Watchalong

Software Requirements Specification

6.0

3/28/2024

Group 11
Manan Shukla
Dylan Rambo
Micah Miller

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

Revision History

Date	Description	Author	Comments
2/1/2024	Version 1.0	Group 11	First Revision
2/7/2024	Version 2.0	Group 11	
2/11/2024	Version 3.0	Group 11	
2/15/2022	Version 4.0	Group 11	Final Requirements Revision
3/14/2024	Version 5.0	Group 11	Added SDS and Test Plan
3/28/2024	Version 6.0	Group 11	Added Data Management Strategy

Document Approval

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

Signature	Printed Name	Title	Date
	Group 11	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

The goal of this document is to draft, outline, analyze, and give insight into the entire **Watchalong Theater Ticketing software system** by defining and outlining requirements needed to create a fully functional software system. This document also focuses on the finer details of creating the software system through its specific requirements and performance ability.

1.1 Purpose

This document is written with the intent to instruct software engineers on the parameters of this project which is a movie ticketing system. This document will provide information of the overall project goals and objectives along with more technical details of the intended functionality and features of the software. It will help future developers maintain and update the existing software and keep the project in line with its core values and goals.

1.2 Scope

The product being produced is a movie ticketing system. This product shall ask the user to input a city, from there they will select a nearby theater, then they can view movie listings that are currently being played for the next few weeks, after selecting a movie they will be able to choose a time the movie is playing, then finally they will be taken to a secure payment page where they can pay for the ticket and print out the details or save a QR code. To put it in brief terms it will let you select a location, movie, time, and allow you to purchase tickets online.

1.3 Definitions, Acronyms, and Abbreviations

Definition/Acronym/Abbreviation	Full Name
API	Application Programming Interface
IT	Information Technology
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
CCV	Card Code Verification

1.4 References

IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

1.5 Overview

The rest of this document will provide a general description, the functional requirements, and analysis of the product's functionality. Section 2 will go over the product's general description including its general functions, user characteristics, constraints, assumptions, and dependencies. Section 3 will describe the external interface requirements, functional requirements, use cases, non-functional requirements, and other requirements.

2. General Description

This section will provide a general description of the product including its relation and similarity with other products, its basic functions, user characteristics, general constraints, and assumptions that are made whilst creating these requirements. This product will be a theater ticketing web app where customers can buy movie tickets and review movies themselves. Administrators from movie theaters can post their showings into the system as well.

2.1 Product Perspective

This product will be in competition with many other movie theater ticket sites. It will need to function well enough to not push customers to competing products. In addition it will need to integrate with other non movie related applications such as a payment method which may require an API to work with banking apps. It will need to interact with email applications to send confirmation emails and other emails such as password resets.

2.2 Product Functions

The application will have numerous functions all of which can be utilized by the user. These functions include, a movie search function, a theater search function, seating assignment, payment menu, review database, and account system. Users will be able to search directly for a movie through a search bar or with filters to find movies of their liking. Similarly, users will be able to search for theaters near them using their zip code or location. Once a user is able to pick out a movie of their choosing, they can reserve their seat using a seat map with tickets that are still available. An option to add a ticket to a shopping cart will bring up a payment menu. Users will also be able to leave reviews on movies they watched and the theater that they watched the movie in. Users can use credentials to sign in to their account in order to buy tickets. Their history of tickets bought, saved payment methods, and reviews will be stored in a database.

2.3 User Characteristics

- Customers: These are the primary users of the application that the software will be intended for. In general users will select a location, browse movies, select a time and make a payment. However there will be various types of customers.
    - Regular customers will be the users who will be making accounts and leaving reviews. This will be beneficial as they will be able to save payment information and interact with the website in more detail.
    - Guest users will be the users who rarely watch movies and would rather use the website with as little commitment as possible. They will be able to purchase tickets and browse reviews but will not be able to leave reviews themselves. Their payment will be a one time transaction and be deleted afterwards.
- Theater Staff: These users are in a more observational role. They will be primarily using the website to monitor capacities, ensure the authenticity of a customer's purchase, also they will be able to upload their movie information to the database for our website.
- Customer Service: These users will be the one performing separate administrative tasks on the account side. They will handle payment issues, account problems, and refunds.
- IT support: These users will handle database issues such as errors in uploading movie listings or problems in the payment system or capacity monitoring system.

2.4 General Constraints

The site will need to be user friendly which may restrict the features. The UI will need to be straightforward or users may get confused and frustrated. This may mean not adding features that work but are unnecessary. The many APIs that will need to be in use to make the app work may also be constraints on what the site can do. The site needs to not be too strenuous to the point where it runs slowly on the users device.

2.5 Assumptions and Dependencies

This product will depend on the user using macOS or Windows OS as their operating system. Any other operating system would possibly change hardware requirements and/or software requirements. This product will also depend on the user having a stable internet connection. Without an internet connection, the user will not be able to access the product. It is also assumed that users will manually enter their information into the system when prompted.

3. Specific Requirements

This section will go over all of the requirements necessary to create a fully functioning software for theater ticketing. These requirements will go into detail about external interfaces, functional requirements, use cases, classes/objects, non-functional requirements, and inverse requirements.

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interface for the software should be compatible with the current popular internet browsers such as Google Chrome, Microsoft Edge, Mozilla Firefox, Opera GX, and Safari. It shall be user friendly for both customers and administrators of the product. There will be several pages to be implemented in either HTML, CSS, or JavaScript.

3.1.2 Hardware Interfaces

The hardware that is necessary for the usage of this software needs to be able to access the internet to be able to view the database and input their selections and payment information into the server. Along with an internet connection a user will also need basic computer peripherals such as a mouse and keyboard to input their data and selections. If a user is accessing the website on their phone from a mobile browser then they will need a touchscreen.

3.1.3 Software Interfaces

The website will not be too resource intensive. All that is necessary is a reasonably up to date web browser.

3.1.4 Communications Interfaces

After purchasing the ticket the website will communicate with theater servers and update the count of occupied seats along with the seats that are claimed.

3.2 Functional Requirements

3.2.1 Account Authentication/ User Sign-In

3.2.1.1 Introduction

The software will need to have an account authentication functionality where user credentials can be confirmed when signing in. If a user wishes to create an account they can do so on the same page. Users can not be signed into the same account on multiple devices.

3.2.1.2 Inputs

The user will provide an input through their device which will consist of an alphanumeric username and a password with the eligible characters. The inputs will be entered within two fields on the login screen.

3.2.1.3 Processing

The software shall process the user-provided input and authenticate if the inputs match pre-existing values in the database. Depending on whether the inputs match or not, the system will display output to the user to give feedback on the success of the authentication. If successful, the user will have a message displayed to them and will be redirected to their account page. If not successful, the user will be prompted to try again.

3.2.1.4 Outputs
There are two possible outputs after the user's inputs are processed. The system shall display either a "Login Successful" message when successful, or a message containing "Incorrect username and/or password".

3.2.1.5 Error Handling
There are common errors that can take place when the user is required to input data. One error handling situation would be when a user is at fault and enters something like an incorrect password. In that case the software would need to recognize that the user input is wrong and display the current error message telling the user what they did wrong.

3.2.2 Movie Scroller

3.2.2.1 Introduction
One page of the system will have a list of movies that are currently available. The user will be able to scroll through a list of movies and is able to click on them to find out more information such as times, theater locations, and a synopsis of the movie.

3.2.2.2 Inputs

The user will be using a mouse or a touchscreen to scroll through the available movies and input from a click of a mouse or a tap from a touch screen.

3.2.2.3 Processing
The software should actively be responding to the user's input in order to scroll down the page or scroll up. It would also need to redirect the user to a new page once something is clicked on.

3.2.2.4 Outputs
The software will output the pages that need to be displayed based on the user's inputs.

3.2.1.5 Error Handling
The system should handle any instances where a user tries to access a page for a movie that is no longer available by displaying a message that says "Movie no longer available".

3.2.3 Search Operation

3.2.3.1 Introduction
This is where the user would navigate to the search bar and enter a movie title.

3.2.3.2 Inputs
The user will enter the search input using a keyboard.

3.2.3.3 Processing
The software will need to search the database using the provided search query.

3.2.3.4 Outputs
The results will output in a user friendly format that shows movie titles matching the result and an image of the movie.

