Final Report

For the Project of:

Web Tool for Testing Environmental Conditions and Procedures for Airborne Equipment

Set forth by:

ZeroAvia, Inc.

Dr. Youcef Abdelli, youcef@zeroavia.com

Bryce Aberg, bryce.aberg@zeroavia.com

Prepared by:

University of Washington Capstone Team

EE 497 A Wi 22: Engineering Entrepreneurial Capstone

With Academic Advisor:

Baosen Zhang

With members:

Eric Singer, esinger1@uw.edu

Steven Zhou-Wright, szwright@uw.edu

Sophie Tao, ztao2@uw.edu

Casey Nguyen, caseyn13@uw.edu

Somnath Mukherjee, som2021@uw.edu

TA:

Harsha Vardhan

Date: 06/08/2022

Contents

Introduction and Abstract

Introduction

ZeroAvia is an aerospace company dedicated to designing the world's first hydrogen fuel cell-powered passenger flight. The hydrogen-electric aircraft needs to meet all existing safety standards currently used by the aviation industry. One of these standards put out by the RTCA (previously known as Radio Technical Commission for Aeronautics) for testing of environmental conditions and procedures is titled RTCA DO-160G. DO-160G contains 16 different sections for testing. The engineer will use this document to guide their physical testing. This document media format is a 600 page book that the test engineer will have to decipher the correct equipment and their corresponding test procedures. The aim of the project is to aid the engineer in quickly being able to identify the correct test based on what equipment they are testing, and give the testing graphs that will tell the engineer the correct values to use for testing. While creating solutions for every section of the DO-160G is derivable. The current project focus will be on two sections. Namely the tests that involve temperature control and/or humidity control. This corresponds with Section 4 and Section 6.

Abstract

The project, titled, “Web Tool for Testing Environmental Conditions and Procedures for Airborne Equipment” seeks to facilitate the meeting of those standards via a web application. The objective is to create a web application that will act as the foundation for taking this DO-160G from a hardback copy to an interactive application. The web application will focus on two sections that have results that can be visualized. Namely temperature and humidity. The project will utilize a front-end web framework that will communicate with a back-end framework and source files. The bulk of the computation will focus on a modular approach for the source files.

Team Roles and Responsibilities

- **Eric Singer**
  Role Title: Team Leader, Model and Optimization Developer, Source code developer
  Contributions: Development of object-oriented input approach and python scripts to store information, management and scheduling of all meetings and general organizational tasks. Set foundation for object oriented approach to Section 4, completed the curve-generating component of Section 6, humidity. Stored all the relevant data in the backend.
- **Somnath Mukherjee**
  Role Title: Model and Optimization Developer
  Contributions: Development of models regarding electrical components, specifically motors, inverters, and power distribution. Development of functional/objective scripting for Section 4, temperature. Created a list of all the categories and tests related to the sections for future reference. Chiefly designed and revised poster and provided inputs for the HMI design.

- **Steven Zhou-Wright**
  Role Title: Model and Optimization Developer
  Contributions: Research into first principle equations and modeling for various powertrain components, script development of hybrid functional/objective approach, finding requested documents. Designed the backend's object oriented and function oriented components to accept the user input and generate the appropriate plot or curve for Section 4, temperature.
- **Sophie Tao**
  Role Title: Full-Stack Application Developer
  Contributions: Design and decision making about data management. Implemented an output webpage and created APIs for input data processing and image transmission between backend and frontend, passing inputs and outputs between them.
- **Casey Nguyen**
  Role Title: Front End Developer and GUI Designer
  Contributions: Graphical design and organization scheme for the HMI, HTML/CSS coding for web applications. Designed the frontend for displaying and storing the test curves generated by the backend. Created an ergonomic flow to select categories and tests. Implemented the HMI design on the webpage.

Work Breakdown:

1. Organization
    a. Meeting scheduling, assignment submission, and correspondence. (Eric)
2. Backend Development
    a. Code Approach/Framework. (Steven, Eric, Somnath)
    b. Data Loading into Backend. (Eric, Somnath)
    c. Development of display scripts for each section
        i. Section 4, Temperature. (Steven, Somnath)
        ii. Section 6, Humidity. (Eric)
3. Frontend Development
    a. Flask App deployment. (Sophie, Casey)
    b. Backend Script Implementation. (Sophie)
    c. User Interface Design and Implementation to the Webpage. (Casey, Sophie)
4. Presentation
    a. Poster Design and Printing. (Eric, Somnath, Casey, Steven, Sophie)
    b. Final Video. (Eric, Somnath, Casey, Steven, Sophie)

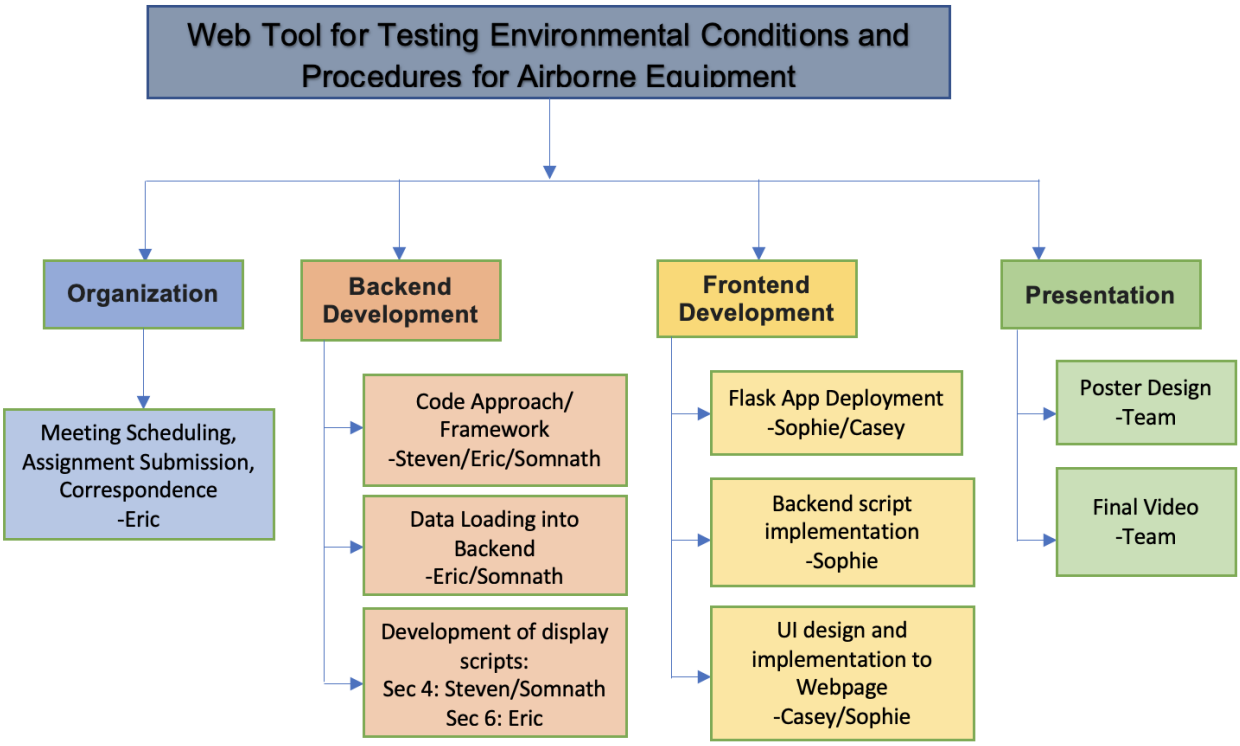

Figure 1: Work Breakdown Structure for the project

Peer Mentor: Harsha Vardhan
Industry Mentor: Bryce Aberg, ZeroAvia
Faculty Mentor: Professor Baosen Zhang, UW Dept. of Electrical & Computer Engineering

