Movie STAR Ticketing System

Software Requirements Specification

Version 1.5

March 28, 2024

Group 2

Leah Bogomolny, Jake Jaro, Alisa Sriphet

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

Revision History

Date	Description	Author	Comments
2/1/2024	1.1	Leah, Jake, Alisa	Initial brainstorming and thoughts
2/6/2024	Worked on 1.2	Leah	Added information
2/14/2024	Continued revisions of 1.2	Alisa	Added information
2/15/2024	Continued additions and revisions of 1.2	Leah, Jake, Alisa	Added information and revised previous work
2/29/2024	worked on and completed 1.3	Leah, Jake, Alisa	Added system diagrams and descriptions for the system
3/14/2024	1.4, test plans added	Leah, Jake, Alisa	added test procedures

Document Approval

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

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

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide a detailed outline of the expectations for the movie ticket distribution software, *Movie STAR Ticketing System*. It is intended for software developers as well as all additional individuals who will be involved in the production of the application.

1.2 Scope

This document specifies requirements, features, and functions of *Movie STAR Ticketing System* as well as how the program will work. Specifically, this document caters towards developers and stakeholders of *Movie STAR Ticketing System* in order to describe the functionality of the system.

1.3 Definitions, Acronyms, and Abbreviations

Definitions, acronyms, and abbreviations used in this document are listed below:

Android:	Android operating system for mobile devices
Customer:	The person or organization that commissioned this software
Developer:	The people that develop and create the software
HTML:	HyperText Markup Language
HTTPS:	HyperText Transfer Protocol Secure
IOS:	iPhone Operating system
IP:	Internet Protocol
macOS:	Operating system for Apple computers
SCRUM	An agile software development life cycle that has work done over sprints
SRS:	Software Requirements Specification
TCP:	Transmission Control Protocol
URL:	Uniform Resource Locator
User:	Ideal audience that will be utilizing this software to buy tickets
Windows:	Operating system developed by Microsoft

1.4 References

This SRS does not reference any outside materials.

1.5 Overview

This SRS contains the various features and requirements for building *Movie STAR Ticketing System*. It is organized through the headers, which serve as a general description before getting increasingly more detailed with specific requirements and use case examples.

2. General Description

2.1 Product Perspective

The application will connect to third-party APIs to process payments. It will also require the use of databases to store theater, movie, and order information.

2.2 Product Functions

Movie STAR Ticketing System will be used by customers in order to search for movies by title and location, select movie times and seats, and buy up to 10 tickets to view a movie in theaters. The customers will also be able to pay online with credit or debit card or PayPal and get refunds or cancel orders as needed. Users will also receive an email confirmation after purchasing a ticket.

2.3 User Characteristics

The software can be utilized by users of any age or education level. However, payments should be made by individuals who are at least 18 years old. Technical expertise is also not required to use the application.

2.4 General Constraints

Our constraints include a short timeline of six months, and a budget of $75,000. Also, the software must be able to function across most web browsers and various mobile devices. It also must follow general protection and security regulations to protect user's sensitive information. Additionally, it must also adhere to regulations regarding the sale of official movie tickets. It must be able to support at least 1000 users simultaneously.

2.5 Assumptions and Dependencies

The software will be developed for Windows, macOS, Android, and IOS and will function on everything Windows 10 and later. For web browsers, it will run on Google Chrome, Microsoft Edge, FireFox, Safari, and DuckDuckGo. It is also assumed that the website and URL are available, and that the theaters will provide *Movie STAR Ticketing System* with the total number of seats, so maximum capacity can be known. Additionally, it is assumed that this site will support multiple theater franchises, with locations nationwide. Finally, it is assumed that the users have a working email address in order to receive tickets.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interfaces will be a computer or mobile device with access to the internet through a browser such as Google Chrome, Microsoft Edge, FireFox, Safari and DuckDuckGo.

3.1.2 Hardware Interfaces

The site requires the use of the internet, so on both the server side and the user's side, the hardware must connect to the internet through the use of ethernet cable for the server, and either ethernet cable, Wifi, or cellular data for the user. The site must function on personal, and workclass computers, along with mobile devices.

3.1.3 Software Interfaces

Movie STAR Ticketing System will interface and interact with Windows, macOS, Android, IOS, the billing software that is used, and the database management system.

3.1.4 Communications Interfaces

The system will use HTTPS protocol when transmitting information in order to keep sensitive user details secure. TCP/IP will also be used to connect devices across the internet.

3.2 Functional Requirements

3.2.1 Search Movie Database by Title and Theater Location

3.2.1.1 Introduction

- Users will be able to scroll through our database of movies ordered by popularity and also be able to use the search bar to search through the databases for the specific movie or theater they desire.

3.2.1.2 Inputs

- The inputs in using the search bar will be the theater name, the movie name, or the theater and movie name together (order does not matter).

3.2.1.3 Processing

- If scrolling through the database, the user can select the movie's poster or name which will be displayed below the poster, and will be brought to another page of just the movie. It will display the theaters that are showing it within 10 miles of the user's location, and under each theater will be listed that theater's offered times. The user can then select the time under their desired theater and will proceed to the ticket buying page.

- If the user uses the search bar, the site will take the input and sort through the database for theater or location depending on what is entered. If both the theater name and movie name are entered, it will go to a database of just that theater and what movies it includes. Then it will display the corresponding results depending on what is entered. The user will be able to select their desired theater/time from there.

3.2.1.4 Outputs

- It will output the theater locations if the search was just the movie name. If the search was just the theater name it will output the supported movies. If the search was the theater name and the movie name, it will display the movie at that theater and the supported times.

3.2.1.5 Error Handling

- If the movie is not being carried anywhere, or is otherwise not recognized, then the search will display a, 'Movie not found' page. If the theater does not exist, is not carried by the system, or is otherwise not recognized, it will have a line saying, 'Could not find theater' then below it will list supported theaters. If the search is a movie and theater, and the movie is not being shown at the theater, or the movie is otherwise not found in that theater's database, it will display the database of other movies being shown at the theater. If the search is a movie and theater, but the theater is not recognized, it will list all other locations of the movie as if only the movie was searched. If the search is the movie and the theater, but neither is recognized, a line will display that says, 'Could not find ___' with their search input in the underlined area. Below the line will display the popular movies in the same format as the homepage.

3.2.2 Select Movie Viewing Time and Date

3.2.2.1 Introduction

- Users of *Movie STAR Ticketing System* will be able to select their movie showing based on the desired time and date.

3.2.2.2 Inputs

- The inputs will include the specific desired movie.

3.2.2.3 Processing

- After selecting the desired movie, the user will be given lists of the available times for the specified movie at various theaters assuming they didn't already pick a theater. If both movie and theater have been previously selected, then the list of times will only include the times from that theater. In both cases it will only display times for the day the search is taking place at the times that have not yet passed. At the latest, users will be able to purchase tickets 10 minutes after the movie showing starts. If the user would like to select a day farther in the future, they can select any time within the following seven days that the search is taking place. The days will be listed at the top above in the time from the selected day. After clicking on a different day, it will reload to show a new list of times that corresponds to the movie showing times for that day.

3.2.2.4 Outputs

- The outputs will be that the user has successfully selected their desired time and date, and will allow them to proceed with selecting seat placement.

3.2.2.5 Error Handling

- If there is no time being provided on a specific day within seven days of the search, selecting the day will result in a 'No times available' message for that day. If the movie is removed after less than seven days from the search date then the amount of days listed will only include up to the last day instead of the full seven.

3.2.3 Selecting Desired Seats

3.2.3.1 Introduction

- The user of *Movie STAR Ticketing System* will be able to select 1-10 seats for a movie showing at a time and location that they choose and may also buy tickets to more than one movie at a time.

3.2.3.2 Inputs

- The inputs to this function will be the time of screening, selected movie, theater location, and selected seat location.

3.2.3.3 Processing

- This functionality will be processed by the user first selecting the seats they choose (up to 10 tickets for each movie and for multiple movies), adding them to cart, reviewing the cart, and then purchasing (discussed below).

3.2.3.4 Outputs

- Output for this functionality will be that the seat has been successfully chosen and will allow the user to continue on to paying for their ticket(s).

3.2.3.5 Error Handling

- Errors will be handled by having a popup that tells the user that there was an error and to try again with a different seat selection. Possible errors include that the seat is already taken, the showing was canceled, or the showing is sold out. If the showing was canceled or is sold out, the pop up will instead say that the showing is unavailable and ask the user to select a different showing or even a different location.

3.2.4 Offer multiple payment methods and send an order confirmation email

3.2.4.1 Introduction

- The system will allow users to purchase tickets via debit card, credit card, and PayPal.

3.2.4.2 Inputs

- The system will receive user debit/credit card information or PayPal login credentials as input, along with the user's email address.

3.2.4.3 Processing

- Confidential payment information will be securely transferred over HTTPS for every payment method. External APIs will be used to access the payment gateways.

3.2.4.4 Outputs

- The system will send the user a confirmation email containing a digital, printable receipt of the transaction, as well as their tickets.

