Fleet Monitoring System

Team 12: sddec12-20

Members: Lorenzo Chavarria, Nicolas De la Cruz, Joe Herrera,

and Marco Yepez

Executive Summary

Development Standard & Practices

- Agile Development Standards
- JavaScript Coding Standards
- Google Java Style Standards
- MongoDB Schema Design Standards

Summary of Requirements

- Develop Android mobile application for Fleet Monitoring System
- Provide GPS location of each vehicle in a fleet
- Provide On-Boarding Diagnostics (OBD) data display for vehicles
- Provide client team chat

Applicable Courses from Iowa State University Curriculum

List all Iowa State University courses whose contents were applicable to your project.

- CPR E 288
- COM S 309
- COM S 363
- COM S 352
- S E 319
- S E 329
- S E 339

New Skills/Knowledge acquired that was not taught in courses

- Server configuration and deployment
- REST API creation
- Hardware equipment knowledge, e.g. I/O pins data sheet

Table of Contents

Design

Overview

With UPS, FedEx, and numerous fleet companies out there, we need something to monitor and manage the vehicle fleets. With our fleet monitoring system, a company would be able to use our user-friendly solution. Our solution revolves around a mobile application to view GPS locations of each vehicle, client chat, and vehicle's On-Board Diagnostics (OBD) dashboard display. Our solution would be based on MongoDB for our database and a server running Node.js REST API for microservices. The fleet would need a PiCAN for each vehicle to be able to send data from the vehicle to the server. Through the connectivity of each component, our solution would provide clients a fleet monitoring system.

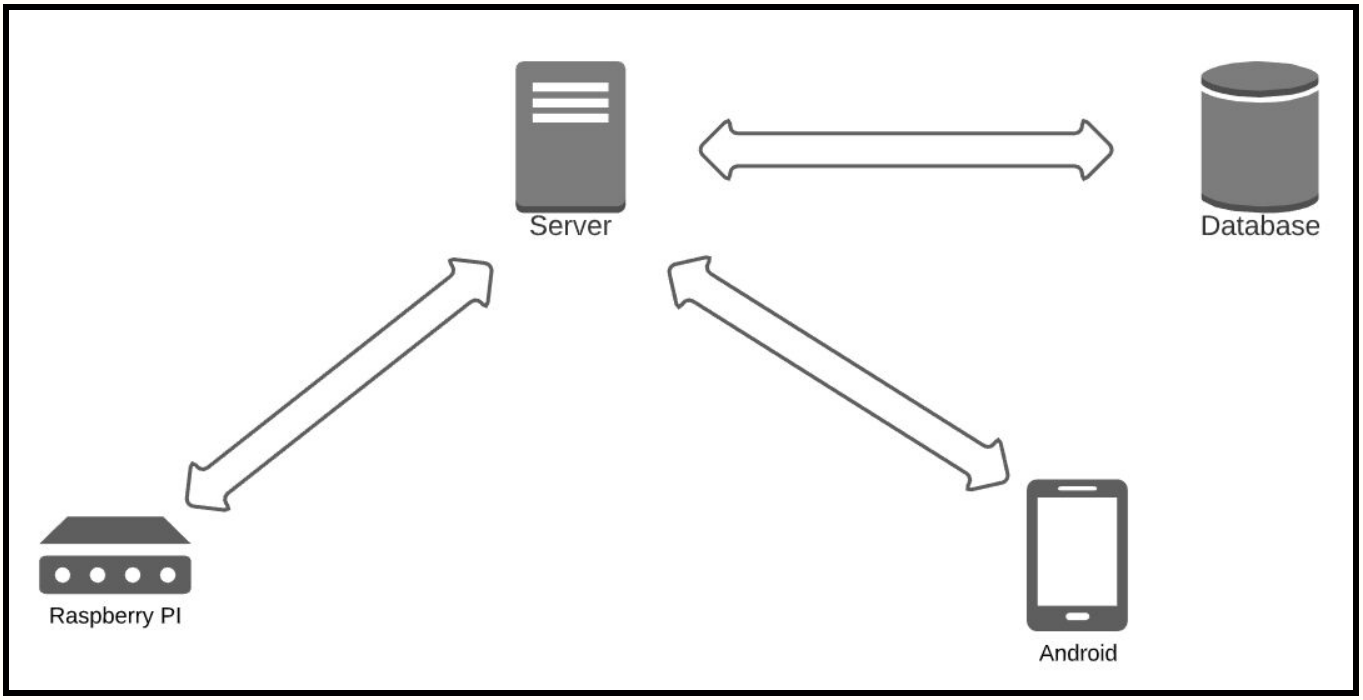
Engineering Constraints and Requirements