Project Schedule

- **Timeline:**
  The timeline for the project is made and recorded using Gantt Chart, as shown in the following figures:

Eric Singer, esinger1@uw.edu Steven Zhou-wright, szwright@uw.edu Sophie Tao, ztao2@uw.edu Casey Nguyen, caseyn13@uw.edu Sommath Mukherjee, som2021@uw.edu	Project: ZeroAvia	Gannt Chart														SP
Key:	Allocated Time	Task Complete														
	Allocated Time/Contrast	Task on Time														
	Buffer Time	Task Behind														
				Jan 3	Jan 10	Jan 17	Jan 24	Jan 31	Feb 7	Feb 14	Feb 21	Feb 28	Mar 7	Mar 14	Mar 21	
Tasks	Descriptions	Completion level	Persons Involved	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	
Winter Quater																
School Assignment Deliverables																
Biweekly report 1	UW Assignment 1	100%	Full Team				DUE: JAN 30									
Sign and Submit NDA	UW Assignment 2	100%	Full Team		DUE:JAN 16											
Project Sketch	UW Assignment 3	100%	Full Team		DUE:JAN 16											
Team Profile and Charter	UW Assignment 4	100%	Full Team	DUE: JAN 9												
Project Plan	UW Assignment 5	100%	Full Team					DUE: FEB 6								
Peer Review 1	UW Assignment 6	0%	Full Team						DUE: FEB 13							
Biweekly Report 2	UW Assignment 7	75%	Full Team						DUE: FEB 13							
Biweekly Report 3	UW Assignment 8	0%	Full Team								DUE: FEB 27					
Peer Review 2	UW Assignment 9	0%	Full Team										DUE: MARCH 13			
Biweekly Report 4	UW Assignment 10	0%	Full Team										DUE: MARCH 13			
Design Report	UW Assignment 11	0%	Full Team											DUE MAR 16		
Technical Tasks (Winter)																
General																
Prototype	Software pack with inputs/outputs with "Model 1"		Full Team										DUE: MAR 12-18?			
Specific to UI Design	A design for user interface		Casey/Sophie													
Show guideline for the user to input		90%	Casey							Due Feb 18						
Provide selections for the output form		90%	Casey							Due Feb 18						
Specific to Front-end	A user-friendly interface based on Web,		Sophie/Casey													
Show guidelines for the user to input		100%	Casey							Due Feb 18						
Show the output forms		50%	Casey							Due Feb 18						
Store the user input in a database.		0%	Sophie								Due Feb 25					
Display the parameters analysis from backend.		0%	Sophie									Due Mar04				
Specific to Back-end	A database design based on PostgreSQL and Flask		Casey/Sophie													
Design tables for the databsase		25%	Casey/Sophie								Due Feb 25					
Provide the API to the database and frontend		25%	Sophie								Due Feb 25					
Test the API according the user inputs		0%	Sophie									Due Mar 04				
Build a server connection based on Flask.		80%	Sophie							Due Feb 18						
Query data from database and test on Backend		0%	Sophie									Due Mar 04				
Data processing on Backend based on tests		0%	Casey/Sophie										Due Mar 11			
Data processing on Backend based on models		0%	Casey/Sophie											Due Mar 16		
Specific to Modeling																
Model 1 Block Diagram		50%	Som/Steven/Eric							Due Feb 18						
Input for Model 1 Design Outline	Mission Flight Profile	25%	Steven/Som/Eric							Due Feb 18						
Major component outline	Propulsion, Inverter, Fuel Cell, Fuel Storage,PDU	25%	Steven/Som/Eric							Due Feb 18						
Subcomponent Mathmatical Model Outline	Efficiencies of power/energy transfer between components	0%	Steven/Eric/Som								Due Feb 25					
Model 1 Development	Completion of First Modeling Task	0%	Steven/Eric/Som								Due Feb 25					
Specific to MDO																
Paramertization of Inputs	(from Model/external resources)	0%	Eric/Steven									Due Mar 4				
Block Diagram	(from Model/external resources)	0%	Eric/Steven									Due Mar 4				
List of Optmization Options	MDO to be done in the spring	0%	Eric/Steven									Due Mar 4				
MDO base line	first one based on efficiency	0%	Eric/Steven									Due Mar 4				
Specific to Software Intergration																
Software Architecture	Block Diagarm of software level interfaces.	0%	Team													
Integrating Frontend/Backend		0%	Team											Due Mar 8-12?		
Integrating Backend/Database		0%	Team											Due Mar 8-12?		
Integrating Model 1		0%	Team											Due Mar 8-12?		
Integrating Inputs		0%	Team											Due Mar 8-12?		
Integrating Introductory MDO base line		0%	Team											Due Mar 8-12?		
Integrating Graphics		0%	Team											Due Mar 8-12?		
Integrating Information Saving		0%	Casey/Sophie											Due Mar 8-12?		

Figure 2: Winter Quarter Gantt Chart

Eric Singer, esinger1@uw.edu Steven Zhou-wright, szwright@uw.edu Sophie Tao, ztao2@uw.edu Casey Nguyen, caseyn13@uw.edu Somnath Mukherjee, som2021@uw.edu	Project: ZeroAvia			SP	Gannt Chart											END
Key:	Allocated Time	Task Complete	Task Removed													
	Allocated Time/Contrast	Task on Time														
	Buffer Time	Task Behind														
					Mar 28	April 4	Apr 11	Apr 18	Apr 25	May 2	May 9	May 16	May 23	May 30	June 6	
Tasks	Descriptions	Completion level	Persons Involved		Week 13	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	Week 20	Week 21	Week 22	Week 23	
Spring Quarter																
School Assignment Deliverables																
Biweekly report 5	UW Assignment 13	100%	Full Team				Due Apr 15									
Biweekly report 6	UW Assignment 14	100%	Full Team						Due Apr 29							
Peer Review 3	UW Assignment 15	100%	Full Team						Due Apr 29							
Standards and Ethics	UW Assignment 16	100%	Full Team							Due May 6						
Biweekly report 7	UW Assignment 17	100%	Full Team								Due May 13					
Biweekly report 8	UW Assignment 18	100%	Full Team										Due May 27			
Peer Review 4: Overall Evaluation	UW Assignment 19	100%	Full Team										Due May 27			
Final Poster	UW Assignment 20	100%	Full Team											Due June 1		
Final Demo Video	UW Assignment 21	100%	Full Team											Due June 1		
Final Report	UW Assignment 22	100%	Full Team												Due June 8	
Self Evaluation and Reflection	UW Assignment 23	0%	Full Team												Due June 10	
Technical Tasks (Spring)																
Implementation of Model 2		0%	Steven/Som/Eric						Due Apr 29							
Implementation of Model 3		0%	Steven/Som/Eric								Due May 13					
MDO Block Diagram		0%	Eric/Som/Steven										Due May 27			
MDO Modeling		0%	Eric/Som/Steven										Due May 27			
MDO Modeling Expansion		0%	Eric/Som/Steven										Due May 27			
DO-160G Section 4 intergrated into working model		100%	Steven/Eric/Som				Due Apr 15									
DO-160G Section 6 implementation		100%	Steven/Eric/Som						Due Apr 29							
DO-160G other sections integration		0%	Steven/Eric/Som									Due May 20				
HMI/GUI Refinement		100%	Sophie/Casey										Due May 27			
Project Hand-off and training		100%	Team												Due June 10	

