Fleet Monitoring System

PROJECT PLAN

sdmay18-18
Lotfi Ben-Othmane
Venecia Alvarez - Point of Contact
Kendall Berner - Project Manager
Matthew Fuhrmann - Report Manager
William Fuhrmann - Test Engineer
Anthony Guss - Technical Lead
Tyler Hartsock - Web Manager
sdmay18-18@iastate.edu
https://sdmay18-18.sd.ece.iastate.edu
Revised: 3-Dec-2017/3.0

Table of Contents

List of Figures

1 Introductory Material

1.1 Acknowledgement

We would like to thank Lotfi Ben-Othmane for his involvement in the project. His contributions to our project as project client and advisor include giving frequent feedback during weekly meetings, providing hardware for the project, and guiding the group members with his significant knowledge in the project domain.

1.2 Problem Statement

Companies spend large amounts of money on their vehicle fleets. Vehicles in a fleet can be a huge waste of money if they are not optimized to run efficiently. They need to be driven in a way that gives an optimal balance between speed and gas usage. Vehicles also require routine maintenance. There are some fleet management applications on the market, but they all have flaws. Some of the products on the market aren't customizable, and some products don't display enough information to fleet managers.

Our goal in this project is to create a complete, scalable, and customizable fleet management system that companies could use and build to work on their vehicles for the purposes they deem useful. We have been provided a Raspberry Pi, a PiCAN2 board, a google cloud server, and an Adafruit Ultimate GPS Breakout board. We will develop a Python application to run on the Raspberry Pi that queries the vehicle's components and relays the collected data to a server. This server will process and store the data. The server will also provide a website with real-time and historical data on the location of all vehicles and internal data from the vehicles such as gas consumption and vehicle diagnostics. The website will also show useful interpretations and statistics of the data for individual drivers, vehicles, and for an entire fleet of vehicles.

1.3 Operating Environment

The Raspberry Pi 3 will need be inside the vehicle located close to the OBD-II port of the vehicle, which is usually in the front of the interior of the vehicle. This means that the Raspberry Pi will operate in the temperature range of the vehicle it is in, which could vary significantly depending on where, when, and how it is being used. The board is qualified to operate between 0°C and 70°C, which may be a concern for the cold temperature operation. The Raspberry Pi 3 will also need to be secured in the vehicle, which should not be difficult to do. Because the PiCAN2 board makes the Raspberry Pi 3 not fit into its case, we will need to develop a new case for it. The server is being deployed to Google Cloud, so we do not have to consider its operating environment.

1.4 Intended Users and Intended Uses

The intended users of our product is primarily the fleet managers. The fleet managers need to be able to view important information about their individual drivers and individual vehicles, as well as aggregate data for their fleet and drivers. This information includes historical data on the location of the vehicles and driver behavior and live data for vehicle tracking. We are also considering the possibility of adding driver functionality using the Raspberry Pi 3, but have not committed to adding any functionality for them yet.

The intended uses of the product are for the fleet manager to log on to the website and view information about their fleet, drivers, and vehicles, both live and historical. Drivers and vehicles will be registered to a fleet manager, and these managers will have access to statistics and live data. The information will be presented primarily in graphs, charts, and maps, and will be simple to use for managers.

1.5 Assumptions and Limitations

Assumptions:

- The vehicles that will be using this product will have an OBD-II port.
- The vehicle will be able to provide power to the Raspberry Pi 3 OR
- A battery will be provided for the Raspberry Pi 3

Limitations:

- The system shall not interfere with the driver's ability to access the pedals.
- The system will not be tested with all types of vehicles: vehicles who do not follow the OBD-II PID standards and were not tested with will not work with the system.
- The system will require mobile data for operation, whether provided by a mobile phone hotspot or by a USB device that receives mobile data using a SIM card.
- The system will use limited calls to Google Maps API to avoid costs.
- The system will use limited processing on the Google Cloud server to avoid costs.
- The system will support a limited number of vehicle types due to schedule limitations.

1.6 Expected End Product and Other Deliverables

The end product will consist of three components: a Raspberry Pi 3 with a Python application and other hardware, a server deployed to the Google Cloud platform that processes the vehicle data, and a website that displays the data processed on the server to the fleet managers. The final version of this product will be delivered by April 23, 2018 to our client.