3.2.3.5 Error Handling
If the user provides something that matches nothing in the database, a message should appear telling the user the movie wasn't found.

3.2.4 Seat Map

3.2.4.1 Introduction
The software shall have a page for a seat map once a user clicks on an option for available seats for a particular movie. Each seat that is not already sold out will be available for the user to click on. The user can only buy 15 tickets for a single movie at a time. Tickets can be bought in a time window right after a showing is posted until 20 minutes before the movie starts. Prices can be changed by administrators to include discounts.

3.2.4.2 Inputs
Input would be the user clicking with their mouse or finger to select a seat of their choosing.

3.2.4.3 Processing
The software would take the seat chosen by the user and interface with the database of seats available which would contain information about prices and quantities. It would need to confirm whether or not the amount of tickets requested is below or equal to 15. Similarly, it would need to confirm that the user is buying tickets within the proper time window.

3.2.4.4 Outputs
The site will display the availability of the ticket and information such as the price.

3.2.4.5 Error Handling
If the seat becomes unavailable, the user should not be able to buy it and should be notified it's taken.

3.2.5 Reviews

3.2.5.1 Introduction
The user will be able to read and leave reviews on both movies and theater facilities.

3.2.5.2 Inputs
The input would be typing on the keyboard. Inputs will consist of integer values ranging from 0-10, and an optional comment in the form of a string variable.

3.2.5.3 Processing

The review submitted by the user will be sent and stored in the review database as the different data types stated above.

3.2.5.4 Outputs
The output would be the review appearing on the user end.

3.2.5.5 Error Handling
One error would be if something goes wrong causing the review to not show up right away. The user might try to hit post again. In this case the software should delete any duplicate reviews and try to terminate the operation before a second one is posted to allow things to load. If the review still doesn't appear, then the post button would function allowing the user to reattempt submitting the review.

3.2.6 Payment Menu

3.2.6.1 Introduction
This would display a page with premade fields showing what the user needs to provide.

3.2.6.2 Inputs
Inputs would be the payment info such as the credit card number, ccv, and expiration date.

3.2.6.3 Processing
The software would use an API to verify that all the info entered is valid. Once valid the system shall mark the seats selected as sold out.

3.2.6.4 Outputs
The output would be a message telling the user whether or not the payment info was received and correct. Tickets will be emailed to the user with the email address they provided when they made their account.

3.2.6.5 Error Handling
The software would need to verify things like the ccv. If the ccv doesn't match what is expected, the software would need to inform the user something incorrect was entered.

3.2.7 Account Manipulation

3.2.7.1 Introduction
This page will allow users of different ranks to edit accounts according to their account ranking.

3.2.7.2 Inputs
The user will be able to edit their own account information. IT users will be able to edit other people's information and use this page to view purchased tickets and issue refunds as well.

3.2.7.3 Processing

The information will be uploaded to the database and overwrite existing information.

3.2.7.4 Outputs
The output would display the same page with the updated information and indicate to the user that the data was updated successfully

3.2.7.5 Error Handling
If the system faces issues accessing the database and writing information then the user will be notified and prompted to try again in a few minutes.

3.2.8 Movie Information Manipulation

3.2.8.1 Introduction
This function will let theater administrators create, view, and manipulate existing information about a movie.

3.2.8.2 Inputs
A movie administrator will input information about the movie ranging from title, description, actors, viewing times, seat capacity, and arrangement.

3.2.8.3 Processing
This data will immediately be uploaded to the database for all users to see.

3.2.8.4 Outputs
The result of this function will be the creation or modification of a movie class object.

3.2.8.5 Error Handling
The program will check for duplicates and notify the administrator if there are any issues before updating the database. The program will also ensure that all sections of the movie information page are filled out before allowing the user to upload into the database.

3.2.9 Theater Information Manipulation

3.2.9.1 Introduction
This function will let theater administrators create, view, and manipulate existing information about a theater.

3.2.9.2 Inputs
A movie administrator will input information about the theater ranging from name, address, amenities, and pictures of the establishment.

3.2.9.3 Processing
This data will immediately be uploaded to the database for all users to see.

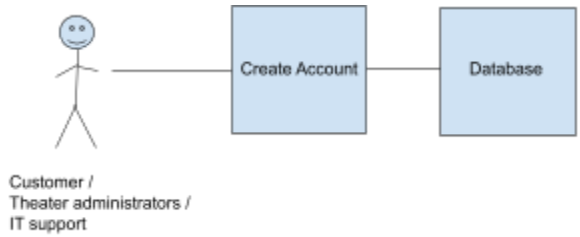
3.2.9.4 Outputs
The result of this function will be the creation or modification of a theater class object.

3.2.9.5 Error Handling
If the system faces issues accessing the database and writing information then the user will be notified and prompted to try again in a few minutes. The program will also ensure that all sections of the theater information page are filled out before allowing the user to upload into the database.

3.3 Use Cases

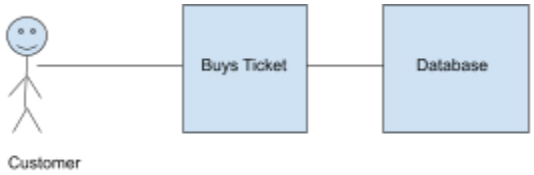
3.3.1 Use Case #1

User Creates Account
Actors: User
Flow of events:
The user searches for the website on a browser or types in the url. Then they are brought to a home page and there will be a separate button to login. Here they can choose to login or create an account. From there they will fill in the account information such as their name, phone number, and a specific safe password. Theater administrators and IT employees will be given an option to enter a passphrase that is active for a limited amount of time to take them to a page where they can enter their employee ID and create an account with more user rights. Then finally the information will be uploaded to the database and create their account.



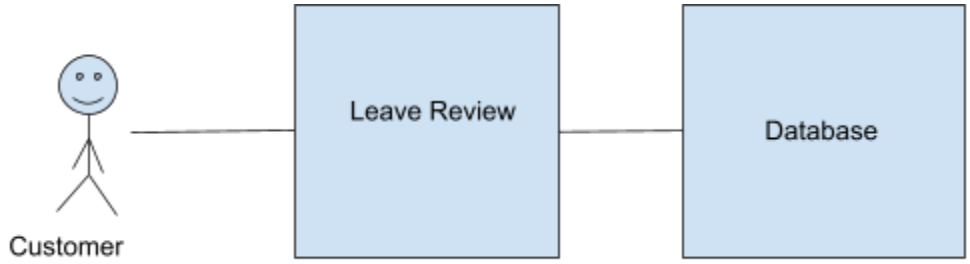
3.3.2 Use Case #2

User Buys Tickets
Actors: User
Flow of events:
User navigates to the ticketing website by looking it up or entering the url. Then if they have an account they would navigate to the login page and enter their username and password. If they are a guest they would hit the guest button and go straight to entering a city zip code, selecting a movie, a time, and then a seat. From there if they are a guest they would need to enter the payment info which will not be saved. A user with an account would still select a movie and tickets but then would be able to use their saved payment info. Then the ticket information will be updated in the database along with the seating information. Lastly they will be taken to a confirmation page showing their ticket information and transaction information and they will be sent an email with their ticket.



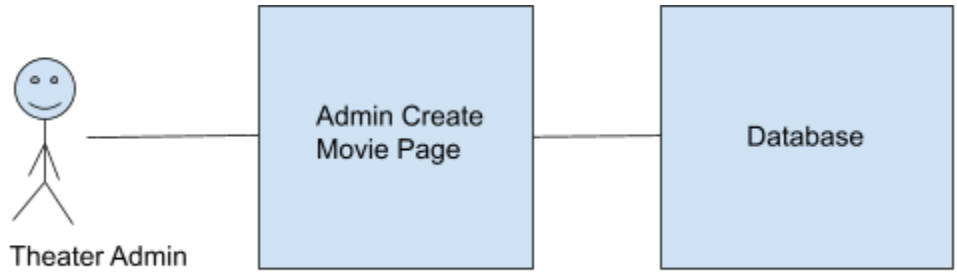
3.3.3 Use Case #3

User Leaves Review For Movie
Actors: User
Flow of events: The user would need to navigate to the login page and login. From there they could search for a movie they want to leave a review for. Instead of buying a ticket, they'd hit the leave a review button and give the movie rating and description. After hitting submit the information is updated in the database and associated with the movie object class. The review would be saved and show up in the movies ratings section where people can read reviews before seeing a movie.



