THIS DOCUMENT CONTAINS DETAILINGS OF A TICKETING SITE, NAMELY AN SRS.
SCROLL TO VIEW

Ticketing System

Software Requirements Specification

Ticket Hub

January 31, 2024

Group #9

Amelia Grevin, Elliot Gambale, Hannah Ferdows

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>
02/14/24	<version 1>	<Amelia Grevin, Hannah Ferdows, Elliot Gambale>	<first revision>
02/24/24	<version 2>	<Amelia Grevin, Hannah Ferdows, Elliot Gambale>	<second revision>
03/13/24	<version 3>	<Amelia Grevin, Hannah Ferdows, Elliot Gambale>	<third revision>
03/13/24	<version 4.	<Amelia Grevin, Hannah Ferdows, Elliot Gambale>	<fourth revision>

Document Approval

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

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

Table of Contents

1. Introduction

This SRS document lays out the process to create an online platform for movie theater guests. You will find a step-by-step description for each engineering team to help construct their specific part of the site.

1.1 Purpose

This SRS is intended for the software developers tasked with creating a user-friendly platform to buy movie tickets.

1.2 Scope

Our products name: Ticket Hub
Product functions:

- display available movie options
- allow for movie selection
- display available seats
- select up to 20 seats
- select movie time
- select movie location
- select child/student/adult/senior
- press checkout button
- put in credit info
- email ticket
- be able to handle 1000+ users
- run in web browser
- block bots with over 20+ ticket requests
- customer feedback
- admin
- take in rotten tomatoes

Ticket booth is a user-friendly, accessible website that enables users to buy tickets to movies while being able to see reviews of given movies.

All clients of the theaters under the umbrella company have access to the website.

1.3 Definitions, Acronyms, and Abbreviations

SQL: Structured Query Language
API: Application Programming Interface
MacOS: Macintosh Operating System
MTBF: Mean time between failures
SRS: Software Requirements Specifications

1.4 References

Software development life cycle – Author: Gus Hanna
Source: Canvas Slides

useCases.ppt – Author: Gus Hanna
Source: Canvas Slides

How to Write a Software Requirements Specification
– Author: Gerhard Kruger & Charles Lane

Date: 1/17/23
Source: (Perforce.com)
https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document

1.5 Overview

The rest of this SRS contains detailed information of the functionality of the website as well as any constraints within the design process. It will also include the distribution of work among the software engineering team assigned to this project.

You can search below for needed information via the following sections of the SRS below: general overview, product functionality, requirement specifications, and use case models.

2. General Description

Find the following descriptions below: product perspective, product functions, user characteristics, general constraints, assumptions and dependencies.

2.1 Product Perspective

This site should work independently from other ticketing systems. It works similarly to popular sites such as amctheatres.com and fandango.com, but a more thorough description of "Ticket booth" site is described below:

Operation Inside Various Constraints:

System interfaces: System Interfaces should allow the site to open on any different hardware and adjust the display to fit the screen and function properly.
User interfaces: The user interfaces are split among 3 different screens; home, selection, and payment. The home screen will display all the available movies, the theater location, and a search bar. The selection screen will appear after a movie and location are chosen and the movie time

and the seat selection screen will appear. Finally, after these selections, the payment screen will appear and the user will be asked to type in credit card/payment information.
Hardware interfaces: The hardware interface is different from user to user. A website such as this one can be opened on a mobile phone, computer, ipad, ect…
Software interfaces:The primary software interfaces to be used are: intellij IDEA(a javaScript coding IDE)...
Communications interfaces:Communication interfaces include – communication between website and gps in order to locate nearest theater, communication between website and pre-existing movie trailers (in order to display on site), communication between site and online review sites (such as rotten tomatoes), communication between site and secure payment site (PayPal), communication between site and wifi and data, and communication between site and captcha (the authorizing service against bots). It should also be noted that communication with cloud databases will be likely and necessary in order to access any stored data regarding the movies and/or showtimes.
Memory: Gives user option to remember option, also gives an option to remember location when using, will need storage for all showtimes, seats, and who is buying what seat. Keep track of how many tickets are sold with a maximum of 20 tickets per theater. Also needs dedicated storage for reviews of movies as well as website experience. Needs to have dedicated memory to display reviews from critics of available movies.
Operations:User operations include the selection of tickets, selection of movie times, inputting payment info, selecting a theater location, as well as a seat.
Administration operations include updating available movies, updating movie times, updating available seats and unavailable seats, debugging any issues with the site.
Site adaptation requirements:In order to ensure the site to run on different platforms we plan on testing it and looking to implement the ability for the site to work properly under any wifi provider. Other means of extra precautions needed to be kept in mind depending on different adaptations of the site includes implementing a strategy for our programming language to cooperate with other possible ones it may need to work with, as well as to ensure that the site is not plagiarizing or copying without allowed access.

2.2 Product Functions

Product functionality depends on the user's needs. The main purpose of the platform is to allow users to purchase tickets via an online platform. This however, is not the only function of Ticket Hub. Ticket Hub provides users a platform to buy tickets, browse available movies in theaters, view online ratings, and view theater locations. The functionality of each individual stage of the buying process is described further in other sections of this SRS.

