Theater Ticketing Software

Software Requirements Specification

Version 3

June 8, 2024

Group 8
Diego Vasquez-Del-Mercado, Rami Azouz

Github Link: https://github.com/RamiAzouz/Theater-ticketing-system.git

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

Revision History

Date	Description	Author	Comments
5/27/2024	Version 1	Diego Vasquez-Del-Mercado	Introduction & Requirements
5/27/2024	Version 1	Rami Azouz	Additional Requirements

Document Approval

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

Signature	Printed Name	Title	Date
	Diego Vasquez-Del-Mercado & Rami Azouz	Software Eng.	5/27/2024
	Dr. Gus Hanna	Instructor, CS 250	5/27/2024

Table of Contents

Revision History ii

Document Approval ii

1. Introduction 1

1.1 Purpose 1
1.2 Scope 1
1.3 Definitions, Acronyms, and Abbreviations 1
1.4 References 1
1.5 Overview 1

2. General Description 2

2.1 Product Perspective 2
2.2 Product Functions 2
2.3 User Characteristics 2
2.4 General Constraints 2
2.5 Assumptions and Dependencies 2

3. Specific Requirements 2

3.1 External Interface Requirements 3
3.1.1 User Interfaces *3*
3.1.2 Hardware Interfaces *3*
3.1.3 Software Interfaces *3*
3.1.4 Communications Interfaces *3*
3.2 Functional Requirements 3
3.2.1 <Functional Requirement or Feature #1> *3*
3.2.2 <Functional Requirement or Feature #2> *3*
3.3 Use Cases 3
3.3.1 Use Case #1 ..... *3*
3.3.2 Use Case #2 ..... *3*
3.4 Classes / Objects 3
3.4.1 <Class / Object #1> *3*
3.4.2 <Class / Object #2> *3*
3.5 Non-Functional Requirements 4
3.5.1 Performance *4*
3.5.2 Reliability *4*
3.5.3 Availability *4*
3.5.4 Security *4*
3.5.5 Maintainability *4*
3.5.6 Portability *4*

4. Software Design Specification 12

5. Software Design Specification Test Plan 21

6. Data Management Strategy 27

1. Introduction

The purpose of this Software Requirement Specification document is to detail the requirements for the Theater Ticketing Software. This document serves as a guide for software engineers to design the software product. It outlines the functionality, constraints, and specific needs of the system.

1.1 Purpose

The purpose of this SRS is to define the functional and non-functional requirements of the Theater Ticketing Software. This document is written for the software engineers, and other stakeholders involved in the development of the software. This document provides a description of the system's features and constraints.

1.2 Scope

(1) The software product to be developed is named "Theater Ticketing System."

(2) The Theater Ticketing System will manage the purchasing of movie tickets for theaters. It will support both online and in-person ticket sales and give real-time data updates. The system will manage ticket availability, pricing and include discounts for students, military, and seniors.

(3) The Theater Ticketing System aims to improve the efficiency and user experience of purchasing movie tickets, both online and in-person.

(a) Some benefits include efficient ticket purchasing process, real-time updates on ticket availability, easy to use interface and quick access to ticket purchasing, Increased customer satisfaction with feedback collection and review display. Some goals are to be able to handle at least 1000 users, ensure secure transactions, prevent bot purchases, and collect feedback for improvement.

1.3 Definitions, Acronyms, and Abbreviations

AMC: American Multi-Cinema, a chain of movie theaters.
SRS: Software Requirements Specification
UI: User Interface
UX: User Experience

1.4 References

(1) As there are no external documents referenced or sourced in this SRS, all information is contained within this document.

1.5 Overview

This subsection should:

(1) The rest of the SRS document includes descriptions of the functions of the Theater Ticketing System and its specific requirements.

(2) The SRS is organized into three parts currently: the introduction, a general description, & the specific requirements.

2. General Description

This section provides a general overview of the Theater Ticketing System. It includes information that helps understanding the product without detailing specific requirements and functionalities.

2.1 Product Perspective

The Theater Ticketing System is a web-based software designed to manage the purchase of movie tickets for theaters. It information on showtimes, ticket availability, pricing, and user data.

2.2 Product Functions

The Theater Ticketing System will perform several functions to manage ticket sales: like, purchasing of movie tickets both online and in-person, maximum of 20 tickets per transaction, user authentication like creating accounts and logging in, and feedback collection.

2.3 User Characteristics

General Customers: Will browse movie showtimes and purchase tickets and require an easy to use interface.
Students, Military Personnel, and Seniors: are eligible for discount pricing.
AMC Members: Use membership cards for ticket reservations.

2.4 General Constraints

Several constraints may include: budget, deadline to finish the project, security, and maintenance of the software.

2.5 Assumptions and Dependencies

Internet Connectivity: Assumes reliable internet for online users.
Database: Requires a database to store all data.
User Feedback: Assumes users will provide feedback that will be used to improve the program.
Security Measures: Assumes the adequate security to prevent unauthorized transactions.

3. Specific Requirements

This section outlines the specific requirements for the Theater Ticketing System.

3.1 External Interface Requirements

3.1.1 User Interfaces

Interaction between the user and the system.
Browser: The system will have a web-based interface accessible through browsers like Chrome and Safari.
Ticket Purchase Interface: Users will be able to browse available movies and purchase tickets. Will provide a summary page of their purchase. Also have a feedback section for users to rate their experience.
Feedback Interface: After each purchase, users will be able to provide feedback on their experience.
AMC Membership: Users can log in with their AMC login. Will also provide options to purchase or renew AMC memberships.

3.1.2 Hardware Interfaces

Interaction between the system and the hardware.
Ticket Printers: will support for real-time printing of tickets upon purchase completion.
Scanners: to support for scanning barcodes used to validate tickets at theater entrances.

3.1.3 Software Interfaces

Interaction between the Theater Ticketing System and other software systems.
Central Database: access to the central database for updating and retrieving ticket and user information.
Payment Portal: secure payment programs to handle payment transactions. Also, support for credit/debit cards and movie gift cards.

