Movie Theater Ticketing System

Software Design Specification

Version 1.2

02/28/2024

Group 21
Alexander Kondan
Luiza Rocha
Shaun Romero

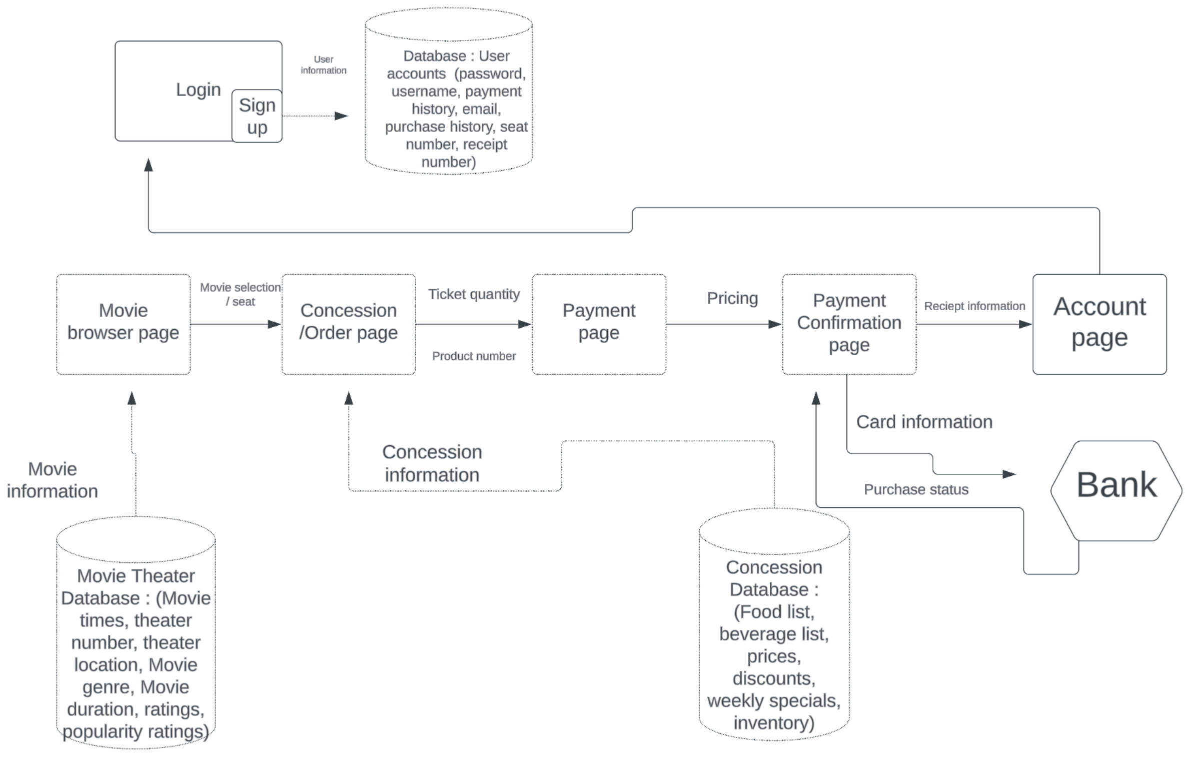
Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Fall 2023

System Description:

The scope of the Movie Theatre Ticketing System is to create a functional website to provide an integrated online real-time reservation, browse movies, list of theaters, payment transaction order and electronic information dissemination such as email confirmation and promotion to its members. This system will allow users to purchase tickets, create user accounts, and provide concession access before ever stepping into the theater. Users will be able to manage their accounts, purchase tickets, view movies, and purchase food and beverages all from their computer or mobile devices. The website's main objective is to provide movie going customers with an easy and pleasant experience.

