CINEMA BOOKING SYSTEM

IBRAHIM AMR

TABLE OF CONTENTS

1️⃣ Executive Summary

The **Cinema Booking System** is a full-stack web application designed to digitize and automate cinema operations.
It provides **secure ticket booking**, **seat selection**, **QR-based tickets**, **food ordering**, and a **comprehensive admin management panel**.

The system is developed using **Flask (Python)** as a backend framework and **MongoDB** as a NoSQL database, following a **layered modular architecture**.

2️⃣ Problem Statement

Traditional cinema booking systems suffer from:

- Manual seat booking errors
- Lack of centralized management
- Poor user experience
- No real-time promotion handling
- Fragmented admin control

This system addresses these issues by providing:

- Automated seat availability checks
- Centralized data management
- Role-based access control
- Integrated promotions and food services

3️⃣ Project Overview

The system supports two main domains:

🎬 Customer Domain

- Cinema selection
- Movie browsing
- Ticket booking
- QR ticket generation
- Food ordering
- Profile management

👨‍💼 Administration Domain

- Movies & screenings management
- Cinema branches management
- Promotions & food management
- User & admin accounts management
- Global booking overview

4️⃣ System Goals & Objectives

- Provide seamless booking experience
- Ensure data integrity
- Support multiple cinema branches
- Enable role-based access
- Facilitate scalability
- Maintain clean code organization

5️⃣ System Scope

✓ Included

- Authentication system
- Cinema-movie mapping
- Seat booking logic
- Promotions (B1G1F, B2G1F)
- QR code ticketing
- Food ordering
- Admin dashboard

❌ Excluded

- Online payment gateway
- Seat locking with WebSockets
- Mobile application

6️⃣ System Actors

Actor	Description
Guest	Visitor without authentication
User	Registered cinema customer
Admin	Cinema staff administrator
System Admin	Super admin managing admins

7️⃣ Overall System Description

The system follows a **multi-layered architecture**:

User → UI → Flask Controllers → Business Logic → MongoDB

Each layer is independent, improving:

- Maintainability
- Scalability
- Testability

8️⃣ Functional Requirements (Detailed)

User Requirements

- Register with strong password rules
- Login with JWT-based authentication
- Select cinema branch
- Browse movies by cinema
- View screenings and promotions
- Book seats securely
- Receive QR ticket
- Order food items
- Manage profile and credentials

Admin Requirements

- Add/Edit/Delete movies
- Schedule screenings
- Manage cinema branches
- Control promotions
- Manage food items
- View and manage users
- Monitor all bookings
- Manage admin accounts (System Admin)

9️⃣ Non-Functional Requirements (Detailed)

Category	Requirement
Security	Password hashing, JWT
Performance	Fast DB queries
Availability	24/7 access
Scalability	Modular architecture
Maintainability	Clean separation of concerns
Usability	Simple UI

🔟 System Architecture

The system uses a **3-Tier Architecture**:

Presentation Layer

- HTML / Jinja templates
- CSS & JavaScript

Application Layer

- Flask routes
- Business logic modules
- Admin & user controllers

Data Layer

- MongoDB collections

---

11 Admin Subsystem Architecture

Admin functionality is isolated as a **logical subsystem**, ensuring:

- Security
- Role isolation
- Clear responsibility boundaries

Admin operations directly interact with:

- Users
- Movies
- Screenings
- Promotions
- Food items
- Bookings

---

12 Data Flow Description

1. User submits request
2. Flask validates input
3. Business logic executed
4. MongoDB queried or updated
5. Response rendered to UI

---

13 Database Design & Schema

Users Collection

```
{
  "_id": ObjectId,
  "name": "User",
  "email": "user@mail.com",
  "password": "hashed",
  "is_admin": false,
  "bookings": []
}
```

Bookings Collection

```
{
  "user_id": ObjectId,
  "movie_id": ObjectId,
  "screening_id": ObjectId,
  "seats": ["A1","A2"],
  "total_price": 20,
  "status": "confirmed"
}
```

1️⃣4️⃣ Module-by-Module Description

Authentication Module

Handles login, registration, JWT validation, and role checks.

Booking Module

- Seat validation
- Promotion application
- QR generation

Food Module

- Menu loading
- Order calculation
- Order persistence

Profile Module

- Personal data update
- Profile picture handling
- Password change

Admin Module

- Full CRUD operations
- Dashboard analytics
- Admin role management

15. User Journey (End-to-End)

User registers .1

Logs in .2

Selects cinema .3

Browses movies .4

Chooses screening .5

Selects seats .6

Applies promotion .7

Confirms booking .8

Receives QR ticket .9

Orders food (optional) .10

---

16. Admin Journey (End-to-End)

Admin logs in .1

Opens dashboard .2

Manages system entities .3

Monitors bookings .4

Updates content dynamically .5

---

17. Authentication & Authorization Model

- JWT token per session
- Role-based access
- Admin-only routes protected
- System Admin privileges isolated

---

18. Validation & Error Handling

- Form validation
- Password strength enforcement
- Seat duplication prevention
- Permission checks
- Graceful error messages

---

19. Security Considerations

- Password hashing
- Session validation
- File upload restrictions
- Admin action isolation

---

20. Performance Considerations

- Indexed MongoDB collections
- Efficient aggregation pipelines
- Reduced DB calls

---

21. Deployment & Environment Setup

pip install -r requirements.txt

python app.py

MongoDB:

mongodb://localhost:27017/cinema_db

---

22 Testing Strategy

- Unit testing for booking logic
- Manual UI testing
- Admin workflow testing

---

23 Limitations & Assumptions

- Assumes stable internet
- Single-region deployment
- No third-party payment integration

---

24 Future Enhancements

- Online payments
- Mobile app
- Seat locking
- Recommendation system
- Analytics dashboard

---

25. Conclusion

This Cinema Booking System is a **robust, secure, and scalable solution** that demonstrates strong system design principles, modular architecture, and real-world applicability.