Requirements Document
Version 1
December 2, 2019
EnginAir

Sponsor:
Harlan Mitchell, Honeywell
(harlan.mitchell@honeywell.com)

Mentor:
Scooter Nowak
(gn229@nau.edu)

Team Members:
Chloe Bates (ccb323@nau.edu),
Megan Mikami (mmm924@nau.edu),
Gennaro Napolitano (gennaro@nau.edu),
Ian Otto (dank@nau.edu),
Dylan Schreiner (djs554@nau.edu)

Accepted as the baseline requirements for the project:

For the client: ______________________________ Date: _________

For the team: ______________________________ Date: _________

Contents

1. Introduction

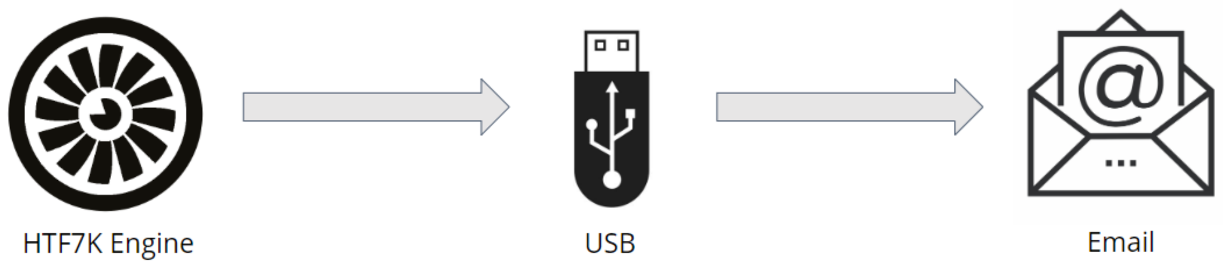
In 1903, the Wright Brothers made history by building and flying the first successful powered airplane. Since then, airplanes have become the most commonly used mode of transportation and helped spark the advancement of the Aerospace and Defense industry. The Aerospace and Defense industry is comprised of the manufacturing, sale, service of aircraft, aerospace parts, space vehicles, and military defense systems. The U.S. Aerospace and Defense industry is the largest in the world having a revenue of $838 billion in 2018 and providing over 2.5 million jobs, 20 percent which comes from manufacturing [1][4]. The top three Aerospace companies in the U.S. based on 2018 revenue are Boeing, Airbus, and United Technology Corporation [3].

Our project focuses on the manufacturing of engines, specifically turbofans. Turbofans are a type of jet engine also called a gas turbine which is primarily used for aircraft propulsion. These jet engines produce thrust through the burning of fuel which gets released as hot gas. This gas is forced through the turbine blades causing them to rotate. Gas turbines, however, require a sufficient amount of airflow between the blades before introducing fuel and starting combustion. If the turbine blades are not pushing enough airflow before this happens, a hot start occurs which causes the engine to overheat and results in damage. Auxiliary Power Units (APUs) are smaller turbine engines that generate high-pressure exhaust which is used to kickstart the turbine blades to prevent a hot start. APUs can also be used as an additional power source for aircraft electrics like lighting, cockpit avionics, and environmental packs that are used to heat and cool the cabin [2].

Our sponsor, Honeywell, is the largest producer of gas turbine APUs, with more than 100,000 produced and over 36,000 still in use today. Honeywell APUs are found on common commercial aircrafts such as the Boeing 747 and Boeing 777. Other Honeywell applications are found on helicopters, military jets, and on the U.S. Army Abrams Tank. Our sponsor contact is Harlan Mitchell and he is the Systems Technical Manager for the HTF7K Controls System Integration Unit. The HTF7K is a turbofan engine family primarily used on business jets like the Cessna Citation Longitude jet.

2. Problem Statement

During a flight, the Engine Control Unit (ECU) saves trending and maintenance data which is reported to the mechanics and engineers via a hardware diagnostic report. Proper upkeep is important to prevent maintenance delays and keeps the engine functioning properly to ensure a safe flight.