3.1.4 Communications Interfaces

Interaction between the system and external communication networks.
Internet Connectivity: Reliable internet and secure data transmission.

3.2 Functional Requirements

This section describes specific features of the Theater Ticketing System.

3.2.1 <Functional Requirement or Feature #1>

3.2.1.1 Introduction:
The ticket purchase functionality allows users to buy movie tickets online and in person, also supports selling regular and premium tickets.
3.2.1.2 Inputs:

User selection of movie, time, and theater. Number and type of tickets (regular or premium). Credit card/payment information, and discount options (student, military, senior).
3.2.1.3 Processing:
Check ticket/time availability, apply discounts, process payment through secure software, and update the movie theater database.
3.2.1.4 Outputs:
Confirmation of ticket purchase(receipt), printable tickets for in person purchases. Digital tickets will be sent to user's email. Update the new seat availability.
3.2.1.5 Error Handling:
Display error messages for errors like invalid payment information, or unavailable tickets, retry options for a failed transactions.

3.3 Use Cases

3.3.1 Use Case #1: Ticket Purchase

3.3.2 Use Case #2: User Authentication

3.4 Classes / Objects

3.4.1 <Class / Object #1>

3.4.1.1 Attributes: Username, Password, Email, MembershipStatus
3.4.1.2 Functions: Register(), Login(), Logout(), UpdateProfile()

3.5 Non-Functional Requirements

These requirements make sure that the system meets the standards of performance, availability, security, maintainability, and portability.

3.5.1 Performance:

Response Time: ticket purchases shall be processed within 2 seconds.
Scalability: the system must support at least 1,000 simultaneous users.
Load Time: the main user interface should load within 3 seconds.

3.5.2 Reliability:

- The system shall have an error rate of less than 0.01% for all transactions.
- All data transactions(ticket sales, user data) must be accurate.
- The system will implement daily backups in case of failure.

3.5.3 Availability:

- The system must be operational 24/7, with scheduled maintenance times.
- The system shall be accessible from all standard web browsers.

3.5.4 Security:

- All sensitive data, including user info and payment information, must be encrypted.

- The system must have secure user authentication like password or two-factor authentication.
- The system must have regular security checks and vulnerability tests.

3.5.5 Maintainability:

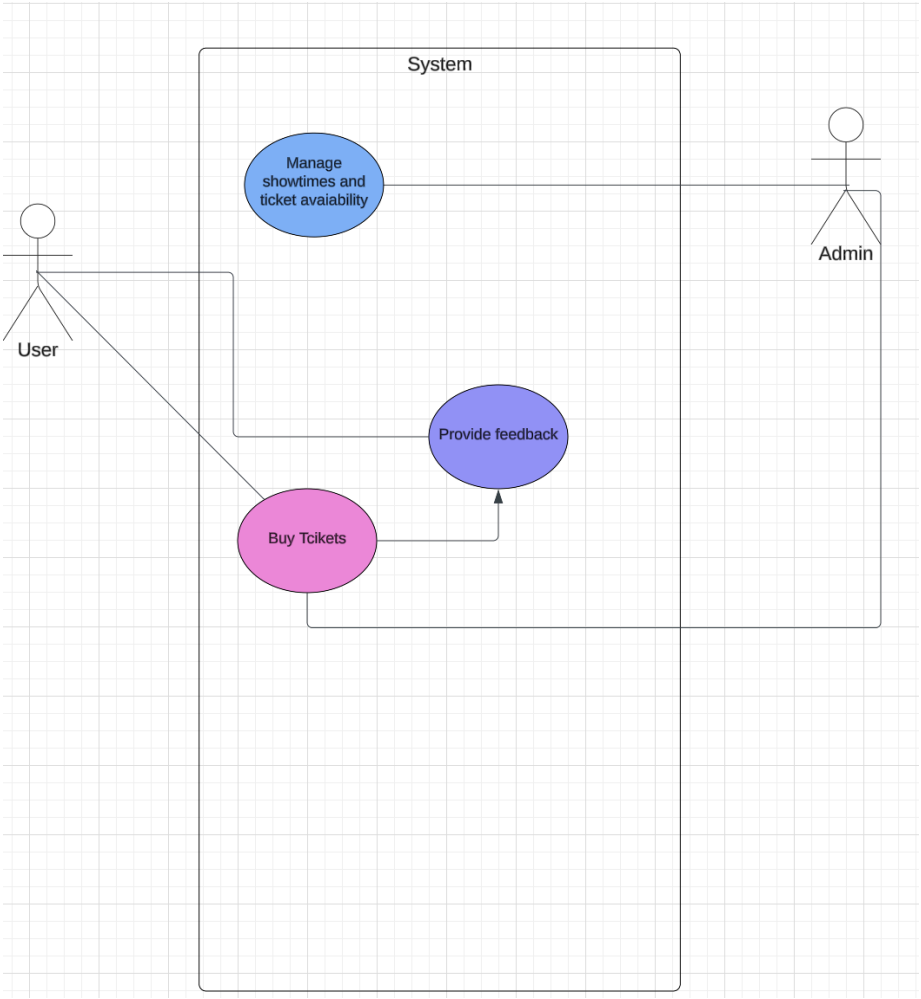
- Code Quality: must adhere to industry best practices.

