# MoviePlug Ticketing System

Software Requirements Specification

Version 4

Nov 2, 2023

Group 5

Ruben Guillen

Justin Pudiquet

Rosa Lisa Silipino

Prepared for

CS 250- Introduction to Software Systems

Instructor: Gus Hanna, Ph.D.

Fall 2023

# **Revision History**

| **Date** | **Description** | **Author** | **Comments** |
| --- | --- | --- | --- |
| Sept-21 | Version 1 | Group 5 | Sect. 1.1 - 3.5, excluding 3.4 |
| Oct-5 | Version 2 | Group 5 | Added Design Portion |
| Oct 19 | Version 3 | Group 5 | Added Test Plan |
| Nov 2 | Version 4 | Group 5 | Added Database Management |

# **Document Approval**

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

| **Signature** | **Printed Name** | **Title** | **Date** |
| --- | --- | --- | --- |
|  | Group 5 | Software Eng. | Nov 2, 2023 |
|  | Dr. Gus Hanna | Instructor, CS 250 |  |
|  |  |  |  |

# **Table of Contents**

[MoviePlug Ticketing System 1](#_heading=h.4k668n3)

[**Revision History 1**](#_heading=h.2zbgiuw)

[**Document Approval 2**](#_heading=h.1egqt2p)

[**Table of Contents 3**](#_heading=h.3ygebqi)

[**1. Introduction 6**](#_heading=h.2dlolyb)

[1.1 Purpose 6](#_heading=h.sqyw64)

[1.2 Scope 6](#_heading=h.3cqmetx)

[1.3 Definitions, Acronyms, and Abbreviations 6](#_heading=h.1rvwp1q)

[1.4 References 7](#_heading=h.4bvk7pj)

[1.5 Overview 7](#_heading=h.2r0uhxc)

[**2. General Description 7**](#_heading=h.1664s55)

[2.1 Product Perspective 7](#_heading=h.3q5sasy)

[2.2 Product Functions 7](#_heading=h.25b2l0r)

[2.3 User Characteristics 7](#_heading=h.kgcv8k)

[2.4 General Constraints 7](#_heading=h.34g0dwd)

[2.5 Assumptions and Dependencies 8](#_heading=h.1jlao46)

[**3. Specific Requirements 8**](#_heading=h.43ky6rz)

[3.1 External Interface Requirements 8](#_heading=h.2iq8gzs)

[3.1.1 User Interfaces 8](#_heading=h.xvir7l)

[3.1.2 Hardware Interfaces 8](#_heading=h.3hv69ve)

[3.1.3 Software Interfaces 8](#_heading=h.1x0gk37)

[3.1.4 Communications Interfaces 9](#_heading=h.4h042r0)

[3.2 Functional Requirements 9](#_heading=h.1baon6m)

[3.2.1 Storage of Customers’ Login Profiles 9](#_heading=h.3vac5uf)

[3.2.2 Personalized Customer Profiles 9](#_heading=h.2afmg28)

[3.2.3 Loyalty Memberships 9](#_heading=h.pkwqa1)

[3.2.4 Movie Listings and Details 9](#_heading=h.39kk8xu)

[3.2.5 Showtimes and Theater Information 9](#_heading=h.1opuj5n)

[3.2.6 Search Functionality 9](#_heading=h.48pi1tg)

[3.2.7 Seat Selection 10](#_heading=h.2nusc19)

[3.2.8 Payment Options 10](#_heading=h.1302m92)

[3.2.9 Payment Security 10](#_heading=h.3mzq4wv)

[3.2.10 Pricing and Discounts 10](#_heading=h.2250f4o)

[3.2.11 Payment Receipt 10](#_heading=h.haapch)

[3.2.12 Refunds and Cancellations 11](#_heading=h.319y80a)

[3.2.13 Ticket Format 11](#_heading=h.1gf8i83)

[3.2.14 Ticket Generation 11](#_heading=h.40ew0vw)

[3.2.15 Ticket Printing and Delivery 11](#_heading=h.2fk6b3p)

[3.2.16 Movie Reviews 11](#_heading=h.upglbi)

[3.2.17 Customer Support 11](#_heading=h.3ep43zb)

[3.3 Use Cases 12](#_heading=h.1tuee74)

[3.3.1 Customer Login Process 12](#_heading=h.4du1wux)

[3.3.2 Customer Searching For Movie 13](#_heading=h.2szc72q)

[3.3.3 Customer’s Payment Process 14](#_heading=h.184mhaj)

[3.4 Classes / Objects 15](#_heading=h.3s49zyc)

[3.4.1 <Class / Object #1> 15](#_heading=h.279ka65)

[3.5 Non-Functional Requirements 15](#_heading=h.meukdy)

[3.5.1 Performance 15](#_heading=h.36ei31r)

[3.5.2 Reliability 15](#_heading=h.1ljsd9k)

[3.5.3 Availability 15](#_heading=h.45jfvxd)

[3.5.4 Security 15](#_heading=h.2koq656)

[3.5.5 Maintainability 16](#_heading=h.zu0gcz)

[3.5.6 Portability 16](#_heading=h.3jtnz0s)

[**4. System Description 16**](#_heading=h.oowoehm74yne)

[4.1. Software Architecture Overview 17](#_heading=h.rz2d4egy66s4)

[4.1.1. Architectural Diagram 17](#_heading=h.advo4hsrai8e)

[4.1.2. SWA Description 18](#_heading=h.1d96cc0)

[4.1.3. UML Class Diagram 19](#_heading=h.2ce457m)

[4.1.4. Description of Classes 19](#_heading=h.rjefff)

[4.1.5. Description of Attributes 20](#_heading=h.3bj1y38)

[4.1.6. Description of Operations 21](#_heading=h.1qoc8b1)

[4.2. Development Plan and Timeline 21](#_heading=h.4anzqyu)

[4.2.1. Partitioning of Tasks 21](#_heading=h.2pta16n)

[4.2.2. Team Member Responsibilities 22](#_heading=h.etpakjs2wrru)

[**5. Verification Test Plan 23**](#_heading=h.3oy7u29)

[5.1. Introduction 23](#_heading=h.q0akdzj5m7yz)

[5.2. Test Plan Scope 23](#_heading=h.j0hshpji67ha)

[5.3. Test Levels 23](#_heading=h.pu4nly864whd)

[5.3.1. Unit Testing 23](#_heading=h.qxm8nom8jbnp)

[5.3.2. Functional Testing 23](#_heading=h.v6iiydyq2rir)

[5.3.3. System Testing 24](#_heading=h.rk4f5tka6k79)

[5.4. Test Cases 24](#_heading=h.d830vb984aps)

[5.4.1. Unit Tests 24](#_heading=h.6ltywqawmmy9)

[5.4.1.1. Test Case 1: Verify System’s Ticket Limit for Ticket Purchases 24](#_heading=h.j9lh9yrwzstq)

[5.4.1.2. Test Case 2: Verify Booking Timeout 24](#_heading=h.q10kfg9jh9dd)

[5.4.1.3. Test Case 3:Verify Senior and Military Discounts 25](#_heading=h.szgytn2xeq2r)

[5.4.2. Functional Tests 26](#_heading=h.i0ro917bdpun)

[5.4.2.1. Test Case 1: Verify Users Can Reset Password Using Forgot Password 26](#_heading=h.4lo0e3ki5b9o)

[5.4.2.2. Test Case 2: Verify User Can Search For Movies 27](#_heading=h.icu4filwq4pl)

[5.4.2.3. Test Case 3: Verify the ticket(s) purchased contain the proper theater, movie, seat information selected. 27](#_heading=h.w3ari3wbmevl)

[5.4.2.4. Test Case 4: Verify that the movies are being shown as "available" apply to those under 10 minutes past screening time or at a later date. 28](#_heading=h.rerypy5c6jhu)

[5.4.3. System Tests 28](#_heading=h.hm7qldqh0mgb)

[5.4.3.1. Test Case 1: Verify Confirmation Email After Transaction is Processed 28](#_heading=h.qhmw9pwvm0cy)

[5.4.3.2. Test Case 2: Verify User Can Login 29](#_heading=h.tsd84cjn31t0)

[5.4.3.3. Test Case 3: Verify Invalid User Login 29](#_heading=h.7hijxrvrz9nu)

[5.5. Test Case Samples 30](#_heading=h.suuab6m4kq33)

[**6. Database Management System 30**](#_heading=h.lfmq5yiz2tkn)

[6.1. Database ERB 31](#_heading=h.qj2d0dg26e15)

[6.2. SQL Database Tables 32](#_heading=h.z5go42d33q59)

[6.3. Data Security 33](#_heading=h.g4pbj7kp7uhv)

[6.3.1. Implement Data Encryption 33](#_heading=h.2albvluue2xb)