Figure 3: Spring Quarter Gantt Chart

- At the initial stage, our project was based on modeling the entire powertrain of an aircraft and then to do a multi-disciplinary optimization. During our discussion with the assigned faculty member we weighed the complexity and effort of the requirements and then decided to just focus on any one of those powertrain components. After pitching this idea to our industry mentor we pivoted to a different project plan. The new project was to design a web tool which will resemble the DO-160G and provide an user-friendly interface. The work done so far helped us in preparing the backbone of our revised project. We already had a model 1 ready with the lowest level of complexity by mid winter quarter. From mid winter quarter the whole team started working on the new frontend and backend aspects of the revised project. We targeted the sections that have some graphical interpretation of the tests and finally managed to provide the deliverables for section 4 and section 6.

- The milestones regarding submissions and tasks are shown in above figures.
  The milestones that triggered our pivot are as tabulated below:

Meetup with Prof. Baosen to discuss project scope	4-Feb
Discussion regarding pivot with industry mentor	11-Feb
Start of revised project	14-Feb

Table1

Project Resources and Budget

Our project required minimal budgeting and purchasing concerns. The only expense was the poster printing cost.

Realistic Constraints

Owing to constraints on the possibility and feasibility of our original multi-disciplinary optimization project we pivoted instead to the development of a testing verification system for DO-160G guidelines. Our faculty mentor has expressed some concerns about our ability to complete this project (it contains 23 sections spread over 600 pages of testing guidelines) but even completion of a few of the most salient and difficult to verify sections will be a worthwhile endeavor. However, even delivering all the sections and the test verification component of the software proved to be too much in our remaining project time.

System Requirements

a. **Functional requirements**
   - **i.** The software shall display the necessary test curve based on the user selected test category, equipment category, and specific test chosen.
   - **ii.** The software shall accept user inputs to generate curves, if necessary. The software shall not request user inputs unless they are needed to generate the curve
   - **iii.** The software shall accept a curve generated by running the tests prescribed in the DO-160G and compare this curve to the one prescribed by the document.
   - **iv.** The software shall determine whether inputted test data successfully emulates the test prescribed by the DO-160G.
   - **v.** The software HMI shall provide the user an option to output data to a Matlab or excel file.
   - **vi.** The software shall store input and output data.
   - **vii.** The software HMI shall provide access to the database on which input and output data is stored.

b. **Performance requirements**
   - **i.** The software shall run with the following minimum system requirements:
     1. 4-core i5 Processor @ 2.6 GHz
     2. 16 GB RAM
     3. Windows 64-bit

c. **Interface requirements**
   - **i.** External interface should receive user input requests and display relative model analysis and graphs. It should include ways for the end-user to find any/all outputs of the backend executions as well.
   - **ii.** Export system performance parameters from the backend as an image file.

d. Software system attributes

i. Reliability

1. Notifications of software failure (unaccepted inputs ,unbound solutions, etc) shall be displayed for the end user through the HMI.

ii. Availability

1. It is a software package which could be displayed as a web app in the user's local device.

2. Guide the user to input information and provide the ability for the input data processing and storage in the database.

3. Output the data analysis and graphs based on the model from the back-end.

4. Let the user decide whether to export the system performance parameters from the backend as a file in Excel and MATLAB format.

iii. Portability

1. This will be a locally hosted web application, requiring a web browser.

Operating Hazards and Requirements

a) Precision and Accuracy Requirements

i) Within each section, each test shall adhere to its specific test parameters accuracy requirements. Such as in Section 6: Humidity, temperature rise and fall shall be within 2% accuracy, as specified.

b) User interface requirements

i) The user interface shall provide instructions on use.

ii) The user interface shall provide drop down menus for selections.

iii) The user interface shall provide informative keywords when manual entry of data is required.

iv) The user interface shall have a print button to print relevant data outputs.

v) The user interface shall be able to return back to the main selection menu for repeat or alternative tests to run.

c) Test and Validation Requirements

i) The Program shall undergo prototype testing where each software generated plot is manually validated against a hand written copy. This validation shall be cross referenced by multiple different people at different times.

ii) There shall be UI usability testing with people unfamiliar with the software, only told basic information one would receive if determining test data by hand via the DO-160G handbook.

iii) The software shall undergo unit testing for all functions that require inputs.

iv) The software shall be tested again over minimum and over maximum values to mitigate inappropriate input values.

d) Electromagnetic Compatibility requirements

i) The program shall interface with commercially available browsers that are pre required to be compatible with immunity testing and emissions testing on the specified hardware.

e) Safety Requirements

i) The program shall not be able to output values that will damage the testing machinery that is eventually to be used. Such as outputting a temperature requirement of 500 degrees fahrenheit.

ii) The software program shall not communicate with outside applications that are not specified within the program.

f) Standard Requirements

i) The software shall adhere to RTCA DO-160G standards.

ii) The software app shall adhere to HTML standards.

iii) The software application shall adhere to HTTP standards.

System Specifications

The final design resulted in a web application that was able to meet basic functionality for displaying testing results for temperature and humidity sections. The temperature section includes all 8 testings and the humidity section can display the three distinct sections. The HMI was able to be improved by being able to use drop down menus that are visible in an easy to read manner. While when the drop down menus require manual inputs, they pop up instead of always being there. It is important to note that the original system requirements were meant for a previous project and does not include the pivot as described later in this paper. When the pivot was introduced mid-way through the project the original goal was to be able to simulate all sections of the DO-160G, not just temperature and humidity. This requirement set forth by the company was an ambitious goal given the time allotment. The project was able to complete two of these sections. There was an additional requirement that the program was able to validate input data. This was considered a stretch goal that was not obtained.

We can isolate the divergence from these company setpoints, mainly due to the time required. The full document of the DO-160G compromised around 600 pages of information. The ability of turning this into an interactive web application would be beyond the scope of work possible for a team that is subjected to less time due to the project pivot. The validation stage of the project was not able to be completed due not having the data to be able to input. There was some early work done into validation, but this was deprioritized as it was not seen as valuable as doing the test setup. Which the majority of the test revolve around.

Design Procedure/Methods

The RTCA DO-160G document has 23 sections (labeled 4.0 to 26.0). These sections will have all their unique inputs and testing requirements laid out via DO-160G. All the tests will conform to the same style of input data via a .csv file and be able to run binary PASS/FAIL if the required information is supplied. As an example we have the block diagram of the first section labeled: Section 4.0 Temperature and Altitude. From this block diagram that reads clockwise starting at “Input Data” we must select the temperature and Altitude test, select the correct equipment category, and the specific subtest they want to run. Our code must then use the reference data tables or accept user input (if the DO-160G doesn’t specify the necessary information to run the test) to create the curve and show the engineer how to run the test.