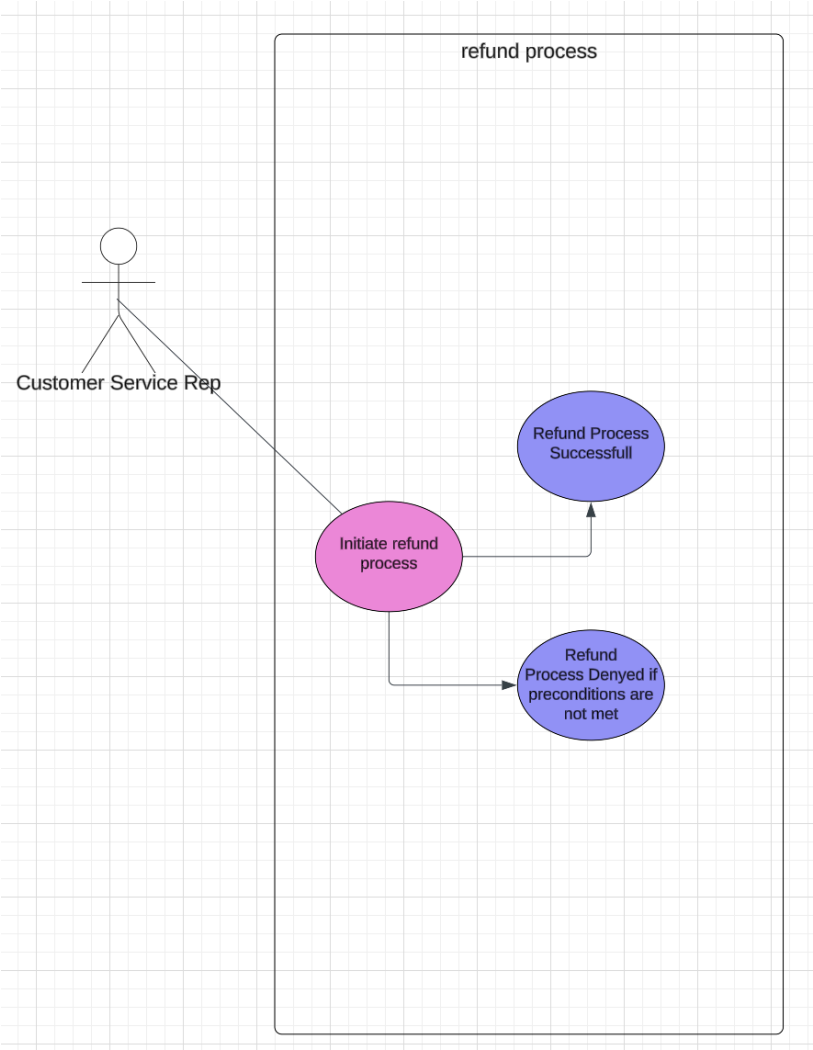
- Updates: must support a smooth update process with minimum downtime to the server.

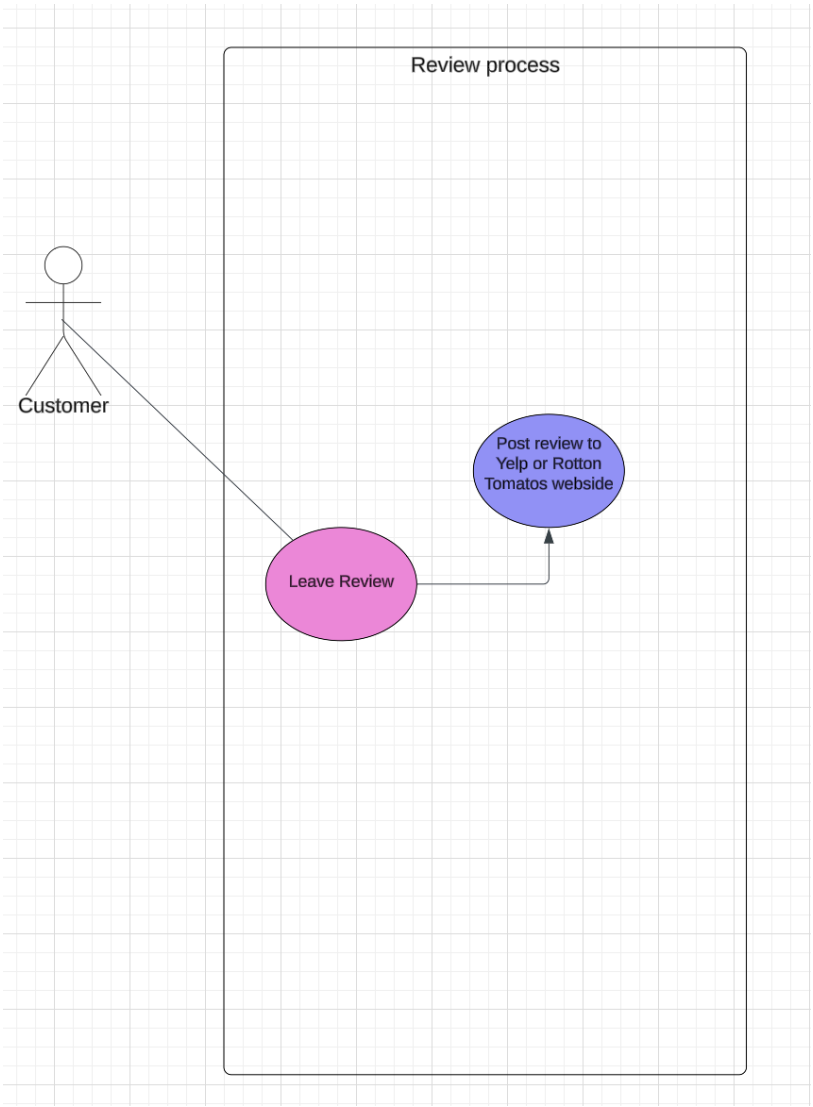
3.5.6 Portability:

- Should be able to run on any web browser like Chrome, Firefox, or Safari.

- When migrating data, it should be easy and secure to transfer data from the existing system to the new system.


