Fleet Monitoring System

sdmay18-18

http://sdmay18-18.sd.ece.iastate.edu/

Venecia Alvarez, Kendall Berner, Matthew Fuhrmann,
William Fuhrmann, Anthony Guss, Tyler Hartsock

Client/Advisor: Lotfi Ben-Othmane

Problem Statement

Problem:

- Managing large fleets of vehicles is costly and time-consuming
- Fleet managers don’t have information to make improvements

Solution:

- Vehicle location data and internal data gathered from on-board Raspberry Pi 3
- Server processes vehicle data, provides data to fleet manager website
- Allow a fleet manager to see real time data of vehicles

Market Survey

Other Fleet Management Applications:

- OBD II or Mobile App
- Live tracking, statistics, vehicle data
- Live map of fleet

Our Application vs Rest of Market:

- Minimize driver interaction
- Useful interpretations of vehicle internal data
- Less focus on fleet administration, more on fleet monitoring

Functional Requirements

The product shall:

- Gather data from a vehicle's OBD-II (On-Board Diagnostics) 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 live data for a certain vehicle (speed, engine temperature)
- Allow managers to register or remove vehicles that belong to a particular fleet
- Allow managers to customize the data being displayed to them

Non-Functional Requirements

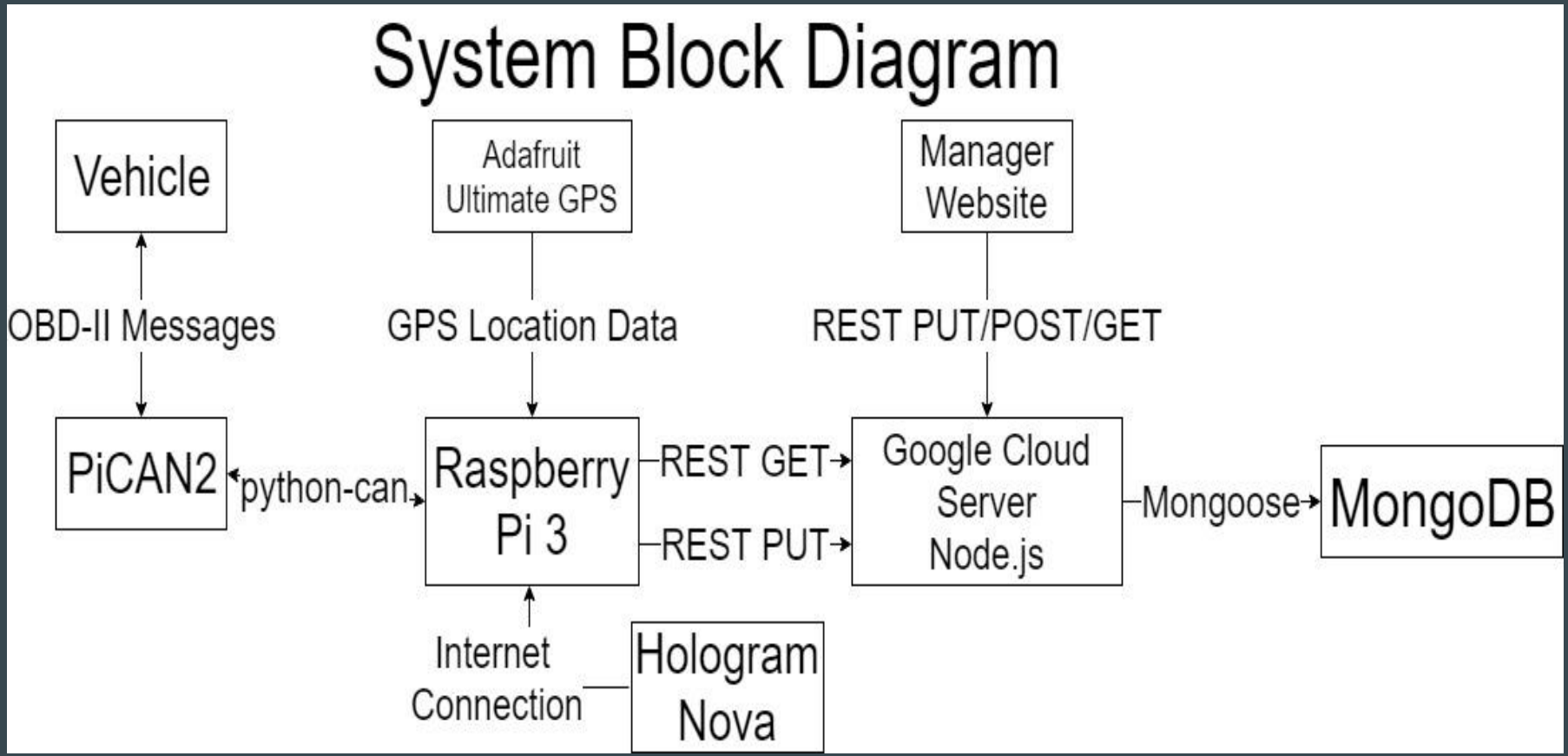
The product shall:

