REPORT

Title : Python-Tkinter Based Real-Time Movie Ticket Booking System with Firebase Integration

NAME : TM SREENAND
R HARI NARAYANAN
N RAMAVARSHINI

CONTENT

1 Abstract

This project is a full-featured, desktop-based movie ticket booking application developed in Python. It utilizes the tkinter library to create a dynamic and user-friendly graphical user interface (GUI) and integrates Google's Firebase Realtime Database as its backend. This client-server architecture allows for real-time data synchronization, ensuring that all information, such as seat availability and pricing, is always up-to-date across all running instances of the application.

2 Introduction

2.1 Project Overview

The Movie Ticket Booking System is a system which is developed to provide a secure, interactive platform for users to book movie tickets with enhanced features compared to traditional systems. It integrates a Tkinter-based GUI with Firebase backend and incorporates advanced concepts like dynamic pricing, last-minute ticket bidding, and pay-later booking options. The system addresses inefficiencies in typical booking platforms by ensuring real-time seat updates, secure login, and fair ticket allocation through bidding on cancelled seats.

2.2 Problem Statement

Conventional movie ticket booking systems fails to accommodate fluctuating demand effectively due to lack of dynamic pricing and flexible payment options. In addition to that, they do not handle canceled tickets efficiently, resulting in wasted opportunities and customer dissatisfaction and increase the chances of tickets to get sold in black. This project seeks to bridge these gaps using data structures and algorithms to optimize seat management, pricing, and ticket allocation dynamically.

2.3 System Objectives

1. Design a user-friendly interface with real-time updates on seat availability and pricing.
2. Implement robust user authentication with secure password management.
3. Develop flexible payment options including pay-now and pay-later with automated reminders.
4. Introduce a bidding system for cancelled tickets to maximize user engagement and revenue.
5. Maintain data consistency through automated cleanup and synchronization mechanisms.
6. Deliver a polished, responsive GUI enhancing overall user experience.

2.4 Technology Stack Used

1. Frontend: Python Tkinter GUI for the user interface,messagebox,simpledialog,Toplevel for dialog and pop-up messages
2. Backend: Firebase Realtime Database for storing data.

3. Security: Password hashing (bcrypt)
4. Core Libraries: Python's datetime, uuid, regex, and os modules

3 System Architecture

3.1 Overall System Design

The application works like a client-server model. Tkinter acts as the client where users interact and books tickets through a graphical interface, and Firebase is the server that keeps all data safe and updated in real time. Data structures like arrays and hash maps help keep data organized and easy to access.

3.2 Frontend (GUI) Components

Built using Tkinter, the frontend offers a suite of dedicated screens (Main Menu, Login, Signup, Show Listings, Booking, Admin Panel). Every screen is realized as a separate Frame, facilitating clear navigation and modularity. Two custom widgets, NeonButton and ValidatingEntry, further enhance usability by providing visual feedback and input validation marks, thus minimizing errors and improving the user experience.

3.3 Backend Database Integration

The backend leverages Firebase Realtime Database, where data is organized in nested dictionaries and lists. Entities such as users, shows, bookings, transactions, and notifications are structured for quick access and modification. All backend logic, including booking management, seat allocation, notifications, and transaction logging, is performed via direct calls to Firebase APIs from the Python codebase, ensuring live synchronization. This architecture ensures that none of the data is lost or overwritten concurrently by any other user (avoiding race condition) therein improving the atomicity of the application ,the atomicity is ensured automatically by the firebase.(Fig : 1).

