Ticket Sales Anywhere

Software Requirements Specification

v1.0.3

6/10/2024

Group 3

Lexa Rao, James Ardilla, Eli Lopez

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

GitHub: https://github.com/LexaRao/Movie-Theater/tree/main

Revision History

Date	Description	Author	Comments
5/26/2024	V1.0.0	Lexa, James, Eli	Included basic tickets sales functionality, advisement capability, and, simple UI.
5/31/2024	V1.0.1	James Ardilla	Revising based on grades
6/3/2024	V1.0.2	Lexa, James, Eli	Revising based on feedback from TA. Added second assignment section.
6/10/2024	V1.0.3	Lexa, James, Eli	Added third assignment section.

Document Approval

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

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

Table of Contents

REVISION HISTORY I

DOCUMENT APPROVAL I

1. INTRODUCTION 1

1.1 PURPOSE 1
1.2 SCOPE 1
1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 6
1.4 REFERENCES 6
1.5 OVERVIEW 8

2. GENERAL DESCRIPTION 8

2.1 PRODUCT PERSPECTIVE 8
2.2 PRODUCT FUNCTIONS 8
2.3 USER CHARACTERISTICS 9
2.4 GENERAL CONSTRAINTS 9
2.5 ASSUMPTIONS AND DEPENDENCIES 9

3. SPECIFIC REQUIREMENTS 10

3.1 EXTERNAL INTERFACE REQUIREMENTS 10
3.1.1 User Interfaces 10
3.1.3 Software Interfaces 10
3.1.4 Communications Interfaces 10
3.2 FUNCTIONAL REQUIREMENTS 10
3.3 USE CASES: 14
3.3.1 Use Case #1 14
3.3.2 Use Case #2 15
3.3.3 Use Case #3 15
3.5 NON-FUNCTIONAL REQUIREMENTS 16

4. SYSTEM DESCRIPTION 17

5. SOFTWARE ARCHITECTURE OVERVIEW 18

5.1 ARCHITECTURAL DIAGRAM 18
5.2 UML CLASS DIAGRAM 21
Class Descriptions 21

6. DEVELOPMENT PLAN AND TIMELINE 24

6.1 PARTITIONING OF TASKS 24
6.2 ESTIMATED TIMELINE OF TASKS 24
6.3 TEAM MEMBER RESPONSIBILITIES 25

7. TEST PLAN AND VERIFICATION 25

7.1 INTRODUCTION 25
7.2 SCOPE 25
7.3 OBJECTIVES 25
7.4 TEST STRATEGY 26
7.5 TEST ENVIRONMENT 26
7.7 TEST CASES 28
7.7.1 Functional Test Cases 28
7.7.2 Unit Test Cases 28
7.7.3 System Test Cases 29

8. VERIFICATION AND VALIDATION PROCESSES 29

9. ENTRY AND EXIT CRITERIA 30

10. DELIVERABLES 30

11. SCHEDULE 30

12. RESPONSIBILITIES 31

13. RISK MANAGEMENT 32

14. APPROVAL 32

15. DATABASE REQUIREMENTS 32

15.1 DATA MODEL DESIGN 32
15.2DATABASE SELECTION 33
15.3 DATA STORAGE AND RETRIEVAL 34
15.4 DATA SECURITY 35
15.4.1 ENCRYPTION 35
DATA AT REST ENCRYPTION: 35
15.5.1 BACKUP STRATEGIES 36
DATA BACKUP FREQUENCY: 36
15.6 SCALABILITY 37
15.6.1 VERTICAL SCALING 37
15.7 DATA CONSISTENCY AND INTEGRITY 37
15.7.1 Data Consistency 37
15.7.2 Validation 37
15.8.2 Monitoring with NRQL: 38
15.8.3 Maintenance 38
15.9 ANALYTICS AND REPORTING 38
15.10 COMPLIANCE AND PRIVACY 38
15.10.1 Compliance: 38
15.10.2 Privacy 38

1. Introduction

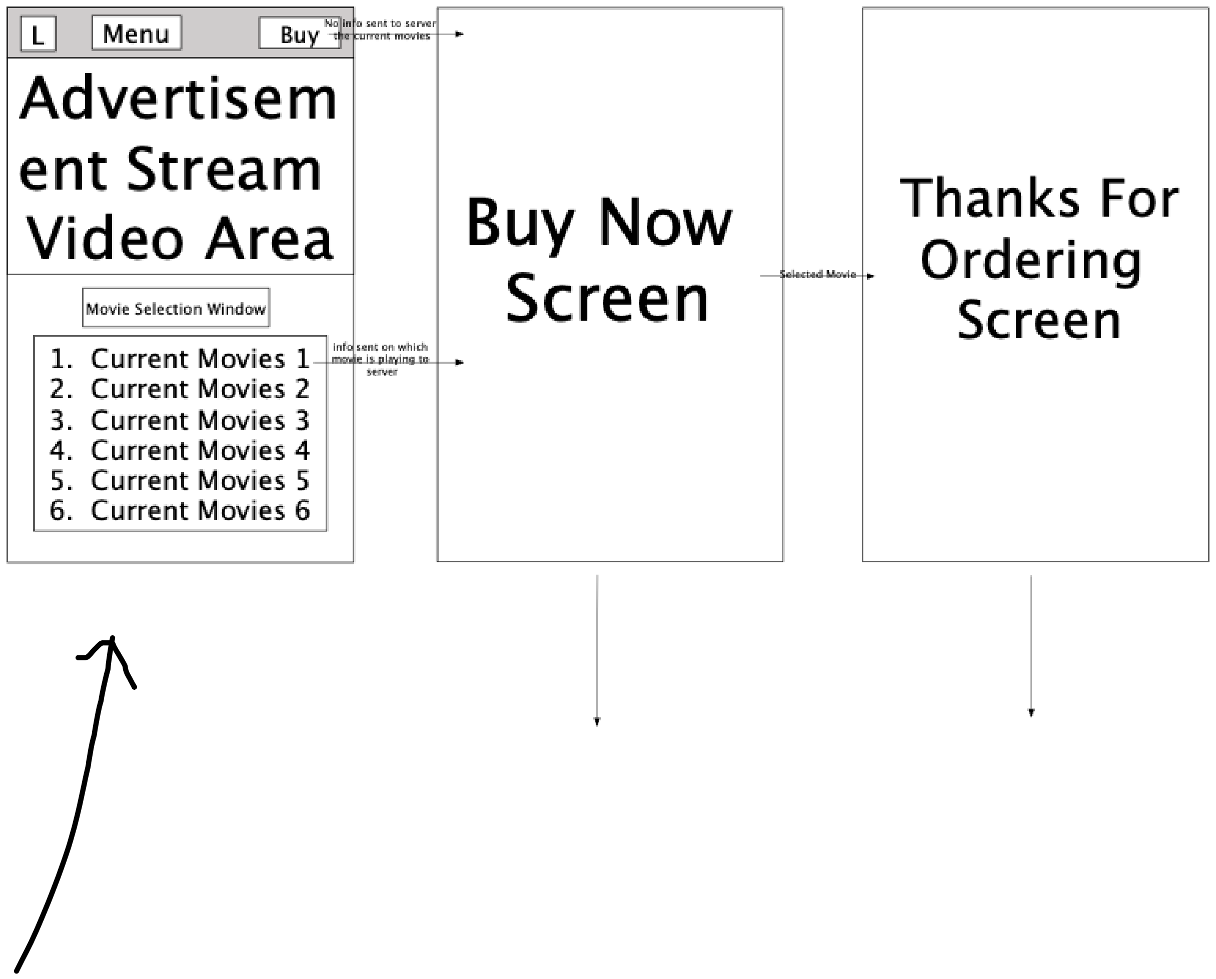
1.1 Purpose

The purpose of this document is to document the ways that the Movie Theater ticketing system functions. It will document both the back and front end of the ticketing system and analyze the ways the user interacts with these components. It is intended to be used by us when creating the specifications from the information our contractor gave us. We will be using the Waterfall Method for the Development of the program.

