Software Documentation Including Requirements, Design, Test Cases, and Test Results

for

Flight Tracking 1

Version 1.3

Prepared by Zackary Hagerty, Matthew Grabasch, Matthew Jolliffe, Devon Casey

DATE: 12/09/2021

Table of Contents

Revision History

Name	Date	Reason for Changes	Version
Devon Casey	11/03/2021	Creation	0.9
Matthew Grabasch	11/13/2021	Initial Information Added	0.9.1
Matthew Grabasch	12/04/2021	Project Updates/Run Test Cases	0.9.2
Zackary Hagerty	12/09/2021	Proofreading and updating	1.0

1. Introduction

1.1 Purpose

The purpose of this program is to aid aviators, enthusiasts, or those working in the aviation industry with tracking current flights. This program is limited to the continental United States and twelve major airlines. The interface allows for easy viewing of real-time flights over user-selected Air Traffic Control (ATC) zones. Users can elect to view flights based on ATC zone, airline, or any combination of the two. Each airplane shown can also depict its call sign if selected by the user. The focus of this program is simplicity, with an emphasis on the essential functionality of a flight tracker. This allows the program to be easy to use and understand. Other solutions for flight tracking have complex user interfaces and controls, making the program difficult to use.

1.2 Intended Audience and Reading Suggestions

The intended audience of this document includes Embry-Riddle Aeronautical University instructors, teaching assistants, or students who are appraising the functionality of the product through the scope of what was required by SE300. Nothing in this document is necessary to enjoy Flight Tracking's functionality and exists solely for the purpose of officially elaborating on the project's scope, design, and implementation.

1.3 Product Scope

The product will consist of a single graphical user interface (GUI)[1] depicting a realistic Mercator[2] projection map of the continental United States. In this GUI, the twenty US ATC zones will be outlined, and commercial flights from twelve major airlines currently in-flight will be displayed on the map according to their location and heading. The location and heading of each plane will be updated to reflect real-time data as often as the Application Programming Interface (API)[3] allows.

The product will allow a user to filter the current flights by airline and ATC zone. This functionality permits the user to reduce the number of planes visible, allowing the user to focus on what they deem necessary. For some users, this may be a single ATC zone, for other users - such as an airline company- this may be the flights from just that airline. Additionally, the program will have a simple help menu describing the program's functionality. The user interface will update regularly, ensuring airplane locations are current.

Other flight tracking programs allow for the user to zoom and pan across the map. It was discovered that this functionality is not possible with the software choices that were made. Animating the planes so that they continue moving between data pulls was also not possible due to software limitations. Tkinter[4], a Python GUI toolkit, was utilized to implement the GUI. However, Tkinter lacks the functionality required for implementing these two features. Because the latitude and longitude coordinates required to perform data pulls on flight data over the United States covers more geographical area than the ATC zones do, planes that are operating beyond the twenty ATC zones (ex planes in Canada or Mexico) will remain visible no matter which ATC zones are filtered in or out. This limitation comes from how the OpenSky data is transferred to the program: data is sent through a rectangular area (the continental United States, approximately), not geographically bound (such as through ATC zones). Furthermore, functionality that would be highly useful to pilots or air traffic control operators is absent; users can not interact with Flight Tracking to determine a plane's exact position in relation to airports or other airplanes, only conveying an airplane's general location.

1.4 References

The following sources have been used to help with the development:

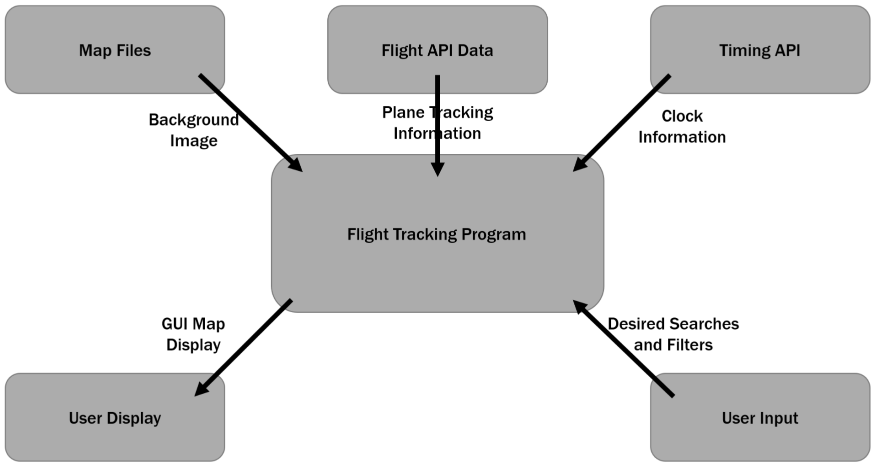
- The OpenSky API[5], which is used to pull airplane data.
  - https://opensky-network.org/apidoc/
- Tkinter, Pillow[6], and Pandas[7] library documentation, used in Python.
  - https://docs.python.org/3/library/tkinter.html
  - https://pillow.readthedocs.io/en/stable/

2. Overall Description

2.1 Product Perspective

At its core, the product is a visual representation of the data pulled from OpenSky's database. After an initial data pull, Flight Tracking uses OpenSky's data to populate a map of the United States, overlaid with planes from twelve commercial airlines. Every twenty seconds, the program pulls new data from OpenSky's database and repopulates the entire GUI with planes. The user can interact with the program to filter airplane visibility by company and ATC zone and can seek help through interaction with Clippy the paperclip, who serves as Flight Tracking's friendly help menu.

A Context Diagram is shown below to depict the structure of the program.



2.2 Product Functions

The main goal of the product is to display real-time commercial flight locations on a map of the continental United States. The locations and heading of each plane must update every 20 seconds when new OpenSky data is pulled. Only planes currently in-flight shall be viewable. Each plane will be depicted by a colored airplane icon representative of the respective airline.

