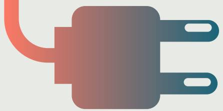
EV Comms

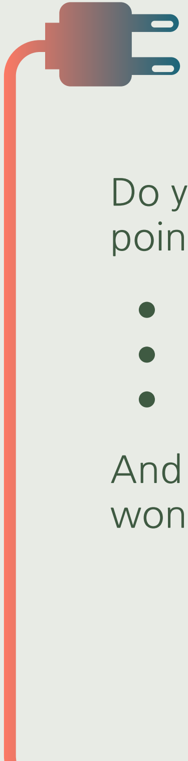
By: Bryce Canillo, James Dao, Aditya Guin, Nam Truong,
Salman Hossain, and Sergei Wong



Overview

1. Objective of the Project
2. Cost Estimation
3. Project Timeline
4. Functional and Non-Functional Requirements
5. Use Case Diagram
6. Sequence Diagrams
7. Class Diagrams
8. Architectural Design
9. Demo of User Interface

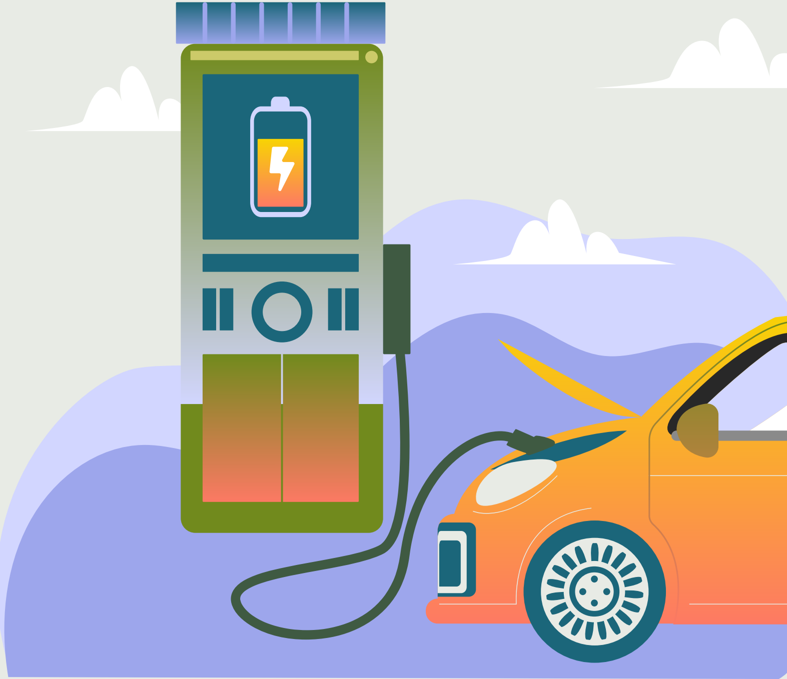
OBJECTIVE OF THE PROJECT



Do you know what helps you make your point clear? Lists like this one:

- They’re simple
- You can organize your ideas clearly
- You’ll never forget to buy milk!

And the most important thing: the audience won’t miss the point of your presentation



COST ESTIMATION

Cloud Computing Services

AWS

- EC2 Instances
- Lambda
- SQS
- Cloudwatch events
- ~$200 - $400/month

Data Analytics

- Confluent
- DataDog
- Splunk
- <$100/month

Other Costs

App Publication

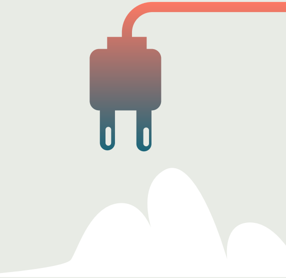
- ~$150

PROJECT TIMELINE

Activities	**Estimated Length of Completion (days)**
Account System	2
Messaging UI	2
Notification System	3
Messaging Functionality(store and retrieve messages from database)	6
Location Tracker and locator algorithm	6
User Input UI	4
Scaling into Distributed Systems	7
Efficiency Improvements	4
Total	**34**

