TLDWorkerBee

Team: Austen Christensen, Morgan Lovato, Wei Song
Sponsor: Harlan Mitchell - Honeywell
Mentor: Austin Sanders

Software Design Document - Version 2
February 26, 2019

Table of Contents

1. Introduction

Planes are the most popular way to travel quickly to anywhere around the world. For instance, a traveler can get from Phoenix to Los Angeles in a little over an hour by plane when it takes 8+ hours to drive. In order to travel such long distances, these machines must maintain a cruising altitude of 33,000 to 42,000 feet. Every day, there are over one hundred-thousand flights scheduled across the globe with up to one million people in the air at any given point in time. This makes it vital for every plane's engine to constantly be working properly since there is nothing stopping a plane from falling out of the sky other than it's own momentum.

Our team is TLD Worker Bee, and we are working on the project Prototype Time Limited Dispatch (TLD) Application for our sponsor, Harlan Mitchell from Honeywell Aerospace. Honeywell Aerospace is a leading manufacturer of all sorts of aircraft engines ranging from helicopters to commercial airliners. They are the leading manufacturer in engine control systems for a variety of private and commercial jets. These engines and their connected systems generate data every flight that is important for the functionality of their product. While in flight, an engine is constantly reading sensor data and storing it on the onboard computer called the Engine Control Unit(ECU). The computer will read the data and create a time-limited dispatch; time-limited dispatch allows the degraded redundancy dispatch of aircraft. Aircraft can be dispatched with certain control system faults and fault combinations for specified periods of time if the failure rates from those configurations meet certification requirements. The various system faults and fault combinations are assigned to dispatch categories according to these failure rates. This gives the dispatch criteria for the system.

Currently, to gather this data, a technician will physically download the data from the ECU through a wire connection. They do not check it after every flight, but will connect periodically to the ECU to retrieve the data. The cumbersome process of physically connecting to a computer and downloading this data on location greatly limits the amount of flight data to collect.

Our prototype will be a webapp that uses an internet connection to connect to the data stored in the cloud for a completely wireless experience. It will verify data integrity before showing the user any data to avoid reading false data. This will ensure the mechanic knows exactly what maintenance to perform on the engine from anywhere in the world.

2. Implementation Overview

The solution our team has in mind to build for our sponsor is a prototype web application that will serve as a viewing tool for data that is stored in a cloud. The web application will be able to download the data files a user is requesting from the cloud and display them in the web browser of choice. The web viewing tool we build will be usable on Google Chrome and Apple Safari.

We have three main components that will each serve a purpose in our solution: cloud storage, parsing and verification services, and a web application. The cloud storage contains the databases for this prototype: one to store data for processing and one to store data that is already processed. These databases communicate with each other using Python to perform operations on the pre-processed data. We will be using Amazon S3 cloud storage to hold the database containing the pre-processed data and Amazon RDS to hold the database containing the processed data. The RDS database is the one that will be accessed when a user makes a data request in the web application. Before the data reaches the web application, it will be sent through a parsing and a verification tool. These tools/services will ensure data integrity throughout the data flow process. For the web application itself, we will be using Django and Python to create a web page to display data that a user requests from the database. This flow of data can be seen below in figure 1.

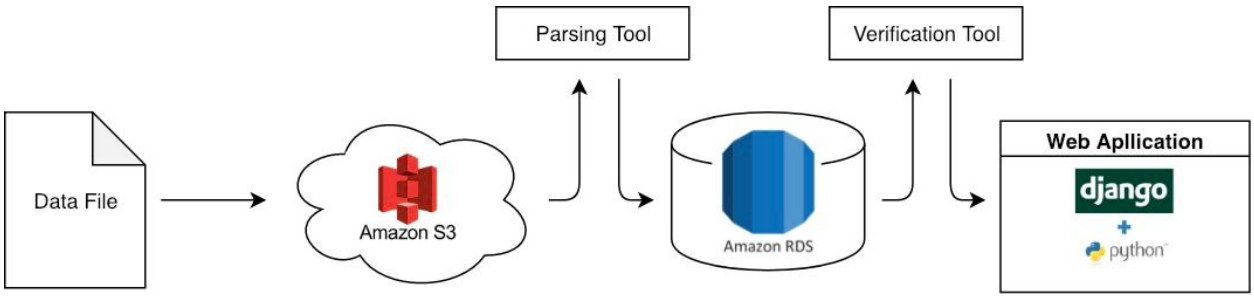

Figure 1 : Data Flow Diagram

The data starts by being collected from the ECU and is then sent to the Amazon S3 cloud database. From there, the data is sent through a parsing tool for data processing and is then sent to the Amazon RDS database. Once the user makes a request for data, it is passed through a verification tool before it is displayed to the user in the web application.