- Constraints
    - The cost of the project must not exceed $200
    - Project deadline is November 25th, 2020
    - Each RaspberryPi is limited to 1mb of data each month
    - Vehicles must be 2009 or newer
    - The Android application must support Android 6.0 and above
- Non-functional Requirements
    - The server must be able to support over 50 clients with a response time of less than 5 seconds.
    - A user must be able to navigate between features of the app in less than 5 seconds on average (demonstrates ease of usability)
    - The application must not crash 99% of the time a user is navigating
    - The server must be running for 100% of the time during business hours. The server may be updated during non-business hours.
- Functional Requirements
    - Communicate data from a vehicle to server
    - Record data into database
    - Display vehicle data on a map for users
    - Allow users to communicate within a messaging system
- Operating Environment

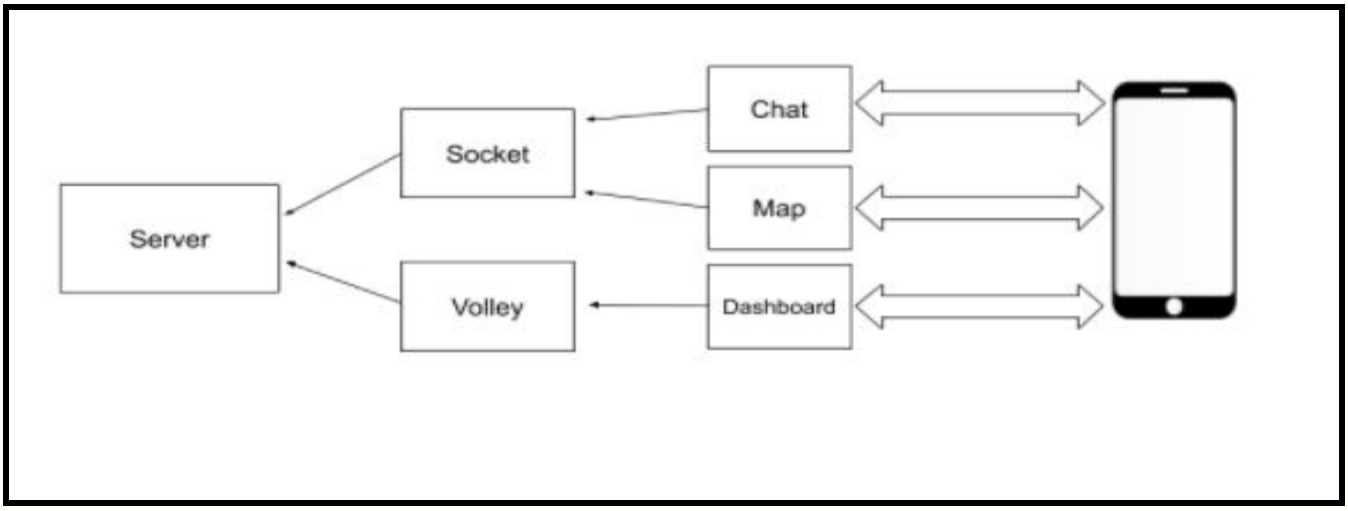
- Windows OS
- Android OS
- Raspberry Pi OS

