Movie Ticket System

Software Requirements Specification

Version 3.0

3/20/2025

Group 3
Isabella King, Vlad Jerohhin, and Kyle Baird

Prepared for
CS 250- Introduction to Software Systems
Instructor: Dr. Gus Hanna
Spring 2025

Revision History

Date	Description	Author	Comments
2/20/25	Version 1	Group 3	SRS 1.0 ~ until 3.6
3/6/25	Version 2	Group 3	SRS 4.0 ~ until 4.4
3/20/25	Version 3	Group 3	SRS 5.0 ~ until 5.3
4/10/25	Version 4	Group 3	SRS 6.0 ~ until 6.3

Document Approval

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

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

Table of Contents

1. Introduction

The Movie Ticket System is a web-based platform that simplifies purchasing movie tickets, enhancing our users' experience. This system allows users to browse theaters, search/filter movies, reserve specific seats, and earn discounts by opting to create an account. Special features such as a rewards program and online snack ordering aim to make for a unique and enjoyable ticketing experience. This Software Requirements Specification (SRS) aims to outline the requirements and process of creating our movie ticketing system. It will specify the ticketing system features, restrictions, and production. The SRS will clarify how the user will interact with our product.

1.1 Purpose

The intended audience is the client/stakeholder purchasing and using our ticketing system. It will give them a vision for the product and confidence in their purchase. Furthermore, it will outline their expectations for the product. Also, this document is meant for our software engineering team. It will guide them through production and give them the layout for what they must produce. It will keep everyone working towards the same goal by providing a structured vision for the product requirements and features. Designers and testers will be able to do a better job as they will understand the big picture.

1.2 Scope

Our movie ticketing system, Ticketing System, will benefit movie theater companies by enabling them to sell tickets more efficiently, improving ticket accessibility, and overall improving customer experience. Our product will accomplish this by creating a system that allows the customer to pick from dozens of movies at multiple locations at the touch of a button. Users can access our system on any device through a browser, but not at a kiosk within the physical movie theater location or through an app. Users will have access to discounts and a special rewards program. Through our system, users can pick a movie theater location within the movie theater company and browse available films within that branch. They can reserve a specific seat, at a certain time, in either a regular or deluxe room within the theater.

Our ticketing system will allow users to sort theaters by rating, prices, and distance on an interactive map to make it easier to select a theater based on what they prioritize. We will offer discounts for members (customers who sign up and visit the theater often), military personnel, students, and teachers. Users also have the option to purchase snacks directly on the site so they can walk into the theater and be seated within a few minutes. The goal is to make a streamlined system for people to purchase tickets quickly and easily online on their computer or mobile device.

Our ticketing system follows the rules and requirements that are set in the System Requirement Specification document from the movie theatre company: "Theatre Architecture Guidelines". That document explains how our ticket system fits in and works together with the other systems used by the Theatre Company. The features included with this system that fit within the larger system are: security, tracking/managing tickets, and working with other parts of the system. This allows for on-site security to have access to our database of all customers'

information in the event of a crime. Also, our database is synced with all of the theatres' databases so any transactions, account creations, seat selections, etc. made in person will be updated with our online system.

1.3 Definitions, Acronyms, and Abbreviations

BII	Basic Identifiable Information
P2P	Peer-to-peer
UX	User Experience
Amazon RDS	Amazon Relational Database Service
API	Application Programming Interface
CCPA	California Consumer Privacy Act

1.4 References

Documents referenced:

Burak, A., & Burak, A. (2025, January 30). Your 2025 guide to writing a Software Requirements Specification – SRS document. *Relevant Software*. https://relevant.software/blog/software-requirements-specification-srs-document/

IEEE Computer Society & Software Engineering Standards Committee. (1998). *IEEE recommended practice for software Requirements Specifications*. IEEE Computer Society.

McClure, R. M. (1969). *NATO SOFTWARE ENGINEERING CONFERENCE 1968* (P. Naur & B. Randell, Eds.).

1.5 Overview

The rest of the SRS will contain a general description of the product: the functions, perspective, characteristics, and constraints. In addition, the requirements provided by the client and our requirements for creating the system will be included. Next, it contains analysis models, the change management process, and an appendix.

The SRS is divided into 5 main sections each of which have subsections. The first section is an introduction/overview of the product: the purpose, scope, definitions, references, and overview can be found. The second section is a more in-depth description of the product including the perspective, functions, characteristics, constraints, and dependencies. Section 3 covers the necessary components needed to create the product: all of the requirements along with the use cases, classes, objects, and constraints can be found there. The fourth section has any analysis models used and the fifth section includes the appendices.

2. General Description

This section provides an overview of the Movie Ticketing System, outlining its perspective, functions, intended users, constraints, and dependencies. Everything outlined sets the foundation for understanding the requirements outlined in later sections.

2.1 Product Perspective

Our ticketing system is independent and not connected to any larger system. It manages its clientele's information through its database which is not conjoined with another company. Our ticketing system's user interface is outlined in the "Theatre Brand Guidelines" and the "Theatre Website Wireframes". The "Theatre Brand Guidelines" provide a reference for how the ticketing system front end should be designed and implemented. It contains the company's logos, color scheme, and typography. The "Theatre Website Wireframes" should be utilized to understand the basic layout of the ticketing system and how the user can interact with it. It solely represents the functions in the ticketing system and not a finished depiction of the visuals. Our ticketing system can be accessed on all devices through a web browser and doesn't require specific hardware. Our ticketing system will allow access to our third-party payment applications, common credit card networks, our database on Amazon RDS, and third-party movie review systems like Rotten Tomatoes. Our movie ticketing system will have a communication interface as it will email customers their receipts and updates on their movie tickets. There are no memory constraints for the user as our ticketing system will run most of the computation on the backend through our servers. Our system will provide backup and recovery functionality through Amazon RDS.

2.2 Product Functions

Overall, the ticketing system will be a website that allows users to purchase tickets from all the locations owned by a specific movie theater company. When the user opens the website they will be presented with a choice. They can continue as a "guest" without access to our reward system and discount options, but they can purchase tickets without losing time making an account. Also, guests will be asked to submit a name and email address so that a receipt and updates on their order can be sent to them. As a guest, they will not be sent spam or advertisements from the company through their email.

The customer can also decide to make an account. They will submit their basic identifiable information (BII) and a password allowing them to access our reward system, discounts, and purchase snacks online through their device. We will keep their information secure through our database and password system. We will not offer a multi-authentication for these accounts.

After choosing to create an account or continue as a guest, the customer can browse movies. Our ticketing system will provide a filtering system where the customer can filter the movies by location, price, time, genre, and location. Once the user selects a movie, our system will recommend locations and alternate times. They can choose between a regular ticket or a deluxe ticket.

