**Module Code (FC723)**

**Class/Group: (Group C)**

**Module Title (Programming Theory)**

**Assessment Title (Project 1)**

**Tutor Name: (****Sophie Norman)**

**Student GUID Number: (3075614)**

**Date of Submission: (6th April 2025 at 23:59)**

**I confirm that this assignment is my own work.**

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

**Part A, Q1**

For the development of the seat-booking system for Apache Airlines, I have chosen the Waterfall Software Development Process. The Waterfall model is a linear and sequential approach to software development, where each phase must be completed before moving on to the next. It consists of the following main stages:

1- Requirements Gathering and Analysis.

2- System Design.

3- Implementation (Coding).

4- Testing.

5- Deployment.

6- Maintenance.

Reasons:

**Clear and Well-Defined Requirements**

The Apache Airlines project has already provided a detailed system description, including a seat layout diagram and a list of required functionalities. Since the requirements are unlikely to change during development, the Waterfall model is suitable because it works best when everything is known from the start.

**Strong Focus on Documentation**

Waterfall places a strong emphasis on documentation at every phase, which aligns with the academic requirement of this project. Each stage (e.g., functional specifications, activity diagrams, UML, and version control steps) can be clearly documented and presented.

**Simple and Easy to Manage**

Waterfall is straightforward to plan and track. It suits small- to medium-sized projects like this one, where all team members (or a single student developer) can follow a step-by-step approach without the need for iterative feedback loops or changing scope.

**Part A, Q3**

**Part A, Q5**

The group\_booking function allows a user to book 2 to 4 adjacent seats in the same row. It scans each row while skipping over the aisle column, looking for a continuous block of free seats ("F"). If such a block is found, the seats are reserved ("R"), and their seat codes are displayed. If no suitable seats are available, the user is notified.

**Part A, Q6**

The first step I took to create a Git repository and commit my code was to set up a public repository on GitHub. I then opened Git Bash and configured my Git identity by entering my name and email address using the git config command. After that, I navigated to my local project directory and used the git clone command to link it to the remote repository. To initialize version control locally, I ran git init, which created an empty Git repository in my project folder. I then added my project files to the staging area using git add . and committed the changes with a message using git commit -m "Initial commit". Next, I set the remote repository using the git remote add origin <URL> command, and finally, I used git push --force to push my committed changes to the main branch of the remote GitHub repository.

Link: <https://github.com/Hydro-XD10/Project-1-FC723.git>

Branch: “master”

**Part B, Q3**

To show and manage updates made to the program using Git, several commands are used during the version control process. The git status command displays the current state of the working directory, including any modified, added, deleted, or untracked files. This helps me review changes before committing. Additionally, the git diff command can be utilized to precisely display changes between the current working directory and the last commit, clearly highlighting line-by-line differences and ensuring the developer understands what has been modified. Once changes are reviewed, the git add . command stages all updated files, making them ready for commit. If a file needs to be removed from both the staging area and the working directory, the git rm filename command is used, deleting the file and staging its removal. After staging, the git commit -m "message" command saves the changes to the local repository with a descriptive commit message. Finally, the git push -f origin master command pushes the committed updates to the remote repository on the master branch.

**Part C**

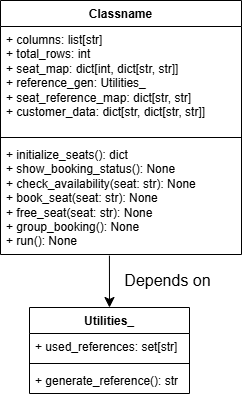
**Introduction**

The Apache Airlines Seat Booking System is a software solution designed to simplify and manage the process of seat reservation within an airline. The primary goal is to allow airline passengers and staff members to efficiently check seat availability, book individual or multiple seats, free previously booked seats, and quickly look up booking details by seat number or reference number. This software is designed with simplicity and clarity in mind, featuring an intuitive command-line interface (CLI) that guides users through all necessary interactions. The implementation includes generating unique booking references and maintaining essential passenger details, all efficiently handled within the application's runtime.

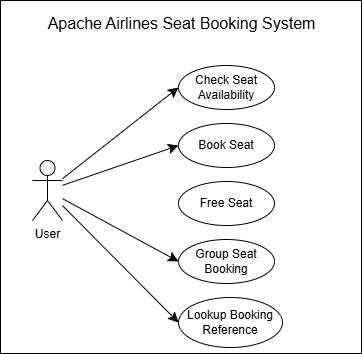
**what the software does and how the software works.**