12 major airlines were selected:

- American Airlines (AAL) - Blue
- Allegiant Airlines (AAY) - Orange
- Air Canada (ACA) - Red
- Air France (AFR) - Light Blue
- Aeromexico (AMX) - Green
- Alaska Airlines (ASA) - Light Gray
- Delta Airlines (DAL) - Burgundy
- Frontier Airlines (FFT) - Dark Green
- Jet Blue Airlines (JBU) - Cyan
- Spirit Airlines (NKS) - Yellow
- Southwest Airlines (SWA) - Dark Blue
- United Airlines (UAL) - Gold

These airlines represent the majority of commercial flights in the continental United States during a typical day. The user has the ability to filter the visible planes based on any number of airlines. The user can select airlines in a drop-down selection through the configuration menu. As a default state, all airlines are selected.

The user also has the ability to filter the visible flights based on the 20 United States ATC zones. The zones are listed below.

- Seattle - ZSE
- Oakland - ZOA
- Los Angeles - ZLA
- Salt Lake - ZLC
- Albuquerque - ZAB
- Denver - ZDV
- Minneapolis - ZMP
- Houston - ZHU
- Ft. Worth - ZFW
- Kansas City - ZKC
- Chicago - ZAU
- Memphis - ZME
- Indianapolis - ZID
- Cleveland - ZOB
- Atlanta - ZTL
- Boston - ZBW
- New York - ZNY
- Washington - ZDC
- Jacksonville - ZJX
- Miami - ZMA

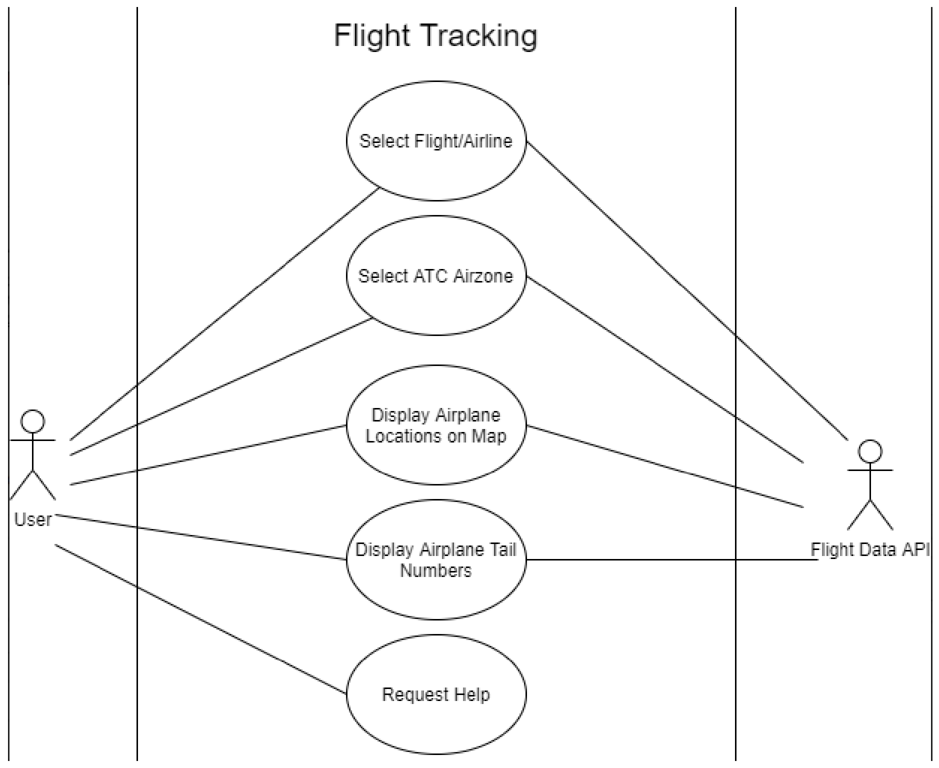
As a default state, only planes flying in the Los Angeles and Jacksonville ATC zones are visible. The user has the ability to filter planes based on any combination of ATC zones across the continental United States. This is through the use of a drop-down selection in the configuration menu. At any time, the borders of the ATC zones are shown on the map but below the planes. This is so that the user can see the extent of the ATC zone and not have to guess if planes are within the bounds of a particular ATC zone.

A help menu is also presented to the user in the bottom right corner. The help menu is activated when the user clicks on the help menu, represented by an anthropomorphized skeuomorphic paper clip reminiscent of the 1995 Microsoft Office Virtual Assistant "Clippit," colloquially named "Clippy." This help menu explains the basic functionality of the program and the user interactivity that is possible.

An essential aspect of this user interactivity is the "hover over planes" function. When a user hovers their mouse cursor over a visible plane, the airline code and callsign are visible.

2.3 User Classes and Characteristics

There is one possible Flight Tracking user, who will utilize Flight Tracking to view plane locations throughout the GUI. The user has the option to filter by company and ATC zone, giving them the ability to dynamically display the locations of airplanes in the GUI. Furthermore, the user can interact with the planes themselves to display airplane info, or can request help from Flight Tracking's help menu, The system's use case diagram has been included below to support the ideas presented here.



2.4 Operating Environment

The program was written in Python 3.9.7 in Windows 10.

- The user must maintain an internet connection throughout the usage of the program
- The user must utilize a laptop or desktop to use the program.
- For achieving full functionality, the program must be run in a Windows operating system
- The user needs to run the program from a virtual environment with the packages included in the requirements.txt file of the final deliverable (found in the GitHub repository).

2.5 Design and Implementation Constraints

The product has several limitations due to the nature of Tkinter and OpenSky.

