Ticket Hub

Software Requirements Specification

1.0

09/25/2025

Group 6
Angel Davila, Luis Villalon, Stephen Conley

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

Revision History

Date	Description	Author	Comments
09/25/25	Version 1	Stephen Conley, Angel Davila, Luis Villalon	First revision

Document Approval

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

Signature	Printed Name	Title	Date
SC, AD, LV	Stephen Conley, Angel Davila, Luis Villalon	Software Eng.	09/25/25
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to define all the functional and non-functional requirements of the movie ticketing software system. This document establishes a common understanding between stakeholders, programmers and clients regarding the objectives, scopes, and features of the system. It serves as a reference throughout the software development life cycle to ensure that the system is designed, implemented, and validated according to the agreed requirements. By outlining the intended functionality, constraints, and performance criteria, this SRS aims to minimize ambiguity, reduce development risks, and provide a solid foundation for the successful delivery of a reliable and user-friendly movie ticketing software system solution.

1.2 Scope

This software product, Ticket Hub, allows customers to browse available movies, view showtimes, purchase tickets, and order food and beverages in advance.

Functionality:

- Enable browsing of current and upcoming movies with information such as title, genre, duration, and showtimes.
- Support real-time seat selection and ticket booking with secure payment processing.
- Provide options for purchasing food and beverages online, with the ability to schedule pickup time.
- Generate and deliver electronic tickets and order confirmations via email and SMS.

The system will not include theater hardware control, third-party entertainment services, or concession stand operations outside of online ordering.

Customers will benefit from more convenient access to tickets and food services, reduced waiting times, and secure transactions. Theater operators will benefit from reduced manual workload, real-time sales tracking, and improved customer satisfaction.

Goals:

- Provide real-time access to movie schedules and seat availability, with updates processed in less than 2 seconds.
- Offer secure and reliable online payments, meeting industry standards for data protection.
- Deliver user-friendly interfaces that require minimal training for customers.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS (Software Requirements Specification):** A document that describes the functional and non-functional requirements of the software system.
- **UI (User Interface):** The part of the system with which the end-user interacts.
- **DBMS (Database Management System):** Software used to manage and store data such as user information, movie schedules, and ticket bookings.

- **API (Application Programming Interface):** A set of functions and protocols that allow communication between different software components, such as payment gateways.
- **Admin:** A system user with elevated privileges who can manage movie listings, schedules, and sales reports.
- **Customer/User:** An individual who uses the system to browse movies, book tickets, and order food or drinks.
- **Payment Gateway:** A third-party service that securely processes online payments.

1.4 References

IEEE Recommended Practice for Software Requirements Specifications

- Std 830-1998

1.5 Overview

The remainder of this document is organized as follows:

- **Section 2: Overall Description** – Provides background information, including product perspective, functions, users, constraints, and assumptions.
- **Section 3: Specific Requirements** – Lists the functional and non-functional requirements in detail.

2. General Description

2.1 Product Perspective

Ticket Hub will operate as a web and mobile platform integrated with external services, including:

- **Database Management System (DBMS):** To store user accounts, movie schedules, booking records, and order histories.
- **Payment Gateway Integration:** To securely process transactions using credit/debit cards or digital wallets.
- **Email/SMS Services:** To deliver booking confirmations, electronic tickets, and food order notifications.

2.2 Product Functions

The product will provide users with the ability to purchase, review, and cancel tickets to a chosen theater in advance. It will also allow users to order concessions prior to arrival. There will be an option to leave reviews of the company.

Administrators will be able to view/manage tickets and concession orders.

2.3 User Characteristics

Users will range from teenagers to elderly, with widely varying technological experience, with at least basic web browsing ability. Users will access the product from various devices, such as mobile phones or personal computers.

2.4 General Constraints

- Internet
    - User must be constantly connected to ensure real-time availability of seats
- Compatibility
    - Must accommodate mobile devices (touchscreen)
    - Must accommodate lower memory devices
- Audits
    - Must have recorded logs of all transactions, errors, and user IP addresses
- User Data
    - Must have third-party handle user's card information
    - Will not store the card information on our server

2.5 Assumptions and Dependencies

- **Assumptions:**

1. Users will have stable internet connections.
2. Customers will use compatible devices (smartphones, tablets, or computers)
3. A secure and reliable payment gateway service will be used
4. Theaters will update the movie schedules, pricing, and seat availability.
5. Users will provide valid contact information for confirmations.
6. The reliable hosting infrastructure will be reliable and appropriately scaled.