The Apache Airlines Seat Booking System is designed to simplify and efficiently manage seat reservations for airline passengers. It enables user to check the real-time availability of seats, book individual or multiple adjacent seats, free previously booked seats, and easily retrieve booking details using either seat numbers or booking references. The system operates via a clear command-line interface that guides the user through each step intuitively, ensuring straightforward interactions. Upon initialization, the software creates a structured seat layout with clearly designated seats categorized as free, aisle, or storage. When passengers request seat bookings, the system verifies availability before prompting for essential personal information such as first name, last name, and passport number. After data entry, the software generates a unique eight-character booking reference using a reliable and straightforward UUID-based algorithm. This booking reference is stored alongside passenger details and seat information in clearly structured internal data maps. The software’s operational logic is organized around a well-defined class structure, comprising a central SeatBookingSystem class responsible for handling user interactions and data management, and a supporting Utilities class focused on generating unique booking references. Each method within these classes handles specific tasks, maintaining a clear separation of responsibilities. Users can perform actions seamlessly through menu-driven prompts that facilitate checking availability, booking or freeing seats, group seat reservations, and retrieving booking details. This structured approach ensures accurate, real-time data management, straightforward navigation, and ease of use. The modular design further supports potential expansions, such as integrating persistent storage solutions, thereby enhancing functionality and usability in future updates. The following Class Diagram demonstrates the main components of the App.

**Class Diagram (Figure 0)**



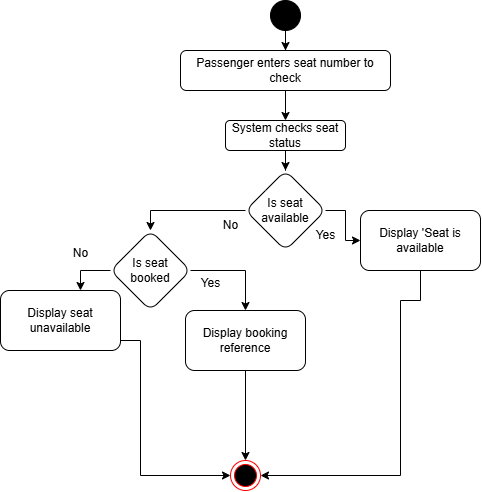
**Use Case Diagram (Figure 1)**



Staff

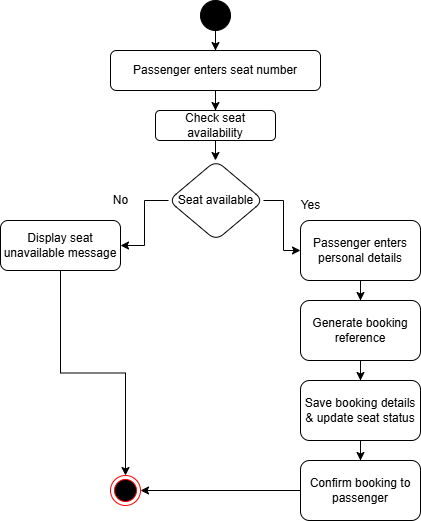
Figure 1 illustrates the Use Case Diagram for the Apache Airlines Seat Booking System, clearly showing the interactions between the main actor, the user (Staff), and the software system itself. The diagram defines five key use cases: Check Seat Availability, Book Seat, Free Seat, Group Seat Booking, and Lookup Booking Reference. The Passenger initiates all interactions directly, clearly representing typical user actions performed within the system. For example, when booking a seat, the Passenger selects an available seat, provides personal details, and the system responds by generating a unique booking reference and updating the seat status accordingly. The straightforward layout of the diagram emphasizes intuitive user interaction and highlights the primary functionalities supported by the booking system.

**check\_availability (Figure 2)**



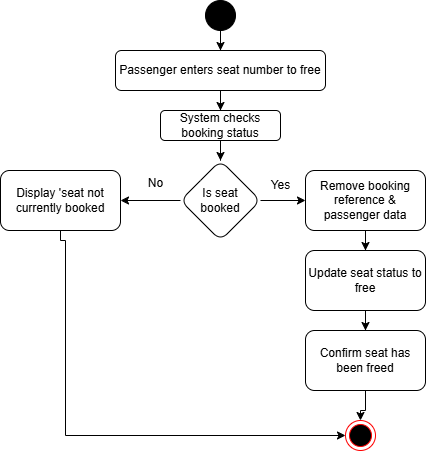
This activity diagram describes the process by which the Apache Airlines Seat Booking System checks seat availability. Initially, the passenger inputs a seat number to query its availability. The system then clearly evaluates whether the seat is free, already booked, or marked as unavailable (such as aisle or storage). Depending on the status, the system displays the appropriate response clearly—informing the passenger if the seat is available, providing the booking reference if booked, or indicating that the seat is unavailable.