3.2.4.5 Error Handling

- If the card is expired or declined, or if the user enters an incorrect card number, the system will display an error message and prompt the user to re-enter their information.

3.2.5 Cancel orders and provide refunds

3.2.5.1 Introduction

- Users shall be allowed to cancel their orders up to 1 hour in advance of the time of their selected movie, and they will be provided a full refund. Canceling an order will mean a refund of all tickets purchased in that order.

3.2.5.2 Inputs

- Users will be able to submit an order cancellation form that will prompt them for their order number, their email address, and their reason for canceling the order. The form can be accessed directly through the website or through a link found in the order confirmation email.

3.2.5.3 Processing

- The system will access the user's payment details and initiate the refund based on the order number provided in the cancellation form. The order number must be valid, and the email address must be the same as the one used to place the order initially. The refund will be immediately transferred to the user's account, and their previously selected seats will be reopened for purchase.

3.2.5.4 Outputs

- Users shall receive an additional email confirming that their order was canceled and their payment was refunded.

3.2.5.5 Error Handling

- If a user attempts to cancel their order less than 1 hour before their scheduled movie, the system will notify them that this action is not allowed and prevent them from submitting the form. If the provided order number is invalid, or the order number and email address do not correspond, the system will indicate that the information is invalid/incorrect and once again prevent the user from submitting the form. The user will then be given the opportunity to correct their input and submit the form.

3.2.6 User Feedback

3.2.6.1 Introduction

- Users of the system will be able to access a help form in order to submit complaints or inquiries.

3.2.6.2 Inputs

- Users will input their email, name, order number (if applicable), and questions/concerns.

3.2.6.3 Processing

- A customer service representative will view feedback from users and respond accordingly.

3.2.6.4 Outputs

- After the user submits a request, they will receive a confirmation email including the contents of their request. There will also be a popup on screen that the request was successfully submitted. Additionally, any response from a customer service representative will be sent to the email provided by the user.

3.2.6.5 Error Handling

- If the email that the user inputs can not receive emails, there will be a popup on the screen asking for the user to input a different email.

3.2.7 Load Bearing

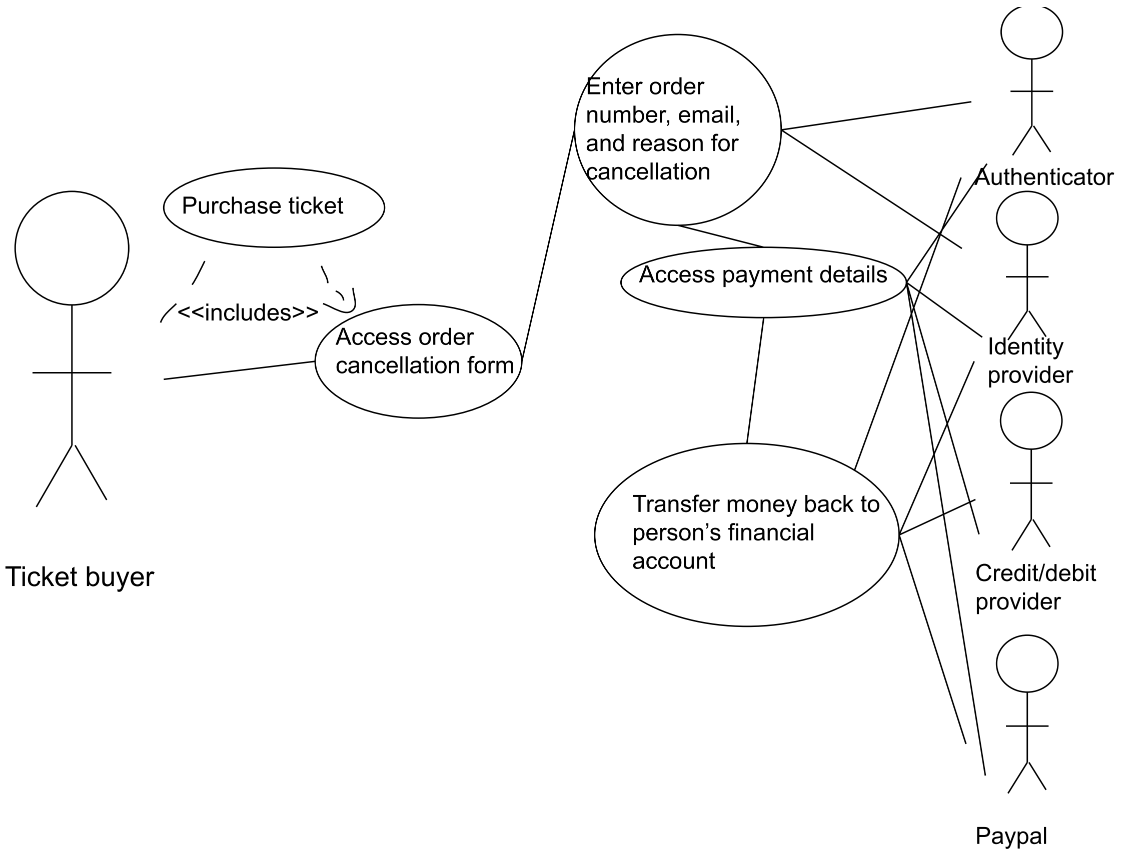
- The system must be able to support at least 1000 users at once.

3.2.8 Robot Blocking

- The system will block bots from purchasing large quantities of movie tickets at once.

3.3 Use Cases

3.3.1 Getting a Refund/Canceling Order


Enter order number, email, and reason for cancellation
Purchase ticket
<<includes>>
Access order cancellation form
Access payment details
Transfer money back to person's financial account
Ticket buyer
Authenticator
Identity provider
Credit/debit provider
Paypal

3.3.1.1 Actor(s):

- The actors are the ticket buyer, the authenticator that authenticates the ticket buyer's identity claim, the identity provider, which provides the proposed identity, the credit/debit provider and Paypal, which serve as the financial institutions.

3.3.1.2 Flow of Events:

1. Ticket buyer connects to the cancellation form through either email or website
2. Ticket buyer enters their order number, email address, and reason for cancellation
3. System checks their validity by connecting to their authenticator and identifier
4. Assuming all is well, proceeds to access payment details, connecting to authenticator to authenticate their action, identity verification to verify their identity, and the credit/debit, or Paypal in order to access the person's financial account.
5. Transfer the money back to the person's respective financial account

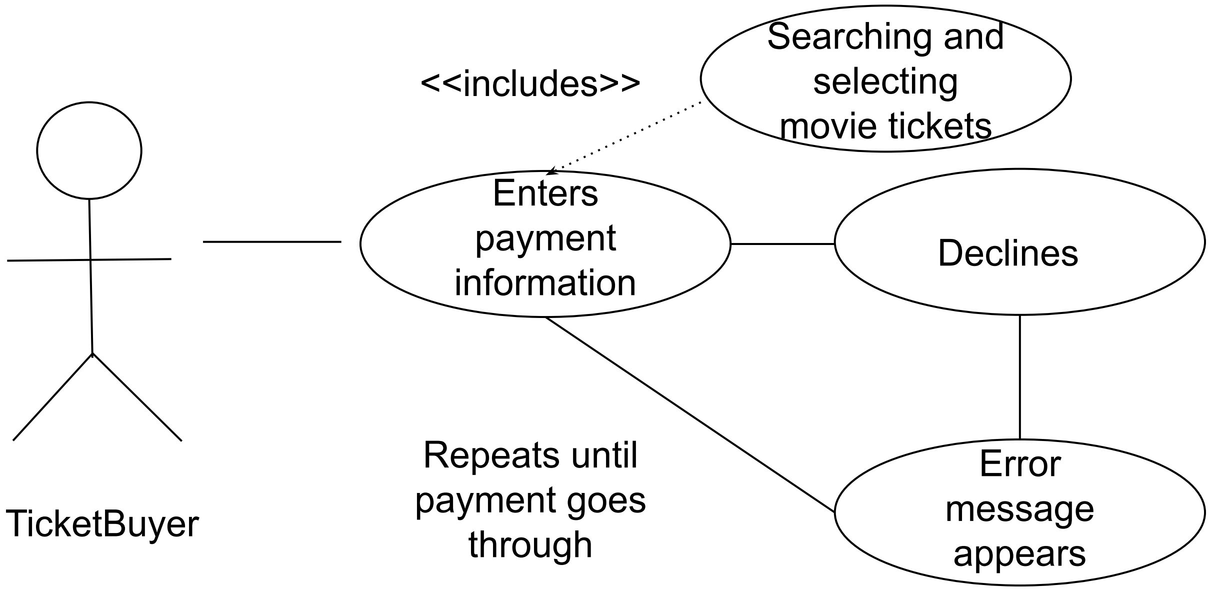
3.3.1.3 Possible Inclusions and Exclusions

- Possible inclusions are purchasing tickets. Possible exclusions are invalid order number or email address.

3.3.1.4 Entry Conditions:

1. internet connection
2. supported browser
3. required information (order number, email address, and reason for cancellation)

3.3.2 Card Declining



<<includes>>
Searching and selecting movie tickets
Enters payment information
Declines
Error message appears
Repeats until payment goes through
TicketBuyer

3.3.2.1 Actor(s)

- The actor in this use case is TicketBuyer, the person who is purchasing a ticket.

