Software Requirements Specification Documentation

Theatre Ticketing System

9/21/2023

Created by:

Group #4

Moe Jawadi

Joel Alvarado

Ernesto Cornejo

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

Revision History

Date	Description	Author	Comments
<date>	<Version 1>	<Your Name>	<First Revision>
9/21/23	Requirement Specification	Moe Jawadi	
9/21/23	Requirement Specification	Joel Alvarado	
9/21/23	Requirement Specification	Ernesto Cornejo	

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 & Overview

The purpose of our website is to allow users to buy tickets to movies that are currently playing or upcoming. In order for the user to access our services they will be required to create an account. The user will be able to choose a time and day a specific movie plays, they will also be able to search through movies that are currently playing or upcoming movies. The user will also be able to filter the movies displayed on screen by choosing a specific genre, actor, showtime, day, and movie title. There will be no refunds however a user can receive credit in their account that doesn't expire which can be used to watch a different movie at any time. The user will be allowed to purchase and reserve up to 20 seats to a movie with our interactive seating chart. An optional newsletter with coupons will be sent out to its users twice a month, which can be disabled at any point.

2. User Requirements

- Access to the internet
- Computer or Laptop
- An Email
- An account on our website
- Debit Card on file or similar purchasing power that meet our requirements
- Optional(game console, or smart watch)

3. System Requirements

3.1 Functional Requirements

1. User Registration

- Users should be able to create an account and login with no pop ups or ads. - System will ask to save password for future logins upon account creation. - The System will provide many ways to login, (Login with Google), (Login with Apple). - Users will be asked to enable and add a (2FA) authentication method upon logging in to help prevent account safety.

2. Profile Management

- Users will be able to change account preferences.
- Users will be able to save their favorite movies.
- Users will be able to save payment methods.
- Users will be able to update payment methods.
- Users will be able to customize their (2FA) authentication preferences to fit wants and needs.

3. Movie Availability

- The front page will display numerous movies.
- The front page will include Showtimes, titles, and Genres.
- The system will have a filter option to help the browsing experience.

4. Analytics & Reviews

- Data of how many tickets were sold within the previous hour for a specific show that is browsed
- Data of what shows are trending, most popular, and best reviewed by our certified customers.
- Ratings of every movie will be shown on the bottom right of every movie icon or (picture)
- Customers will be able to see the positive/negative reviews on every show available by simply clicking on the show.

5. Payment Processing & Movie Selection

- Payment Options of Debit cards, Credit Cards, Apple Pay, CryptoCurrency (BTC & ETH), and Paypal.
- Users will have the ability to click on any movie, select any seat and time that is available upon searching.
- Users will be able to view an interactive seating chart to view the seats they are interested in.
- Users will be able to select up to (20) tickets of purchase at once upon each transaction to limit the amount of tickets that can be purchased by a single individual.

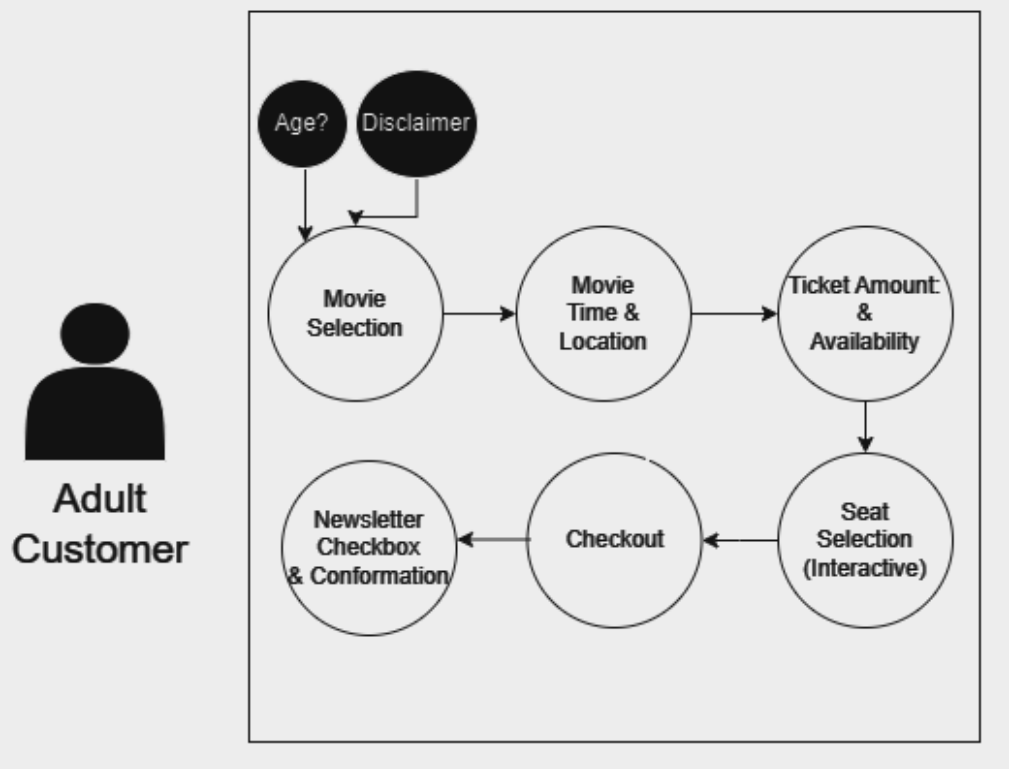
6. Receipts & Purchase History

- After every purchase, a confirmation page will appear as well as a confirmation email will be sent.
- Purchase history of all movies will be saved and found in the Profile Preferences under “Previous Purchases”.