- **Dependencies:**

7. Third-party services (payment gateways, email/SMS providers) for transactions and notifications.
8. Database management system (DBMS) for consistent data storage and retrieval.
9. Hosting provider (e.g., cloud servers) for availability and performance.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- Visualization of each theater seating arrangement
    - Show seat availability, special accommodation, screen location
- Map of theater locations
- List showing times per movie, separate into different sections for different levels of theater technology

3.1.2 Hardware Interfaces

- Works on multiple devices (desktop, tablet, mobile, etc.)
- QR scanner (at location of sale)

3.1.3 Software Interfaces

- Payment via Stripe
    - Use Stripe API to authorize payment online
- The Movie Database (TMDb)

    - Pull information regarding movies
        - Trailers, summarization, actors
- MySQL database management
    - Store user credentials, reviews, theater tickets, etc.

3.1.4 Communications Interfaces

- Email communications with users
    - Confirmation, reminders, promotions

3.2 Functional Requirements

3.2.1 Ticket Ordering

3.2.1.1 Introduction

Serves as the main method of ordering tickets and viewing different movies and showtimes. Will include selective seating and reserves tickets in the customer's cart for a few minutes to prevent their seat from being purchased by another customer while checking out.

3.2.1.2 Inputs

- Email/Password for account login and email receipt
- Seat Selection(s)
- Movie selection
- Theater Selection

3.2.1.3 Processing

- System checks available theaters, movies, and corresponding showtimes
- System validates payment method
- System checks available seating
- System temporarily reserves chosen seating
- System validates account/email

3.2.1.4 Outputs

- Receipt with purchase confirmation
- Email to account/email with order information
- Updates seating database removing the availability of seats just purchased.

3.2.1.5 Error Handling

- If specific seating or showtime becomes unavailable prior to purchase, then an error screen will pop up removing ineligible seat(s) from cart and prompting customer to try again from the seating availability screen.
- If the payment method fails, bring up an error screen and prompt customer to try again or with a different method.

3.2.2 Canceling & Refunding Purchases

3.2.2.1 Introduction

Allows customers to cancel or refund past purchases as long as it adheres to the company policies.

3.2.2.2 Inputs

- User email and password to login
- Selection of purchase
- Preferred destination of refund (if refundable)

3.2.2.3 Processing
- Verify the request with policy restrictions
- Verify the purchase has not been canceled/refunded already
- Verify payment destination is correct and matches billing address and home address

3.2.2.4 Outputs
- Email verification
    - Either confirmation or notice of failure (with description)
- Update database for seat availability or concession requests
- Flag user purchase as refunded/canceled to prevent double attempt

3.2.2.5 Error Handling
- Boolean flag for if the purchase is already canceled
- Flag if request is not within policy agreement

3.2.3 Food & Beverage Ordering

3.2.3.1 Introduction

This function allows customers to browse, select, and purchase food and drink items through the movie ticketing software. Customers can add items to their cart, select a pickup date and time, and complete payment.

3.2.3.2 Inputs
- User-selected food and drink items
- Quantity of each item
- Preferred pickup date and time
- Contact information (email, phone number)
- Payment details (credit/debit card)

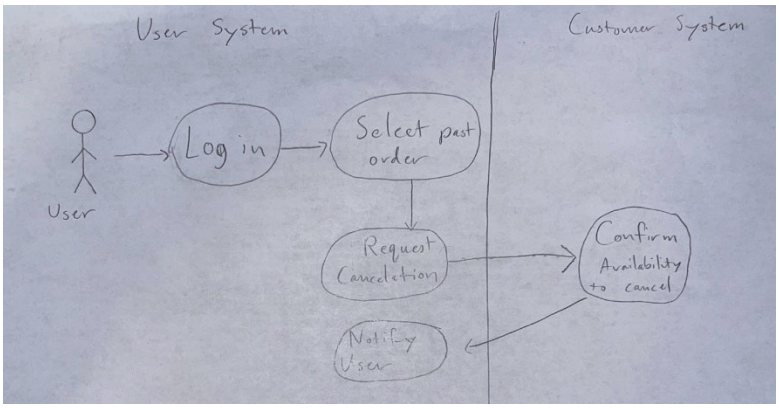
3.2.3.3 Processing
- The system validates item availability from the menu database.
- The selected items are added to the customer's cart.
- The system records the chosen pickup date and time.
- The system validates contact and payment details.
- Upon successful payment, the order is confirmed and stored in the system’s database.