3. Architectural Overview

The components discussed in the previous section will be used to create software that adheres to the Model View Presenter (MVP) model, a key part of which is that data is handled and represented in three separate layers. These layers are as follows:

- Database Layer: where the data used by the software is stored(Model)
- Presentation Layer: where the data used by the software is displayed(View)
- Service Layer: where the data used by the software is parsed and verified.(Presenter)

This separation of responsibilities surrounding the data into separate layers ensures a level of security with data parsing and makes sure that the data that is displayed is accurate. This configuration is shown in figure 2.

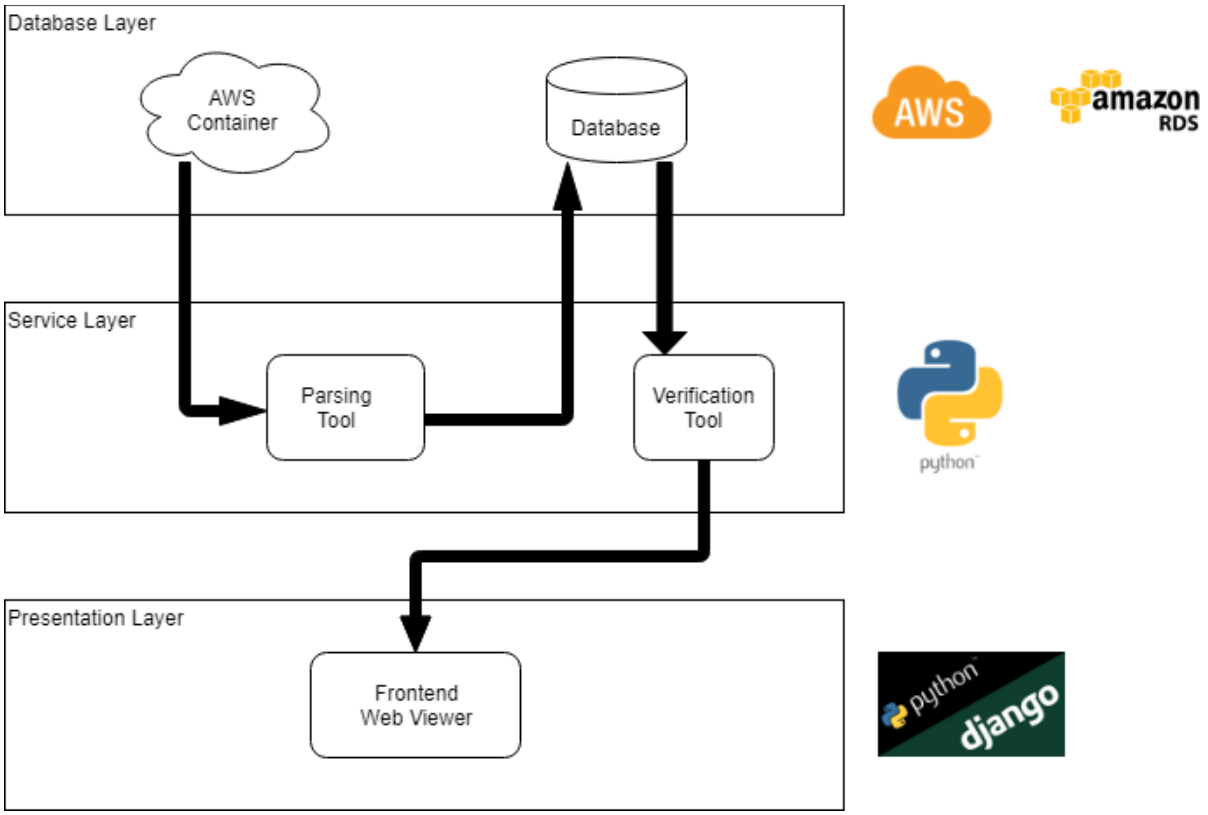

Figure 2 : High-Level Architecture Diagram

The Database Layer consists of two major components: Amazon RDS and S3 cloud storage. The S3 storage is where the data is sent from the ECU and is stored for processing. The Amazon RDS database will be used to store SQL tables which will house the processed data.