3.3.4 Use Case #4

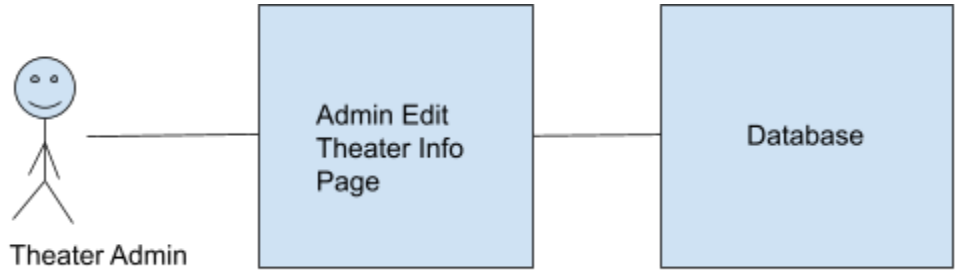
Creating Movie Listing
Actors: Theater Admin
Flow of event: The admin from a particular theater will be able to post their movies on an admin page that is only available to them on admin accounts. They will be able to fill in fields such as title, description, actors, viewing times, seat capacity, and arrangement. These fields will be uploaded to the database so that the program can process it and output it to users.



3.3.5 Use Case #5

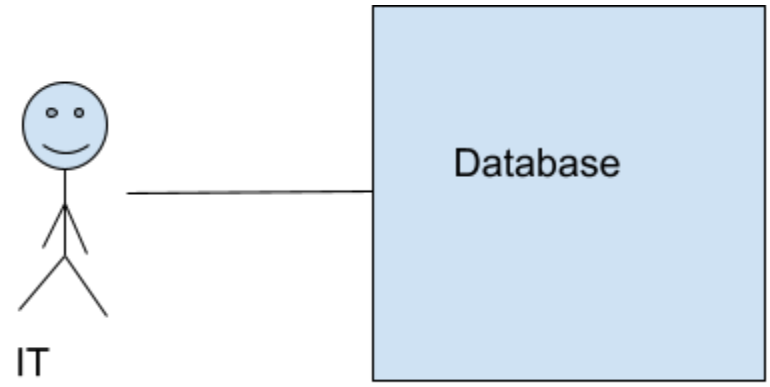
Creating Theater Information
Actors: Theater Admin
Flow of event: Admins will also have access to an admin-only page where they will be able to add information such as theater name, address, amenities, and pictures of the theater. Each field

will require input from the admin. Once they are ready to submit, the data will be uploaded to the database and be output onto the website for the customer to see.



3.3.6 Use Case #6

Issuing Refund
Actors: I.T.
The I.T. user must login with the credentials. From there they can access the database of users. Then they can search for a user using their userID. Then they verify a ticket object associated with their account and they can see if the movie has already passed or not. If it hasn't, the IT user can remove the ticket object, invalidate the purchase, and manually issue a refund to their credit card.



3.4 Classes / Objects

3.4.1 Guest Class

3.4.1.1 Attributes

payment information: a temporary subclass that will hold the subclass payment information. This subclass is described in more detail below.
ticket: a temporary subclass that will hold ticket information. This subclass is described in more detail below.
temporary id: gives the guest a temporary id.

3.4.1.2 Functions

create account: allow the guest to create an account and upgrade the user status
search movie: search the database for a movie
buy ticket: purchase ticket that gets them a seat for a movie at a specific time and theater. Creates an object of the ticket class

input payment information: creates the payment information class. If the guest user does not upgrade the account then data will be wiped upon ticket purchase.

3.4.2 Customer Class

3.4.2.1 Attributes

name: holds the name of the account holder
tickets purchased: has subclasses that hold the ticket information
reviews: has subclasses that hold review information
permanent id: gives the user an id number associated with the account.

3.4.2.2 Functions

change name: modify account name
delete account: remove all information of this account from the database
add payment information: create a payment information object with all of the details
remove payment information: remove the payment information class
search movie: search for a movie in the database
buy movie: create a ticket object and associate it with the account
create review: create a review class object to leave a review on a movie. This review is also linked to your account.

3.4.3 Theater Admin Class

3.4.3.1 Attributes

name: holds the name of the admin user
role: indicates role and permissions that they are allowed to perform
employee id: holds the id of the employee

3.4.3.2 Functions

create account: create an employee account
create movie: create a movie object
edit movie: edit an existing movie object
create theater: create a theater listing
edit theater: edit the details for a theater listing

3.4.4 IT Support Class

3.4.4.1 Attributes

name: holds the name of the admin user
role: indicates role and permissions that they are allowed to perform
employee id: holds the id of the employee

3.4.4.2 Functions

remove any any rank account: removes any account from customer to theater administrator
create any rank account: creates an object of any account class
remove ticket purchases: remove ticket objects off of people's accounts

refund purchase: reverses payments back into an account

3.4.5 Movie Class

3.4.5.1 Attributes

name: holds the name of the movie
image: holds the image of the movie
summary: holds the summary of the movie
time: holds the timing of the movies
location: holds the locations the movie is being shown at
reviews: holds review objects that pertain to that movie
seat count: holds the movie theater seat occupancy count in a 2 dimensional array.

3.4.5.2 Functions

create movie: this is a method used by other administrator classes to create movie objects.
edit movie information: this is a method used by other administrator classes to edit information on the movie objects.

3.4.6 Ticket Class

3.4.6.1 Attributes

movie: holds the name of the movie a ticket is being purchased for
price: holds the price for the specific ticket
time: holds the time for the movie
movie theater: holds objects of the movie theater subclass that hold movie information
seat: holds the seat number that is being purchased
discount: indicates if the user is a senior citizen, military or student and applies the discount accordingly.

3.4.6.2 Functions

create ticket class: creates a ticket object. This method is used by the user classes.
delete ticket: deletes ticket object. This method is used by the administrative classes.
process payment: This method is used along with the creation of the ticket. It accesses payment information and uses it to process the payment.
refund payment: This method is used by administrative classes to refund tickets that a customer calls to get removed.

3.4.7 Payment Information Class

3.4.7.1 Attributes

name: stores the name of the cardholder
card number: stores the card number
expiration date: stores the expiration date of the card
cvv: stores the card verification value
billing address: stores the billing address of the card

3.4.7.2 Functions

create payment object: used by all classes but mainly by the user classes. This method create an object storing payment information
delete payment object: removes payment information.
access payment: Secure method to access payment information when purchasing a ticket.

3.4.8 Movie Review Class

3.4.8.1 Attributes

user: holds the class for the user to be accessed through comments.
movie watched: holds the movie class for what movie was watched
theater watched: holds the class of the theater the movie was viewed at.
comment: holds the actual comment that is being written
rating: holds a number out of 10 to portray the writing.

3.4.8.2 Functions

create movie: creates a movie object. Method used by administrative classes.
remove movie: removes a movie object. Method used by administrative classes.
check seats: shows a display of the available seats. Users see this when purchasing a ticket. Administrators can see this whenever they want.
modify seats: Remove seats or reserve seats as needed.

3.4.9 Theater Class

3.4.9.1 Attributes

name: holds the name of the theater
address: holds the address
rating: holds the average rating of movies at that theater

3.4.9.2 Functions

create theater: create a theater object. used by administrative classes.
edit theater information: edit information for a theater object. used by administrative classes.

3.5 Non-Functional Requirements

3.5.1 Performance

The website performs simpler operations so under average load it will perform operations, updates, and take inputs at almost instant speeds. Some factors that would alter the average performance and slow it down would be if there is an unusually high amount of users on the website all making inputs (such as seat reservations and payments). Another situation would be if the connection on the user's side is exceptionally slow causing data that is being sent or received from the server to face some delays. Another possible issue for performance would be server side connection issues which could very rarely but are also very unlikely to occur often.

3.5.2 Reliability

The system should be constantly operating. Very rarely there will be scheduled maintenance that will be planned and stated on the website ample time in advance. In the circumstance of an error it will be made apparent to the user as soon as they submit any input to the servers. This means that every time the user submits something the program will ensure that the data is saved and uploaded to the proper database before continuing to the next page. If an error occurs the operation will terminate and the user will be alerted to try again or contact support. The operation will be terminated to ensure that no issues involving double inputs occur such as double reserving a seat or double charging the user.

