Fleet Monitoring System

DESIGN DOCUMENT

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: 12-3-2017/2.0

Table of Contents

1 Introduction

1.1 Project statement

Our project is to make a fleet monitoring system. We have been provided a Raspberry Pi with a PiCAN2 board 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.

1.2 Purpose

The purpose of this project is to give fleet managers a better way of tracking information on the vehicles in their fleet. Fleet managers often find it difficult to know the current location of their vehicles or where their vehicles have been, and knowing this information helps them ensure that their vehicles are being used correctly and efficiently. Fleet managers also need better ways of predicting maintenance needs and vehicle operation costs. Data on fuel consumption and vehicle diagnostics can help fleet managers predict the needs of their vehicles. Having statistics on fleet operation will help fleet managers who are trying to improve the efficiency of their fleet.

1.3 Goals

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. In regards to soft skills, our team has set a goal to learn more about project management and agile development. Through this project experience, we will see what the entire software development process and life cycle really look like.

2 Deliverables

Our project will have three components. The first is the Python application for the Raspberry Pi that will get data from the OBD-II port of a vehicle and forward it to the server for processing. The second will be a server that will take in data sent by the Raspberry Pi and store it, as well as provide necessary data to the website. The third is a website that will serve as a dashboard for fleet managers to be able to view specific data about their vehicles.

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 API will be implemented using Swagger. This will allow easy documentation of our API. 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 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.

3 Design

3.1 System Specifications

Our back-end is required to be written in Node.js.

3.1.1 Non-functional

The product shall:

- Be used by vehicles at any time and location
- Utilize Google Cloud services
- Only allow managers to view fleet data on the dashboard
- Only allow managers to view vehicles in their fleet
- Have the server side code made in Node.js
- Use AngularJS on the client side

3.1.2 Functional

The product shall:

- Gather data from a vehicle's ODB-II port
- 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

3.1.3 Standards

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.

3.2 Proposed Design/Method

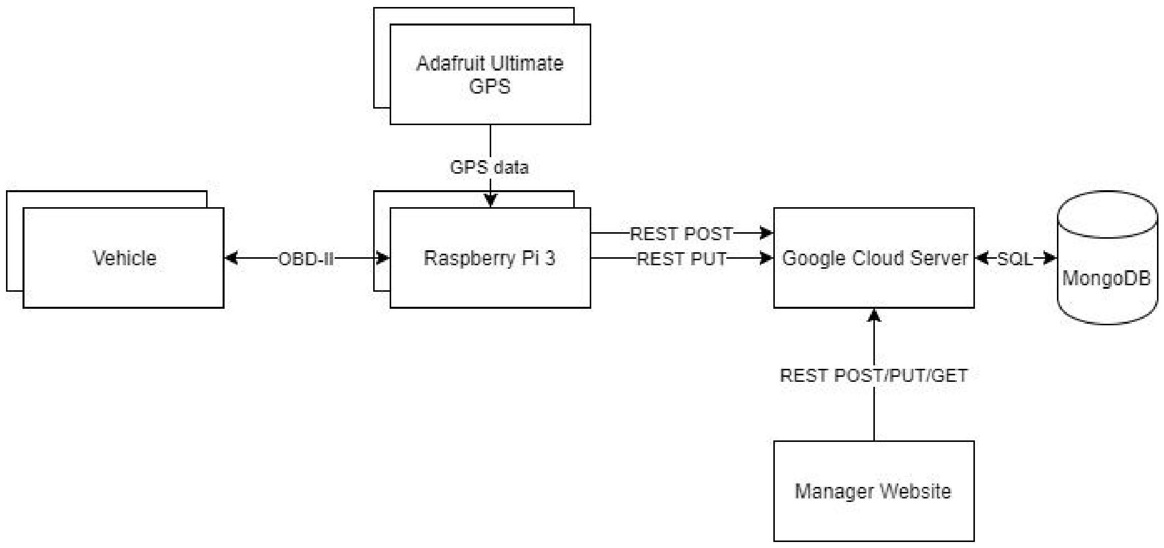

Figure 1: Diagram showing components and interfaces between component

Our team will develop three different components that will provide the manager with the information that they need on their fleet: an on-board Raspberry Pi microcontroller that will query the OBD-II port of the vehicle and forward the data it collects to a server, a server used to handle incoming data from the Raspberry Pi device and send necessary data to the website, and a website that will be the interface that shows important information to a manager about their fleet. The server is required to be in Node.js, and will provide APIs for putting data into the database for vehicles and APIs for the front-end to get the information that they need. The front-end will be an AngularJS website that presents location data on a map and vehicle and fleet statistics using charts and graphs.

The on-board Raspberry Pi device uses the Adafruit Ultimate GPS to get location data. The on-board Raspberry Pi device does have the capability to use Wi-Fi that can be provided by a mobile phone's Wi-Fi hotspot feature. The Raspberry Pi 3 will have several Python modules for sending data to the server and for interfacing with the OBD-II port and the GPS board.

The server will be designed to be extremely modular. This will be done by separating our implementation of the API into a model, a controller and a route to the API. This will allow each implementation of the API to easily be modified, and will make it easy to add new API calls to the server. We have created an example API that can be used as an example of how to implement a new API call. Implemented APIs are a Vehicle API and a Manager API. The server will handle incoming data from the Raspberry Pi device through POST and PUT API calls. The server will send data to the web front end through GET and DELETE API calls.