System
Manage showtimes and ticket avaiability
Admin
User
Provide feedback
Buy Tcikets


refund process
Customer Service Rep
Refund Process Successfull
Initiate refund process
Refund Process Denyed if preconditions are not met


Review process
Customer
Post review to Yelp or Rotton Tomatos webside
Leave Review

Software Design Specification

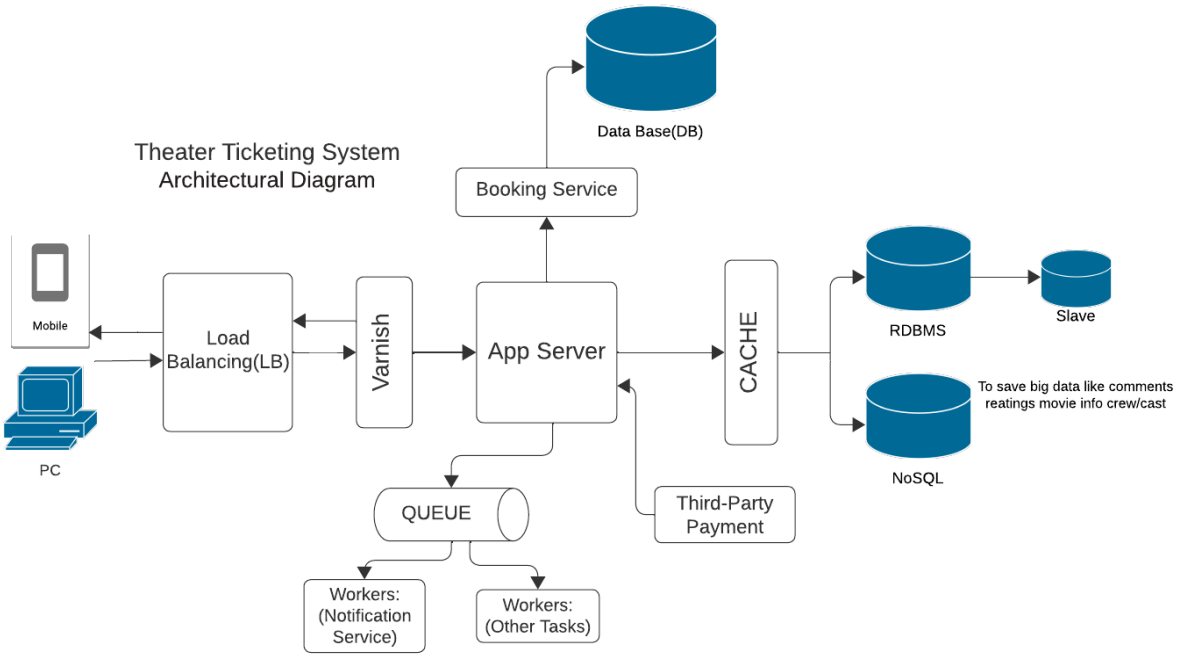
Team Members: Diego Vasquez-Del-Mercado, and Rami Azouz
June, 2, 2024

Brief Overview of the System:

The Theater Ticketing System is an online solution aimed at streamlining the ticket purchasing process for theater performances. It offers users the ability to explore different shows, select their preferred seats, complete ticket purchases, and handle their user profiles efficiently. This document should contain all the necessary information to complete a theater ticketing system application.

Github Link: https://github.com/RamiAzouz/Theater-ticketing-system.git

Architectural Diagram:



Software Architecture Overview:

The architectural diagram for the Theater Ticketing System shows the various components to handle that occurs when the user buys a ticket,also affecting caching, load balancing, and database interactions. The diagram is segmented into the following main components:

1. Users (Mobile and PC): These are the clients that will interact with the system. Users will order tickets by using a mobile device, a computer, or even accessing through a browser on a mobile device.
2. Load Balancing (LB): Distributes incoming requests to make sure that the servers are not overwhelmed, to ensure efficient and continuous reliability and to minimize downtime.
3. Varnish: A caching layer that helps to speed up the request processing by storing copies of frequently accessed data.

4. App Server: This is the main application, it is a server that processes requests, manages business logic, handles interactions with other services, allows for online payment or links to a third party payment service, and allows users to leave reviews that could also link to third-party review websites like Yelp or Rotten Tomatoes, and sends out conformation receipts. All coding should be compatible to handle all of these requirements and should maintain fast loading speeds.
5. Booking Service: Manages booking-related operations and interactions with the database. Should be able to access the database to check for available tickets, check the database for which seats are taken and not taken, and either allow the user to buy tickets directly online or through a third party payment process.
6. Cache: Stores frequently accessed data to reduce database load and improve performance.This is an important component of the application, because it keeps fast loading times of the system and ensures minimum downtime and should be implemented into the code with care.
7. Third-Party Payment: Handles payment processing by integrating with external payment services. This can either be directly coded into the application or be substituted with redirecting the user to a third party payment gateway(preferred, because if you have to code the payment process yourself, then you are also are responsible with keeping a secure database with the credit card details of the users, which will require programming more security protections, which will increase labor costs.)
8. Queue: Manages background tasks and notifications. This will queue up the task needed to be done in a first come first serve basis to be fair for users that process their tasks first.
9. Workers: Processes queued tasks such as sending notifications and other asynchronous operations. This refers to the "workers" portion of the architectural diagram, which will work on sending out the notifications for the users upcoming movie dates and times, and sending out email confirmation receipts to users after their payment process is complete. This step is crucial because it gives the application better reliability, by providing users with confirmations and good communication between the program and customer.
10. Database (DB): The primary data store for the application, consisting of an RDBMS and NoSQL databases. This stores all of the necessary data for the theater ticketing system, like available tickets, available seats, ticket prices, user login information, etc…
11. Slave: A replica of the primary RDBMS for read-heavy operations.

