Software Design Document for Operations Data Analysis and Management System

Version 1.3

Prepared by Jesus Mendoza, Maximillion Orozco, Nathan Lee, and Richard Balbuena

Project Sponsor: Boeing

Created: November 1st, 2019
Table of Contents

Table of Contents 2
Revision History 4

1. Introduction 5
   1.1 Purpose 5
   1.2 Document Conventions 5
   1.3 Intended Audience and Reading Suggestions 5
   1.4 System Overview 5

2. Design Considerations 6
   2.1 Assumptions and Dependencies 6
   2.2 General Constraints 6
   2.3 Goals and Guidelines 7
   2.4 Development Methods 7

3. Architectural Strategies 8

4. System Architecture 8
   4.1 Data Storage Module 10
      4.1.1 Responsibilities 10
      4.1.2 Collaborations and Rationale 10
   4.2 User Interface Module 10
      4.2.1 Responsibilities 10
      4.2.2 Collaborations and Rationale 11
   4.3 Graphing Module 11
      4.3.1 Responsibilities 11
      4.3.2 Collaborations and Rationale 11
   4.4 Report Generator/Exporter Module 11
      4.4.1 Responsibilities 11
      4.4.2 Collaborations and Rationale 11

5. Policies and Tactics 12
   5.1 Choice of which specific products used 12
   5.2 Plans for ensuring requirements traceability 12
   5.3 Plans for testing the software 12

6. Detailed System Design 13
6.1 Front End (Module)
  6.1.1 Responsibilities
  6.1.2 Constraints
  6.1.3 Composition
  6.1.4 Uses/Interactions
  6.1.5 Resources
  6.1.6 Interface/Exports
6.2 Web server (Module)
  6.2.1 Responsibilities
  6.2.2 Constraints
  6.2.3 Composition
  6.2.4 Uses/Interactions
  6.2.5 Resources
  6.2.6 Interface/Exports

7. Detailed Lower level Component Design
  7.1 Success View
  7.2 DBEmail View
  7.3 Index View
  7.4 DBWriteFile
  7.5 File View
  7.6 Delete_file
  7.7 Build_response
  7.8 Components of Satellite
  7.9 Recent_measurements

8. Database Design

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

10. Requirements Validation and Verification

11. Glossary

12. References
# Revision History

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Reason For Changes</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>All members</td>
<td>11/1/19</td>
<td>Created initial draft</td>
<td>1.0</td>
</tr>
<tr>
<td>All members</td>
<td>11/15/19</td>
<td>Updated document for second draft</td>
<td>1.1</td>
</tr>
<tr>
<td>Richard Balbuena</td>
<td>11/21/19</td>
<td>Format document, ensure cohesiveness</td>
<td>1.2</td>
</tr>
<tr>
<td>Jesus Mendoza</td>
<td>11/26/19</td>
<td>More formatting, added table of contents</td>
<td>1.3</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Purpose

This document will specify the requirements for the first release of the project known as Operations Data Analysis and Management System, also referred through this document as ODAS. This document will cover a comprehensive overview of the complete system including its required modules and their intercorrelation. How the project works, how it should be interacted with and the purpose of each of the modules and their role in the project will be described and analyzed for the reader.

1.2 Document Conventions

For this document, every requirement statement described should be assumed to have its own priority. If applicable, italicized terms, words or phrases indicate an added emphasis of the terms, words or phrases in question.

1.3 Intended Audience and Reading Suggestions

For customers, this document is intended for ensuring each requested design requirement was implemented successfully.

For project managers, users, developers and testers, the document intends to allow the reader to become familiar with the functions and expected behavior of different modules, and for personal usage to understand the project structure.

The recommended sequence for reading this document is to begin with the overview sections first, then to proceed to the specific sections most pertinent to the reader’s specific area of interest.