Software Architecture Overview:

- Architectural Diagram: (swa diagram)


Login
Sign up
User information
Database : User accounts (password, username, payment history, email, purchase history, seat number, receipt number)
Movie browser page
Movie selection / seat
Concession /Order page
Ticket quantity
Product number
Payment page
Pricing
Payment Confirmation page
Reciept information
Account page
Card information
Purchase status
Bank
Movie information
Concession information
Movie Theater Database : (Movie times, theater number, theater location, Movie genre, Movie duration, ratings, popularity ratings)
Concession Database : (Food list, beverage list, prices, discounts, weekly specials, inventory)

- **Data Management Strategy:**

1. Database Design:
   - Single Database Approach: Utilizing a single SQL database for the entire ticketing system simplifies data management and ensures consistency.

2. Database Schema:
   - Movies: Contains information about movies such as title, genre, duration, movie times etc.
   - Theaters: Stores details about theaters including location, seating capacity, etc.
   - Customers: Holds customer information like name, email, phone number, etc.
   - Tickets: Records details of tickets purchased including movie, showtime, seat number, etc.
   - Showtimes: Manages information about movie showtimes including start time, end time, theater, etc.

3. Normalization:
   - Employ normalization techniques to reduce redundancy and improve data integrity, for example: store customer information separately from ticket details to avoid duplication.

4. Security: Implement access control mechanisms, encryption, and other security measures to keep sensitive data safe like customer payment and information.

5. Alternatives and Tradeoffs of SQL:

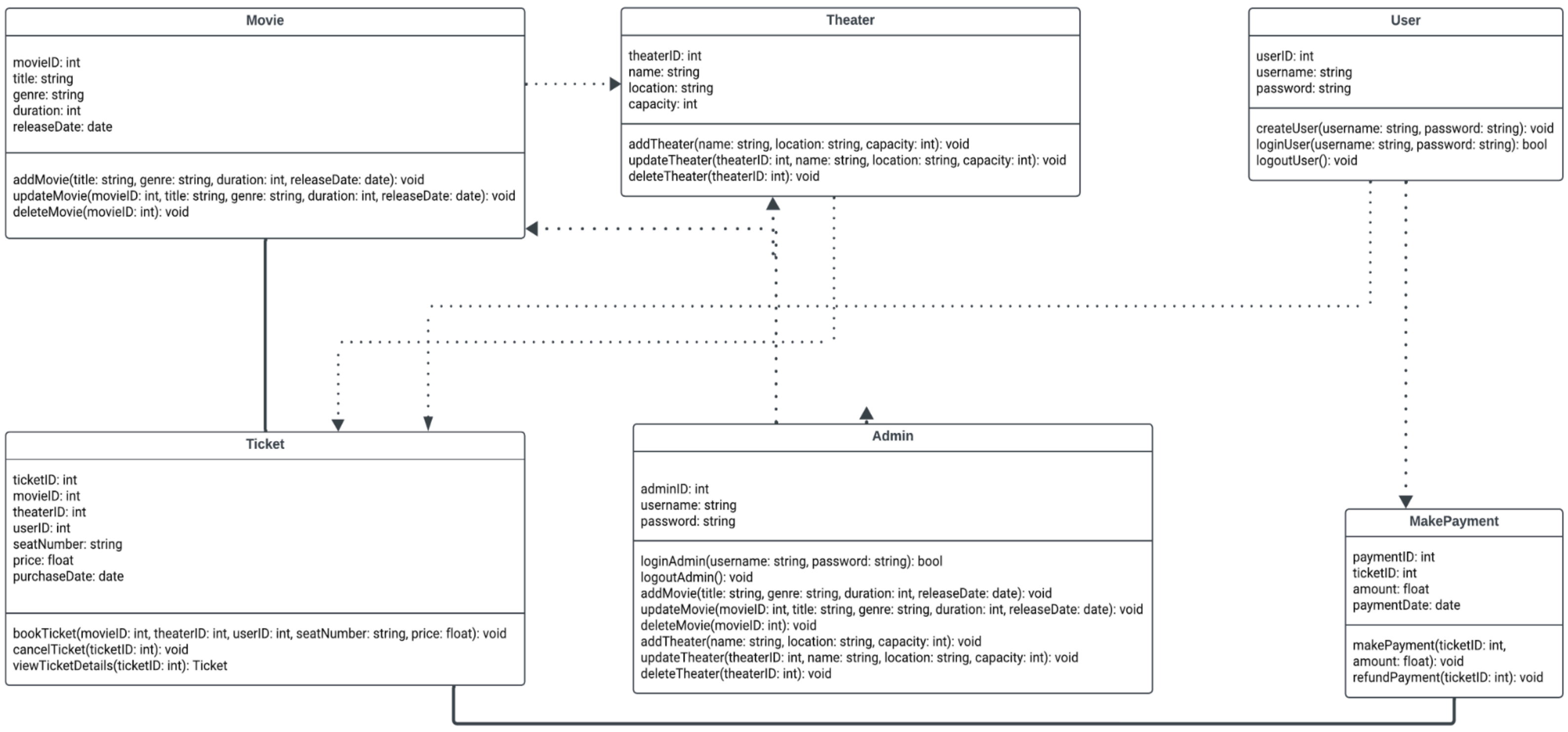
Pros:
- Data Integrity and Consistency: The transactional nature of ticket sales demands strong data integrity and consistency, which SQL databases are well-equipped to provide through features like ACID (Atomicity, Consistency, Isolation, Durability) properties.
- Security: SQL databases offer robust security features such as user authentication, access control, and encryption, which are crucial for protecting sensitive customer information like payment details.
- Structured Data: Ticketing systems deal with structured data such as customer information, showtimes, seating availability, etc., making SQL databases a natural fit due to their support for well-defined schemas and complex queries.
- Maturity and Reliability: SQL databases have been around for decades, offering a mature and reliable solution for data management in various industries, including entertainment and hospitality.

