Movie Theater Ticketing System

Software Requirements Specification

Version 2.0

7/21/2025

Guled Hussein

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

Revision History

Date	Description	Author	Comments
7/15/2025	Version 1.0	Guled Hussein	Specified the Requirements
7/21/2025	Version 2.0	Guled Hussein	Software Design Specified

Document Approval

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

Signature	Printed Name	Title	Date
	Guled Hussein	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

Table of Figures:

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) aims at giving a comprehensive and thorough description of the Movie Theater Ticketing System (MTTS) to be developed. It is a contractual and technical base of all stakeholders in the project. It provides a shared terminology on the behavior, functions, limitations, and interfaces of systems to guarantee congruity among developers, testers, project managers, and representatives of the clients.
The intended audience includes:

- **Software developers** responsible for implementation.
- **Quality assurance teams** verifying system conformance.
- **System architects and UI designers** ensuring adherence to requirements.
- **Theater management** overseeing operational goals and ensuring business needs are met.

The objective of this document is to reduce ambiguity, facilitate traceability and ensure that the system fulfills the user expectations and business goals.

1.2 Scope

The Movie Theater Ticketing System (MTTS) is a software product run over the web environment, which will be used to support the process of the ticket reservation, the choice of seats and make the schedule of a physical movie theater along with the reporting. Customers can use the system to navigate the online resource to see the shows and buy tickets, as well as use self-service terminals, however, containing a back-end interface that allows theater employees to control schedules, pricing, and reporting.

Included Features:

- **Online Booking**: Customers can book tickets via desktop or mobile devices 24/7.
- **Seat Selection**: Real-time visual seat map for selection with automatic availability updates.
- **Ticket Issuance**: E-tickets sent via email or printable from kiosk terminals.
- **Schedule Management**: Admins can add/edit movie listings, showtimes, and screen allocations.
- **Reporting Tools**: Generate daily and weekly attendance, sales, and occupancy reports.
- **User Authentication**: Secure registration and login system for both customers and admins.

Excluded Features (Current Version):

- streaming or digital rentals.
- Online food/concession ordering.
- Third-party ticket platform integrations (e.g., Fandango, BookMyShow).
- Loyalty programs or promotional discount systems (may be added in future versions).

System Assumptions and Dependencies:

- A reliable internet connection is assumed for all online transactions.
- The system assumes the availability of third-party payment gateway APIs (e.g., Stripe, PayPal).
- Users will access the platform via modern web browsers (Chrome, Firefox, Safari).

Primary User Classes:

- **Customers (Moviegoers)**: General users who browse listings, select seats, and purchase tickets.
- **Administrators (Theater Staff)**: Employees responsible for scheduling movies, managing prices, and generating reports.
- **Managers**: Senior personnel who analyze trends and oversee theater performance.
- The aim of the MTTS system is to mitigate bottlenecks in operations and increase customer satisfaction, and to offer end-to-end seamless ticketing services under measurable performance criteria (e.g. <5 seconds transaction time, >99% uptime).

1.3 Definitions, Acronyms, and Abbreviations

This section defines all specialized terms and acronyms used in this SRS. A more extensive glossary may be provided in Appendix A. Key definitions include:

Term / Acronym	Definition
MTTS	Movie Theater Ticketing System – The software application described in this SRS.
SRS	Software Requirements Specification – This document, defining system requirements.
GUI	Graphical User Interface – Visual interface through which users interact with the system.
API	Application Programming Interface – External software interfaces (e.g., for payments).
ACID	Atomicity, Consistency, Isolation, and Durability – properties ensuring data transactions consistency and reliability
Customer	End user who uses the system to book, view, or cancel movie tickets.
Administrator	Theater staff with system access to manage movies, schedules, and pricing.
Ticket	A digital or printed reservation confirmation for a movie showing.
Showtime	Scheduled date and time for a movie screening in a particular auditorium.

1.4 References

The following documents and standards have informed the structure and content of this SRS:

1. IEEE Std 830–1998, *Recommended Practice for Software Requirements Specifications*, IEEE Computer Society, 1998.
2. IEEE Std 610.12–1990, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Computer Society, 1990.
3. Sommerville, I. (2015). *Software Engineering* (10th ed.). Addison-Wesley.
4. Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill.
5. CS250 Course SRS Template, Instructor: Dr. Gus Hanna (Fall 2023).
6. Lecture-Use-Cases and Project Notes, CS250 Software Systems Course (2025).

All cited sources are accessible through course resources, university library databases, or the IEEE Xplore Digital Library.

1.5 Overview

This SRS is organized to provide a clear and systematic understanding of the Movie Theater Ticketing System:

- **Section 2 – General Description:** Provides a description of the product from the user's point of view, foremost including the functions, the user characteristics, the constraints, the dependencies.
- **Section 3 – Specific Requirements**: Details of functional and non-functional system requirements, interfaces and use cases.
- **Section 4 – Analysis Models**: This section gives the system behavior models including sequence diagrams, state diagrams and data flow diagrams.
- **Section 5 – Change Management**: Outlines the process of updating of this document based on the changing requirements.
- **Appendices**: Include such details as definitions, schemes, or an example of test data formats.

2. General Description

2.1 Product Perspective

Movie Theater Ticketing System (MTTS) is an independent software product developed to modernize and automate the ticketing reservation and scheduling process of a real physical movie house. It is not based upon legacy system and is created as new system out of scratch. MTTS is, however, supposed to be able to interface other services like payment gateways (e.g. Stripe or PayPal) and notification services (e.g. email or SMS API), to provide secure payment and e-communication with customers.
The system architecture will follow a three-tier structure:

- **Presentation Layer (Frontend)**: Web-based GUI accessible via browsers and responsive to mobile devices.
- **Application Layer (Backend)**: Business logic, scheduling algorithms, and user/session management.
- **Data Layer (Database)**: Centralized relational database managing user data, movie listings, seat inventory, and transactions.