Currently, Honeywell engine operators are required to download and send engine diagnostic data reports once a month. This process requires:

1. a manual port connection using a USB device and a cable,
2. transferring a file to a USB drive, and
3. sending the file via email.

This process is tedious and collects a small data set containing basic maintenance information. Because this process occurs once a month, it can result in infrequent data collections and missed maintenance opportunities.

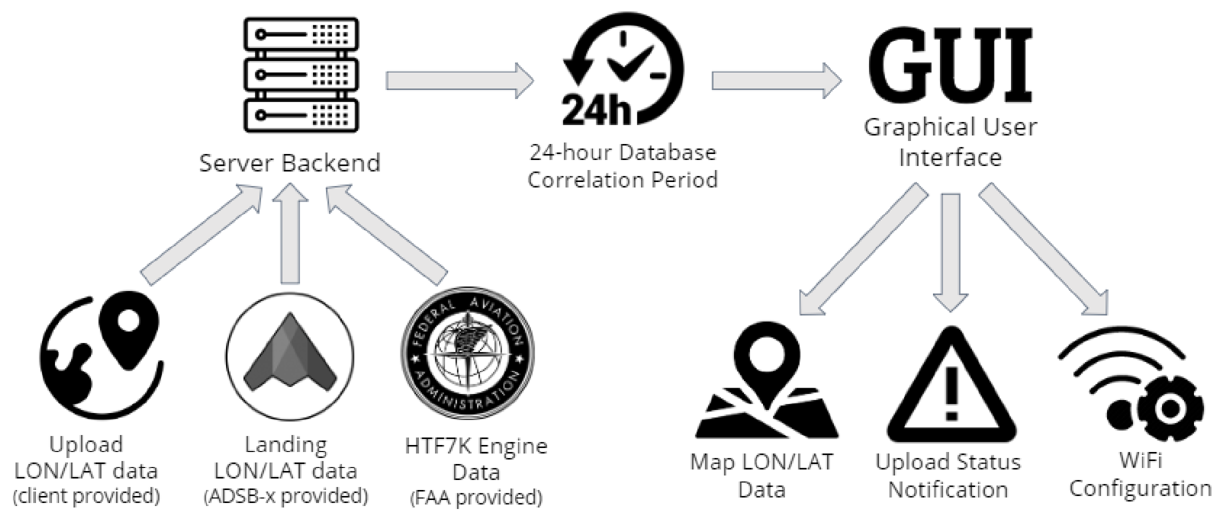
To better this process and collect flight data more frequently, Honeywell is currently developing a connected engine product called the Connected Engine Data Access System (CEDAS). CEDAS allows engines to autonomously upload engine data wirelessly to a cloud. The CEDAS is hosted on an embedded computer located in the aircraft along with a WiFi antenna. If the WiFi connection is spotty or nonexistent at a certain location, the diagnostics may not send. When this happens and the data upload is not on schedule, it is difficult to determine the status of the aircraft; whether it is grounded, inflight, or if there is a potential problem with the engine.

Our team, EnginAir, is designing a solution to help account for the diagnostic data gaps and provide a visual representation of WiFi configurations.

3. Proposed Solution

To solve our client's problem, our team has come up with the Connected Engine Upload Status System (CEUSS). Two primary goals of this system are: [1] provide airplane operators a way to know where to park their aircraft for the highest chance of upload success and [2] provide engineers a way to be notified and help visualize where and when potential problems occur. CUESS is composed of two main components: the server-side backend software which is primarily responsible for the data import and execution of database correlations, and the administrative front end web panel which is primarily used as a graphical interface to display notifications and LON/LAT locations.

