

Software Design Document

Los Angeles County Fleet Management System

(LACFMS)

Software Design Document
Version 1.0

Prepared by

Flinner, Patrick
Ghazarian, Avik
Karapetyan, Robert
Romero, Aaron
Shimauchi, Ken
Valadez, Benjamin



California State University of Los Angeles
April 10, 2017

Los Angeles County Fleet Management System

Software Design Document
Version 1.0

Prepared By:

Flinner, Patrick
Ghazarian, Avik
Karapetyan, Robert
Romero, Aaron
Shimauchi, Ken
Valadez, Benjamin



Approved By:

Dr. Russ Abbott

Date

Dr. Sue Lim

Date

Los Angeles County Liaison

Date

Table of Contents.....	pg. 2
Revision History.....	pg. 3
1. Introduction.....	pg. 4
1.1. Purpose.....	pg. 4
1.2. Document Conventions.....	pg. 4
1.3. Intended Audience and Reading Suggestions.....	pg. 4
1.4. System Overview.....	pg. 5
Design Considerations.....	pg. 6
2.1. Assumptions and dependencies.....	pg. 6
2.2. General Constraints.....	pg. 7
2.3. Goals and Guidelines.....	pg. 8
2.4. Development Methods.....	pg. 8
Architectural Strategies.....	pg. 9
System Architecture.....	pg. 11
4.1.	pg. 11
4.2.	pg. 12
Policies and Tactics.....	pg. 15
5.1. Specific Products Used.....	pg. 15
5.2. Requirements traceability.....	pg. 15
5.3. Testing the software.....	pg. 16
5.4. Engineering trade-offs.....	pg. 20
5.5. Guidelines and conventions.....	pg. 21
5.6. Protocols.....	pg. 21
Detailed System Design.....	pg. 23
6.x Name of Module	
6.x.1 Responsibilities	
6.x.2 Constraints	
6.x.3 Composition	
6.x.4 Uses/Interactions	
6.x.5 Resources	
6.x.6 Interface/Exports	
Detailed Lower Level Component Design.....	pg. 32
7.x Name of Class or File	
7.x.1 Classification	
7.x.2 Processing Narrative(PSPEC)	
7.x.3 Interface Description	
7.x.4 Processing Detail	
Database Design.....	pg. 33
User Interface.....	pg. 34
Requirements Validation and Verification.....	pg. 41
Glossary.....	pg. 44
Hardware Schematics.....	pg. 47
References.....	pg. 51

Revision History

Name	Date	Reason For Changes	Version
LACFMS	1/18/2018	Initial draft in progress	1.0
Updated SDD	4/10/2018	Final Draft	1.1

1. Introduction

Parks and Recreation has a fleet of 600 vehicles, and it is made up of different makes and models. Currently all vehicles use paper logs for the purpose of tracking vehicle usage and safety checks. This process can be time consuming for the drivers, service personnel and relies on the accuracy of manually supplied information. The Los Angeles County Department of Parks and Recreation has teamed up with California State University: Los Angeles for a multi-year project to combat these issues.

An automated system will be developed to retrieve information from County motor vehicles. This system is intended to eliminate the paper based process, improve reporting, and enhance vehicle maintenance capabilities. An authentication component is also part of the interface for every vehicle. Only employees of Los County Parks and Recreation should be driving County vehicles, so all trips that are completed by unauthorized drivers will be flagged. A later system will prevent the vehicle from being used. Implementation of this system should reduce paper waste footprint, improve County resources, improve control over maintenance costs, and is a simplified process for authorizing and reporting usage of County vehicles.

1.1 Purpose

This document contains the software design specification for Los Angeles County Fleet Management System application project. The document is similar to IEEE specification 1016, with slight modifications to subsections which are for clarification purposes. All information here is prioritized and has not been committed for release. This is version 1.0 revision N/A at this time. Further, this document is part of another document, the Software Requirements Specification which will be the governing requirements for the implementation described here. Currently, this design specification will be broken up into three major phases and is currently focusing on the the edge and platform phase. The Edge phase of the project will include, hardware prototyping and data ingestion of sensory data, which will be stored locally on Edge devices. The Platform phase of the project will focus on the communication between Edge devices to infrastructure, secure messaging, and data accessibility and storage. Lastly, the Enterprise phase will focus on predictive analytics on stored datas.

1.2 Document Conventions

This document follows MLA Format. Bold-faced text has been used to emphasize section headings of each subsection . Italicizing text words is to point out words that have been defined in the glossary. Every requirement statement in this document will have its own priority.

1.3 Intended Audience and Reading Suggestions

This document is intended for developers, testers, and project managers who wish to read it. This document is arranged according to standard IEEE specification 1016. **Bold** typeface is used to highlight subsections. *Italics* are used to emphasis words that have been defined in the

Glossary. Further, the software document is intended to be used by members of the project team that will implement and verify the correct functioning of the system.

1.4 System Overview

The Los Angeles County Fleet Management System, abbreviated (LACFMS), is a software application used to monitor and report vehicle usage. The edge, is comprised of hardware components and is managed by a data flow file management technology. The *LACFMS* shall interface with these hardware components to extract, store, and transmit vehicle diagnostics and *GPS* data using *IoT* best practices. The web application shall act as an intermediate allowing employees of Los Angeles County to submit information as a replacement to the County Vehicle Mileage and Safety Check Form. Employees shall authenticate using a web interface. Only Managers will be permitted to view detailed trip descriptions and trip visualizations. This document is currently in the Edge phase of implementation which focuses on data ingestion and data storage, while the Platform and Enterprise phases will be postponed until the Edge is fully functional. The Platform interface, Apache NiFi and Apache MiNiFi should be configured in this phase to communicate with one another to transfer data securely from edge device to infrastructure via site-to-site transmission. Data transferred should be stored in a reliable data storage for accessibility. The Enterprise phase should integrate data analytics.

2. Design Considerations

- 2.0.1 RFID device operating on 125kHz which is compatible with HID badge system the County currently uses.
- 2.0.2 Storing sensory data on a Raspberry Pi 3 sim card, for an extended period of time, may overload the capacity of the storage device.
- 2.0.3 Transmitting large files may cause the devices to overheat due to load.
- 2.0.4 Transmitting large files may take considerable amount of time while at County Facility.
- 2.0.5 Loss of power may corrupt file system.
- 2.0.6 Device failure may cause certain data to become out of sync

2.1 Assumptions and Dependencies

Software Dependencies

- 2.1.1 Software libraries currently used by edge devices may need to be reconfigured on newer devices due to hardware changes and software changes.
 - Raspberry PI3 Model B uses bluetooth on serial port ttyAMA0, which interferes with NEO 6M GPS hardware. Consequently, NEO 6M GPS hardware needs to be configured on a different serial port, for example, use serial port ttyS0, or another available serial configuration.
- 2.1.2 Operating System for Edge Devices Raspberry PI 3 Model B
<https://www.raspberrypi.org/downloads/> version 2.4.4
- 2.1.3 Python-OBD library
<https://python-obd.readthedocs.io/en/latest/>
- 2.1.4 GPSD library
<https://github.com/wdalmut/libGPS>

- 2.1.5 Apache NiFi version 1.40 (current)
<https://cwiki.apache.org/confluence/display/NIFI/Release+Notes>
- 2.1.6 Apache MiNiFi version 0.20(current)
<https://cwiki.apache.org/confluence/display/MINIFI/Release+Notes>
- 2.1.7 Raspberry Pi3 Model B device for sensory data collection and storage uses a file flow management technology Apache MiNiFi, this technology is better equipped on Edge devices than Apache NiFi due to the constrained hardware limitations. Apache MiNiFi, is a reduced set of Apache NiFi, as the name suggests it requires less hardware resources and is better suited for hardware constrained devices.
- 2.1.8 Operating System on Raspberry Pi 3 Model B is Raspbian, this software is currently open-source but may change significantly enough over time to have an effect on used libraries and software dependencies. This may have a negative impact on maintainability of the products life cycle.

2.2 General Constraints

- 2.2.1. RFID capabilities which enable or disable vehicle starter are not feasible, all available resources suggest that such a feature would require interrupting the vehicle starter system which requires vehicles to have wires cut in order to successfully implement.
- 2.2.2. Apache MiNiFi is to be housed on Edge devices which it is designed to do; however, considerations need to be taken to ensure proper shutdown and to ensure cold start operations do not corrupt the files system of the device.
- 2.2.3. Software environment on the Edge needs to support batch transmission in the event that streaming data is not possible due to network connectivity interruptions.
 - Storing sensory data locally for an unknown time for an available network to become available may not be feasible, considerations on stored data formats need to be tested to see what data format uses the least amount of storage capacity.
- 2.2.4. Considerations must be taken when transmitting large amounts of data.
 - 2.2.4.1 Overheating on Edge Device, time frame of how long it will take to transmit.
 - 2.2.4.2 Difficulty may arise in targeting a software implementation for Toyota Prius 2007. Feedback is necessary from the vehicles that are currently being tested.

2.2.4.3 Currently storage volatility does exist, Specifically, data is stored on local micro-sd, it has not had a reoccurring effect on project test runs. Cautions needs to be taken when powering off the Raspberry Pi 3.

2.2.4.4 No subscription based internet providers for the edge device

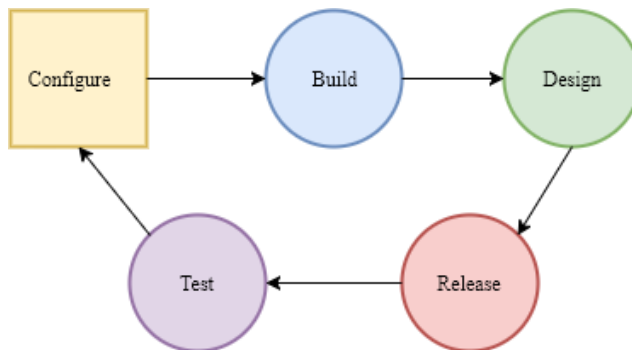
2.3 Goals and Guidelines Requirements

2.3.1 Proof of Concept prototype has an estimated delivery date April 2018.

2.3.2 The Software requirements specification is a guideline to the implementation of the SRD.

2.4 Development Methods

Agile Development methods were used in order to promote rapid development in a constrained time frame. While the waterfall paradigm approach uses a similar iterative process as Agile methodologies, the tendencies for waterfall methodology tend to be more rigid and less flexible which causes software development and deployment to be slower and costly. This project employs a flexible methodology, where prototyping hardware has significant impact on design and deadline. Common practice is as follows.



