

# Theater Ticketing System

## Software Specification

Version 2.1

16 October 2023

Group 2

Afnan Algharbi, Jasmine Rasmussen,  
Everett Richards

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

## Revision History

Date	Description	Author	Comments
09/07/23	Version 1.0	Group 2	First Draft
09/14/23	Version 1.1	Group 2	Various revisions
09/21/23	Version 1.2	Group 2	Final draft for Part 1
10/05/23	Version 2.0	Group 2	Final draft for Part 2
10/16/23	Version 2.1	Group 2	Added testing docs for Part 3

## Document Approval

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

Signature	Printed Name	Title	Date
	Afnan Algharbi	Software Engineer	10/05/2023
	Jasmine Rasmussen	Software Engineer	10/05/2023
	Everett Richards	Software Engineer	10/05/2023
	Dr. Gus Hanna	Instructor, CS 250	10/05/2023

# Table of Contents

<b>Revision History.....</b>	<b>2</b>
<b>Document Approval.....</b>	<b>2</b>
<b>1. Introduction.....</b>	<b>1</b>
1.1 Purpose.....	1
1.2 Scope.....	1
1.3 Definitions, Acronyms, and Abbreviations.....	2
1.4 References.....	2
1.5 Overview.....	2
<b>2. General Description.....</b>	<b>3</b>
2.1 Product Functions.....	3
2.2 User Characteristics.....	3
<b>3. Specific Requirements.....</b>	<b>4</b>
3.1 External Interface Requirements.....	4
3.1.1 User Interfaces.....	4
3.1.2 Hardware Interfaces.....	5
3.1.3 Software Interfaces.....	5
3.1.4 Communications Interfaces.....	5
3.2 Functional Requirements.....	5
3.2.1 Payment Processing.....	6
• Payments should be processed using credit, debit, PayPal, and gift cards / reward points.....	6
3.2.2 Temporary Ticket Reservations.....	6
• Customers should be able to reserve seats in their virtual cart for up to 5 minutes before actually purchasing them.....	6
3.2.3 E-Mail Notifications.....	6
• Receipts and notifications should be sent to customers through E-Mail.....	6
3.2.4 Refund Processing.....	6
• Refunds should be payable via theater gift cards.....	6
3.3 Use Cases.....	6
3.4 Classes / Objects.....	9
3.5 Functional Requirements.....	10
3.5.1 Performance.....	10
3.5.2 Reliability.....	10
3.5.3 Availability.....	10
3.5.4 Security.....	10
3.5.5 Maintainability.....	11
3.5.6 Portability.....	11
3.6 Inverse Requirements.....	11
3.7 Design Constraints.....	11

3.8 Logical Database Requirements.....	12
<b>4. System Description.....</b>	<b>13</b>
4.1 Brief overview of system.....	13
5.1 Architectural diagram of all major components.....	13
5.2 UI Landing Page Mockup.....	14
5.3 UML Class Diagram.....	15
<b>6. Classes, Objects, &amp; Attributes.....</b>	<b>17</b>
6.1 Theater.....	17
6.1.1 Attributes.....	17
6.1.2 Functions.....	18
6.1.3 Methods.....	18
6.2 Showtime.....	18
6.2.1 Attributes.....	18
6.2.2 Functions.....	18
6.2.3 Methods.....	19
6.3 Movie.....	19
6.3.1 Attributes.....	19
6.3.2 Functions.....	19
6.3.3 Methods.....	19
6.4 User.....	20
6.4.1 Attributes.....	20
6.4.2 Functions.....	20
6.4.3 Methods.....	20
6.5 ShoppingCart.....	21
6.5.1 Attributes.....	21
6.5.2 Functions.....	21
6.5.3 Methods.....	21
6.6 Membership.....	21
6.6.1 Attributes.....	21
6.6.2 Functions.....	21
6.6.3 Methods.....	22
6.7 Ticket.....	22
6.7.1 Attributes.....	22
6.7.2 Functions.....	22
6.7.3 Methods.....	22
6.8 DiscountCategory.....	23
6.8.1 Attributes.....	23
6.8.2 Functions.....	23
6.8.3 Methods.....	23
6.9 Genre.....	23
6.9.1 Attributes.....	23