The MTTS will operate within the theater's IT infrastructure and is expected to be hosted on a cloud-based or on-premise server depending on operational decisions.

2.2 Product Functions

The system will provide the following functions:

- **User Registration and Login**
  They are able to register accounts and log in to access their profiles, reserve with a history of their bookings, and make booking reservations.
- **Movie and Showtime Browsing**
  The customers are able to read about current and upcoming movies with their description, trailers, genre, and timings.
- **Seat Selection, Booking Payment and Ticket Generation**
  Customers are now able to select their own preferred seats via real-time seat maps. The system eliminates cases of double-booking through atomic processing of the transactions.
- There is integrated online payment options customers may use to transact. When a transaction is successful, e-ticket is created and sent via email to a user.
- **Ticket Cancellation**
  The theater policy will provide the window when the booked tickets can be canceled.
- **Admin Functions**
  Admin users can add/retract/edit movie listings, configure showtimes and theaters, pricing, and report.
- **Reporting**
  Internal generating reports are daily sales, occupancy figures, and logs on customer activities.

2.3 User Characteristics

The system will support the following categories of users, each with specific characteristics:

1. **Customers**
   - Non-technical users accessing the system via browser or mobile.
   - Require intuitive, guided user interfaces.
   - May be first-time or recurring users.
2. **Administrators**
   - Trained theater staff who manage schedules, pricing, and reports.
   - Require secure login, data entry interfaces, and reporting tools.
3. **Managers**
   - Executive users with access to high-level reports and analytics.
   - Require minimal interaction but expect summarized, visual dashboards.

The responsive design will be taken into consideration in order to make it mobile compatible, and the navigation will be simple to understand by non-expert users.

2.4 General Constraints

- The system should also adhere to data privacy laws (e.g., the GDPR or national alternatives).
- Web frontend should be based on modern browsers (Chrome, Firefox, Safari, Edge).
- All user communications have to be done through secure HTTPS links.
- Database should be compliant to ACID to keep integrity of transactions.
- The booking should be atomic to avoid race conditions (e.g. two users reserving the same seat).
- The system has to support at least 1000 concurrent users at peak times.

2.5 Assumptions and Dependencies

The assumptions and external dependencies applicable to the development and operation of MTTS include the following:

- The third-party payment processing services (e.g., Stripe, PayPal) will be accessible and usable.
- There will be access to computers and reliable internet connection to carry out backend administration by the theater staff.
- Email/SMS APIs (e.g. SendGrid, Twilio) will be added to send confirmation of tickets.
- The system shall run under a Linux based server unless where otherwise indicated.
- Clients will have rudimentary knowledge of internet platforms and online payment.

3. Specific Requirements

This section provides summary of functional and the external interface requirements of the Movie Theater Ticketing System (MTTS). Such requirements determine what the system will do and how it will communicate with the users, the existing hardware, other software systems, and the communication technologies. All the requirements are unambiguous, testable and traceable to user requirements or business aims.

3.1 External Interface Requirements

3.1.1 User Interfaces

The MTTS shall provide two main user interfaces:

1. **Customer Interface (Web/Mobile Browser):**
   The customer interface will be a responsive web application accessible via latest browsers on desktops and in mobile devices. This interface will facilitate, searching movies, check showtimes, seat selection through interactive seat maps, make purchase of tickets and manage personal bookings. It will be focused on usability, having a clean and intuitive layout catering to all users with different technical sophistication. Their profile dashboard will enable them to choose a booking history and digital tickets as well as to log in and register on the site safely.
2. **Administrator Interface (Dashboard):**
   An administrator interface will be well-secured web-based portal, which only certain staff can use. In this interface, a dashboard and data input forms about movie listing, showtime, prices and auditorium setups etc will be provided. The administration will be able to access cancellation records, audit logs, and the setting of systems as well. Each of the interfaces will be created so that navigation is easy, they meet accessibility requirements (e.g., WCAG 2.1 AA), and they have cross-browser compatibility.

3.1.2 Hardware Interfaces

A well-encrypted web-based portal of an administrator will be an administrator interface, and only specific staff members will be able to use this administrator interface. In such an interface, a dashboard and data input forms regarding movie listing, showtime, prices and auditorium setup etc would be given. The administration will also have an access to cancellation history, audit logs and system set-up. All the interfaces will be designed in such a way that it is easy to navigate through them, they will be accessible (e.g. WCAG 2.1 AA), and they will be cross-browser compatible.

3.1.3 Software Interfaces

MTTS will rely on the connection to multiple external software services. It will be linked with third-party payment gateway API (e.g., Stripe, PayPal or a local provider) to securely process online payments. On receipt of the payment, the system will integrate with an e-mail notification API (e.g. SendGrid) to send electronic tickets and confirmations. The frontend and backend services will internally communicate via RESTful APIs and data will be stored in a relational database such PostgreSQL or MySQL, possibly based on deployment preferences. The server-side part will be created on the framework, e.g., Django or Node.js, and the frontend technologies, e.g., React or Vue.js, will be used to render and engage with the users.

3.1.4 Communications Interfaces

The transmission of any information between the client and server components will be implemented through HTTPS-based secure communications. The APIs within the system are going to be REST-based, with elements getting listed by utilizing GET, reservations being made with POST, updating per-user or per-seat with PUT, and canceling the reservation with the DELETE. The database will be stored on a confidential server and will be secured against malicious attacks namely firewalls, access control lists and encryption-at-rest policies. Transaction records and logs will be timestamped and auditable in order to enable a post-event review and compliance.

3.2 Functional Requirements

In this section, essential characteristics of Movie Theater Ticketing System (MTTS) will be outlined. All features are described in the same way: introduction, inputs, processing, outputs and errors. Section 3.3 includes the use-case descriptions, and use-case diagrams where the user can find the non-functional requirements in 3.5.

