Ticket Booking Site

Software Requirements Specification

2.0

3/27/24

Group 20

Jake Linnell, Osvaldo Mendez, Bryan Zavala

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Spring 2024

Revision History

Date	Description	Author	Comments
2/14/24	Version 1	Jake Linnell, Osvaldo Mendez, Bryan Zavala	First Revision
2/28/24	Software Design Specification	Jake Linnell, Osvaldo Mendez, Bryan Zavala	
3/13/24	SDS: Test Plan	Jake Linnell, Osvaldo Mendez, Bryan Zavala	Testing the Software Design Specification
3/27/24	Architecture Design w/Data Mgmt.	Jake Linnell, Osvaldo Mendez, Bryan Zavala	Architecture and Data Management Plan

Document Approval

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

Signature	Printed Name	Title	Date
	Group 20	Software Eng.	2/14/24
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

This Software Requirement Specification (SRS) defines the requirements for the Cin-Emma's Online Movie Booth website.

1.1 Purpose

This document outlines the requirements for a website that allows customers to book movie tickets at Cin-Emma's theatres.

1.2 Scope

The Cinema Ticket Booking website will offer customers a platform to view movie times, select seats, and purchase tickets online, interfacing with the cinema's system to manage bookings and payments. This SRS will detail the user interface focused on ease of use, the process of selecting a movie and seats, and the integration with the cinema's existing systems to understand our website's role and functionalities.

1.3 Definitions, Acronyms, and Abbreviations

Abbreviation	Brief Explanation
DDoS	distributed denial-of-service
TLS	Transport Layer Security. Widely adopted security protocol.

1.4 References

- https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/
- Resource explaining TLS.

1.5 Overview

The remainder of the document will cover a general description of the website including the user interface, hardware and software considerations, user considerations, and functional requirements.

2. General Description

2.1 Product Perspective

Our website will be designed such that a consumer can access and properly view and utilize it on desktop or mobile browser. It will collect interface with a database to display movies and showtimes and collect other data for the theatre such as ticket purchases. The payment processing will be done through a major third party such as PayPal or Stripe.

2.2 Product Functions

The website will require a login for the sake of data collection and identification. Users then select movies from displayed options and purchase tickets which are tracked and stored in the client's database.

2.3 User Characteristics

System users should be able to access the internet and understand the technology to access it. It will be assumed that the user has a minimal knowledge of technology.

2.4 General Constraints

The website needs to display on the major browsers in both desktop and mobile modes. Ticket purchasing will also be limited to the seats available in the theatres.

2.5 Assumptions and Dependencies

The client is in a low-population area which may not see much traffic. It can be assumed that we should aim to support several thousand concurrent users. As an online service, we must also assume that it may be targeted by a DDoS, or some users may try to use scripts to buy tickets in mass.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The website must have big buttons using colors consistent with the theatre's theme.
The website must use movie posters for selection.

3.1.2 Hardware Interfaces

The website must be able to display on mobile or desktop devices.

3.1.3 Software Interfaces

The website will be compatible with Safari, Chrome, Mozilla Firefox, and Edge.

3.1.4 Communications Interfaces

The website will communicate with the theatre's database.