Once they decide on a film, location, time, and ticket type, they can check out. It will prompt them to choose a seat in the theater room signifying which seats are available with a color scheme. The system will hold their seat selection for a 10-minute interval while they make their purchase. We will provide a secure system that keeps their sensitive information safe in our database. Our ticketing system will allow users to purchase tickets using a debit card, credit card, or popular peer-to-peer (P2P) payment application like Venmo. They will receive a confirmation email, after checking out within the allotted time frame.

If they opt to create an account, customers will gain access to our rewards system, special discounts, and online snack service. As customers make purchases through our online system, they will receive points that can be redeemed. Some rewards include a free ticket, free snacks, a free upgrade to a deluxe ticket, and more. Also, they can access deals for cheaper tickets on certain days and large group discounts. On the other hand, they can access our discounts for military, students, teachers, and healthcare professionals. They will need to verify their identity through our secure system and then they can access 10% of tickets and 15% of food.

In addition to rewards and discounts, customers with accounts can purchase snacks online and pick them up before their movie starts. They can browse snack options and add them to their cart. Also, they can save their favorite snack options to quickly purchase their favorite snack combos more efficiently. The checkout process is the same for snacks and tickets.

2.3 User Characteristics

The prospective users of our movie ticketing system are people in the middle class who can afford movie tickets and don't have a better movie system at home. Anyone aged 16 and up can use it, but we predict that people under 45 will use it. Considering that multiple online market websites and applications are available, we assume that our consumer already knows how to utilize the system. Our development team will need to focus on a proficient UX design, not teaching our clients how to use technology. They can refer to our “Theatre Brand Guidelines” and “Theatre Website Wireframes” references. Our system design should accommodate the average user by making each button a common icon. For instance, our consumers should know how to check out but not have to search for the checkout button. The checkout button should resemble a shopping cart in the top menu bar.

2.4 General Constraints

Our system will enforce regulatory policies that will ensure consistent maintenance on our website every month where our maintenance team will fix bugs and add new features. Also, we guarantee reliability by ensuring that the reaction time of our website is 2 seconds, meaning that after the customer clicks on a button it must react immediately. We can do this by enabling parallel query execution so when customers purchase something on our website it will quickly

respond. Furthermore, our development team will be required to implement a database stored on the cloud through Amazon RDS due to its increased security and reliability. To create and manage the database we will use MySQL. By utilizing common e-commerce creation systems, we can quickly develop our system by hiring experienced developers and reviewing thorough online documentation. To ensure the security of our database, we will enforce strict policies on who can access our API and how much data they can manipulate or extract. Additionally, we will follow the California Consumer Privacy Act (CCPA) to protect the rights of our customers. They have the right to know how we utilize their information, delete it from our database, and limit the amount of information we collect.

2.5 Assumptions and Dependencies

The Movie Ticketing System relies on many assumptions and dependencies that impact its functionality. To start, it assumes that the user will have a stable and decent internet connection, as slow or unreliable internet may cause issues using the platform. The system can also be accessed with most modern web browsers such as Google Chrome, Firefox, Microsoft Edge, Safari, etc. However, outdated or less common browsers could cause issues. Also, the database is built on Amazon RDS which could affect our system if pricing changes, policy changes, or outages occur. Another dependency the system has is third-party payment networks including Visa, MasterCard, American Express, Venmo, and PayPal. Any API changes, outages, or policy changes could affect our system. Compliance with legal and regulatory requirements such as the California Consumer Privacy Act (CCPA) is another important assumption that could require changes to how user data is collected and stored if new laws are passed. Our website's front end will rely on three key languages, Javascript, CSS, and HTML. We will use these to create and maintain the website. The back end will be built on PHP for database connection and SQL to manage our database through Amazon RDS. Our system will use React, an open-source JavaScript library to connect files more efficiently. As these open-source languages update and change, they might not support later versions of the language. This dependency will cost money as we might have to update our code sooner rather than later.

3. Specific Requirements

This section details the software's specific requirements, including functional, non-functional, use cases, external interfaces, classes, objects, and inverse requirements. All requirements in this section are essential to the implementation of the Movie Ticketing System.

3.1 External Interface Requirements

3.1.1 User Interfaces

- The system will have a web-based interface that can be accessed on mobile and desktop devices. We will not create an app or kiosk system.
- The home/main page will have a navigation menu with options to browse movies, view rewards, see user accounts, and view cart/checkout.
- On each page, there will be a consistent color scheme, branding, and interactive elements including filters, buttons, and dropdowns. Please refer to our "Theatre Brand Guidelines" for

a representation of our system's visual style. Then, refer to our "Theatre Website Wireframes" for an interactive model for our website's basic layout.
- The seating map page will use a color-coded system to show the status of seats: yellow for pending orders, red for taken seats, and green for available seats.
- The checkout page will allow users to input their payment details and confirm within a 10-minute time limit.
- The interface will have a feedback system, where users can give movies a thumbs-up or thumbs-down along with optional written feedback. We will encourage user interaction by sending them an email reminder.

3.1.2 Hardware Interfaces

- The system can be accessed with any hardware capable of web browsing such as desktop computers, laptops, tablets, and smartphones.
- There are no specific hardware requirements other than an internet connection. Personal devices are common, so we don't have to plan to accommodate for those without devices. These outliers will still be able to purchase tickets in person.

3.1.3 Software Interfaces

- The system will use third-party payment APIs such as Venmo and PayPal as well as major credit/debit card networks such as Visa, Mastercard, and American Express.
- Amazon RDS will securely store customer data, movie details, and transaction information.
- The movie database will be synced to the system to show movie listings, times, and availability in real time.
- Rotten Tomatoes or other third-party review systems will have their APIs linked to our system.
- Our ticketing system will connect with the in-person food court area within the movie theatre and give live updates to orders. They will send them an email receipt to document online purchases.

3.1.4 Communications Interfaces

- The system will send email notifications for order confirmations, promotion of upcoming movies, and any changes to seat availability. They will also receive an email, reminding them to fill out our feedback form.

3.2 Functional Requirements

3.2.1 Account Creation

3.2.1.1 Introduction
- The system will allow users to create an account or continue as a guest.
- Users must provide basic identifiable information (BII) such as name, email address, phone number, and password to register.

3.2.1.2 Inputs
- The user will provide BBI like name, phone number, email, and password upon creating an account. If the user decides to continue as a guest, they will be required to input their name and email address for the sole purpose of a ticket confirmation email with their receipt.

3.2.1.3 Processing

- Validate user inputs by checking if the email is valid and if the password meets requirements.
- Store user data in the Amazon RDS database.

3.2.1.4 Outputs
- Confirmation messages and emails will be sent at account creation.

3.2.1.5 Error Handling
- If a user inputs invalid information, the system will display an appropriate error message.

3.2.2 Movie Ticket Purchase

3.2.2.1 Introduction
- Users will be able to browse/search movies, filter/sort them, select showtimes, choose seats, and purchase tickets.

