# Movie Ticketing System

## Software Requirements Specification

Version 1.3

9/18/2025

Group18
James Shumate
Philip Revak
Anthony Diego

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Fall 2025

## Revision History

| Date | Description | Author | Comments |
|---|---|---|---|
| 09/18/25 | Version 1 | Group-18 | Document Creation |
| 10/09/25 | Version 1.1 | Group-18 | Added Section 6. Software Design Specifications |
| 10/23/25 | Version 1.2 | Group-18 | Added Section 7 Test Cases |
| 11/6/25 | Version 1.3 | Group-18 | Added Section 8 Data Management Strategy |

## 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

## 1.1 Purpose

The purpose of this document is to define requirements for the movie ticketing system and to detail the software's functionality and any constraints. The intended audience for this document is developers and investors of the company.

## 1.2 Scope

The movie ticketing website allows anyone with a stable internet connection and a modern, majorly supported browser, to view information regarding movie showings across our multiple movie theaters and allows them to select seats and purchase tickets. It includes a login page where customers can login and view their purchase history and receive special discounts. When an administrator logs in, they will have administrative view and access to the website, allowing them to view graphs and statistics regarding earnings, see individual purchases, and modify them by cancelling and refunding the order. The statistics page is customizable by date, time, theater, and movie. The website will have network protection, preventing DDOS attacks. We will have a database containing encrypted data of the user's account information and payment history, as well as data of all ticket purchases. Credit card information will not be stored. The deadline is 12/11/2025.

## 1.3 Definitions, Acronyms, and Abbreviations

UI: User Interface
DDOS: distributed denial-of-service
Special characters: Characters that aren't (A-Z, a-z, or 0-9), such as $!,@,#,$,%,^,&,*,etc.

## 1.4 References

Software Architecture Diagram (SWA)
https://docs.google.com/drawings/d/1K5nIu70fzIAlyaKlleOwsdDxzrIhDZJ2XuAlhCZlkBk/edit?usp=sharing

UML Class Diagram
https://github.com/JCoding8127/CS250-04-Group18/blob/main/UML%20Class%20Diagram.drawio

Test Cases
https://github.com/JCoding8127/CS250-04-Group18/blob/main/Test%20Cases.xlsx

## 1.5 Overview

This document contains information on the purpose of our product, its functionality, data on the users of the product, and use-case information. We split the information into three sections.

The first section is an introduction to the document, the second section explains the functions of our system along with constraints. The third section follows the requirements our products follow.

## 2. General Description

### 2.1 Product Perspective

There are many movie theater ticketing websites that provide a number of extra functionalities that our website will not contain such as the ability to purchase food online. However we believe the advantage our software will provide is that without many of these extra functionalities it will be simpler, faster, and easier to use for customers most of whom do not benefit from these extra functionalities. For example it is estimated only 1 in 10 people have ever preordered food at a theater. The absence of these functionalities will also allow us to work with a larger range of theaters by not imposing as many requirements onto them, like preparing and distributing preordered food.

### 2.2 Product Functions

Users will first be presented with a list of nearby theatres available to select. Once selected, they will be shown a catalog of movies which they can select. After movie selection, the user will choose the number of their party and their seating arrangement based on available seats. Finally, they will purchase their tickets and receive their receipts. Logging in is optional, by default users are considered guests.

Tickets may qualify for a discount depending on age.

### 2.3 User Characteristics

The eventual users of the movie ticketing software will be members of the general public that are capable of purchasing movie tickets online. Customer users of this software will not be required to have more than basic computer skills so the software will need to be easy to use even for teens. The software will also need to support access from customer service members in the company who will have better knowledge of the website and the ability to modify customer orders.

### 2.4 General Constraints

Some constraints the software will have are constraints from the browser it is running on, constraints from the payment processor, server and network restrictions, as well as constraints of theaters that are supported.

### 2.5 Assumptions and Dependencies

It is assumed that the user's device has enough computational power, memory, resources, and a stable network connection in order to navigate the website as intended. It is also assumed that the user is using the latest major version of a supported browser.

