Movie Theater System

Software Requirements Specification

Version 01

05/26/2024

Group 5
Sushil Rawtani
Nhu Vo
Adrian Arguelles

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

Revision History

Date	Description	Author	Comments
<05/27/24>	<Version 1>	<Group 5>	<First Revision>

Document Approval

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

Signature	Printed Name	Title	Date
	<Your Name>	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to provide a detailed description of the Movie Theater system. This document will outline the functional and non-functional requirements, as well as the use cases for the system. The intended audience includes software developers, project managers, and stakeholders involved in the project.

1.2 Scope

The Movie Theater system will allow theater administrators to manage movie showtimes, ticket bookings, and customer information. It will also enable customers to browse movie schedules, book tickets, and manage their bookings online. The system aims to streamline theater operations and improve the customer experience.

1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirements Specification
- UI: User Interface
- DB: Database
- Admin: Administrator

1.4 References

- IEEE 830-1998 Recommended Practice for Software Requirements
- All documents provided in the lecture

1.5 Overview

This SRS document contains an introduction to the system, a general description of its functions and characteristics, and detailed specific requirements. The document is organized into sections that address each aspect of the system comprehensively.

2. General Description

This section of the SRS should describe the general factors that affect 'the product and its requirements. It should be made clear that this section does not state specific requirements; it only makes those requirements easier to understand.

2.1 Product Perspective

The Movie Theater system is a website designed to replace manual booking systems in theaters. It will integrate with existing payment gateways and support multiple theaters.

2.2 Product Functions

- Schedule movies
- Manage ticket bookings
- Handle customer information
- Generate reports on sales and occupancy

2.3 User Characteristics

- Administrators: Manage movies and showtimes, require administrative access.
- Cashiers: Handle ticket sales and customer queries, require cashier-level access.
- Customers: View movie listings, book and cancel tickets, manage personal information.

2.4 General Constraints

- The system must integrate with existing payment gateways.
- It should be compatible with common web browsers and mobile devices

2.5 Assumptions and Dependencies

- Users have basic computer literacy.
- The theater has reliable internet access.
- Payment gateway services are operational.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- The user interface for this website will be constructed using something like JavaScript/React. Users will be able to search for movies, filter movies by genre and title. Users will be able to select movies with available showtimes, and buy tickets for said movies, selecting where they want to sit.

3.1.2 Hardware Interfaces

- Working mouse and keyboard to click on buttons select movies and a keyboard to search for movies and input payment information.

3.1.3 Software Interfaces

- a Data Management System for storing user account information
- Integrate Paypal, to make purchases easier.

3.1.4 Communications Interfaces

The system shall use secure protocols for all data transmissions.

3.2 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

3.2.1 <User should be able to create accounts on the website, and account data should be stored and secured>

3.2.1.1 Introduction

The system will let **user be able to create accounts on the website in order to purchase tickets**

3.2.1.2 Inputs

Inputs include email, username, and password. Username shall be 3-20 characters, password shall be around 8 to 48 characters. Username and password will be letters, numbers, and basic symbols such as (, . : ; * & _). Usernames will be unique (only one person will be allowed to have a specific username).

3.2.1.3 Processing

The data above shall be processed and stored in the database.

3.2.1.4 Outputs

The account will have been created.

3.2.1.5 Error Handling

If the username is not unique or uses an illegal character, prevent the creation of the account and prompt the user to pick another username.

3.2.2 <Available movies and showtimes will be shown and cataloged on the website. >

Introduction: The system will contain available movies cataloged on the website, with available showtimes.

