Software Requirements Specification Template

San Diego Theater Ticketing App

Software Requirements Specification

Version 1

Feb 20, 2025

Group 11

Carlyne Ada, Nick Stoneking, Theo Leonard

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Spring 2025

Revision History

Date	Description	Author	Comments
02/20/25	Version 1	Group 11	Version 1 Implementation
03/06/25	Assignment 2	Group 11	"Software Design Spec"
03/18/25	Assignment 3	Group 11	Test Plan

Document Approval

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

Signature	Printed Name	Title	Date
	Group 11	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

This Software Requirements Specification (SRS) document outlines the functional and nonfunctional requirements of a new Theater Ticketing System designed to support both online and in-person ticket sales for multiple theaters in San Diego. The system is intended to allow customers to browse, select, and purchase tickets for different movies, while providing features like movie-specific data, ticket prices, available seatings, and more. This document is structured to provide the necessary information for the software team to design and implement the system in a way that meets all the functional and nonfunctional requirements.

1.1 Purpose

The purpose of this document is to outline the software requirements for the San Diego Theater Ticketing Application. This application is intended to allow customers in San Diego to browse, purchase, and manage movie tickets for local theaters via a web interface. Customers will be able to partake in further features such as memberships and reviews as they explore the webpage. This document will provide the detailed specifications necessary for the design, development, and testing of the product. The system also supports administrative functionalities, such as managing showtimes, handling customer feedback, and ensuring an optimal ticketing experience. This document provides specific requirements for the design, development, deployment, and testing of the system to make sure it is reliable, can scale, and overall provide user satisfaction.

1.2 Scope

(1) Software Product(s): Theater ticketing system for San Diego-based movie theaters using web pages.
(2) The software will allow customers to browse, purchase, and manage tickets for movies across multiple theaters in San Diego.
(3) Application of the Software:
    (a) - Primary goal: To provide an intuitive, real-time ticketing system that allows customers to purchase tickets, view available seats, and apply discounts if applicable.
    - Customer Experience: Simplified, browser-based platform to ensure smooth purchasing experiences with a ten-minute timer for any abandoned tickets.
    - Administrative Goal: Efficient management of showtimes, refunds, and user feedback.
    - Scalability: The system is designed to scale for further growth, handling up to 10 million users and supporting multiple theaters in the San Diego area.
    (b) This system aligns with the business goal of improving the efficiency of theater ticket sales while maintaining scalability for system expansions.

1.3 Definitions, Acronyms, and Abbreviations

User: A customer who interacts with the system to purchase tickets or browse movies.

Bots: Automated programs that attempt to purchase tickets for resale at inflated prices.
Scalping: The resale of tickets at prices higher than their original value.
API: Application Programming Interface for interacting with external data sources.
Web Scraper: A tool to collect and display movie reviews from external websites.
Stack: a set of technologies that work with each other to perform a specified functionality
Redis/Caching: Data storage that uses maps and is stored in memory instead of disk
Cron Job: A recurring time-based trigger for a service or process

1.4 References

IEEE Guide to Software Requirements Specifications.
Web API documentation from theater companies.
Documentation for any third-party web scraping tools used.

1.5 Overview

The rest of this document will contain the following in order:

- General Description: For exposure to general information regarding product requirements without using specific requirements themselves.
- Specified Requirements: For containing clear and specific product requirements for each relevant aspect of the product.
- Analysis Models: For containing a list of analysis models used with details containing the use case for the listed models.
- Change Management: For clear, documented, and procedural ways to handle change in product design direction.

2. General Description

This section will overview the flow of the web app system. The product will be discussed with enough detail for a basic amount of functionality, with descriptions on main components and features. Finally, constraints, assumptions and dependencies will be discussed in the following.

2.1 Product Perspective

The San Diego Theater Ticketing Application is a comprehensive, web-based system designed to streamline the process of ticket purchasing for moviegoers across San Diego. The product will consist of two main components: a frontend and a backend.

The frontend will provide an intuitive user interface where customers can browse available movies, check showtimes, and purchase tickets for local theaters. Users will be able to search for movies based on genres, showtimes, or proximity to their location, with the option to filter results to find the best available tickets. The system will be available in three languages which are Swedish, Spanish, and English and currency will be automatically converted based on the geographical location of the user. Customers can review their purchased tickets, view discounts or promotions, and provide feedback on their experience.

On the backend, the application will interface with databases that contain information about theaters, movies, showtimes, ticket availability, and pricing. It will also handle user data, including profiles, preferences, and ticket purchase history. A system for managing user feedback and reviews will verify whether comments are associated with verified ticket purchases. Backlend will also support an administrator mode for theater staff to manage showtimes, handle customer feedback, and view transactions.

The system will be developed to handle at least 10 million concurrent users, allowing for management during peak loads such as high-demand movie releases. Anti-bot measures will be in place to protect against ticket scalping. With a focus on streamlining the entire process, the application enables users to purchase tickets within two weeks of a showtime, with a grace period of 10 minutes after the movie begins for late purchases. This product will provide a centralized platform for customers to engage with San Diego theaters, providing convenience and improving the ticket purchasing process.

2.2 Product Functions

The product functions will include: being built to handle at least 10 million users at one time; blocking bots from scalping tickets of high demand movies; contain an interface to display the showtimes and ticket information from a database; a user purchase limit of 20 tickets at a time; ticket purchasing restraints only being available between 2 weeks before show time and 10 minutes after the movie starts; contain an administrator mode; a customer feedback/review system; have discount support at checkout for tickets; and contain a web scraper to use critique movie reviews from other websites.

2.3 User Characteristics

User characteristics will include: being a San Diego resident, having an average to below average amount of technical experience, access to the internet, and a device that can open webpages.

The typical user of the system will be:
Demographics: A resident of San Diego, typically aged 18-45 familiar with basic internet usage, and possibly unfamiliar with advanced technical features. The system will cater to users with average to below-average technical skills, which is why it should be intuitive and user-friendly.
Technical Experience: Users are expected to be familiar with browsing the internet and interacting with web applications but may not have extensive experience with more technical aspects such as troubleshooting.