3.2.3.4 Outputs
- Order confirmation is displayed on the screen.
- Electronic receipt and pickup details sent via email or SMS.
- Updated inventory of available food and drink items.

3.2.3.5 Error Handling
- If an item is out of stock, the sytem notifies the cusotmer and removes it from the cart.
- If payment fails, the order is canceled, and the customer is prompted to retry.
- If contact details are invalid, the system requests correction.

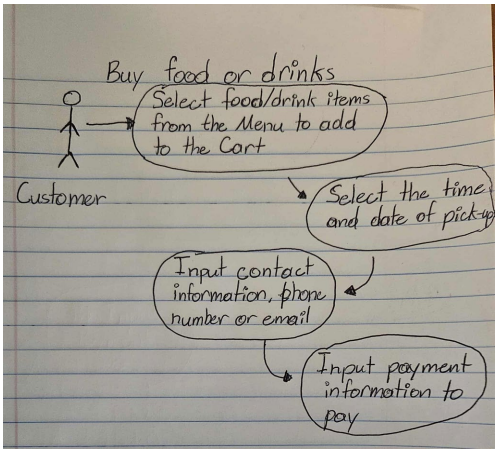
3.3 Use Cases

3.3.1 Reviewing/Canceling Past Orders



1. User will log in to their account
2. User will select purchase they wish to review/cancel
3. User will request cancelation (if applicable)
4. Customer system will verify cancelation
5. User will receive confirmation of success or failure

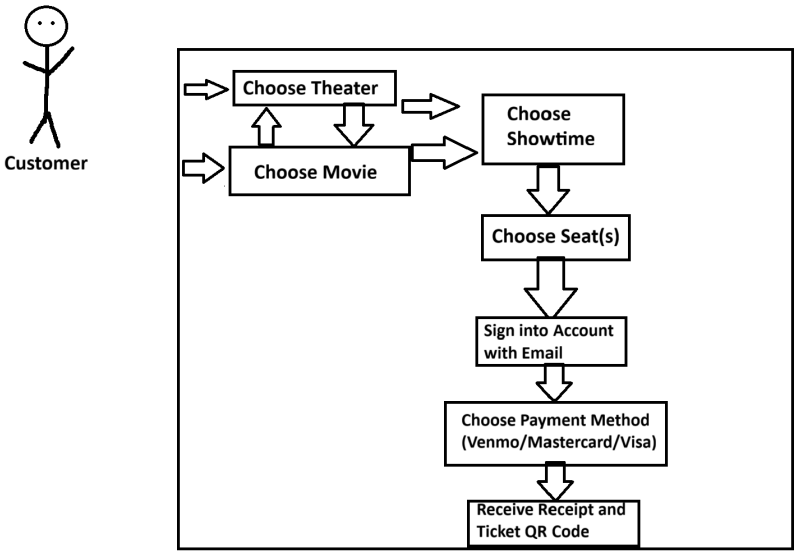
3.3.2 Customer Purchases Food and Drinks



Name of Use Case: Buy food or drinks
Actor(s): Customer
Flow of events:

1. The customer will select various food items and drinks to add to their cart.
2. The customer will select the time and date they wish to pick up their order.
3. The customer will input their contact information, either their phone number or email.
4. The customer will input their payment information to pay for the order.

3.3.3 Customer Purchases Tickets



1. Customer starts by logging into their account, or they can start by selecting a movie/theater.

2. After the customer chooses both a theater and movie, they will be prompted showtimes followed by seat availability.

3. Then they'll sign into their account if they haven't already before getting to checkout screen.

4. Finally, they'll get the receipt with their ticket QR code and email confirmation

3.4 Classes / Objects

3.4.1 Class 1: customerAccount

3.4.1.1 Attributes

- Customer Email
- Customer Password
- Customer Movie Tickets / Order History
- Billing info (Card info)

3.4.1.2 Functions

- retrieveTicket (Recieves ticket info likely in QR code format)
- chargeCustomer (Utilizes billing info for purchase of tickets/food)
- updateBillingInfo(Recieves the Customer's inputted Billing information and updates the current value)

