Requirements for Flight Simulator for Comfort Airlines

Team Name: Foobar
Teammates: Anthony Cox, Corey Lawrence, Dylan Hudson, Parker Blue, Will Wadsworth, Zach Christopher

Project Summary:

Comfort Airlines has contracted to rent 55 planes and has secured takeoff and landing agreements with 30 airports. The goal of this project is to provide a simulation and timetable adhering to the customer-provided specification document. This simulation and timetable will allow Comfort Airlines to assess the robustness of their business plan. The timetable should be well constructed and comprehensive. The simulation should be sufficiently realistic to generate approximations of the net profit for a two-week period, assuming a 2% market share.

Phase 1 - Gathering and Generating Data: In this phase, initial data is manually compiled. Then a series of python scripts will transform the data into the required input for the simulation and timetable generation software.

Note: CSV (comma separated value) files are being used for data storage and management because of their ease of use, and integration with programming software. This allows for more features to be delivered during this early stage.

1. **Read over all documentation provided by C.A.**
2. **Gather technical specifications on each plane type and create aircraft.csv (manually generated):**
    a. Fuel Capacity
    b. Cruise Speed
    c. Passenger Capacity
    d. Maximum Range

3. **Gather all information needed to create the master starting record, called airports.csv:**
    a. airport
    b. iata code
    c. city, state
    d. metro area
    e. metro population
    f. latitude, longitude

4. **Create master record called airports.csv (manually generated):**
    a. Columns: rank(index position), airport, iata code, city, state, metro area, metro population, latitude, longitude for each input airport.
    b. A header row should be used, and column names should be specified as seen above. (Excel is an excellent tool to manage and manipulate csv files).

5. **Create and run flight_weighted_distances.py:**
    a. **Purpose:** account for the flight time gained or lost due to the rotation of the earth.
    b. **Description:** This script is designed to give the weighted distances between all airports on the airports.csv. For 30 airports, the output file should contain 900 rows (30 x 30).
    c. **Input:** Airport name, latitude, longitude (for all airports on doc)
    d. **Output:** Source Airport (name, latitude, longitude), Destination Airport (name, latitude, longitude), weighted distance between the two, calculated using the haversine method.
    e. **Generates:** flight_weighted distance.csv

6. **Create and run flight_demand.py:**
    a. **Purpose:** Calculate the estimated number of passengers C.A. expects to fly with them each day, given the market share details provided in the documentation.
    b. **Description:** For each flight, the script calculates the proportion of the destination airport's metro population compared to the total metro population of all airports serviced by the source airport. This is done by dividing the metro population of the destination airport by the sum of the metro populations of all destination airports serviced by the source airport.
    c. num_passengers=round(metro_population[source_airport]×DAILY_DEMAND×MARKET_SHARE× percent_flight_demand)
        i. metro_population[source_airport] represents the metro population of the source airport.
        ii. DAILY_DEMAND is a constant representing the proportion of the population that wants to fly each day (0.005 or 0.5%).
        iii. MARKET_SHARE is a constant representing the market share of the airline company (0.02 or 2%).
        iv. percent_flight_demand is the calculated proportion of flight demand for the specific flight based on the destination airport's metro population relative to all destination airports serviced by the source airport.
        v. This process ensures that the number of passengers for each flight is estimated based on factors such as the size of the source airport's metro population, the demand for flights, and the market share of the airline company.
    d. **Input:**
        i. airports.csv: metro population for each airport
        ii. flight_weighted_distances.csv: source airport, destination airport
    e. **Output:** source airport, destination airport, company flight demand
    f. **Generates:** flight_demand.csv

7. **Create and run flight_fuel_capacity.py:**
    a. **Purpose:** Generate the fuel required for each potential flight.
    b. **Description:** This script reads in data from aircraft.csv and flight_weighted_distances.csv to calculate the fuel needed for each route.
    c. **Input:**

i. aircraft.csv: Aircraft Name, Maximum Fuel Capacity, Miles Per Gallon (MPG)
ii. flight_weighted_distance.csv: Source Airport, Destination Airport, Weighted Distance

d. **Output:** Source Airport, Destination Airport, Fuel for each aircraft type (in Gallons)
e. **Generates:** flight_fuel_capacity.csv