[6.3.2. Role-Based Access Control (RBAC) 33](#_heading=h.c08lu54wb3x)

[6.4. Scalability 34](#_heading=h.d3tcljknhtnf)

[6.4.1. Horizontal Scaling 34](#_heading=h.9970c1cbwhcx)

[6.4.2. Caching 34](#_heading=h.sww4h2g3jf3v)

[6.5. Data Integrity 35](#_heading=h.fosmmiiszzyn)

[6.5.1. Foreign Keys 35](#_heading=h.nfcdzpndfxnm)

[6.5.2. Constraints 35](#_heading=h.1fznwz2apbzc)

[6.6. Backup and Recovery 36](#_heading=h.bb20jritzd15)

[6.6.1. Regular Backups 36](#_heading=h.nf3dyjn08uve)

[6.6.2. Disaster Recovery Plan 36](#_heading=h.9njt01y93rcz)

[6.7. Performance Optimization 37](#_heading=h.bnw02j41wn5i)

[6.7.1. Query Optimization 37](#_heading=h.8ryvrpsvdtfd)

[6.7.2. Connection Pooling 37](#_heading=h.ofl9yvs6g740)

[6.8. Logging and Monitoring 38](#_heading=h.iv9cewvskryk)

[6.8.1. Implement Logging 38](#_heading=h.q4b5emanpv8p)

[6.8.2. Monitoring 38](#_heading=h.8qwkd2q9vc0u)

[6.9. Data Purging and Archiving 38](#_heading=h.617sf3iaofqv)

[6.9.1. Purging Policies 38](#_heading=h.f9mb05hiwkjj)

[6.9.2. Archive Legacy Data 38](#_heading=h.iexg7hj9toh0)

[6.10. Redundancy and High Availability 39](#_heading=h.mc75xsqtj8d6)

[6.10.1. Redundancy 39](#_heading=h.kvrn0h1t5ohv)

[6.10.2. High Availability (HA) 39](#_heading=h.kl0b12lw8qxj)

[6.11. Compliance 39](#_heading=h.6wdi14t7rftl)

[6.11.1. California Consumer Protection Act (CCPA) 39](#_heading=h.dsvwkzr4nfcg)

[6.12. Regular Maintenance 39](#_heading=h.7vjsaquwxh3j)

[6.12.1. Maintenance Tasks 39](#_heading=h.esrswj8hl7t3)

[6.13. Alternative Data Management 40](#_heading=h.nfx3mfofpdmo)

[6.13.1. NoSQL Database Alternative 40](#_heading=h.9uv8i13amm2o)

[6.13.2. SQL vs. NoSQL 40](#_heading=h.1d6yyoqrxcb2)

[6.13.3. Justification 40](#_heading=h.xhq0o0xgnjwx)

# **Introduction**

## Purpose

This Software Requirements Specification (SRS) is an in-depth description of the requirements for theMoviePlug Ticketing System. The document provides an overview of functional and nonfunctional requirements of movie ticketing software, constraints that impact how the system will function, and overall purpose of the system. This SRS assists and optimizes experience for developers, stakeholders, and investors with our system’s software development life cycle (SDLC) to produce an optimal system for our customers.

## Scope

The purpose of MoviePlug Ticketing Systemis to ease customers' movie ticket purchasing experience. By providing a convenient online web-based platform that has access to 20 partnered theaters across the San Diego region offering access to both regular and deluxe theater seating reservations. The system shall interface with several APIs and a relational database that streamlines processes such as movie searches, ticket purchases, seat reservations, payment and reviews. Customer accounts are available and loyal customers reap rewards through their loyalty points saved to their account, with their commitment to our product. Overall, the system will provide a user-friendly portal for movie-goers, that offers the ability to skip long lines on movie night.

## Definitions, Acronyms, and Abbreviations

| **Term** | **Definition** |
| --- | --- |
| Americans With Disabilities Act (ADA) | Act where all electronic information and technology must be accessible to those with disabilities. |
| Application Programming Interface (API) | Interface that allows two applications to communicate with each other. |
| Cascading Styling Sheets (CSS) | Styling sheet language associated with customization of HTML. |
| Frequently Asked Questions (FAQ) | Section populated with frequently asked questions and answers for users to read. |
| Graphical User Interface (GUI) | User interface that allows users to interact with the system. |
| Hypertext Markup Language 5 (HTML5) | Standard markup language for web browser sites. |
| Software Development Life Cycle (SDLC) | Process that assists with software development through planning, creation, testing. deployment processes. |
| Software Requirements Specification (SRS) | A document that completely describes all of the functions of a proposed system and the constraints under which it must operate. |
| World Wide Web Consortium (W3C) | Organization who developed international web standards. |

## References

TIEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

## Overview

The remaining sections of the SRS consist of the following:

**Section Two**: Provides a general informal overview of distinct characteristics, and functions that MoviePlug Ticketing System has to offer. This section also briefly mentions software constraints and dependencies that limit the system, covered in later sections..

**Section Three**: Takes a deeper-dive on both functional and nonfunctional requirements for the system. As well as interfaces utilized by the movie ticketing system to provide the optimal customer experience.

**Section Four**: Offers several models, and diagrams that depict analysis done for developing certain requirements.  
**Section Five:** Standard operating procedures (SOP) for updating this document.

## General Description

**Note:** This section provides a limited general overview of factors that affect the system.

## Product Perspective

MoviePlug Ticketing System is an independent web-based system that essentially acts as a hub for customers purchasing movie ticket reservations for 20 San Diego theaters in our network.

## Product Functions

MoviePlug shall provide a user-friendly interface for movie ticket purchasing. Allowing purchase options for guests and ability to create user accounts so customers can gain loyalty points that can be redeemed at 20 partnered locations.

## User Characteristics

Customers are expected to be tech-savvy enough to navigate the system’s simplistic internet platform via specified internet browsers to perform searches, finalize reservations and movie ticket purchase. Customers are also expected to navigate a comprehensive loyalty system connected to associated accounts to redeem rewards.

Administrators are expected to maintain access to manage the system to ensure optimal performance, to upkeep the system and override customer errors.

## General Constraints

The following are constraints that impact capabilities for MoviePlug Ticketing System:

* System’s platform shall be web-based accessible only on internet browsers and will be most compatible on browsers such as Microsoft Edge, Google Chrome, Safari.
* System must implement a queuing system to limit high traffic movie showing requests.
* System’s database must be able to showcase movies from 20 partnered theaters.
* System’s must utilize secured transactions utilizing Stripe API with the only acceptable forms of payment of Bitcoin, PayPal, and personal customer credit cards.
* Users’ personal information will be saved in our database and must be protected.
* Must prompt for logout of session if detection of multiple devices for single user login.
* System’s database must be capable of storing ten million concurrent users accounts.
* System must compile daily logs in the database of customer ticket purchases to assist admins and partnered theaters with overrides and refunds.
* System’s tickets must equip a unique NFT security protocol to prevent replication and resale.

## Assumptions and Dependencies

The following section lists factors that may impact of affect the system requirements:

* Incompatible browser usage may impact customer’s ability to perform tasks.
* Internet connectivity is a must in order to access the system's platform for customers.
* Movie tickets can only be purchased as far as two weeks prior to showtimes.
* Database will immediately refresh upon ticket purchase to ensure no overbooking.

# **Specific Requirements**

## External Interface Requirements

### **User Interfaces**

The user interface for the software shall be compatible with any browser such as Microsoft Edge, Mozilla, Chrome, and Safari by which user can access to the system. Must be scalable to fit screen to provide optimal visual experience for any internet connected device ie. mobile phone, laptop, tablet. Interface will comply with Americans with Disabilities Act (ADA)

### **Hardware Interfaces**

Since the application must run over the internet, all the hardware required to connect to the internet will be hardware interfaces for the system. As for e.g. Modem, WAN – LAN, Ethernet Cross-Cable.

### **Software Interfaces**

MoviePlug Ticketing Systemwill be supported the following:

* An interface connected to a database of showtimes and tickets available in the San Diego network of theaters.
* Stripe API that supports multiple payment methods such as Bitcoin, PayPal, and Credit Card.
* An interface to push and pull up-to-date ratings and reviews from RottenTomatoes and IMDB.
* Youtube API to support movie trailer videos.

### **Communications Interfaces**

MoviePlug Ticketing Systemshall utilize the HTTP protocol for communication over the internet and for the intranet communication will be through TCP/IP protocol suite.

## Functional Requirements

### **Storage of Customers’ Login Profiles**

* + - 1. System must store existing profiles within the database.
      2. System’s database shall store new user profiles
      3. System shall authenticate user credentials at login.

### **Personalized Customer Profiles**

* + - 1. System shall show purchase history within the profile.
      2. System will allow customers to set watch preferences.
      3. System shall recommend movie titles from previous history and preferences.