Cons:

- NoSQL, database sharding, microservices architecture, or hybrid approaches offer scalability and flexible benefits, however they introduce complexities and tradeoffs in terms of data consistency, and transaction management which we want to avoid seeing as we're valuing our clients' transactional data the most.

6. Data manag

UML Diagram:


Movie
movieID: int
title: string
genre: string
duration: int
releaseDate: date
addMovie(title: string, genre: string, duration: int, releaseDate: date): void
updateMovie(movieID: int, title: string, genre: string, duration: int, releaseDate: date): void
deleteMovie(movieID: int): void
Theater
theaterID: int
name: string
location: string
capacity: int
addTheater(name: string, location: string, capacity: int): void
updateTheater(theaterID: int, name: string, location: string, capacity: int): void
deleteTheater(theaterID: int): void
User
userID: int
username: string
password: string
createUser(username: string, password: string): void
loginUser(username: string, password: string): bool
logoutUser(): void
Ticket
ticketID: int
movieID: int
theaterID: int
userID: int
seatNumber: string
price: float
purchaseDate: date
bookTicket(movieID: int, theaterID: int, userID: int, seatNumber: string, price: float): void
cancelTicket(ticketID: int): void
viewTicketDetails(ticketID: int): Ticket
Admin
adminID: int
username: string
password: string
loginAdmin(username: string, password: string): bool
logoutAdmin(): void
addMovie(title: string, genre: string, duration: int, releaseDate: date): void
updateMovie(movieID: int, title: string, genre: string, duration: int, releaseDate: date): void
deleteMovie(movieID: int): void
addTheater(name: string, location: string, capacity: int): void
updateTheater(theaterID: int, name: string, location: string, capacity: int): void
deleteTheater(theaterID: int): void
MakePayment
paymentID: int
ticketID: int
amount: float
paymentDate: date
makePayment(ticketID: int, amount: float): void
refundPayment(ticketID: int): void

Student to GRADER NOTE: Couldn't get image to appear clear on doc. Please follow link to see more clearly:
https://lucid.app/lucidchart/01e71a52-b6bd-428c-b80f-12b644530d1e/edit?viewport_loc=-2358%2C-492%2C2750%2C1291%2C0_0&invitationId=inv_43cab007-a494-46d8-9da2-fb84c61e9685

UML Class Description:

1. Class: User

Description of Class:

This class represents a user of the movie ticketing system. Users can create accounts, log in, and book tickets for movies.

Attributes:

- userID: int - uniquely identifies each user in the system.
- username: string - stores the username of the user.
- password: string - stores the password of the user's account.