2.3 User Characteristics

This site should be accessible and available to all users regardless of their technical expertise and experience. It is meant to be extremely user friendly and intuitive.

2.4 General Constraints

Must follow all regulatory policies for selling tickets such that laws or illegal activity is broken or taking place. Devices must be able to have access to a browser as well as the internet. Website will require cookies from the user in order to process transactions. The website will also log activity by each user, logging what they buy and when they buy it. The website will also ask the user for their location in order to determine nearby theaters. All admins to the website will have access to these logs. Users on the website will be able to determine what movie they want to see and what kind of ticket they will purchase. This website should be able to operate with at least 1000 users at a time and still operate functionally. If the software were to fail, this would be costly to revenue as each minute is a potentially ticket lost. This website will not store the user's data, for safety to the user. It will also be programmed in such a way to keep the software decrypted from unwanted users, such as bots. This can be achieved by captchas to see if users are human.

2.5 Assumptions and Dependencies

Be weary of any changes in the way the website functions depending on what web browser is being used, and account for them if any. Also be wary of how website display may change depending on the hardware the user is using.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

HTML, CSS, JavaScript and React framework will help build a user interface and front end.

3.1.2 Hardware Interfaces

Mobile view, Laptop/computer view, iPad (tablet) are the most likely means of hardware interfaces we must look to work around and be compatible with.

3.1.3 Software Interfaces

This site will be built using Python and JavaScript. JavaScript will primarily be used to design the website itself, and Python will be used with all communication aspects.

- Django framework for maintenance, integration efficiency and code reusability

3.1.4 Communications Interfaces

MONGODB &/or MySQL – (outside databases communication languages in order to retract information and data). Another communication interface we may need to plan for is building or using a pre-existing API to access cloud computed information/

3.2 Functional Requirements

Find, in this section, specific features of the software project.

3.2.1 Extracting Review Data from Outside Sites

3.2.1.1 Introduction
Due to our website's unique planning to display reviews as seen on sites such as Rotten Tomatoes, below we will discuss in detail the logistics of doing so.

3.2.1.2 Inputs
The inputs necessary to extract are a rating out of 5 stars, and the top 2 reviews.

3.2.1.3 Processing
In order to get this data and process it, we must scrape it with an SQL server, and upload it to our cloud for easy access through APIs.

3.2.1.4 Outputs
Our website will therefore output said rating, an image displaying the stars visually, and the 2 reviews as quotes to reference at the bottom of the movie page, for those seeking such.

3.2.1.5 Error Handling
Some errors which may be intercepted during this process are a movie with no reviews, in which case will be caught by the error handling message saying “No reviews currently available”. Another is the movie not being on the second party website at all, where the same message could handle it. Lastly, it shall display this message again in case the data fails to load.

3.2.2 Updating Seating Maps for Tickets Purchased in Person

3.2.2.1 Introduction
Due to the need to consider that some tickets may be purchased in person, while the seating map may fill according to users’ choices online, we must also implement the functionality for employees to fill the map as tickets are sold at the theater.

3.2.2.2 Inputs
The inputs will be a seat number, and seat row to fill for each ticket.

3.2.2.3 Processing
Given that the site will need an administrative side, in order to make any adjustments, updates, etc. there should also be an employee view where seat maps can be filled and adjusted.

3.2.2.4 Outputs
The seat map should ideally update live to the website as employees adjust and fill it, showcasing seats that are taken (by grayed out colors and other signifiers).

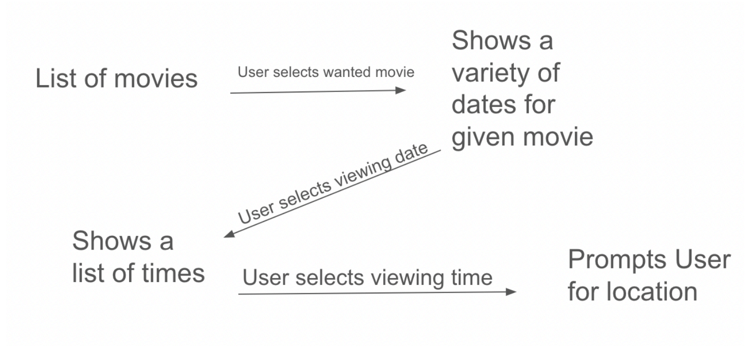
3.2.2.5 Error Handling
In the case in which something crashes, or parts are not communicating well with each other, the system is to display, “Seat map currently not available. Check with your local theater.” This way,

customers will have to be seated in person until the map is up and running again, and server connection is restored. One other error to handle is wanting to purchase a certain number of tickets, when there are less seats available than tickets desired. In this case as well, there will be an error message prompting the user to return to the last screen and try again with a different number of tickets, or to be redirected to the movies page in order to choose a new showing.