Reference to functional requirements and/or use cases

- **Functional Requirement:** Ticket Ordering, Reviewing & Canceling Purchase (see Section 3.2.1, 3.2.2)
- **Use Case:** Customer Purchases Tickets, Reviewing/Canceling Past Orders

3.4.2 Class 2: foodOrder

3.4.2.1 Attributes

- **OrderID** – Unique identifier for each food/drink order.
- **CustomerID** – Reference to the customer account that placed the order.
- **Items[]** – List of selected food and drink items (with quantity).
- **PickupTime** – Date and time selected by the customer for pickup.
- **ContactInfo** – Email or phone number for sending order confirmations.
- **OrderStatus** – Tracks status (Pending, Confirmed, Canceled, Completed).
- **PaymentInfo** – Payment method details used for the order.
- **TotalPrice** – Calculated total cost of the order.

3.4.2.2 Functions

- **addItem(item, quantity):** Adds a food/drink item to the order list.
- **removeItem(item):** Removes a food/drink item from the order.
- **calculateTotal():** Calculates the total cost of all items in the order.
- **confirmOrder():** Finalizes the order and updates the system database.
- **sendConfirmation():** Sends order confirmation via email or SMS.

Reference to functional requirements and/or use cases

- **Functional Requirement:** Food & Beverage Ordering (see Section 3.2.3)
- **Use Case:** Customer Buys Food or Drinks

3.5 Non-Functional Requirements

3.5.1 Performance

- The system shall support at least 500 concurrent users performing transactions without degradation in performance.
- The system shall ensure that movie schedules load within 2 seconds under normal network conditions.
- The system shall process payment transactions within 5 seconds of submission.
- The system shall handle peak usage during weekends and holidays without downtime.

3.5.2 Reliability

- The system shall ensure accurate and consistent booking records by preventing double-booking of the same seat.
- The system shall recover gracefully from unexpected failures without loss of confirmed transactions.
- The system shall maintain error logs and monitoring tools to track failures and support rapid issue resolution.
- The system shall guarantee that confirmed bookings remain stored in the database and retrievable at all times.

3.5.3 Availability

- The system shall be available online 24/7 with the exception of occasional maintenance downtime, which shall be announced prior to.

3.5.4 Security

- All transactions shall be done via a third-party service, with adequate encryption, such as TLS 1.3.
- There will be authentication for user logons, utilizing two-factor authentication when applicable.

3.5.5 Maintainability

- System will be maintained through a central system that automatically updates movie catalogue and showtimes per theater as information is provided

3.5.6 Portability

- System will be accessible through web browser and app
- Browser will feature differing user interface that better fits desktop functionality
- App will be more condensed for better phone and other small device use

3.6 Inverse Requirements

*State any *useful* inverse requirements.*

3.7 Design Constraints

Specify design constrains imposed by other standards, company policies, hardware limitation, etc. that will impact this software project.

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.