1.4 System Overview

The software system is an interface that takes an open source spacecraft command and control system (COSMOS) and develop a data analytics system to analyze the operational data produced from the spacecraft command and control system. It has several modules; the database module stores data and bridges COSMOS to ODAS, the user-interface module facilitates user interaction, the report generator module produces reports from the ODAS system, the graph module graphs data in ODAS, and the machine learning module applies analytics approaches to the data. From the ODAS interface, the user will access all other modules through the user interface; all modules are directly connected to allow for intercommunication between relevant parts of the program in order to meet the requirements.
2. Design Considerations

2.1 Assumptions and Dependencies

Because the main way a user will utilize our software is via our web-interface, we can mostly limit the system demands and dependencies that the user will need. Since ODAS performs almost all of the file-processing, storage handling, and predictive analysis on remote servers, we only require that a user have access to modern browser and reliable internet connection.

Therefore, we plan that ODAS will be compatible with any operating system as with a modern browser and reliable internet connection. However, ODAS will be recommended for use on a computer with a large enough screen. So, we assume that the end user will either be on a Desktop/Laptop, rather than a mobile device, since scaling the webpage may disrupt the user experience on smaller devices.

2.2 General Constraints

As mentioned above, our team will mostly dedicate time to ensuring compatibility on larger displays, like normal desktop / laptop displays. Since we will not be focusing on developing ODAS for a mobile platform, we cannot ensure that the user experience will be consistent to those for which ODAS is intended for.

In terms of performance and compatibility when using ODAS, these will mostly depend on the individual use-case for each user. For instance, if a user requires a large file to be uploaded, they must ensure that their network connection is fast enough to speed up their upload time. Due to the fact that this may decrease user experience, ODAS may provide network connection specifications for optimal performance.

As of now, we have not begun stage two in our development of ODAS, where stage two consists of predictive analysis using machine learning. This is because, we have yet received data that accurately represents our target data; so, performing and training machine learning algorithms on false or hand-curated data would be catastrophically unproductive.

Subsequently, ODAS does not currently constrain file types or data structure when being uploaded for analysis. However, this will definitely change as we receive data and enter stage two of development.
2.3 Goals and Guidelines

One of the implicit principles our team follows during development is: “Form follows function”. Meaning that we much prefer that our features function as they are intended, and reliably so, rather than limiting our performance with the intention of improving our design.

While design and user experience are important, it is more important to simplify user interaction to improve accessibility to all of the well-functioning features ODAS will provide. With regard to deadlines and deliverable products, our liaisons instructed us to focus on having a deliverable product by the end of the academic year. That being said, it will be beneficial to the team to plan out and have a foundational understanding of how some aspects of our project may be accomplished, in case we do not have enough time to implement them.

2.4 Development Methods

For our development methods, we mainly used Agile Development. As part of Agile, we planned our a rough plan for the year of how we suspected to work on our project. That consisted of general epochs that were also broken down into shorter stories which tackled smaller subtasks of how each epoch would be completed. Afterwards, from the stories, we would create tasks which would guide us through completing our goals. We also had Demos to present our working accomplishments to the liaisons.

In order to improve our workflow, we set up a Task Board using Trello. Trello allowed us to keep track of the tasks we had pending, the tasks for each week, as well as the upcoming tasks. Once we started using Trello, we became more productive since we were able to create task Cards, and add descriptions and checklists for subtasks which helped us stay organized and on task. In addition, we would add deadlines to our tasks, so that when we would meet as a group, we would be able to expect our tasks to be completed, and if not, then we would always discuss any issues we may have had and allocate time or reorganize tasks as necessary.

Furthermore, we used Git and Github to work on our project as a team. For communication, we used Slack. Furthermore, to simplify our workflow, we created a channel on slack which was directly tied to our Trello board and Github project. This allowed notified us whenever tasks were completed, added, etc; and whenever our repository was updated. This was helpful in keeping us all up to date.
3. Architectural Strategies

After careful consideration and discussion with project leaders and advisors it was decided that the application would make use of a Django web server written in Python, MySQL database for storage management, and Javascript (mainly ReactJS and PlotlyJS) for the front end design. All technologies are highly scalable and easily maintained. Since these technologies are some of the most popular in the industry, there is a high availability of documentation.

Since the application will be used to gather and analyze hundreds of thousands of data points from each user, scalability was an important factor in deciding on which technologies to use. Django is able to spread the load of processes over multiple application servers. Due to its ability to handle load balancing well it was a top choice for our project.