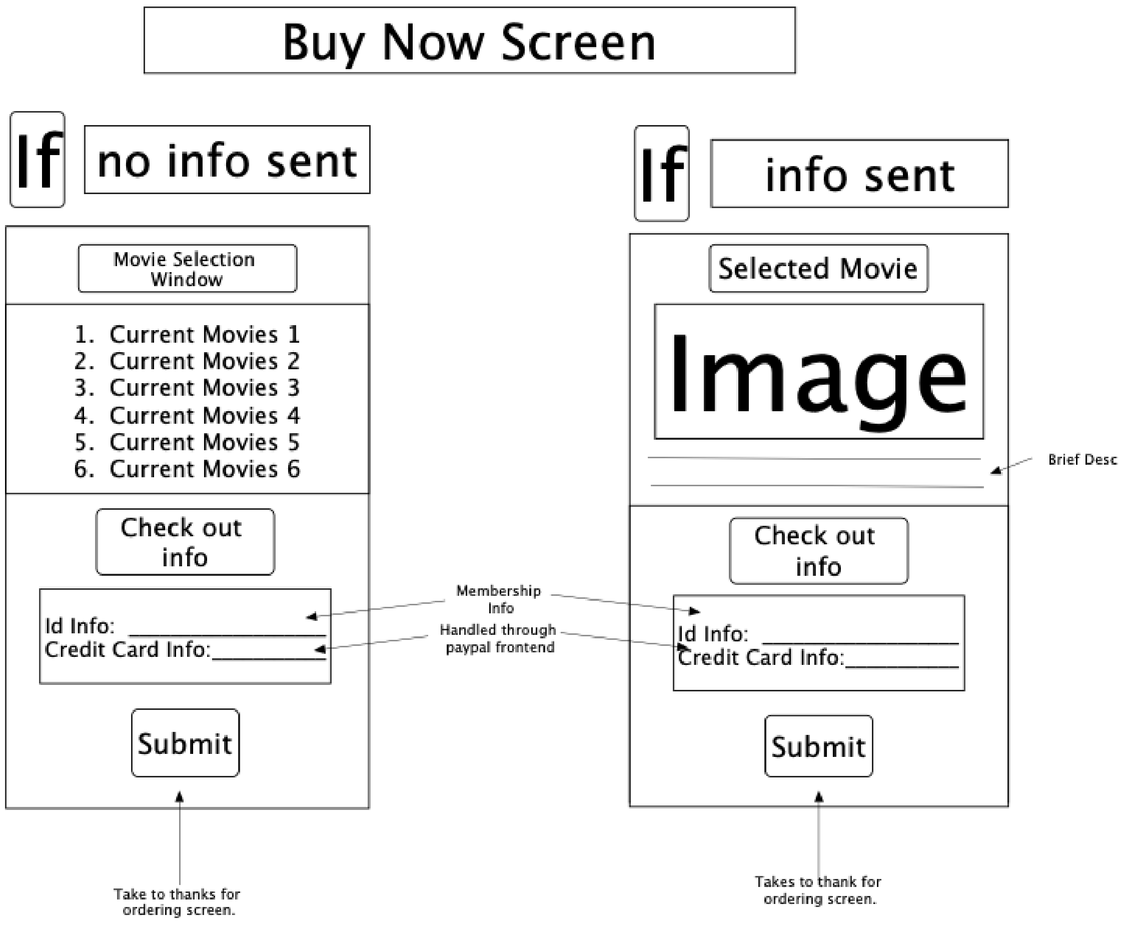
1.2 Scope

This project uses a client-server model to allow users to buy tickets from the vendor. To do this three separate software's will be designed that will work together to allow users to interact with the ticket vendor both in person and online. Ticket Server will be the software that runs on the server end. It will be responsible for a few different tasks that are split into subprograms. One task is indexing past and currently available movies including their descriptions. This means that it will add movie/movie descriptions to its currently playing library at the admin's request or allow for an admin to upload a new movie/movie description to it. No movies that are uploaded will be deleted by default when new movie/movie descriptions are uploaded. A second task the servers will be responsible for is loading randomized clips of trailers that will be accessible to the client with a URL. The server will have a timer set for the length of the video and rotate the videos it is serving to the client every time this timer ends. A second URL will display the timer value in plain text. Both will be used by the clients to display the trailers to the user. A third functionality will be that the server will provide information about any movie currently playing to the client when it is requested. This will be searched either by the current playing index or the name of the movie.

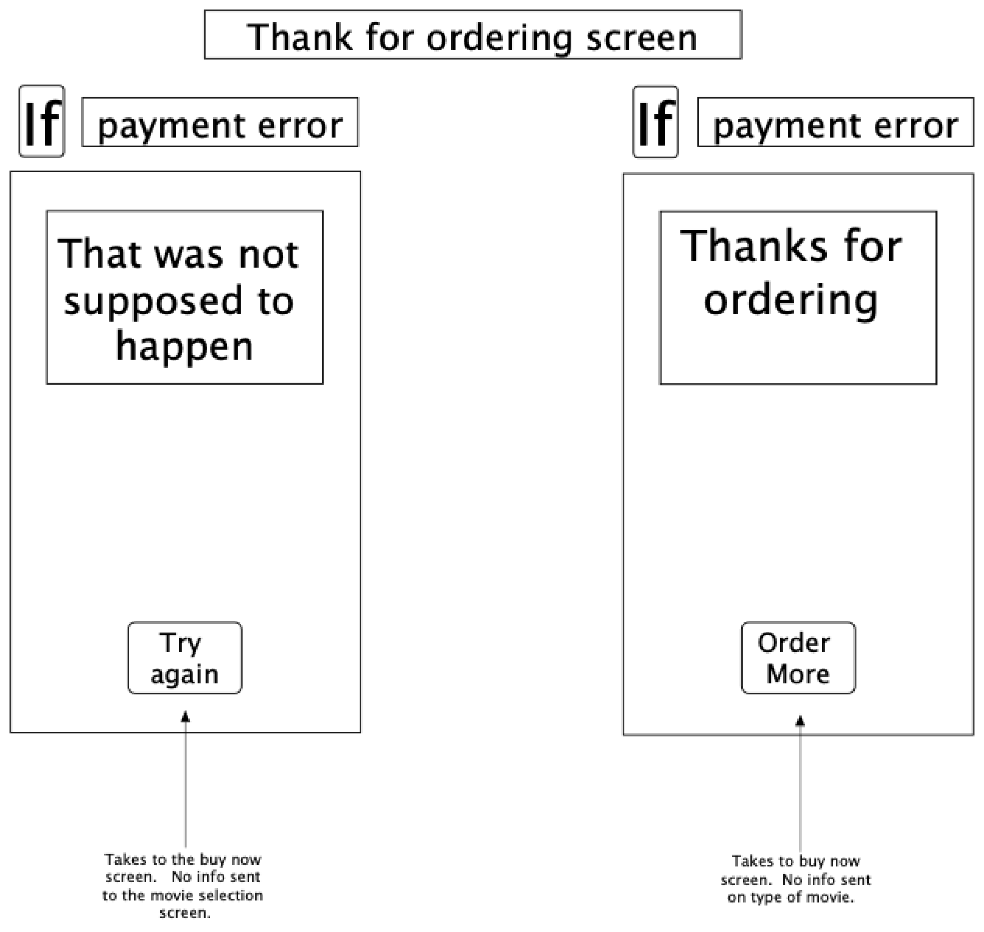
The first of the two clients will be called Ticket Sales Anywhere. It will allow users to buy tickets from anywhere based on their requested order.



This page allows people to see movies currently playing. It will display a current stream of movie trailers that the theatre is currently playing or movies that have been played in the past. It will also allow users to select from one of the current movies that are playing in theaters. Hovering over a title will bring up a trailer and description of the movie. If the user clicks buy then they will get sent to the buy now screen without any info about the current movie they have selected in the website catch if not they will click on the movie they want and then get set to the buy now screen with the movie they wanted loaded into the website catch.

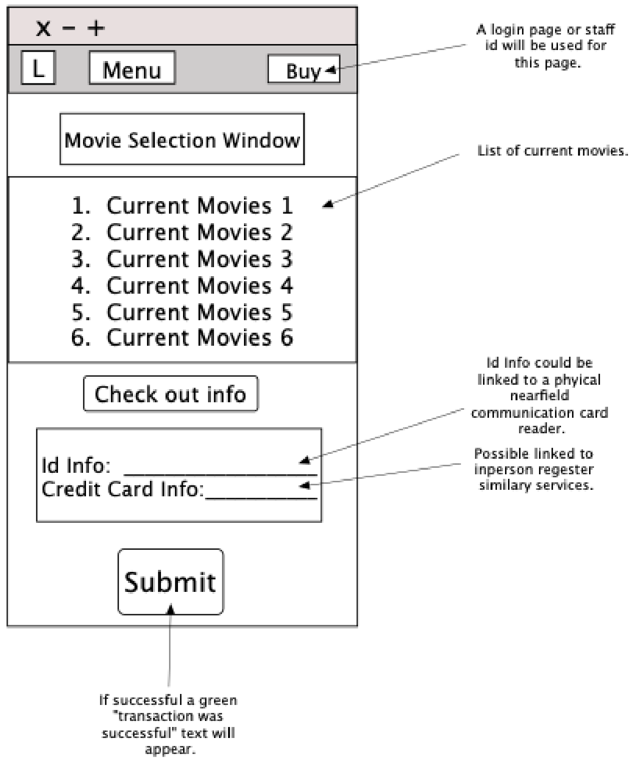


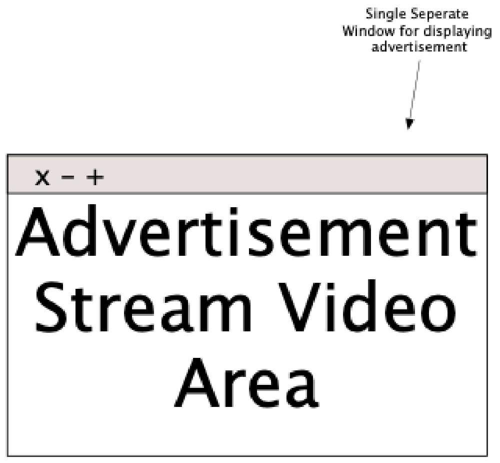
This screen is responsible for letting a user pick out a ticket or tickets if they have not already. It then takes their membership ID for identification purposes and their credit card info using PayPal. When the submit button is hit the system will place this information from the form into the website catch and then go to the URL of the "thanks for ordering screen".


Thank for ordering screen
If
payment error
That was not supposed to happen
Try again
Takes to the buy now screen. No info sent to the movie selection screen.
If
payment error
Thanks for ordering
Order More
Takes to buy now screen. No info sent on type of movie.

This screen is responsible for displaying a loading screen and then posting information about the chosen movie, identification information, and PayPal / Payment information to the Ticket Server. This will then process the information. When done it will modify the html page so that it displays one of the two screens above. If successful it will then display an image with a caption over it saying thanks for ordering. It will then wipe the information from the local browser catch. If not, it will display an image with a caption over it saying that was not supposed to happen. In both situations, there will be a button at the bottom that redirects back to the Buy Now Screen. However, the text in the button may be different as you can see.

The second of the two clients will be a simple client made for staff members. The name of the program will be Ticket Sales Assistant. The web app will be written within JS nodes so it can run as an HTML and JavaScript-based app. It will have the following layout.


x - +
L
Menu
Buy
A login page or staff id will be used for this page.
Movie Selection Window
List of current movies.
1. Current Movies 1
2. Current Movies 2
3. Current Movies 3
4. Current Movies 4
5. Current Movies 5
6. Current Movies 6
Check out info
Id Info could be linked to a phyical nearfield communication card reader.
Id Info: ____________
Credit Card Info: ________
Possible linked to inperson regester similary services.
Submit
If successful a green "transaction was successful" text will appear.