### **Loyalty Memberships**

* + - 1. The system will give customers the option to purchase loyalty memberships.
      2. The system shall give free, monthly and annual memberships that will offer tiered perks and redeemable rewards.

### **Movie Listings and Details**

* + - 1. The system shall display a list of currently showing movies.
      2. The system shall provide detailed movie information (title, synopsis, ratings, genre, cast, director)
      3. The system shall display movie posters and trailers.
      4. The system shall display upcoming movie releases and release dates.

### **Showtimes and Theater Information**

* + - 1. The system shall display showtimes for each movie.
      2. The system shall display theater information.
      3. The system shall allow users to select from the 20 partnered theater locations.

### **Search Functionality**

* + - 1. The system shall provide users with the ability to search for movies.
      2. The system shall include a user-friendly search interface.
      3. The system shall include search fields: movie title, genre, release date, and location.
      4. The system shall allow users to search a movie using one or more criteria simultaneously.
      5. The system shall display a list of movies that match the search criteria.
      6. Each movie result shall include movie title, showtimes, poster, and brief synopsis.
      7. Users shall have the option to sort search results by criteria: release date and showtime.

### **Seat Selection**

* + - 1. The system shall allow customers to choose between deluxe or regular seating..
      2. The system shall display 2D layout of theater for customers to pick deluxe seating.
      3. Customers shall not have the option to reserve specific seats in regular theaters.
      4. Upon reservation for both deluxe and regular seating, information is updated in the database and relayed to partnered theaters.

### **Payment Options**

* + - 1. The system shall support credit and debit card payment methods.
      2. The system shall support mobile wallet payment methods.
      3. The system shall support online banking payment methods

### **Payment Security**

* + - 1. The system shall integrate with Stripe a secure payment system service to process online transactions.
      2. The system shall utilize transport layer security (TLS) to increase data integrity.
      3. The system shall verify the payment details for accuracy and authorization.
      4. The system shall detect insufficient funds or failed authorization and notify users with the appropriate error message.

### **Pricing and Discounts**

* + - 1. The system shall support credit and debit card payment methods.
      2. The system shall support mobile wallet payment methods.
      3. The system shall support online banking payment methods.
      4. The customer shall be given the option to apply for a senior discount.
      5. The customer shall be given the option to apply for a youth discount.
      6. The customer shall be given the option to apply for a military discount.

### **Payment Receipt**

* + - 1. A payment receipt shall be generated and sent to the user’s registered email address.
      2. The receipt should contain transaction details, including ticket information, date, time, and the amount paid.

### **Refunds and Cancellations**

* + - 1. Users shall have the ability to request ticket refunds or cancellation within a specified timeframe.
      2. Refund requests will be subject to the refund policy and will be clearly communicated to users.

### **Ticket Format**

* + - 1. Ticket template should include essential information such as movie title, showtime, seat number, theater location and unique ticket identifier.

### **Ticket Generation**

* + - 1. The system shall send customers a confirmation email for their purchase.
      2. Separate ticket information without customers details will be sent to the partnered theater location.
      3. Generated email will contain QR code acting as a mobile ticket.
      4. Customers can choose to scan this code at the kiosk to receive a printed ticket.

### **Ticket Printing and Delivery**

* + - 1. The system shall provide customers with the option to receive electronic tickets via email or through a mobile app for digital storage.
      2. The system shall provide customers with the option to print tickets via printer.
      3. The system shall provide customers with the option to choose between different ticket formats, such as standard paper tickets or eco-friendly alternatives.

### **Movie Reviews**

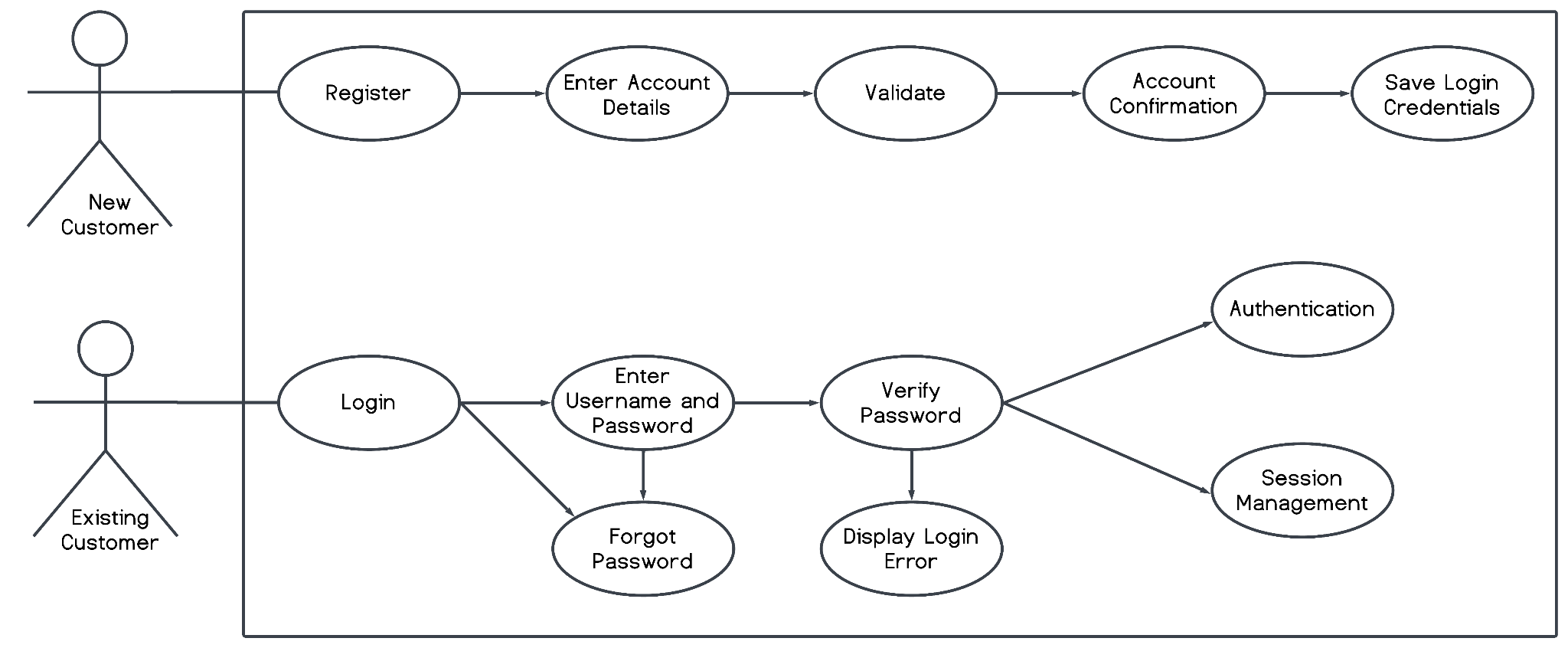
* + - 1. The system shall interface with Rotten Tomatoes and IMDB to pull live ratings and reviews of current showings.
      2. Customers shall have the option to post personal ratings through the interface.

### **Customer Support**

* + - 1. The system will clearly display helpdesk contact information on the website.
      2. The system will offer a Frequently Asked Questions(FAQ) section to answer commonly asked questions.
      3. The system shall offer onsite support that connects to admininstrators via minimizable chat in the bottom right-hand corner.

## Use Cases

### **Customer Login Process**



**Actors**

Existing and new customers.

**Brief Description**

Before this use case can be initiated customers must be on MoviePlug webpage. New and existing customers can log into the web-based interface of the system to browse, reserve seating, and purchase movie tickets.

**Initial Step-By-Step Description**

On the web page GUI in the upper-right corner an icon of a person shows. Upon clicking, the customer is brought to the login screen.

**New Customers:**

1. Customers without an existing account will click sign up to create a loyalty account.
2. A form will populate on the web page, prompting the customer to fill in required fields
3. Upon submitting the form, the customer’s information will be validated to check for any errors or fields not completed.
4. Customers will receive an email asking to verify email address and account creation.
5. New customer loyalty account information will be saved in the database, and converted to existing customers.
6. The system will redirect the customer back to the login page.

**Existing Customers:**

1. Customers will enter their username and password.
2. User authentication will occur to verify input, displaying error messages if not valid.
3. Three attempts will be given, on the third failed attempt account will be locked.
4. Customers can use “forgot password” if the account is locked or needs to reset password.
5. Upon valid input, the system will authenticate the user and ensure only one account session is active and return the user to the home page.

### **Customer Searching For Movie**



**Actors**

Existing or new customer

**Brief Description**

This use case describes how a customer, both new and existing, searches for a movie within the online movie ticketing system.

