**Functional Requirements Specification**

Module Code: FC723

Class/Group: Foundation Certificate for Science and Engineering | Group B

Module Title: Programming Theory

Assessment Type: Project

Module Tutor Name: Callum Birkett

Student ID Number: P470158

Date of Submission: 2025.4.4

Word Count: 2852 Words

GitHub link: 👉 <https://github.com/Ddddd917/apache-airlines-seat-booking>

I confirm that this assignment is my own work.

Where I/we have referred to academic sources, I have provided in-text citations and included the sources in the final reference list.

# 1. Introduction

## 1.1 Purpose

This document describes the functional requirements for the seat booking system to be developed for Apache Airlines. The system is intended to support the expansion of the company’s operations through the introduction of a digital solution for booking passenger seats on their new fleet of Burak757 aircrafts. This document outlines the key features, user interactions, and performance expectations of the software, and serves as a communication bridge between developers, stakeholders, and end users.

A diagram of a person with text

Description automatically generated

Use case Diagram

## 1.2 Background

Apache Airlines is undergoing an expansion of its commercial flight services and has acquired a fleet of Burak757 passenger jets. With the growth in operations, the airline has identified a need for a streamlined, software-based solution to handle seat reservations and enhance customer experience. A digital booking system would significantly reduce the reliance on manual processes, minimize human errors, and offer passengers greater transparency and flexibility when choosing seats.

This project simulates a real-world software development task, requiring the use of structured analysis, software engineering principles, and practical programming skills. It offers an opportunity to apply the concepts taught in the FC723 Programming Theory module to design, implement, and document a working system. The goal is to ensure that the software not only meets functional expectations, but is also efficient, maintainable, and user-friendly.

## 1.3 Scope

The Apache Airlines Seat Booking System is designed to support the reservation of seats on Burak757 aircraft. It provides users with the ability to check seat availability, book and cancel reservations, and view the overall seating layout. The system is implemented as a command-line interface (CLI) application to ensure simplicity, accessibility, and ease of use.

Key features include:

• Seat status tracking using symbols: **“F”** for free, **“R”** for reserved, **“X”** for aisles, and **“S”** for storage areas.

• Validation to ensure only valid seats can be booked.

• Prevention of bookings on aisle or storage area seats.

• Booking reference generation with 8-character alphanumeric codes.

• Integration with a SQLite database to persist booking and customer data.

• Support for additional features such as group booking and seat preference recommendations.

In **Version 4**, the system was fully refactored using object-oriented programming (OOP) principles. Each core function is encapsulated in modular classes (e.g., BookingManager, SeatMap, Validator, DatabaseManager), and a centralized controller class BookingSystem coordinates the program flow. This refactoring improves maintainability, scalability, and aligns the system with best practices in software design.

# 2. Software Development Process

## 2.1 Selected Model: Agile Development Model

For this project, the **Agile development model** has been selected due to its flexibility, iterative structure, and suitability for small-scale software systems with evolving requirements. Agile focuses on delivering functional software in short cycles, promoting continuous feedback, and encouraging adaptive planning throughout the project lifecycle.

### Rationale for Choosing Agile:

### 1. Incremental Development

The system can be built and improved step-by-step — starting from core functions such as booking and viewing seats, then adding advanced features like database support and group bookings.

### 2. User Feedback and Testing

After completing each functional block, tests can be performed and results reviewed, allowing improvements before moving to the next stage. This helps ensure the system meets expectations.

### 3. Adaptability

Agile supports changes during development — for example, if a new requirement such as seat preference search is introduced later, it can be integrated without disrupting the whole project.

### 4. Focus on Working Software

Agile prioritizes the delivery of working software over excessive documentation, which aligns well with this practical project where functionality is key.

### 5. Improved Collaboration and Clarity

By breaking the project into small, manageable tasks with clear goals, Agile helps the developer stay organized, track progress, and stay motivated throughout the process.

## 2.2 Development Phases

The development of the Apache Airlines seat booking system will follow a structured, multi-phase approach, aligning with Agile methodology. Each phase focuses on completing a specific set of tasks and producing a functional system that can be tested and iterated upon. Git version control will be integrated throughout the entire process to track changes and support incremental development.

Because of the size of the picture, it cannot be clearly displayed in the document, please check the Diagram folder

### Phase 1: Implementing Core Menu Functions

Develop the basic menu-driven Python application including:

• Check seat availability

• Book a seat

• Free a seat

• Show booking status

• Exit the program

This phase focuses on establishing a functional CLI-based system foundation.