Single Seperate Window for displaying advertisement
x - +
Advertisement Stream Video Area

The figure to the left is the staff portal screen. It will have a way for staff members to identify themselves. Login info will be provided with a popup page. The simple web-based app will display a list of current movies that are being played. Then a dialog box on the bottom will show the customer info entry screen. The two boxes for input which are respectively the ID info followed by a line and credit card info followed by a line will be represented in a form that will allow people to enter more info into it. Both could be linked to external services or peripherals such as a NEF communication card reader or a cash register. The submit button will upload the information to the Ticket Server. During the post and processing time, a small loading wheel will appear. After success, the server will change the HTML page so that a “success” text will be displayed, and if failed a small “failed, please try again” text will appear. If the logo is pressed a separate dialog page will appear that allows for the staff to input new movie information and trailers to the server.

The figure to the right is a small separate dialog box that will appear having an advertisement stream of movies that have been played in the past and are currently playing. These will be taken from the video and time stamp function on the Ticket Server. The time stamp at the opening of the window will be used to tell how far into the video to play and when the video is complete so the page can reload without showing the user or view and fetch the next video from the URL described above.

1.3 Definitions, Acronyms, and Abbreviations

ID Info:
Information is relevant to a current user or customer that is unique enough to differentiate from other customers.

Credit Card Info:
The information needed for a financial transaction between the theater and customer.

NEF Communication:
Near-field communication is used to transfer information over short distances.

Ticket Server:
Used for processing transactions, loading content about the movies, and managing user data.

Ticket Sales Anywhere:
Website for customers to order tickets from the theater online. The transactions are handled by the backend Ticket Server.

Ticket Sales Assistant:
Web app for displaying movie information and helping users to order tickets. Can interface with peripherals like a cash registrar and NEF communication reader to get information relevant to the transactions. The transactions are handled by the backend Ticket Server.

Advertisement Stream Video Area:
An area where past and present trailers and movies are played. This may be played in the background and is useful for keeping the customer engaged.

1.4 References

- What Is MV3:
  - Title: What Is MV3
  - Report Number: N/A
  - Date: N/A
  - Publishing Organization: Chrome Developer

- IEEE Recommended Practice for Software Requirements Specifications:
  - Title: IEEE Recommended Practice for Software Requirements Specifications
  - Report Number: 830-1998
  - Date: N/A
  - Publishing Organization: IEEE Computer Society

- Paypal Client and Server API:
    - Title: Paypal Client and Server API
    - Report Number: N/A
    - Date: 04/24/2024
    - Publishing Organization: Paypal Developer

- Bitcoin RPC API
    - Title: Bitcoin RPC API
    - Report Number: N/A
    - Date: N/A
    - Publishing Organization: Bitcoin Developer
- Web server Clustering: High Availability Setup
    - Title: Web Server Clustering: High Availability Setup
    - Report Number: N/A
    - Date: 05/20/2024
    - Publishing Organization: Alibaba Cloud

- HTTP Requests
    - Title: HTTP Requests
    - Report Number: N/A
    - Date: N/A
    - Publishing Organization: Code Academy

- DevOps and Security Glossary Terms:
    - Title: DevOps and Security Glossary Terms
    - Report Number: N/A
    - Date: N/A
    - Publishing Organization: Sumo Logic

- The Ultimate Guide to Review Scraping in 2024
    - Title: The Ultimate Guide to Review Scraping in 2024
    - Report Number: N/A
    - Date: 01/27/2024
    - Publishing Organization: AIMultiple

- Access Control: Models and Methods
    - Title: Access Control: Models and Methods
    - Report Number: N/A
    - Date: N/A

- Publishing Organization: Delinea

1.5 Overview

The rest of this document will be responsible for documenting information about the product. This could include information such as what systems the product runs on, how the program integrates with other systems at a lower level, and how will be using it.   As well as other user interfaces not documented in the screen above. As well as documenting any software-based dependencies our program will have. It will also include the hardware, software, and communication interfaces, as well as the functional and non-functional requirements.

The second section will give a general description of the product.   This will include the information on what the system is as mentioned above. The third section will go into the specification requirements of the program. As mentioned above it will split these requirements into several sections.

2. General Description

2.1 Product Perspective

This application is designed to operate within a cloud-based infrastructure of the Amazon Web Services (AWS) platform. Our decision to host most of the server-side content on AWS containers is driven by the need for scalability, reliability, and security AWS offers. Consequently, the application's performance and availability are linked to AWS's operational status and uptime.
We also are reliant upon the uptime and performance of credit card payment processing companies such as Visa and Mastercard. Downtime on the company's end would cause issues with our software.
We are also dependent on the uptime of the bitcoin network, which is a decentralized system that processes payments. Bitcoin payment processing times can vary wildly depending on the time of day as well as the volume of transactions at any given time.

2.2 Product Functions

This product has 4 main functions:

1) The user's ability to purchase tickets.
2) The client ability to enter an admin mode and perform various customer service actions.
3) The application will register and save user info either for customer use or for client purposes.
4) The application will use a queue system to handle influxes of users and limit purchases.

2.3 User Characteristics

There are two main user groups: customers and theater workers.

Theater workers can be given higher levels of access to the software, since they will need to access customer information to perform customer service work. It can be assumed that the theater will have done background checks on workers before allowing them to access this information.
Customers should be given a much more restrictive version of the software, since it is a public facing application on the open internet. We should take steps to ensure that the software is highly accessible and has minimal ambiguity. Accessibility for customers with disabilities such as vision or hearing impairment should be considered too.

2.4 General Constraints

Regulatory: Since this software accepts credit cards and bitcoin, we must adopt an appropriate logging policy to comply with local and federal laws. Bitcoin related purchases should be especially focused on due to the often-rapid changing legal status of cryptocurrencies. Additionally, we must handle sensitive user data appropriately and comply with state and federal data privacy laws, such as California's data privacy acts.

Reliability: The client has requested that the software handles at most 1000 people and has a queue system to control purchasing. We must ensure that we utilize appropriate load-balancing methods to manage this.

Audits: Similar to the above policy with CC and bitcoin we must provide appropriate logging for tax purposes in line with federal and state codes.

Safety: While not entirely in our control, we must be mindful of age-rating requirements for movies as well as content moderation for reviews. Another concern may be logging all chats with customer service representatives to document any inappropriate interactions.

2.5 Assumptions and Dependencies

This software is a web-based application designed to function on modern browsers. We assume that modern browsers such as Chrome/Firefox/Safari will support this software without significant issues. Our development team will conduct tests to ensure compatibility with the latest stable versions of these browsers.

Web browsers are updated frequently, and we will update this SRS accordingly. One major change that will occur later this year is Chrome manifest V3. This could impact performance and security. We should be researching how these changes will affect our software and any fixes that may be needed.

Additionally portions of this application rely on external dependencies including backend services, libraries and various APIs. Changes in any of these could result in security or performance issues to the application.

We can mitigate the effect of these changes by regularly monitoring for any major changes from these systems and by also implementing automated testing to detect issues when dependencies are updated. Creating fallback mechanisms may also be a viable solution for these issues as well.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

Do:

- Short error messages for easy customer support
- Visible purchasing UI, all buttons should be easy to find on their page
- Language selection should be offered upon first accessing the site and should be logged to users' account
- Splash page should have users account info easily readable upon login ie points, past purchases etc.

3.1.2 Hardware Interfaces

We should always ensure that our designed system does not pull too many resources from the clients' AWS instances. We should implement rate-limiting to prevent accidental DOS from users attempting to purchase tickets.

3.1.3 Software Interfaces

As of v1.0.0 we will require the following software:
- AWS
- JS Nodes (Server Application)
- Electron JS (Client Application)
- Paypal Client and Server API
- RPC API (Bitcoin)

3.1.4 Communications Interfaces

AWS staff handle most sever side communications interfaces, but we should still focus on ensuring that customers information is always transmitted between components in a secure manner with HTTPS and properly configured certifications.

3.2 Functional Requirements

3.2.1 User Handling

3.2.1.1 Introduction:

-The system will handle concurrent access by at least 1000 users.

3.2.1.2 Inputs:

- User requests for browsing, purchasing tickets, or accessing account information.

3.2.1.3 Processing:

- Implement load balancing and optimized database to deal with high traffic.

3.2.1.4 Outputs:

- Correct responses to different user interactions

- Queued responses when there are excess users.

3.2.1.5 Error Handling:

- Display error messages if the system is overloaded.

- Implement a queue system to manage excess users smoothly.

3.2.2 Web Browser Operation

3.2.2.1 Introduction:

-The system will entirely operate within a web browser environment,

3.2.2.2 Inputs:

- URL requests from user's web browsers.

3.2.2.3 Processing:

- Create the UI

- Make sure the system is compatible with modern browsers (Chrome, Firefox, Safari, Edge).

3.2.2.4 Outputs:

- Interactive web pages for browsing, purchasing, and managing tickets.

3.2.2.5 Error Handling:

- If the user's browser is not compatible display error mssage

3.2.3 Bot Prevention

3.2.3.1 Introduction:

- The system will block bots attempting to buy tickets.

3.2.3.2 Inputs:

- Ticket purchase requests from users.

3.2.3.3 Processing:

- Implement a verification system

- Monitor and analyze request patterns to identify and block bots.

3.2.3.4 Outputs:

- Blocked transaction attempts from identified bots.

- Successful transactions from legitimate users.

3.2.3.5 Error Handling:

- Message for users when verification system interaction is required.
- Log and alert administrators of suspected bot activity.

3.2.4 Database Interface

3.2.4.1 Introduction:

- The system will interface with the database of showtimes and available tickets.

3.2.4.2 Inputs:

- Requests for showtimes, ticket availability, and user transactions.

3.2.4.3 Processing:

- Perform CRUD operations on the database.

3.2.4.4 Outputs:

- Updated info on showtimes and ticket availability.
- Confirmation of successful transactions.

3.2.4.5 Error Handling:

- Show error messages for issues when interacting with database.
- Log database errors for admin.