PROJECT TIMELINE

Activity	Start Date	End Date
Account System	November 14, 2022	November 16, 2022
Messaging UI	November 14, 2022	November 16, 2022
User Input UI	November 15, 2022	November 21, 2022
Messaging Functionality	November 21, 2022	November 29, 2022
Location Tracker	November 24, 2022	November 32, 2022
Notification System	November 25, 2022	November 30, 2022
Scaling into Distributed Systems	November 30, 2022	December 9, 2022
Efficiency Improvements	December 10, 2022	December 15, 2022



Requirements

Functional Requirements

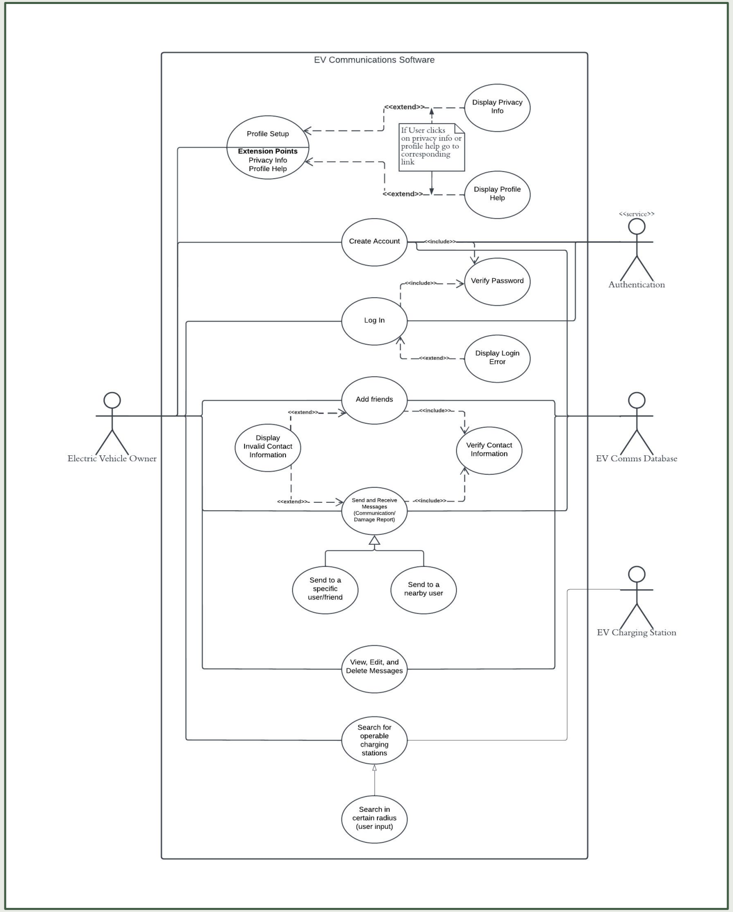
1. The user shall be able to do various actions on messages; view, edit, delete, send, and reply
2. The app shall be able to uniquely identify cars based on its given attributes and current geolocation.
3. The user shall be able to navigate to specific cars to message through the app
4. Each user shall be uniquely identified by their car and its attributes
5. The app shall notify users when they receive a message