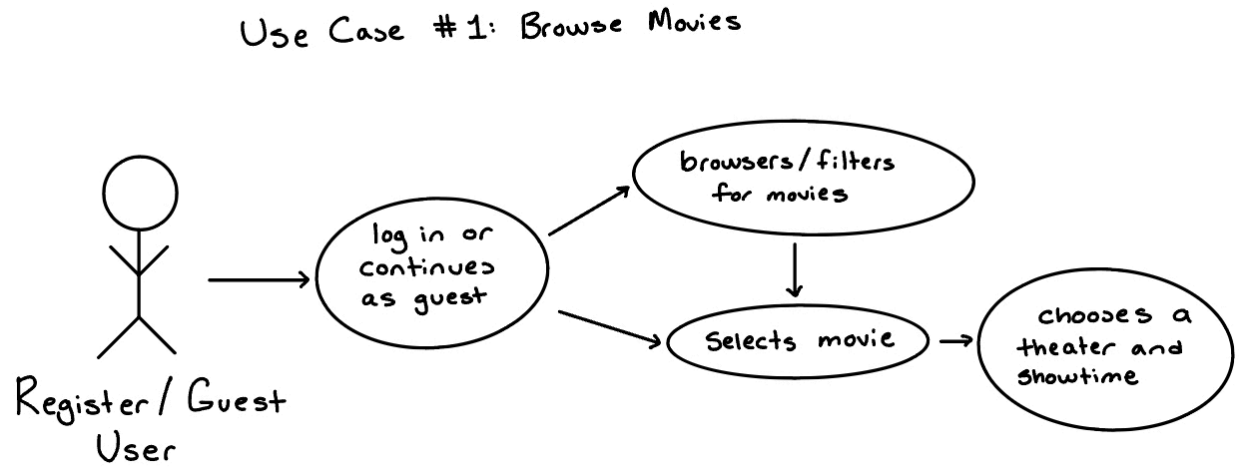
3.2.2.2 Inputs
- The user selects a movie, location, time, and seat(s).
- User inputs payment details: credit, debit, or other P2P options.

3.2.2.3 Processing
- The system will keep seat availability up to date in real-time, reserve the selection seat(s) for 10 minutes, and process payment.
- The system will update the seating map and confirm availability.

3.2.2.4 Outputs
- A payment confirmation message with ticket details.
- Email with ticket confirmation and seat information.

3.2.2.5 Error Handling
- If payment fails, the user will be notified and given an option to retry.
- If the user exceeds the 10-minute window for checkout, the seat will be released and the user will be taken back to the seat selection page.

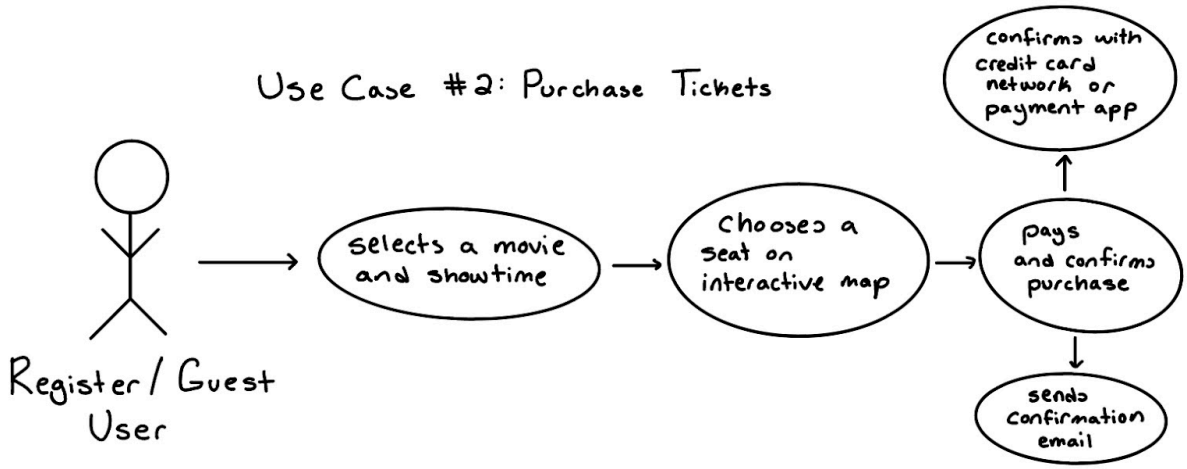
3.3 Use Cases



3.3.1 Use Case #1: Browse Movies

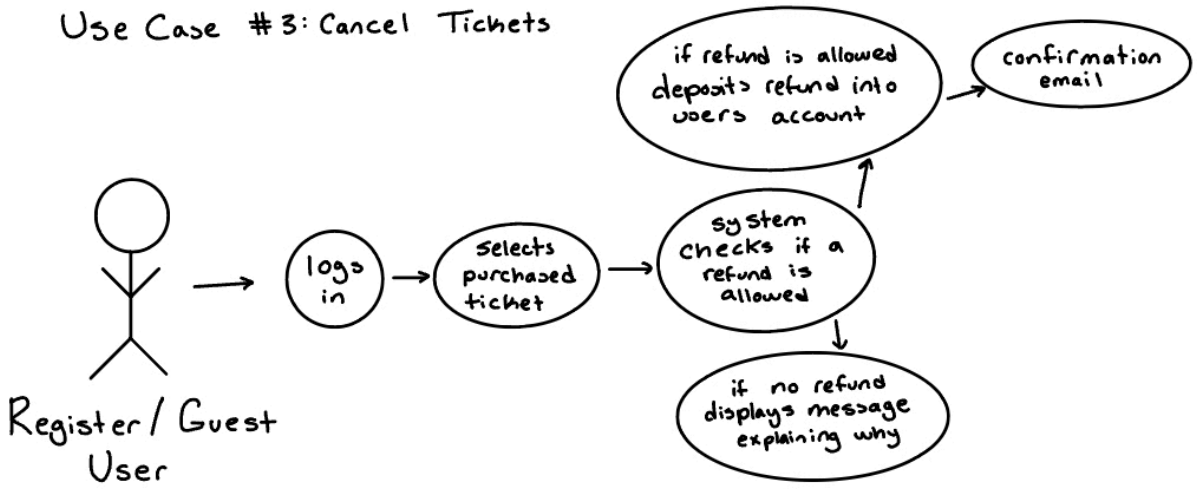
- Actor: Registered or Guest User

- Description: A user selects their preferred movie, views available showtimes, and chooses the desired theater location.
- The flow of events:
  1. The user logs in or continues as a guest.
  2. The user searches/browses for movies and filters movies by genre, location, and time.
  3. The user selects a movie and is presented with available theaters and showtimes.



3.3.2 Use Case #2: Purchase Ticket

- Actor: Registered or Guest User
- Description: The user selects a movie and seat(s) and then proceeds with the payment.
- The flow of events:
  1. The user selects a movie and showtime.
  2. The user chooses a seat on the interactive seating map.
  3. The user enters payment information and confirms the purchase.
  4. Confirmation email and ticket are sent to the user.