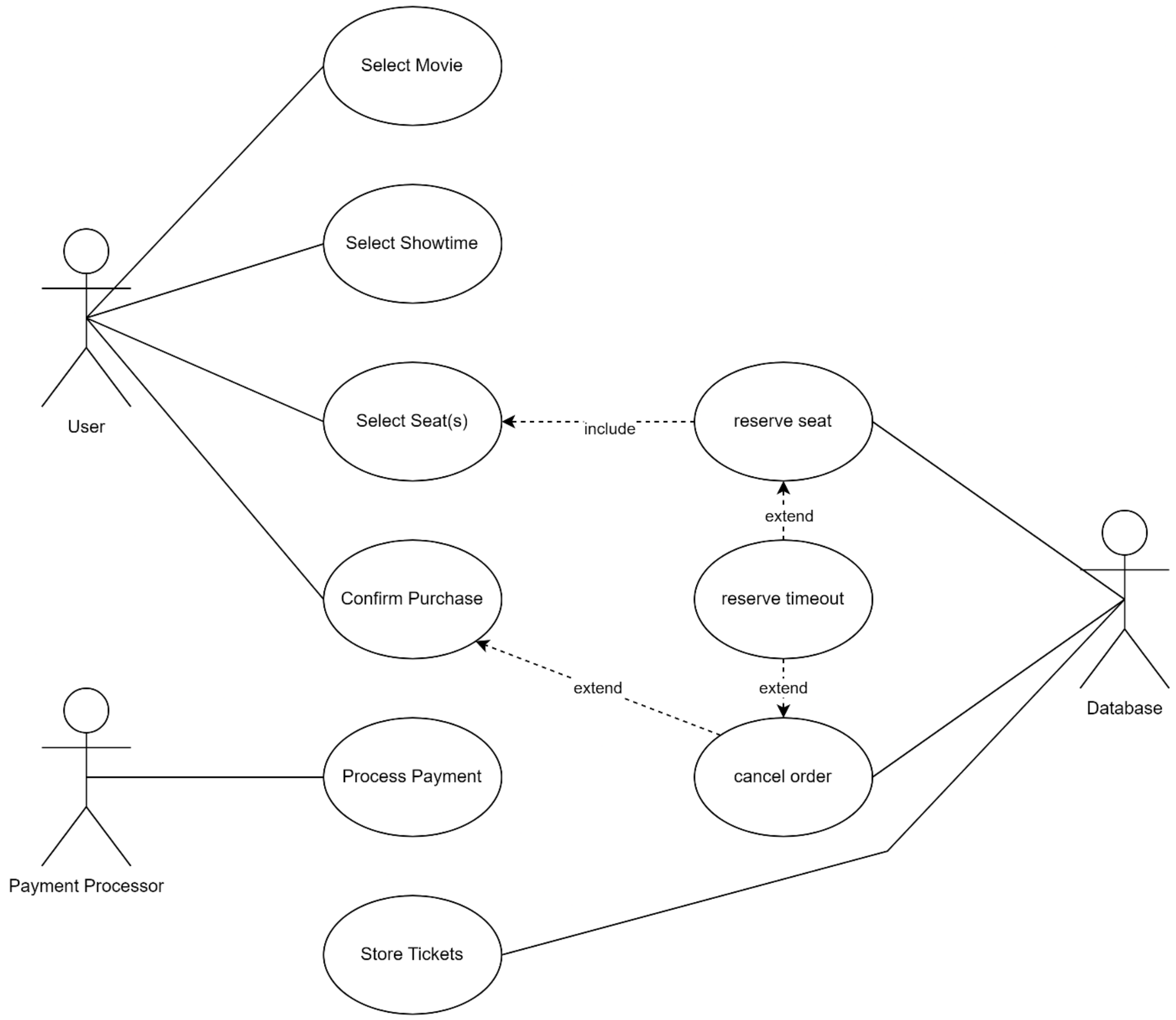
3.2 Functional Requirements

3.2.1 The website needs to be able to handle at least 5000 people at once.
3.2.2 The website needs to be run in a web browser (not as an app).
3.2.3 The website must block bots who are trying to buy a lot of tickets to high-demand movies.
3.2.4 The website must interface with the database of showtimes and tickets available.
3.2.5 The website needs to support administrator mode.
3.2.6 The website needs to support different discount tickets.
3.2.7 The website limits tickets to available seating.
3.2.8 The website allows specific seat selection.