8. **Create and run flight_combine.py:**
    a. **Description:** This script aggregates information from three input files (flight_weighted_distances.csv, flight_demand.csv, and flight_fuel_capacity.csv) to create a combined list of flights. It combines data on flight distances, passenger demand, and fuel capacities for different aircraft types.
    b. **Purpose:** The purpose of the script is to consolidate data from separate sources regarding flight distances, passenger demand, and aircraft fuel capacities into a single comprehensive list of flights for use in subsequent data generation.
    c. **Input:**
        i. flight_weighted_distances.csv: source airport, destination airport
        ii. flight_demand.csv: demand per location
    d. **Output:** source airport, destination airport, distance (weighted in km), number of passengers for 2% market share (flight demand), aircraft name, fuel required for route.
    e. **Generates:** flights.csv
9. **Create and run flight_profit_or_loss.csv:**
    a. **Description:** The script calculates the profitability or loss of various flight combinations using different aircraft types over multiple routes. It generates separate CSV files for profitable and loss flights based on monetary calculations, considering only generic takeoff and landing fees and fuel costs.
    b. **Purpose:** This script will provide the profitability information for each route in the "flight_master_record.py" seen below.
    c. **Input:**
        i. flights.csv: Source Airport, Destination Airport, Distance (in km), Demand
        ii. aircraft.csv: Aircraft, Passenger Capacity, Cruise Speed (km/h), Miles Per Gallon (MPG)
    d. **Output:** The script generates a CSV file named flight_profit_or_loss.csv, which contains details about the profitability or loss of each flight combination. Each row in the output file represents a flight combination and includes the following information:
        i. Source Airport
        ii. Destination Airport
        iii. Aircraft Type
        iv. Total Cost v. Break Even Cost Per Ticket
        v. Profit/Loss Per Flight
        vi. Percentage Full

10. **Create and run flight_times.py:**
    a. **Description:** The script estimates the flight times between pairs of airports, considering various factors such as taxi times, ascent, and descent times, ramp-up and ramp-down times, and

cruising speeds. It calculates these flight times for different aircraft types and generates separate CSV files for each aircraft type.

b. **Purpose:** This script will be used in the main simulation to provide flight time for each route.
c. **Input:**
    i. airports.csv: Airport Name, Latitude, Longitude, Population (Metropolitan Area), Hub Status
    ii. flight_weighted_distances.csv: Origin Airport, Destination Airport, Weighted Distance (km)
d. **Output:** The script generates a CSV file named flight_estimated_times.csv, which contains the estimated flight times between pairs of airports for different aircraft types. Each row in the output file represents a flight combination and includes the following information:
    i. Origin Airport
    ii. Destination Airport
    iii. Aircraft Type
    iv. Flight Time (Minutes)

11. Create and run flight_master_record.py:

a. **Description:** The script aggregates data from multiple CSV files to create an aggregated list containing various flight-related information. It combines data on flight routes, profitability, and estimated flight times for different aircraft types.
b. **Purpose:** This is the only data being fed into the simulation. All previous work is compiled is compiled here for easy importing into the simulation.
c. **Input:**
    i. flights.csv: Source Airport, Destination Airport, Distance (weighted in km), Number of Passengers for 2% Market Share (Flight Demand), Name, Fuel
    ii. flight_profit_or_loss.csv: Source Airport, Source Airport, Destination Airport, Airplane Type, Total Cost, Break Even Cost Per Ticket, Profit/Loss Per Flight, Percent Full
    iii. flight_times.csv: Source Airport, Destination Airport, Airplane Type,
d. **Output:** The script generates a CSV file named flight_master_record.csv, which contains the aggregated flight-related information. Each row in the output file represents a flight combination and includes the following information:
    i. Source Airport
    ii. Destination Airport
    iii. Distance (weighted km)
    iv. Fuel
    v. Number of Passengers
    vi. Aircraft Type
    vii. Expected Time
    viii. Ticket Cost

12. Data generation summary:

A suite of .csv data files is generated in Phase One of the project, which are utilized by subsequent scripts. The following list outlines each file along with its contents:

aircraft.csv:

Contents: Aircraft name, passenger capacity, max speed (in km/h), max fuel (gallons), max range (km), and miles per gallon.

airports.csv:

Contents: Rank (based on most popular), airport name, IATA code, city, state, metropolitan area, metropolitan population, latitude, longitude.

flight_fuel_capacity.csv:

Contents: Source airport, destination airport, fuel for Boeing 737-600 (gallons), fuel for Boeing 767-800 (gallons), fuel for Airbus A200-100 (gallons), fuel for Airbus A220-300 (gallons).

flight_master_record.csv:

Contents: Source airport, destination airport, distance (weighted km), fuel, number of passengers, aircraft type, expected time, ticket cost, net profit. This file aggregates data from other scripts.

