# Just3Guys

# Tickets'R'US

## Software Requirements Specification

## Version 1.0.0

Mar 28, 2024

Group 12
Matthew Tran, Samuel Walls, Victor Tepordei

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

## Revision History

| Date | Description | Author | Comments |
|---|---|---|---|
| 2/1/2024 | Initial project creation | Matther Tran | |
| 2/5/2024 | Introduction | All | |
| 2/9/2024 | Description and Requirements | All | |
| 2/15/2024 | Final draft | All | |
| 2/22/2024 | Software Design Specification | All | |
| 2/29/2024 | SDS | All | |
| 3/14/2024 | Test cases | All | |
| 3/28/2024 | Data Management Strategy | Samuel Walls | |
| 3/28/2024 | Software Architecture Diagram | Victor Tepordei | |

## Document Approval

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

| Signature | Printed Name | Title | Date |
|---|---|---|---|
| | Dr. Gus Hanna | Instructor, CS 250 | |
| | | | |

# Table of Contents

# 1. Introduction

This Software Requirements Specification (SRS) document is intended to provide a detailed overview of the functionalities, system constraints, user interactions, and technical requirements of Tickets'R'US. This SRS aims to ensure clear understanding and communication among stakeholders, developers, and users, facilitating a streamlined development process and a system that meets or exceeds expectations for user experience.

## 1.1 Purpose

The purpose of this SRS document is to provide a detailed description of the software product provided. This necessary document defines the hardware, software, and user interface requirements aimed towards the needs of the consumer while also outlining the development for the developers.

## 1.2 Scope

Tickets'R'US  is designed to be an interactive platform where users can browse, select, and purchase tickets for various movies. Tickets'R'US will have features such as:

- User registration and profile management
- Browsing performances by date, genre, and venue
- Viewing detailed information about performances, including synopsis, cast, and venue information
- Online ticket purchasing and payment processing
- Digital ticket delivery and order history tracking
- User reviews and ratings for performances

Tickets'R'US will cater to theater enthusiasts, casual attendees, and venue operators, offering an intuitive interface and a secure, efficient ticketing process. It will not encompass the management of performance content or the operational aspects of theaters and venues.

## 1.3 Definitions, Acronyms, and Abbreviations

*This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.*

| | |
|---|---|
| HTML | Hyper-Text Markup Language, a markup language which our website relies on. |
| API | Application Programming Interface |
| SRS | Software Requirement Specifications |

| | |
|---|---|
| NAS | Network Attached Storage |
| Salt (cryptography) | Random data added to encrypted information |
| Pepper (cryptography) | Semi-random data added to encryption, typically based on a hardware security module |
| CLI | Command line interface |
| MTBF | Mean time between failures |
| 2FA | Two factor authentication |

### 1.4 References

*This subsection should:*
*(1) Provide a complete list of all documents referenced elsewhere in the SRS, or in a separate, specified document.*
*(2) Identify each document by title, report number - if applicable - date, and publishing organization.*
*(3) Specify the sources from which the references can be obtained.*
*This information may be provided by reference to an appendix or to another document.*

### 1.5 Overview

The following information provided within the document contains product hardware an important database features. Section 2 defines general constraints when building Tickets'R'US and the overall description of the software. Section 3 summarizes functional requirements, use cases, database requirements. Section 3 also describes the external and communication interface. Section 4 covers diagrams helpful to the streamlined development of **Just3Guys.**

## 2. General Description

The General Description section of this SRS provides an overview of the general factors that influence the theater ticket website and its requirements. This section does not define specific requirements but rather outlines the broader context to help stakeholders understand the nuances of the product's development, integration, and operation.

### 2.1 Product Perspective

This subsection of the SRS puts the product into perspective with other related products or projects. The theater ticketing application operates as a standalone system, interacting with external entities such as users, payment gateways, and promotional partners. It interfaces with the theater's backend system for real-time updates on ticket availability and show schedules.

## 2.2 Product Functions

This subsection of the SRS provides a summary of the functions that the software will perform.

**User Services:**

- **User Account System:** The application will facilitate user registration, login, and account management.

**Product Services:**

- **Purchasing Tickets:** Users can browse available shows, select seats, and purchase tickets seamlessly.
- **Sales:** The system will track and manage ticket sales efficiently.
- **Advertisements:** Advertisements for upcoming shows and promotions will be displayed to users.
- **Promotions:** Special promotions and discounts can be applied during the ticket purchase process.
- **Sponsors:** Integration with sponsor information for promotional and partnership purposes.

## 2.3 User Characteristics

The users of the theater ticketing application include moviegoers, who may range from casual viewers to frequent patrons. User characteristics such as technological proficiency, preferences, and purchasing behaviors will influence the design and functionality of the application.

## 2.4 General Constraints

This subsection of the SRS provides a general description of any other items that will limit the developer's options for designing the system. Constraints may include budgetary limitations, time constraints, or hardware specifications.