3.5.3 Availability

Movies are seen at most hours of the day. This means the site needs to be functional during these important hours. Any errors during the normal times people see movies need to be handled quickly. Any maintenance that might require the site to be down will likely have to be done very early in the morning.

3.5.4 Security

This software will need to provide secure handling of personal data. This will include safely transferring and storing the data. Payment information is especially important because if the information gets into the wrong hands and does damage to the customer, the fault and consequences would lie with us. Passwords should not be weak and things like two factor authentication should be used for any non customers using the software.

3.5.5 Maintainability

This SRS and future documentation can help support the maintainability of the software. Comments should be used properly to ensure future developers will not need to redo the software and can simply build off what exists. The code should be concise to ensure it is also not confusing. Future additions should likely already be somewhat in place to ensure the hardware and software do not become limiting factors for future attempts to add to the software.

3.5.6 Portability

As the movie ticketing system is solely web-based to promote simplicity and compatibility it is almost certain to be able to run on any platform that has the previously mentioned browsers. The only features to be added for mobile usage is touch screen support and on screen keyboard support. The website should make sure on the mobile version of the webpage that the content is still reasonably viewable with the keyboard as well and that the frame of the webpage adjusts accordingly. The content of the webpage should be able to scale down in terms of width to be convenient to read on mobile devices. This implies that the webpages should be designed for single column viewing so that the width can be adjusted accordingly depending on the native resolution and dimensions of the screen.

3.6 Inverse Requirements

3.6.1
Users can only modify their accounts and their own reviews only.

3.6.2
Only theater admins can edit theater information and movie information.

3.6.3
IT can modify user accounts and have the rights of the other two users since they can manipulate the databases.

3.6.4
Payment will be stored securely so access is limited to only the user and IT.

3.6.5
Operations will spend extra time to ensure no conflicts occur. For example when seats are reserved the operation will wait a second to check against other current reservations to make sure no one is reserving the same seat. If two people try to reserve the same seat at the same time the user with the id number that comes first alpha numerically will get the seat and the other user will be told the seat is already reserved.

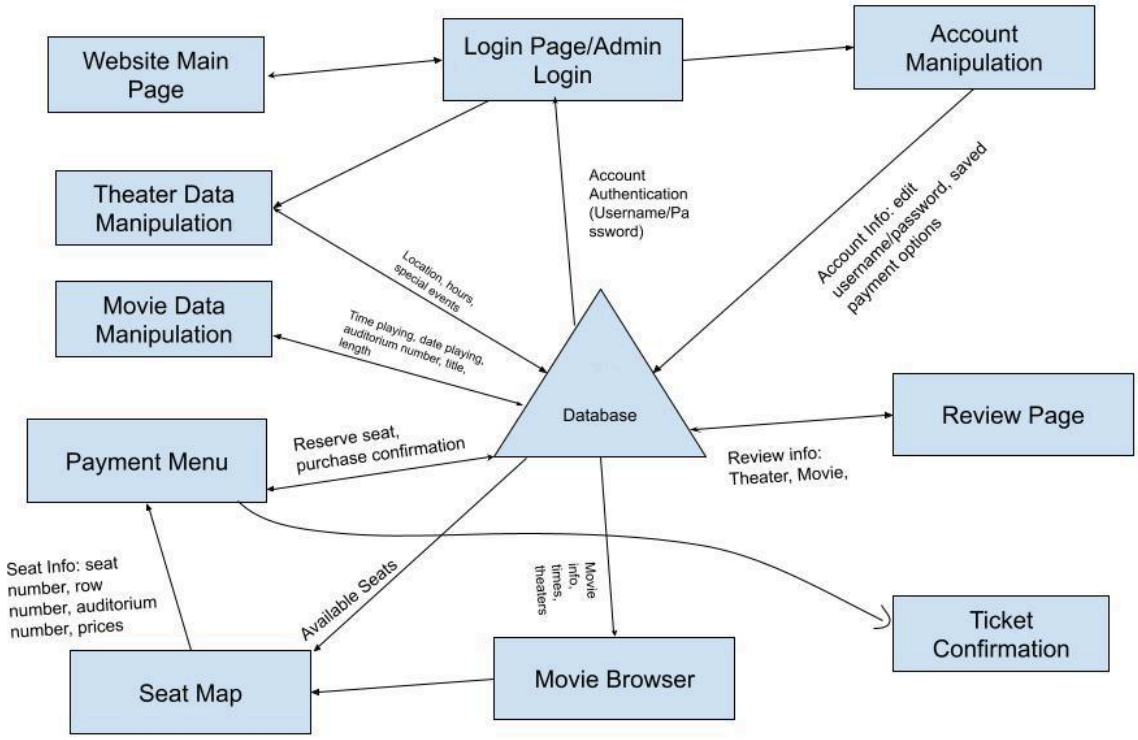
3.6.6
Only card types from within the country will be taken to prevent dealing with international payment issues.

4. System Description

The entire software system consists of multiple pages for the user to view and use, as well as a database that will be heavily utilized. These pages include a website main page, login page for regular customers and theater admins, account manipulation to edit account information, a movie browser, movie seat map, payment menu, ticket confirmation, review page, and theater/movie data manipulation for admins.

5. Software Architecture Overview

5.1 SWA Diagram

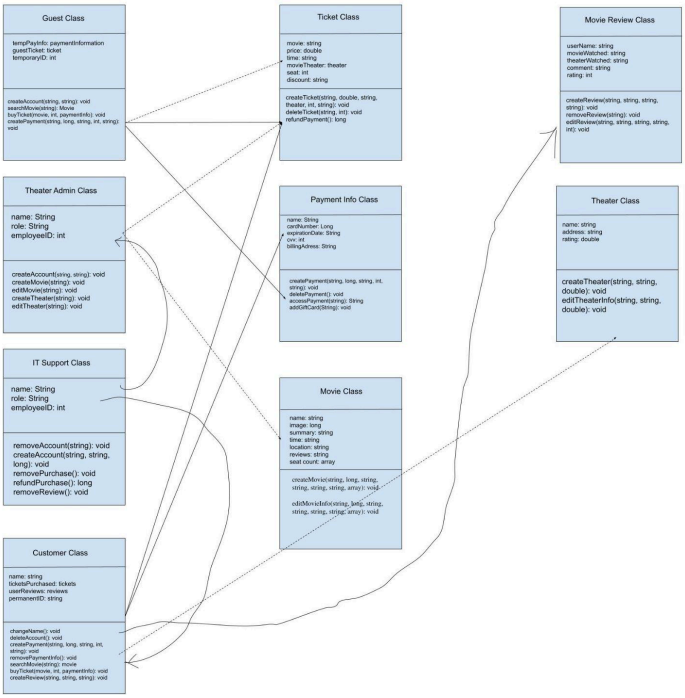


5.1.1 Description

The overall design of our software's architecture is heavily dependent on communicating with our database. The software will have 11 components that work together to bring a working theater ticketing system to a customer. The first component is the website's main page. This will mainly be a user interface meant to display to the user what the software actually does. From the main page the user will be able to navigate to the login page or admin login page depending on who is trying to access their login. Once the user enters their username and password the software will bring the associated fields from the database to compare and will authenticate the user if they match. Once successfully logged in, the user will be able to manipulate their existing account information such as their username, password, and saved payment options. This data will be updated in the database by the software. A user may wish to browse movies to watch at a local theater. In this case, the movie browser will take information from the database about available movies to watch, such as a synopsis, showing times, and theater locations. Once an option is clicked on a seat map will be displayed. The seat map will take information from the database about available seats such as seat number, row number, auditorium number, and prices for each seat. That information will be transferred to the payment menu once the user decides to buy a ticket(s). The payment menu will prompt the user for their payment information where the database will send information back to the software to check if the payment has been processed. Then the software will update the available seats in the database. Once the seats have been

confirmed to be reserved and paid for, the software will send a confirmation message to the user. Separate from the other pages for purchasing tickets is the review page. Information about past reviews will be brought from the database to the review page. Reviews made by a user will be sent to the database so that it is added to the rest of the reviews. The last two components are the theater and movie manipulation pages for admins. Both will take existing data from the database in case an admin wants to edit existing theater or movie information. New movies or theaters can also be created by admins which will be updated in the database. The information to be transferred between these pages and the database are theater location, theater hours, theater special events, movie times/dates, auditorium number, length of movie, and title.