A high-level workflow of our solution is outlined below:



3.1. Server Backend

The server backend software will implement many of the core features of our overall system. This software is responsible for accessing the engine upload Excel files and extracting critical information such as tail numbers and upload LAT/LON coordinates. The software will then interact with the ADSB-Exchange API to grab landing LON/LAT locations for the aircraft tail numbers which have an HTF7K engine. Due to security and privacy policies at Honeywell, they were not able to provide us a list of aircraft tail numbers that they worked with. So, our team has used the FAA public API to compile a list of tail numbers that contain an HTF7K engine as sample data within this project. This information will be added to our database which will be correlated with the landing and upload databases to calculate aircraft upload status and possible upload failure explanations.

Our team elected to build this software using Java due to our familiarity with the language and the availability of open-source libraries.

3.2. Administrative Front End Web Panel

The front-end web panel will implement the graphical interface of the system that will be used by both aircraft operators and Honeywell engineers. The goal of this administrative panel is to provide a visual representation of the information correlated from the landing and upload databases like LAT/LON coordinates, upload failure notifications, and WiFi connectivity and configurations.

Our team has elected to use NGINX, Node.js, MongoDB, and Bootstrap for our web application. NGINX was chosen for its superior load balancing and caching capabilities and Node.js was chosen for its easily integrable routing framework and multithreading library. MongoDB was chosen because it is easily compatible with JSON files, Java and Node.js. Bootstrap was chosen for its simplicity and ease of use in creating sleek and modern web pages.

4. Project Requirements

We describe our requirements in the form of user stories to explain the functionality for the different types of end-users: engine technicians and airplane operators. Engine technicians are more interested in knowing the technical aspects of the aircraft upload process (i.e., upload failure explanations, landing locations, status of upload entry, etc.) while the aircraft operators (i.e. pilots) are more concerned with the big picture like where to park the aircraft to ensure a successful upload.

The eight high-level user stories are listed below:

As an engine technician, I want to be able to:
- view all aircraft landing locations, every 24 hours.
- visualize all flights that are currently in progress.
- simulate various locations and their corresponding WiFi configuration.
- know the status of each landing/upload entry.
- visualize the status of each upload entry
- run a report to determine the cause of a failed upload.

As an aircraft pilot/operator, I want to be able to:
- visualize locations on where to park the aircraft for an upload success.
- simulate locations and their WiFi strength.

This section further describes the Functional, Non-Functional, and Environmental requirements for each of the user stories listed above. The General Requirements are requirements essential to the entire system whereas an individual user story.

4.1. General Requirements

The following naming convention for General Requirements is as follows: [<G><Type><Number>.<Type><Number>] with Type: Functional (F), Non-Functional (P) and Environmental (E) requirement. Note: not all fields must be occupied.

[GF1] System shall create an Upload Database that contains the following information regarding an upload from the ECU to the cloud database:
- Engine roll-down GPS location
- Engine roll-down time/date
- Engine start GPS location
- Engine start time/date
- Upload GPS location (assumed to be halfway between roll down and start location)

- WAP signal strength
- WAP ID
- Airport Code

[GF2] System shall create an **Access Database** (Wireless Application Protocol (WAP) Database) contains the following information regarding aircraft landing locations and WiFi configurations:
- WAP ID
- WAP GPS Position
- Airport Code

The following attributes are needed for each entry but are not necessary for the scope of this project:
- SSID
- Password
- MAC Address
- Public/Private Boolean
- Honeywell WAP Validated (Yes/No)
- WAP Model Number

[GF3] System shall create a **Landing DB** data shall include with ADSB-x data that provides LAT/LON locations of aircraft, landing and takeoff locations.

- **[GF3.E1]** Data is used from downloadable archived JSON files from the first of the month, using the last three months.

[GP1] Databases shall be populated and updated every 24 hours.

[GP2] System shall correlate database entries every 24 hours.

[GP3] Backend server shall run and collect upload data every 24 hours.

[GP4] Backend web application shall have an average TTFB of 200ms/request.

[GP4] Database queries shall be executed within 5 seconds.

[GE1] All graphical interface components shall be compatible with Google Chrome, Safari, and Firefox Internet browsers.

[GE2] All data files shall be in JSON or Excel format.

For purposes of graphing or identifying LAT/LON locations:
[GE3] The scope of the locations shall be restricted to the that within the US.