2.4 General Constraints

Developing for computer web use, no need for focus on mobile formatting.

Platform Requirements: The product will be designed for computer web browsers and is not required to be optimized for mobile devices.
Performance Requirements: The system should be able to support at least 1,000 simultaneous users. Response times should not exceed 2 seconds for any page requests, and the application should be capable of managing multiple transactions that occur at once without degrading the performance.

2.5 Assumptions and Dependencies

Assumptions and dependencies for the product that are being made are the following: an available api from theater companies that allow requests for movie information, ticket information and pricing, and critique comments; etc.

3. Specific Requirements

This section will contain the necessary specific requirements the web application will need at a baseline. The following will contain in-depth descriptions on each specific requirement outlined.

3.1 External Interface Requirements

3.1.1 User Interfaces

First time users need to see a log in/sign up screen when first interacting with the web application.
There should always be a navigation bar that contains access to the user's profile, a search engine to find movies, local theaters in proximity to the user's location, and access to all main pages from every page.
After signing up, log-in, or the user is a returning user, the user should be able be shown a home page. This home page will display the following: the top 5 rated, playing movies available in theaters in San Diego; recommended movies based on what movies the user has seen, bought tickets for, and genres they follow.
Each theater page will contain current showing movies, location reviews, movies close to selling out and future releases coming soon.
Each movie page will contain critique and audience reviews, when applicable; show times and prices, and the option to add a review if the user has purchased a ticket from our web application and the screening has ended for said movie.

3.1.2 Hardware Interfaces

No necessary hardware interface. All physical aspects of this web app will be handled by the user's own computer. This includes the gps component, to get the user's current location, will be handled locally by the computer and users web browser of choice as the user is prompted to enable showing their location.

3.1.3 Software Interfaces

The system will interface with multiple third-party systems and APIs to collect theater and movie data.

- Application will be utilizing a third-party API to fetch real-time movie showtimes, ticket, availability, and pricing information.
- The system will be using a secure payment gateway for processing tickets efficiently and safely.
- The web scraper will be implemented to retrieve movie reviews from other trusted sources (e.g. Rotten Tomatoes, IMDb, et al.).

3.1.4 Communications Interfaces

The communication between all parts of the web app should be reasonably quick. Although the entire database of movies and theaters may be needed, each page only needs to request a limited amount of data for full functionality; the same is to be said with user reviews/comments.

Page Requesting: The user interface will send HTTP requests to the backend, which will handle data retrieval from the database, process ticket purchases, and return appropriate responses to the user. The system should respond within a reasonable time of roughly 2 seconds to provide a smooth user experience.

3.2 Functional Requirements

3.2.1 Dashboard

3.2.1.1 Introduction
A dashboard with a navigation bar that allows the user to view tickets
3.2.1.2 Inputs
Navigation inputs
3.2.1.3 Processing
Fetching the tickets that are available in the database
3.2.1.4 Outputs
Visual output for the user to see what tickets are available
3.2.1.5 Error Handling
Shows a 404 not found for pages that don't exist, other errors should be handled by other processes.

3.2.2 Ticket Purchase Limit

3.2.1.1 Introduction
To prevent bulk ticket purchases, users will be limited to purchasing a maximum of 20 tickets per transaction and a window of 2 weeks to 10 minutes before showtime..
3.2.2.2 Inputs

Users will input the number of tickets they wish to purchase.
3.2.2.3 Processing
The systems will check if the number of tickets exceeds the limit and/or the time window and prompts the user to adjust the order if needed.
3.2.2.4 Outputs
If the number of tickets is over the limit, the system will notify the user and block further purchases until the limit is respected.
3.2.2.5 Error Handling
The user will see a message such as “You can only purchase up to 20 tickets per transaction.”

3.2.3 Authentication

3.2.3.1 Introduction
Page/authentication flow for users to log in to the product.
3.2.3.2 Inputs
User/pswd
3.2.3.3 Processing
Create unique JWT/other auth token for secure browsing, as well as the ability to protect user information that they might want to save for future transactions
3.2.3.4 Outputs
Auth token and navigation to dashboard
3.2.3.5 Error Handling
Handling for a failed authentication request, notification that auth failed

3.2.4 Ticket Purchases

3.2.4.1 Introduction
Ability for the user to navigate and purchase tickets.
3.2.4.2 Inputs
Order information chosen by the user through the front end
3.2.4.3 Processing
Order info is sent to the system for validation, fund management, and ticket transfer.
3.2.4.4 Outputs
Ticket emailed to user.
3.2.4.5 Error Handling
Users will be informed by pop-up or notification.

3.2.5 ReCaptcha

3.2.5.1 Introduction
A captcha system to prevent botting
3.2.4.2 Inputs
Captcha input from user
3.2.4.3 Processing
Captcha API or component

3.2.4.4 Outputs
Transfer to ticket purchase page
3.2.4.5 Error Handling
Handled by Captcha

3.2.6 Admin Page

3.2.6.1 Introduction
An administrator-view dashboard that allows you to view all current tickets, unrestricted, as well as site feedback, and a special indicator by your name when commenting on any feedback forms
3.2.6.2 Inputs
auth creds and navigation to dashboard
3.2.6.3 Processing
displays all data properly with no filters/restrictions
3.2.6.4 Outputs
Ability to view everything on the site
3.2.6.5 Error Handling
regular auth error handling

3.2.7 Customer Feedback System

3.2.7.1 Introduction
Customer feedback system that allows for people to comment on a built in form to leave reviews of movies. This should validate whether or not the user has purchased a ticket through the site, to verify they have watched the movie.
3.2.7.2 Inputs
Navigation to page, scrolling to render more messages, user id (to determine if it is read only), possibly user message.
3.2.7.3 Processing
loading the first x amount of messages for the user to see
3.2.7.4 Outputs
Displays reviews and comments user's have left regarding the movie that was commented on
3.2.7.5 Error Handling
Notification/placeholder when data cannot be retrieved/rendered