Requirements

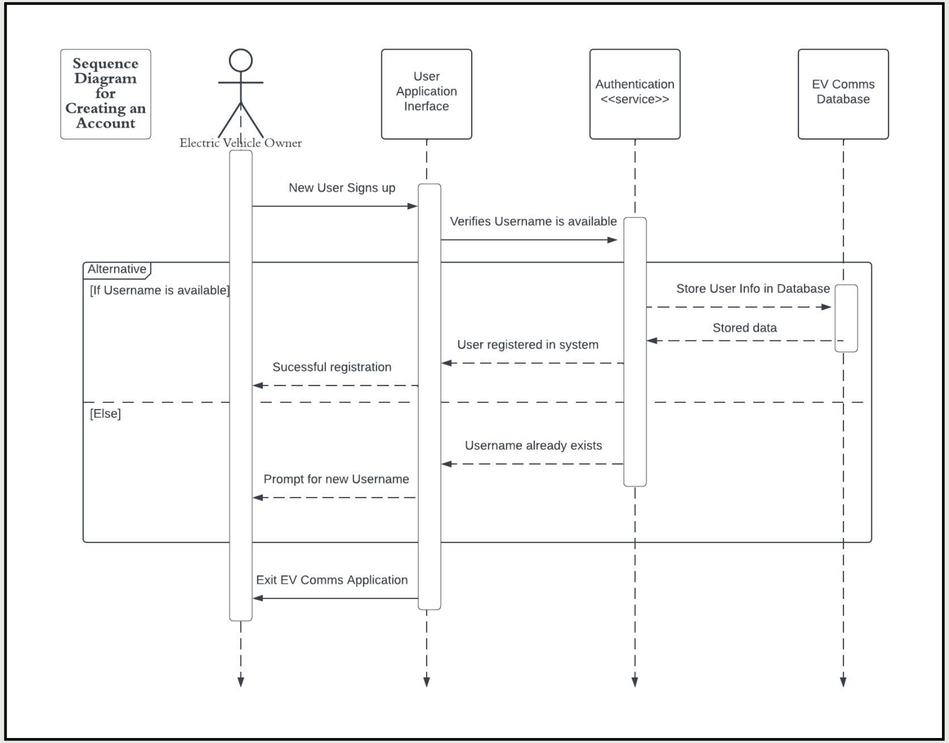
Non-Functional Requirements

1. Usability
   a. The usability shall be easy to learn and navigate through.
2. Performance
   a. Messages shall be sent at most with 0.2 seconds delay.
3. Dependability
   a. The app shall accurately track cars and send messages to the right users.
4. Security
   a. The app shall ask for permission on accessing geolocation and disable location tracking when the user specifies not to.
5. Development
   a. The app will be written in a client-server architecture.
6. Regulatory
   a. The app shall adhere to location tracking standards.

