**AIRLINE SEAT RESERVATION SYSTEM**

GitHub Repository: https://github.com/adityar225/Airplane-Reservation-System.git

Hey there! I'm Aditya, and this is a command-line seat booking system I built for a take-home assignment.

# About the Project / Project Goal

This project is a command-line based seat reservation system for a fictional airline. The aircraft has 20 rows (A to T), each with 8 seats arranged in a 2-4-2 format. The system allows users to book or cancel seats through CLI commands, while ensuring that seat assignments remain valid and no double bookings occur.

The reservation state is stored persistently in a JSON file, allowing the system to maintain data across multiple executions. The tool is designed to be simple, fast, and non-interactive, following the problem statement's requirements to accept inputs via CLI and return only `SUCCESS` or `FAIL`.

This solution is also modular and extensible, with room for future enhancements such as support for batch bookings, nearest seat suggestions, and separate seat maps for different cabin classes.

Table of Contents

[About the Project / Project Goal 1](#_Toc203098565)

[How to Use 2](#_Toc203098566)

[Core Features 2](#_Toc203098567)

[Design Rationale 3](#_Toc203098568)

[Validation Logic 3](#_Toc203098569)

[What’s in the Folder 3](#_Toc203098570)

[Test Cases 4](#_Toc203098571)

[Assumptions Made 5](#_Toc203098572)

[Final Thought 5](#_Toc203098573)

[Project Planning and Task Tracking 6](#_Toc203098574)

[Tips: 7](#_Toc203098575)

# How to Use

You can run the program like this:

python3 airline.py < BOOK | CANCEL > < SeatNumber > < Count >

Example:

python3 main.py BOOK A2 3

python3 main.py CANCEL A2 3

# Optional: View the current seat map

python3 main.py VIEW

# Core Features

- Book or cancel one or more consecutive seats using a clean command-line interface.

- Seat layout includes 20 rows (A to T) and 8 seats per row (0 - 7) - configurable via `config.py` only if needed in the future

- Seat availability is stored in a JSON file (‘seats.json’), so bookings are remembered across runs.

- Input is validated using a dedicated validation module (‘validator.py’) for row, seat format, bounds, and action type.

- Actual booking and cancellation logic is handled in ‘seats\_utils.py’ with safeguards to prevent overbooking or invalid access.

- Only SUCCESS or FAIL is printed to STDOUT and all warnings and errors are written to ‘error.log’ using Python's logging module.

- Optional ‘VIEW’ command lets you see the current seat map in a 2-4-2 layout, useful for debugging or demonstration.

- Includes unit tests (`test\_airline.py`) to check key scenarios like booking, double booking, invalid cancellation, and seat overflow.

- Modular file structure keeps business logic, validation, configuration, and CLI handling clean and separate.

# Design Rationale

I kept the design modular to make the code clean, easy to maintain, and scalable. Each part of the system has its own role like the main.py handles user input, seats\_utils.py manages booking logic, validator.py checks input validity, and config.py stores constants.

I used a JSON file to store seat data since it's simple, readable and works well with Python. Logging is handled using Python’s built-in logging module to capture errors and warnings in a separate file (error.log) keeping the output clean.

This structure also makes the system easier to test and extend later. If I wanted to add features like seat pricing or a web interface, I could do that without rewriting the core logic.

Validation Logic

- That the row is valid (A to T)

- That the seat number is within bounds (0 - 7)

- That the seat format is correct (e.g. A2)

- That the action is either BOOK or CANCEL

- That all seats in the requested range are valid for the action

If any of these checks fail, the system prints FAIL and exits gracefully.