The webpage will be made for use by fleet managers. It will have a simple and intuitive design made for ease of use. This web page will display a live map tracking the locations of all vehicles in the fleet in real time. A single vehicle can be selected, which will bring up a

profile that will display information about that vehicle such as remaining gas, miles driven, driver, etc. The webpage will also have a page that details the stats of the fleet as a whole, so that the manager can see how much gas the entire fleet uses over a month, for instance. Graphs will be provided to make the statistics and information easier to understand.

Our server will be created using Node.js. The server will handle incoming raw data from the embedded devices and process this information into usable information that it will then put into a database. The server will also handle data request from the web page. Both the incoming and outgoing data will be transmitted through REST API calls. This will allow us to use a single protocol across our entire system. The Node.js server will be hosted on the Google cloud platform, allowing us to scale our resources to the demands of our services.

The Raspberry Pi 3 application will be a Python application that queries and pulls data from the OBD-II port of a vehicle and forwards it to the server. It will also acquire location information from the Adafruit Ultimate GPS Breakout board and forward it to the server. This application will need to be easy to install for new vehicles, and should correlate sent data to both the driver currently using the vehicle and the vehicle's unique identifier.

2 Proposed Approach and Statement of Work

2.1 Functional Requirements

The product shall:

- Gather data from a vehicle
- Transmit data from the vehicle to the server
- Process raw data from the vehicle on the server
- Record vehicle data into a database
- Display a map with a location of all vehicles in the fleet
- Display historical data for a certain vehicle (location, gas usage)
- Allow managers to register vehicles that belong to a particular fleet
- Display vehicle information only to users who have that vehicle registered to their fleet

2.2 Constraints Considerations

The product shall:

- Be able to be used by vehicles at any time and location
- Utilize Google Cloud services
- Only allow managers to view fleet data on the dashboard
- Have the server implemented using Node.js
- Have communication between components implemented using a REST API
- Use angularJS on the client side

Our coding standards will align with the basic coding standards for each programming language used. We will also use vulnerability testing to ensure that the data can only be accessed by processes or processors with permission. This ensures that we will not be doing anything unethical. Coding standards are important for our project, because the client intends to modify our software in the future to fulfill the needs of future clients. Following coding standards for the programming languages we use will make fixing, and modifying the software set much quicker, easier, and cheaper.

We plan to plan, develop, and test using IEEE standards. We will use the IEEE Software Development Planning Standard to plan the development of our project. We will use the IEEE Software Testing Standard to test our code base. We will use the IEEE Standard for Software Reviews to review our code. These standards will help ensure that we our project remains ethical.

2.3 Technology Considerations

The Raspberry Pi 3 is a very versatile device, which will make it easy to add new functionality for the later versions of the product. Python is also a versatile language, which will work well with the Raspberry Pi 3 and with the Adafruit Ultimate GPS and PiCAN2 boards. Python will allow us to develop our modules separately for each hardware device that is needed, and will allow for rapid development of applications dealing with hardware or user interface.

The server code will use Node.js because it is a client requirement. A disadvantage of using Node.js is that it is slower than a Java server. Node.js is also single-threaded, so it is slow when many requests are being made. The main advantage of Node.js is that it has high throughput, and therefore it has high scalability. The server will be hosted on Google Cloud. Google Cloud will be useful to us because it is well-documented and easy to use. A disadvantage is that although they have competitive prices, it is still a cost that our team has to account for. The server will be connecting to a Mongo database for data storage. Mongo at its core is a simple database to use, and usage can be improved by using

Mongoose on the server. Using Mongoose will allow us to easily add and edit models for our Mongo database.

For the front end, HTML and JavaScript will be the best way to display information. AngularJS reduces the need for functions. It is faster because it modifies the page DOM directly instead of adding inner HTML. AngularJS gives multiple ways to accomplish a task. This can be a disadvantage because it is difficult to know what is the best way to do one thing. It is not a very scalable framework because it requires a lot of refactoring when trying to extend existing functionalities. Data will be plotted using the Google Maps API and Chart.js which are well-documented and easy to implement. They work very well with our design plans for the front end.

2.4 Safety Considerations

The only safety concern for the project is ensuring that the Raspberry Pi 3 on the vehicles is not a safety concern for the vehicles in case of a crash and does not impede with the driver's ability to operate the car. We plan to evaluate the methods to secure the device to the vehicle.

