Movie Theater System

Architecture Design w/Data Management

Version 1.0

04/24/25

Group 13
Ines Ling Delgado Dominguez
Silvia Perez Servando

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Fall 2023

Revision History

Date	Description	Author	Comments
<04/24/25>	<Version 1>	<Group-13>	<First Revision>

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date
	<Your Name>	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

This Software Requirements Specification (SRS) provides a structured and comprehensive overview of the Movie Theater System (MTS). It is intended to define the purpose, scope, and requirements of the system to ensure clarity for developers, stakeholders, and end users throughout the software development lifecycle.

The Movie Theater System is being developed to streamline ticketing, seat reservation, user management, and showtime administration for both moviegoers and theater staff. It emphasizes a responsive, user-friendly experience for customers while enabling efficient backend operations for administrators.

This document outlines both functional and non-functional requirements, as well as the system's integration with hardware, external services, and supporting infrastructure. It serves as a foundational artifact that guides system design, development, testing, and future enhancements.

1.1 Purpose

The purpose of this document is to detail the functional and technical requirements of the Movie Theater System. The software will support key features such as:

- User registration, login, and profile management
- Browsing current and upcoming movies
- Showtime and seat selection
- Online ticket booking and payment
- Notifications and cancellations
- Administrative control over listings and operations

It provides a reference for project stakeholders to understand the system's capabilities, expectations, and compliance with industry standards.

1.2 Scope

The Movie Theater System is a web-based platform intended for:

- Moviegoers: who will interact with the system to search for movies, select seats, make purchases, and manage reservations.
- Theater Administrators: who will manage schedules, seating arrangements, ticket availability, and pricing.
- Support Staff: who will resolve technical and transaction issues and assist with account management.

The system includes modules for user accounts, movie/showtime listings, ticket booking and payment integration, notification services, and an admin dashboard. It will be optimized for major browsers and responsive to mobile screen sizes, though it is not a mobile-native app.

1.3 Definitions, Acronyms, and Abbreviations

Term	Description
MTS	Movie Theater System
UI	User Interface
API	Application Programming Interface
PII	Personally Identifiable Information
GCS	Google Cloud Services
2FA	Two-Factor Authentication
SQL	Structured Query Language
STD	State Transition Diagram
DFD	Data Flow Diagram

1.4 References

The references are:

- Course Materials – CS 250: "Lecture 3-ReqGathering"
- IEEE Guide for Software Requirements Specifications
- Your 2025 Guide to Writing a Software Requirements Specification (Andrew Burak)
- https://allassignmentexperts.com/sample_assignments/programming/Software%20Requirements%20Specification%20-%20Gym%20App.pdf

1.5 Overview

This document details the Movie Theater System software's functional and non-functional requirements, providing design constrains, interface requirements, and use case scenarios. Section 2 is an overall description of the software including the product perspective and functions, and the user characteristics. In section 3, a full list of requirements is explained.

2. General Description

This section outlines the broader factors influencing the system and its requirements without specifying exact functional or technical details. It describes the system's context, high-level functionality, user types, operating conditions, constraints, and underlying assumptions.

2.1 Product Perspective

The Movie Theater System (MTS) is a standalone, web-based platform that provides both consumer-facing and administrative capabilities. Inspired by existing ticketing systems such as Fandango or AMC Theatres, it is designed to support real-time ticket booking, seat selection, and showtime browsing through an intuitive user interface.

While the MTS operates independently, it integrates with third-party services for payments, notifications, and analytics. It will also rely on cloud-based hosting for performance, scalability, and availability.

Key integration points include:

- Payment gateways: to securely process online transactions
- Cloud storage services: to manage media assets like movie posters or trailers
- Communication APIs: for sending confirmation emails and notifications
- Database backend: for storing user accounts, bookings, and movie data

The system is designed to support future expansion into mobile apps or kiosk-based ticketing modules.

2.2 Product Functions

The Movie Theater System will deliver the following core functionalities:

- User Account Management: Users can register, log in, and update profiles.
- Movie Listings: Users can browse available movies, view trailers and descriptions.
- Showtime Scheduling: Real-time display of available showtimes by location.
- Seat Selection: Interactive seat maps enable users to reserve specific seats.
- Ticket Booking: Users can purchase tickets using secure online payment methods.
- Admin Dashboard: Administrators can add or edit movies, manage showtimes, and monitor sales.
- Notifications: Automated alerts for bookings, reminders, and promotional messages.
- Cancellation and Refunds: Ticket cancellations are processed according to predefined rules.
- Analytics & Reporting: Admins can access sales metrics, occupancy rates, and user activity.

2.3 User Characteristics

The MTS is designed to serve three core user groups, each with distinct goals, access privileges, and usage patterns:

1. Consumers (Moviegoers)
   - Goals: Easily book tickets, choose seats, and manage reservations
   - Device Usage: Primarily mobile and desktop browsers
   - Technical Skill: Basic; expects intuitive navigation and clear visuals
   - Key Activities:
     - Create/manage account
     - Browse showtimes and movie information
     - Purchase and cancel tickets
     - Receive and manage notifications

2. Administrators
   - Goals: Manage operational logistics for the theater
   - Device Usage: Desktop/laptop with secure dashboard access
   - Technical Skill: Intermediate to advanced; comfortable with forms, reports, and analytics tools
   - Key Activities:

  - Add or modify movie listings and showtimes
  - Adjust pricing and seat layouts
  - Monitor system status and user activity
  - Handle exceptions, e.g., outages, bulk cancellations

3. Support Staff
- Goals: Provide real-time assistance to users and manage transactional errors
- Device Usage: Internal systems (desktop preferred)
- Technical Skill: Intermediate; troubleshooting and administrative support
- Key Activities:
  - Assist users with login, payment, or booking issues
  - Process refunds or rebookings
  - Communicate with admins and technical teams

2.4 General Constraints

- Platform: The system must support major web browsers (Chrome, Firefox, Safari, Edge)
- Mobile: Mobile-friendly design, but not a native app
- Security: Must implement role-based access control and encryption for PII
- Performance: System should support real-time updates and handle concurrent users during peak demand
- Regulations: Must comply with data privacy and e-commerce regulations (e.g., PCI-DSS)
- Scalability: Backend should scale to accommodate growth in user volume and number of theaters
- Technology Stack: Development will utilize modern languages and frameworks such as JavaScript (Node.js), SQL, and Firebase

2.5 Assumptions and Dependencies

- Users will access the system using supported browsers and devices
- A stable internet connection is required for booking and payment
- Payment processing relies on third-party APIs (e.g., Stripe, PayPal)
- Theater admins are trained to use the backend management dashboard
- The hosting infrastructure (e.g., Google Cloud) will remain available and reliable
- Local laws may affect refund policies or payment integrations

3. Specific Requirements

This section provides detailed functional and non-functional requirements of the Movie Theater System (MTS). It includes descriptions of the system's interfaces, core functions, use cases, and system architecture to ensure consistent development, testing, and maintenance throughout the project lifecycle.

3.1 External Interface Requirements

3.1.1 User Interfaces

The MTS will feature an intuitive, responsive, and browser-compatible interface that serves three user types: consumers, administrators, and support staff.

