Software Requirements Specification Data Dictionary & Data Flow Diagram

Project name : 🎬 Movie Booking Website

Developers : Krish Patel , Jayesh Patil

Table of Contents

🎬 Movie Booking Website - SRS , Data Dictionary & Data Flow Diagram

Software requirements specification

📌 1. Introduction

📖 1.1 Purpose

This document outlines the software requirements for the **Cinema Booking System (CBS)**, a web-based platform developed using the **MERN stack**. It describes the expected behavior of the system, the functionalities it should offer, and the constraints and limitations the system should adhere to.

🎯 1.2 Scope

The **Cinema Booking System** is designed to provide a seamless and intuitive interface for users to book movie tickets online. The system offers functionalities such as movie listings, seat bookings, ticket management, and administrative controls. The system will be deployed as a **responsive web application** that supports multiple user roles: **Viewer**, **Registered User**, and **Administrator**.

🔍 1.3 Definitions and Abbreviations

- **MERN Stack**: A combination of MongoDB, Express.js, React.js, and Node.js for full-stack web development.
- **JWT**: A secure method of transmitting information between parties in a compact format.

📚 1.4 References

- **React** v18.2.0
- **Node.js** v16.14.0
- **MongoDB** v5.0
- **Express.js** v4.18.2
- **Tailwind CSS** v3.3.3

🌍 2. Overall Description

💻 2.1 Product Perspective

The **Cinema Booking System (CBS)** will operate as a **single-page application (SPA)** for both the **viewer and registered users**. The administrative functionality is integrated into the same platform but with restricted access. Users can access the system on both mobile devices and desktops with a responsive design that adjusts based on the screen size.

⚙️ 2.2 Product Functions

- **Movie Listings**: Display a list of currently available movies with details like genre, runtime, rating, etc.
- **Showtime Selection**: Users can select the desired showtime based on their preferred movie.
- **Seat Booking**: Interactive seating chart allowing users to select available seats.
- **Ticket Management**: View past bookings, download tickets, and cancel bookings.
- **Admin Dashboard**: Admin can manage cinemas, movies, and showtimes.

👤 2.3 User Characteristics

- **Viewer**: Users who can browse movies, check showtimes, and view cinema details.
- **Registered User**: Users who have an account and can book tickets, view their booking history, and manage their profile.
- **Administrator**: System users who have permission to manage movies, cinemas, showtimes, and user roles.

⚠️ 2.4 Constraints

- **Browser Compatibility**: The system must be compatible with all major browsers (Chrome, Firefox, Safari, etc.).
- **Security**: All personal user information must be protected using industry-standard encryption techniques (e.g., bcrypt for password hashing).
- **Scalability**: The system must be able to scale to handle increasing numbers of users, especially during peak hours (e.g., during movie premieres).

3. Functional Requirements

3.1 User Authentication

- **Registration**: New users can register by providing a username, email, and password.
- **Login/Logout**: Returning users can log in securely using their credentials.
- **JWT Authentication**: Secure authentication method with token-based sessions. JWT will be issued on login and used for subsequent requests to authenticate users.

3.2 Movie and Showtime Management

- **Movie Browsing**: Users can view available movies, including descriptions, cast, ratings, and showtimes.
- **Admin Features**: Admins can add, edit, and delete movies and showtimes. Admins can also update movie availability based on showtimes.

3.3 Ticket Booking

- **Showtime Selection**: Users can select from a list of available movie showtimes.
- **Seat Selection**: Interactive seating chart displaying available (▨) and unavailable (▧) seats.
- **Booking Confirmation**: After seat selection, users will receive a booking confirmation email and see a confirmation page.
- **Booking History**: Registered users can view their booking history and rebook tickets if needed.

3.4 Seat Selection

- **Seats Availability**: Each movie's available seats will be displayed. Seats can be color-coded as available, reserved, or unavailable.
- **Multiple Seat Booking**: Users can select multiple seats in one go.
- **Dynamic Availability**: Once a user selects seats, they become unavailable to others until the booking process is completed.

🛠️ 3.5 Administrative Features

- **Cinema Management**: Admin can add, edit, and delete cinemas, and assign theaters to each cinema.
- **User Management**: Admin can view user profiles, change user roles (admin/user), and deactivate user accounts.
- **Schedule Management**: Admin can add and manage showtimes for movies across different cinemas.

🚀 4. Non-Functional Requirements

⚡ 4.1 Performance

- The system must be able to handle at least **500 concurrent users** at any given time during peak hours.
- The system should perform all interactions (e.g., booking tickets, searching for movies) within **3 seconds** under normal usage.