1. Configure:
2. Build:
3. Design:
4. Release:
5. Test:

3. Architectural Strategies

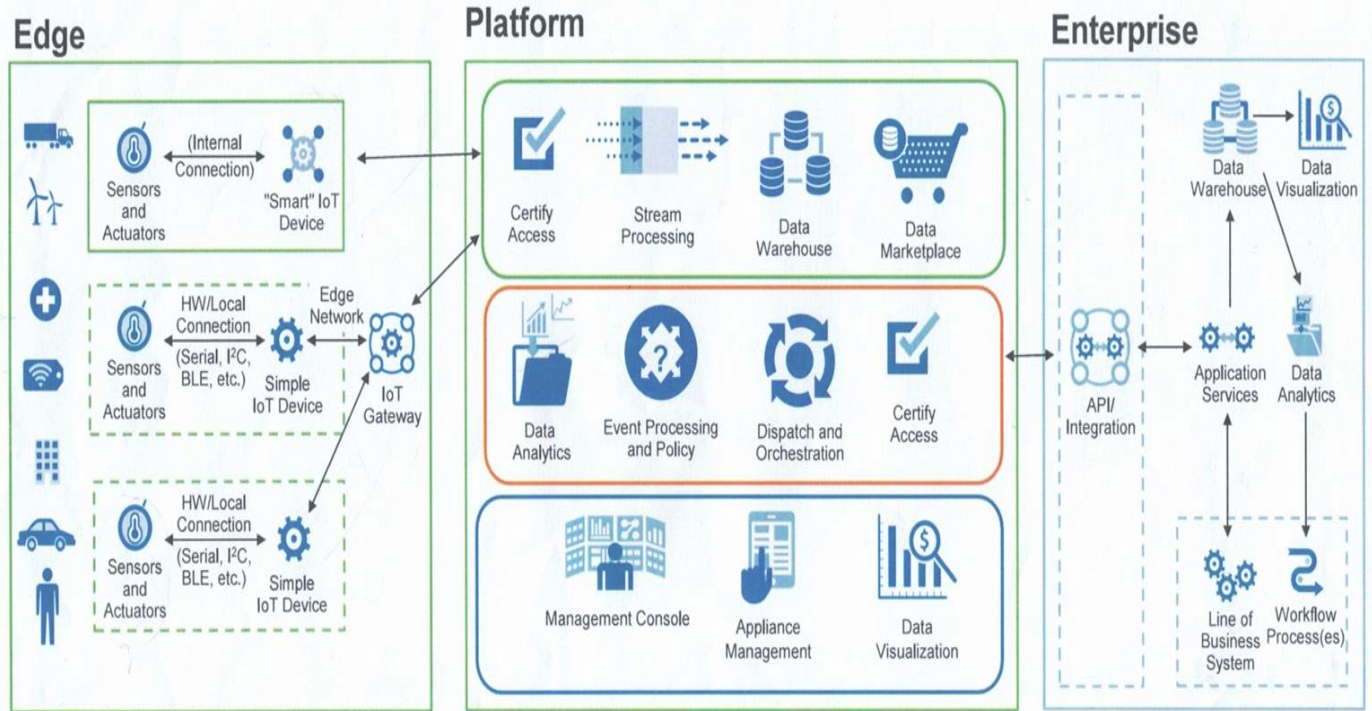
The system architecture has been divided into three main build phases.

- 3.0.1 The Edge phase includes hardware devices at the edge for data ingestion and temporary storage to support batch transmission.
- 3.0.2 The Platform phase includes platform technology, Apache NiFi to be configured to communicate with IoT gateway, Raspberry Pi 3 Model B using Apache MiNiFi. These technologies assist data flow, simplifying the transit of stored data from ingestion to Infrastructure.
- 3.0.3 The Enterprise phase
Apache NiFi configured to transmit to Apache Spark or Maximo.(This is still being evaluated)
- 3.0.4 Design Decision
 - 3.0.4.1 Design Decisions were made to use a file flow management technology, Apache NiFi in conjunction with Apache MiNiFi. These technologies will be used to manage the flow of data from Edge devices to Infrastructure, e.g. The Platform.
 - 3.0.4.2 *GPS* is collected every time a certain distance between two points has been reached to reduce storage capacity.
 - 3.0.4.3 The predominate computer language used for scripting is the python language since it was a native language to the Raspberry PI operating system.
 - 3.0.4.4 C-Sharp is used for use of a RESTful Service.
 - 3.0.4.5 A standalone Microsoft SQL database is used to store key-value of pairs which pertain to *GPS* data and County Safety Data. This is being used to improve data input from Apache NiFi.
- 3.0.5 The Raspberry Pi Model 3 was chosen due to it having more processing power than other microcontrollers.
- 3.0.6 WaveID is used because it is compatible with 125kHz HID badges

- 3.0.7 Microsoft SQL Server is used because Los Angeles County Department of Parks and Recreation utilizes Microsoft products.
- 3.0.8 The Raspberry Pi will have no user interface, outside of the RFID scanner.

4. System Architecture

4.1 Architectural Overview



The Edge

The Edge represents hardware components used for data ingestion. The Edge devices operate two fold, gathering sensory data; *GPS* and Vehicle Actuator data will be the main focus. This data is then stored and made available for batch transmission. Streaming data should be considered as an alternative method of transmission which should not be implemented at this time. The microdevice is configured to operate as a server using (Apache MiNiFi) which then encrypts and broadcasts the data. The data is consumed in another instance of the File flow manager technology (Apache NiFi).

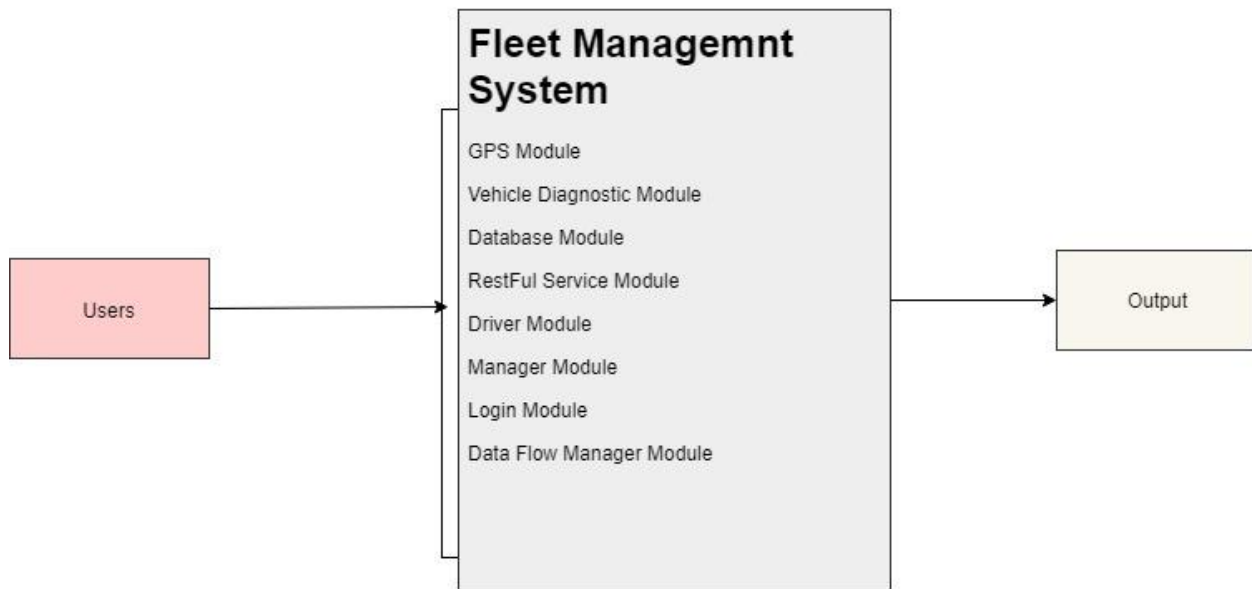
The Platform

The platform consumes data being sent by devices on the edge. Data transmission should include a secure message protocol. Data stored in platform should be accessible through a web interface.

The Enterprise

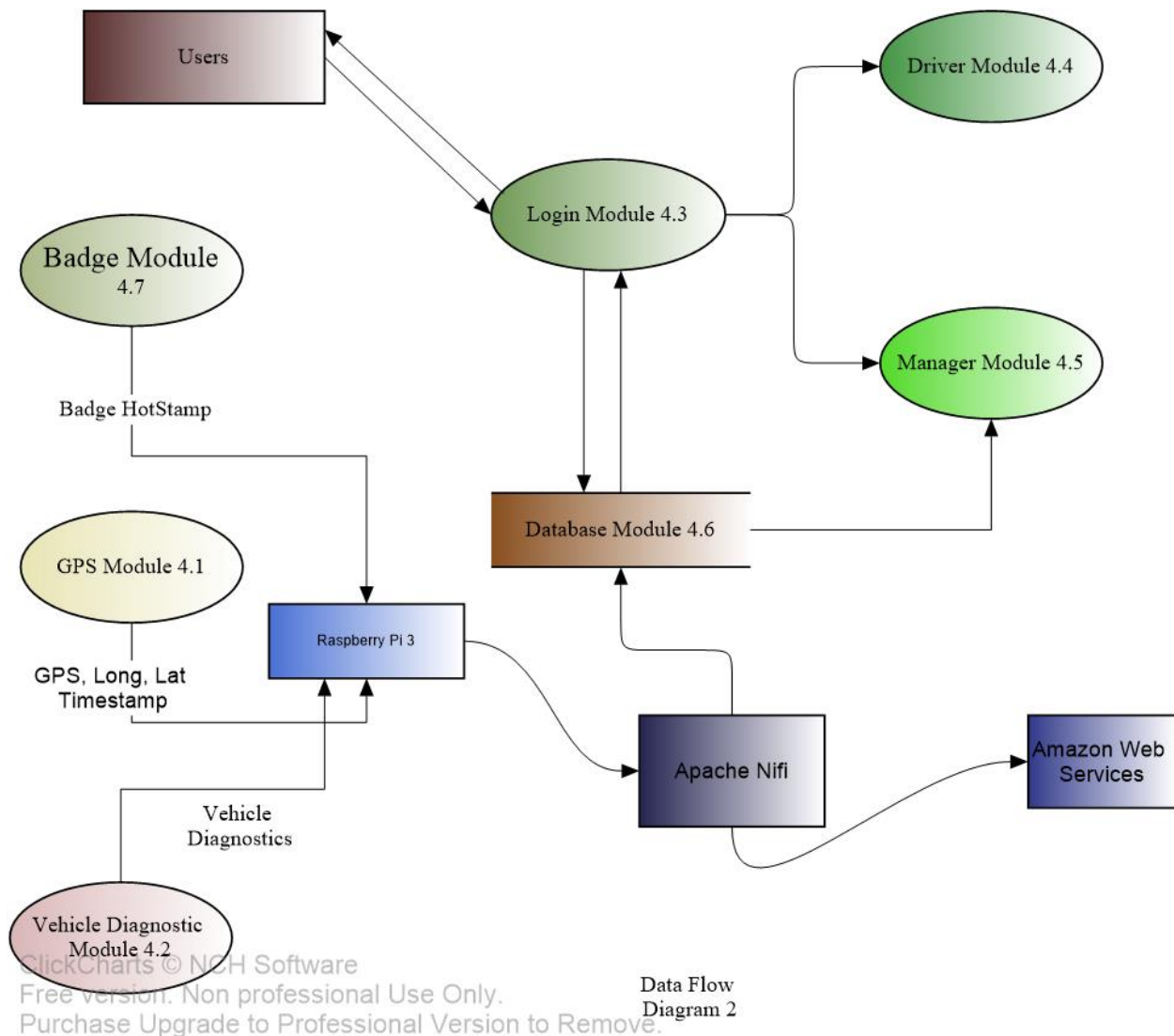
The enterprise phase will integrate data analytics on stored data.

