Compliant Project Report: Airport Management System (AMS)

Acomprehensive console-based system for managing critical operational data across multiple airport departments, demonstrating essential software engineering principles through modular design and structured data handling.

Introduction

Modern airport operations demand high efficiency and rapid coordination across multiple departments. The AMS project addresses this need by providing a fundamental console-based system for managing critical operational data. Developed using Python, this solution serves as a foundational prototype demonstrating essential software engineering principles, including modular design and structured data handling. The project aligns with the goal of applying subject concepts in a real-world context by identifying a problem and designing a technical solution.

Problem Statement

Small to medium-sized airports often rely on disparate spreadsheets or manual processes to track essential operational data across multiple domains like flights, passengers, and employees. This leads to data silos, inefficiencies, and difficulty in generating quick operational reports. The problem is to design and implement a centralized, streamlined, and user-friendly system to manage core airport entities and generate basic, yet critical, analytical reports from a single application interface.

Data Silos

Fragmented information across multiple systems

Inefficiencies

Manual processes causing delays and errors

Reporting Challenges

Difficulty generating quick operational reports

Functional Requirements (FRs)

The AMS features three major functional modules: Airline/Flight Management, Passenger Management, and Employee Management.

1

Secure Access (User Management)

Implement a username/password login for administrative access with error handling for incorrect attempts.

2

CRUD Operations

Provide complete Create, Read, Update, and Delete (CRUD) capability for all four main entities (Airline, Flight, Passenger, Employee).

3

Flight Status Tracking

Allow administrators to input and modify key flight parameters such as Flight No., Gate Assignment, and Departure/Arrival status.

4

Employee Reporting and Analytics

Generate analytical reports on employee data, specifically calculating and displaying the Average, Minimum, and Maximum salaries.

5

Clear Input/Output Structure

Ensure all user interaction follows a clear, menu-driven workflow (input) and uses formatted tables (output).

Non-Functional Requirements (NFRs)

At least four non-functional requirements must be specified.

NFR1: Usability

The system must be entirely menu-driven and intuitive for an administrator using the Command Line Interface (CLI).

NFR2: Security

The system must restrict unauthorized access via a mandatory login and limit login attempts.

NFR3: Maintainability

The code must be modular, separated into distinct files/functions for each management module to facilitate easy maintenance and future upgrades.

NFR4: Error Handling

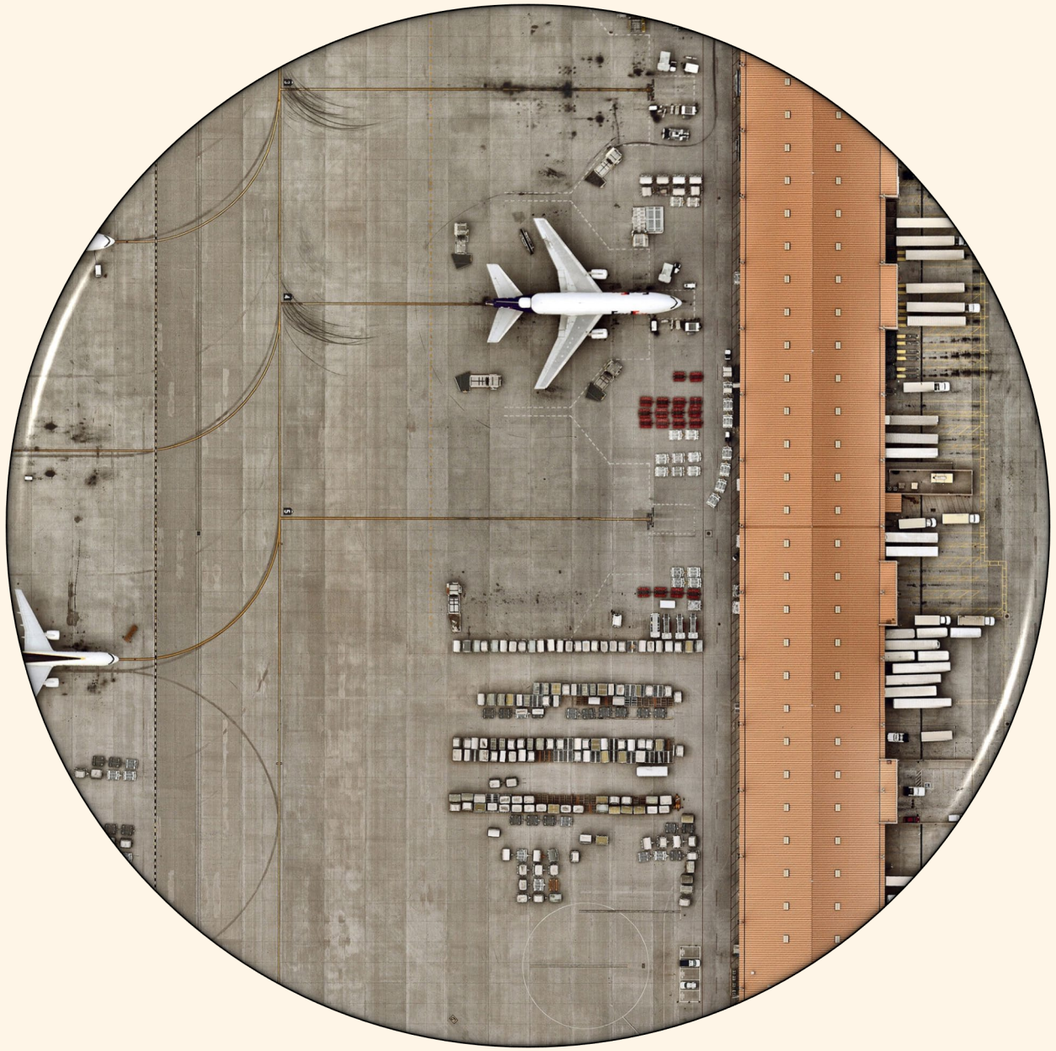
The system must include a robust error handling strategy (e.g., using try...except blocks) to gracefully manage invalid user inputs (non-integer inputs, searching for non-existent IDs, etc.).