Description of Components

1. Users (Mobile and PC)
    - Description: User interfaces for accessing the theater ticketing system.
    - Attributes: Device type, IP address.
    - Operations: Send request, receive response.
2. Load Balancing (LB)

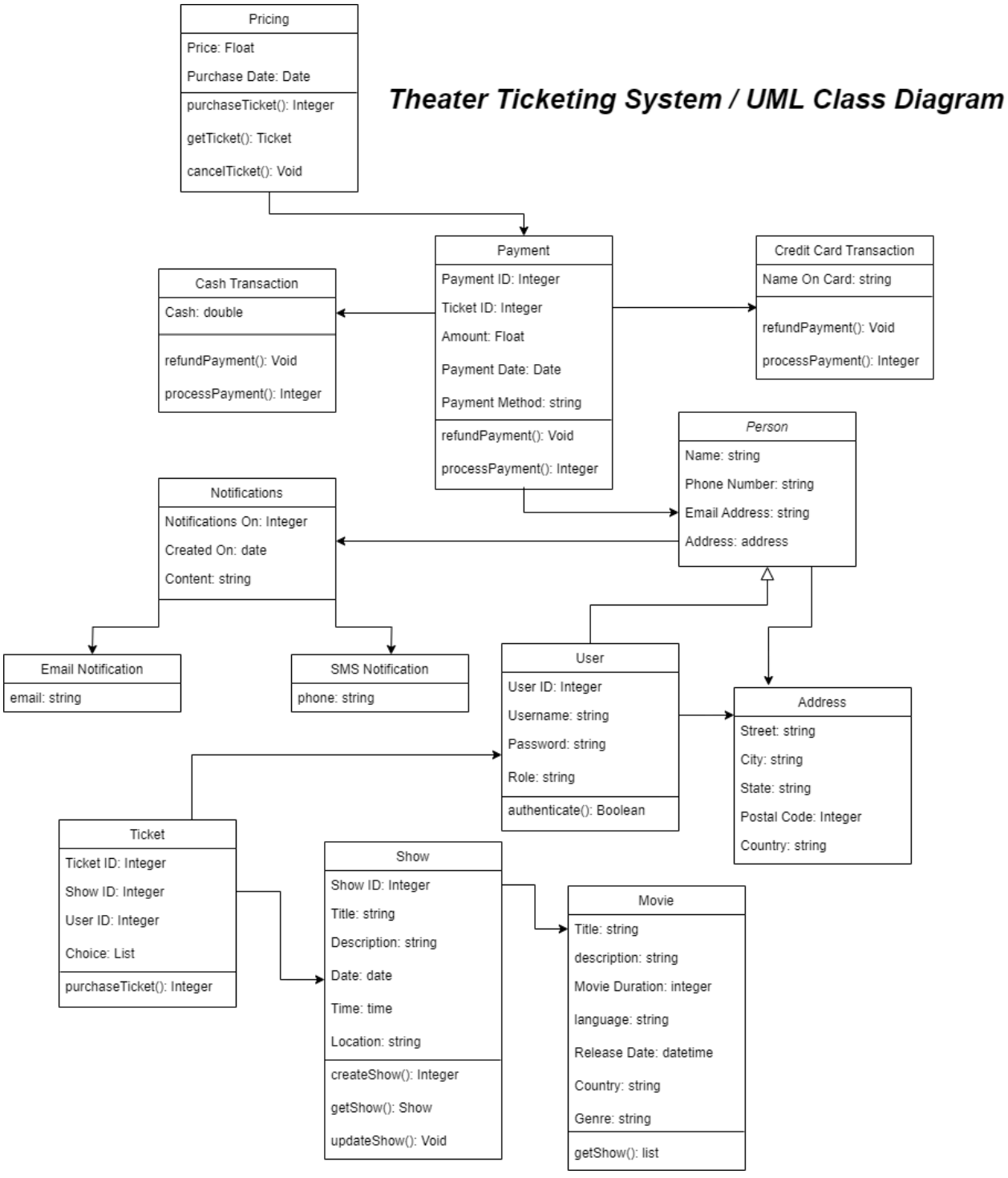
- Description: Distributes incoming traffic to multiple app servers to ensure even load distribution.
- Attributes: Load balancer ID, algorithm type.
- Operations: Distribute request, monitor server health.

3. Varnish
    - Description: A caching layer that stores copies of frequently accessed resources.
    - Attributes: Cache ID, size, TTL (Time To Live).
    - Operations: Store cache, retrieve cache, invalidate cache.
4. App Server
    - Description: The main server that handles business logic and processes requests.
    - Attributes: Server ID, CPU usage, memory usage.
    - Operations: Process request, interact with database, call external services.
5. Booking Service
    - Description: Manages ticket booking operations.
    - Attributes: Booking ID, user ID, show ID, seat number.
    - Operations: Create booking, update booking, cancel booking, retrieve booking details.
6. Cache
    - Description: Stores frequently accessed data to reduce load on the primary database.
    - Attributes: Cache key, value, expiration time.
    - Operations: Store data, retrieve data, clear cache.
7. Third-Party Payment
    - Description: Handles payment processing through external payment services.
    - Attributes: Payment ID, amount, payment method.
    - Operations: Process payment, verify payment status, refund payment.
8. Queue
    - Description: Manages background tasks and ensures they are processed asynchronously.
    - Attributes: Queue ID, task ID, priority.
    - Operations: Enqueue task, dequeue task, monitor queue.
9. Workers
    - Description: Processes tasks in the queue such as sending notifications and other background operations.
    - Attributes: Worker ID, task type, status.
    - Operations: Execute task, update task status, retry task.
10. Database (DB)
    - Description: The primary data store, consisting of both RDBMS and NoSQL databases.
    - Attributes: Database ID, type, size, connection string.

- Operations: Insert data, update data, delete data, query data.

11. Slave
    - Description: A read replica of the primary RDBMS to handle read-heavy operations.
    - Attributes: Replica ID, lag time, read-only status.
    - Operations: Synchronize with primary, handle read queries.

UML Class Diagram



Description of UML Class Diagram for Theater Ticketing System

Software Requirements Specification

The UML Class Diagram for the Theater Ticketing System illustrates the system's structure, showcasing various classes, their attributes, operations, and the relationships among them. Below is a summarized description of the key components:

Class: Person

Attributes:

- Name: String - Captures the person's name, is string to capture the users name in string format..
- Phone Number: String - Stores the person's phone number for contact purposes.
- Email Address: String - Stores the person's email address for communication.
- Address: Address - Links to the Address class to store detailed address information.