3.3.2.2 Flow of Events

1. TicketBuyer inputs payment information and purchases
2. Payment declined
3. System displays an error message and prompts TicketBuyer to re-enter payment information

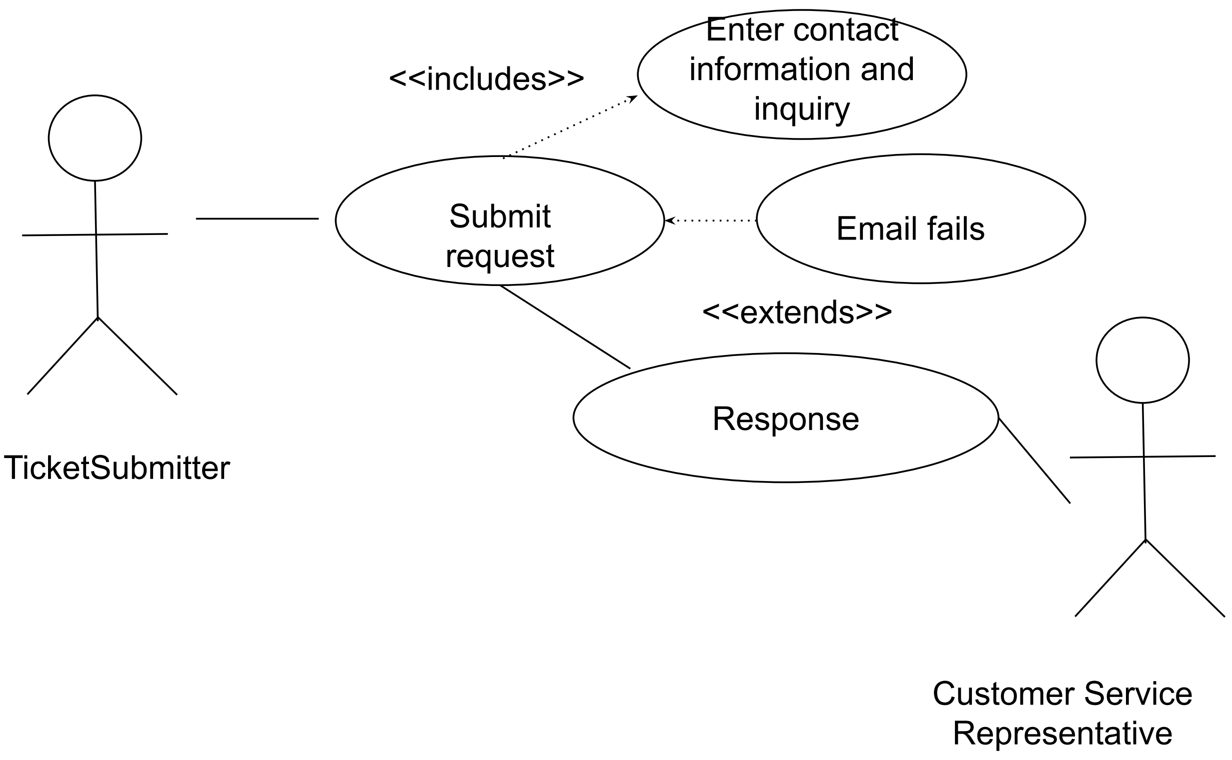
3.3.2.3 Possible Exclusions and Inclusions

- Inclusions to this situation would be searching the website and selecting a movie and seats.

3.3.2.4 Entry Conditions

- Entry conditions for this situation would be a connection to the internet, a supported browser, and payment information.

3.3.3 Submitting a Request



3.3.3.1 Actor(s)

- The actors are the TicketSubmitter who is a user that has an inquiry and is submitting some sort of request. The second actor is the Customer Service Representative who is an employee that responds to the inquiry submitted by TicketSubmitter.

3.3.3.2 Flow of Events

1. TicketSubmitter submits request after filling out contact information and inquiry
2. Then the Customer Service Representative sends the TicketSubmitter a response

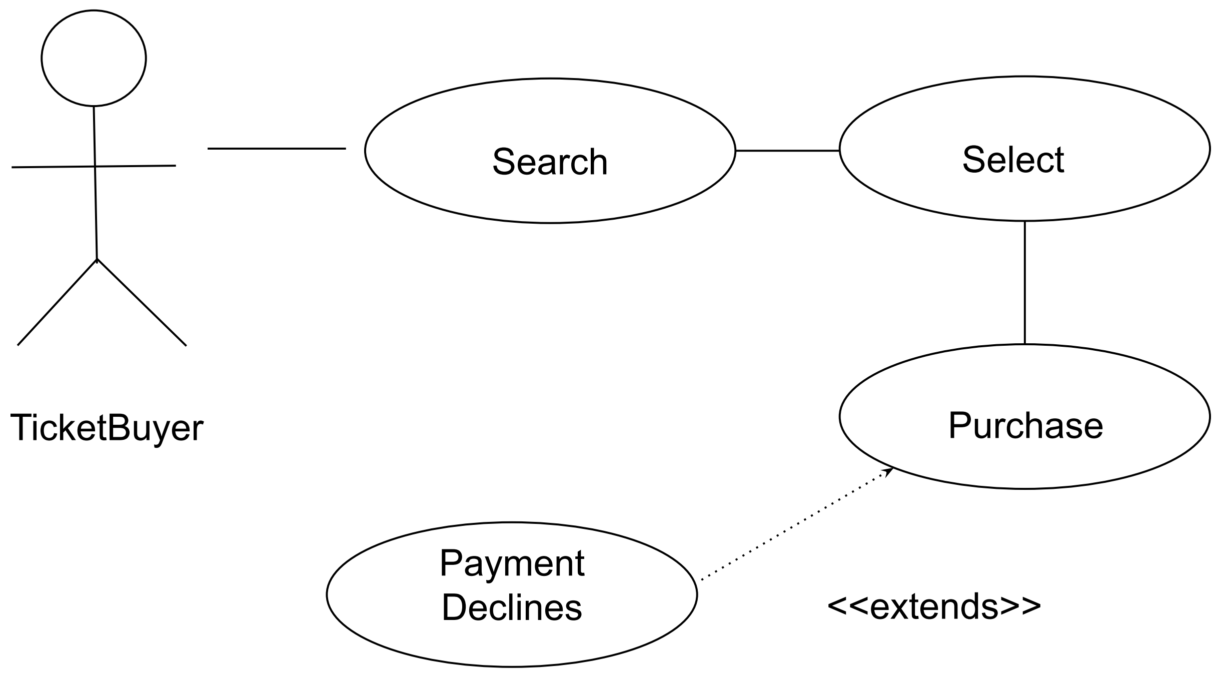
3.3.3.3 Possible Exclusions and Inclusions

- A possible exclusion is if the email that the TicketSubmitter submits does not exist or can not receive emails. If this happens, the TicketSubmitter is prompted to submit a different email.

3.3.3.4 Entry Conditions

- Entry conditions for this situation would be a connection to the internet, a supported browser, and an email account.

3.3.4 Purchasing a Ticket



3.3.4.1 Actor(s)

- The actor for this use case is someone who is looking to purchase a ticket through *Movie STAR Ticketing System,* who will be called "TicketBuyer"

3.3.4.2 Flow of Events

1. TicketBuyer connects to the website
2. TicketBuyer searches the database to find a movie showing to buy tickets for
3. TicketBuyer selects desired showing and seats
4. TicketBuyer goes to cart, inputs payment information, and purchases tickets
5. TicketBuyer receives an email confirmation for the order

3.3.4.3 Possible Exclusions and Inclusions

- Possible exclusions for this use case include that the TicketBuyer's search query may not return results, the payment information may be declined, or they may have selected a taken seat. See other use cases to see how these are handled.

3.3.4.4 Entry Conditions

- Entry conditions for this situation would be a connection to the internet, a supported browser, and valid payment information.

3.4 Classes / Objects

3.4.1 Movie Class

3.4.1.1 Attributes

- Title
- Release date

- Rating
- Description/summary

3.4.1.2 Functions

- ‘Get’ methods for each attribute

3.4.2 Venue Class

3.4.2.1 Attributes

- Venue name
- Venue address
- Current showings array

3.4.2.2 Functions

- ‘Get’ methods for each attribute

3.4.3 Theater Class

3.4.3.1 Attributes

- Theater number

3.4.3.2 Functions

- ‘Get’ methods for each attribute

3.4.4 Showing Class

3.4.4.1 Attributes

- Movie object
- Date
- Time
- Seats

3.4.4.2 Functions

- ‘Get’ methods for each attribute
- isSeatAvailable(Seat)

3.4.5 Seat Class

3.4.5.1 Attributes

- Showing object
- Seat row
- Seat number
- Availability
- Price

3.4.5.2 Functions

- ‘Get’ methods for each attribute

3.4.6 Order Class

3.4.6.1 Attributes

- Time the order was placed
- Date the order was placed
- Order cost
- User email

- User first name
- User last name
- User billing address
- Tickets purchased, and the quantity of each
- Payment object

3.4.6.2 Functions

- 'Get' methods for each attribute

3.4.7 PayPalPayment Class

3.4.7.1 Attributes

- User email
- User password

3.4.7.2 Functions

- userDetails()

3.4.8 CardPayment Class

3.4.8.1 Attributes

- Card number
- Card security code
- Card expiration date

3.4.8.2 Functions

- userDetails()

3.5 Non-Functional Requirements

3.5.1 Performance

Performance standards are that 95% of transactions will be processed in less than 5 seconds.\

3.5.2 Reliability

The mean time between failure and downtime of the system will be greater than 30 days.