3.2.1 User Registration and Authentication (FR1)

3.2.1.1 Introduction

New users will be able to register and old users will be able to authenticate in the system. This feature allows customization of services including booking history, ticket cancellation and saved preferences.

3.2.1.2 Inputs

- Full Name (text)
- Email Address (validated, unique)
- Password (must meet complexity rules)
- Login credentials (email and password)

3.2.1.3 Processing

When registering the system checks uniqueness of email and password validity. Passwords are salted and hash. In the process of logging-in, authentication takes place against the database based on credentials. Secure tokens are used to create a session.

3.2.1.4 Outputs

- Success confirmation on registration.
- Session token on successful login.
- User redirected to dashboard.

3.2.1.5 Error Handling

- Invalid email format or existing email: Display error and block submission.
- Weak password: Prompt user to follow complexity rules.
- Incorrect login credentials: Display message with retry option.
- Multiple failed logins: Lock account temporarily and notify user.

3.2.2 Browse Movies and Showtimes (FR2)

3.2.2.1 Introduction

Users can explore available movies and showtimes through a categorized and filterable interface.

3.2.2.2 Inputs

- Filter selections (genre, date, time, language)
- Search keyword (movie title or actor)

3.2.2.3 Processing

The system to access the movies and related metadata in the database, query them using user input and output available showtimes in the results list. Every film listing entails title, poster, description and screenings schedule.

3.2.2.4 Outputs

- A dynamic list of movie cards with showtimes.
- Visual timeline of showings by date and auditorium.

3.2.2.5 Error Handling

- No results found: Display "No movies match your criteria."
- Database connection failure: Show fallback message and retry option.

3.2.3 Seat Selection, Booking Payment and Ticket Generation (FR3)

3.2.3.1 Introduction

Customers can select specific seats from a visual seat map and proceed to book them through a guided checkout process.

3.2.3.2 Inputs

- Selected showtime
- Selected seats
- User authentication token (session)
- Payment method and details

3.2.3.3 Processing

The system displays the diagram of the place and seats; depending on the layout of the auditorium and also indicates booked seats. Pages with chosen seats are locked within a certain amount of time (e.g. 5 minutes). Once the payment gets approved through the payment gateway, the seats are then closed permanently and a confirmation is created.

3.2.3.4 Outputs

- Confirmation page with booking ID
- E-ticket generated (PDF format or QR code)
- Email confirmation sent to user

3.2.3.5 Error Handling

- Attempted double booking: Prevent and prompt user to reselect.
- Payment failure: Notify user and unlock held seats.
- Session timeout: Expire seat lock and request rebooking.

3.2.4 Ticket Cancellation (FR4)

3.2.4.1 Introduction

Customers may cancel their ticket(s) through the system within an allowable cancellation period and, depending on policy, receive a refund.

3.2.4.2 Inputs

- Booking ID
- User authentication (must match booking account)

3.2.4.3 Processing

The system confirms and the booking and eligibility of cancellation. In case it is valid, the booking is canceled, the seat is freed, and the refund is done using the payment API.

3.2.4.4 Outputs

- Cancellation confirmation message
- Email notification to the customer
- Refund transaction (if applicable)

3.2.4.5 Error Handling

- Cancellation deadline passed: Inform user and disallow action.
- Unauthorized cancellation attempt: Reject and log event.
- Refund API failure: Retry and escalate to admin if needed.

3.2.5 Admin Movie and Schedule Management (FR5)

3.2.5.1 Introduction

Authorized theater staff can add, edit, or remove movie listings and configure auditorium schedules and pricing.

3.2.5.2 Inputs

- Movie metadata (title, description, poster, duration, rating)
- Showtimes (date, time, auditorium ID)
- Pricing tiers
- Admin credentials/session token

3.2.5.3 Processing

Validated inputs are held in the backend database. Updates of the movie listings are reflected on to the user facing interfaces instantaneously. The system eliminates time conflicts and checks auditorium capacity.

3.2.5.4 Outputs

- Confirmation of new or updated entries
- Immediate visibility of changes on the customer portal

3.2.5.5 Error Handling

- Schedule conflicts or overlapping showtimes: Reject entry and notify admin.
- Missing or malformed fields: Display input validation messages.
- Unauthorized access: Redirect to login and log attempt.

3.2.6 Reporting (FR6)

3.2.6.1 Introduction

The system shall support internal reporting features to help theater managers and administrators monitor performance, identify trends, and make informed operational decisions.

3.2.6.2 Inputs

- Date range
- Movie title (optional)
- Auditorium ID (optional)

- Report type (sales, occupancy, booking trends)

3.2.6.3 Processing

Approved inputs are stored in a backend database. Any updates of the movie listings are projected onto user facing interface in real time. The technology removes time clash and auditorium confirmation.

3.2.6.4 Outputs

- PDF or CSV exportable reports
- On-screen summaries with charts and key metrics
- Dashboard view for quick insights

3.2.6.5 Error Handling

- No data for selected criteria: Display “No results found” message
- System/database error: Log issue and alert administrator

3.3 Use Cases

In this part, the following core use cases are outlined in structured way. Any use case consists of actors, preconditions, normal flow of events, and extensions. It aims at explaining interactive/reactive behaviour of system and aids design and testing.

Use Case 1: Book Movie Ticket

- **Use Case ID:** UC1
- **Use Case Name:** Book Movie Ticket
- **Actor(s):** Customer
- **Description:** This use case describes the process by which a registered user browses showtimes, selects seats, makes a payment, and receives a digital ticket.
- **Preconditions:**
    - The user must be logged into the system.
    - The movie and showtime must be available.
- **Main Flow:**

1. The customer navigates to the movie listings.
2. The customer selects a movie and showtime.
3. The system displays a seat map for the selected show.
4. The customer selects one or more seats.
5. The system locks the selected seats temporarily.
6. The customer proceeds to payment and enters payment details.
7. The system processes the payment via a payment gateway.
8. On successful payment, the booking is confirmed.
9. The system updates seat status to “booked.”
10. An e-ticket is generated and sent to the customer via email.