4.2 System Overview



Context Diagram DFD 0

The Fleet Management system described above has been modularized for simplicity. More modules will be added or removed throughout the implementation phases. This is likely to evolve as the project evolves, and is no way representative of a finished product.



- 4.1 GPS Module provides GPS, formatted in NMEA 0183, for more information on NMEA. https://en.wikipedia.org/wiki/NMEA_0183 Data will be stored locally on Edge devices, which will later be transmitted in a batch process.
- 4.2 OBD-II Scanner requests vehicle diagnostics data from ECU. The ECU responds with Vehicle Subsystem data. OBD-II scanner is capable of reading all vehicle engine protocols. Data is stored in JSON format locally which can be used by Apache MiNiFi to Transmit data to infrastructure.
- 4.3 Login Module provides a user interface where driver and manager can view web content. Allows authorization for Los Angeles Park and Recreation to block non-users from viewing specific web content and allows credentialed users access

- 4.3.1 An improvised Login Module will be created using a database which will act as a stand-alone Login Feature. It will provide a mockup of a login system until more information is gathered about existing network and whether the user must login to the network prior to submitting credentials to this system. For the time being the login module will be web-based and will not take in account an active directory, or any other network related log in or permission.(A session may be looked at when doing this).Login feature should simulate a possible solution that is already in place, for now a web-based system should be considered.
- 4.4 Driver Module automates County Vehicle Mileage and Safety Check Form. Allows drivers to input this information in a digital manner. Data is stored in a persistent database which can be accessible.
- 4.5 Manager Module is a web page which only managers can view. This web page displays a list of trips with driver information. A map visualization of each trip which includes a route along with a speed or velocity for each route taken.
- 4.6 Database Module is used for persistent storage for Driver input information, along with GPS data. Data stored in this module is for the creation of a Map Visualization, and also made accessible for administrative purposes. Apache NiFi uses PutSQL processor along with textual manipulation to INSERT data into the stand alone database.
 - 4.6.1 MSSQL Tables for *GPS* and Employee Data Submission
- 4.7 The Badge Module is a COTS hardware device which is capable of reading, Los Angeles County Parks and Recreations Employee Identification badges. These Badges will be used to swipe and verify the employee driver of each vehicle that the LACFMS system is installed in. This Module is responsible for verifying the employee of each vehicle. The employee identification will be read from the card and routed, within a json file and stored in a SQL database.
 - 4.7.1 The Badge Module is a hardware device which will get the Employee Identification and route that to an external MSSQL Table.
- 4.8 Apache NiFi and MiNiFi is a technology which will govern the flow of data from the Edge to County Infrastructure.

Other Modules will be added as need be. Currently, this represents the general structure.

5. Policies and Tactics

5.1 Choice of which specific products used Software

5.1.1 Visual Studio 2017, .NET Core Technologies Compiler

Visual Studio was chosen because it makes .NET Core development smoother for the team.

5.1.2 Angular Framework version 5.0

The main functionality of the web application would remain the same, but implementation would be different if we had chosen to go with React or Vue. Angular was not picked for any specific design reason.

5.1.3 JRE 1.8 version

JRE 1.8 was the most recent and tested JRE when the project started.

5.1.4 Java version 8

Java is required for Apache NiFi and Java 8 is what we had on the machines. We did not know if Apache NiFi was compatible with Java 9.

5.1.5 GPSD library for Raspbian OS

GPSD is one of the more widely used *GPS* libraries for the Raspberry Pi.

5.1.6 OBD library for python for Raspbian OS

Many python OBD libraries exist and we chose one that had better documentation.

5.1.7 Apache NiFi version 1.4.0

Apache NiFi version 1.4.0 was the most recent edition

5.1.8 Apache MiNiFi version 0.2

Apache MiNiFi version 0.2 was the most recent edition

5.1.9 GPS NEO 6M Ublox

Many GPS modules exist for purchase. The NEO 6M was purchased for its price point and for its reviews within the community.

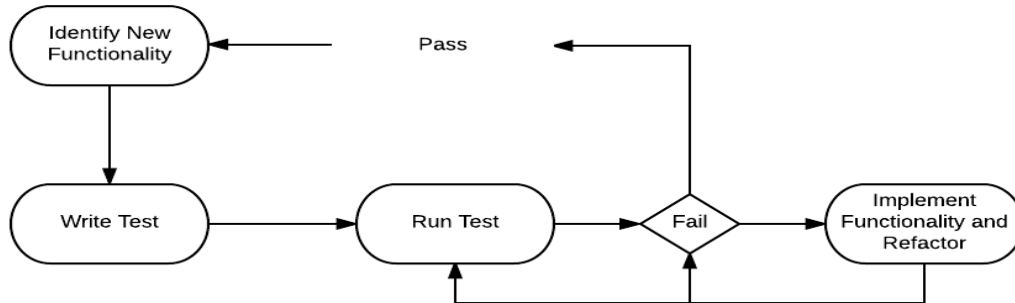
5.1.10 OBD-II Scanner

Many different OBD Scanners exist for purchase. A cheap model was purchased to lessen costs.

5.2 Plans for ensuring requirements traceability

- Overview of requirements are in sections 9 discusses each module which is responsible for verifying these requirements.

5.3 Plans for testing the software



Scenario Testing for RestFul Web Service

5.3.1 RESTful Services tests will be conducted using JavaScript versions “es5 and es6”, which allows for the testing of various functions of the RESTful API,

Test for Get Request functionality GetTests.js

```

const http = require('http');
let get_request = (url) => {
  http.get(url, (res) => {
    console.log('statusCode:', res.statusCode)
    console.log('headers:', res.headers)
    let data = ""
    res.on('data', (chunk) => {
      data += chunk
    });
    res.on('end', () => {
      console.log(data)
    })
  }).on('error', (e) => {
    console.error(e)
  })
}

```

```

}
// Employee Controller
// get_request('http://localhost:56889/api/Employee')
// get_request('http://localhost:56889/api/Employee/1')
// get_request('http://localhost:56889/api/Employee/history/1')

// Vehicle Controller
// get_request('http://localhost:56889/api/Vehicle')
// get_request('http://localhost:56889/api/Vehicle/1')
// get_request('http://localhost:56889/api/Vehicle/history/1')

// Trip Controller
// get_request('http://localhost:56889/api/Trip')
// get_request('http://localhost:56889/api/Trip/1')
// get_request('http://localhost:56889/api/Trip/path/1')

```

Test for Get Request functionality PostTests.js

```

var request = require('request')
let post_request = (url, obj, v_id, e_id) => {
  let isTrip = false
  request({
    url: url,
    method: 'POST',
    json: obj
  }, (error, response, body) => {
    {
      console.log('error:', error);
      console.log('statusCode:', response && response.statusCode)
    }
  })
}

post_request('http://localhost:56889/api/Trip',
{
  "tripId": 1,
  "projectTitle": "Dance",
  "tripPath": "Some Path",
  "purpose": "Some Purpose",
  "projectFunction": "Some Function",
  "taskOrder": "Some Order",
  "initialOdometer": 1546,
  "finalOdometer": 1570,

```

```

"vehicle": {
  "vehicleId": 1,
  "make": "Audi",
  "model": "A8",
  "plateNumber": "ARM14",
  "color": "Blue",
  "modelYear": "0001-01-01T00:00:00.0002017",
  "modelTrim": "4 Wheels",
  "odometer": 1546,
  "lastService": "2017-12-03T00:24:12.4788414",
  "nextService": "2018-02-03T00:24:12.4792875"
},
"employee": {
  "employeeId": 3,
  "firstName": "Some First Name",
  "lastName": "Some Last Name",
  "middleName": "Some Middle Name",
  "dateOfBirth": "1996-01-01T20:25:16.583533",
  "phoneNumber": "000 000 0000",
  "licenseNumber": "Some Licence Number",
  "password": "Some Password",
  "manager": false
},
"startTime": "2018-01-01T12:00:00",
"finishTime": "2018-01-01T15:00:00",
"approval": false,
"legal": false,
"reviewed": false
}, 2, 3)

```

Test for Put Request PutTests.js

```

var request = require('request')
let put_request = (url, obj, v_id, e_id) => {
  let isTrip = false
  if ("tripPath" in obj) isTrip = true
  request(
    {
      url: (isTrip) ? url : url + '/'
        + v_id + '/' + e_id,
      method: 'PUT',
      json: obj
    }, (error, response, body) =>
    {

```

```

        console.log('error:', error);
        console.log('statusCode:', response && response.statusCode)
    })
}

```

```

// Vehicle Controller
// put_request('http://localhost:56889/api/Vehicle',
// {
//   "make": "Audi",
//   "model": "A8",
//   "plateNumber": "ARM14",
//   "color": "Blue",
//   "modelYear": "0001-01-01T00:00:00.0002017",
//   "modelTrim": "4 Wheels",
//   "odometer": 1546,
//   "lastService": "2017-12-03T00:24:12.4788414",
//   "nextService": "2018-02-03T00:24:12.4792875"
// } )

```

```

// Employee Controller
// put_request('http://localhost:56889/api/Employee',
// {
//   "firstName": "Some First Name",
//   "lastName": "Some Last Name",
//   "middleName": "Some Middle Name",
//   "dateOfBirth": "1996-01-01T20:25:16.583533",
//   "phoneNumber": "000 000 0000",
//   "licenseNumber": "Some Licence Number",
//   "password": "Some Password"
// } )