**Initial Step-By-Step Description**

**New Customers:**

1. The new customer opens the online movie ticketing system’s website.
2. The system presents the customer with a homepage.
3. The customer clicks on a search or browse movie option.
4. The system displays a search bar or a list of movie categories and genres.
5. The customer enters a search query into the search bar or selects a category/genre.
6. The system processes the customer’s query and presents a list of matching movie results.
7. The customer clicks on a movie from the list to view more details.
8. The system displays detailed information about the selected movie.
9. The customer can choose a showtime and proceed with booking tickets, or they can return to the list of movies and continue browsing.

**Existing Customers:**

1. The existing customer logs into their account on the online movie ticketing system.
2. The system presents the customer with their account dashboard or a personalized homepage.
3. The customer clicks on a search or browse movie option.
4. The system displays a search bar or a list of movie categories and genres.
5. The customer enters a search query into the search bar or selects a category/genre.
6. The system processes the customer’s query and presents a list of matching movie results.
7. The customer clicks on a movie from the list to view more details.
8. The system displays detailed information about the selected movie.
9. The customer can choose a showtime and proceed with booking tickets, or they can return to the list of movies and continue browsing.

### **Customer’s Payment Process**

**Actors**

New Customers, Existing Customers, Guests.

**Brief Description**

New and/or existing customers will need to access *MoviePlug* webpage so that they may be able to purchase tickets to a movie time. Using the web-based purchase process customers can purchase tickets for any available viewing. Customers should log in or proceed as a Guest. From the main page Customers can search for available showtimes and movies and proceed with the purchase process.

**Initial Step-By-Step Description**

1. Log in or continue as a Guest.

2. On the main there will be different movie titles displayed to choose from as well as an interactable search bar. If the desired movie isn’t available use the search bar to find the desired movie, if available.

3. Customer selects the desired movie tile.

4. Customer selects desired theater location.

5. Customer selects desired viewing time(s)

6. Is the theater “Deluxe Theater”?  
 *If not,* skip to **STEP 7**.  
 *If so,* select the desired seat or seats if purchasing multiple tickets.

7. Click “Proceed to Checkout”

8. Select preferred payment method (Visa, Mastercard, Apple Pay, Online banking).

9. Click Next, and input billing information if New Customer or Guest, or verify/edit existing billing information if Existing Customer.

10. Click “Proceed to Purchase”.

11. Once a transaction has been successfully processed, customers will be taken to a confirmation page where they can download/print/email their movie tickets.

12. Click the “X” from the confirmation page to exit and go back to the main page.

## Classes / Objects

### **<Class / Object #1>**

* + - 1. Attributes
      2. Functions

## Non-Functional Requirements

This section of the document overviewsMoviePlug Ticketing System’snon-functional requirements. It follows specific constraints placed on the system that impact the system’s overall performance, reliability, security, etc.

### **Performance**

MoviePlug Ticketing System’sperformance on the customer’s end depends on overall personal internet access of the user's personal equipment ie. their router/modem connectivity speeds and device hardware capabilities.

### **Reliability**

To ensure our system’s reliability and up-time, hourly database synchronization will occur and daily logs will be saved to prevent data loss. Bi-weekly updates will occur to upkeep the database to prevent any security faults.

### **Availability**

MoviePlug Ticketing System shall be accessible 24/7 to customers barring any hosting site and

database server faults.

### **Security**

MoviePlug Ticketing Systemshall utilize multiple security procedures and protocols to ensure the safety of our customers and partnered theaters:

* User logins will be capped at three password attempts, if attempts are exceeded the account is locked.
* Customer credit card information will never show the full number at purchase, only the last four digits.
* Databases shall be encrypted to protect customer’s personal information.
* Bi-weekly updates to system databases shall occur,
* Daily logs and synchronization shall occur to prevent data loss.
* IP detecting block bots are deployed to prevent bulk ticket purchases.
* Transport Layer Security (TLS) shall be utilized during purchase and login actions.

### **Maintainability**

The system’s database shall utilize a redundant array of independent disks (RAID) with minimal data redundancy to prevent data loss. Web page will be hosted by a reliable web-hosting server. Third-party plugins shall be verified for compatibility regularly. MoviePlug Ticketing Systemshall have a dedicated administrator team to upkeep web interfaces.

### **Portability**

The GUI interface of the system shall utilize Hypertext Markup Language 5 (HTML5), Cascading Style Sheets (CSS), and JavaScript to enhance web page experience that conforms to World Wide Web Consortium (W3C) standards. Ensuring accessibility on the majority of internet browsers and all customer operating systems. Our database will operate using a Microsoft SQL Server as our relational database to store records, accounts, and titles.

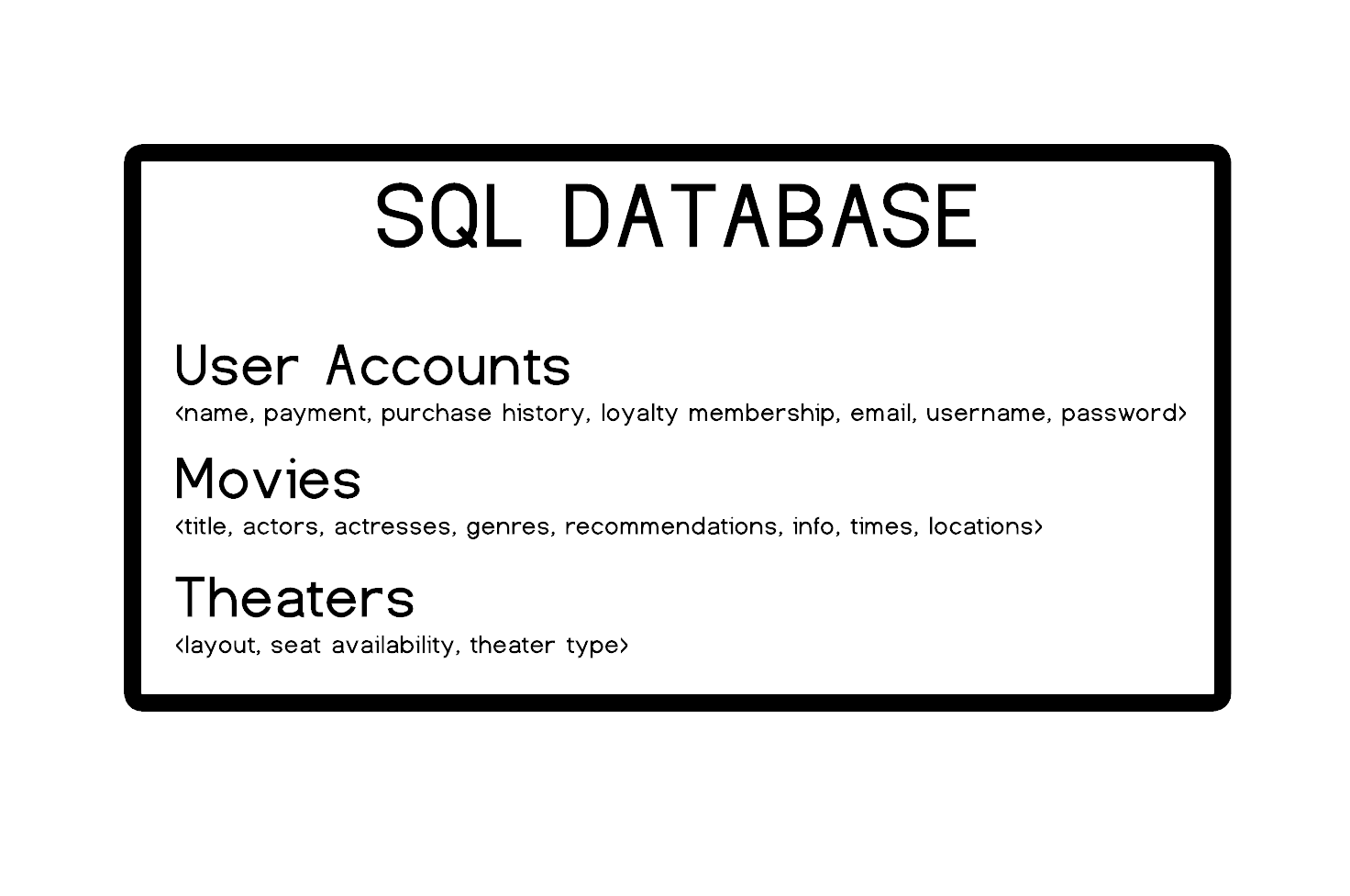
# **System Description**

The purpose of MoviePlug Ticketing Systemis to ease customers' movie ticket purchasing experience. By providing a convenient online web-based platform that essentially acts as a hub for customers purchasing movie tickets. It has access to 20 partnered theaters across the San Diego region offering access to both regular and deluxe theater seating reservations. The system shall interface with several APIs and a relational database that streamlines processes such as movie searches, ticket purchases, seat reservations, payment, and reviews. Customer accounts are available and loyal customers reap rewards through their loyalty points saved to their account, with their commitment to our product. Overall, the system will provide a user-friendly portal for movie-goers that offers the ability to skip long lines on movie night.