MySQL has a structured data architecture which makes it a good choice for our particular use case.

Javascript is a popular choice for front end development due to its versatility and use of the Node package manager. Since Javascript is so widely used many powerful frameworks have been developed for it such as ReactJS and PlotlyJS which help speed up development significantly. The modularity of ReactJS allows the application to be far more easily expanded and maintained.

4. System Architecture

For the ODAS system, each module was created in order to address a different aspect of functionality as requested through the requirements. The ODAS modules were partitioned based on their relationship to using the received data or other relationships with other modules. As a user may only need to use a certain part of the program at a certain time, dividing the responsibilities of ODAS allows for more specific, dynamic and efficient component usage. Furthermore, modularization enables clearer understanding of components’ individual requirements, functions and architecture.
The system’s main responsibilities are to store operational data produced from COSMOS and the target data producers, provide a User Interface to perform data retrieval, graphing functions and report generation to the user. Along with the data retrieval and reporting capabilities, ODAS incorporates applications/tools to perform analytics on the data sets. The data may be structured(stored or real-time spacecraft telemetry, spacecraft command history) or unstructured(application logs).

The system also is responsible for having a data storage solution that can store the representative data sets, that ensures good clean data and be able to not lose data if the system fails and that is highly available, meaning that an instance of the data system can go down and and the user is still able to store data without interruption . A primary role of ODAS is to aid users in analyzing data; the system has partitioned its functions into various modules that follow below.

4.1 Data Storage Module

4.1.1 Responsibilities

The data storage applies elastic principles in the solution and leverages COTS(commercial off-the-shelf) to the largest extent possible. The module also demonstrates high availability principles in the data storage principles. The module also shows that it can scale to large data sets. The use of a database is employed to help with these various duties.

4.1.2 Collaborations and Rationale

The data storage module is utilized by all other modules. The creation of this module was necessary in order to isolate and properly store the data received from the user.

4.2 User Interface Module

4.2.1 Responsibilities

The user interface module is responsible for giving users the ability via a web-based interface to create a query for a list of one or more telemetry points over a user-specified time period(absolute time period), create a query for a list of one or more telemetry points over a relative time period(like recent 12 hours). The user interface module must also be able to create a query to search the unstructured data, and should have the ability to develop and save a query with a label for later use or as a source of data for plots or exports. Exporting data to a file in one of several formats(PDF, MS Office, JPEG, PNG, etc.) is also a responsibility of the module, as well as providing and displaying the graph utility. The user interface module also has the ability to connect the user to a function that monitors ODAS system performance. Lastly, the user
interface module is also given the task of allowing the user to perform ODAS system administration.

4.2.2 Collaborations and Rationale

The user interface module will interact with the Database Module, Graphing Module, Report Generator Module, and most other modules as well, since the user interface will provide a channel of access for the user to use the other modules’ functions. Having a separate user interface helps organize the program so the user may receive only what is desired without unnecessary access to other functions.

4.3 Graphing Module

4.3.1 Responsibilities

The graphing module is responsible for plotting data for one or several telemetry points through the plot format of a time plot display. The module allows the user to view data points through a tabular chart. In addition, the telemetry plots can be formatted; auto-scaled, zooming, and panning plot areas are allowed. The user is also allowed to use the pause function in the graph at any time.

4.3.2 Collaborations and Rationale

The graphing module will utilize the Database Module for its input and will display itself in conjunction with the User Interface Module. The complexity of graphing incoming data demands this component to be its own discrete module. This module will be heavily interacting with the User Interface Module.

4.4 Report Generator/Exporter Module

4.4.1 Responsibilities

The report generator/exporter module allows users to create reports from ODAS data based on specific queries created on the fly, or using saved queries. Reports can be executed one-time, or scheduled to be automatically executed on a recurring basis such as once per orbit, daily, weekly, monthly. Recurring reports would then be distributed via email or to a file server, or made available via a web-service as an on-demand capability with email notification and links.