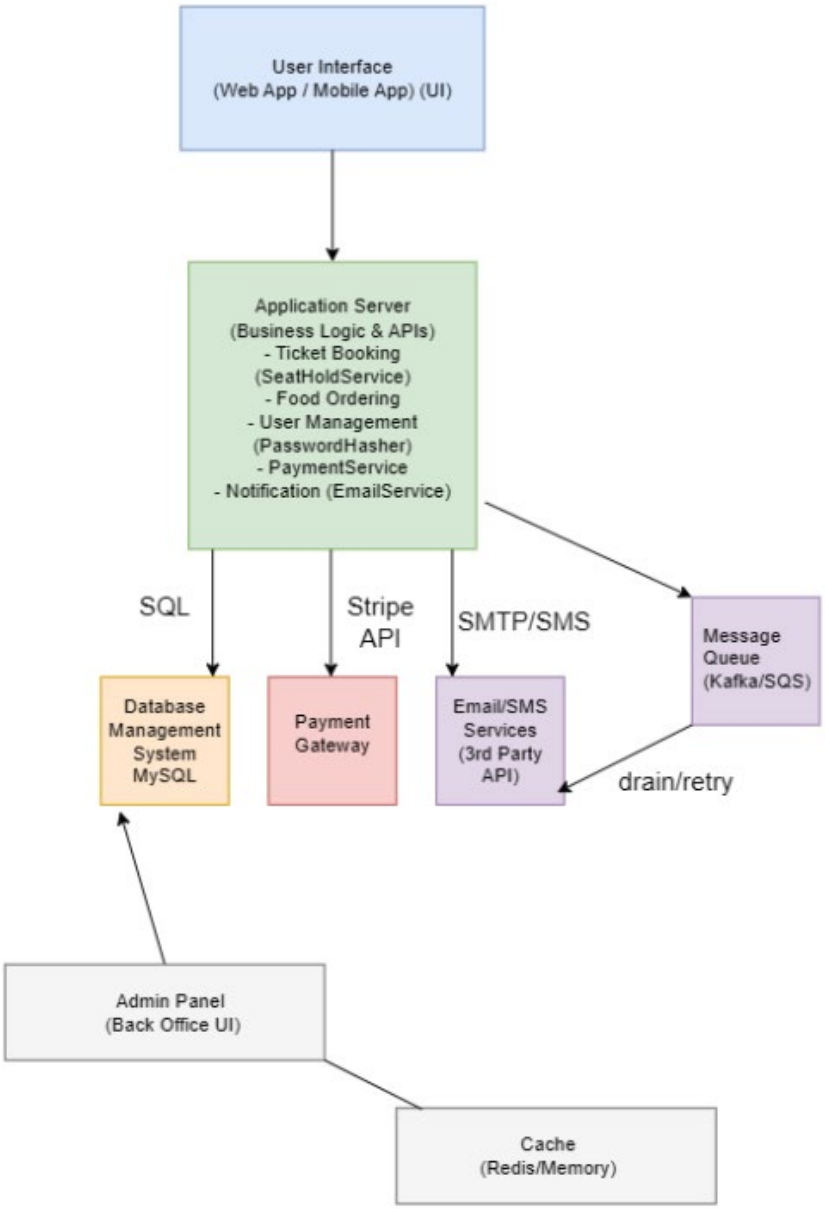
3.9 Other Requirements

Catchall section for any additional requirements.

4. Software Design

4.1 Software Architecture Overview

4.1.1 SWA Diagram



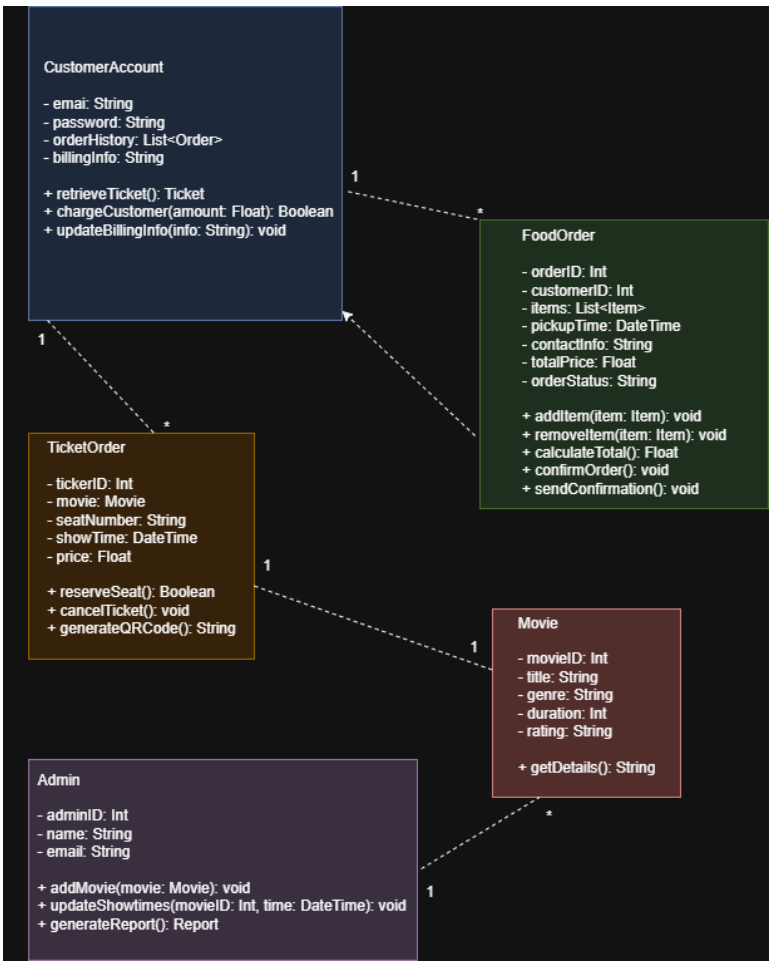
Caption:
The Ticket Hub architecture follows a client–server pattern. The UI communicates with the Application Server via REST. The server persists data in MySQL and integrates with third-party services (Stripe for payments, SMTP/SMS for notifications). A read cache accelerates hot reads (showtimes, seat maps), and a message queue buffers critical writes during transient failures to meet reliability goals and the DB-outage recovery test. (See SRS 3.1.3 for Stripe/MySQL and 2.4 “User Data” constraints; 3.5.1–3.5.2 for performance/reliability.)