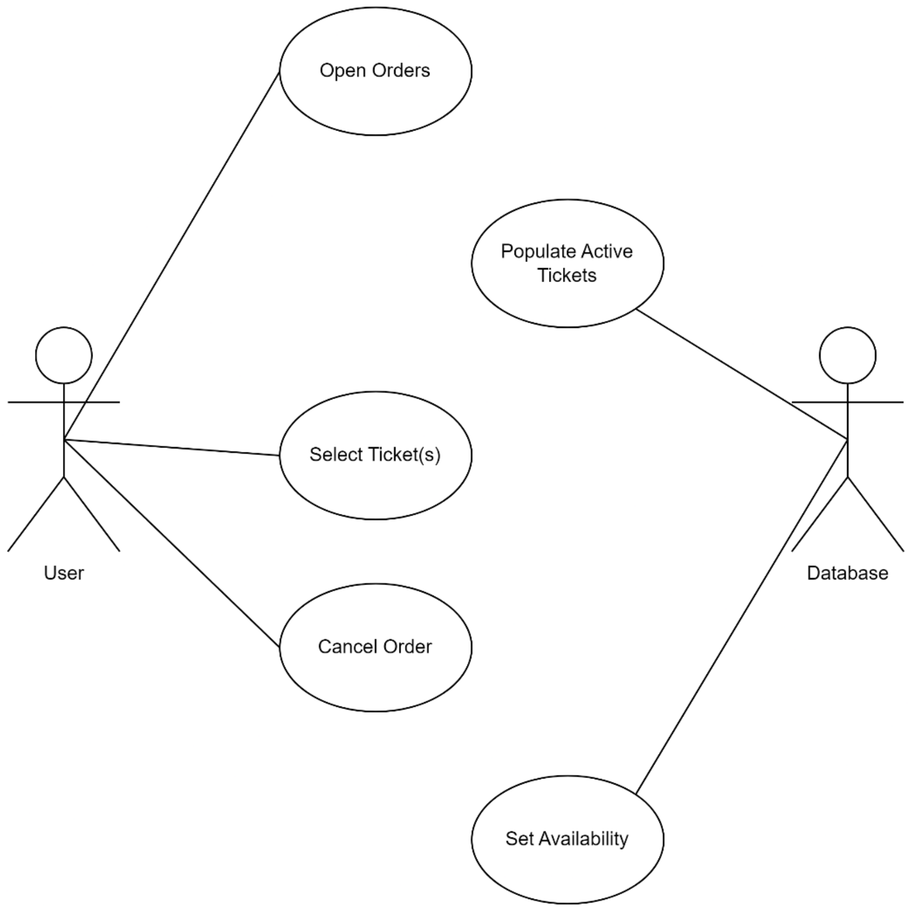
3.2.9 The website must allow for cancelling tickets.

3.3 Use Cases

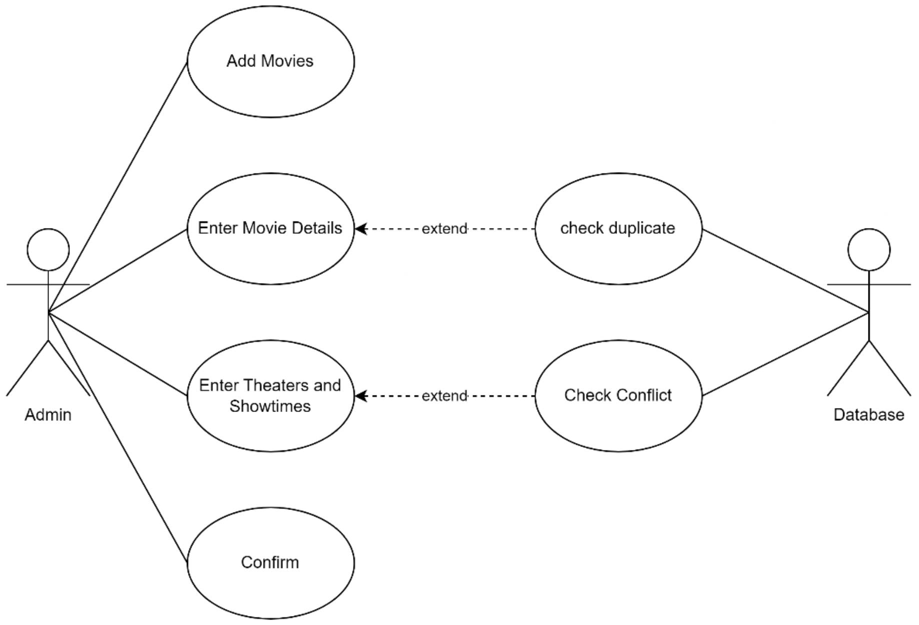
3.3.1 Happy Path



3.3.2 Cancel Tickets



3.3.3 Add Movie(s)



3.4 Non-Functional Requirements

3.4.1 Performance

New pages should load <0.25s

3.4.2 Availability

The website must always be reachable with an allowance of 4 hours a month during closing hours for maintenance.