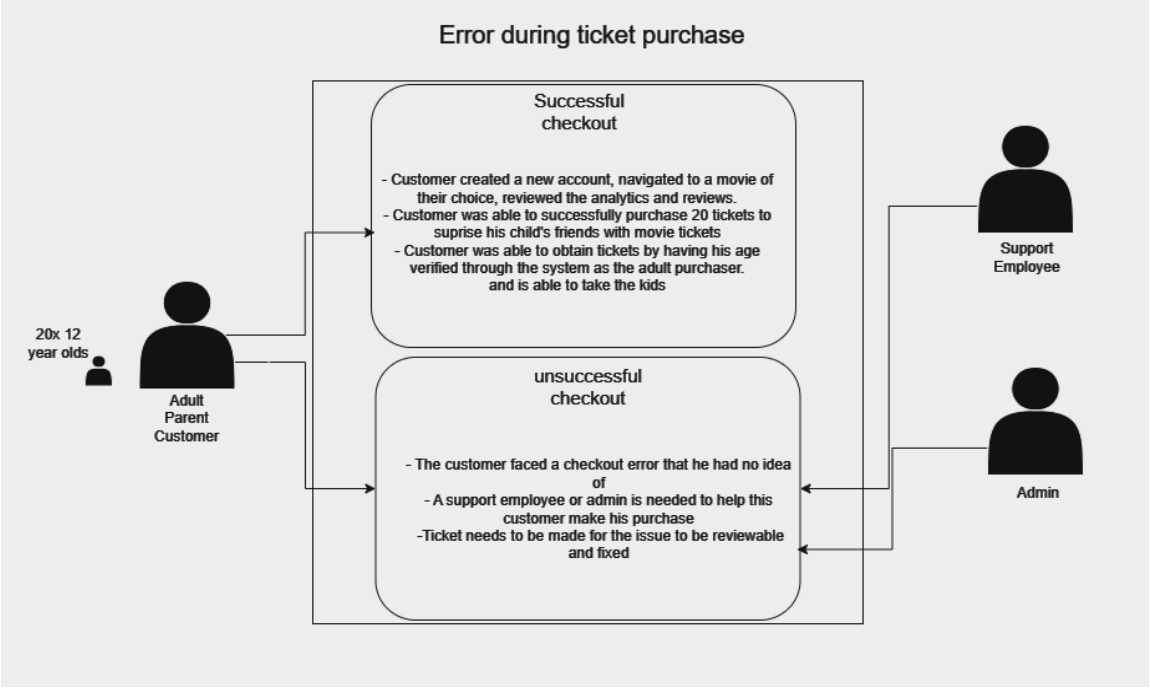
7. Email Newsletter

- Discounts and Coupons will be sent out commonly twice a month to loyal returning customers who agreed to the terms and conditions of the Email Newsletter upon purchase the newsletter can be disabled by the user at any point.

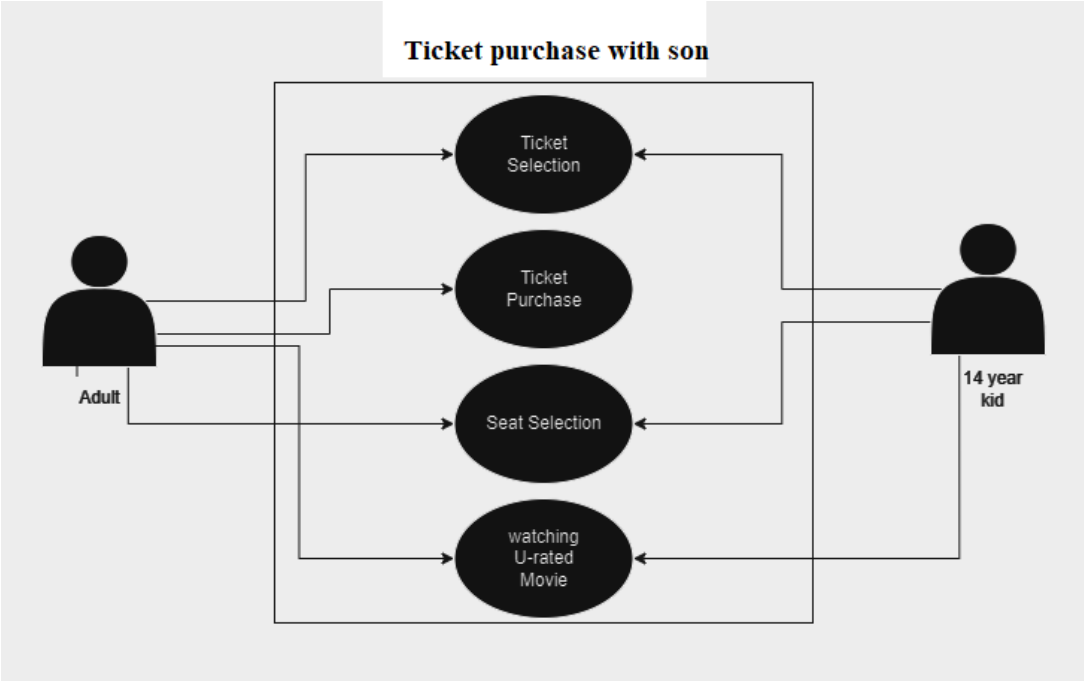
8. *Use Case Diagram #1:*



9. *Use Case Diagram #2:*



10. *Use Case Diagram #3:*



3.2 Non-Functional Requirements

1. Security

- Information such as user information, passwords, or payment methods will be secured and preventable from any data leaks or breaches.
- User information will also be secured against brute-force login by limiting a user to a total of 5 attempts, once those attempts have been reached they will get an email letting them know
- All customer data is encrypted, both in transit and stored. Admin access uses multi-factor authentication. Account brute force attacks are safeguarded against.
- Customers have the ability to enable 2 step authentication (2FA) to take any precautions against any unpreventable attacks.

2. UI Configuration

- UI will be very easy to use as it will be designed to be very friendly, easy to use and navigate.

3. Servers & Response time

- The average website response rate is 2.5, our website's response time is 2.0, so it will be ahead of many websites and will be lag-free.
- Server can handle over 10,000 orders during peak with no problem

4. Portability

- UI can be accessed through all available web browsers.
- UI can be accessed through all available devices including game consoles and smart watches.

5. Reliability

- The system should maintain over 99.9% uptime measured monthly. No more than 1% of transactions should fail due to technical issues.

6. Logical Database Requirements

- The System uses a SQL Database
- Stores information such as shows, times, seatings, charts, availability, customer accounts and orders, payments, admin access and activity logs.
- Database normalization principles reduce redundancy. Foreign keys enforce data relationships.

7. Fraud

- Fraud detection identifies suspicious purchasing patterns. Customer service can access order histories to assist users. Notifications are sent when shows or sales are added.

8. Availability

- The site should be up 24/7 unless we have scheduled maintenance. We want unplanned downtime to be less than 1% of the year.
- Movies will be updated frequently by administrators including movie times, seats, etc.

9. Inverse Requirements

- The System will not sell your information to any person or company.
- The System will have no security breaches to damage or have any stolen content or private information of customers.
- The System will not automatically save your payment information without asking.
- The System will not email you if you have not checked the news letter box upon a purchase.