3.3 Use Cases

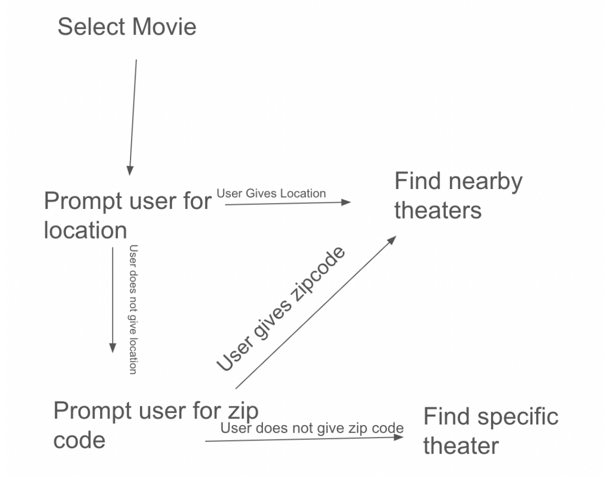
3.3.1 Use Case #1

The first use case for the website would be to display all available movies, and their showtimes each day into a month in advance, so customers can plan ahead and come accordingly.


List of movies
User selects wanted movie
Shows a variety of dates for given movie
User selects viewing date
Shows a list of times
User selects viewing time
Prompts User for location

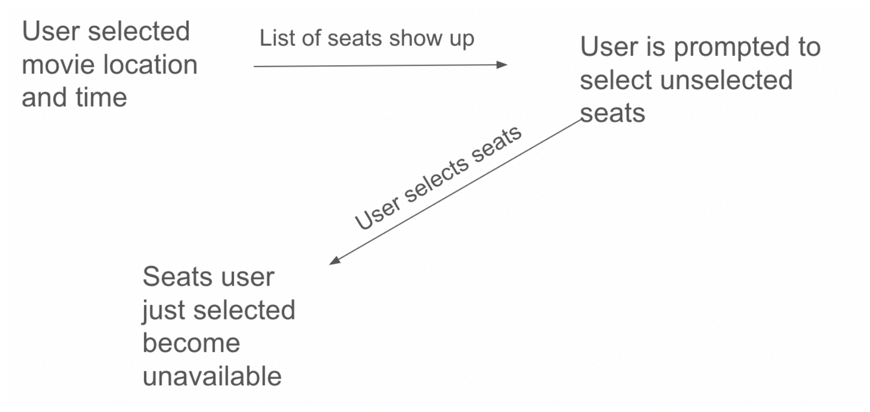
3.3.2 Use Case #2

The second use case for our ticketing website would be for purchasing a ticket – users will need specifications! like which location they're looking at and choice of movie.



3.3.3 Use Case #3

The final use case for the user is for them to view specific seats/ a seating map and whether certain seats that they'd like are open for them to buy or sit in. (Also, if there are a certain number of seats open together in the case in which someone wants to come and sit with their friends)


User selected movie location and time
List of seats show up
User is prompted to select unselected seats
User selects seats
Seats user just selected become unavailable

…

3.4 Classes / Objects

3.4.1 User / User Profile

The class ‘user’ will consist of the class object userProfile, which will be used to access methods and *functions* regarding view available tickets, purchase tickets, view upcoming showtimes, all locations available, and seat maps for a specific movie and time as chosen by the user.

Movie Details / Movie

The class ‘movieDetails’ will consist of the class object movie, which will be used to access method and *functions* such as the date of the specific movie, its showtimes, the name, number of seats available, whether the movie has started yet or not (after a showtime is chosen) and its reviews – which will be externally extracted from movie review websites and stored as one of its components.

<Reference to functional requirements and/or use cases>

…

3.5 Non-Functional Requirements

3.5.1 Performance

Our application should be able to run and to load tickets efficiently, regardless of the user count. During normal run time, our site should take no more that 85% of CPU usage, ignoring any errors that may occur during high capacity. To lessen the chance of this happening, the implementation of a 1000 user cap will be placed so that the performance will not be drastically affected.

3.5.2 Reliability

Ensure the website remains functioning while the user is active. Also, assign a specific period of maintenance time for the admin to ensure that the site is up and functioning all other times. Make sure the site can signal admin to fix any bugs/errors. The site should not crash any user's device. We look to add a cue to ensure that no user is kicked from the site in case of too many users, and there is an orderly line.

3.5.3 Availability

application and web browser available, should be available on as close to 100% of browser applications as possible
– (browsers: Google, Safari, Firefox, Bing – research more browsers that are commonly used and can be necessary to achieve ~100%)

3.5.4 Security

Secured checkout, and basket/cart available for the user to add their ticket to. Can use 3rd party security measures – such as secured paypal checkout. Otherwise data can be encrypted and stored in a cloud database. Also authorizations such as reCaptcha should be involved to ensure identities and human users.

3.5.5 Maintainability

The application should have an automatic check for updates as well as automatically administer them across all platforms (live editing)

- Has routine checkup dates (ex. every 2 weeks required to check on functionality and software versions)
- Regular updates to the website in order to tweak minor bugs and keep the website modern and maintained.

3.5.6 Portability

Website will be designed on macOS, but will be compatible with all operating systems.