## 2.5 Assumptions and Dependencies

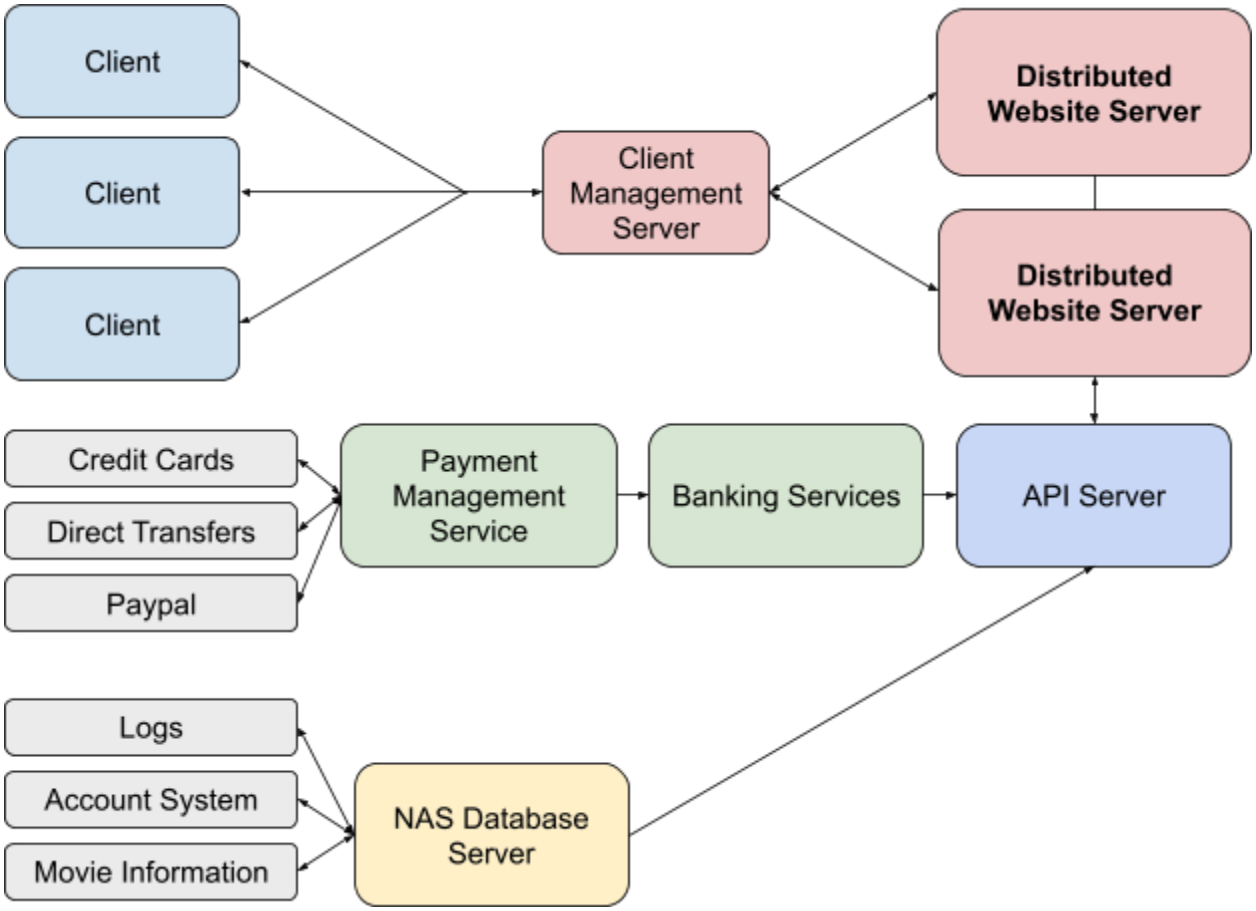
This subsection of the SRS lists each of the factors that affect the requirements stated in the SRS. Assumptions include the expectation that users will access the application using modern web browsers. Any deviation from these assumptions may impact the functionality and performance of the system. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption might be that a specific operating system will be available on the hardware designated for the software