```

```

// Trip Controller
put_request('http://localhost:56889/api/Trip',
{
  "projectTitle": "Clean Up",
  "tripPath": "[{1,2}, {3,4}, {5,6}, {7,8}, {9,10}]",
  "purpose": "Some Purpose",
  "projectFunction": "Some Function",
  "taskOrder": "Some Take Order",
  "initialOdometer": 1546,
  "finalOdometer": 1570,
  "vehicle": null,

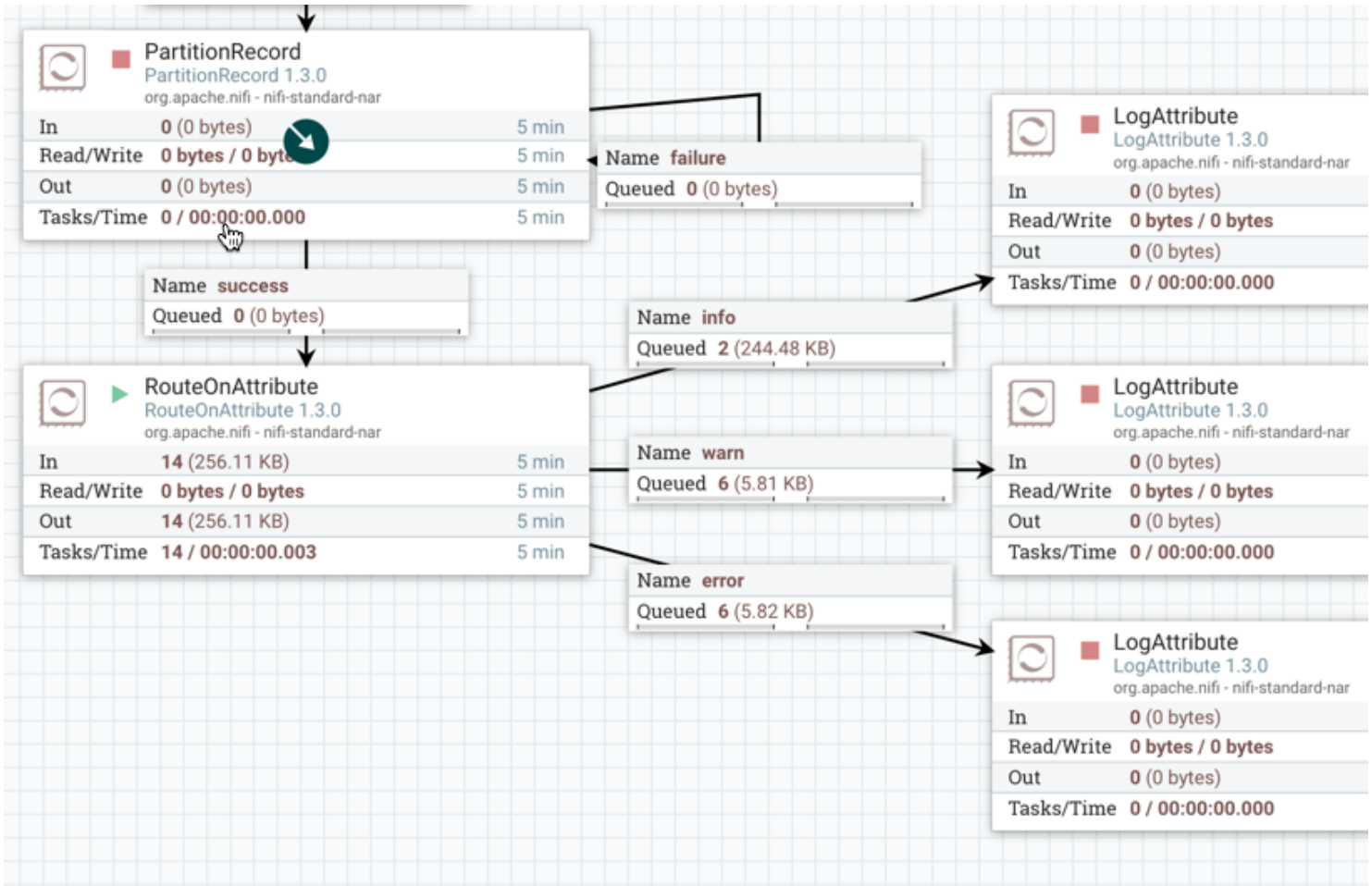
```

```

"employee": null,
"startTime": "2018-01-01T12:00:00",
"finishTime": "2018-01-01T15:00:00"
}, 2, 3)

```

Apache NiFi, is being used to Log failed data between Running Processes. This assists in logging the data where it fails and re-routes the data back to be resubmitted.



5.4. Hardware Performance Testing

5.4.1 GPS Processing Test

- Performance Testing on Edge Device Load
 - 5.4.1 GPS processing at 1Hz or 1 cycle per second.

5.4.2 GPS processing at 5Hz or 5 cycles per second.

- Trade-Offs

Advantage to accuracy-There is no beneficial gain when dealing with car vehicle velocity. Positional data is accurate and sufficient at 1 cycle per second, or 1Hz.

Testing was conducted with Ublox software which allows configuration changes at the hardware level. See, <https://www.u-blox.com/en/product/u-center-windows> as a reference.

- Advantage to performance
None- 1Hz GPS sensory collection is sufficient.

Trade-offs exist there is more benefits to reducing storage capacity by collecting GPS at 1 second versus 5 collections per second. Theoretically, reducing storage capacity 5x over the same time frame.

Overview of hardware limitations

- Velocity accuracy 0.1m/s
- Heading accuracy 0.5 degrees
- Operational Limits Dynamics 4g
- Altitude 50,000 m (meters)
- Velocity 500 m/s (meters per second)

5.4.2 OBD-II Scanner Test

- See Manufacturer Datasheet for exact specifications, usage, and other benchmarks.

5.4.3 Los Angeles County Parks and Recreation RFID card reader

- See Manufacturer Datasheet for exact specification, usage and other benchmarks.

5.5. Engineering trade-offs

Many trade-offs exist when deciding hardware device integration, easy of to use, scalability and at what cost(s) to the overall functionality of the system design.

5.6. Maintaining the software

Nominal maintenance is expected with the overall system. Currently, this is a plug and play automated solution. However, the system does rely on some third party API's which have their own dependencies and costs associated with them. One example is the Raspberry Pi(3) operating system which is continually being updated, and it maybe a good idea to keep a

preserved copy of the operating system. Secondly, Google's Road Maps API charges when requests exceed a thousand transactions. In these cases it may be a good idea to move to a less dependent and in house solution for mapping visualizations. A cost analysis should be conducted to determine what is the best choice for the consumer. Despite these dependencies the overall system should operate with minimal update, and should be manageable through at least the life cycles associated with the hardware product life cycle.

6. Detailed System Design

6.1 GPS (Module)	
6.1.1 GPS Component	<p>GPS Module, consists of GPS COTS Hardware device which when powered and configured continually gives GPS coordinates of the device's current location. Data is given in NMEA 0183 sentence specification which is timestamped for each pair of longitudes and latitudes.</p>
6.1.2 Constraints	<ul style="list-style-type: none"> ● GPS timing can be an issue as it relies on getting a satellite fix which can often have a different time then the device location time. ● Storage capacity could also be a potential issue as the GPS hardware gives GPS coordinates every second. Currently, python scripts are being used to interact with the device's data output to eliminate redundant data storage while not compromising overall positional accuracy.
6.1.3 Composition	<p>Hardware Device, the exact composition and features is defined by the manufacturer. Python Scripts are being used to get GPS, longitude, latitude and timestamp, that is part of the NMEA sentencng. This data is then routed using Apache NiFi and placed into the SQL Database.</p>
6.1.4 Uses/Interactions	<p>The GPS Module data output is being interacted on by a python script which is run on the raspberry pi3. This data is being stored on the Raspberry Pi 3 locally and is broadcasted using a secure messaging protocol. Apache MiNiFi and Apache NiFi are being used to transmit the data to infrastructure using a site-to-site protocol. The GPS Module's output data is sanitized and stored in Microsoft SQL database where it is accessible to database administrators but will be visually displayed to authorized personnel e.g. managers, on a manager's page which will have elevated permissions for viewing.</p>
6.1.5 Resources	<p>https://www.u-blox.com/en/product/neo-m8-series</p>
6.1.6 Interface/Exports	<p>Hardware, Automated file flow, python scripts</p>

6.1.7 NMEA 0183 (Composition)

NMEA 0183 defines a specification for marine communications which include GPS receivers. The COTS GPS hardware device generates NMEA 0183 as output. If configured using the GPSD python library, then the output is configure to display on the local host at 2947.

```
1. import GPS
2.
3. # Listen on port 2947 (GPSd) of localhost
4. session = GPS.GPS("localhost", "2947")
5. session.stream(GPS.WATCH_ENABLE | GPS.WATCH_NEWSTYLE)
6.
7. while True:
8.     try:
9.         report = session.next()
10.            # Wait for a 'TPV' report and display the current time
11.            # To see all report data, uncomment the line below
12.            # print report
13.            if report['class'] == 'TPV':
14.                if hasattr(report, 'time'):
15.                    print report.time
16.            except KeyError:
17.                pass
18.            except KeyboardInterrupt:
19.                quit()
20.            except StopIteration:
21.                session = None
22.                print "GPSD has terminate"
```

Example NMEA 0813 Sentence at localhost//2947

UBLOX Hardware

```
$GPRMC,162254.00,A,3723.02837,N,12159.39853,W,0.820,188.36,110706,,A*74
$GPVTG,188.36,T,,M,0.820,N,1.519,K,A*3F
$GPGGA,162254.00,3723.02837,N,12159.39853,W,1,03,2.36,525.6,M,-25.6,M,,*65
$GPGSA,A,2,25,01,22,,,,,,,,,2.56,2.36,1.00*02
$GPGSV,4,1,14,25,15,175,30,14,80,041,,19,38,259,14,01,52,223,18*76
$GPGSV,4,2,14,18,16,079,,11,19,312,,14,80,041,,21,04,135,25*7D
$GPGSV,4,3,14,15,27,134,18,03,25,222,,22,51,057,16,09,07,036,*79
$GPGSV,4,4,14,07,01,181,,15,25,135,*76
$GPGLL,3723.02837,N,12159.39853,W,162254.00,A,A*7C
$GPZDA,162254.00,11,07,2006,00,00*63
```

The GPS code which is written in python, shall be kept on a private repository for future usage.

