# System Design Document

## For

## RTube NeMo Team

Team members:
Maegan Lucas, Maxwell Moolchan, Darian Hopkins, Taylor Sumlin, Adam Fitch

| Version | Author | Date | Description of Changes |
|---|---|---|---|
| 1.0 | Maegan Lucas | 26 September, 2023 | Initial creation of document |
| 1.1 | Maegan Lucas | 26 September, 2023 | Updated document name |
| 1.2 | Adam Fitch | 28 September, 2023 | Added Section 1.2, Edited Section 1.2 |
| 1.3 | Darian Hopkins | 28 September, 2023 | Added Sections 1.4, 3.1, 4.2, 4.3, 5.1, 5.2, 6, Team Member Names, and gave credit to previous team |
| 1.4 | Adam Fitch | 29 September, 2023 | Added Section 1.3, Edited Sections 1.2, 1.2.1 |
| 1.5 | Adam Fitch | 29 September, 2023 | Added Section 1.2.3 |
| 1.6 | Taylor Sumlin | 29 September, 2023 | Updated Table of Contents |
| 1.7 | Adam Fitch | 29 September, 2023 | Added Sections 2.1, 2.2 |
| 1.8 | Maegan Lucas | 29 September, 2023 | Added Section 1.3.2, 1.6, 4.1, Edited Sections 4.2, 5.1, 5.2, Minor text edits through entire document |
| 1.9 | Maegan Lucas | 29 September, 2023 | Edited Sections 4.3, 5.1, 5.2, 6 |
| 2.0 | Maxwell Moolchan | 30 October, 2023 | Edited Team names to be full names, Updated table of contents, added detailed log of all document changes up to this point, added page numbers & revision comments |
| 2.1 | Maxwell Moolchan | 30 October, 2023 | Edited Section 1.2, 1.2.3, 1.5, 2.1, 4.2, Minor edits through the entire document |
| 2.2 | Maegan Lucas | 31 October, 2023 | Edited Section 1.1, 1.2 |
| 2.3 | Maxwell Moolchan | 31 October, 2023 | Added Term to Section 1.5 |
| 2.4 | Maxwell Moolchan | 31 October, 2023 | Added Section 1.2.2, Edited Section 1.4 |
| 2.5 | Darian Hopkins | 31 October, 2023 | Added Diagram to Section 4.2, 4.3 |
| 2.6 | Maegan Lucas | 31 October, 2023 | Added ER Model and made final updates |
| 3.0 | Maegan Lucas | 6 November, 2023 | Added diagrams |
| 3.1 | Adam Fitch | 16 November, 2023 | Edited Section 4.1, 4.2 |
| 3.2 | Adam Fitch | 17 November, 2023 | Edited Section 1.2.3, 4.1, 4.2 |
| 3.3 | Taylor Sumlin | 20 November, 2023 | Edited All Sections |

| | | | |
|---|---|---|---|
| 3.4 | Darian Hopkins | 20 November, 2023 | Edited Section 3.1, 4.2,<br>Added Diagrams to Section 4.3 |
| 3.5 | Maegan Lucas | 20 November, 2023 | Added updated ER model in section 4.3,<br>Final updates |
| 3.6 | Taylor Sumlin | 21 November, 2023 | Updated Section 1.2, 4.2 |
| 3.7 | Maegan Lucas | 1 December, 2023 | Made updates based on V2 comments. |

Previous Version: Speech Recognition for Air Traffic Control (Capstone 2022-2023)
Previous Version Team members : Braeden Burnett, Jakob Haehre, Kira McFadden, Tyler Carr

# TABLE OF CONTENTS

**1 INTRODUCTION..... 4**
1.1 Purpose and Scope..... 4
1.2 Project Executive Summary..... 4
1.2.1 System Overview..... 5
1.2.2 Design Constraints..... 6
1.2.3 Future Contingencies..... 7
1.3 Document Organization..... 7
1.4 Project References..... 8
1.5 Glossary..... 9
**2 SYSTEM ARCHITECTURE..... 9**
2.1 System Hardware Architecture..... 9
2.2 System Software Architecture..... 10
2.3 Internal Communications Architecture..... 11
**3 HUMAN-MACHINE INTERFACE..... 12**
3.1 Inputs..... 12
3.2 Outputs..... 12
**4 DETAILED DESIGN..... 14**
4.1 Hardware Detailed Design..... 14
4.2 Software Detailed Design..... 15
4.3 Internal Communications Detailed Design..... 20
**5 EXTERNAL INTERFACES..... 26**
5.1 Interface Architecture..... 26
5.2 Interface Detailed Design..... 26
**6 SYSTEM INTEGRITY CONTROLS..... 26**

# SYSTEM DESIGN DOCUMENT

## 1 INTRODUCTION

### 1.1 Purpose and Scope

This project consists of two teams working on this ongoing project. The Kaldi team, which is in charge of developing and innovating upon the existing AI technologies to be able to convert ATC (Air Traffic Control) speech to text. The purpose of this is to be able to utilize text data to be able to better understand communications for pilots, ATC Controllers, and other relevant parties. The NeMo team focuses on integrating NVIDIA NeMo models, and in the future, the Kaldi model, into the web application as well as developing a database to be able to store audio communications between aircraft and ATC along with airport and flight data. The NeMo team also focuses on updating the existing user interface of the web application.

The first semester of the project will focus on the creation of a communications database, learning about and initial execution of NVIDIA NeMo, and some visual changes to the existing user interface. The second semester of the project will focus on establishing the connections between the communications database and the interactive map, the database and NVIDIA NeMo, further optimization of NVIDIA NeMo transcriptions, and more ser interface updates that will involve both the database and transcriptions. Anything not described in this document will be considered out of the scope of the project.

### 1.2 Project Executive Summary

The purpose of this project is to develop an application that is capable of listening to live Air Traffic Control communications and be capable of transcribing them to be used to study, research and learn. This project is primarily for flight instructors to assist flight students during their first solo flights. Additionally, flight students will be able to access the tool to be able to understand their communication skills or be able to listen in as a new student to begin to understand ATC.

The project was planned to be developed over two semesters using the Agile Scrum workflow method during the development of the project. Each semester was divided into 3 sprints varying in length. Semester 1's focus was integrating the concepts learned from the previous group and building those concepts in software that was faster and better suited for the purposes of this app, with the primary example being moving the application's user interface from Python to JavaScript & HTML and the creation of a database. The team utilized NVIDIA NeMo as the tool to be able to transcribe Air Traffic Control (ATC) communications data. NVIDIA NeMo is being used during the first semester as it has been a consistent model, offering a word error rate of approximately 30% given current training models. Currently, the second team working on this project is developing and training another ASR model, known as Kaldi. This model, currently as a work-in-progress, is not able to be fully implemented into the ASR Web Application. Kaldi will eventually replace the NVIDIA NeMo model once it is capable of being fully implemented and has a better word error rate than its predecessor.

The ASR Web Application is an on-going development process that tracks nearby aircraft surrounding Daytona Beach, Florida. This application is able to track flight paths and moments of ATC communication. The goal of this application is to allow for the ASR model, being either NeMo or Kaldi, to be implemented allowing for the direct transcription of ATC communications to the web app. This application will additionally have a tied database containing all of the relevant information of the application as well as audio files and transcribed text files that NeMo will utilize and output.

### 1.2.1 System Overview

There are two features that are in this system:

*1. ASR Web Application:*

This gives the user the ability to switch between a series of VFR Aeronautical maps and a geographic map. These maps include a zoom-in and zoom-out function with the ability to click on the plane icons, providing details about the associated aircraft such as callsign, position, and heading on the left side of the interface. With the updating tracking, the user has the ability to click and track all the aircrafts or airports. When an aircraft is clicked, it will display the flight path of the plane along with multiple circles showing where the aircraft communicated. When an airport is clicked on, a menu on the left side of the interface will display the information about the airport and a drop down menu of any available live radio frequencies.

*2. Transcription:*

Communications between ATC and an aircraft will be stored, along with the transcription of this audio once complete, will be stored in the communications database. From this database, the transcriptions and communication data, such as the longitude and latitude where the communication took place, will be able to be displayed to the user.

*Figure 1* depicts the use-case diagram for the NeMo system. It shows in more detail what interacts within the system to provide the features of NeMo. While there are different types of users intended to interact with the NeMo system, they currently are able to access the same functionalities, so they have been simplified into one user for the sake of the diagram. The user is able to *View Live Maps*. This includes a geographic map as the default map setting. This use case also includes an aeronautical map as an optional map selection. The following APIs interact with this case internally: OpenSky, VFRMap, and OpenStreetMap.

Users are able to *Listen to Live ATC Audio*. Internally, Near Aero provides the audio. Similarly, users are able to *Transcribe Live Audio*. Near Aero provides the audio, the NVIDIA NeMo application transcribes the audio which is then saved to the Communications database, which is then displayed to the user.

As more functionalities are added to the system, and those functionalities start to vary between user types, the Use-Case Diagram will be updated accordingly.

