**Response to ATI Questions**

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

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.

# Part A Software Implementation Reflections

**1.** **“Describe a software development process in detail that you would choose for the development for the above description. Your answer must provide at least three rationales for the choice you make. ” (From Part A in ATI)**

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. “As a systems analyst, study the description provided by the Apache airlines and produce a formal Functional Requirements Specification document to capture the requirements of the prospective software system of Apache airlines. At least five requirements must be identified in the Functional Requirements Specification document.” (From Part A in ATI)**

## **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.

## **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.

## **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”).

## **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.

## **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. “Produce an activity diagram that will represent the identified functionalities of the prospective system.”**Because of the size of the picture, it cannot be clearly displayed in the document, please check the Diagram folder

**A diagram of a flowchart

Description automatically generated**

**Activity Diagram of Version 1**

**A diagram of a company

Description automatically generated**

**Activity Diagram of Version 2**

**A diagram of a company

Description automatically generated**

**Activity Diagram of Version 3**

The Apache Airlines Seat Booking System was developed iteratively across four major versions, each enhancing the system’s functionality and architecture.

• Version 1 implemented the core features: checking seat availability, booking a seat, cancelling a booking, and displaying the seat layout using a command-line interface.

• Version 2 introduced three additional user-friendly features: group booking for up to three adjacent seats, seat preference search (window/aisle/middle), and a detailed booking summary with seat statistics.

• Version 3 added persistent data handling through SQLite database integration, enabling secure storage of passenger identity data and automatically generating 8-character booking reference codes. A new search function was added to retrieve bookings by full name and passport number.

• Version 4 focused on code refactoring using Object-Oriented Programming (OOP). All functions were encapsulated into well-structured classes such as BookingManager, SeatMap, and DatabaseManager, making the system more maintainable, reusable, and modular.

**4. “Develop a seat-booking application in Python that will provide the following functionalities given below. The functionalities must be listed in a menu as options. The menu must be available to the used until the program is terminated. Provide comments within your programs in such a way that the comments can be later used for producing documentation of your program. ”(From Part A in ATI)  
Basic functionalities:**

**1. Check availability of seat   
2. Book a seat   
3. Free a seat   
4. Show booking status   
5. Exit program**

The seat-booking application with all five required functionalities has been fully implemented in Python and is submitted as part of the project attachment. The source code contains clear comments to support future documentation and understanding.

**5. Describe and implement a common functionality that may be available in a airline booking system but not described in Apache airlines description above (you may have to update your activity diagram). (From Part A in ATI)**

## **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.

## **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.

## **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.”

**6. “ In order to maintain version control of your code base, create a publicly accessible GitHub repository within your account and commit your code to the remote repository. Describe in detail the steps you followed to achieve this task. Provide the link to your repository. ” （From Part A in ATI）**

**Describe the git command that can be used show the updates you have made to the program. Give an example using your own repository. (From Part B in ATI)**

To maintain version control throughout the development of my Apache Airlines Seat Booking System, I used GitHub and GitHub Desktop to manage and track the evolution of my code. The steps I followed are outlined below:

**1. Initial Setup**

   - I created a new local folder on my computer to contain the project files.

   - Then, I logged into GitHub and created a public repository titled `apache-airlines-seat-booking`.

   - I linked the local folder with the remote repository using GitHub Desktop.

**2. Project Structure and Version 1 Commit**

 In the local folder, I built a modular project structure using multiple Python files (e.g., `main.py`, `booking.py`, `seatmap.py`, `validation.py`, etc.). These files implemented the five core functionalities required in Version 1.

I wrote the initial `README.md` file to describe the system and added a `version\_1\_description.txt` to explain the features implemented in Version 1.

Using GitHub Desktop, I committed and pushed all files to the remote repository with appropriate commit messages.

A screenshot of a computer

Description automatically generated

Project Structure of Version 1

**3. Development of Version 2**

I created additional Python files to implement new features such as group booking and seat preference search (e.g., `preference.py`).

I updated the `main.py` and other modules to integrate these new features while maintaining modular structure.

I expanded the `README.md` file to include:

     - A summary of Version 1 features

     - A detailed explanation of Version 2 improvements

     - A future plan for Version 3 (database and persistence)

 I also wrote and committed a `version\_2\_description.txt` file summarizing the updates.

