Theater Ticketing System Project

Software Requirements Specification

Version <4.0>

03/27/2024

Group #16
Kaylie Pham
Aditya Bhagat
Ryan Hanna

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

Revision History

Date	Description	Author	Comments
02/14/2024	Version <1.0>	Group #16	Software descriptions (Sections 1-3)
02/28/2024	Version <2.0>	Group #16	Software Design Specification (Section 4)
03/13/2024	Version <3.0>	Group #16	Verification Test Plan & Test Cases (Section 5)
03/27/2024	Version <4.0>	Group #16	Architecture Design with Data Management (Section 6)

Document Approval

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

Signature	Printed Name	Title	Date
	Kaylie Pham, Aditya Bhagat, & Ryan Hanna	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

This Software Requirements Specification (SRS) document defines and lists the functional and nonfunctional requirements for the development of a Theater Ticketing System. This document is intended for project managers, stakeholders, software engineers, and software developers involved in the design, development, and implementation of the Theater Ticketing System.

1.1 Purpose

The purpose of this document is to provide a detailed specification of the requirements for the development of a Theater Ticketing System. This system aims to assist and simplify ticket purchasing for customers. This system allows customers to either purchase in person at the theater through a digital kiosk or purchase from an online website.

1.2 Scope

This document identifies the software product, its characteristics (functional and nonfunctional) and constraints. It describes the application of the software, including its benefits, objectives, and goals. The Theater Ticketing System will execute ticket purchasing for customers, whether in person at theater kiosks or through an online website.

1.3 Definitions, Acronyms, and Abbreviations

- *SRS: Software requirements specification*
- *DBMS: Database Management System*

1.4 References

The references are:

- "Software Requirements Specification document with example" by Ravi Bandakkanava, May 8, 2023 https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database
- "How to Write a Software Requirements Specification (SRS Document)" by Gerhard Krüger and Charles Lane, January 17, 2023 https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document
- "IEEE Recommended Practice for Software Requirements Specifications" by the IEEE Computer Society, October 20, 1998

1.5 Overview

This document contains specific requirements and information for the Theater Ticketing System, including general description, external interface requirements, functional requirements, non-functional requirements, design constraints, and logical database requirements.

2. General Description

2.1 Product Perspective

The Theater Ticketing System will interact with databases of showtimes, available tickets, and user accounts. The application operates independently from other software/hardware which means it will be a standalone web-based application.

2.2 Product Functions

The system will perform many different functions, the main being ticket purchasing and interfacing with databases. For security and management purposes, other functions include blocking bots from buying tickets in high-demand, managing through administrator mode, and collecting customer feedback.

2.3 User Characteristics

Users of the system include customers purchasing theater tickets, software engineers maintaining the system and software, and administrators that manage and control the system.

2.4 General Constraints

Constraints include handling at least 1000 users at once, preventing scalping/reselling and selling unique, non-replicable tickets, supporting multiple languages such as English, Spanish, and Swedish, and following discount policies.

2.5 Assumptions and Dependencies

Assumptions include the availability of hardware and software needed for this ticketing system to work and operate as intended.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- *The system will have user interfaces for ticket purchasing, account management, and feedback submission.*
- *Language options include English, Spanish, and Swedish*.

3.1.2 Hardware Interfaces

- *For web browsing, the system will interface with standard hardware components.*

3.1.3 Software Interfaces

- *The system will interact with databases for tickets, showtimes, and customer accounts.*

3.1.4 Communications Interfaces

- *The system may use email for ticket delivery and communication.*

3.2 Functional Requirements

3.2.1 Ticket Purchasing

3.2.1.1 Introduction

- *Users can access the system via a digital kiosk in person or online through a web browser, where they will be presented with an interface that displays showtimes and ticket availability for either a deluxe or regular theaters depending on the theater's location.*

3.2.1.2 Inputs

- *The user selects a desired showtime and specifies the number of tickets they would like to purchase. If the theater is a deluxe theater, the user will then be prompted to select what specific seats they would like to claim. To complete the transaction, the user chooses an accepted payment method and whether they would like their ticket to be provided via email or in person at the box office.*

3.2.1.3 Processing

- *The system will process the payment method that the user inputs and check error handling for any issues that may have occurred during the input process.*

3.2.1.4 Outputs

- *The user receives confirmation for their purchase.*
- *The system will ask for feedback by displaying a range of smiley faces for the user to select.*

3.2.1.5 Error Handling

- *Users can purchase tickets for specific showtimes.*
- *A maximum of 20 tickets can be purchased in a single transaction.*
- *Tickets are available for purchase two weeks prior to showtime and up to 10 minutes after the show starts.*
- *Once the purchasing process begins, the system will initiate a timer with a 5 minute countdown. If the user does not complete the transaction within this time frame, their ticket will be abandoned.*

3.2.2 Bot Blocking

- *The system must be secure to protect users and to prevent automated bot purchases. Tickets purchased must be unique and non-replicable.*