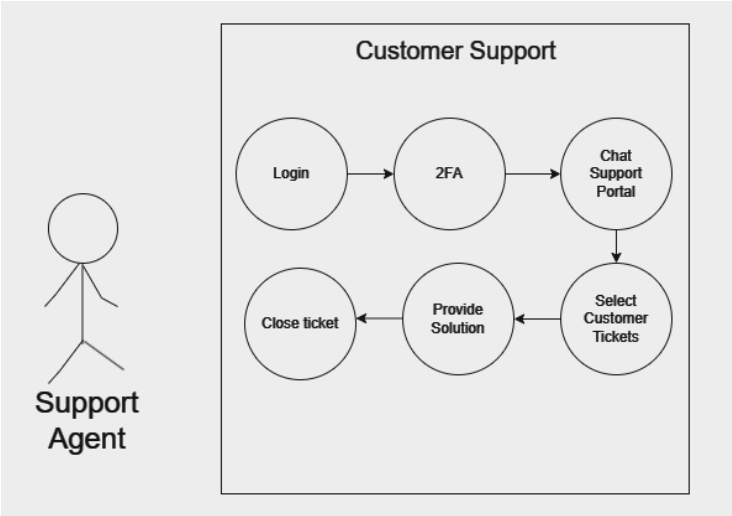
10. Maintainability

- The web application will support regular updates to ensure compatibility with latest technology.
- The web application will apply bug fixes regularly.
- The web application will apply security patches when needed.

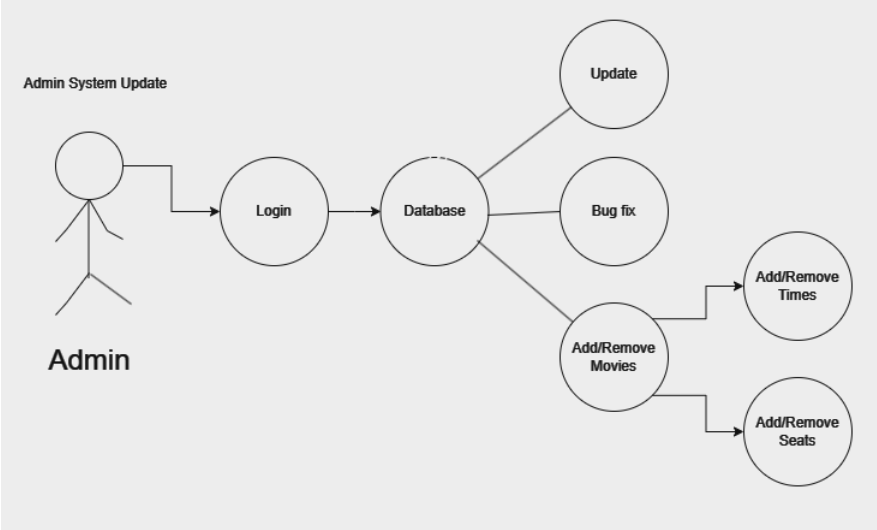
11. Live Support

- On the bottom right of the website we have a live support chat that is 24/7 to help ensure customers get the help they need.
- Live support agents can help with purchase problems, ticket problems and any errors the customer might have

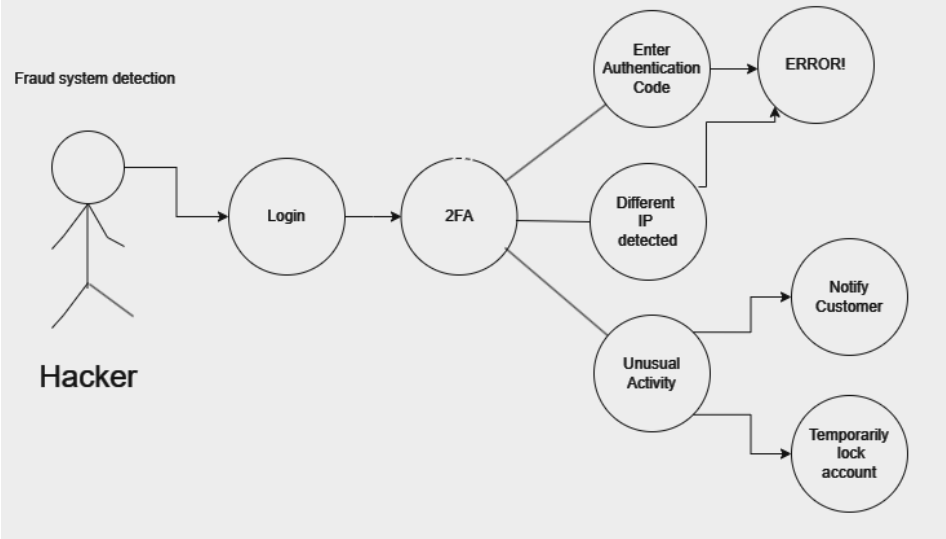
12. *Use Case Diagram #1:*



13. *Use Case Diagram #2:*



14. *Use Case Diagram #3:*



4. Analysis Model Diagram

5. Conclusion

In conclusion, we have one of the best websites that will cater to your movie ticket needs 24/7 with our filter tool that helps you search the right movie for you and your friends. Our website's response time, security, fraud detection, as well as a personalized account based on previous purchases is also something that will surely catch your attention.

Software Design Specification

Movie Ticket System

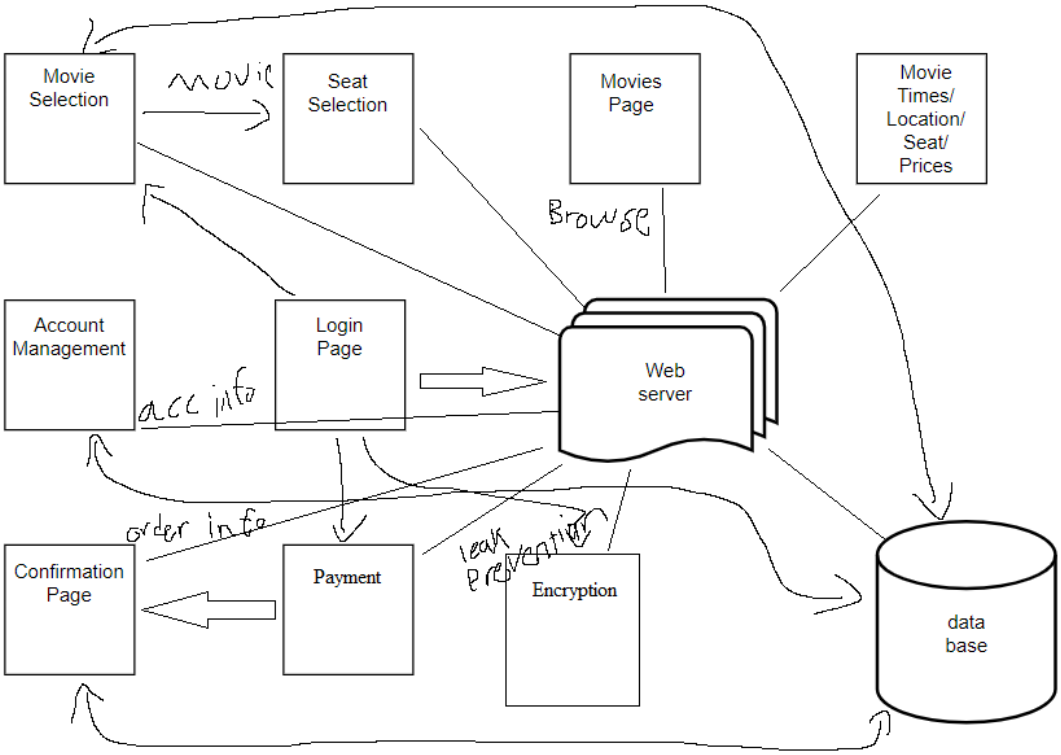
Moe Jawadi
Joel Alvarado
Ernesto Cornejo