OpenSky

- The data is pulled into a .json file from the OpenSky database, and the program must transform the file into data that is usable by Flight Tracking. This process freezes Flight Tracking for a few seconds, and the user cannot interact with the software during this time.
- OpenSky updates its database every fifteen seconds, so it is useless to pull from the database more than once in a fifteen-second time span. Therefore, Flight Tracking only performs a data pull every twenty seconds.

and Tkinter

- Tkinter has an unchangeable limit on the memory it can access, limiting the number of planes that can be displayed. Therefore, the program is designed to only display planes from a set of twelve major airlines.
- There is no way to interact with system memory through Tkinter, meaning performance optimization cannot be within the scope of the project.

2.6 Assumptions and Dependencies

- This product requires a stable and constant internet connection. This is because the program relies on live data from the OpenSky API. Constraints for OpenSky are discussed in section 2.5.

- This product also assumes that the number of planes visible on the map at any given time does not exceed approximately 5,000; Tkinter only has access to a limited amount of CPU memory, so when there are more than 5,000 planes in the sky, the program will crash due to an unfixable memory error thrown by Tkinter.

3. External Interface Requirements

3.1 User Interfaces

The user interface requirements are listed in Section 4.1. This section has been repurposed to display the functionality of Flight Tracking in respect to Section 4.1's requirements.

Figure 1 depicts the default state when the user initializes the program. Planes are only visible in ZLA and ZJX ATC zones (as well as outside the bounds of the continental United States ATC zones).

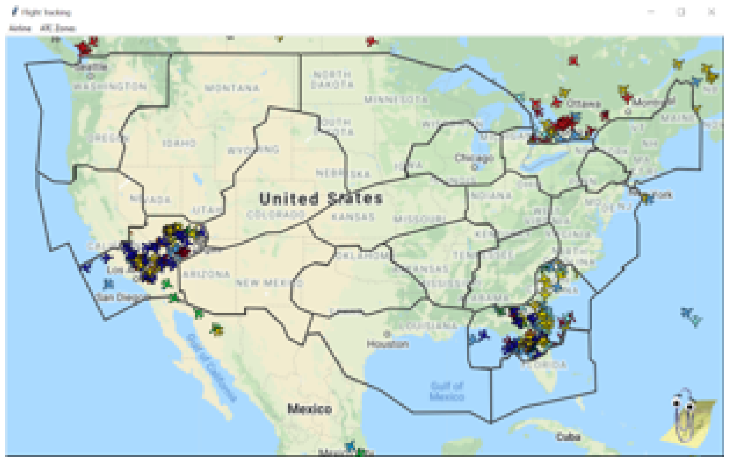


Figure 1

Figure 2 shows the selection menu for the different airlines. This menu can be found in the configuration menu.

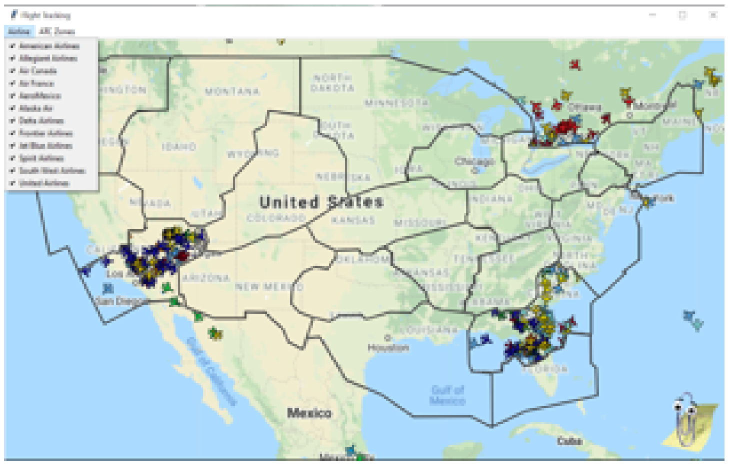


Figure 2

Figure 3 shows the selection menu for the different ATC zones. This menu can be found in the configuration menu.

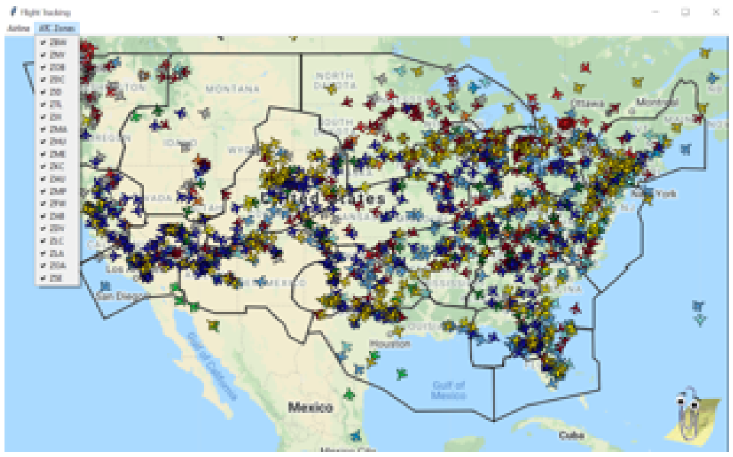
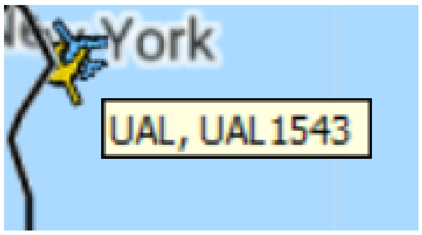


Figure 3

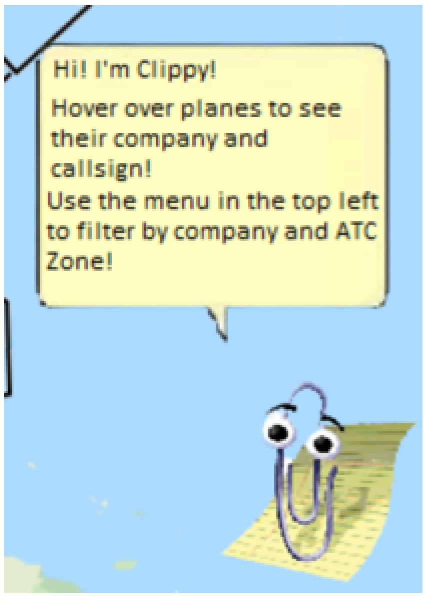
Figure 4 shows the flight information when the user hovers over a plane.