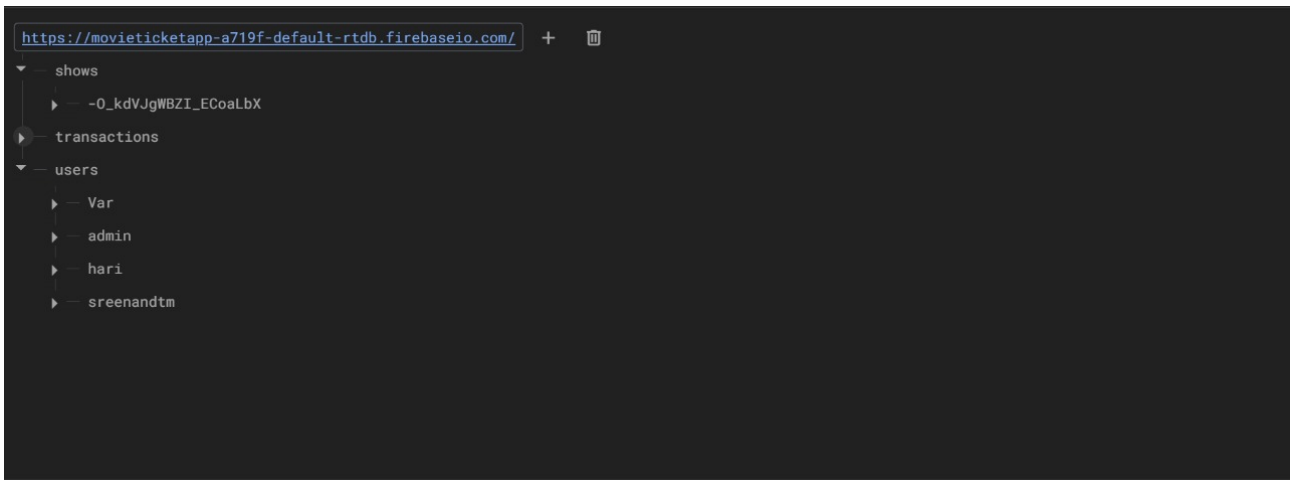

Figure 1: firebase dataset

3.4 Authentication System

The authentication subsystem guarantees that only verified users and admins can access the application. Passwords are hashed with bcrypt and then stored in the firebase database , and the login process checks the stored bcrypt hash upon attempts. Session management is handled in-memory by the Tkinter app, maintaining the current user's identity during the active session.

4 Data Structures Implementation

4.1 User Management Data Structure

User data is maintained as Firebase objects where each user has a unique record. The user structure features fields such as encrypted password, user role (admin or regular user), and associated bookings.

4.2 Show/Movie Data Organization

A dedicated data structure stores all movie shows, including metadata like movie title, showtime, language, base price, total seat count, and bookings status. Seats are managed as dictionary keys within each shows object, allowing fast lookup and atomic updates in both booking and cancel operations. Bid data and pay-later reservations are distributed within the same shows node, improving integrity and reducing latency.

4.3 Booking Management System

Bookings are referenced per-user and per-show, consolidating seat numbers, payment status, and expiry information where applicable. Each booking is timestamped and can carry multiple seats in a single logical object, facilitating batch management and reducing redundancies in data storage.

4.4 Seat Allocation Arrays

Seats are represented as arrays (Python lists) and dictionaries for efficient seat tracking. Booked and reserved seats are stored as dictionary keys for instantaneous verification of seat status, enabling concurrency-safe operations during seat selection and booking processes.

4.5 Transaction History Storage

Transactions are logged with universally unique identifiers (UUIDs), storing details such as username, movie title, showtime, seats, status, and payment amount. These records form a comprehensive audit trail, enabling transparency and accountability across all user and administrative actions.

5 Core System Features

5.1 User Registration and Authentication

Robust input validation and bcrypt encryption are central to the registration and login process. Usernames must be unique, and passwords must satisfy complexity requirements. Upon successful signup, passwords are hashed and stored exclusively in encrypted (bcrypt)format in the firebase , ensuring high levels of security. All credential checks during login use bcrypt comparison functions, minimizing the risk of unauthorized access.

5.2 Movie Show Management

Admins are empowered to create and delete movie shows, with advanced validation blocking the addition of past events or incomplete data entries. Shows are dynamically listed with all relevant information, and past shows are periodically cleaned up from the database using a dedicated function, keeping the system up to date and uncluttered. The system makes sure the show times are correct and deletes old shows automatically.

5.3 Ticket Booking System

The main operation of the application lies in its booking module, which verifies user credentials, seat selection, and payment status before confirming ticket allocation. Adaptive validation rules ensure seat numbers fall within valid bounds, eliminate duplicates, and prevent overbooking of reserved/booked seats. Post-booking, seats are marked as occupied or reserved as per payment method, and all transactions are logged for audit purposes.(Fig : 2)