3.6 Inverse Requirements

Website will have no more than 1000 users at a given time, if so, an error message occurs and the queue will begin forming.
Website will not sell more than 20 tickets for a given movie, if exceeded will portray a sold out message.

Section 1-3 conclusion…

The above SRS describes the baseline needed to start an online ticketing system. You will find, in this document, information provided for; the creation, functionality, and the platforms needed to construct the site. Section 1 explains more thoroughly the contents of the SRS sections 2 and 3. Section 2 describes in much more detail, the functions of the site, and an overview of all the site operations. Finally, Section 3 explains more dynamics of the site, specifically, what it can and cannot do and what outside information/connections are allowing it to successfully run smoothly and efficiently.

3.7 Design Constraints

Ticket hub will run into a few constraints imposed by other standards, these standards include; company policies and hardware limitations. Below are constraints listed for each of these standards.

Company policies:

- Privacy Policy: this policy ensures that the privacy of the user remains intact, that no credit card information is at risk of obstruction and personal information should remain private.
- Terms of use: When users access the site, it should be noted that their information should be truthful and accurate. The password information should not be shared and is in the hands of the user, it is not the responsibility of the admin. Ensure good user conduct, there should be no abuse of the site. In any of these cases there should be a user ban for a predetermined time.
- ticket policy: return policy for tickets should be declared and rules should be strictly followed by users and enforced by administration. Simple ticket rules should ensure there is only one ticket given out for each seat and a maximum number of tickets should be given

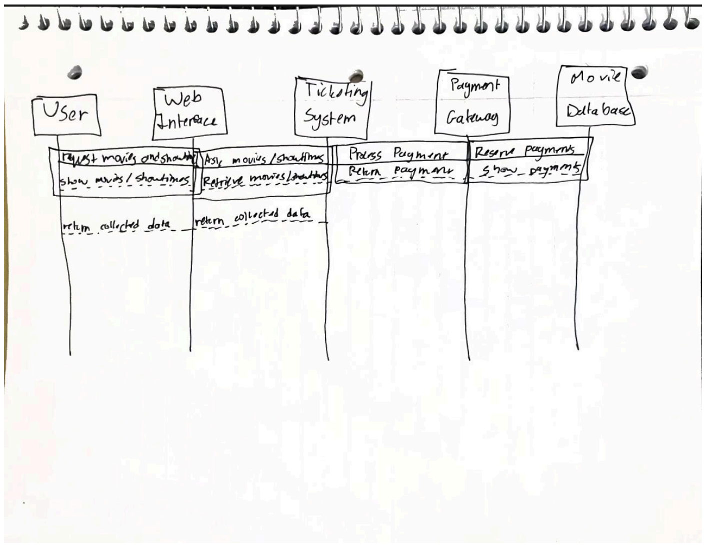
Hardware Limitations:

Hardware limitations often depend on the user and admin devices. The site should not overwhelm a user's device/cpu and should be scaled to adjust to any different model/brand/company's device. Software should be formatted to fit any screen size/ be available in different formats for phone vs computers. Hardware constraints during production period/admin accessing, may include; memory storage space, internet connection, battery life, and heat management. One major hardware restriction for both admin and users is wifi/connectivity. The user should be connected to the internet to access the site.