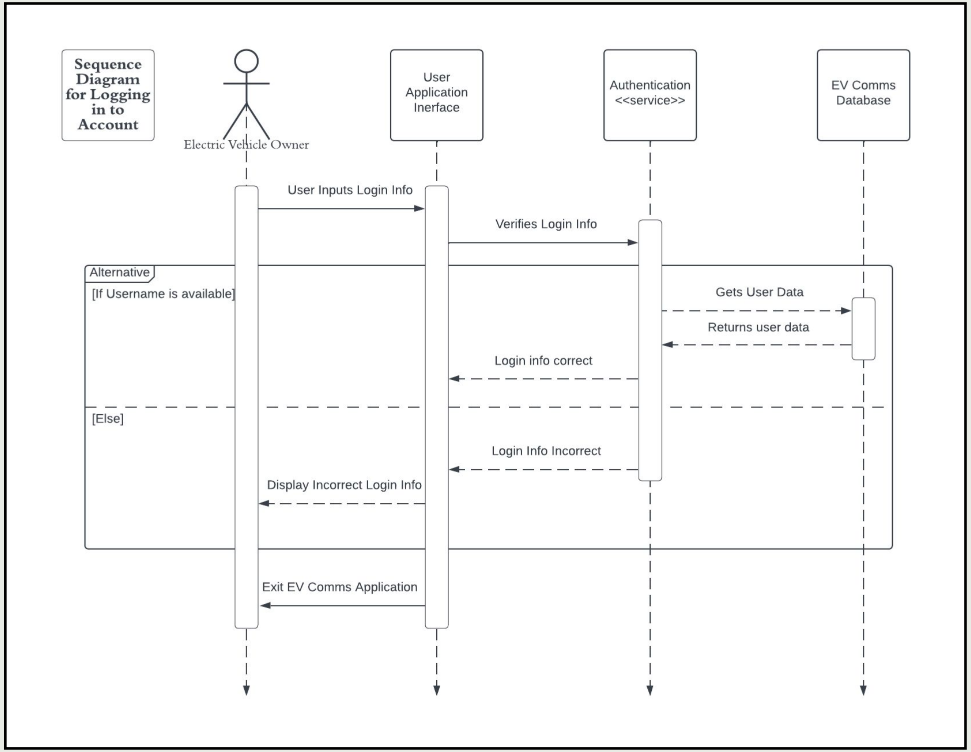
USE CASE DIAGRAM



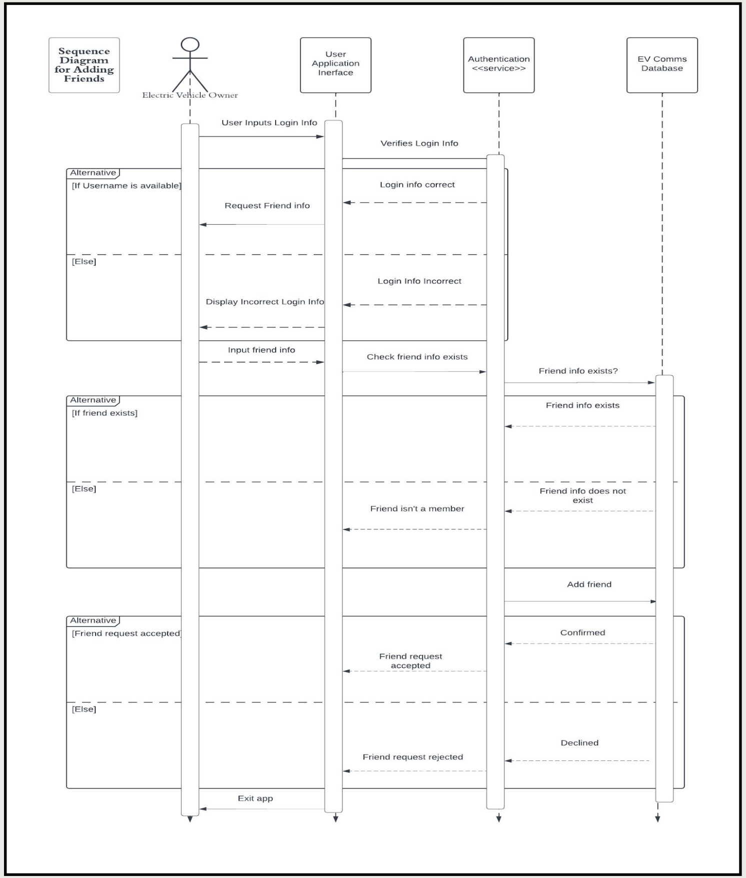
SEQUENCE DIAGRAM


Sequence Diagram for Creating an Account
Electric Vehicle Owner
User Application Inerface
Authentication <<service>>
EV Comms Database
New User Signs up
Verifies Username is available
Alternative
[If Username is available]
Store User Info in Database
Stored data
User registered in system
Sucessful registration
[Else]
Username already exists
Prompt for new Username
Exit EV Comms Application