# 3. Specific Requirements

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

The page a first time user sees is either the theater location selector or move search. The theater location selector will consist of a list of buttons containing the city and name of the theater, in order from closest to farthest. After selecting the theater, the user will be shown a catalog of movies with their poster art and details. This page will also contain a date selector which will bring them to a movie catalog page only containing movies from that date. If the user searches by movie they will be shown theaters and times when the movie is playing. After selecting a movie and theater, the user will be shown a seat selector, visually showing all of the seats available and taken by color code and key. Available seats can be temporarily "reserved" by interacting with their UI element. After seat selection, the payment page will be displayed that shows the user's total and has textboxes requiring payment information and discount code(s). Finally, after purchase, the user will be presented with a receipt page that shows their ticket details and purchase receipt. There is a login page that requires an email and password to be entered in textboxes. There is also an account info page, where logged in users can view information about their account and change their details. There is a separate section of user interfaces for logged-in administrators. They will see purchase histories, buttons to refund tickets, and account information of logged-in users and guests. There will be a page that shows statistical graphs containing revenue earned with category selection and timeframe selection.

### 3.1.2 Hardware Interfaces

No hardware interface is required beyond the user's device.

### 3.1.3 Software Interfaces

The website will involve using multiple external software systems to do its core functionality. It will need to be reliant on major web browsers such as Google Chrome, Microsoft Edge, Safari, and Firefox for today's modern standards. We will also need a database management system,(Not sure what we are gonna use), to retrieve user account data, ticket payments, and theater information. All of these connections will be encrypted through TLS. Payments will be handled through a third-party payment gateway API, such as PayPal, making sure that we do not store credit card information on the website. The system will also need an email or notification service to send receipts and ticket information. For user login, we will use traditional email and password registration. (maybe add google login?)

### 3.1.4 Communications Interfaces

The website will communicate using standard internet protocols to ensure secure and reliable interactions between users, the server, and external services. All client-to-server communication will use HTTPS for data confidentiality. TLS encryption will be applied to protect sensitive information like login info and payment information. The website will communicate with the payment gateway using secure API requests to authorize transactions.

## 3.2 Functional Requirements

### 3.2.1 User login or registration

#### 3.2.1.1 Introduction
Allows users and administrators to login or register an account, if not already logged in. It also supports resetting passwords.

#### 3.2.1.2 Inputs
Registration requires the user's full name, email, password, and optionally their phone number. Login requires the user's email and password. Resetting one's password requires their email and the email verification code sent to their email and a new password.

#### 3.2.1.3 Processing
Password validation checks whether the password contains at least one special character, no spaces, numbers, both lower and upper case letters, and a minimum of 8 characters. Email validation checks whether an email is already registered. All login and registration data is securely encrypted and stored.

#### 3.2.1.4 Outputs
Outputs include informing the user a new account has been created, an email already exists, an email doesn't exist, the password is incorrect, the password is invalid, the email is invalid, the user has made too many new attempts, a code has been sent to the user's email, the user has successfully logged in.

#### 3.2.1.5 Error Handling
All registration and login errors are outputted to the user. Backend errors such as being unable to fetch whether an email already exists is logged in a database visible only to admins.

### 3.2.2 <Functional Requirement or Feature #2>

#### 3.2.2.1 Introduction
This feature allows users to select a movie, choose their seats, and buy their tickets. Both guest users and logged-in users are supported.

#### 3.2.2.2 Inputs
The inputs include the user's chosen theater, movie name, showtime, amount of tickets, and seat selections. At the purchase stage, users must provide payment information. This is where users can also add optional discounts or promos.

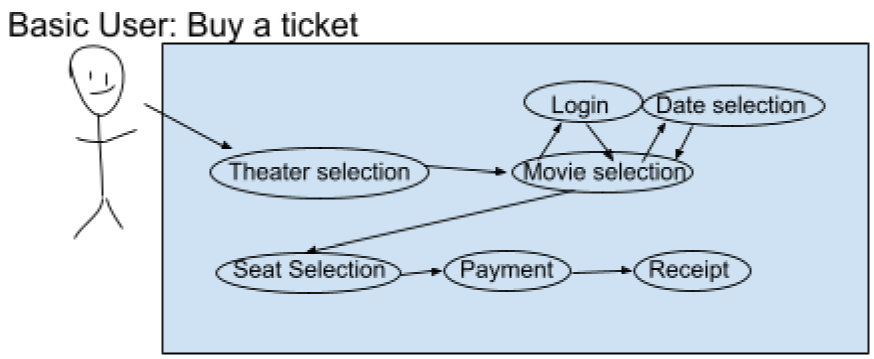
#### 3.2.2.3 Processing
The system checks seat availability in real time and prevents seats being selected twice. It calculates the total price with the applied discounts and taxes. We use the payment gateway API to process the transaction. If payment is successful the selected seats are reserved and stored in the database under the users account or guest registration.

#### 3.2.2.4 Outputs
The outputs include an on-screen confirmation showing the transaction ID, movie information, seat numbers, and the amount of money spent. A receipt along with the digital tickets are sent to the registered email address.