- **Postconditions:**
    - Seats are reserved and marked as booked.
    - A booking record is stored in the user’s account.
- **Exceptions/Alternate Flows:**
    - Payment fails: User is notified and seats are released.
    - Session expires: User is redirected to login and must restart the booking.

Use Case 2: Cancel Ticket

- **Use Case ID:** UC2
- **Use Case Name:** Cancel Ticket
- **Actor(s):** Customer
- **Description:** This use case describes how a customer can cancel a previously booked ticket and request a refund.
- **Preconditions:**
    - The user must be logged in.
    - The booking must be within the allowable cancellation window.
- **Main Flow:**

1. The customer navigates to the "My Bookings" page.
2. The customer selects the booking to cancel.
3. The system checks if the booking is eligible for cancellation.
4. If eligible, the user confirms the cancellation.
5. The system updates the booking status to "cancelled.
6. Seats are released and marked available.
7. A refund request is sent to the payment gateway.
8. A cancellation confirmation is shown and emailed to the user.

- **Postconditions:**
    - The ticket is marked cancelled and seat is freed.
    - A refund process is initiated.
- **Exceptions/Alternate Flows:**
    - Cancellation deadline has passed: Inform user and disallow action.
    - Payment refund fails: Escalate to admin with notification.

Use Case 3: Add Movie and Schedule (Admin)

- **Use Case ID:** UC3
- **Use Case Name:** Add Movie and Schedule Showtimes
- **Actor(s):** Administrator
- **Description:** This use case describes how an admin adds a new movie and schedules showtimes with specific auditorium and pricing configurations.
- **Preconditions:**
  - Admin is logged into the system.
  - Auditorium exists and has available time slots.
- **Main Flow:**

1. Admin accesses the dashboard and navigates to "Add Movie."
2. Admin enters movie title, description, genre, language, rating, and upload poster.
3. Admin selects available auditoriums and times for showings.
4. Admin sets seat pricing and seat configuration.
5. The system checks for scheduling conflicts.
6. If valid, the movie and showtimes are saved in the database.
7. New listings appear on the customer portal immediately.

- **Postconditions:**
  - The movie is added to the system.
  - Showtimes and seats are available for booking.
- **Exceptions/Alternate Flows:**
  - Schedule conflicts detected: System prompts admin to reschedule.
  - Missing mandatory fields: Show validation errors and prevent submission.

3.4 Classes / Objects

It is in this section that the main classes (objects) that make up the core of Movie Theater Ticketing System are identified. These classes are mentioned as the main entities which will be modeled during the design stage of the system and which will be implemented into the code. Every class has a short overview of its properties and its connections with other things.

3.4.1 User

3.4.1.1 Attributes

- user_id (Integer): Unique system-generated identifier for the user.
- name (String): Full name of the user.
- email (String): Unique login identifier, validated format.
- password_hash (String): Encrypted password for authentication.
- phone_number (String): Optional contact number for communication.
- role (String): Either 'customer' or 'admin', used for access control.
- created_at (DateTime): Account creation timestamp.

3.4.1.2 Functions

- Register a new account (Refer to Functional Requirement 3.2.1, Use Case UC1)
- Log in to the system (Refer to Functional Requirement 3.2.1, UC1)
- View and manage profile
- View booking history (Linked to Booking object)
- Perform authorized actions based on role (admin functions – UC3)

3.4.2 Movie

3.4.2.1 Attributes

- movie_id (Integer): Unique identifier for each movie.
- title (String): Name of the movie.
- description (String): Brief synopsis of the movie.
- genre (String): Classification such as Action, Drama, etc.
- language (String): Language of the film.
- duration (Integer): Runtime in minutes.
- poster_url (String): Path to the poster image.
- rating (String): Age or content rating (e.g., PG-13).

3.4.2.2 Functions

- Display movie listings for customers (Refer to Functional Requirement 3.2.2, Use Case UC1)
- Add/edit/delete movie records (Admin only; Refer to UC3)

3.4.3 Auditorium

3.4.3.1 Attributes

- auditorium_id (Integer): Unique identifier for the auditorium.
- name (String): Label or name of the auditorium.
- total_seats (Integer): Capacity of the room.
- seat_layout (JSON/Object): Layout structure defining seat rows and columns.

3.4.3.2 Functions

- Store configuration for seat map rendering (Referenced in UC1 and UC3)
- Associate with specific showtimes for scheduling
- Enable seat assignment (Functional Requirement 3.2.3)

3.4.4 Showtime

3.4.4.1 Attributes

- showtime_id (Integer): Unique identifier for a scheduled show.
- movie_id (Foreign Key): Links to a specific movie.
- auditorium_id (Foreign Key): Links to a specific auditorium.
- start_time (DateTime): When the movie begins.
- end_time (DateTime): Automatically computed from duration.
- base_price (Decimal): Ticket cost before discounts or category surcharges.

3.4.4.2 Functions

- Associate movie and auditorium for booking purposes (UC1)
- Display showtime availability to users (Functional Requirement 3.2.2)
- Allow admins to configure new showings (UC3)

3.4.5 Seat

3.4.5.1 Attributes

- seat_id (Integer): Unique identifier for the seat.
- showtime_id (Foreign Key): Links to a specific showtime.
- seat_label (String): Code like A1, B3, etc.
- seat_type (String): Standard, Premium, VIP, etc.
- status (String): Available, Reserved, Booked.

