DATA ANALYTICS FOR SNOW PLOW TRUCK-DATA
Design Document

SDMAY18-23
Client: Henderson Products
Advisor: Dr. Goce Trajcevski
Design Lead: Alex Smola
Git Master: Alan Peine
Test Engineer: Colin Heinrichs
Report Manager: Evan Warych
Meeting Reporter: Michael Entin
Meeting Facilitator & Communication Lead: Zach Wilson
Team Email: sdmay18-23@iastate.edu
Team Website: http://sdmay18-23.sd.ece.iastate.edu/

Design Document 10/12/17 Draft #1

Table of Contents

List of figures/tables/symbols/definitions

NOTE: This template is a work in progress. When in doubt, please consult the project plan assignment document and associated grading rubric.

1 Introduction

1.1 Acknowledgement

Thank you to James Timmermann and Henderson Products for working with us and providing us with the information for our project. Thank you, Dr. Goce Trajcevski, for your help and guidance with the the direction and management of our project.

1.2 Problem and Project Statement

The main goal of the project is to help with the visualization of data that is being received from sensors equipped to select snowplows. Currently the data is being sent from these trucks containing information about the plow including its coordinates, fuel consumption, and select statistics about the plows performance. That data is then being stored on a server, but is currently in an undesired format to be analyzed. Ultimately the product will be a web application that will display these statistics about the truck that can be studied to forecast the trucks performance in the future. With this knowledge, quality issues can be spotted before they affect performance of the truck, saving money from a potential breakdown.

The solution to this as previously stated is to create an application that hosts and helps visualize the data being received. At its current state the data is being stored on a server hosted by Henderson Products. So our application will be constantly pulling new data as it makes its way to that server, and will convert it into a readable format to be stored in our database. We will then design an interface that will allow the info to be easily read and analyzed. Most likely the product will allow clients to log on and allow them to view and monitor their own trucks and data.

1.3 Operational Environment

As our project deals with transferring, converting, storing, and accessing data; we don’t expect our product to be exposed to any notable conditions. We expect our end product to run on one or more servers, and be capable of being accessed by employees working at Henderson most likely in an office setting. Due to the fact that it will be running on servers, it may be important to account for possible failures in that domain. It’s important to note that while the data will be coming from loggers on Henderson

vehicles, the actual capturing of data is already implemented and is considered out of scope for this project.

1.4 Intended Users and uses

The intended users of the CANBus data app will be the good employees of Henderson products and possibly even their clients. The clients using the web app, however, is a bit of an assumption (as stated in the next section). Any other intended users could involve the operators of the dispensing units so that they may see the data of the vehicles they operate. It will be important to make sure that only the designated people can use this software so that Henderson's data does not get into the wrong hands.

There could be many end use cases for this app. First of all, it will reduce the need for calculations done by hand by our company contact James. This had been a very large waste of time, taking nearly half an hour to convert 3 minutes of data. Key details about the data could inform the employees when things are not going right with the CANBus system. Whether that means a part on the vehicle is broken the hydraulics readings are not what they should be, it is essential to know when things are going wrong so they can fix it as soon as possible. The data they retrieve from this app will also be used to determine better ways to create products for Henderson's clients, making their snowplows able to withstand the tests of time and the harsh environment of winter.

1.5 Assumptions and Limitations

1.5a. Assumptions

1. We will receive all information on how data converts from hexadecimal bits into relevant data.
2. We will receive access to Henderson Products server in order for us to be able to pull files from it and convert.
3. We will be able to contact our client for important information within a reasonable amount of time.
4. The web application will have users with unique logins to guarantee security and data integrity.
5. Vehicles will each have their own unique identifier.
6. Clients of Henderson Products will be able to view data sent from their own vehicles
7. The maximum amount of simultaneous users shall not exceed 100.

8. The web application will be a single page that allows the manipulation of data to be viewed and read effectively and efficiently.
9. The web application will adhere to the branding of Henderson Products
10. The completed product will not be viewed outside of the United States
11. The website will only need to be in English.
12. New desired features will be given with the understanding that time may be a factor on successful implementation.
13. We will not need to develop for the sensor, all data being sent is all data needed.