## Software Architecture Overview

### **Architectural Diagram**

### **SWA Description**



The following describe the architecture and functionality of the UI interface for customers.

## 

## 

## 

## 

## 

## 

## 

## 

### UML Class Diagram

## 

### **Description of Classes**

● **User()** - Contains user information and validates information with DB.

● **Booking()** - Contains booking information like booking id, the user who is booking and screenings.

● **Payment()** - Contains user input such as card information and uses 3rd party API to process payment, and transaction information.

● **Ticket()** - Contains all ticket details such as user name, booking confirmation, and assigned seat if applicable.

● **Seat()** - Contains the user designated seat provided that the theater in question is an assigned seating theater.

● **Theater()** - Contains theater branch information such as name, theater id and capacity among other things.

● Screening() - Contains information in regards to actual screening date, time, theater location, film name.

● **Movie()** - Contains actual movie information such as title, genre, date released, and movie identifier (internal purposes).

### **Description of Attributes**

● **User()**

○ user\_id - string - user display name

○ username - string user login

○ password - string user password

○ email - string email address

● **Booking()**

○ booking\_id - string - booking confirmation

○ user - - strings - information coming from the User() class containing user info

○ screening - multiple data types coming from the Screening() class

● **Payment()**

○ payment\_id - string - payment confirmation id returned from 3rd party API

○ amount - int - total amount being charged for ticket(s)

○ payment\_date - int - date transaction was made in xx/xx/xxxx xx:xx:xx format

● **Ticket()**

○ ticket\_id - int - ticket id unique to theater, movie, and user

○ booking - multiple data types coming from the booking class

○ seat - string - passed from the Seat() class

● **Seat()**

○ seat\_id - string - composed from the theater information as well as seat number

○ theater - string/int - information passed from the Theater() class

○ seat\_number - int - seat number within a given row

○ is\_occupied - bool - flag that determines if seat is already taken

● **Movie()**

○ movie\_id - int - unique ID that corresponds to movie within theater (internal use)

○ title - string - movie title

○ genre - string - movie genre

○ release\_date - int - contains date of when the movie will be released in theaters in xx/xx/xxxx format

● **Screening()**

○ screening\_id - unique ID that is composed of movie information such as location, movie title, movie times (internal use)

○ movie - string - information brought over from the Movie() class contains movie

○ theater - object - contains theater information

○ date\_time - int - Date and time of the screening

● **Theater()**

○ theater\_id - int - Unique identifier for the theater (internal use)

○ name - string - Name of the theater

○ capacity - int - Maximum seating capacity of the theater.

### Description of Operations

● **User()**

○ Register() - Registers user from user input information

○ Login() - Using user input information validates credentials with DB information

● **Booking()**

○ Add\_Ticket() - Adds ticket to the purchase, marks seat as unavailable (if applicable), updates theater capacity, and will queue for printing at a later step.

● **Payment()**

○ Process\_Payment() - 3rd party API that processes payment

## Development Plan and Timeline

## 

### **Partitioning of Tasks**

To ensure the four month timeline is met these have been assigned to the following teams:

● **Planning:**

* **Project Manager** - POC: Justine Rosario Rubenero

● **Research**:

* **UX/UI Team** - POC: Shanique Doodlebob

● **Gather Requirements:**

* **Project Manager** - POC: Justine Rosario Rubenero
* **UX/UI Team** - POC: Shanique Doodlebob

● **Design:**

* **UX/UI Team** - POC: Shanique Doodlebob

● **Build:**

* **Front-End Development Team** - POC: Terrance Clifford III
* **Back-End Development Team** - POC: Chris P. Baycoon

● **Testing:**

* **Front-End Development Team** - POC: Terrance Clifford III
* **Back-End Development Team** - POC: Chris P. Baycoon
* **QA Team** - POC: Naomi Yerba
* **Tester Team** - POC: Wallace Steffingshire

● **Final Review:**

* **Project Manager** - POC: Justine Rosario Rubenero
* **Software Architect** - POC: Davy Jones
* **QA Team** - POC: Naomi Yerba

● **User Testing:**

* **UX/UI Team** - POC: Shanique Doodlebob
* **QA Team** - POC: Naomi Yerba
* **Tester Team** - POC: Wallace Steffingshire

● **Refining:**

* **Software Architect** - POC: Davy Jones
* **Front-End Development Team** - POC: Terrance Clifford III
* **Back-End Development Team** - POC: Chris P. Baycoon

● **Launch:**

* **Project Manager** - POC: Justine Rosario Rubenero
* **Software Architect** - POC: Davy Jones

### **Team Member Responsibilities**

● **Project Manager**: Justine Rosario Rubenero

* Develop a project plan and lead product planning meetings with teams.
* Develop functional specifications.
* Plan, estimate, and manage resources and the project budget.
* Conduct initial interviews with stakeholders.
* Evaluate and ensure timeline processes.
* Launch and release.

● **UX/UI Team |*****UX/UI Lead***: Shanique Doodlebob

* Conduct user interviews.
* Design website wireframe.
* Research competition.
* Gather Requirements

● **Front-End Development Team | *FE Lead*:** Terrance Clifford III

* Build the features laid out in the Sprint.
* Update the status of the software project to the Project Manager and Lead.

● **Back-End Development Team** | ***BE Lead:*** Chris P. Baycoon

* Integrate web services and APIs.
* Build servers, frameworks, databases.
* Integrate with front-end capabilities.

● **QA Team | *QA Lead:*** Naomi Yerba

* Evaluate execution of processes and/or production.
* Identify and document deviations in SOPs.
* Provide feedback about the results of the quality assurance.

● **Tester Team | *Test Lead:*** Wallace Steffingshire

* Understand requirements, and constraints to create and run test cases.
* Build and run test cases to detect bugs and report them.

● **Software Architect:** Davy Jones

* Define the technical and functional architecture of the overall system.
* Guide developers in the design and implementation of the solution.
* Ensure functionality of the final product.

# **Verification Test Plan**

## Introduction

The purpose of this Verification Test Plan is to provide an in depth overview of functional, unit and system test plans that satisfy business requirements.

## Test Plan Scope

The verification tests will target the following aspects of the software system:

* Unit Testing - Individual class methods and functions.
* Functional Testing - The interaction between different classes and functions.
* System Testing - The end-to-end behavior and performance of the software system.

## Test Levels

### **Unit Testing**

Unit tests analyze and evaluate at a basic level of testing individual components of MoviePlug Ticketing System software. These components can include objects and classes within the system.

### **Functional Testing**

Functional testing is a test level that focuses on verifying the functional aspects and features of the software application. The primary objective of functional testing is to ensure that the software behaves as expected, meets its functional requirements, and delivers the intended functionality to end-users.

### **System Testing**

System testing focuses on assessing the system's behavior and performance as a whole rather than individual components. System testing verifies that all the components work together seamlessly and meet the system's non-functional requirements, such as performance, security, and scalability.

## Test Cases

### **Unit Tests**

#### **Test Case 1: Verify System’s Ticket Limit for Ticket Purchases** A black and white sign Description automatically generated

* + **Description:** This test will verify that the seating() class function communicates with the event management system (EMS) API to ensure conditions are set to limit the number of premium seats selected on the user's interface. The maximum number of seats a user can reserve is 20 seats utilizing MoviePlug. Upon finalizing movie, theater and showtime selection, the user will be navigated to the seat selection page if a premium theater is chosen. A graphical 2D layout of the theater allows users to select the specific seat(s) they chose. After exceeding the maximum number of seats (20) a popup error message should alert the user that the ability to choose any more seats is unable to be processed. This should mitigate surplus unfair ticket purchases. Possible faults that could occur is that the condition isn’t properly set within the external API or that seating() class doesn’t pull seating information from the database.
  + **Test Input:**
    - Navigate to the seating selection page.
    - Select 21 seats on the 2D-seating map.
    - Click the "Next" button to navigate to the ticket selection page.
  + **Expected Output:** User receives an error message indicating the maximum number of seats have been selected.
  + **Pass/Fail Criteria:  
    Pass:** 
    - System alerts number of seats selected exceed limit.
    - Booking() function

**Fail:**

* + - System allows users to select over 20 seats.