3.5.3 Availability

The system's downtime will not exceed greater than 5 minutes per day.

3.5.4 Security

This system aims to have 95% of transactions be secure and aims to report any fraudulent activity within a day of it occurring. Additionally, this system will strive to ensure that no user data is leaked or hacked.

3.5.5 Maintainability

If the system does fail, it will be fixed and running again within 5 minutes.

3.5.6 Portability

This program runs on common web browsers such as Google Chrome, Microsoft Edge, FireFox, Safari and DuckDuckGo.

3.6 Inverse Requirements

- The site will not offer tickets for any non-movie showings at the theaters.
- It will not be possible to buy concessions through the site.
- The site will not offer a map to, or of, the theater(s), but the site will provide the address of the selected theater location.

3.7 Design Constraints

- The site must obtain all necessary licenses and follow all regulations to display images and handle the selling of tickets for the carried movies.
- The site must have a sleek background that does not draw attention away from the movie posters.
- All devices must have a web browser in order to access the site.
- All products displayed on the site must be easily accessible and clearly displayed for the user.

3.8 Logical Database Requirements

Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.

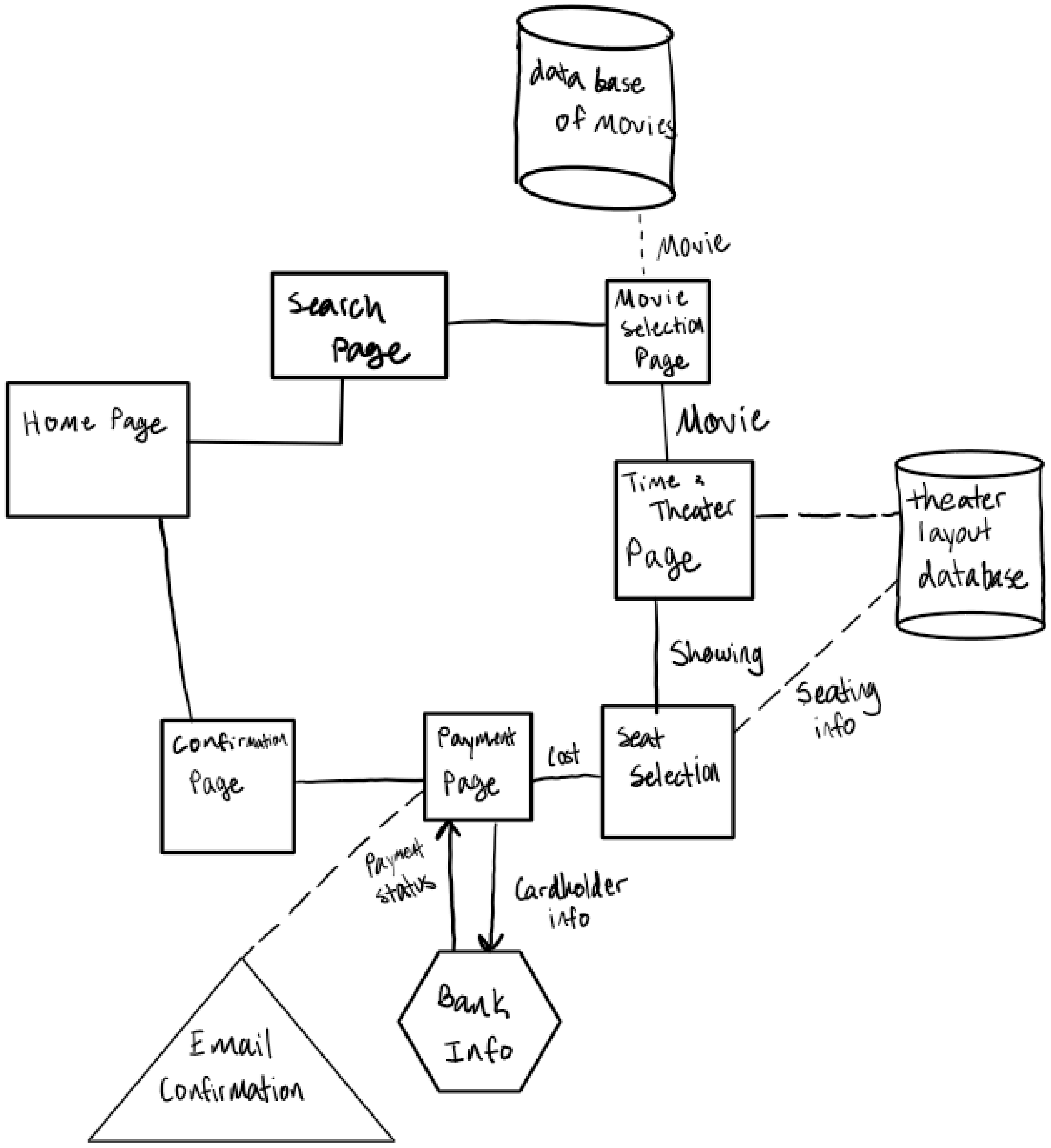
3.9 Other Requirements

Catchall section for any additional requirements.

4. Analysis Models

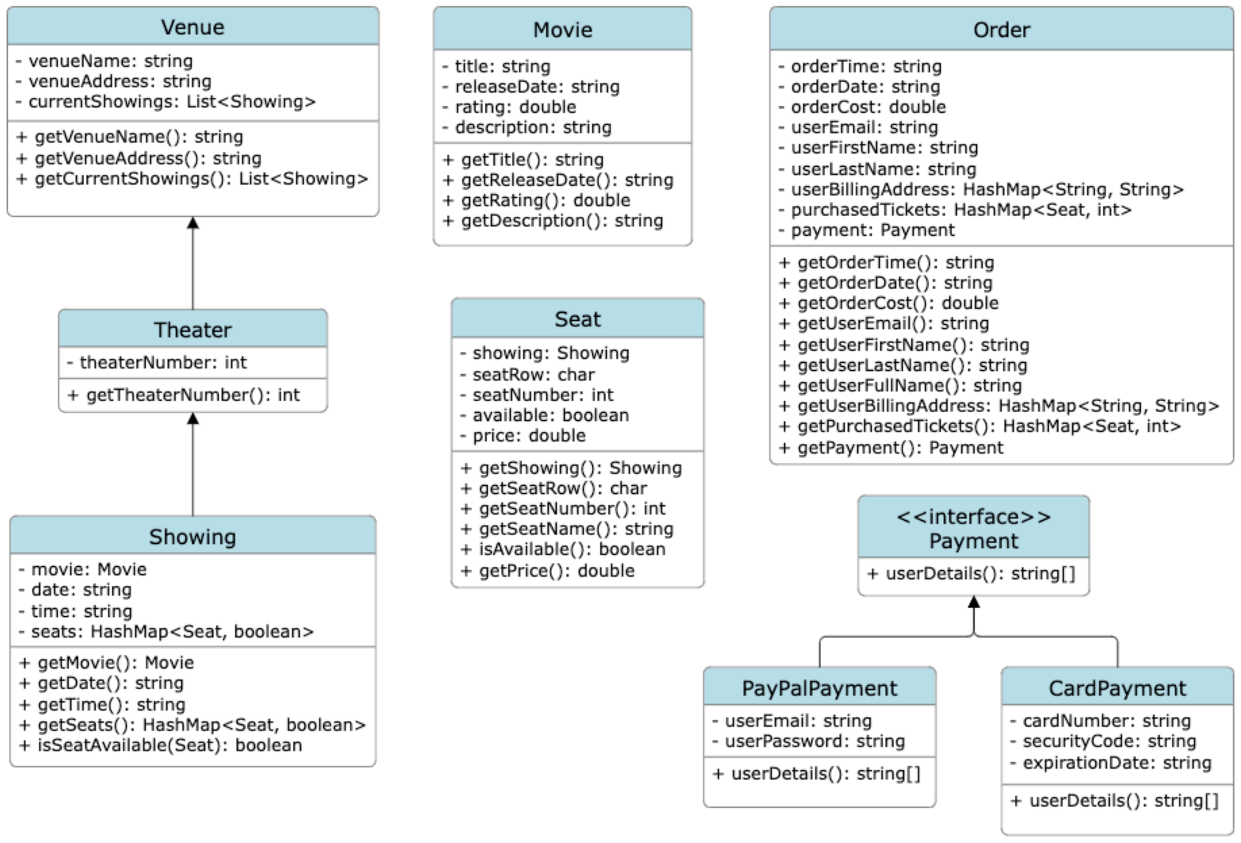
As shown in the Architecture Diagram, the system will include multiple pages: home, search, movie selection, time and theater selection, seat selection, payment, and the confirmation of payment, for the user to progress through. As the project is built, we will follow the hierarchy outlined in the UML Diagram, which includes Venue, Theater, and Showing. Each of these classes will have their own methods and variables. We plan to use a SCRUM development life cycle (more details under Task Partitioning) and should complete the development in eight weeks.