3.2.8 Dynamic Scraper

3.2.8.1 Introduction
A dynamic web scraper that can extract useful information from websites like imdb, and other critic websites. Could be run as a cron job
3.2.8.2 Inputs
List of movies
3.2.8.3 Processing

goes through the list of movies, and stores scraped data (maybe LLM processed?) and orders it into a database along with the movie information stored

3.2.8.4 Outputs

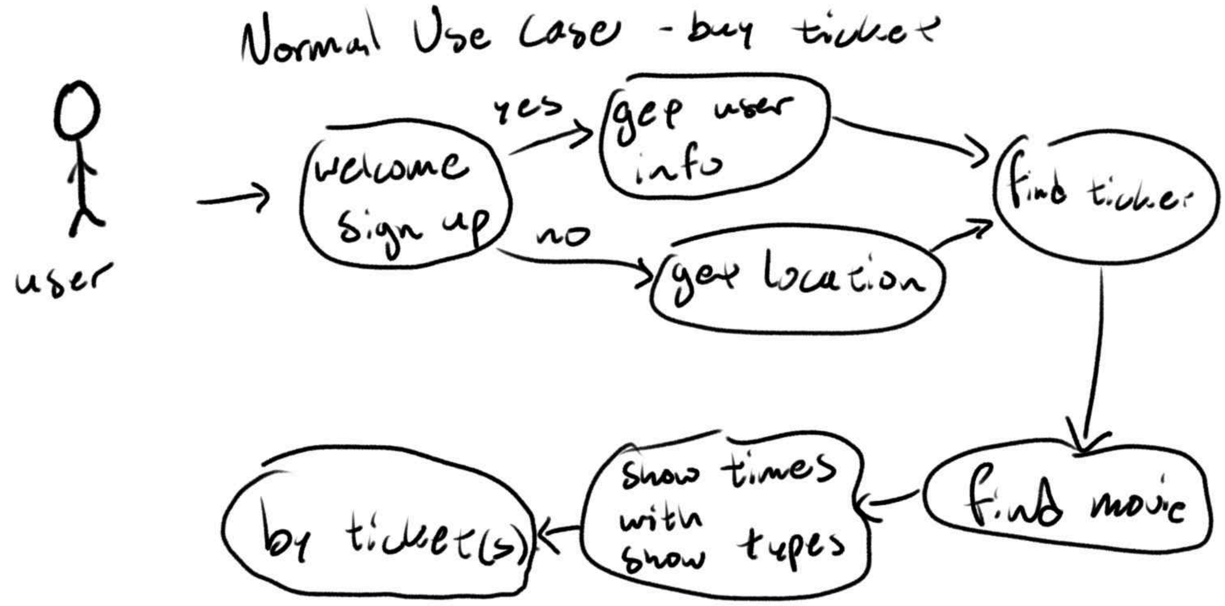
List of movies that failed (this job should scrape, clean, and store the data programmatically)

3.2.8.5 Error Handling

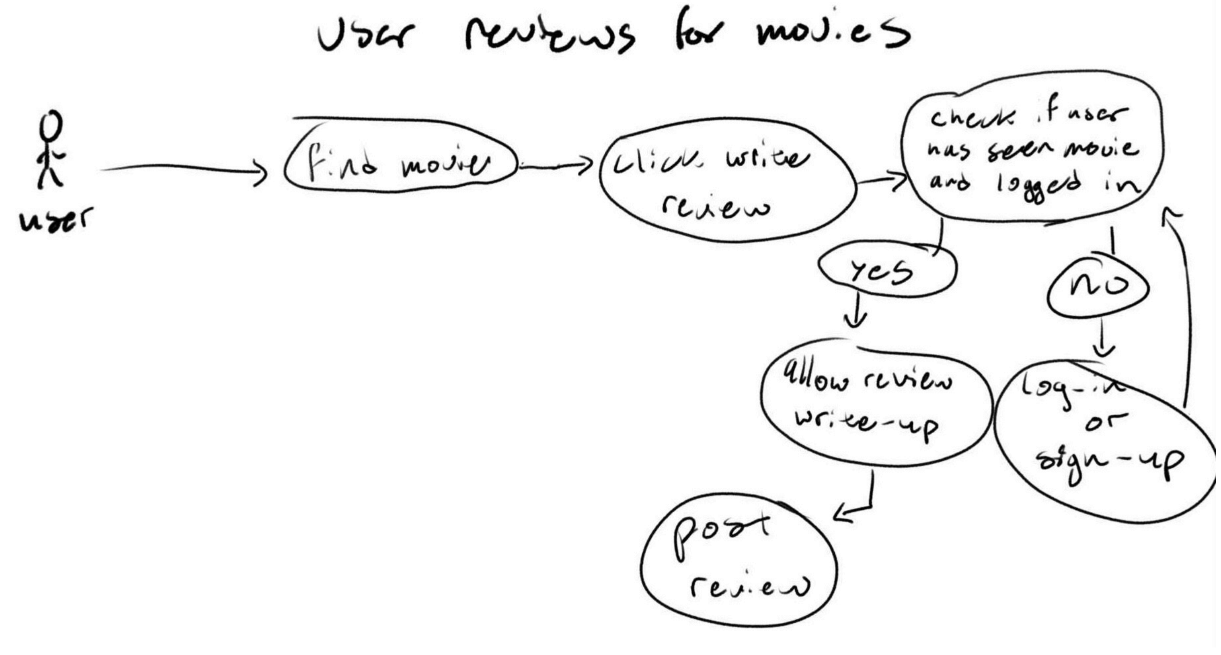
Each movie that fails to get scraped, should be added to the returned list and skipped over