#### 3.2.2.5 Error Handling
If selected seats are unavailable at the time of checkout, the system prompts the user to select new seats. If payment fails due to invalid information, insufficient funds, or the card declines, the user will be notified and asked to retry or choose another payment option.

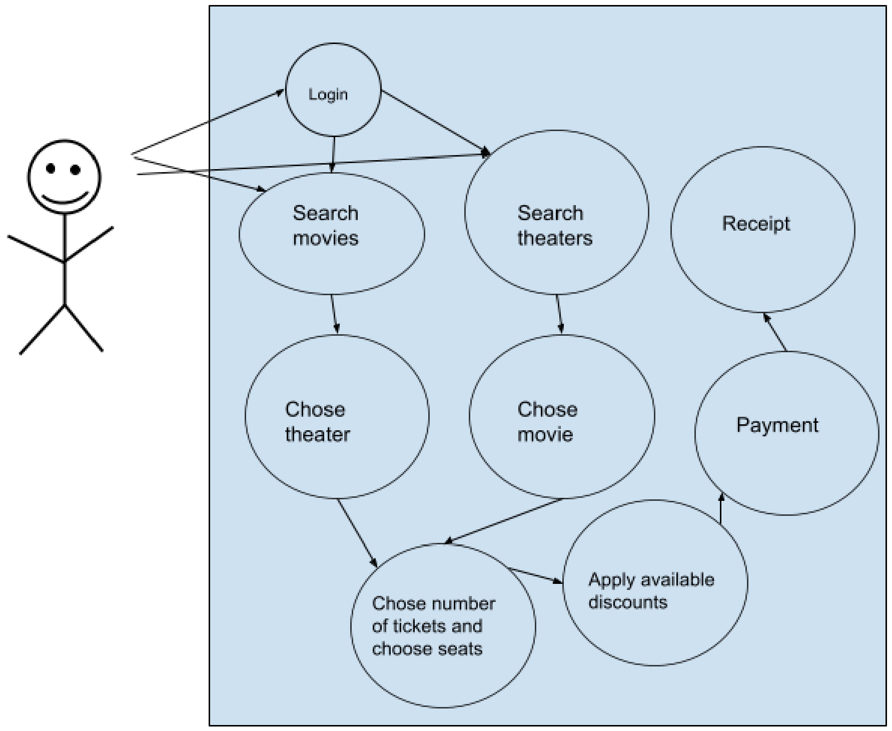
## 3.3 Use Cases

### 3.3.1 Use Case #1



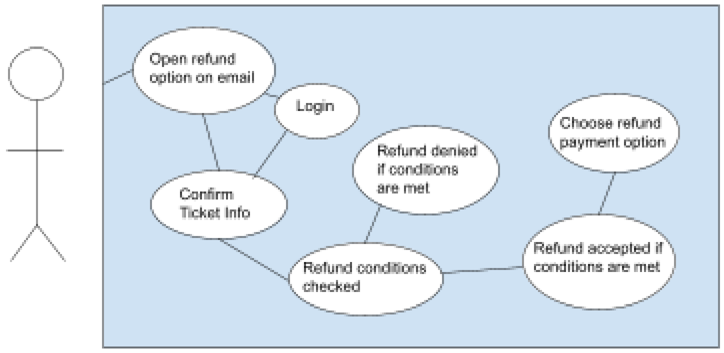
### 3.3.2 Use Case #2

Basic User: Multiple ways to chose



### 3.3.3 Use Case #3

Basic User: Request Refund



Open refund option on email
Login
Refund denied if conditions are met
Choose refund payment option
Confirm Ticket Info
Refund conditions checked
Refund accepted if conditions are met

## 3.4 Classes / Objects

### 3.4.1 <Class / Object #1>

3.4.1.1 Attributes

- userID (int)
- firstName(string)
- lastName(string)
- email (string, unique)
- passwordHash (string, hashed)
- phoneNumber (string, optional)
- role (enum: GUEST | CUSTOMER | ADMIN)
- createdAt, updatedAt (timestamp)
- lastloginAt (timestamp)
- isLocked (boolean, throttling/too many attempts)
- failedLoginAttempts (int)

3.4.1.2 Functions

- register (firstName, lastName, email, password, phoneNumber?) - userAccount
- authentication (email, password) - AuthResult
- requestPasswordReset (email) - void
- resetPassword(email, code, newPassword) - void
- updateProfile(fields)
- listPurchases() - Order

### 3.4.2 <Class / Object #2>

3.4.2.1

- orderID
- userID
- theaterID
- movieID
- showtimeID
- seats
- subtotal, discounts, tax, total
- transactionID

3.4.2.2

- reserveSeats (showtimeID, seats)
- applyDiscount (code)
- calculateTotal()
- checkout(paymentMethod)
- confirm()
- emailReceipt()

## 3.5 Non-Functional Requirements

### 3.5.1 Performance

The website will be capable of running on most devices but performance will vary depending on the device's hardware components. The server will need to process transactions in under 5 seconds and should not take more than 3 seconds to process any other requests.