3.2.3 Administrator Mode

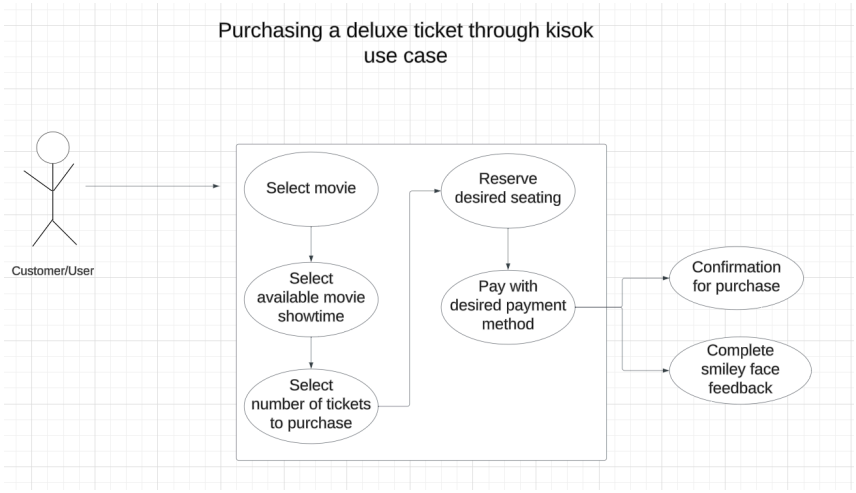
- *Administrators can access a secure mode for managing and controlling the ticketing system such as overriding errors made by users and setting showtimes and theaters.*

3.2.4 Feedback System

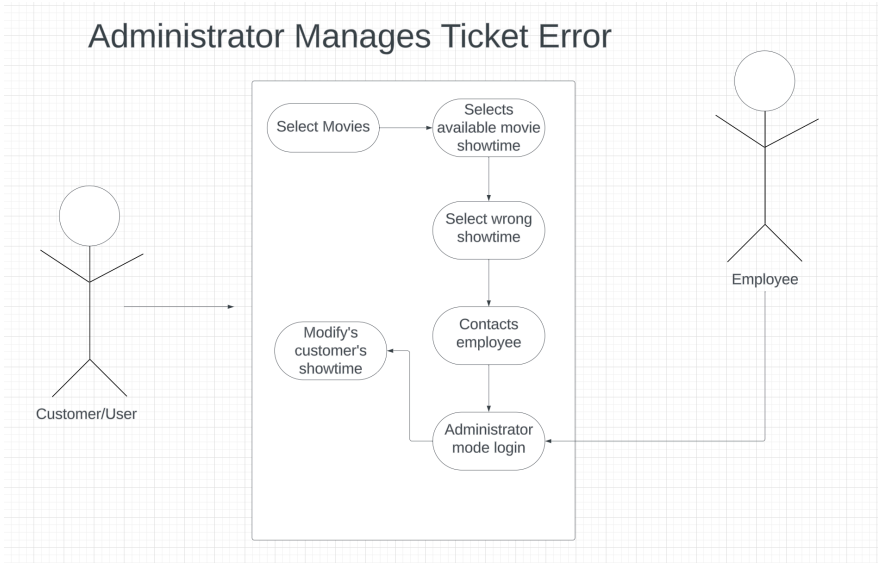
- *Customers can provide feedback on their ticketing experience. After each customer purchases tickets, whether via kiosk in-person or via the online website, smiley faces will be displayed for people to select to represent their evaluation of the theater.*

3.3 Use Cases

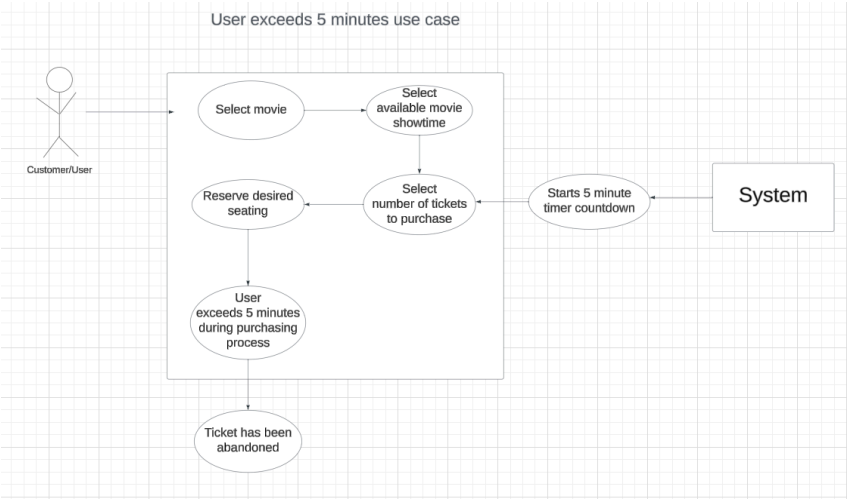
3.3.1 Use Case #1: User purchases deluxe tickets



3.3.2 Use Case #2: Administrator Manages Ticket Error



3.3.3 Use Case #3: User Exceeds 5 Minutes



3.4 Classes / Objects

3.4.1 User Class

3.4.1.1 Attributes

- *User account name and password*
- *Can register loyalty accounts to store payment/purchase history, personal information, and loyalty points*

3.4.1.2 Functions

- *Purchasing tickets by selecting showtime, theater, and number of tickets*
- *Pay with credit card, paypal, bitcoin*
- *Reserve seats*
- *Can return or exchange tickets before showtime*

3.4.2 Administrator Class

3.4.2.1 Attributes

- *Admin account name and password*
- *Access Control*

3.4.2.2 Functions

Overrides any errors made by users
Set showtimes and prices
Change theaters as needed
User management
Security Management
Financial management
Customer/Collaboration communication