With our task at hand, we had to select the tools we were going to use. Early on, Python was an obvious choice for the backend coding language. Its flexibility and ease of use in comparison to other languages made this choice quick. There was some concern that the optimization tasks required later in the project may tax the language. However, given the industry sponsor’s expectation that this program would run on middle-of-the-road computers led us to conclude that any computing tasks would not be too immense. How we intended to present and store the information was a slightly more complex process. Initially, we wanted to move towards a remote-hosted website using Azure or AWS, but the industry mentor pushed back on this, citing the possible unreliability and potential maintenance concerns for a remotely-hosted database. This led us to pursue instead software development packages and the creation of an executable tool. However, two salient concerns arose. First, this would require the purchase of a new software development IDE, whereas previously we had been using free platforms to write up scripts. The purchases could be made through the school, but this would take time and require conversion of our working scripts to the new platform - altogether a minor setback. The more worrying issue was the overall lack of software development experience amongst our team members. Mostly each team member was familiar with the element of their project (scripting, database development, GUI development) but not at all familiar with how to integrate each of these in a software package. Using something like PyCharm, for example, would require every team member to learn the new platform, which could slow us down development. In the end, we decided our program would be run on a locally-hosted Flask webpage. This allowed us to best leverage our existing skills, while still meeting the industry requirements.

For the database, we chose PostgreSQL, again for team familiarity and compatibility with Python. However, as time went on and we became more familiar with the project, we realized that a database is unnecessary because the number of inputs are limited based on different user’s selections and also the amount of data can easily be stored in .csv files and called in with the backend.

Software Implementation

Description

This project aims to design and implement a Web App for testing environmental conditions and procedures for airborne equipment. The software tools used in this project are as the following figure shows.

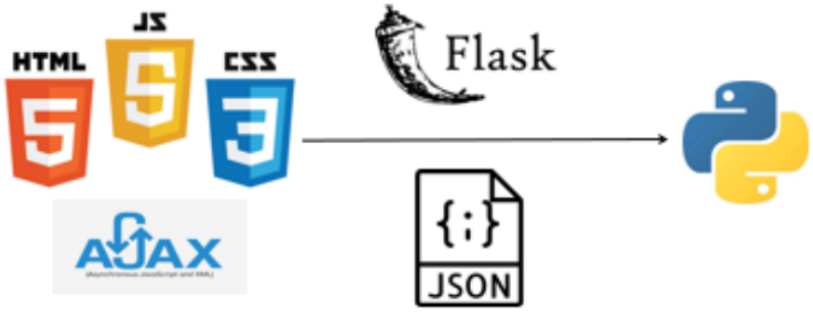

Figure 4: Application languages and Tools

It's a full-stack web development. The web main has two pages, ons is the input page which could send user's inputs and selections to the backend and another is the output page where can display and download the test result, and return back to the test part (input page). For the frontend, the programming languages like HTML, CSS, AND JavaScript are used and a framework Bootstrap is applied for the purpose of building a responsive, and dynamic webpage. For the backend, Python as the main language for programming and the micro-framework Flask was applied to process routing, and request handling.

Implementation

Frontend-Input Page

There are three parts for the input page. All the decorations about the web pages are implemented by CSS and Bootstrap Framework.

- **Guideline**
  It is set as a user navigation which includes an introduction for the project, input rules, submission rules and the refresh function. This part is one of the static files of a HTML file in the backend.

- **Users' Selection and Input:**
  This part would produce more dropdown options as required while navigating the "tree of inputs" based on user's selections. This dynamically content changing is achieved by the combination of JavaScript and HTML.

- **Refresh Button:**
  This button could reload the current page in order for user's to change selections.

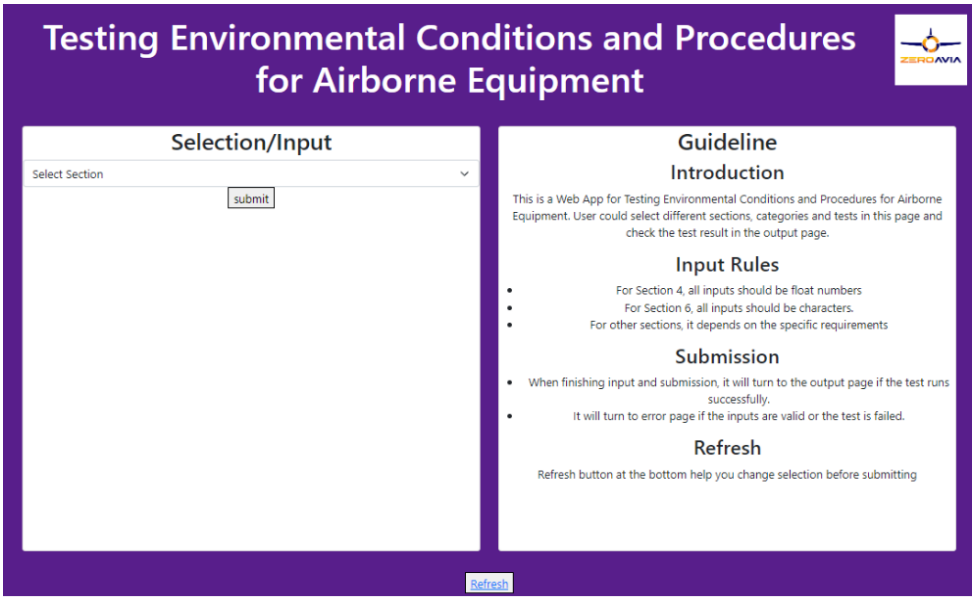

Figure 5: Input Page for users' selection and inputs

Frontend-Output Page

There are three parts for the output page:

- **Test Result Display**
  It is set as a user navigation which includes an introduction for the project, input rules, submission rules and the refresh function. This part is one of the static files of a HTML file in the backend.

- **Download Button**
  This button is used for the user to download the test results which are usually graphical images in the project. The image name is related to the test section and current time, which is corresponding to a unique URL in the backend.

- **Return Button**
  The button could send a HTTP request to the backend for redirecting back to the test section (input page).