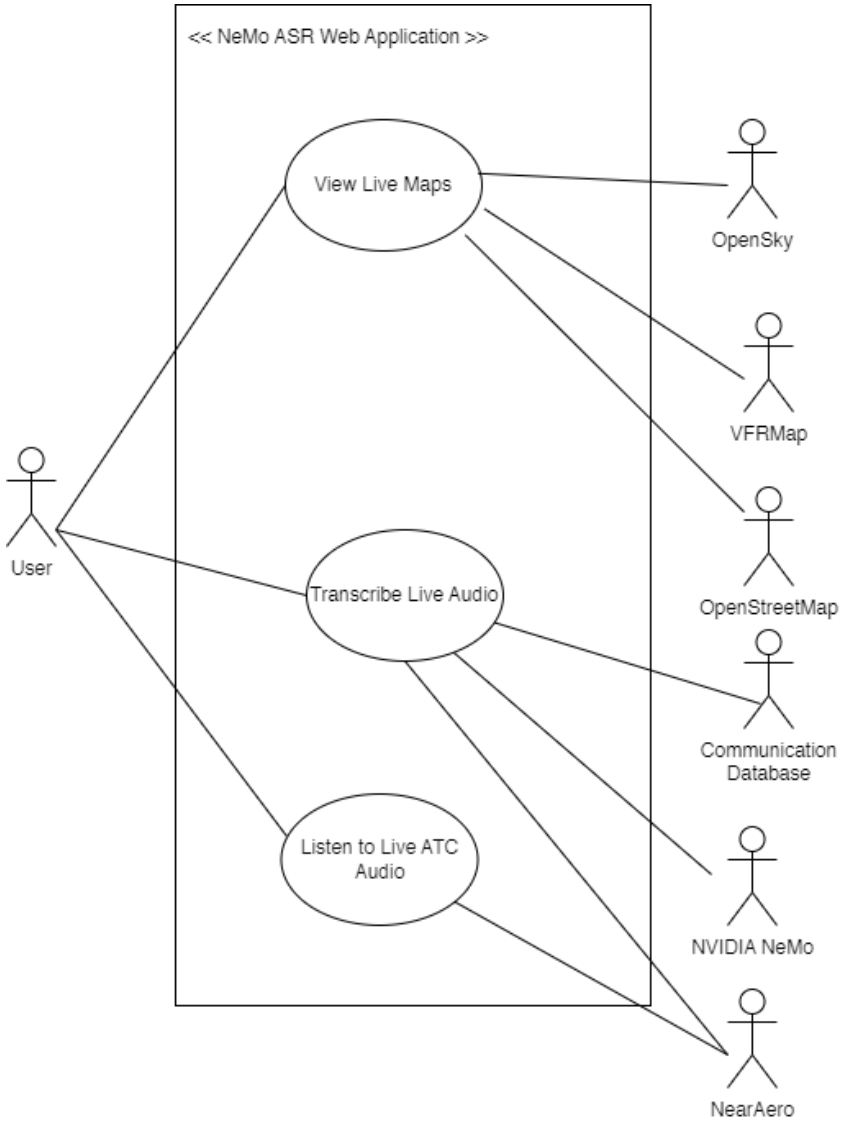


*Figure 1: Use-Case Diagram for NeMo system.*

### 1.2.2 Design Constraints

One of the key assumptions we made is that the data obtained from the APIs such as Opensky and the tilesets for the maps from OpenStreetMap and VFRMap are sufficiently correct information to be displayed to the user and that the audio from Near Aero and other sources is the correct audio communication that is to be translated to text. We also are aware of current limitations regarding ASR technology still having a sufficiently high WER that may result in incorrect and or unreadable text. We concluded that based on the ASR limitations the end user should be made aware of these limitations and display the limitations on the “about” page of the website.

Another key constraint while designing and creating this web application is the requirement for developers, testers, and the machine acting as a host to have a NVIDIA GPU in order to

run NVIDIA NeMo for the ASR models even though we are not creating or training any models.

Additionally, when developing this web application there was a tradeoff between getting certain systems working and getting them working efficiently for the majority of end users. This includes the optimization and balance between taking more memory and calling data from sources such as the database or an API. Many of the APIs have limitations on how often they can be called.

### 1.2.3 Future Contingencies

The primary contingency that could arise would be implementing Team Kaldi's model into Team NeMo's code. A potential way to overcome this would be waiting until Team Kaldi has a usable and stable model that has support to take in a stream of audio and output text. Currently the application will use NVIDIA NeMo running a trained model specifically for ATC Communications. Team Kaldi's model will either replace or work in conjunction with the NVIDIA NeMo model when ready.

Another contingency that will likely arise is speech-to-text transcriptions or other data not being fully accurate, resulting in incorrect text and/or false information being displayed to the end user. Due to current limitations with Automatic Speech Recognition (ASR) technology a low or near-zero Word Error Rate (WER) is difficult to achieve. A good ASR Model using current technology can only achieve a WER of about 0.5 - 0.10 (about 5% to 10% error rate). In the current implementation of NVIDIA NeMo, the WER is approximately 30%. In the application "about" page end users will find information that transcriptions and data including aircraft type, speed, altitude, and position pulled from third-party APIs are not guaranteed to be 100% correct, and should not be used as the only source of information for a pilot. The FAA and official audio channels should always be used as the primary source of information.

Another future contingency would be the airplane icons not showing up because of the Opensky API not being able to pull requests. Every 30 seconds the Opensky API should be able to pull a new request to update the position of the airplane icons. Sometimes the pull request fails because Opensky is unable to retrieve the information. This will cause problems in new changes to the icons of the airplanes. One way to overcome this challenge would be to freeze the data of the aircrafts in order to keep them from disappearing. This will help with changing the design of the airplanes.

## 1.3 Document Organization

The System Design Document starts with the introduction section describing in full detail what the project is about and what the purpose of this project is about. After the general overview of the project is discussed, the document goes into a more in depth view of the System Architecture. Then the document goes over the human-machine interface, detailed design, external interfaces, and system integrity controls.

## 1.4 Project References


Team NeMo 2023 - 2024. (2023). *System Requirements Specification Document.* https://github.com/maeganlucas/CS490-ATC/blob/main/NeMo/Documents/System%20Requirements%20Specification%20-%20NeMo%20-%20Final.docx.pdf

Team NeMo 2023 - 2024. (2023). System Test Plan. https://github.com/maeganlucas/CS490-ATC/blob/main/NeMo/System%20Test%20Plan%20-%20NeMo%20%20Final.pdf

LeafletJS. (2023). *LeafletJS Reference.* https://leafletjs.com/reference.html

Flask. (2023). *Flask User's Guide.* https://flask.palletsprojects.com/en/3.0.x/

Opensky Network. (2023). *Opensky Network API Documentation.* https://openskynetwork.GitHub.io/opensky-api/index.html

Sphinx. (2023). *Sphinx Documentation.* https://www.sphinx-doc.org/en/master/

GitHub. (2023). *NVIDIA NeMo GitHub.* https://GitHub.com/NVIDIA/NeMo

NVIDIA. (2023). *NVIDIA NeMo tutorials.* https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/starthere/tutorials.html

Team NeMo 2023 - 2024. (2023). *Team NeMo documentation GitHub.* https://GitHub.com/maeganlucas/CS490-ATC

Team NeMo 2023- 2024. (2023). *Team NeMo active GitHub.* https://GitHub.com/TheCreepOfWar/asr-webapp

Team NeMo 2022 - 2023. (2023). Team previous NeMo team GitHub. https://GitHub.com/Burnetb8/Senior-Capstone

### 1.5 Glossary

| # | Term | Abbreviation | Description |
|---|---|---|---|
| 1 | Air Traffic Control | ATC | Refers to tower communication to or from aircraft. |
| 2 | Automatic Speech Recognition | ASR | Software that recognizes speech patterns. |
| 3 | Word Error Rate | WER | Refers to the error rate in ASR technology typically on a scale of 0 to 1 but can exceed 1 depending on sample comparisons. |
| 4 | Federal Aviation Administration | FAA | United States government agency that oversees and regulates civilian aviation in the United States. |
| 5 | NVIDIA NeMo | | NVIDIA NeMo is a generative AI framework that allows for conversational AI to function and transcribe ATC communications within the web application. |
| 6 | Graphical Processing Unit | GPU | A hardware component used to improve computer graphics. |
| 7 | International Civil Aviation Organization | ICAO | The special agency of the United Nations that governs aviation. |
| 8 | International Air Transportation Association | IATA | An aviation trade association. |

## 2 SYSTEM ARCHITECTURE

### 2.1 System Hardware Architecture

The web application will be hosted on a system or a server with a NVIDIA GPU. The NeMo ASR model is unable to transcribe ATC Communications without a NVIDIA GPU. However, due to the web application being hosted by a server, the end user will not have to meet the specific GPU requirement and will only need to have a device that can connect to the internet.

Currently, there is no ability to access the application through a mobile version. The end user (client) does not require any special hardware to load the application.