- Only allow managers to view fleet data on the website
- Only allow managers to view vehicles in their fleet

Constraints

- Have the server side code made in Node.js
- Utilize Google Cloud services

System Block Diagram



Detailed Design - Front End

- Emphasis on visualizing data on the main page
- Other pages
  - Login
  - Register
  - Edit fleet
  - Edit view
- Technologies Used
  - AngularJS
  - Chart.js
  - Google Maps API
  - Bootstrap

Detailed Design - Server

Technologies Used: NodeJS, MongoDB, Mongoose, Bcrypt.js

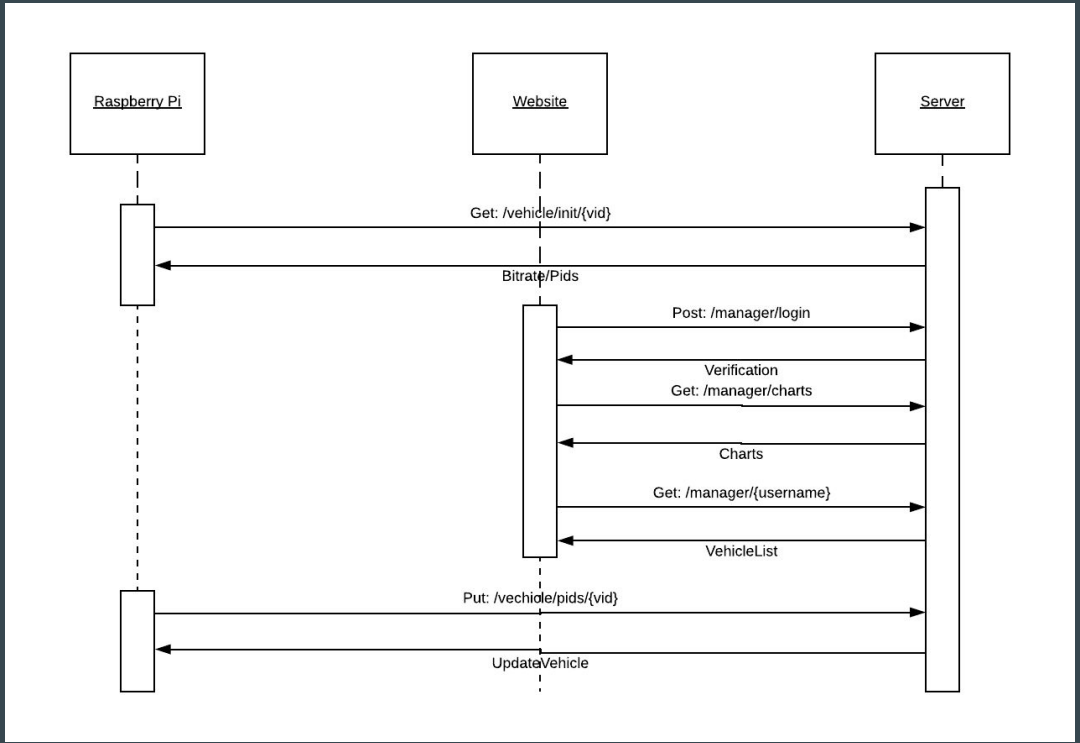
RESTful API

PID Processing

Swagger Documentation

Hosted on Google Cloud Compute Engine

Server Sequence Diagram



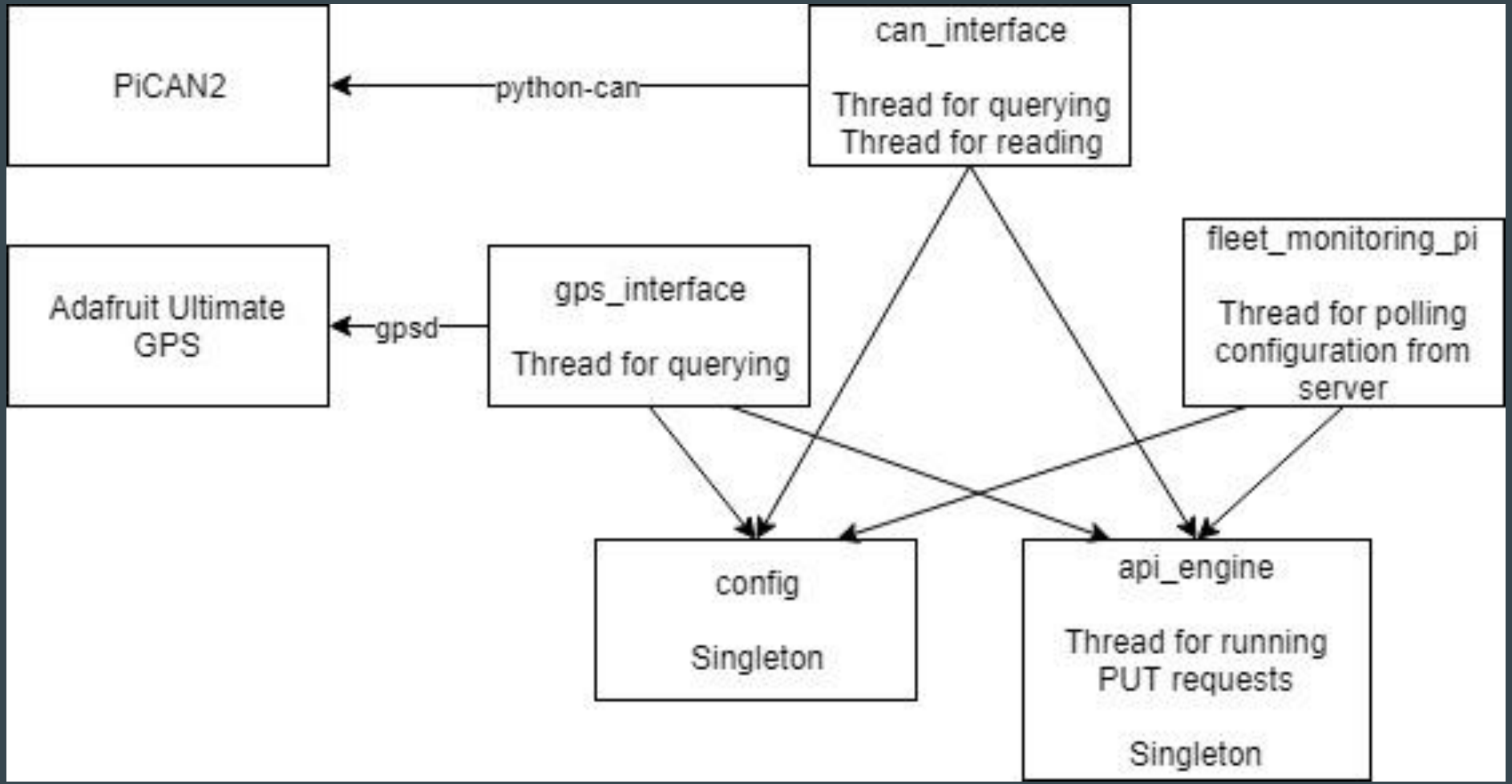
Detailed Design - Raspberry Pi

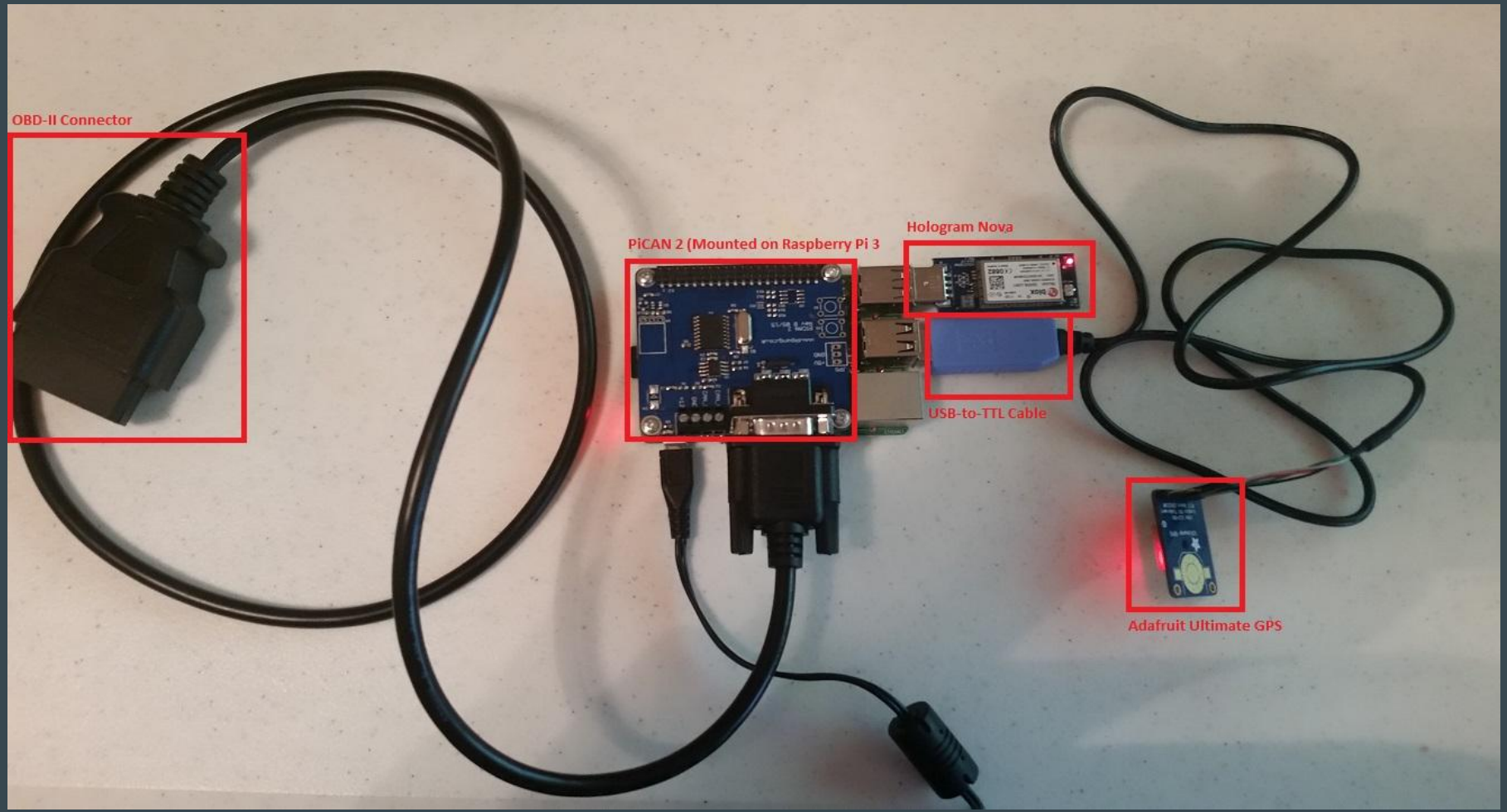
Hardware Used: PiCAN2, Adafruit Ultimate GPS, Hologram Nova