1.5b. Limitations

1. The project will be completed in its entirety by May 2018.
2. The project will not require any hardware design.
3. The project budget shall remain at zero dollars.
4. The project will not use more cellular data than the initial transfer from vehicle to server.
5. No data will be lost in transformation or translation.
6. Data will be moved to the web application within 24 hours of creation.
7. Adding a new user will take less than 2 minutes.
8. Clients will only be able to view data from their own vehicles.

1.6 Expected End Product and Deliverables

Data Log Parser

The data log parser will take the log files from the trucks, parse them, and return the data in a more manageable form. Initially this parser could return a human readable file. However, the end goal for the parser is to take the data it returns and insert it into a database.

Database

We expect to deliver the design for the database that will allow our client to store, access, and organize the truck data. The database will be populated with data extracted from log files by the parser. The database will be accessed by the client through the web application.

Web Application

The web application will be used by the client to view the data being sent from their trucks. The application will offer different ways for the client to view the information such as: graphs, maps, and tables.

2. Specifications and Analysis

Design Decisions:

List a couple of options for each, and maybe state considerations relating to client needs. This will help us justify our design decision, and can be included in the report.

- **Data Log Parser:**
    - **ANTLR**
        - **A popular language parser generator**
        - **Might be too advanced for what we need**
    - **Build our Own Parser**
        - **Can use a language that we know well**
        - **Does not need to be complex**
        - **Can set up configuration files to allow different kinds of data to be recognized**

- **Data Storage:**
    - **SQL database**
        - **Henderson works with SQL a lot**
        - **Familiar with the syntax and technologies**
        - **Short setup time**
        - **Consistent & reliable**
    - **Data Warehouse**
        - **Works better with companies in general**
        - **Can store large amounts of data**
        - **A new concept most of us aren't familiar with**
    - **MongoDB**
        - **Non-relational, which may not be good for this scenario**
        - **Can work much faster than an SQL database**

    - **Apache Cassandra**
        - **Designed to work well with large amounts of data**
        - **Also Non-relational**
        - **No experience from team working with it**

- **Web Application:**
    - **Frontend**
        - **AngularJS**
            - **Uses a MVC architecture for developing frontend applications**
            - **Experience from multiple team members**
        - **ReactJS**
            - **Uses uni-directional data flow for organizing components**
            - **Organizes views into reusable components.**
            - **Makes it easy to port to mobile with React Native**
    - **Backend**
        - **NodeJS**
            - **Can have full stack javascript**
            - **Some of our team has exposure to Node**
        - **Laravel Framework**
            - **PHP is usually not fun to code in**
            - **The framework is really great & can make some good looking apps**
            - **May not have the graphical functionalities we're looking for; can't be used to create single page apps on its own.**
        - **Java Spring**
            - **Nearly all of our team knows Java**
            - **Used widely, has great documentation**
        - **Apache Thrift**
            - **RPC framework that won't limit us in choice of language.**
            - **Gives us more freedom in terms of software design.**
            - **Forces us to define our service interfaces and the models that they work with, leading to a better design.**
            - **Abstracts away communication between software modules. (In our case the client and server)**
    - **AngularJS & NodeJS**
        - **Javascript is really great for making webapps**
        - **Angular is one of the leading techs for single page apps**
        - **Some of our team is very familiar with both and others have some experience in both**