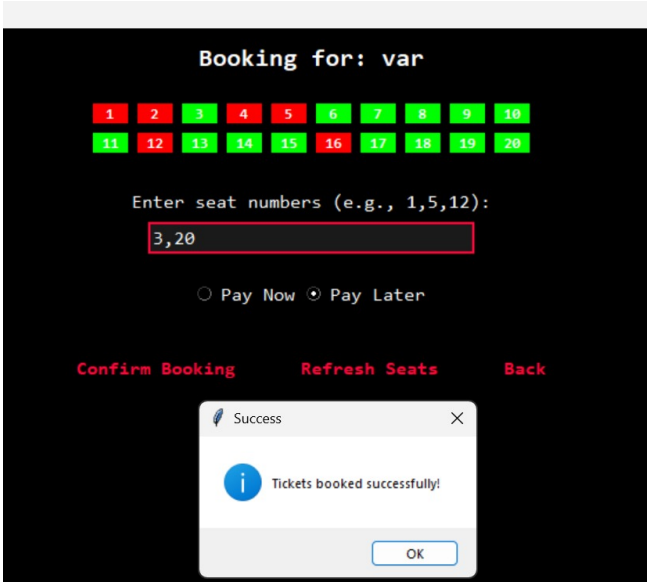

Figure 2: User Ticket Booking

5.4 Dynamic Pricing Algorithm

Ticket prices are not static; instead, they evolve in response to current seat occupancy. The algorithm multiplies the base ticket price by a scaling factor dependent on the occupancy percentage, thus encouraging early bookings and maximizing revenue for near-full shows. Administrators can monitor real-time pricing changes directly from the admin dashboard.The formula for dynamic pricing is New Price=Base Price$\times$(1+0.005$\times$Occupancy Percentage)

5.5 Payment Processing (Pay Now/Pay Later)

Users can pay immediately to confirm bookings or reserve seats with a deadline. The system recognizes both immediate ('Pay Now') and deferred ('Pay Later') payment models. 'Pay Now' tickets allocate seats instantly and close the transaction as paid, while 'Pay Later' reservations introduce expiry times and global seat limits to minimize misuse. Expiring reservations are tracked with automatic deadlines and reminders, optimizing resource utilization and user satisfaction.

5.6 Seat Bidding System

If someone cancels, that seat goes into a bidding pool, users may bid for formerly reserved seats. The system enforces a minimum bid increment and prevents repeated self-bidding, ensuring fair competition. Bidding closes after a designated timeout period, and the highest bidder is automatically awarded the seat. Completed transactions and bid histories are archived in Firebase for future reference.

5.7 Automatic Show Cleanup

To prevent database bloat, regularly scheduled routines delete outdated movies and related bookings. The cleanup module cross-references showtimes to the current system date.(Fig:3)

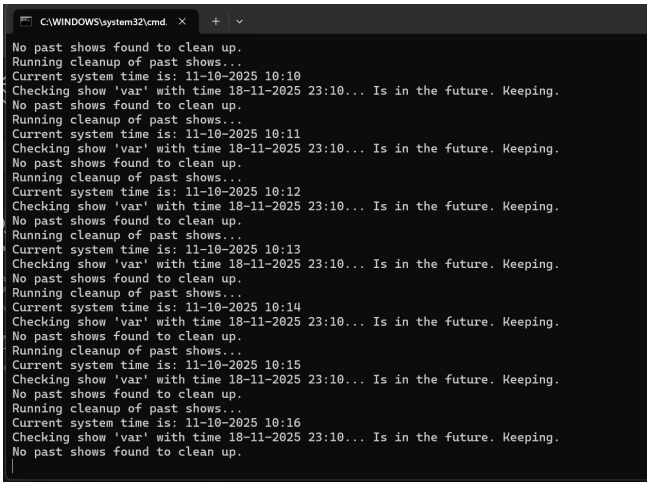

Figure 3: Past Show Clean Up

6 Advanced Features

6.1 Real-time Seat Availability

The system provides a constantly updated visualization of seat availability, enabling users to make highly informed booking decisions. The GUI employs a color-coded layout where each seat's status like booked, reserved (for pay-later), or available are distinguishable by color such as red for booked, orange for reserved, and green for available. This dynamic representation directly reflects the current state of the backend database, ensuring users and administrators always get the latest view without delays or outdated information. Leveraging this mechanism, double-bookings and conflicts are effectively minimized, and the booking process remains intuitive and transparent.(Fig : 4).