4.2. Technician Requirements

The following naming convention for Technician Requirements is as follows: [<T><Number>.<Type><Number>] with Type: Functional (F), Non-Functional (P) and Environmental (E) requirement. Note: not all fields must be occupied.

[T1] As an engine technician, I want to be able to view all A/C, with an HTF7K engine, landing locations, every 24 hours.

- **[T1.F1]** A list of aircraft tail numbers, with HTF7K engines, must be created to be used for reference.
- **[T1.F2]** The landing DB shall include an entry for each aircraft containing an HTF7K engine.
- **[T1.E1]** The list of tail numbers shall be obtained via the 2019 FAA aircraft registration database.
    - The registration database contains the following necessary attributes.
        - N-Number: Identification number assigned to aircraft
        - Engine Model Code: Code assigned to the engine model
- **[T1.E2]** WAP DB information shall be obtained via client provided test data.
    - The test data includes the following:
        - ESN: Upload identifier
        - Longitude/Latitude Coordinates: Location coordinates of a completed upload accurate to two decimal points (1.11km)
        - GMT date/time: Greenwich Mean Time Zone date and time upload occurred

[T2] As an engine technician, I want to be able to visualize all flights that are currently in progress to predict when an upload should occur.

- **[T2.F1]** A graphical interface shall be created to allow user to view all flights in progress.
    - **[T2.F2]** The interface shall plot LAT/LON points of aircraft flights in-progress ("in-progress" is defined as flights that are not grounded).
    - **[T2.F3]** The interface shall provide an "update" feature that renders the illustration with the most recent LON/LAT locations.
- **[T2.F4]** The system shall locate possible landing location date and time.
- **[T2.F5]** The graphical interface shall integrate with the Upload DB.

- **[T2.P1]** The update feature shall load new LAT/LON locations within 5 seconds.

[T3] As an engine technician, I want to be able to simulate various locations to predict if an upload will be successful based on the WiFi7 configuration.

- **[T3.F1]** The graphical interface shall provide a mechanism for the user to simulate landing scenarios to test probability of upload success.
    - **[T3.F2]** User shall be able enter number of hours parked at a current landing location as well as a next flight landing location.
    - **[T3.F3]** User shall be able to navigate a map.
    - **[T3.F4]** User shall be able to view WiFi configuration at specific points.
    - **[T3.F5]** System shall be configured to highlight test aircraft.
- **[T3.F6]** System shall integrate with the Upload DB.
- **[T3.E1]** WiFi configuration shall be obtained from the Upload DB.

[T4] As an engine technician, I want to know the status (upload occurred - green, upload pending - yellow, upload failed - red) of each landing/upload entry.

- **[T4.F1]** Upload DB entries shall be classified using the following criteria:
    - Within 24 hours of landing, if there has been uploaded recorded, the entry is tagged GREEN.
    - Within 24 hours of landing, if there has not been an upload recorded and another flight has not been started, the entry is tagged YELLOW.
    - Within 24 hours of landing, if there has not been an upload recorded and another flight has been started, the entry is tagged RED.
    - Note: “flight” refers to the same tail number
- **[T4.F2]** Upload DB shall integrate with Landing DB.
- **[T4.E1]** Engine roll-down and start data shall be obtained from the client provided test data.

[T5] As an engine technician, I want to be able to visualize the upload status of each upload DB entry.

- **[T5.F1]** The graphical interface shall render information from the upload DB to correlate upload location with upload status.
- **[T5.F2]** The graphical interface shall allow a user to graphically view the landing DB status on a map.
    - **[T5.F3]** The graphical interface shall render the upload status as colored dots matching the status color (green, yellow, red).

  - o **[T5.F4]** The graphical interface shall allow a user to view the upload entries by status (i.e. view all green status, all yellow status, red status).

[T6] As an engine technician, I want to be able to run a report to determine the cause of the failed upload.