6.2 Vehicle Diagnostics(Module)	
6.2.1 Vehicle Diagnostics Component	Vehicle Diagnostics Module consists of a COTS ELM327 bluetooth enabled OBD-II scanner. The data will be collected and stored using python scripts. The python scripts can be programmed using the python scheduler library to automate continued execution as long as the Raspberry Pi 3 is powered and the ELM327 device is functional. Apache MiNiFi, will assist in transferring stored datas into a configured instance of Apache NiFi. NiFi will be used to manipulate, transform and route data to its desired locations.
6.2.2 Constraints	<ul style="list-style-type: none"> ● Power needed for the OBD Diagnostics Reader to vehicle ECU. ● If device fails, diagnostic's will not be captured. ● When vehicle is powered off, Raspberry Pi 3 will not capture new information.
6.2.3 Composition	Hardware Interface, using bluetooth connection to request data from vehicle Engine Control Unit. Automation of this process is being done using BASH scripting language and storage of data is write to file.
6.2.4 Uses/Interactions	Processing device's data is automated using Apache NiFi
6.2.5 Resources	Hardware Datasheets
6.2.6 Interface/Exports	Hardware, Automated file flow

6.3 Login (Module)	
6.3.1 Login Component	Login Component will be a stand alone system, which provides varying access to web-application content. Managers should only be able to view the visual routes taken by vehicles.
6.3.2 Constraints	<ul style="list-style-type: none"> • Managers should only be able to view the map visualization. • County employee should only be able to login and submit data pertaining to their particular work requests.
6.3.3 Composition	The Login is part of the web application written Angular which is composed of a mock, web interface which prompts the employee to submit, proper credentials.
6.3.4 Uses/Interactions	<p>Users logs in at specified url, to gain different features of the web-application.</p> <ul style="list-style-type: none"> • Managers have access to visual routes taken. • Non-manager employee shall be blocked from seeing visual display of routes taken by various employees. • Driver employee(s) need to login to submit data pertaining to the work-requests of various assignments.
6.3.5 Resources	https://angular.io/api
6.3.6 Interface/Exports	Interfaces, web application part of the front end page and the database module.

6.4 Driver (Module)	
6.4.1 Driver Component	Driver component is the web application where Employee's of Los Angeles County Parks and Recreation may select from a list of trips and provide details, to be submitted and stored into the Microsoft SQL database.
6.4.2 Constraints	<ul style="list-style-type: none"> ● Google Maps Api, can be somewhat slow in terms of delivering address data from requests made to the API. This has caused a slowed response which we are currently working on to resolve.
6.4.3 Composition	<ul style="list-style-type: none"> ● Software interface of a web page. Written in Angular 5.0 interfaces with Database Module to retrieve routing data from TRIP TABLE in the Database Module.
6.4.4 Uses/Interactions	Database Module 4.6, 6.6; Used to retrieve Trip data from the Trip Table in Database. Creates a list that the driver chooses to specify which trips were involved for a given work request. Note: Multiple trips may encompass, a single work request.
6.4.5 Resources	https://angular.io/api , geocoding
6.4.6 Interface/Exports	Interface with database module, google maps api's, and the user. Connection with persistence database, to prompt user(Drivers) with details, on how to submit information pertaining to vehicle routes which are currently be done manually on paper log.

6.5 Manager (Module)	
6.5.1 Manager Component	Manager component is part of the web application where managers can select a trip and view the route taken by the driver. There will be a visual display of the route taken which will be displayed the vehicle's velocity. The mapping feature needs to have permissions such that only managers can view the mapping function.
6.5.2 Constraints	<ul style="list-style-type: none"> • Managers should only be able to view the map visualization. • County employee should only be able to login and submit, data pertaining to their particular work requests.
6.5.3 Composition	Software Interface, desktop application which is written in Angular 5.
6.5.4 Uses/Interactions	Managers, login and are directed to website where they can choose a driver, or trip from a list. Data is fetched from a database and the Manager is given specific the specific driving route taken. This is displayed on a Map, which visually show the path.
6.5.5 Resources	https://angular.io/api
6.5.6 Interface/Exports	

6.6 Database (Module)	
6.6.1 Database Component	The database module will store all relevant data for the <i>LACFMS</i> . This data will be stored in Microsoft SQL database(section 8.1). The database will be made up of tables for Employees, Managers, Trips, Vehicles, and Badge Information,.
6.6.2 Constraints	<ul style="list-style-type: none"> • Please see maximum capacity specification for MS SQL.
6.6.3 Composition	Stand Alone MS SQL Database.
6.6.4 Uses/Interactions	Login Module, Trip Data(GPS storage), Manager Module to display a visual of Trip Data with routing.
6.6.5 Resources	https://docs.microsoft.com/en-us/sql/
6.6.6 Interface/Exports	Microsoft SQL Application Software

6.7 Name (Module)	Badge Module
6.7.1 Component	The Badge Module is a COTS hardware device which is capable of reading, Los Angeles County Parks and Receptions Employee Identification badges. These Badges will be used to swipe and verify the employee driver of each vehicle that the FMS system is installed in. This Module is responsible for verifying the employee of each vehicle. The employee identification will be read from the card and routed, within a json file and stored in a SQL database.
6.7.2 Constraints	<ul style="list-style-type: none"> • RFID reader needs power to properly run and collect Employee Identification
6.7.3 Composition	RFID Hardware Device
6.7.4 Uses/Interactions	User swipes, taps Los Angeles County Badge on Badge Reader
6.7.5 Resources	Card Reader Make and Model WAVE ID pcProx RFIDeas RDR-6081AKU-78X FCC ID: M9MPCROXHUSB100 IC: 6571A-RDR6061 Serial No: H097653 https://www.rfideas.com/files/rfideas/files/support/doc/manuals/POE_Manual.pdf
6.7.6 Interface/Exports	Hardware Device RFID reader. Connects to the Raspberry Pi

6.8 Name (Module)	File Flow Manager (Apache NiFi)
6.8.1 Component	Apache NiFi
6.8.2 Constraints	<ul style="list-style-type: none"> • Data transfer rates can be upwards in the 10Gbs of data transfer at the edge provided the network being used is capable. • New technology that is still having more capabilities added
6.8.3 Composition	Apache NiFi is composed of commonly used processes used in software development. At its core the components are written in the Java programming languages.
6.8.4 Uses/Interactions	Apache NiFi, is being used on Edge devices and Infrastructure to consume and ingest data being transported site to site.
6.8.5 Resources	https://nifi.apache.org/developer-guide.html https://cwiki.apache.org/confluence/display/NIFI/NiFi+Components
6.8.6 Interface/Exports	The interface of Apache NiFi, provides a drag and drop web page on the localhost, which allows user groups to configure a multitude of commonly used techniques in software development, in attempt to expedite: ETL or complete automate these processes.

7. Detailed Lower Level Component Design

7.1 TripGather.py

7.1.1 Function

7.1.2 Runs whenever the vehicle is in use and shutdowns when the vehicles location does not move after a designated amount of time.

7.1.3 No interface

7.2 DiagnosticGather.py

7.2.1 Function

7.2.2 Runs whenever the vehicle is in use and is disabled when the scheduling script determines the car is not in use

7.2.3 No interface

7.3 RFID.py

7.3.1 Function

7.3.2 Runs whenever the Raspberry Pi is on. Waits for an employee to swipe their ID badge

7.3.3 External interface that connects to the Raspberry Pi. The interface is proximity based

7.4 Odometer/VehicleID/EmployeeID/TripCounter/DiagnosticCounter/.txt

7.4.1 Files

7.4.2 Used for data persistence on the Raspberry Pi.

7.4.3 No Interface

7.5 SchedulingScript.sh

7.5.1 Function

7.5.2 Automates all functionality on the Raspberry Pi. Uses the files listed above to know when to shut off certain functionalities.

7.5.3 No Interface

7.6 Persistence Layer

7.6.1 Package

7.6.2 Contains all the classes relevant to the database tables. CRUD methods are created for each table and stored within the specified class file.

7.6.3 No interface

7.7 Web API

7.7.1 Package

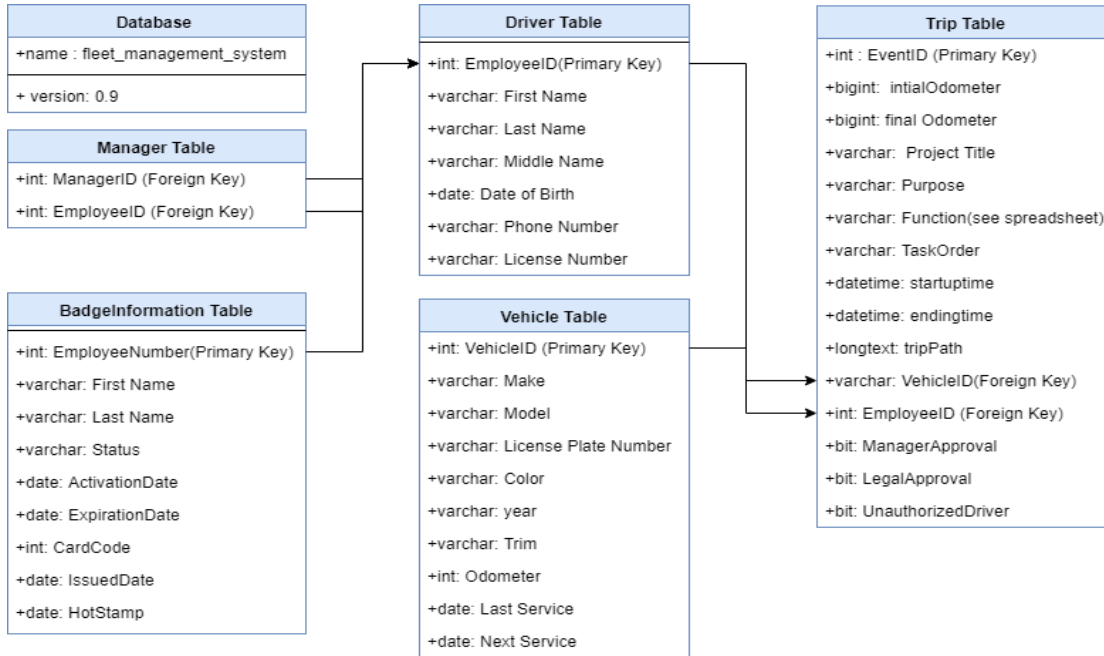
7.7.2 Contains all the classes relevant to the RESTful commands. Each class has appropriate GET/POST/DELETE/PUT commands for the CRUD methods.

7.7.3 No interface

8. Database Designs

8.1 Database Design

8.1.1 For routing data from the Raspberry PI3 into this database model see sections 4.6.2.



8.1 Manager Table

Cross reference table that allows Employees to manage one another

8.2 BadgeInformation Table

Every employee has an identification badge and this table is related to how Los Angeles County keeps track of the information on each badge.

8.3 Employee Table

Basic employee information table that Los Angeles County collects

8.4 Vehicle Table

Basic vehicle information to identify various vehicles

8.5 Trip Table

The relations of the Trip Table come from the Los Angeles County Safety Check and Mileage Form. Most of the data points are automated, but employees must fill in four fields that cannot be automated.