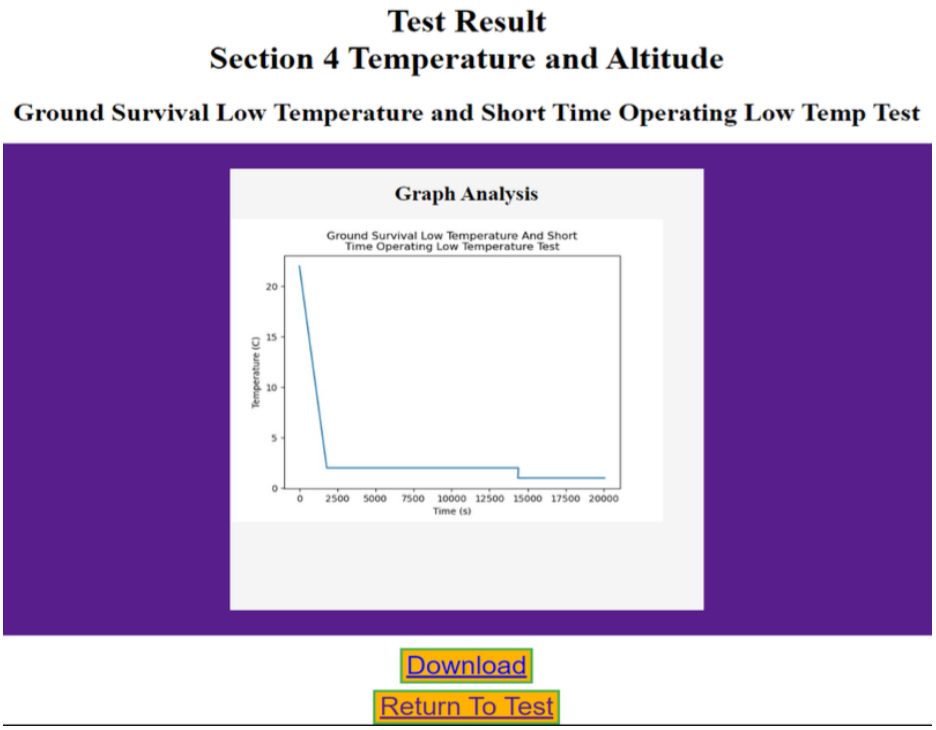

Figure 6: Output Page for displaying and downloading the test results

Data Processing Between Frontend and Backend

- **Data Transformation**
  The input data is stored in a JSON file by JavaScript, which could be accessed by Flask server. The image shown test result output is sent by a unique URL based on time from the backend. All the images for the results are stored in a static directory in the backend.

- **Initial Database Design**
  At the beginning of the project, a PostgreSQL database was used in order to collect the data about inputs and send them to the backend for testing. However, Since the required input data for each section is really less, usually less than 5 inputs for different tests, the data could be transmitted to the backed through JSON file, that is more efficient in time consumption.

Based on the design and implementation, this web application could assist the engineer when performing RTCA DO-160G testing, which consists of standards for environmental conditions and the corresponding test procedures for airborne equipment of all types of aircrafts. Although the DO-160G contains 16 different sections, the web application will focus on temperature and humidity testing.

Test Design and Procedure

Test Specifications

The importances of test specifications lies in that the output of the web application has to exactly match what is required from the DO-160G. This test, if the output is incorrect, the testing engineer who is responsible for doing the DO-160G test will not have performed the correct test. This can have issues in two distinct ways. Either the test is incorrect in understanding the requirements, or it is overstating the requirements. In the overstating case we do not have critical damage, as the piece of equipment is being subjected to a higher constraint. There is a non-critical issue in that in this scenario the piece of equipment is over engineered to sustain a higher - unnecessary requirement. Or in the overstate case and failure (false negative), the piece of equipment can be damaged costing money and time. With a piece of software that isn't known to be incorrect might be a difficult bug to discover. The issue is when the equipment to be tested is subjected to an understate test. In which a false positive occurs. There is a multifaceted way to address this problem. With exhaustive testing and with hard requirements. These specifications need to be met at the individual test level. While each test needs its own specific test parameters. Common groups can be made for tests that share variable limits. Such as in the Temperature Test: parameter bounds are made for upper and lower operating temperatures.

Test Cases

There are two sections that are of interest for this project. The Temperature section and the Humidity section. Both sections have values that are hard coded into source files. There are mutable files but they are not overwritable during the standard use of the project. The mutable values that are manual entry by the test engineer have hard upper and lower limits. These values are dedicated by the manufacturing based on the equipment, so a wide range is required.

The more nuanced case requires validation on an indusitual test basis. As from Figure 2 we can see the 8 various tests and the 19 possible equipment categories. The test validation process is to generate all 8 tests for the 19 possible equipment categories. This leads to 152 different (but similar) output results. These 152 possible graphs, some of which contain manufacturing inputs, can be tested with random values or min/max values, of which it is the input person's responsibility for. These 152 graphs are then matched with a manual search via the DO-160G book to validate their results. Each graph has approximately 6 key timing values with 2-4 temperature values of importances.

Because the foundation of the web application is to validate software with hardcopy values there isn't a readily available way to validate aside from manually looking up the values. The benefit of this, is it should only have to be validated once.

Results and Analysis

Source Code Results

The best way to understand the test results is to compare and contrast the work that was involved between the temperature and humidity tests. The temperature test was a much more inclusive test that required a significant amount more work. First to look at the test code for the humidity tests. We can see from the code below, there is a class oriented approach. The humidity test falls between three different equipment category selections, and there are preset values depending on each one. This then can be displayed based on what category is selected and the results can be displayed. This is the simplified version of the temperature test. The temperature test has significantly more information.

Humidity Classification Test

```
# Testing for section 6 Humidity
# Info:
# 3 specific categories (A,B,C)
# Three types of humidity tests
import matplotlib.pyplot as plt


class humidity_test():
    def __init__(self, equipment_category_input):
        self.equipment_category = equipment_category_input  # A,C or C
        # self.rate_of_change  # = get from table  or function (need to create)
        self.relative_humidity_sp1 = int  # set via cat. selection *** sp1 - set point 1
        self.relative_humidity_sp2 = int  # set via cat. selection **** sp2 - set point 2
        self.temp_sp1 = int  # tempature at which to stablizate
        self.temp_sp2 = int  #
        self.title_for_plot = str
        self.num_of_cyles = int
        # title_for_plot
        # Category A = Standard Humidity Enviorment
        # Category B = Severe Humidity Enviroment
        # Category C = External Humidity Enviroment

    def set_category_variables(self):
        if self.equipment_category == "A":
            self.relative_humidity_sp1 = 85  # +/-4
            self.relative_humidity_sp2 = 95  # +/-4
            self.temp_sp1 = 38  # +/- 2
            self.temp_sp2 = 50  # +/- 2
            self.title_for_plot = "Standard Humidity Enviorment"
            self.num_of_cyles = 2  # 48 hours

        elif self.equipment_category == "B":
            self.relative_humidity_sp1 = 85  # +/-4
            self.relative_humidity_sp2 = 95  # +/-4
            self.temp_sp1 = 38  # +/- 2
            self.temp_sp2 = 65  # +/- 2
            self.title_for_plot = "Severe Humidity Enviroment"
            self.num_of_cyles = 10  # 240 hours

        elif self.equipment_category == "C":
            self.relative_humidity_sp1 = 85  # +/-4
            self.relative_humidity_sp2 = 95  # +/-4
            self.temp_sp1 = 38  # +/- 2
            self.temp_sp2 = 55  # +/- 2
            self.title_for_plot = "External Humidity Enviroment"
            self.num_of_cyles = 6  # 144 hours

    def plot_equipment_category(self, path=''):
        print(path)
        f = plt.figure()
        plt.xlabel('Time (hours)')
        plt.ylabel('Temperature and Humidity')
        temp_time = [0, 2, 8, 24]
        temperature_levels = [self.temp_sp1,
                              self.temp_sp2, self.temp_sp2, self.temp_sp1]
        humidity_time = [0, 2, 8, 24]
        humidity_levels = [self.relative_humidity_sp1, self.relative_humidity_sp2,
                           self.relative_humidity_sp2, self.relative_humidity_sp1]
        plt.plot(temp_time, temperature_levels, label="Temperature")
        plt.plot(humidity_time, humidity_levels, label="Humidity")

        plt.legend()
        plt.title(self.title_for_plot)
        plt.savefig(path)
        plt.close()
        pass
```

Figure 7: Figure above referenced in Source Code results

The humidity test can be similarly compared to a single test for the temperature with more variables and more complexity. First we can take a look at the class build. We can see that the temperature class has to reference external files, these external files are permanent files that are included in the project. As more sections are included in the project, similar files will need to be created.