- **[T6.F1]** The system shall allow the user to run a diagnostic test to determine cause of upload failure.
- **[T6.F2]** The system shall correlate cause of failed upload based on the following criteria:
  - o **[T6.F3]** The system shall execute the corresponding action.
  1. Airport has no WiFi or has not been configured by any aircraft (i.e. No landing aircraft or any other aircraft has the airport in their WAP DB).
     - ➢ The system shall display a list of airport codes and number of landings at that airport.
  2. Aircraft does not have WiFi configured at the airport (i.e. Other aircraft have had successful uploads from the airport but not the current aircraft).
     - ➢ The system shall display a list of aircraft that have visited an un-configured airport over 5 times in a given month.
  3. Possible broken EDG-100 (i.e. Aircraft has landed at three airports that have a WiFi configuration and have uploaded in the past, but a current upload has not occurred).
     - ➢ The system shall display a list of aircraft.
  4. WAP credentials have changed (i.e. Aircraft has uploaded data at a specified airport in the past, but not this time and aircraft has uploaded data at least once since landing, in question, at another airport).
     - ➢ The system shall display a list of aircraft.

4.3. Aircraft Operator Requirements

The following naming convention for Aircraft Operator Requirements is as follows: [<A><Number>.<Type><Number>] with Type: Functional (F), Non-Functional (P) and Environmental (E) requirement. Note: not all fields must be occupied.

[A1] As an aircraft pilot/operator, I want to be able to visualize locations on where to park the AC for the best upload success (based on previous uploads).

- **[A1.F1]** The graphical interface shall render information from the upload DB to correlate upload location with upload status and WiFi strength.

- [A1.F2] The graphical interface shall allow a user to graphically view the landing DB status on a map.
    - o [A1.F3] The graphical interface shall render the upload status as colored dots matching the status color (green, yellow, red).
    - o [A1.F4] The graphical interface shall allow a user to view the upload entries by status (i.e. view all green status, all yellow status, red status).
- [A1.F5] The graphical interface shall allow a user to visualize different WiFi configurations/strengths.
    - o [A1.F6] The graphical interface shall render the strength of WiFi as circular areas, varying in opacity.

[A2] As an aircraft pilot/operator, I want to be able to visualize locations and their WiFi strength via a simulation to determine which location is the best for a successful upload.

- [A2.F1] Graphical interface shall provide a mechanism for the user to simulate landing scenarios to test probability of upload success.
    - o [A2.F2] User shall be able enter number of hours parked at a current landing location as well as a next flight landing location.
    - o [A2.F3] User shall be able to navigate a map.
    - o [A2.F4] User shall be able to view WiFi configuration at specific points.
    - o [A2.F5] System shall be configured to highlight test aircraft.
- [A2.F6] System shall integrate with the Upload DB.
- [A2.E1] WiFi configuration shall be obtained from the Upload DB.

5. Potential Risks/Challenges

With the complexity of our solution, we predict there will be some potential risks and challenges that may impact our project progression. A risk is potential problem that could have detrimental effects whereas a challenge is an inconvenience that we will need to assess and work around. We have analyzed three risks and two challenges and have thought of ways to handle each issue as described below.

5.1. Risks

RISK	DESCRIPTION	SEVERITY	LIKELIHOOD
ADSB-x Downtime	Database Corruption, unable to check for upload failures	High	Low
ADSB-x Accuracy	Locations reported by ADSB-x are not always accurate, and are crowdsourced.	Low	Medium
Data Import Inconsistencies	Database Corruption, schema modifications	Medium	Medium

5.1.1. Flight Data API Downtime

Our system relies heavily on using a third-party flight data API called ADSB-Exchange (ADS B-x) in order to determine the landing locations and flight occurrences of an aircraft. If the ADSB-x server were to malfunction, our software would be unable to retrieve flight data during this downtime and thus would not be able to correlate this with the upload data. This can be mitigated by writing a common generic interface in the server-side software which could be replaced easily without having to modify any code elsewhere in our codebase. For example, we can abstract the common operations of our database framework, such that no matter which actual database software we use, it will work the same on our server-side application.