9. User Interface

- 9.1. Overview of User Interface
- 9.2. Screen Frameworks or Images
- 9.3. User Interface Flow Model

Web-Application user-interface from perspective of user(s).

Driver Module 4.4

Developed in HTML5 and Angular 5.

Initial Web Page Display

- Navigation Bar
Logout feature
Displays Name of the Logged in user.

Driver Page



Driver Page, When the Logo is “Depressed,” user is redirected to a Trips Page.

- Login Functions are defined in the Manager Module. The Login functions operates using tokens an is essential for accessing map visualization of routes taken by drivers.

- Logout Function simply logs out the current user and blocks users from viewing the map visualizations.

Trips Page

User(s) Choose which trips correspond to their work requests. User(s) fill in information pertaining to, Task Order, Function, XXX,XXX as per County Vehicle Mileage and Safety Check Form. Data is stored in Trip Table described in section 8.1

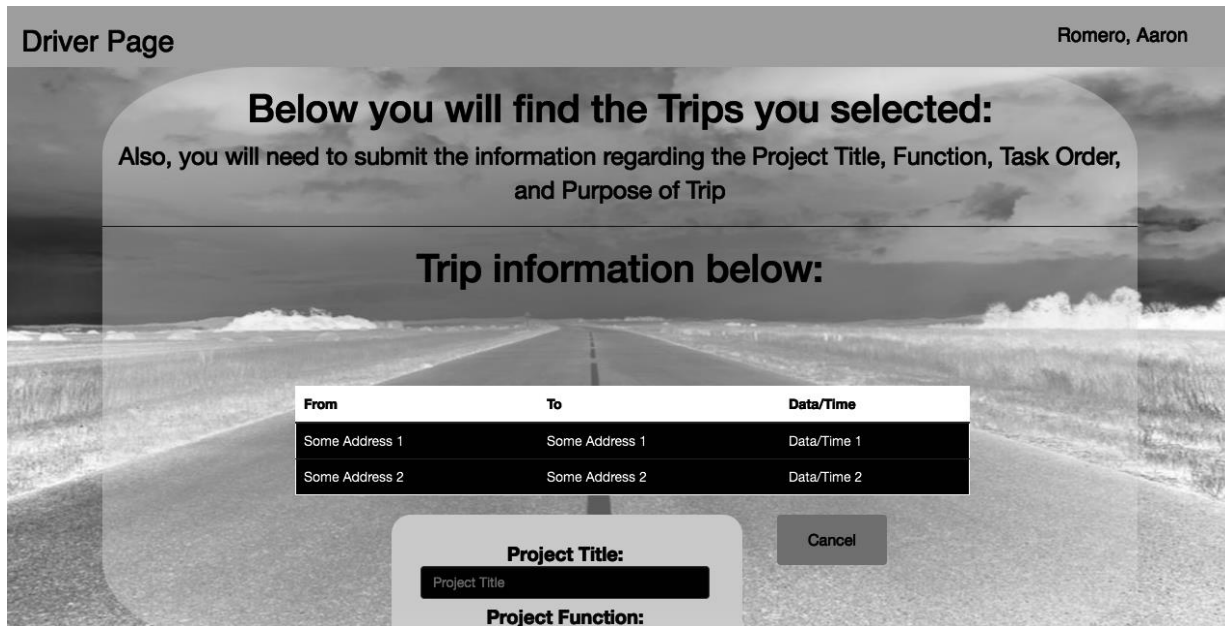
The screenshot shows the 'Driver Page' for user 'Romero, Aaron'. It features a table with trip data and a modal form for entering trip details.

From	To	Data/Time
Some Address 1	Some Address 1	Data/Time 1
Some Address 2	Some Address 2	Data/Time 2

The modal form contains the following fields and buttons:

- Project Title:** Project Title
- Project Function:** Project Function
- Task Order:** Task Order
- Purpose Of Trip:** Purpose Of Trip
- Submit** button
- Cancel** button

This portion of the driver page is essential in gathering information from the drivers pertaining to a particular work request. Multiple trips may be chosen and may correlate to a single trip.



These Four fields correspond to the county paper form which will be used to gather information for Los Angeles County purposes in attempt to streamline this process. The address are being generated using reverse geocoding.

- Project Title
- Project Function
- Task Order
- Purpose of Trips

Manager Module 4.5.

Developed in HTML5 and Angular version 5.

Initial Web Page Display with Login Function:

- Navigation Bar layout. Upper position of web page, includes the following buttons.
 - Login/out , Registration
 - Home Button:
 - Driver List Button:
 - Trips List Button:
 - Vehicle List Button

Login Page

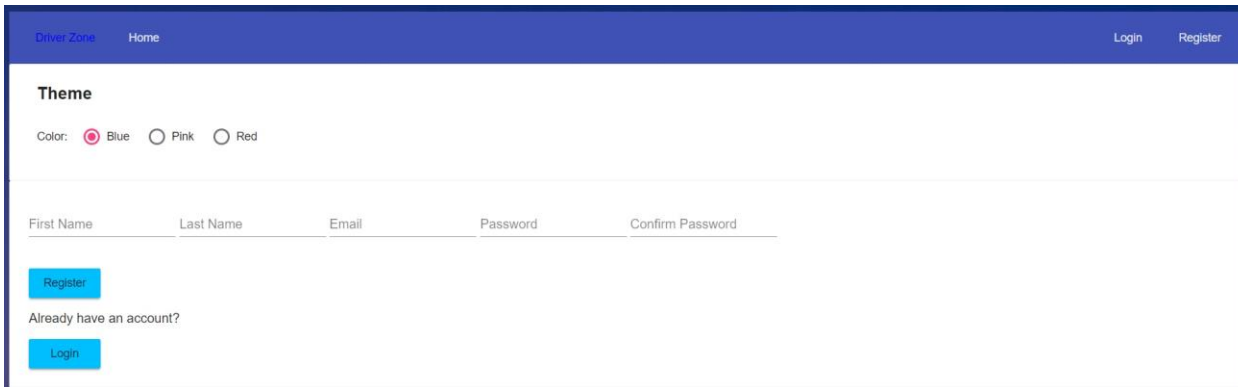
Prompts the User To Login, If the user is does not have credentials, they may create one by registering.



A simple login form with two input fields labeled 'Email' and 'Password', and a blue 'Login' button to the right.

Registration Page

Prompts users to create username, password, supply email. All of which is submitted to database for storage and are cross referenced for validation purposes.



A registration form with a blue header containing 'Driver Zone', 'Home', 'Login', and 'Register'. Below the header is a 'Theme' section with radio buttons for 'Blue' (selected), 'Pink', and 'Red'. The main form has five input fields: 'First Name', 'Last Name', 'Email', 'Password', and 'Confirm Password'. Below these fields is a blue 'Register' button, followed by the text 'Already have an account?' and a blue 'Login' button.

- Driver List Button: Lists all drivers from the SQL Database



A page titled 'Driver List' with a blue header containing 'Driver Zone', 'Home', 'Drivers List', 'Trips List', 'Vehicles List', 'Welcome Evik', and 'Logout'. Below the header is a 'Theme' section with radio buttons for 'Blue' (selected), 'Pink', and 'Red'. The main content is a list of driver names with dropdown arrows to the right: Patrick Flinner, Joe Smith, Duke Nguyen, Brad Smith, and Pat Fli.

If a driver is chosen, it list information on the driver.

Fields Include

1. Employee ID
2. First Name
3. Last Name
4. Middle Name
5. Date of Birth
6. Phone Number

7. Driver License Number

The screenshot shows a web application interface with a blue header. The header contains navigation links: "Driver Zone", "Home", "Drivers List", "Trips List", and "Vehicles List". On the right side of the header, it says "Welcome Evik" and "Logout". Below the header, there is a section titled "Theme" with three radio buttons: "Blue" (selected), "Pink", and "Red". Below the theme section, there is a profile card for "Patrick Flinner" with a small circular profile picture. The profile card lists the following information: "Employee ID: 1", "First Name: Patrick", "Middle Name: T", "Last Name: Flinner", "Date of Birth: 1992-02-22T00:00:00", "Phone Number: 6265322803", and "Driver License Number: F12345".

- **Trips Button**

Lists all the trips that have been recorded by the GPS and Which have NOT been verified by the Manager.

Initial State of Web Page

The screenshot shows the same web application interface as above, but the "Trips List" navigation link is selected. Below the theme section, there is a table with five rows, each representing a trip. Each row contains the text "Trip ID: 1" through "Trip ID: 5" and a small downward-pointing arrow icon on the right side of each row.

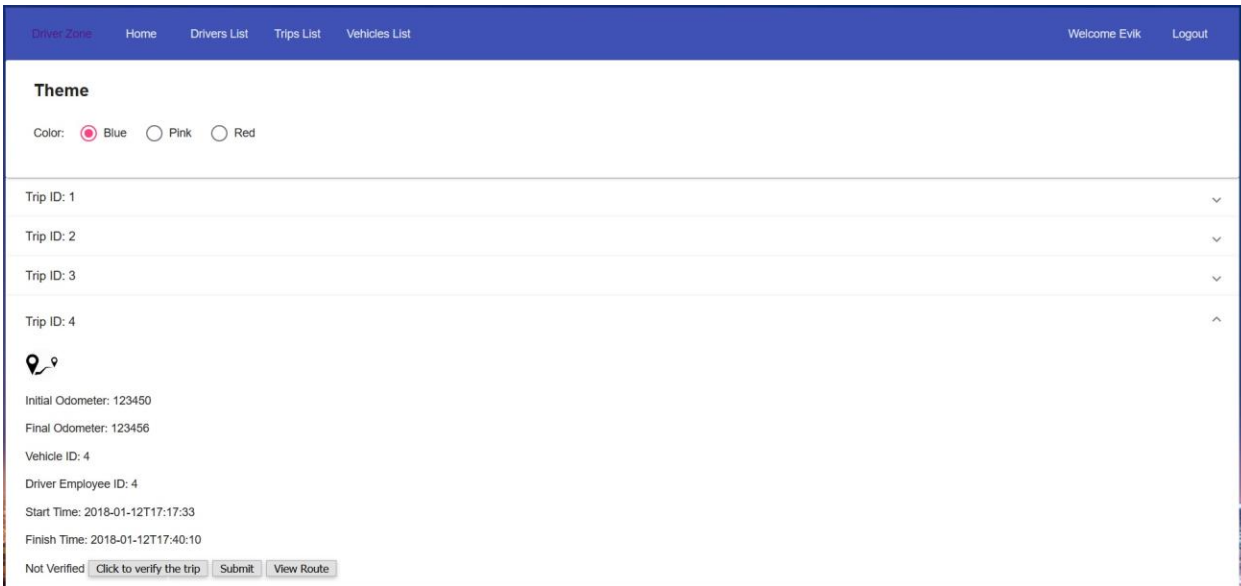
If a Trip has not been verified, then the Trip appears to the Manager, this view includes several fields.