2.5 Previous Work and Literature

We researched several commercial fleet management applications to get an idea of what kind of features people would want, as well as to see where there are opportunities in the market. There are two main categories that we feel most fleet management applications fit into: applications that had the driver have a mobile application, and applications that had the vehicle have a device that is installed into the OBD-II port.

Most of the applications where the drivers have a mobile application sent the mobile device's' GPS and data manually entered by the drivers to a web application used by the fleet manager. The web application the fleet managers used generally had ways to interact with the drivers in the fleet, such as messaging and dispatching. Examples of this type of application in our market study were collectiveFleet, ManagerPlus, and OnFleet. Generally, our application is covering a different niche than applications from this category because we will be providing internal data from the vehicle that these applications do not have access to, and we do not plan to provide much functionality for the driver for interacting with their fleet managers. Our product will provide useful interpretations of vehicle internal data for fleet vehicles instead of functionality for administration of a fleet.

We also noticed that a lot of things needed to be entered manually by drivers for these applications. We intend to improve on these systems by automating as much as possible. For example, several apps required the drivers to enter how much fuel they got at a gas station. We are going to try to get information automatically to minimize the impact of our product on the driver.

Most of the applications where the vehicles have an installed OBD-II device have a similar goal to our product, which is to provide live tracking, statistics, and useful interpretations of vehicle data to fleet managers. Examples of this type of application in our market study were FleetComplete, Fleetmatics Now, and High Point GPS. While our application is covering a similar niche in functionality, our use of the Raspberry Pi 3 provides us with many more customization options for the OBD-II device and will provide much more potential for adding additional functionality for drivers.

With the hardware provided, we see two main opportunities for differentiating ourselves from other fleet monitoring/management applications: adding driver functionality using the Raspberry Pi 3's resources, or adding more functionality to our website that is not represented in the market. Overall, there are many fleet management applications that provide lots of different functionality, making it difficult to provide completely new functionality. We will have to use the versatility of the Raspberry Pi 3 and the large amount of data provided by the OBD-II port to provide a distinct product.

Common features from both categories were fuel and speed tracking and the real time map displaying the locations of other drivers. For the first version of our product we selected the most ubiquitous and most useful seeming features to include: fuel tracking, vehicle location tracking, and vehicle speed tracking.

See the complete list of fleet management software we reviewed in the references section.

2.6 Possible Risks and Risk Management

One of our major risks for the project from the start was that we would be unable to query the OBD-II port with our hardware. The original hardware that the project was using was an Android microcontroller, the NXP i.MX6 Android Board. Unfortunately, this risk occurred when we were not provided with the cables needed to connect the board to the OBD-II port, and the cable was not a standard type of cable. The board also did not come with sufficient documentation, and the library the board was supposed to use to access the OBD-II data was provided only as a compiled library. This made it difficult to understand how to fix the problem of the board connection, and the vendor of the board was not responsive to queries asking to get the part we needed or the documentation of the library. For these reasons, our client ordered a Raspberry Pi 3 and other hardware to help us continue the project. This occurred in early November, and put our development for the vehicle interface significantly behind schedule.

Another risk that occurred is relating to our knowledge of OBD-II. We did not understand how to program for OBD-II PIDs or how they worked. However, through sharing knowledge and resources with each other, we have developed a sufficient understanding of the OBD-II PIDs and message system that will allow us to work with this standard for the remainder of the project.

A possible risk for the project is that the Raspberry Pi 3 is not able to correctly query and retrieve OBD-II data from a real vehicle. We have been successful in retrieving this data from an OBD-II simulator, so we are not concerned with establishing an OBD-II connection. However, real vehicles may not follow the OBD-II standard for PIDs. If this is the case, our plan for mitigating this risk is to change from uploading values to the server to uploading raw OBD-II message data to the server and using OBD-II data and notes from test drives of the vehicle to try and figure out how the OBD-II data correlates to the information that we need.

Another possible risk is that the server using Google Cloud and the website using Google Maps API could result in increased expenses is the project. To avoid this risk, we plan to be cautious when deploying code that uses these to make sure that the usage does not exceed normal levels.

On the front end, there were many challenges with asynchronous requests. The front end was developed with static data to begin with, and then integrated with the server api. When we started to make actual calls to the database to display on the front end, we found that the page was rendering before the necessary data was returned from the request. This is a risk that we were not expecting, so we managed it by doing a lot of redesign to ensure the data was available before the relevant elements were loaded on the page.