In the case that a user is running the application on a local system, a NVIDIA GPU will be required only for the ability to transcribe ATC Communications data.

Despite this, the user is still capable of tracking aircraft across the map and has the ability to listen to live ATC communications.

## 2.2 System Software Architecture

A user will interact with the NeMo system through the frontend of the application. This will make the appropriate requests to the backend portion of the system.

For visual maps, the backend will make requests to the respective APIs, OpenStreetMap for geographic maps and VFRMap for aeronautical maps, and display them to the user. To display the active planes, the backend will make requests as necessary to the OpenSky API to determine plane data. The system will then complete the necessary logic to add a plane icon of the correct type to the map and display it to the user. The system will also query the Communications database when an airplane is selected to pull the waypoint data for the specified plane for display.

The main system software being used in this is the NVIDIA NeMo ASR models which are using Python. NVIDIA NeMo is focused on Automatic Speech Recognition which is used by Python's machine learning modules. Using NVIDIA NeMo, transcriptions of live Air Traffic Control communications will be possible. This will occur when a user selects a transcription button on the frontend. From there, the backend portion of the system will make a query from the database to retrieve the live audio recording. It will be passed into NVIDIA NeMo to be transcribed and this will be added to the database entry for the audio. This entry will then be displayed to the user. With current training and implementation, the transcription may have up to approximately a 30% word error rate (WER).

The NeMo architecture can be seen in Figure *2* below.

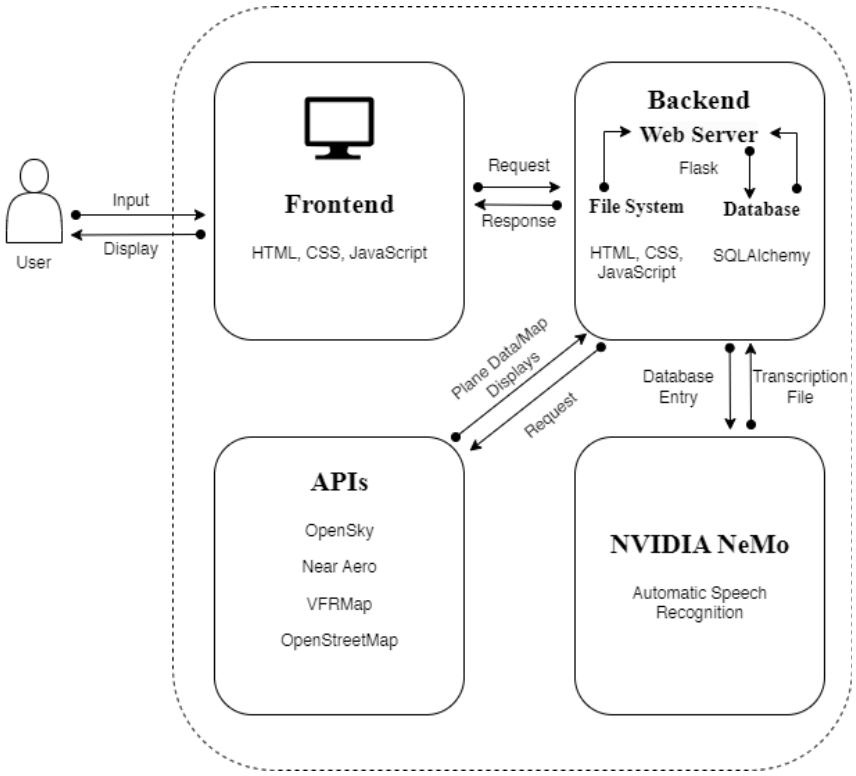


*Figure 2: System Architecture Diagram for NeMo.*

## 2.3 Internal Communications Architecture

The internal communication architecture for the NeMo web application is depicted in *Figure 3*. The user of the system represents one source of the diagram as they input airplane, airport, and button selections into the system. Other sources include the various APIs that the application utilizes. OpenSky is a source for the live plane data. OpenStreetMap and VFRMap are the sources for the geographic and aeronautical map tiles respectively. Near Aero provides the live ATC audios to the system. The Communication Database developed by the team can also be considered a source as plane and audio data are queried from the entries within the database into the system to be transcribed by NVIDIA NeMo. The database will also provide the written transcriptions back to the system to be displayed to the user.

The sinks for this system are the Communication Database and the Display to the user. For the Communication Database, transcriptions for given audios are then saved to their respective entry within the database. The Display receives all visual aspects such as airplane and airport icons, airplane and airport details, flight paths, transcriptions, and waypoint indicators.

More detailed diagrams of how data flows within the system can be seen in section 4.3, *Internal Communications Detailed Design*.

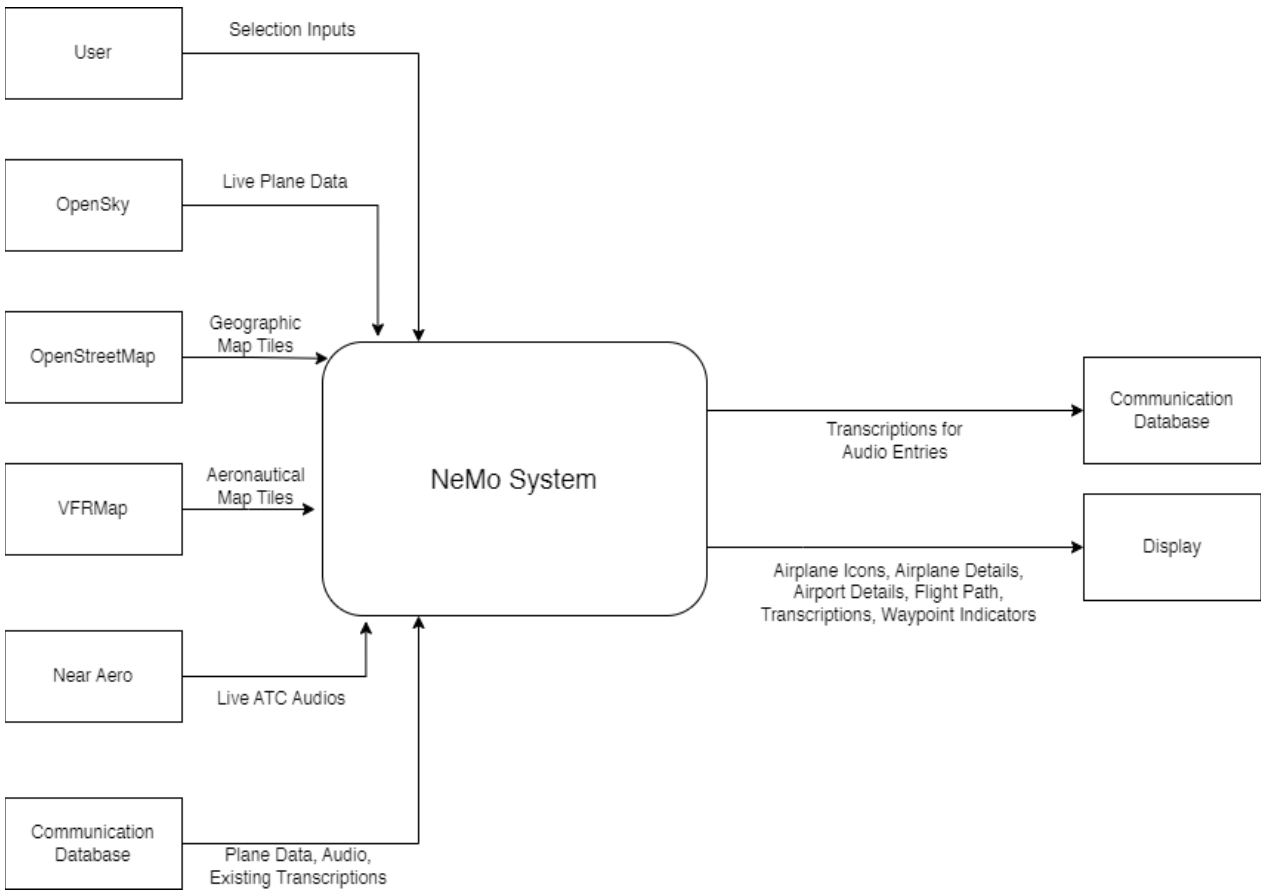


*Figure 3: Context Diagram for NeMo system.*

## 3 HUMAN-MACHINE INTERFACE

### 3.1 Inputs

The main form of input into the system is the live audio feeds in the form of streamed .mp3 files which will later be converted to .wav files. The live audio feeds will be pulled from various online sources such as LiveATC and Near Aero. The user will give input to the system with mouse interaction on the map. Using buttons in a drop-down menu, the user can choose between a geographical map, sourced from Openstreetmap.org, and a series of aeronautical maps, from VFRMap.com. The user will be able to select which aircraft's flight data they want to access, by selecting that aircraft's icon on the aeronautical map.