Phase 2 - Simulation and Timetable Generation: During this phase, a series of classes are created to facilitate the running of a simulation. After all classes are created, the timetable-generator/simulation main program can be created. During this initial simulation, a greedy algorithm will select for the most profitable flights. This will output a timetable optimized for profitability. The simulator will also capture a snapshot of its current state every minute. This data will be used in the next phase which is the user simulation.

1. **Create Aircraft Class:**
    a. **Description:** The Aircraft class represents a single aircraft object with various attributes. It also includes enumerated classes for aircraft type and status, as well as an aircraft factory class for creating and initializing aircraft objects.
    b. **Data Members and Requirements:**
        i. name: (string) Name of the aircraft.
        ii. model_type: (string) Model type of the aircraft.
        iii. status: (AircraftStatus enum) Current status of the aircraft.
        iv. location: (string) Current location of the aircraft.
        v. tail_number: (string) Tail number of the aircraft.
        vi. passenger_capacity: (integer) Maximum passenger capacity of the aircraft.
        vii. cruise_speed: (float) Cruise speed of the aircraft.
        viii. fuel_level: (float) Current fuel level of the aircraft.
        ix. fuel_capacity: (float) Maximum fuel capacity of the aircraft.
        x. fuel_efficiency: (float) Fuel efficiency of the aircraft.
        xi. max_range: (float) Maximum range of the aircraft.
        xii. wait_timer: (dictionary) Dictionary of wait timers for different statuses.

2. **AircraftFactory Class:**
    a. **Description:** The AircraftFactory class is responsible for determining the next tail number and initializing aircraft objects.
    b. Data Members and Requirements:
        i. __next_tail_number: (method) Calculates the next tail number for the aircraft.
        ii. create_aircraft: (method) Creates and initializes an aircraft object with appropriate values.
3. **Airport Class:**
    a. **Description:** The Airport class represents a single airport object with attributes such as name, IATA code, city, state, metropolitan population, hub status, available gates, latitude, longitude, gas price, takeoff fee, landing fee, and a tarmac queue. It also includes an enumerated class for airport type.
    b. **Data Members and Requirements:**
        i. name: (string) Name of the airport.
        ii. iata_code: (string) IATA code of the airport.
        iii. city: (string) City where the airport is located.
        iv. state: (string) State where the airport is located.
        v. metropolitan_population: (integer) Population of the airport's metropolitan area.
        vi. hub_status: (AirportType enum) Hub status of the airport.
        vii. available_gates: (integer) Number of available gates at the airport.
        viii. latitude: (float) Latitude coordinate of the airport.
        ix. longitude: (float) Longitude coordinate of the airport.
        x. gas_price: (float) Price of gas at the airport.
        xi. takeoff_fee: (float) Fee for takeoff from the airport.
        xii. landing_fee: (float) Fee for landing at the airport.
        xiii. tarmac_queue: (list) List representing the tarmac queue at the airport.

4. **Flight Class:**
    a. **Description:** The Flight class represents a single flight object with attributes such as flight number, scheduled time, aircraft type, flight path (route), and number of passengers.
    b. **Data Members and Requirements:**
        i. flight_number: (string) Flight number.
        ii. scheduled_time: (datetime) Scheduled departure time of the flight.
        iii. aircraft_type: (string) Type of aircraft operating the flight.
        iv. flight_path: (list) List representing the flight path or route.
        v. num_passengers: (integer) Number of passengers on the flight.