These two components communicate with the service layer, which will be the core backend of this project, consisting of several modules written in Python that perform operations on the data. These modules will primarily be responsible for parsing through the data in the cloud storage to store in the database and validating data integrity.

After the data is stored in the database, it will wait for the user to request to view. Once the user makes a request, the data will be passed into a verification tool that will verify data integrity before displaying it to the user. In the Presentation Layer, the data will be displayed in a way that is easy for an experienced user to understand.

While this arrangement of layers may seem unnecessary or redundant, it is necessary to ensure data integrity throughout the process. This is due to the importance of an engine to be maintained properly.

4. Module and Interface Descriptions

In this section we will illustrate the finer points of our system's architecture. For each module in the design, we will provide a brief description of its purpose, create a diagram that outlines how this module fits into the larger whole, and describe exactly how and when this module is interfaced. These technical descriptions will serve as the blueprint to the TLD Application and establish how such a system should be created.

4.1 Module: Database Layer

The database layer will consist of two different databases: one to hold pre-processed data and one to hold processed data. The data collected from the ECU will be stored in the cloud; we will be using Amazon S3 to store this raw data in the cloud for our solution. From here, it will go through a parsing tool before reaching the database. We will be using Amazon RDS to store this processed data for our solution. When a user makes a request from our web app, they will be requesting access to data in the Amazon RDS database; this data will go through a verification tool before reaching the user. Figure 3 illustrates this flow of data.

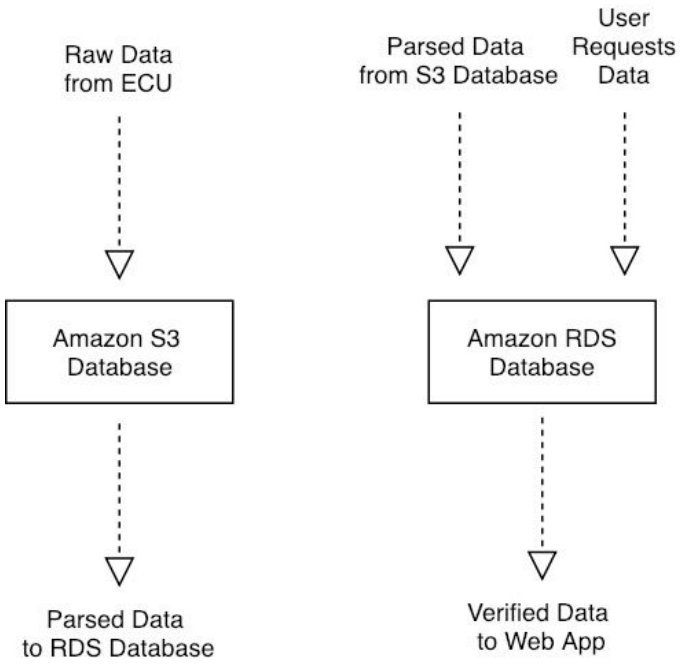

Figure 3 : Overview of Database Layer Dataflow

4.2 Module: Service Layer

The service layer will be responsible for two major components: maintaining data integrity and parsing the raw data into a database. When the data file gets stored in the cloud, our parsing tool will begin parsing through the data and passing it into the database along with an MD5 hash code. Once the user makes a request for the data, the data will then get passed into our data integrity checker which will cross check the stored hash with the one created on the user's machine upon receiving. If the MD5s do not match, then the request is made again. If the hashes don't match three times in a row, the data stored in the cloud (if it still exists) will be reparsed into the database. If the file does not exist, an error message will be sent to the Presentation Layer. Figure 4 illustrates this flow of data.