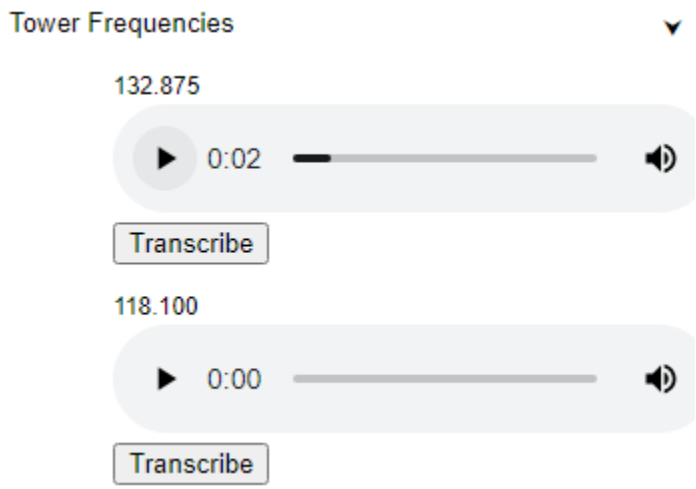


*Figure 4: Example of the users option to listen to live ATC communications & be able to transcribe live audio*

### 3.2 Outputs

The system will output a Flask-based website containing a geographical map and a series of aeronautical charts, determined through user input, with icons displaying the locations of airplanes and airports. This website can be seen depicted in *Figures 5 and 6* When one of these icons is clicked, an information popup on the left side of the interface will be displayed with the selected airplane or airport's information and any available live audio frequencies. For airplanes, this information will include the airport's International Civil Aviation Organization (ICAO) 24-bit address, callsign, origin country, time of last position report, last contact time, position (longitude and latitude), geometric altitude, velocity, heading, vertical rate, squawk, position source, category, and whether the airplane is on the ground. This display can be seen in *Figure 7*. For airports this information will include the airport's identifier, name, position (longitude and latitude), elevation, region, municipality, GPS code, International Air Transportation Association (IATA) code, local code, and website. This can be seen in *Figure 8*.

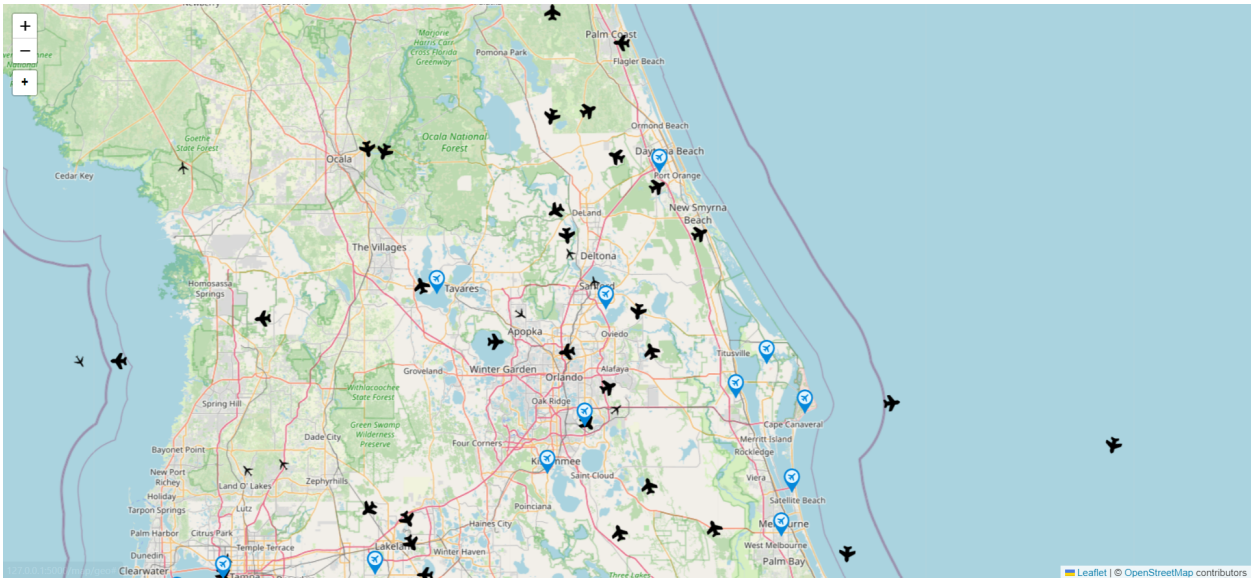


*Figure 5: User interface shown to users upon opening web application with geographic map display.*

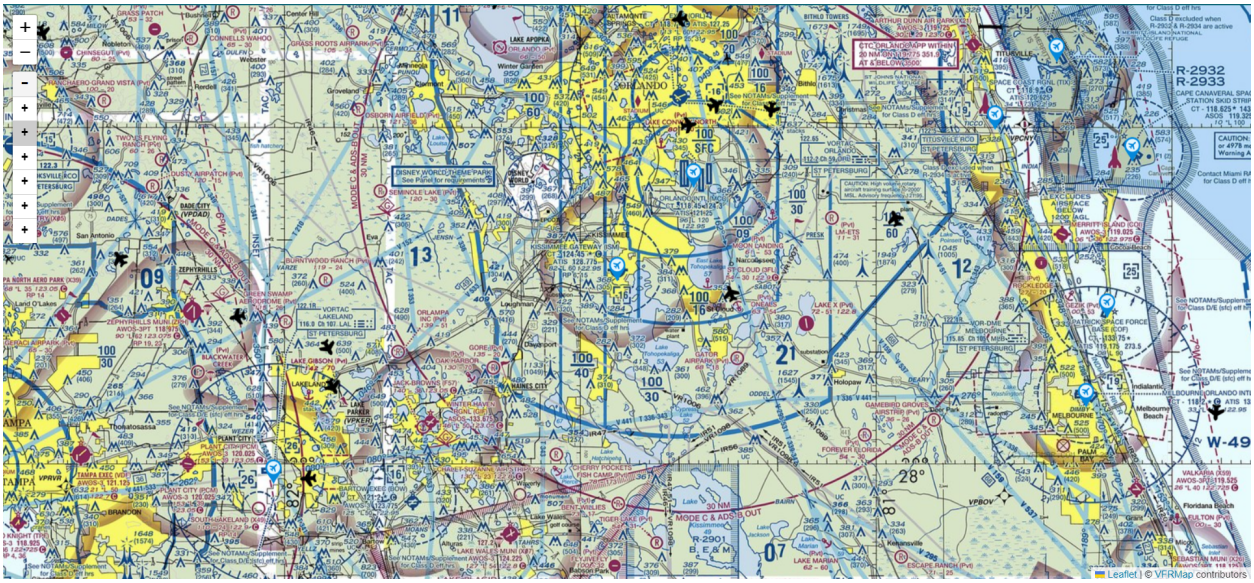


*Figure 6: User interface shown to users when aeronautical map option is selected.*