3.2.5 Ticket Purchase Limitations

3.2.5.1 Introduction:

-The system will enforce ticket purchase limitations.

3.2.5.2 Inputs:

- User selected showtimes and ticket quantities.

3.2.5.3 Processing:

- Validate the max ticket quatinty of 20
- Confirm that purchase is 2 weeks prior to showtime or 10 minutes after showtime.

3.2.5.4 Outputs:

- Confirmation of ticket purchase.
- Error messages for exceeding ticket quantity limits or outside purchase times.

3.2.5.5 Error Handling:

- Notify users if they attempt to buy more than 20 tickets or purchase outside the allowed timeframe.

3.2.6 Administrative Mode

3.2.6.1 Introduction:

- Admins will have more access than customers. In admin mode they will be able to modify movie information.

3.2.6.2 Inputs:

- Movie details such as descriptions or movie names.

3.2.6.3 Processing:

- Add new movies to the database and update the existing database.
- Validate input details

3.2.6.4 Outputs:

- Confirmation message for successful description update or movie addition.
- The movie will show the added movie or changed description.

3.2.6.5 Error Handling:

- Show error message for invalid inputs.
- Handle database issues

3.2.7 Customer Feedback System

3.2.7.1 Introduction:

- The system will include a customer feedback system.

3.2.7.2 Inputs:

- Customer feedback submissions.

3.2.7.3 Processing:

- Store feedback in the database.
- Alert admin to new feedback entries.

3.2.7.4 Outputs:

- Confirmation messages for customers.
- Feedback reports for administrators.

3.2.7.5 Error Handling:

-Error messages for failed feedback submissions.

3.2.8 Discount Ticket Support

3.2.8.1 Introduction:

- The system will support various discount ticket types.

3.2.8.2 Inputs:

- Discount codes

3.2.8.3 Processing:

- Validate discount eligibility.
- Apply discounts to ticket purchases.

3.2.8.4 Outputs:

- Updated ticket pricing based on discounts.
- Confirmation of discounted purchases.

3.2.8.5 Error Handling:

- Notify users of invalid discount codes.

3.2.9 Online Review Scraping

3.2.9.1 Introduction:

- The system will scrape online review sites to display reviews and critic quotes of movies.

3.2.9.2 Inputs:

- URLs of review sites and movie identifiers.

3.2.9.3 Processing:

- Extract review data.
- Update the database with new reviews and critic quotes.

3.2.9.4 Outputs:

- Display reviews and critic quotes on the movie detail pages.

3.2.9.5 Error Handling:

- Handle failures in scraping by notifying administrators.
- Show placeholder content if reviews cannot be fetched.

3.3 Use Cases:

3.3.1 Use Case #1

Name: Purchase Movie Tickets
Actor: Customer

Flow of Events:

The customer accesses the movie theatre's website. The customer navigates to the main menu section to browse currently available movies. The customer selects a movie they wish to watch. The system displays movie details such as showtimes, available seats, and ticket prices. The customer selects the desired showtime and number of tickets. The system validates the ticket quantity and showtime eligibility. The customer proceeds to checkout. The system prompts the customer to log in or create an account if not already logged in. The customer enters payment information and completes the transaction. The system confirms the ticket purchase and provides a confirmation number.

Assumptions about Entry Conditions: The customer has access to a compatible web browser and an internet connection. The customer has selected a movie from the available options and has valid payment information.

3.3.2 Use Case #2

Name: Admin Modify Movie Information
Actor: Admin

Flow of Events:

The admin accesses the admin portal of the movie theatre's website. The admin logs in using their credentials. The system presents a list of currently available movies. The admin selects the movie they want to modify. The system displays options to edit movie details such as descriptions, showtimes, or trailers. The admin makes the desired modifications. The system validates the changes and updates the database accordingly. The admin receives a confirmation message of successful modification.

Assumptions about Entry Conditions: The admin has appropriate permissions to access the admin portal. The admin is logged in and authenticated. The movie selected by the admin exists in the database.

3.3.3 Use Case #3

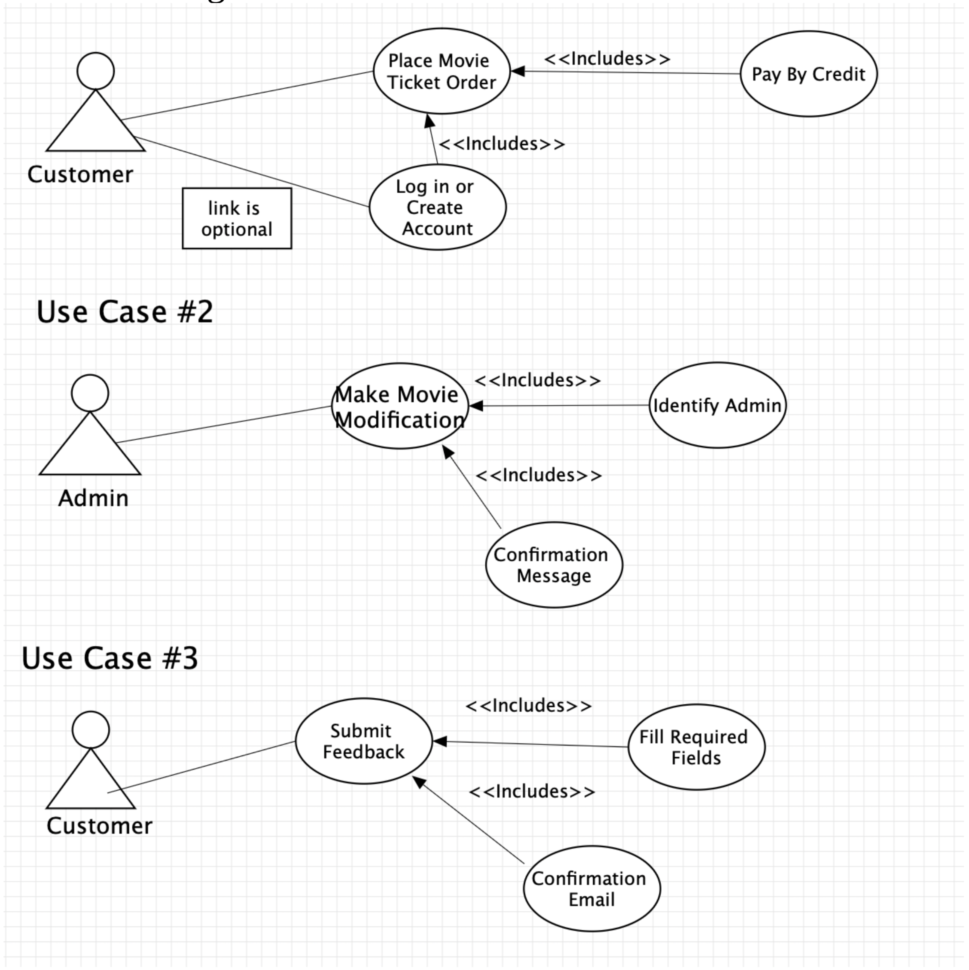
Name: Submit Customer Feedback
Actor: Customer

Flow of Events:

The customer accesses the movie theatre's website. The customer navigates to the "Feedback" section. The system provides a form for the customer to submit feedback. The customer fills in the required fields such as name, email, and feedback message. The customer submits the feedback. The system validates the input and stores the feedback in the database. The system sends a confirmation email to the customer acknowledging receipt of feedback.

Assumptions about Entry Conditions: The customer has access to a compatible web browser and an internet connection. The feedback form is accessible and functional. The customer provides valid account information.

Use Case Diagrams:



3.5 Non-Functional Requirements

3.5.1 UI:

- The user interface will be designed to be intuitive and user-friendly to make website navigation easier for inexperienced users.
- Response times for user interactions will be optimized for minimal wait times.

3.5.2 Security:

- The system will implement security measures to safeguard user data, payment information, and system integrity.

- User authentication will be enforced for all user interactions, with secure password storage in the database.

- Access controls will be implemented to ensure that only authorized users have access to sensitive functions.

- Payment transactions will comply with industry standards.

- Regular security audits and vulnerability audits will be conducted to identify and address potential security threats.

3.5.3 Performance

- The system should have a response time of less than 2 seconds for 95% of user interactions.
- The system must handle up to 10,000 concurrent users without performance issues.

3.5.4 Reliability

- The system must gracefully handle errors and provide meaningful feedback to users without crashing.
- Regular backups must be performed, and a recovery plan will be in place to restore operations quickly.

3.5.5 Availability

- The system must be accessible to users 24 hours a day 7 days a week.
- Scheduled maintenance should be minimized and people informed in advance.

3.5.6 Maintainability

- The system should follow coding standards and best practices to ensure code quality and readability.
- Comprehensive documentation must be provided for system architecture, design, and code.

3.5.7 Portability

- The system should be capable of running on various operating systems (Windows, macOS…).
- The system should support all major web browsers (Chrome, Firefox, Safari…).

4. System Description

This software system will be used as both a ticketing and customer service application for a movie theater. The client has requested that this software is a web-based application, that will accept both credit cards as well as bitcoin. The client has additionally requested that the customer side of the software features a queue system for crowd control as well as a customer reward system. Focusing on the queue system, it should manage in demand movies and block detected bots from purchasing tickets to these movies.

The customer-service side of the application should be able to accept customer feedback in the form of a 5-star style system. The clients' workers should be able to enter an "admin" mode that allows for them to perform customer service actions such as refunds or account support.

Since this software has two parts, we will be using a hybrid architecture pattern. This is because we must combine the event driven pattern and the microservice pattern. The event driven pattern will be used for the front-end website features like previews and the rating system. The microservice pattern will be used for the payment processing systems and well as anti-bot measures.