5.2 UML Class Diagram



*updated UML Diagram as of Version 5.0(Assignment 3)

5.2.1 Description of Classes/Attributes and Functions

5.2.1.1 Guest Class

This class will allow a user to perform operations on this website without an account

5.2.1.1.1 field variables:

- Holds a variable for tempPayInfo which holds a paymentInformation object. This field variable will be holding an object of the paymentInformation class. It will be deleted once the Guest session is ended along with the rest of the class.
- The guestTicket field variable holds a ticket object. This will be created upon purchase.
- The guestClass will have a temporaryID field variable to be used in the ticketBuying operation.

5.2.1.1.2 operations:

- The createAccount operation will take in strings for name and role. This exists to give the guest an option to create a permanent account.
- The searchMovie function will take a string for the movie name and search the database for the specified movie. Then if found it will return a Movie object.
- The buyTicket operation takes in a movie class, an integer for the userID, and a string for the payment information. This will return a ticket object and an email with ticket information when successful.
- The createPayment operation creates a payment information object. It will take in a string for the name on the card, a long for the card number, a string for the expirationDate, an int for the CVV, and a string for the billingAddress. This will create a payment information object.

5.2.1.1.3 dependencies:

- The guest uses the ticketClass because it will create ticket classes associated with the account
- The guest class uses the payment class to create an object to temporarily store payment info.

5.2.1.2 Theater Admin Class

This class will allow movie theater employees to have an account mainly used to perform operations that update movie information and movie theater information

5.2.1.2.1 field variables:

- The name variable is a string holding the name of the Admin user.
- The role variable is a string that will hold the status of the employee.
- The employee ID variable is an int that will hold a unique number to identify the employee.

5.2.1.2.2 operations:

The class will have several operations.

- The createAccount function will create a theater admin object.
- The createMovie function will take in a string and create a movie object in the database. The editMovie will be able to change the movie info.
- createTheater will take in a string and create a theater object.
- editTheater will update theater info.

5.2.1.2.3 dependencies:

- The theater admin class updates the theater and movie classes.

5.2.1.3 IT Support Class

This class allows IT support staff to fix problems such as adding and removing accounts and refunding purchases.

5.2.1.3.1 field variables:

- The name variable is a string that holds the name of the IT support user.
- The role variable is a string that holds the role of the IT user.

- The employee ID variable is an int that will hold a unique number to identify the IT user.

5.2.1.3.2 operations:

- The removeAccount function will take in a string of the account to remove and will delete the object.
- The createAccount function will take in the necessary info and create an account object.
- the removePurchase function will delete a ticket object.
- The refundPurchase function returns a long of the money being refunded.
- *The removeReview function is a void function that allows IT support to remove a review*

5.2.1.3.3 dependencies:

- This class uses the Theater admin and customer classes because they have user rights to manipulate and create account information as needed.

This class also uses the ticketClass to refund purchases.

5.2.1.4 Payment Info Class

This class will take in the user payment information when they want to buy movie tickets.

5.2.1.4.1 field variables:

- Hold a string variable for name on the card
- Holds a long for the card number
- Holds a string for the expiration date
- Holds an int for CVV.
- Holds a string for a billing address.

5.2.1.4.2 operations:

- The createPayment operation is a constructor for this class that will take in a string, long, string, int, and a string for the field variables.
- deletePayment is an operation that will remove a paymentInfo object and all of its information.
- accessPayment will return a string with the requested information.
- *addPaymentGiftCard will be a void function that will take in a string to add a gift card as a payment method*

5.2.1.4.3 dependencies:

- This class is not really dependent on other classes. It is a core class that other classes rely on.

5.2.1.5 Customer Class

This class allows a general user a more permanent account to use that saves past movie information and allows the user to leave reviews.

5.2.1.5.1 field variables:

- The name variable is a string that holds the name of the customer.
- The tickets purchased variable is of type tickets.
- The user reviews variable is of type reviews.
- The permanent ID variable is a string that will identify the user.

5.2.1.5.2 operations:

- The changeName function will allow the user to update their username.
- The deleteAccount function will allow the user to delete their account.
- The createPayment function takes in the relevant banking info and creates an object.
- removePayment will delete the object.
- The searchMovie function will take in a string of the movie name and return a movie object.
- The buyTicket function takes in the move, cost, and payment info and creates a ticket object.
- The createReview function will take in strings of the review and create an object.

5.2.1.5.3 dependencies:

- This class is dependent on the Ticket class because it creates objects of the ticket class upon purpose.
- This class is dependent on the payment info class because it saves a field variable of the payment info class
- This class is dependent on the movie review class because this class creates objects of the movie review class.

5.2.1.6 Movie Class

This class is how movies will be stored in the database and will hold information about the movie

5.2.1.6.1 field variables:

- Holds a string for the name of the movie
- Holds a long to store the binary for the image of the movie cover
- Holds a string for the summary of the movie
- Holds a string for the time of the movie
- Holds a string for the location of the movie
- Holds a string for the reviews of the movie
- Holds a 2 dimensional array to represent the taken seats and the vacant seats

5.2.1.6.2 operations:

- The createMovie operation takes in a string, long, string, string, string, array to update the mentioned the variables above when creating an object
- The editMovieInfo takes in a string, long, string, string, string, array to update the mentioned variables above.

5.2.1.6.3 dependencies:

- This class is not dependent on other classes but many classes are dependent on this class.

5.2.1.7 Movie Review Class

This class will be associated with movie objects and be created by customer class objects.

5.2.1.7.1 field variables:

- The userName variable is a string that holds the name of the user leaving the review.
- The movieWatched variable is a string that holds the name of the movie being reviewed.
- The theaterWatched variable is a string that holds the theater the user saw the movie at.
- The comment variable is a string holding the review the user is leaving.
- The rating variable is an int holding the number rating.

5.2.1.7.2 operations:

- The createReview function takes in strings and creates an object of the review.
- The removeReview function deletes the instance of the review.
- *editReview is a void method and takes in values for all of the field variables in order to edit an existing review*

5.2.1.7.3 dependencies:

- This class isn't dependent on any other classes.

5.2.1.8 Ticket Class

This class will be created every time a user purchases a ticket and stores relevant information about the purchase.

5.2.1.8.1 field variables:

- The movie variable is a string holding the name of the movie that the ticket is for.
- The price variable is a double that holds the price of the ticket.
- The time variable is a string variable that holds the time of the movies showing.
- The seat variable is an int holding the seat number.
- The discount variable is a string holding the discount code if used.

5.2.1.8.2 operations:

- The createTicket function takes in the relevant ticket info and creates a ticket object with it.
- The deleteTicket function deletes objects created by the createTicket.
- The refundPayment function returns a long of the ticket price being refunded.

5.2.1.8.3 dependencies:

- This class is not dependent on other classes but other classes are dependent on it.

5.2.1.9 Theater Class

The theater class will be manipulated by movie theater employees and will store relevant information about the theater.

5.2.1.9.1 field variables:

- Holds a string to holds the name of the theater
- Holds a string for the address of the theater
- Holds a double for the rating of the movie

5.2.1.9.2 operations:

- The createTheater operation takes in a string, string, and double to populate the variables mentioned above.
- The editTheaterInfo takes in a string, string, and double to update the variables mentioned above.

5.2.1.9.3 dependencies:

- The class is not dependent on other classes but is a core class that other classes.

6. Development Plan and Timeline

There will be three teams of developers on this project.
Group 1 will be led by Manan and will handle the guest, IT support, and customer classes.
Group 2 will be led by Micah and will handle the ticket, payment info, and movie classes.
Group 3 will be led by Dylan and will handle the theater admin, movie review class, and theater class.
Each group will take 2-3 weeks to complete their classes including the implementation of the attributes and functions.
After this period each group will complete testing on their classes and will have to implement new code if needed.
This period of testing will take 10 weeks.
All three groups will reconvene and test the entire system for a period of 2 weeks.

7. Verification Test Plan

7.1 Introduction

7.1.1 Purpose

The purpose of this test plan is to verify the functionality of all aspects of the software so that the requirements can be fulfilled. Tests will be conducted to target at the unit, functional, and system levels.

7.1.2 Scope

The test plan covers the entire architecture of the system, including the website main page, login page, account manipulation, theater data manipulation, movie data manipulation, review page, movie browser, search bar, seat map, and payment menu.