6.9.2 Functions.....	23
6.9.3 Methods.....	23
6.10 Employee.....	23
6.10.1 Attributes.....	23
6.10.2 Functions.....	24
6.10.3 Methods.....	24
<b>7. Task Management and Development Timeline.....</b>	<b>24</b>
7.1.1 Beginning Development.....	24
7.1.2 Full System Tests.....	24
7.1.3 Completion of Minimum Viable Product.....	25
7.1.4 Final Product Due.....	25
7.2 Delegation of Tasks.....	25
7.2.1 UX/UI.....	25
7.2.3 QA Testers.....	25
7.2.4 Backend Team.....	25
7.2.5 Database Team.....	26
7.2.6 Customer Service.....	26

# 1. Introduction

The following document discussed the potential production of an effective Movie theater ticketing system that is designed to efficiently serve customers and organize a flowing work environment for all users involved. The main goal of this document is to provide an adequate blueprint for a project that is driven to offer the most updated and systematically streamlined functionalities and features to provide the most optimal performance and potentially enhance the theater's ticketing system even more in future revs. Before delving into the heart of the specific requirements and essential elements that the system must uphold, it is crucial to discuss the purpose of the ticketing system and potential features that will be included in the scope.

## 1.1 Purpose

The main purpose for creating the Theater Ticketing System is to create a program that addresses efficient issues found in other ticketing systems. The main goal is to maintain strong capabilities and features that will facilitate customer's satisfactions via technical alterations. The ticketing system will also target optimizing resources to guarantee operational productivity.

## 1.2 Scope

The following section outlines the overall goals and benefits of the Theater Ticketing System, as well as the future goals of the program that will be discussed in depth in future iterations.

### Current scope:

- Theater Ticketing System is intended for independent theaters in the direct area. This system provides a quality system at affordable prices.
- Provide a responsive, and user-friendly experience for accessibility purposes
- Provide a metric system for upper management to keep track of ticket sales, with the ability to customize refresh rate
- Offer many features for different parties operating the system to make the Ticketing System comprehensive

### Future scope:

- First-come, first serve seating events. We will not implement a separate system to disable assigned seating until future builds, as the theater will not be holding special events at this time
- One month advance ticket purchases. Ticket purchases will be limited to a two week advance purchase. In the future, special events will allow customers to purchase up to one month in advance
- Theater availability. Only local theaters are listed, as the system does not have enough resources to handle a large number of users. We expect that traffic will be minimal initially

## 1.3 Definitions, Acronyms, and Abbreviations

In Section 3.4, the word “Belongs” refers to a hierarchical one-to-many relationship in SQL data structures. For example, if a “Showtime” object belongs to a “Movie” object, then there are many Showtimes (i.e. Theater 14-09, 9/12/2023, 9:30 AM) that correspond to one Movie (i.e. Despicable Me 2).

### 3.1.4. Communications Interface

HTTPS - Hypertext Transfer Protocol Secure


TLS - Transport Layer Security


SSL - Secure Sockets Layer

TCP/IP - Internet Protocol suite

## 1.4 References

[1] “System Description,” in *Software Design Specification*. pp. 13-26.

[2] *Test Cases*. [Online].  CS 250 Test Cases

[3] *Verification Test Plan Version 1.0*. [Online].  Verification Test Plan

## 1.5 Overview

One of the most important components for coordinating the ticketing system is to create a web application where a plethora of users can utilize and interact with. The focus from this point onward will discuss the functionality of the web application.

The ticketing system is set to be designed to meet certain requirements which create a base for the product functionality and user characteristics. The document will also discuss the very vital specific requirements that will guide the next stages of the project that will focus on the design, implementation, and testing of the ticketing system. Part of the specific requirements will address the user and communication interfaces as well as the software and hardware that will be mostly recommended to be used for the ticketing system to perform supremely.

Furthermore, the functional requirements will cover the technical operations to be executed by the ticketing system such as the process of payments and refunds as well as ticket reservations. Use Cases will also be used to determine the actors of the system and how they interact with the program. Additionally, the non-functional requirements will also be addressing the attributes of the ticketing system in terms of concepts of reliability and security. Furthermore, in this document, an outline of the class and objects that are to be used in the coding cycle is introduced.

## 2. General Description

This section will discuss the general factors that will be implemented in the ticketing system. That includes the functionality of the program and the users’ characteristics.

## 2.1 Product Functions

The Theater Ticketing System will provide methods for customers, theater employees, and system administrators to perform various operations, as outlined below.