5. Software Architecture Overview

5.1 Architectural diagram

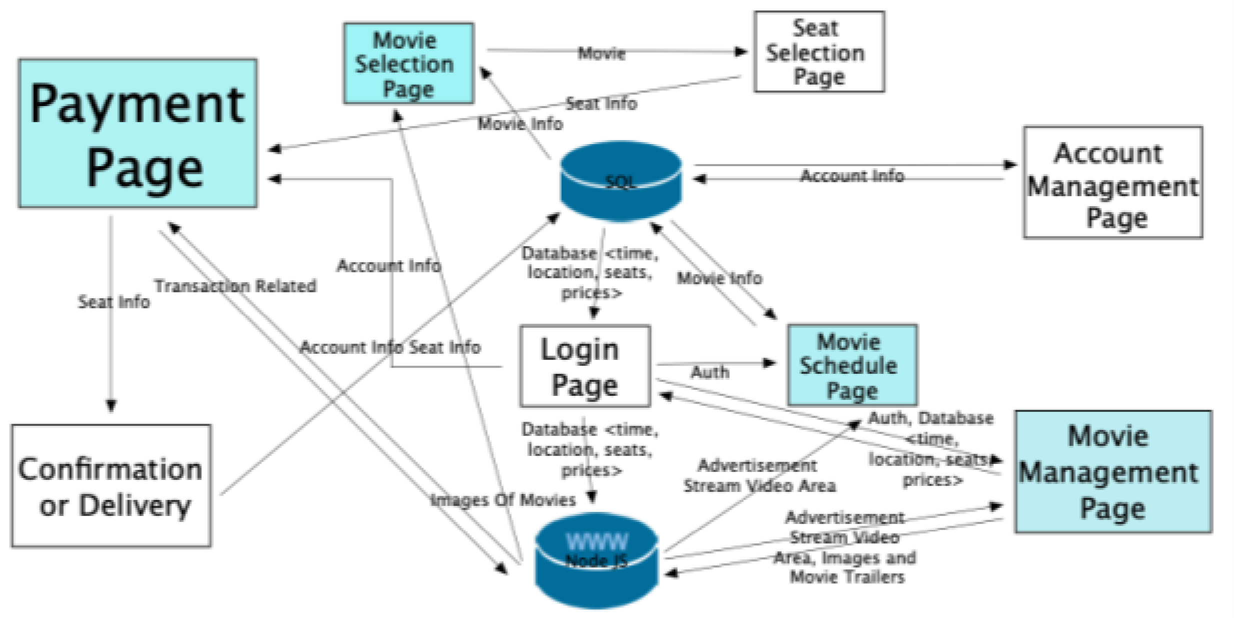


Diagram of the Architecture for Ticket Sales Anywhere and Ticket Sales Assistant

This is a diagram of what the Ticket Sales Anywhere and Ticket Sales Assistant client's architecture should look like. The diagram has the SQL servers which are responsible for storing information about the Time, location, seats, and movie prices. The Node JS server is responsible for serving movie images of movies currently playing to the movie selection screen, and clips to the user on the movie selection page which could be seen as the greeting or welcome screen that also displays a list of currently playing movies within it. The Payment Page has information being sent to it from the Node JS server because PayPal and Bitcoin transactions require a server to be used while making payments and keeping track of them.   All three of these screens have blue boxes to show they are using information from the Node JS server is being used in them. The movie management page is only logged into when an administrator logs in and displays the Ticket Sales Assistant UI which is what allows administrators to manage movies. It also uses the Node JS backend so it can change data in it and manage available movies.

Login Page:

Displays login information for users when they get to the website.

Movie Schedule Page:

A page that displays options for current movies that are playing as well to schedule the movie at a specific theater and date.

Account Management Page:

A separate page a user can get to when pressing a button somewhere on the top of the UI. It allows users to configure their accounts and manage their information.

Seat Selection Page:

A page that allows the user to select which seat they are currently sitting at. It displays the row of seats available.

Movie Selection Page:

Displays information about the current movies available and allows you to select the one you are choosing to watch within the theater.

Payment Page:

Allows you to select a payment option. It will include a PayPal payment and a Bitcoin Payment area.

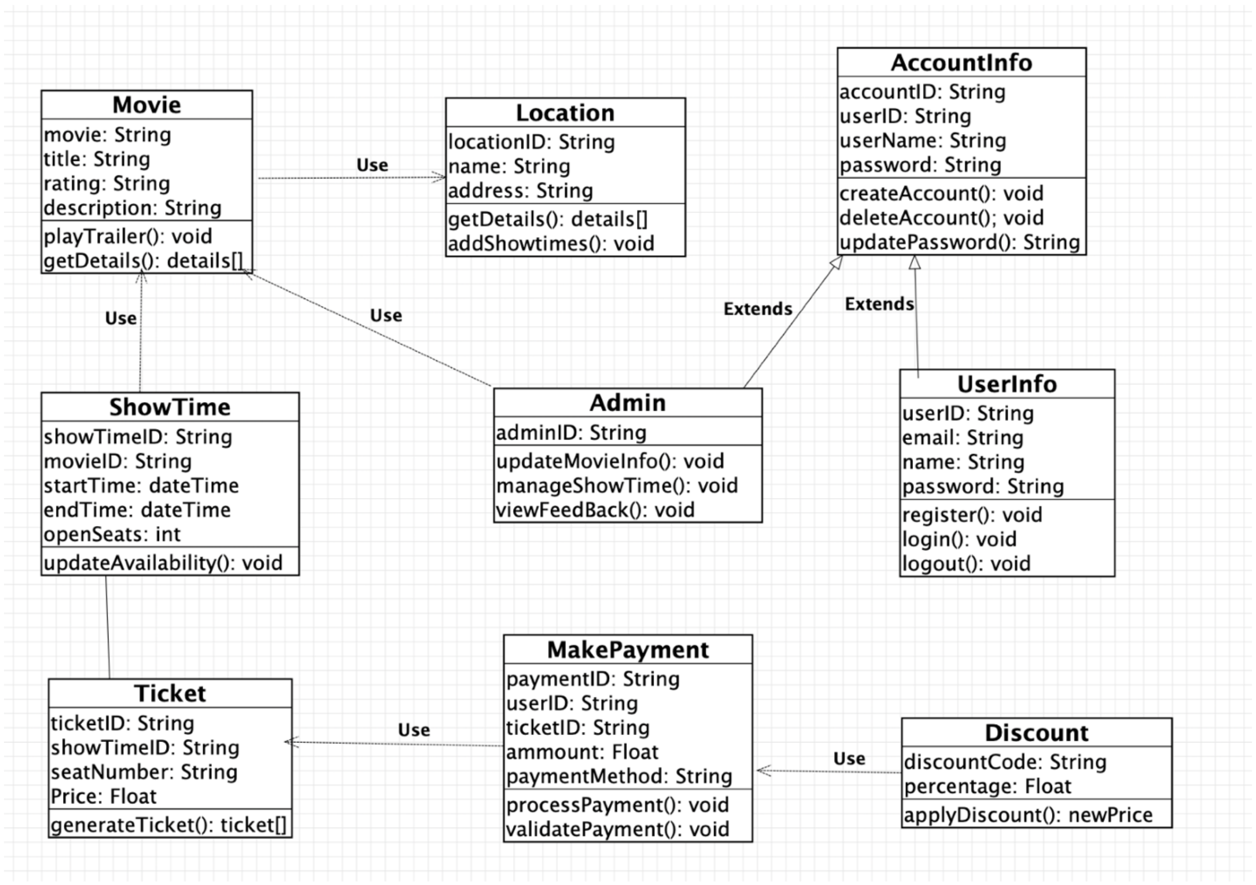
Confirmation and Delivery Page:

A ticket will be emailed to the user that will have more information on the ticket they just purchased.

Movie Management Page:

Uses the Ticket Sales Assistant UI and is responsible for managing movies currently playing. It allows them the upload new movies and delete old ones from the currently playing list when they are no longer playing. It also has a pop-up website that can be maximized and displays the Advertisement Stream Video Area.

5.2 UML Class Diagram



Explanations of UML Diagram: The UML Class Diagram represents a movie ticket booking system where movies are associated with locations and showtimes. Users can register and manage their accounts, purchase tickets, make payments, and apply discounts. Admins have special privileges to manage movie information and showtimes.

Overview of Classes: The classes in the UML Diagram are Movie, Location, AccountInfo, Admin, UserInfo, ShowTime, Ticket, MakePayment, and Discount.

Class Descriptions

Movie:

Attributes:
movie: String
title: String
rating: String
description: String

Operations:
playTrailer(): void - Plays the trailer of the movie.

getDetails(): details[] - Retrieves the details of the movie.

Location:

Attributes:
locationID: String
name: String
address: String

Operations:
getDetails(): details[] - Retrieves the details of the location.
addShowtimes(): void - Adds showtimes to the location.

AccountInfo:

Attributes:
accountID: String
userID: String
userName: String
password: String

Operations:
createAccount(): void - Creates a new account.
deleteAccount(): void - Deletes an account.
updatePassword(): String - Updates the password for the account.

Admin (Extends AccountInfo):

Attributes:
adminID: String

Operations:
updateMovieInfo(): void - Updates information about movies.
manageShowTime(): void - Manages showtimes for movies.
viewFeedBack(): void - Views feedback from customers.

UserInfo (Extends AccountInfo):

Attributes:
userID: String
email: String
name: String
password: String

Operations:
register(): void - Registers a new user.

login(): void - Logs in an existing user.
logout(): void - Logs out the user.

ShowTime:

Attributes:
showTimeID: String
movieID: String
startTime: dateTime
endTime: dateTime
openSeats: int

Operations:
updateAvailability(): void - Updates the availability of seats for the showtime.

Ticket:

Attributes:
ticketID: String
showTimeID: String
seatNumber: String
Price: Float

Operations:
generateTicket(): ticket[] - Generates a ticket for the showtime.

MakePayment:

Attributes:
paymentID: String
userID: String
ticketID: String
amount: Float
paymentMethod: String

Operations:
processPayment(): void - Processes the payment for the ticket.
validatePayment(): void - Validates the payment information.