UAL, UAL1543

Figure 4

Figure 5 shows the help menu, toggleable by clicking on Clippy in the bottom right corner of the GUI (visible in Figures 1, 2, and 3).



Hi! I'm Clippy!
Hover over planes to see their company and callsign!
Use the menu in the top left to filter by company and ATC Zone!

Figure 5

3.2 Hardware Interfaces

This product is assumed to be run with the following hardware constraint:

HI.1 From a computer, not an external drive so that it can access system memory

3.3 Software Interfaces

The software this program interfaces with is listed below:

SI.1 OpenSky API

4. System Features

4.1 Functional Requirements

Use Case ID: FF1
Use Case Name: Flight Filter - American Airlines
Goal: User will be able to filter airlines so the GUI only displays American Airlines flights
Actors: User
Pre:

- Active Internet Connection
- OpenSky API is operational

Post:

- Only American Airlines flights are displayed on the GUI

Main Success Scenario:

Step	Actor Action	Step	System Reaction
1	User launches the program	2	The program pulls live flight data from the OpenSky API
		3	Program presents the default GUI state
4	User deselects all airlines besides American Airlines	5	Program hides all flights not belonging to American Airlines

Exceptions (Alternative Scenarios of Failure):

- No American Airlines flights are active at the time
- Program is unable to access OpenSky API flight data
- The User has all ATC Zones deselected, so all flights are hidden

- Req. 1: The system shall pull live data from the OpenSky API
- Req. 2: The system shall present the default state for the GUI
- Req. 3: The system shall allow the user to filter flights based on the airline
- Req. 4: The system shall display only American Airlines flights (as selected by the User)

Use Case ID: ATCF1

Use Case Name: Air Traffic Control Filter - ZBW
Goal: User will be able to filter airlines so the GUI only displays flights within the ZBW ATC Zone
Actors: User
Pre:
- Active Internet Connection
- OpenSky API is operational

Post:
- Only flights within the ZBW ATC Zone and outside the United States are displayed

Main Success Scenario:

Step	Actor Action	Step	System Reaction
1	User launches the program	2	The program pulls live flight data from the OpenSky API
		3	Program presents the default GUI state
4	User deselects all ATC zones besides ZBW	5	Program displays only those flights within the ZBW ATC zone and outside of the continental United States

Exceptions (Alternative Scenarios of Failure):
- No flights within the ZBW ATC zone at the time
- Program is unable to access OpenSky API flight data
- The user has all airlines currently deselected, so no flights are visible anywhere on the map

- Req. 1: The system shall pull live data from the OpenSky API
- Req. 2: The system shall present the default state for the GUI
- Req. 3: The system shall allow the user to filter flights based on the ATC zone
- Req. 4: The system shall display only the flights within the ZBW ATC zone and outside of the continental United States (as selected by the User)

1) Use case: The user will be able to view airplane locations on a map of the United States

1.1) As a user, I want to display airplanes so that I can view their current location

A. *Req. 1:* The system shall display all planes operated by American Airlines over the continental United States
B. *Req. 2:* The system shall display all planes operated by Allegiant Airlines over the continental United States
C. *Req. 3:* The system shall display all planes operated by Air Canada over the continental United States
D. *Req. 4:* The system shall display all planes operated by AirFrance over the continental United States
E. *Req. 5:* The system shall display all planes operated by AeroMexico over the continental United States
F. *Req. 6:* The system shall display all planes operated by Alaska Airlines over the continental United States
G. *Req. 7:* The system shall display all planes operated by Delta Airlines over the continental United States

H. *Req. 8:* The system shall display all planes operated by Frontier Airlines over the continental United States
I. *Req. 9:* The system shall display all planes operated by JetBlue over the continental United States
J. *Req. 10:* The system shall display all planes operated by Spirit over the continental United States
K. *Req. 11:* The system shall display all planes operated by Southwest over the continental United States
L. *Req. 12:* The system shall display all planes operated by United Airlines over the continental United States
M. *Req. 13:* The system shall display each airline in a unique color
N. *Req. 14:* The system shall display each flight in the direction of its heading
O. *Req 15:* The system shall update the location of each flight every 20 seconds

1.2) As a user, I want to view callsign information for each flight
A. *Req. 16:* The system shall display callsign information for each flight upon hovering over the airplane on the GUI

1.3) As a user, I want to have a help menu so I can understand how to use the program
A. *Req 17:* The program will have a help menu displayed upon clicking on Clippy

2) Use case: The user will be able to filter airplane visibility by company
2.1) As a user, I want to filter airplanes based on company so I can view all flights belonging to a specific airline
A. *Req. 18:* The system shall contain a dropdown menu of airlines for selection by the user
B. *Req. 19:* The system shall allow filtering of American Airlines flights
C. *Req. 20:* The system shall allow filtering of Allegiant Airlines flights
D. *Req. 21:* The system shall allow filtering of Air Canada flights
E. *Req. 22:* The system shall allow filtering of AirFrance flights
F. *Req. 23:* The system shall allow filtering of AeroMexico flights
G. *Req. 24:* The system shall allow filtering of Alaska Airlines flights
H. *Req. 25:* The system shall allow filtering of Delta Airlines flights
I. *Req. 26:* The system shall allow filtering of Frontier Airlines flights
J. *Req. 27:* The system shall allow filtering of JetBlue flights
K. *Req. 28:* The system shall allow filtering of Spirit flights
L. *Req. 29:* The system shall allow filtering of Southwest flights
M. *Req. 30:* The system shall allow filtering of United Airlines flights