3.5 Non-Functional Requirements

3.5.1 Performance: *The system must handle at least 1000 concurrent users efficiently and the system must be easy to use by customers. The system must also be secure and protect customer information. Tickets distributed through the system must be unique and non-replicable.*

3.5.2 Reliability: *In cases where there are a high volume of requests of more than 1000 people for a single showing at once, a queuing system will be used so that customers can wait in line to purchase their tickets according to when they attempted their purchases.*

3.5.3 Usability: *The user interface must be simple, straightforward, and easy to use by customers purchasing tickets. Customers will receive unique, non-replicable ticket(s).*

3.5.4 Security: *The user interface must be secure to prevent data breaches and unapproved access to customers' accounts and information. To protect customers and the potential for unauthorized purchases, only one device can log into a user account at once. Signing into a user account using a different device will log out the previous device.*

3.5.5 Maintainability: T*he system will keep a systemwide daily log of ticket purchases and will have an administrator mode that is available. The administrator mode will override any errors made by customers and will set theaters, showtimes, and tickets available for purchase.*

3.5.6 Portability: *The entire theater ticketing system uses a single database that is divided by individual theaters allowing the system to function smoothly across different systems and individual theaters.*

3.6 Inverse Requirements

The system has various inverse requirements. The user has 5 minutes after starting the ticket purchasing process to finish purchasing their ticket before the ticket is abandoned. The range in which users may purchase tickets is a maximum of two weeks in advance and up to 10 minutes after showtime. Users may purchase up to a maximum of 20 tickets at once. User accounts can be logged into only one account at a time and can purchase using credit card, paypal, or bitcoin. There are also only assigned seats for deluxe theaters.

3.7 Design Constraints

- *The system must be web-based (not app-based) and accessed through standard web browsers.*
- *Ticket validation is limited to the San Diego theater chain (20 different theaters) and Pacific Time Zone.*

3.8 Logical Database Requirements

The system will use a database for storing customer information, user accounts, showtimes, tickets, reviews and critics quotes of the movies, and feedback data.

3.9 Other Requirements

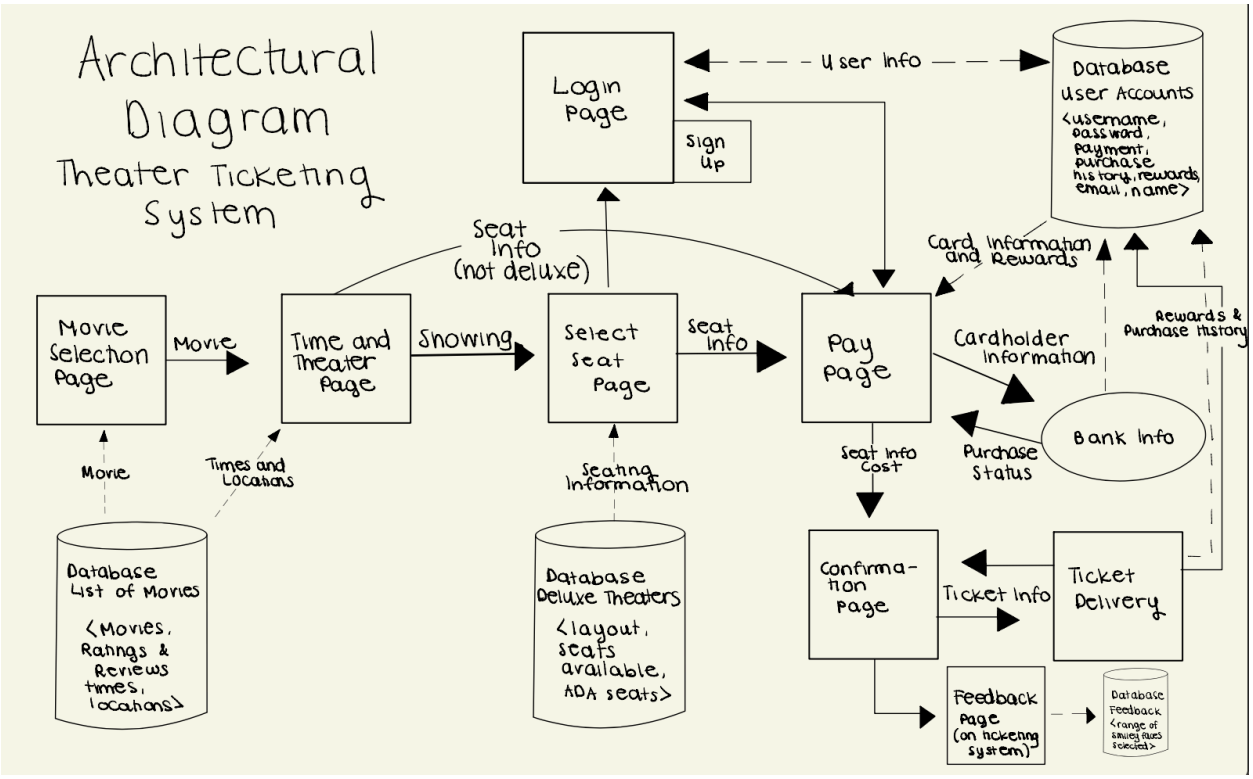
The system must offer discounts for students, veterans, and military during weekdays. The system must also allow users to register for loyalty accounts that store purchase history, personal information, payment information, and loyalty points. Users with accounts may return or exchange their theater tickets before the showing.