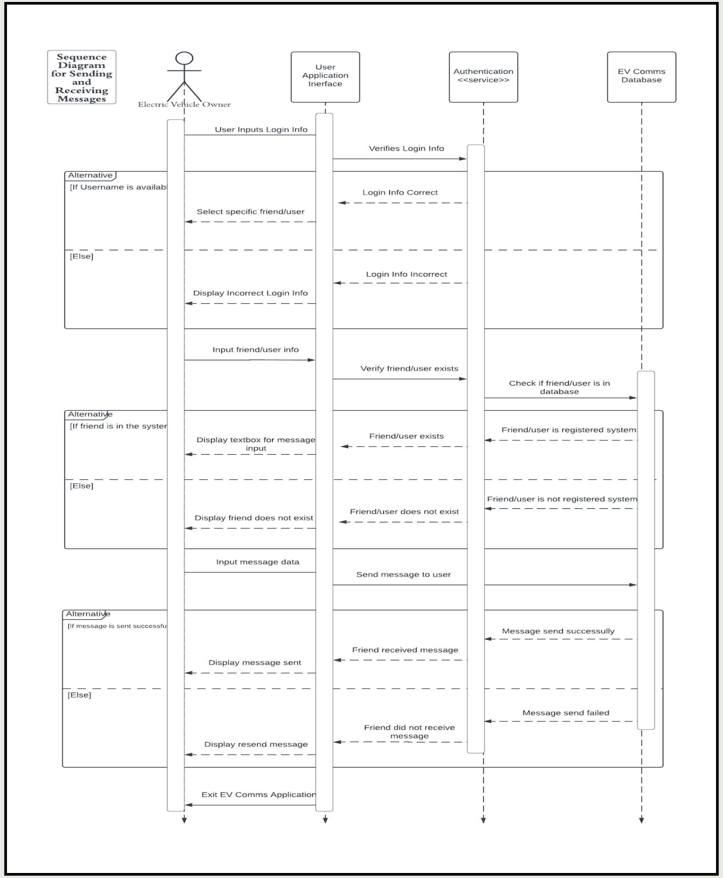
SEQUENCE DIAGRAM


Sequence Diagram for Logging in to Account
Electric Vehicle Owner
User Application Inerface
Authentication <<service>>
EV Comms Database
User Inputs Login Info
Verifies Login Info
Alternative
[If Username is available]
Gets User Data
Returns user data
Login info correct
[Else]
Login Info Incorrect
Display Incorrect Login Info
Exit EV Comms Application

SEQUENCE DIAGRAM



SEQUENCE DIAGRAM



Sequence Diagram for Sending and Receiving Messages

Electric Vehicle Owner

User Application Inerface

Authentication <<service>>

EV Comms Database

User Inputs Login Info

Verifies Login Info

Alternative

[If Username is availabl

Login Info Correct

Select specific friend/user

[Else]

Login Info Incorrect

Display Incorrect Login Info

Input friend/user info

Verify friend/user exists

Check if friend/user is in database

Alternative

[If friend is in the syster

Friend/user is registered system

Friend/user exists

Display textbox for message input

[Else]

Friend/user is not registered system

Friend/user does not exist

Display friend does not exist

Input message data

Send message to user

Alternative

[If message is sent successfu

Message send successully

Friend received message

Display message sent

[Else]