3.4.3 Security

The website will utilize TLS.

3.4.4 Maintainability

The website shall be maintained by the client and a representative may be dispatched in case further assistance is required.
Maintenance may be scheduled by the client at their discretion.

3.4.5 Portability

The website will be put on a docker that shall allow for easily changing between existing website infrastructures.

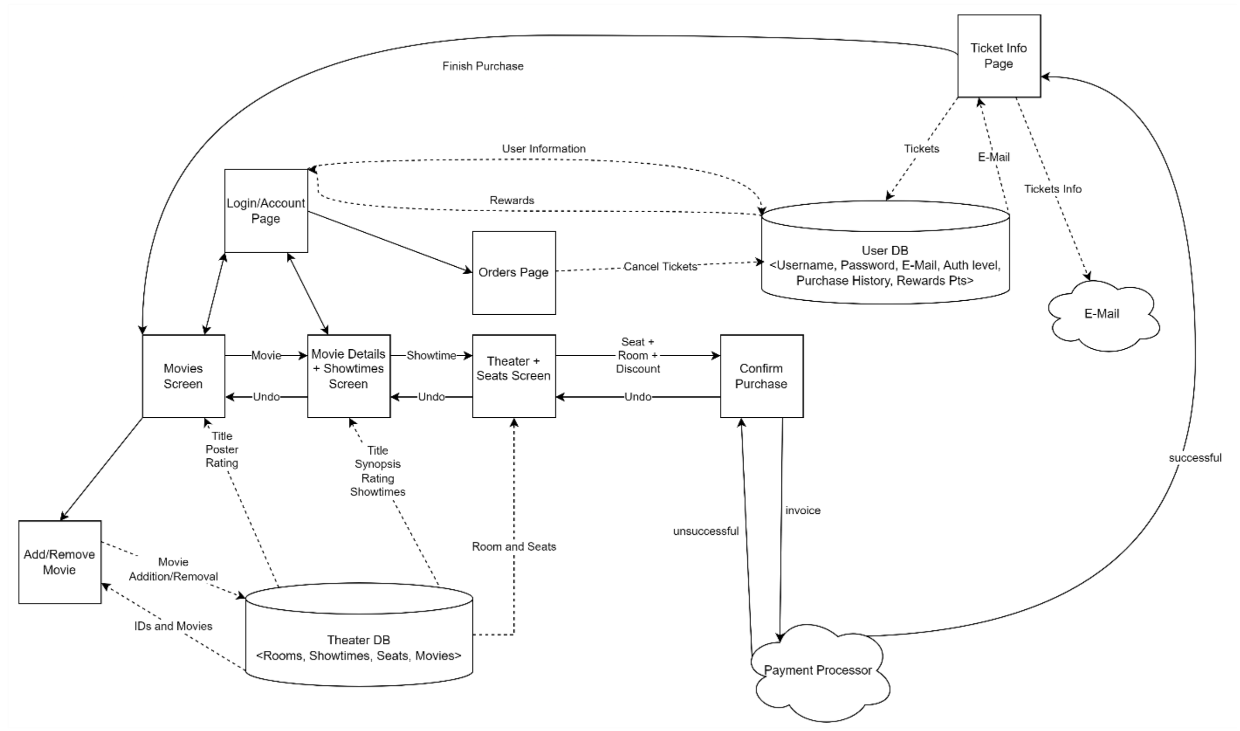
3.5 Inverse Requirements

3.5.1 The website will not need a customer feedback system.
3.5.2 The website will not need to be able to scrape online review sites.

4. Analysis Models

This system will be a ticketing site for a local theater with a few rooms. Each room will be numbered and have a set number of seats that can be sold and will be showing specific movies at different showtimes throughout the day. The site will allow users to log in and choose their movie, showtime, and seats. Purchases will be collected and tracked in a database both to have a record of the tickets purchased and for potentially offering certain users discounts.