Technologies Used: Python, gpsd, gpsd-clients, python-gps, python-can, requests

Modules:

- fleet_monitoring_pi: Startup the application, poll configuration in a thread
- can_interface: Use python-can to interact with OBD-II port using threads
- Gps_interface: Use python-gps to interact with gpsd reports using a thread
- api_engine: Singleton, handle API calls
- config: Singleton, handle configuration storage based on polling, startup arguments


PiCAN2
python-can
can_interface
Thread for querying
Thread for reading
Adafruit Ultimate GPS
gpsd
gps_interface
Thread for querying
fleet_monitoring_pi
Thread for polling configuration from server
config
Singleton
api_engine
Thread for running PUT requests
Singleton


OBD-II Connector
PiCAN 2 (Mounted on Raspberry Pi 3
Hologram Nova
USB-to-TTL Cable
Adafruit Ultimate GPS

Testing Plan

- Front End: UwAmp/XAmpp for local testing, confirm database updates show up live on the website
- Raspberry Pi: Verify hardware functionality by running Python modules individually, verify that api_engine results are correct by comparing with Postman calls. OBD-II testing done by OBD-II simulator and rented U-Haul van, as well as overall system testing using the real vehicle
- Server: Automated API Testing Using Postman

Testing Results

GPS module tested in operating vehicle - successful

OBD-II module tested in operating vehicle - successful

Raspberry Pi data sent from operating vehicle using Hologram Nova - successful

Website tested to confirm live update of location and statistics - successful

Server tested using Postman runners for all API calls - successful

Alternative Designs

- Java Spring Microservices
- Android Microcontroller
- R Data Analytics
- Google Roads API

Questions?