UML

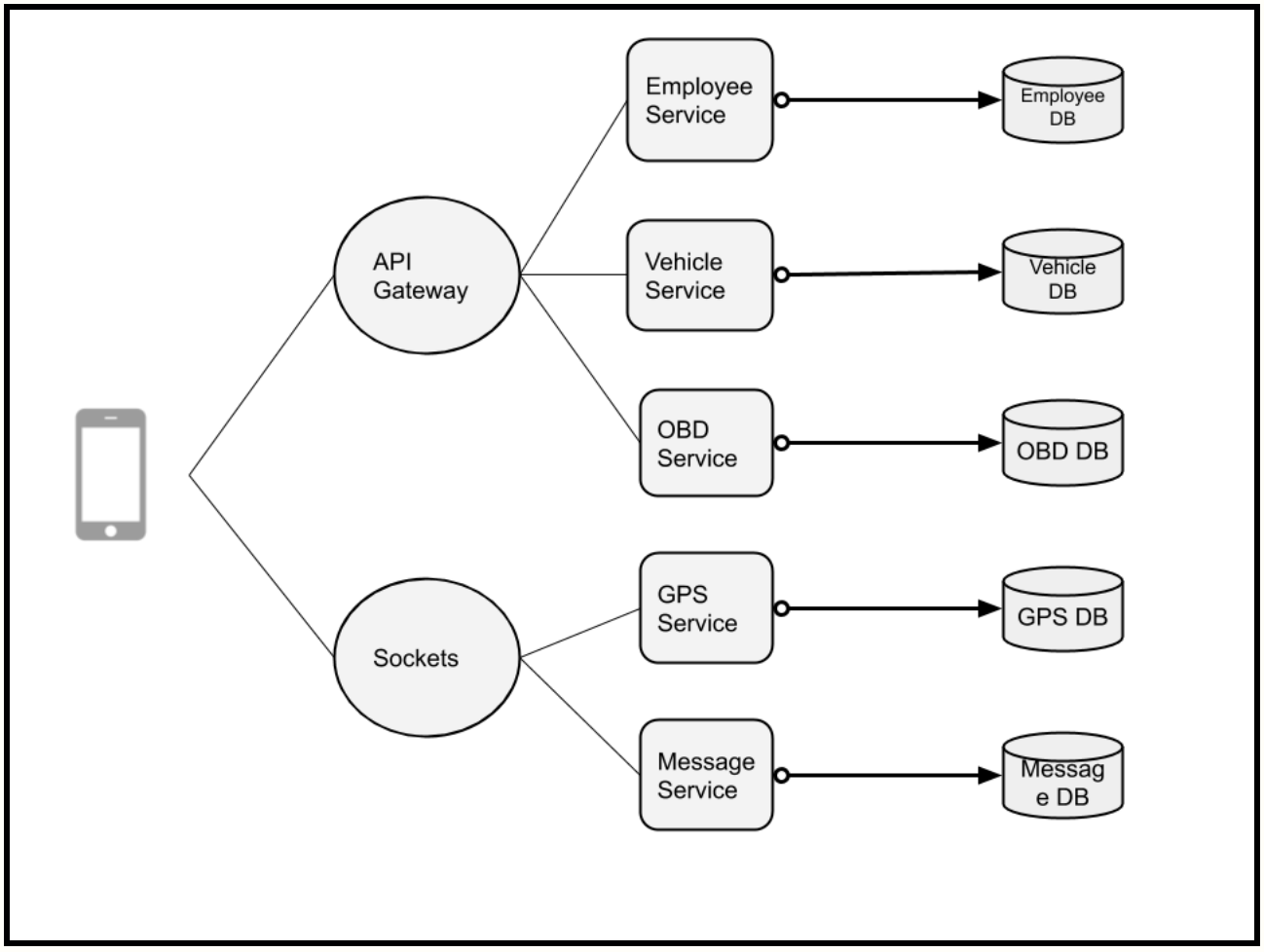
Overview



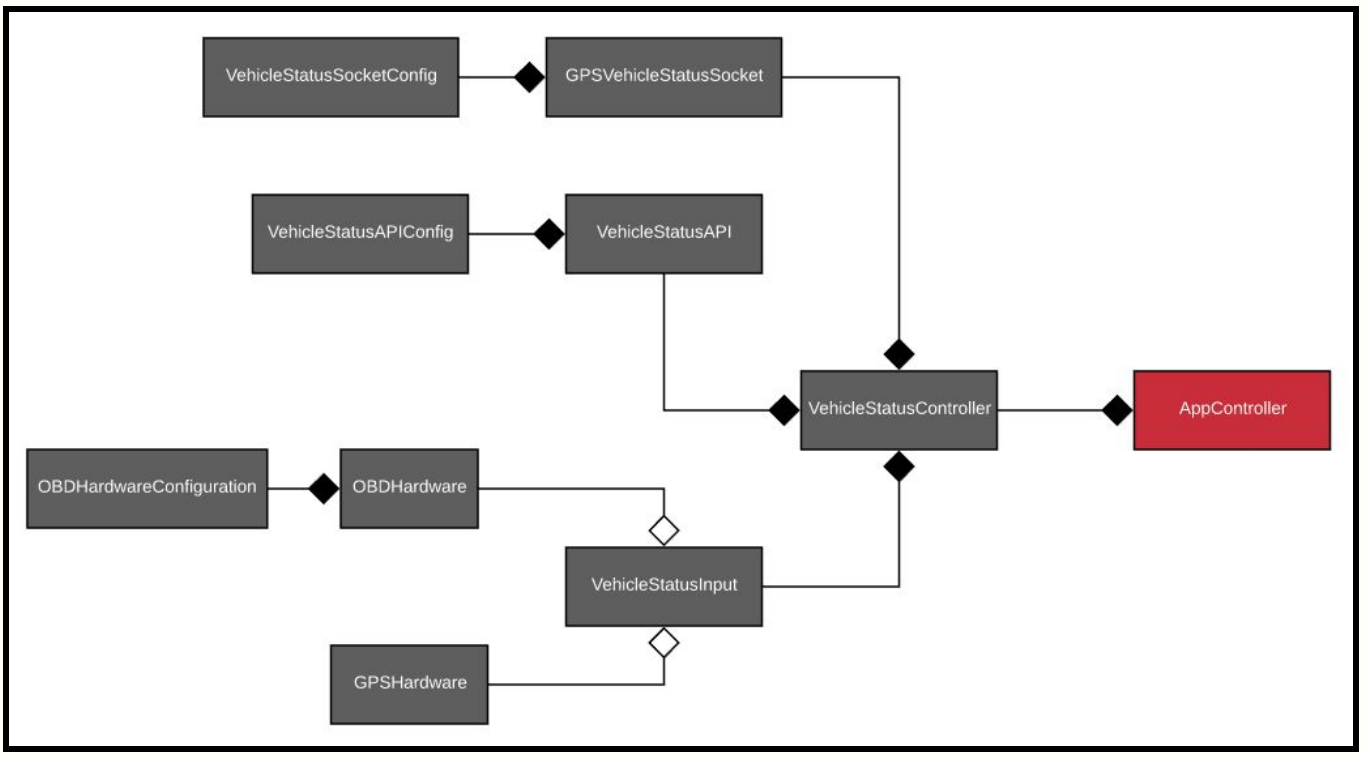
Android



Backend



Raspberry Pi



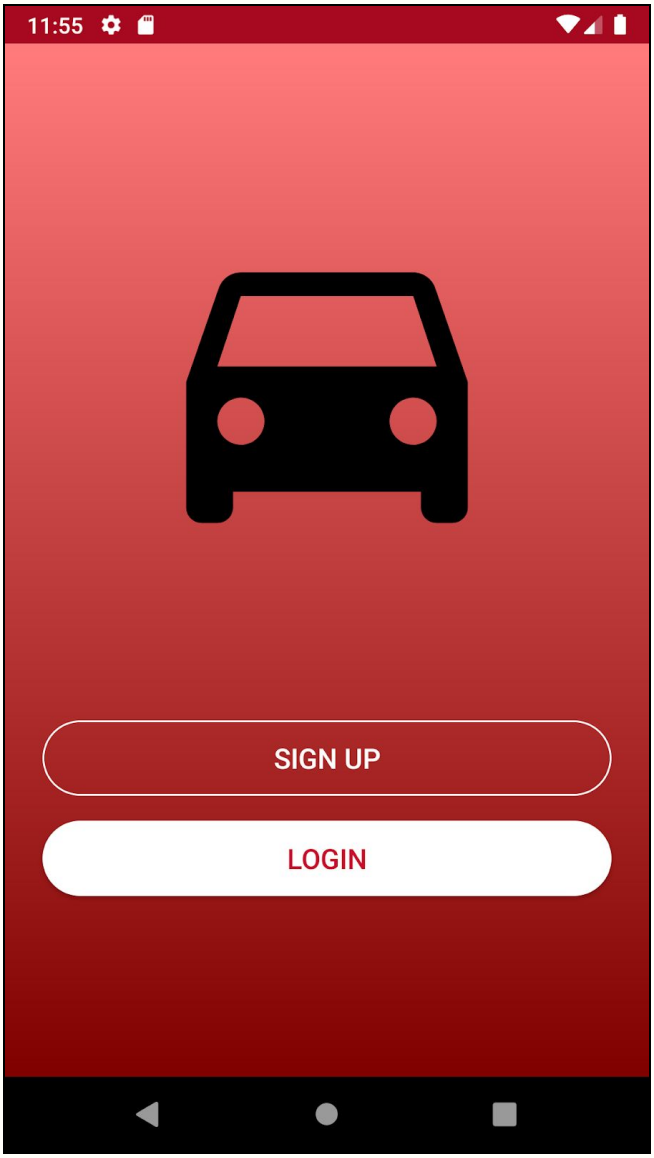
Implementation Details

Android

For the Android application, Android Studio was used using Java. Volley, a HTTP library, was used for communication between the application and server. The vehicle data was received from the server, which got the information from the database. For the messaging feature, a web socket was used for displaying each message that was sent from each user. For the map feature, a Google Maps API, called Maps SDK, was integrated for Android. This API allowed us to add a map based on Google Maps data to the application. This API also allowed for pins to be displayed which represent each vehicle. Each vehicle pin is also clickable and displays data for that vehicle when it is clicked on. This data as well as the location of that pin is read using a websocket that reads the information in real-time from the server.