5. **Passenger Class:**
    a. **Description:** The Passenger class represents a single passenger object with attributes such as source airport, location, destination, number of flights taken, and a unique passenger ID. It

includes methods to retrieve expected departure time, actual departure time, expected arrival time, and actual arrival time.

b. Data Members and Requirements:
    i. source_airport: (string) Source airport of the passenger.
    ii. location: (string) Location of the passenger.
    iii. destination: (string) Destination of the passenger.
    iv. num_flights_taken: (integer) Number of flights taken by the passenger.
    v. passenger_id: (string) Unique ID of the passenger.
    vi. expected_departure_time: (method) Returns the expected departure time of the passenger.
    vii. actual_departure_time: (method) Returns the actual departure time of the passenger.
    viii. expected_arrival_time: (method) Returns the expected arrival time of the passenger.
    ix. actual_arrival_time: (method) Returns the actual arrival time of the passenger.

6. Route Class:

a. **Description:** The Route class represents a single route object with attributes such as aircraft type, source airport, destination airport, flight path distance, daily passengers, estimated flight time, and fuel requirement.

b. **Data Members and Requirements:**
    i. aircraft_type: (string) Type of aircraft flying the route.
    ii. source_airport: (string) Source airport of the route.
    iii. destination_airport: (string) Destination airport of the route.
    iv. `flight_path

7. Simulation Class:

a. **Description:** This is the class that will provide the timetable, and output the data needed to run the user simulation. It generates the necessary aircraft, airports, and route objects. Additionally, it uses a greedy scheduling algorithm to generate a timetable that optimizes net profit. A state snapshot of the simulation will be output each minute. This will become the starting point for the simulation. The timetable and simulation input will be generated for a two-week period.

b. **Objects Created:**
    i. Set of 55 aircraft based on the requirements specification (15 Boeing 737-600, 15 Boeing 737-800, 12 Airbus A200-100, and 13 Airbus A220-300)
    ii. Instances of 30 required airports

c. **Data Utilized (through importation in the main function):**
    i. flight_master_record.csv: Source airport, destination airport, distance (weighted km), fuel, number of passengers, aircraft type, expected time, ticket cost, net profit.

d. **Output (CSV Format):**
    i. List of all flights scheduled by the simulation (the "timetable", including the flight number, time, source airport, destination airport, number of passengers, scheduled

departure time, scheduled arrival time, actual departure time, actual arrival time, aircraft tail number).

ii. List of all financial transactions (profits and losses), including the item, net profit/loss, time, and location (if applicable).

iii. List of all passengers, including their source airport, destination airport, the list of all the flights they took (list of flight UUIDs delimited by semi-colons), scheduled departure time, actual departure time, scheduled arrival time, and actual arrival time.

iv. List of all airport data including all departed and arriving flights, their departure and arrival times, the number of passengers on the flights, and the gates used by the flights, aircrafts which underwent maintenance, and when that maintenance was performed.

e. **Utilizes the following classes:**

i. Aircraft Class

ii. AircraftFactory Class

iii. Airport Class

iv. Flight Class

v. Passenger Class

vi. Route Class

8. **Logging Class:**

a. **Description:** The Logging class provides logging functionality using the strutlog library. Log records are formatted in JSON format and include various fields such as current real time, current simulation date and time, source code location, and a unique log record ID.

b. **Methods:**

i. log_event(event): Logs an event along with the current real time, simulation date and time, source code location, and a unique log record ID.

Phase 3 – User Simulation and User Interface:

1. **User Simulation Class:**

a. **Description:** The UserSimulation class provides functionalities for users to interact with the simulation output. Users can check ticket prices, flight times, available seats, and flight status for any given flight. Additionally, users can obtain information about layovers, including layover airport details and combined flight time. The class also supports generating reports for ticket sales, expenses, and revenue within a specified window, ranging from one day to two weeks.

b. **Data Members:**

i. flights_data: A data structure containing information about all scheduled flights, including flight number, time, source airport, destination airport, number of passengers, scheduled departure time, scheduled arrival time, actual departure time, actual arrival time, and aircraft tail number.

ii. ticket_prices: A data structure storing ticket prices for different routes.

iii. flight_times: A data structure holding flight times for all flights.
iv. available_seats: A data structure tracking available seats on each flight.
v. flight_status: A data structure indicating the status of each flight, whether it is direct or layover.
vi. layover_info: A data structure providing information about layovers, including layover airport details and combined flight time.
vii. reports_data: A data structure containing information for generating reports, including ticket sales, expenses, and revenue for different time windows.

