Movie Ticket Booking System - Project Report

*Project Title: EasyBook –Movie Ticket Booking System

*Course/Subject:** Python Programming

*Submitted By: BHUMIKA DAS

*Date:** November 23, 2025

1. Cover Page

(Already covered above)

2. Introduction

EasyBook is a simple console-based Movie Ticket Booking System developed in Python. The objective of this project is to demonstrate core programming concepts such as functions, dictionaries, lists, control structures, and basic user authentication while simulating a real-world online ticket booking experience.

The system allows users to:

- Register and log in
- View available movie shows
- Book tickets
- View their bookings
- Cancel bookings
- Automatically update seat availability

This project is entirely text-based and uses in-memory data structures (no external database).

3. Problem Statement

Design and implement a menu-driven console application that enables users to book, view, and cancel movie tickets with the following constraints:

- Multiple shows of different movies on different dates and times
- Limited seats per show (initially 60)
- User authentication (registration + login)
- Prevent over-booking
- Allow cancellation with seat count restoration
- Simple and intuitive interface

4. Functional Requirements

1. User Registration
2. User Login
3. View all available shows (only shows with seats > 0)
4. Book tickets for a selected show
5. View personal booking history
6. Cancel an existing booking and restore seats
7. Logout and Exit functionality

5. Non-Functional Requirements

- Response time: Instant (console-based)
- Usability: Clear menus and formatted output
- Reliability: Proper input validation and error handling
- Security: Basic password protection (stored in plain text – for educational purpose only)
- Scalability: Limited (in-memory storage)
- Portability: Runs on any system with Python 3.x

6. System Architecture

The system follows a simple procedural architecture with modular functions:

Main Loop

├— Registration Module

├— Login Module

└— User Session Loop

  ├— View Shows

  ├— Book Ticket

  ├— My Bookings

  └— Cancel Booking

Data is stored in three global structures:

- `shows` → List of dictionaries (show details)

- `users` → Dictionary {username: password}

- `bookings` → Dictionary {username: list of booking strings}

No external libraries or database used.

7. Design Diagrams

○ Use Case Diagram (Textual representation)

Actors: Guest User, Registered User

Guest User:

- Register
- Login
- Exit

Registered User:

- View Available Shows
- Book Ticket
- View My Bookings
- Cancel Booking
- Logout

○ **Workflow Diagra**m (Main Menu Flow)

Start → Clear Screen → Display Main Menu

├— 1 → Register → Back to Main Menu

├— 2 → Login → Success → User Menu

| └→ Fail → Back to Main Menu

└— 3 → Exit

User Menu:

├— 1 → View Shows → Book Ticket

├— 2 → My Bookings

├— 3 → Cancel Booking

└── 4 → Logout → Main Menu

```

○ **Sequence Diagram** (Booking a Ticket)

User → System: Select "Book Ticket"

System → User: Display available shows

User → System: Enter Show ID & No. of tickets

System: Validate show existence & seat availability

System → Database: Reduce seats_left

System → Database: Add booking to user's bookings

System → User: "BOOKED SUCCESSFULLY! Total: ₹XXX"

○ Class/Component Diagram

(No classes used – procedural design)

Main components (functions):

- `register()`, `login()`, `book_ticket()`, `my_bookings()`, `cancel_booking()`, `view_shows()`

○ ER Diagram

Not applicable (no persistent database). Data is stored in Python dictionaries/lists.

8. Design Decisions & Rationale

| Decision | Rationale |
|---|---|
| In-memory storage (dicts/lists) | Simplicity and no external dependencies |
```

| Plain text password storage | Educational project – real systems must hash passwords

| Global data structures | Easy access across functions without passing parameters

| Procedural over OOP | Keeps code simple and readable for beginners

| Fixed price ₹200 per ticket | Simplifies calculation logic

| Booking stored as formatted string | Easy display in "My Bookings" without complex objects

| Input validation using try-except| Prevents crashes from non-integer inputs

9. Implementation Details

- Language: Python 3.x

- IDE: Any (VS Code, PyCharm, IDLE, etc.)

- Key Techniques Used:

  - Dictionaries for fast user lookup

  - List comprehension and `next()` with generator for finding shows

  - String parsing during cancellation to restore correct seats

  - `clear_screen()` using print("\n"*50) for better UX

  - Formatted printing using f-strings and .center(), .ljust()

10. Screenshots / Results

```
==================================================
            EASYBOOK - MOVIE TICKETS
==================================================
1. Register
2. Login
3. Exit
Choose (1-3): 1

==================================================
                    REGISTER
==================================================
Enter username: BHUMIKA
Enter password: 2334
Registration successful!

Press Enter to continue...
```

```
1. Book Ticket
2.My Bookings
3. Cancel Booking
4. Logout
Choose (1-4): 1

AVAILABLE SHOWS
--------------------------------------------------
-----------------
ID   Movie                Date         Time
Seats Left
--------------------------------------------------
-----------------
1    Avatar 3             2025-11-25   10:00 AM
60
2    Avatar 3             2025-11-25   06:00 PM
60
3    Wicked               2025-11-26   02:00 PM
60
4    Deadpool 3           2025-11-27   09:00 PM
60
5    Moana 2              2025-11-28   11:00 AM
60
--------------------------------------------------
-----------------

Enter Show ID: 1
Number of tickets: 2

BOOKED SUCCESSFULLY! Total: ₹400
```

11. Testing Approach

- Manual testing of all user flows
- Edge cases tested:
  - Booking more tickets than available
  - Cancelling non-existent booking
  - Invalid show ID / non-integer input
  - Registering duplicate username
  - Logging in with wrong credentials
  - Cancelling and re-booking same show (seat count correctly restored)

All critical paths verified working.

12. Challenges Faced

1. Parsing booking string during cancellation to restore correct seat count
   → Solved using `split(" | ")` and extracting ticket count
2. Matching exact show during cancellation (same movie can have multiple shows)
   → Solved by comparing movie + date + time string
3. Screen clutter in console
   → Implemented `clear_screen()` and formatted headers
4. Global variable mutation risk
   → Accepted for simplicity in educational context

13. Learnings & Key Takeaways

- Importance of data structure choice (dicts for O(1) lookup)

- Input validation is critical in console apps
- Separation of concerns using functions improves readability
- Real-world systems need persistent storage and proper security
- User experience matters even in simple console applications
- String parsing can be powerful but fragile

14. Future Enhancements

1. Convert to Object-Oriented design (User, Show, Booking classes)
2. Add persistent storage using JSON/file database or SQLite
3. Seat selection (specific seat numbers, 2D grid)
4. Admin panel (add/remove shows)
5. Payment gateway simulation
6. Email/SMS confirmation
7. GUI using Tkinter or web version using Flask/Django
8. Password hashing using `bcrypt`
9. Search and filter shows by movie/date
10. Multiple theaters and locations

15. References

- Python Official Documentation – https://docs.python.org/3/
- GeeksforGeeks – Python Dictionary & List tutorials
- Real Python – Formatting strings in Python
- Lecture notes on procedural programming and console applications

Declaration:

I hereby declare that this project is my original work and has been developed solely for academic purposes.