7.2 Test Objectives:

The goal of the testing process is to catch any immediate problems as well as identify areas that may pose future problems. At the most fundamental level, the unit tests seek to ensure each section of the code performs its intended task which is separate from the overall functionality of the program that the user experiences. For example, one of the unit tests, tests whether or not the theater admin class properly updates the theater objects name. This type of testing can be done in the software and does not need to be done via the actual software. The next stage of testing, functional testing, will seek to ensure that the program functions well from the level of using the software as a user would without looking at the code. This is a type of black box testing. Testing should be performed by looking at all the functions a user can perform and attempting to use them as intended and even as not intended to see how the software performs. Examples include things like creating a user account and then attempting to make updates to user info such as the username and password. Success would be seen in-website. The final stage of testing would be system testing which would seek to test whether all functions of the system work in unison. This includes things such as testing whether the bank api for payments works and whether the system can handle a large amount of traffic. This stage of testing might need to be performed at both the user and coding level. For example high amounts of traffic could be simulated using test objects whereas testing whether the software works on different platforms would be performed from the UI level. At each level of testing, attempts should be made to break the software or system to ensure things will run smoothly. For example testing weird inputs to see whether the software allows it should be done. If done improperly, this could slip by and not cause any compiling errors but it could break the functionality of the system. This is why both unit and functional testing need to be done.

7.3 Test Environment

7.3.1 Hardware:

These tests will take place on mobile devices, laptops, and desktops. However among these devices of various different types will be tested that operate off different operating systems. This will be addressed in more detail in section 4.3.2. Laptops and desktops will both produce relatively the same results so main testing will be performed on a desktop computer. However extra cautionary testing will be performed on the laptop but it will be minimized to reduce time and costs.

7.3.2 Software:

Among mobile devices the main operating systems will be tested. So the website will be tested using the native browser of IOS devices, Android devices, and HarmonyOS that runs on Huawei devices. Among computers the website will be accessed on Microsoft Windows, Linux, and MacOS. Since browsers among these systems will produce relatively the same results we just need to make sure that the pages display properly among the three operating systems but main functionality testing will occur in Windows. Among browsers the main ones to be tested will be Google Chrome, Microsoft Edge, Microsoft Explorer, Firefox, Safari and Android browser. Tests will be performed within each of these browsers. One important note is that while tests will be performed on the devices, operating systems, and browsers listed above; the most in-depth and comprehensive testing will be performed on a desktop computer running a windows operating system and utilizing the google chrome browser.

7.4 Test Scenarios:
Our testing will need to be very extensive, however some of the specific testing scenarios will be described here.

Our first test case example is testing the payment menu. This is a unit test that tests whether payment info can only enter correctly such as the CCV only being an int. This can be done with a test object.

The next thing we will test is whether the createTheater function can update the database properly. This can also be done as a unit test with an object.

The edit movie test will see if the editMovieInfo function updates the movie object's field variables. This is a unit test and can be done with a theater admin object. A simple print output test with expected versus actual results could work for all the unit tests.

The final unit test example is to test whether the buyTicket Function works. This can be done within the guest and customer class and checking to see if a ticket object is created.

The changePasswordTest will be a functional test to see if from a user's perspective their password changes in the login info section. This can be done by acting as a user and seeing if the password shows up correctly.

The removePurchase_dataBase_update will be another functional test to see if an IT support user can delete a payment and the corresponding payment object gets deleted.

The movieSearchTest will be another functional test to see whether the search function works properly. This can be done by attempting every scenario a user might run into. This includes the tester trying stuff like entering a word, entering a phrase, and entering a genre. Any errors may require unit testing to resolve.
The next test is the seatAvailabilityUpdate test. This is a unit test to test the whole system functionality. When a user buys a ticket, the system needs to almost instantly update the database to show that seat as taken so that if another user is trying to buy that ticket it will show up as taken. Huge issues would arise if a ticket that no longer is available is able to be purchased.

The accountInfoUpdate test will be a system test to ensure that when information is updated in one section of the system, it is concurrently updated in other areas. For example if a user leaves a review, and then updates their username, the old review should show as left under the updated username and not the old one.

Another test example is the ChangeofOS test. This is a third system test that will ensure that the software can be used equally well on any operating system. Further testing beyond these specifics will be necessary.

We will need to test that redundancies in the database do not occur so the system will not permit two of the exact same theater to be created. Testing will involve ensuring the system compares theater addresses to make sure duplicates do not exist
We will need to test that a valid email address is required for the user to make an account. An invalid email address should give the user some sort of message to inform them their email is invalid.
Guest class users should not be able to leave a review on any movie. The system must ensure this by checking and making sure that the ID type is not of the guest type before allowing them to leave a review.

A test scenario for account creation is to verify that a user should only be able to create an admin account as long as they have a valid admin ID. This test would be a unit test to make sure that only a valid ID already in the database can be used to create an admin account.

Testing should be done to ensure that the deleteAccount method wipes the corresponding account data from the database. Holding on to unnecessary information creates security risks for the user and company. This can be done by deleting a test account object and then attempting to access that information and hopefully not finding it.

When a user selects to leave a review the system will check if the user has a purchase for a ticket of that movie in their ticket history. If they do not they will be notified that they have not watched this movie so they cannot leave a review

Removing a review from the review page also needs to be tested. This would be a unit test to verify that the data from the review was deleted from the database. This would protect the user from having a permanent review on the website.

7.5 Example Test Cases Spreadsheet

	A	B	C	D	E	F	G	H	I	J	K
1	"Watchalong" Test Cases										
2	**TestCaseId**	**Component**	**Priority**	**Description/Test Summary**	**Pre-requisites**	**Test Steps**	**Expected Result**	**Actual Result**	**Status**	**Test Executed By**	Granularity
3	PaymentMenu	Payment Info Class	P2	Verify that when a user enters in their payment information, they will be restricted to specific characters and length of values	The user has already selected a ticket and clicked the pay option	1. Type cardholder name in the "name" field. 2. Type the credit/debit card number in the card number field. 3. Type the exipiration date in the expiration date field. 4. Enter ccv code in ccv code field. 5. Enter in billing address in billing address field.	The user will be directed to a payment confirmation page if the credentials were correct	A payment confirmation page is displayed with a confirmation number	Pass	Dylan	unit
4	CreateTheaterTest_1	createTheater function within the Theater Class	P0	Verify that the createTheater function accurately updates the database with the new information	A theater admin has already navigated to theater manipulation and is prompted with fields to enter in the theater name, address, and rating	1. Type the theater name into the theater name field. 2. Type the theater's address into the address field. 3. Type the theater's rating into the rating field. If it does not have an existing rating then select n/a 4. Check the database for the new values.	The database shows a new Theater object with associated values for name, address, and rating	A Theater object is in the database with the correct values for name, address, and rating	Pass	Dylan	unit
5	EditMovieTest_1	editMovieInfo function within the Movie Class	P1	Verify that the editMovieInfo function accurately updates the Movie object's field variables	A theater admin has already navigated to movie data manipulation and is promted with fields to enter in the new info for an existing movie such as name, image, summary, time, location, and seat count	1. Type in the name in the name field. 2. Select an image to be associated with the movie 3. Type in the summary in the summary field. 4. Type in the time for the movie in the time field. 5. Type in the location for the movie in the location field. 6. Type in the seat count in the seat count field. 7. Check the database for the edited values that are supposed to be in place of the old values	The database shows an existing Movie object with the edited values for the fields mentioned	The existing Movie object already exists with the edited information corresponding with its field variables	Pass	Dylan	unit
6	buyTicket Function	creates a ticket object	P3	Tests the buyTicket function within the Guest class and Customer class Checks for proper creation of the ticket class. Ensures program grabs proper data from the user's session and populates the class with the correct data	A movie must be selected along with a location, date, and time. Along with that discount type and a vacant seat must be selected.	1. Navigate to the website www.watchalong.com 2. Enter login information or select continue as guest (the following steps will be the same either way) 3. Search for a movie 4. Select a location 5. Select an available date 6. Select an available time 7. Select a vacant seat 8. At this point the system will temporarily reserve your seat and create a tentative ticket object (This ticket object should populate with all the previously selected data) 9. The system will give you an option to pay 10. Once payment goes through the ticket object and seat reservation will become permanent 11. The system should navigate to a page that shows all the ticket details. This page can be printed by the user. 12. Another important step to check is that specifically for users that are logged in an email of the ticket information should be sent to the user's provided email.	In step 11 This page should be checked to ensure proper output. This step is crucial in this test case to ensure that the data populated and saved properly. In step 12 the email should be checked to ensure the proper data was sent from the database and not just from the current user session. That should be checked to make sure the proper information was delivered.	Proper movie ticket object was created. This was verified when the ticket page listed the data that was previously selected. Along with that an email was recieved with the exact same data	Pass	Manan	unit
7	changePasswordTest	account manipulation	P5	Verify that a user can change their password and successfully re-login to their account	User has already created an account with an associated username and password. User has already navigated to the account manipulation page	1. Click on button to change password 2. Enter in new password with the required amount and type of characters 3. Confirm new password 4. Navigate back to login page 5. Re-sign in with new credentials 6. Check if login is successful	The user will be successfully logged in with the new user-created password and is brought to their account page	User was signed in with the new password and was redirected to their account page	Pass	Micah	functional