product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

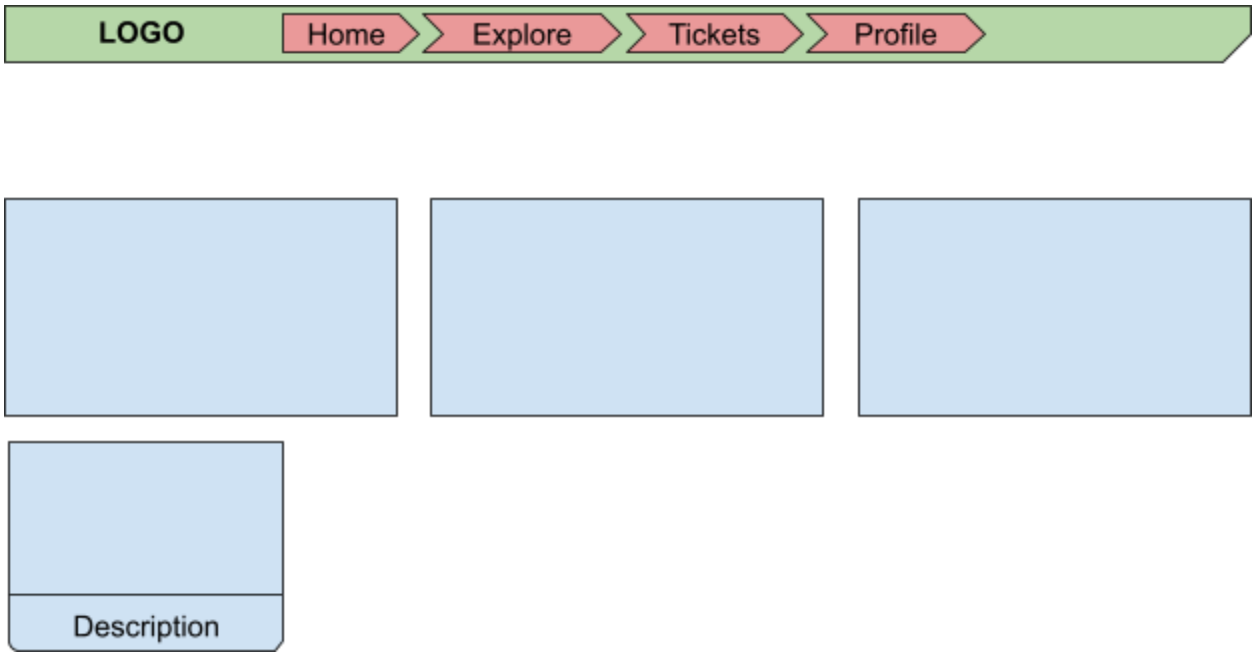
# 3. Specific Requirements

*This will be the largest and most important section of the SRS. The customer requirements will be embodied within Section 2, but this section will give the D-requirements that are used to guide the project's software design, implementation, and testing.*

## 3.1 External Interface Requirements



### 3.1.1 User Interfaces




**3.1.1.1** Web-access: Clients should access the ticketing system through the website.
**3.1.1.2** API: There should be API support for accessing certain services through external services. The API server manages account information, movie information, promotional sales, etc.
**3.1.1.3** CLI: Developer command line interface for monitoring the system logs.

### 3.1.2 Hardware Interfaces

**3.1.2.1** Hard disk: Program reads and writes from the file system when performing operations such as reading system variables and monitoring system loads. This information is logged to the NAS database server to help optimize server performance.
**3.1.2.2** On-site kiosk: Potential API integration for on-site devices. Provides direct access to purchasing tickets.
**3.1.2.3** Printing: Tickets may be printed with identifiable information, including the purchasing user. This prevents resale as tickets are associated with user accounts. In the case of a ticket purchased without an account, the user must provide information showing proof of purchase.

### 3.1.3 Software Interfaces

**3.1.3.1** Cryptogry: API server has an interface with a cryptographic module responsible for secure random, encryption, decryption, etc.

**3.1.3.2** Short-term Database: Short term database server for storing real-time information such as currently active users and current movie statuses. This allows for real-time high frequency data access to clients.

### 3.1.4 Communications Interfaces

**3.1.4.1** Client Management Server: Server which accepts client connections and routes them to the appropriate web server. This distributes high traffic and helps lessens slow load times.

**3.1.4.2** Banking Service: Banking service integration moves payment handling to 3rd parties. Integration interacts with API server to allow direct payments from clients, handling cases such as overdrawing and refunds.

**3.1.4.3** NAS Database Server: Database server holds important information such as user data, logs, movie information, times, and more. This data is accessed through the API server which communicates with the NAS server.

**3.1.4.4** User account service servers: This service manages account related information such as signing in, signing up, logging out, deleting accounts, etc.

**3.1.4.5** Database servers: Users need an account system which stores and queries user information in a short-term way. This data is periodically backed up to the NAS server.

## 3.2 Functional Requirements

### 3.2.0 Advertisements

**3.2.0.1** Discounts for students, elderly, military.
**3.2.0.2** Featured films
**3.2.0.3** Discounts based on day/time

### 3.2.1 Movie Searching

**3.2.1.1** Search bar
- **3.2.1.1.1** Search by title, director, description
- **3.2.1.1.2** Advanced search which includes character names, actors, directors

**3.2.1.2** Recommended films
- **3.2.1.2.1** Recommend similar movie to previously searched films
- **3.2.1.2.2** Recommend popular films in your area
- **3.2.1.2.3** Recommend films your friends/family have watched
- **3.2.1.2.4** Have recommended films for users not signed in

### 3.2.2 Account Creation

**3.2.2.1 Sign in page**
- **3.2.2.1.1** Username input: Accepts usernames with a max length of 16 characters.
- **3.2.2.1.2** Password input: Accepts passwords with a max length of 32 characters.

**3.2.2.1.3** Remember me: Checkbox which stores a public encryption key generated by the server as a cookie on the user's device. When the server sees this cookie it knows the client is a certain user.

**3.2.3.1** Movie Transaction
**3.2.3.1.1** Discounts/Coupons: The system shall support discounts and coupons for movie purchases, allowing users to avail of promotional offers during the transaction.

**3.2.3.1.2** Multiple Tickets (Family Plan): The system allows the purchase of multiple tickets under a single transaction, supporting family plans or group bookings.

**3.2.3.1.3** Payment Options: The system shall provide various payment options for users to complete movie transactions securely.

**3.2.4** Support
**3.2.4.1** User Feedback and Notifications
**3.2.4.1.1** Email Confirmation: The system shall send email confirmations to users after successful movie transactions or account-related activities.

**3.2.4.1.2** Phone Reminder: The system shall provide phone reminders or notifications to users for upcoming movie showtimes, ensuring a timely and convenient movie-going experience.

**3.2.5** Home Page
**3.2.5.1** User Interface Elements
**3.2.5.1.1** Title: The home page displays the ticketing logo.

**3.2.5.1.2** Search: The home page has a large centered search bar allowing users to quickly search for movies, theaters, or specific showtimes.
**3.2.5.1.2.1** Auto-fill based on recommended movies

**3.2.5.1.3** Popular Movies: The home page showcases a section highlighting popular and trending movies, building user attention while advertising new movies.

**3.2.5.1.4** Sign Up/Log In: The home page should include prominent options for users to sign up for a new account or log in to their existing accounts.

**3.2.5.1.5** Featured Movies: The home page should present a curated selection of featured movies, promoting specific films or events.