3.3 Use Cases

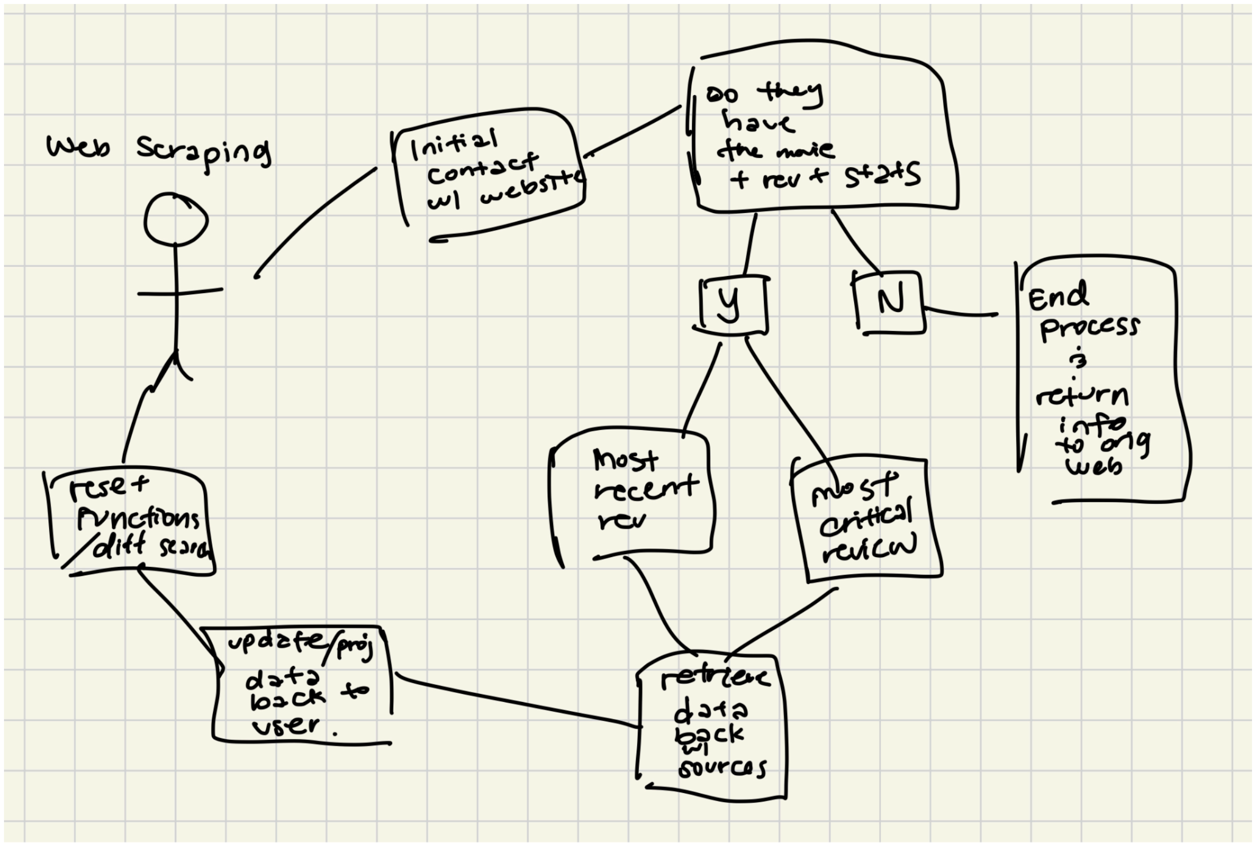
3.3.1 Use Case #1



3.3.2 Use Case #2



3.3.3 Use Case #3


Web Scraping
Initial contact w/ website
Do they have the movie + rev + stats
Y
N
End Process & return info to orig web
most recent rev
most critical review
retrieve data back w/ sources
update/proj data back to user.
reset functions /diff search

3.4 Classes / Objects

3.4.1 Order Object

3.4.1.1 Attributes
- user-id
- order-id
- List<Ticket>
- PaymentInfo

3.4.1.2 Functions
- create
  - lets you create an order object
- validate/submit
  - validates payment and submits order

<Ticket Purchase/Limits>

3.4.2 Ticket/Movie

3.4.1.1 Attributes
- event info
- time/date
- price
- name

- Customer Feedback Component

3.4.1.2 Functions

<Ticket Purchase/Limits>

3.4.3 Scraper

3.4.3.1 Attributes

- List of movies
- List of urls that can be found from movies given

3.4.3.2 Functions

- scrape

<Dynamic Scraper>

3.4.4 Customer Feedback Component

3.4.4.1 Attributes

- Comments left by users (includes likes, subcomments)