“Changes since Assignment 2”:
Compared to A2, we added a Cache to satisfy performance targets and a Message Queue to satisfy outage-recovery requirements validated by our DBOutage_1 test, while keeping the existing modules (SeatHoldService, PaymentService, EmailService) intact.

4.1.2 SWA Description

- **UI**: Handles views and input; communicates with servers
- **Application Server:**
  - Ticket Booking
    - SeatHoldService
  - Food Ordering
  - User Management
    - PasswordHasher
  - Payment Processing
    - PaymentService
  - Notification
    - EmailService
- **Database:** Source of users, movies, showtimes, seats, bookings, reviews, and any other necessary information
- **Payment Gateway:** Creates purchase and refund intents with Stripe
- **Email SMS:** Outbound notifications to users
- **Admin Panel:** Staff UI for managing data and viewing reports

4.2 UML Diagram



4.3 UML Class Descriptions

Class: CustomerAccount
Represents a registered user of the Ticket Hub system. Each CustomerAccount holds personal credentials, billing details, and order history for both movie tickets and food orders. The class provides functions to manage billing information, retrieve ticket data, and process customer payments.

Attribute	Data Type	Description
email	String	The unique email address used by the customer for login and notifications.
password	String	The encrypted password associated with the account.
orderHistory	List<Order>	A list of all previous orders, including ticket and food purchases.

billingInfo	String	Encrypted billing details or a reference token from the payment gateway.

Operation	Parameters	Return Type	Description
retrieveTicket()	ticketID: Int	Ticket	Retrieves the details of a specific movie ticket by ID, including seat number and showtime.
chargeCustomer()	amount: Float, paymentMethod: String	Boolean	Initiates a charge through the payment gateway using stored billing information. Returns true if payment succeeds.
updateBillingInfo()	newInfo: String	void	Updates the stored billing information for the customer after validation.

Class: FoodOrder

Handles all operations related to concession orders. The FoodOrder class allows customers to select, update, and confirm food or beverage orders, as well as schedule pickup times and receive confirmation notifications.

Attribute	Data Type	Description
orderID	Int	A unique identifier for each food and drink order.
customerID	Int	Identifier referencing the CustomerAccount that created the order.
items	List<Item>	A collection of selected food and drink items with their quantities.
pickupTime	DateTime	The chosen date and time for pickup of the order.
contactInfo	String	Customer’s contact email or phone number for order confirmation.
totalPrice	Float	The total calculated cost of the order.

orderStatus	String	Represents current status: “Pending”, “Confirmed”, “Canceled”, or “Completed”.

Operation	Parameters	Return Type	Description
addItem()	item: Item, quantity: Int	void	Adds a food or drink item to the current order with the specified quantity.
removeItem()	itemID: Int	void	Removes the specified item from the order list.
calculateTotal()	—	Float	Calculates the total price of the order based on selected items and their quantities.
confirmOrder()	—	Boolean	Finalizes the order, triggers payment processing, and updates the database. Returns true if confirmation succeeds.
sendConfirmation()	contactInfo: String	void	Sends a digital receipt and pickup details to the customer via email or SMS.

Class: TicketOrder

Manages the process of purchasing and managing movie tickets. The TicketOrder class handles seat selection, ticket confirmation, and QR code generation for entry validation.

Attribute	Data Type	Description
ticketID	Int	A unique identifier assigned to each ticket order.
movie	Movie	The Movie object associated with the ticket.
seatNumber	String	The selected seat number within the theater.
showtime	DateTime	The date and time of the selected movie showing.
price	Float	The price of the purchased ticket.

Operation	Parameters	Return Type	Description
reserveSeat()	seatNumber: String	Boolean	Temporarily reserves a seat for checkout. Returns true if reservation succeeds.
cancelTicket()	ticketID: Int	void	Cancels the ticket and updates seat availability in the database.
generateQRCode()	—	String	Generates and returns a QR code string representing the ticket confirmation for entry scanning.