4.4.2 Collaborations and Rationale

The report generator/exporter module will be collaborating with the Database Manager in the backend. As the report generator and export functions will be exclusively dealing with
system output to the user, it is beneficial to have this module delineated separately from the rest of the program.

5. Policies and Tactics

5.1 Choice of which specific products used

The choice of products vary greatly between team members. For the development of the front end of the application several things are being used such as WebStorm IDE by JetBrains and Notepad++ to write the code. The majority of the code is written in the ReactJS framework. Where graph visualization and consolidation is concerned, PlotlyJS is being used in tandem with ReactJS. In addition to those frameworks, CSS is being used to support where styling is needed. The backend of the project is being handled by Python, mainly the Python web server framework Django. MySQL is being used to manage the data that Django will be both serving and gathering. Python is also being used for the FTP server, email sender, and report generator. One of the most important technologies being used by the team as a whole is the version control technology Git and GitHub. GitHub hosts our code repositories and tracks the history of changes in our project. For task assignments and organization the team is using Trello.

5.2 Plans for ensuring requirements traceability

Requirements will be tracked through our choice of version control, GitHub. All advancements on the project code will be tracked on our online repository. All goals that have been set for the team including ongoing and completed goals are being tracked through a task board website called Trello.

5.3 Plans for testing the software

Testing on the application is done at every step of development. When integrating the application modules testing is done by using the modules as intended. In regular use the modules reveal bugs and other faults that can be remedied. As development continues we plan on doing rigorous testing of each of the modules. Methods of testing have not yet been decided beyond using the application as intended.
6. Detailed System Design

6.1 Front End (Module)

6.1.1 Responsibilities

The purpose of the front end module is to give the user an interface to interact with. The interface will essentially give the user a simple and intuitive way of sending queries and viewing the data that is stored on the backend servers of the application.

6.1.2 Constraints

The application will be hosted on a free tier of Amazon Web Services. Its storage and performance limitations will be according to the limitations of the free tier of an Ubuntu virtual machine on AWS. This module will be unable to do much of the calculations needed to handle the data that is being collected. Analysis and calculations will be restricted to the server module. This module is also unable to store data long term. Long term storage will also be dealt with by another module.

6.1.3 Composition

The front end will be composed of several pages which will each offer a different function. The main page will allow the user to sign up for update reports and the ability to save their reports and easily access data specific to them. Other pages allow the user to upload or query datasets related to their satellites and equipment.

6.1.4 Uses/Interactions

The user interface will consist of a main page where they shall be able to authenticate their identity with an email and password. Their login will be associated with satellite objects that they create. Their satellites will be associated with components and the components will be associated with the measurements they collect. Using a series of dynamically generated menus and checkboxes the user will be able to create high level queries which will be translated to SQL. The user will also be able to upload log files containing data they have collected on their own. From those log files they will be able to store information to a database and create queries.

6.1.5 Resources

This application will need a computer and an internet connection in order to develop and access it. The application needs to be hosted on a cloud service as well. The chosen cloud service is AWS. Development is done with several libraries for Python and Javascript including Django.
and ReactJS. Django is needed to manage connections to the interface and load balancing in the app. Memory and storage requirements for basic usage is low however it can rise significantly depending on the amount of data being dealt with.

6.1.6 Interface/Exports

This component provides a visualization of the application allowing the user to transverse it and take full advantage of its features. This module returns strings and views for the user to interact with. One main function of the UI is to display graphs that are generated in part by PlotlyJS with the backend serving the data that is to be reported.

6.2 Web server (Module)

6.2.1 Responsibilities

The server of the application will be responsible for processing data and requests it is receiving from the front end of the application. The user will be able to upload data through the interface which will then send the data to the server. The server will organize and store the data into a SQL database. When the front end requests information the server will serve the data as requested. The requested data will in most cases be sent to the front end where it will be displayed in tabular and chart forms by PlotlyJS. The server will also analyze that data upon the user’s request. The analysis will be done using machine learning to give the user an organized view of the data that has been collected.