3.4.4.2 Functions

- CRUD operations

<Customer Feedback System>

3.4.5 Dashboard

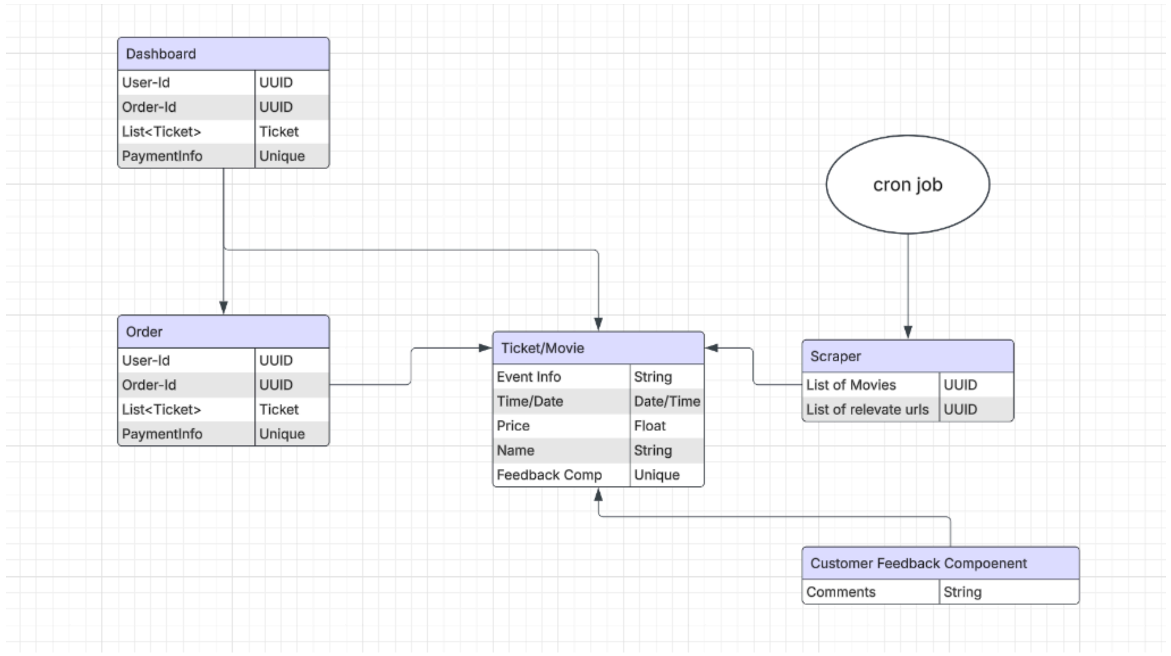
3.4.5.1 Attributes

- list of available tickets

3.4.5.2 Functions

- CRUD operations

<Dashboard>



3.5 Non-Functional Requirements

3.5.1 Performance

- Application shall have a response time of 2 seconds or less upon user input for all core functions, including browsing movies, searching, and navigating pages.
- Payment processing shall be handled through an external partner's payment gateway, ensuring a secure and efficient process.
- If a customer does not complete the payment within 10 minutes, the system shall prompt the user to confirm their presence before automatically redirecting them to the homepage.
- The web application should be optimized to ensure smooth performances even under peak load conditions, such as weekends and major movie release dates.

3.5.2 Reliability

- The system shall maintain an uptime of at least 99%, with planned downtimes only occurring during scheduled maintenance.
- The system must be designed to handle high concurrency, which ensures stability even when many users are simultaneously accessing it.
- Critical system failures must be minimized, and an automated error recovery mechanism will be in place to quickly restore service in case of unexpected failures.

3.5.3 Availability

- Scheduled maintenance windows shall occur outside peak movie-watching hours and should not exceed 1 to 2 hours.

- Users must receive advance notifications about maintenance periods, displayed as a banner at the top of the webpage.
- During maintenance, a maintenance mode message should be displayed, and users should not be able to make new transactions.

3.5.4 Security

- Users must authenticate themselves before performing actions such as purchasing tickets or leaving reviews.
- The system will implement multi-factor authentication for administrator accounts.
- Data encryption will be used for members logging on to the website to protect user information when checking out or inputting sensitive information.

3.5.5 Maintainability

- The system will support modular updates, which allows individual components to be fixed without affecting the entire system.
- A dedicated maintenance team shall track system performance, apply necessary patches, and ensure ongoing improvements.
- A versioning system will be in place to document and track all updates, with rollback options in case of critical errors.

3.5.6 Portability

- The application shall be accessible from any modern web browser.
- While the system is optimized for desktop users, it should be functional on mobile devices, even though mobile optimization is not a priority.
- Users must be able to access the website from any geographical location, provided they have an internet connection, except in cases where restrictions are legally required.
- The system will be designed for accessibility, ensuring ease of use for users with limited technical experience.

3.6 Inverse Requirements

- The website shall not display movie listings without official movie posters approved by the administrators.
- User IDs shall not exceed 18 characters, and users shall be prompted to enter a username that complies with this limitation.
- Only one active session per user is allowed at a time:

If a user attempts to log in from a second device, they will receive a notification stating that only one session is permitted at a time.

- The system shall provide the option to log out of the other session if the user wishes to switch devices.
- The system shall not allow ticket purchases to proceed without verifying the payment through the external payment gateway.
- The website shall not allow unverified users to submit movie reviews. Only users who have purchased a ticket for a specific movie and attended the screening shall be allowed to leave a review.

- The system shall not store plaintext passwords; all passwords must be hashed and salted before storage.
- Automated web scraping from unauthorized sources shall not be allowed. The system must comply with data-sharing policies and legal agreements with third-party review providers.

3.7 Design Constraints

Software Design Specification (SDS)

3.7.1. Introduction

This document provides a detailed Software Design Specification for the San Diego Ticketing System. It expands over the Software Requirements Specification by defining the system architecture, UML class structure, and any key design constraints.

3.7.2. Software Architecture

The architecture of this application follows a three-tiered architecture system.
Presentation (Frontend)

- User interface for movie browsing, ticket selection, and making payments.
- Built using React.js for a more responsive web experience.
- Includes authentication and session management.

Business Logic (Backend)