**3.2.5.1.6** Explore More: The home page should offer a section for users to explore additional movie genres, theaters, and special categories.

**3.2.5.1.7** Airing Near You Tickets: The home page should feature a section showcasing movie tickets for shows airing near the user's location, enhancing the personalized user experience.

**3.2.5.1.8** Support: The home page shall include a support section with links to user guides, FAQs, and customer service contact information.

### 3.2.7 Progressive Web App (PWA)

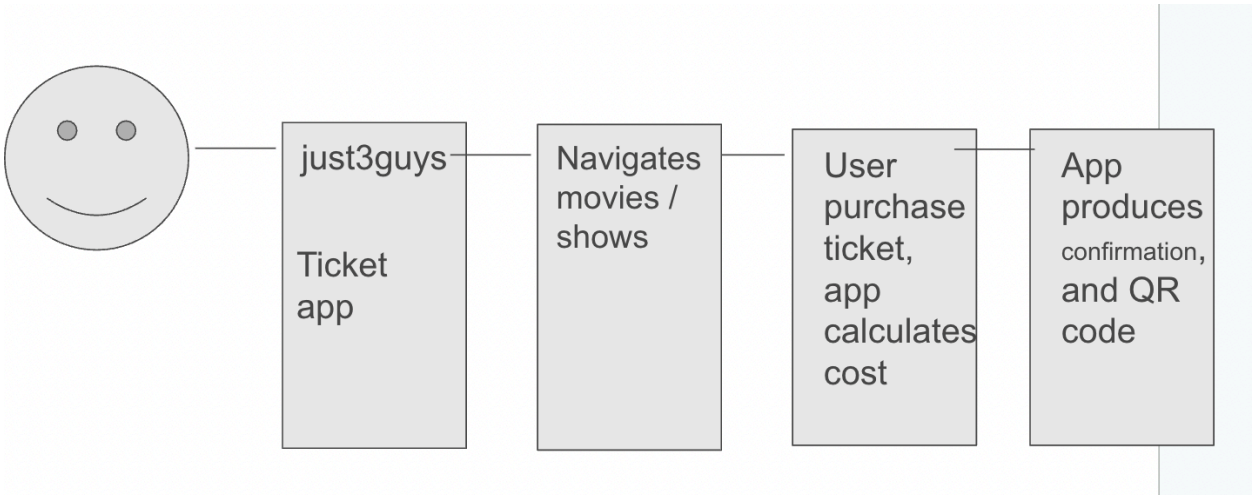
**3.2.7.1** Offline access which informs the user that the internet is required.

**3.2.7.2** Service worker optimization which allows for cached images and other information. This can significantly improve performance and lighten server load.

## 3.3 Use Cases

### 3.3.1 Use Case #1: User Purchases ticket

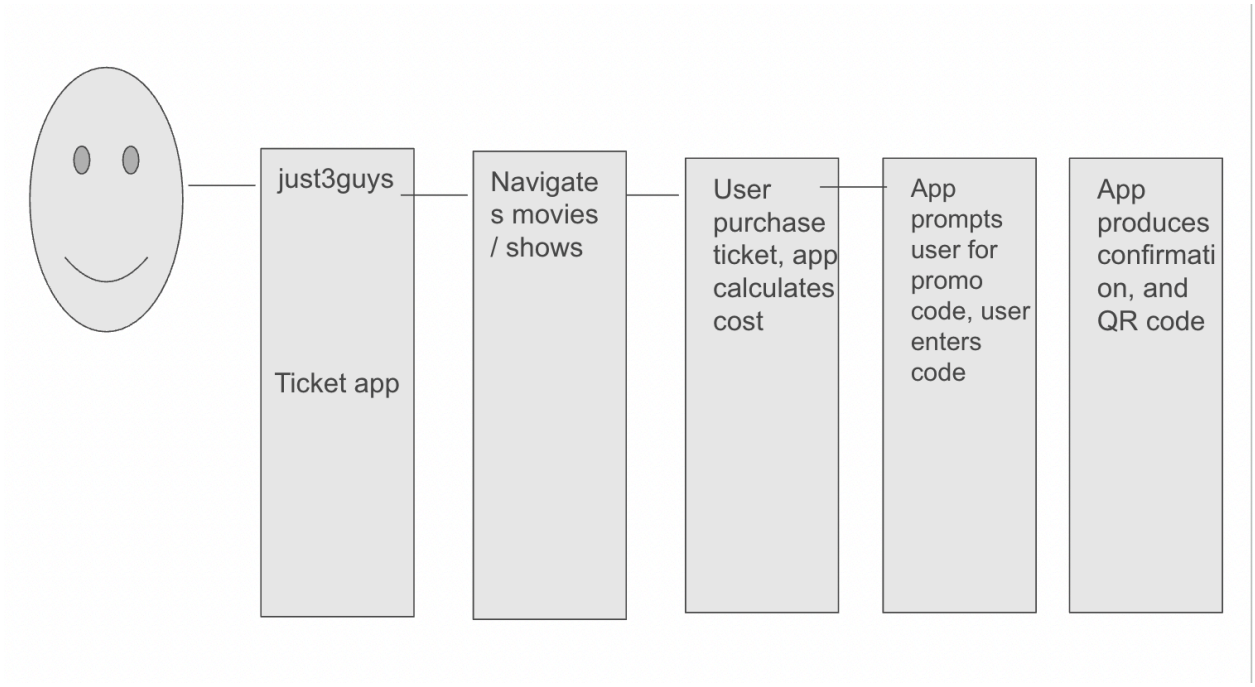
The user accesses the theater ticketing application through a modern web browser and logs into their account. Upon logging in, the user navigates to the list of available movies and showtimes. The user selects a preferred movie, showtime, and seating preferences. The application validates the selected seats' availability and calculates the total cost. The user proceeds to the payment section, providing necessary payment details. The system securely processes the payment transaction through integrated payment gateways. Upon successful payment, the user receives a confirmation of the ticket purchase, which includes details like the booking reference and QR code. The application updates the ticket availability in real-time, ensuring accurate information for subsequent users.