3.3.2 Use Case #3: Cancel Ticket

- Actor: User who has purchased ticket(s)
- Description: The user cancels a purchased ticket before the showtime and receives a refund depending on the cancellation time.
- The flow of events:
  1. The user logs into their account.
  2. The user navigates to the “My Tickets” page and selects the ticket(s) they wish to cancel.
  3. The system checks if the cancellation is allowed and prompts the user to confirm.
  4. Refund is given according to how far in advance the cancellation occurs. (Full refund if 24+ hours, partial refund if within 24 hours)
  5. The user receives a confirmation email with details and the seat is released for others to book.

3.4 Classes / Objects

3.4.1 User Class

3.4.1.1 Attributes

- **userID**: Unique identifier for the user.
- **name**: Full name of a user who signs up to become a member.
- **email**: A member's preferred email address, used for both login and general emails.
- **password:** Encrypted password used to authenticate a user.
- **userType:** Defines whether the user is a (temp) guest with no login, a member, or a premium member.
- **savedPayments:** List of the user's saved payment methods for in-app purchases.
- **purchaseHistory**: List of past movie ticket purchases as well as past food orders

3.4.1.2 Functions

- **register():** Allows the user to create an account and become a member.
- **login():** Authenticate a user and allow them to log in to an existing account.
- **savedPaymentMethods():** Allows the user to see, add, modify, and remove payment methods from their account.

- **viewHistory():** Allows the user to view their past purchases including both tickets as well as food items
- **updateZipcode():** Let the user change their home address zip code so that movie theaters can be recommended/sorted by distance.
- Reference to functional requirements and/or use cases
- User authentication relates to login() and register().
- Managing their profile related to savedPaymentMethods() and updateZipcode().

3.4.2 Movies class

Attributes:

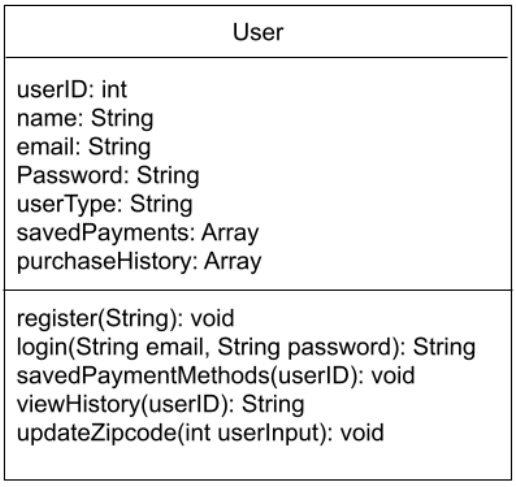
- **movieID:** Unique ID for every movie.
- **title:** Movie title.
- **genre:** Movie genre. (Horror, comedy, action, etc)
- **duration:** Movie runtime.
- **rating:** Movie ratings from IMDB.
- **ageRating**: MPAA movie age restriction (PG, PG-13, R)
- **showtimes:** List of available showtimes for the chosen movie

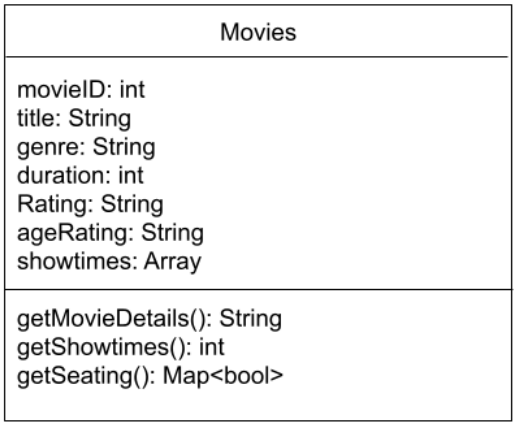
Functions

- **getMovieDetails()**: Displays the title and a short description of the movie as well as the online trailer for it.
- **getShowtimes():** Shows the dates the movie is showing as well as the times available on those dates.
- **getSeating():** Displays a visual theater layout with available seating to choose from.

Reference to functional requirements and/or use cases

- Browsing Movies references getMovieDetails()
- Selecting showtimes relates to getShowtimes() and getSeating()





3.5 Non-Functional Requirements

3.5.1 Performance

- The system will process 95% of transactions in less than 2 seconds
- Movie search and showtime retrieval will not be greater than 1 second for 90% of searches
- Payment processing will be completed in 5 seconds or less

3.5.2 Reliability

- Uptime is expected 99.98% of the time
- Any system failure will recover within 5 minutes and attempt to save any temp data like a user selecting seating

3.5.3 Availability

- Available 24/7.
- Schedules downtime will be announced at least 48 hours in advance
- No more than 1 planned downtime a month
- Downtime should not exceed 30 min

3.5.4 Security

- User passwords will be encrypted and stored securely using industry-standard encryption (AES-256)
- The system will support 2FA and use CVC verification or phone unlock security such as faceID or fingerprint reader when paying.
- Payment transaction and card details will also use industry-standard security and encryption (AES-256)

3.5.5 Maintainability

- Updates will have minimal interference with users during updates
- The system will be documented well for easy maintainability

3.5.6 Portability

- Available on all web browsers
- Our system will work seamlessly with nondesktop devices and have an optimized UI for mobile

3.6 Inverse Requirements

- The system will not store unencrypted credit cards or store cards automatically without user permission.
- The system will not allow ticket refunds/cancellations *online* if the movie starts in less than 2 hours and no refunds after the movie has passed.
- The system will not allow payments without fully verifying the transaction.
- Unregistered users will not be able to earn points/rewards

4. Software Design Specification

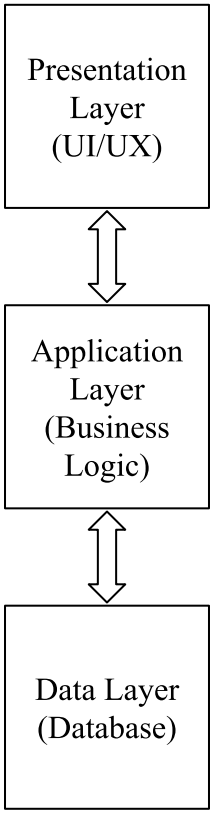
This section describes the software architecture, design, and development plan for the Movie Ticket System. It is intended for developers who will implement and maintain the system.