c. **Methods:**
   i. generate_revenue_report(start_date, end_date, airport=all): Generates a report of revenue for all flights within the specified time range. If an airport is specified, it generates revenue for flights departing from or arriving at the specified airport.
   ii. generate_expenses_report(start_date, end_date, airport=all): Generates a report of expenses for all flights within the specified time range. Expenses include airplane rental, maintenance, takeoff/landing fees. If an airport is specified, it generates expenses for flights departing from or arriving at the specified airport.
   iii. generate_flight_times_report(start_date, end_date, airport=all): Generates a report of flight times per airport for all flights within the specified time range. If an airport is specified, it generates flight times for flights departing from or arriving at the specified airport.
   iv. generate_passenger_status_report(start_date, end_date, airport=all): Generates a report of passengers served vs passengers left unserved per airport for all flights within the specified time range. If an airport is specified, it generates passenger status for flights departing from or arriving at the specified airport.

2. User Interface:

Option 1 - User Interface Description using PyQt:

The user interface developed with PyQt offers a robust platform for users to interact with and access information generated by the UserSimulation class. Leveraging PyQt's powerful features, the interface provides intuitive navigation, seamless data retrieval, and efficient report generation functionalities.

Key Features of the User Interface:

Main Window: The interface consists of a main window where users can navigate between different sections and access various functionalities.

Navigation Toolbar: A navigation toolbar is located at the top of the main window, offering easy access to different sections of the application. Users can switch between flight information queries and report generation with just a click.

Flight Information Section: In this section, users can input specific flight details such as flight number or airport in designated input fields. Upon submission, the interface retrieves relevant flight information such as ticket prices, flight times, available seats, and flight status.

Report Generation Section: Users can generate reports by selecting the desired report type (e.g., revenue, expenses, flight times, passenger status) from a dropdown menu. They can specify the start and end dates for the report and optionally choose an airport parameter if needed. The interface then generates the report based on the provided parameters.

Results Display Area: The interface includes a results display area where retrieved flight information and generated reports are presented in a structured format. Flight information queries display detailed data, while generated reports offer summarized information based on the specified parameters.

User Input Forms: Input forms are provided for users to enter query parameters and report generation parameters. Users can input flight numbers, airports, and date ranges using these forms to retrieve specific flight information or generate reports.

Interactive Elements: Interactive elements such as buttons, dropdown menus, and input fields facilitate user interaction with the interface. Users can trigger actions such as querying flight information or generating reports by interacting with these elements.

Option 2 - Web interface utilizing JavaScript and HTML:

The user interface implemented in JavaScript and HTML provides a platform for users to access and interact with the information generated by the UserSimulation class. The interface displays various functionalities for querying flight-related data and generating reports based on the provided data members and methods of the UserSimulation class.

Features of the User Interface:

Navigation Menu: The interface includes a navigation menu for easy access to different sections of the application, such as querying flight information and generating reports.

Flight Information Section: This section allows users to query flight-related information such as ticket prices, flight times, available seats, and flight status. Users can input specific flight details, such as flight number or airport, to retrieve relevant information.

Report Generation Section: Users can generate reports based on specified time ranges and optional parameters, such as airport. The interface provides options to generate revenue

reports, expenses reports, flight times reports, and passenger status reports. Users can input the start and end dates for the report and select an optional airport parameter if needed.

Results Display Area: The interface displays the results of queries and generated reports in a structured format. For flight information queries, it presents relevant details such as ticket prices, flight times, available seats, and flight status. For generated reports, it presents summarized data based on the specified parameters.

User Input Forms: Input forms are provided for users to input query parameters and report generation parameters. Users can enter flight numbers, airports, and date ranges as required.

Interactive Elements: The interface includes interactive elements such as buttons and dropdown menus for user interaction. Users can trigger actions such as querying flight information or generating reports by interacting with these elements.

Appendix:

1. **Scope:** The scope of the project is to develop a standard schedule (colq. "timetable") for the set of 55 airplanes rented by Comfort Airlines and then use the timetable to simulate two weeks of flight time. The simulation will be used to generate a report of business profits, expenses, and general business statistics. The standard schedule developed will be based on chapters 2 and 3 of the IATA Standard Schedules Information Manual: "Information Required for Standard Schedules (Data Requirements, Data Representation, Data Elements and Data Element Identifiers)" and "Standard Print Layouts for Schedules Information (Data Elements Required, Code Sharing Flights, Plan Change, Examples)".