### 3.3.2 Use Case #2: User Applies Promotional Code

The user, during the ticket purchase process, reaches the section for applying promotion codes. The user enters a valid promotion code provided by the theater or through a promotional campaign. The application validates the promotion code and calculates the discounted ticket price based on the applied promotion. The user reviews the updated pricing and continues to the

payment section. The system processes the payment transaction considering the applied promotion code. Upon successful payment, the user receives a confirmation of the ticket purchase with the applied promotion details.




**3.3.3 Use Case #3:** A user searches for movies to watch with their family.

The user opens the website and searches for movies tagged with “family friendly”. The user scrolls through the movies until one catches their eye. The user decides they want to purchase tickets for their entire family, but is hesitant due to the cost of buying tickets for multiple family members. However, the user sees that students get 50% off and movies are much cheaper on weekdays at night. The user adds his family members to the ticketing service and finds a cheap time and day for the tickets. The user proceeds to purchase the tickets and prints them off at home. He then receives a confirmation email with his tickets in addition with his applied discounts and price.

## 3.4 Classes / Objects

### 3.4.1 User session

**3.4.1.1** Attributes

- **3.4.1.1.1** Search history
- **3.4.1.1.2** Associated account, null if not signed in

**3.4.1.2** Functions

- **3.4.1.2.1** Create new session: Creates new recommendation based on history.
- **3.4.1.2.2** Delete old session: Removes session from memory.

**3.4.1.2.3** Get associated account: Checks if cookies match any saved accounts.

**3.4.1.2.4** Save search history: Updates the user's search history with the latest queries and interactions.

**3.4.1.2.5** Clear search history: Clears search history if the user wants their information to be cleared.

**3.4.2 User Account**

**3.4.2.1** Attributes

**3.4.2.1.1 User ID**: A unique identifier for each user account, allowing for easy identification and retrieval of user-specific information.

**3.4.2.1.2 Username**: The chosen username of the account holder, used for login and public display within the system.

**3.4.2.1.3 Email Address**: The unique email address associated with the user account, used for communication and account recovery.

**3.4.2.1.4 Password**: The secure, hashed representation of the user's password, ensuring confidentiality.

**3.4.2.1.5 Payment Information**: The stored payment details of the user, used for quick and secure transactions.

**3.4.2.1.6 Preferences**: User-specific preferences, such as preferred genres, notification settings, or language preferences.

**3.4.2.1.7 Cookie token**: Unique random token which is associated with the user account. This cookie is deleted after 30 days.

**3.4.2.2 Functions**

**3.4.2.2.1 Create New Account**: Initiates the creation of a new user account, requiring essential information such as username, email, and password.

**3.4.2.2.2 Update Account Information**: Allows the user to update their account details, including username, email, password, and preferences.

**3.4.2.2.3 Delete Account**: Permanently removes the user account from the system, including associated data such as payment information and preferences.

**3.4.2.2.4 Change Password:** Enables the user to change their account password.

**3.4.2.2.5 Retrieve Payment Information**: Retrieves the stored payment information associated with the user account for secure transactions.

## 3.5 Non-Functional Requirements

*Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transaction shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc).*

### 3.5.1 Performance

The application should support 10,000,000 concurrent users during peak hours without significant degradation in performance.

### 3.5.2 Reliability

**3.5.2.1** Error Handling

The application will notify the users of errors and will send a detailed log to the server, ensuring that both the client and server have a clear understanding of the issue. Error logs should be generated and sent to system administrators to troubleshoot issues.

**3.5.2.2** System Stability

The software should have a Mean Time Between Failures (MTBF) of at least 30 days, ensuring reliable and stable operation.

### 3.5.3 Availability

The system shall have a minimum uptime of 99.9%, excluding scheduled maintenance. In addition, the website should have translations for multiple languages including but not limited to English, Spanish, and Swedish.

### 3.5.4 Security

**3.5.4.1** Authentication

User authentication must be secure, utilizing up-to-date industry-standard encryption protocols such as salted Argon2id. Passwords should be stored securely using salt and peppered hashes. Identification cookies (for users who selected “remember me”) should be randomly generated and should have no identifiable information about them.

**3.5.4.2** Authorization

Role-based access control (RBAC) will be used to verify users have access to the appropriate information. This authorization will be securely implemented using local networks directly connected to the server.

**3.5.4.3** Data Encryption

Sensitive user information, including payment details, shall be encrypted during transmission using Transport Layer Security (TLS) to prevent unauthorized access.

**3.5.4.1** Backup
A complete recovery plan should be available hourly, including data backup and restoration processes. These backups should be saved to the NAS server. These backups should be archived and sent to long-term storage monthly.

### 3.5.5 Maintainability

**3.5.5.1** Code Documentation
All code will be commented and documented in order to benefit maintenance and future enhancements.