Class: User

Attributes:

- User ID: Integer - A unique identifier for each user, ensuring distinct user records.
- Username: String - The users login name, essential for authentication.
- Password: String - The users password, Stored securely for authentication.
- Role: String- Defines the users role, for example the user could be a customer, an employee, etc… Its important for access control or permission.

Operations:

- Authenticate(): Boolean - Verifies user credentials to allow or deny access.

Class: Ticket

Attributes:

- Ticket ID: Integer - A unique identifier for each ticket, ensuring distinct records.
- Show ID: Integer - Links the ticket to a specific show.
- User ID: Integer - Links the ticket to the user who purchased it.
- Choice: List - Stores seat choices or preferences.

Operations:

- purchaseTicket(): Integer - Facilitates the ticket purchasing process.

Class: Show

Attributes:

- Show ID: Integer - A unique identifier for each show, ensuring distinct records.
- Title: String - The title of the show, important for identification.
- Description: String - A brief overview of the show.
- Date: Date - The date of the performance.
- Time: Time - The time of the performance.
- Location: String - The venue where the show is held.

Operations:

- createShow(): Integer - Facilitates the creation of a new show.
- getShow(): Show - Retrieves details of a specific show.
- updateShow(): Void - Updates the details of an existing show.

Class: Movie

Attributes:

- Title: String - The title of the movie.
- Description: String - A brief overview of the movie.
- Movie Duration: Integer - The length of the movie.
- Language: String - The language of the movie.
- Release Date: DateTime - The release date of the movie.
- Country: String - The country of origin.
- Genre: String - The genre of the movie.

Operations:

- getShow(): List - Retrieves a list of shows for the movie.

Overall, these attributes and operations are designed to encapsulate the essential data and functionalities needed to manage the theater ticketing process efficiently, ensuring that all relevant aspects are covered and interactions are handled seamlessly.

Development plan and timeline:

Diego was responsible for creating the architectural diagram of all major components.
Rami was responsible for developing the UML Class Diagram, including the description of classes, attributes, and operations.

1. UML Class Diagram: Will be created by Rami, estimated timeline will be 2 hours.
2. UML Description of classes, attributes and operations: Will be done by Rami, estimated timeline will be 3 hours.
3. ARCH Diagram: Will be Created by Diego, estimated timeline will be 2 hours.
4. ARCH Description: Will be done by Diego, estimated timeline will be 2 hours.
5. Document overview and Description: Will be done by both team members Rami & Diego, estimated timeline will be 2 hours.

5. Software Design Specification Test Plan

Software Design Specification Test Plan

Team Members: Diego Vasquez-Del-Mercado, and Rami Azouz
June, 8, 2024

Github Link: https://github.com/RamiAzouz/Theater-ticketing-system.git

The 10 test case samples are attached in an excel document.

5.1 Introduction

This test plan for the Theater Ticketing System is designed to ensure that the software works correctly, performs well, and is reliable. It covers detailed descriptions of the parts of the system being tested, the strategies we will use, and the specific tests we will run. Our goal is to thoroughly check all the important features of the system at the unit, functional, and system levels.

Test Plan Overview

The testing process is divided into three main levels:

- **Unit Testing:** Focuses on individual components or modules to ensure they function correctly in isolation.
- **Functional Testing:** Validates the software against the functional requirements to ensure it behaves as expected.
- **System Testing:** Evaluates the system's compliance with specified requirements in an integrated environment.

Each test plan includes detailed descriptions of the target features, specific test cases, and expected outcomes.

5.3 Test Plans and Test Cases

5.3.1 Unit Test Plan

Test Plan 1: User Registration

- **Target Component:** User_Registration_Module
- **Description:** Verify that a new user can successfully register with valid details.
- **Test Steps:**
    1. Navigate to the registration page.
    2. Enter valid username, password, email, and other required details.
    3. Click the "Register" button.
- **Expected Outcome:** User should be registered successfully and redirected to the login page.
- **Test Case:** UserRegistration_1

Test Plan 2: User Login

- **Target Component:** User_Authentication_Module
- **Description:** Verify that a registered user can log in with valid credentials.
- **Test Steps:**
    1. Navigate to the login page.
    2. Enter valid username and password.
    3. Click the "Login" button.
- **Expected Outcome:** User should be logged in successfully and redirected to the homepage.
- **Test Case:** UserLogin_1

5.3.2 Functional Test Plan

Test Plan 3: View Movie List

- **Target Component:** Movie_List_Module
- **Description:** Verify that the user can view the list of available movies.
- **Test Steps:**
    1. Log in to the system.
    2. Navigate to the "Movies" section.
- **Expected Outcome:** The list of available movies should be displayed for the user.
- **Test Case:** ViewMovieList_1

Test Plan 4: Search for a Movie

- **Target Component:** Search_Module
- **Description:** Verify that the user can search for a movie using the search bar.
- **Test Steps:**
    1. Log in to the system.
    2. Enter the movie title in the search bar.
    3. Click the "Search" button.
- **Expected Outcome:** The search results should display the relevant movie.
- **Test Case:** SearchMovie_1

5.3.3 System Test Plan

Test Plan 5: Purchase Ticket

- **Target Component:** Ticket_Purchase_Module
- **Description:** Verify that a user can purchase a ticket successfully.
- **Test Steps:**

1. Log in to the system.
2. Select a movie and showtime.
3. Choose seats and proceed to payment.
4. Enter payment details and confirm the purchase.