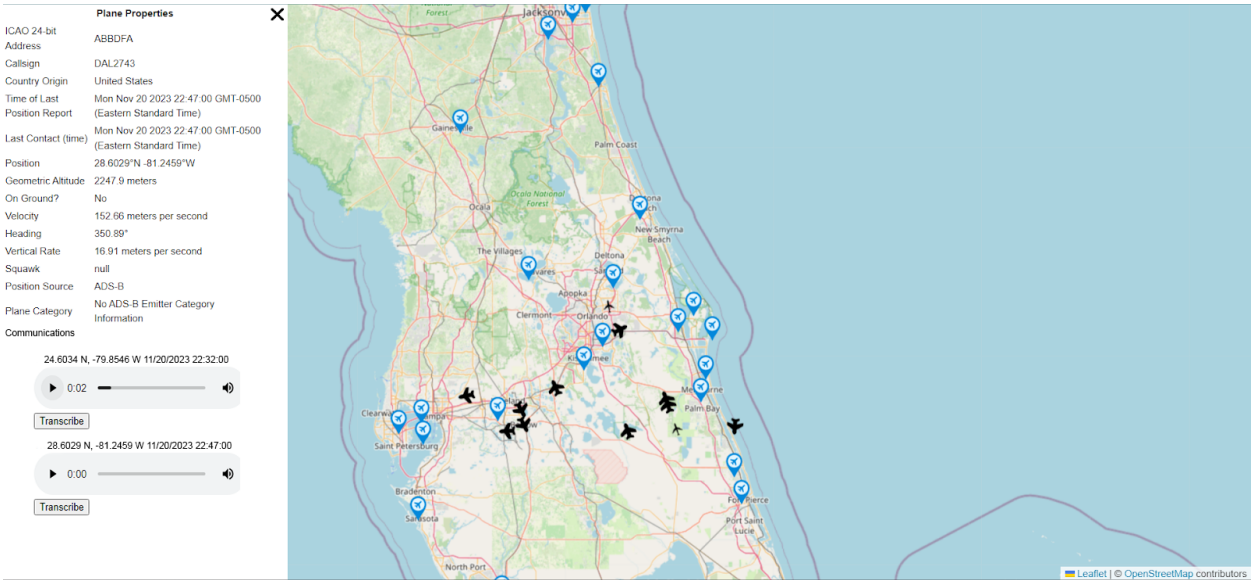


*Figure 7: Plane properties display. Note: Communications portion is a mockup.*

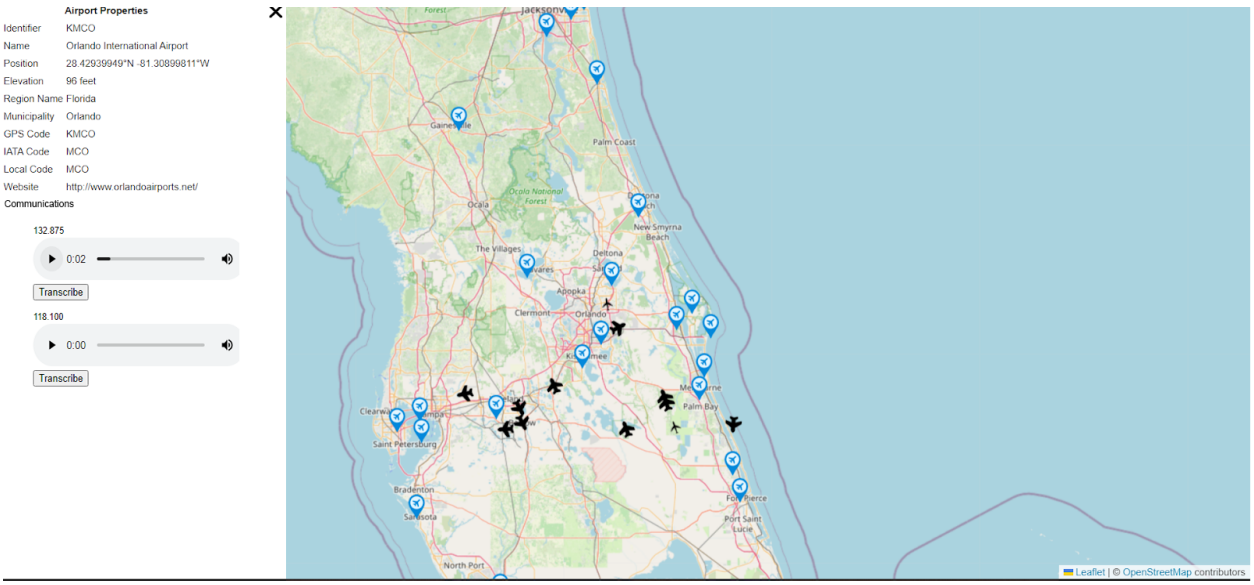


*Figure 8: Airport properties display. Note: Communications portion is a mockup.*

# 4 DETAILED DESIGN

## 4.1 Hardware Detailed Design

No hardware will be designed by the NeMo group. All required hardware is already embedded in an average laptop or personal computer to be able to host or run the web application. However, as previously stated, a NVIDIA GPU will be needed on a running server or local machine acting as the host for the project, otherwise the speech-to-text transcription part of this project will not function.

## 4.2 Software Detailed Design

***Python Version 9***, ***Pandas Version 2.0.1***, and ***OpenPyxl Version 3.1.2*** are required for the backend to operate.

***Flask*** web framework for creation and framework of the application, managing and creating templates, pages, communication between front end HTML interface and python and JavaScript scripts which do backend processing and communication with the other APIs.

***OpenskyAPI*** is a network API used to obtain live aircraft data. Airplanes seen by the OpenSky Network obtain their flight information via ADS-B and Mode S messages. Utilizing this API, the application is able to track data including: icao24, callsign, origin country, time position, last contact, longitude, latitude, geospatial altitude, ground status, velocity, true track (heading), vertical rate, squawk, position source, and category of the aircraft. Within the application, it is possible to use this API to derive aircraft type and display it to the user via a specific icon pertaining to that aircraft type. This allows the application to determine the difference between a smaller civilian aircraft such as a Cessena or a long-haul 4-engine heavy aircraft such as a commercial Airbus A380.

***LeafletJS*** is an open source JavaScript library that is used to create, display, and control interactive maps. This includes the creation of the icons on the map, the base which tilesets are connected to, and the zoom controls. In this application, LeafletJS allows for the creation of two separate maps. This being the geographic map as well as the aeronautical map. This tool additionally is what allows for a user to be able to click on specific icons to be able to then receive the relevant flight information and listen to or transcribe ATC communications.

***OpenStreetMap*** is used for the tileset (set of images that comprise the map) for the geographical map display & VFRMap is used for the tileset for the various aeronautical maps.

***NVIDIA NeMo*** is an end-to-end, cloud based framework to develop and train generative AI models. This tool will be utilized within this project to transcribe ATC communications data and output the data to the relevant user. NeMo is an early stepping stone in this project, as there is no requirement from the team to develop a model or continuously train the NeMo model as it will not be a final solution for the RTube project. NeMo will be utilized to ensure the capabilities and early testing for ASR transcription of ATC communications. This project however will take predefined classes and functions NVIDIA NeMo provides to take in and process audio files and streams and provide a text output. When the user decides to transcribe live ATC transmissions (from an aircraft currently in flight), transcribable audio transmissions will be livestreamed to NeMo and segmented into predetermined audio fragments (for example, 30 second data chunks). To ensure no communication data is lost or distorted, each audio fragment will overlap with its prior and subsequent audio data chunk (for example, 5 second overlap per data chunk). Python

version 3.9 or 3.10 and Pytorch version 1.13.1 or 2.0+ are the prerequisites in order to run NVIDIA NeMo along with Ubuntu Linux.

***ASR Web-App Database*** contains recorded ATC audio communication data, as-well-as relevant information associated with each ATC communication. This relevant information includes transmission time-stamp, callsign, and related Opensky API flight data. This database will additionally connect audio files obtained by LiveATC and store them to then be able to be queried and transcribed by NeMo. Once transcribed, the output can then be stored back into the database and attached to its relevant callsign and the time that the communication happened. The web application also ustilizes ***SQLAlchemy*** to enable interaction between the database, Flask, and Python.

The ***ASR Web Application*** is the primary host of the project. This web-based application utilizes Python, Javascript, HTML, and CSS files to create an online environment that the application will eventually be hosted on. These files allow for the manipulation of every relevant information that is displayed through the application. Once fully implemented, NeMo will be attached to the web application and allow for the direct transcription of live ATC communications.

**Use Cases:**

I. Use Case 1: Opening Web Application
- Description: The user initially opens the web application.
- Actors: User, OpenStreetMap, OpenSky
- Precondition: No preconditions.
- Trigger condition: The web application is loaded.
- Flow of Events
    1. The user selects the geographic map button from the map selection drop-down menu.
    2. The system request the map tiles from OpenStreetMap (ALT 1).
    3. The system displays the geographic map.
    4. The system requests the airport data from internal sources.
    5. The system displays the airport markers.
    6. The system requests airplane data from OpenSky (ALT 2).
    7. The system displays airplane icons.
    8. End of use case.
- Alternative Cases
    - ALT 1. Step 2: OpenStreetMap does not return data.
        - 2.1. System throws internal error.
        - 2.2. System sends another request to OpenStreetMap.
        - 2.3 Return to step 3.
    - ALT 2. Step 6: OpenSky does not return data.
        - 6.1. System throws internal error.
        - 6.2. Return to step 6.

II. Use Case 2: Select Geographic Map

- Description: The user selects the geographic map button from the map selection drop-down menu to display the geographic map.
- Actors: User, OpenStreetMap, OpenSky
- Precondition: The user must be on the aeronautical map display.
- Trigger condition: The geographic map button is selected from the map selection drop-down menu.
- Flow of Events
    1. The user selects the geographic map button from the map selection drop-down menu.
    2. The system request the map tiles from OpenStreetMap (ALT 1).
    3. The system displays the geographic map.
    4. The system requests the airport data from internal sources.
    5. The system displays the airport markers.
    6. The system requests airplane data from OpenSky (ALT 2).
    7. The system displays airplane icons.
    8. End of use case.
- Alternative Cases
    - ALT 1. Step 2: OpenStreetMap does not return data.
        - 2.1 System throws internal error.
        - 2.2 System sends another request to OpenStreetMap.
        - 2.3 Return to step 3.
    - ALT 2. Step 6: OpenSky does not return data.
        - 6.1 System throws internal error.
        - 6.2 Return to step 6.

III. Use Case 3: Select Aeronautical Map

- Description: The user selects the aeronautical map button from the map selection drop-down menu to display the aeronautical map.
- Actors: User, VFRMap, OpenSky
- Precondition: The user must be on the geographic map display.
- Trigger condition: The aeronautical map button is selected from the map selection drop-down menu.
- Flow of Events
    1. The user selects the aeronautical map button from the map selection drop-down menu.
    2. The system request the map tiles from VFRMap (ALT 1).
    3. The system displays the aeronautical map.
    4. The system requests the airport data from internal sources.
    5. The system displays the airport markers.
    6. The system requests airplane data from OpenSky (ALT 2).
    7. The system displays airplane icons.
    8. End of use case.
- Alternative Cases
    - ALT 1. Step 2: VFRMap does not return data.
        - 2.1 System throws internal error.
        - 2.2 System sends another request to VFRMap.