3) Use case: The user will be able to filter airplane visibility by ATC Zone
3.1) As a user I want to filter airplanes based on company so I can view all flights within a specific ATC Zone
A. *Req. 31:* The system shall contain a dropdown menu of ATC Zones for selection by the user

B. *Req. 32:* The system shall allow filtering of flights within the Seattle (ZSE) ATC Zone
C. *Req. 33:* The system shall allow filtering of flights within the Oakland (ZOA) ATC Zone
D. *Req. 34:* The system shall allow filtering of flights within the Los Angeles (ZLA) ATC Zone
E. *Req. 35:* The system shall allow filtering of flights within the Salt Lake (ZLC) ATC Zone
F. *Req. 36:* The system shall allow filtering of flights within the Albuquerque (ZAB) ATC Zone
G. *Req. 37:* The system shall allow filtering of flights within the Denver (ZDV) ATC Zone
H. *Req. 38:* The system shall allow filtering of flights within the Minneapolis (ZMP) ATC Zone
I. *Req. 39:* The system shall allow filtering of flights within the Houston (ZHU) ATC Zone
J. *Req. 40:* The system shall allow filtering of flights within the Fort Worth (ZFW) ATC Zone
K. *Req. 41:* The system shall allow filtering of flights within the Kansas City (ZKC) ATC Zone
L. *Req. 42:* The system shall allow filtering of flights within the Chicago (ZAU) ATC Zone
M. *Req. 43:* The system shall allow filtering of flights within the Memphis (ZME) ATC Zone
N. *Req. 44:* The system shall allow filtering of flights within the Indianapolis (ZID) ATC Zone
O. *Req. 45:* The system shall allow filtering of flights within the Cleveland (ZOB) ATC Zone
P. *Req. 46:* The system shall allow filtering of flights within the Atlanta (ZTL) ATC Zone
Q. *Req. 47:* The system shall allow filtering of flights within the Boston (ZBW) ATC Zone
R. *Req. 48:* The system shall allow filtering of flights within the New York (ZNY) ATC Zone
S. *Req. 49:* The system shall allow filtering of flights within the Washington DC (ZDC) ATC Zone
T. *Req. 50:* The system shall allow filtering of flights within the Jacksonville (ZJX) ATC Zone
U. *Req. 51:* The system shall allow filtering of flights within the Miami (ZMA) ATC Zone

3) Use case: The user will be able to filter airplane visibility by ATC Zone

4.2 Non-functional Requirements

4.2.1 Performance Requirements

[P Req 1]	The system shall only display the number of planes it can without crashing
[P Req 2]	The system shall not crash under normal operating circumstances
[P Req 3]	The system shall not be run for more than 8 continuous hours
[P Req 4]	The system shall run smoothly between data pulls
[P Req 5]	The system shall temporarily freeze user input while updating flight data

4.2.2 Security Requirements

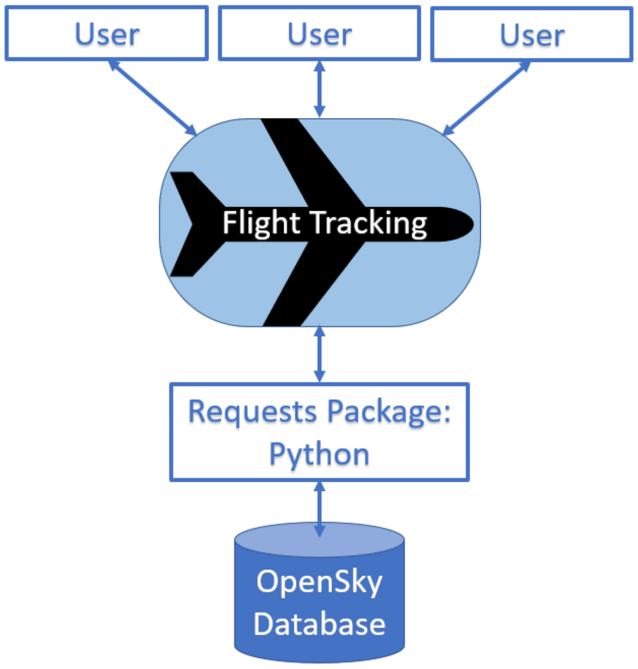
SE.1 The system shall comply with all FAA security rules and laws when handling flight data

SE.2 The system shall comply with all FCC communications rules and laws when handling flight data