2.7 Project Proposed Milestones and Evaluation Criteria

Version 1.0 - 12/4/2017 - Complete

The first milestone is the integration of the server, Raspberry Pi interfacing with the OBD-II, and the website displaying data for the fleet managers. This milestone was scheduled for completion by 12/4/2017, and involves the Raspberry Pi sending GPS information from a GPS board and gas and speed information from an OBD-II simulator to the server, which provides that data to the website along with fake data generated to demonstrate it working with multiple vehicles. We have completed the functionality for this milestone, and demonstrated it to our client on 11/30/2017. The Raspberry Pi data was verified by monitoring the calls that it was sending to the server and confirming that the data sent matched what was expected. The server was tested both for receiving data from the Raspberry Pi by showing that it received the data in the database, and that it provided the data to the website by testing the calls to retrieve the data from the server and making sure that the returned data was accurate. The website was tested by use, with the user interface being used to show data returned by the server and the data being displayed in the charts being checked for accuracy.

Version 2.0 - 4/23/2017 - Not Yet Started

The second and final milestone is the integration of the server, Raspberry Pi interfacing with the OBD-II, and the website displaying data for the fleet managers. While the first milestone was a simple proof of concept for the component interactions and the OBD-II connection, this milestone expects for their to be significantly more functionality to the fleet managers, with much more server processing of data and many more types of information to display. The Raspberry Pi 3 application and the server will be updated to work with real vehicles that may not work like the simulator, and a sufficient physical installation method for the Raspberry Pi 3 will be prepared. We have not yet decided which functionality will be added, but we plan to add at least some functionality related to identifying driver behavior, aggregating fleet data, and integrating vehicle tracking information with third-party functionality such as geofencing and mapping location information to roads. Additionally, we will look into adding driver functionality for the Raspberry Pi 3, but it might be out of budget or scope to get a screen for the car.

2.8 Project Tracking Procedures

Our group plans to use an Agile development method with weekly sprints for developing new features. Our sprints will correlate with the project timeline, which will show which weeks a task should be scheduled for and how long the tasks should last. Project tasks will be tracked as GitLab issues in the team's provided GitLab, as it allows us to have all of the features we need such as sprint tasking, task due dates and assignment, and comments for task completion that help us address issues that are created during development. We will work closely with our client to develop the schedule for next semester: we have yet to decide the functionality for the Version 2.0 fully, and once we have our specifications for that we can create and schedule the tasks needed to complete the project.

2.9 Objective of the Task

We want to create a system that efficiently gathers and transmits information from vehicles, processes and stores that data, and displays it in a meaningful way for fleet managers. Our project has several different components involved. Our main goal is to have all those components working and integrated with each other before we get to our final deadline. The final version of the product will be expected to have the Raspberry Pi prepared to be physically installed into a vehicle, the server configured to provide all functionality needed by the website and microcontroller and deployed to Google Cloud, and a website available through the team website provided by Iowa State for our senior design project.