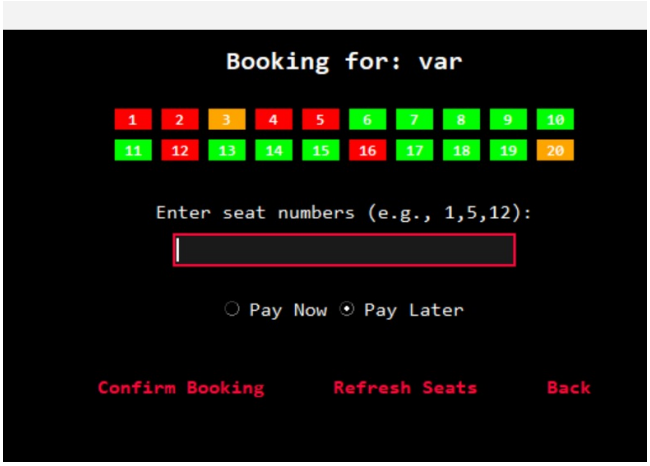

Figure 4: Display of tickets

6.2 Deadline Management for Reserved Seats

To ensure optimal resource utilization and prevent misuse of the 'Pay Later' feature, the system includes a sophisticated deadline management mechanism for reserved seats. Each pay-later booking is associated with a clear expiry time, after which, if payment hasn't been completed, the reservation is automatically canceled. Built-in background jobs, running via Tkinter's after scheduling, periodically check for expired reservations and instantly release unclaimed seats back into the pool.(Fig : 5,6)

6.3 Transaction Logging

A vital aspect of the system's transparency and auditability is its transaction logging engine. Every significant action—ticket booking, payment, cancellation, bid placement, or administrative operation—is recorded as a unique transaction, complete with details such as timestamp, involved parties, seats, payment status, and total amount. This not only guarantees a reliable history for accountability but also allows users and administrators to review past activity, verify payments.(Fig : 7)