6.2.2 Constraints

This module will not be able to give the user a visualization of the data that is collected and served up. It will be limited to gathering and analyzing data according to the user’s requests. Like the front end module, this module’s performance and storage capacities will be limited by the free tier AWS Ubuntu virtual machine. Having set up on a free tier the storage capacity is limited to 30GB of memory. The limits of performance and speed of processes are untested but will suffice for small tasks and will be readily available any time of the day.

6.2.3 Composition

The server is composed of a Django web server that creates the end points and makes them available for the front end to access. The server is accompanied by a MySQL database that allows the server to store data long term. The server accesses the data and formats it into a JSON response to be more easily read by the Javascript in the front end.
6.2.4 Uses/Interactions

The user will never directly interact with the server of the application. The front end interface will act as a tool for the user to access the server. The server analyzes and serves the data so that the front end can render and serve it to the front end module.

6.2.5 Resources

This module contains Django software. Much of this software is provided in the framework already. All that is needed to access the server is a terminal window. The Django endpoints can be accessed through any browser on a computer. The Django server will access a MySQL database that has a supported client that is easily installed in dependencies.

6.2.6 Interface/Exports

The interface on this module will only be accessible by developers. The endpoints have a UI integrated by Django that allows the developers to see the available data that is being stored in a JSON format. Developers may need a JSON formatter to read the raw data more easily. Formatters often come in the form of a browser extension. The interface in this module is mainly for reference and ease of development.
7. Detailed Lower level Component Design

7.1 Success View

7.1.1 Type: Django View
7.1.2 Simply returns a success message that is tied with the index view
7.1.3 No interface

7.2 DBEmail View

7.2.1 Type: Django View
7.2.2 Just a testing view to display the results from the satellite table from the database
7.2.3 Simple html text interface

7.3 Index View

7.3.1 Type: Django view
7.3.2 Uses Django form to gather email address to send to and a subject and delivers data from a database
7.3.3 Two form fields available: user email address and a subject field

7.4 DBWriteFile

7.4.1 Type: Django View
7.4.2 Writes a file to a specific directory to simulate sending a file to a file server.
7.4.3 No interface
7.4.4 When the view is executed, for now it just displays a success message and the file is created within the media folder.
7.4.5 Program has no input for now but work is in progress to allow user to select what database query they want to send to file server as a .txt file. Other file format options are also being explored

7.5 File View

7.5.1 Type: Django View
7.5.2 Renders data to an html page that shows the complete file list of all the files within the folder ‘media/files/uploads’. This view works, together with the other views:
7.5.3 The interface is an html page to display the file results in a table.
7.5.4 This view is currently being used to test that the files are being written here correctly and the delete function works to simulate a file system online, currently works within a local folder

7.6 Delete_file

7.6.1 Type: Django View
7.6.2 Deletes a file from the file list that is displayed by file_view
7.6.3 Interface is a button that the user can press to delete the file

7.7 Build_response

7.7.1 Type: Django View
7.7.2 Creates a data response that is used by other views to display using JSONResponse
7.7.3 No interface
7.7.4 Takes in a query set in the form of a list as a parameter. This view then splits the query into a readable format into a variable, data, which contains things such as measurements, satellites, and various fields for the components of this satellites

7.8 Components of Satellite

7.8.1 Type: Django View
7.8.2 Retrieves table sets from Django Model of Components table
7.8.3 No interface, just a json response for now
7.8.4 Takes in the satellite ID as a parameter to the field and uses that ID to retrieve the components from the database and returns a json response. Needs to interact with build_response view in order to get the data in a printable format

7.9 Recent_measurements

7.9.1 Type: Django View
7.9.2 Retrieves table sets from Django Model of Measurements table
7.9.3 No interface, just a json response for now
7.9.4 Takes in the satellite ID as a parameter to the field and uses that ID to retrieve the measurements from the database and returns a json response. Also uses a parameter from the URL to identify the quantity of the measurements that the user wants to display in the json response. They are ordered by the time measured
8. Database Design

The database for our project proved to be an essential part of our design. The difficulty in this regard, arose because we were still unclear about what kind of data and how the data would be provided. We were not sure whether it would in a specific file format, or how that data would be stored in those files. With that in mind, we wanted to design a database that was flexible and scalable, since we would need to accommodate for potentially millions of telemetry data points. Furthermore, we continue to iterate our design as more details are provided about the data we’ll be receiving. Below, is a depiction of our current database schema, followed by descriptions of the tables and columns along with our reasoning for this design.