4.1 Architecture Diagram



The user begins on the home page and from there can go to the search page in order to search for a movie, using the movie database. Then, the user can select a movie and move to the time and theater page, passing the movie that the user has selected. Then the user selects a showing and moves to the seat selection page, while passing the selected showing. The user then selects a seat through the theater layout database, which passes seating information. Then the user continues to the payment page which has a parameter of cost. The user then inputs payment information which is sent to an API of bank information which exchanges cardholder information and payment status. After the payment goes through, the user ends up at the confirmation page and also receives an email confirmation. Then, the user can return to the home page if desired.

4.2 UML Class Diagram



4.2.1 Overview

- The primary class hierarchy is Venue > Theater > Showing. The logic behind this inheritance chain stems from the fact that each venue contains multiple theaters, which each provide certain showings throughout the day. The other classes are not related through inheritance, but they are still utilized within one another. Additionally, most of the classes contain getter methods for most, if not all, of their fields.

4.2.2 Venue

- This class represents the actual building/location that offers the movie showings. It contains fields for the most essential information about the location, like its name, address, and current movie offerings.

4.2.3 Theater

- This class represents the room where people actually sit down and watch movies. Every venue has several different theaters. The class inherits from the Venue base class, and also includes a field for the theater number used to identify a certain theater within a venue.

4.2.4 Showing

- A showing is a specific movie at a specific time and location that users will go watch. The Showing class stores these details, as well as a HashMap that contains the seats for that showing and their respective availability. The isSeatAvailable() function returns true if the given seat is available for purchase for that particular showing.

4.2.5 Movie

- This class simply holds the basic information about a movie, like its title, rating, release date, and a short description.

4.2.6 Seat

- A seat is what users will ultimately be purchasing. Every showing has a selection of seats that are available for purchase, which are identified by their row and number (e.g A2 is seat 2 of row A). The 'available' field indicates whether or not a seat is still able to be selected and bought.

4.2.7 Order

- The Order class stores all of the information that is entered when a user checks out. This includes the date and time of the purchase, the user's contact information, the contents of the order, and payment details.

4.2.8 Payment (Interface)

- This interface primarily serves as a way to relate the two payment methods and allow for polymorphism. It contains one method, userDetails, which must be implemented by both child classes. All sensitive information will be encrypted to ensure security.

4.2.9 PayPalPayment

- If the user chooses PayPal as their payment method, this class will store their login credentials in the 'userEmail' and 'userPassword' fields.

4.2.10 CardPayment

- If the user opts to use a credit or debit card, their card information will be stored in this class, through the 'cardNumber', 'securityCode', and 'expirationDate' fields.

4.3 Task Partitioning

This system will be developed using the SCRUM development life cycle. Each sprint will focus on a different database and the associated functionality with that database. The first sprint will focus on the movie database and how to implement the search functionality as well as the home page, search page, and movie selection. Team 1 will work on the database, Team 2 on the home page, Team 3 on the search page, and Team 4 on the movie selection page After this first sprint, which will take around two weeks, the team will move on to the next sprint which will focus on showing and seat selection pages and incorporate the database of theaters and their layouts. Team 1 will work on the showing/theater and time selection page, Team 2 on the seat selection page, Team 3 and 4 on the database of theaters and their layouts. This sprint will also aim to be around two weeks. The final sprint will be to complete the payment page while utilizing the API for processing payments and also sending a confirmation email to the user. It will also include the confirmation page and take around two weeks. Team 1 will work on the payment page, Team 2 will work on the API functionality, Team 3 will work on processing payments and sending confirmation emails, and Team 4 will work on the confirmation page. After this sprint, the team will regroup and check for any errors that have been missed, since testing will also occur during each sprint. If a team finishes their tasks and testing before the sprint is over, they will support the other teams in what they are working on. This system will take about eight weeks to complete if a conservative estimate is used.

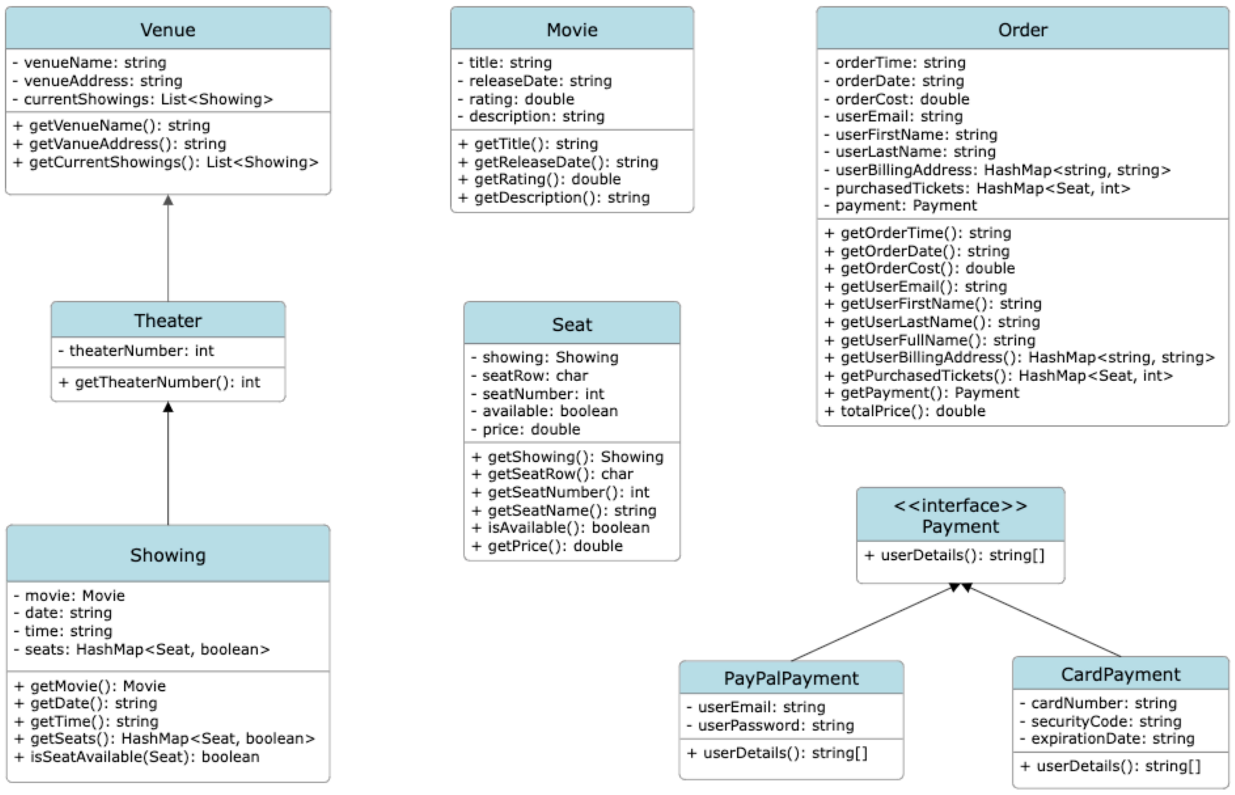
Github Commits

Leah Bogomolny - link
Jake Jaro - link
Alisa Sriphet - link

5. Verification and Validation

Identify and describe the process that will be used to update the SRS, as needed, when project scope or requirements change. Who can submit changes and by what means, and how will these changes be approved.

5.1 Updated UML Class Diagram



5.1.1 Modifications

The only change was that the function totalPrice() was added to the Order class.

5.2 Test Plan - Searching

5.2.1 The testing process

In order to test if the search capability works properly, it will be tested at the unit, functional, and integration levels. The unit level testing will verify that conducting a search will return results. Functional testing will test that each movie and location is searchable and that applying these filters to the search will return correct results. Functional testing will also include testing what the search returns if someone searches a movie or location that is not active or searches something that can not be identified. It will also test that searching a movie will return results sorted by location and searching by location will return results sorted by movie. The integration level testing will ensure that selecting a movie, location, and time will take the user to the seat selection page. All of these tests will be conducted for a substantial proportion of possible movies and locations.