### A diagram of a flowchart Description automatically generated

Activity Diagram of Version 1

Because of the size of the picture, it cannot be clearly displayed in the document, please check the Diagram folder

### Phase 2: Implementing Enhanced Features

Add advanced real-world features to enrich the system:

• Group booking

• Seat preference search

• Display booking summary with seat map

This phase enhances the user experience and expands the functionality scope of the application.

**A diagram of a company

Description automatically generated**

Activity Diagram of Version 2

Because of the size of the picture, it cannot be clearly displayed in the document, please check the Diagram folder

Phase 3: Adding Data Persistence and Database Integration

Refactor the program to:

• Store booking references and passenger data

• Generate unique 8-character booking codes

• Integrate a SQLite3 database

This phase satisfies Part B requirements and ensures long-term data handling.

### A diagram of a company Description automatically generated

Activity Diagram of Version 3

Because of the size of the picture, it cannot be clearly displayed in the document, please check the Diagram folder

### Phase 4: Refactoring with Object-Oriented Programming (OOP)

This final development phase involves restructuring the entire application using object-oriented programming principles. All core functionalities were encapsulated into dedicated Python classes to improve modularity, maintainability, and scalability.

The previous main script (`main.py`) was replaced with a new controller class (`BookingSystem`) in `booking\_system.py`, which coordinates all system modules.

This phase satisfies Part C requirements for demonstrating class-based design, maintainable architecture, and documentation readiness.

### Phase 5: Documentation and UML

Prepare:

• Functional requirements specification

• Activity, Use Case, and Class diagrams

• Final 1000-word explanation essay

This final phase ensures clear communication of the system’s logic, structure, and functionality.

## 2.3 Developing Tools and Environment

|  |  |
| --- | --- |
| Programming Language | Python |
| IDE | Pycharm |
| Database | SQLite3 |
| Git | Local Git + GitHub |
| Documentation | Microsoft Word |
| UML Diagrams | Draw.io |

# 3. Functional Requirements

## 3.1 Requirement 1:  Check Seat Availability

The system must allow the user to check whether a specific seat is available for booking. When the user inputs a seat identifier (such as “46A”), the system will validate the input and return the current status of the seat. Although the internal system stores seat status as “F” (Free) and “R” (Reserved), the program must present clear messages to the user such as “The seat is available” or “The seat is already reserved.”

The system must also handle edge cases, including invalid seat identifiers and seats located in non-bookable areas such as aisles (“X”) and storage areas (“S”). In these cases, an appropriate error message should be displayed.

## 3.2 Requirement 2: Book a Seat

The system must allow the user to reserve a seat by entering a valid seat identifier (e.g., “14C”). Upon receiving the input, the system will first check whether the seat exists and is available (i.e., marked as “F”). If the seat is free, the system will change its status to “R” (Reserved) and display a success message such as “Seat 14C has been successfully booked.”

If the seat is already reserved, or located in an aisle (“X”) or storage area (“S”), the system must not complete the booking and must display an appropriate message indicating why the action failed.

## 3.3 Requirement 3: Free a Seat

The system must allow the user to cancel a seat reservation by entering the seat identifier (e.g., “20D”). Upon receiving the input, the system will validate the seat and check its current status. If the seat is currently reserved (marked as “R”), the system will change its status back to “F” (Free) and display a confirmation message such as “Seat 20D has been successfully freed.”

If the seat is already free, or located in an aisle (“X”) or storage area (“S”), the system must inform the user that the seat cannot be freed and explain the reason (e.g., “This seat is not currently reserved”).

## 3.4 Requirement 4: Show Seats Layout and Booking Status

The system must provide functionality to display the entire seat map of the Burak757 aircraft, showing the booking status of each seat. Each seat should be represented by a symbol according to its current state:

• “F” for free

• “R” for reserved

• “X” for aisle

• “S” for storage area

The seat map must be displayed in a readable and structured format, allowing the user to easily identify which seats are available or booked. For enhanced clarity, the system should include row and column labels, and possibly group seats by section (e.g., A–C on the left, D–F on the right of the aisle). This output helps users make informed booking decisions and provides a complete overview of current reservations.

## 3.5 Requirement 5: Exit Program

The system must provide an option to safely exit the program from the main menu. When the user selects the “Exit” option, the system should terminate all operations gracefully, ensuring that any in-memory data structures are properly saved or that the program ends without errors.

The exit process must include a clear confirmation message such as “Thank you for using Apache Airlines Booking System. Goodbye!” This enhances user experience and confirms successful termination. No further input should be accepted once the program is exited.