- **Expected Outcome:** Tickets should be purchased successfully, and a confirmation should be sent to the user's email address.
- **Test Case:** PurchaseTicket_1

Test Plan 6: Apply Discount

- **Target Component:** Discount_Module
- **Description:** Verify that a discount is applied correctly for eligible users (e.g., students).
- **Test Steps:**
    1. Log in to the system.
    2. Select a movie and showtime.
    3. Choose seats and proceed to payment.
    4. Enter payment details and apply the discount code.
- **Expected Outcome:** Discount should be applied to the total amount, and the user should see the discounted price.
- **Test Case:** ApplyDiscount_1

Test Plan 7: Print Ticket

- **Target Component:** Ticket_Print_Module
- **Description:** Verify that the system can print a ticket successfully.
- **Test Steps:**
    1. Navigate to the "My Tickets" section.
    2. Select the ticket to print.
    3. Click the "Print Ticket" button.
- **Expected Outcome:** Ticket should be printed successfully.
- **Test Case:** PrintTicket_1

Test Plan 8: Send Email Notification

- **Target Component:** Email_Notification_Module
- **Description:** Verify that an email notification is sent to the user after purchasing a ticket.
- **Test Steps:**
    1. Purchase a ticket.
    2. Check the user's email inbox.
- **Expected Outcome:** Email notification should be received in the user's inbox.

- **Test Case:** SendEmailNotification_1

Test Plan 9: View Booking History

- **Target Component:** Booking_History_Module
- **Description:** Verify that a user can view their booking history.
- **Test Steps:**
    1. Log in to the system.
    2. Navigate to the "Booking History" section.
- **Expected Outcome:** The user's booking history should be displayed.
- **Test Case:** ViewBookingHistory_1

Test Plan 10: Cancel Ticket

- **Target Component:** Ticket_Cancellation_Module
- **Description:** Verify that a user can cancel a purchased ticket.
- **Test Steps:**
    1. Log in to the system.
    2. Navigate to the "My Tickets" section.
    3. Select the ticket to cancel.
    4. Click the "Cancel Ticket" button.
- **Expected Outcome:** Ticket should be canceled successfully, and a refund should be processed.
- **Test Case:** CancelTicket_1

5.4 Test Strategy

The test strategy involves using various methodologies to ensure comprehensive coverage of the system's functionalities:

- **Black Box Testing:** Focuses on testing the functionality without knowledge of the internal code structure.
- **White Box Testing:** Involves testing the internal structures or workings of the application.
- **Integration Testing:** Ensures that different components of the system work together as expected.
- **Regression Testing:** Verifies that new changes have not adversely affected existing functionalities.

5.5 Test Execution and Reporting

The QA team will run the tests and record the results in the provided Excel template. Any bugs or issues found will be logged, tracked, and fixed before the system is deployed.

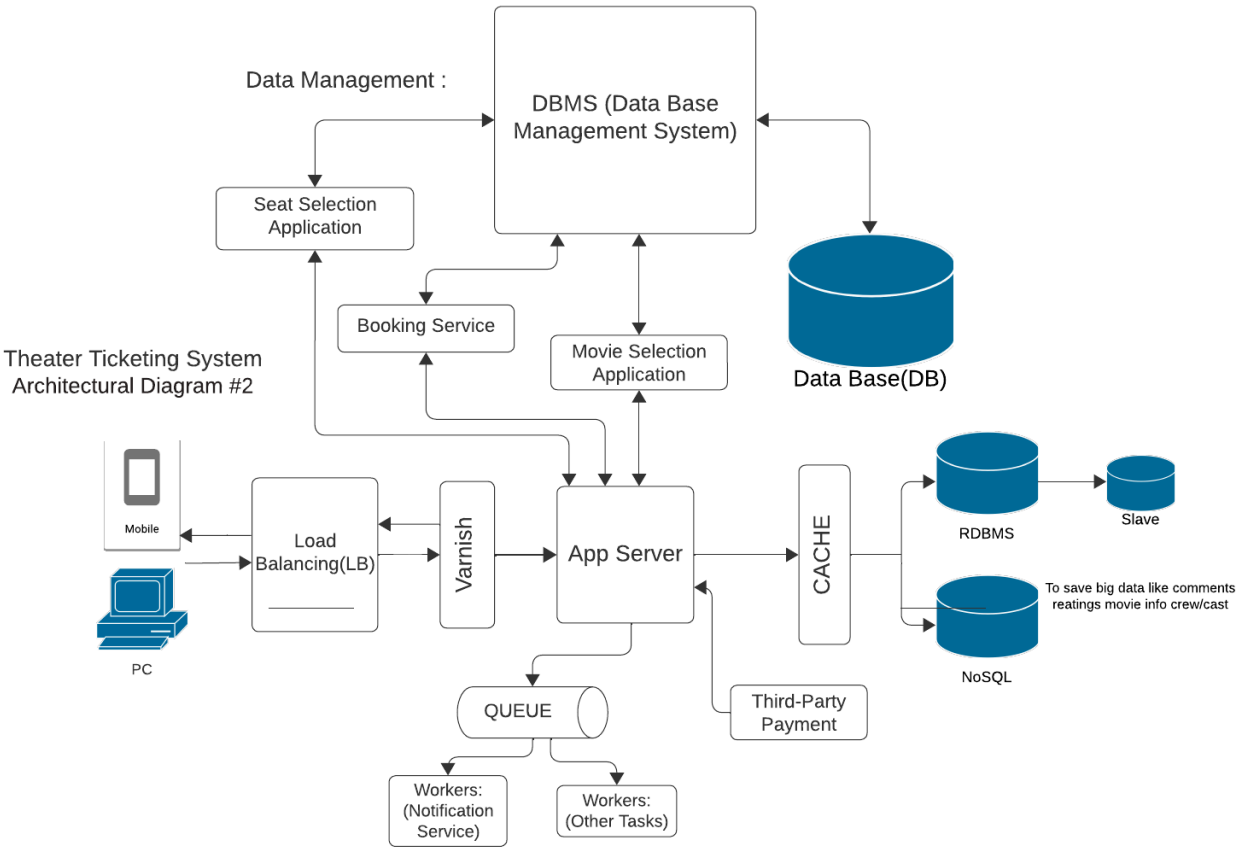
Chapter 6: Architecture Design w/Data Mgmt.