Class: Movie
Represents a movie listed in the theater system. The Movie class stores metadata such as title, genre, duration, and rating, and interacts with external APIs (like TMDb) to fetch information and posters.

Attribute	Data Type	Description
movieID	Int	Unique identifier for each movie in the database.
title	String	The title of the movie.
genre	String	The genre of the movie (e.g., Action, Comedy, Drama).
duration	Int	Movie runtime in minutes.
rating	String	Audience or critic rating (e.g., PG-13, R).

Operation	Parameters	Return Type	Description
getDetails()	—	String	Retrieves formatted movie information including title, genre, duration, and rating.

updateSchedule()	showtime: DateTime	void	Updates the movie's showtime schedule (admin use only).

Class: Admin
Description
Represents an administrative user who manages system operations such as adding movies, modifying showtimes, and generating reports. The Admin class provides functions for maintaining and monitoring theater operations.

Attribute	Data Type	Description
adminID	Int	Unique identifier for each admin account.
name	String	Full name of the administrator.
email	String	Contact email address for notifications and login.

Operation	Parameters	Return Type	Description
addMovie()	movie: Movie	void	Adds a new movie to the system database.
updateShowtimes()	movieID: Int, time: DateTime	void	Updates showtimes for a given movie.
generateReport()	reportType: String	Report	Generates a sales or performance report based on the given type.

4.4 Development Plan & Timeline

4.4.1 Team roles

- Project Manager: Planning/goals, reviews repository merges, manages meetings
- Backend Lead: Manages API, databases, and unit tests
- Frontend Lead: Manges UI and overall user experience, integration of UI with backend
- Quality Assurance/documentation: Manages overall testing, product readiness, and documentation

4.4.2 Timeline

Phase 1: Requirement Analysis
- Duration: 1-2 Weeks
- Tasks:
    - Develop and approve Software Requirements Specification (SRS)

Phase 2: System Design
- Duration: Week 3-4
- Tasks: Create Software Design Specification (SDS), including UML and architecture diagrams.

Phase 3: Development
- Duration: Week 5-8
- Implementation of modules and databases

Phase 4: Testing & Debugging
- Duration: 9-10 Weeks
- Tasks: Conduct unit, integration, and acceptance testing. Conduct front-end user interference testing, test back-end services for security measures.

Phase 5: Deployment
- Duration: Week 11-12
- Deploy final version of Ticket Hub

5. Data Management Strategy

5.1 Storage Model

We use **MySQL** as the authoritative Online Transaction Processing (OLTP) store for users, movies, showtimes, seat maps, orders, and refunds (SRS 3.1.3). Card data is **never stored**; we persist only in Stripe **payment intent IDs / tokens** to comply with the constraint that a third party handles card info (SRS 2.4 “User Data”). Read performance is improved with a **Redis cache** for hot datasets (movies, showtimes, seat availability). A **Message Queue** buffers critical writes/notifications so transactions complete even during short DB outages (validated by test DBOutage_1).