4.1 Software Architecture Overview

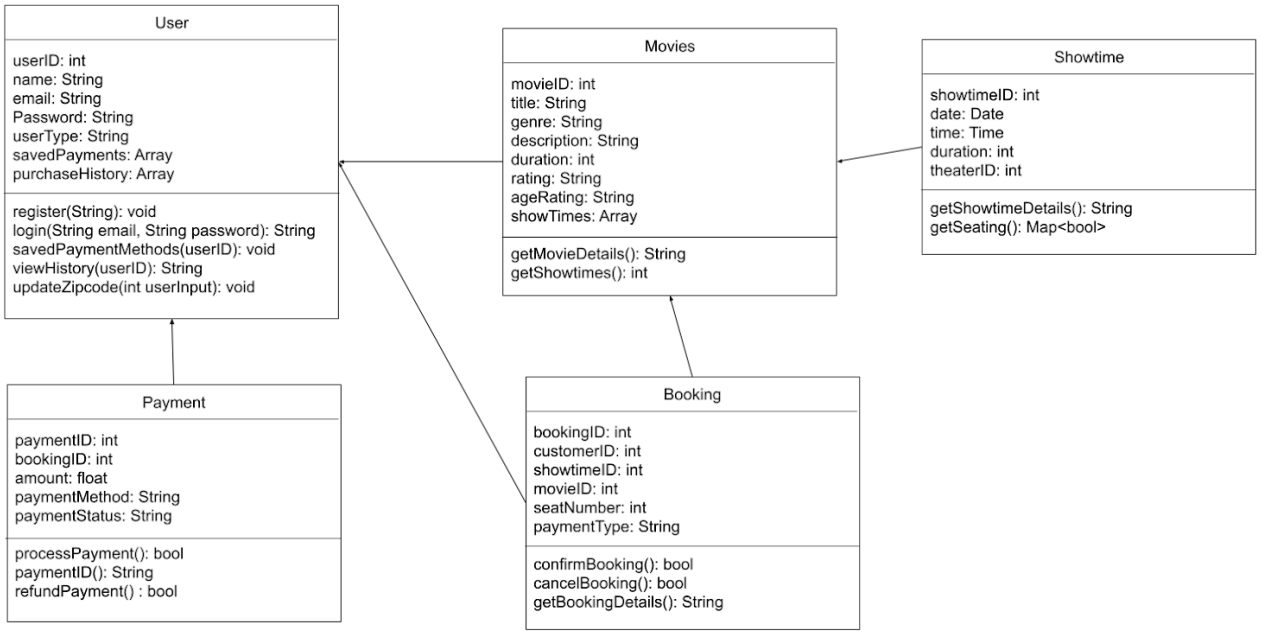
The Movie Ticket System follows a three-layer architecture comprising of the Presentation Layer, Application Layer, and Data Layer.

- **Presentation Layer:** This layer handles the user interface and user interactions. It includes the web interface which can be accessed on desktop and mobile devices.
- **Application Layer:** This layer contains the business logic and processes user requests. It handles functionalities such as account creation, ticket booking, payment processing, and rewards.
- **Data Layer:** This layer manages data storage and retrieval. It uses Amazon RDS to store user data, movie details, transaction records, and seating information.

Diagram:



4.2 UML Class Diagram



4.3 UML Class Description

User Class

- **userID**: Unique identifier for the user.
- **name**: Full name of a user who signs up to become a member.
- **email**: A member's preferred email address, used for both login and general emails.
- **password:** Encrypted password used to authenticate a user.
- **userType:** Defines whether the user is a (temp) guest with no login, a member, or a premium member.
- **savedPayments:** List of the user's saved payment methods for in-app purchases.
- **purchaseHistory**: List of past movie ticket purchases as well as past food orders

User Class Functions

- **register():** Allows the user to create an account and become a member.
- **login():** Authenticate a user and allow them to log in to an existing account.
- **savedPaymentMethods():** Allows the user to see, add, modify, and remove payment methods from their account.
- **viewHistory():** Allows the user to view their past purchases including both tickets as well as food items
- **updateZipcode():** Let the user change their home address zip code so that movie theaters can be recommended/sorted by distance.

Movies class

- **movieID:** Unique ID for every movie.
- **title:** Movie title.
- **genre:** Movie genre. (Horror, comedy, action, etc)
- **duration:** Movie runtime.

- **rating:** Movie ratings from IMDB.
- **ageRating**: MPAA movie age restriction (PG, PG-13, R)
- **showtimes:** List of available showtimes for the chosen movie

Movies Class Functions

- **getMovieDetails()**: Displays the title and a short description of the movie as well as the online trailer for it.
- **getShowtimes():** Shows the dates the movie is showing as well as the times available on those dates.

Showtime Class

- **showtimeID**: Unique identifier for each showtime.
- **date**: The date when the movie is scheduled.
- **time**: The time at which the movie will start.
- **duration**: The total duration of the show.
- **theaterID**: The identifier of the theater where the show is playing.

Showtime Class Functions

- **getShowtimeDetails()**: Returns detailed information about the showtime.
- **getSeating()**: Returns the seating availability for the showtime as a map of seats.

Booking Class

- **bookingID**: Unique identifier for each booking.
- **customerID**: The user ID of the customer making the booking.
- **showtimeID**: The ID of the showtime being booked.
- **movieID**: The ID of the movie for which the booking is being made.
- **seatNumber**: The assigned seat number for the booking.
- **paymentType**: The type of payment method used for the booking.

Booking Class Function

- **confirmBooking()**: Confirms the booking and generates a ticket.
- **cancelBooking()**: Cancels an existing booking.
- **getBookingDetails()**: Retrieves the details of the booking.

Payment Class

- **paymentID**: Unique identifier for each payment.
- **bookingID**: The booking ID associated with the payment.
- **amount**: The total cost of the transaction.
- **paymentMethod**: The method used for payment (credit card, PayPal, etc).
- **paymentStatus**: The current status of the payment (pending, completed, refunded).

Payment Class Function

- **processPayment()**: Processes the payment and completes the transaction.
- **paymentID()**: Retrieves the payment ID associated with a transaction.
- **refundPayment()**: Issues a refund for a canceled booking.

4.4 Development Plan and Timeline

This section describes the movie ticketing system as a whole. It outlines the tasks to create the system, when each task must be completed, and who will be responsible for completing each task.

4.4.1 System Overview

The Movie Ticketing System will allow customers to buy tickets and snacks online. They will be able to filter through locations, prices, and movies. Once they pick a movie, they can choose their specific seat. Afterward, they can purchase snacks to pick up at a certain time.

Customers will have the choice to continue as guests who can access the basic features of our software. If they create an account or log in, they can access rewards and deals. Also, certain groups like students, teachers, healthcare workers, and the military can get discounts.