### 3.5.2 Reliability

The website and database will be backed up at least once a day to save transaction logs and user information ex passwords so that in the event of unforeseen issues the website will minimize any data lost. The website will be tested before any major updates for reliability, any bugs that are identified will be fixed ASAP. Stress tests will be conducted to ensure the website remains stable in scenarios where there is a large fluctuation of users.

### 3.5.3 Availability

The website will be available to users 24/7 unless under scheduled maintenance which will warn users for at least 1 week before. The website will be available anywhere in the United States although most areas will not have supported theaters nearby.

### 3.5.4 Security

Security is a top concern and steps including encryption of all information will be taken to protect users information and privacy. The user's login information will be stored securely on a secure server. Any credit card information will be erased as soon as the transaction is completed and not stored anywhere in our website. The website will have network security protection against denial-of-service and other attacks.

### 3.5.5 Maintainability

We will continue to maintain the system for the foreseeable future. We plan to do regular updates for compatibility with browsers, bug fixes, and security updates.

### 3.5.6 Portability

The latest major version of the most popular desktop browsers and mobile browsers will be supported, such as Google Chrome for desktop, and Safari for mobile.

## 3.6 Inverse Requirements

*State any *useful* inverse requirements.*

## 3.7 Design Constraints

Constraints on the system design will be the requirements of the browsers we support, limitations of servers and databases that are processing and storing data about movie availability and purchases.

## 3.8 Logical Database Requirements

*Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.*

**Data Management Strategy**

**1) Overview & Goals**

Our data strategy prioritizes correctness, security of PII, and operational simplicity. We use a relational SQL database as the system of record for all transactional data, plus Redis for short-lived locks and hot reads. Payments are tokenized via the payment gateway, we do not store raw card data.

**Diagrams**

**2) Number of Databases & Logical Split**

- **One primary PostgreSQL database** for all OLTP (orders, seats, showtimes, users).
- **Optional read replica(s)** for admin dashboards/reporting to keep writes isolated.
- **Redis** (separate service) for ephemeral reservations and caching.
- **Object storage** (S3/Cloud Storage) for non-relational artifacts.
- **Optional analytics warehouse** (BigQuery/Snowflake) via nightly ETL for trend reports.

**Why this split?**

- Keeps **ACID guarantees** for seat allocation and orders.
- Offloads **hot transient state** to Redis without polluting the relational model.
- Protects OLTP performance by pushing heavy reads to **replicas/warehouse**.

**3) Logical Data Model (Core Entities)**

**Core tables**

- **user**: account & role (GUEST/CUSTOMER/ADMIN), password hash, login throttling flags.
- **theater, room, seat**: physical layout; **seat** has a natural unique key (room_id,row,col).
- **movie, showtime**: what/when/where.
- **reservation**: short-lived “locks” per seat+showtime while a purchase is in progress.
- **order, order_item**: authoritative purchase record and seat ownership.
- **payment, refund**: gateway outcomes (provider_charge_id / provider_refund_id).
- **promo**: discounts with validity windows and max usage.
- **audit_log**: immutable admin/user actions for forensics & compliance.

**Key constraints**

- Foreign keys across all relationships.
- UNIQUE(showtime_id, seat_id) in **order_item** to enforce one paid owner per seat.
- Optional EXCLUDE constraints to prevent time overlaps if ever needed.

## 4) Transactions & Concurrency (No Double Bookings)

- **Seat selection** creates a **Redis lock**: reservation:{showtime_id}:{seat_id} with TTL.
- Checkout runs a **single DB transaction**:
    1. Re-validate seat availability (SELECT … FOR UPDATE on the targeted seats).
    2. Insert order + order_item rows (unique constraint prevents duplicates).
    3. Record payment after gateway success; set order status PAID.
- If any step fails, the transaction rolls back; Redis locks expire automatically.

## 5) Data Retention, Archival & Purging

- **PII (user profile)**: retain until account deletion request; pseudonymize in analytics.
- **Orders & payments**: retain at least 7 years for tax/audit.
- **Reservations**: TTL 5–10 minutes in Redis; no long-term storage.
- **Audit logs**: retain 2 years online, then archive to cold storage.
- **Anonymization**: after chargeback windows, detach promo metadata from specific users while keeping aggregate reporting.