2.3 Return to step 3.

- ALT 2. Step 6: OpenSky does not return data.
  - 6.1 System throws internal error.
  - 6.2 Return to step 6.

IV. Use Case 4: Map Reposition

- Description: The user repositions the map by click and drag or zoom which can lead to UC 5.
- Actors: User, OpenSky
- Precondition: Web application must be running.
- Trigger condition: The user repositions the map by click and drag or zoom.
- Flow of Events
  1. The user clicks and drags map to new location or zooms in or out (ALT 1).
  2. The system determines the new longitude and latitude of the map bounds.
  3. The system requests the airport data from internal sources.
  4. The system displays the airport markers.
  5. The system requests airplane data from OpenSky (ALT 2).
  6. The system displays airplane icons.
  7. End of use case.
- Alternative Cases
  - ALT 1. Step 1: User zooms out to zoom limit
    - 1.1 UC 5
    - 1.2 Return to step 2.
  - ALT 2. Step 6: OpenSky does not return data.
    - 6.1 System throws internal error.
    - 6.2 Return to step 6.

V. Use Case 5: Zoom Blur

- Description: The user zooms past the zoom limit for the map display.
- Actors: User, Display, OpenSky
- Precondition: The web application must be running.
- Trigger condition: The user zooms out past zoom limit.
- Flow of Events
  1. The user zooms out past the zoom limit.
  2. The system pauses requests.
  3. The system blurs map.
  4. The display shows message to zoom in.
  5. End of use case.
- Alternative Cases

VI. Use Case 6: Live ATC Audio - Plane

- Description: The user wants to listen to live ATC audio from a specific flight which can lead to UC 8.
- Actors: User, Communcations Database, OpenAero, OpenSky

- Precondition: The web application must be running.
- Trigger condition: The user clicks an airplane icon.
- Flow of Events
    1. User selects airplane icon.
    2. The system determines airplane data from OpenSky results.
    3. The system queries communications database for audio with specific callsign. (ALT 1).
    4. Airplane data and communications are displayed to the user on a popup with transcribe buttons for each audio (ALT 2).
    5. End of use case.
- Alternative Cases
    - ALT 1. Step 2: The database query returns no data.
        - 2.1 Data is left empty.
        - 2.2 Return to step 3.
    - ALT 2. Step 3: User selects transcribe button.
        - 3.1 UC 8
        - 3.2 Return to step 4.

VII. Use Case 7: Live ATC Audio - Airport

- Description: The user wants to listen to live ATC audio from an airport which can lead to UC 8.
- Actors: User, Communcations Database, OpenAero, OpenSky
- Precondition: The web application must be running.
- Trigger condition: The user clicks an airport icon.
- Flow of Events
    1. User selects airport icon.
    2. The system determines airport data from communications database.
    3. The system queries communications database for audio with specific callsign. (ALT 1).
    4. Airport data and communications are displayed to the user on a popup with transcribe buttons for each audio (ALT 2).
    5. End of use case.
- Alternative Cases
    - ALT 1. Step 2: The database query returns no data.
        - 2.1 Data is left empty.
        - 2.2 Return to step 4.
    - ALT 2. Step 3: User selects transcribe button.
        - 3.1 UC 8
        - 3.2 Database is queried again for callsign.
        - 3.3 Transcription is displayed under audio.
        - 3.4 End of use case.

VIII. Use Case 8: Audio Transcription

- Description: The user wants to transcrie live ATC audio.
- Actors: User, Communcations Database, NVIDIA NeMo
- Precondition: Audio must be .WAV file.

- Trigger condition: The user clicks selects transcription button next to communication audio.
- Flow of Events
    1. User selects transcribe button next to communication audio.
    2. The system passes audio file from database to NVIDIA NeMo (ALT 1).
    3. NVIDIA NeMo transcribes data and returns a .txt file. (ALT 2)
    4. Database stores .txt file to specific time and callsign entry.
- Alternative Cases
    - ALT 1. Step 2: Audio not a .WAV file.
        - 2.1 Audio is converted to .WAV file.
        - 2.2 The system passes audio file to NVIDIA NeMo.
        - 2.3 Return to step 3.
    - ALT 2. Step 3: NVIDIA NeMo does not return a file.
        - 3.1 System throws internal error.
        - 3.2 End of use case.

### 4.3 Internal Communications Detailed Design

The NeMo website interfaces with the OpenStreetMap, VFRMap, and OpenSky APIs for maps and airspace information. Near Aero provides all live ATC communication. A transcription of this audio will be displayed to the user after being passed through the NVIDIA NeMo ASR model. A database will be created to hold all communication data that will then be queried based on user input to pull specific airplane or airport data, audios and audio data, and transcriptions. Below are a level zero (*Figure 5*) and level one Data Flow Diagram (*Figure 5*) representing the flow of data within the ASR web application.

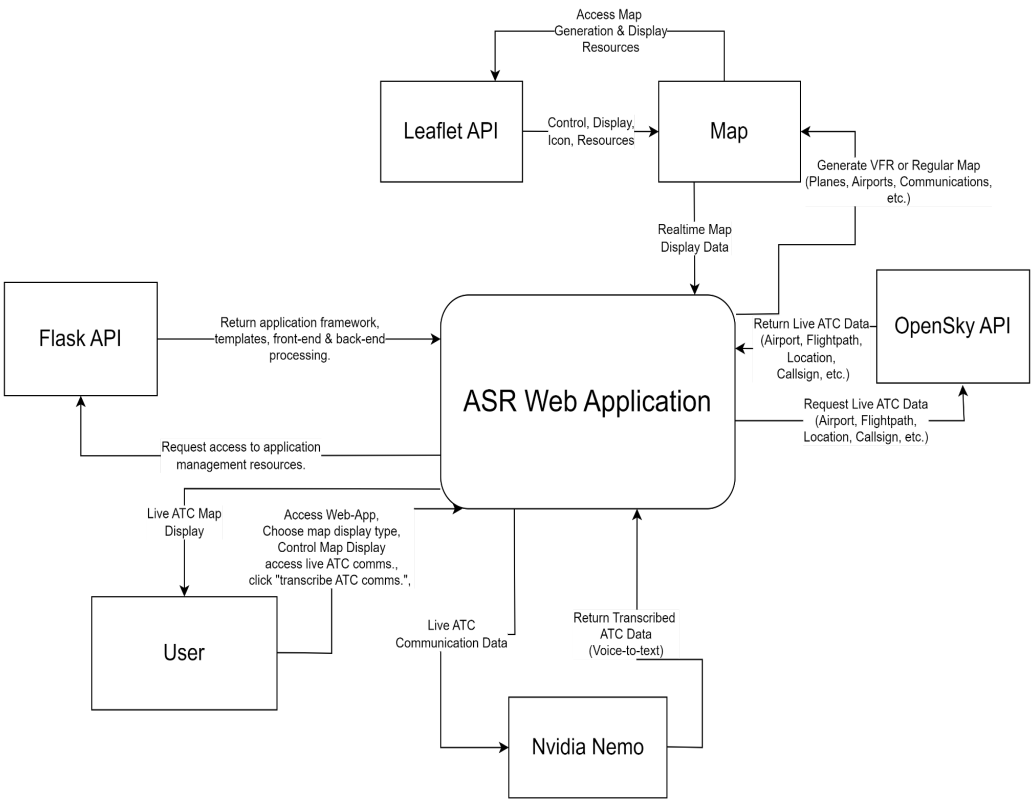


*Figure 9: The above diagram is a high level view detailing the data communications between the "components" of our system.*