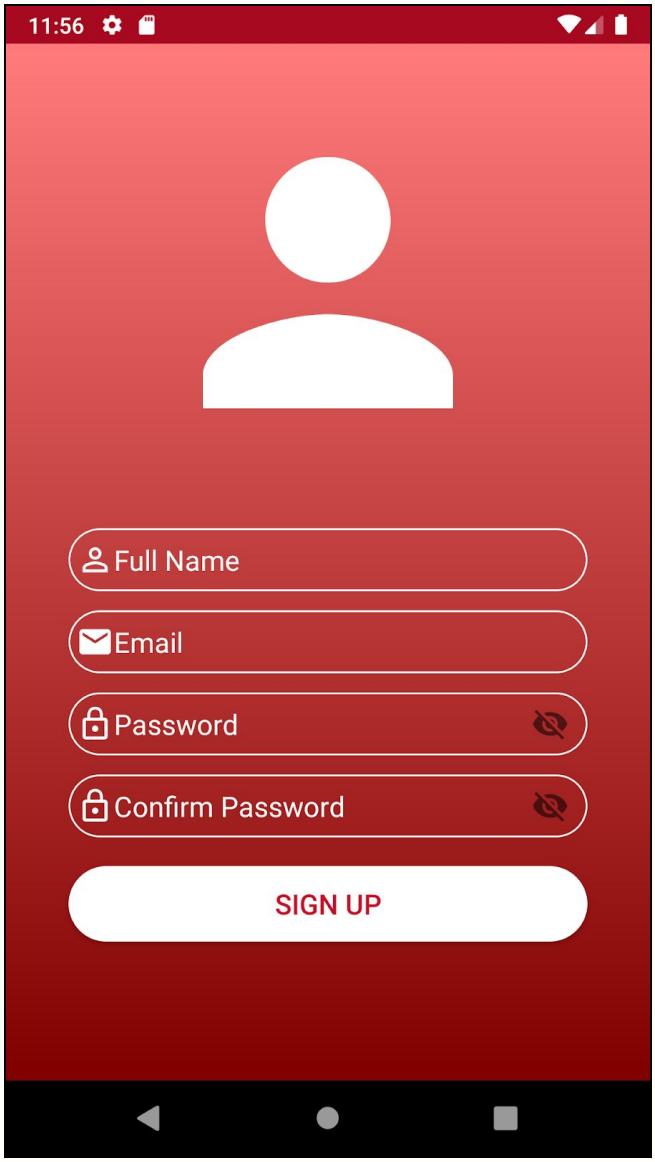
GUI

Landing Page



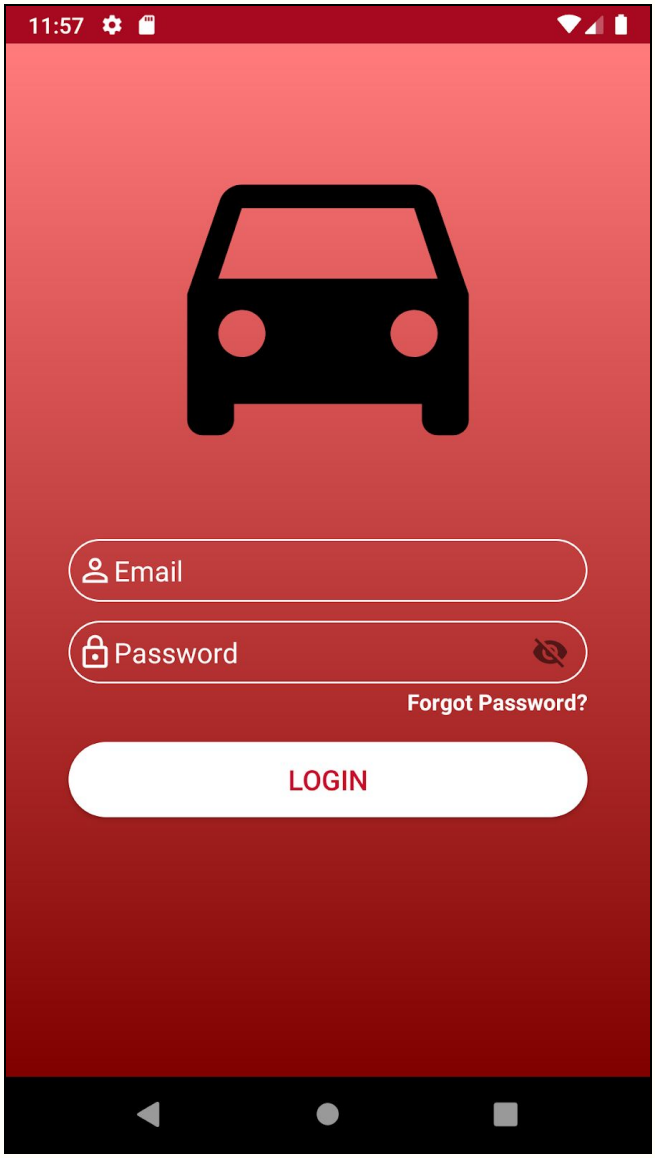
The landing page includes the user two options to select: sign up or login.