3.4.5.2 Functions

- Display real-time seat map to users (3.2.3, UC1)
- Prevent double booking through seat locking (3.2.3)
- Allow cancellation and release of seat (UC2)

3.4.6 Booking

3.4.6.1 Attributes

- booking_id (Integer): Unique identifier for the booking.
- user_id (Foreign Key): Refers to the user who booked.
- showtime_id (Foreign Key): Refers to the scheduled movie.
- seat_ids (List/Array): Set of selected seats.
- payment_status (String): Paid, Unpaid, Refunded.
- booking_time (DateTime): Timestamp of booking.
- cancellation_time (Nullable DateTime): Timestamp if cancelled.

3.4.6.2 Functions

- Store confirmed ticket data (UC1)
- Support viewing history (3.2.1)
- Enable cancellation (3.2.4, UC2)
- Integrate with Payment and Seat objects

3.4.7 Payment

3.4.7.1 Attributes

- payment_id (Integer): Unique identifier for the transaction.
- booking_id (Foreign Key): Associated booking.
- amount (Decimal): Total paid by the user.
- method (String): Payment channel (e.g., credit card, PayPal).
- status (String): Completed, Failed, Refunded.
- timestamp (DateTime): Time of transaction.

3.4.7.2 Functions

- Process transactions securely (3.2.3)
- Enable refunds during cancellations (3.2.4)
- Update booking status based on transaction result

3.5 Non-Functional Requirements

The quality attributes and the performance limitations of the Movie Theater Ticketing System include the following non-functional requirements. These are system-wide requirements, which maintain reliability, security, usability, maintainability, and performance of all modules.

3.5.1 Performance Requirements

- At peak time, the system must serve a minimum of 1,000 concurrent users without declining responsiveness.
- 95 percent of the transactions (booking of tickets, payment, cancellation) will be performed within 3 seconds on normal load.
- The number of available and sold seats as well as seat reservation will be updated in real time with a no greater than 1s latency of synchronization.
- Summarizational reporting and data analysis: Daily reporting and data analytics will produce summary reports (e.g., revenue, occupancy) in less than 5 minutes of query.
- The response time of the basic queries (e.g., movie listings, showtimes) should not be more than 2 seconds on average (99-percentile).

3.5.2 Availability and Reliability

- The system will have an uptime of 99.5 percent and above in a month, minus scheduled maintenance.
- Routine maintenance should not take more than 2 hours per week and it should take place at least at off-peak times (12 AM- 6 AM local time).
- The system should accommodate automatic recovery of the system to restore the service in not more than 5 minutes in case of server failure.
- Mean Time Before Failures (MTBF) will be above 30 days.
- There will be no single point of failure; it should have redundancy of the database and application servers.

3.5.3 Security Requirements

- User passwords will be stored through a hash and a salt that meets industry encryption standards (e.g., SHA-256 or better).

- Communication between clients and servers should be in HTTPS/TLS (TLS 1.2 or later) in all cases to avoid intercepting data in communication.
- The payment processing will be in line with PCI-DSS of secure credit cards.
- The system will use role-based access control (RBAC) to differentiate among customers, administrators and managers.
- Account lockout shall be in place due to failed logins of 5 logins in number consecutively to last a maximum of 15 minutes, after which the process can be unlocked manually.

3.5.4 Usability Requirements

- The system will have a Graphical User Interface (GUI), webpage and mobile browser (UIResponsiveness) accessible.
- Users should be able to fill and book tickets in not more then 5 clicks or steps from movie selection to confirmation.
- The interface should comply with accessibility standards, it should be able to navigate through the interface with a keyboard and screen readers (WCAG 2.1 Level AA).
- On-screen Tooltips and error messages SHALL lead users through frequent tasks and form corrections.

3.5.5 Maintainability and Supportability

- The system will be modular in order to enable it to make individual modules (e.g. payment module, user management) to be upgraded in a singular fashion.
- System logs and errors will be stored with timestamps and will have to be available to the administrator within 2 minutes.
- New movies, showtime, and prices updates are allowed to be configured by the administrators using the dashboard without alteration of code.
- Documentation (developer API, deployment guide, admin manual) will be updated and supplied every release cycle.

3.5.6 Portability

- The MTTS application will be platform-independent and will run on all modern operating systems (Windows, Linux, macOS), also with JavaScript-enabled browsers.
- The system will be fully functional on Chrome, Firefox, Safari, and Edge with 95+ percent consistency between versions.
- It will be containerized during the deployment (e.g. Docker) to enable the replication of the environment at both development and service servers.

4. Software Design Specification

4.1 System Description

A modular multi-tiered software, the Movie Theater Ticketing System (MTTS) is tailored in such a way that it gives optimization of the ticket-management process of a cinema facility. As a developer, MTTS can be viewed as service-oriented system, in which loosely coupled but highly cohesive components are combined into a service-based system, communicating using clearly specified interfaces.
In principle, MTTS intends to facilitate the processing of two key types of the user: Customers (moviegoers) and Administrators (theater staff). Customers are able to view movie listings, showtimes, pick seats and buy tickets either on line or via kiosk. Administrators can set up shows, modify movie content, layout auditoriums and produce reports. Its architecture makes the system scalable, responsive and consistent in data.
In order to facilitate such features, the system will logically consist of the following parts:
Presentation Layer: This entails the customer facing and admin facing. The customer user interface gives users ability to select movie and book seats and payment, and the administrative dashboard gives the user the ability to upload movies and edit as well as change schedule.
Application Layer: The business logic of the system is here. It manages the action of seat primer, transaction clearance, cancellation policy, and schedule conflict.
Data Access Layer: This layer negotiates with the database. It summarizes all the logic that it needs to extract, insert, modify, or delete the records pertaining to the movies, show times, users, seats, and payments.
External Service Integration: This has secure 3rd party payment gateway porting, email ticket services, and analytics. The API contracts make them resilient and fallback in case of failed transaction.
Database Layer: Structured data will be stored and handled using a relational database. The records will be kept in tables containing information about the users, booking, seat, schedule, transaction and audit logs.
The system will use a Model-View-Controller (MVC) architectural pattern that encourages maintainability, testability and separation of concerns. MTTS will be designed to support concurrent users so that the availability of seats is always updated in real-time and business rules (e.g. cancellation cutoffs, pricing policies etc) are applied to all modules.
The design is cross-platform but will be realized with the help of modern web development technologies, including Python/Django backend and ReactJS frontend. This makes the system future-proof so that they can be installed in a cloud and on-premises environment.