- Homepage: Displays featured movies, current releases, and a search bar
- Movie Listings: Filter and browse by genre, title, rating, or release date
- Showtime and Seat Selection: Interactive maps showing real-time seat availability
- Checkout: Secure ticket purchasing flow with payment options
- User Account Dashboard: View booking history, manage payment methods, cancel or rebook
- Admin Panel: Accessible by authorized staff to manage showtimes, movies, and system data

3.1.2 Hardware Interfaces

- Client Devices: System is optimized for desktops, laptops, and mobile browsers
- Theater Kiosks (future): Potential hardware integration for in-person ticket purchases
- Cloud Hosting: MTS will run on a scalable cloud infrastructure (e.g., Google Cloud or AWS)

3.1.3 Software Interfaces

- Payment Gateways: Integration with APIs such as Stripe and PayPal
- Email/SMS Services: Notification integration via SendGrid or Twilio
- Movie Database APIs: Optional integration for fetching metadata, trailers, etc.
- Authentication APIs: Firebase Authentication or OAuth2 support for login and security
- Database Backend: SQL-based system (PostgreSQL or MySQL)

3.1.4 Communications Interfaces

- All data transferred between client and server will use HTTPS protocols
- RESTful APIs will be used for frontend-backend communication
- Notifications (email/SMS) will be sent through third-party communication APIs
- JSON will be the standard format for API data exchange

3.2 Functional Requirements

3.2.1 User Registration and Authentication

3.2.1.1 Introduction

The system must allow users to register and securely log into their accounts. It will handle credential management, account creation, and authentication with added security features like CAPTCHA and optional Two-Factor Authentication (2FA).

3.2.1.2 Inputs

- Email address
- Password

- Full name
- CAPTCHA code
- Optional: Phone number (for 2FA)

3.2.1.3 Processing

- Verify that all required fields are filled in and correctly formatted
- Check if the email already exists in the database
- Hash and store passwords securely
- Generate and send confirmation email
- On login, validate credentials against stored values and allow access

3.2.1.4 Outputs

- Confirmation email for successful registration
- Success message and redirection upon login
- JWT or session token created for authenticated access

3.2.1.5 Error Handling

- Display error if email already exists or format is invalid
- Return “Incorrect password” for failed login attempts
- Lock user out after 5 consecutive failed login attempts
- CAPTCHA must be validated before processing

3.2.2 Movie Browsing and Showtimes

3.2.2.1 Introduction

Users will be able to browse and search for movies, view listings by date or location, and access showtime information.

3.2.2.2 Inputs

- Movie title (search query)
- Genre, rating, date, or location filter
- Optional: User-selected favorites or previous history

3.2.2.3 Processing

- Filter the movie database based on query parameters
- Return matching movie records with showtimes
- Load associated media and metadata (poster, trailer)

3.2.2.4 Outputs

- List of movies and available showtimes
- Movie details page with poster, synopsis, cast, and trailer

3.2.2.5 Error Handling

- Show “No results found” if nothing matches the search
- Display loading errors if media fails to load
- Log invalid input or search term errors

3.2.3 Seat Selection and Ticket Booking

3.2.3.1 Introduction

Users can select seats visually from an interactive map and proceed with ticket booking for a selected movie and showtime.

3.2.3.2 Inputs

- Movie ID
- Showtime ID
- Selected seat(s)
- Ticket quantity and type (Adult, Child, Senior)

3.2.3.3 Processing

- Lock selected seats for 5 minutes during checkout
- Validate availability and pricing
- Calculate total cost
- Prepare data for payment

3.2.3.4 Outputs

- Confirmation of seat lock and transition to checkout
- Summary of booking (movie, time, seat, total price)

3.2.3.5 Error Handling

- Notify if seat is already booked
- Timeout seat lock if payment is not completed
- Highlight and refresh unavailable seats in real-time

3.2.4 Payment Processing

3.2.4.1 Introduction

The user will be able to pay for tickets using credit/debit cards, PayPal, or stored payment methods. The payment will be processed through a secure third-party gateway.

3.2.4.2 Inputs

- Credit card or PayPal account details
- Billing information
- Promo codes (optional)

3.2.4.3 Processing

- Validate payment form inputs
- Send encrypted request to payment gateway
- Process transaction and update database

3.2.4.4 Outputs

- Success message with ticket confirmation
- Receipt with QR code
- Email or SMS confirmation

3.2.4.5 Error Handling

- Display gateway error messages (e.g., declined card)
- Reopen seat selection if payment fails
- Retry option for failed transactions

3.2.5 Ticket Cancellation and Refunds

3.2.5.1 Introduction

Users can cancel booked tickets and request a refund if within the allowed cancellation window.

3.2.5.2 Inputs

- Booking ID
- User authentication
- Reason for cancellation (optional)

3.2.5.3 Processing

- Check eligibility based on movie start time
- Process refund through the original payment method
- Update seat availability in the database

3.2.5.4 Outputs

- Confirmation message and refund receipt
- Updated booking status in user dashboard
- Email confirmation of cancellation

3.2.5.5 Error Handling

- Deny cancellations past the deadline
- Handle refund API failures with retry option
- Alert support staff if refund processing fails

3.2.6 Notifications and Reminders

3.2.6.1 Introduction

The system shall notify users about booking confirmations, reminders before showtime, ticket cancellations, and promotional offers (if opted in). Notifications will be sent via email and optionally SMS.

3.2.6.2 Inputs

- User contact information (email/SMS)
- Booking confirmation or cancellation event
- Showtime data
- Notification preferences

3.2.6.3 Processing

- Generate appropriate notification content
- Format message according to the selected channel (email/SMS)
- Queue and dispatch via third-party services (e.g., Twilio, SendGrid)

3.2.6.4 Outputs

- Confirmation emails and SMS messages
- Reminders 1 hour before showtime
- Promotional messages for opted-in users

3.2.6.5 Error Handling

- Retry sending on first failure
- Fallback to alternative channel if available (e.g., email if SMS fails)
- Log failures and notify admin panel if delivery repeatedly fails

3.2.7 Security and Privacy

3.2.7.1 Introduction

The system shall protect user data through encryption, secure authentication, and compliance with data protection regulations. Users must have control over their data sharing and visibility settings.

3.2.7.2 Inputs

- User credentials
- Personal identifiable information (PII)
- Privacy preferences

3.2.7.3 Processing

- Hash and store credentials securely
- Encrypt PII both in transit and at rest
- Enforce access control policies based on roles (User, Admin, Support)

3.2.7.4 Outputs

- Secure login sessions
- Obfuscated data views for unauthorized roles
- Notifications for account changes or login from unknown devices

3.2.7.5 Error Handling

- Lock account after repeated failed login attempts
- Expire sessions after inactivity
- Alert users and support on suspicious activities

3.2.8 Admin Management Dashboard

3.2.8.1 Introduction

Authorized administrators can manage showtimes, movie listings, prices, and view reports through a dedicated admin dashboard.