5.2.2 Tested items

- Search functionality
- Search filters
- Movie showing selection

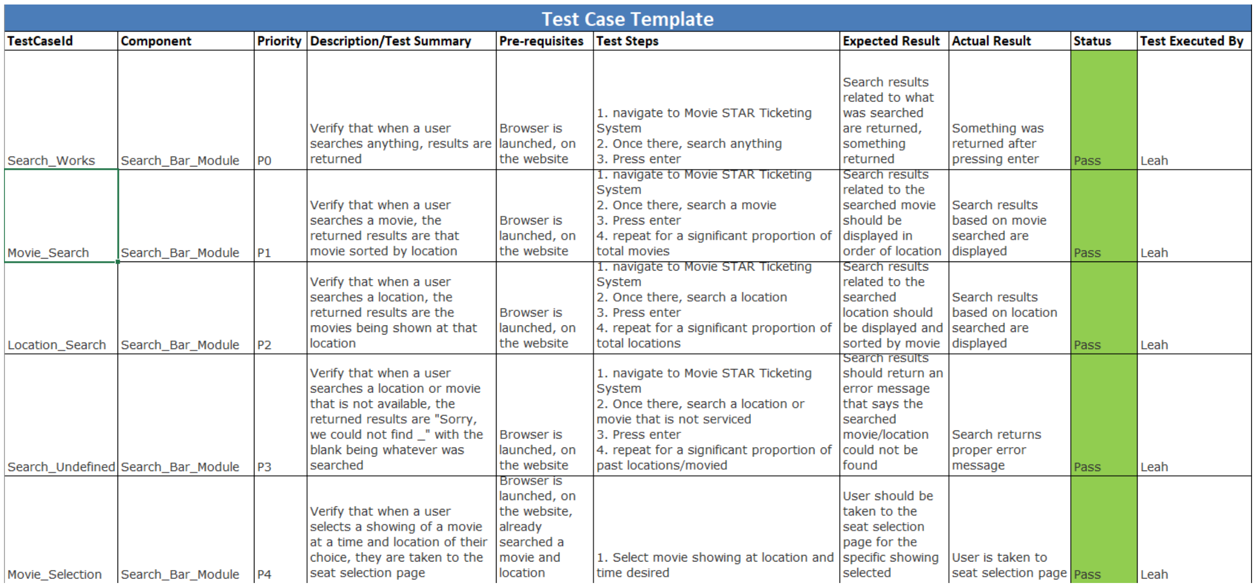
5.2.3 Hardware and software requirements

Hardware and software requirements are that the tester must use a device that can access the internet, and the software must be compatible with most current browsers.

5.2.4 Constraints

Constraints include that the movie and location database must be set up and up to date and that it is updated every time a new movie or location is added or removed. Additionally, there is only about two months to complete testing with only four teams to work on it.

5.2.5 Test Cases



Test Case Template

TestCaseId	Component	Priority	Description/Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed By
Search_Works	Search_Bar_Module	P0	Verify that when a user searches anything, results are returned	Browser is launched, on the website	1. navigate to Movie STAR Ticketing System 2. Once there, search anything 3. Press enter	Search results related to what was searched are returned, something returned	Something was returned after pressing enter	Pass	Leah
Movie_Search	Search_Bar_Module	P1	Verify that when a user searches a movie, the returned results are that movie sorted by location	Browser is launched, on the website	1. navigate to Movie STAR Ticketing System 2. Once there, search a movie 3. Press enter 4. repeat for a significant proportion of total movies	Search results related to the searched movie should be displayed in order of location	Search results based on movie searched are displayed	Pass	Leah
Location_Search	Search_Bar_Module	P2	Verify that when a user searches a location, the returned results are the movies being shown at that location	Browser is launched, on the website	1. navigate to Movie STAR Ticketing System 2. Once there, search a location 3. Press enter 4. repeat for a significant proportion of total locations	Search results related to the searched location should be displayed and sorted by movie	Search results based on location searched are displayed	Pass	Leah
Search_Undefined	Search_Bar_Module	P3	Verify that when a user searches a location or movie that is not available, the returned results are "Sorry, we could not find _" with the blank being whatever was searched	Browser is launched, on the website	1. navigate to Movie STAR Ticketing System 2. Once there, search a location or movie that is not serviced 3. Press enter 4. repeat for a significant proportion of past locations/movied	Search results should return an error message that says the searched movie/location could not be found	Search returns proper error message	Pass	Leah
Movie_Selection	Search_Bar_Module	P4	Verify that when a user selects a showing of a movie at a time and location of their choice, they are taken to the seat selection page	Browser is launched, on the website, already searched a movie and location	1. Select movie showing at location and time desired	User should be taken to the seat selection page for the specific showing selected	User is taken to seat selection page	Pass	Leah

5.3 Test Plan - Selecting

5.3.1 The testing process

For this test plan, it will describe how the selection of seats will be tested. Starting at a unit level, we will test that clicking on the image of an open seat will redirect the user to the payment page and close off the seat to other customers. It must only redirect if the seat is open. For integration, we will test to ensure that after the user selects their movie and theater location, that they are presented with a page containing an interactive image of the seats of the given theater. These seats must be colored differently to reflect their availability status. After testing unit and integration, for functional testing we must check that an *Invalid* message is given if the selected seat is already taken. We will also have to test that after 10 minutes, if the user has not proceeded with payment, the seats will be forfeited, and return to an open status. If the user selected open seats, and proceeded with payment, the seats will be recorded under their order, and must be present on their ticket.

5.3.2 Tested items

- Seat selection button
- Seat display page
- Timer that will redirect user after it runs out

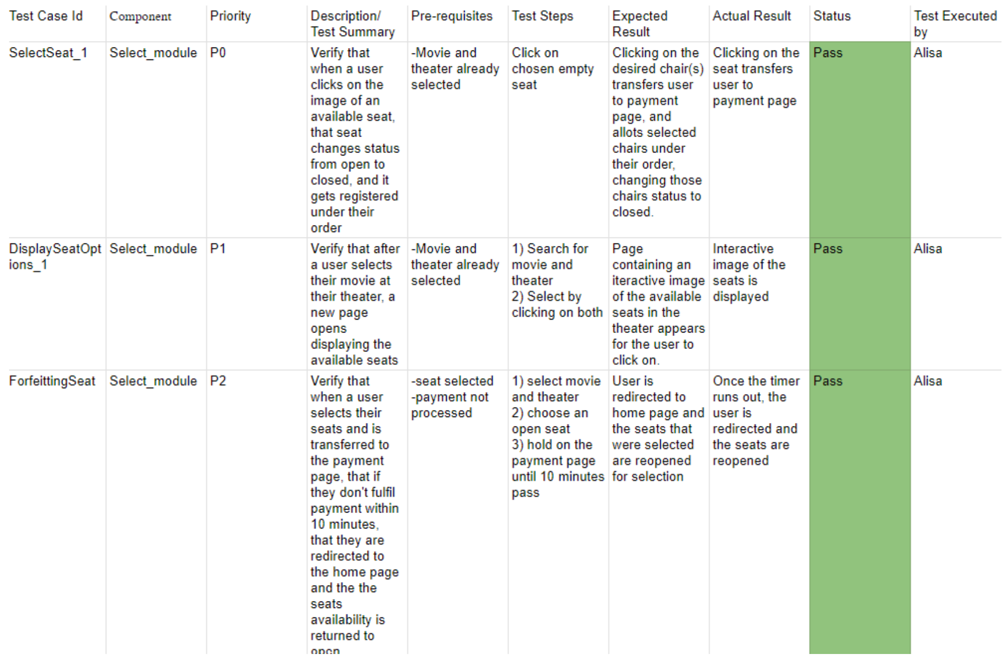
5.3.3 Hardware and software requirements

Must use a device that can access the internet, and the software must be compatible with most current browsers.

5.3.4 Constraints

There would only be roughly two months to complete all the testing, and we only have four teams available to work on it.

5.3.5 Test Cases



Test Case Id	Component	Priority	Description/ Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed by
SelectSeat_1	Select_module	P0	Verify that when a user clicks on the image of an available seat, that seat changes status from open to closed, and it gets registered under their order	-Movie and theater already selected	Click on chosen empty seat	Clicking on the desired chair(s) transfers user to payment page, and allots selected chairs under their order, changing those chairs status to closed.	Clicking on the seat transfers user to payment page	Pass	Alisa
DisplaySeatOptions_1	Select_module	P1	Verify that after a user selects their movie at their theater, a new page opens displaying the available seats	-Movie and theater already selected	1) Search for movie and theater 2) Select by clicking on both	Page containing an iteractive image of the available seats in the theater appears for the user to click on.	Interactive image of the seats is displayed	Pass	Alisa
ForfeittingSeat	Select_module	P2	Verify that when a user selects their seats and is transferred to the payment page, that if they don't fulfil payment within 10 minutes, that they are redirected to the home page and the the seats availability is returned to open	-seat selected -payment not processed	1) select movie and theater 2) choose an open seat 3) hold on the payment page until 10 minutes pass	User is redirected to home page and the seats that were selected are reopened for selection	Once the timer runs out, the user is redirected and the seats are reopened	Pass	Alisa

5.4 Test Plan - Purchasing

5.4.1 The testing process

This test plan outlines how the process of purchasing tickets will be tested. The procedures will be broken down into unit, integration, and system tests, with the primary focus being placed on the order form and the other complementary components that allow a user to place an order. These additional components include things like the database and API. Purchasing_01, as seen in the table below, is a unit test case; it verifies that the function totalPrice() correctly calculates the total price of an order. Purchasing_08 and Purchasing_09 are system level test cases. The former covers the process of canceling an order, from the initial form submittal to the final refund. The latter simulates the entire process of buying tickets, which involves searching, selecting, and purchasing. The remaining tests (02-07) work at the integration level, as they check to make sure the different components of the frontend and backend work seamlessly together. They test the interactions that take place amongst the frontend components, the API, and the database.