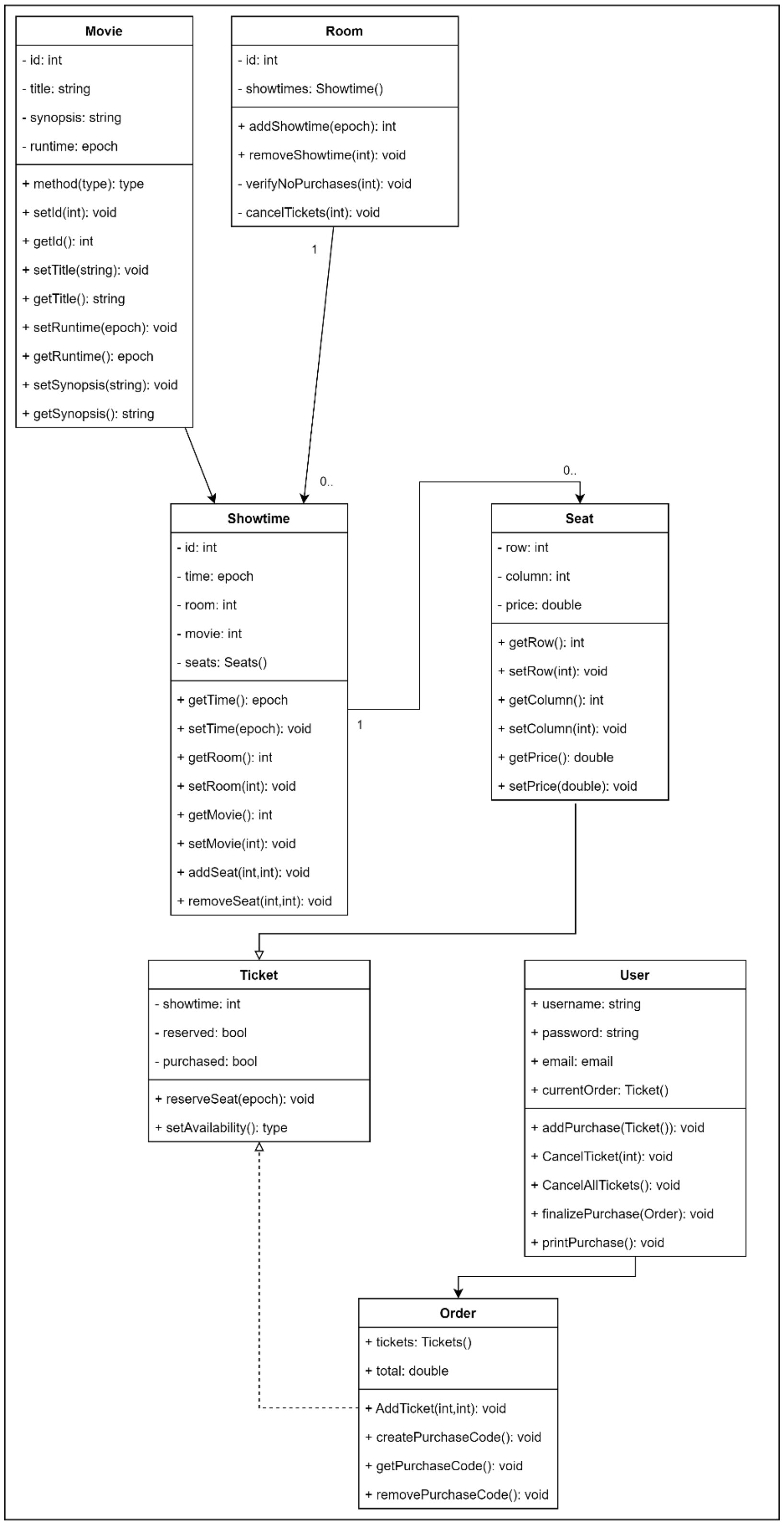
4.1 SWA Diagram



The movie page will be the first page of the site that displays the movies on selection that may be clicked on. Clicking on a movie takes one to a page showing the details, such as synopsis and rating, and showtimes that may be selected to proceed to the next page. The theater room and seats page will contain the available options for purchase that may be selected before finally proceeding to the confirmation page. The confirmation page will display the current choices and the total, which will be handled by a third-party payment processor. The tickets page will display the finalized information, which will be stored in the purchase history of the user.