4.4.2 Tasks

- Presentation Layer
    - Hire a team of UI/UX designers and frontend developers to work together to create the presentation layer
    - Obtain color scheme, graphics, and other company design requirements
    - Speak as a team to make a timeline and assign tasks to each team member
    - Fully understand who the users are to make a product oriented towards them and them alone
    - Speak to real potential customers to get their feedback and start to discover their needs
    - Create wireframes for each page to outline the flow of the application
    - Write text for each page, button, and error message. It needs to hook customers and is easy to understand
    - Obtains photos to add to the website of the movie theater and for each film
    - Create the wireframes into an actual working website
    - Have a set of potential customers to test each function within the website and give feedback
    - Fix the website based on the given feedback
    - Continuous maintain the website as new bugs arise and new movies are added to the theater
- Application Layer
    - Gather a team of software developers within the company or outsource this task to another company to work on the software
    - Everyone must review the SRS to fully understand what product must be created and review the given resources
    - Split into teams that will work on each aspect of the website, like the account creation feature or the ticket booking feature
    - Speak as a team to make a timeline and assign tasks to each team
    - Review the design specifications within the SRS document
    - Work on each part of the project
    - Test the software individually
    - Have Dev Ops put all the pieces together
    - Test the product as a whole
    - Have a set of potential customers to test each function within the website and give feedback

    - Improve the software and test again
    - Maintain the software and fix bugs as they arise
- Data Layer
    - Speak with an Amazon RDS representative about the benefits and downsides of their services
    - Review the SRS for which languages and resources are required or recommended for our system
    - Create a team to work on the database layer with people with expertise in building databases and database security
    - Review the SRS for limits and constraints on the construction of the Movie Ticketing Systems database
    - Plan it out as a team and figure out who would be good at doing what
    - Create the database and implement it into the website
    - Add some test data into the database to ensure that it works for holding accounts, and guest emails, and securely holds the information
    - Release the website and fix any bugs that arise

4.4.3 Timeline

They will implement the project at the start of summer in May 2025 and it should be completed by 2026.

[May]

- Create teams for each layer of the project and assign a leader
- Leaders should review and discuss the SRS document
- Assign preliminary tasks to each team member within each of the 3 teams
- Solidify funding for the website with the movie theater company

[June]

- Database: Speak to Amazon RDS and create the database
- UI/UX: Interview potential customers to understand their needs and expectations for the website
- Application: Create a specific plan for creating the software and assign people to each feature in the website

[July]

- Database: Organize the database for the needs of the website
- UI/UX: Create wireframes for each page to outline the flow of the application
- Application: Implement the basic structure of the website

[August]

- Database: Ensure the Database is secure and working properly

- UI/UX: Implement the design of the website as per the requirements in the SRS and the company's design requirements
- Application: Implement every desired feature for the website

[September]
- Database: Add test data to the database and ensure it is working properly for each function created by the application layer team
- UI/UX: Add more details to the application
- Application: Implement every desired feature for the website

[Octover]
- Database: Test the application with potential users
- UI/UX: test their UI/UX design with potential customers
- Application: Test each feature for it's functionality

[Novermber]
- Database: Take feedback from the tests and fix anything that needs improvement
- UI/UX: take feedback from potential users and fix anything that needs improvement
- Application: Have dev ops ensure each feature is connected properly

[Decmeber]
- Release the website and fix bugs as they arise

4.4.4 Task Assignment

All the tasks listed above in part 6.4.2 have been organized into 3 separate parts per the software architecture overview. Each part will be assigned a leader who will hire people to work as a team to complete the tasks. We will have a group for each part which includes the presentation (UI/UX), application, and database.

5. Test Plan

The test plan section outlines the goals and steps for verification and validation It contains our testing strategy and target features for verifying and validating the software system. It is intended to be reviewed by both the testing and developer teams to ensure that the desired functionality is produced.

5.1 Verification

The goal of verification is to ensure that we are properly implementing the product in a way that exhibits all of the required functionality.

5.1.1 Test Strategy

Black box and white box testing have their weaknesses, so to counter them we will implement both. We will split our testing team into two groups and each will focus on either black or white box testing.

Black box testing is good for identifying which inputs or user interactions can create errors. It will allow our product to prepare for unusual user inputs. In other words, it will make our product idiot-proof. Black box testing is less systematic, so we will use white box testing to verify that our developers properly implemented the product. White box testing requires testers to go over the source code, ensuring that our code doesn't have any bugs.

<u>Black box testing:</u>

Testable Features:

- account creation
- movie ticket purchase
- discount application
- search functionality
- Filter functionality
- user profile
- display showtimes and seat map

We will select our dataset of inputs at random because it will allow us to test a variety of features while preparing for unusual inputs from users. Also, we do not have the budget to test every possible input to 100% ensure that we have covered everything.

Each test case created must abide by the test case sample format. It will contain an ID, component category, priority, a test summary, prerequisites, test steps, expected results, and actual results. Then, there must be a column for status to mark whether the test has been completed or not and who executed the test case.

Based on the list of testable features and the required format, testers will create a collection of tests. This collection of tests must test everything from unit tests to integrated tests, to system tests.

<u>White box testing:</u>

To develop tests for the code, testers must first better understand the structure of the code. They are responsible for reviewing the ULM class diagram and the software architecture diagram.

Test requirements/coverage:

- each line of code must be reviewed and checked for errors (statement coverage)
- each branch or possibility of how the code can proceed from start to finish must be reviewed (branch coverage)

- each conditional/boolean statement must be reviewed to ensure that it produces the desired result (conditional coverage)
- each loop must be viewed to ensure that it won't result in an infinite loop or produce undesired results (loop coverage)
- each independent path must be reviewed (path coverage)

We will have a spreadsheet that contains a checklist of each component listed above that must be tested. The tester will indicate this in the spreadsheet to avoid time wasted by having it tested again.

As per the format listed above and in the test case samples, developers must create test cases that fulfill full coverage of the system as listed in the test coverage section.

Regression testing:

After finding and fixing errors within the program, we must conduct regression testing. Regression testing reviews every unit, integrated unit, and system to ensure that nothing else is broken during the improvement process. This can be conducted by redoing the tests created in the black box testing section. Also, developers are encouraged to use a tool called test manager which makes regression testing more efficient.

5.1.2 Test Components / Target Features

This section will identify the target features, including components in the unit, integrates, and system levels.

Unit Tests:
Unit tests ensure that each component (variables, methods) individually apart from the overall system is working properly. Below is a list of components that must be tested individually. Although this list is not comprehensive, it points out important components and serves as a starting point for testers to expand upon.

- Showtime.getShowtimeDetails()
- Showtime.getSeating()
- Payment.paymentID()
- getBookingDetails()
- viewHistory()

Integrated Tests:
Integrated tests ensure that key sections (classes, methods) of the program are functioning properly. Below is a list of important classes and methods, which must be tested.

- Booking interacting with Movies

- Showtimes interacting with Movies
- Payment interaction with User
- Movies interacting with User
- Booking interaction with User

System Tests:
System testing ensures that every component within the system is working in unison without error. It focuses on testing the overall functionality of each feature. Below is a list of important features that must work for the Movie Ticketing System to be successful.

- register
- login
- save payment methods
- view purchase history
- update location

5.2 Validation

The goal of validation is to ensure that we are properly implementing the product required by the client and stakeholders

5.2.1 Overview

The validation section includes an overview of the tests that ensure that the project is fulfilling the SRS requirements. Testing everything will take about 1,000 person-hours and overall 3 months to complete. The testing team will be responsible for completing the testing processes, but the development team will be responsible for fixing bugs and explaining their implementation when necessary. The requirements for this process will be addressed below.

