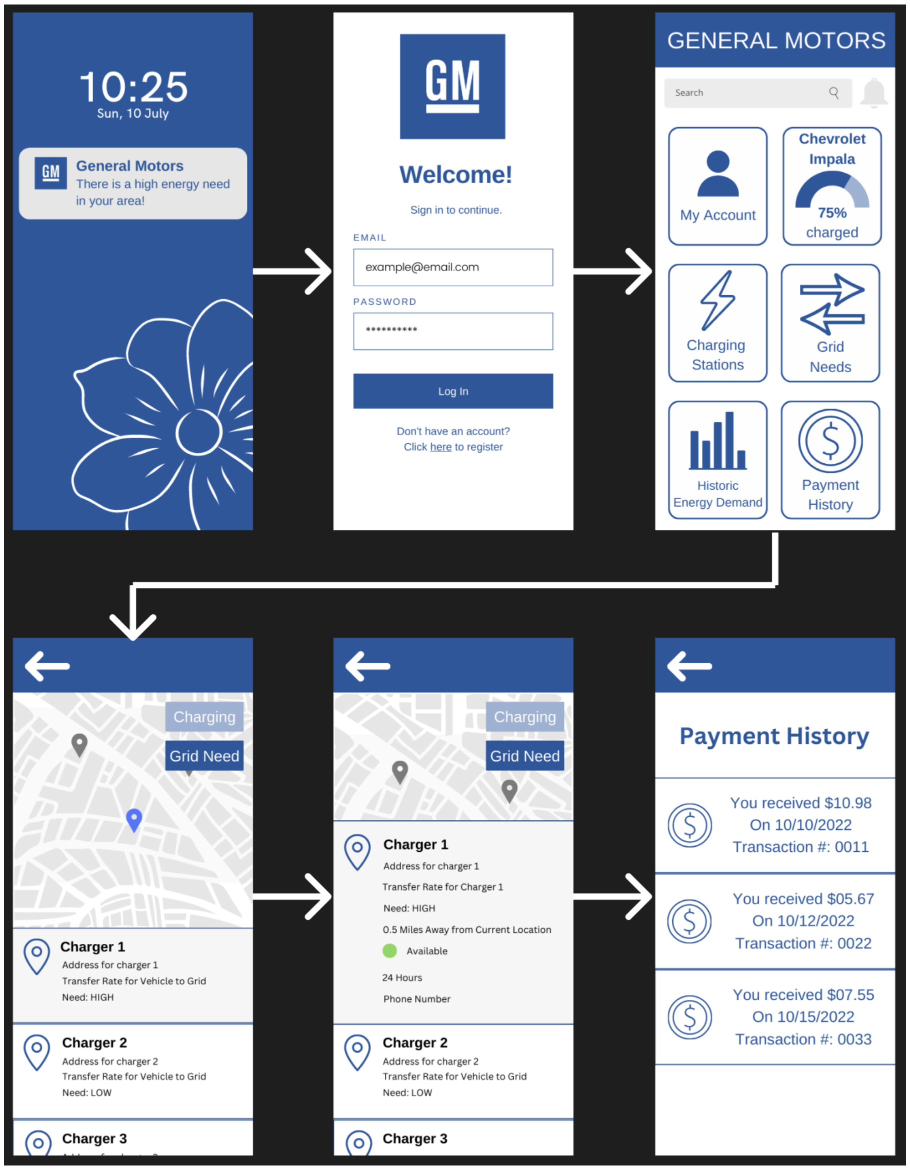
You received $10.98
On 10/10/2022
Transaction #: 0011

You received $05.67
On 10/12/2022
Transaction #: 0022

You received $07.55
On 10/15/2022
Transaction #: 0033

Example Use Case

Below is an example of how a user will utilize our application. The user will receive a notification that there is a high grid need in their area. Grid need is calculated using information from the Genability API, and the notification will be sent to all users within a certain range of the area of grid need (assuming the user has notifications enabled and has allowed their application to share their location with us). The user logs in to their app and navigates to the grid needs page. They then select a station nearby in high demand for energy to view more information. The user will then travel to this station and transfer some of their excess energy back to the grid. The user can now view a record of this transaction on their application using the 'Payment History' screen.