**3.5.5.2** Modularity
All systems should be modular, which not only allows the addition of new features but allows for updates without disrupting users. Utilizing modular DLL server files can lead to much high uptime as changes can be made without restarting the server.

### 3.5.6 Portability

**3.5.6.1** Web-access

**3.5.6.2** Browser: Ticketing system is browser based which maximizes portability.

**3.5.6.3** Mobile: Website has mobile support and correctly scales to smaller devices.

**3.5.6.2** Cross-Browser Compatibility
The application must be compatible with major web browsers such as Chrome, Firefox, Safari, and Edge. Older browsers may or may not be compatible, however, in these cases a warning notification will be shown.

**3.5.6.3** Javascript: Javascript must be enabled in order for the website to function, an error message will be displayed if Javascript is not enabled.

**3.5.6.4** CSS: CSS must be enabled, if not, an error message will be displayed.

## 3.6 Inverse Requirements

**3.6.1** Denial of service: In cases of extremely high load and suspected denial of service, all accounts not previously seen will be denied.

**3.6.2** Reliability: In cases of slow-response times, administrators should be notified with relevant system information.

## 3.7 Design Constraints

The system used will conform to IBM's CUA standards and will be built using standard web development tools.

Computers accessing the web based product should be equipped with web browsers such as Google.
Users are expected to have standard computer knowledge to access the product.
There is no significant load on memory for computers.

## 3.8 Logical Database Requirements

*Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.*

## 3.9 Other Requirements

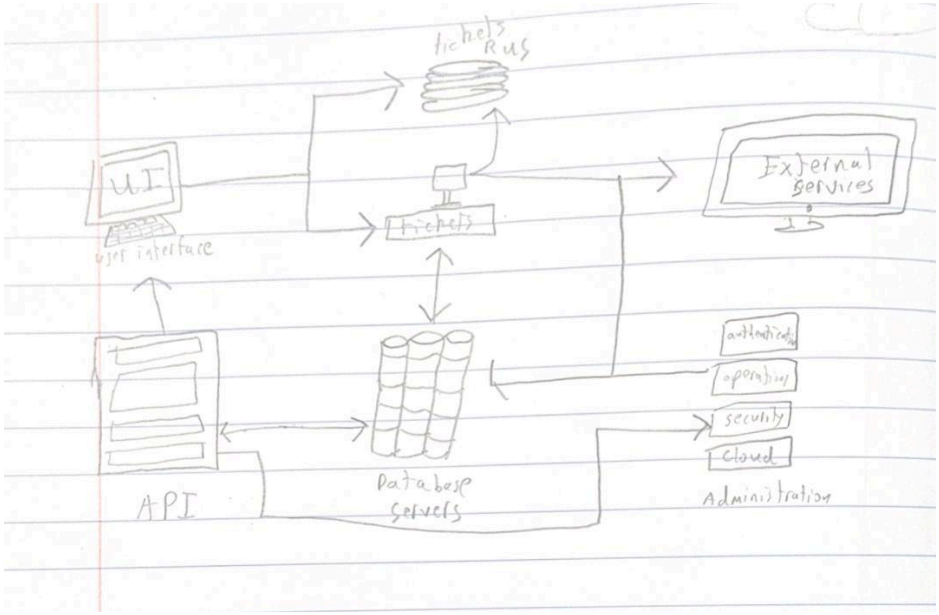
*Catchall section for any additional requirements.*

# 4. Analysis Models

*List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable to the SRS's requirements.*