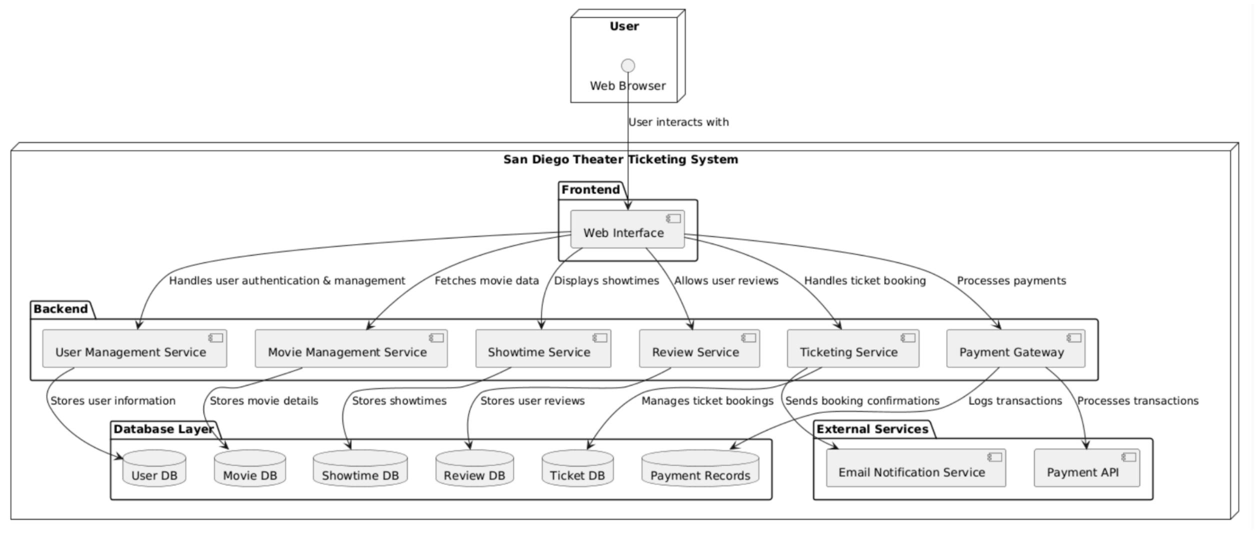
- Handling user authentication, ticket purchasing, and seat reservations.
- Built with Node.js
- Integrates with an external payment gateway (Paypal, etc).

Data Storage (Database)

- Stores movies, theaters, user accounts, and transactions.
- Ensures data integrity

UML Class Diagram

The UML Class Diagram defines the classes and their interactions explaining the core functionality of how frontend requests are handled by their respective backend technology.

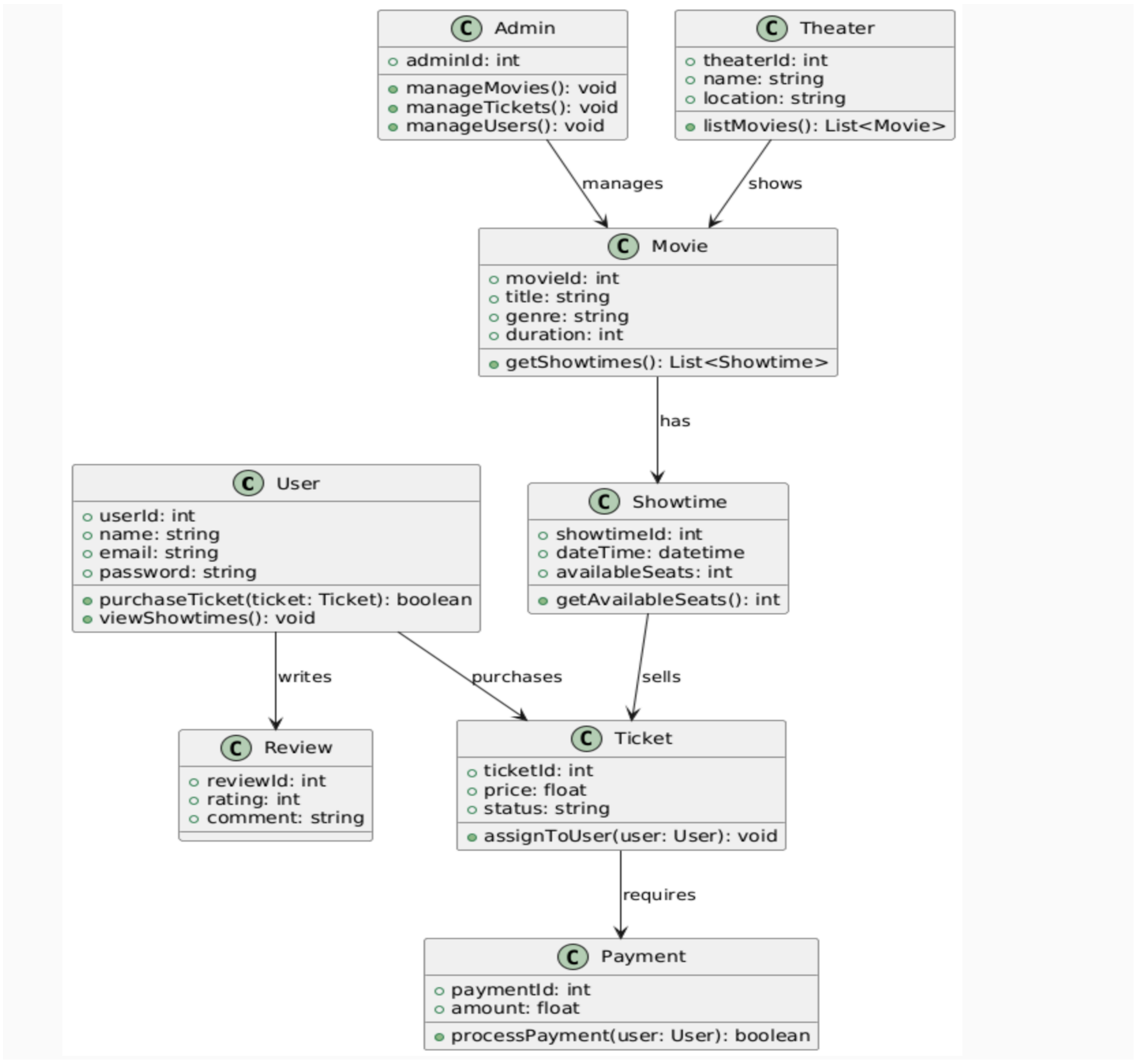


3.7.3 Class Descriptions:

- The order class can contain multiple ticket objects, meaning one order can include several tickets.
- The user class contains user information along with being linked to orders and payments.
- The movie class presents showtime information for ticket purchases.
- The payment class handles transaction processing.

SWA Diagram:

The SWA Diagram is the envisioned model of how the software architecture should be laid-out to have our main components efficiently interact with what is necessary.