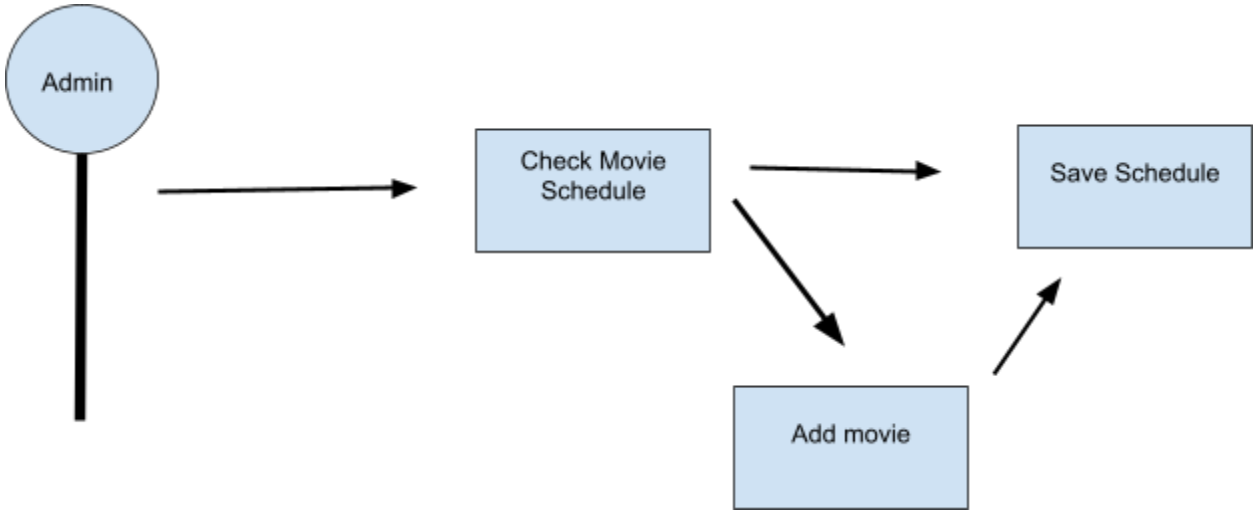
Inputs: **Click on a movie to view details and view available showtimes.**
Processing: **The system will access the webpage/information for the movie that was clicked on.**
Outputs: **The movie's details and showtimes will be displayed.**
Error: **If the movie's details are not available, redirect to a webpage that displays "This movie is not yet available."**
…

3.3 Use Cases

3.3.1 Use Case #1: Schedule Movie

- Actors: Administrator
- Description: The administrator adds a new movie schedule to the system.
- Preconditions: The administrator is logged in to the website.
- Postconditions: The new movie schedule is available for customers to view and book.

  Administrator navigates to the scheduling section.
  Administrator enters movie details and showtime.
  Administrator saves the schedule.



3.3.2 Use Case #2: Book Ticket

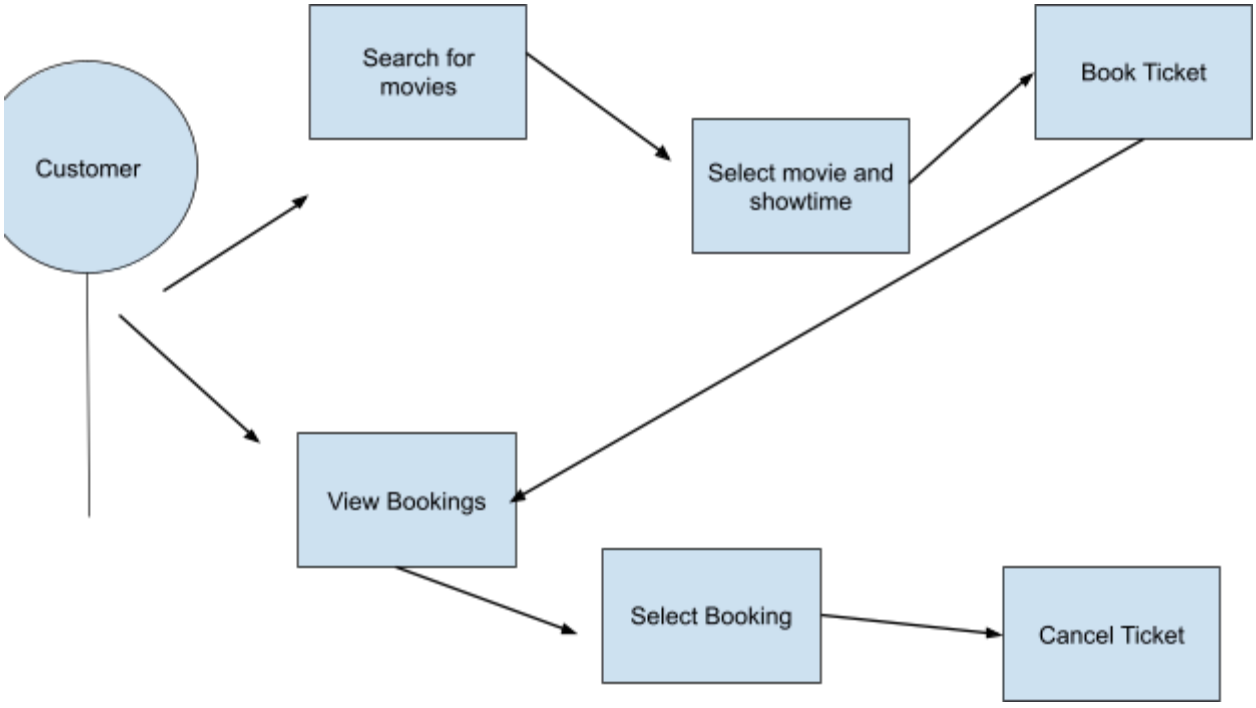
- Actors: Customer
- Description: A customer books a ticket for a movie.
- Preconditions: The customer is registered and logged in.
- Postconditions: The ticket is reserved, and the customer receives a confirmation.