3.2.8.2 Inputs

- Admin credentials
- Movie/showtime information
- Report filters (date range, ticket type, movie)

3.2.8.3 Processing

- Authenticate admin access
- Retrieve and update movie/showtime data
- Generate sales or booking reports in real time

3.2.8.4 Outputs

- Admin dashboard view
- Updated movie and schedule listings
- Downloadable reports (CSV or PDF)

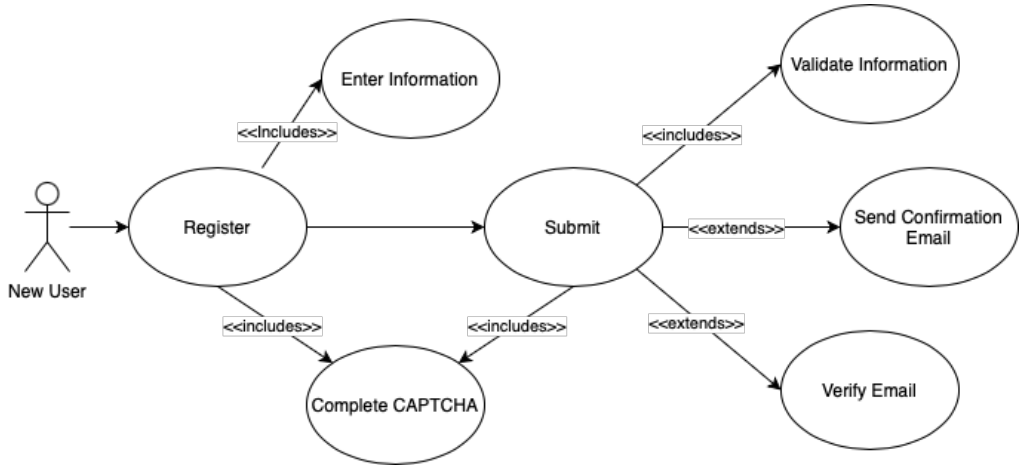
3.2.8.5 Error Handling

- Deny access if permissions are insufficient
- Roll back changes on failed data update
- Log unauthorized access attempts and notify system monitor

3.3 Use Cases

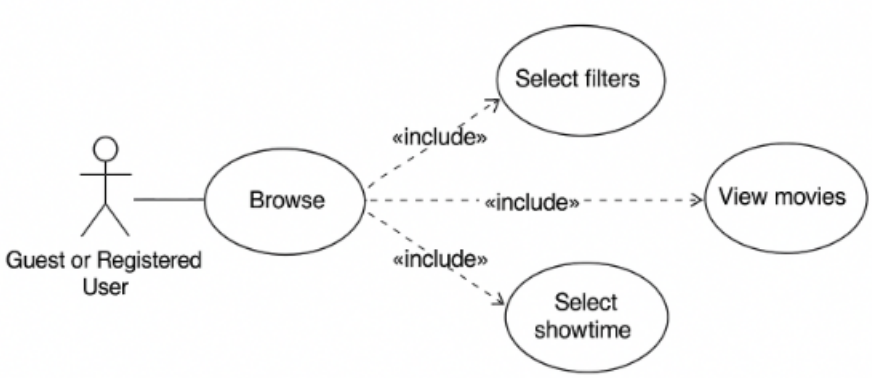
3.3.1 User Registration

- Actor: New User
- Description: A new user registers an account on the system to book tickets and manage reservations.
- Preconditions: User is not currently registered.
- Main Flow:
    1. User navigates to the registration page.
    2. Enters full name, email address, and password.
    3. Completes CAPTCHA verification.
    4. System validates and stores data securely.
    5. Confirmation email is sent to user.
    6. User verifies the email and is redirected to login.
- Alternate Flow:
    - If email is already in use, system prompts user to log in or reset password.
- Postconditions:
    - A verified user account is created and stored in the system.



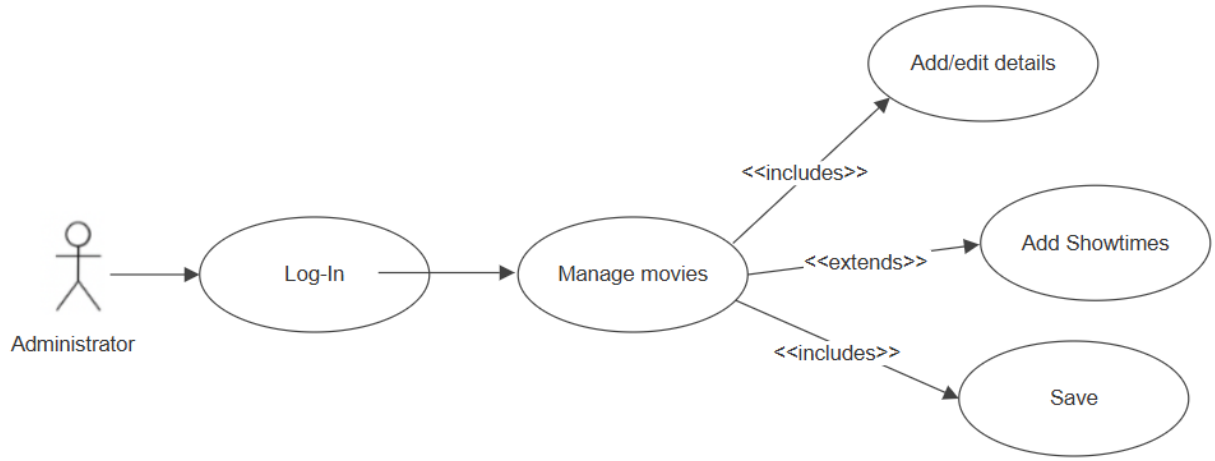
3.3.2 Browse Movies and Showtimes

- Actor: Guest or Registered User
- Description: A user browses the catalog of movies and their respective showtimes.
- Preconditions: None.
- Main Flow:
    1. User accesses the "Movies" or "Showtimes" section.
    2. Selects filters (genre, language, date, location).
    3. System displays matching movie entries.
    4. User selects a movie to view available showtimes.
- Postconditions:
    - User may proceed to booking or exit browsing.



3.3.3 Admin manage Movies and Showtimes

- Actor: Administrator
- Description: Admin updates movie listings and schedules showtimes.
- Preconditions: Admin is authenticated and authorized
- Main Flow:
    1. Admin logs into the dashboard.
    2. Navigates to “Manage Movies”.
    3. Adds or edits movie details (title, synopsis, rating).
    4. Adds showtimes for selected theaters and dates
    5. Changes are saved and synced with the booking system.
- Postconditions:
    - Movies and showtimes become visible to users.



3.4 Classes / Objects

More information about specific classes, methods, and attributes can be found in Section 5.3.

3.5 Non-Functional Requirements

The following section describes the non-functional requirements that the Movie Theater System must fulfill. These relate to the overall system behavior and constraints, such as performance,

security, scalability, and usability, which are critical for providing a robust and satisfactory user experience.

3.5.1 Performance

- The system shall respond to 95% of all user actions (e.g., login, seat selection) within 2 seconds.
- Ticket booking and payment confirmation processes shall be completed within 5 seconds under normal load.
- The system must support at least 200 concurrent users without performance degradation.
- Database queries (e.g., for movie listings, seat availability) should return results in less than 1 second.
- During high-traffic events (e.g., blockbuster premieres), the system shall maintain stable performance up to 500 concurrent users via autoscaling.

3.5.2 Reliability

- The system shall maintain 99.9% uptime over any 30-day period, excluding scheduled maintenance.
- All transaction operations (e.g., bookings, refunds) shall be atomic, ensuring data integrity during partial failures.
- The system shall have automated monitoring and recovery mechanisms in place to handle unexpected crashes or downtime.
- In the event of a critical failure, recovery should occur within 1 minute, and affected services must resume with no loss of data.