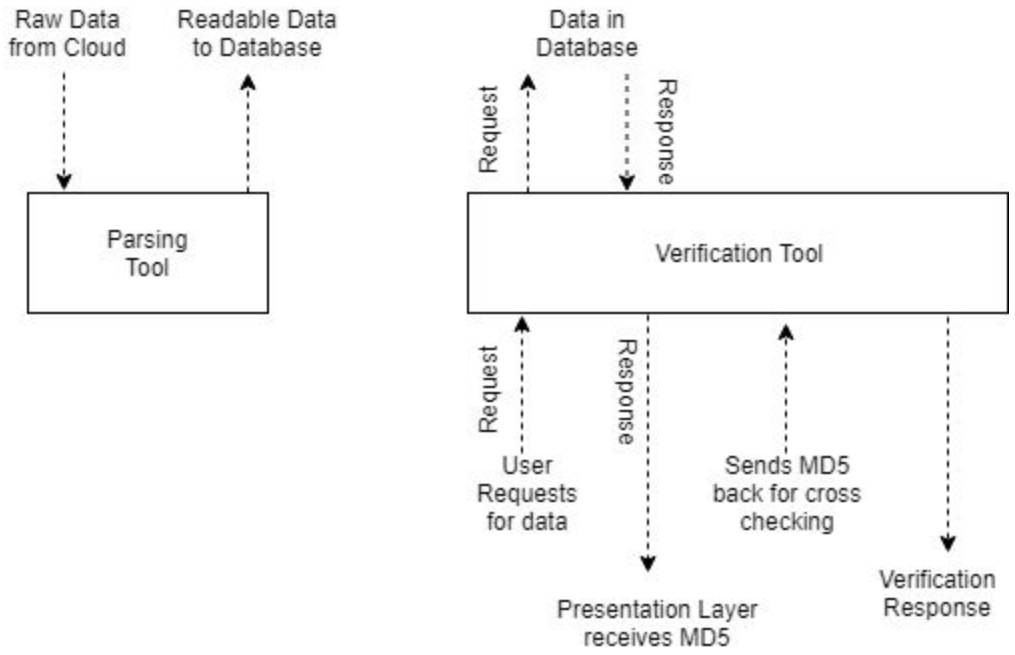

Figure 4 : Overview of Service Layer Dataflow

As mentioned above, the Service Layer is composed of two parts. The first part is the parsing tool. The parsing tool is a major factor in the system because it creates the MD5 hashes for each entry in the table so the data can be verified by the verification tool. The data will be passed into the parsing tool as a raw data file which the parsing tool will then break the data up into its correct format based on the corresponding config file. The data will then be passed into the database to be stored along with its MD5 hash. Figure 5 displays this flow and manipulation of data.

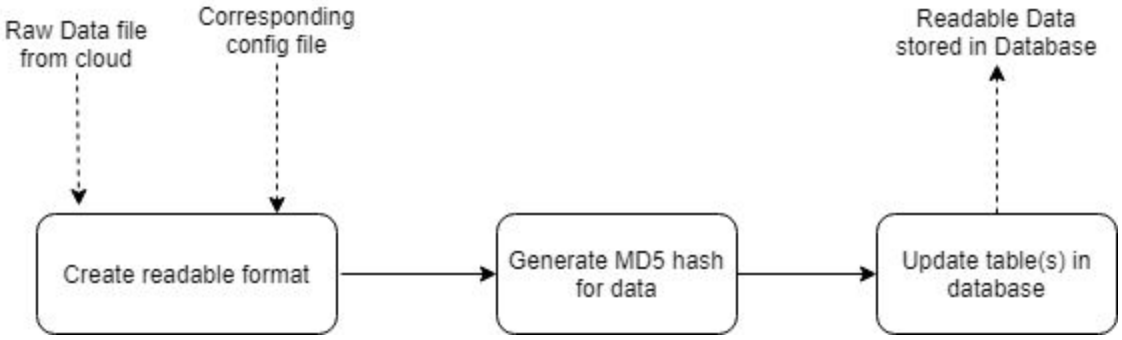

Figure 5 : Parsing Tool Dataflow

The second part of the Service Layer is the verification tool. The verification tool is the key component of the system. Since it is so vital that these airplanes are serviced properly, the data needs to maintain its integrity 100% of the time throughout the

system. If at any point in time the data changes or loses its integrity, the system needs to know right away and fix the problem. This happens by verifying the MD5 hash before the user sees the data. If the hash doesn't match, then it will try to retrieve the data again. If the data keeps failing, the parsing tool will try to reparse the datafile if it still exists in the cloud. This process is outlined in figure 6.