4. Software Design Specification

The Software Design Specification describes the software organization, development plan, and architectural design of the theater ticketing system. The user class and administrator class have access to different components of the system. While the user is able to purchase tickets, make an account, and select movies and showings of their choice, the administrator has access to all databases and information submitted on the web application and may make changes to the application. The Architectural Diagram focuses on the components, user interfaces, and connections between interfaces. The UML Diagram focuses on more detailed information about the software structure and behavior.

4.1 Software Architecture Overview

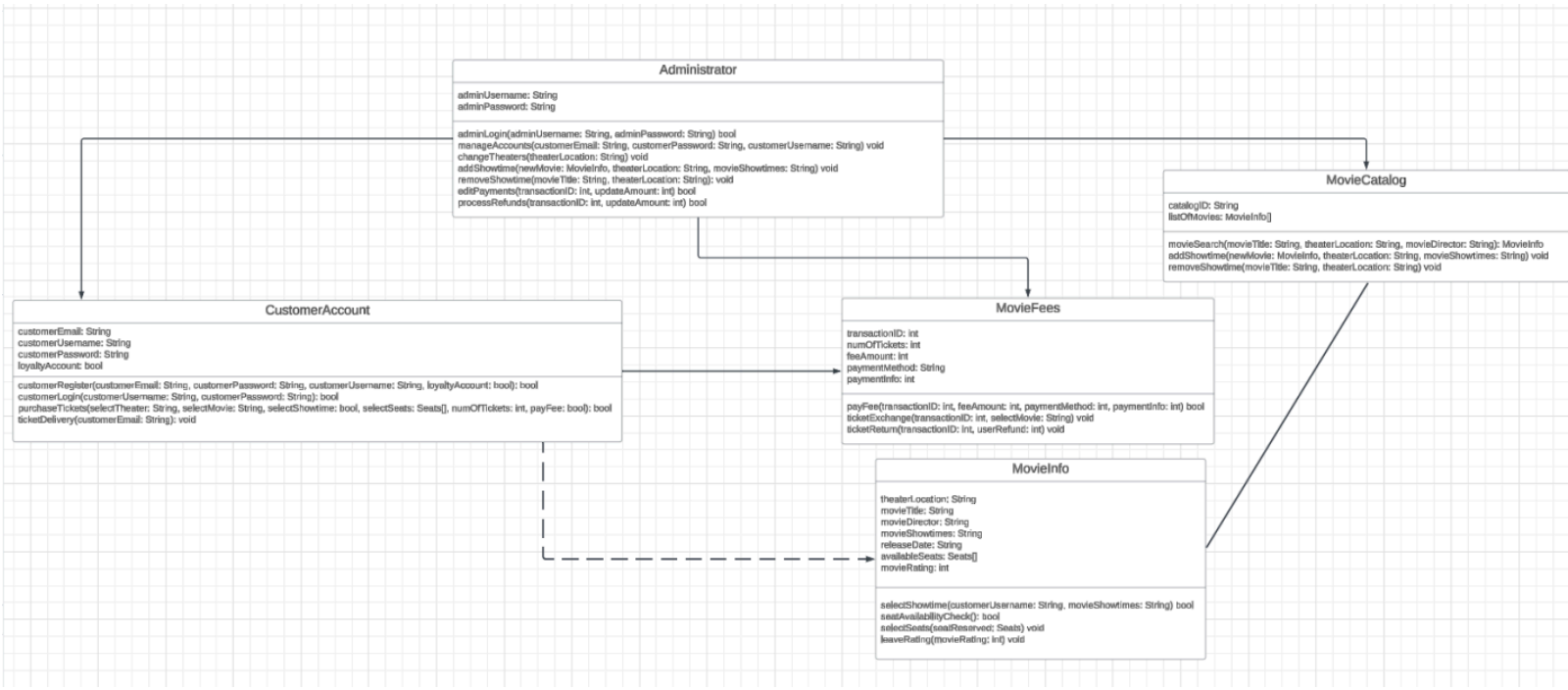
4.1.1 Architectural Diagram



4.1.2 Explanation of the Architectural Diagram

This architectural design serves as a blueprint for the Theater Ticketing System. It outlines the key elements of the system such as its components, interactions, and structure. Its components include databases, functions, and user interfaces. When users first open the web application, they are directed to the "Movie Selection Page" which includes information from the "List of Movies" database. The "List of Movies" database includes information about movies, their ratings and reviews, times, and locations. Users can then select a movie of their choice. This redirects them to the "Time and Theater Page" which contains information about the times and locations from the "List of Movies" database. On this page, users select an available showing with a time and location of the movie of their choice. If the user selects a deluxe theater, they are directed to the "Select Seat Page" because users are able to select their seat(s) if they select a showing in one of the deluxe theaters. The "Select Seat Page" contains seating information from the "Deluxe Theaters" database which includes information about the layout, seats available, and ADA (Americans with Disabilities Act) seating. Once the user selects their seating information, they are redirected to the "Pay Page." If the user does not select a showing from the deluxe theaters, they are not able to select their seats and are directly brought to the "Pay Page" from the "Time and Theater Page." From the "Pay Page," users are given the option to log in with an existing account or sign up with an account for the web application. User information from the log-in/sign-up page is stored in the "User Accounts" database which contains the user's username, password, payment information, purchase history, rewards, email, and name. Once the user is logged in, they are brought to the "Pay Page" to finish their transaction. If the "User Accounts" database does not already include the user's payment and bank information, the user is redirected to the "Bank Info" page to enter their card information. Once the payment information has been entered and the purchase status is confirmed, the user is once again directed to the "Pay Page" to finish their transaction. Once possible rewards, card information, cost details, and theater showing information are confirmed by the user, the user is directed to the "Confirmation Page" where they may open their ticket delivery information on the "Ticket Delivery Page" and/or open the "Feedback Page" where they may rate the ticketing system by picking from a range of smiley faces. The user feedback information is stored in the "Feedback" database which contains a range of smiley faces selected by users. Information from the "Ticket Delivery Page" such as the rewards earned and purchase history is stored in the "User Accounts" database. Finally, the user has successfully purchased their theater tickets. The administrator can access information from all databases and change information such as available movies, times, locations, prices, seatings, rewards, discounts, and more.