3.5.3 Availability

- The system shall be available 24/7, including for users in different time zones.
- All core services (e.g., booking, browsing, login) must be deployed with failover redundancy.
- System maintenance shall be scheduled during low-traffic hours and announced to users in advance.
- Availability of third-party APIs (e.g., payment gateways, email/SMS services) shall be monitored, and fallback procedures shall be in place (e.g., queuing payment requests)

3.5.4 Security

- All personal identifiable information (PII) and payment details shall be encrypted using AES-256 at rest and TLS 1.2+in transit.
- Authentication shall include strong password policies and support for multi-factor authentication (2FA).
- The system shall use CAPTCHA to prevent bot attacks during registration and login.
- Role-based access control (RBAC) must be enforced to restrict unauthorized access to administrative features.
- All authentication tokens and session cookies shall be securely managed and expire after 30 minutes of inactivity.
- A security audit log shall track user logins, failed access attempts, and account modifications.
- The system shall be compliant with data protection laws like GDPR and CCPA.

3.5.5 Maintainability

- The codebase shall follow consistent, documented coding standards and be modular for easier updates.
- The system architecture shall be loosely coupled, allowing independent updates to components like booking or notifications.
- All configuration values (e.g., API keys, database credentials) must be stored in environment-specific secure files.
- Comprehensive developer documentation (API docs, database schemas, component diagrams) shall be provided and updated regularly.
- The system shall include automated testing (unit, integration, end-to-end) to support continuous integration and deployment (CI/CD).
- Maintainability score (as measured by tools like SonarQube) should not fall below a "B" rating.

3.5.6 Portability

- The system must function correctly across all major web browsers: Chrome, Firefox, Safari, and Edge.
- The user interface must be responsive, adapting to different screen sizes, including desktops, tablets, and smartphones.
- All core services shall be containerized using Docker to ensure platform independence.
- The system shall be deployable on multiple cloud environments (e.g., Google Cloud, AWS, Azure) with minimal configuration changes.
- The frontend and backend must be separately deployable, facilitating portability for microservices or hybrid environments.

3.6 Inverse Requirements

This section defines behaviors that the system must explicitly avoid, to ensure user safety, data integrity, and proper access control.

3.6.1 The system shall not allow booking without user authentication.

Only logged-in users may proceed to book, pay, or manage tickets.

3.6.2 The system shall not allow double booking of seats.

A seat cannot be selected by multiple users at the same time. Temporary locks prevent conflicts during checkout.

3.6.3 The system shall not store passwords or personal data in plain text.

All sensitive information must be encrypted and securely stored.

3.6.4 The system shall not allow cancellations after showtime.

Refunds are only permitted before the movie begins, as per the refund policy.

3.6.5 The system shall not expose admin features to standard users.

Admin panels and functions are only accessible to users with proper roles and permissions.

3.6.6 The system shall not retain session data after logout.

All tokens and session info are cleared once the user logs out.

4. System Description

4.1 Overview

The Movie Theater System is a multi-user web application with distinct modules for customers, administrators, and support staff. It follows a three-tier architecture, consisting of:

- Frontend (Presentation Layer): Web interface for users and admins
- Backend (Business Logic Layer): Handles core functionalities such as booking, seat management, user authentication, etc.
- Database (Data Layer): Stores persistent data such as user accounts, movie info, bookings, and transactions

The system interacts with third-party APIs for payment processing, email/SMS notifications, and optional external movie metadata.

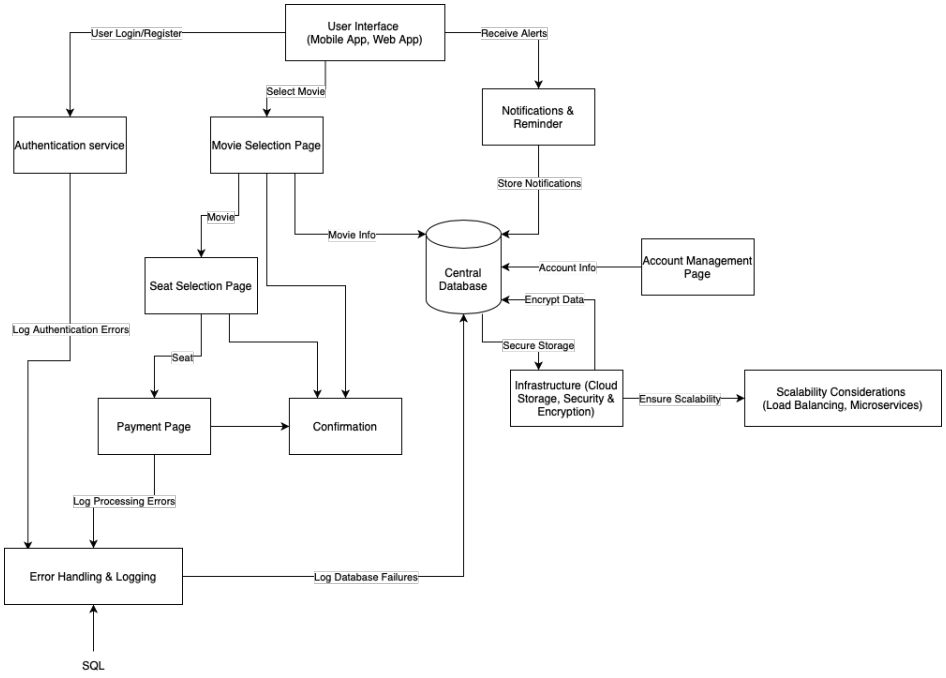
4.2 Data Processing Flow

The main data flow in the system operates as follows:

1. User Input: Users interact via the frontend to search for movies, select seats, and initiate bookings.
2. Request Handling: The backend processes these actions, applying business rules and verifying seat availability or payment info.
3. Database Operations: The backend queries or updates the SQL database for user, booking, or movie data.
4. Third-Party Services: Payment details are securely forwarded to external payment gateways. Notifications are sent using messaging APIs.
5. System Response: The frontend receives confirmation messages, updates the user interface, and provides feedback (e.g., ticket confirmation or error messages)

5. Software Architecture

5.2 Architectural Diagram



This diagram shows how the diferent parts of the Movie Theater System (MTS) work together to let users browse movies, choose seats, make payments, and manage their accounts through a web or mobile app. Everything starts at the User Interface, which is where users interact with the system. When a user logs in or registers, their information is handled by the Authentication Service, which checks if the login is valid and sends any errors to the Error Handling & Logging system.