1. Customer searches for the movie.
2. Customer selects a showtime.
3. Customer enters payment details.
4. System confirms booking.

3.3.3 Use Case #3: Cancel Ticket

- Actors: Customer
- Description: A customer cancels a previously booked ticket.
- Preconditions: The customer is logged in and has a valid booking.
- Postconditions: The booking is canceled, and the seat is available for others.

1. Customers navigate to their bookings.
2. Customer selects the ticket to cancel.
3. System processes the cancellation.



3.4 Classes / Objects

3.4.1 <Book Ticket>

3.4.1.1 Attributes

Username: The username of the account that purchased the ticket
Movie: The movie being booked
Seat: What seat was booked
Date of Movie: The date the movie is playing
Price: The price of the ticket
Purchase Details: The purchase details, to identify the specific purchase.

3.4.1.2 Functions

startBooking(): make a new booking
getPrice(): Get and calculate the price of the ticket
confirmBooking(): Confirm the booking of the ticket
getDetails(): get the details of the booking.

3.4.2 <Movie>

Attributes:
movie: The name of the movie
genre: the genre of the movie

cast: the cast of the movie
rating: the age rating of the movie
synopsis: what the movie is about

Functions:
createMovie() - create movie with all the details;
updateMovie() - update an existing movies details
getMovieDetails() - get the details of the movie

3.5 Non-Functional Requirements

3.5.1 Performance

The system will be designed to perform 90% of user requests within 1-2 seconds, and create bookings within 3 seconds.

3.5.2 Reliability

The system will be designed to keep backups of user data and the catalog of movies in order to maintain reliability. Any errors that occur during the operation of the system will be handled within one hour - two hours.

3.5.3 Availability

The system shall be available at all times, except for when the website needs full maintenance.

3.5.4 Security

The system will be designed in a way so that user account data is secure in the database. Since we are dealing with account data and payment information, we will prioritize a high level of security in our system to prevent sensitive data being stolen.

3.5.5 Maintainability

The system will be maintained by the team working on this document.

3.5.6 Portability

The system will be designed to work and be accessible on common web browsers.

_

3.6 Inverse Requirements

*State any *useful* inverse requirements.*

3.7 Design Constraints

Specify design constrains imposed by other standards, company policies, hardware limitation, etc. that will impact this software project.

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 the SRS's requirements.