	A	B	C	D	E	F	G	H	I	J	K
8	removePurchase_dataBase_update	removePurchase	P6	Verify that the removePurchase function performs timely updates for information displayed sitewide	valid movie ticket purchase must already be placed	1. Purchase should already be made. This can be through either the guest account or through an actual user account 2. If a user decides to refund their purchase within a timely manner they must call IT support to perform the refund process 3. Part of that refund process is to perform the removePurchase function that is a part of the IT support class 4. Once performed the ticket information should become invalidated since the userID assigned with it will not have that purchase in their account anymore. 5. With guest users since the whole account is temporary nothing more needs to be done. However with registered users the purchase must be dissasociated from their account. This will also prevent them from leaving reviews 6. Within the database the tester must check that from the list of ticket objects associated with the user's account the ticket in question is removed 7. The tester must also check that the user cannot create reviews for a movie they have not watched. 8. Lastly the tester must ensure that the seat count for that specific movie at a certain place and time is updated. So one specific movie object must be updated to where the seat that was previously purchased is now vacant.	One result will be that the ticket object associated with the account does not exist in the database anymore Another result will be that the user cannot leave any reviews of that movie if they have never watched that movie before The last result will be that the specifc movie object from the movie class is updated so that its seatCount array is updated to have that seat shown as vacant within a minute or two from ticket deletion.	The ticket object was deleted and not appearing in the users previous purchases The user cannot leave a review if they have never watched the movie before The seat count was updated system wide approximately 30 seconds after ticket deletion	Pass	Manan	functional
9	movieSearchTest	searchMovie	P4	This test will ensure the search function works properly	The user should already be at the movie search page	(The tester should enter various search criteria and make sure they all output relevant results) 1. The tester should enter a word 2. The tester should enter a phrase 3. The tester should enter a genre (such as comedy) in the genre search selection	1. The output should have only movies involving those words 2. The output should only have movies involving those phrases 3. The output should only have movies with those genres listed in their summary descriptions (when performing a genre search)	1. The results were only movie titles with that word in it 2. The results were only movies with that phrase in it 3. The results were only movies being shown in that specific city	Pass	Micah	functional
10	seatAvailabilityUpdate	buyTicket / seatCount	P9	This test will ensure that after purchasing a ticket that the seat count updates accordingly	The user should have a movie selected ready to purchase	1. After searching and selecting a movie at a certain location, date, and time the tester will "pay" for the ticket 2. Within a few seconds after this occurs another instance of the website will be open with a person searching for the exact same movie, location, date, and time 3. The second instance will check the seat availability 4. To further solidify the test the second instance user will try to purchase a ticket from the sam seat	When the second instance user checks the seat availability the seat in question should show up as occupied for them When the second instance user tries to purchase a ticket for the same seat their purchase should become rejected before they reach the payment screen.	While running the second instance the seat showed up as occupied The user was rejected from trying to purchase a seat that was already taken and recieved a notificaiton instructing them to pick another seat since that one was taken.	Pass	Manan	system
11	acountInfoUpdate_test1	changeName function / movieReview objects	P8	Verify that account changes show up systemwide	An account should already be created with existing movie reviews written	1. Create an account 2. Purchase a ticket for a movie to be allowed to leave a review for that movie 3. Leave a test review for that movie 4. Now edit the account name 5. With another user instance go to reviews of that movie and check on the review originally created	The second user instance should be able to see that the username changed from the original to the new username that was selected	After changing the username the movie review changed the account name of the user who left the review to the newly entered username	Pass	Manan	system

	A	B	C	D	E	F	G	H	I	J	K
12	Change_of_OS_Test	Overall System	P7	Verify that a user can use the application on different operating systems	User has a working operating system that includes either macOS or Windows OS	1. Use a system that uses Windows OS first 2. Navigate to the website www.watchalong.com 3. Test all functions including website main page, login page, account manipulation, theater data manipulation, movie data manipulation, review page, movie browser, seat map, etc. 4. If any of these fail, then the test has failed for Windows OS 5. Once Windows testing is done, move on to a machine with macOS 6. Repeat steps 2-3 7. If any of these fail, then the test has failed for macOS	All functions of the website will work on both macOS and Windows OS	All of the functions were operational and met our requirments on both macOS and Windows OS	Pass	Micah	system

*Link: Spreadsheet also on the Github

7.6 Testing Approach:

Much of the content held and displayed on this website is user generated. The accounts, the movies, and reviews are some of the core data on this system which is all heavily reliant on user input. With such being the case much of the testing will have to be done manually.
Some pre-programmed inputs can be developed and then executed simultaneously to check for edge cases where there are two conflicting inputs such as for buying the same seat. This can also be utilized to load test the website and ensure it operates smoothly with moderate levels of usage. Testers will have to manually check much of the functionality such as systems for creating movie listings, theaters, leaving reviews, and editing accounts and purchase information.

7.7 Test Schedule:

Unit Tests will take approximately 1 week to complete. Bug fixes will be applied in the midst of testing.
Functional Tests will be performed afterwards and will take 4 weeks to complete. This ensures cohesion between the software modules. The first week will be used to identify errors within the program. The second week will be used to resolve issues found within the code. Then the last 2 weeks will be spent confirming the proper functionality of the code and fixing any smaller issues found during the testing. System Tests will be performed afterwards for the next 5 weeks. The first 2 weeks will be spent on administrative side testing. The next 2 weeks will be spent on client side testing. The last week will be spent ensuring systemwide communication operates properly.

7.8 Risks and Mitigation:

One risk is receiving invalid payment information.

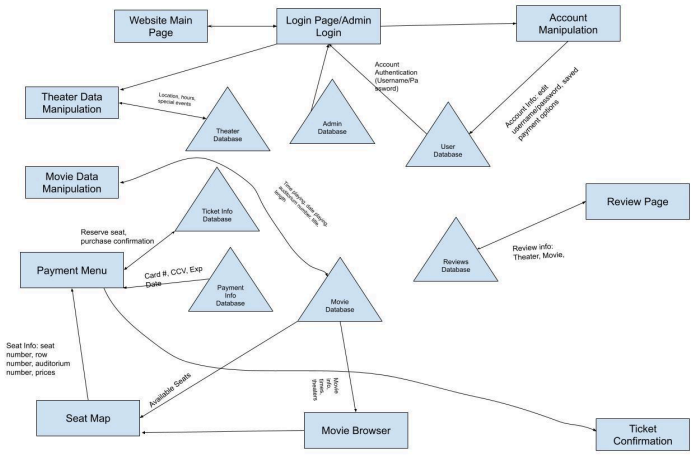
To solve this problem simple misinputs that do not meet the required amount of characters can be caught by the system. However invalid cards that are of proper length will need to be checked through a bank before a bank API before the transaction is finalized.
Another risk is with identical account creation. If hypothetically two users were to create an account with the same name and password how would the system differentiate the two at login time. The solution to this issue is having the userID that is associated with every user account to be required to be entered at login. This way everyone has a unique identifier and no information could completely overlap.
Another risk is when people purchase the same seat. If two users bought the same seat at about the exact same time there would be issues. This issue can be solved by having a condition that if two users bought a seat at the same time and neither of them was warned that the other already bought the seat then while the payment for the seat is pending the system will look at the userID for both customers and whoever's userID has the lower value will get the seat and the other will have their transaction canceled and be notified that the seat was taken.
One more risk is with payment security. There is no simple solution to this problem. The only proper way to manage the risks is to keep payment information in a smaller database that is much more secure and has much higher restricted access. Along with that it will have to be maintained with up to date security patches.