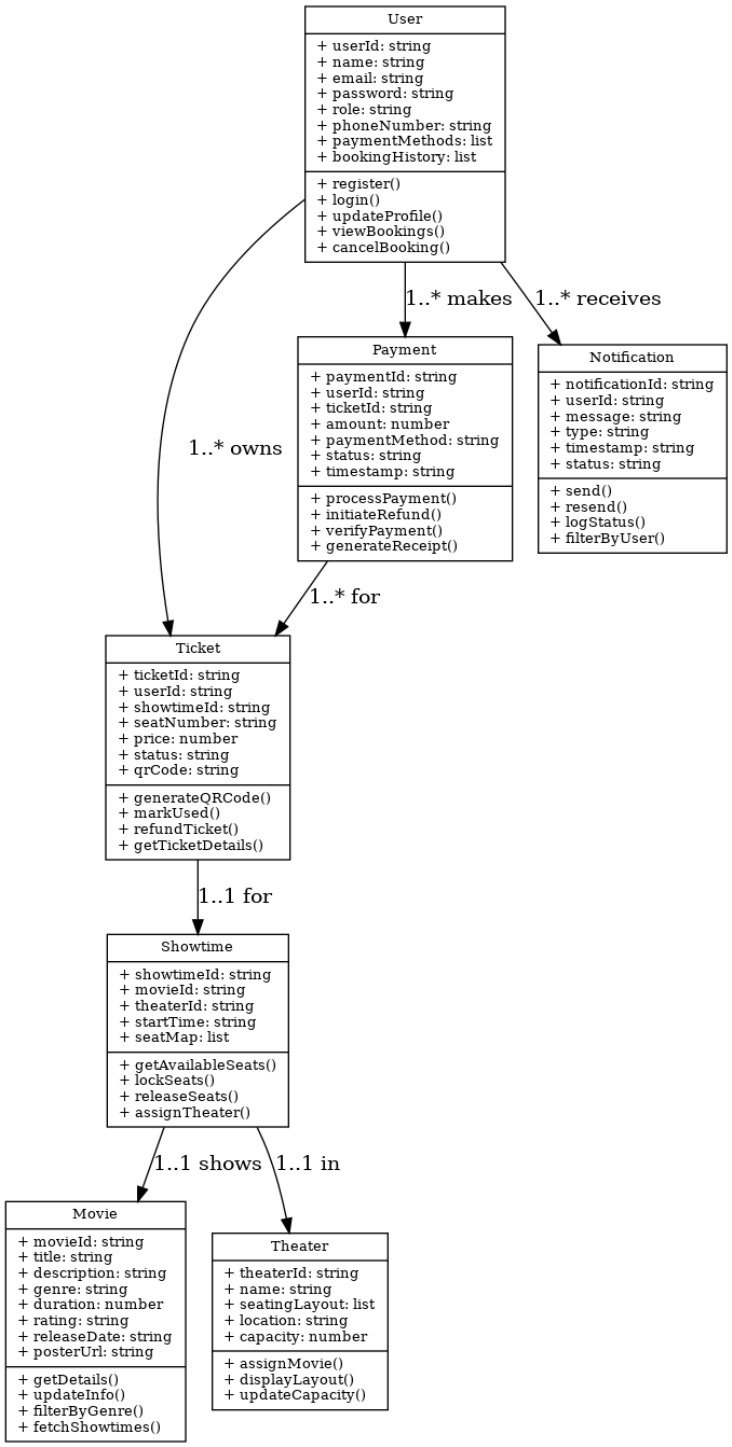
Once logged in, users can go to the Movie Selection Page to see a list of movies. The system gets this information from the Central Database, which stores all the data about movies, seats, showtimes, user accounts, and payments. After choosing a movie, users are taken to the Seat Selection Page, where they can pick their preferred seats. The seat selection is saved in the database and also passed to the Payment Page when the user is ready to pay.

The Payment Page collects payment details, and once the transaction is successful, the system shows a Confirmation Page so the user knows everything went through. During all of these steps, important data like movie info, seat info, and payment records are saved and encrypted in the Central Database to keep it secure. The system also includes a Notifications & Reminder feature that can send alerts to users (like payment confirmations or reminders for upcoming movies), and an Account Management Page where users can view or update their personal details, payment methods, or booking history.

In the background, the system is supported by Infrastructure that handles secure storage and encryption, and Scalability Considerations to make sure it can handle lots of users at once, especially during busy times. All errors—whether related to login, payment, or database—are tracked by the Error Handling & Logging component to help developers fix problems quickly.

Overall, this architecture helps make the system safe, smooth, and easy for users to navigate from start to finish.

5.2 UML Class Diagram



5.3 Classes, Attributes and Functions

5.3.1 User Class

Person using the system, such as a customer, admin, or support staff.

Attributes:

- userId *(string)* – A unique ID to identify the user in the system
- name *(string)* – The user's full name
- email *(string)* – The user's email address (used for login and notifications)
- password *(string)* – The hashed password for authentication
- role *(string)* – User type: "customer", "admin", or "support"
- phoneNumber *(string, optional)* – The user's contact number
- paymentMethods *(list)* – A list of stored payment methods
- bookingHistory *(list)* – A list of past or current ticket bookings

Functions:

- register() – Create a new user account
- login() – Authenticate credentials
- updateProfile() – Modify personal or contact info
- viewBookings() – Display user's booking history
- cancelBooking() – Cancel a ticket if allowed

5.3.2 Movie Class

Represents a movie available for booking.

Attributes:

- movieId *(string)* – Unique identifier for each movie
- title *(string)* – Name of the movie
- description *(string)* – Brief plot summary
- genre *(string)* – Genre or category (e.g., "Action", "Comedy")
- duration *(number)* – Duration in minutes
- rating *(string)* – Age rating (e.g., "PG-13")
- releaseDate *(string/date)* – The date the movie premieres
- posterUrl *(string)* – Link to movie poster or trailer

Functions:

- getDetails() – Fetch movie info for display
- updateInfo() – Admin function to update movie data
- filterByGenre() – Search/filter movies by genre
- fetchShowtimes() – Get showtimes linked to this movie

5.3.3 Showtime Class

Represents a scheduled movie showing in a specific theater at a certain time.

Attributes:

- showtimeId *(string)* – Unique identifier for the showtime
- movieId *(string)* – The movie being shown
- theaterId *(string)* – The theater room where it's playing
- startTime *(string/date-time)* – Start time and date

- seatMap *(list of booleans or objects)* – Seat availability (true = taken, false = available)

Functions:

- getAvailableSeats() – Returns list of unbooked seats
- lockSeats(seatIds) – Temporarily hold selected seats
- releaseSeats(seatIds) – Release unbooked seats
- assignTheater() – Assign a theater room to the showtime

5.3.4 Ticket Class

Represents a purchased movie ticket

Attributes:

- ticketId *(string)* – Unique code per ticket
- userId *(string)* – User who owns the ticket
- showtimeId *(string)* – The showtime the ticket is for
- seatNumber *(string)* – The seat assigned (e.g., "B12")
- price *(number)* – Total price of the ticket
- status *(string)* – Status: "active", "used", "cancelled"
- qrCode *(string)* – Encoded data for scanning on entry

Functions:

- generateQRCode() – Create a QR code for digital entry
- markUsed() – Mark ticket as used after scanning
- refundTicket() – Cancel and request refund
- getTicketDetails() – Show ticket info to user or staff

5.3.5 Payment Class

Represents a completed or pending transaction.

Attributes:

- paymentId *(string)* – Unique ID for the transaction
- userId *(string)* – Who made the payment
- ticketId *(string or list)* – Related ticket(s)
- amount *(number)* – Total charge in local currency
- paymentMethod *(string)* – Type: "card", "PayPal"
- status *(string)* – "success", "pending", or "failed"
- timestamp *(string/date-time)* – Time of transaction

Functions:

- processPayment() – Charge the user using gateway
- initiateRefund() – Reverse the payment
- verifyPayment() – Confirm transaction status
- generateReceipt() – Output a receipt for user or admin

5.3.6 Theater Class

Represents a physical theater room within the venue.

Attributes:

- theaterId *(string)* – ID or number of the theater room

- name *(string)* – Name or label of the room
- seatingLayout *(list or 2D array)* – Arrangement of rows/seats
- location *(string)* – Branch or city (for multi-location theaters)
- capacity *(number)* – Total number of seats

Functions:

- assignMovie(movieId) – Schedule a movie for this theater
- displayLayout() – Show seats and their arrangement
- updateCapacity() – Modify seating limits or layout

5.3.7 Notification Class

Represents messages sent to users, like confirmations or alerts.