5.2 Logical Schema

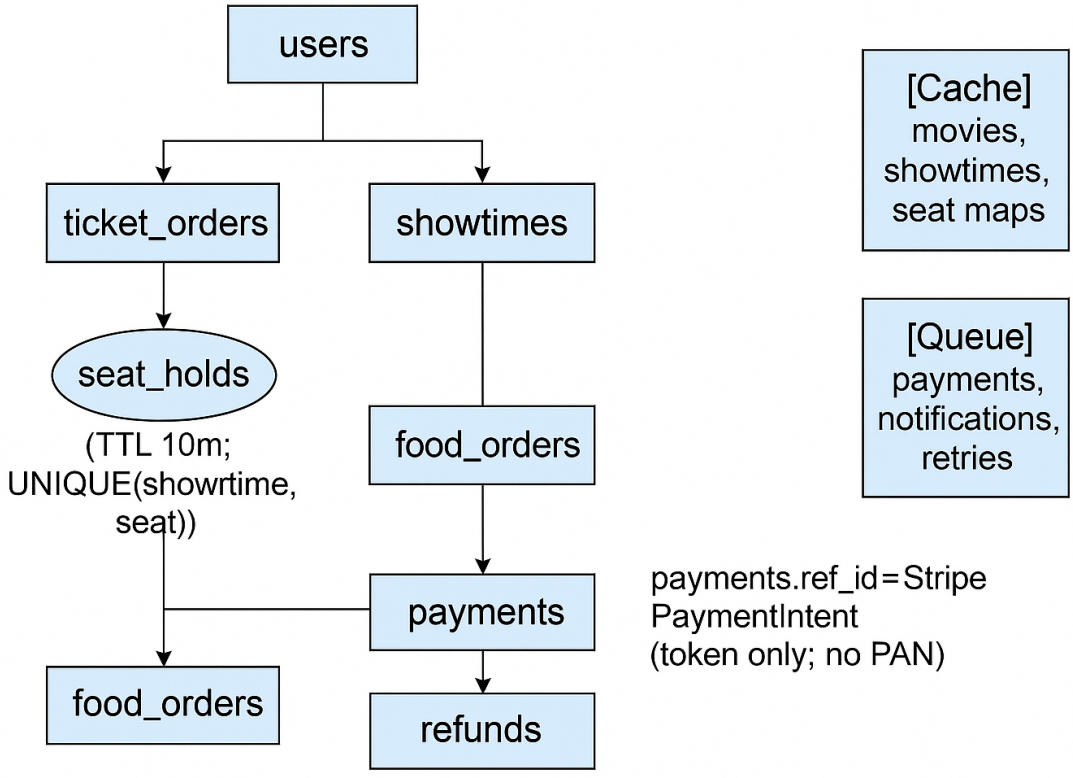
- **users**(user_id PK, email UNIQUE, password_hash, created_at)
- **movies**(movie_id PK, title, genre, duration_min, rating)
- **showtimes**(showtime_id PK, movie_id FK, theater_id, starts_at)
- **seats**(seat_id PK, theater_id, row, number, accessible BOOLEAN)
- **seat_holds**(hold_id PK, showtime_id FK, seat_id FK, user_id FK, **expires_at**; UNIQUE(showtime_id, seat_id))
- **ticket_orders**(ticket_id PK, user_id FK, showtime_id FK, seat_id FK, price, status, **qr_code**; UNIQUE(showtime_id, seat_id) to prevent double-booking)
- **food_orders**(order_id PK, user_id FK, pickup_time, contact_info, total_price, status)

- **food_order_items**(order_id FK, item_id FK, qty, price_at_purchase; PK(order_id, item_id))
- **payments**(payment_id PK, order_ref (ticket|food), ref_id, **stripe_intent_id**, amount, status, created_at)
- **refunds**(refund_id PK, payment_id FK, amount, status, created_at)
- **notifications**(notif_id PK, user_id FK, channel, payload, status, created_at)

5.3 Data Flow & Integrity

- **Seat protection:** create a *seat_holds* row with TTL (10 min) when user enters checkout (maps to SeatHoldService); commit order → delete hold → insert ticket_orders. (Supports 3.2.1 Processing and error handling.)
- **Payments:** backend creates **Stripe PaymentIntent**, stores only *stripe_intent_id* in **payments**; on success, finalize order; on failure, roll back and release seat. (SRS 3.1.3, 3.2.1.5).
- **Email/SMS:** enqueue a **notifications** job so emails are idempotent and retriable.
- **Auditing:** log errors and events to meet audit constraints and reliability goals (SRS 2.4, 3.5.2).

5.4 Mini Data Diagram



5.4.1 Why SQL (and why one primary DB)?

- Strong **ACID** transactions and relational constraints prevent double-booking

- Workload is **transactional** (OLTP) with clear relations (users↔orders↔seats).
- One primary DB keeps ops simple

5.5 Alternative & Tradeoffs

- **NoSQL (e.g., DynamoDB/MongoDB):**
  - Pros: Horizontal scale, flexible schema
  - Cons: Harder to enforce seat uniqueness and multi-row transactions
- **Multiple databases (orders vs catalog):**
  - Pros: Isolation, scale independently
  - Cons: More ops complexity, cross-DB joins become app logic.
- **No cache:**
  - Pros: Simplicity
  - Cons: Hard to meet "fast schedule loads" and "500 concurrent users" targets.
- **No queue:**
  - Pros: Simplicity
  - Cons: Risk lost writes during transient outages.

6. 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.

6.1 Sequence Diagrams

6.3 Data Flow Diagrams (DFD)

6.2 State-Transition Diagrams (STD)

7. 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.

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