5.1.2. Flight Data API Inaccuracy

Because our system must properly assign LON/LAT landing locations to a specific airport, our data must be accurate. Data received from ADSB-x however, is completely crowdsourced information which could skew the accuracy of particular locations. These potential inaccuracies could result in a misassignment of landing locations. A simple error-detection system that detects when there is an impossible change in position, during a flight, will prevent flights from having invalid landing locations.

5.1.3. Data Import Inconsistencies

One of our environmental constraints is that our import system for upload data must accept Excel documents. Because Excel does not have a fixed schema, it opens our software up to schema inconsistencies which could result in failed uploads or could corrupt our database. This could be mitigated by implementing a schema validation system which we check to make sure that the Excel document has the same structure as the previous Excel documents.

5.2. Challenges

CHALLENGE	DESCRIPTION	SEVERITY	LIKELIHOOD
MongoDB Speed	Indexes could be inefficient in storing our data	Medium	Low
Node.js Scalability	Some operations within Node.js are unable to scale	Medium	Low

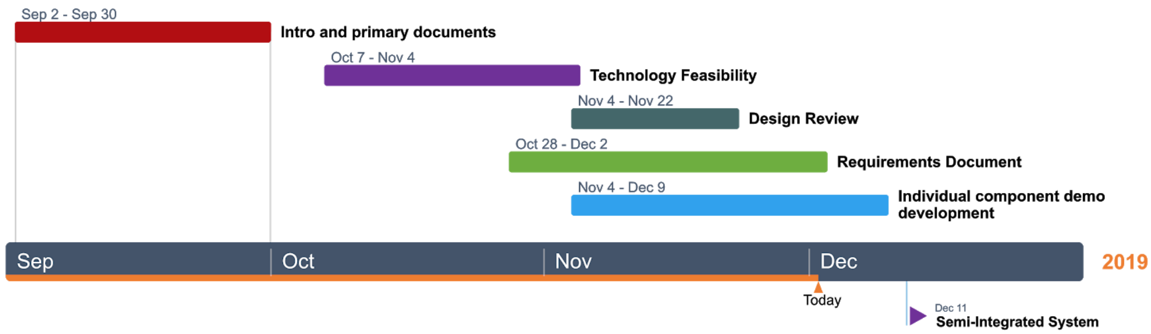
5.2.1. MongoDB Index Speed

Our software will be ingesting about 20GB of aggregate flight data per day from ADSB-x. As such, we need to ensure that our MongoDB queries are using indexes rather than searching through the entirety of the historical data. Since we will be ingesting a large amount of data at regular intervals, our indexes will grow too large to be efficient. This will result in slower database queries and limit the performance of the web interface and server-side software of our solution. To alleviate the severity of this issue, an index Time-To-Live (TTL) can be set such that the historical data still exists in the database, but it is no longer accessible.