System Description

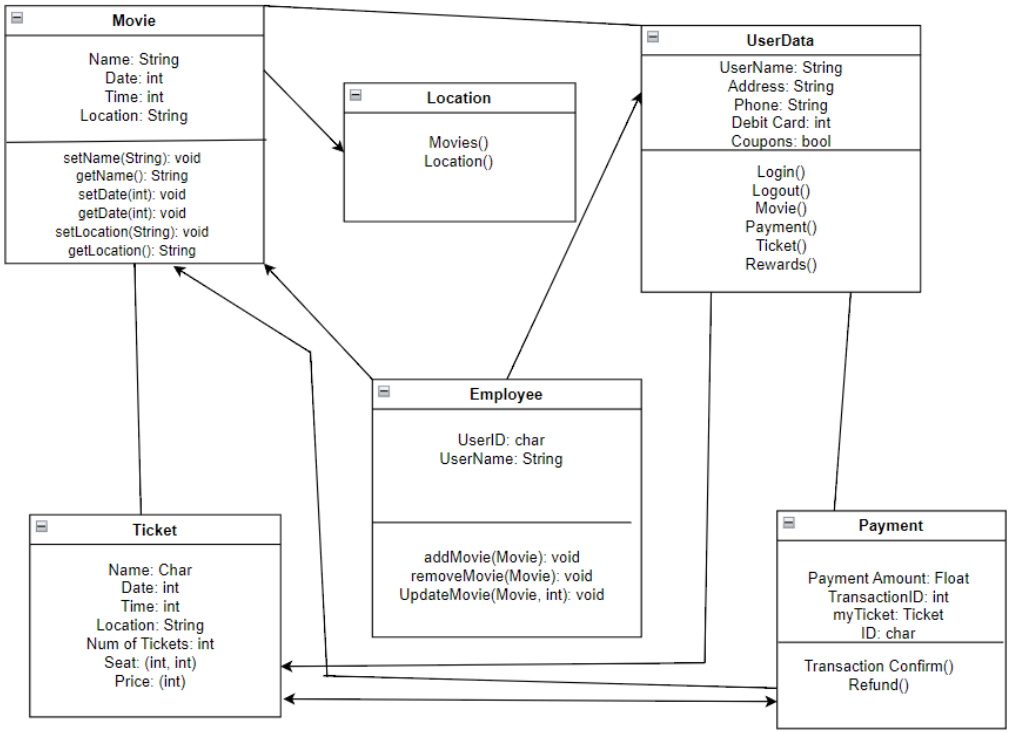
The movie ticket system is a web application that allows users to purchase movie tickets online. Users can browse movies by showtime, genre, title, etc and select seats on an interactive seating chart. The system manages user accounts, profiles, payment processing, and sends email receipts/confirmations. Admins can add new movies, showtimes, and seating availability. The movie ticket system was developed using a responsive design approach that will allow users to connect to our website via tablet, desktop, smartphone, laptop, smartwatches, and video game consoles. It will have a database that will store information about shows, times, seatings, customer accounts and their saved orders. Our system will have security measures such as two factor authorization, key encryption, and SSL certificate in order to protect user information.

6. Software Architecture Overview

- 6.1.1 ARCHITECTURAL DIAGRAM



- 6.1.2 UML DIAGRAM OF SOFTWARE DESIGN



-

Description of classes

Movie class: The Movie class is set to give us provide a lot of information about any movie that is selected.

Location class tells us what is the movie and where is its location

UserData class showcases any user based information of a customer

Employee class gets special access to UserData and Movie

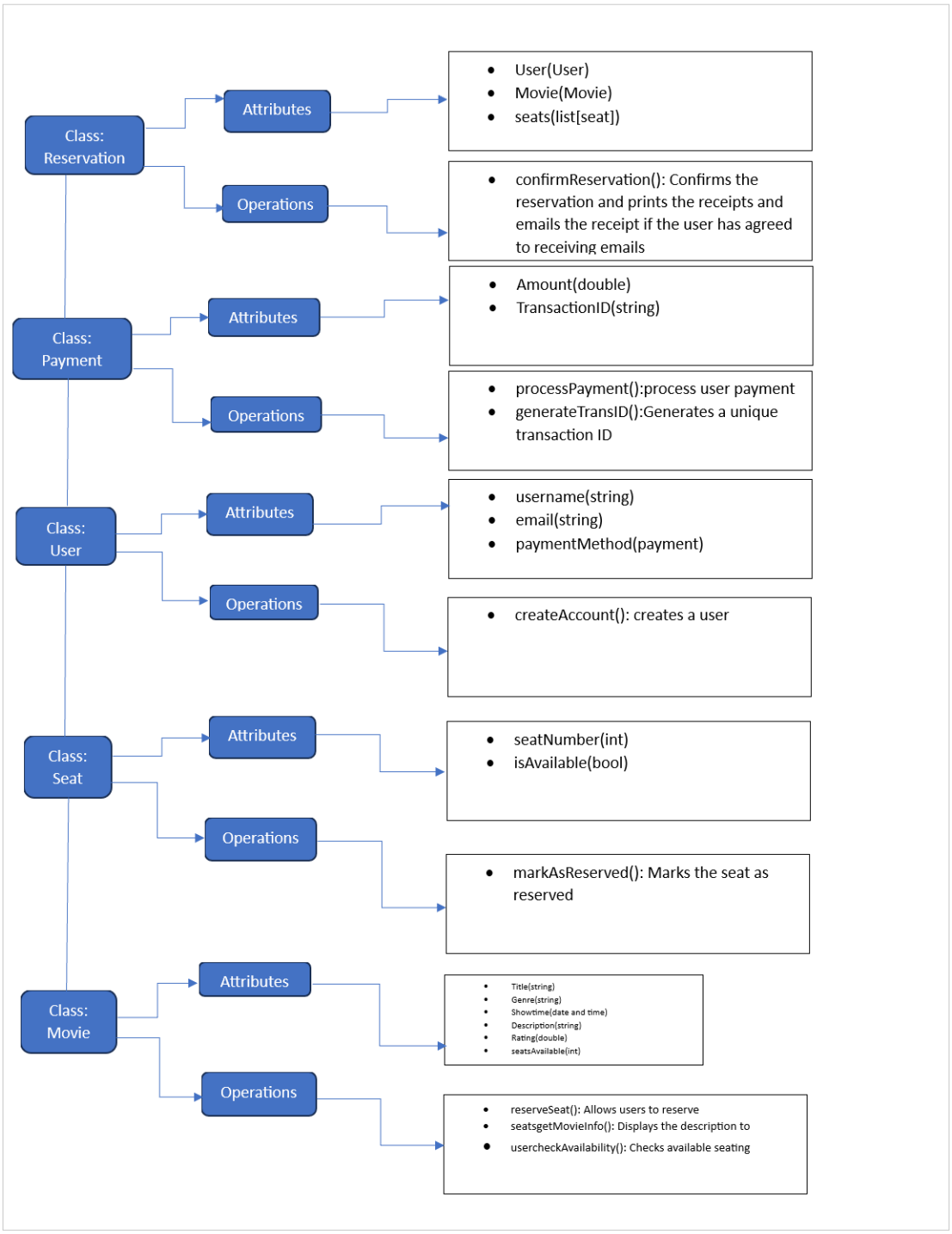
Ticket class includes all information regarding purchased or browsed tickets.