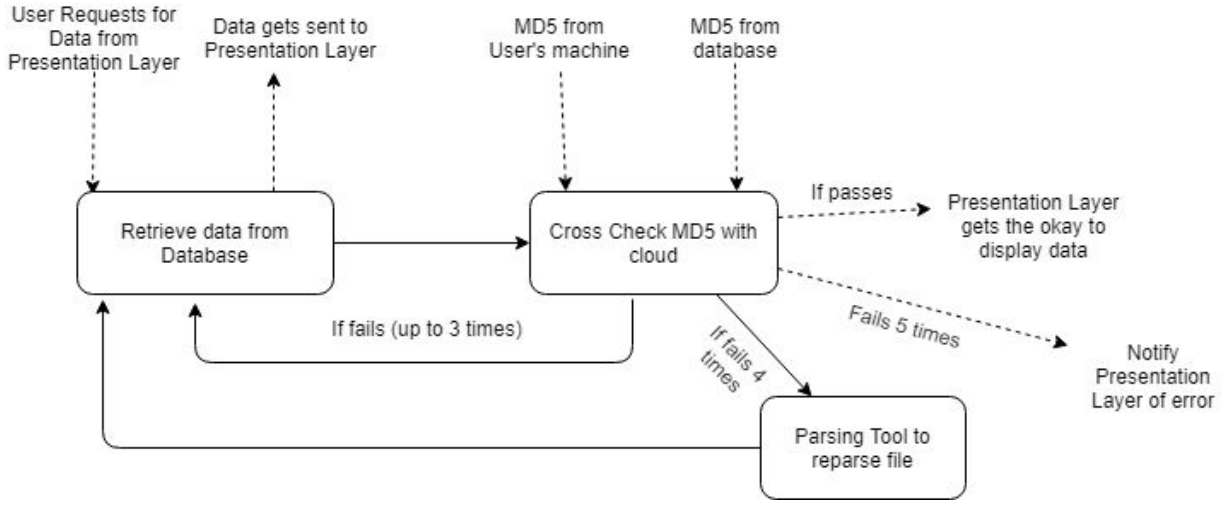

Figure 6 : Verification Tool Dataflow

4.3 Module: Presentation Layer

The presentation layer module consist of several components in order to present the validated TLD data to the clients and users. After being downloaded from the cloud module and processed by the service module, we have make sure the integrity of TLD data, and is ready to present them to the users. Before presenting the data, we still need to compute and compare their MD5 hash value in order to guarantee that the data being displayed is accurate. This module is able to provide an organized ways to arrange and display the TLD data by using easy-to-read table and grid structure. And in order to help user locate the data they want to access more conveniently, we designed several components and tools like search bar and filter tool for the specific plane and TLD data selection. The UML class diagram for this module is presented below.