 These changes were committed and pushed to GitHub under version tag `v2.0`.

A screenshot of a computer

Description automatically generated

Project Structure of Version 2

**4. Development of Version 3**

   - I created a new module `database.py` to implement SQLite3 integration for data persistence.

   - I significantly updated `booking.py` to:

     - Generate unique 8-character booking reference codes

     - Collect and store passenger identity information

     - Store all data into a database table (`passengers`)

   - I added validation for reference codes and passport formats in `validation.py`, and centralized constants in `constants.py`.

   - I again updated the `README.md` file to reflect all current features and version history.

   - I created and committed `version\_3\_description.txt` to document the changes introduced in Version 3.

   - After thorough testing and final review, I pushed the final version to GitHub and tagged it as `v3.0`.

A screenshot of a computer

Description automatically generated

Project Structure of Version 3

**5. Development of Version 4**

• I replaced the procedural main.py with a new class-based controller module called booking\_system.py.

• I refactored all existing modules using object-oriented programming (OOP) principles. Each major functionality was encapsulated into a dedicated class:

- BookingManager for handling bookings and cancellations

- SeatMap for managing and displaying seat layouts

- Validator for all input validation

- DatabaseManager for database interaction

- SearchManager for identity-based reference lookup

- SeatPreference for preference-based seat recommendations

• I modified each module to improve readability, reduce duplication, and enhance maintainability.

• I added or adjusted validation logic for passport and reference codes where necessary.

• I updated README.md to include Version 4 descriptions and new class-based architecture.

• I created and committed a new version\_4\_description.txt to document this refactoring phase.

• After successful testing, I committed and pushed Version 4 to GitHub and tagged it as v4.0.

A screenshot of a computer

Description automatically generated

Project Structure of Version 4

**6. Repository Accessibility**

 The GitHub repository is public and accessible at the following URL:

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

Throughout the process, GitHub and GitHub Desktop allowed me to organize my work clearly, track changes across versions, and ensure a professional workflow that reflects real-world software development practices.

I chose to write and maintain the `README.md` file using Markdown and keep all versions of the project on GitHub not only to meet the coursework submission requirements, but also to begin building my personal programming portfolio.

By making the repository publicly accessible and clearly documenting each stage of development, I aim to use this project as part of my academic and professional profile. In the future, this GitHub repository can serve as evidence of my software development skills when applying for postgraduate studies or job opportunities in the technology sector.

# Part B Extended Features and Version Control

**1. “ Develop a functionality based on an algorithm that will produce a random booking reference. The booking reference must have exactly eight alphanumeric characters. When a new reference is produced the system must make sure the reference is repetitive. You must comment your code and describe in detail the implementation logic of your algorithm.  ”**

During the implementation of the booking reference generation feature, I first defined a constant in the constants.py file to specify that the reference code must be exactly 8 characters long, consisting of uppercase letters and digits. This approach ensures that all parts of the system follow a unified format standard when generating reference codes, which improves code consistency and maintainability.

Next, I created a dedicated function in the booking.py file specifically for generating reference codes. This function randomly selects 8 characters from the predefined character set, and then checks whether the generated code already exists in the database by calling the get\_booking\_by\_reference function. If the code is already taken, the function will regenerate a new one until a unique reference code is created. This guarantees that every passenger receives a unique reference code that does not conflict with any existing ones.

The entire process is simple, efficient, and implemented in a modular way, with clearly separated responsibilities, making it easy to maintain and reuse. This reference generation logic is integrated into both individual and group booking functions, ensuring that valid and unique references are consistently generated across all booking scenarios.

A screen shot of a computer

Description automatically generated

Code of reference generation part

**2. Refactor your functionalities from part A in such a way that when a booking is made the reference is stored in the data structure and customer data is stored on a database table. Likewise, when a seat is freed the letter “F” is stored and any booking details from the database is removed. On completion of this task, make the second commit to the remote repository.**

 (8 marks)

The functionalities from Part A have been successfully refactored. Upon booking, a unique reference code is generated and stored in the in-memory data structure, while customer information is saved to a SQLite database. When a seat is freed, the seat status is reset to “F” and the corresponding booking record is removed from the database. This updated version has been committed to the remote GitHub repository.