1. Trip ID
2. Initial Odometer Reading of vehicle
3. Final Odometer Reading from vehicle
4. Driver Employee ID
5. Start time
6. Finish Time

A Not Verified field which has two choices, and one view route button

1. Click to verify the trip (click to verify)
2. Submit to the SQL database.

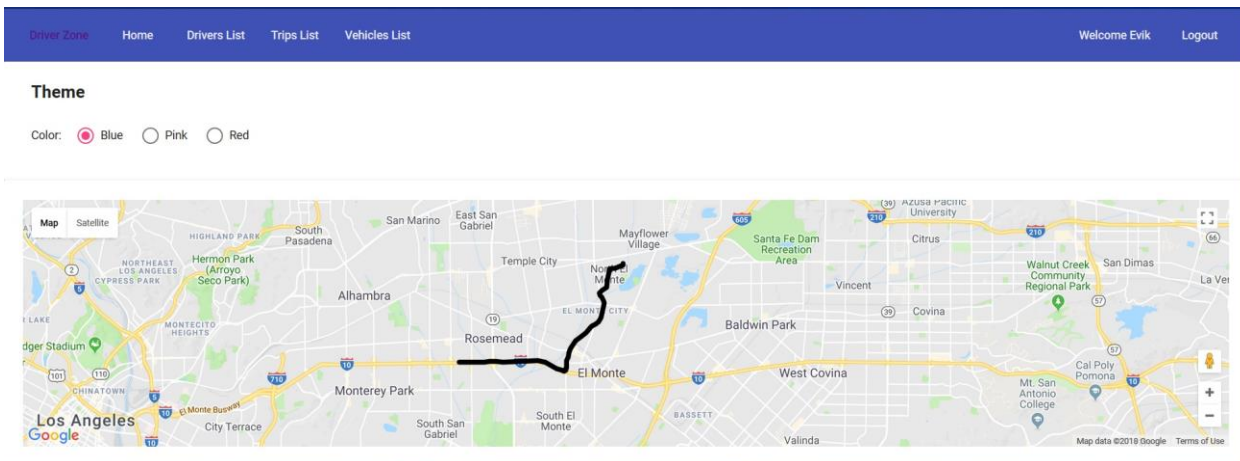
A Verified Trip will not appear in this list.



- **View Route Button**

Displays the route taken for a particular Trip. This is integrated into GOOGLE ROADS API, which is display in angular container.

- **Satellite View:**
- **Map View:**
- **Zooming Function:** Allows the user to zoom in on a particular GPS location.



- **Vehicle List Buttons**

Lists all vehicles that have been added to the Database.

Vehicle ID: when depressed does a drop down menu with

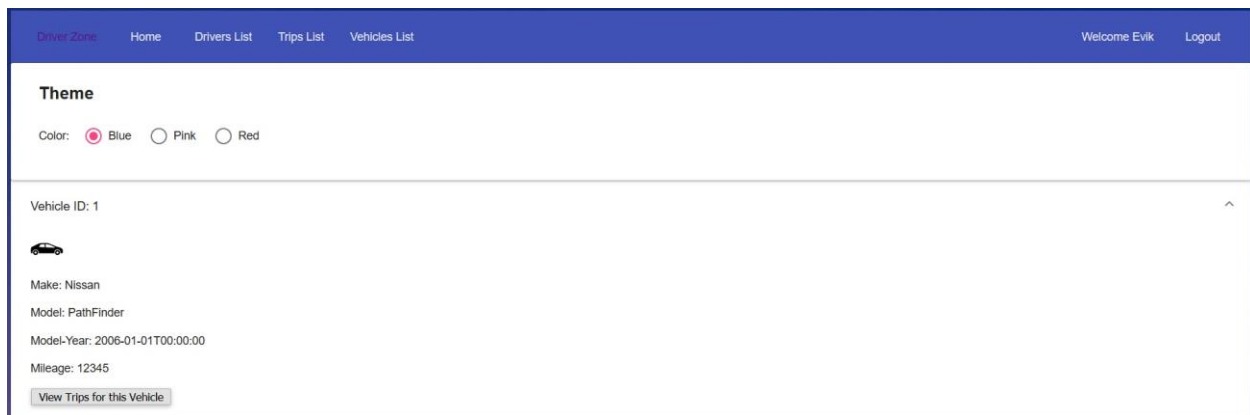
Vehicle ID:

Make:

Model/Year:

Current Mileage:

If the user chooses “view trips for this vehicle,” web application displays Trip that the vehicle has taken most recently.



Displays Vehicle Information when button is depressed

10. Requirements

Requirement	Component	Test
	Module	
<p>10.4.1 Positional data collected shall include the following, longitude, latitude and timestamp. Positional data in this context refers to global satellite positioning which will be used in conjunction with trip data. The timestamp shall have a date and the time from a 24 hour clock.</p>	<p>GPS Module see section(s) 4.1, 6.1</p>	<p>Hardware Controlled, by Ublox Software. See Section 5.4.1</p>
<p>10.4.2 Vehicle Diagnostics data will be made up of speed, current odometer, and all pertinent data which is collected by the data ingestion device. This data should be all the sensors that exist on the car. Discretion will be used to determine what sensor data does not need to be stored. This data will be used to determine if the vehicle has any operating abnormalities. Vehicle diagnostics data will be stored within a data lake for later data analysis. The analysed data will be available for maintenance and mechanics.</p>	<p>Vehicle Diagnostics Module see section(s) 4.2, 6.2</p>	<p>COTS, ELM327 with bluetooth capabilities.</p>
<p>10.4.3 <i>LACFMS</i> shall display the path traveled on a map</p>	<p>Manager Module 4.5</p>	<p>Test:</p>

which will be accessible by the management user. The path or route shall display the velocity of the vehicle along each route traveled.		Dependencies:
10.4.4 The system shall interconnect microcontrollers with appropriate technology to perform high bandwidth transactions for proper data ingestion, storage and transmission of stored data.	Apache NiFi Technology	Maximal Transfer is dependent on the network, but the technology itself can handle 10Gbs/from edge devices.
10.4.4 The system shall require driver validation.	Badge Module 4.7, 6.7	COTS, Hardware
10.4.5 The driver of a Los Angeles County vehicle shall sign in to a vehicle by swiping their identification badge. The authentication system may disable the vehicle from starting.	GPS Module 4.1, 6.1	
10.4.6 All data will be sent to a Los Angeles County server infrastructure and stored into a database. Vehicle Diagnostic Data will be saved to a data lake. Trip Data will be saved to a relational database. Trip Data must be queryable by applications for supervisors and managers.	Apache NiFi Technology Manager Module 4.5, 6.5 (Displaying Trip Data) GPS Module 4.1, 6.1(Trip Data)	
10.4.7 All employees shall input specific trip information into the web application.	Driver Module 4.4, 6.4	

10.4.8 The <i>LACFMS</i> system shall have a web application which shall register new employees if they do not currently exist in the database.	Driver/Manager Modules 4.4, 6.4/ 4.5, 6.5	
10.4.9 The <i>LACFMS</i> system shall display <i>GPS</i> data with vehicle locations.	Manager Module 4.5, 6.5	
10.4.10 All data gathered from driver will be stored in a relational database such as SQL.	Database Module 4.6, 6.6	
10.4.11 Employees must be identified with each vehicle that they have driven.	Driver Badge Module 4.7,6.7	
10.4.12 Installation of <i>LACFMS</i> devices must not splice into a vehicle's wiring.	Installation Requirement	
10.4.13 The system should allow for the generation of vehicle usage history for any given time period.	Database/Manager Module 6.6, 8.2	

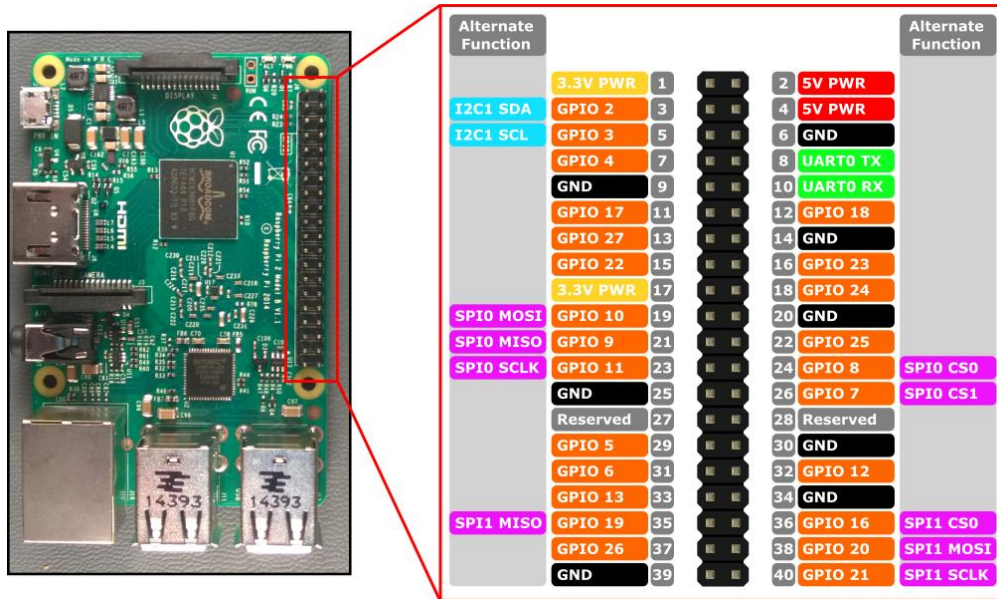
11. Glossary

CAN	A Controller Area Network (CAN bus) is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer
COTS (Commercial off the shelf)	Refers to ready-made merchandise that is available for sale, defined by market need, significant functionality and complexity, and self-contained.
ECU	The Parts of the Engine the ECU Controls. The ECU , also known as the car computer, provides controls for a variety of systems within the engine. The following sections will examine these systems, including the control of air:fuel ratio, ignition timing, and idle speed.
Global Positioning System	is a radio navigation system that allows land, sea, and airborne users to determine their exact location, velocity, and time 24 hours a day, in all weather conditions, anywhere in the world.
Graphical user interface (GUI /gu:i:/)	is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.
Internet of Things (IoT)	The internet of things is the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive.