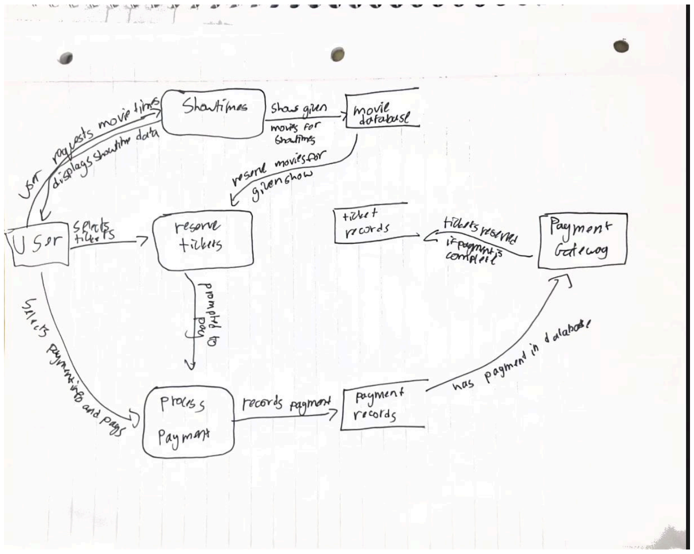
4. Analysis Models

…

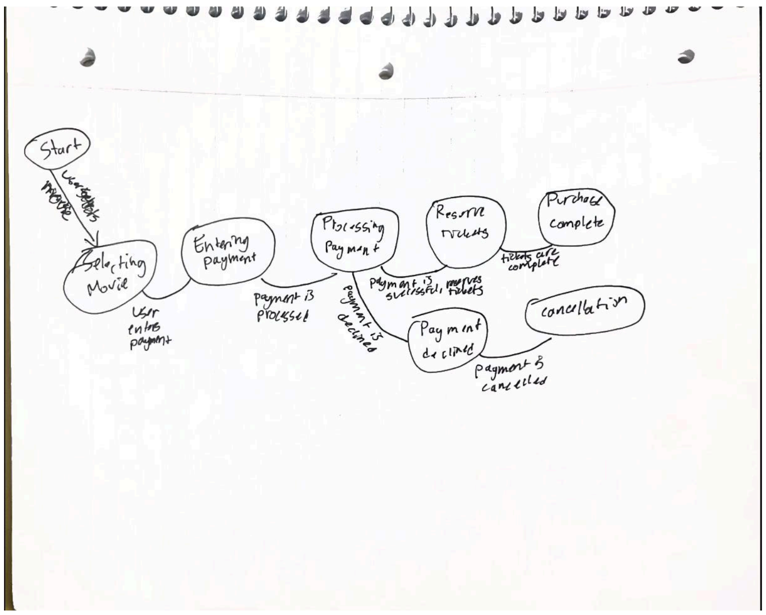
4.1 Sequence Diagrams



4.3 Data Flow Diagrams (DFD)



4.2 State-Transition Diagrams (STD)

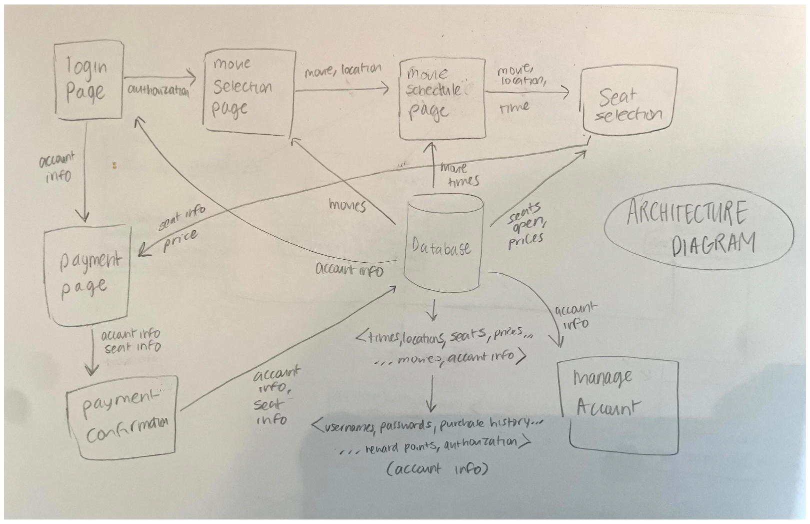


Brief Description of System:

This Software Design System has been established in order to regulate and establish architectural, visual and working components of our application, TicketHub. Below you will see and read about the data components involved, as well classes, attributes, functions and surrounding specifications needed in order to help us efficiently design and bring to life our software surrounding movie ticket purchasing.

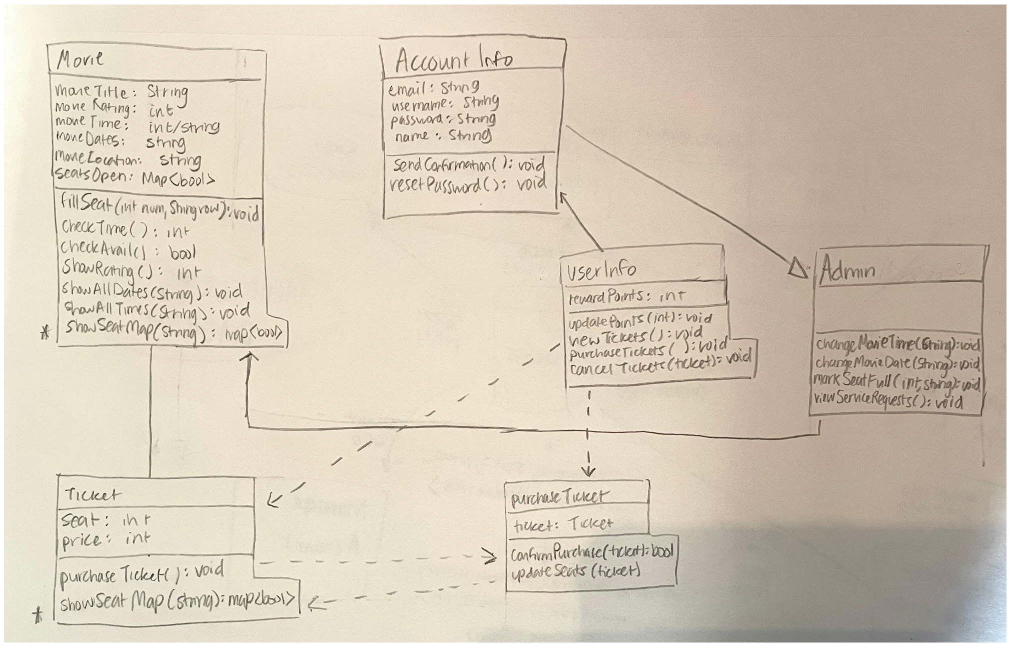
Software Architecture Overview

Software Architecture Diagram:



UML Class Diagram:

Ticket Hub



Classes:

The classes consist of *Movie, Ticket, PurchaseTicket, UserInfo, Admin*, and *AccountInfo*

Attributes:

Movie – movie consists of movieTitle, movieRating, movieTime, movieDates, movieLocation, and seatsOpen. The title, date and location and time will all be held and stored as Strings within the class, while the rating will be an integer, and finally the seats open will be stored within a map where boolean values will determine each individual seats availability.