Attributes:

- notificationId *(string)* – Unique ID per message
- userId *(string)* – Recipient of the notification
- message *(string)* – Content of the message
- type *(string)* – "email", "SMS", "system"
- timestamp *(string/date-time)* – When it was sent
- status *(string)* – "sent", "pending", "failed"

Functions:

- send() – Trigger notification to user
- resend() – Retry sending a failed message
- logStatus() – Track if it was delivered or not
- filterByUser(userId) – Get user-specific message history

6. Development Plan and Timeline

The development of the Movie Theater System (MTS) will follow a structured, phased approach based on standard software engineering practices. This ensures logical progression from planning to deployment, allowing continuous feedback, manageable workloads, and timely delivery.

6.1 Partitioning of Tasks

6.1.1 Phase 1: Requirement Gathering & Analysis

Duration: 1 week

Objectives:

- Identify and document all functional and non-functional requirements.
- Define core user personas: Consumers, Admins, Support Staff.
- Establish clear system goals, constraints, and boundaries.
- Create early drafts of use cases and system flow.

Key Tasks:

- Stakeholder interviews and survey (if applicable)
- Draft use case scenarios and feature list
- Research compliance needs (e.g., privacy, data handling)

Deliverables:

- Software Requirements Specification (SRS)

- Approved feature set and system scope
- Initial Use Case Diagram

6.1.2 Phase 2: System Design

Duration: 2 weeks
Objectives:

- Translate requirements into detailed system architecture.
- Design UI/UX wireframes for key flows (e.g., booking, login).
- Model database schema, object classes, and relationships.
- Select technology stack for frontend, backend, and infrastructure.

Key Tasks:

- Create architecture diagram (3-tier or microservices model)
- Design user journeys and screen mockups
- Define APIs and endpoints for modules
- Choose frameworks, libraries, and deployment tools

Deliverables:

- Architecture Design Document
- Database Schema (ER Diagram)
- UML Class Diagrams and Sequence Diagrams
- UI Wireframes and Component Design

6.1.3 Phase 3: Implementation (Development)

Duration: 4 weeks
Objectives:

- Build a fully functional application with both consumer and admin views.
- Implement key modules: registration, movie browsing, seat selection, payments.
- Integrate secure authentication and third-party APIs.

Frontend Tasks:

- Develop responsive UI using React, Vue, or HTML/CSS/JS
- Implement key screens: Home, Movie Detail, Checkout, Profile

Backend Tasks:

- Create RESTful APIs using Node.js, Python (Django), or Java (Spring Boot)
- Implement logic for seat locking, booking, and refund workflows
- Set up secure user authentication (JWT, OAuth, Firebase)

Deliverables:

- Fully functional web application
- Working backend APIs with documentation
- Integrated payment gateway (e.g., Stripe or PayPal)
- Email/SMS notification system
- Role-based access for users and admins

6.1.4 Phase 4: Testing and QA

Duration: 2 weeks
Objectives:

- Validate all user stories and requirements through test cases.
- Ensure app reliability under real-world conditions.
- Detect and fix bugs, security holes, and edge case failures.

Testing Types:

- Unit Testing (function-level logic)
- Integration Testing (API and database flow)
- End-to-End Testing (user scenarios from login to purchase)
- Security Testing (login abuse, data exposure)
- Compatibility Testing (browser and screen sizes)

Deliverables:

- Test Plan and Test Cases Document
- Bug Report Log
- Verified Fixes and Regression Test Results
- Final Acceptance Test Results

6.1.5 Phase 5: Deployment & Maintenance

Duration: 1 week

Objectives:

- Launch system in a cloud environment (e.g., Google Cloud, AWS).
- Conduct final user acceptance testing (UAT).
- Provide basic post-launch monitoring, documentation, and support plan.

Deployment Tasks:

- Containerize application using Docker
- Set up CI/CD pipelines for smooth deployment
- Apply domain and SSL for secure web access
- Monitor logs and health of application

Deliverables:

- Live web deployment
- Final system documentation (deployment guide, user manual)
- Maintenance Checklist and Future Features Roadmap

6.2 Project Timeline Summary

Phase	Duration	Timeframe	Milestone
Requirement Gathering	1 week	Week 1	Finalized SRS and project goals
System Design	2 weeks	Week 2-3	Approved architecture and UI designs
Implementation	4 weeks	Week 4-7	MVP with working booking system
Testing	2 weeks	Week 8-9	Bug-free version ready for release
Deployment & Maintenance	1 week	Week 10	Live deployment + support readiness

6.3 Team Member Responsibilities

6.3.1 Project Manager (PM)

- Create and maintain the project timeline and task board
- Coordinate team meetings and ensure on-time deliverables
- Manage communication with the instructor or stakeholders

6.3.2 Fronted Developer (FD)

- Design and implement the user-facing interface (home, booking, admin views)
- Ensure responsive design and cross-browser compatibility
- Integrate frontend components with backend APIs

6.3.3 Backend Developer (BD)

- Develop APIs for user authentication, booking, and payment processing
- Implement business logic such as seat locking and refunds
- Ensure secure data handling and input validation

6.3.4 Database Designer/Administrator (DBA)

- Design the relational database schema with proper normalization
- Create and manage tables for users, movies, showtimes, and bookings
- Optimize queries and enforce data integrity with constraints

6.3.5 QA Engineer

- Write test cases for major user flows (register, book, cancel)
- Perform functional and regression testing on frontend and backend
- Report and track bugs throughout development and testing phases

6.3.6 DevOps Engineer

- Set up and maintain the deployment environment using Docker or cloud services
- Create CI/CD pipelines for automated testing and deployment
- Monitor logs and system health post-deployment

6.3.7 UI/UX Designer

- Design wireframes, mockups, and screen flows for the application
- Ensure accessibility, intuitive navigation, and user-centered design
- Collaborate with frontend dev to maintain design consistency

6.3.8 Security Engineer

- Implement encryption, secure authentication, and API rate-limiting
- Conduct vulnerability assessments and fix security flaws
- Ensure compliance with privacy standards (e.g., GDPR, CCPA)

6.3.9 Support Analyst

- Draft help content or FAQs for end users
- Simulate real-world usage scenarios for UAT testing
- Gather feedback to improve system usability

7. Test Plan

7.1 Overview

This section lays out test plans for verification and validation. It includes detailed test cases covering various system components, ensuring the software meets design and performance expectations.

7.2 Testing Approach

- Functional testing: This will ensure that all features and functions of the MTS act as expected. It includes testing user registration and login, browsing movies, seat selection, ticket booking, payment, and administrative tasks such as changing movie listings and scheduling showtimes. Each function must react correctly based on user input and business rules.

- **Usability testing:** This will help ensure how user-friendly and simple the system is to use for both administrators and users. It is interested in ease of getting around the site, ease of selecting movies and times, ease of completing bookings, and how simple administration tools are to use to control content. The goal is to reduce friction and increase the user experience for all types of users.

- **Performance testing:** This will test how the system behaves with different loads, especially during high usage times (e.g., when a blockbuster movie is opening). It will check the number of users the system can support accessing it simultaneously, the speed of pages loading, and how the bookings are handled without delay. The testing of real-time seat availability updates and handling of transactions will also be done.