5. Software Design

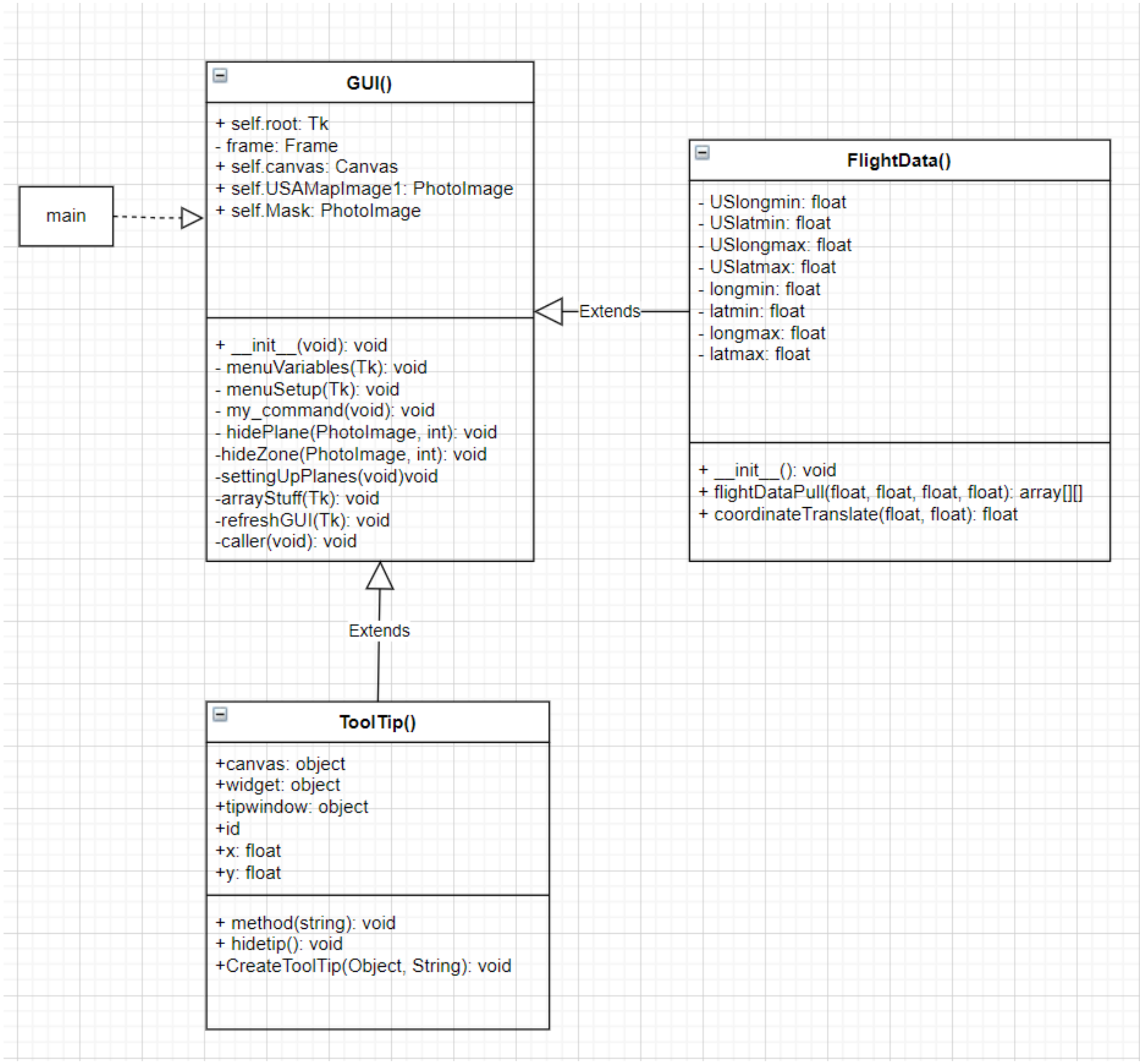
5.1 High-Level Design

This software uses a joint software architecture between the service-oriented and the client-server architectures. The main software operates on the service-oriented principle where the user is interacting with the producer through the User Interface, and a service broker handles which producers are shown. The data pull and interface with the Open Sky API is the client-server part of the software, where the service broker from the last architecture now serves as the client which interfaces with the Open Sky API Server. The software incorporates both architectures to allow the final product to be presented to the user.



5.2 Detailed Design

This section will discuss each class in Flight Tracking's code, describing its purpose, the functionality of its methods, and how it interacts with other classes. The UML diagram of the program is below.



1. class GUI()

The purpose of GUI(), and its methods, are to facilitate the creation and updating of Flight Tracking's GUI. GUI()'s methods pull and filter OpenSky data, then create and update a Tkinter instance based on OpenSky's data pull.

1.1. GUI() methods

1.1.1. main()

Creates an instance of tkinter, root, and passes it to the GUI object.

1.1.2. __init__()

GUI()'s constructor method. Receives a Tkinter instance from main, and populates the Tkinter instance with a Frame and a Canvas, establishing a functional GUI. Populates the GUI with an image of the United States, then calls functions menuVariables(), menuStetup(), settingUpPlanes(), and arrayStuff(), establishing the Flight Tracking's update cycle.

1.1.3. menuVariables()

menuVariables() defines the variables necessary for the filtering options available through the configuration menu at the top of the GUI.

1.1.4. menuSetup()

menuSetup() creates the configuration menu, establishing the functionality needed to filter planes by company and ATC zone.

1.1.5. my_command()

This method facilitates the functionality of the help menu- when the help menu (represented by an image of Clippy the Paperclip) is clicked, this method is called to create a text box explaining Flight Tracking's functionality

1.1.6. hidePlane()

This function is called when the user interacts with the configuration menu's company filtering options. The user can deselect a company, which results in every one of that company's planes disappearing. When the company is reselected, the company's planes will repopulate.

1.1.7. hideZone()

This function is called when the user interacts with the configuration menu's ATC zone filtering options. The user can deselect an ATC zone, which results in all planes in that specific ATC zone disappearing. When the ATC zone is reselected, the planes in that ATC zone will repopulate.

1.1.8. settingUpPlanes()

settingUpPlanes() initializes every plane sprite, every ATC zone sprite, and the photo of the duck.

1.1.9. arrayStuff()

arrayStuff() triggers a OpenSky data pull, storing all info in a 2D array before filtering that data into smaller company-specific arrays, which hold each plane's call sign, longitude, latitude, and heading.

1.1.10. refreshGUI()

refreshGUI() plots all planes and ATC zone masks dependent on what is currently selected in the config menu. This method maintains the current state, between data refreshes, of whatever filtering options the user has selected.