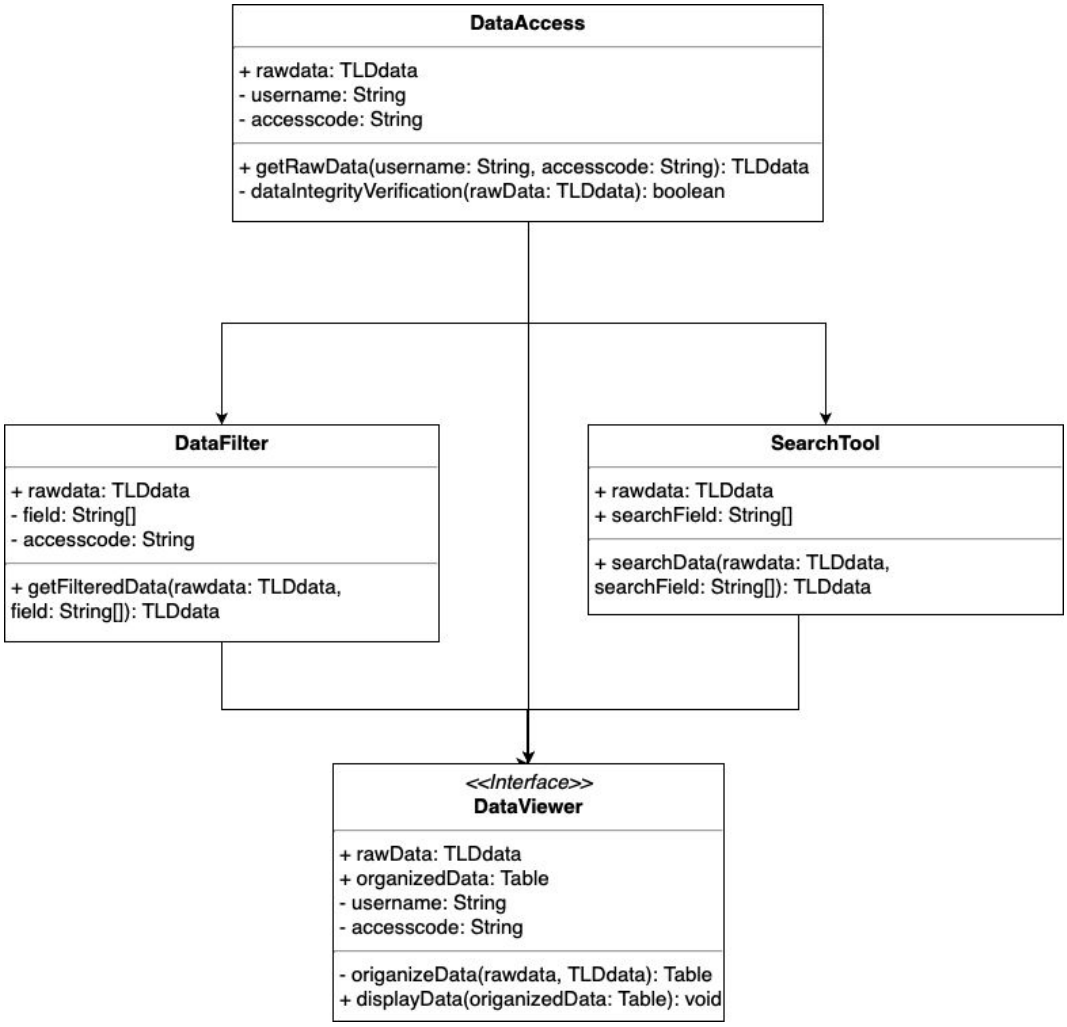

Figure 7 : UML Class Diagram of the Presentation Layer

Since users can not modify or interact or modify the TLD data, the public interface of the component that this module provide to display the data will be the DataViewer component. This DataViewer component will in charge of getting all the data from database, or only fetching the data that the users request by the searching bar or filter component. It contain one public method to display the TLD data: displayData(), which return void but print the data on the screen. It also take one parameter: originizedData, which is table-formatted data type and generate from raw TLD data by using private originizeData.

There are other components in this module that contains the public methods.
1. getFilteredData. This method take the raw TLD data and the filter field as its parameters, and return the filtered result.

2. searchData. This method take the raw TLD data and the searching keyword as parameters, and return the searching result.
3. getRawData. This method take the user information like username and access code as parameters in order to make sure that the TLD data can only be accessed by the users with correct authorization. It return the raw TLD data.

5. Implementation Plan

In order to plan out the schedule for implementation, there are several things on which we need to focus. The first one is to divide the system into several modules. The advantage of this is we can have a better overview of the development process, understand the importance of each modules and make sure that the important and fundamental modules have top priority in the whole implementation process. The second one is to ensure a smooth transition between each tasks and modules, and consider the unexpected circumstances of any delays. The third is to increase parallelism with multiple modules being worked on any given time, which will help to improve efficiency.

The Gantt chart that outlining the plan of implementation and schedule is presented below. It shows the order on how parts of this project will be done. Six high level overview of tasks and modules are included in this Gantt chart, which contain a substantial amount of work in each tasks.