Payment class displays the payment amount, transaction ID and the payment ID. **-**

Description of attributes

Attributes and characteristics of each class are showcased in the diagram above, when we set something to its value, such as Name to String, Date to int, Time to int, Location to String, add movie, remove movie, update movie and etc. we used these attributes to help us get the name of our movie, date of our movie, time of our movie, location of our movie, ability to update, add or remove a movie without any complications.

- 6.1.3 SWA DIAGRAM OF SOFTWARE DESIGN



Description of Classes

The createAccount() allows users to create an account

The reserveSeat() allows users to reserve up to 20 seats

The getMovieInfo() displays a synopsis of the movie

The userCheckAvailability() checks for any seats available for a particular movie and show time

The markAsReserved() marks seats that have been reserved in order to not allow a different user to choose that same seat

The confirmReservation() confirms the users reservation and prints the receipt or emails the receipt to the user

The processPayment() processes user payment

The generateTransID() generates a unique transaction ID that authenticates user purchase

Description of Attributes

We will start off with user attributes such as username which takes in a string value as well as email, but for paymentMethod we will take in a payment object. For the Movie attributes we will have a few string data types such as title, genre, and description. We will also have showtime which will be a date and time, we will also have rating which is of double data type, and seatsAvailable which is an int. Our attributes for seat are only seatNumber, which is an int value and isAvailable which is a boolean data type. Our Reservation has attributes that take in objects as their parameters such as User(User) which shows which user reserved certain seats, and Movie(Movie) which shows what movie the reservations were made for, and lastly Seats(list[seats]) Which holds the list of the seats that are reserved for a given movie. Our last attributes are of the payment class. The attributes are as follows, Amount which is of double data type and displays the total balance the user needs to pay, and TransactionID which is a string data type and is a unique transaction identifier.

7. Development Plan and Timeline

- 7.1 Partitioning of tasks

Moe: Implement user registration, login, profiles and settings pages (Est: 3 weeks)
Joel: Implement movie browsing, selection, seating chart and payment pages (Est: 4 Weeks)
Ernesto: Implement database schema, movie/showtime admin features (Est: 5 weeks)

- 7.2 Timeline

Month 1

Week 1: Set up development environments, build basic UI pages
Week 2: Create database schema, begin work on user account pages
Week 3: Complete user registration and login pages
Week 4: Finish user profile and settings pages

Month 2

Week 1: Start on movie browsing and selection pages
Week 2: Complete movie browsing, begin selection flows
Week 3: Wrap up movie selection, start on seating charts
Week 4: Complete seating charts, begin payment pages

Month 3

Week 1: Finish payment pages, start order management
Week 2: Complete order management functionality
Week 3: Begin admin features for movie and showtime management
Week 4: Complete admin features

Month 4

Week 1: Testing across user flows and ecosystems
Week 2: Fix bugs, refine flows based on tests
Week 3: Finalize documentation, deploy to production

- 7.3 Test Case Template

Test Case Template									
TestCaseId	Component	Priority	Description/Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed By
userLogin_1	Website_Login_Interf	P01	Verify the user's login email matches the user login password	An active account in our website	1.Input valid email address 2.Input password attached to the email 3.Press enter	User logs in and is presented with a personalized account	User logs in and is presented with a personalized account	Pass	TesterK
userLogin_1	Search_Bar_Module	P01	Verify the user's login email matches the user login password	An active account in our website	1.Input valid email address 2.Input password attached to the email 3.Press enter	User logs in and is presented with a personalized account	User is prompted "Create an account or re-enter email or password" message	Fail	TesterK
userRegistration_	New_User	P01	Verify that a new user can successfully create an account with valid information	Valid email address	1.Navigate to the registration page 2.Enter Valid Information (name, email, password) 3.Click "Create Account"	account is created, and they can log in with provided information	User account created	Pass	TesterK
userRegistration_	New_User	P01	Verify that a new user can successfully create an account with valid information	Valid email address	1.Navigate to the registration page 2.Enter Valid Information (name, email, password) 3.Click "Create Account"	account is created, and they can log in with provided info	User account not created	Fail	TesterK
securityMeasure_1	Secure_Account	P00	Test the systems response to a potential security breach, such as a brute-force login attempt	N/A	1.Attempt to login with incorrect credentials multiple times 2.Exceed the allowed number of failed login in attempts (5 attempts)	The system locks the users account temporarily and will also receive an email letting them know	User was received an email after the 5 attempts were made, and their account was temporarily locked	Pass	TesterK
securityMeasure_1	Secure_Account	P00	Test the systems response to a potential security breach, such as a brute-force login attempt	N/A	1.Attempt to login with incorrect credentials multiple times 2.Exceed the allowed number of failed login in attempts (5 attempts)	The system locks the users account temporarily and will also receive an email letting them know	User was not received an email and the account wasn't locked	Fail	TesterK
PaymentProccess_	User_Purchase	P00	Test the payment process for purchasing a movie ticket	Movie, showtime, and number of seats selected to purchase	1.Select Movie and showtime 2.Choose Seats 3.Proceed to payment with "Purchase." 4.Enter valid payment information 5.Confirm the purchase	The user successfully completes the payment process, and a ticket is generated and an email is sent	User was not charged but the reservation was made and email was sent	Fail	TesterK
PaymentProccess	User_Purchase	P00	Test the payment process for purchasing a movie ticket	Movie, showtime, and number of seats selected to purchase	1.Select Movie and showtime 2.Choose Seats 3.Proceed to payment with "Purchase." 4.Enter valid payment information 5.Confirm the purchase	The user successfully completes the payment process, and a ticket is generated and an email is sent	User was charge, reservation was made, and they were sent an email	Pass	TesterK
profileUpdate_1	Update_User_Profile	P01	Verify that users can update their profiles and payment methods	Valid user account	1.Login into the system 2.Naviagte to user profile page 3.Update user information(email address or payment method)	User information was updated, and changes were saved	User information was not saved	Fail	TesterK
profileUpdate_1	Update_User_Profile	P01	Verify that users can update their profiles and payment methods	Valid user account	1.Login into the system 2.Naviagte to user profile page 3.Update user information(email address or payment method)	User information was updated, and changes were saved	User information was updated and saved	Pass	TesterK