1.1.11. caller()

caller()'s sole purpose is creating a huge, transparent image underneath the user's cursor before Flight Tracking refreshes, severing the connection between the user's cursor and any planes. If this is not done, any instances of the ToolTip() class will fail to disappear between refreshes.

2. class ToolTip()

The purpose of ToolTip(), and its methods, are to facilitate the creation and destruction of text boxes that display a plane's company and call sign when the user hovers over it with their cursor.

2.1. ToolTip() methods

2.1.1. __init__()

ToolTip()'s constructor. Assigns a passed canvas and airplane to the class's variables to be used in tip creation and destruction.

2.1.2. showtip()

showtip() receives a string from refreshGUI(), calculates an x and y position dependent on the user's cursor location, and creates a text box near the user's cursor with that text,

2.1.3. hidetip()

hidetip() deletes any currently open text boxes created by showtip()

2.1.4. CreateToolTip()

This method binds showtip() to the user's cursor hovering over a plane, and hidetip() to the user's cursor leaving a plane.

3. class FlightData()

The purpose of FlightData is to carry out a data pull on OpenSky's flight database. The class is only accessed as an object but includes a method to convert pulled latitude and longitude data to x and y coordinates that can be used to plot airplanes.

3.1. FlightData() methods

3.1.1. __init__()

FlightData()'s constructor. Contains a minimum longitude variable, a minimum latitude variable, a maximum longitude variable, and a maximum latitude variable necessary to request

3.1.2. flightDataPull()

flightDataPull() connects to OpenSky's database and pulls the data into a JSON file, which is then translated to a 2D array and returned.

3.1.3. coordinateTranslate()

coordinateTranslate() is passed a latitude and longitude value, which is translated to x and y coordinates and returned so that planes can be plotted.

5.3 Detail Code

Detailed Code can be found here:
Github:https://github.com/ZackHagerty/FlightTrackingProject

6. Test Cases and Results

Test ID: FT_1	**Designed by: Devon Casey**	**Executed by (on date): Matt Jolliffe (12/4/2021)**	
Purpose	Verify that the map and the planes appear		
Dependencies	• none		
Pass/ Fail Conditions	Program starts and shows map of USA with planes on it		
Pre-conditions:	Internet Connection, OpenSky API is functional		
Test Data:	N/A		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Press Run or open program	Program should launch and a map of the USA with multicolored planes on top should appear	P
Comments	Works as required		

Test ID: FT_2	**Designed by: Devon Casey**	**Executed by (on date): Matt Jolliffe (12/4/2021)**	
Purpose	Verify planes have pop-ups with plane information		
Dependencies	• FT_1		
Pass/ Fail Conditions	Plane will show a pop up with information when hovered over		
Pre-conditions:	Internet Connection, OpenSky API is functional		
Test Data:	None		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Hover over a plane on the map with mouse cursor	Pop up appears with information about the callsign and airline it is associated with	P
Comments	Stalls during data update, but after 2 seconds resumes working		

Test ID: FT_3	**Designed by: Devon Casey**	**Executed by (on date): Matt Jolliffe (12/4/2021)**

Purpose	Verify planes are in the proper location and heading based on real data		
Dependencies	• ***FT_1*** • ***FT_2***		
Pass/ Fail Conditions	Plane location must match with data from Flight Radar 24		
Pre-conditions:	Internet Connection, OpenSky API is functional		
Test Data:	None		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Hover over a displayed airplane	Plane should show its call sign when hovering	P
2.	Enter call sign into Flight Radar 24	Callsign should appear as a plane currently operating in the sky on Flight Radar 24	P
3.	Ensure that the location on Flight Radar 24 matches the one on the map	Relative locations should be the same	P
4.	Ensure that the heading on Flight Radar 24 matches the one of the planes on the map	Relative headings should be the same	P
Comments	Works as required, most accurate immediately after data pull, but not a noticeable difference after 20 seconds.		

Test ID: FT_4	**Designed by: Matt Jolliffe**	**Executed by (on date): Matt Jolliffe (12/4/2021)**	
Purpose	Verify airplane colors match airline		
Dependencies	• FT_1 • FT_2		
Pass/ Fail Conditions	Each plane has the proper color assigned to it based on the key below		
Pre-conditions:	Internet Connection, OpenSky API is functional		
Test Data:	The following key is needed: American Airlines (AAL) = Blue Allegiant Airlines (AAY) =Orange Air Canada (ACA) = Red Air France (AFR) = Light Blue Aeromexico (AMX) = Green Alaska Airlines (ASA) = Light Gray Delta Airlines (DAL) = Burgundy Frontier Airlines (FFT) = Dark Green Jet Blue Airlines (JBU) = Cyan Spirit Airlines (NKS) = Yellow Southwest Airlines (SWA) = Dark Blue United Airlines (UAL) = Gold		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Hover over a plane	Verify the color of the plane matches the airline of the plane as displayed by the call sign	P
2.	Repeat for 2 planes of every color	Same as step 1	P

Comments	Works as required

Test ID: FT_5	**Designed by: Devon Casey**	**Executed by (on date): Matt Jolliffe (12/4/2021)**	
Purpose	Verify that planes move after a period of time		
Dependencies	• FT_1 • FT_3		
Pass/ Fail Conditions	Every plane in the sky will move every 20 seconds		
Pre-conditions:	Internet Connection, OpenSky API is functional		
Test Data:	None		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Note the position of an arbitrary plane	Planes should appear and a specific plane should be noted by the tester	P
2.	Find the same plane on Flight Radar 24, as was done in steps 1-3 on FT_3	Plane should appear in Flight Radar 24 at approximately the same location	P
3.	Wait 20 seconds	Plane displayed by program GUI should move approximately the same distance as on Flight Radar 24 during that duration of time	P
4.	Repeat steps 1-3 for 5 more planes	Same as above steps	P
Comments	With the scale of the map, this was difficult to verify in its entirety however it is definitely close enough for confidence in the program		