There is a high energy need in your area!

example@email.com

****

Charger 1
Address for charger 1
Transfer Rate for Vehicle to Grid
Need: HIGH

Charger 2
Address for charger 2
Transfer Rate for Vehicle to Grid
Need: LOW

Charger 1
Address for charger 1
Transfer Rate for Charger 1
Need: HIGH
0.5 Miles Away from Current Location
Available
24 Hours
Phone Number

You received $10.98
On 10/10/2022
Transaction #: 0011

You received $05.67
On 10/12/2022
Transaction #: 0022

You received $07.55
On 10/15/2022
Transaction #: 0033

4.3 Risks

Risk	Risk Reduction
Breach in payment information	Have a two-step authentication and keep sensitive information encrypted.
Location leak	We will only use location service when finding a charging station. Location data will not be stored in any database.
User confidentiality	Keep as much personal information localized and encrypted as possible. Information that is not used does not need to be on any server. User passwords will be stored in the Firebase database to ensure that they will be safe even in the worst-case scenario.

4.4 Tasks

1. **Planning**
   a. Use background and related works to understand the requirements and scope of the application
   b. Gain an understanding and knowledge of Swift, decide on additional frameworks to use if necessary
   c. Become familiar with iOS development
   d. Decide on which location service to use (Google Maps API or the native Maps API)
   e. Understand the needs and plan the schema of the database
   f. Look into cloud database hosting options
   g. Create a schedule to keep progress
2. **Design**
   a. Design database schema
      i. User Profiles
         1. Stored in Firebase
            a. First and Last Name
            b. Password
         2. Energy sold
         3. Phone Number
         4. Email
      ii. Vehicle Data
         1. Make
         2. Model
         3. VIN
      iii. Payment Info
         1. Routing Number
         2. Account Number
         3. Bank
         4. Stripe API

- b. Design a detailed architecture using MVVM of the application where each activity interaction is shown
    - i. Learn MVVM
    - ii. Apply MVVM architecture to our iOS application throughout the building process
- c. Define APIs
    - i. FireStore
    - ii. Google Maps API
    - iii. Firebase Authentication
    - iv. Genability API
    - v. Stripe API
- d. Draw and design UX pages
    - i. Create a user story diagram

3. **Development**
    - a. Backend
        - i. Set up a local database with MongoDB in Swift for ease of development
        - ii. Set up a cloud database for production
            1. Google Firestore
            2. Implement database schema
            3. Connect the database to the Swift application
        - iii. Asking for and keeping track of the user’s location
            1. Location Services
                - a. GPS
                - b. Wireless/Cellular
                - c. Geocoding
        - iv. Payment functionality and Bank Information
            1. Authentication
            2. Retrieving bank information and balance with Stripe API
            3. Calculates the balance of the user’s current sale of their electricity
        - v. Connecting to vehicles and their charge level
            1. Connect to the user’s car API to get charge level
        - vi. Keeping track of charging stations
            1. Communicates with Google Maps API to locate charging stations
        - vii. Connect to power grid
            1. Establish communication APIs with the power grid in order to track needs
        - viii. Notifications
            1. Set up using Swift and iOS notification alarm scheduler
        - ix. Implement encryption and security
            1. User Profiles
                - a. Authentication
            2. Payment Process Security
            3. Database Protection
                - a. SQL Injection
    - b. Frontend
        - i. Implement UX pages

        1. Login
            a. Beginning screen the user is shown when the user opens the application
        2. Creating an account
            a. Accounts are saved into the database backend through user input
            b. Single-sign-on such as Gmail, Facebook, etc …
        3. Main
            a. The main screen to help navigate to the other screens of the application after logging in.
        4. Find Charging Stations
            a. Implemented with Google Maps API
        5. See Areas of Grid Need
            a. Implemented with Google Maps API and Genability API
        6. Payment
            a. Lets the user make a sale of electricity with the bank of their choice
            b. Stripe API
        7. Vehicle Charge Levels
            a. Lets the user see the current charge of their vehicle
            b. Apache Kafka
        8. Update Profile
            a. User customization
            b. User profile details
            c. Updated in database
        9. Bank Info/Account Balance
            a. View balance and banking information
    ii. Connecting backend and frontend
        1. Every user input should be implemented and handled correctly and securely by the backend
    iii. Notifications
        1. Shows if the sale went through successfully
        2. Shows how much charge is left in the vehicle
        3. Notifies user of nearby grid needs