- **Security testing:** This will define and establish vulnerabilities in storage and transmission of individual information. This includes authentication of security of user data (email addresses, passwords, names), payment information, and administrator data. The system must meet security standards to prevent data leakage and secure communication among users and servers.

- **Compatibility testing:** This will ensure that the MTS runs perfectly on a variety of devices (desktop, tablet, smartphone) and operating systems (Windows, macOS, Android, iOS). The system should display correctly in all major browsers (e.g., Chrome, Firefox, Safari) and function fully irrespective of the platform being used by the user.

7.3 Test Case Categories

7.3.1 User Registration and Authentication Successful user registration, login, and email verification

- Invalid login attempts (incorrect password, email not registered)
- Password reset functionality via secure link
- Multi-factor authentication (MFA) for administrative accounts

7.3.2 Movie Browsing and Showtime Selection Searching and filtering of movies by genre, language, date, and location

- Display of detailed movie information (title, synopsis, rating, duration)
- Selection and preview of available showtimes by date and theater
- Dynamic update of listings with admin changes

7.3.3 Seat Booking and Payment Selecting available seats using interactive seat maps

- Booking tickets successfully and generating a booking confirmation
- Handling concurrent seat selection and preventing double bookings
- Making payments securely using third-party gateways (e.g., Stripe, PayPal)
- Handling payment declines and transaction timeouts gracefully

7.3.4 Notifications and Confirmations Sending booking confirmation via email or SMS

- Ensuring cancellation or change notifications are triggered correctly
- Setting up reminder configurations (e.g., 2 hours before the movie start time)
- Ensuring compatibility with third-party notification APIs (e.g., Twilio)

7.3.5 User Profile and Reservation Management Modifying user profile information (e.g., name, email, preferences)

- Displaying future and past bookings in the reservation dashboard
- Canceling or modifying future bookings
- Downloading digital tickets and receipts from account history

7.3.6 Admin Functionality

- Logging into the admin dashboard with proper access control
- Adding and modifying movie metadata (title, rating, description, poster)
- Booking showtimes for target theaters and dates
- Modifying prices and verifying sync with booking interface
- Managing user accounts, permissions, and resolving support tickets

7.3.7 Performance and Compatibility

- Performance testing under high-traffic situations (e.g., new release evening)
- Response time testing for API calls and database queries

- Testing for consistent functionality across devices (desktop, mobile, tablet)
- Cross-browser compatibility testing (Chrome, Safari, Firefox, Edge)

7.3.8 Security and Data Protection

- Testing for common attack protection (SQL injection, XSS)
- Verification of HTTPS encryption for all sensitive transactions
- Ensuring payment and user data are stored securely and anonymized
- Role-based access control for admin vs. user permissions

7.3.9 Error Handling and Edge Cases

- Handling failed payments, lost internet connection, and session timeouts
- Displaying friendly error messages when there is a service outage
- Graceful recovery from failed booking form submission or server errors
- Logging and alerting for unexpected system exceptions

7.4 Test Case Samples

7.4.1 Email Duplication Check During Registration

TestCaseId	Component	Priority	Description/Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed By
Email Duplication Check During Registration	User_Registration_Module	P0	Unit test for validating unique email during registration.	User data with existing email in the database.	1. Enter an already registered email. 2. Submit the registration form.	System prompts that the email is already in use.	The system correctly identifies the existing email and prompts the user to log in or reset their password.	Pass	Silvia Perez

This test ensures that the registration process prevents duplicate accounts by checking if the email already exists in the system. It helps maintain database integrity and avoids user confusion caused by multiple registrations.

7.4.2 Secure Password Validation Logic

Secure Password Validation Logic	Authentication_Module	P0	Unit test for password hashing and validation.	Valid and invalid password inputs for a known user.	1. Enter password input. 2. Call authentication function.	Correct password is accepted, incorrect is rejected.	Valid passwords grant access; invalid passwords are rejected. Passwords are not stored in plain text.	Pass	Silvia Perez

This test focuses on the backend logic for password handling. It confirms that the system uses secure password hashing and that valid credentials are properly authenticated while invalid ones are rejected, contributing to overall system security.

7.4.3 Filter Movies by Genre and Show Date

Filter Movies by Genre and Show Date	Movie_Browsing_Module	P1	Functionality test for filtering movies by genre and date.	Movie catalog loaded with various genres and dates.	1. Select 'Comedy' genre and a date filter. 2. Click apply.	Only 'Comedy' movies scheduled for that date are shown.	Only movies matching the selected genre and date are shown in the listings.	Pass	Silvia Perez

This test verifies the movie browsing filters. It ensures users can narrow down movie listings based on their preferences, providing a smoother, more personalized browsing experience.

7.4.4 Prevent Concurrent Seat Bookings

Prevent Concurrent Seat Bookings	Seat_Selection_Module	P0	Functionality test to ensure double booking is prevented.	User is on the seat selection page for a popular showtime.	1. Attempt to book the same seat from two sessions.	Only one booking is confirmed; second receives an error.	The first session successfully books the seat. The second session receives an “Already Booked” error.	Pass	Silvia Perez

This case checks the seat reservation logic under high-demand conditions. It prevents two users from booking the same seat simultaneously, which is essential for maintaining trust and accuracy in the ticketing system.

7.4.5 Complete Booking and Payment Flow

Complete Booking and Payment Flow	Payment_Module	P0	System test for completing a full booking flow with payment.	User is logged in and has selected seats.	1. Proceed to checkout. 2. Enter valid card info. 3. Confirm payment.	Payment is successful and confirmation email is sent.	Payment is processed successfully, and the user receives a confirmation on screen and via email.	Pass	Silvia Perez

This end-to-end system test simulates a full user journey from movie selection to payment. It validates the critical functionality of the platform, ensuring users can book tickets seamlessly and securely.

7.4.6 Send Notifications After Ticket Purchase

Send Notifications After Ticket Purchase	Notification_Service	P2	System test to verify SMS and email notifications after booking.	Booking is completed with contact information provided.	1. Complete a booking. 2. Check for email and SMS confirmation.	Booking confirmation is received via both channels.	Both SMS and email confirmations are received within 10 seconds after booking.	Pass	Ines Ling

This test confirms that users receive confirmation messages via email and SMS after completing a booking. It enhances the user experience by providing assurance and important event details.

7.4.7 Restrict Admin Dashboard Access to Staff Only

Restrict Admin Dashboard Access to Staff Only	Admin_Dashboard	P1	Unit test to validate admin access rights.	Different users with varied access levels.	1. Login as regular user. 2. Attempt to access admin tools.	Access denied for non-admin users.	Non-admin users receive an “Access Denied” message when attemptin	Pass	Ines Ling

							g to use admin tools.		

This test validates role-based access control, ensuring that only authorized admins can access sensitive dashboard features. It helps safeguard administrative tools from unauthorized users.

7.4.8 Reflect Admin Movie Updates to Public Listings

Reflect Admin Movie Updates to Public Listings	Movie_Management_Module	P1	Functionality test to verify movie updates reflect on user view.	Admin logged into dashboard.	1. Update movie title or poster. 2. Refresh user movie list.	Updated movie info appears in user view.	The updated title and poster are visible to users within 2 seconds of admin submission.	Pass	Ines Ling

This case checks the synchronization between admin content updates and the user-facing movie catalog. It ensures real-time visibility of changes, such as new titles or schedule updates.