Discount:

Attributes:
discountCode: String
percentage: Float

Operations:

applyDiscount(): newPrice - Applies a discount to the ticket price.

Explanation of Relationships:

Use Relationships:
Movie uses Location.
Movie uses ShowTime.
Ticket uses MakePayment.
MakePayment uses Discount.

Associations:
ShowTime associated with Ticket.

Generalization Relationships:
Admin extends AccountInfo.
UserInfo extends AccountInfo.

6. Development Plan and Timeline

6.1 Partitioning of Tasks

Account Management
Payment System/Payment Processing
Ticketing
Movie Managment
Security/Queue System
Verification/Inspection

6.2 Estimated Timeline of Tasks

Based on the client budget of 500k and our current team size of 3 people, we estimate a total development time of about 7 months for the project. This timeline includes the build time, the time for verification of the application, and post launch feedback.
Estimated time for each of the above tasks is approximately 1 month for each task with an additional month and a half for quality control and for client feedback.

Below is a high-level overview of the development timeline. For now, this timeline is monthly and can be changed based on team feedback.

Month 1: Initial Planning and Requirements Gathering
Month 2: Dev Team Design Period
Month 3: Development – Front End
Month 4: Development – Back End
Month 5: Testing and QA
Month 6: Bug Fixing and Product Launch
Month 7: Post-Launch and Maintenance

6.3 Team Member Responsibilities

James - Verification/Inspection, Security
Lexa - Author – Account Management, Payment Processing
Eli - Author – Security, Ticketing, Movie Management System

7. Test Plan and Verification

GitHub: https://github.com/LexaRao/Movie-Theater/tree/main

7.1 Introduction

The purpose of *Section 7 Test Plan and Verification* will describe how we will test and verify the movie ticketing system. In this section we will go over the scope of what is being tested, the objectives for testing, our strategy for testing, the functional requirements for the system, example test cases, the verification and validation process, entry and exit criteria, the deliverables, our schedule for completing the testing, the responsibilities given to different members during testing, risk management, and lastly approval.

7.2 Scope

The scope of testing and verification will include all functional and non-functional aspects of the movie ticketing system. This includes verifying the user interface for both customers and theater staff, validating the integration of external payment systems (PayPal), making sure the AWS-hosted servers perform well, and confirming compliance with security and regulatory requirements. Additionally, testing will cover user authentication, data processing, error handling, and system maintainability across different web browsers and operating systems.

7.3 Objectives

Verify Functionality: Ensure all features and functionalities, such as ticket purchasing, admin operations, and customer feedback, are working as intended.

Ensure Performance: Confirm the system can handle up to 10,000 concurrent users without significant performance degradation.

Check Security: Validate that user data, payment information, and system integrity are protected through robust security measures.

Assess Reliability: Ensure the system can handle errors gracefully, providing meaningful feedback to users and performing regular backups.

Validate Compatibility: Confirm the system is compatible with various operating systems (Windows, macOS) and supports all major web browsers (Chrome, Firefox, Safari), ensuring a consistent user experience across different platforms.

Verify Integration: Ensure integration and interaction between different modules. Verifying that data flows correctly and processes are synchronized.

7.4 Test Strategy

Verification:

Code Reviews: Conduct thorough reviews of the code to ensure it adheres to the design specifications and coding standards.

Unit Testing: Perform unit tests to verify the functionality of individual components and modules.

Static Analysis: Use automated tools to analyze the source code for potential issues such as security vulnerabilities and code complexity.

Integration Testing: Conduct integration tests to verify the correct interaction between different modules.
System Testing: Perform system testing to verify that the entire system works together as intended.

Validation:

Functional Testing: Conduct functional tests to ensure that all features and functionalities, such as ticket purchasing and admin operations, work as expected from a user perspective.

Security Testing: Carry out security tests to validate that user data, payment information, and system integrity are protected.

Usability Testing: Conduct usability tests to ensure that the system provides a user-friendly interface and a consistent user experience across different platforms and devices.

7.5 Test Environment

Hardware:

Servers: Multiple servers to host the movie ticketing system components, including the Ticket Server, database server, and web servers.

Client Devices: Including desktop computers, laptops, tablets, and smartphones, to test the compatibility and responsiveness of the movie ticketing system across different platforms and screen sizes.

Software:

Operating Systems: The servers should run on supported operating systems, such as Windows, macOS, Android… For client devices.

Web Browsers: A range of web browsers, including Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge should be installed on client devices to test the compatibility and functionality of the movie ticketing system across different browsers.

Database Management System : The movie ticketing system relies on a database to store and retrieve data related to movies, showtimes, tickets, and user information. A DBMS should be installed and configured for testing.

Development Tools: Tools for code development, debugging, and testing, such as integrated development environments (IDEs) should be available for developers to identify and resolve issues efficiently.

7.6 Functional Requirements

User Registration and Login:
-Verify that users can successfully register for an account with valid information.
-Test the login functionality to ensure users can access their accounts.

Movie Listing:
-Confirm that the system accurately lists currently available movies, including showtimes, descriptions, and trailers.

Seat Selection:
-Verify that users can select seats for their desired movie showtime from an interactive seating chart.
-Test the seat availability feature to prevent double booking of seats.

Ticket Booking:
-Validate the ticket booking process including selecting a movie, choosing seats, and confirming the booking.
-Test the system's ability to handle multiple ticket bookings simultaneously without errors.

Payment Processing:
-Test the integration with payments such as PayPal or credit card processors, to make sure there is reliable payment processing.

Confirmation and Notification:
-Confirm that users receive a confirmation message or email after completing a ticket booking.
-Test the notification system

Admin Functions:
-Validate admin functions such as adding, editing, or removing movies from the system.
-Test the admin interface to ensure it provides sufficient control.

Error Handling:
-Verify that the system provides meaningful error messages and prompts users to correct input errors.

Performance:
-Test the system's performance under normal and peak load conditions.
-Validate response times for key functions such as page loading, seat selection, and payment processing.

Security:
-Test the system's security measures.
-Validate that user data and payment information are stored and transmitted securely.

7.7 Test Cases

7.7.1 Functional Test Cases

TU01 Verify email notification on purchase

Description: Test that upon purchase the user will be sent an email confirming their purchase and be sent a confirmation number and ticker information. This test is successful if the test walks through the entire purchase process and receives a confirmation email to the test email account.

TU02 Verify user account creation

Description: Test that the user can create and access an account with valid information. This test is successful if the tester is able to create an account with valid information.

TU03 Verify customer service login

Description: Test that the client will be able to login to the administrator account to perform customer service functions. This test is successful if the administrator account is successfully logged in to.

TU04 Verify movie preview function

Description: Test that a movie preview will show upon clicking on a title after navigating to the title from the home page. The test is successful if the preview plays upon clicking on the preview image

7.7.2 Unit Test Cases

TU05 Verify the function of the email receipt module

Description: Verify the software can send properly emails. The tester should ensure that emails are not flagged as spam and are correctly formatted to client specifications.

TU06 Verify the function of the queue system module

Description: The tester should verify that a movie with high demand has a queue system function. A successful test of this function should result in the user being taken to the queue page upon attempting to purchase tickets for this movie.

TU07 Verify the function of the movie add/delete module

Description: The tester should be able to enter the administrative account and add or remove movies. This should be tested by using the administrative account to perform both functions and a successful test should result in the sample movie being created and deleted.

TU08 Verify the function of the seat selection module

Description: The test should verify that the user is able to select any seat during the purchase process. A successful test results in the dummy user account selecting any seat in the movie.

7.7.3 System Test Cases

TU09 Verify integration between the software and CC payment provider systems.

Description: This test verifies that credit card information is correctly parsed to the payment processor's network. The tester should check that the entered information is properly formatted to the payment processor’s specifications.

TU10 Verify integration between the software and the PayPal API

Description: This test verifies that upon selecting the PayPal option in the ‘choose your payment’ option at checkout that the user is sent to the PayPal payment API. A successful test should result in a completed payment and the user being returned to the software upon completion of the transaction with PayPal.

TU11 Verify integration between the software and the Bitcoin payment API

Description: This test verifies that upon selecting the Bitcoin option in the ‘choose your payment’ option at checkout that the user is sent to the Bitcoin payment API. A successful test should result in a completed payment and the user being returned to the software upon completion of the transaction with the Bitcoin payment API.

TU12 Verify the load balancing upon hitting 10k concurrent users

Description: This test is done in addition to the queue tests to verify that upon hitting 10k users the application begins load balancing functions. This can be done by performing a stress test of the software to similate 10k users. The load balancing test is successful if the first connection after 10k users is either rejected or is sent ot the queue system.

8. Verification and Validation Processes

- **Verification:** Conduct code reviews using software we have coded to test the program and manual review of the code, and, use scripts written in Edureka to test the UI to make sure it does what we expect it to. Make sure to do this regularly throughout the design process.

- **Validation:** Make sure our system conforms to the functional, and non-function requirements and is validated against all system requirements. Make sure our system functions as it should within the given state of the system.

9. Entry and Exit Criteria

Entry Criteria:

All the requirements for our system have been thoroughly documented and have been approved. These requirements should conform to the functional and non-functional requirements that our contractor has created for us.

- The test environment includes software like Edureka should be set up. We should already have the testing scripts written based on the requirement list above.
- Data related to the tests is prepared.
- The tests we will be running are reviewed and approved.