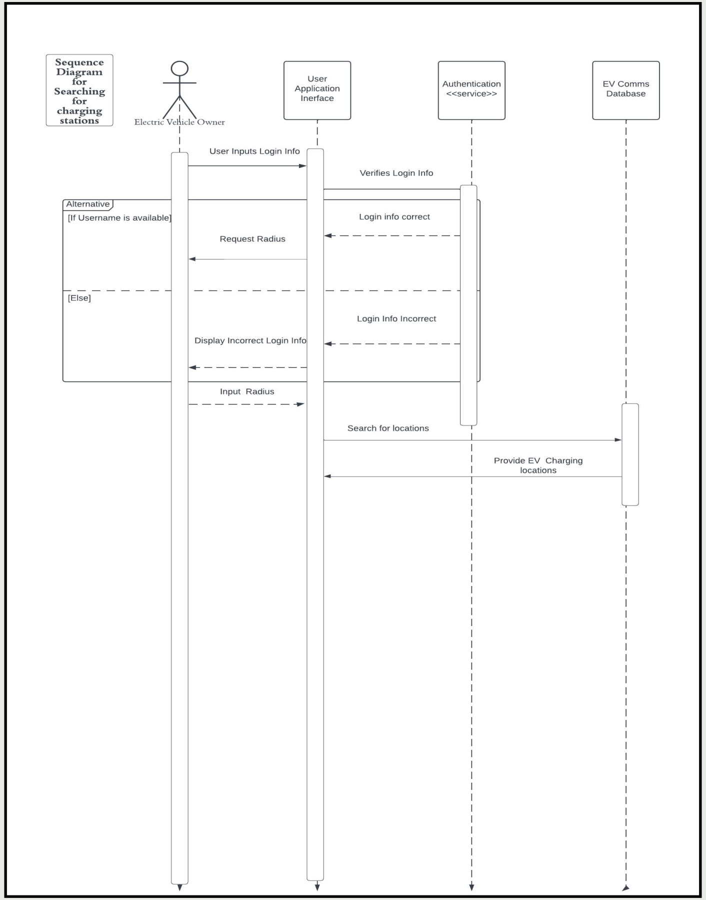
Message send failed

Friend did not receive message

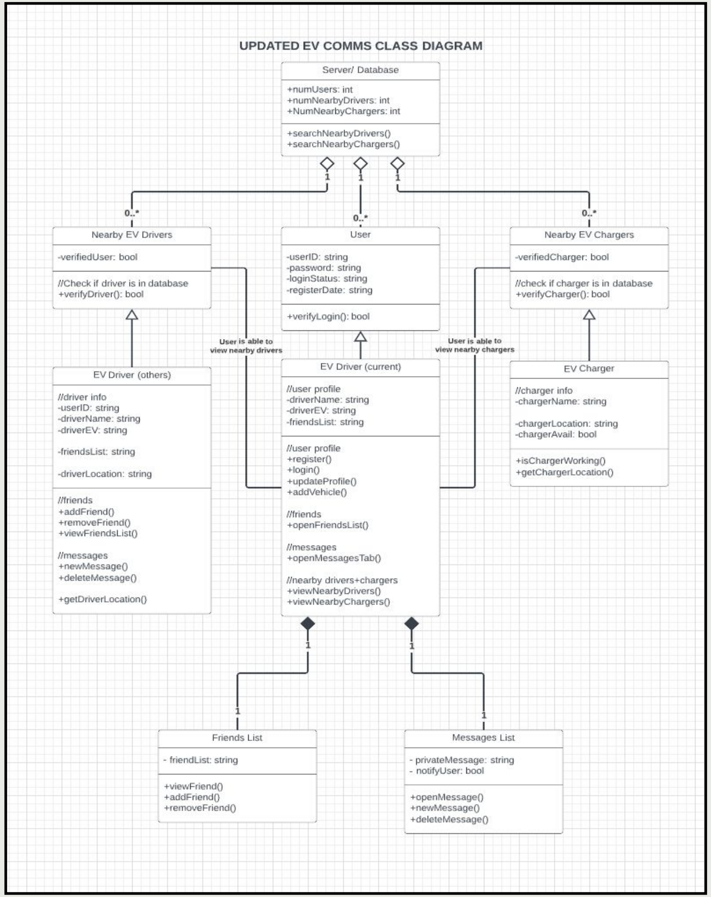
Display resend message

Exit EV Comms Application

SEQUENCE DIAGRAM


Sequence Diagram for Searching for charging stations
Electric Vehicle Owner
User Application Inerface
Authentication <<service>>
EV Comms Database
User Inputs Login Info
Verifies Login Info
Alternative
[If Username is available]
Login info correct
Request Radius
[Else]
Login Info Incorrect
Display Incorrect Login Info
Input Radius
Search for locations
Provide EV Charging locations

CLASS DIAGRAM



UPDATED EV COMMS CLASS DIAGRAM

Server/ Database

+numUsers: int
+numNearbyDrivers: int
+NumNearbyChargers: int

+searchNearbyDrivers()
+searchNearbyChargers()

Nearby EV Drivers

-verifiedUser: bool

//Check if driver is in database
+verifyDriver(): bool

User

-userID: string
-password: string
-loginStatus: string
-registerDate: string

+verifyLogin(): bool

Nearby EV Chargers

-verifiedCharger: bool

//check if charger is in database
+verifyCharger(): bool

EV Driver (others)

//driver info
-userID: string
-driverName: string
-driverEV: string

-friendsList: string

-driverLocation: string

//friends
+addFriend()
+removeFriend()
+viewFriendsList()

//messages
+newMessage()
+deleteMessage()

+getDriverLocation()

EV Driver (current)

//user profile
-driverName: string
-driverEV: string
-friendsList: string

//user profile
+register()
+login()
+updateProfile()
+addVehicle()

//friends
+openFriendsList()

//messages
+openMessagesTab()

//nearby drivers+chargers
+viewNearbyDrivers()
+viewNearbyChargers()

EV Charger

//charger info
-chargerName: string

-chargerLocation: string
-chargerAvail: bool

+isChargerWorking()
+getChargerLocation()

Friends List

- friendList: string

+viewFriend()
+addFriend()
+removeFriend()

Messages List

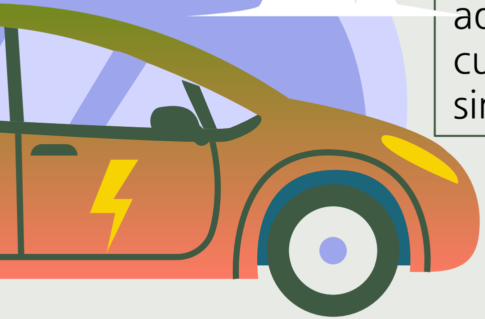
- privateMessage: string
- notifyUser: bool