## 6) Security & Privacy

- **No card storage**: gateway tokenization only; we store provider_charge_id and amounts.
- **Transport**: HTTPS/TLS 1.2+ everywhere.
- **At rest**: DB disk encryption; server-side encryption for object storage.
- **RBAC**: separate DB roles for app, read-only analytics, and admin maintenance.
- **Row-level security (optional)** for multi-tenant or least-privilege admin tooling.
- **Secrets** in a **vault** (API keys, DB creds), never in code or environment files checked into VCS.
- **Hashing**: Argon2id for passwords.
- **Audit logging** for all admin actions (refunds, price changes).
- **Back-office access** gated behind SSO + MFA.

## 7) Backups, DR & High Availability

- **Point-in-time recovery**: daily full + WAL archiving; verify restores weekly.

- **RPO/RTO** targets: RPO ≤ 5 minutes, RTO ≤ 1 hour.
- **Multi-AZ** Postgres deployment, read replicas can be promoted on primary failure.
- **Redis**: managed service with replication + automatic failover.

## 8) Indexing & Performance

- B-tree composite indexes on frequent filters:
  - showtime(start_time), order(created_at), order_item(order_id).
  - reservation(showtime_id, seat_id), payment(order_id).
- Partial indexes (only reservation.status='LOCKED').
- **Partitioning** (native Postgres) by **date** for order, payment, audit_log to keep tables hot and VACUUM fast.
- **Caching**: seat maps and "what's playing today" in Redis with short TTL.

## 9) Alternatives Considered & Tradeoffs

- **MySQL**: similar maturity; Postgres wins on constraints, EXCLUDE, JSON, and partitioning UX.
- **MongoDB / NoSQL**: flexible docs, complex multi-doc consistency. We need strict ACID for seats.
- **DynamoDB**: massive scale and simple ops; modeling cross-entity joins is complex; transactions limited vs Postgres.
- **SQLite**: great for local/dev; lacks HA and concurrency for production.
- **Event-sourcing**: excellent auditability; higher engineering overhead and steeper learning curve for this course project.

## 10) Environments & Data Separation

- **Prod / Staging / Dev** databases with seed data.
- **Strict separation** of credentials and networks; no cross-environment data mixing.
- Redacted production snapshots may be used in staging after scrubbing PII.

## 11) Testing the Data Layer

- **Unit tests**: constraint checks (cannot insert duplicate seat ownership).
- **Property tests**: random seat reservations under load, ensure uniqueness and rollback safety.
- **Integration tests**: end-to-end checkout including payment sandbox & email receipt.
- **Failure drills**: kill primary DB to validate failover/RTO; expire Redis locks to ensure graceful seat release.

## 12) Operational Monitoringwww2

- **Metrics**: lock TTL expiries, failed payments, checkout latency, DB lock wait times.
- **Logs**: structured app logs and DB audit logs shipped to SIEM with alerts.
- **Dashboards**: seats sold per showtime, refunds per day, promo redemption r

### 3.9 Other Requirements

*Catchall section for any additional requirements.*

## 4. Analysis Models

*List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable the SRS's requirements.*

### 4.1 Sequence Diagrams

### 4.3 Data Flow Diagrams (DFD)

### 4.2 State-Transition Diagrams (STD)

## 5. Change Management Process

*Identify and describe the process that will be used to update the SRS, as needed, when project scope or requirements change. Who can submit changes and by what means, and how will these changes be approved.*