🔓 4.2 Security

- **Data Encryption**: All sensitive data, including passwords and payment details, must be encrypted using **SSL/TLS** encryption during transmission.
- **Role-based Access Control (RBAC)**: The system will implement strict access control measures for each role (Viewer, Registered User, Admin).

💻 4.3 Usability

- **Responsive UI**: The interface should adjust for mobile, tablet, and desktop use cases.
- **Accessibility**: Ensure the platform is accessible to users with disabilities, including screen reader compatibility and keyboard navigability.
- **Error Handling**: Users should receive clear, actionable error messages.

🏗️ 5. System Architecture

🛠️ 5.1 Technologies Used

- **Frontend**: React (v18.2.0), Tailwind CSS, React Router for routing.
- **Backend**: Node.js, Express.js for API routes, MongoDB for database management.
- **Database**: MongoDB with Mongoose ORM for database interactions.
- **Authentication**: JWT-based authentication for secure logins and user management.
- **Deployment**: Client hosted on **Vercel** and backend hosted on **AWS** using EC2 instances.

☁️ 5.2 Deployment and Hosting

- **Client Hosting**: Vercel provides scalable hosting for the frontend, optimizing for speed and security.
- **Backend Hosting**: The backend is hosted on an AWS EC2 instance, connected to **MongoDB Atlas** for cloud-based database management.
- **Backup & Disaster Recovery**: Regular database backups will be performed to ensure data integrity in case of system failure.

🔗 6. Assumptions and Dependencies

- **Internet Connectivity**: A stable internet connection is required for accessing the Cinema Booking System.
- **Database Availability**: The MongoDB database will be maintained and regularly updated to ensure data availability.
- **Third-party Integrations**: The payment gateway will rely on third-party services (e.g., Stripe or PayPal) for transaction processing.

📄 7. Appendices

UI Screenshots

Screenshots of the **movie listing page**, **seat selection page**, and **admin dashboard** will be included as references for developers and designers.

API Documentation

A detailed list of RESTful API endpoints, including methods (GET, POST, PUT, DELETE), required parameters, and responses for each endpoint.

Use Case Examples

Use Case 1: Booking a Movie Ticket

- **Actors**: Registered User
- **Preconditions**: User is logged in.
- **Flow**:
    1. User selects a movie.
    2. User selects a showtime.
    3. User selects seats.
    4. User confirms the booking.
    5. System sends a booking confirmation.

Use Case 2: Admin Managing Movies

- **Actors**: Admin
- **Preconditions**: Admin is logged in with the necessary privileges.
- **Flow**:
    1. Admin accesses the movie management page.
    2. Admin adds, edits, or deletes a movie.
    3. Changes are saved to the database.

🎬 Movie Booking Website Data Dictionary

1. Cinema 🏢

Description: Physical cinema location containing multiple theaters

Attribute	Type	Description	Constraints	References
name	String	Unique cinema name	Required, Unique, Trimmed	-
theaters	Array[ObjectId]	Theaters in this cinema	-	Theater
createdAt	Date	Creation timestamp	Auto-generated	-
updatedAt	Date	Update timestamp	Auto-generated	-

🔗 Relationships:

- One-to-Many with Theater
- Cascading delete: Cinema → All Theaters

2. Movie 🎥

Description: Movie available for screening

Attribute	Type	Description	Constraints	References
name	String	Movie title	Required, Trimmed	-

length	Number	Duration in minutes	Required	-
img	String	Poster image URL	Required, Trimmed	-
createdAt	Date	Creation timestamp	Auto-generated	-
updatedAt	Date	Update timestamp	Auto-generated	-

🔗 **Relationships:**

- One-to-Many with Showtime
- Cascading delete: Movie → All Showtimes

3. Showtime 🕒

Description: Scheduled movie screening in a theater

Attribute	Type	Description	Constraints	References
theater	ObjectId	Screening location	Required	Theater
movie	ObjectId	Movie being shown	Required	Movie
showtime	Date	Screening date/time	Required	-
seats	Array	Booked seats	-	-
seats.row	String	Seat row (A-Z)	Required	-
seats.number	Number	Seat number	Required	-
seats.user	ObjectId	Seat owner	-	User
isRelease	Boolean	Showtime status	-	-