4.1 Sequence Diagrams

4.3 Data Flow Diagrams (DFD)

4.2 State-Transition Diagrams (STD)

5. Change Management Process

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

A. Appendices

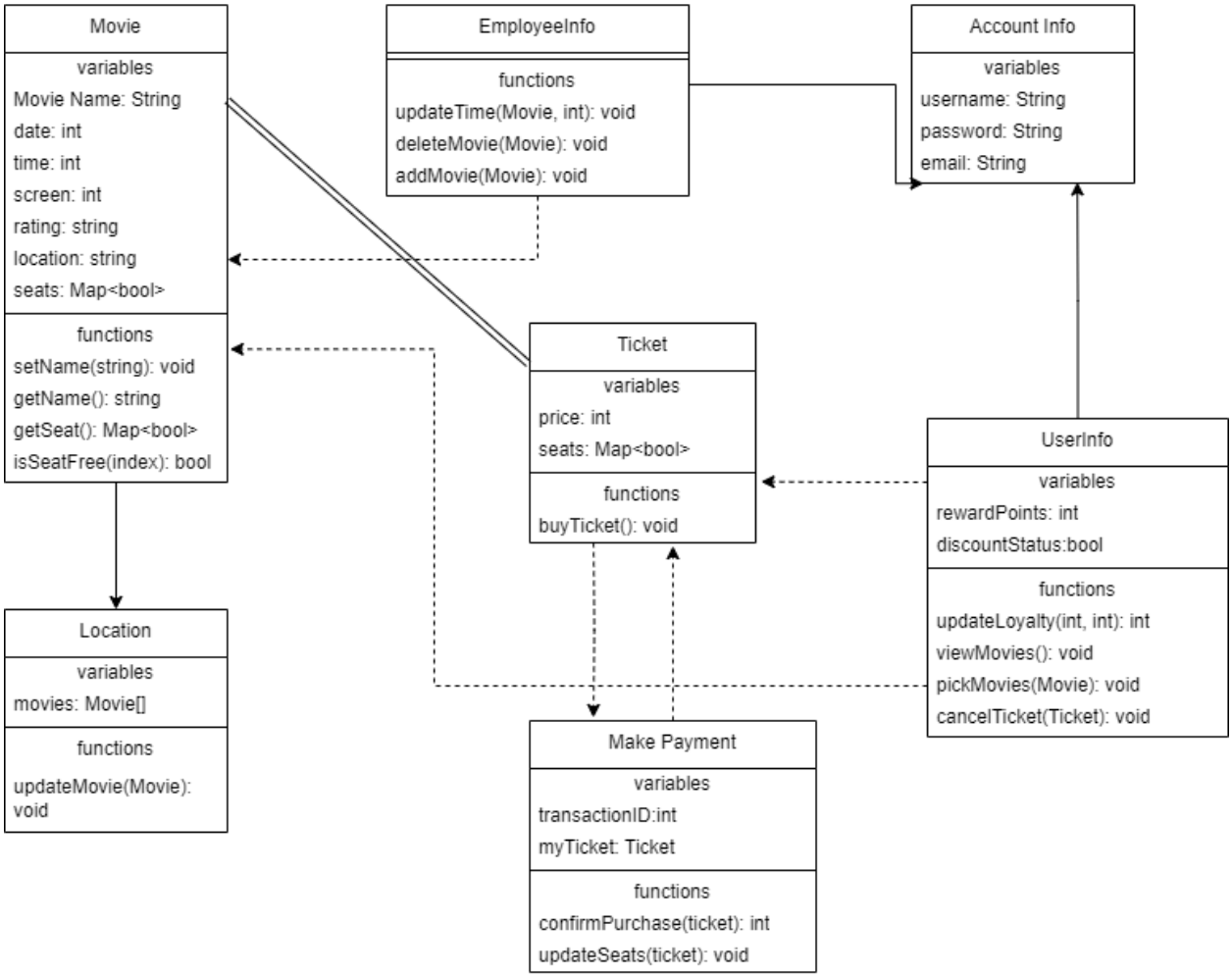
Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements.

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

A.1 Appendix 1

A.2 Appendix 2

UML Class Design



UML Class Design Descriptions

The above diagram shows our class diagram, consisting of 7 classes. This section will go in depth of each class and their variables and parameters.

The Movie class contains variables, relevant details that are needed for a movie. This includes, of course, the movie name (string), the date(int), time(int), rating(int), etc. of a given movie. This class contains simple functions such as a setter and getter for the movie name, getSeat(), a boolean function isSeatFree() to check if a seat is available.

The Location class simply contains a list of movies available at a current location, and a function, updateMovie(Movie) to update the movies at a given location.

The EmployeeInfo class is responsible for managing the details and operations related to employees who handle movie data. This class contains various functions to update, delete, and add movies. Specifically, it includes the following functions: updateTime(Movie, int) : void is the function that updates the time of a specified movie. deleteMovie(Movie) : void is the function that deletes a specified movie from the system. addMovie(Movie) : void is function adds a new movie to the system.

The Account Info class manages the login credentials of the users. It contains variables that store the username, password, and email address of the account holder: username : String is the username of the account holder. password : String is the password associated with the account. email : String is the email address linked to the account.

The Ticket class manages the details related to a movie ticket. This includes the ticket price and a map representing the availability of seats. The class includes a function to buy a ticket: price(int) is the price of the ticket. seats : Map<bool> is a map indicating the availability of seats. buyTicket() : void is the function facilitates the purchase of a ticket.

The UserInfo class manages user-related details and operations. It includes variables to store the user's reward points and discount status. Additionally, it provides several functions for user operations: rewardPoints : int is the reward points accumulated by the user. discountStatus : bool is the discount status of the user. updateLoyalty(int, int) : int is the function that updates the user's loyalty points. viewMovies() : void allows the user to view available movies. pickMovies(Movie) : void enables the user to select a movie. cancelTicket(Ticket) : void allows the user to cancel a ticket.

The Make Payment class handles the payment process for purchasing tickets. It includes variables to store the transaction ID and the ticket associated with the transaction. The class also provides functions to confirm the purchase and update seat availability: transactionID : int: is the ID of the transaction. myTicket : Ticket is the ticket associated with the transaction. confirmPurchase(ticket) : int to confirm the purchase of a ticket and returns a transaction ID. updateSeats(ticket) : void to update the seat availability after a purchase.