Temperature Class Structure

```
class temperature_tests:
    def __init__(self, equipment_category):
        # each value below pulls from table_4_1 based on the equipment category
        self.equipment_category = equipment_category
        self.operating_low_temp = table_4_1.at['operating_low_temp',
                                               self.equipment_category]
        self.operating_high_temp = table_4_1.at['operating_high_temp',
                                                equipment_category]
        self.short_time_operating_low_temp = table_4_1.at[
            'short_time_operating_low_temp', equipment_category]
        self.short_time_operating_high_temp = table_4_1.at[
            'short_time_operating_high_temp', equipment_category]
        self.ground_survival_low_temp = table_4_1.at['ground_survival_low_temp',
                                                     equipment_category]
        self.ground_survival_high_temp = table_4_1.at['ground_survival_high_temp',
                                                      equipment_category]
        self.loss_of_cooling_test = table_4_1.at['loss_of_cooling_test',
                                                 equipment_category]
        self.altitude_thousands_of_meters = table_4_1.at[
            'altitude_thousands_of_meters', equipment_category]
        self.altitude_thousands_of_feet = table_4_1.at['altitude_thousands_of_feet',
                                                       equipment_category]
        self.decompression_test = table_4_1.at['decompression_test',
                                               equipment_category]
        self.overpressure_test = table_4_1.at['overpressure_test',
                                              equipment_category]

        # parameter_dict is important for updating "notes" that are in the database instead of values
        self.parameter_dict = {'operating_low_temp': self.operating_low_temp,
                               'operating_high_temp': self.operating_high_temp,
```

Figure 8: Partial display of temperature class structure above

Moving on from the temperature class structure, there is the need for some of these variables to be manually imputed. We can see from below there is a function that allows us to identify what information is missing and have the user be able to manually input.

Check For Manufacturing Notes

```
    def check_for_notes(self, *args, input=''):  # pass in the necessary variables
        list = []
        for key in args:
            if "Note" in str(self.parameter_dict[key]):
                # #print({key}, "->", {self.parameter_dict[key]})
                # #self.parameter_dict[key] =
                # val = input(f"For Category: {key} and equipment_category {self.equipment_category}. Values are
supplied by the manufacturer. Please enter an manufacuturer's value for: {key} now: ")
                # #print({key}, "->", {self.parameter_dict[key]})
                # list.append(val)
                list.append(str(input))
                print(list)
            else:
                val = self.parameter_dict[key]
                list.append(val)
        return list
```

Figure 9: Function display for manual entries required

“Checking for notes” is a function name based on the DO-160G notation of calling manufacturing inputs based on what ‘note’ they have associated with. The notes may have slightly different information, but the end result is the same. A requirement of the user to input their specific manufactured setpoints. These manufacturing setpoints are not consistent and vary based on what equipment selection is made. From this point the next requirement is to generate the graph to show the required testing.

Generating plottable response

```
def figure_4_2_test_baseline_requirements(stab_time=30*60,  category="", input='', path=''):
    note = temperature_tests(category).check_for_notes(
        'operating_low_temp', input=input)
    dT0, temp0, op0 = 0, 22, 1
    dT1, temp1, op1 = 30*60, float(note[0]), 1
    dT2, temp2, op2 = stab_time, float(note[0]), 1
    dT3, temp3, op3 = 7200, float(note[0]), 1
    dt = [dT0, dT1, dT2, dT3]
    temps = [temp0, temp1, temp2, temp3]
    plt.xlabel('Time (s)')
    plt.ylabel('Temperature (C)')
    plt.plot(np.cumsum(dt), temps)
    plt.title('Operating Low Temp Test Requirements')
    plt.savefig(path)
    plt.close()
    pass

    # Ground Survival High Temperature and Short Time Operating High Temp Test
```

Figure 10: Above is code to generate a specific test: Ground Survival High Temperature and Short Time Operating High Temperature Test

Challenges

Front-End

- The image displayed in the output page is the same with the previous test result and isn't updated. This was solved by a related image with the current time rather than the name for the test section.
- The download function could work but always download the result for the first user, when other users choose different tests and obtain the output images, these images couldn't be downloaded successfully. This problem was solved by selecting the disable cache in the Network Section in the web browser. The reason is that by default, the browser creates a cache of the content that the user is viewing on the web so that it doesn't have to download it again when the user open the same webpage, but in our project, when user return back to the input page, it is actually an updated page than the previous one.

Back-End

- Working on a team project, the back end source code files were initially not connected to the front end framework. This was done to facilitate the quickness of developing the back end source files, while keeping in mind this issue would need to be resolved later to complete the project. This meant the hardcoded links would need to be integrated with the web application.

Changes

Front-End

- Based on the requirement from backend, most test results from DO-160G are shown as images rather than text format, so the output page remains the graph section and deleted the text section.
- The test sections in the Input page are valid based on the current work from the backend.

Back-End

- There were significant changes from how each section was coded. Originally the coding process was more based on a procedural style of coding. While this technically worked, it produced hard to read code that was lengthy to chance down bugs. The move to create more object oriented code was implemented. And it was easier to see where incorrect values were being produced.

Project Success Criteria

Implement graphical display of DO-160G graphs (complete for sections 4 and 6) Ultimately, we got the main industry concern complete and displayed the curves for sections 4 and 5 (Temperature and Humidity, respectively) However, there are many more sections

Implement PASS/FAIL verification of test conditions given in DO-160G (incomplete)

Creation of database to store inputs and outputs (dropped)

- Users' inputs from the frontend (web) could be successfully stored as different model sections in the local database (75% complete, SPRING QUARTER)
- The data from diverse model parts could be searched and retrieved successfully from the database for the Optimization Developer in the backend. (50% complete)
- Ultimately, we decided that a dedicated database wasn't necessary for the project and just stored all our data in the code. The data we're using isn't really large enough to require anything else.

All the testing results from the backend could be stored in the database and output in a suitable format, such as .xlsx or .csv file. (pending,SPRING QUARTER)

Creation of locally-hosted web application

- The website will run on localhost and display all the components from GUI Design. Users could interactively create requests with the web and obtain the response from server-side. (completed)

Design and implementation of human-machine interface

- Design of a simple, efficient human-machine interface that first time users could feasibly use unaided. (completed)
- Creation of user manual and guide giving instructions on use and limitations of software. (dropped) The program is so simple to use that a manual isn't necessary.

Impact and Consequences

1. The project is entirely software based and it is a Graphical User Interface (GUI) representation of the RTCA DO-160G document.
2. The DO-160G document was prepared by Special Committee 135 (SC-135) and approved by the RTCA Program Management Committee (PMC) on December 8, 2010. The document is based on the 'Environmental Conditions and Test Procedures for Airborne Equipment'. We have used a copy of this document for our project. This project can only be used by ZeroAvia, as the RTCA document is copyrighted.
3. Our chief ethical and safety concern is avoiding a 'false positive,' that is saying a component passes a test (or that a test was carried out correctly) when it isn't in reality. If we say some component is ready for flight, it turns out it isn't, and the component fails in flight, then the aircraft and anyone on it will be at risk.
4. The environmental effects of the program are not directly linked to the program itself, but rather the outcome the program provides. The program provides insight into the requirements set forth by the environmental document. It will make mistakes and missed information occur less often, therefore being an indirect benefit to the environment.