## 4.1 Sequence Diagrams

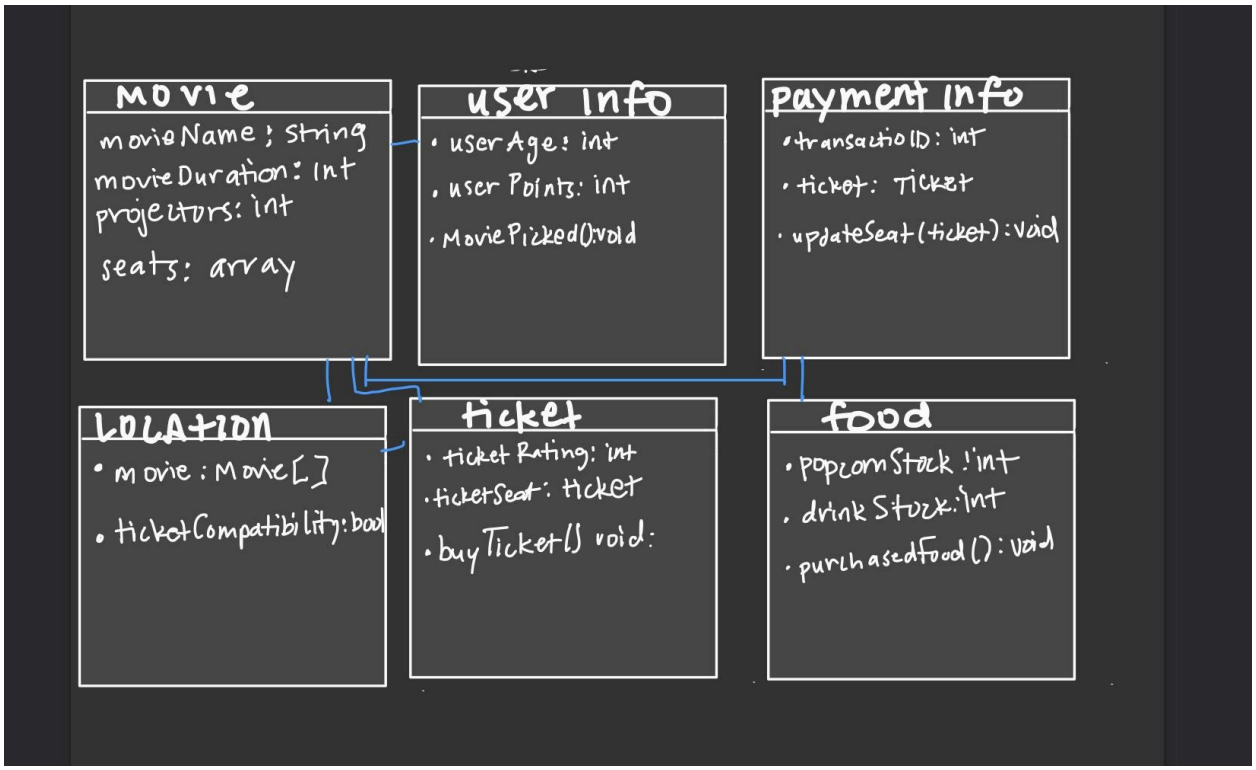
### 4.1.1 SWA Diagram




- **User Interface:** Represented by icons of a desktop indicating the software is accessible via web and mobile platforms.

- **API Layer:** Shown at the base of the diagram, symbolizing the foundation that handles communication between the user interface and other system components.
- **Tickets Module:** Positioned centrally above the API layer, representing the core functionality of the system where ticket-related operations are processed.
- **Database Servers:** Illustrated with a stack of cylinders, suggesting data storage capabilities, possibly for user data, ticketing information, and performance details.
- **External Services:** Depicted with cloud icons, which indicate integration with third-party services such as payment gateways, email services, or other APIs.
- **Administration:** Represented by a stack, implying tools or dashboards for system administrators to manage and monitor the software's operations.

### 4.1.2 UML Diagram




- Description of classes
  - Movie:
    - The Movie class entails the current film being projected, the duration of the movie, and the projectors displaying the movie. It also has an array of seats available and seats taken.
  - UserInfo:
    - The user info monitors the user's age which determines the rating of the movie they can view, and also keeps track of their account's points and the movie they picked as well.
  - PaymentInfo:

        - PaymentInfo covers the transaction ID of the ticket purchased, keeping a database of all tickets procured.The paymentinfo class also updates the seat chosen, altering the seating array in the Movie class.
    - Location:
        - The location is a simple class, containing the movie and the compatibility of the user's ticket and the location.
    - Ticket:
        - Ticket describes the rating of the ticket, such as Rated R or Rated E, and also keeps track of the seat and includes a buyTicket method which is the process of buying a ticket.
    - Food:
        - Food simply keeps track of the theater's food stock, which when it is low, means that popcorn, hot dogs, and drinks need to be restocked for additional customers who want to purchase food.
- Description of attributes
    - movieName is a string as it just includes the name of the movie, and movie duration + projectors includes the length of the movie and the amount of working projectors in the theater. Seats are ordered in arrays containing individual seats and their availability.
    - userAge and userPoints are both integer values as they only require numbers. MoviePicked is a method without a return value.
    - transactionID returns an integer value while ticket returns a ticketID called "ticket" which is used in updateSeat() function which updates the seating in the theater.
    - Movie is an array which has a list of movies showing for the week associated with the corresponding location for the showing of each movie. Ticket Compatibility is a boolean value which returns true if the ticket is valid in the location and false if it is not.
    - ticketRating returns an integer value of the rating type, e.g. rating R is a value of 1, etc. ticketSeat returns the "ticket" value as well which is used to determine the location of the seat. buyTicket() is a function used when the user is in the process of purchasing a ticket.
    - The integer values in food are keeping track of the current food in the stock, and purchasedFood() is an example of a receipt which confirms the food has been purchased.
- Description of operations

### 4.3 Data Flow Diagrams (DFD)

### 4.2 State-Transition Diagrams (STD)

## 5. Test Verification Plan

### 5.1 Unit Testing

We will perform unit testing which will test individual pieces of code. We will test smaller pieces of code such as the apply promo code function (as shown in test promoCode_1). This will verify that the software is performing correctly in regard to payments. We will also test ticket handling through tests such as updateTicket_1. This will test code related to ticketing information to ensure information is correctly processed and handled. In addition, pieces of code responsible for searching for movies (as shown in test searchMovie_1) will be tested to ensure accurate retrieval of movie information.

#### 5.1.1 Test Case: promoCode_1

**Objective**: To verify that the apply promo code function works correctly, ensuring accurate application of discounts during the payment process.

**Test Description**:

1. Enter a valid discount code at checkout.
2. Complete the purchase.

#### 5.1.2 Test Case: updateTicket_1

**Objective**: To ensure that the update ticket function accurately handles modifications to ticket information.

**Test Description**:

1. Picks ticket
2. User goes to ticket checkout
3. Ticket is purchased"

#### 5.1.3 Test Case: searchMovie_1

**Objective**: To verify that the search movie function retrieves accurate and relevant movie information.

**Test Description**:

1. Log in to the application.
2. Access the browse shows section.
3. Select different filters (date, genre, venue).

## 5.2 Integration Testing

We will perform integration testing which will incorporate the separate modules and their interactions. This verifies the integration of individual components into larger subsystems.