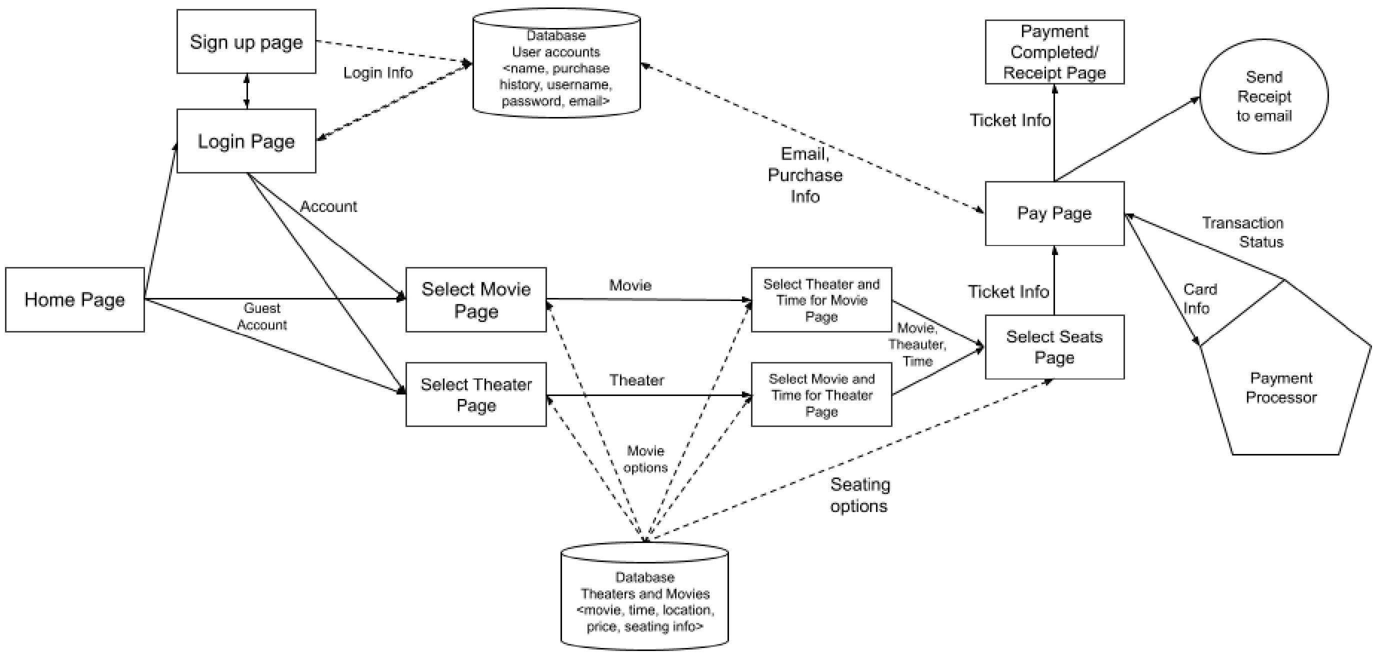
## 6. Software Design Specification

### 6.1 Overview of system

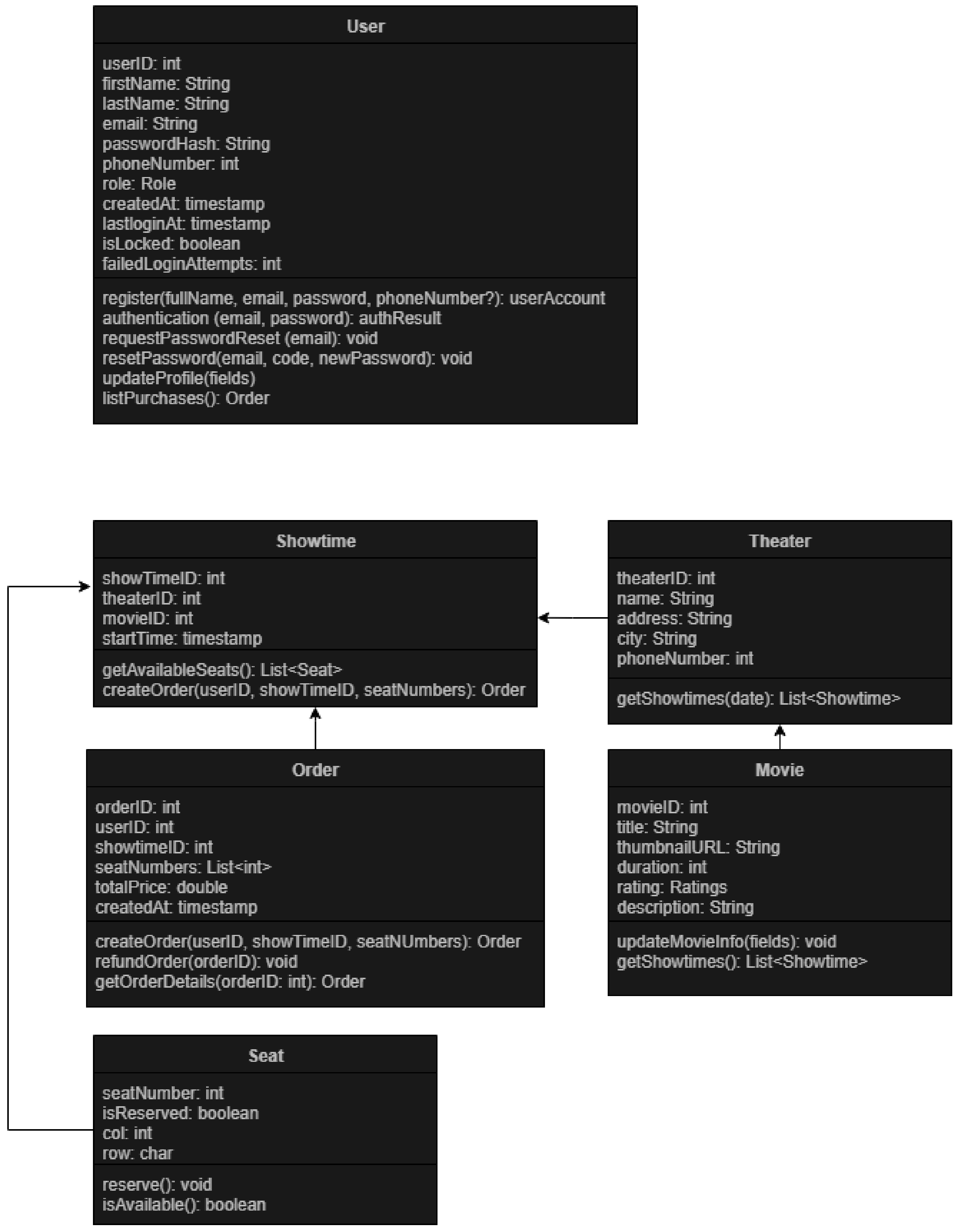
The Movie Ticketing System will allow users to purchase movie tickets from available nearby theaters through the website. The system will include optional account login, currently available movies, seat selection based on availability, secure payment with credit card, and sending an email with a receipt.