Tables and columns:

Satellites:

- **id** - A unique identifier acting as a primary key, used to refer to a specific satellite
name  - The name of the particular satellite (this may not be provided in the final result, since this information may be deemed classified)

Components:

id  - A unique identifier to refer to a specific component

satellite id  - The id for the specific satellite that this component pertains to

name  - The name of the component referring the component (and in some cases, a way to refer to one of multiple of the same component)

model  - The model number or model name for this component
category  - A categorical label that refers to a component’s function or intended task
description  - A brief description of the role this component contributes to the satellite

Units:

id  - A unique identifier to refer to a specific type of units

units  - A string that is commonly used to refer to these type of units

Measurements:

id  - A unique identifier to refer to a specific measurement

satellite id  - The id of the satellite that this measurement was measured on

component id  - The id of the component that this measurement was measured from

units id  - The id of the units for this type of measurement
time stamp  - The datetime stamp that this measurement was measured on the satellite

value  - The actual value this measurement recorded

We decided on this schema because it provided us with enough flexibility to accommodate for most, if not all satellites, as well as making it easy to add specifications about a satellite. For instance, our database design allows us to describe all of the components of a satellite may have. By using a satellite's id as a foreign key, we can determine exactly which components a satellite contains.

Furthermore, our database makes it simple to modify any specifications of the satellite, making it easy for the user to make any revisions. For example, if a component on a satellite fails, and the user decides that it is no longer necessary to keep track of that component, they can easily remove it. Or on the other hand, if a user needs to specify a component that wasn’t initially detected, they can easily add it, regardless of the function of that component.

This schema allows us to generalize how each satellite is represented. Also, another key aspect that we need to focus on was time-based data points. Since we would be heavily dependent on
when measurements occurred and predicting future potential behaviours of a satellite and/or its components, it was vital for us to highlight that time and date were features that would allow us to accurately trace a measurement to. So by making a single table to store the measurements, it would be easy to filter by time, and query for the most recent and ranges of time. As opposed to having individual tables to track the time that a measurement was taken for each component.

9. User Interface

9.1 Overview of User Interface

The interface was designed with simplicity in mind so that the application will have a flow that the user will be able to follow with ease. The screen must be kept simple so that it does not become too busy and overwhelm the user. The user is given a small limit of actions to perform per page so that they are able to choose without much difficulty. A page that acts as a dashboard for the user will provide a point for them to return to and begin the process from the beginning. Overall the user interface is a tool for the user to be able to easily access the application’s features.

9.2 Screen Frameworks or Images

![Operations Data and Management System](image-url)
9.3 User Interface Flow Model
## 10. Requirements Validation and Verification