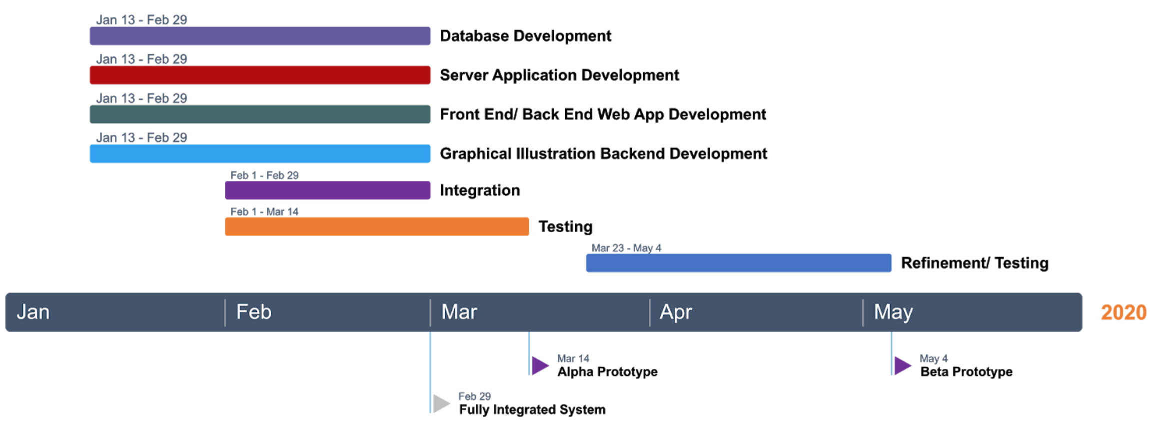
5.2.2. Node.js Scalability

One of our performance requirements states that our web application must be able to support 100 requests per minute. However, Node.js is by nature a single-threaded programming environment. This could result in the performance of our application plateauing at a certain level of requests. To resolve this, we can implement a forking web app framework, such as PM2, to allow our software to have multiple threads, at the expense of added complexity.

6. Project Plan



With only a few weeks left in the semester, there is only one more major milestone left, the semi-integrated demo. In this demo, each of our selected technologies will be showcased to prove their feasibility outlined in the technology feasibility document. This demo will demonstrate the integration between the database and the backend server applications while the other components will be shown independently executing simple tasks to confirm their ability to fulfil the requirements.



The spring semester begins the development of our project where we will start implementing all components. We have split up the project into four pieces, the database, the server application, the front and back-end web application, and the GUI application backend. The first milestone for the year will be on February 29th which is a fully integrated system. Testing will begin shortly after. This is where we will work out any major bugs and discover any flaws with our implementation. The next milestone is an Alpha prototype set for March 14th. This will be the first version of our product. The remaining of the semester includes refinements and further testing before our third milestone, a Beta release of the project. This will be the final product incorporating any and all refinements and fixed bugs from our testing phases.

7. Conclusion

In conclusion, the US Aerospace industry is highly profitable and supplies millions of jobs. Honeywell is the largest producer of gas turbine APUs and ranks 13 out of the top 50 aerospace companies. Honeywell's current engine data download process is tedious and results in a small data set and although Honeywell has upgraded to wireless uploading via their CEDAS system, it is limited by the adequacy of WiFi connection. Our solution helps predict when and where an upload should occur and helps predict why an upload failed. This document outlines our functional requirements as user stories from the perspectives of the engine technician and the airplane operator. We anticipate our project will have risks and a few challenges to overcome but we do not believe that they will pose a big problem to the overall progress of the project. As we approach the end of the semester, we are working towards a semi-integrated demo showcasing the database and backend server. The other components of the system will be demonstrated independently. Next semester, we look forward to integration and testing phases resulting in a beta prototype by May 2020. We are team EnginAir and we are confident that our project plan is on track to provide our client a software product that helps better the maintenance process to keep engines working properly.

8. References

1. Arbor, Ann. "Global Aerospace Industry Worth $838 Billion, According to AeroDynamic Advisory and Teal Group Corporation." Home - Teal Group. Teal Group Corporation, July 12, 2018. https://www.tealgroup.com/index.php/pages/press-releases/53-global-aerospace-industry-worth-838-billion-according-to-aerodynamic-advisory-and-teal-group-corporation.
2. Quora. "How Do Jet Airplanes Start Their Engines?" Forbes. Forbes Magazine, April 14, 2017. https://www.forbes.com/sites/quora/2017/04/14/how-do-jet-airplanes-start-their-engines/#11309fa9ab4b.
3. "Top Aerospace Companies: Top 50 Lists." AviationOutlook. AviationOutlook.com, October 10, 2019. https://aviationoutlook.com/top-aerospace-companies/#kcmenu.
4. "2019 STATE OF THE AMERICAN AEROSPACE AND DEFENSE INDUSTRY." AIA Aerospace Industries Association. www.AIA-AEROSPACE.rg . Accessed November 10, 2019. https://www.aia-aerospace.org/wp-content/uploads/2019/06/AIA-2019-Facts-and-Figures.pdf.