#### **Test Case 2: Verify Booking Timeout**

* + **Description:** This test case is designed to verify the behavior of the Booking() class and its transmission to the Payment() class. When a user initiates the movie booking process but does not proceed to payment for an extended period (five minutes). The expected result is that the system should detect the booking timeout condition and display an error timeout pop up prompt.
  + **Test Input:**
  + Open movie ticketing system application.
  + Log in as a registered user or use a test account.
  + Navigate to the “Movie Selection” section.
  + Select a movie for booking.
  + Choose the number of tickets and select seats.
  + Proceed to the payment page.
  + At this point, pause the test to simulate the user’s inactivity.
  + Wait for an extended period (e.g. 10 minutes) without taking any further action.
  + **Expected Output:**
    - After the defined waiting period, the system should detect the user’s inactivity and booking timeout.
    - The system should display an error timeout page with a clear message informing the user that the booking session has timed out due to inactivity.
    - The error timeout page may also provide instructions on how to restart the booking process or return to the movie selection page.
  + **Pass/Fail Criteria:**
    - The test passes if the system correctly identifies the booking timeout condition and displays the error timeout page.
    - The error timeout message should be clear and informative.
    - The user should be guided on how to proceed or restart the booking process.

#### **Test Case 3:Verify Senior and Military Discounts**

* + **Description:** The purpose of this test is to verify that Users who have been identified as either a Senior Citizen or a Military Service member or an affiliate receive a discount on the purchase of movie tickets regardless of the movie, theater, or screening selected. This test case verifies that the Payment() class receives the valid information containing the discount and that it is properly applied to the transaction.
  + **Test Input:**
    - Go through the process of selecting a movie, screening, theater location, seat (if applicable).
    - Just prior to confirming purchase look at the Cart Summary there should be an inline credit named "Discount" in parentheses for "$xx.xx" dollar amount.
    - After submitting payment the proper amount is charged to the payment method.
  + **Expected Output:** Cart Summary shows "x" amount of tickets added to cart minus "$xx.xx" dollar amount for Senior/Military Discount. After purchasing, the amount reflects ticket price minus appropriate discount.
  + **Pass/Fail Criteria:**

**Pass:**

* + - The test passes if the total amount charged reflects the amount of a ticket or multiple tickets minus the amount of the discount. (i.e. if ticket is $15, and Military/Senior Discount is 10% then the amount charged per ticket for a customer who is discount eligible would be $13.50/ticket)

**Fail:**

* + - This test will fail in any event where the total amount charged is anything other than $13.50/ticket for users who are discount eligible.

### **Functional Tests**

#### **Test Case 1: Verify Users Can Reset Password Using Forgot Password** A screenshot of a computer screen Description automatically generated

* **Description**: This black box test will verify functionality of User() class function , integration with Email API, database updates as well as server communications. Main focus is to ensure the user(s) can properly reset passwords when either locked out or the password is no longer known. Forgot Password button is found on the login page, once clicked it will trigger password recovery/reset transactional email sent by API. User(s) should receive an email containing a link that will navigate them to the SSL secure password reset page and will be prompted to input a new password. Upon successful input of a valid password, the new password should be updated within MoviePlug’s internal database and authentication server and work during the next login. Possible faults could occur at the internal database with failure to update the user's password information. Process is dependent on the Email API to send email to the user for the reset process to occur.
* **Test** **Input**:
* Navigate to the login page.
* Click the "Forgot Password" button.
* Enter valid email associated with account.
* Click the "Reset Password" button.
* Receive password reset email.
* Redirect to password reset page.
* Enter a valid new password and verify.
* Click "Confirm" Button.
* Return to the login page and try the new password.
* **Expected** **Output**: User enters a valid email associated with an account to get sent a password reset link to secure SSL encrypted page to reset their password. Upon valid input of a new password that follows password requirements and user confirmation to reset password, the password associated with the account will be changed for login credentials.
* **Pass/Fail Criteria**:

**Pass:**

* Password meets security criteria and is accepted by authentication.
* Database updates and saves the new password.

**Fail:**

* User() class failed to retrieve new password from database.
* The internal database didn’t store the updated password.
* Email API didn’t successfully initiate transactional email.

#### **Test Case 2: Verify User Can Search For Movies**

* **Description**: This test case is designed to verify the functionality of the Movie() class and Screening() class search feature. The purpose is to confirm that when a user searches for a specific movie title, the system correctly displays the corresponding movie listings.
* **Test Input**:
* Open the movie ticketing system application.
* Navigate to the “Search” or “Find Movies” section.
* In the search bar, enter the title of a specific movie (e.g. “The Avengers”).
* Initiate the search by pressing the “Search button” or hitting “Enter”.
* **Expected Output**:
* The system should process the search query and retrieve movie listings matching the entered movie title.
* The displayed listings should include movies with titles that contain or match the search query, including exact title matches.
* Each movie listing should provide essential details, such as the movie title, showtimes, and theater locations
* **Pass/Fail Criteria**:
* The test passes if the system successfully retrieves and displays movie listings that match the entered movie title.
* The displayed listings should accurately correspond to the search query.
* The listings should provide relevant movie details.

#### **Test Case 3: Verify the ticket(s) purchased contain the proper theater, movie, seat information selected.**

* **Description**: The purpose of this test is is to ensure that the Booking(), Ticket(), Seat(), Theater(), Movie(), and Screening() classes all function accordingly and that they all pass the proper information over to the Ticket() class which will print out the adequate data as selected and paid for by the user.

* **Test Input**:
* Search for a movie.
* Select a theater.
* Select a screening time.
* Select number of desired tickets.
* If applicable, select a seat(s).
* Confirm cart & proceed to finalize purchase.
* Confirmation page and ticket(s) will populate and prompt you to print/save.
* Confirm that ticket data concurs with all options chosen by the customer.
* **Expected Output**:Printed/Saved ticket will contain all information as chosen by customer (i.e. customer chose to buy 3 tickets for movie "A" in theater "X" and seats "E8,E9,E10", printed tickets will reflect that information
* **Pass/Fail Criteria**:

**Pass:**

* This test case passes if the tickets printed properly reflect the users order (i.e. if User selected movie “X”, at screening time “xx:xx”, in a premium theater “T” and select seat “A1”. Then the ticket should reflect the user name with movie “X” in theater “T” allocated to seat “A1”.

**Fail:**

* This test case fails if any component of the ticket does not match the user’s order (i.e. incorrect movie selection, screening time, theater location or seat selection *(if applicable)*).

#### **Test Case 4: Verify that the movies are being shown as "available" apply to those under 10 minutes past screening time or at a later date.**

* **Description**: The purpose of this test is to ensure that the system will check multiple classes throughout the system like the Movie(), and Screening() class to make sure that the system will allow a user to purchase a ticket for a movie that has not been playing for longer than 10 minutes past the screening time.
* **Test Input**:
* Search for a movie.
* Select a theater. (note that some locations will be available while others are grayed out meaning they are not available)
* Select a screening time, for those theaters that are marked as available.
* **Expected Output:** Movie "A" will be available to be seen at theater "X" assuming the movie hasn't been rolling for more than 10 minutes or if it is for any screening at a later time.
* **Pass/Fail Criteria**:

**Pass:**

* This test case passes if the application allows the user to purchase a ticket for a screening that is either at a later time or if it has not been more than 10 minutes past the screening time. (i.e. movie starts at 4pm, system should allow user to purchase ticket until 4:10pm)

**Fail:**

* This test case fails if the application allows the user to purchase a ticket for a screening that is more than 10 minutes past the screening time.

### **System Tests**

#### **Test Case 1: Verify Confirmation Email After Transaction is Processed**

A diagram of a payment process

Description automatically generated

* + **Description**: This test will verify system’s communication between internal database storing user contact information, and external email API that delivers transactional email triggered by purchase from MoviePlug website. Main task is to verify Payment() class functions integrated with other class functions. After the user navigates through the payment page and payment is processed between bank and external payment API for preferred payment method, the user will automatically be navigated to the confirmation page. Upon successful payment, corresponding external QR Code and Email APIs will be triggered and users should receive a confirmation email containing the details of their purchase, along with access to their purchased ticket(s). Possible faults can be isolated to external APIs, whether it be payment status not confirmed, QR Code not generated, Email API not receiving corresponding trigger message to initiate email. User error that can occur is an incorrect email address stored in the database.
  + **Test Input**:
  + Navigate to the payment page.
  + Pay with preferred payment method.
  + Confirm payment information
  + Click "Checkout" Button
  + **Expected Output**: User receives on screen confirmation with text stating confirmation email with tickets information has been sent to email address on file. Upon checking the user checking their personal email inbox, corresponding email from MoviePlug will be delivered confirming purchase details and NFT ticket access.
  + **Pass/Fail Criteria**:

**Pass**:

* + - Transactions must be processed.
    - User(s) must receive a confirmation email sent from transactional email API.

**Fail**:

* User(s) does not receive the system's automatically generated confirmation email.