2. **Document Conventions:** This document uses Markdown formatting. The document is divided into major sections denoted by section headers declared with a single # character and minor sections denoted by section headers declared with two consecutive # characters. Markdown is traditionally compiled and rendered as HTML, but can also be rendered in other formats such as PDF for flexibility. For more information about Markdown's features, visit the Markdown standard, RFC 7763.

3. **Intended Audiences:** The intended audience of this document is the client company that contracted the work, Comfort Airlines, and technical and non-technical team members working on the project, including but not limited to the project manager, product owner, and software developers. Knowledge of technical jargon used in software development and project management is assumed.

4. **External References**
    a. Databases: 3.5 Normal Form
    b. FAA: Aircraft Registration Guidelines
    c. IATA Standard Schedules Information Manual (SSIM)
    d. IEEE: Markdown Standard - RFC 7763
    e. ISO 8601: Date and Time Standardization

f. MariaDB: Overview
g. DockerHub: Official MariaDB Image
h. DockerHub: Official Python 3 Image
i. Pep 8: Official Python Style Guide
j. PyTest: Overview
k. Wikipedia: Charles de Gaulle Airport
l. Wikipedia: Top 30 Busiest Airports in the United States
m. Wikipedia: Universal Coordinated Time (UTC)

5. **Pre-research Research and Data Collection**
    a. Functional Specification
    b. Prior to initiating the simulation, the following static data about the aircraft, airports, and simulation must be collected:
        i. Aircraft Information Collection:
        ii. Gather all necessary information to model an aircraft, including its name, passenger capacity, cruise speed, fuel capacity, fuel efficiency, and maximum range.
        iii. Ensure compliance with FAA aircraft registration guidelines for aircraft tail numbers (format: N00000).
        iv. Airport Information Collection:
        v. Collect all pertinent details to model an airport, such as its name, IATA code, city, state, metropolitan population, available gates, coordinates (latitude and longitude), gas price, and takeoff/landing fees.
    c. Aggregate airport attributes from Wikipedia into a CSV file.
    d. Flight Data Derivation:
        i. Calculate the Cartesian product of airports and aircraft to derive all possible flights.
        ii. Determine flight attributes for each aircraft, including fuel required, flight duration, and net profit.
        iii. Sort flights by profitability.

6. **Deployment Requirements:**
    a. All deliverables must be operating-system independent, compatible with systems featuring a 64-bit CPU and a minimum of 8 GB of RAM.
    b. The simulation will be deployed as a two-part application-and-database system. Each component of the application, namely the app and the database, will be self-contained and interconnected within an internal bridge container network.

7. **Reliability Requirements:** To ensure reliability, the following requirements are mandated:
    a. Health Checks: All deployable software components (e.g., containers) must have a health check that is verified at least once per minute.
    b. Automatic Restart: In the event of a failure, all deployable software components must automatically restart.

c. Timely Recovery: All deployable software components must be fully recovered within 5 minutes of detecting a failure.
d. Clear Error Indication: All errors must clearly indicate the cause and location of the error for efficient debugging and troubleshooting.

8. **Security Requirements:**
    a. To uphold security standards, the following requirements must be met:
    b. Vulnerability-Free Software: The project must not incorporate any external software projects or packages known to have vulnerabilities.
    c. Data Confidentiality: Simulation data is classified as company confidential and must not be transmitted online without encryption. All data transmission over the internet must be encrypted.
    d. Network Isolation: Docker containers must be isolated over the network, utilizing an internal bridge network configuration and ensuring that no ports are exposed to external access.

9. **Scalability Requirements:**
    a. The simulation should be scalable such that it can simulate profits for any percent market share or change in airports, aircraft, or other simulation components; at a minimum, the simulation must be scalable from a 2% to 5% market share.