Operations:

- createUser(username: string, password: string): void
    - Description: Creates a new user account with the given username and password.
- loginUser(username: string, password: string): bool
    - Description: Logs in the user with the provided username and password. Returns true if login is successful, false otherwise.
- logoutUser(): void
    - Description: Logs out the current user from the system.

2. Class: Movie

Description of Class:

This class represents a movie available in the system. Admins can add, update, or delete movies.

Attributes:

- movieID: int - uniquely identifies each movie in the system.
- title: string - stores the title of the movie.
- genre: string - stores the genre of the movie.

- duration: int - stores the duration of the movie in minutes.
- releaseDate: date - stores the release date of the movie.

Operations:

- addMovie(title: string, genre: string, duration: int, releaseDate: date): void
  - Description: Adds a new movie to the system with the provided details.
- updateMovie(movieID: int, title: string, genre: string, duration: int, releaseDate: date): void
  - Description: Updates the details of the movie identified by movieID with the provided information.
- deleteMovie(movieID: int): void
  - Description: Deletes the movie with the specified movieID from the system.

3. Class: Theater

Description of Class:

This class represents a theater where movies are screened. Admins can add, update, or delete theaters.

Attributes:

- theaterID: int - uniquely identifies each theater in the system.
- name: string - stores the name of the theater.
- location: string - stores the location of the theater.
- capacity: int - stores the maximum seating capacity of the theater.

Operations:

- addTheater(name: string, location: string, capacity: int): void
  - Description: Adds a new theater to the system with the provided details.
- updateTheater(theaterID: int, name: string, location: string, capacity: int): void
  - Description: Updates the details of the theater identified by theaterID with the provided information.
- deleteTheater(theaterID: int): void
  - Description: Deletes the theater with the specified theaterID from the system.

4. Class: Ticket

Description of Class:

This class represents a ticket booked by a user for a movie screening.

Attributes:

- ticketID: int - uniquely identifies each ticket in the system.
- movieID: int - stores the ID of the movie for which the ticket is booked.
- theaterID: int - stores the ID of the theater where the movie is screened.
- userID: int - stores the ID of the user who booked the ticket.
- seatNumber: string - stores the seat number allocated for the ticket.
- price: float - stores the price of the ticket.
- purchaseDate: date - stores the date and time when the ticket was purchased.

Operations:

- bookTicket(movieID: int, theaterID: int, userID: int, seatNumber: string, price: float): void
  - Description: Books a ticket for the specified movie, theater, and user with the provided seat number and price.
- cancelTicket(ticketID: int): void
  - Description: Cancels the ticket with the specified ticketID.
- viewTicketDetails(ticketID: int): Ticket
  - Description: Retrieves and returns the details of the ticket with the specified ticketID.

Development Plan and Timeline:

Week 1-3: Project design preparation
Week 4-6: User Interface and database design
Week 5-6: User and Login System development
Week 7: Hardening: bug fixes
Week 8-9: Payment feature development
Week 10: Hardening: bug fixes
Week 11-12: Data integration
Week 13-14: Security Implementation and Testing
Week 15: Hardening: bug fixes
Week 16-17: Product testing
Week 18: Hardening: bug fixes
Week 19: Final push for development and release

Team Member Responsibilities:

[Alex]: Database design and management, data integration
[Shaun]: UI design, user registration and login system, security and final testing.
[Luiza]: management of transactions

Verification Test Plan:

Test Case 1: Verify User Sign Up

- Explanation: This test ensures that the user sign-up process functions correctly. It covers the feature of creating a new user account in the system.
- Test Sets/Vectors:
  - Pre-requisites include having access to the sign-up page.
  - Test steps involve launching the browser, navigating to the URL, accessing the sign-up tool, and entering required personal credentials.
- Targeted Feature(s): User account creation functionality.
- Coverage: This test ensures that users can successfully create an account and verifies the proper rendering and functionality of the sign-up page.

Test Case 2: Verify User Sign In

- Explanation: This test validates that users can successfully sign in to their accounts. It tests the authentication mechanism of the system.
- Test Sets/Vectors:
    - Pre-requisites include access to the sign-in page.
    - Test steps involve navigating to the sign-in page, entering username and password, and initiating the login process.
- Targeted Feature(s): User authentication and login functionality.
- Coverage: This test ensures that the login process works smoothly and users can access their accounts securely.