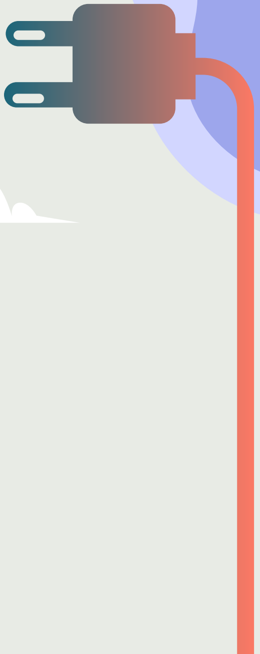
+openMessage()
+newMessage()
+deleteMessage()

1
1
1
0..*
0..*
0..*
User is able to view nearby drivers
User is able to view nearby chargers
1
1
1
1

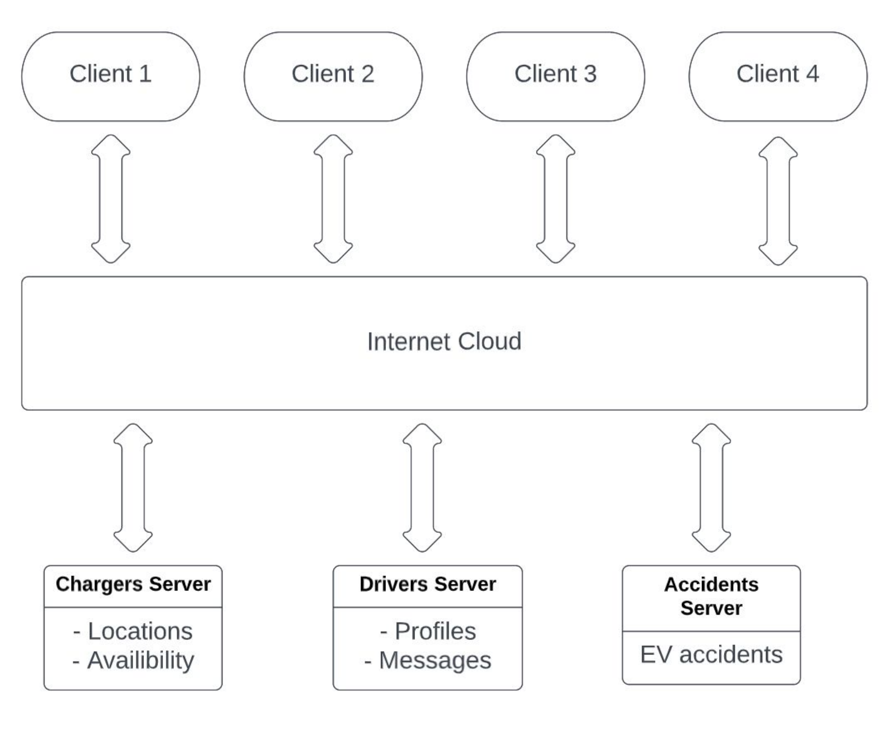
ARCHITECTURAL DESIGN

After a group discussion, we decided that the Client-server architecture is the best model for our application. In our application, different electric-vehicle drivers will create their own account, and be able to communicate with other drivers. In addition, if there are any updates on non-usable charging stations, or any major accidents involving electric vehicles, this message would need to be relayed. Therefore, having servers across the world (since electric vehicles are driven globally) would be necessary. In addition, because of the number of users (5+ million EVs currently), there needs to be multiple servers, as having a single server (repository model) would cause crashes.





ARCHITECTURAL DESIGN DIAGRAM



Conclusion

As we all know, the era for EVs is inevitable. We want to provide a seamless transition from petrol to electric cars. To achieve this, our app provides information and communication that would be convenient for all EV users. Since EV's are still far from perfection we hope that in the future our app would become more refined and include a larger database.

- Quicker load times
- Less server crashes
- More charging locations

Some changes that could be implemented would be the duration of how long our app would take to complete. The FP algorithm calculated that it would take 2 weeks, but we believe it will take much longer.

Thank You