#### **Test Case 2: Verify User Can Login**

* + **Description**: This test case aims to verify the functionality of the software system’s login authentication process, specifically assessing the interactions with the User() class and its associated functions. The test objective is to confirm that when a user logs in with valid credentials, the User() class, in conjunction with the system, correctly authenticates the user and displays the user’s page.
  + **Test** **Input**:
  + Open the software system application.
  + Navigate to the “Login” or “Sign In” section.
  + Enter valid user credentials, which include a username or email and a password.
  + Initiate the login process by clicking the “Login” or “Submit” button.
  + **Expected** **Output**:
    - The system should process the login credentials and perform user authentication.
    - Upon successful authentication, the system should redirect the user to the user’s page, dashboard, or main account page.
    - The user’s page should display relevant information, user-specific information, or a personalized dashboard.
  + **Pass/Fail Criteria**:
    - The test passes if the system successfully authenticates the user with valid login credentials.
    - The system should then direct the user to the user’s page.
    - The user’s page should display user-specific information and content.

#### **Test Case 3**: **Verify Invalid User Login**

* + **Description**: The test case is designed to assess the behavior of the software system’s login authentication process when a user attempts to log in with incorrect or invalid credentials. The primary objective is to confirm that the system correctly identifies incorrect login attempts and responds by displaying an error page with clear error messages.
  + **Test Input**:
    - Open the software system application.
    - Navigate to the “Login” or “Sign In” section.
    - Enter invalid user credentials, which include a username or email and a password. These credentials intentionally contain errors, such as an incorrect password.
    - Initiate the login process by clicking the “Login” or “Submit” button.
  + **Expected Output**:
    - The system should process the login credentials provided by the user.
    - When incorrect credentials are detected, the system should recognize the login attempt as unsuccessful.
    - The system should redirect the user to an error page that provides clear error messages, indicating that the login attempt has failed due to incorrect credentials.
    - The error page may also include instructions on how to recover the account, reset the password, or return to the login page.
  + **Pass/Fail Criteria**:
    - The test passes if the system correctly identifies incorrect login attempts and redirects the user to the error page.
    - The error page should clearly communicate the reason for the failed login attempt (incorrect credentials).
    - The test should ensure that the error message is user-friendly and provides guidance for the user on how to proceed.

## Test Case Samples

View Test Case Document Here: [Test Case Matrix](https://github.com/rosasilipino/MoviePlug-Ticketing-System/blob/main/SDS%20TEST%20PLAN/Movie%20Plug%20Test%20Case%20Matrix.xlsx)

# **Database Management System**

A data management strategy is a comprehensive plan that outlines how an organization collects, stores, manages, secures, processes, and ensures the quality of its data assets. The primary purpose of a data management strategy is to enable an organization to effectively and efficiently use its data to achieve its business objectives.

## Database ERB

## SQL Database Tables

SQL Database Tables can be found here: [SQL Database Tables](https://github.com/rosasilipino/MoviePlug-Ticketing-System/blob/main/Architecture%20Design%20w%3AData%20Mgmt./Database%20Tables.xlsx)

## Data Security

Data security within the MoviePlug ticketing system is of paramount importance to protect sensitive user information and maintain the integrity and confidentiality of data. The system employs a mulit-faceted security strategy, which include **Implementation of Data Encryption** and **Role-Based Access Control (RBAC)**.

### **Implement Data Encryption**

One of the cornerstones of our data security approach is the implementation of robust data encryption. This encompasses the encryption of sensitive data, including encryption of sensitive user data, including not only passwords, but also financial information, to ensure that such data remains inaccessible to unauthorized parties.

**Password Encryption**

During the user registration and login processes, user passwords are subjected to advanced encryption methods. This ensures that the actual password is never stored directly in the database. Instead, a cryptographic hash, generated using a secure and industry-recognized algorithm, is stored in the database. This hash is virtually impossible to reverse-engineer, offering a high level of protection against password-related breaches.

**Financial Data Encryption**

Financial information, such as credit card details, undergoes encrypted before transmission and storage. Secure communication protocols, such as TLS/SSL, are employed to safeguard data during transmission between the user's browser and the system's servers.

### **Role-Based Access Control (RBAC)**

To maintain data security and ensure that access to various parts of the database is meticulously controlled, the system implements a Role-Based Access Control (RBAC) framework. RBAC assigns users to specific roles based on their responsibilities and permissions within the system.

**User Roles**

Users are classified into distinct roles, each of which grants specific access rights and privileges. For example, database administrators are assigned an "Admin" role, which provides them with comprehensive access to manage and administer the database. Meanwhile, regular users, such as customer support staff, are assigned roles with tailored access, typically limited to querying user information for support purposes.

**Access Control Policies**

Access control policies are meticulously defined and applied to ensure that individuals are granted only the access rights necessary to fulfill their roles. This prevents unauthorized access to sensitive data and system components.

**Fine-Grained Control**

RBAC allows for fine-grained control over who can perform what actions within the system, reducing the risk of data breaches resulting from unnecessary or excessive access permissions.

## Scalability

Scalability is a fundamental consideration in the architecture of the MoviePlug ticketing system, ensuring that the system can efficiently handle increased traffic, maintain responsiveness, and support growing user demand. The system employs a combination of strategies to achieve these scalability goals which include **Horizontal Scaling** and **Caching**.

### **Horizontal Scaling**

The system is designed with horizontal scalability in mind, allowing it to effectively accommodate heightened traffic levels. To achieve this, the following techniques, such as sharding or partitioning, are implemented

**Sharding**

Sharding involves the division of the database into smaller, more manageable partitions, known as shards. Each shard contains a subset of the data. For example, showtime and booking data for different geographic regions or movie categories may be stored in separate shards. This technique enables parallel data processing, reducing the load on any single database server and distributing the traffic evenly across multiple servers.

**Partitioning**

Partitioning involves breaking down large database tables into smaller, more manageable partitions based on specific criteria, such as time, location, or movie title. For example, the bookings table may be partitioned by date, ensuring that data for different time intervals is stored separately. This enhances query performance and facilitates load distribution.

**Load Balancing**

Load balancers are deployed to distribute incoming traffic across multiple database servers. This ensures that the system can efficiently allocate and manage resources, preventing any single server from becoming a performance bottleneck.

### **Caching**

To further enhance system performance and alleviate the load on the primary database, caching mechanisms, such as Redis, are integrated

**Redis Caching**

Redis, an in-memory data store, is utilized to store frequently accessed and relatively static data, such as movie listings, showtimes, and theater details. This data is temporarily stored in memory, allowing for rapid retrieval and reducing the need to query the database repeatedly. By caching this data, the system significantly improves response times for users and conserves database resources for more complex queries and transactional operations.

**Cache Invalidation**

To ensure data consistency, the caching mechanism employs cache invalidation strategies that update or remove cached data when underlying database records are modified. This guarantees that users are always presented with the most current information, even when data is frequently changing.

## Data Integrity

Data integrity is a fundamental aspect of the MoviePlug ticketing system, ensuring that data remains accurate, consistent, and reliable. The system employs several techniques to enforce and maintain data integrity which include the use of **Foreign Keys** and **Constraints** for data validation.

### **Foreign Keys**

The system enforces referential integrity by strategically using foreign keys to establish relationships between tables within the database. Foreign keys ensure that data dependencies and relationships are maintained, preventing inconsistencies and data anomalies.

**User-Booking Relationship**

To maintain data integrity, the system uses foreign keys to associate bookings with specific users. When a user makes a booking, the user's unique identifier (user\_id) is stored in the "Bookings" table as a foreign key. This establishes a link between the user and their booking, ensuring that bookings are attributed to the correct users and preventing orphans (unassociated) records.

**Movie-Showtime Relationship**

Foreign keys are employed to link movies to their respective showtimes. The "Showtimes" table contains a foreign key that references the movie's unique identifier (movie\_id). This establishes a clear connection between movies and the showtimes during which they are screened, ensuring data consistency.

### **Constraints**

The system employs data validation constraints, including CHECK constraints, to maintain data integrity and prevent the entry of invalid or inconsistent data.

**Seat Availability Constraint**

To ensure that only valid seat selections are made during the booking process, a CHECK constraint is applied to the "Seats" table. This constraint verifies that the status of a seat corresponds to its availability for the selected showtime. Seats marked as unavailable cannot be booked, maintaining data accuracy and preventing overbooking.

**Date and Time Validity Constraint**

CHECK constraints are utilized to validate the accuracy of date and time entries, particularly in the "Showtimes" table. These constraints verify that showtimes are scheduled for valid dates and do not overlap, preventing scheduling conflicts and data inconsistencies.