Test Case 3: Verify Sign Out

- Explanation: This test ensures that users can properly sign out of their accounts. It verifies the functionality related to user session management.
- Test Sets/Vectors:
    - Pre-requisites include being logged into the system.
    - Test steps involve clicking on the sign-out button on the appropriate page/tab.
- Targeted Feature(s): User session management and logout functionality.
- Coverage: This test confirms that users can securely log out of their accounts, preventing unauthorized access to their accounts.

Test Case 4: Verify Payment Information

- Explanation: This test ensures that users can add and save their payment information for future purchases. It covers the functionality related to managing payment details.
- Test Sets/Vectors:
    - Pre-requisites include being logged into the payment page.
    - Test steps involve adding card information or payment preferences and saving them in the user's account.
- Targeted Feature(s): Payment information management.
- Coverage: This test verifies that users can securely store their payment details, enabling smooth transactions in the future.

Test Case 5: Browse Available Movies

- Explanation: This test verifies that users can browse the available movies and read their descriptions. It assesses the functionality related to movie discovery and exploration.
- Test Sets/Vectors:
    - Pre-requisites include being logged into the system and accessing the movies available page.
    - Test steps involve browsing through the movie options and reading their descriptions.

- Targeted Feature(s): Movie browsing and description display.
- Coverage: This test ensures that users can easily explore the available movie options and make informed decisions.

Test Case 6: Book Ticket

- Explanation: This test validates the process of selecting a movie and tickets for purchase. It covers the functionality of selecting movie showtimes and ticket types.
- Test Sets/Vectors:
    - Pre-requisites include being logged in and accessing the movies available page.
    - Test steps involve software displaying available movies and time slots, and the user choosing tickets for the desired movie.
- Targeted Feature(s): Movie ticket selection.
- Coverage: This test ensures that users can successfully select movie tickets for purchase, facilitating the booking process.

Test Case 7: Verify Ticket Availability

- Explanation: This test ensures that users can select the number of tickets they want to purchase. It assesses the functionality of checking ticket availability and making purchases.
- Test Sets/Vectors:
    - Pre-requisites include selecting a movie and being logged in.
    - Test steps involve software displaying the number of tickets available, and the user selecting the desired quantity for purchase.
- Targeted Feature(s): Ticket availability and selection.
- Coverage: This test verifies that users can conveniently select the desired number of tickets for their chosen movie, ensuring a smooth purchasing experience.

Test Case 8: Verify Ticket Payment

- Explanation: This test ensures that users can successfully complete the payment process for their selected tickets. It covers the functionality related to secure payment processing.
- Test Sets/Vectors:
    - Pre-requisites include being on the checkout page and logged in.
    - Test steps involve software displaying checkout information and the user proceeding to the payment page.
- Targeted Feature(s): Payment processing.
- Coverage: This test confirms that users can securely complete the payment for their selected tickets, facilitating seamless transactions.

Test Case 9: Verify Confirmation of Purchase

- Explanation: This test ensures that users receive confirmation of their ticket purchase. It covers the functionality of providing users with transaction confirmation.
- Test Sets/Vectors:
    - Pre-requisites include being logged into the user account.
    - Test steps involve software generating a confirmation message, which the user receives in their account.
- Targeted Feature(s): Purchase confirmation.
- Coverage: This test verifies that users receive timely confirmation of their ticket purchase, providing assurance and transparency in the transaction process.

Test Case 10: Verify User Watches Selected Movie

- Explanation: This test ensures that users can access and watch the movie they selected after purchase. It covers the functionality of delivering purchased content to users.
- Test Sets/Vectors:
    - Pre-requisites include being logged in to the user account.
    - Test steps involve the software making the movie available at the selected time, and the user accessing the movie page to watch it.
- Targeted Feature(s): Movie access and playback.
- Coverage: This test confirms that users can seamlessly access and enjoy the movie they purchased, enhancing their overall user experience.