Conclusion and Recommendation

The overarching goal that is beyond the direct scope of the project is to speed up the development of hydrogen-electric powered aircrafts. Developing tools that don't contribute directly, but in an indirect way is where this project finds its motivation. Building this project from the ground up without legacy constraints has allowed the project to develop in a way that will not only allow but encourage future design modifications and development. The project succeeded in developing tools to aid in Temperature and humidity based testing. And by including the dynamic range of web based formatting, allows for future sections to bring in their specific requirements. Such as the future potential to include photo uploads to a database to record information from Mold testing. By focusing on a web application, not only can the tools within the program grow, but the reach of users can grow as well. The ability to host the program on a single user, multiple users via a local server or even pushing the same code to the web allows for total control of how the tool gets used. While the tool did not get pushed into production and real world testing, this is the next stage that allows for feedback for the continuously growing project.

Lessons Learned:
The project did not start out as a web application tool for the DO-160G. The original project was a multi-disciplinary optimization for the electric powertrain of an aircraft powertrain. Half way through the project the preemptive conclusion was made that the project would require significantly more time than what was allotted. This was made with the decision of our academic advisor. The team formulated a spread of options for minor adjustments, such as focusing on a single component rather than the full powertrain. But it was seen as more

valuable to pivot in a substantially different direction, the DO-160G. The goal was to merge what had previously been created, which was a web based tool for the previous project and collect what could be salvaged and was usable in the new project. This was the underlying foundation for the new project. The wire frame for the web application should be similar, the HMI/GUI would have to be modified, and the source files would need the largest rework. If given this project at the start, we believe most of that time used on the previous project would be spent doing review and modifications to the project to fine tune it. We were able to achieve the majority of the same goals.

References, Acknowledgements, and IP

“Environmental Testing and Evaluation using DO-160G” software package will be under standard copyright held by ZeroAvia. There will not be a set license other than said copyright. The software package will include a license stack based on its package dependencies that will include: Python Software License (PSL), Berkeley Software Distribution license (BSD), Massachusetts Institute of Technology license (MIT) and information held within the public domain. The combination of these license dependencies do not conflict and are commonly included in software that is: sold, private, open-source, closed source and/or proprietary software. It is up to the discretion and future modification by ZeroAvia how this software is licensed.

Packages and licenses used

- Python Language
    - Base type language used. No further dependencies.
    - PSF License (Python Software License)
- Flask
    - dependencies included: Werkzeug,Jinja, markupSafe, ItsDangerous, Click
    - Three clause BSD license
- Matplotlib
    - Dependencies included: Python, FreeType, Libpng, NumPy, setuptools, cycler, dateutil, kiwisolver, pyparsing
    - “MDT” License Agreement (based on PSF License)
- Conda
    - Versioneer.py located in the public domain
    - ProgressBar package: redistributed under the BSD 3-clause license
    - 3-clause BSD License with addudemum of set forth by 2017 Continuum Analytics, Inc. (dba Anaconda, Inc.)

Standards

1. This project is built based on Web Application. There are 2 standards applied in the project, WHATWG and ECMA:
   -WHATWG maintains the living standards for the HTML language.
   -ECMA which publishes the standard for ECMAScript, which JavaScript is based on.
2. There are general standard for web applications:
   - Must be easy and intuitive to use for the target audience.
   - Must function in a logical manner for the target audience.
   - Must use styles that are consistent throughout the application and within the associated website, including the use of capitalization, punctuation, error messages must appear in a consistent location and style, consistent use of any web document notations, layout/spacing, and descriptive metadata titles and descriptions.
   - Must adhere to industry best practices.
3. Web applications must be thoroughly tested in all required browser versions.
4. For database design, ER model is used to define an information structure which can be implemented in a database, typically a relational database.
5. Standard database object naming conventions are followed to identify database objects correctly and perform the proper administration tasks. When creating an object in PostgreSQL, every table and every column should have a name. PostgreSQL uses a single data type to define all object names, which is called name. A value of type name is a string of 63 or fewer characteristics. A name must start with a letter or an underscore; the rest of the string can contain letters, digits, and underscores.
6. This project uses Flask Framework to build a web app, all the behaviors such as Routing, Unique URLs, Redirection Behavior, HTTP methods and Sessions are followed on the Official Flask Documentation.
7. DO-160G is itself a document of environmental standards and tests that various aircraft components must meet.
8. The project is entirely software based and it is a Graphical User Interface (GUI) representation of the RTCA DO-160G document.
9. The DO-160G document was prepared by Special Committee 135 (SC-135) and approved by the RTCA Program Management Committee (PMC) on December 8, 2010. The document is based on the ‘Environmental Conditions and Test Procedures for Airborne Equipment’. We have used a copy of this document for your project.
10. Our technical ethical and safety concern is avoiding a ‘false positive,’ that is saying a component passes a test (or that a test was carried out correctly) when it isn’t in reality. If we say some component is ready for flight, it turns out it isn’t, and the component fails in flight, then the aircraft and anyone on it will be at risk.
11. The environmental effects of the program are not directly linked to the program itself, but rather the outcome the program provides. The program provides insight into the requirements set forth by the environmental document. It will make mistakes and missed information occur less often, therefore being an indirect benefit to the environment.

Ethical Considerations

The ethical considerations of a software program need to specifically address based on the ‘shareholder’ personnel that would be affected by this product. This includes not only the people who directly use the product, but the individuals who are indirectly affected by it, and who may not be aware of the program at all. The full list can be considered inexhaustible, but by identifying the primary stakeholders and the ethical considerations of the individual we can group ethical considerations by those affected. We can gain a better understanding of the full picture. The program centers around a document that is created by RTCA. The DO-160G is a private document that requires purchasing. Because ZeroAvia has purchased this document it is well within their ability to facilitate a program that aids them in using said document. The privacy considerations come into effect, only personnel at ZeroAvia would be able to use this document. Outside of ZeroAvia, distributing this document would be distributing a program that was under a different company's licenses and would need their permission to do so. The project has an indirect impact on our environment, with climate change being a growing consideration. The move to more renewable energy is the primary ethical consideration. This project aims at assisting the overall goal. Which is to grow the hydrogen-electric aircraft industry.

Appendices

Book-keeping: Github link: https://github.com/Ziw127/ZeroAvia
The following figure shows the file structures. The directories of static templates are used for front-end development and the virtual directory holds are the library used in the project including a python virtual environment. The file app.py is the main file for Flask running and a connection between frontend and backend. The file test.py is just a test file for backend developers. For the backend, the files section6.py, table4_1, and test_451_szw.py are related to DO-160G. In addition, an instruction for user operates in the terminal is in connection.txt.