7.9 Conclusion:

After the steps depicted in the verification test plan are completed the website will be in a state where there is a high confidence level for the system to execute as intended. Any major issues should not arise. However with the increase in usage, less severe issues may be found that can be resolved with the help of the IT class and later fully patched by the development team.

8. Data Management Strategy

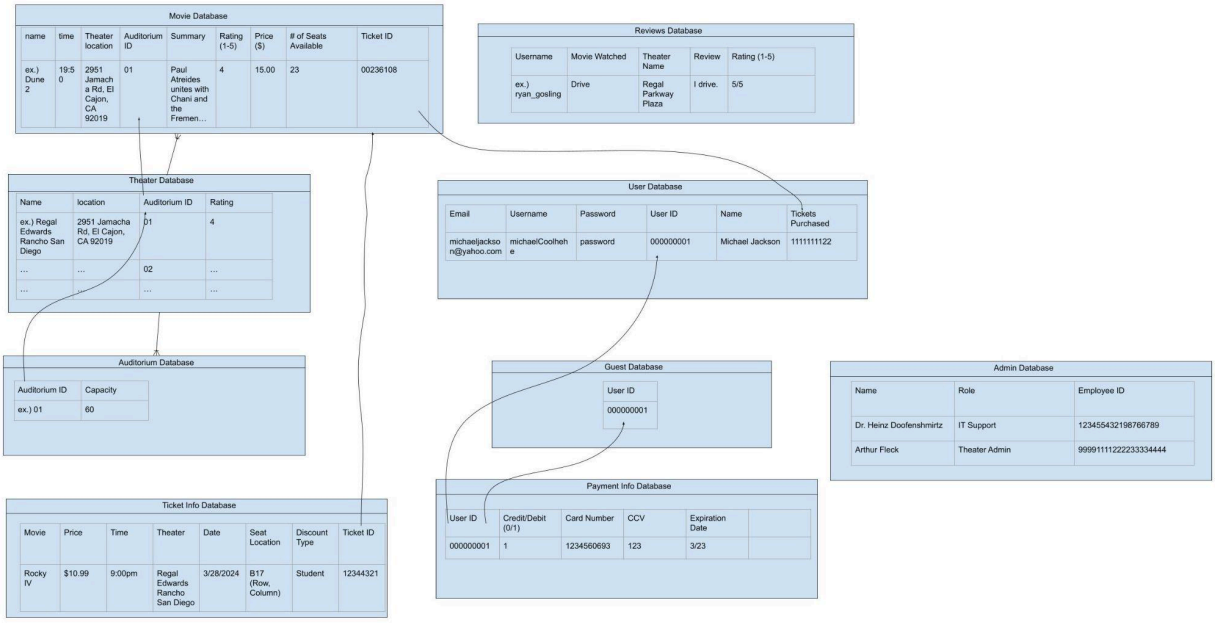
8.1 Updated SWA Diagram



8.1.1 Description

The updated SWA diagram integrates our data management strategy of using multiple databases for each of our specific parts of the program. Instead of one generic database, there are 7 databases containing information of which each part of the system uses. These include the theater, admin, user, review, movie, payment info, and ticket info database. Specific data transfers between each page and each database will be more straightforward with the new architecture design.

8.2 Data Management Diagram



8.2.1 Description

The above diagram is a visual representation of our data management strategy. Our data will be separated into 9 different databases which are all SQL. Every database except for the review and admin databases have connections between other fields in the other databases. For example, the Ticket ID field in the ticket info database has a direct connection to the movie database where individual movie tickets will have a unique ID based on the data in the ticket info database. Another example of the connections between databases is between the user, guest, and payment info databases. Every user will have a unique user ID and every guest will also have a unique user ID. These will be able to connect to the payment info database where a customer's payment information will be associated with each individual user and guest. The difference between the user ID field for the user database and the user ID for the guest database is that the guest ID will be permanently deleted after a transaction is completed. There is also a

connection between the movie database's ticketID field and the user database's tickets purchased field in order to store the tickets that the user has purchased in the past. These 9 databases will help store our data in a straightforward and optimized manner.

8.3 Explanation of Strategy

The database management strategy that we chose was SQL. The reason for this choice was because it was most optimized for the manner we wish to store our data. Data in SQL is stored in a format that shows the field variables of a class in different columns and each row represents a new object of that class. This is optimal in our case because our data follows a rigid structure that can be stored in this format to allow for ease of understanding the data that is being stored. Since the structure is very rigid and doesn't vary between different objects of the same class we can clearly show the correlation between the data and how they interact with each other. SQL also has many tools to prevent or also rectify corruption issues. This is good because the data being stored is meant to be stored for long term usage. For example purchase data needs to be stored so that users have the ability to write reviews later on. If that information was lost then users would lose access to some of the privileges they get through the purchase of a movie. SQL is also optimal for accessing, removing, appending, and other data altering commands.This is useful especially for the admin classes that would need to often manipulate theater data, movie data, and user data. Most data stored in our database is not static and will likely need to be changed frequently. SQL is also reliable when multiple users access the databases so it makes it useful for our application where thousands of people may be looking at different movies, writing reviews, purchasing seats, or editing account information at the same time. SQL also supports security and has functions built in just for that. This will be needed especially for specific databases that store payment information.

8.4 Database Organization

The databases were divided into 9 groups. These were representative of the main classes that many objects would be created from. These databases were the movie database, theater database, auditorium database, ticket info database, review database, user database, guest database, payment info database, and admin database. It is necessary to divide the databases as such because the objects from these classes all require varying levels of security. So it makes sense that depending on whatever data is accessed most frequently versus whatever data is most private the access to the database is limited and filtered accordingly. For example most people will be accessing movie info, theater info, auditorium info,and review info so those databases are more open for users since that information is open to the public anyways and does not need to be secured. The only thing to monitor is that hackers do not get in to modify the information into something that gives incorrect information. Next is the guest database. We need to temporarily create an object of the guest class for a singular user session for a customer that does not want to create an account. At the end of the session the object will be wiped. This database is unique because of how often objects are created and wiped. It will almost always be performing an operation on the contents within its database. User info database and ticket info database need to be more secure because those databases hold more personal information that only really needs to be available to the singular user. So access to this database is much more filtered. The admin database is a step up in security just so that admin credentials and employee information cannot

be taken and used to interfere with the website. Lastly is the payment info database. This database needs to be one of the most secure databases. The information being held here needs to be secured in such a manner that only the sole user associated with that payment information can access it. Also this database will not be altered as much or accessed that frequently so the operations in this database will be much more limited. In summary, the main points for determining the division of the databases was how frequently accessed the database would be this is correlated with the level of security necessary with the database. Obviously every database would be made with the intent of being secured but some databases need to be more secure than others which is why special precautions such as limited access will be put into place which obviously does not make sense to keep on every database as it will hinder system processes. Lastly is dynamic allocation of memory. Some databases such as the guest, user, and movie database will be expanding rapidly and so there should be tools put in place to help allow the databases to expand rapidly as needed. Obviously admin database and theater databases are things that expand much less frequently so it would not make sense to allocate as much storage to it as once. This is vital when considering cost efficiency.

8.5 Data Organization

The data in each database is split up based on each data type contained within the database. Each row of data in each database corresponds to a singular collection of data for that associated object. For example, each row in the user database is split up based on each individual user and their unique user ID. Each column is split based on the different data types associated with each individual object. Our data is organized into rigid partitions in order to be distinguishable from one another. These partitions are also important in making connections between databases and the data within them.

8.6 Possible Alternatives

We could have chosen to use a form of non-SQL such as a key-value database. If we chose this route we would likely have structured the database with far less subdivisions because each data input is identifiable by its key value. Another non-SQL option could’ve been some sort of tree.

8.7 Discussion

Choosing SQL with the table structure allows for very neat data organization with relationships between the different tables/databases. This will likely sacrifice speed of searching and potential updating it as well. Choosing non-SQl, especially a tree, would make searching very efficient, and also adding and removing stuff as well. A key-value database would require less subdivisions of the data but would be more difficult to interpret.