5.2.2 Test Estimation

Estimated number of tests: 289 tests

Average Test Development Time: 2.4 person-hours/test

Estimated Test Development Time: 693.6 person-hours

Average Test Execution Time: 1.1 person-hours

Estimated Test Execution Time: 317.9 person-hours

Estimated Regression Testing (50%): 158.95 person-hours

Total Estimated Test Execution Time: 476.85 person-hours

5.2.2 Processes

The validation processes for the Movie Ticket System follow the V&V reporting structure outlined in the IEEE standards document. These processes ensure that documentation, analysis, and resolution of issues while maintaining alignment with the SRS requirements.

- Tasks: Validation tasks, such as verifying account creation or ticket purchases, are documented. Each task report included objectives, results, and status updates, these provide a clear report of progress and defects.
- Methods and Procedures: Designed to ensure a thorough evaluation of the software from the requirements outlined in the SRS. These methods and procedures are structured to cover all phases of the software lifecycle.
- Inputs: Refer to the test strategy (7.1.1) where it lists how to get inputs for each test case.
- Outputs: The output of each task should result in a test case completed or a defect found, which will result in a fix and regression testing.
- Schedule: The entire testing process must be completed within 4 months. It is encouraged for regression testing to begin within the last month.
- Resources: The testing team is required to use the proper documentation forms.
- Risks and Assumptions: The testing process is at risk of a lack of technical aid. Due to budget constraints, we may lack the man power to complete the testing process in the required timeframe.
- Roles and responsibilities: The test team is responsible for completing all of the testing and the development team is responsible for fixing any bugs that arise throughout the testing process. Also, the test team must be available to speak with the test team if they have any questions on the code.

5.2.3 Test Development and Informal Validation

During informal validation, developers must review the SRS document and ensure that all product functions have been implemented. During informal validation, new features may be added to aid in meeting the requirements outlined in the SRS. This is where they will help refine the reporting requirements listed below before formal validation must be completed.

5.2.4 Formal validation

During formal validation, testers must review the SRS one more time to ensure that every requirement of the product has been met. No more features may be added and the tests must be completed. The testers must execute all of the tests to find defects or areas of improvement. Then, the developers are responsible for making the required improvements. Then, the testers must review the test design document below. Every test case must be run and any test that fails must be documented and fixed by developers. After all bugs have been fixed, regression testing must be completed.

5.2.5 Reporting Requirements

Task Report:

- Software SRS Analysis ~ a document that lists all of the SRS requirements where a tester must review and complete
- Access Analysis ~ a document outlining all the ways that a user can access the software and ensure that they all work

Activity Summary Report:
- Test Completed ~ log that the specific test with ID number has been completed
- Test Failure ~ log which test failed and why
- Failure Solution ~ include some recommendations for possible fixes

Anomaly Report:
- Priority Fix Report ~ explain why this bug must be fixed before other bugs
- Cause of Anomaly ~ explain why the anomaly may have occurred
- Recommendation form ~ explain who is responsible for fixing the anomaly

V&V Final Report:
- Summary of activities ~ a log of what has been done throughout the V&V process
- Summary of tasks ~ a log of which task have been completed from executed test cases to test case development
- Summary of anomalies ~ provides a high level description of how each anomaly was fixed
- Future improvements ~ what must be done in the future

5.2.6 Administrative Requirements

Whenever a bug arises during formal validation, a document must be completed elaborating on the bug and who must fix it. Then, the document must be marked as done once regression testing has been performed.

Based on our schedule and budget, we are going to require regression testing to be completed after each bug is fixed to ensure that the improvements didn't cause any other issues in the system.

If any changes to the test plan are made, a document must be completed outlining why this course of action was taken. Also, it will include what the new test plan consists of and how this will affect the quality control of our system.

5.2.7 Documentation Requirements

The first required document is this test plan in the SRS. Also, it will need to include the test design, a document containing information of the purpose of each test and how it provides full test coverage of the system. Then, an expanded spreadsheet following the formatt of the table shown below of ten test case samples is required to be included. It must include all 298 tests. Furthermore, it must include the document called Test Procedure. It includes the systematic process for completing all of the tests. In summary, a tester must be with a developer while reviewing the code to provide quick feedback and a better understanding of the code. Finally, the document called test results must be included. The test results document includes a checklist of all the steps that must be completed during the testing process.

5.2.8 Resource Requirements

The test team is encouraged to use a test case tool called Test Rail. It helps to manage tests cases and test fail reports. It provides a way to schedule test runs. Also, it can integrate with Github to promote collaboration among the testing team.

5.2.9 Test Completion Criteria

For Validation Testing to be completed, all tests must have been executed and reviewed. All bugs and reported errors must have been resolved. Regression testing must be completed and errors that have been resolved must be tested again. After all errors have been resolved, the documentation of the system must be updated.

5.3 Test Cases Samples (Hyperlink)

Notes:

1. Priority Levels:
   P0: Critical functionality (must pass for system to work)
   P1: Important functionality (should pass but not critical)
   P2: Nice-to-have functionality (optional)
2. Status:
   Pass: Test case executed successfully
   Fail: Test case did not meet the expected result

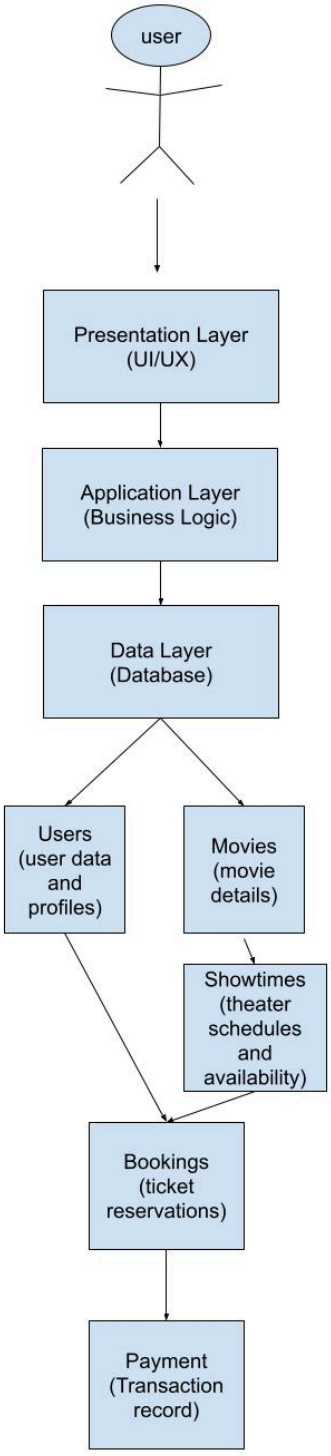
6. Data Management

The data management section outlines our data management strategy and explains why we set it up this way. Also, we include a software architecture diagram that visually represents how the database interacts with the other features.

6.1 Software Architecture Diagram

The software architecture diagram demonstrates how the software interacts with the database. It is used to visualize the interactions.