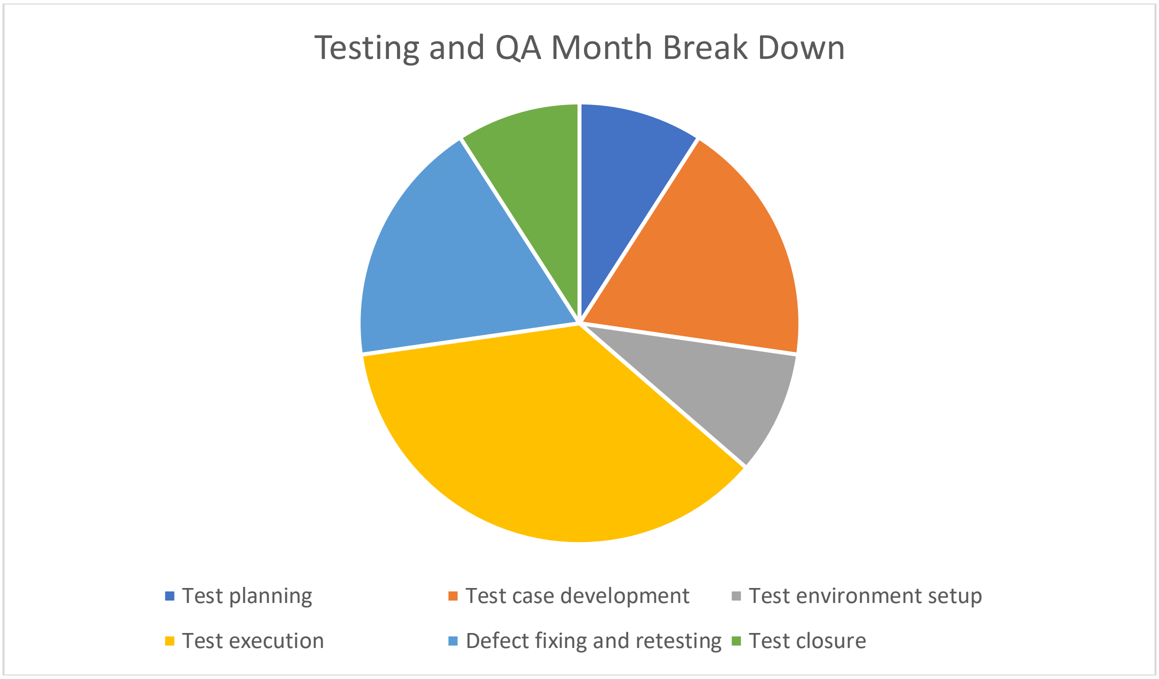
Exit Criteria:

- All of the tests should be executed. Test using software will be conducted with a real person responsible for watching the test as they are being run and executed.
- Resolve all defects in the program that may result in the final project not working properly.
- Create a Test Summer Report based on information we have gathered from above.
- Make sure we meet the acceptable criteria.

10. Deliverables

- Test Plan Document that conforms to the test requirements that we have created above.
- Test Cases
- Test Scripts (both UI and non-UI test scripts should be returned)
- Test Execution reports
- Defect Report that should include information about which defects have been resolved in this version of the program.
- Test summer Report

11. Schedule



- Test planning: 0.09 months
- Test case development: 0.18 months
- Test environment setup: 0.09 months
- Test execution: 0.36 months
- Defect fixing and retesting: 0.18 months
- Test closure: 0.09 months

12. Responsibilities

- **Test Manager and Engineers:** Responsible for planning and coordination tester. They will be responsible for creating the test case design, executing the test, and reporting the system defects.
- **Developers:** Responsible for fixing the bugs that were reported.
- **Business Analysts:** Goes through and reviews the test cases and makes sure that the requirements are currently being met.

13. Risk Management

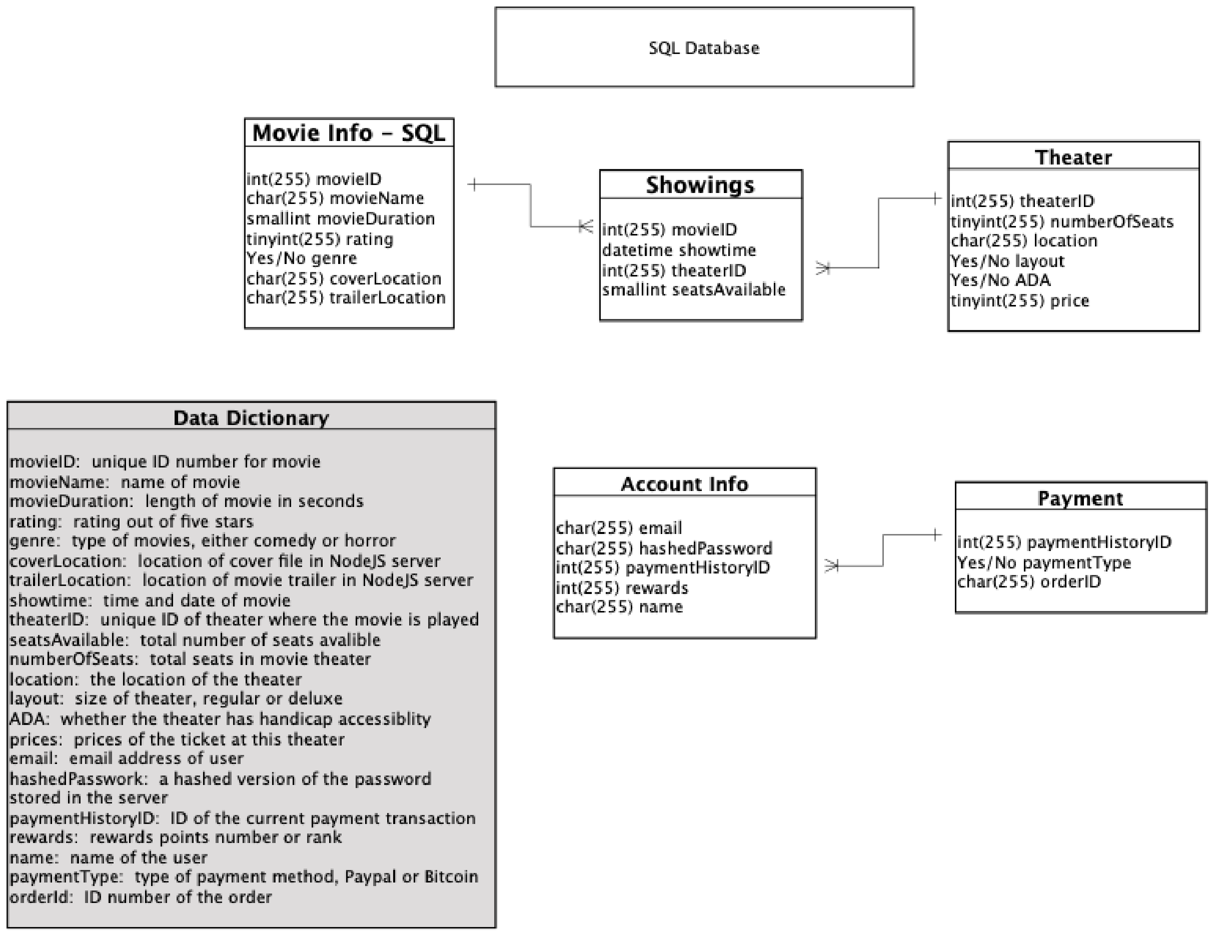
- **Risk:** Delays in the setup of the testing environment extend the amount of time this process takes.
  - **Mitigation:** Plan for this face by having testing software and requirements ready. Make sure to allocate at least half a month as buffer time.
- **Risk:** Incomplete requirements
  - **Mitigation:** Continuous communication with our contractor will help prevent this from happening. We will meet with them so we can answer any questions they might have in a QA interview format.
- **Risk:** High defect rates:
  - **Mitigation:** Review code regularly. If tests are done in an automated format using a preexisting testing script make sure to run it on a nightly basis and make sure the developers know what the defects are.
- **Risk:** Usage of obsolete or overly complicated components that may increase the total development time.
  - **Mitigation:** Make sure to speak sure to factor into account the amount of time it may take to maintain the code. Make sure to keep a list of alternate software dependencies if it is needed for the project to function.

14. Approval

This plan should be approved by our contractors and the test of our team. Since our team consists of three people, we are being contracted by a small independent contractor and we should work together as a team to make sure this plan fits everyone's needs.
This test plan ensures that we can test all the code to make sure it conforms to the requirements of the movie ticketing systems. It makes sure that the system is functional and creates a more structured approach to testing the theater ticking system.

15. Database Requirements

15.1 Data Model Design



Data Dictionary

movieID:	unique ID number for movie
movieName:	name of movie
movieDuration:	length of movie in seconds
rating:	rating out of five stars
genre:	type of movies, either comedy or horror
coverLocation:	location of cover file in NodeJS server
trailerLocation:	location of movie trailer in NodeJS server
showtime:	time and date of movie
theaterID:	unique ID of theater where the movie is played
seatsAvailable:	total number of seats avalible
numberOfSeats:	total seats in movie theater
location:	the location of the theater
layout:	size of theater, regular or deluxe
ADA:	whether the theater has handicap accessiblity
prices:	prices of the ticket at this theater
email:	email address of user
hashedPasswork:	a hashed version of the password stored in the server
paymentHistoryID:	ID of the current payment transaction
rewards:	rewards points number or rank
name:	name of the user
paymentType:	type of payment method, Paypal or Bitcoin
orderId:	ID number of the order

- **Entities:** Movie Info, Showings, Theater, Account Info, and Payment.
- **Relationships:** The relationship between Movie Info and Showings was one to many. The relationship between Showings and Theater was many to one. The relationship between Account Info and Payment was many to one.

15.2 Database Selection

- **Relational Database:** An SQL database will be used to store numbers, strings, and booleans relevant to the website. We will be using PostgreSQL since it is available on Linux and our server will be based on Linux Debian.
- **Disk Image in Ext4:** A separate disk image on the server file system will be responsible for storing Videos and Images needed by the server that are uploaded to it. Ext4 format will be used since it is wildly used on Linux Debian already and has the compatibility we need to build into it. Weekly snapshots will be created of this disk image. These disk images are exposed to the website from the Node.js VM running locally on the Server.