4.1.3 UML Diagram



4.1.4 Explanation of UML Class Diagram

The system's architecture is composed of multiple classes that manage the operations of a movie theater which has been displayed in the UML diagram above. The Administrator class is the backend management of the system, consisting of attributes such as adminUsername and adminPassword that allow for secure access control to managing aspects of the system. These administrator privileges are included as operations in the UML Diagram such as being able to manage customer accounts (manageAccounts), theater/movie information (changeTheaters, addShowtime, removeShowtime), and edit payments (editPayments, processRefunds).

The CustomerAccount class allows user interaction with the system, holding important information within attributes such as customerEmail, customerUsername, customerPassword, and a boolean for loyaltyAccount to denote whether the user would like to join the loyalty program. Allowing the system to store the aforementioned information allows for user account creation and sign-ins through the use of the customerRegister and customerLogin operations respectively. Along with this, this class is also responsible for letting users purchase tickets (purchaseTickets) and allows a user to set a delivery preference for receiving their ticket (ticketDelivery).

The MovieCatalog class serves as a directory of movies that are available at a designated theater, each uniquely identified by a catalogID. This class lists movies through an array of MovieInfo instances and includes functionality to search (movieSearch), add (addShowtime), and

remove showtimes (removeShowtime), thereby maintaining an up-to-date and accessible catalog of movies.

MovieFees is responsible for the monetary functions of the theater's transactions. The attributes that encapsulate these functions are transactionID, numOfTickets, feeAmount, paymentMethod, and paymentInfo which are all essentially steps that calculate what a person will owe before purchasing their tickets. The following operations use these attributes to process payments (payFee), allow for ticket exchanges (ticketExchange), and returns (ticketReturn), ensuring that users have an accurate financial experience within the system.

Within the MovieInfo class is a repository of detailed movie information listed as attributes such as theaterLocation, movieTitle, movieDirector, movieShowtimes, releaseDate, availableSeats, and movieRating designed to easily categorize movies by the most important information about them. This class is designed to facilitate customer decisions and engagement through a set of operations that allow them to select showtimes (selectShowtime), see proper seat availability (seatAvailabilityCheck), reserve specific seats (selectSeats), and provide feedback by rating the movie (leaveRating).

4.2 Development Plan and Timeline

As a team, we first analyzed the information provided about the theater ticketing system and described the system in more detail for the first part of this assignment for parts 1-3. We provided the system's purpose, overview, functions, characteristics, constraints, interfaces, functional and non-functional requirements, classes, and use cases. This was completed by February 14, 2024. Next, we completed the "Software Design Specification" in part 4 of the specification. As a team, we used our knowledge of the system to create the system's architectural diagram and UML diagram. These diagrams display the system's components, interactions, and functions in more detail. The architectural diagram displays different user interfaces and databases, and the UML diagram displays the classes, functions, and behaviors. This was completed by February 28, 2024. Next, we completed the test plan. As a group, we developed a verification test plan including target features/functions, methods of testing the system, and how to test important components of the system using test cases. In addition, we created 10 test cases in which at least three are unit testing, three are system testing, and three are functional testing. This was completed by March 13, 2024.

5. Test Cases

5.1 Verification Test Plan

This verification test plan strives to test the condition and functionality of the theater ticketing system. The plan identifies features and functions to be tested, methods of testing, and how test cases will cover the features. The plan also addresses and considers possible failures and expected outcomes. This verification test plan should verify that the system operates as expected.

5.1.1 Features to be tested

The listed features will be the targets/focus of the verification test plan and test cases:

- *Administrator's Ability to Manage the System*
- *Security Management*
- *Customer Ticket Purchasing*
- *Bot Blocking*
- *Processing Refunds/Ticket Exchanges*
- *Testing Performance Under Load*

5.1.2 Administrator's Ability to Manage the System

We are going to perform specific unit testing on the administrator class in order to verify the class's ability to manage customer accounts, theater/movie information, and payments. This includes administrators attempting to edit user accounts, add, remove, or edit movie showtimes/theater locations, and process refunds or change payment amounts for customers.

- *Testing for Theater/Movie Management:*

1. *Using the Administrator interface, add a new movie showtime and theater location to the catalog*
2. *Edit the previously created movie showtime by changing the showtime/theater to something different*
3. *Delete the showtime/theater entirely*

5.1.3 Security Management

The goal of testing this feature is to verify that the web application is secure and the users' accounts and information are protected. Each account can only be logged into on one device. If a user signs into their account on another device, the previous device will automatically log out of the user's account.