6.1.1 Diagram



6.1.3 Diagram Explanation

The software architecture diagram visually represents how the software interacts with the database. Furthermore, it breaks down the database into different sections that are in charge of

holding different information about the movie company and users. Then, it shows how each section interacts with each other.

6.1.2 Diagram Changes

The original software architecture diagram includes a box for how the software interacts with the database. It was simply represented by a box labeled “Database Layer”. This new diagram, presented above in section 8.1.1, breaks down the database into different database structures. It shows how the database layer is made up of structures that hold the user data as well as the movies and showtimes. Then it connects the user data and movie/showtime data to make purchases. After booking the movie you need to update the payment database so the company has a record of all transactions. Additionally, I added a user at the beginning which interacts with the presentation layer or the browser where our software is visually displayed to the user. From there the arrows show how the user interacts with all the hidden parts of our software.

6.2 Data Management Strategy

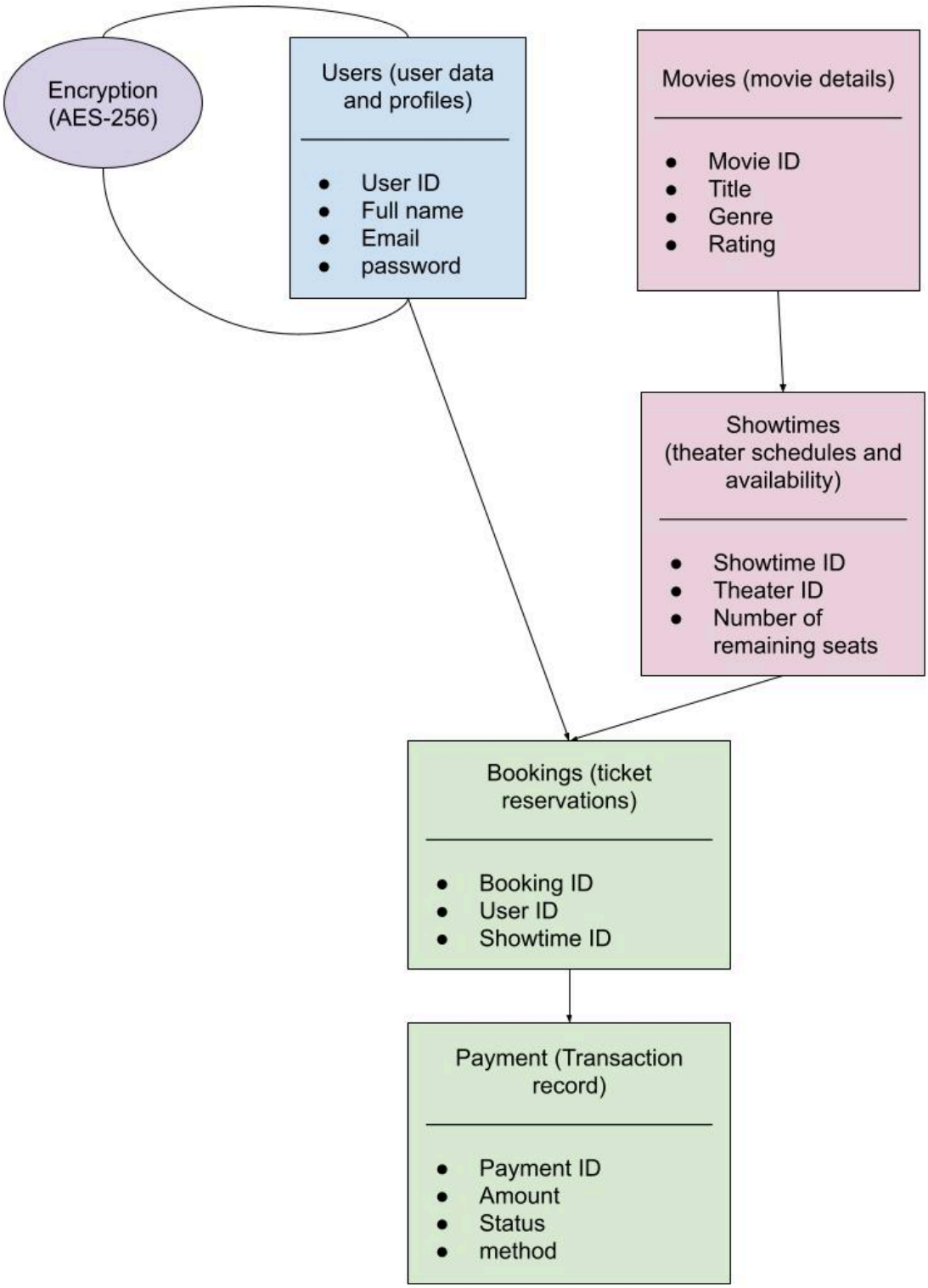
Database Structure:

Table	Purpose	Fields
Users	Stores account details and links to Bookings and Payments	userID, fullName, email, password
Movies	Contains movie details and data	movieID, title, genre, rating
Showtimes	Manages theater schedules and seat availability in real time	showtimeID, theaterID, remainingSeats
Bookings	Ticket reservations	bookingID, userID, showtimeID
Payments	Records transaction status and payment method	paymentID, amount, status, method

Example of User Table:

userID	fullName	email	password (encrypted)
u1	Example Name1	example1@email.com	********
u2	Example Name2	example2@email.com	********
u3	Example Name3	example3@email.com	********

Diagram:


Encryption (AES-256)
Users (user data and profiles)
User ID
Full name
Email
password
Movies (movie details)
Movie ID
Title
Genre
Rating
Showtimes (theater schedules and availability)
Showtime ID
Theater ID
Number of remaining seats
Bookings (ticket reservations)
Booking ID
User ID
Showtime ID
Payment (Transaction record)
Payment ID
Amount
Status
method

Design Decisions:
We use SQL for structured, transactional data. This was chosen to allow for ticket reservations as well as complex filtering of movies. Also, a single database was chosen to simplify maintenance and allow all tables to be synced.

Security & Compliance:
Encryption: AES-256 for passwords/payments.
CCPA Compliance: Users can request data deletion.

6.3 Tradeoff Discussion

Why SQL?
We chose SQL because it works well with structured relational data and supports transactions which are important for things like booking tickets and making payments. It makes it easier to run more complex searches like checking available showtimes and storing a user's past history.

Why Single database?
Using one database keeps things simple and easy to manage. It's easier to manage, back up, and keep everything in sync. It also helps with data consistency, especially since different parts of the system are closely connected (like users, bookings, and payments).
Alternative: We could have split the system into multiple databases (for example, separating user info from booking data). That might improve performance or security in the long run, but it would also make the system more complex and harder to maintain which is important for a system like ours.

Security tradeoff:
Using a strong encryption like AES-256 for passwords and payments adds processing time making the process slower but is a well worth tradeoff for the protection provided.

Privacy and compliance:
Our system is CCPA compliant so users can request to delete their data. This adds steps in the backend but it's important to be compliant not only for legal reasons but also for our users to trust us.

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