4.2 Software Architecture Overview

Movie Theater Ticketing System (MTTS) adheres to the layered modular approach that is based on Model-View-Controller (MVC) paradigm. The given system is segmented into the modules to which presentation, business logic, data access, and external integrations are encapsulated which makes the system to be scalable, maintainable, and testable. The architecture will be web/mobile

client-compatible and will have a lean back-end suitable to run administrative activities and generate reports.

4.2.1 System Components and Responsibilities

1. **Client Interfaces**
   The system allows the use of several access points of users:
   - Web App (developed in React.js)
   - Mobile Application which can be developed using React Native or Flutter.
   - Kiosks (On-site touch-screen enabled browser interface)

Under these interfaces, the consumers are able to sign up, log in, navigate movies, pick some seats, pay and obtain electronic tickets.

2. **Authentication and Session Management**
   Intermediate authentication module deals with:
   - User login and registration
   - Role-based access control (admin, customer)
   - Tokenized sessions (e.g., JWT)
3. **Controller Layer**
   The controller layer is a traffic director that distributes a HTTP request to a service logic. Each endpoint is made to support a particular exchange like ticket booking, getting movies, or updates on an admin.
4. **Service Layer (Business Logic)**
   This is the core of the system, holding upon it the services that enforce the fundamental rules of the business. Some major services include:
   - **UserService**: Manages profiles, authentication, and history.
   - **MovieService**: Handles movie listings, metadata, and categorization.
   - **ScheduleService**: Manages showtimes, auditorium assignment, and availability.
   - **BookingService**: Facilitates real-time seat selection, ticket issuance, and confirmation.
   - **PaymentService**: Coordinates with external payment gateways (e.g., Stripe) for secure transactions.
   - **ReportService**: Generates real-time and historical sales reports for admins.
5. **Data Access Layer (DAL)**
   Components of DAL take care of queries and updates on the PostgreSQL database. All services have a clear way of communicating with the DAL which needs to only be well-defined repositories to ensure logic and persistence separation.
6. **PostgreSQL Database**
   The relational database is where all persistent data will be found:
   - User accounts and credentials
   - Movies and showtime schedules
   - Seat maps, bookings, and payment records
7. **External Integrations**
   - **Payment Gateway**: Used for secure credit/debit card transactions.
   - **Email Service**: Sends booking notifications and tickets to the users by e-mail (e.g., via SendGrid).

4.2.2 Architectural View (Diagram)

See the architecture diagram in the following that illustrates the layers in the system and the data flows:

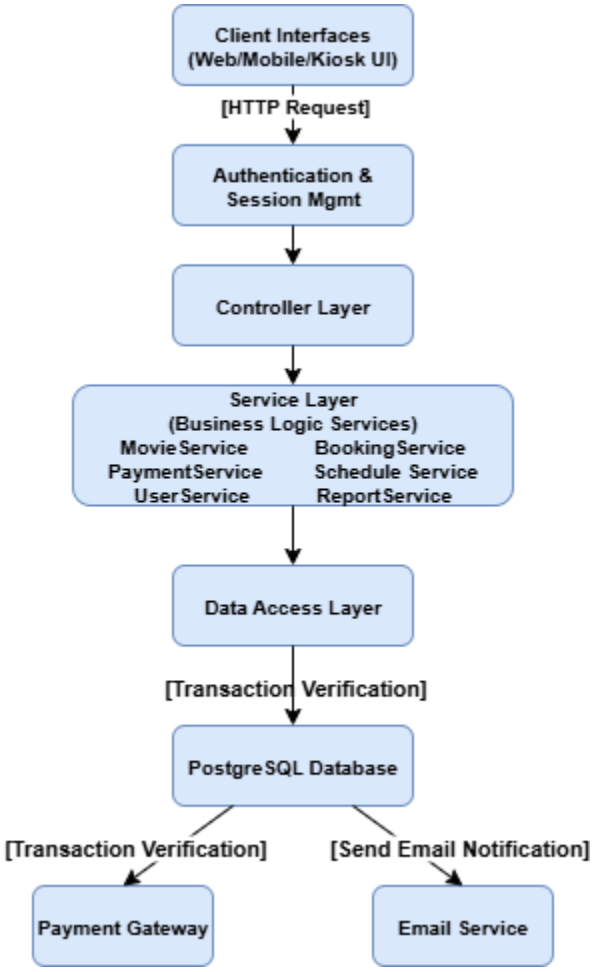

Figure 1:System Architecture Overview

4.2.3 Design Principles Applied

- **Separation of Concerns**: Each layer has its own responsibilities and this eliminates coupling and makes debugging easy.
- **Scalability**: Horizontal scaling is possible because of the stateless controller and service layers.
- **Extensibility**: The incorporation of the new services (e.g. loyalty points) does not require a great deal of modification to the previously written code.
- **Security**: All endpoints of the API can be authenticated through security middleware and API endpoints are secured.

4.3 UML Diagram Description

Movie Theater Ticketing System (MTTS) UML Class Diagram represents the static components of the system, but it reveals the classes that are important to the system and their attributes, operations and depict the relations that exist between the classes. The diagram is the basis of system implementation and it is a simple way to give an actual picture of the interaction between core components of the system.