Ticket – ticket consists of an assigned seat value, and price for the ticket, which will be two attributes stored as integers within the class.

PurchaseTicket – the class purchaseTicket has the attribute of ticket, which is an object (ticket) of the Ticket class.

UserInfo – userInfo will have rewardPoints, where the number of points gained on their account will be stored, each being unique to the individual account, stored as an integer.

Admin – since the Admin class inherits the attributes of the class AccountInfo, it does not have any other components outside of those it inherits. (listed below)

AccountInfo – this class consists of the attributes of the user's email, the accounts username and password – as well as the name of the accounts owner – which will all be stored as Strings.

Operations:

Movie – within the movie class, you will find the operations fillSeat, checkTime, checkAvail, showRating, showAllDates, showAllTimes, showSeatMap. With fillSeat, between the ticket class, and purchaseTicket, when a customer is to purchase a ticket the information will update seatAvailability() (function of purchaseTicket), which will update the ticket class' showSeatMap, finally sent over to the movie class where it will specifically then mark the seat for that movie at the customer's specific time and location - receiving the values of seat number (int) and row (String) in order to do so. Next, there is checkTime() and checkAvail() which when called will print all available times for the chosen movie, and whether the showtime is still available for more viewers. For showRating allDates and allTimes, the method will print the extracted rating of the movie, the available dates for the movie to be shown, and the times at which it will be shown. Finally, showSeatMap() will allow the customer to view which seats are available visually rather than by numbers or a listing, in order to see if the movie that they have chosen has their ideal seating arrangement for purchase. To see dates, times and the seat map however it must be mentioned that the method will need to receive the movie title to provide the appropriate information.

Ticket – for the ticket class, there will be two operations, being purchaseTicket(), and the inherited method of showSeatMap() which will be used to validate whether purchasing a ticket for the client's specified seat is allowed. purchaseTicket will first call showSeatMap() where the parameter of which movie to show the seat map for is accepted. Once viewing the seat map, the user will be able to choose a seat to then have purchaseTicket() called on.

PurchaseTicket – confirmPurchase() will receive the parameter of the object ticket, which will hold the seat number and price (inherited from Ticket) and will return a confirmation if successfully executed (bool), and if else (false) will let the customer know the purchase did not go through. For updateSeats(), the same ticket will then be passed to this method if the purchase confirmation is successful, in order to update the seat map which is essential to the functionality of both Movie and Ticket.

AccountInfo – account info will need to send a confirmation that the account has been logged into, leading to sendConfirmation() operation, as well as the potential need for a password reset, which is why resetPassword() function will clear the password and allow the user to enter and save a new one to their specific account.

UserInfo – for userInfo, since it already holds all the inherited information from account info regarding the general necessities of an account, this class instead has the specific details of the customer, leading for the need of the functions updatePoints(), viewTickets(), cancelTicket(), and purchaseTickets(). When a ticket purchase goes through, the user's points will be updated, leading to update points adding the value to the existing amount. View tickets will allow the user to view any tickets they've already purchased, and cancel tickets will cancel any, given that it is given the parameter of which ticket to cancel. Finally, the user will have the option available to purchase more tickets if needed, where the function purchaseTickets() will send them to purchaseTicket in order to confirm the sale.

Admin – the Admin will need the ability to override certain things, update information – yet will also have an account, leading it to inherit the attributes of an account from AccountInfo. The admin will be able to change the available movie times with changeMovieTime(), with the parameter of the movie to change being received (as a String) and it will also be able to do the same with the movie date. Next, the admin will be able to manually mark a seat as full in case a ticket is purchased at the theater in person, or any other need to do so. This leads the method markSeatFull() to need to receive the seat number (int) and the row (String). Lastly, the administrator will be able to view any submitted service requests about the website/application using the parameterless method viewServicerequests().

Development plan and timeline

*Partitioning of tasks:*To assign tasks to the different teams working on this project we must first divide the work and assign tasks to the proper teams. We want to ensure that the work is divided evenly enough that no single team member is overloaded with responsibilities. Find the list of responsibilities for each team and their member counts in order for an efficient product production

Team member responsibilities:

Web developers(30 members)-The web developers are the engineers responsible for the coding and main construction of the website. After receiving instructions from the visual design teams and the analysts, they do the primary block of work. This team is the largest because the work is the most time consuming, the developers will be working on a software level and will implement code to ensure the quality of the new site. They are also in charge of debugging and fixing any errors during the testing process. This team can reference the srs, specifically section 3 as a guide for the construction of the site. Following the srs should ensure all requirements/specifications are met and properly executed throughout the software development side of the project.