Data Management Strategy for Movie Ticketing System

Team Members: Diego Vasquez-Del-Mercado, and Rami Azouz
June, 17, 2024

Github Link: https://github.com/RamiAzouz/Theater-ticketing-system.git

Updated Architecture Diagram: Our updated design diagram, labeled "Theater Ticketing System Architecture Diagram #2" is provided below. The changes were made in order to improve system performance, scalability, and reliability. The main component we added is the DBMS (Database Management System) which is in charge of managing incoming data, and outputting data in a secure, efficient, and reliable way. We also included some of the applications the software uses in the diagram including: Movie Selection Application, Booking Service, Seat Selection Application.



1. Data Model Design

- **Entities:** The parts of the system that our movie ticketing software require are listed here: Movies listing, Theaters locations, Screens available, Showtimes, Ticket costs, User logins, Payments, Seat location & availability, and Reviews.

- **Relationships:** the theater ticketing system should be able to connect to all of the other systems at different movie theaters, providing ticket availability and showtimes across the platform.

2. Database Selection

- **Relational Database:** Our system includes SQL databases specifically a DBMS and a RBMS. PostgreSQL or MySQL are possible options to choose when implementing the program, because these are good for handling structured data and managing relationships effectively.

3. Data Storage and Retrieval

- **Normalization:** Organize the database to reduce redundancy and maintain data accuracy.
- **Partitioning:** Make sure to break down the large datasets into smaller components to keep speed up and make data more organized.

4. Data Security

- **Encryption:** Very important to keep sensitive information protected, like user data and payment details, by encrypting it both when it's stored and when it's being transmitted. Two options are available for this, either design the encryption code yourself to hire an outside provider to encrypt sensitive data.
- **Access Control:** Make a secure admin system so that only authorized users can access data and program management system.
- **Regular Audits:** Make sure to do regular security checks to test the vulnerability of the program, ensuring the security of the data.

5. Backup and Recovery

- **Regular Backups:** Take regular backups of the database to ensure that no data is lost incase of program failure. Store backups in a secure database.
- **Disaster Recovery Plan:** Create a detailed plan that aids in the process to recover lost data in case of errors, failure, or outside circumstances. Include a second backup storage in a secure location somewhere.

6. Scalability

- **Vertical Scaling:** Increase program speed and database performance by adding more CPU and RAM.

- **Load Balancing:** Use load balancers to distribute the workload of incoming queues.

7. Data Lifecycle Management

- **Archiving:** Move old, rarely accessed data to an archive to lighten the load on the main database.
- **Retention Policies:** Set up data retention rules to meet legal and business standards.

8. Data Consistency and Integrity

- **Transactions:** Maintain data consistency and integrity by using transactions that follow ACID principles.
- **Validation:** Implement data validation in the application and database to ensure accuracy.

9. Monitoring and Maintenance

- **Performance Monitoring:** Use tools like New Relic or Datadog to keep an eye on database performance and query speed.
- **Regular Maintenance:** Perform regular maintenance tasks like rebuilding indexes and vacuuming the database to keep it running smoothly.

10. Analytics and Reporting

- **Data Warehousing:** Set up a data warehouse to handle complex queries and generate reports.
- **ETL Processes:** Use ETL (Extract, Transform, Load) processes to gather and consolidate data from different sources into the data warehouse.
- **Reporting Tools:** Utilize tools like Tableau, Power BI, or custom dashboards for data analysis and reporting.

11. Compliance and Privacy

- **GDPR/CCPA Compliance:** Make sure your system follows data protection laws like GDPR and CCPA.
- **User Consent:** Get clear permission from users before collecting their data and give them options to delete it.

12. Documentation and Training

- **Documentation:** Keep detailed records of your data model, schema, and processes.
- **Training:** Provide training for database administrators and developers on best practices and security protocols.

13. Future-proofing

- **Technological Advancement:** Keep up with the latest database technologies and best practices.
- **Scalability Testing:** Regularly test the system to make sure it can handle more users as it grows.

1. Designing the Database for a Movie Ticketing System

1. Person

- **Attributes:** Name, Phone Number, Email Address, and Address

2. User

- **Attributes:** User ID, Username, and Password

3. Ticket

- **Attributes:** Ticket ID, Show ID, User ID, and Choice

4. Show

- **Attributes:** Show ID, Title, Description, Date, Time, and Location

5. Movie

- **Attributes:** Title, Description, Movie Duration, Language, Release Date, Country, and Genre

2. Define Relationships

- Person - Login: One-to-One (One person can make one account)
- User - Ticket: One-to-Many (One user can book multiple tickets)
- Ticket - Shows: One-to-One (One ticket can only have one show)
- Show - Seats: One-to-many (One show can have multiple seats)
- Movie - Ticket: One-to-many (One Movie can have multiple tickets)

3. Normalization and Indexing

- Make sure the database is at least in the third normal form (3NF) to avoid redundancy.
- Add indexes to fields that are often searched, like UserID, MovieID, TheaterID, and ShowTime.

4. Scalability Considerations

- Use database partitioning (horizontal sharding) for very large datasets.
- Implement caching solutions, like Redis, for frequently accessed data such as movie listings and showtimes.
- Think about using a read replica setup to help distribute read traffic across the database.

5. Justification of design decisions:

- We include Movies, Theaters, Screens, Showtimes, Tickets, Users, Payments, Seat Categories, and Reviews, ensuring interconnected data. We use MySQL for structured data and relationships, with normalization to reduce redundancy and partitioning to improve performance. Data is protected with encryption and role-based access, and we implement automated backups and disaster recovery plans. Performance is enhanced through scaling, load balancing, and regular monitoring with tools like New Relic, while compliance with GDPR/CCPA and user consent are ensured.