The diagram has been updated to show that returning to previous pages will safely allow one to undo a selection. An administrator will be able to add or remove movies from an option in the main screen available only to administrators. Also, the login page turns into an account page for users logged in, with an extra page allowing users to cancel active tickets. Furthermore, the number of databases went from three down to two, as movies and showtimes could be in the same database as part of two different entities.

4.2 UML Class Diagram



4.2.1 Description of Classes

- Room: Room class holds and controls the showtimes for each individual show room.
- Showtime: Showtime class consists of the movie, the seats which make up the show and the time.
- Seat: Seat class will hold the row and column, and the price of the seat.
- Movie: The movie class holds a movie object that can be independently called by the showtime class to get relevant information.
- Ticket: The ticket object will be a child object inheriting from the seat class. It will control the reservation and purchase of seats during specific times.
- User: The user class creates and holds relevant user information and calls the ticket class when the user purchases a ticket.

4.2.2 Description of Attributes

- Room: Room will hold the room number, and an array of the showtime objects for that day.
- Showtime: Showtime will hold the time for the object, the movie being shown at that time, and an array of the seats in that room.
- Seat: Holds the row, column, and price.
- Movie: Holds the title, and synopsis as strings. The runtime will be stored as an epoch.
- Ticket: Holds the information that the user will be shown once they receive the ticket. The object will also control the availability of a seat whether by purchase or temporary reservation.
- User: Holds the sign-up information for the user that will be used to distinctly identify each user.

4.2.3 Description of Operations

Showtime:
- setMovie: Takes an integer value that represents the id of the movie. To set which movie is being shown at that time.
- makeTicket(): Takes no parameter and creates a ticket object assigning the time, movie and seat.

Movie:
- Has getter and setter methods for each of its attributes. The ID takes the integer value, the rest of the attributes take String values.

Seat:
- Setter method that takes a Boolean value for the availability of the seat.

Ticket:
- getFreeSeats(): Returns an array of available seats.
- setMovie(int): Takes an integer value and sets the current movie according to the integer ID value provided.
- setSeat(int, int): Takes two integer values that represent the column and the row of the seat.

User:
- addPurchaseTicket(Ticket<>): Takes an array of tickets consisting of all the tickets in a purchase. Adds them to the purchase database.

4.3 Development Plan and Timeline

Bryan: Title page, Description of operations
https://github.com/OMendez2/CS-250-Ticket-Pulling-System/blob/main/image.png

Jake: System description, Description of classes
https://github.com/OMendez2/CS-250-Ticket-Pulling-System/blob/main/System%20Design%20Specification_commit.pdf

Osvaldo: Architectural diagram, UML class diagram, description of attributes
https://github.com/OMendez2/CS-250-Ticket-Pulling-System/blob/main/System%20Design%20Specification.pdf

4.4 Test Plan

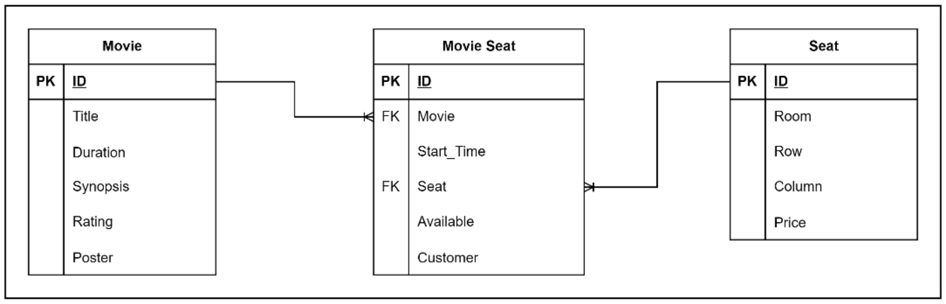
Reference the spreadsheet provided on github in the following link.