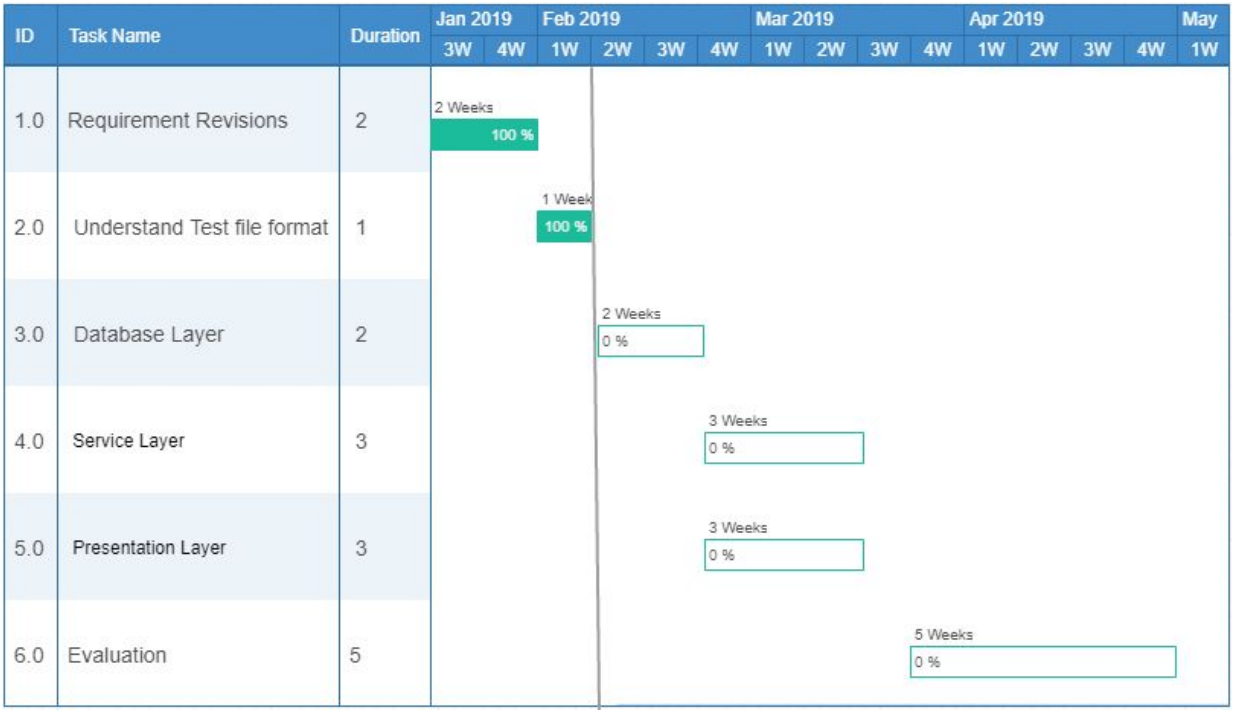

Figure 8 : Gantt Chart

During the last semester, we had built a minimum viable product, and modify the requirements based on the feedback from our client. The tasks we have completed during the first three weeks of this semester include the requirements revisions and understanding the test file format. The requirements revisions took the first two weeks, which we have modified the requirement documents based on the suggestions from our client and mentor. This helps us to have a better understanding of what our client really need and what will be our tasks for the capstone project this semester.

From week 4 to week 5, we will focus on the implementation of the Database layer, which includes the tasks to establish the cloud and database. These are the fundamental components of our system since we can not build other modules without establishing the database and cloud. And it can only be implemented after the full understanding of the format of data files in the third week.

After making sure the database layer functionality is working properly, then our tasks will be implementing both the service layer and presentation layer in parallel. Our plan is to implement these two parts together for the following 3 weeks, from week 6 - week 9. The service layer is in charge of maintaining data integrity and parsing data to the database, and the presentation layer is in charge of downloading the data from the database and display them. Since the only connection between these two modules is the database, we can put some testing data into the database at first so that the presentation layer can be implemented without the actual TLD data, and these two layers can be implemented in parallel without any conflict.

Week 10 is vacant for the spring break, and also for the unexpected delays during the implementation process. After that, the team will finish our evaluation and assessment during the next 5 weeks. In this period, we will focus on testing the whole applications in order to make sure the application has met all the requirements. Each individual also needs to be evaluated to ensure they are all working as expected. We may still improve the GUI or add new functionality during this period, if time permitted. At the end of this semester, an acceptance test will be performed before deliver the final product to our client.

6. Conclusion

With planes being a necessary source of travel for some and a leisure for others, it is vital to human safety that they are always functioning properly. Our client, Honeywell, produces many plane engines and it is their job to make sure they are always in working order. If something does go wrong, technicians need to be able to quickly identify and

fix a problem. Currently, technicians need physical wire connections and people on site collecting data to analyze their engines. This process is tedious and time consuming, slowing down the analysis of a potential engine failure. The current method of data collection can be greatly improved with the use of our prototype web application. With cloud based databases and wireless data transfer, our sponsor can quickly and more accurately analyze the data they are getting off their engines. Plane technicians will be able to open a web browser to our application, search for the plane data they are interested in, and view that data faster and with more accuracy than they can currently.