10. **Maintainability Requirements:** To ensure maintainability of the project, the following requirements must be followed:
    a. Pep 8 Compliance: All Python code must adhere to the guidelines outlined in Pep 8, the official Python style guide.
    b. File Documentation: Each code file must begin with metadata including the development team's name, members, creation date, last update date, purpose summary, and all preconditions and postconditions.
    c. Python Docstrings: All Python modules, functions, and classes (major structural components) must include Python docstrings describing their general function, precondition(s), and postcondition(s).
    d. Modular Code Design: Code should be modular, with well-typed and well-defined contracts connecting modules across boundaries to facilitate agility and testability.
    e. Version Control with Git: The codebase must be stored remotely on GitHub and managed using the Git version control system to track changes and facilitate collaboration.
    f. Use of LTS Releases: All packages utilized in the project must prefer Long-Term Support (LTS) releases whenever available to ensure stability and compatibility.

11. **Standardization Requirements:** To maintain consistency and facilitate interoperability, the following standardization requirements must be adhered to:
    a. Scientific Units: All units used in the project should adhere to scientific standards to ensure clarity and precision in measurements.

b. Time in UTC: Time data should be stored and processed in Coordinated Universal Time (UTC) to avoid confusion and facilitate synchronization across different systems. Time representation should follow the ISO 8601 standard to ensure compatibility and consistency in time formats.

12. **Documentation:** Documentation is crucial for understanding the project's thought process and execution details. All documentation is provided in Markdown format for consistency and accessibility. Below is a breakdown of the documentation files and directories:
    a. Individual Documents
    b. Algorithm: Contains definitions of key phrases, program input and output specifications, aircraft statuses, ledger entries, and pseudocode for the main algorithm.
    c. Docker: Provides instructions on using Docker, along with links to official Docker documentation for further reference.
    d. Logging: Details the usage and implementation of logging information in the project.
    e. Requirements: Software requirements specification document, outlining the project's functional and non-functional requirements (this document).
    f. Testing: Describes the types of testing employed in the project and the testing methodology used.
    g. Directories
    h. Diagrams: Contains outlines of project modules and an Entity-Relationship Diagram (ERD) representing the database structure.
    i. Meeting minutes: Notes from group meetings, documenting discussions and decisions made during project development.
    j. Process: Provides an introduction to the project process, offering insights for anyone looking to recreate the project.
    k. Standards: Project specifications for code, documentation, and general project standards.
    l. Timeline: A rough timeline outlining the production schedule of the project

13. **Client Feedback**
    a. To ensure alignment with client expectations and project progress, the development team is required to meet with the client at least once per week. During these meetings, the team will provide updates on their progress and present development plans for the following week, seeking approval from the client. The presentation should encompass the team's understanding of both functional and non-functional requirements to be planned or implemented in the upcoming week.

    b. By the end of each meeting, the development team and the client should have a clear understanding of the project requirements and expectations moving forward. This iterative feedback loop ensures that the project stays on track and meets the client's needs effectively.

14. **Software Testing:** To ensure that the developed software meets the client's requirements and maintains a high level of quality, rigorous testing procedures will be implemented throughout the development process.

a. **Testing Framework:**
    i. The application code will be tested using PyTest, a widely-used testing framework for Python applications. PyTest offers robust capabilities for writing and executing tests, making it suitable for various types of testing.
b. **Code Coverage:**
    i. A minimum of 50% code coverage is required for all application code. This ensures that a significant portion of the codebase is tested, reducing the likelihood of undetected defects.
c. **Types of Tests:**
    i. Unit Tests: These tests focus on individual components or units of code, verifying that each unit performs as expected in isolation.
    ii. Integration Tests: Integration tests validate the interactions between different components of the system, ensuring that they function correctly when integrated.
    iii. System Tests: System tests evaluate the system as a whole, testing end-to-end functionality to ensure that it meets the specified requirements.
d. Software tests will be integrated into the development workflow using continuous integration (CI) practices. This integration ensures that any code commit does not inadvertently introduce regressions or break existing functionality.
e. By incorporating testing into the development process, the team can identify and address issues early, leading to higher-quality software and more efficient development cycles.

15. Glossary

Term	Definition
Client	Comfort Airlines, the contracting company
Company Confidential	Proprietary information owned by Comfort Airlines
Timetable	A standard schedule as defined by the International Air Transport Association (IATA)

16. Revision History

Date	Added
2024-02-15	SRS 5 major sections and major section content
2024-02-21	Added more description functional requirements
2024-03-19	Added Deliverables and Unattempted Functionality sections
2024-03-22	Complete rewrite of requirements document including ample detail on scripts, data csv files, simulation/time-table generator, and user interface.