Test ID: FT_6	**Designed by: Devon Casey**	**Executed by (on date): Matt Jolliffe (12/4/2021)**
Purpose	Verify airline filtering menu works	
Dependencies	• FT_1 • FT_3 • FT_4 • FT_5	
Pass/ Fail Conditions	Normal operation as detailed by prior test cases but with a filter attached	
Pre-conditions:	Internet Connection, OpenSky API is functional	
Test Data:	The following key is needed: American Airlines (AAL) = Blue Allegiant Airlines (AAY) =Orange Air Canada (ACA) = Red Air France (AFR) = Light Blue Aeromexico (AMX) = Green Alaska Airlines (ASA) = Light Gray Delta Airlines (DAL) = Burgundy Frontier Airlines (FFT) = Dark Green Jet Blue Airlines (JBU) = Cyan Spirit Airlines (NKS) = Yellow Southwest Airlines (SWA) = Dark Blue United Airlines (UAL) = Gold	

Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Click the Airline Button in the top left	Drop-down page should be visible with all airlines listed and selected	P
2.	Deselect all airlines on the list other than American Airlines	Only Blue planes should remain on the map and they should operate as described in every other test case using Flight Radar 24 data to ensure proper location and movement	P
3.	Select Allegiant airlines from the airlines drop-down menu	All Allegiant planes (colored Orange) should appear at their current location	P
4	Repeat step 3 for every other airline until all are selected again	Each airline's aircraft should appear in their respective color when selected in the drop-down menu	P
Comments	None		

Test ID: FT_7	**Designed by: Devon Casey**	**Executed by (on date): Matt J (12/4/2021)**	
Purpose	Verify ATC Filtering Menu Works		
Dependencies	• FT_1 • FT_3 • FT_4 • FT_5 • FT_6		
Pass/ Fail Conditions	When an ATC zone is selected, all planes within that region will appear		
Pre-conditions:	Internet Connection, OpenSky API is functional		
Test Data:	None		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Click the ATC Zone dropdown menu in the top left	Dropdown menu should be visible and all ATC Zones should appear selected	P
2.	Deselect all ATC Zones other than Seattle (ZSE)	Each ATC Zone other than ZSE should no longer display any aircraft	P
3.	Select each other ATC Zone, one at a time, from the menu	Airplanes within each ATC Zone should appear once selected	P
Comments	None		

Test ID: FT_8	**Designed by: Devon Casey**	**Executed by (on date): Matt J (12/4/2021)**
Purpose	Verify help button operation	
Dependencies	• ***FT_1***	
Pass/ Fail Conditions	Help button displays help description	
Pre-conditions:	Internet Connection, OpenSky API is functional	
Test Data:	N/A	

Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Click on the help button, Clippy	The help menu should display	P
2.	Click on the help button again	The help menu should disappear	P
3.	Click on the help menu once the program refreshes	The menu should display the same as in step 1	P
Comments	None		

6.1 Traceability Matrix

For each requirement, there will be one or more test cases.

Requirement	Test case	Results
1.1.1	FT_6	P
1.1.2	FT_6	P
1.1.3	FT_6	P
1.1.4	FT_6	P
1.1.5	FT_6	P
1.1.6	FT_6	P
1.1.7	FT_6	P
1.1.8	FT_6	P
1.1.9	FT_6	P
1.1.10	FT_6	P
1.1.11	FT_6	P
1.1.12	FT_6	P
1.1.13	FT_4	P
1.1.14	FT_3	P
1.1.15	FT_5	P
1.2.16	FT_2	P
1.3.17	FT_8	P
2.1.18	FT_6	P
2.1.19	FT_6	P
2.1.20	FT_6	P
2.1.21	FT_6	P
2.1.22	FT_6	P
2.1.23	FT_6	P
2.1.24	FT_6	P
2.1.25	FT_6	P
2.1.26	FT_6	P
2.1.27	FT_6	P

2.1.28	FT_6	P
2.1.29	FT_6	P
2.1.30	FT_6	P
3.1.31	FT_7	P
3.1.32	FT_7	P
3.1.33	FT_7	P
3.1.34	FT_7	P
3.1.35	FT_7	P
3.1.36	FT_7	P
3.1.37	FT_7	P
3.1.38	FT_7	P
3.1.39	FT_7	P
3.1.40	FT_7	P
3.1.41	FT_7	P
3.1.42	FT_7	P
3.1.43	FT_7	P
3.1.44	FT_7	P
3.1.45	FT_7	P
3.1.46	FT_7	P
3.1.47	FT_7	P
3.1.48	FT_7	P
3.1.49	FT_7	P
3.1.50	FT_7	P
3.1.51	FT_7	P

Appendix A: Glossary

Requirements:

[1] Graphical User Interface (GUI): A way to interact with a computer system visually, such as with buttons, menus, or icons, removing the need to interact with the code of a program directly.
[2] Mercator: A cylindrical representation of the globe, orienting up as north and down as south everywhere. This is the most common type of map for navigation.
[3] Application Programming Interface (API): any set of functions or procedures that let one application interact with or pull data from another application or data service.
[4] Tkinter: A GUI interface package specifically built for Python
[5] OpenSky API: An API that allows for the retrieval of real-time flight data for non-profit, research purposes.
[6] Pillow (Python Imaging Library): A Python library specifically designed for manipulating image files, including using them within GUIs, such as through the use of Tkinter.
[7] Pandas: A software library for Python that is used primarily for manipulating large data sets and tables