- 7.3.1 (2 Unit Tests: userRegistration, both pass and fail test)
- 7.3.2 (2 Functional tests: userLogin, both a pass and fail tests)
- 7.3.3 (2 System tests: paymentProcess: both pass and fail tests)

8. Verification Test Plan for the New Ticketing System

1. Objective:

Thoroughly testing this new ticketing system is crucial for a smooth rollout. We want to validate all the key functions inside and out to catch any bugs before users start relying on it. It is recommended we take a structured approach with multiple phases so we cover everything.

2. Unit Testing:

Objective: Check that individual parts work right on their own.

Description: The tech team will test pieces like the login, ticket creation, and notification tools.

Test Sets/Vectors: Trying out different user logins and ticket info.

Coverage: Making sure each separate component functions properly.

3. Integration Testing:

Objective: Confirm different pieces work well together.

Description: Testing interactions like user accounts with ticket forms, and connections between ticketing and notifications.

Test Sets/Vectors: Varying user roles and actions.

Coverage: Catching issues between components early.

4. System Testing:

Objective: Validate entire workflows from start to finish.

Description: Testing major processes like account creation, ticket opening, comments, and closure across user roles.

Test Sets/Vectors: Walking through scenarios for different users.

Coverage: Ensuring the full system operates smoothly.

5. User Acceptance Testing (UAT):

Objective: Test real-world use cases.

Description: Starting with internal team trials then expanding to select external customers. Testing on different devices, operating systems and browsers.

Test Sets/Vectors: Simulating real customer interactions.

Coverage: Getting feedback to improve user experience.

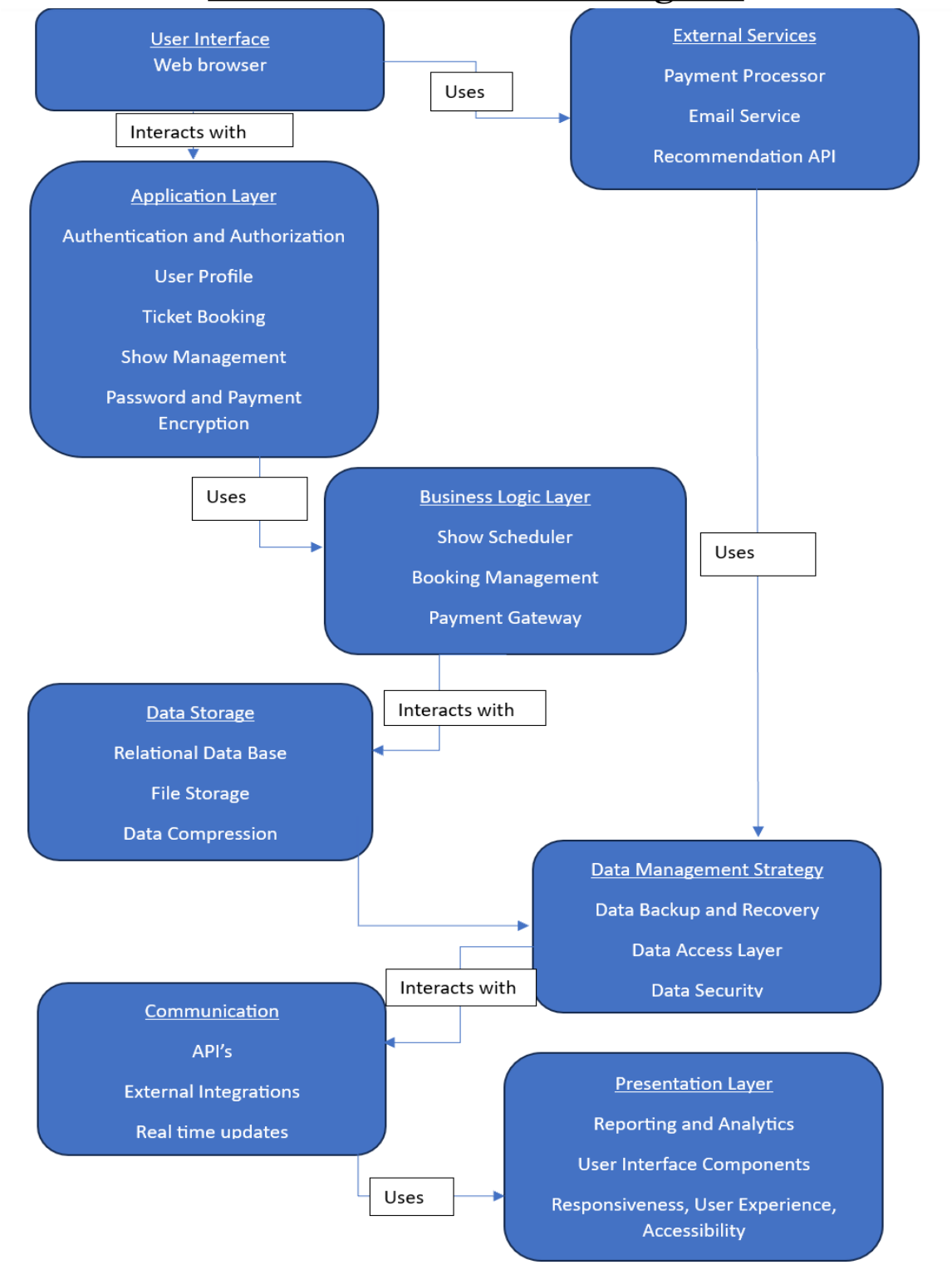
6. **Environments**:

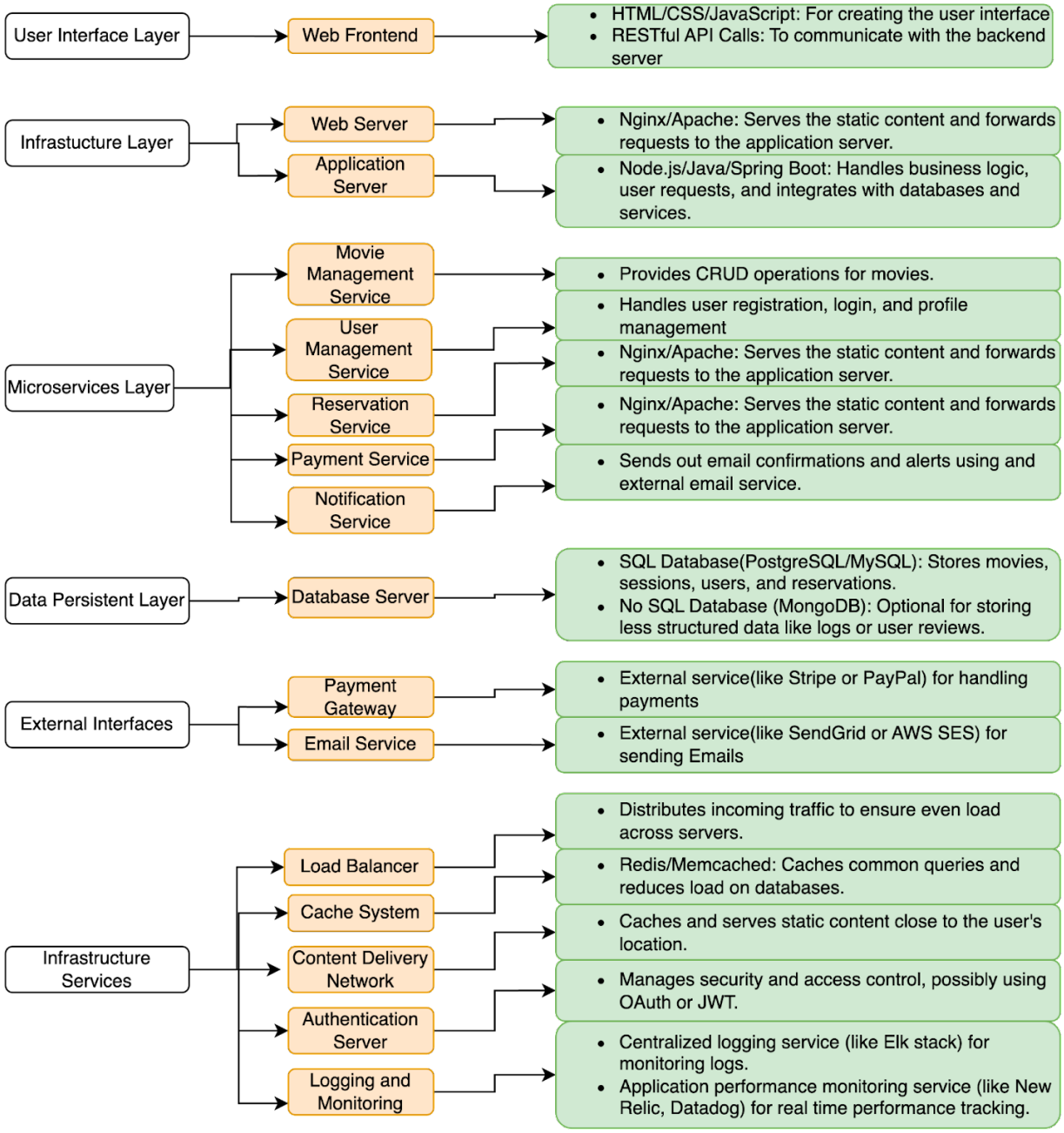
In terms of environments, we'll want to test locally in dev first, before promoting builds to the staging and production environments. Testing directly in production as part of UAT reduces surprises down the line. Speaking of environments, we also need to test across different hardware like desktops, laptops, mobile devices with varied OS and browser combinations.

7. **Quality Evaluation**:

Throughout all testing, we need to continually evaluate quality criteria like functionality, reliability, security, and performance. Leveraging checklists and tracking progress in a tool like TestRail will keep testing structured. Frequent sync-ups with stakeholders and the team will ensure we're covering everything necessary to deliver a solid ticketing system that delights customers.

Software Architecture Diagram




User Interface Layer
Web Frontend
HTML/CSS/JavaScript: For creating the user interface
RESTful API Calls: To communicate with the backend server
Infrastucture Layer
Web Server
Application Server
Nginx/Apache: Serves the static content and forwards requests to the application server.
Node.js/Java/Spring Boot: Handles business logic, user requests, and integrates with databases and services.
Microservices Layer
Movie Management Service
User Management Service
Reservation Service
Payment Service
Notification Service
Provides CRUD operations for movies.
Handles user registration, login, and profile management
Nginx/Apache: Serves the static content and forwards requests to the application server.
Nginx/Apache: Serves the static content and forwards requests to the application server.
Sends out email confirmations and alerts using and external email service.
Data Persistent Layer
Database Server
SQL Database(PostgreSQL/MySQL): Stores movies, sessions, users, and reservations.
No SQL Database (MongoDB): Optional for storing less structured data like logs or user reviews.
External Interfaces
Payment Gateway
Email Service
External service(like Stripe or PayPal) for handling payments
External service(like SendGrid or AWS SES) for sending Emails
Infrastructure Services
Load Balancer
Cache System
Content Delivery Network
Authentication Server
Logging and Monitoring
Distributes incoming traffic to ensure even load across servers.
Redis/Memcached: Caches common queries and reduces load on databases.
Caches and serves static content close to the user's location.
Manages security and access control, possibly using OAuth or JWT.
Centralized logging service (like Elk stack) for monitoring logs.
Application performance monitoring service (like New Relic, Datadog) for real time performance tracking.