Users will interact with pages of the website to choose and pay for movie tickets. The system will communicate with servers for information regarding available movies and information for the user's account. Payment history will be saved with the user's account in the event of later issues regarding the purchase (no credit card information will be saved).

## 6.2 Architectural Diagram of Major Components (UPDATE)



## 6.3 UML Class Diagram



## 6.4 Description of Classes, Attributes, and Operations

### 6.4.1 Class 1: User

Description: Represents a user which is a guest by default and a customer or administrator if logged in. It handles account data and login-related operations.
Attributes:

- userID: int - Identify the account, never changes.
- firstName: String
- lastName: String
- email: String - unique, used for login and code verification.
- passwordHash: String
- phoneNumber: int
- role: Role - enum (GUEST | CUSTOMER | ADMIN) to grant privileges.
- createdAT: timestamp - time account was created.
- lastLoginAT: timestamp
- isLocked: boolean - Whether the account is locked due to too many failed login attempts.
- failedLoginAttempts: int - failed login counter

Operations:

- register(firstName: String, lastName: String, email: String, password: String, phoneNumber?: String): User - creates a new User
- authenticate(email: String, password: String) : AuthResult
- requestPasswordReset (email): void - Sends a code to the user’s email.
- resetPassword(email, code, newPassword): void - Changes the user’s password.
- updateProfile(fields) - Changes the user’s information such as their name, email, or phone number.
- listPurchases(): Order- Returns a list of the user’s ticket order history.

### 6.4.2 Class 2: Movie

Description: Stores movie data and connects to Showtimes.
Attributes:

- movieID: int
- title: String
- thumbnailURL: String - link to the poster image.
- duration:int - duration of the movie in seconds.
- rating: Ratings - enum (G, PG,  PG-13, R, NC-17)
- description: String

Operations:

- updateMovieInfo(fields): void - Update the duration, rating, description, title, or thumbnailURL for the movie.
- getShowtimes(): List<Showtime>

### 6.4.3 Class 3: Showtime

Description: The ShowTime is a showing of a movie at a theater and contains the seat data.

Attributes:
- showTimeID: int
- theaterID: int
- movieID: int
- roomNumber: int
- startTime: timestamp - The time the movie starts.

Operations:
- getAvailableSeats(): List<Seat>

### 6.4.4 Class 4: Theater
Description: Represents a physical theater location.
Attributes:
- theaterID: int
- name: String
- address: String
- city: String
- phoneNumber: int

Operations:
- getShowtimes(date): List<Showtime>

### 6.4.5 Class 5: Seat
Description: Represents a seat in a theater room.
Attributes:
- seatNumber: int
- isReserved: boolean
- col: int
- row: char

Operations:
- reserve(): void
- isAvailable(): boolean - returns whether the seat is available or reserved.

### 6.4.6 Class 6: Order
Description: Represents a ticket purchase.
Attributes:
- orderID: int
- userID: int - The userID of the customer that made the purchase.
- showtimeID: int - The showtimeID of the movie the customer purchased the ticket for.
- seatNumbers: List<int> - The seats the customer selected when purchasing the ticket.
- totalPrice: double - The total amount the customer was charged.
- createdAt: timestamp - The time the order was created.