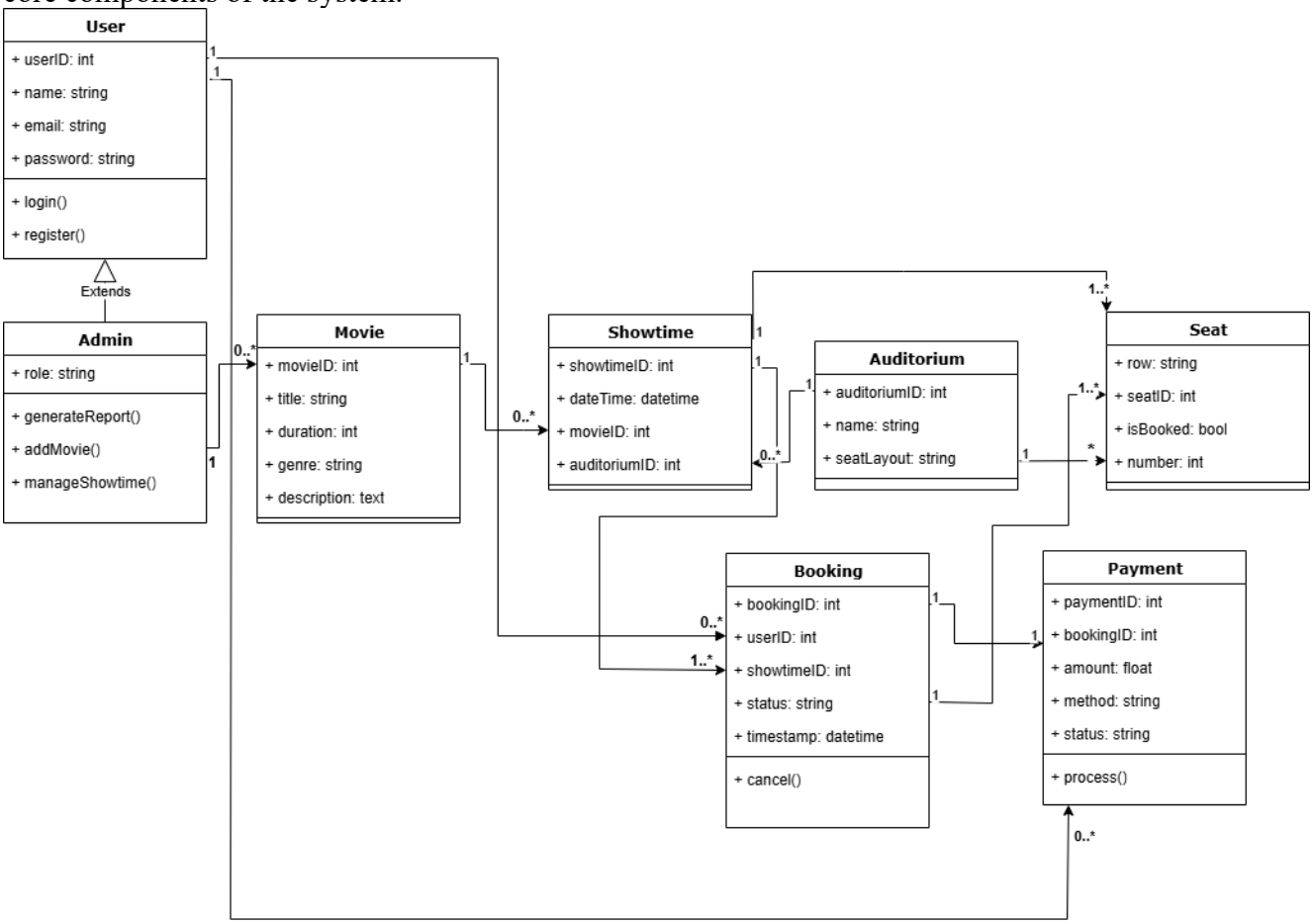

Figure 2:UML Class Diagram

4.3.1 Class Descriptions:

4.3.1.1 User

- **Attributes:**
  - userID: int – Unique identifier for each user.
  - name: string – User's full name.
  - email: string – Email used for login and contact.
  - password: string – Encrypted password string.
- **Operations:**
  - login(): void – Authenticates the user.
  - register(): void – Registers a new user.
- **Relationships:**
  - One User can have **zero or more (0..*)** Booking records.
  - The Admin class extends User.

4.3.1.2 Admin (inherits from User)

- **Attributes:**
  - role: string – Specifies the role (e.g., "admin").
- **Operations:**
  - generateReport(): void – Generates system usage or financial reports.
  - addMovie(): void – Adds a new movie entry to the system.
  - manageShowtime(): void – Manages and modifies showtime data.
- **Relationships:**
  - Inherits all relationships and behaviors of User.

4.3.1.3 Movie

- **Attributes:**
  - movieID: int – Unique movie identifier.
  - title: string – Name of the movie.
  - duration: int – Duration in minutes.
  - genre: string – Genre such as Action, Drama, Comedy.
  - description: text – Summary or storyline.
- **Relationships:**
  - One Movie is associated with **zero or more (0..*)** Showtime entries.
  - Each Showtime is linked to exactly **one (1)** Movie.

4.3.1.4 Showtime

- **Attributes:**
  - showtimeID: int – Unique identifier for the showtime.
  - dateTime: datetime – Date and time of screening.
  - movieID: int – Reference to the associated movie.
  - auditoriumID: int – Reference to the auditorium.
- **Relationships:**
  - One Showtime is for **one (1)** Movie.
  - One Showtime occurs in **one (1)** Auditorium.

- One Showtime may have **zero or more (0..*)** Booking entries.

4.3.1.5 Auditorium

- **Attributes:**
  - auditoriumID: int – Unique identifier.
  - name: string – Auditorium name (e.g., Hall A).
  - seatLayout: string – Layout details in encoded format.
- **Relationships:**
  - One Auditorium can host **zero or more (0..*)** Showtime sessions.
  - One Auditorium contains **one or more (1..*)** Seat entries.