The original class diagram was just a basic sketch of the software design. We've now fleshed it out into a full system that could actually be launched. Whereas the old diagram only showed the classes, the new design has all the components needed to build a real working product. There's a user interface added so people can interact with the system through a nice graphical screen. The backend has been split up - one part serves the simple static content like images, while another part handles the complex dynamic information like customer accounts. Instead of one giant piece of software, we've broken it into smaller independent pieces that each handle specific features like managing movies or reservations. This way each piece can be updated and scaled as needed. We're now including databases to actually store and manage all the data like customer info and payment details. One database stores structured data like names and addresses, while another stores looser unstructured data like product images and descriptions. We're integrating with payment systems to actually handle money transfers and take bookings. And we can connect to email services to send notifications and confirmations to users. Behind the scenes, we're adding load balancers to handle traffic, caching to speed things up, and content delivery systems to reduce lag. There's also proper logins, permissions and monitoring to manage security and issues. Overall, the design has gone from a basic idea to a real working architecture that can handle things like performance, scaling, reliability and security needed to launch and run a business.

Data Management Strategy

We first will start with our Database Selection, Structure, Partitioning, Backup and Emergency recovery, Security, and lastly finish off with the Trade-offs.

Selection -

SQL Database: Enroll a relational database using MySQL to ensure user information, purchase record, as well as product specifications.

NoSQL Database: for Partially-Structured data such as user reviews, comments, referrals, and customer feedback. using NoSQL databases we are able to leverage and stretch what we can or cannot accomplish.

Structure

SQL Database: We will be using 4 types of Databases to set up the structure of our SQL Database.
Starting with a User database that is able to maintain user information. The second will be our record database and that will have any purchase history of any products and purchase-related information.
Our third will be our product specification, keeping information related to our product mainly our stock, availability and pricing. and lastly, our fourth will be product analytics where everything will be documented from purchase time, till expiration.

NoSQL Database: We will be using 2 types of Databases to set up the structure of our NoSQL Database.
We will begin with our Partial-Structured database, it is responsible to help develop and maintain our content on a strict database that will develop and maintain information history of reviews, comments, referrals and customer feedback. Our second database will be our user management, that is able to enable, customize user profiles and preferences.

Partitioning

Implementation of sharding to help allow us to maintain high levels of oncoming users as well as orders. We are implementing sharding will help us manage bits and pieces across multiple nodes with the use of this administration. This will help with any potential effects of the website overload to help maintain stability and scaling as more data enters.
Document storage in any document based format with NoSQL will increase the read performance by enabling swift and efficient data retrieval which ultimately dismisses the use for any lower layer storage.

Backup & Emergency recovery -

Commonly and Routinely backup all data including SQL databases and NoSQL Database to prevent any danger of any breaches or leaks. This will help keep data safely protected and automatically restored incase of an emergency.
In case of a special case emergency there will be a nearby private storage that keeps a copy of all encrypted data from all databases incase of an unknown.

Security -

Assurance of data encryption that protects Database accessibility. This will prevent any unauthorized access without the specific “keys”. We want to make sure your personal information stays safe and secure. So, we'll be using encryption to scramble stored data that way, only people who should see it can read it. Whenever we need to send information from one place to another, we'll use secure ways of transmitting it. In order to protect data integrity and prevent unauthorized access we will conduct regular audits, as well as penetration testing on the database to ensure security measures are up to date. For our databases, we'll take precautions to avoid hacking. By taking these common sense steps, we aim to fully protect the privacy and accuracy of your data. Our goal is simple: keep your information totally secure while it's in our care.

Trade Offs

Difficulty- Multiple Databases comes with a more complication, for example website synchronization does its own thing where it has to maintain consistent and up to date data through all databases.
Maintenance is very important as every database has a different set of preconditions and layouts. Development often needs additional effort as it is sometimes extremely difficult to deal with any possible errors or database failure that may occur.

Benefits - Databases will come with better performance which can lead to faster response times and less customer delay. As well as Scalability which helps with the growth of each database depending on what we need. and Flexibility helps with database match types for each application which creates a faster process.

Theater Database

TheatreID	# of seats	Layouts ADA Location
15	15	Regular X San Marcos
11	13	Regular X National City
8	20	Regular X El Cajon
13	6	Deluxe X Poway

User Database

Username	Email	Password Address Date of Birth
Jma050	Jmao667@gmai l.c om	Samantha9316 553 Wells ave 03/11/2002
RulerMm	JohnSmith59@ ya hoo.com	BbCcDd7412 992 Lincoln Ave 09/29/1998
mikeJared2 00 1	Mikereaper20 01 @gmail.com	Olpacio50 1047 Houston Ln 05/1/2001
idontkno	gorlockthedestr oy er@yahoo.com	GorloMthy943n 6419 Parkway 04/21/1996 Plaza

Life Cycle Model

We wanted to make buying tickets, using accounts, and tracking sales numbers an easy process. Our system uses data storage, payment processors, and flexible design that adjusts across devices. We tested pieces individually and end-to-end. We talked about adding more complex options like customized recommendations. We took standard security protections like password scrambling, info encoding, and carefully vetting user inputs to stop attacks. We also stay updated on the latest protections and fixes. Our development group took an evolutionary prototype model approach in order to add to our project week-to-week. First just account tools, then movie browsing abilities, seating charts, payments etc. Checking each part fully before moving forward. This flexible work method served the project size well, though mapping all the specifics earlier would have focused our direction sooner.

Summary

In summary, we built an easy to use service for browsing upcoming movies in the user's area, viewing showtimes, reserving good seats, and getting tickets fast with no hassles required! The steps included combining modern technologies like smooth databases, responsive design systems and robust security protocols into one user-friendly website. User accommodations like saved payment methods, one-click reorders and group seating optimizers save a lot of time. In the future we would like to develop intelligent features to suggest movies based on past preferences and simplify group outing coordination. And there's room to grow and potentially become a one-stop-shop for comparing theaters across cinema mega chains. But for now, establishing solid infrastructure provides capacity to gradually refine rough spots while expanding capabilities users truly desire over time. Because at the end of the day, this platform isn't about software modules or architectures. It's built piece-by-piece to fit real lifestyles, by understanding daily frustrations and goals around grabbing those perfect seats with friends to catch the latest movie

release. And if we listen closely to ongoing feedback from the fans reserving tickets. Well, then more helpful features will unveil themselves.

Conclusion

In conclusion, we learned a lot while working on this movie ticket website during all stages of coming up with ideas, making plans, and building it. Getting positive and constructive feedback from the TA provided immense value in leveling up skills and refining the end product over time. But a few key lessons stood out as especially impactful. First, clearly formatting user requirements upfront makes lives easier for engineers and customers alike. We initially overlooked ease-of-understanding, but simplifying key details makes a system more user-friendly overall which is an easy win. Second, supplementing diagrams with written descriptions creates helpful redundancy for explaining complicated system diagrams. We favored visuals yet left some context gaps. Offering both formats improves understanding all around. And finally, putting in effort early on to set realistic deadlines helped us coordinate our plans better. Overall, the good teamwork experience left us truly thankful for matching goals with what's possible to make a website that works well.