- *Test Steps for Verifying Security Measures:*

1. *Log into multiple devices at once to verify that one account can be logged in at once*
2. *Attempt unauthorized access to account and personal information*

5.1.4 Customer Ticket Purchasing

The goal of testing the customer ticket purchasing process is to verify that users can purchase tickets at a time, location, and showing of their choice. Users may purchase up to a maximum of 20 tickets at once and have 5 minutes to complete their transaction before the application closes and the ticket is abandoned. Users may also purchase tickets up to two weeks in advance and up to 10 minutes after the movie starts airing.

- *Test Steps for Purchasing Tickets:*

a) *1. Log into an account*
   2. Select a movie, time, location, and appropriate number of tickets (under 20)
   3. Select seat(s) if deluxe theater is chosen
   4. Enter payment information and purchase tickets
   5. Verify ticket purchase is successful and verify delivery information

6. Verify customer feedback feature by selecting a smiley face

b) *1. Attempt to purchase more than 20 tickets*
2. Attempt to purchase tickets past the 5 minute mark
3. Attempt to purchase tickets more than 2 weeks in advance
4. Attempt to purchase tickets more than 10 minutes after showing begins
5. Verify that ticket purchase is unsuccessful because user exceeded system's limitations

5.1.5 Bot Blocking

The goal of testing the bot blocking feature is to verify that the system can identify and prevent automated bot purchases of tickets in high-demand. Tickets also must be unique and must not be replicable. Managing bot purchases through bot blocking is important because they can lead to traffic load and can slow down the server.

- *Test Steps for Blocking Automated Bot Purchases:*

1. *Attempt to rapidly purchase tickets*
2. *Verify that the system detected rapid bot-like purchases and blocked ticket purchasing*

5.1.6 Processing Refunds/Ticket Exchanges

Our purpose in testing this feature of processing refunds/ticket exchanges is to make sure an important aspect of our system that is integral in customer satisfaction works. Ticket exchange should work when a customer already has bought a ticket and would like to swap it out for another one. As for refunds, if a customer has bought a ticket and would like their money back it would then be returned to them.

- *Testing for Ticket Exchanges:*

1. *Successfully purchase a ticket on a test customer account*
2. *Attempt to exchange the ticket for another applicable ticket of a different movie*
3. *Check if the customer account has the newly requested ticket and if the previous ticket is available for others to purchase/exchange*
4. *Submit refund request for the ticket and see if funds are transferred back to the customer*

5.1.7 Testing Performance Under Load

An important non-functional requirement of the system is that it must be able to handle at least 1000 concurrent users efficiently and if the system is under too much load a queue system will be put in place for users ensuring the system maintains stability.

- *Testing System Under Load:*

1. *Run two tests one with a 500 customer accounts and the other with 1000+ customer accounts*
2. *Have all the accounts try to purchase tickets at the same time*
3. *Test the system's performance and increase the number of accounts to test the system's limits*
4. *Implement the queue system to see if it is working properly*
5. *Test the system's performance after the queue has been implemented*