### 5.2.1 Test Case: purchaseTicket_1

**Objective**: To verify the integration of various modules involved in the ticket purchasing process.

**Test Description**:

1. User prompts server for movies
2. Server checks availability of seats for selected movie
3. Server returns "full" if all seats are taken"

**Test Coverage**: This test ensures the seamless integration of modules responsible for user authentication, movie selection, seat allocation, promo code application, payment processing, and ticket creation. It validates the entire ticket purchasing workflow, from user interaction to database updates, ensuring that all components work together cohesively.

### 5.2.2 Test Case: theaterLocation_1

**Objective**: To validate the integration between location-based services and movie listings.

**Test Description**:

1. User searches movies
2. Server prompts user's location
3. Provides list of theaters with requested movie nearby

### 5.2.3 Test Case: purchaseSnacks_1

**Objective**: To test the integration between ticket purchasing and snack ordering functionalities.

**Test Description**:

1. User nears completion of ticket purchase
2. UI prompts food selection
3. User clicks food selection and purchases in tandem with ticket

**Test Coverage**: This test evaluates the integration between the ticket purchasing module and the snack ordering module. It ensures that users can seamlessly add snacks to their ticket orders and that both ticket and snack purchase information are properly synchronized and stored in the system.

## 5.3 System Testing

We will perform system testing to ensure that the final product is working as intended. System testing involves testing the entire software system as a whole to ensure that it meets the specified requirements and functions as intended in a real-world environment. This includes testing system functionality, load and stress testing the product to ensure reliability.

#### 5.3.2 Test Case: sqlInjection_1
**Objective**: To test the system's resilience against SQL injection attacks.

**Test Description**:
1. Attempt to inject SQL code via input fields (search, login)
2. Verify that the system detects and prevents SQL injection attempts.
3. Confirm that the system responds with appropriate error messages

**Test Coverage**: This test evaluates the system's security measures against SQL injection attacks. It ensures that the system can identify and block malicious SQL code injections, protecting the security of the underlying database.

#### 5.3.3 Test Case: performanceUnderload_1
**Objective**: To assess the system's performance under normal operating conditions.

**Test Description**:
1. Simulate multiple users accessing the application simultaneously.
2. Perform various actions (browse shows, purchase tickets).

# 6. Data Management Strategy

We will be using SQL as it provides a more structured format for storing and retrieving data. This allows for better data management and security as programs will be using more reliable libraries and APIs for interacting with the databases. SQL provides support for complex queries, transactions, and relational data integrity, which is essential for bringing together the different forms of data in the ticketing system. A single, well-structured SQL database will be utilized for simplicity and to ensure transactional integrity across different parts of the application. This database will use multiple tables which will each manage the different sections of the program. The scope of the Tickets'R'US system and its data interrelations are effectively managed within a single database, reducing complexity and potential connectivity issues between separate databases.

## 6.1 Security

**Encryption**: Use of industry-standard encryption methods for sensitive data such as passwords and payment information. We will be using Argon2 to encrypt and decrypt information such as account details, passwords, and tokens. Argon2 is designed to be memory-hard, which makes it resistant to various types of attacks, including GPU-based attacks. It is considered the current state-of-the-art for password hashing.
**Backups**: We will employ regular backups to a separate storage solution such as a NAS storage device, ensuring data integrity and availability.

## 6.2 Data Storage and Retrieval

**Data Entities**: We will be storing user accounts, movie details, ticket transactions, seat reservations, payment information, and promotional codes.
**Data Relationships**: We will utilize foreign keys and indexed tables to efficiently manage relationships between movies, showtimes, users, and tickets. This allows for quick and efficient data retrieval to ensure low server costs.

## 6.3 Possible Alternatives

**NoSQL Database**: A NoSQL database is very useful for its scalability and flexibility with unstructured data. However, because of the relational nature of ticketing data, SQL was chosen for its relational data integrity and because it is better suited for transactional support. In addition SQL is an industry standard and has better support and documentation.
**Multiple Databases**: Distributing data across multiple databases was considered for scalability but was not used to maintain transactional integrity and simplify system architecture.

## 6.4 Trade-offs

**SQL vs. NoSQL:**

**SQL**: Offers good support for complex queries and transactions, which is essential for the ticketing system, at the cost of less flexibility in handling unstructured data.
**NoSQL**: Provides scalability and flexibility with unstructured data but lacks the transactional and relational integrity required for the ticketing system.

**Single Database:**

**Pros**: Simplifies data management, maintains transactional integrity.
**Cons**: Potentially limits scalability compared to distributed database systems.

**Cloud Computing:**

**Pros:** Abstracts code and allows for better data management and scalability. Utilizing cloud services such as AWS and Cloudflare can prevent attacks such as DDOS attacks in addition to supporting high data loads.
**Cons:** Requires programmers with experience utilizing cloud computing services. Adds additional fees to maintain uptime.

# 7. Change Management Process

*Identify and describe the process that will be used to update the SRS, as needed, when project scope or requirements change. Who can submit changes and by what means, and how will these changes be approved.*

# A. Appendices

*Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements.*

*Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.*

## A.1 Timeline

### A.1.1 Matthew Tran

In charge of developing and implementing section 1 and section 4.1.1.
Section 1 should be completed by February 15.
Section 4.1.1 should be completed by February 29.

### A.1.2 Victor Tepordei

Completed section 2 and section 4.1.2.
Section 1 should be completed by February 15.
Section 4.1.1 should be completed by February 29.

### A.1.3 Samuel Walls

In charge of section 3 and section A.1.1.
Section 1 should be completed by February 15.
Section 4.1.1 should be completed by February 29.