Figure 5: Deadline Management

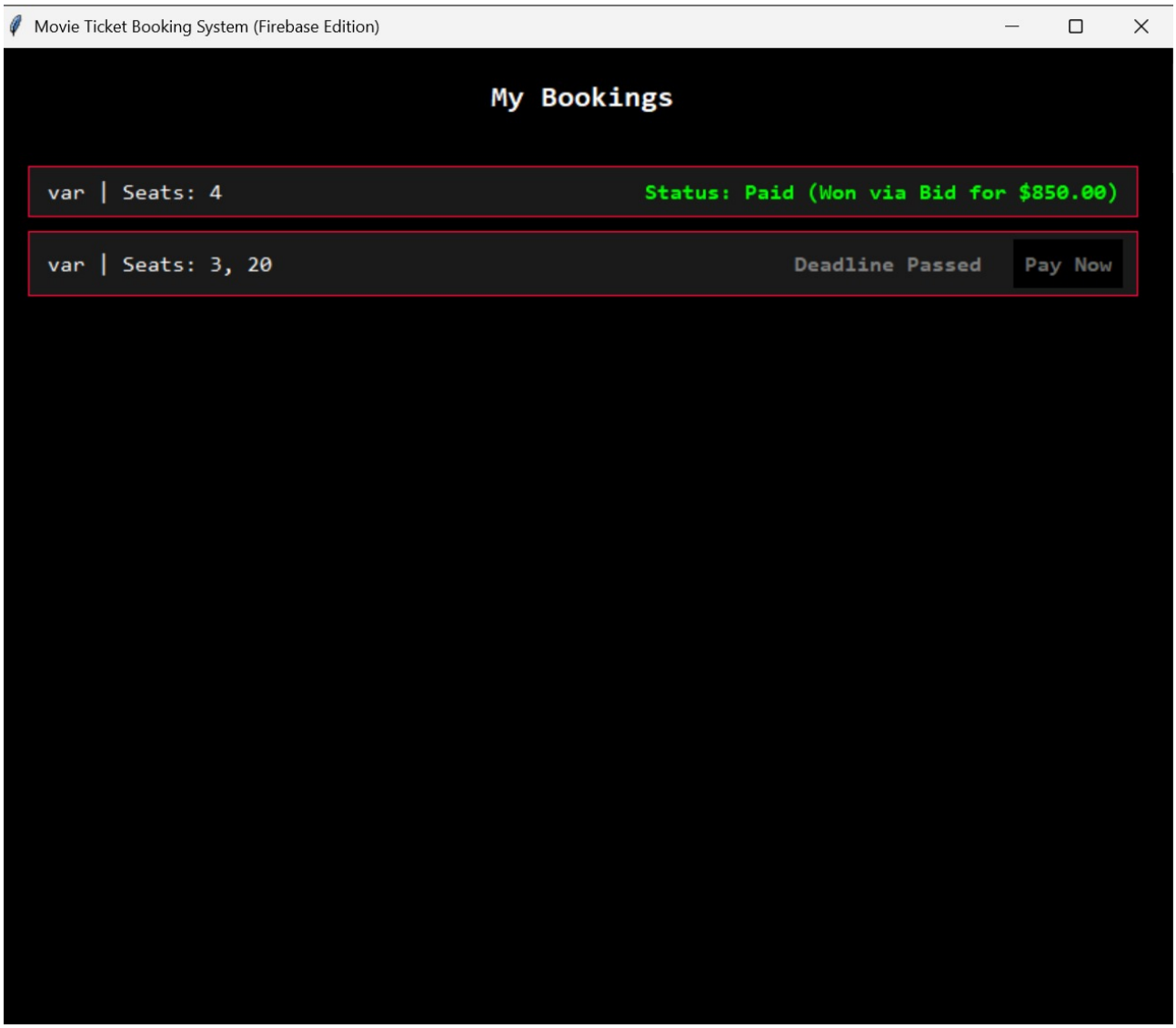

Figure 6: Deadline Management

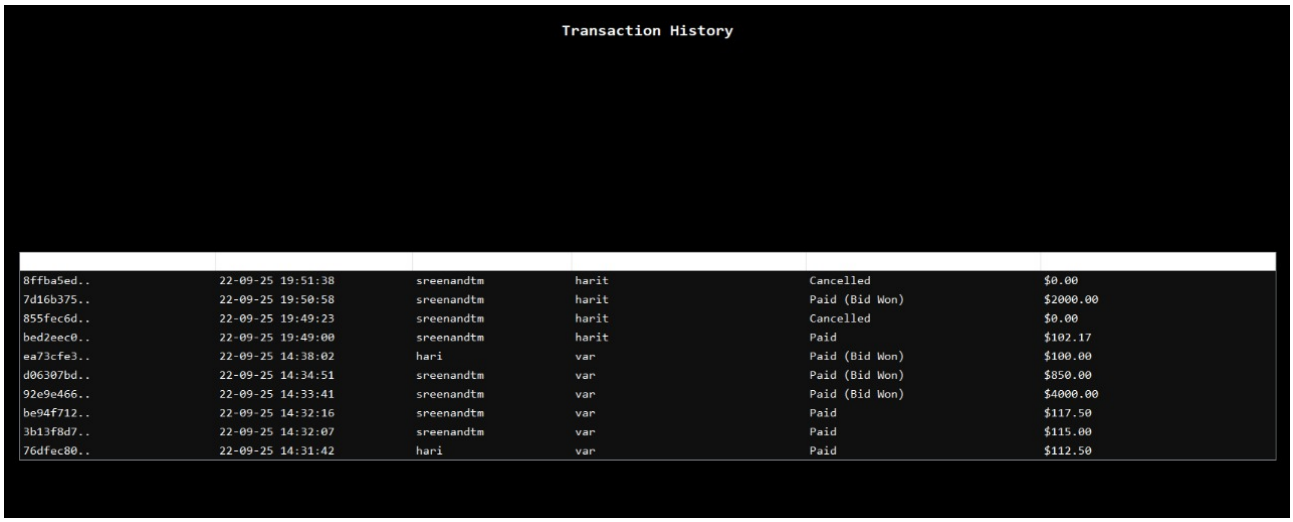

Figure 7: Transaction Logging : Admin Panel

6.4 Admin Management Panel

Administrative users have access to a comprehensive management dashboard designed for overseeing the entire ticketing ecosystem. Within this dedicated section, admins can manage shows (add/remove), view all user bookings, resolve bid battles for cancelled seats, monitor real-time transaction history. The panel is constructed to empower administrators with clear data visibility and efficient control mechanisms.(Fig : 8).

7 GUI Implementation

7.1 Main Menu Design

The main menu is designed as the principal navigation hub for both new and returning users. It offers clearly labeled options for logging in, signing up, viewing show listings, and exiting, ensuring first impression.Use of Tkinter frames allows for instant screen transitions and maintains context for back navigation, preserving user workflow and enhancing the overall usability.(Fig : 9)

7.2 User Interface Components

Each role is served by purpose-built Tkinter Frames that encapsulate specific workflows and minimize distractions. All data-intensive displays such as ticket bookings, show listings, and the transaction log—are rendered using Table (Treeview) widgets for clarity and interactivity. Users receive input prompts through dialog boxes and real-time error flags, reducing frustration and promoting accurate data entry. Feedback on successful or failed actions is provided visually and textually, helping users correct mistakes quickly.(Fig : 11)

7.3 Custom Widget Classes

Key to the user experience are custom widget classes like NeonButton which provides distinctive, neon-effect action buttons with dynamic hover states—and ValidatingEntry, which

highlights valid or invalid user input actively as the user types. These enhancements are not merely aesthetic; they provide immediate feedback, reduce error rates, and make the GUI visually modern and memorable.