🔗 **Relationships:**

- Many-to-One with Theater and Movie
- Cascading update: Showtime → User tickets

4. Theater 🪑

Description: Physical screen/room in a cinema

Attribute	Type	Description	Constraints	References
cinema	ObjectId	Parent cinema	Required	Cinema
number	Number	Theater number	Required	-
seatPlan.row	String	Row identifier	Required, Max 2 chars	-
seatPlan.column	Number	Seats per row	Required	-
showtimes	Array[ObjectId]	Scheduled showtimes	-	Showtime

🔗 Relationships:

- Many-to-One with Cinema
- One-to-Many with Showtime
- Cascading delete: Theater → All Showtimes

5. User 👤

Description: System user for ticket booking

Attribute	Type	Description	Constraints	References
username	String	Unique identifier	Required, Unique	-
email	String	Contact email	Required, Unique, Regex validated	-
role	String	User permissions	Enum: ['user', 'admin'], Default: 'user'	-
password	String	Hashed secret	Required, Min 6 chars	-
tickets	Array	Booked tickets	-	-
tickets.showtime	ObjectId	Showtime reference	-	Showtime
createdAt	Date	Account creation date	Auto-generated	-

🔗 Relationships:

- One-to-Many with Tickets
- Dependent on Showtime references

📌 System Notes

⚡ Cascading Actions:

- 🎟️ Cinema deletion → All Theaters → All Showtimes
- 🎬 Movie deletion → All associated Showtimes
- 🪑 Theater deletion → All associated Showtimes
- ⏰ Showtime deletion → User ticket references removed

🔐 Security Features:

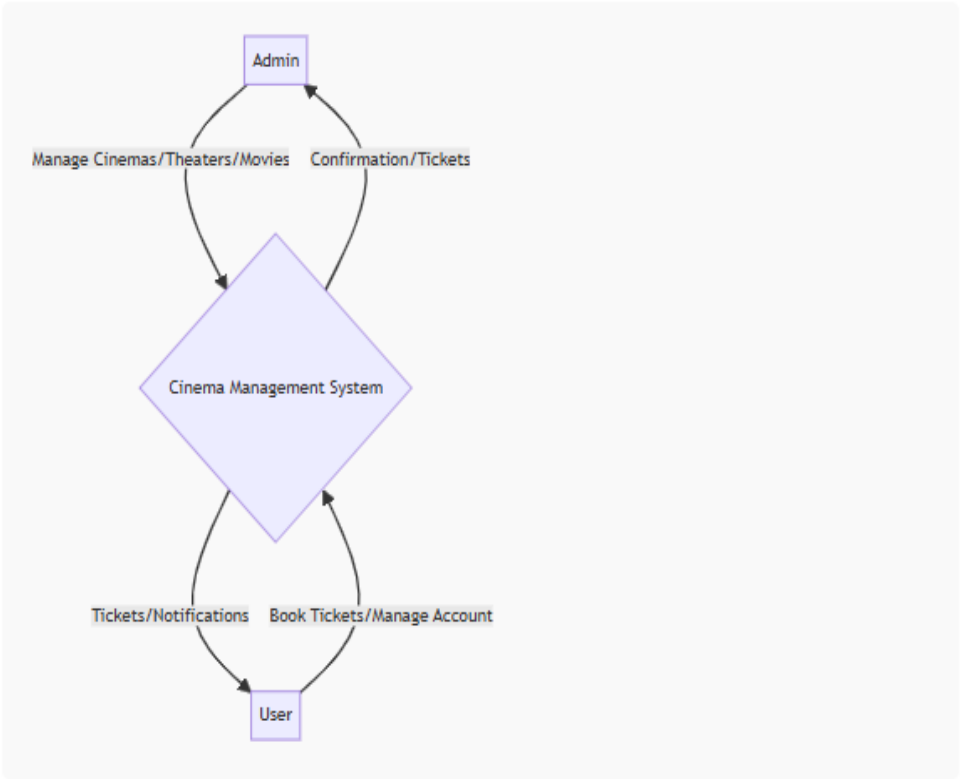
- 🔑 Password hashing with bcrypt
- 🛡️ JWT authentication system
- 👮 Role-based access control

⚠️ Important Notes:

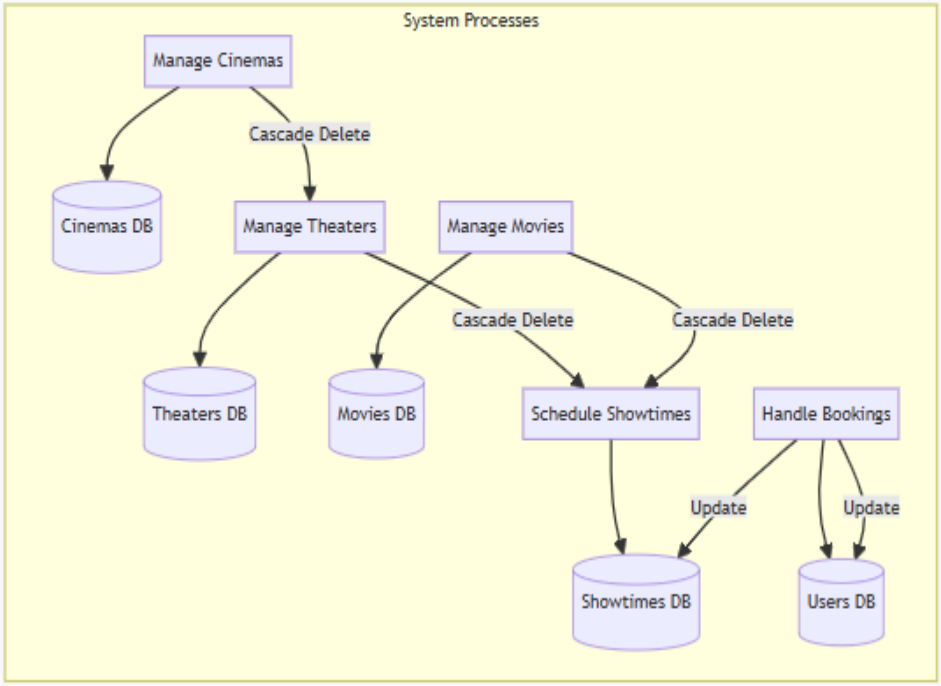
- 📧 Email validation using regex pattern
- 👥 User role has typo in schema ('defalut') but functions correctly
- 💺 Seat plan limited to 2-character row identifiers

Movie Booking Website DFD

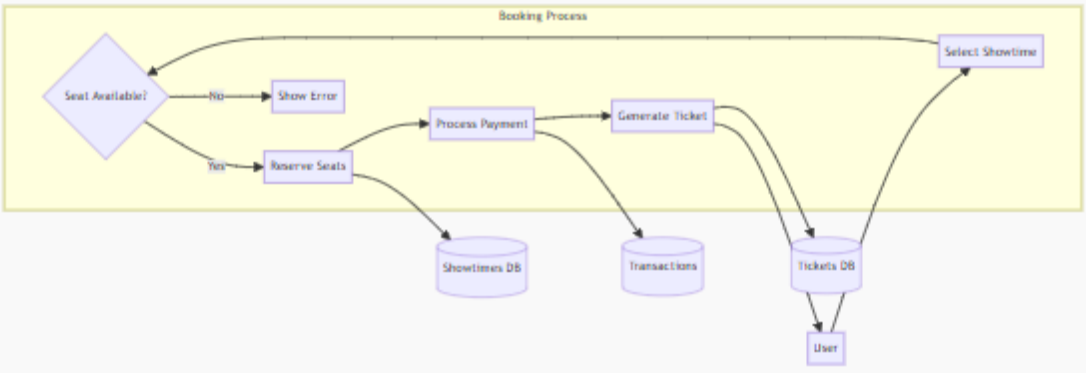
Level 0: Context Diagram



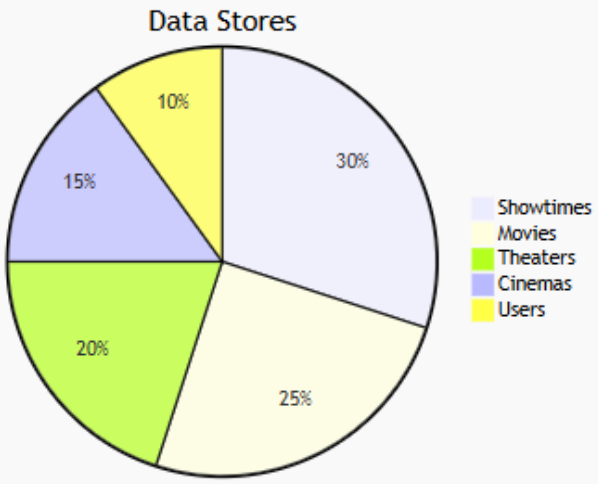
Level 1: Process Decomposition



Level 2: Detailed Booking Process



Data Dictionary



Data Flow Matrix