15.3 Data Storage and Retrieval

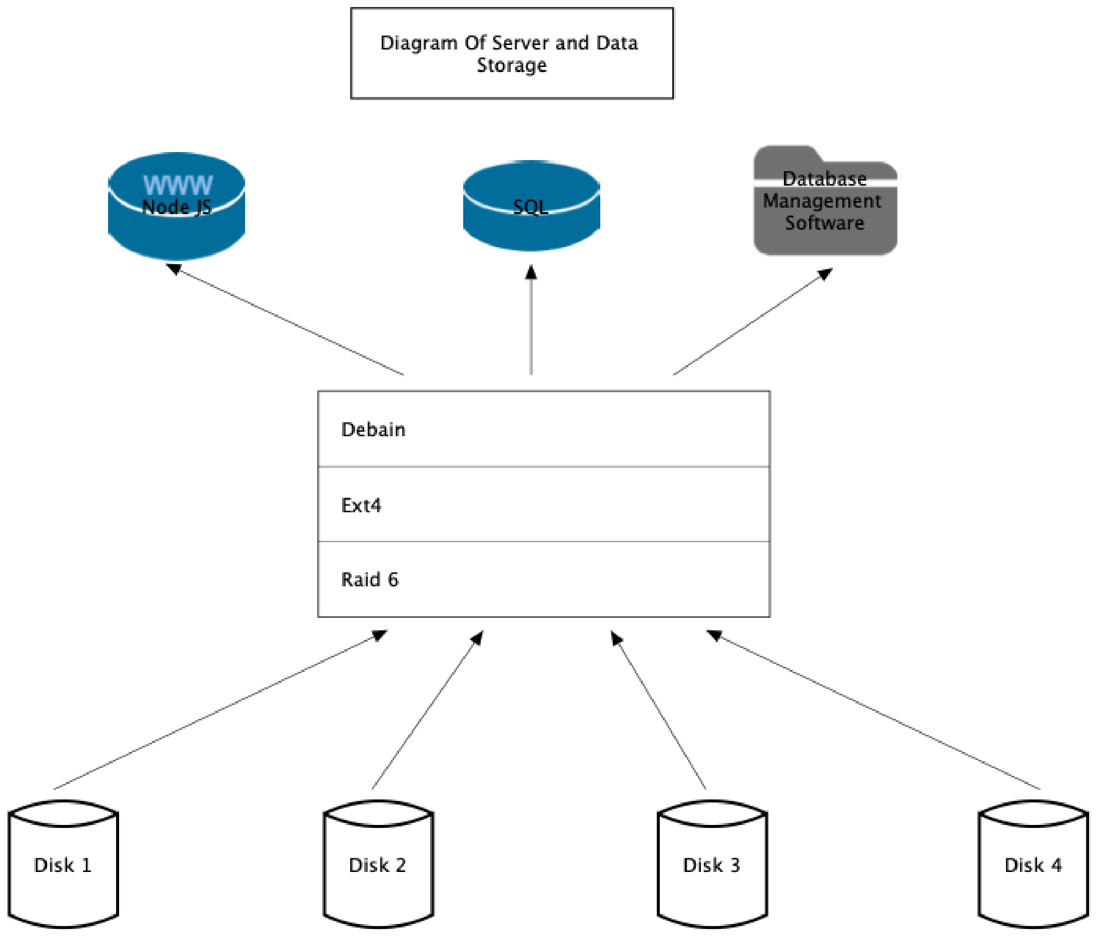


Diagram Of Server and Data Storage
WWW
Node JS
SQL
Database Management Software
Debain
Ext4
Raid 6
Disk 1
Disk 2
Disk 3
Disk 4

- **HDD: A** total of 4 1 TB HDD will be used in this server. We will use HDD since the drives do not need to have extremely high access times and this website will not likely have a lot of user traffic. The extra cache will be used to improve performance.
- **Raid: For** data redundancy and to ensure data correction with little effort from someone maintain the server when something gets corrupted or lost. We will use raid 6 since it allows for easier and more effective maintenance with smaller windows of downtime than other raid configurations.
- **Indexing: Keep** local indexes of all currently available movies and users so that they may be accessed more quickly when needed. Maintain indexes for both the SQL databases and the movie images and trailer video. Make sure critical information we need like names of movies, show times, and user IDs readily available since they are accessed more often. Taken care of by the high-level database management software.

- **Redundancies: Make** sure to have multiple local copies of SQL database sets so that if one version of the file is corrupted another can be restored without interrupting the server functionality. If there is a way built into SQL to do this without storing the data set in two separate files, then do this instead. This makes sure data is free of corruption and improves the theoretical performances of the system if one of the data sets is locked by the OS due to being accessed too many times for a writing attempt at the same time. Taken care of by the high-level database management software.
- **Partitioning: Similar** to what is mentioned above have these data sets separated to make them more manageable and improve the access time. To make it more manageable separate the data sets by their entries or their entries relationships. Taken care of by the high-level database management software.

15.4 Data Security

15.4.1 Encryption

Data at Rest Encryption:

- All sensitive data stored in a PostgreSQL database, including user passwords, payment information, and personal details, will be encrypted.
- Disk images storing videos and images will also be encrypted.

Data in Transit Encryption:

- All communications between client devices and the server will be encrypted.

15.4.2 Authentication and Authorization

User Authentication:

- We will implement strong user authentication mechanisms, including MFA for admin accounts.
- Access to different parts of the system will be restricted based on user roles.
- Admin functions, such as managing movies and processing refunds, will be accessible only to authenticated admin users.

15.4.3 Access Controls

Database Access Controls:
- Strict access controls will be used at the database level to make sure that only authorized users can access sensitive data.
- Database credentials will be stored securely and rotated regularly.

System Access Controls:
- Access to the server and application will be limited to authorized personnel only.

15.4.6 Regular Audits

Internal Audits:

- We will conduct regular internal audits of our data security practices such as access controls, encryption methods, and authentication mechanisms.
- Audits will ensure that our system works efficiently with our security policies and identify areas for improvement.

External Audits:

- We will conduct regular external audits by independent security firms. They will validate our security measures against industry standards.
- Audit findings will be used to improve our data security.

15.5 Backup and Recovery

15.5.1 Backup Strategies

Data Backup Frequency:

- Important data such as user information, ticketing transactions, and system configurations will be backed up daily.
- Incremental backups will be performed hourly for transaction data.

Backup Retention Policy:

- Backup retention periods will be decided based on business needs.
- Older backups will be automatically archived or rotated to make sure the storage system is efficient and so that historical data is accessible.

15.5.2 Disaster Recovery

Disaster Recovery Plan Development:

- A Disaster Recovery plan will be developed and it will document procedures for recovering critical systems and data in the event of a disaster.
- The Disaster Recovery plan will include roles and responsibilities, communication protocols, and escalation procedures.

Disaster Recovery Testing:

- Regular testing of the Disaster Recovery plan will be conducted to validate its effectiveness and identify areas for improvement.
- Testing scenarios will include simulated disasters such as server failures, data corruption, and network outages.

15.6 Scalability

15.6.1 Vertical Scaling

Hardware Upgrades:

- Upgrading CPU, RAM, and storage capacity of our servers to accommodate increased demands.
- Monitoring resource usage to identify when upgrades are necessary to maintain performance levels.

Database Optimization:

- Optimizing database configurations and performance to improve efficiency and handle larger datasets.
- Improving database instances by upgrading to higher performance database servers.

15.6.2 Horizontal Scaling:

Scaling Out Web Servers:

- Adding more web server instances behind a load balancer to distribute incoming traffic evenly.
- Using an auto scaling system to adjust the number of server instances based on traffic patterns.

15.6.3 Load Balancing

Load Balancer Configuration:

- Implementing an efficient load balancer to evenly distribute incoming requests across multiple web server instances.
- Doing load balancer health checks to monitor server availability.

15.7 Data Consistency and Integrity

15.7.1 Data Consistency

We should use ACID compliant transaction policies to ensure the durability and atomicity of any given database entry. This is especially ideal for ticket purchases and payments, since a corrupt entry in either of these could result in losses to the buisness.

15.7.2 Validation

The database for this software will be using SQL so we have a wild variety of tools at our disposal to ensure data validity and to remove any corrupt or invalid entries. The UNIQUE, NOT NULL and CHECK SQL constraints can all be used to ensure data validity. An example of this would be using the UNIQUE and NOT NULL constraints to ensure that when a customer creates an account their username and password are not empty, and that the username is unique.

15.8 Monitoring and Maintenance

15.8.2 Monitoring with NRQL:

New Relic will be used for monitoring our database performance and query efficiency. New Relic provides a wild variety of monitoring systems and provides its own monitoring language for databases: NRQL. Supporting documentation for NRQL can be found here.

The use of NRQL also has the added benefit of being very similar to the SQL language already used in our databases, therefore reducing the workload of our developers in this project.

15.8.3 Maintenance

Maintenance tasks should be run in the early morning around 4am to 5am in the client's time zone (PST). This is an idea time to run any updates/patches that may cause system downtime as there is unlikely to be any users online during this time frame.

15.9 Analytics and Reporting

Ideally, we should ensure we can extract KPI's and other critical pieces of information from the database. Most of the sever side content is handled through AWS in this project, so we can use available AWS tools to perform data analytics and reporting.

AWS provides tools such as QuickSight for business intelligence reporting, which would be ideal for generating reports from customer transaction data and making it readable to the client. For other data such as rewards info or movie related data, Amazon Kenises is available to process and generate reports from given data.

15.10 Compliance and Privacy

15.10.1 Compliance:

Since both our client and our software development team is based out of the state of California, meaning that we will have to comply with the CCPA. This means that we must provide the user with the ability to both request their data from our client as well as be able to make a request to delete the information as well.

Another key compliance issue is that we must be very careful with logging any bitcoin transactions that occur between our client and users of the apps to comply with anti-money laundering (AML) regulations. While this is mostly handled through our payment providers i.e. PayPal, the ever-changing regulatory issues of crypto make it especially important to log.

15.10.2 Privacy

Along with compliance with known government regulations, we should also provide the user with the option to control how much data we collect. This should be a cookie collection choice sheet at the bottom of the webpage.

Additionally, as discussed in the security portion of this document, we must take steps to encrypt and protect any collected customer data.