5.4.2 Tested items

- totalPrice()
- Order Form
- API and Database
- Order Cancellation Process
- Ticketing Buying Process

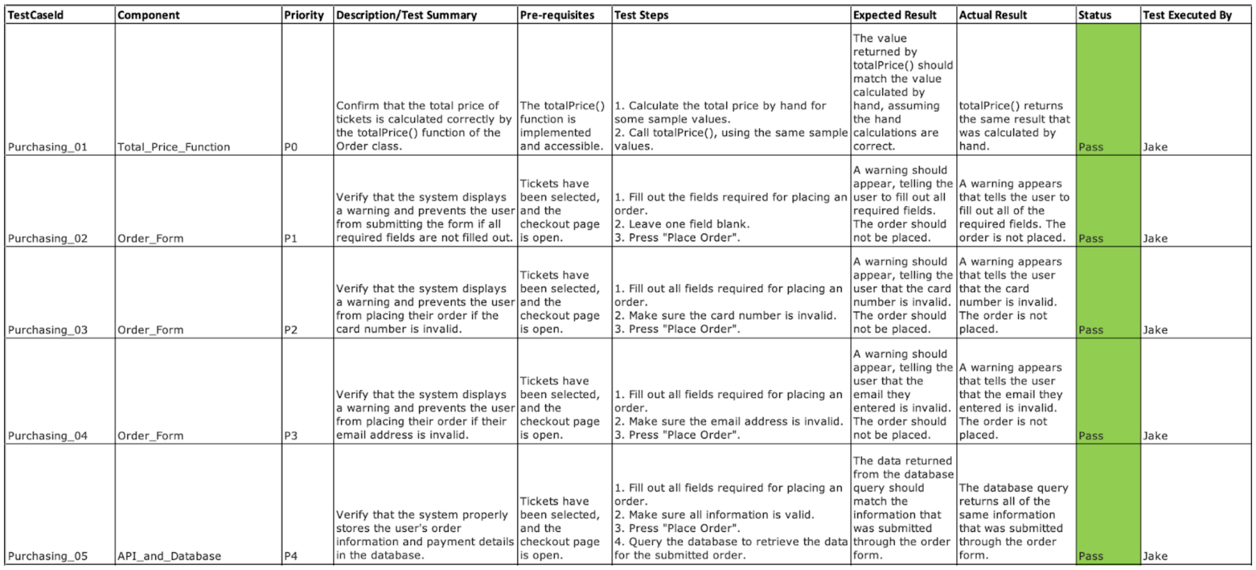
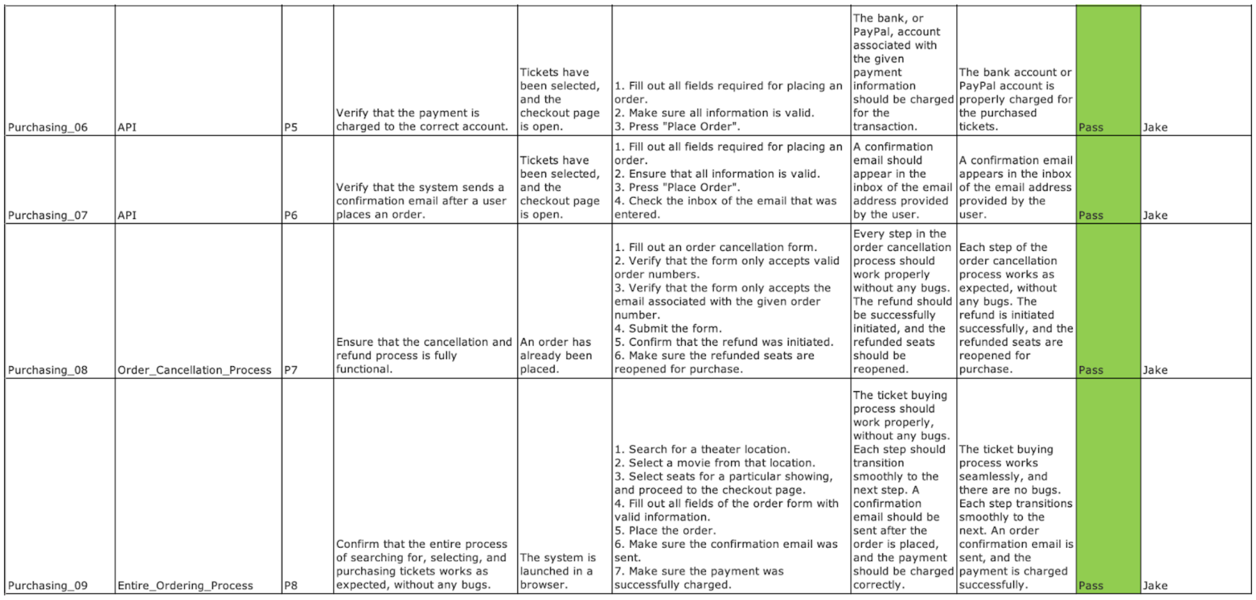
5.4.3 Hardware and software requirements

The device running the system must be connected to the internet. Also, the web browser and operating system must be up to date.

5.4.4 Constraints

There will be four teams available to conduct the testing, and they will have a two-month time frame to get the testing done.