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Where is it implemented?</th>
<th>How was this requirement verified?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1 Data Storage Solution</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>4.1.1.1 The application shall have a database that is capable of scaling to large data sets</td>
<td>Django Model</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>4.1.2 API for External Applications</td>
<td>build_by_response View</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>4.1.2.1 The application should allow the capability of ingesting, retrieving and generating reports using an API</td>
<td>Email and DBWrite View</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>4.1.2.2 The application should be able to export data into common ASCII formats including CSV, Space Separated, Tab-Delimited, JSON, and XML</td>
<td>n/a to date</td>
<td></td>
</tr>
<tr>
<td>4.1.3 Web Based User Interface</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>4.1.3.1 The application shall have a web based interface</td>
<td>UI package</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>4.1.3.1.1 The application should be able to create a query for a list of one or more telemetry points over a user specified time period</td>
<td>Ui Package</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>4.1.3.1.2 The application should be able to create a query for a list of one or more telemetry points over a relative time period</td>
<td>Ui Package</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>4.1.3.1.3 The application should be able to create a query to search unstructured data</td>
<td>Ui Package</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>4.1.3.1.4 The application should be able to develop and save a query with a label for later use or as a source of data for plots or exports</td>
<td>Ui Package</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>4.1.3.1.5 The application should be able export data to a file in one or more available formats</td>
<td>Ui Package</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>4.1.3.1.6 The application should be able to plot data for one or more several telemetry points in one of several plot formats</td>
<td>Ui Package</td>
<td>Manual Testing</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Implementation</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>4.1.3.1.7</td>
<td>The application should be able to format telemetry plots</td>
<td>Graphing Package</td>
</tr>
<tr>
<td>4.1.3.1.8</td>
<td>The application should be able to Monitor ODAS system performance</td>
<td>n/a to date</td>
</tr>
<tr>
<td>4.1.3.1.9</td>
<td>The application should be able to perform ODAS system administration</td>
<td>n/a to date</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Report Generator</td>
<td>--</td>
</tr>
<tr>
<td>4.1.4.1</td>
<td>The application shall have a report generating function to create reports from ODAS data based on specific or saved queries</td>
<td>Email and DBWrite View</td>
</tr>
<tr>
<td>4.1.4.2</td>
<td>The application shall be able to send reports using email as a one time execution or a recurring basis such as daily, weekly or monthly</td>
<td>index View</td>
</tr>
<tr>
<td>4.1.4.3</td>
<td>The application shall be able to send reports to a file server as well</td>
<td>fileView and fileUploader View</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Data Search</td>
<td>--</td>
</tr>
<tr>
<td>4.1.5.1</td>
<td>The application shall have a query interface that allows for easy searches on data and displaying the results as well as filter queries</td>
<td>n/a to date</td>
</tr>
<tr>
<td>4.1.6 Data Analytics</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>4.1.5.1 The application shall have data analytics tool suites that allow the user to exploit the data using the ODAS system to make predictive fault predictions or predictive fault data discovery</td>
<td>n/a to date</td>
<td></td>
</tr>
</tbody>
</table>
11. Glossary

Agile Development

Software development approach where requirements and solutions evolve through self-organization and collaboration within teams and with customers, clients and end users.

Analytics

Data analytics or analysis referring to data inspection, analysis, and transformation for extracting more effective, useful information and improving conclusions. Used in data science.

AWS (Amazon Web Services)

Cloud-computing and hosting services platform from Amazon used for web hosting.

COSMOS

Ball Aerospace User Interface for Command and Control of Embedded Systems. The documentation may be found at cosmosrb.com.

COTS (Commercial Off-The-Shelf)

Type of packaged software solution product dynamically adapted to the customers’ potentially changing needs and requirements.

Endpoints

Unique URL representing a JSON Object or collection of JSON objects.

Foreign Key

In SQL, a database column linking data in two separate tables.

Front End

The portion of the software application the user directly interacts with.
GB Gigabytes

Unit of information.

Interface

Program allowing a user or one component of a system to communicate with a separate component of a system exchange.

JSON

JavaScript Object Notation, a common open-standard file format used to transmit data objects as human-readable attribute-value pairs and arrays.

Machine Learning

Study of statistical models and algorithms used by software to perform a task effectively without explicit instructions from a human. Prediction, patterns, and inference are utilized in the application of machine learning.

Module

Independent part of a program used to construct a more complex software application.

ODAS Operations Data Analysis and Management System

The name of the software program this document refers to.

Open Source

Software where original source code is free, redistributable and accessible.

Operating System

Supporting software for basic functions, like executing applications.

Predictive Analysis

Advanced analytics using previous and new data with machine learning and prediction in order to forecast activity, behavior and trends and draw conclusions.
Repository

The specific central location in Github where data is stored and managed.

Server

Program managing access to a centralized network service or resource.

Telemetry

(A spacecraft’s) instrument readings’ records and transmissions.

User Interface (UI)

The interface where a user and software system interact, specifically with input software.

Virtual Machine

The emulation of a computer system within a computer.

12. References

Documentation for technologies used:

Django: <https://docs.djangoproject.com/en/2.2/>

React.js: <https://reactjs.org/docs/getting-started.html>

Plotly.js: <https://plot.ly/javascript/>