Sign Up Page



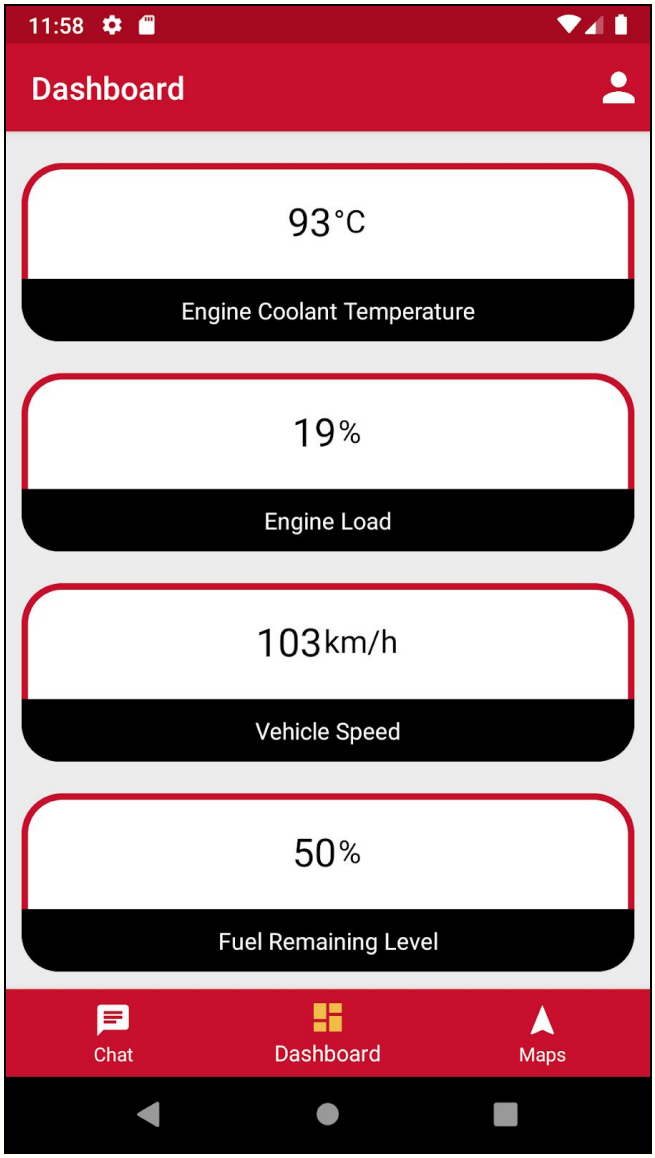
A user can sign up as a driver using their full name, email, and a password. After signing up, the user will be presented with the dashboard.

Login Page



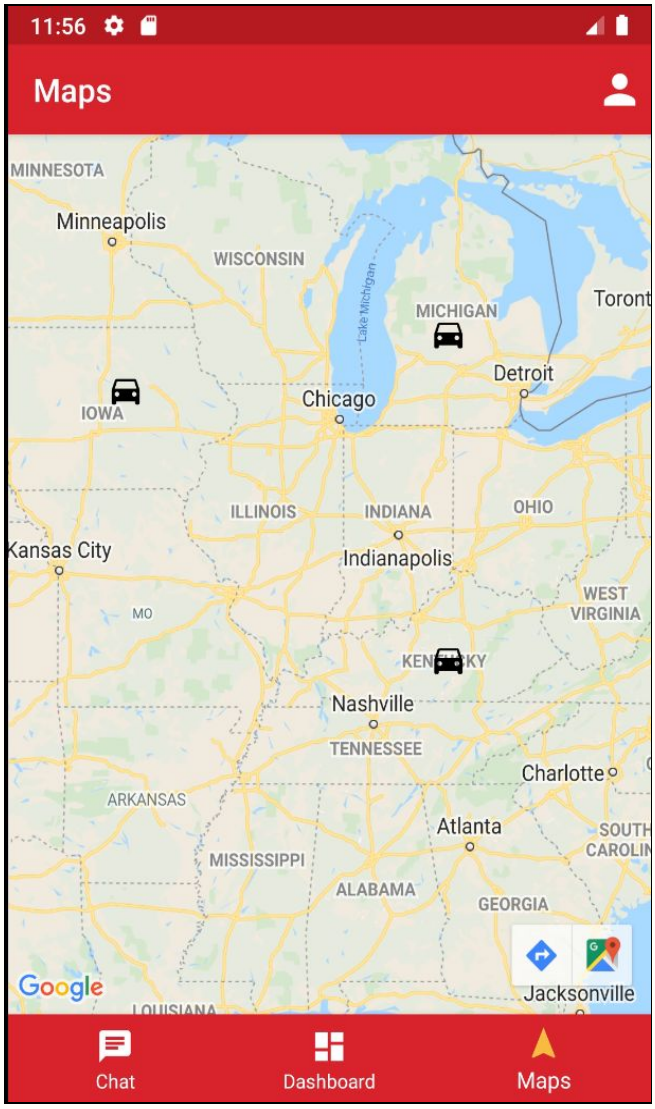
A user must login using a valid email and password that belongs to their account.