3.7.4 SWA Diagram:

- The Admin and Theater objects need the access and ability to make changes to the Movie class
- The Movie object will collect and display the necessary information
- The Showtime object will then take the specific data needed from the Movie object to display the date, times and available seating.
- The User object then comes into play to handle ticket selection and purchasing
    - The User is also allowed to affect the Review object as well
- The Payment object is then handled on our end as our last call in this process of buying a ticket.

3.7.5 Company Policies:

This section is non-applicable to this document

3.7.6 Hardware Limitations:
There should be no hardware limitations due to how lightweight the stack should be.

3.8 Logical Database Requirements

There should be a database for all listed services, with keys being shared in what was shown in the relationship diagrams. In our case, all dbs should be SQL or relational, except for the inclusion of some caching for commonly accessed data (initial dashboard list of movies). This should be done using redis and logic to cache our default data.

3.9 Other Requirements

3.9 Other Requirements

Cron jobs are set to run the following operations:

Data Scraping: Pulling new movie data nightly via third-party APIs.

Redis Cache Refresh: Ensures dashboard freshness and up-to-date showtimes.

Review Moderation: Auto-flagging inappropriate content every hour.

3.10 SDS: Test Plan Cases

3.10.1 Test Objectives

States the overall goals of testing, such as verifying that all functional and nonfunctional requirements are met. This ensures system stability and validates user interactions.

3.10.2 Test Scope and Coverage

Our test covers:

- Unit testing: Focused on individual components like the Movie class, user authentication, and ticket purchase logic.
- Functional Testing: End-to-end tests verifying features like the ticket purchase process, user sign-in/sign-up, customer feedback submission, and administrative page functionality.
- System Testing: Complete system tests that stimulate real-world user interactions to ensure that all modules work together.

3.10.3 Test Strategy and Methods

Test Documentation

- Detailed test cases are documented on the XLSX document which includes 10 test case samples.

Manual and Automated Testing

- While our current test cases are manually executed, it is designed to support future automation of repetitive test scenarios.

Test Vectors

- Each test case includes input vectors, (e.g., number of tickets purchased, valid/invalid user, CAPTCHA responses), expected results, and error handling scenarios.

Environment Setup

- Testing is performed in a controlled environment simulating the production setup, including a web browser interface for front-end tests and access to the backend database and external APIs.

3.10.4 GitHub Repository Link

Group11 SDS Test Cases

3.10.5 Test Execution and Reports

Execution

- Each test case is executed by designated team members. Test results are recorded in the Excel document and any deviations are recorded as well.

Reporting

- A test report is generated which summarizes the pass and fail status of each test case.

Continuous Updates

- The test plan will be updated as needed based on the testing cycles and any modifications to the system design.

4. Analysis Models

4.1 Sequence Diagrams

- Use Case: Ticket Purchase
  - Actors: User, Showtime Service, Ticketing System, Payment Gateway
- Sequence
  - User selects the movie.
  - Requests available showtimes.
  - User chooses seats.
  - Ticket generated and payment initiated.
  - Confirmation returned and tickets are saved.

4.2 State-Transition Diagrams (STD)

- State Flow: Ticket Status
  - States: Available -> Selected -> Pending Payment -> Confirmed -> Used/Expired

4.3 Data Flow Diagrams (DFD)

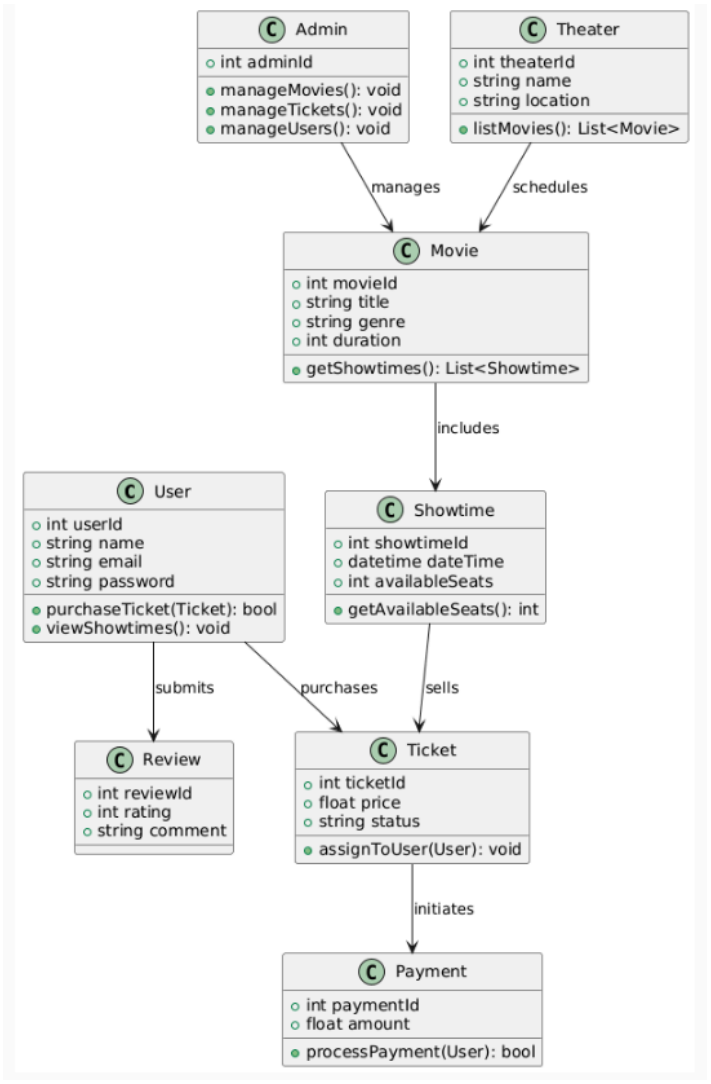
- Level 1 DFD shows interactions between User, Ticketing System, Payment Processor, and Databases.