System Architecture

The AMS utilizes a Procedural and Modular Architecture. The system is composed of several modules/files: main.py, auth.py, airline_module.py, flight_module.py, passenger_module.py, and employee_module.py. Acentral main.py file acts as the controller, hosting the top-level menu and managing the flow of execution between the distinct functional modules. Data is stored in in-memory Python lists, residing within the corresponding module scope.

Authentication

Login, roles, token management



Airlines

Manage carriers and fleets

Flights

Schedule, status, routing

Passengers

Profiles, bookings, check-in

Design Diagrams

Use Case Diagram

This diagram shows the scope of the system from the perspective of the Administrator (the single user role).

Workflow Diagram (Process Flow)

This diagram illustrates the logical steps a user takes to interact with the system.

Sequence Diagram

This diagram shows the time-ordered sequence of interactions for a key process, such as Admin Login.

Class/Component Diagram

Given the procedural nature, this acts as a Component Diagram, illustrating the modular files/components.

Design Decisions & Rationale

Python Lists as Data Store

Rationale: Chosen for rapid prototyping and meeting the requirement for a core technical solution without introducing external dependencies (like a full database), thus simplifying the project scope.

Modular File Structure

Rationale: Adopts a clean and modular implementation standard, separating authentication logic, core application menu, and the CRUD functions for each entity into dedicated files. This directly supports the NFR for Maintainability.

tabulate Library

Rationale: Used to format data output professionally in tables, enhancing Usability (NFR1) and improving the overall quality of the console-based reporting.

Implementation Details, Testing & Learnings

The system is built entirely on Python 3.x. The implementation focuses on the correct application of data structures (lists of dictionaries) and algorithms (linear search for CRUD operations, aggregation for reporting).

Example : Key Implementation Snippet: Salary Aggregation

The Employee reporting feature utilizes basic list comprehensions and Python's built-in functions to satisfy FR4.

```
def generate_salary_report(employee_list):    # This block executes if there is data    if employee_list:        salaries = [emp['salary'] for emp in employee_list]        print(f"Average Salary: {sum(salaries) / len(salaries):.2f}")        print(f"Minimum Salary: {min(salaries)}")        print(f"Maximum Salary: {max(salaries)}")    else:        print("No employee data available.")
```

Testing Approach

The project employed a combination of Unit Testing and System Testing. Unit Testing involved testing individual functions (e.g., add_airline(), delete_passenger(), login()) in isolation to ensure they performed their singular purpose correctly, including boundary condition checks (e.g., inputting a negative salary, checking an ID that doesn't exist). System Testing (Validation Tests) involved testing the complete workflow (e.g., logging in, navigating to the employee module, adding an employee, running the salary report, and exiting), ensuring all modules integrate seamlessly and the user interaction is logical.

Challenges Faced

- **Data Integrity on Updates:** Ensuring that updating a record (e.g., changing an employee's salary) correctly located the record via its unique ID and updated only that specific dictionary within the master list was a key challenge that required careful index management.
- **Robust Error Handling (NFR4):** Implementing input validation for non-numeric data (e.g., preventing a string input when an integer ID was expected) in all CRUD functions was time-consuming but essential for system stability.

Learnings & Key Takeaways

This project provided hands-on experience in Modular Programming by successfully applying subject concepts through breaking down a complex application into manageable, independent Python files and functions. It demonstrated practical application of Data Structures using Python lists and dictionaries to model real-world entities (flights, passengers) and manage their relationships. The project showed how to translate abstract Functional and Non-Functional Requirements (FRs and NFRs) directly into implementable features and code structure.

Future Enhancements

- **Data Persistence:** Implement database connectivity (e.g., SQLite) or file I/O (JSON/CSV) to save data between sessions.
- **Advanced Reporting:** Introduce filtering capabilities (e.g., search flights by status, report employees by department).
- **API Integration:** For a truly realistic system, incorporate external APIs to fetch real-time flight data.
- **GUI:** Develop a simple Graphical User Interface using Tkinter or PyQt for improved user interaction.

References

- Python Documentation (For core language syntax and built-in functions).
- The tabulate Library Documentation (For table formatting).
- Preeti Arora Computer Science textbooks, CBSE, Class 11 and 12th

PROJECT BY:
Rishabh Dhaulakhandi
25BAS10039
B.Tech Aerospace Engineering