Security(5 members)- The security team should also be well versed in coding languages as well as software security. The security team is responsible for the part of the site that deals with payment methods(such as a direct payment to certain banking systems). They should be able to ensure the safety of a user's card information, as well as the safety of a user's device. This team should be in charge of the firewalls that prevent viruses from taking over the site on any administration or users device. The security team's main purpose is to ensure reliability of the site.

Visuals Team(4 members)-The visuals team are just what it sounds like, they are in charge of designing the website's graphics. They create templates/layouts of the site that should be displayed on the users devices, they should take into account the sizing modifications for different devices. The work done here should be handed off to the web developers to execute.

Analyst(1 member)-It is the analyst's job to ensure that both client and company needs are met. The analyst should work closely on the front end of the project, ensuring that the guidelines for the site will satisfy the customer and follow the company policies.

Test Plans:

Test Plan 1-

Test plan 1 describes the software attributes of our new website, as well as its limitations. This includes almost any of the work done by the site to accommodate users/administrators.

Verification:

- The Design:

  -In order to execute this test plan, test cases should be used to verify the accuracy of the site and its functionality. We plan to use multiple cases, described more thoroughly in the testCase excel sheet linked below, in order to identify any potential errors in the system code/program. In order to execute the test cases we will hire another team of programmers/developers to access the site and run tests on each specific method. If a specific test case fails, they should re-write the method or fix what is causing the error so that the site runs as intended.
- Test Strategies:

  -Unit testing- test individual components of the software in search of any errors.

  -Integration testing- test the software together as one and bugfix.

  -Performance/stress testing- test the software on a high amount of user capacity to see how the performance varies.

  -Security testing- ensure the software is secure and adheres to all legal precedents.
- Vectors:

  - Login functionality- test the user’s login process and make sure their username and password is secure.
  - Product search- test that the movie and time selection are functional
  - Checkout test- make sure the user receives their tickets after purchasing

Test Plan 2-

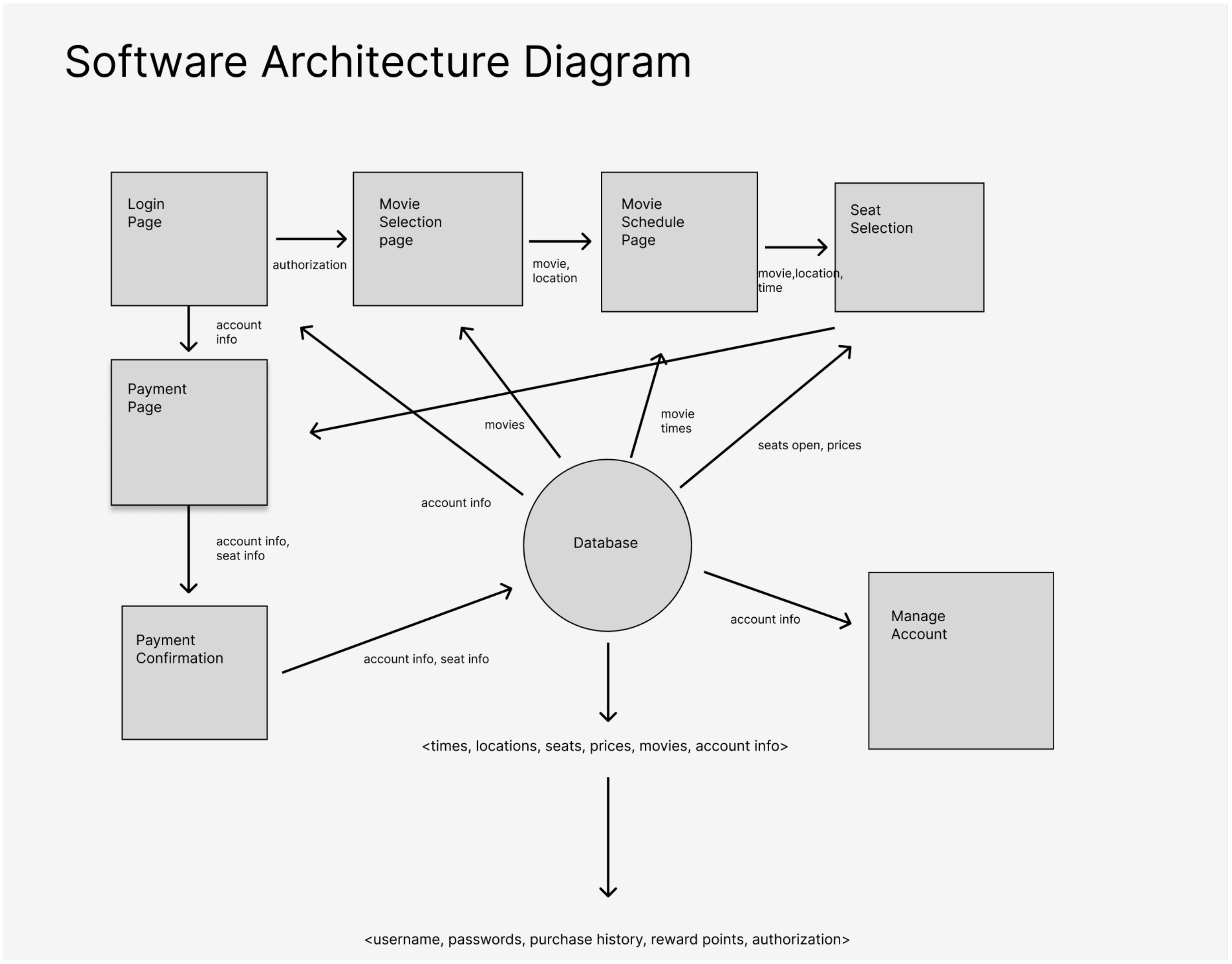
Test plan 2 will test the hardware limitations of the users. These tests will include any formatting or layout changes, processing speed, and any other limitations depending on the hardware the user is using.