4.3.1.6 Seat

- **Attributes:**
  - seatID: int – Unique identifier.
  - row: string – Row label (e.g., "A").
  - isBooked: bool – Indicates booking status.
  - number: int – Seat number within the row.
- **Relationships:**
  - Each Seat belongs to exactly **one (1)** Auditorium.

4.3.1.7 Booking

- **Attributes:**
  - bookingID: int – Unique booking reference.
  - userID: int – Reference to the booking user.
  - showtimeID: int – Reference to the showtime.
  - status: string – Booking status (e.g., Confirmed, Cancelled).
  - timestamp: datetime – When the booking was made.
- **Functions**
  - cancel(): void – Cancels the booking and updates related entities.
- **Relationships:**
  - Each Booking is linked to exactly **one (1)** User.
  - Each Booking refers to exactly **one (1)** Showtime.
  - One Booking has exactly **one (1)** Payment.

4.3.1.8 Payment

- **Attributes:**
  - paymentID: int – Unique payment reference.
  - bookingID: int – Reference to the related booking.
  - amount: float – Total payment amount.
  - method: string – Payment method used.
  - status: string – Status (e.g., Completed, Failed).
- **Functions:**
  - process(): void – Executes the payment transaction.
- **Relationships:**
  - Each Payment is linked to exactly **one (1)** Booking.

4.3.2 Design Summary:

The model is modular, maintainable and its responsibilities are separated very well:

- Individual important domain entities (movie, showtime, user, booking, and so on) are modeled in an individual way.
- Referential integrity exists between components as a result of foreign key relationships.
- Methods like cancel() and process() allow explaining important class level behaviours.
- The structure is extendable, e.g. go ahead and add Review, Concession, or Discount classes in the future.

The diagram has been developed in such a way that essential system flows, including browsing a movie, selection of a seat, booking and payment should be traced through the relationship between the objects and their operations.

4.4 Development Plan and Timeline

Designing of Movie Theater Ticketing System (MTTS) is an iterative and systematic procedure. This section provides the stages of development of the project, the personal roles, project schedule plan, as well as the practices of the contribution. Since it is a one-member project, the related issues of development such as design, implementation and testing will be carried out by one person.

4.4.1 Phased Timeline

The process of development is conceptualized into different, sequential stages which are spaced out within 5 weeks. Every stage will feature critical deliverables that will facilitate the fulfillment of the MTTS.

Phase	Week	Key Deliverables
Phase 1: Requirements Analysis	Week 1	Completion of Software Requirements Specification (SRS), Use Cases
Phase 2: Design Specification	Week 2	Software Architecture Diagram, UML Class Diagram, Class Descriptions
Phase 3: Core Implementation	Week 3–4	Development of core modules: registration, seat booking, payment, and admin panel
Phase 4: Testing and Debugging	Week 5	Execution of test plans, validation of use cases, bug fixes, performance tuning

4.4.2 Responsibilities

Because I am the only developer, I will be in charge of:

- SRS documentation and SRS requirements elicitation
- Software architecture and class modeling Developing software architecture and class models
- Front end and backend implementation
- Incorporation of 3 rd party services (e.g. payment gateway)
- Unit and integration testing Conducting unit and integration testing
- Version control and use of GitHub repository
- Capturing of progress and keeping the system traceable

4.4.3 GitHub Contribution and Tracking

On the basis of course requirements:

- I will source code and documentation, diagrams to the group GitHub repository
- The version control conventions (a clear commit message, well-structured branches) will be adhered to at all times
- This repository will picture entire history and traceability of work throughout the phases
- Every major deliverable will have at least one significant commit made

4.4.4 Development Tools and Stack

- **Frontend**: HTML, CSS, JavaScript (React optional)
- **Backend**: Node.js with Express or Python Flask
- **Database**: PostgreSQL or MySQL
- **Version Control**: Git, hosted on GitHub
- **Design Tools**: Draw.io (Architecture + UML diagrams)
- **IDE**: Visual Studio Code

A. APPENDICES

A.1 Appendix A – Glossary of Terms

Term	Definition
MTTS	Movie Theater Ticketing System – the software product described in this document.
SRS	Software Requirements Specification – this formal document detailing all system requirements.
GUI	Graphical User Interface – the interface used by users to interact with the system visually.
Admin	Authorized theater staff who manage movies, showtimes, and reporting features.
Customer	A general user (moviegoer) who uses MTTS to browse, book, and cancel tickets.
Showtime	The scheduled start time for a particular movie screening.
Booking	A reservation made for a seat(s) for a specific showtime.
E-ticket	An electronic ticket generated upon successful booking, sent via email.
UML	Unified Modeling Language – used to visually model classes and relationships in software design.
Commit	A version control action that saves a set of changes to the GitHub repository.
Change Request	A formal proposal to modify the requirements or design of the system.

A.2 References

1. IEEE. (1998). *IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998)*. The Institute of Electrical and Electronics Engineers, Inc. https://standards.ieee.org/standard/830-1998.html
2. Sommerville, I. (2015). *Software Engineering* (10th ed.). Boston, MA: Addison-Wesley. ISBN: 978-0133943030
3. Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education. ISBN: 978-0078022128
4. ISO/IEC/IEEE 29148:2018. *Systems and software engineering — Life cycle processes — Requirements engineering*. https://www.iso.org/standard/72089.html
5. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN: 978-0201633610
6. Glinz, M. (2007). *On Non-Functional Requirements*. In *15th IEEE International Requirements Engineering Conference (RE 2007)*. https://doi.org/10.1109/RE.2007.45
7. Hanna, G. (2025). *CS250 Course Lecture Notes and Software Requirements Template*, Department of Computer Science, [Your University Name].
8. Hanna, G. (2025). *CS250 Use Case Lecture Examples*, [Course Handouts].