2.10 Task Approach

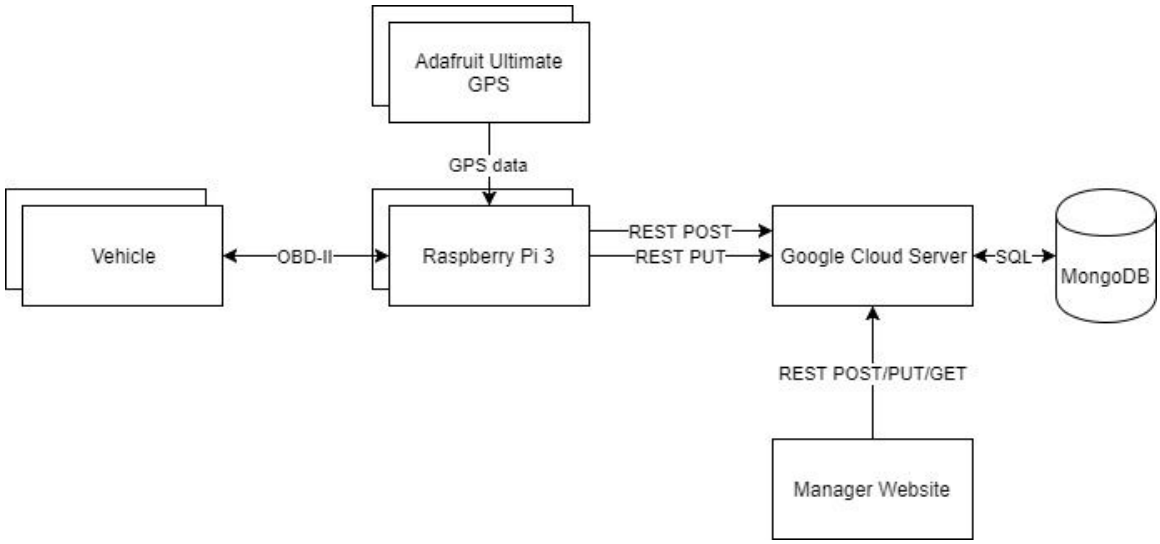

Figure 1: Diagram showing components and interfaces between components

The vehicles in the block diagram represent the vehicles in the fleet that have a Raspberry Pi 3 connected via the OBD-II port. Vehicles will respond to queries made over the OBD-II port to provide the data we need. The Raspberry Pi 3 will have a Python application that will take in data from the OBD-II port as well as additional sensors, including at least the Adafruit Ultimate GPS Breakout board, and send the important data to the server via Rest PUT calls. The Raspberry Pi 3 will register both the driver and the vehicle to the server via Rest POST calls. The server will be hosted using the Google Cloud platform. The server will receive data from the Raspberry Pi 3 and store the data in a database. This database is managed with MongoDB and Mongoose. The server will also receive commands from the website, such as creating new manager accounts via Rest POST calls and updating vehicle information via Rest PUT calls. The server will also send requested data to the website with Rest GET calls, which the website uses to provide graphical representation of fleet data such as Google Maps display for vehicle tracking and Chart.js visualizations of fuel consumption for a vehicle or fleet. The website will use the Angular.js framework.

2.11 Expected Results and Validation

Our desired result from our project is to have a Raspberry Pi that will connect to a vehicle and stream data to our dedicated server. We will have a manager website that can access information they want about their assigned vehicles. We will use manual test cases as well as automated cases to ensure the validity of our data on our server. We will have to go through the data that the raspberry pi is getting from the vehicle manually to ensure that it is collecting the correct data. Finally we will test our front end website to confirm that we are displaying the correct data that the final end user is requesting.

3 Estimated Resources and Project Timeline

3.1 Personnel Effort Requirements

Second Semester

Team Member	Effort Estimate (hours)	Explanation
Venecia Alvarez	68	Implementing register user functionality (8), login functionality (15), utilizing more AngularJS features (10), changes based off of midterm client feedback (35)
Kendall Berner	63	Implementing the page to edit a manager's fleet (18), improving look and feel of the website (5), determining and creating more useful charts (5), changes based off of midterm client feedback (35)
AJ Guss	71	Convert current API to Swagger API (8), update api to take in raw can data (10), create database model for valid PIDs for vehicles (8), implement data processing for new features for 2.0 release (10), changes based off of midterm client feedback (35)
Tyler Hartsock	65	Implement server side code for register and login functionality (10), create api functionality for editing fleets (20), changes based off of midterm client feedback (35)
Matthew Furhmann	68	Implement queries for new needed vehicle components (10), test accuracy of measured data (8), update code that sends data in server API format (5), check for supported PIDs of vehicle (5), fix start-up and close down of python application to prevent bugs (5), changes based off of midterm client feedback (35)
William Furmann	65	Figure out the packing and installation of the Pi into the vehicle (20), send raw CAN data to server from Pi (10), changes based off of midterm client feedback (35)

Figure 2: Effort Estimations

3.2 Other Resource Requirements

Raspberry Pi 3 - $50

PiCAN2 - $48

Adafruit Ultimate GPS - $30

USB to TTL cable - $10

3.3 Financial Requirements

Server Upkeep Cost

During development: 30-50 per month

Production: Estimated 120$ per month

Database Cost:

During development: 0$

Production: Estimated 180$ per month

3.4 Project Timeline

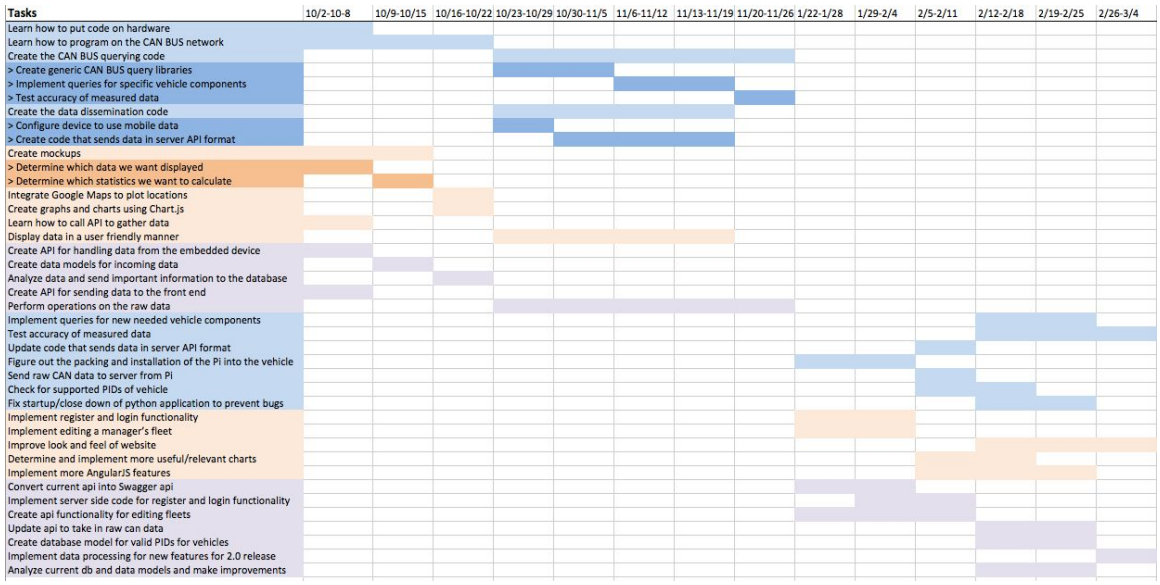

Figure 3: Project Timeline

3.4.1 First Semester

Tasks for embedded side (Matthew Fuhrmann and William Fuhrmann):

Learn how to put code on hardware (10/2-10/9)

Learn how to program on the CAN BUS network (small example prototypes for querying, network interaction/connection) (10/2-10/23)

Create the CAN BUS querying code (10/23-11/27)

Create generic CAN BUS query libraries (10/23-11/6)

Implement queries for specific vehicle components (11/6-11/20)
Test accuracy of measured data (11/20-11/27)
Create the data dissemination code (send data to server) (10/23-11/20)
Configure device to use mobile data (10/23-10/30)
Create code that sends data in server API format (10/30-11/20)

Tasks for front end (Venecia Alvarez and Kendall Berner):
Create mockups (10/2-10/16)
Determine which data we want displayed (10/2-10/9)
Determine which statistics we want to calculate (10/9-10/16)
Integrate Google Maps to plot locations (10/16-10/23)
Create graphs and charts using Chart.js (10/16-10/23)
Learn how to call API to gather data (10/2-10/9)
Display data in a user friendly manner (10/23-11/20)

Tasks for the server (Anthony Guss and Tyler Hartsock):
Create data models for incoming data (10/2-10/16)
Create API for handling incoming data from the embedded device (10/9-10/16)
Analyze data and send important information to the database (10/16-10/23)
Create API for sending data to the front end (10/2-10/9)
Make improvements to models and data to contain more relevant data for the website (10/23-11/20)

3.4.2 Second Semester

Embedded (Matthew Fuhrmann and William Fuhrmann):
Implement queries for new needed vehicle components (2/12-2/26)
Test accuracy of measured data (2/12-3/5)
Update code that sends data in server API format (2/5-2/12)
Figure out the packing and installation of the Pi into the vehicle (1/22-2/5)
Send raw CAN data to server from Pi (2/5-2/12)
Check for supported PIDs of vehicle (2/5-2/19)
Fix start-up and close down of python application to prevent bugs (2/12-2/26)

Front-end (Venecia Alvarez and Kendall Berner)
Implement register and login functionality (1/22-2/5)
Implement editing a manager's fleet (1/22-2/5)
Determine and implement more useful/relevant charts (2/5-2/19)
Implement more AngularJS features (2/5-2/26)
Improve look and feel of website (2/12-3/5)

Server (Anthony Guss and Tyler Hartsock):

- Convert current api into Swagger api (1/22 - 1/29)
- Implement server side code for register and login functionality (1/29-2/12)
- Create api functionality for editing fleets (1/22-2/12)
- Update api to take in raw can data (2/12-2/26)
- Create database model for valid PIDs for vehicles (2/12 - 2/26)
- Implement data processing for new features for 2.0 release (2/26-3/4)
- Analyze the current database and data models and make improvements (1/12-2/26)

The last couple months of the project have not been concretely determined yet. This is because we plan to use that last time to work closely with our client to ensure everything we have worked on in our current schedule is the desired functionality. At that point, we will use our remaining time to develop more features the client would like, or make changes that he feels necessary. This extra time will also be useful in case our current plan underestimates the actual development effort.