LACFMS	Los Angeles County Fleet Management System(LACFMS): Describes the overall software in this document.
Low Frequency (LF) RFID	The LF band covers frequencies from 30 KHz to 300 KHz. Typically LF RFID systems operate at 125 KHz, although there are some that operate at 134 KHz. This frequency band provides a short read range of 10 cm, and has slower read speed than the higher frequencies, but is not very sensitive to radio wave interference.
MQTT Protocol Specifications	<i>MQTT</i> v3.1.1 is an OASIS Standard. ... <i>MQTT-SN</i> is a publish/subscribe messaging protocol for wireless sensor networks (WSN), with the aim of extending the <i>MQTT</i> protocol beyond the reach of TCP/IP infrastructure for Sensor and Actuator solutions
<i>NMEA 0813</i>	NMEA 0183 is a combined electrical and data specification for communication between marine electronic devices such as echo sounder, sonars, anemometer, gyrocompass, autopilot, GPS receivers and many other types of instruments. It has been defined by, and is controlled by, the U.S.-based National Marine Electronics Association. It replaces the earlier NMEA 0180 and NMEA 0182 standards. In marine applications it is slowly being phased out in favor of the newer NMEA 2000 standard. NMEA-0180 and 0182 are very limited, and just deal with communications from a Loran-C (or other navigation receiver, although the standards specifically mention Loran), and an

	autopilot.
<i>OBD-II</i> (On-board diagnostics Parameter IDs)	are codes used to request data from a vehicle, used as a diagnostic tool. SAE standard J/1979 defines many <i>PIDs</i> , but manufacturers also define many more <i>PIDs</i> specific to their vehicles.
RFID	A radio frequency identification reader (RFID reader) is a device used to gather information from an RFID tag , which is used to track individual objects. Radio waves are used to transfer data from the tag to a reader . RFID is a technology similar in theory to bar codes.
TBD	To be determined

12. Hardware Schematics



Neo-6M RPI

VCC to Pin 1, which is 3.3v

TX to Pin 10, which is RX (GPIO15)

RX to Pin 8, Which is TX (GPIO14)

Gnd to Pin 6, which is Gnd

Alternatives exist, depending hardware additions. Configuration can follow the GPIO layout as above.



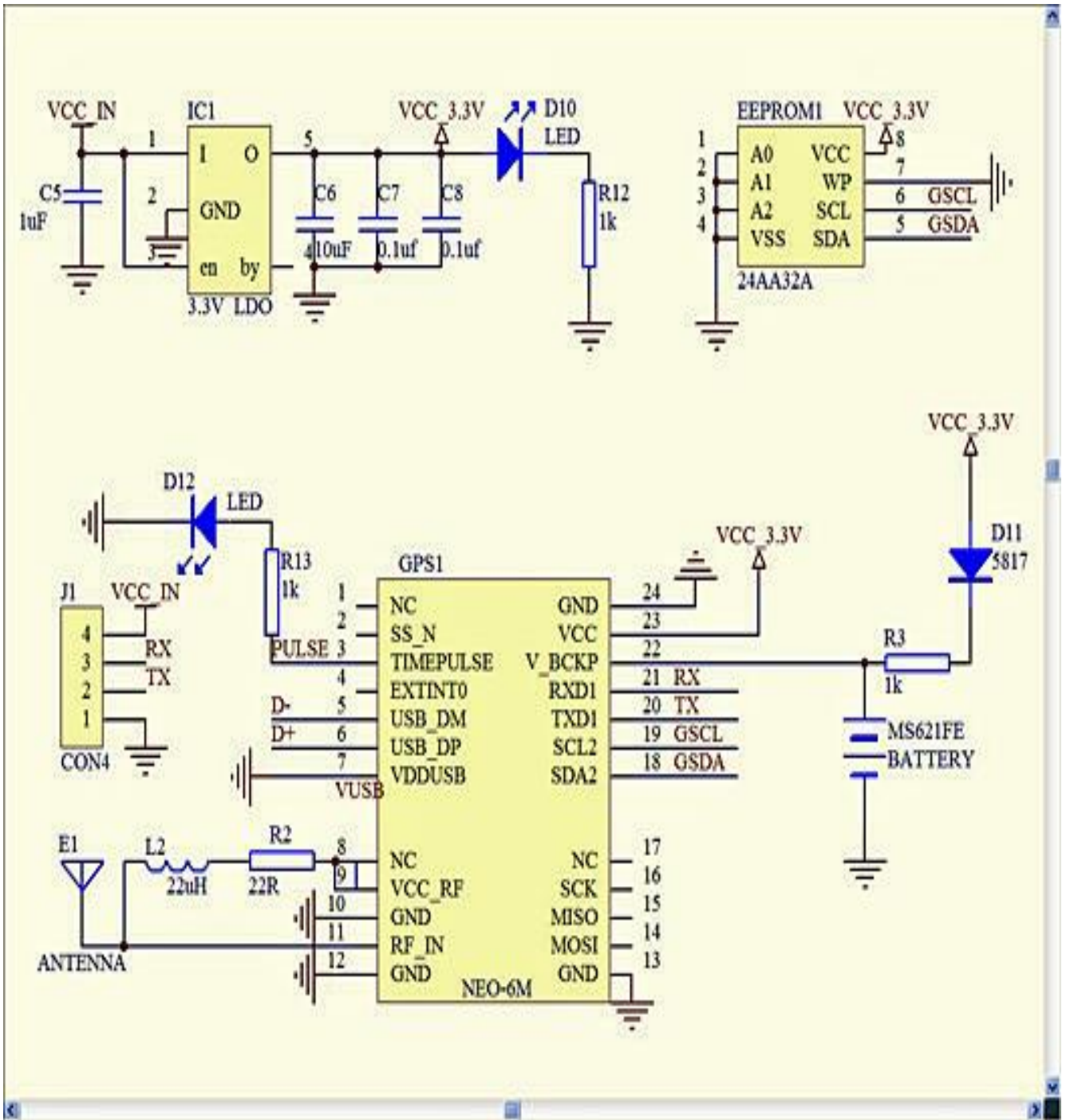
1. ELM327 connect to OBD-II interface
2. ELM327 connected to Raspberry Pi using Bluetooth.

Communication
Bluetooth to Raspberry Pi



WAVE ID
pcProx
RFIDeas
RDR-6081AKU-78X
FCC ID: M9MPCROXHUSB100
IC: 6571A-RDR6061
Serial No: H097653





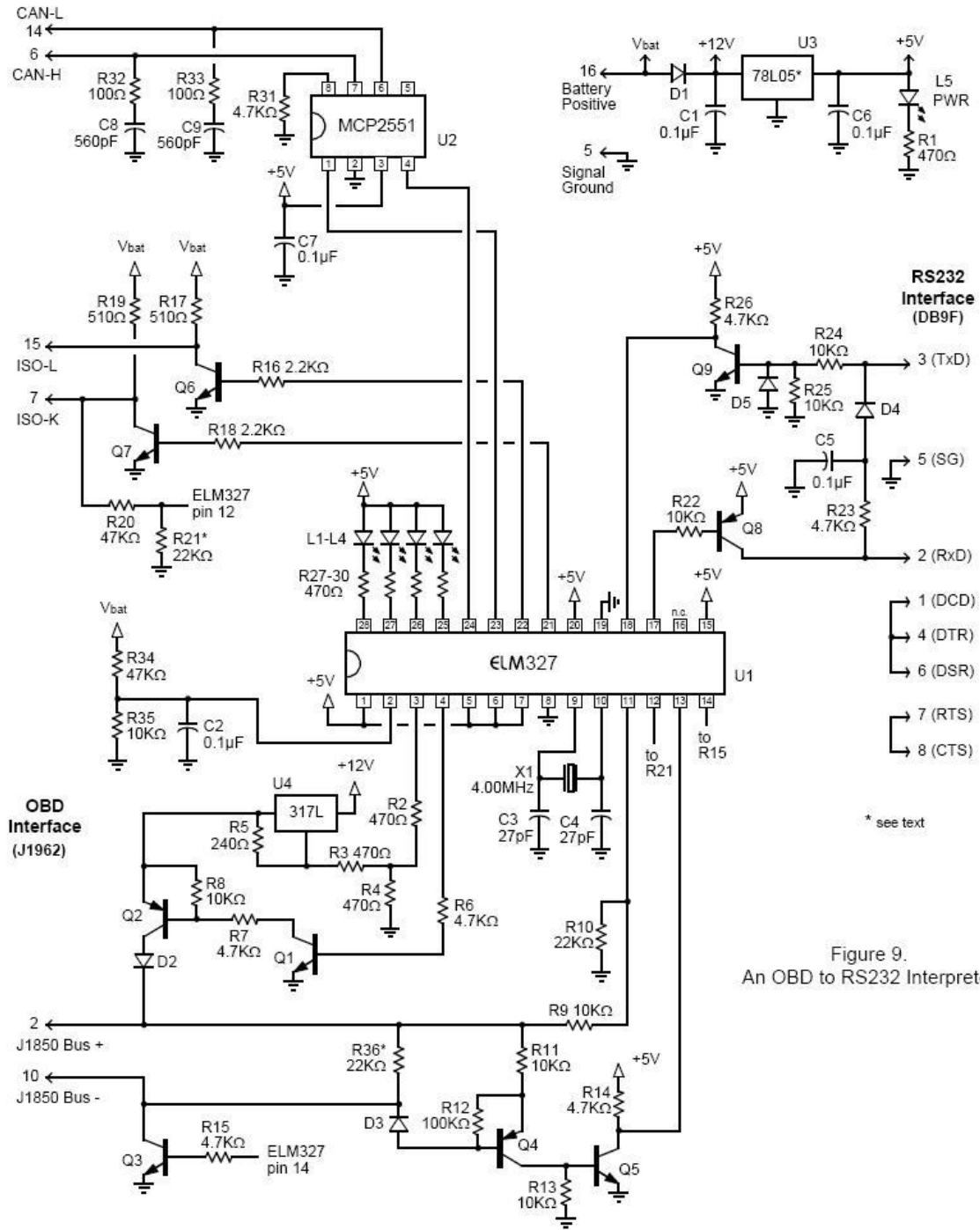


Figure 9.
An OBD to RS232 Interpreter

13. References

- 12.0 Apache NiFi Documentation
<https://NiFi.apache.org/docs.html>
- 12.1 IEEE specification 1016-2009
<https://standards.ieee.org/findstds/standard/1016-2009.html>
- 12.2 Neo 6M GPS Hardware Datasheet
https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf
- 12.3 OBD-II 327 ELM Datasheet
<https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf>
- 12.4 Raspberry PI3 Model B Datasheet
https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf
- 12.5 RFID reader
<https://www.rfideas.com/products/readers/pcprox>