7.4 Event Handling System

Central to the app's responsiveness is its integrated event handling framework, where all button clicks, input changes, and frame transitions are tightly bound to their respective Python functions and methods. Tkinter's event system is leveraged for both direct user actions and scheduled background tasks, such as checking pay-later deadlines and updating seat maps. The application's structure ensures that every possible event—user-initiated or automatic—is managed in real time and processed without blocking or lag, resulting in a seamless user journey from login to logout.(Fig : 12,13)

8 Database Operations

8.1 Firebase Integration

Upon program initialization, the app loads database credentials from a JSON file and establishes a secure connection to a cloud-hosted Firebase instance. All operations such creating users, storing bookings, updating seat status, logging transactions, and pushing notifications are performed through the official Firebase Python API, which ensures reliability, redundancy, and real-time data updates accessible to all clients.(Fig : 10)

8.2 Data Validation and Error Handling

Every database-modifying operation is preceded by detailed input validation using both regular expressions and programmatic checks. Should any error (from user input, network failure, or logic bugs) occur during these operations, the system responds with user-friendly error popups and, in many cases, gracefully rolls back incomplete changes. This guarantees data consistency and minimizes user impact from transient or unexpected issues, reflecting the robust error handling philosophy built into every layer of the application.(Fig : 12)

8.3 Atomic Transactions

Performance and data consistency, especially during highly concurrent seat bookings and bid resolution, are safeguarded by Firebase's transactional update mechanism. By wrapping high-conflict operations such as confirming a booking, placing a bid, or marking a cancellation in atomic transactions, the system ensures that race conditions or network aberrations do not result in inconsistent data states or double allocations. These guarantees are crucial for an environment where multiple users may interact with the same resource nearly simultaneously.

8.4 Real-time Data Synchronization

A core benefit of Firebase's architecture is instant data propagation. Any change committed to the backend be it a new booking, changed seat status, or sent notification is available in real time to every connected client. Combined with the GUI's periodic refreshing behaviors, this creates a truly live system where users and admins always interact with the most current data, dramatically reducing the confusion and errors typically seen in older, batch based systems.

9 Security Features

9.1 Password Encryption (Bcrypt)

Security is non-negotiable; all user and admin passwords undergo bcrypt hashing with robust salting before storage, eliminating the risk of password leakage or simple dictionary attacks. The login and signup processes are tightly integrated with bcrypt's hash generation and verification routines, ensuring no plaintext password is ever read or stored.

9.2 Input Validation

All externally facing input fields are subjected to thorough validation routines. Passwords must meet strict complexity rules, usernames must be unique and within length constraints, and seat selection is checked for validity and duplicate entries. Invalid entries generate immediate, visually prominent feedback, sharply reducing common errors.(Fig : 14)

9.3 User Session Management

The application tracks active sessions purely in local (in-memory) state using Tkinter, differentiating between user and admin roles. Sessions are securely wiped upon logout, and no session information is persisted across restarts or exposed outside the local process.

10 Testing and Validation

10.1 Error Handling Mechanisms

Popups and error dialogs provide immediate feedback when failures occur, and in critical database sections, failed operations are detected early and either rolled back or isolated from the main data set, keeping service interruptions minimal and confusion rare.(Fig :15)

10.2 Data Integrity Checks

Each ticket, seat status, and booking undergoes multi-level integrity checks to thwart duplication, collision, or out-of-range values. Before a seat is booked or payment processed, its present status is verified directly against the most recent backend state. This ensures that even in edge-case scenarios with network lag or multiple users, system invariants are always maintained and no invalid state enters the system.

10.3 User Input Validation

Input fields such as signup, booking, payment, show addition deploy advanced real-time validation using regular expressions and custom logic. Mistyped usernames, insecure passwords, or ill-formatted seat selections are instantly caught and highlighted, significantly elevating user experience and trust. The system's combination of frontend and backend checks means that only high-quality, accurate data is ever allowed into the database.

11 Conclusion

11.1 Key Achievements

The Movie Ticket Booking System delivers a robust, user-centric experience that blends modern technology and best practices in software engineering. Dynamic pricing, secure authentication, rich transaction history, live seat visualization, deadline and bidding-based booking mechanisms, and comprehensive admin tooling establish this platform as a benchmark for similar projects.