4 Closure Materials

4.1 Conclusion

The goals for our team are to create a viable fleet monitoring system that provides managers with relevant information on the vehicles in their fleets by collecting data from the vehicles, and to gain more experience with the development process.

We have successfully created prototypes for the map and various charts that we would like to have on our fleet manager dashboard. At this point, the client can make calls to the server API to grab data and display that data on the front-end.

We have created a prototype Raspberry Pi application that can forward interpreted OBD-II data and GPS data to the server. This application uses libraries such as gpsd, Requests, and PyCan to interface with hardware such as a PiCan2 and a Adafruit Ultimate GPS board to acquire information. There are currently three Python modules developed for sending data to the server, querying the GPS, and querying the OBD-II port.

We have successfully created a prototype for our server. This includes basic API calls that allow incoming data from the Raspberry Pi device, and outgoing data to the website. We have created a good base for our server, and it will be easy to add new data as well as new API calls to implement new features going forward.

This is the best possible plan because it helps us create the framework to provide managers information while focusing on the aspects that will be the most difficult and are causing us the most uncertainty for estimation. The success of our initial prototypes gives us confidence in our ability to provide a valuable product to our client by the end of the

project using our market research and client interaction to determine which features are most important to add to our product throughout the second semester.

4.2 References

Introduction to AngularJS. [Online]. https://www.w3schools.com/angular/angular_intro.asp. [Accessed: 2-Oct-2017].

AngularJS API [Online]. https://docs.angularjs.org/api. [Accessed 2-Oct-2017].

Chart.js. [Online]. http://www.chartjs.org/. [Accessed: 30-Sept-2017].

Google Maps API. [Online]. https://developers.google.com/maps/documentation/javascript/tutorial. [Accessed: 30-Sept-2017].

High Point GPS. [Online]. Available: http://www.highpointgps.com/. [Accessed: 10-Sep-2017].

Fleet Complete Fleet Tracker. [Online]. Available: https://www.fleetcomplete.com/en/products/fleet-tracker/. [Accessed: 10-Sep-2017].

Fleetmatics Now. [Online]. Available: https://www.fleetmatics.com/now. [Accessed: 10-Sep-2017].

collectiveFleet. [Online]. Available: http://www.collectivedata.com/fleet_management_software.html. [Accessed: 11-Sep-2017].

ManagerPlus. [Online]. Available: http://www.managerplus.com/lps/managerplus. [Accessed: 11-Sep-2017].

Rhino Fleet Tracking. [Online]. Available: http://www.rhinofleettracking.com/. [Accessed: 11-Sep-2017].

Linxup. [Online]. Available: http://www.linxup.com/. [Accessed: 11-Sep-2017].

Silent Passenger. [Online]. Available: https://vehicletracking.com/features/. [Accessed: 11-Sep-2017].

Azuga Fleet. [Online]. Available: https://www.azuga.com/. [Accessed: 11-Sep-2017].

Onfleet. [Online]. Available: https://onfleet.com/. [Accessed: 11-Sep-2017].

Fletilla FleetFACTZ. [Online]. Available: https://fleetilla.com/products/fleet-management-solutions/real-time-tracking. [Accessed: 11-Sep-2017].

Fleetio. [Online]. Available: https://www.fleetio.com/. [Accessed: 11-Sep-2017].

MongoDB. [Online]. Available https://www.mongodb.com/. [Accessed: 15-Sep-2017]

Swagger. [Online]. Available https://swagger.io/. [Accessed 28-Nov-2017]

Google Cloud Platform Documentation. [Online]. Available https://cloud.google.com/docs/. [Accessed 10-Sep-2017]

PiCan2 User Guide. [Online]. Available http://skpang.co.uk/catalog/images/raspberrypi/pi_2/PICAN2UGB.pdf. [Accessed 15-Nov-2017].

Adafruit Ultimate GPS Documentation. [Online]. Available https://www.adafruit.com/product/746. [Accessed 17-Nov-2017].