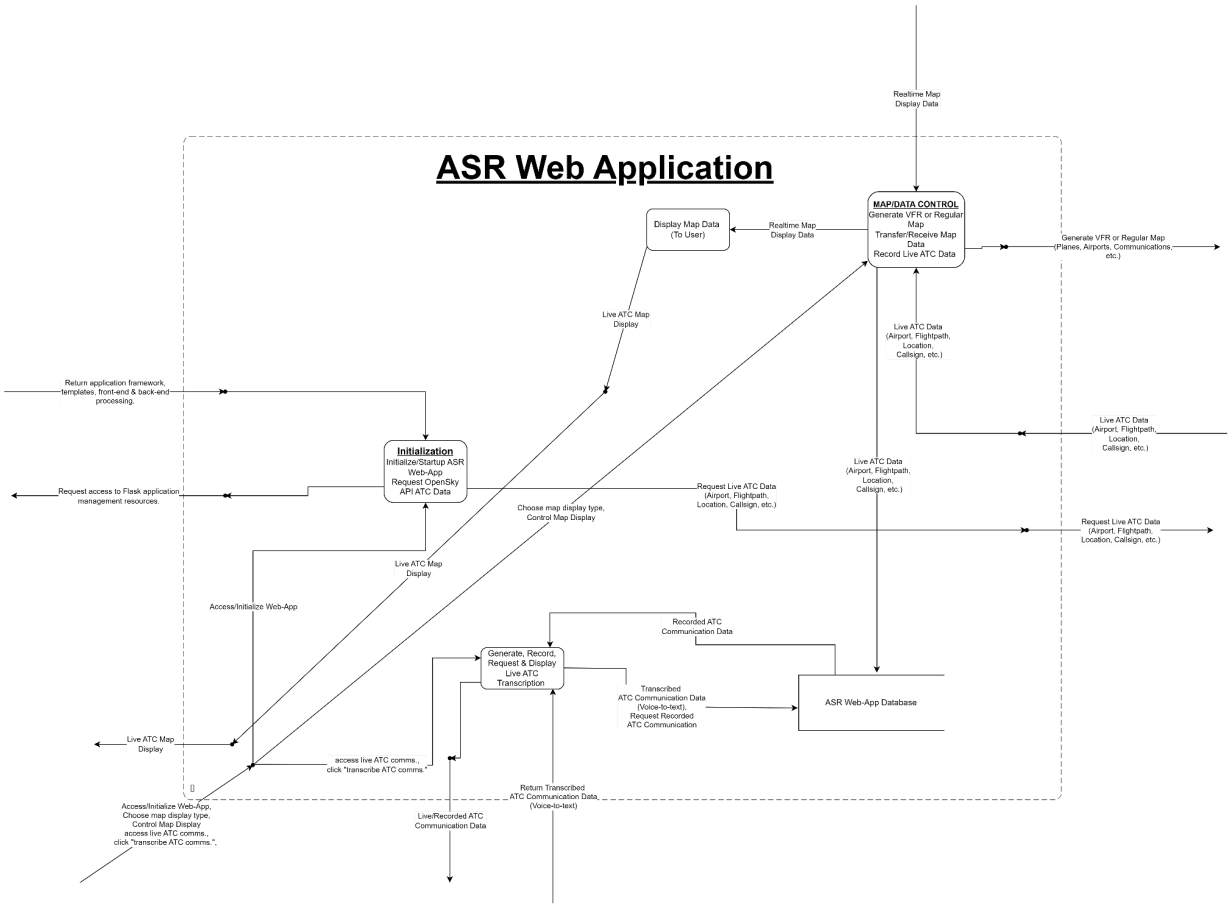


*Figure 10: The above diagram is a level 1 Data Flow Diagram detailing the processes and flow of data within the ASR Web Application (please zoom for clarity or see attached documentation).*

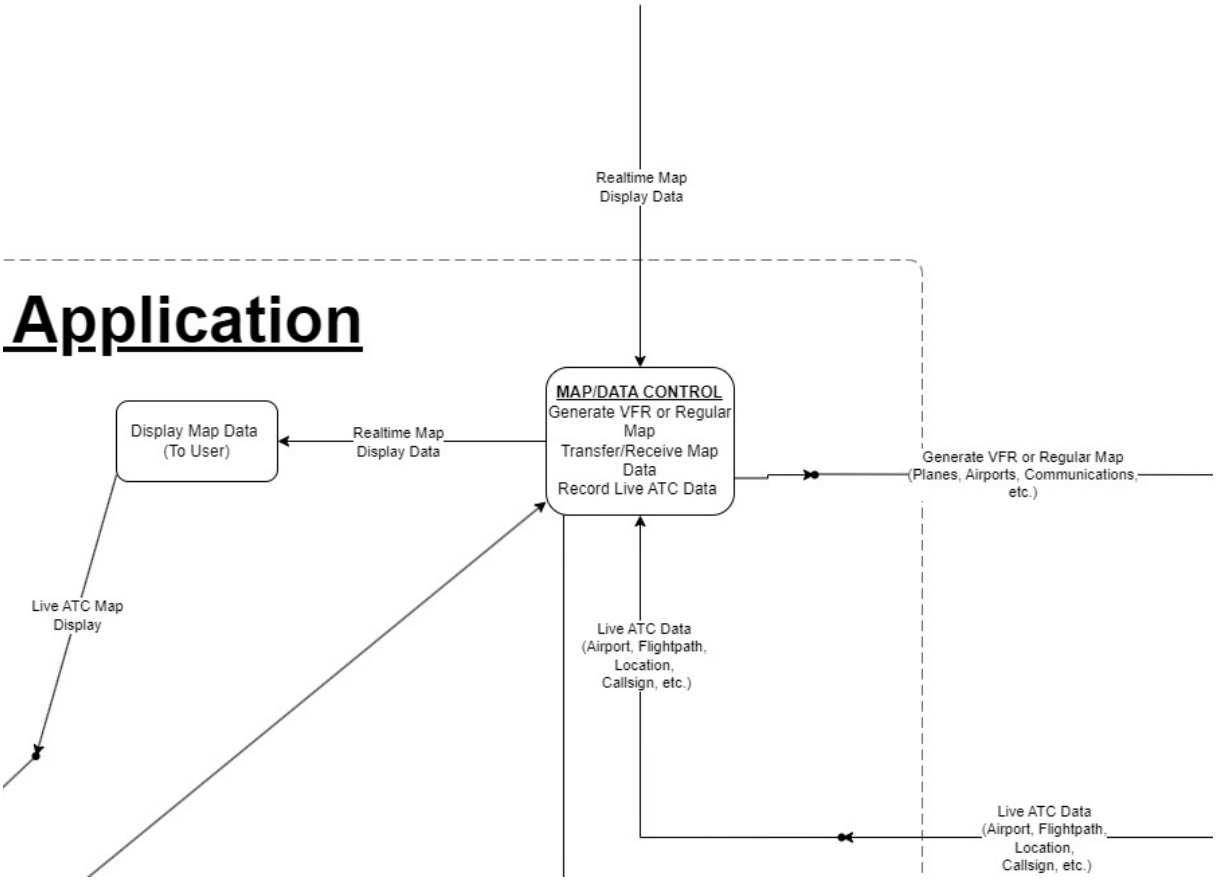


*Figure 11: The above diagram is an expanded view of the top-right section of Figure 10 [see pg.18]. (Focuses on data flow for ATC data and Map/Data Control processes).*

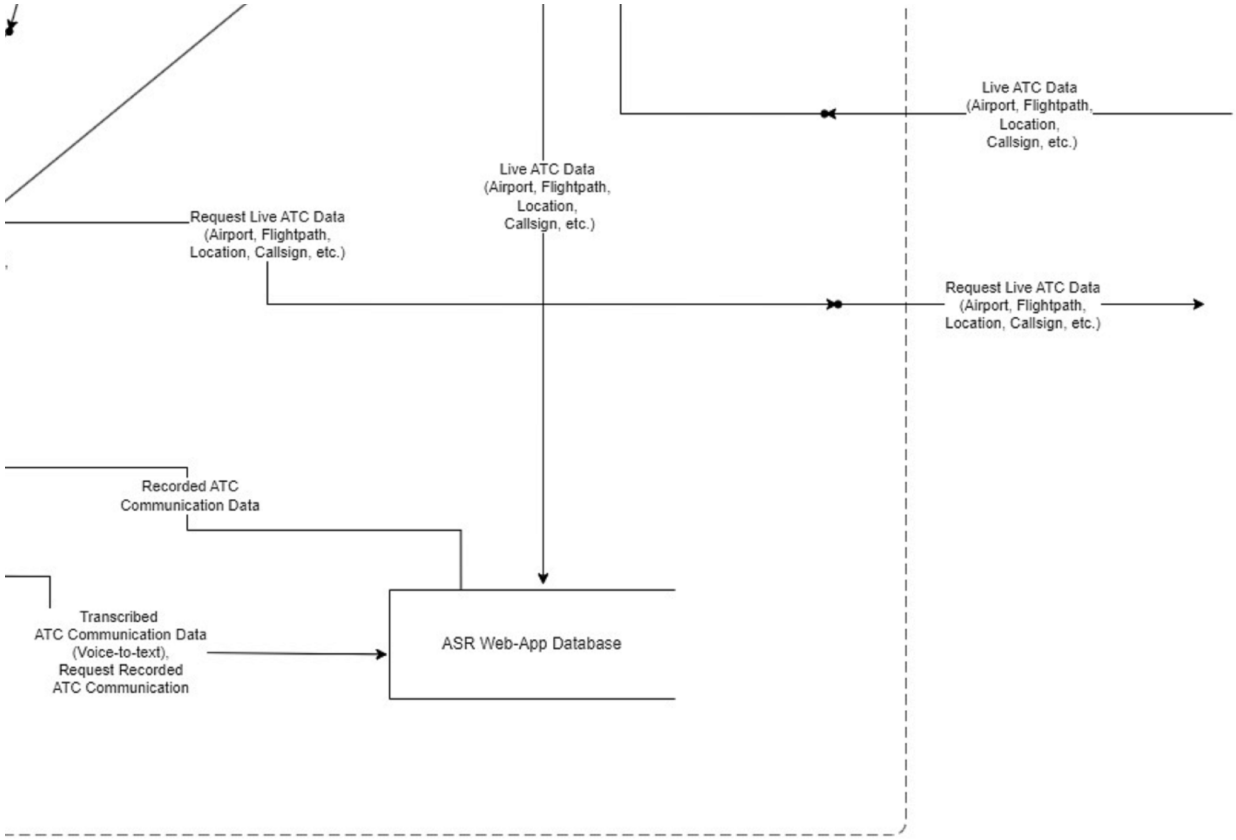


*Figure 12: The above diagram is an expanded view of the bottom-right section of Figure 10 [see pg.18]. (Focuses on data flow for ATC data, audio communication data, and the ASR Web-App Database).*