Operations:

- createOrder(userID, showTimeID, seatNumbers): Order - Creates an order.
- refundOrder(orderID): void - Cancel and refunds the order.
- getOrderDetails(orderID): Order - Returns specific order details to an administrator.

### 6.5 Development Plan and Timeline

#### 6.5.1 Partitioning of Tasks

All tasks will be divided into subsections and team members are able to choose which incomplete subsections they would like to complete. Team members are responsible for completing about ⅓ of the subsections for every task.

#### 6.5.2 Timeline

10/23: Test Plan
11/6: Architecture w/Data Mgmt.
11/20: Ethics and Communications
12/4: Final Project Report Draft
12/11: Final Project Report

# 7. Test Plan

### 7.1 Test 1: CreateAccount_1

This tests the process of creating an account. This is based on the User class. The test verifies that the system accepts valid usernames and passwords and successfully creates user accounts. This depends on the system properly storing and verifying the information.

### 7.1 Test 2: CreateOrder_2

This test verifies that the system correctly creates a new order when valid inputs are provided and properly handles invalid selections by displaying errors and not completing an order. The Order module is tested by initializing the order creation function with valid inputs such as the user's ID, showtime ID, and available seats. The test is repeated using invalid inputs. The system should create a new order for the valid inputs and display an error message for the invalid inputs, not creating a new order.

### 7.1 Test 3: SystemTest_3

This test validates the functionality of the entire system from login to checkout to ensure proper integration across all modules. This entire process is tested by first signing in with a valid user account, selecting a theater, movie, and seats, then completing the payment and viewing the order history. Each step is reviewed to ensure proper and expected results occur, valid inputs result in valid outcomes, and invalid inputs result in error messages.

### 7.2 Test 4: LoginAuth_4

This test verifies that only valid users can access the system using the correct email and password combination, and that all invalid login attempts are rejected with appropriate error messages. This test also covers systems responses to multiple failed login attempts.

### 7.3 Test 5: SeatSelect_5

This test confirms that users can select available seats for a chosen movie and showtime while preventing double-booking from multiple devices or sessions. The seat selection module is tested by having one user reserve seats while another user attempts to select the same ones simultaneously. The system should show the seats are “Reserved” to the second user.

### 7.4 Test 6: Payment_6

This test validates that the Payment Gateway Module correctly processes transactions using valid credit card information while handling invalid, expired or declined cards. The system must successfully complete payment transactions using valid cards and send an email receipt to the users. Invalid transactions should display an error message without charging the customer.

### 7.5 Test 7: Refund_7

This test confirms that only administrators can issue refunds and that each refund correctly reverses the order and updates the system. This is tested by logging in as an admin, selecting an order, and performing a refund action.

### 7.8 Test 8 PasswordReset_8

This test verifies that users can securely reset their passwords through an email verification process. The User Account Module is tested by initiating a password reset request using a valid registered email, receiving a verification code, and creating a new password. The system must accept the new password, allow login using the new credentials, and reject attempts with the old password.

### 7.9 Test 9: PromoCode_9

This test verifies that valid promo codes correctly apply discounts correctly and invalid codes display an error and don’t provide a discount. The Payment Gateway module is tested by entering both valid and invalid promo codes on the payment page. The total cost should update for valid promo codes and remain the same for invalid promo codes.

### 7.10 Test 10: UserOrderIntegration_10

This test validates integration between the User and Payment modules by confirming that the completed purchases appear correctly in the user’s order history. This test is performed by logging in as a user, selecting a theater, movie, and seats, then making a purchase and viewing the purchase history. The order should accurately reflect the purchase details such as the theater, movie, seats, and act as a receipt.

## 8. Data Management Strategy

### 8.1 Overview

The system needs to manage important data such as user accounts, booked seats, and process transactions. The data needs to be stored properly to guarantee security and reliability.

We will use a SQL design for the databases with 2 databases.

1 User Information Database: Stores user information such as username, password, email, name, and purchase history.

2 Showings Database: Stores information about showings such as movie title, time, theater, auditorium number, theater location, seat availability, and seat price.

Sensitive information especially in the user information database will be encrypted and stored with extra safety precautions. Only authorized individuals and the backend of the system will be allowed access to ensure privacy and security.

### 8.2 Database Design

### 8.3 Rationale and Tradeoffs

## A. Appendices

*Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements.*

*Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.*

### A.1 Appendix 1

### A.2 Appendix 2