5.4.5 Test Cases

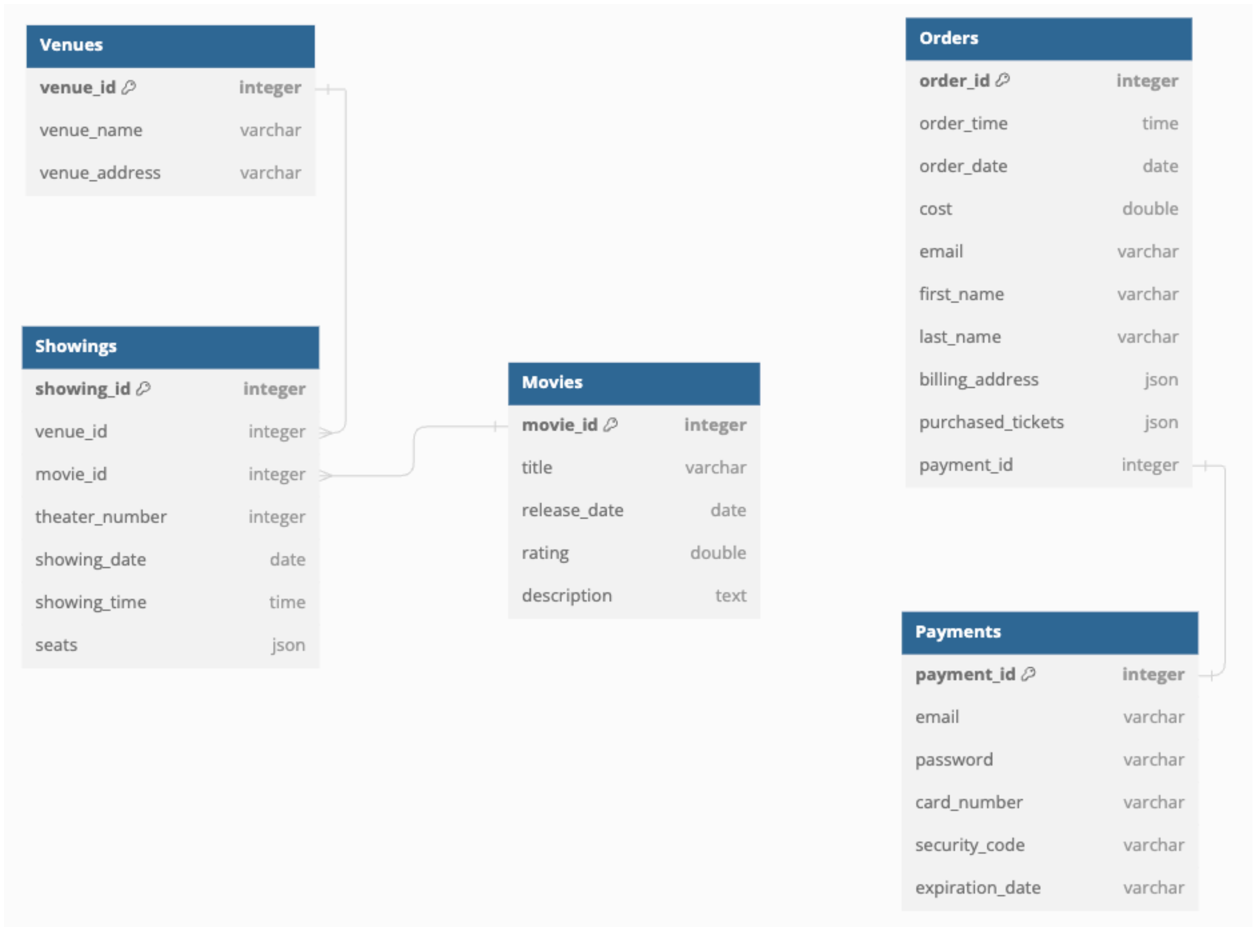
TestCaseId	Component	Priority	Description/Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed By
Purchasing_01	Total_Price_Function	P0	Confirm that the total price of tickets is calculated correctly by the totalPrice() function of the Order class.	The totalPrice() function is implemented and accessible.	1. Calculate the total price by hand for some sample values. 2. Call totalPrice(), using the same sample values.	The value returned by totalPrice() should match the value calculated by hand, assuming the hand calculations are correct.	totalPrice() returns the same result that was calculated by hand.	Pass	Jake
Purchasing_02	Order_Form	P1	Verify that the system displays a warning and prevents the user from submitting the form if all required fields are not filled out.	Tickets have been selected, and the checkout page is open.	1. Fill out the fields required for placing an order. 2. Leave one field blank. 3. Press "Place Order".	A warning should appear, telling the user to fill out all required fields. The order should not be placed.	A warning appears that tells the user to fill out all of the required fields. The order is not placed.	Pass	Jake
Purchasing_03	Order_Form	P2	Verify that the system displays a warning and prevents the user from placing their order if the card number is invalid.	Tickets have been selected, and the checkout page is open.	1. Fill out all fields required for placing an order. 2. Make sure the card number is invalid. 3. Press "Place Order".	A warning should appear, telling the user that the card number is invalid. The order should not be placed.	A warning appears that tells the user that the card number is invalid. The order is not placed.	Pass	Jake
Purchasing_04	Order_Form	P3	Verify that the system displays a warning and prevents the user from placing their order if their email address is invalid.	Tickets have been selected, and the checkout page is open.	1. Fill out all fields required for placing an order. 2. Make sure the email address is invalid. 3. Press "Place Order".	A warning should appear, telling the user that the email they entered is invalid. The order should not be placed.	A warning appears that tells the user that the email they entered is invalid. The order is not placed.	Pass	Jake
Purchasing_05	API_and_Database	P4	Verify that the system properly stores the user's order information and payment details in the database.	Tickets have been selected, and the checkout page is open.	1. Fill out all fields required for placing an order. 2. Make sure all information is valid. 3. Press "Place Order". 4. Query the database to retrieve the data for the submitted order.	The data returned from the database query should match the information that was submitted through the order form.	The database query returns all of the same information that was submitted through the order form.	Pass	Jake
Purchasing_06	API	P5	Verify that the payment is charged to the correct account.	Tickets have been selected, and the checkout page is open.	1. Fill out all fields required for placing an order. 2. Make sure all information is valid. 3. Press "Place Order".	The bank, or PayPal, account associated with the given payment information should be charged for the transaction.	The bank account or PayPal account is properly charged for the purchased tickets.	Pass	Jake
Purchasing_07	API	P6	Verify that the system sends a confirmation email after a user places an order.	Tickets have been selected, and the checkout page is open.	1. Fill out all fields required for placing an order. 2. Ensure that all information is valid. 3. Press "Place Order". 4. Check the inbox of the email that was entered.	A confirmation email should appear in the inbox of the email address provided by the user.	A confirmation email appears in the inbox of the email address provided by the user.	Pass	Jake
Purchasing_08	Order_Cancellation_Process	P7	Ensure that the cancellation and refund process is fully functional.	An order has already been placed.	1. Fill out an order cancellation form. 2. Verify that the form only accepts valid order numbers. 3. Verify that the form only accepts the email associated with the given order number. 4. Submit the form. 5. Confirm that the refund was initiated. 6. Make sure the refunded seats are reopened for purchase.	Every step in the order cancellation process should work properly without any bugs. The refund should be successfully initiated, and the refunded seats should be reopened.	Each step of the order cancellation process works as expected, without any bugs. The refund is initiated successfully, and the refunded seats are reopened for purchase.	Pass	Jake
Purchasing_09	Entire_Ordering_Process	P8	Confirm that the entire process of searching for, selecting, and purchasing tickets works as expected, without any bugs.	The system is launched in a browser.	1. Search for a theater location. 2. Select a movie from that location. 3. Select seats for a particular showing, and proceed to the checkout page. 4. Fill out all fields of the order form with valid information. 5. Place the order. 6. Make sure the confirmation email was sent. 7. Make sure the payment was successfully charged.	The ticket buying process should work properly, without any bugs. Each step should transition smoothly to the next step. A confirmation email should be sent after the order is placed, and the payment should be charged correctly.	The ticket buying process works seamlessly, and there are no bugs. Each step transitions smoothly to the next. An order confirmation email is sent, and the payment is charged successfully.	Pass	Jake

6. Database Management

6.1 Overview

We have selected to use SQL in order to build our database. We chose to use SQL because of how user friendly it is and how easy it is to learn. Another reason we chose to use SQL is that it is a relational database. Using a relational database allows us to more easily connect movie showings and the related theater that it will be shown at. Additionally, relational databases also provide good structure while allowing for easy interaction with and manipulation of data.

6.2 Database Diagram



6.2.1 Summary

The design of the database is heavily based on the system's classes. It contains tables for the venues, showings, and movies, which each have fields that resemble the attributes of their respective classes. There is also a table for past orders as well as one for the payments associated with those orders. Note that there is only one table for storing payments, despite there being two types of payment classes (see 4.2). The "Payments" table contains fields for the attributes of both the PayPalPayment class and CardPayment class. Every row of the table utilizes certain fields depending on the payment type, leaving all unused fields blank, or null.

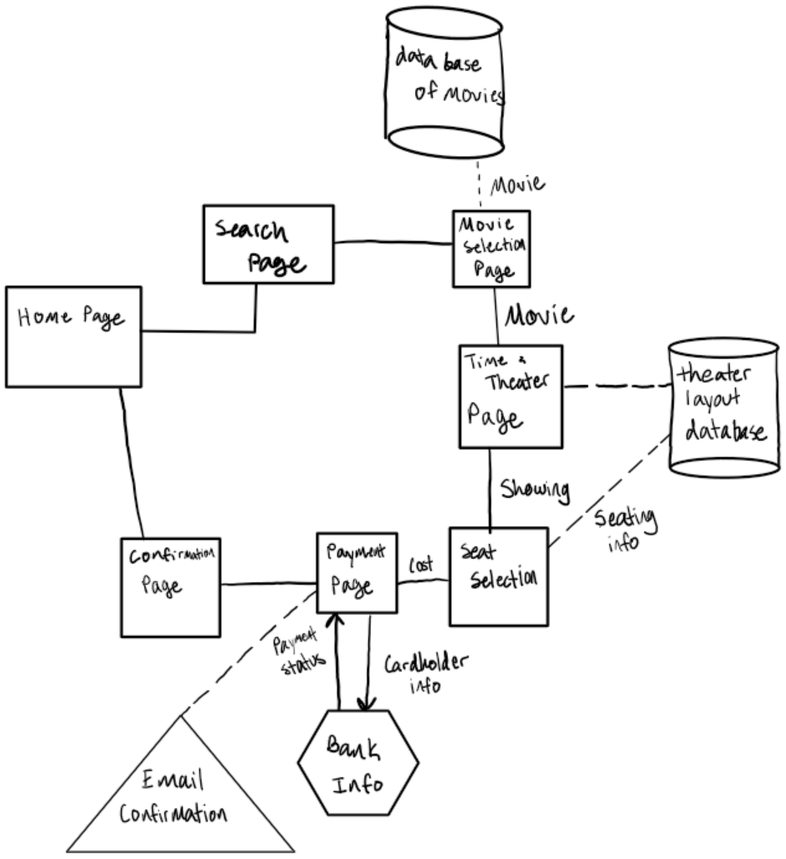
6.2.2 Relationships

There is a one-to-many relationship between the "Venues" table and the "Showings" table. This relationship is necessary because every venue contains multiple showings, and there must be some indication of which showings are associated with which venue. Additionally, there is a many-to-one relationship between the "Showings" table and the "Movies" table. This is due to the fact that each showing is for a particular movie and multiple showings can exist for the same movie. Lastly, there is a one-to-one relationship between the "Order" table and the "Payments" table because each order is linked to only one payment, and vice versa.

6.3 Trade-offs

We will be implementing a SQL database. While nonSQL is less rigid in that the schemas don't have to be predefined, this also means that there's no standard language and would require a steep learning curve and more experienced staff to operate which would raise our costs overall. Meanwhile, SQL is very user-friendly and managing the database can be done with simple keywords and very little coding. Also, compared to SQL, nonSQL is less established and therefore would be harder to solve any sort of undocumented issues that may appear. Additionally, the data in SQL is more consistent, having ACID compliance instead of just BASE. While scaling SQL databases is more difficult and costly than nonSQL, we are prioritizing the additional data security, consistency, and integrity that SQL offers. Additionally, since our data is very structured, we wouldn't need the accommodations that a nonSQL database offers.

6.4 Software Architecture Diagram



No changes were made to the Software Architecture Diagram. Refer to 4.1 for explanation.

A. Appendices

Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements.

Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.

A.1 Appendix 1

A.2 Appendix 2