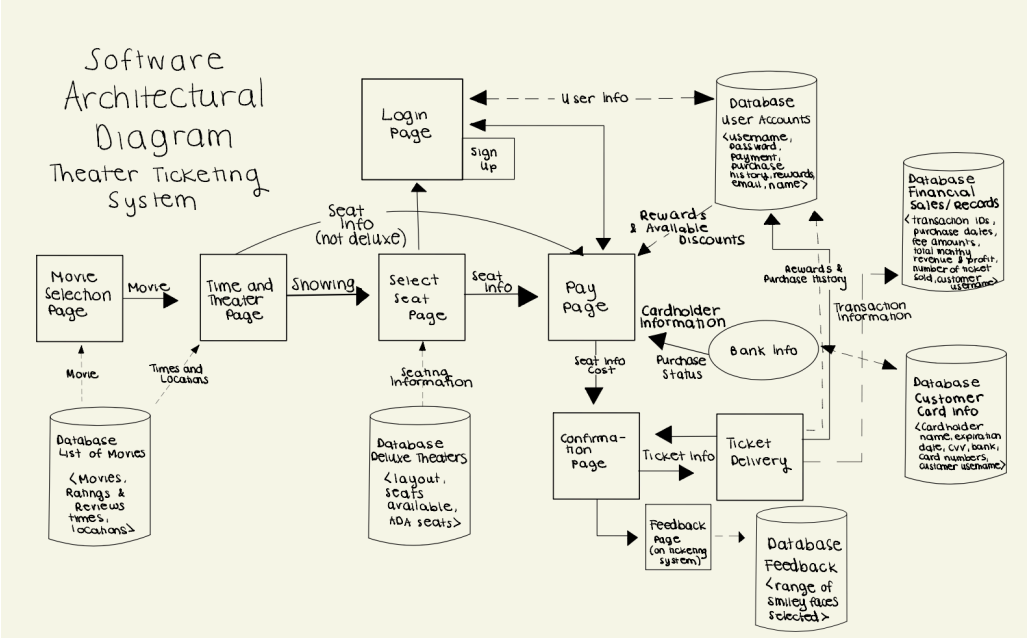
5.2 Test Case Samples

Test Cases (Theatre Ticketing System)									
TestCaseId	**Component**	**Priority**	**Description/Test Summary**	**Pre-requisites**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**	**Test Executed By**
UNIT_TC_001	**User_Account_Module**		Test the login functionality of the User Class.	User account exists in the database.	1. Enter valid email. 2. Enter correct password. 3. Click on the login button.	User should be successfully logged in.	User successfully logs into the system.	Pass	
UNIT_TC_002	**Theatre_Catalog_Module**		Test the browse theatre catalog functionality of the User Class.	User is logged in.	1. Navigate to the time/shows section.	User should be able to view the catalog.	User can view the catalog with available shows.	Pass	
UNIT_TC_003	**Administrator_Mode_Module**		Test if administrators can access and manage the ticketing system effectively.	Ensure admin credentials are valid and administrator mode is accessible.	1. Log in with administrator credentials.	Administrators should be able to perform all designated administrative tasks without errors.	Administrators successfully managed the system without encountering issues.	Pass	
SYS_TC_001	**Bot_Blocking**		Test if the system can detect and block automated bot purchases effectively.	Ensure the bot detection module is integrated and configured properly.	1.Simulate a bot attempting to purchase tickets.	The system should identify the bot and prevent the purchase, displaying an error message.	Bot detected and purchase blocked.	Pass	
SYS_TC_002	**Payment_Module**		Test if users can choose different payment options seamlessly.	User is logged in and purchased a ticket	1. Proceed to payment during ticket purchase. 2. Select different payment methods (credit card, PayPal, Bitcoin).	Users should be able to select and use different payment methods without encountering errors.	Payment methods selected and processed successfully.	Pass	
SYS_TC_003	**Feedback_Module**		Test if users can submit feedback successfully.	User is logged in and purchased a ticket	1. Proceed to payment during ticket purchase. 2. Select different payment methods (credit card, PayPal).	Feedback submission should be successful, and users should receive confirmation.	Feedback successfully submitted and user receives confirmation.	Pass	
FUNC_TC_001	**Ticket_Purchasing_Module**		Test if users can purchase tickets seamlessly.	Ensure the ticket purchasing interface is accessible and functional.	1. Log in to the ticketing system. 2. Select a showtime and specify the number of tickets. 3. Proceed to payment and complete the transaction.	Users should receive confirmation of ticket purchase.	Users successfully purchased tickets.	Pass	
FUNC_TC_002	**Timeshow_Selection_Module**		Test if users can select showtimes accurately.	Ensure showtimes are correctly listed in the system.	1. Log in to the ticketing system. 2. Navigate to the showtime selection interface. 3. Select a showtime.	The selected showtime should match the user's choice	Users successfully selected showtimes	Pass	
FUNC_TC_003	**Seat_Reservation_Module**		Test if users can reserve seats accurately (applicable for deluxe theaters)	Ensure the system supports seat reservation for deluxe theaters.	1. Select a showtime for a deluxe theater. 2. Choose specific seats for reservation.	Selected seats should be successfully reserved and reflected in the ticket purchase confirmation.	Reserved seats confirmed in the ticket purchase.	Pass	
FUNC_TC_004	**Ticket_Management_Module**		Test if users can return or exchange tickets before showtime.	Ensure the system allows ticket returns and exchanges within the specified timeframe.	1. Return or exchange tickets before the showtime.	Users should be able to return or exchange tickets.	Ticket return or exchange processes completed successfully.	Pass	

