* + - Module Code (e.g. FC723)
    - Class/Group: (e.g. Group B)
    - Module Title (e.g. Programming Theory)
    - Assessment Title (e.g. Portfolio Project 1)
    - Tutor Name: (Callum Birkett)
    - Student GUID Number: (3075459T)
    - Date of Submission: (date)

## 1. Introduction

This software project was developed in response to a system request by Apache Airlines. The airline needed a robust and user-friendly system to manage seat bookings for their newly acquired Burak757 fleet. The software enables users to check seat availability, book and cancel reservations, and view seating status through a menu-driven console application. The application was implemented in Python, utilizing object-oriented programming, random algorithms, and an SQLite database for data persistence. The system was built in two phases: Part A focused on basic booking features, and Part B extended the functionality to include booking references and customer information storage. That is why I chose Agile Software Development for this project. It is flexible and iterative, which fits the two-phase structure of the task.

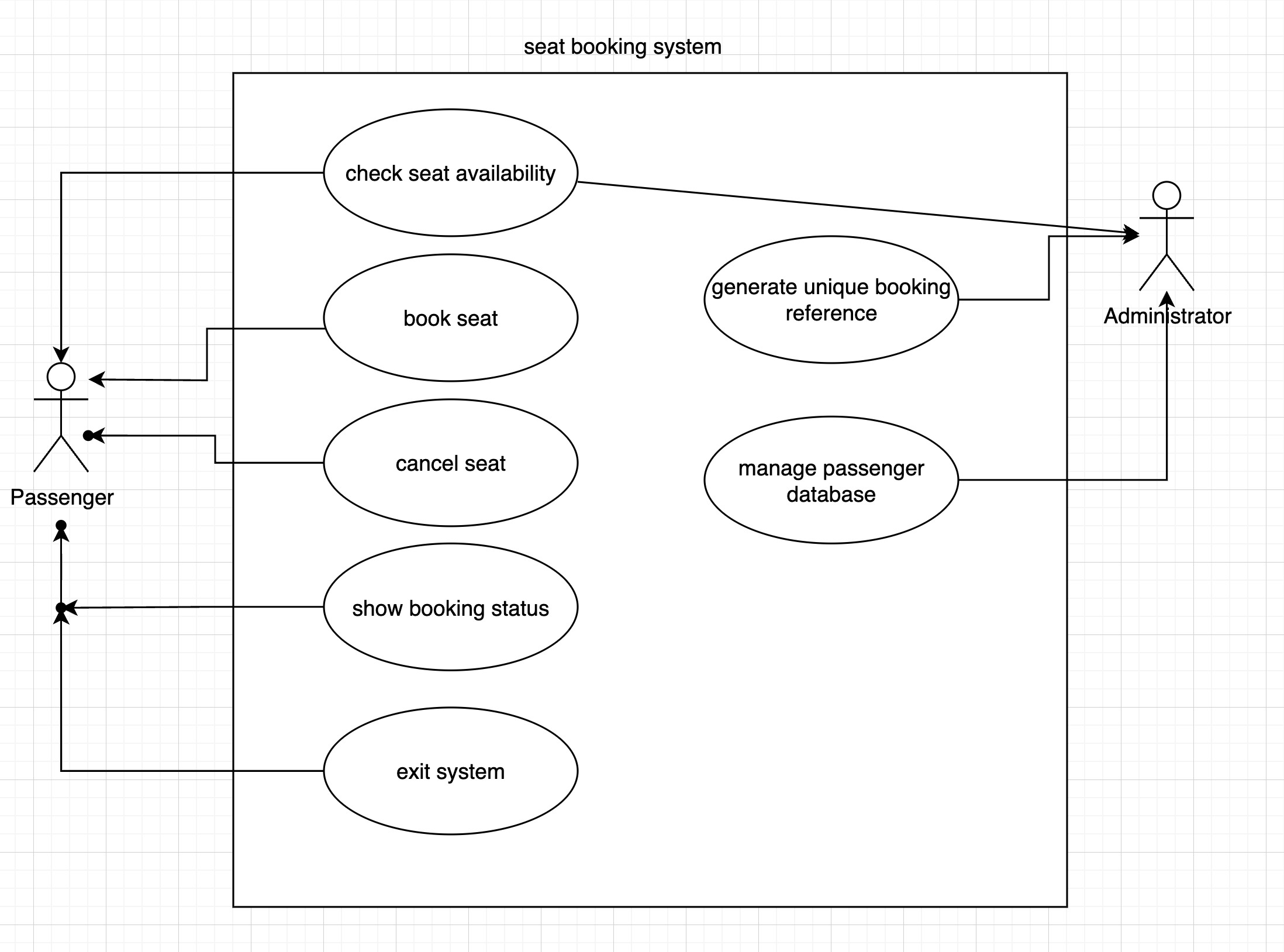
1. Agile allows the software to evolve over time. In Part A, I built the core booking system. Later, in Part B, I added new features like booking references and a customer database without restarting the whole system.
2. Agile supports early testing and feedback. After each function was added, I tested it immediately to catch bugs early.
3. Agile makes the code easier to maintain. Since each function was built as a small module, changes were easy to apply.