Dashboard Page



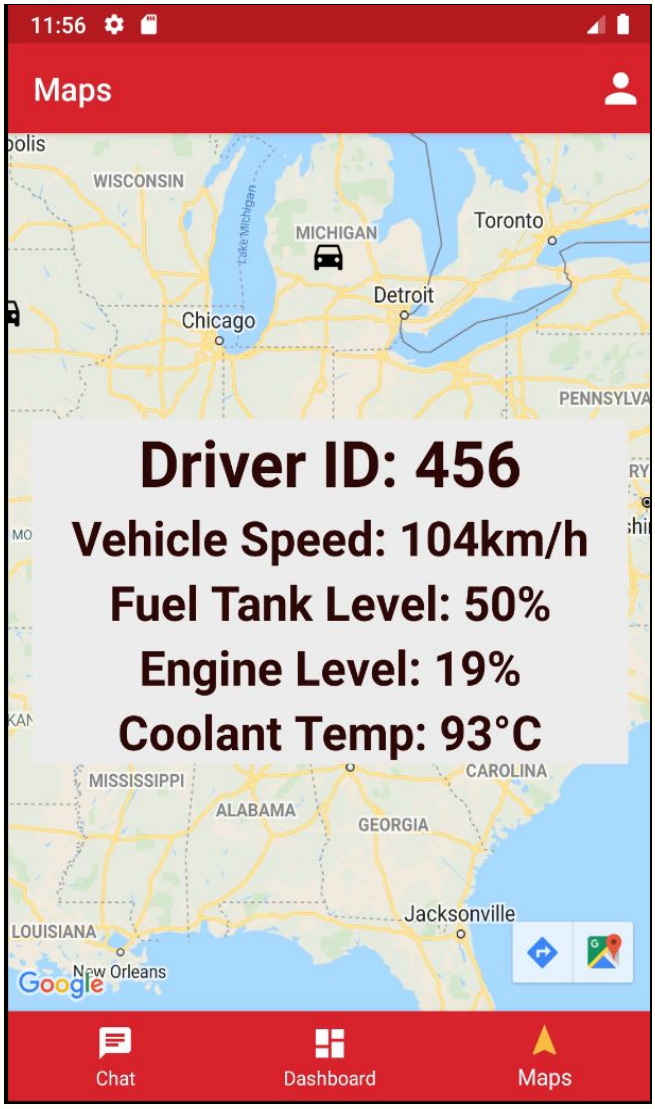
The dashboard displays data about the user's current vehicle. The vehicle data available includes engine coolant temperature, engine load, vehicle speed, and fuel remaining level.

Map Page 01



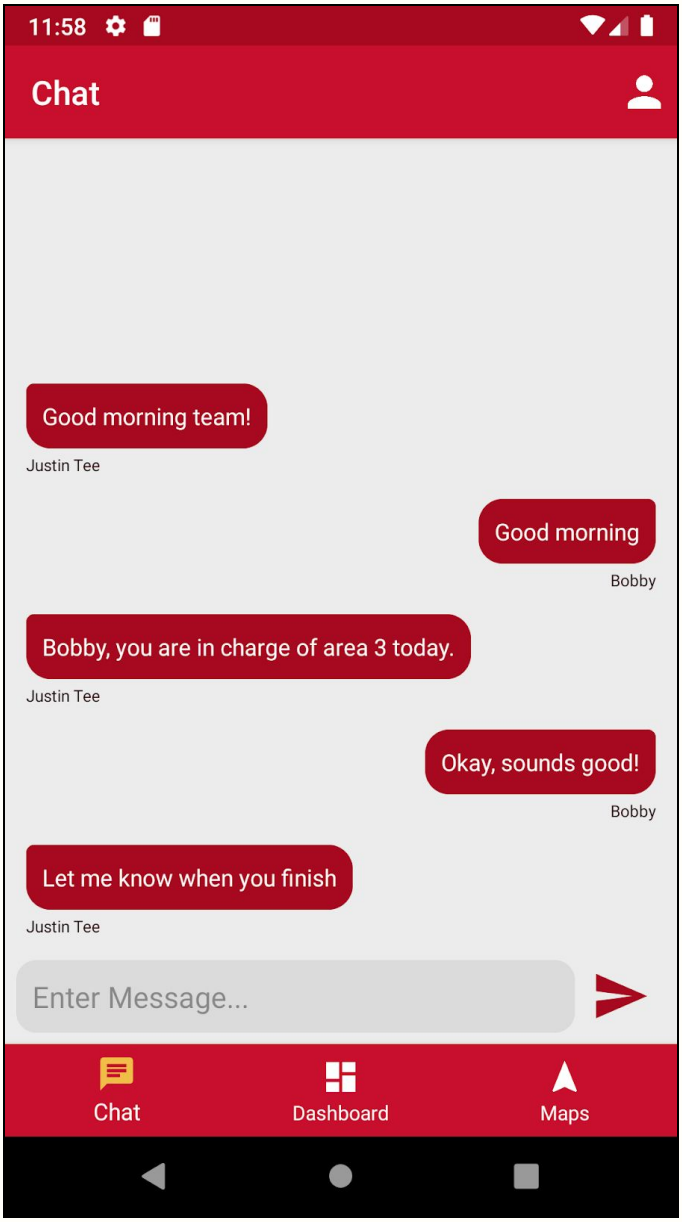
The GPS map contains the locations of other drivers. Each vehicle pin is updated in real-time.