Verification:

- The Design:
    - Specification- the device must be required to have stable internet connection as well as access to the internet.
- Test Strategies:
    - Functional testing- test normal functionality on each device, and note any differences.
    - Stress testing- test the functionality of each device in an extreme scenario, for example, while the device is running hot or overloading information and note the performance.
    - Performance Testing- test the performance of the website and record how long it takes to get movie tickets and compare the times.
- Vectors:
    - Zero input test: test to see if 0 + 0 = 0 to test the adder on each device.
    - Max value test: add MAX_VALUE + MAX_VALUE to see how each device handles the MAX_VALUE considering the MAX_VALUE varies per device.
    - Carry bit test- add numbers with various carries, to see if all bit operations are functional and if there are any variations between devices.
    - Boundary Test- test numbers both inside and outside of each hardware’s boundaries and see how the results vary, this is especially important as these variations could lead to a change in performance between the devices.

Link to Excel Sheet with Test Cases:

https://sdsuedu-my.sharepoint.com/:x:/g/personal/agrevin0627_sdsu_edu/ESAa9--fsRtKvrv17VC_aI0BqfmxSE_omE1ClBzu3nH4Ew?e=tasmpF



Data Management Strategy:

Data Management Strategy: SQL. Our Strategy follows the structure of a single database. The database is large enough to provide accessible information to both the users of the base and the administrators. This database is somewhat large as it is all being contained in one base, this means that we will need to include many objects in order to maintain readability and organization (these can be found below in the Logical Data Division Section). However, a single database provides data independence, and it simplifies the data application maintenance, meaning that the database can manage concurrent data efficiently.

Database Count: This site creation project should utilize one large database. Other database formatting often utilizes multiple databases to separate what the user has access to vs what the project creators have access to. Our idea of having a single database allows for the project information to be completely available to both users and administrators. The database should be available to users for viewing and to the engineers as editable.

Ticket Hub

Logical Data Division: Due to the database withholding mainly the same types of information for just different types of movies, it made the most sense to keep all the data in one base. This way many users can access the database effectively and easily, a function that is necessary in order for multiple employees to update seat and movie information. However, this database will be sectioned into parts as follows: 5 parts, (1)cost, (2)models, (3)indexing, (4)security, and (5)Maintenance. Further descriptions of each section below:

1:Cost: Cost is hidden in the database, covered up by security codes from the public. Any pricing situations are recorded in the database, this includes pricing for both hardware and software used, as well as the price negotiated with site publishers and everyone involved in the publishing of the site.

2:Modeling: virtual diagrams of seat maps for each individual theater, and different rooms will be stored here, in order to be assigned for movie times, and updated accordingly by employees as seats fill up.

3:Indexing:The Indexing divides the database into accessible sections. It makes scanning through the lengthy base easier for both users and administrators. Indexing can divide the project into the parts that are explained above in the software architecture diagram. The diagram has 7 sections as follows; seat selection, movie schedule page, movie selection page, login page, payment page, payment confirmation, and managed account. Index separates these 7 sections and allows an easier viewing of the database. By indexing, we make the job of maintenance much easier, as well as making the viewability of the database easier and more pleasant for users.

4:Security:Security is in charge of ensuring that any payment information is hidden from the public, a risk with the single database is hiding information from users that should not be made public. Security ensures that all affairs decided in the cost section of the database are hidden. Security is also in charge of establishing user authentication vs admin authentication.

5:Maintenance: This section of the database is accessible with a decipherable key that is protected by the security section. Only the administrators of the project have access to the key code, this code allows them to edit and update anything on the public database

Alternatives: The other alternative to a single database approach is to split the data into two bases, a front-end base, and a back-end. This idea would just help to split data for a different organizational approach, where things like interest reports, site traction and rewards points could possibly live- and the back-end to take care of data tables which keep track of seats and everything which pertains to a movie (seat, price, availability, time, etc). Once again though, logically we went with the singular database idea due to the majority of the data being stored for our site being mainly what we described the ‘back-end’ holding, and because it also conveniently goes to keep acquisition costs lower.

Github Link:

Amelia's Link to Github

Hannah's Link to Github

Elliot's Link to Github

****END****