6. Architecture Design with Data Management

6.1 Software Architecture Diagram

6.1.1 Updated Software Architecture Diagram

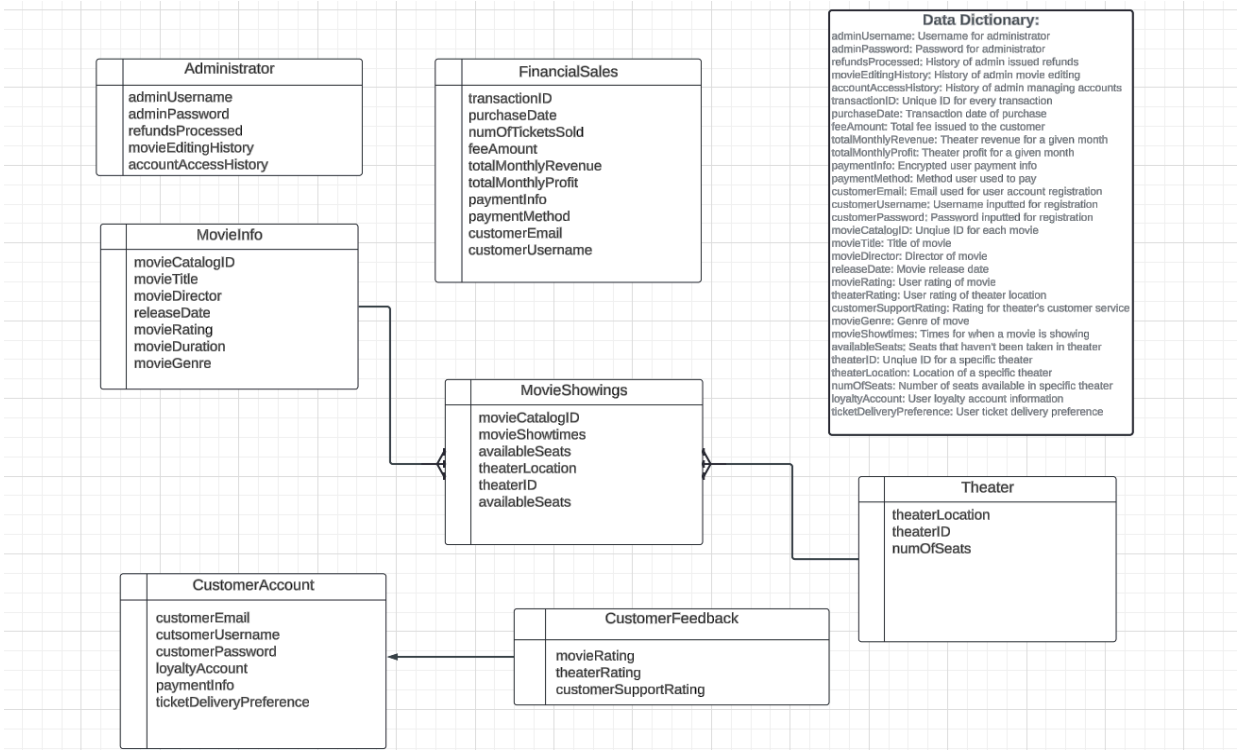


6.1.2 Architecture Diagram Modification Description

In this modified version of the Software Architecture Diagram from section 4.1.1 of this document, components were added to improve the organization of the diagram and to reflect the data management strategy developed for this system. The "Financial Sales/Records" database and the "Customer Card Information" database were added to the architecture diagram. The "Financial Sales/Records" database includes transaction IDs, purchase dates, fee amounts, total monthly revenue, total monthly profit, number of tickets sold, and purchasers' usernames. This database tracks each transaction and its fee amounts, adding it to the total monthly revenue and the total monthly profit, and is utilized by the administrator to assess business performance and monthly purchasing statistics. Information from the "Ticket Delivery" page is stored in this database. The "Customer Card Information" database includes cardholder name, card expiration date, CVV number, card numbers, and customer username. This database stores the card information of customers who have made purchases or who have entered their banking information by accessing their accounts through the website. This makes it easier for customers to make purchases following their first purchase because their card information is stored and is made a payment option on the "Pay Page" of the website. Information from the "Bank Info" page is stored in this database, and the "Bank Info" and "Pay Page" pages utilize information from this database.

6.2 Data Management Strategy

6.2.1 Entity-Relationship Diagram



6.2.2 Strategy Description

The choice of SQL for our Data Management Strategy diagram stems from its proven reliability and robustness in handling structured data with clear relationships between datasets. SQL databases are perfect for applications that need complex queries and multi-row operations/transactions because they are excellent at maintaining normalized data over numerous tables. Effective data modeling and normalization strategies are made possible by SQL databases, which also support efficient information storage, maintenance, and updating of information. The use of Entity-Relationship Diagrams (ERDs) allows for clear visualization of the database schema, helping in understanding the data structure and relationships between entities. Overall, SQL's stability, performance, and comprehensive querying capabilities make it the optimal choice for our data management needs.

6.2.3 Sample Databases

Theater Database

Theater Id	No. of Seats	Location
01	50	Delmar
02	85	Fashion Valley

Movie Database

Movie Id	Showtimes	Available Seats	Theater Id	Location
1011	9:30 pm	28	01	Delmar
1012	11 am	14	02	Fashion Valley

Movie Info Database

Movie Id	Title	Director	Release Date	Genre	Rating	Duration
1011	Spiderman	Joseph Zito	03/29/2018	Action/Comedy	8.4	2h 1m
1012	Ready Player One	Steven Spielberg	05/03/2003	Action/Thriller	7.5	2h 20m

Customer Account Database

Email	Username	Password	Loyalty Account	Payment info	Ticket Delivery Preference
xyz@gmail.com	Tom_JAzz	formovies@44	Yes		Email
abc@gmail.com	James1	formovies@44	No		Email

Customer Feedback Database

Movie Rating	Theater Rating	Customer Support Rating
7	5.5	10
8	7	9

Admin Database

Username	Password	Refunds Proccessed	Movie Update History	Account Access History
admin1	itsAdmin			
admin2	itsAdmin@34			

Financial Database

Customer Email	Transaction Id	Purchase Date	Fee Amount	Payment Info	Total Monthly Revenue	Total Monthly Profit
xyz1@gmail.com	#70611589	02/24/2024	$44		$60000	$45000
abc876@gmail.com	#87655676	11/02/2023	$60		$80000	$65000

6.2.4 Design Decisions

For the Entity-Relationship (SQL) diagram, seven databases were chosen and added to organize and split the data within the ticketing system. The seven databases are Administrator, Customer Account, FinancialSales, MovieInfo, MovieShowings, Theater, and CustomerFeedback. The data is split logically into different account access (customer and administrator) and different informational databases. For instance, the movie information such as rating, duration, and genre are listed in MovieInfo, and the showtime information such as showtimes, seats, and location are stored in MovieShowings. The Theater database then stores information about specific theaters. The FinancialSales and CustomerFeedback databases are accessed and utilized by the administrator to assess business performance and application performance, respectively.

6.2.5 Possible Alternatives and Tradeoffs

Alternatively, a noSQL database could have been used instead of a SQL database for this system. A noSQL database follows the Brewers CAP theorem which states "Consistency, Availability, and Partition tolerance." That being said, a noSQL database would be easier to implement and utilize. A noSQL database is also cheaper to scale. However, SQL databases are best for high-transaction applications, and the theater ticketing system processes many purchases and operates against account breaches and bot purchases. Overall, the SQL database handles more complex applications and has the ability to perform dynamic operations which is better fit for this system.

7. 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 to the SRS's requirements.

7.1 Sequence Diagrams

7.2 Data Flow Diagrams (DFD)

7.3 State-Transition Diagrams (STD)

8. 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