3.3 Design Analysis

We originally planned to use AngularJS on our front-end. This has proven to be a slight challenge, and our team is considering if it is necessary to use this framework at all anymore. We plan on continuing to experiment with AngularJS and trying to integrate it with our existing prototypes for the map and charts for at least another week. Our plan is to pivot towards a different framework if AngularJS seems to be more trouble than it's worth moving forward.

The work for the Raspberry Pi on-board device so far has mostly entailed understanding the hardware, finding out what other hardware we need, and writing Python code to query the OBD-II port. We have prototyped data querying functionality including querying speed and gas usage from the OBD-II port.

We originally intended to implement our server using a micro-service architecture using Java and Spring. However, our client made it a requirement that our server be written in Node.js. Because of this, we lost a week of time researching and prototyping the old server. After transitioning into a Node.js server, we were able to prototype our API that will be used by the Raspberry Pi device and the Website. Currently, the API is using a basic model for the data. The Vehicle model currently logs speed, location, and gas usage data. This model will be changed and improved as we add new features going forward.

4 Testing/Development

4.1 Interface Specifications

We are using an Raspberry Pi microcontroller to query data off of the OBD-II port of a vehicle. We will develop an Raspberry Pi app that will query the OBD-II port for useful information and send that data along with location data from the Raspberry Pi device to our google cloud server.

There is fairly extensive interface definitions for the server on the first part of the project available at http://sdmay18-18.sd.ece.iastate.edu/swagger-ui-master/swagger-ui-master/dist/index.html.

4.2 Hardware/Software

We will use Python unitest.py to unit test for the Raspberry Pi application. We will also use a OBD-II simulator that allows the application to query for dummy data and ensure that all the parts are functioning as intended. We will verify that the data retrieved from the OBD-II port for real vehicles matches the data expected based on what we record was happening while driving the vehicle at that time.

In order to test our server, we will be using Postman to verify our API and perform regression testing. Postman allows us to setup automated tests that will run whenever we update the API for the server. These tests will allow us to easily verify our API implementation. We will also create unit tests for the serve once we begin implementing data analytics.

4.3 Process

The Raspberry Pi has currently been tested mostly for valid GPS coordinate and simulator readings. These have been verified easily because it is trivial to know your GPS coordinates and the simulator can be programmed to return whatever values you want, making it easy to test different edge cases for values or variable values returning from the OBD-II.

The server was tested using Postman. This allowed us to validate that the API calls were working correctly. We used Postman to setup automatic tests to ensure that the API calls remain correct. This is done by calling a specific API call and validating the response from the server. These tests will grow more complex as our models and data grow as we implement new features. Our API documentation will be created using Swagger. Swagger allows us to create documentation with an easily readable UI. Swagger will also allow people to test calls directly from the UI.

Front-end code is always first written and tested locally, using UwAmp and XAmpp to simulate the code being run on a proper server. Testing is performed manually, most often by the developer. Testing is very frequent, with the page being reloaded with every small change to the codebase to make sure everything still works properly. The developer console on the browser is also useful for client side testing and debugging. Logging helpful messages to the console helps to confirm the desired functionality is working. When deploying new changes to our hosted website, manual regression testing is done on the interface to ensure that everything deployed correctly and successfully and that all other parts of the application still work as expected.

5 Results

On the front end, we have successfully created several prototypes of what we like our final dashboard for fleet managers to look like. It will use the Google Maps API to display a map with the location of all of the vehicles in the fleet. It will also use Chart.js to make multiple different types of charts that will display vehicle/fleet statistics. So far, prototypes of maps and charts have been made that can be modified going forward to meet the specific requirements of our front-end and to use real data from vehicles. One challenge we recently faced was setting up our environment to run our server in order to hit our API endpoints on the front-end. Through collaboration and research, we were finally able to get our application up and running. Another challenge we are facing on the front-end is using AngularJS to its full capabilities. To progress in this area, we plan on doing a lot more research and practicing AngularJS with examples we find online.

As far as Raspberry Pi programming, we have created a prototype application that gets latitude and longitude and a prototype application that builds a JSON object from dummy data gathered from the CAN tester and sends it to the server via POST API. This should give us a good foundation for when we use the OBD-II data from a real vehicle. To complete this part of the project, we need to make design decisions as to what types of data the fleet manager will want to see from the vehicle.

On the server, we have successfully prototyped some basic API calls. The first API call allows the Raspberry Pi device to send data containing the date and time, longitude and latitude, and speed. The Raspberry Pi device also sends a unique identification number (UID) with this data which is used to assign the data to a specific vehicle. Using a GET API call, the website can access this data based upon the UID. In order for the data to make sense for a specific manager, we created a manager model. This model contains a login system for managers to login, and shows which vehicles the manager owns. This will allow the website to pull the specific vehicle data for a manager from the server. We have also implemented the ability to delete vehicles, both from the data set and from a manager. This allows old data to be removed if, for example a vehicle has been sold or is no longer in use. The data used for the vehicles is currently minimal containing only location, speed and date. This will expand as we add new features. For example, a feature we want to add is fuel management. To do this, we will have to have data regarding the current levels of fuel. Once we have this data, the server will accept this data and make it available to the website. The website will in turn use this data to provide the manager information about fuel usage.

6 Conclusions

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.

The successes of our prototypes to this point give us confidence in our design going forward. With the basic implementation of our components completed, we have lots of versatility in expanding and improving the functionality of our system to provide value to our client and intended users.

7 References

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].
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].