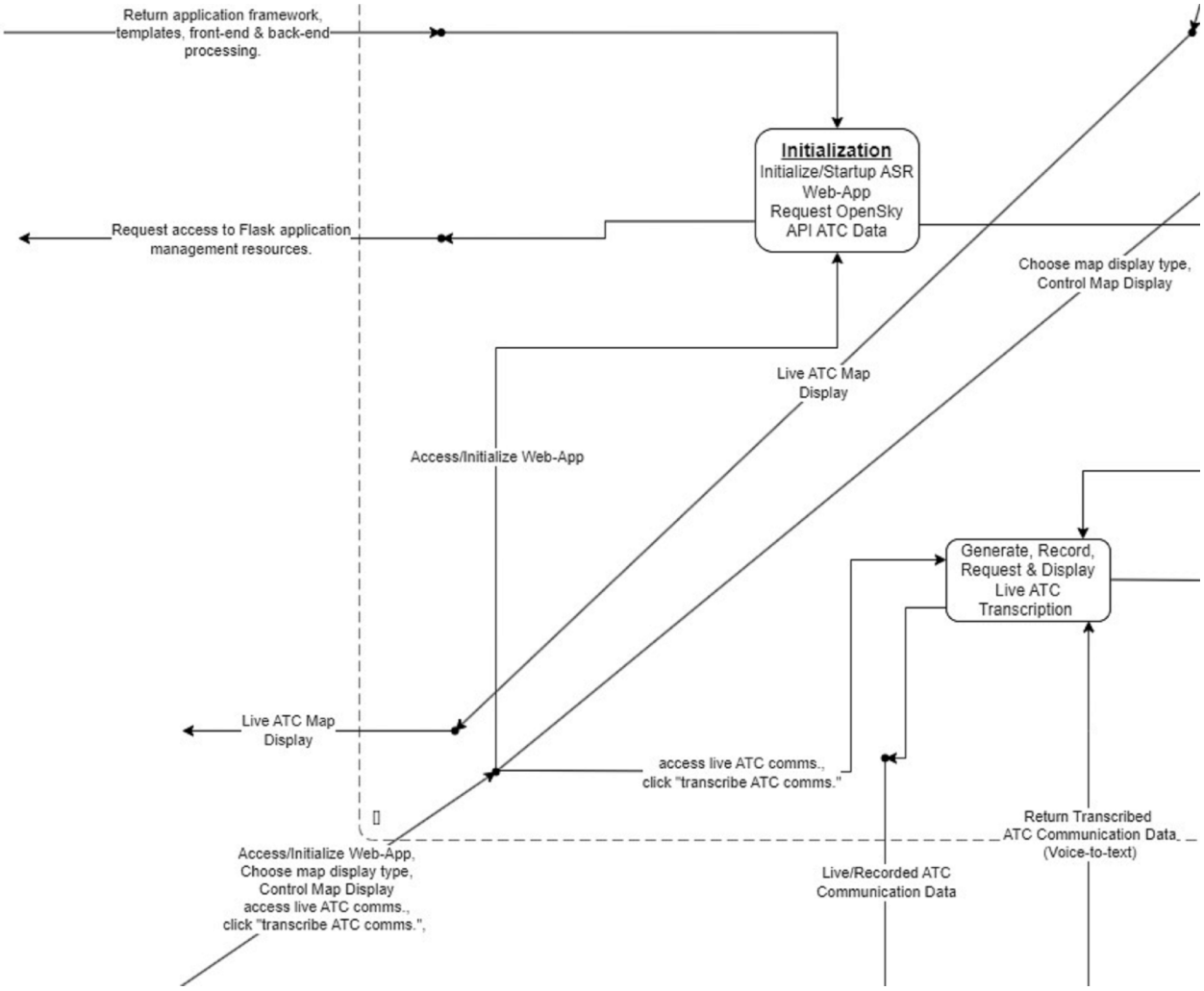


*Figure 13: The above diagram is an expanded view of the bottom-left section of Figure 10 [see pg. 18]. (Focuses on data flow for ASR Web-App Initialization, audio communication data, and NeMo transcript data generation).*

**Internal ASR-Web Application Processes:**

- Map/Data Control - Handles the transfer of vital map data. Directly Interfaces with Map source and indirectly interfaces with the Leaflet API.
- Display Map Data - Displays readable map data to user.
- Generate, Record, Request & Display Live ATC Transcription - Sends and receives live ATC voice data (to be transcribed) to Nvidia Nemo. Sends newly transcribed data to the internal ASR Web-App database. In addition, can request previously recorded voice communications from the internal ASR Web-App database
- Initialization - Contacts the Flask API and OpenSky API and requests required resources for NeMo website application initialization.
- ASR Web-App Database - Stores ATC communications and flight data from OpenSky API. In addition, stores transcribed ATC communications received from Nvidia NeMo.

The Entity-Relationship Model for the communication database can be seen in *Figure 3* below.

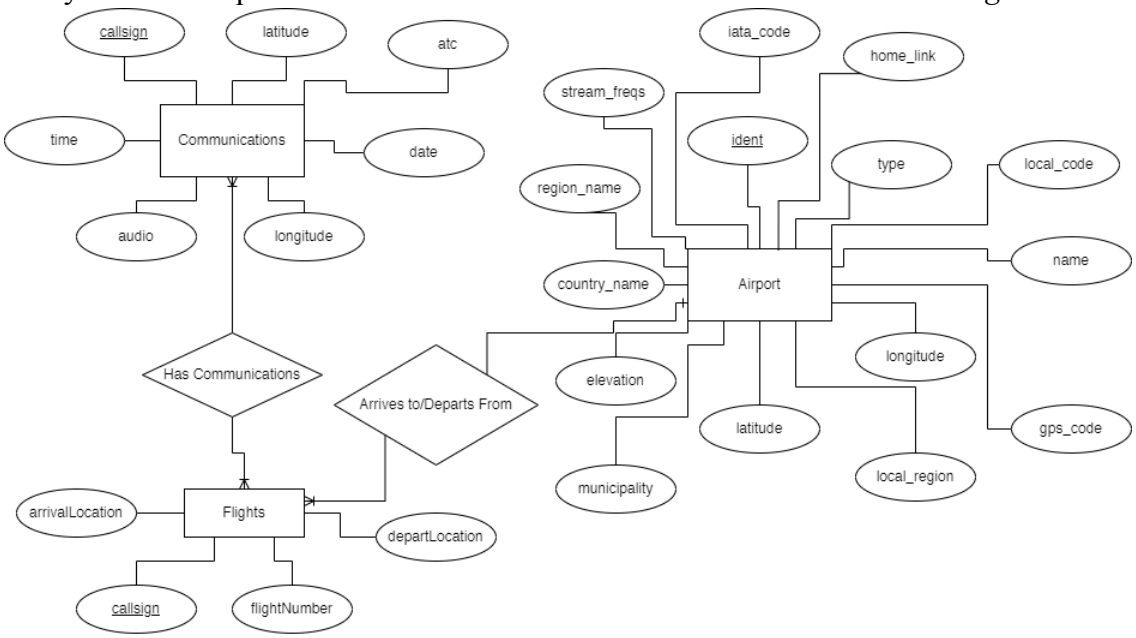


*Figure 14: Entity-Relationship Model for the communication databases.*

# 5 EXTERNAL INTERFACES

## 5.1 Interface Architecture

A database will be developed to hold all communication data. This database will be queried from user selection of airplane and/or airport icons to produce the relevant data for the selection.

## 5.2 Interface Detailed Design

Data will be stored and queried using a database. This database will store an audio file of a communication, a text transcription of the audio, the date and location that the communication took place, the callsign of the party making the communication, and a Boolean value indicating whether the party making the communication is ATC. This will be queried when a specific airplane or airport location is selected by the user to find the matching data. Information about the flight such as departure location, arrival location, callsign, and flight number will also be stored to the database and will be able to be queried if necessary. Airport information such as its identification, name, type, longitude, latitude, elevation, region, municipality, country, GPS code, IATA code, local code, stream frequencies, and website link will also be saved to the database for future development of the web application.

# 6 SYSTEM INTEGRITY CONTROLS

This project does not require system integrity specifications as it does not handle any sensitive data.