## 3.6 Requirement 6: Display Seats Layout and Booking Statues

The system must generate and display a booking summary along with the seat layout whenever the user selects the option “Show Seats Layout and Booking Status” from the main menu. After displaying the visual layout of all seats with their current status (“F”, “R”, “X”, “S”), the system should print a summary containing:

• The total number of seats currently reserved

• A list of all reserved seat identifiers (e.g., 4A, 5B, 10C, etc.)

This summary helps users quickly understand the overall reservation situation without having to scan the seat map manually. It also serves as a useful feature for system administrators or staff who wish to monitor current occupancy levels.

This functionality is automatically triggered when Option 4 is selected from the main menu, and does not require separate user input.

## 3.7 Requirement 7: Group Booking

The system must provide functionality for booking multiple seats in a single transaction. Users can input a list of seat identifiers (e.g., “12A 12B 12C”), and the system will verify that **all** specified seats are valid and currently available. If every seat in the group is free, the system will mark all of them as reserved (“R”) and display a confirmation message listing all successfully booked seats.

If one or more seats in the group are already reserved or located in non-bookable areas (such as “X” for aisle or “S” for storage), the system must cancel the entire transaction and inform the user that group booking has failed. A clear message must indicate which seat(s) caused the failure.

This feature supports real-world booking needs for families, groups of friends, or travel parties who wish to sit together, and improves user experience by minimizing repetitive actions.

## 3.8 Requirement 8: Seat Preference Search

The system must allow users to search for available seats based on their seating preference. When selecting this function, users can choose from one of the following options:

• Window seat

• Aisle seat

• Middle seat

The system will then search for seats that match the user’s preference and are currently available for booking. Based on the aircraft layout:

• **Window seats** are located in **columns A and F**

• **Aisle seats** are located in **columns C and D**

• **Middle seats** are located in **columns B and E**

To enhance comfort and usability, the system must prioritize available seats starting from the front of the aircraft (i.e., from row 1 upward to row 80). A maximum of four matching, unreserved seats will be recommended in each search result. The user can then choose one of the suggested seats to proceed with the booking.

If no matching seats are found, the system must display a message such as “Sorry, no available aisle seats found.”

## 3.9 Requirement 9: Class-Based System Refactoring

The system must be refactored to follow an object-oriented structure. Each core module should be represented as a class with encapsulated logic, including:

• BookingManager – handles booking and cancellation

• SeatMap – generates and displays the seat layout

• DatabaseManager – interacts with the SQLite3 database

• Validator – performs seat and input validation

• SearchManager – enables searching bookings by identity

• SeatPreference – manages seat recommendation based on preference

A new main controller class, BookingSystem, coordinates all modules and replaces the previous procedural main function.

This restructuring ensures better separation of concerns, supports future extensions, and meets the maintain

# 4. Non-Functional Requirements

## 4.1 Performance

The system must respond to user inputs and execute operations (e.g., checking seat availability, booking, cancelling) within **one second** under normal usage conditions. The seat map should render instantly upon selection. The application should maintain a stable runtime without crashing, even during repeated operations. The adoption of object-oriented design in Version 4 helps structure logic more efficiently, which contributes to consistent response times even under frequent user operations.

## 4.2 Usability

The system should have a clear, menu-driven interface with user-friendly prompts and instructions. Input validation must be implemented to prevent invalid seat numbers or actions. The menu should remain visible after each operation, allowing users to perform multiple actions without restarting the program. With the modular class-based structure, each user interaction is handled by dedicated components, simplifying logic flow and improving prompt clarity and consistency.

4.3 Security

Although this is a local, standalone system, basic security measures must be in place:

• Users must not be allowed to modify non-bookable areas (aisles “X” and storage “S”).

• Booking references and customer data (Part B) must be accurately stored and removed only by authorized program logic.

• Database access should be handled through controlled, structured queries to prevent corruption.

Encapsulating booking and database operations in well-defined classes (e.g., BookingManager, DatabaseManager) ensures controlled access to critical data and enforces proper logic flow for secure data handling.

# 5. Assumptions and Constrains

## 5.1 Assumptions

1. Users have basic familiarity with command-line interfaces.

2. Seat identifiers (e.g., 12A, 25C) are formatted consistently and are valid.

3. The Burak757 seat layout remains fixed throughout the system lifecycle.

4. Only one user interacts with the system at a time (single-user mode).

5. All operations are performed in real-time, and no data is preloaded from external sources.

## 5.2 Constraints

1. Aisles (“X”) and storage areas (“S”) must never be modified or booked.

2. Seat input must be validated against predefined seat codes and layout.

3. System is CLI-based with no graphical interface.

4. All data is stored locally; no remote access or cloud database is implemented.

5. The application must be submitted as part of an academic project within defined deadlines.