**AIRLINE SEAT RESERVATION SYSTEM**

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

# About the Project

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 1](#_Toc203072773)

[How to Use 2](#_Toc203072774)

[Core Features 2](#_Toc203072775)

[Design Rationale 2](#_Toc203072776)

[Validation Logic 2](#_Toc203072777)

[What’s in the Folder 3](#_Toc203072778)

[Test Cases 3](#_Toc203072779)

[Final Thought 4](#_Toc203072780)

[Project Planning and Task Tracking 5](#_Toc203072781)

[Tips: 6](#_Toc203072782)

My main goal here was to build something clean, functional, and easy to understand while making sure I followed the requirement of only printing SUCCESS or FAIL to the terminal.

# How to Use

You can run the program like this:

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

Example:

python3 airline.py BOOK A2 3

python3 airline.py CANCEL A2 3

# Core Features

- Keeps track of seat bookings row by row (A - T), 8 seats per row (0 - 7).

- Stores the seat map in a JSON file, so bookings are remembered between runs.

- Lets you book or cancel a group of consecutive seats but only if all of them are available or booked.

- If anything goes wrong (invalid input, partially unavailable range, etc.), it just prints FAIL - no extra noise.

# Design Rationale

I chose to use JSON for maintaining the seat reservation state because it's lightweight, easy to read, and integrates well with Python for structured data management.

While I didn't include task breakdown tools like JIRA directly in the submission, I did plan the structure and logic in a similar way behind the scenes. I'm happy to walk through that thought process if helpful. I just wanted to keep this version focused and uncluttered.

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

What’s in the Folder

airline-booking/

├── airline.py # The CLI booking tool

├── seats.json # Gets created on first run

└── 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

- Aisle breaks or seating zones

- Logging to a file

- Multiple commands in one run

- Alternate seat suggestions

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

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 airline.py

* After that, you can run it like:

./airline.py BOOK A2 3