## 2. Use Case Description

The system is designed to be used by an airline booking agent. The agent interacts with the system through a terminal interface to carry out routine booking operations. The key use cases include:

1. Checking seat availability
2. Booking a seat
3. Cancelling (freeing) a seat
4. Viewing the current seating layout
5. Exit system

Each use case corresponds to one of the menu options displayed in the program’s main loop. The user selects an option and provides necessary input, such as a seat number or passenger details. Invalid operations—such as booking an aisle or storage seat—are handled with appropriate error messages.



## 3. Activity Diagram

The activity diagram below illustrates the main flow for the "Book a Seat" functionality. When the user selects the booking option, they are prompted to enter a seat number. If the seat is available, the system proceeds to collect customer information (passport number, first name, last name). Then, a unique 8-character alphanumeric booking reference is generated and stored both in the seat data structure and in the SQLite database. If the seat is unavailable or invalid (aisle/storage), the system terminates the flow with an error message. The common functionality I added is waitlist function. A waitlist allows customers to register for fully booked seats. When a seat is freed, the first waitlisted passenger is automatically assigned(no python implement).

This approach ensures that only valid bookings are stored, and the seat map remains synchronized with the database.

图示

AI 生成的内容可能不正确。

## 4. Class Diagram

The system employs object-oriented programming with modular design. The major classes are:

* Seat: Represents each individual seat with attributes like seat\_id, status, and booking\_ref.
* Airplane: Manages the entire seating layout by initializing and accessing `Seat` objects.
* Booking System: Handles user interactions and connects logic across modules.
* Utils: Contains helper functions, including the booking reference generator.
* Database: Manages SQLite operations such as table creation, record insertion, and deletion.

This modular structure allows for easy future upgrades, such as implementing user login or online access.

图示

AI 生成的内容可能不正确。

## 5. Technical Implementation

The application uses a console-based menu interface driven by a while-loop in BookingSystem.run(). When a seat is booked, the book\_seat() method checks seat validity and availability, prompts for user information, generates a booking reference using the generate\_booking\_reference() function in utils.py, and stores both seat and customer details.

The system prevents booking conflicts by checking the seat’s current status before proceeding. If the booking is successful, data is written to the database using insert\_booking() in database.py. When a seat is freed via free\_seat(), the system updates the seat’s status and removes the corresponding customer record with delete\_booking().

The database (‘bookings.db’) is initialized at runtime with a ‘bookings’ table that stores passenger details and seat assignments. The system guarantees booking reference uniqueness via a Python set object that tracks previously generated references in memory.

Robust input validation ensures that storage and aisle seats cannot be booked. This is hardcoded during ‘Airplane’ initialization, based on seat position (e.g. ‘C’ seats every 5th row are marked as ‘X’).

The codebase is organized into separate modules, which improves maintainability and readability.

## 6. Conclusion

This software successfully meets the functional requirements set by Apache Airlines, offering a clean and extensible Python implementation of a seat booking system. The use of modular programming, object-oriented structure, and persistent storage via SQLite ensures both efficiency and scalability. The system could be further improved in the future by implementing a graphical user interface or allowing bookings through a web-based platform. In summary, the project demonstrates how core programming principles can be applied effectively to solve real-world problems in a structured, maintainable way.