**Rating and Genre Validity Constraint**

When movie details are entered or updated, CHECK constraints are applied to ensure that movie ratings and genres conform to predefined criteria. This guarantees that movie information remains accurate and adheres to industry standards.

## Backup and Recovery

Backup and recovery are vital components of the MoviePlug ticketing system, providing mechanisms to safeguard data and ensure the system’s continued operation, even in the face of unforeseen disruptions or failures. The system employs **Regular Backups** and a **Disaster Recovery Plan** in order to achieve this.

### **Regular Backups**

The system rigorously adheres to a schedule of regular database backups, ensuring that data remains safeguarded against loss in the event of system failures, data corruption, or human errors.

**Scheduled Backup Frequency**

The system is configured to conduct regular, automated database backups at predefined intervals, such as daily or hourly, depending on the system's requirements. Frequent backups minimize the potential data loss in case of an adverse event.

**Backup Storage**

Backups are securely stored in redundant and geographically separate locations to prevent data loss due to local incidents, such as hardware failures or natural disasters. This redundancy ensures that a copy of the data is readily available for recovery purposes.

**Point-in-Time Recovery**

In addition to full backups, the system implements point-in-time recovery to allow restoration to specific moments in time. This feature is particularly useful in addressing issues such as data corruption.

### **Disaster Recovery Plan**

The system has a comprehensive disaster recovery plan in place to ensure the continuity of database operations and system functionality in the event of unexpected failures, catastrophic events, or disasters.

**Backup Restoration Procedure**

The disaster recovery plan outlines the step-by-step procedures for restoring data from backups in case of a disaster. This includes procedures for initiating a full system recovery and reestablishing database operations.

**Testing and Drills**

Regular testing and drills are conducted to validate the effectiveness of the disaster recovery plan. This includes simulated recovery scenarios and procedures to train personnel on emergency response.

**Data Restoration Validation**

The disaster recovery plan includes processes to validate the integrity of restored data to ensure that the recovered database is free from corruption or inconsistencies.

## Performance Optimization

Performance optimization is a critical aspect of the MoviePlug ticketing system, ensuring that the system operates efficiently and provides users with a responsive and seamless experience. The system employs Query Optimization and Connection Pooling to achieve this.

### **Query Optimization**

The system places a strong emphasis on query optimization to minimize response times and enhance the efficiency of database operations.

**Indexing**

The system utilizes indexing on database tables to expedite data retrieval. Indexes are carefully designed and implemented on columns that are frequently used as search criteria, such as user IDs, movie titles, and showtimes. Proper indexing significantly accelerates query performance.

**Caching**

To further reduce the load on the database, caching mechanisms are utilized. Frequently accessed data, such as movie listings, showtimes, and theater details, are temporarily stored in memory (e.g., Redis). This minimizes the need to execute repetitive queries, resulting in faster response times for users.

**Denormalizing**

In certain cases, data denormalization is employed to consolidate data from multiple tables into a single table. This approach can optimize specific queries by reducing the need for complex joins and data retrieval operations.

### **Connection Pooling**

Connection pooling is a key strategy used to efficiently manage database connections, reduce overhead, and enhance system performance

**Pooling Configuration**

The system is configured with connection pooling mechanisms to maintain a pool of pre-established database connections. This pool is managed and shared among multiple application components, minimizing the overhead associated with frequently opening and closing connections.

**Resource Management**

Connection pooling includes resource management features that automatically release idle or unused connections, ensuring that resources are efficiently allocated and preventing resource exhaustion.

**Scalability**

Connection pooling mechanisms are designed to scale with increasing user demand. As traffic grows, additional connection pools can be added, and the system can efficiently distribute connections across multiple database servers.

## Logging and Monitoring

Logging and monitoring are integral components of the MoviePlug ticketing system, providing insight into database activities, performance, and the overall health of the system. These measures help ensure accountability, diagnose issues, and optimize system performance. The use of Implement Logging and external **Monitoring** will achieve this.

### **Implement Logging**

Logging records activities within the database, such as login attempts, data modifications, and schema changes, which are crucial for auditing and compliance with regulations like CCPA. Logs can be used to track and detect security incidents, unauthorized access, or unusual activities within the database.

### **Monitoring**

We maintain MoviePlugs efficiently by tracking the performance of the database, including response times, query execution times, resource utilization (CPU, memory, disk I/O), among other things to identify performance bottlenecks. We also monitor the utilization of system resources to ensure they are within acceptable limits and to plan for scalability when resource consumption increases.

## Data Purging and Archiving

Data Purging policies are designed and implemented to systematically remove outdated or no longer needed records, maintaining a lean and efficient database. Key components of this strategy include **Purging Policies** and the ability to **Archive Data**.

### **Purging Policies**

MoviePlug purging policies are implemented by removing unused or irrelevant data, data purging can help improve the performance of database queries and transactions, as there is less data to scan and process. We also contribute to compliance by purging personal data when it is no longer needed.

### **Archive Legacy Data**

Due to the nature of our ticketing system. We are required to archive data for many reasons, we archive data that we deem to be valuable for historical analysis, reporting, trend analysis. We also archive data that is required for regulatory compliance for audits, it is preserved while reducing the burden on the active database.

## Redundancy and High Availability

Redundancy and High Availability (HA) are implemented in MoviePlugs DB architecture to provide a reliable and seamless use of our ticketing system. These components include backup servers, and data replication.

### **Redundancy**

We practice Data Redundancy, by storing the same data in multiple locations or tables. This helps ensure data availability in case of failures. As well as, Server redundancy by using multiple database servers that replicate data, enabling seamless failover in case one server goes down.

### **High Availability (HA)**

In MoviePlug we use Failover Clustering which involves Implementing a cluster of database servers where one server automatically takes over if another fails. This ensures minimal downtime in case of a server failure. We also practice Load Balancing Distributing incoming database queries and transactions across multiple servers to ensure even workloads and improve response times. Load balancers can also route traffic away from failed servers.

## Compliance

MoviePlug Ticketing System’s DB contains customer information as such we are held to California Consumer Protection Act (CCPA) which is similar to the GDRP that is used in the EU.

### **California Consumer Protection Act (CCPA)**

That is that we are subject to safeguarding customer data from Personal Identifiable Information to private information such as billing/physical addresses to banking information. It is just as important that we comply with (CCPA) and alert our users in the event of a data breach as well as to disclose our data management process. This is to include the right to delete user information handled on an individual basis.

## Regular Maintenance

Maintenance is paramount to the successful operation of MoviePlug’s Ticketing System. As new technologies arise so do the APIs we use for our financial transactions that ensure an efficient an end to end safe transaction. Thus, regular maintenance and application update is continuous.

### **Maintenance Tasks**

Schedule regular database maintenance tasks, including index reorganization and performance tuning. Things in this category include the identification and handling of bottlenecks. Also, the tuning and improvements of queries that directly influence performance.

## Alternative Data Management

### **NoSQL Database Alternative**

Provides non-relational/distributed data. Typically employed in big data and real-time applications. Based on data with dynamic schema for unstructured data. The language convention provides queries that are focused on collection of documents and syntax varies from database to database. Examples of NoSQL include MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j and CouchDb.

### **SQL vs. NoSQL**

| **SQL** | **NoSQL** |
| --- | --- |
| SQL database not well-suited for hierarchical data storage. | NoSQL databases best suited for hierarchical data storage as it follows the key-value pair method for storing the data |
| From a commercial perspective, SQL databases are generally classified as open source or closed source. | They are classified on the basis of the way they store data as key-value store, document store, graph store, column store, and XML store. |
| SQL databases properly follow ACID properties (Atomicity, Consistency, Isolation & Durability) | NoSQL databases properly follow Brewers CAP theorem (Consistency, Availability, and Partition tolerance) |
| Adding new data in SQL database requires some changes to be made like backfilling data, altering schemas. | New data can be easily inserted in NoSQL databases as it does not require any prior steps. |
| **Excellent vendor support and community Support is available for all SQL databases.** | Only limited community support is available for NoSQL databases. |
| **Best fit for a high transaction-based application.** | You can use NoSQL for heavy transactional purposes. However, it is not the best fit for this. |
| Not suitable for hierarchical data storage. | Suitable for hierarchical data storage and storing large data sets (E.g. Big Data) |

### **Justification**

We decided to choose a SQL Database because of the nature and setup of our Ticketing System. Our data within the Ticketing System is structured, it is to include tables containing information like our movies, customer information among other things. We also have the need to conduct ACID transactions regularly, a very important characteristic that a SQL based Database provides. MoviePlug interacts with customer/users PII and financial information and SQL provides an edge over NoSQL, providing reliable mechanisms for access control and data protection and data compliance. MoviePlug Ticketing System involves financial transactions, thus we are subject to regulatory audits, SQL databases are well-suited for historical data and maintaining a complete audit trail of changes over time.