2.1 Proposed Design

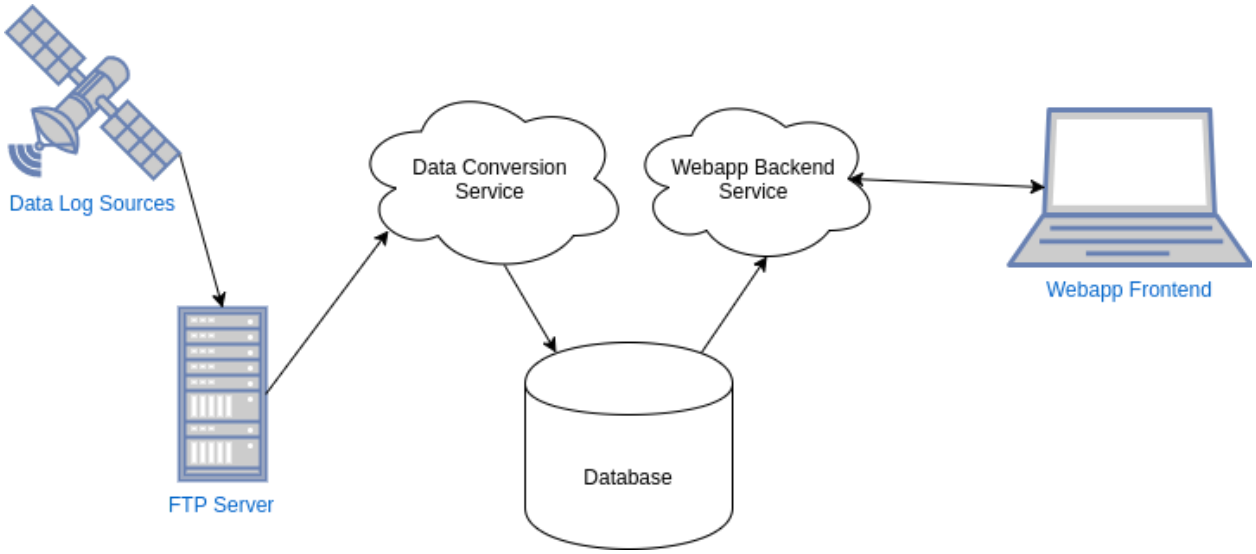

Fig. 1: Design diagram describing the layout of the different parts of our application

Our team has decided to implement a web app to solve the problem of having to convert the CANBus data by hand. You can refer to the figure 1 as I describe the different parts of our design. We will be using Henderson Product's FTP Server to retrieve the data for conversion & send it to a data conversion service we will need to create. This meets the data conversion functional requirement. From there, we will store the data in a SQL database in order to meet the data storage requirement. We have chosen SQL because the Henderson uses primarily SQL databases and it will make the transition of providing the product off to them at the end of the project much smoother. The backend service will be implemented with a RESTful API framework because it will provide the most effective means for querying truck data from the database, meeting the responsiveness non-functional requirement. Our frontend will be written using a frontend library/framework in order to meet the data analysis requirement. By doing this we can make use of frontend graphing libraries that will allow users to better visualize the data. As no requirements have been given to us by Henderson for security, we will implement an authentication service for logging in and accessing the data. All justifications for these choices can be found in the design decisions portion.

1. Functional requirements

- **Data Conversion -** Currently data is being hosted on an ftp site in the form of obfuscated log files filled with hexadecimal values. This data needs to be converted into something that makes more sense for humans.
- **Data Storage -** The converted data will need to be stored somewhere where it can be queried for in useful ways for analysis.
- **Data Access -** An interface will need to be build that gives an intuitive way to query for data that will be useful.

- **Data Analysis -** An interface will be needed to outline useful trends in data using graphing tools.

2. Non-functional requirements

- **Security Requirements -** Our project should limit access to data to the people that need it within Henderson products. This could mean locking it down to an internal network, or requiring some sort of authorization to access, or some combination of the two.
- **Responsive Requirements -** Since our project is a web app, our applications should be fast enough to meet the needs of a company. Any request for data should take no longer than 5 seconds to complete and populate on the page.

2.2 Design Analysis

Separating the solution into three distinct parts allows for us to develop and test the different parts concurrently. It also allows us to be able to use different technologies for the different components (such as a php backend and a angular frontend). We then have to option to switch out technologies without having to rework the other components.

3 Testing and Implementation

3.1 Interface Specifications

Testing for our project will mostly involve testing software, which we will split primarily into unit tests and integration tests.

Unit tests will be created to test small units of functionality in our software, and will ran both during development and before being pushed into the main branch of our repository.

Integration tests will be designed to test the connections between different modules of our system -- e.g. Database, backend service, frontend, etc. -- to ensure that they work as expected. In order to facilitate our integration tests we will need to build a separate copy of our production system that will act as a pre-production stage. This will provide us with an environment to test changes and ensure correctness before they make it to production.