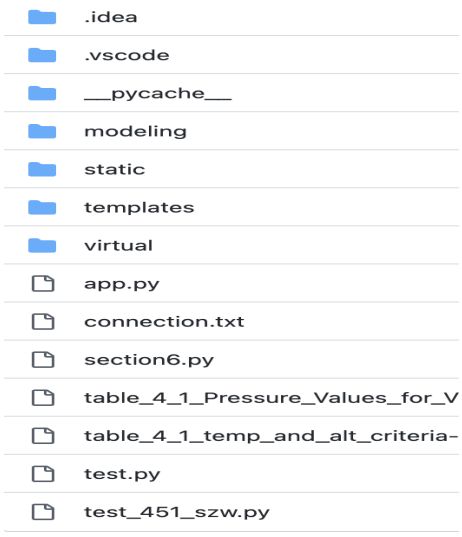

Figure 11: File Structures of the project

User Manual Information: Here is a brief introduction into the project for new users. This section constitutes a “one page” introduction into what the project is and how to get started as quickly as possible.

Intro: The goal of this project is to help facilitate the engineer when running DO-160G tests or validating data from a previously run test. There are 23 sections of DO-160G that pertain to testing. Each section has its own unique set of requirements and forms of validation. Ranging from specific time intervals that need to be met such as temperature to *more* subjective evaluations such as fungus resistance. Most of the test sections are further broken down into categories depending on the type of equipment being tested. Some of these require further input based on the manufacturer specifications. This project aims to allow for these inputs..

Goal: Interactive Web App for engineers to select categories, preview of curves, and load files.

Method: The foundation of the program is based around a “flask application.” Flask is a python based web framework that can be used when developing web based applications. While the goal of the project is not to deploy a public website, this allows for use of rich web based tools. The ‘website’ can be hosted via the same machine it runs the program on via its own loopback address and a specified port (127.0.0.1:5000) from the local web browser.

Requirements: When developing a program that has a large number of dependencies [matplotlib, numpy, sql, ect.] that also have specified versions. It suggests using a tool that can manage these dependencies while aiding in quick deployment to various machines for when the program needs to migrate. The migration could be to another local computer, a dedicated local/remote server, or possibly to a public/private website for remote access. For this project, being python language based, conda is the managing program to handle these types of migrations. Conda is a package/dependency manager and a virtual machine. There is the option to download to the local machine all of the dependencies and avoid using conda, But this can be considered ‘messy’ for the new user.

Example of transferring program [abbreviated]:

- Download Conda [via miniconda ->conda]
- Download program [git hosted]
- Activate environment [conda env activate DO160_Project_Name]
- In the terminal(Git bash for Windows OS system and default terminal for Mac OS system), the following command could also be seen in the file connection.txt.
- Run flask application [python app.py]
- Open web browser via loopback.port ip [http://127.0.0.1:5000/
- [after time passes]
- Run a script program that automatically starts conda nm environment, opens web browser, connects to hosted location. This will be OS specific!

Applicable Images and Tables:

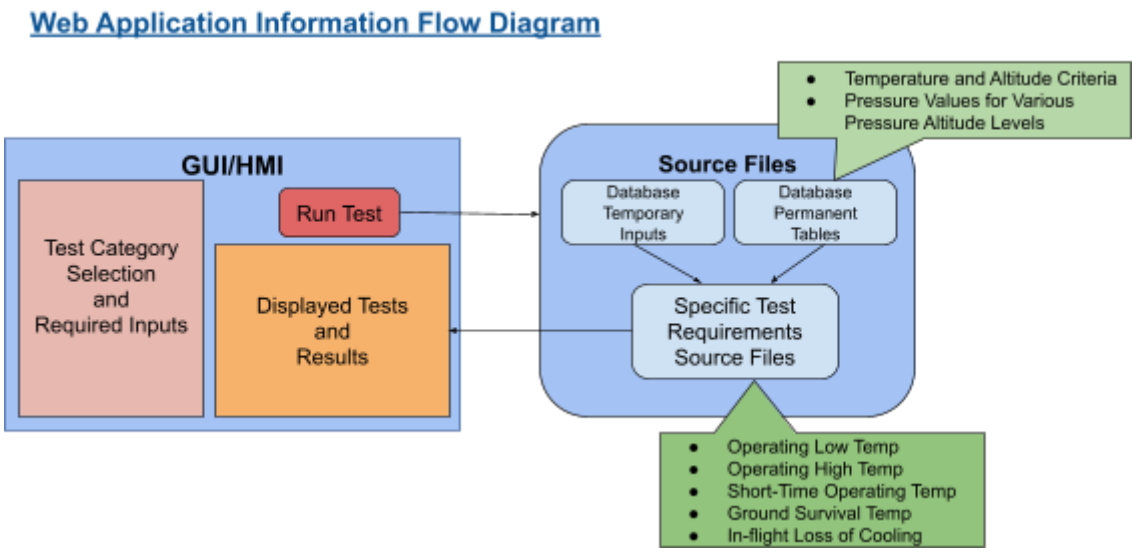

Figure 12: Web Application information Flow Diagram

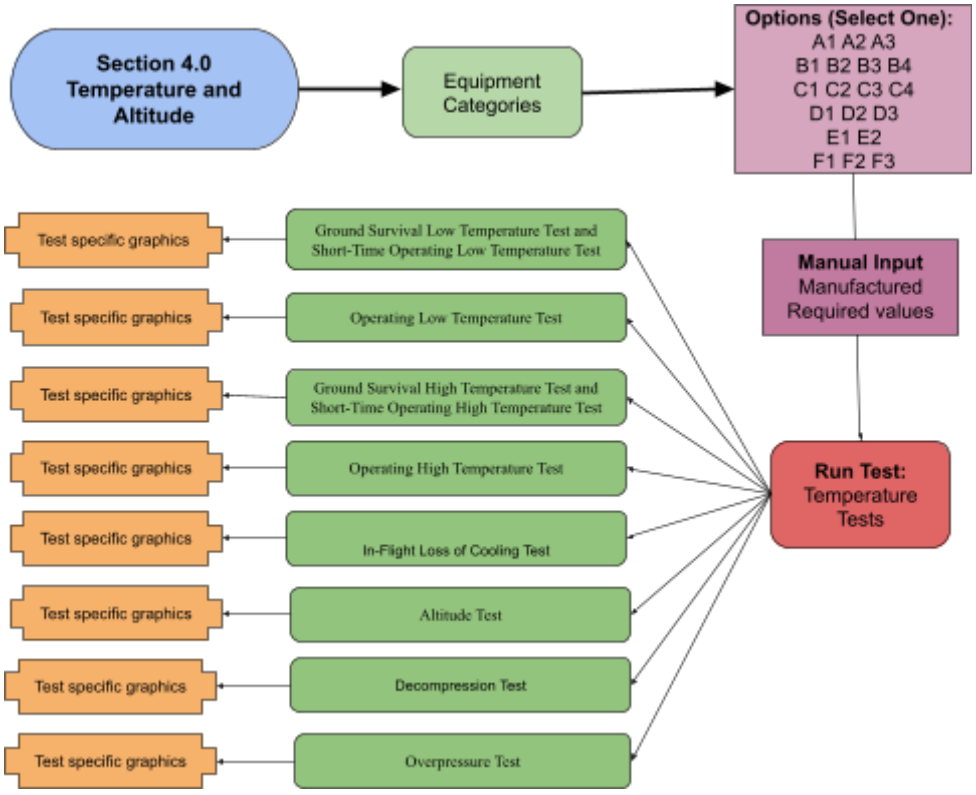

Figure 13: Information Flow For Section 4: Temperature and Altitude