4. Testing and Documentation

a. Test database
b. Test backend/frontend interactions and ensure functionality
c. Test communication with vehicles and grid
d. Test location service accuracy
e. Test user-interface visibility and readability on various platforms
f. Test performance issues and fix them as needed
    i. Battery Drain
    ii. CPU/Ram Usage
g. Survey for feedback on user interaction
h. Document issues, fixes, and design choices

4.5 Schedule

Tasks	Dates
1. Background Investigation a. Research and familiarize ourselves with Swift and XCode. Finalize framework/languages to be used. b. Look into location service options. Google Maps API research c. Research database options	10/24-11/2
2. Design Database Schema a. User data b. User vehicle information c. Payment information	11/2-11/9
3. Design UI a. Define specific pages b. Create a user story diagram	11/9-11/18
4. Design/Define APIs	11/18-11/25
5. Finalize Design with GM Team	11/25-11/28
6. Create Databases using FireStore a. User/car info b. Charging station info	1/17-1/30
7. Create UX Pages a. Front end view of all user-facing endpoints via application pages (as defined previously)	1/17-1/30
8. Backend Development a. Backend connections to all APIs established b. Data interpretation established	1/30-2/13
9. Account creation a. Connect front-end account creation page to backend and databases	1/30-2/13
10. Authentication Service Set Up a. Create authentication processes for user account creation and payment systems	2/13-2/27
11. Set up location services a. Keeping track of user location data b. Accessing charging station location data	2/13-2/27
12. Configure Communication Services a. Establish communication with the user's vehicle b. Establish communication with the power grid	2/27-3/31
13. Connect Backend and Frontend	2/27-3/31

14. Set Up User Notifications a. Charge level low b. Nearby grid needs c. Payment went through/receipt	3/31-4/10
15. Testing and Refinement	4/10-5/4

4.6 **Deliverables**

- Design Documentation: Comprehensive list of required hardware and software components, as well as additional plugins and packages necessary to install for application use.
- Database Schema: Information regarding the design of the MongoDB and Firebase databases used, including specific schema (such as username, password, email, car model, etc).
- Website Code: All necessary code for hosting the website containing project information.
- Mobile Application Code: Swift code for front-end application (to be run on XCode).
- Backend Code: Swift code for the backend of our mobile application (as well as other languages if necessary)
- Final Report: Final report detailing the holistic design of our application, including schema, diagrams, and design decisions.

5.0 Key Personnel

Julia Brixey– Julia is a senior Computer Science and Psychology major in the Computer Science and Computer Engineering Department at the University of Arkansas with minors in Mathematics and Data Analytics. She has completed coursework in Software Engineering, Database Management Systems, Programming Paradigms, and Artificial Intelligence. She has worked as a Software Engineer Intern for Walmart Global Technology, and an Intern for start-up companies AMBOTS and Moment AI.

David Hammons- David is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He also is getting his minor in Data Analytics. He has completed relevant coursework in Software Engineering, Database Management Systems, Mobile Programming, and Data mining. He also has work experience in Data Analytics as an intern for Gainwell Technologies.

Joseph Taylor– Joseph is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas, with a minor in Psychology. He has completed relevant coursework in Software Engineering, Database Management Systems, Mobile Programming, and Artificial Intelligence. He has worked as an Application Developer Intern for J.B Hunt Transport since early 2021.

Chiyou Vang- Chiyou is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas, with a minor in Japanese. He has completed relevant coursework in Software Engineering, Database Management Systems, Mobile Programming, and Artificial Intelligence. He is currently assisting in the Computer Vision Lab at the University of Arkansas.