7.4.9 Avoid Overlapping Movie Showtimes

Avoid Overlapping Movie Showtimes	Showtime_Scheduler	P2	System test for overlapping showtimes in the same hall.	Existing showtime entries for one theater hall.	1. Try to add a new showtime that overlaps. 2. Save changes.	System prevents overlapping schedule and shows error.	System blocks the overlapping showtime and displays an error explaining the conflict.	Pass	Ines Ling

This test prevents administrators from scheduling conflicting showtimes in the same hall, protecting the operational schedule and avoiding booking errors.

7.4.10 Responsive UI on Multiple Devices

Responsive UI on Multiple Devices	Responsive_UI	P2	System test for cross-device compatibility.	Access to browser testing tools or physical devices.	1. Open booking page on desktop, tablet, and phone. 2. Interact with UI elements.	UI adapts correctly on all devices; no layout issues.	All UI elements are properly scaled and functional across desktop, tablet, and mobile views.	Pass	Ines Ling

This test ensures that the user interface adapts properly across desktops, tablets, and smartphones, providing consistent functionality and visual clarity regardless of the device used.

8. Data Management Strategy

8.1 General Description

The data management strategy for the Movie Theater System (MTS) is designed to ensure secure, reliable, and efficient handling of transactional data, user credentials, and content-related assets. Given the system's critical role in managing real-time seat availability, user profiles, and payment processing, it relies on a combination of technologies optimized for performance, consistency, and scalability.

- Primary Database (Relational Database): A PostgreSQL database is used as the main data store to manage structured data such as:
    - User accounts (credentials, roles, contact info)
    - Movie listings (title, genre, duration, rating)
    - Showtimes (theater ID, movie ID, start time)
    - Ticket records (seat number, status, pricing)
    - Payment history and transaction details
    - Notifications and reminders

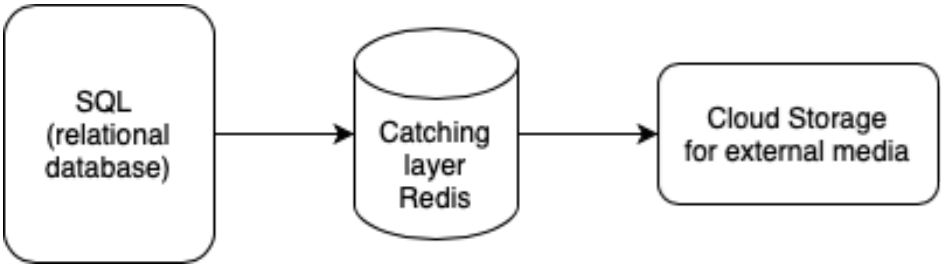
- Caching Layer (Redis): To reduce latency and enhance performance, Redis will temporarily store:
    - Real-time seat availability during booking sessions
    - Active user sessions and preferences
    - Frequently queried movie and showtime data

- Cloud Storage (AWS S3 or Firebase Cloud Storage): Cloud storage is used to offload large assets like:
    - Movie posters, promotional images, and trailers
    - Admin-uploaded media content
    - Auto-generated booking receipts and QR code images

Key Considerations:

- Data Security: All personal and payment data is encrypted both at rest and in transit. Role-based access control and audit logging help enforce data protection policies.
- Scalability: The architecture supports future scaling in both data volume and traffic, with provisions for horizontal expansion.
- Real-Time Operations: Redis and API integration ensure that data like seat reservations, notifications, and confirmations update live across the system.
- Consistency: Relational modeling with strict constraints ensures data integrity and synchronization between linked entities.
- Recovery & Backup: Automated daily backups and disaster recovery protocols are configured to ensure system reliability and business continuity.

8.2 Diagram



8.3 Relational Databse (SQL)

Use of a structured SQL database ensures well-defined, enforced data relationships, secure user authentication, and accurate tracking of bookings and payments. Structured Query Language (SQL) supports robust searching, filtering, and reporting.
PostgreSQL was selected due to its balance between performance, scalability, and modern features, including:

- Advanced indexing
- ACID compliance
- JSON support for flexible data storage (e.g., custom seating layouts)

This allows seamless operation across multiple modules while maintaining high standards of data integrity and performance.

8.4 Single Database Design

For this system implementation, a single relational database instance contains all base tables and relationships. This design simplifies:

- System maintenance

- Backup and restoration processes
- Access control across modules

It also enables easier cross-functional queries, such as linking users with their booking history or identifying movie popularity trends.
While separating the database into multiple services (e.g., for users, payments, and movies) could offer modular scalability, such a design would introduce unnecessary coordination overhead at this stage. The centralized approach promotes ease of development, high reliability, and data integrity across the platform.

8.5 Table (Database) Structure

The following are the main tables that support core functionalities of the system:

8.5.1 Users Table

- user_id (Primary Key)
- name
- email
- password_hash
- role (customer, admin, support)
- phone_number

8.5.2 Movies Table

- movie_id (Primary Key)
- title
- description
- genre
- duration
- rating
- release_date

8.5.3 Showtimes Table

- showtime_id (Primary Key)
- movie_id (Foreign Key referencing Movies)
- theater_id
- start_time
- seat_map (JSON or array)

8.5.4 Tickets Table

- ticket_id (Primary Key)
- user_id (Foreign Key referencing Users)
- showtime_id (Foreign Key referencing Showtimes)
- seat_number
- status (active, used, cancelled)
- price
- qr_code

8.5.5 Payments Table

- payment_id (Primary Key)
- user_id
- ticket_id
- amount
- method
- status
- timestamp

8.5.6 Notifications Table

- notification_id (Primary Key)
- user_id
- message
- type (email, SMS)
- timestamp
- status

8.6 Foreign Key Relationships

Foreign keys enforce consistency between entities:

- The Showtimes.movie_id field references Movies.movie_id, linking scheduled showings with film details.
- The Tickets.user_id and showtime_id fields link customer purchases to specific users and movie times.
- The Payments.ticket_id field links transactions to specific bookings.

These relationships support accurate reporting, data validation, and operational traceability.

8.7 Alternatives Considered

Several technologies and designs were considered during planning:

- NoSQL (MongoDB): Evaluated for schema flexibility and horizontal scaling, but dropped due to a lack of robust relational enforcement required for seat-level bookings.
- Firebase/Firestore: Attractive for mobile integration and real-time sync, but found inadequate for complex queries and analytics needed by the admin dashboard.
- Multi-SQL database model: Considered for isolating services (e.g., users and payments), but rejected to avoid complications in joins, cross-service backups, and reporting workflows.

8.8 Trade Off Discussion

8.8.1 SQL vs. NoSQL

PostgreSQL, a relational SQL database, was chosen for its data consistency, transaction support, and relational modeling. NoSQL was not selected due to its lack of native joins and ACID guarantees.

8.8.2 Single vs. Multiple Databases

A single database design improves performance, simplicity, and maintainability. Multiple databases could isolate services but increase coordination complexity and latency in cross-entity queries.

8.8.3 Caching with Redis

Redis improves performance for real-time seat selection and session management, but introduces the need for cache invalidation strategies to prevent stale or incorrect booking data.

8.8.4 Cloud Storage for Media

Large assets such as trailers, posters, and QR-coded tickets are stored in AWS S3 or Firebase to optimize primary database speed. This separation introduces extra effort in access control and latency handling but significantly reduces query overhead and improves content delivery.

GitHub Link

https://github.com/idelgadod/CS250-Spring-2025/tree/main/Movie%20Theatre%20System