4.5 Data Management Strategy

4.5.1 General Strategy

Our system will be using 2 databases for storing and managing our data. The split can allow for securing sensitive user information in another more focused database. The system will be using a SQL table since the system is simple without very simple relationships. The system may scale somewhat but the use of SWL as opposed to NoSQL means that scaling will be harder. This may work for our simple system, but future development or expansion may require redesign. We can change our system to NoSQL, but that would trade scalability for performance. We considered this option, but we found that using SQL for our rigid and simple datatypes is more useful.

4.5.2 Theater Database



Movies

ID	Title	Duration	Synopsis	Rating	Poster
001	Men in White	2:00:13	Two agents...	4	MiW.jpg
002	Objective Possible 5	3:15:37	One man must...	5	OP5.jpg

Seats

ID	Room	Row	Column	Price

001	1	1	1	$12.00
002	1	1	2	$12.00

Screening Seats

ID	Title	Start_Time	Room	Row	Column	Available	Customer
045	Men in White	10:00:00	1	1	2	FALSE	John Smith
049	Objective Possible 5	15:00:00	3	5	3	TRUE	

Overview

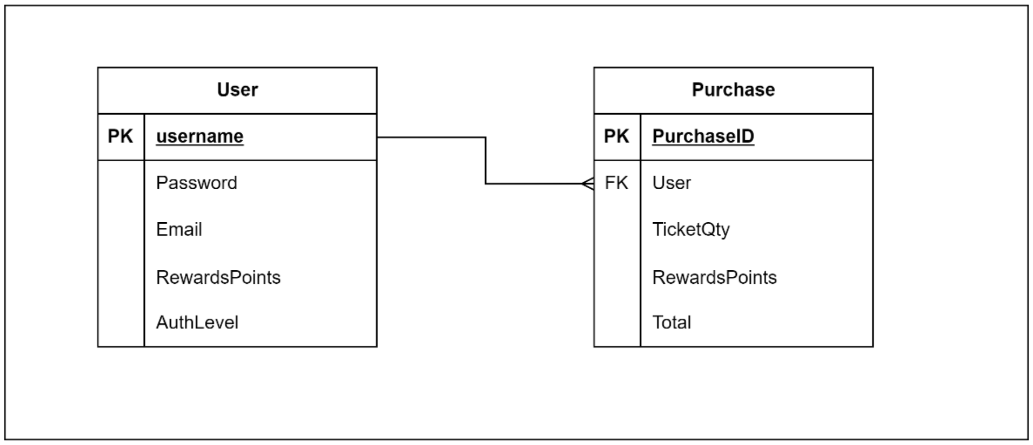
The table is based around a seat during a specific screening. Movies and Seats are their own entities, and screenings are a combination of a movie, time and a seat. Theater rooms are not entities in this model. Another model where rooms are treated as an entity with possible additional attributes may be helpful for scalability, though that could risk complicating the system.

Using integers, specific ID's and data values are mainly used to simplify each theatre and seat operations. Using SQL's vertical structure allows for us to keep all data on a single server, compared to NoSQL's multiple server setup. SQL allows for structuring of data. Our data will not likely change, thus we can settle for some rigidity.

Cons

The simple relationship means that the database is compact but does not have that much room to scale all that well, as further attributes could be completely unrelated to our current entities. If we were to add "Popcorn Spills" as an attribute, that would be difficult to incorporate into our current SQL table, as there is not a specific spot for that attribute.

4.5.3 User Database



Users

Username	Password	Email	Rewards_Points	Authorization
Jack_Black22	sOR01word!	blackjack@yahoo.com	0	Admin

johnDoe_1	keyword*!	johndoe@gmail.com	381	User

Overview

The tables are split between users and their purchases in a simple one-to-many relationship. It may be productive to consider a third entity specifically for rewards points and their benefits. The tables are simple and compact. This will be more space efficient, and there will be less to encrypt. However, rewards may be restrictive without more room for expansion by making them their own entity. The current design focuses on simplicity and efficiency with less focus on future expansion which may require a redesign sooner rather than later.

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.

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