11.2 System Benefits

The key benefits manifested by this system include real-time reliability, enhanced security, operational efficiency, clarity of roles and permissions, and transparency in all transactions. Its scalability, cloud-first design, and deep validation make it suitable for anything from a single cinema to a multiplex chain, with administrative tools to support both growth and day-to-day stability.

11.3 Future Enhancements

Plan to add mobile support, AI for seat suggestions, more payment options, and support more languages.And other improvements include integration with online payment gateways to further streamline transactions, the development of a mobile-responsive or web-based frontend to expand accessibility, real-time analytics dashboards for advanced admin insights.

12 Bibliography

- https://ieeexplore.ieee.org/document/9468015
- https://www.researchgate.net/publication/359158596_A_Systematic_Literature_Review_of_Dynamic_Pricing_Strategies

13 Reference Figures

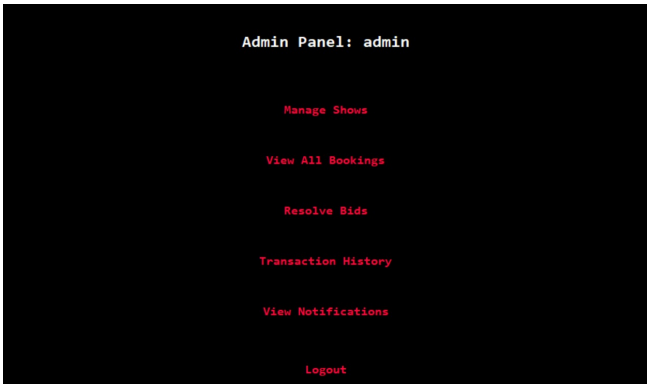

Figure 8: Admin Panel

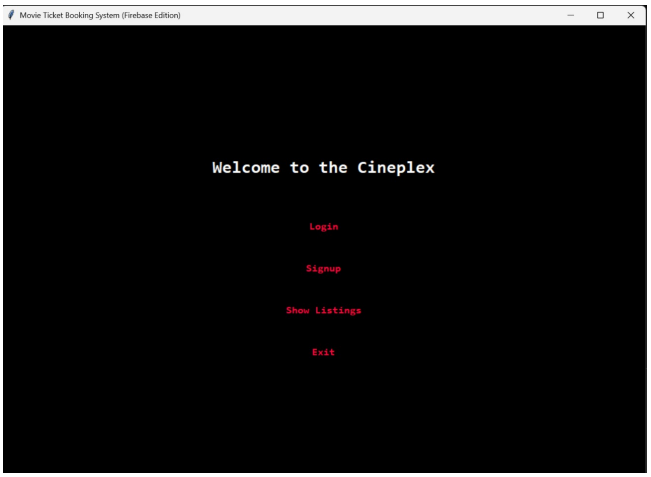

Figure 9: Main Menu Design

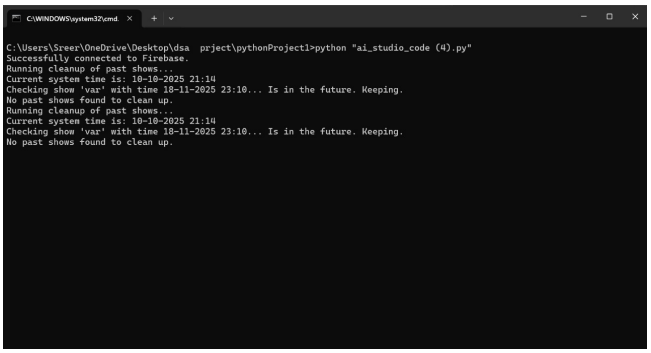

Figure 10: Firebase initalization

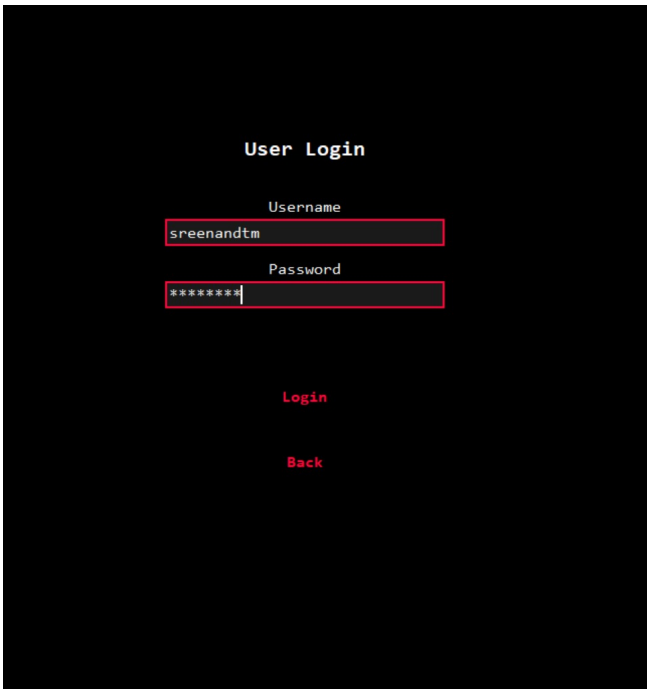

Figure 11: User Login

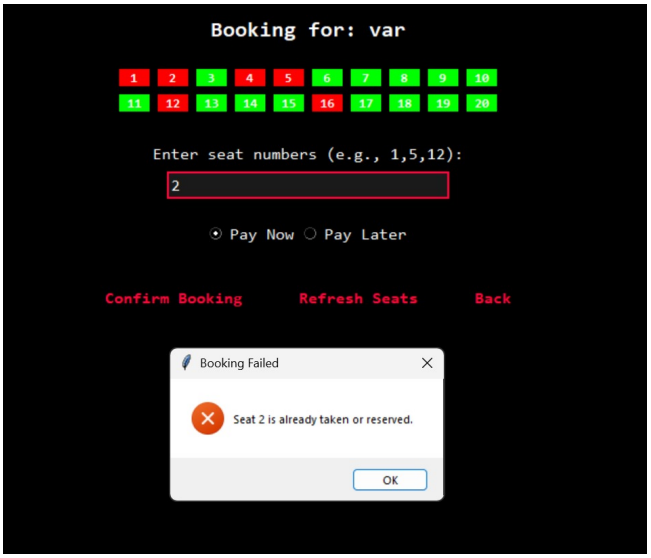

Figure 12: Error Handling

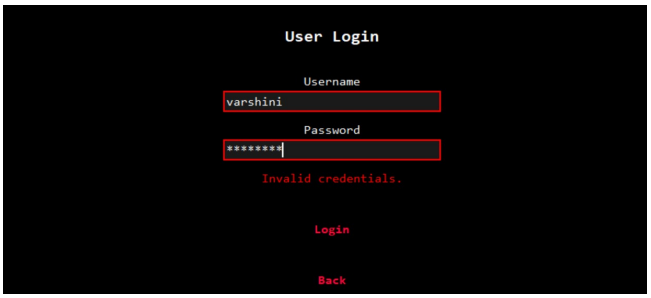

Figure 13: Error handling

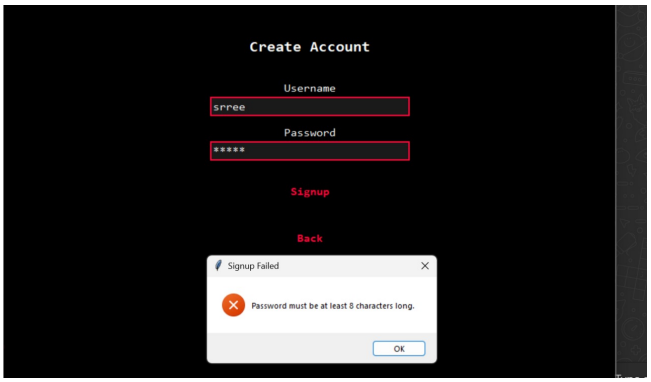

Figure 14: Input Validation

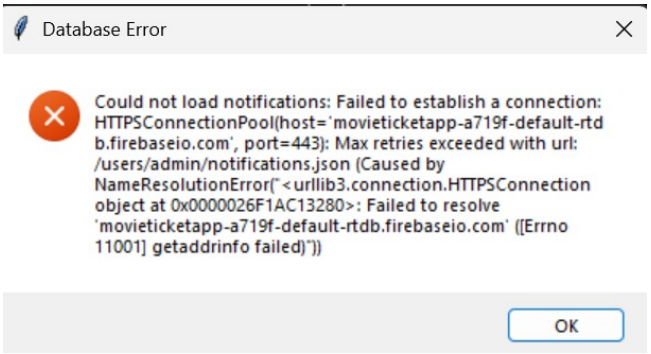

Figure 15: Error Handling Mechanism