**book\_seat (Figure 3)**



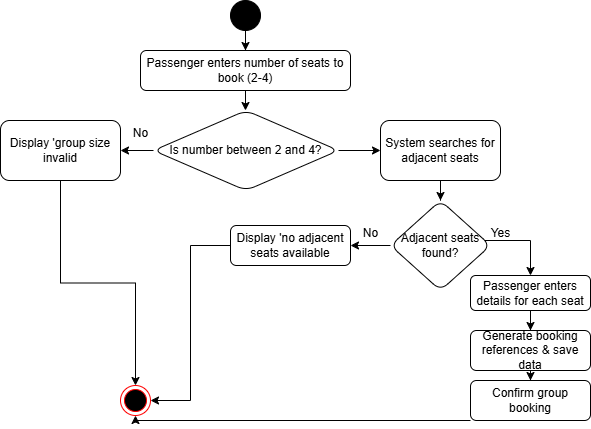
This diagram clearly represents the workflow for booking a single seat within the system. The passenger begins by selecting a desired seat. The system promptly checks availability, and if the seat is available, requests passenger details such as first name, last name, and passport number. Following this, a unique booking reference is generated, records are updated, and a clear booking confirmation is presented to the passenger. Should the seat be unavailable, the passenger receives a clear notification.

**free\_seat (Figure 4)**



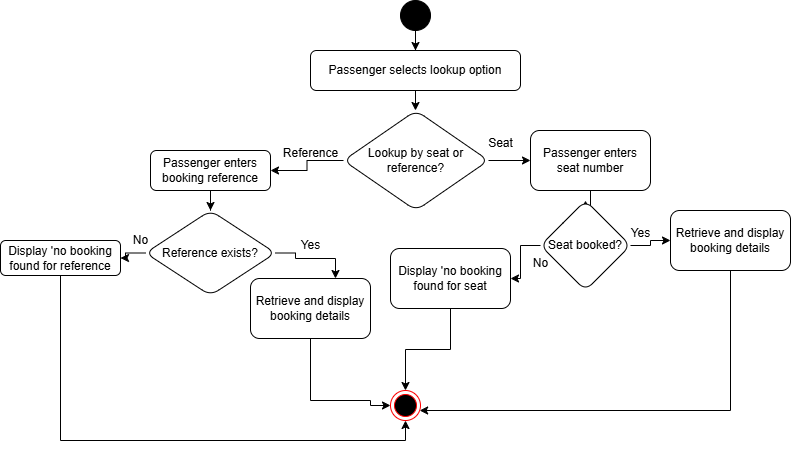
The activity diagram for freeing a seat clearly outlines the steps involved in making a previously booked seat available again. It begins with the passenger providing the seat number they wish to free. The system then verifies whether the seat is currently booked. If the seat is booked, the system clearly proceeds by removing the associated booking reference and passenger details, updating the seat’s status to free, and confirms this change to the user. If the seat was not previously booked, the passenger is clearly informed.

**group\_booking (Figure 5)**



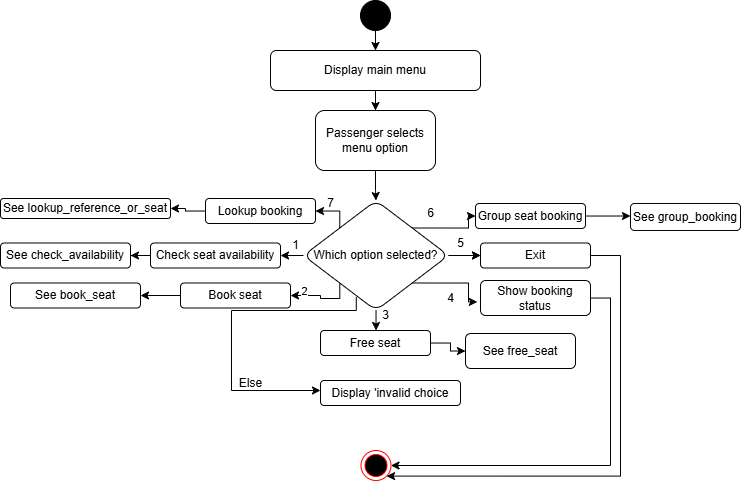
This diagram clearly illustrates the steps involved in booking multiple adjacent seats simultaneously. The passenger specifies the desired group size (between two and four). The system verifies the validity of the request and searches for adjacent available seats. Upon successfully locating suitable seats, passenger details for each seat are sequentially collected, unique booking references are generated, and the system confirms the group booking clearly to the passenger. If adjacent seats cannot be found or the group size is invalid, the system explicitly communicates this information.

**lookup\_reference\_or\_seat (Figure 6)**



The activity diagram for lookup functionality clearly details the process by which passengers retrieve booking details using either a seat number or booking reference. Initially, the passenger chooses their lookup preference. If looking up by seat number, the system checks whether the seat is booked, and if so, displays passenger and booking details clearly. For lookup by booking reference, the system verifies the reference and retrieves associated passenger details if the reference is valid. If no booking details are found for either lookup method, the passenger receives a clear notification.

**Run() (Figure 7)**



The main menu activity diagram clearly represents how users interact with the Apache Airlines Seat Booking System. The system initially presents a clear menu of options, allowing passengers to check availability, book or free seats, view the seat layout, perform group bookings, lookup bookings, or exit the system. Based on the passenger’s selection, the corresponding functionality is executed. Any invalid selections are clearly handled by returning passengers to the main menu with an explicit notification.