What’s in the Folder

airline-booking/

├── main.py # Entry point for CLI actions

├── seats\_utils.py # Core booking/cancel logic and seat map display

├── validator.py # Parses and validates user input

├── config.py # Stores constants like ROWS and SEATS\_PER\_ROW

├── test\_airline.py # Unit tests for booking and cancelling

├── seats.json # Auto-generated seat data file

├── error.log # Log file for any invalid inputs or system errors

└── README.txt # This file

Test Cases

|  |  |  |
| --- | --- | --- |
| **Command** | **OUTPUT** | **Notes** |
| BOOK A0 1 | SUCCESS | First booking of seat A0 |
| CANCEL A0 1 | SUCCESS | Successfully cancels A0, making it available again |
| BOOK A0 1 | SUCCESS | A0 was available after cancellation, booking again is valid |
| BOOK A0 1 | FAIL | A0 is already booked, cannot double book |
| BOOK A1 1 | SUCCESS | A1 is available, booking succeeds |
| BOOK A2 4 | SUCCESS | Booking the middle block A2 - A5, all seats are free |
| BOOK A5 1 | FAIL | A5 is already booked as part of the A2-A5 block |
| BOOK A6 3 | FAIL | A6 - A8 exceeds row limit (only seats 0–7 exist per row) |
| BOOK A8 1 | FAIL | A8 is invalid, seat index out of range |
| BOOK U1 1 | FAIL | U is an invalid row, only A to T are valid |
| **OUTSIDE THE GIVEN TEST CASES** |  |  |
| BOOK Q3 -2 | FAIL | Negative seat count - invalid input |
| BOOK B0 0 | FAIL | Invalid: trying to book \*\*0 seats\*\* |
| BOOK D0 10 | FAIL | Booking extra seats than the capacity in one row |
| BOOK E2 4 | SUCCESS | E2 to E6 is available |
| CANCEL K0 2 | FAIL | Trying to cancel seats that were never booked |
| RESERVE A2 3 | FAIL | Invalid input |

Potential Enhancements

* Alternate seat suggestions when requested seats are unavailable
* Aisle breaks or seating zones for realistic seat layout
* Batch command execution using input files
* Seat pricing and revenue tracking
* Admin view to show total booked/available seats per flight

# Assumptions Made

* It is assumed that the flight is a non-stop direct flight (no layovers or segment-specific bookings).
* Only rows A to T (20 rows total) and seats 0 to 7 (8 seats per row) are valid.
* Seat numbers must be consecutive for both booking and cancellation actions.
* The system only accepts two actions: BOOK and CANCEL.
* The system prints only SUCCESS or FAIL to stdout with no extra messages.
* The seats.json file is created on the first run and used to persist seat data across executions.
* The CLI tool assumes a single command per run; batch commands are not supported in this version.

Final Thought

This was a really enjoyable build. It was practical, and a nice balance of constraints and creativity. Thank you for reviewing it!!

- Aditya

# Project Planning and Task Tracking

I used Trello to manage tasks and track progress for the development of this project.

Please find the project planning board at this link: https://trello.com/invite/b/687076dbd4e07749705b592a/ATTIb1614591c9967c8f2886f09a19b685ca3A20D529/airline-seat-reservation-system

If you are unable to access the link or face difficulty signing into Trello, kindly refer to the attached screenshot of the board below.

Screens screenshot of a chat

AI-generated content may be incorrect.

# Tips:

1. **Visibility Tip: View JSON File**

Let users know how they can peek into the seat map (optional for debugging):

Viewing the Seat Map (for debugging)

If you want to see the current seat reservation state, you can open `seats.json` in the bash with this command:

cat seats.json

or use python3 main.py if display function block isn’t commented

1. **Change the rows and seats\_per\_rows ( Scale )**

Type this in bash to change behaviour dynamically

export ROWS=25

export SEATS\_PER\_ROW=10

export SEAT\_FILE=airline\_test.json

python3 airline.py BOOK A0 2

To reset :

unset ROWS

unset SEATS\_PER\_ROW

unset SEAT\_FILE

Advantages : No need to hardcode values or pass extra command-line arguments.

Your code becomes flexible and portable.

You can test multiple configurations (like different planes) without modifying your source code

1. **Make it easier to run in the bash (optional)**

If you prefer not to type `python3` every time, you can make the script directly executable:

Add this to the top of `airline.py`:

#!/usr/bin/env python3

Then run:

chmod +x main.py

After that, you can run it like:

./airline.py BOOK A2 3