Map Page 02



After clicking on a vehicle pin, a popup with information about that specific vehicle will be displayed.

Chat



A user can use the chat to communicate with other drivers on the fleet management system.

Backend

The server for this project was not only a bridge between client and data, but it was multiple bridges. Our server's architecture is using microservices to be modularly independent between different services. This allowed maintenance to take place in one service and let the remaining services function as they normally would. When the GPS feature came around we found out the microservices were not going to be sufficient alone, so we needed to use sockets. We opted for sockets to eliminate a continuous while-loop queuing the database every n-seconds to allow our application to have real-time communication between the vehicle and the clients. For our database, we are using MongoDB's NoSQL style database to prepare any companies with a high-scalable database to accompany our NodeJS server.

Raspberry Pi

The Raspberry Pi was the engine of our website. Implementing this part of the project consisted of several hardware and software components. There were three main hardware components used: Raspberry Pi, vehicle simulator, and GPS module. For software components, there were several used from the python pip repository. Some libraries included gps, socketio, and python-can. To develop, modify, and deploy the project, PyCharm was used. PyCharm was an essential tool because it provided many remote tools that allowed development to be done in a more powerful computer than the Raspberry Pi while allowing deployment and running the project using the Raspberry Pi interpreter and libraries.

Testing Process & Results

Android

- Dynamic testing
    - Functionality was visually inspected to ensure goals were met throughout the project
    - Server communication was tested by having data from the Raspberry Pi to go to the server and then displayed on the android application

Backend

- POSTMAN for testing API routes
    - Mimicked as a basic HTTP call to show any and all results from each call. This call also displayed the return status code of each call
- Mock testing for websocket
    - Allowed a proxy client to communicate with other proxy clients and the data sent between each client was also stored in the database

Raspberry Pi

- Used vehicle simulator to provide OBD queries and outputs
    - Simulator was able to return proper data
- To test gps location, user can just run the module using the linux terminal
    - GPS module displayed current location

Appendices

Appendix I: Operation Manual

Raspberry Pi

Operating the Raspberry Pi is simple and, with some understanding, anyone can set it up.

Hardware

- PiCAN
- Vehicle Simulator
- GPS Module

Resources

- OBD-II PIDs
- PiCAN
- GPS Module

Recommendations

- PyCharm Professional
    - Useful for remote interpreter and transferring files to a Raspberry Pi

Setup

1. Connect the GPS module and the simulator to the PiCAN
2. Copy the embedded system project files to the Raspberry Pi
3. Ensure all the libraries and configurations are correct
4. Run the AppController.py file on a terminal
5. After running the file, you should expect outputs on the terminal showing a log

Server & Database

Services Needed

- MongoDB Account

- Web Server

Recommendations

- Unless experienced with other Linux versions, Ubuntu works well for server
- Server SSH access with install permissions allowed

Setup

1. Clone project into desired folder in server
2. Run all microservice files named “server.js” or “socket_server.js” using the command:

node <service file name> &

Android Application

Requirements

- Android 6.0 or above
- Connection to cellular data or wifi
- Fleet Monitoring System mobile application downloaded and installed

Setup

1. Download and install application on mobile device
2. Open application and sign up for an account
3. Upon signing up, the user will be redirected to the dashboard
4. Navigate between dashboard, chat, or the map as the user desires

Appendix II: Alternative/Other Versions of the Design

References

Latest Design Document

- http://sddec20-12.sd.ece.iastate.edu/docs/design/Design%20Document%20v3.pdf

Appendix III: Code

References

GitLab

- https://git.ece.iastate.edu/sd/sddec20-12