5. Change Management Process

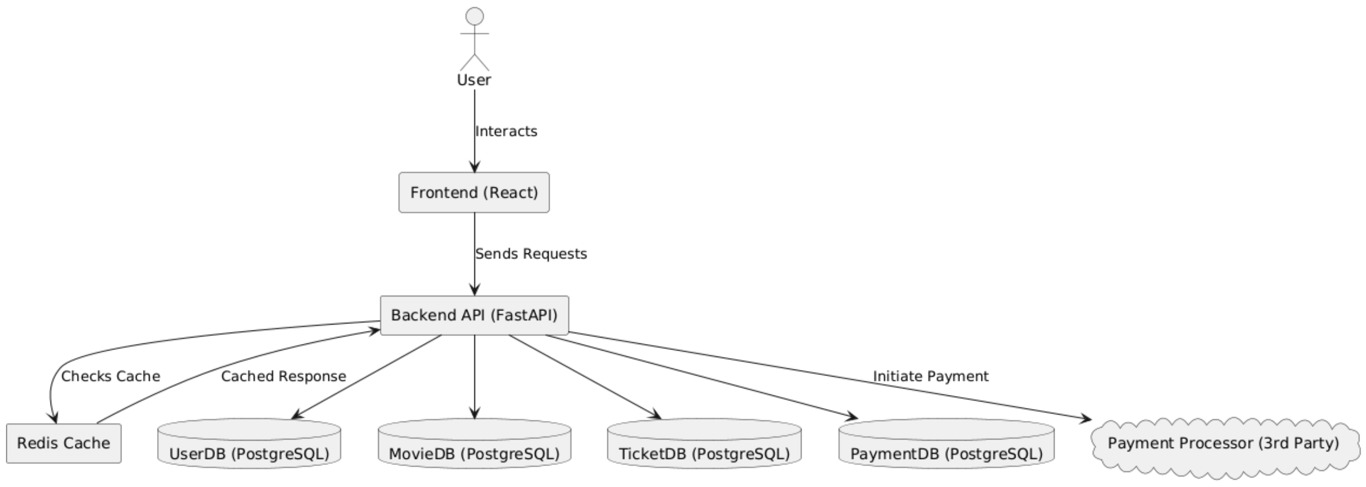
- Change Submission: All team members may submit requests via GitHub Issues or project management tools.
- Approval Process:
    - Change requests must be reviewed and approved by a lead developer and one peer.
    - Changes to core system components (database schema, auth logic) must pass a full test suite before merge.

A. Appendices

A.1 Appendix 1 UML Class Diagram



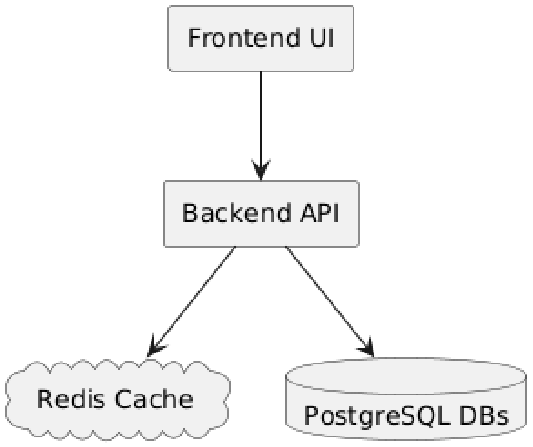
A.2 Appendix 2 Software Architecture Diagram



Appendix B: Data Management Strategy

- Chosen Approach: Hybrid SQL + Redis Caching

Diagram Overview:



- User Data → UserDB → secured with hashed passwords and OAuth support.
- Movies & Showtimes → MovieDB → normalized schema linking movies to theaters and showtimes.
- Tickets → TicketDB → fast-access indexes for ticket lookups.
- Payments → PaymentDB → stores only transaction metadata; sensitive info sent directly to a third-party processor.

Alternatives Considered

Option	Pros	Cons
MongoDB	Schema-flexible, rapid prototyping	Poor relational integrity
Single SQL DB	Simple deployment	Slower queries, no logical separation
Firebase	Real Time sync	Vendor lock-in, limited SQL features

Justification

- PostgreSQL was chosen for its balance of power, open-source nature, and support for complex queries and indexing.
- Redis improves performance for high-traffic, read-heavy endpoints.
- Logical separation of data aids scalability and clarity.

Appendix C: API Endpoints

*GET, POST, and DELETE are all HTTP methods that help with managing data
- GET: retrieve data
- POST: send/create data
- DELETE: delete data

* possible API endpoints may include but are not limited to

- **Movies:**
  - GET /movies: for a list of current available movies
  - GET /movies/id: for the specific movie details when requested
  - GET /movies/:id/showtime for the user to select a time to purchase tickets for
- **Movie Booking:**
  - POST / bookings: for creating a new user showtime booking
  - Get /bookings/:id: to log, manage, and review a specific booking made
  - DELETE /bookings/:id to allow for cancellations of specific movie bookings
- **Users:**
  - POST /users/register: used in user account creation
  - POST /users/login: for user authentication, session creation, and logging
  - GET /users/:id/bookings: allows for users to see their own specific movie bookings

Appendix D: Potential Tech Stacks

*Tech Stack is a term used to describe the main tools and core components used in creating apps

1) **M.E.R.N Stack:**

- frontend:	React
- backend:	Node.js and Express
- database:	MongoDB
- auth:	JWT or Auth0
- payment:	Stripe
- hosting:	Vercel

2) **M.E.V.N Stack:**

- frontend:	Vue.js
- backend:	Node.js and Express
- database:	MongoDB
- auth:	JWT
- payment:	Stripe
- hosting:	Vercel-frontend and Heroku-backend

3) ***Serverless* Stack:**

*not recommended because of high cost with scaling, but good for demoing and prototypes

- frontend:	React
- backend:	Firebase-functions
- database:	Firestore - no SQL, real time database
- auth:	Firebase
- payment:	Stripe
- hosting:	Firebase or Vercel