Ben Worthington - Ben is a senior in the Computer Science department at the University of Arkansas. He has completed coursework in Software Engineering, Database Management Systems, Programming Paradigms, and Advanced Data Structures. He is currently taking Cryptography as that is the field he is interested in. He has been working in IT for the University since freshman year and had an internship with Walmart Technologies in his senior year of high school. He wants to go into the Cybersecurity sector with an emphasis on cryptography and networking.

Dr. Matthew Patitz, Professor - Dr. Patitz is an Associate Professor for the Department of Computer Science and Computer Engineering at the University of Arkansas. He received his Bachelor's, Master's, and Ph.D. in Computer Science from Iowa State University. He has received research grants from the National Science Foundation and has published over 60 peer-reviewed conference and journal papers. His research interests are DNA computing, algorithmic self-assembly, and theoretical computer science.

Johnathan Michlik, Software Development Manager - Michlik is a Software Development Manager at General Motors supporting OnStar marketing campaigns and various customer notifications. He joined General Motors in 2016 and has past experience supporting data and analytics projects for Finance, Sales, and Marketing. He graduated from the University of Arkansas Walton college of business in 2011. He will be our primary contact for this project.

Vanessa Black, Telematic Data Asset Manager - Black is a Telematic Data Asset Manager at General Motors. She works on supporting products that provide telematic data to a number of third-party contracts. She joined the General Motors team in 2014. She obtained her Bachelor's in Mathematics in 2011 from the University of Central Oklahoma, and her Master's in Mathematics in 2013 from the University of Arkansas. She will be one of our main contacts for this project.

Garrett Bartlow, Software Developer – Bartlow has been a Software Developer at General Motors for 5 months on the Big Data Infrastructure Enterprise team. He graduated in May 2022 from the University of Arkansas with a Bachelor's in Computer Science and a minor in Mathematics. He has experience in creating and maintaining large-scale databases, application development, large-scale data streaming, and collection. He will be one of our main contacts for this project.

Ally Maumba, Lead Software Developer - Maumba is a Lead Software Developer at General Motors on the Retail Programs and Certification Used Car Marketplace team. He graduated from the University of Arkansas in 2017 with a Master's in Information Systems. He will be one of our main contacts for this project.

6.0 Facilities and Equipment

XCode- IDE

MacBooks- Development of iOS Application

Personal iPhones - Application Testing and Development

Simulated iPhones through XCode - Supplemental Application Testing and Development

Servers - Database and Backend Hosting

7.0 References

[1] *What is an electric vehicle?* Electric For All. (2022, April 12). Retrieved October 20, 2022, from https://www.electricforall.org/what-is-an-electric-car/

[2] EV Connect. (2021, December 20). *What is vehicle-to-grid for electric vehicles?: Ev Connect*. EV Connect. Retrieved October 20, 2022, from https://www.evconnect.com/blog/what-is-vehicle-to-grid-for-electric-vehicles#:~:text=What%20Is%20Vehicle%2Dto%2DGrid%20Technology%3F,cells%20for%20the%20electrical%20grid.

[3] *Selling solar electricity back to the grid: What you need to know*. RocketSolarMasthead-No Descriptor. (n.d.). Retrieved October 20, 2022, from https://www.rocketsolar.com/learn/energy-efficiency/selling-solar-electricity-to-the-grid

[4] Jperez. (2022, June 1). *Vehicle to grid (V2G) technology*. IEEE Innovation at Work. Retrieved October 20, 2022, from https://innovationatwork.ieee.org/vehicle-to-grid-v2g-technology/

[5] Black, Douglas, Jason MacDonald, Nicholas DeForest, and Christoph Gehbauer. Lawrence Berkeley National Laboratory. 2017. *Los Angeles Air Force Base Vehicle-to-Grid Demonstration*. California Energy Commission. Publication Number: CEC-500-2018-025. Retrieved October 23, 2022, from https://www.energy.ca.gov/sites/default/files/2021-06/CEC-500-2018-025.pdf

[6] Shahan, Zachary. “First V2G (Vehicle to Grid) System on Launches NYC Grid.” *CleanTechnica*, 22 Aug. 2022, https://cleantechnica.com/2022/08/21/1st-vehicle-to-grid-system-on-nyc-grid-launches/.