3.2 Hardware and software

Junit

Junit is a testing framework for Java development that is widely adopted both in industry and in academics. It provides easy to use test fixtures that make it easy for developers to write and run unit tests. We plan on using Junit (or a similar testing framework if we decide to use a language besides Java) while developing our backend services.

Mockito

Mockito is a framework for creating mocks and stubs for Java objects and methods respectively. A mock of an object is a “dummy” object that is used while writing unit tests to isolate units of software. Mockito makes it easy to create mocks of objects and to stub out their methods to return something predetermined by the tester. This becomes useful when the unit of software under test makes a method call that we aren’t testing. By mocking the object that is being called and stubbing the method call we can ensure that our tests are consistent when they are ran.

Mocha

Mocha is a framework for testing javascript code running in the browser. This will help us ensure that our web app is working well on the client side.

3.3 Process

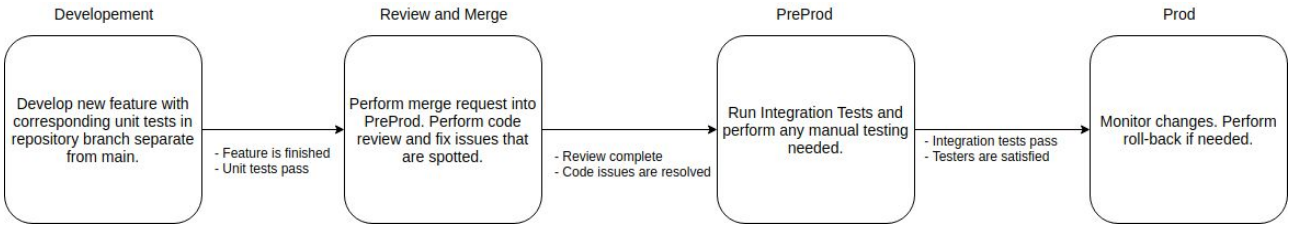

Fig 3.1 outlines our process for testing and ensuring code quality. It is split up into four stages: Development, Review and Merge, Pre-prod, and Prod. The process begins in the “Development” stage with a new feature that is being developed. In this stage the developer creating the feature will write the code implementing it, as well as an appropriate number of unit tests -- which need to pass before the “Review and Merge” stage can be reached.

In the next stage the code that has been developed is looked over by another team member. This is achieved by performing a merge request into the pre-prod branch of our repository. Once the merge request has been made a code review needs to performed to verify the code change is satisfactory. Once another team member has accepted the merge request the change can then be merged into the preprod branch of our repository; leading us into the “PreProd” stage of our testing process.

The Preprod stage is where we will test our changes in an environment that is similar to our production stage. It exists to allow us to spot bugs before they can have negative effects such as downtime or loss of data. It is also where we will perform integration tests between the different modules of our system. Once the integration tests pass and the testers are satisfied, the change can then be pushed into production.

3.4 Results

– List and explain any and all results obtained so far during the testing phase

· – Include failures and successes

· – Explain what you learned and how you are planning to change it as you progress with your project

· – If you are including figures, please include captions and cite it in the text

• This part will likely need to be refined in your 492 semester where the majority of the implementation and testing work will take place

4 Closing Material

4.1 Conclusion

The goal for our project is to make it easier for our client to understand the data that they are receiving from their trucks. Currently, the information is being stored on a server hosted by Henderson Products, and then manually converted from hexadecimal values. We are going to be creating a web application that will help convert this information faster. This information is being used to predict the performance of each individual truck and will help the clients to better monitor their own trucks. We will be formatting the information that they are receiving so that they will be able to see how the quantities of each piece of information over time.

4.2 References

https://thrift.apache.org/ We might be using this website's software framework for our design.

4.3 Appendices

Any additional information that would be helpful to the evaluation of your design document.
If you have any large graphs, tables, or similar that does not directly pertain to the problem but helps support it, include that here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc. PCB testing issues etc. Software bugs etc.