### 2.1.1 Customers

Customers using our web application will be able to:

- query Showtimes to find relevant movie showings in their geographic vicinity,
- select their preferred seats in a selected showing,
- sign up or sign in to an existing account,
- purchase movie tickets,
- purchase premium memberships,
- cancel or refund tickets within a reasonable timeframe,
- apply for discounts on applicable purchases, and
- navigate various pages on our website that provide relevant information, such as concession menus and theater policies.

### 2.1.2 Theater Employees

Employees will be able to:

1. use our web application (using a dedicated SSO portal) to perform various occupational duties
2. manage in-person ticket transactions using a dedicated interface
3. distribute refunds as needed
4. (managers only) manually override specific operations, such as providing a refund to a customer, or applying a ticket to someone's account
5. (managers only) view certain business statistics, such as the sales for certain movies and refund/cancellation rates

### 2.1.3 System Administrators

System Administrators (and relevant parties) will have access to behind-the-scenes operations of the web server and database. They will have the ability to update and modify data structures, to modify code, to manage internal files, and to assist management with any requested tasks.

## 2.2 User Characteristics

A very significant part of the project is to determine the users' characteristics who will be interacting with the ticketing system. The ticketing system is designed to be utilized by a group of users that include the customers, theater employees and management, as well as the IT admins.

### Customers:

- can be of varying ages, abilities (*see section 3.5.6*), and demographic backgrounds
- have basic technological knowledge to use the ticketing system web application



#### Theater Employees/ management:

- need employee authentication before accessing the ticketing system web application
- need moderate technological proficiency in order to utilize the system's features

#### IT Admins:

- need Advanced technological proficiency to code and debug the system
- have access to both the customers' and employees' web portals

## **3. Specific Requirements**

The following section of the document is designed to offer a solid foundation for the ticketing system's development. That includes the coverage of precise functionalities and features as well as applied use cases.

### **3.1 External Interface Requirements**

#### **3.1.1 User Interfaces**

- drop-down lists for location selection
- input fields
  - user information
  - search
    - drop down for filters
      - accessibility options
      - genre options
      - showing time periods
      - movie lengths
- modal windows
  - user log-ins
  - system downtime notification
  - confirmation of user canceling order before confirmation
- slider bar for theater distance
- visual seating chart for ease of seat selection
- cookie settings system
- buttons
  - navigation
  - add to cart
  - payment confirmation
  - social media navigation
- movie poster carousel
- progress bar for purchase steps
  - ticket selection
  - check out as guest or sign in
  - enter info or autofill info based on log-in

- order recap
- payment confirmation
- APIs
  - reviews
  - press releases related to movies available
  - external payment (ex. Paypal)

### **3.1.2 Hardware Interfaces**

The system will require a stable internet connection between web clients, web servers, and databases. This will be accomplished by connecting the web servers and databases to Wi-Fi networks, which are connected to broadband internet.

### **3.1.3 Software Interfaces**

An updated browser will be required to access the website. The system only provides explicit support for the following browsers:

- Google Chrome (latest version)
- Firefox 116.0.3 and up
- Opera (latest version)
- Microsoft Edge (latest version)
- Internet Explorer 8 and up

### **3.1.4 Communications Interfaces**

- APIs
  - SMS, MMS, RCS for new movie releases
  - email
  - frequent polling of servers for updates
- webhooks
  - frequent communication with servers
- HTTPS used for TLS/SSL encryption for data transfer(suggestion instead of client-side hashing)
- TCP/IP to communicate with servers, send data packets

## **3.2 Functional Requirements**

For the following section, the functionality of the ticketing system is outlined to meet the users' expectations. This includes detailing the overall structure of the system in regards to reserving/purchasing processes as well as highlighting distinctive use cases.

### **3.2.1 Payment Processing**

- Payments should be processed using credit, debit, PayPal, and gift cards / reward points.
- Appropriate APIs will be accessed to manage payment on the backend.
- If an error occurs while processing a payment, the user will be notified and the purchase will be canceled.

### **3.2.2 Temporary Ticket Reservations**

- Customers should be able to reserve seats in their virtual cart for up to 5 minutes before actually purchasing them.
- When a user selects a seat, the web server will take note of the time stamp and then remove the reservation after 300 seconds (5 minutes).
- Only the user who “reserved” a seat may purchase it during the 5-minute window.

### **3.2.3 E-Mail Notifications**

- Receipts and notifications should be sent to customers through E-Mail.
- Use an API to send emails to users to remind them of showtimes for tickets they purchased, as well as occasional deals and events.
- Emails should be sent securely from the theater’s own domain.

### **3.2.4 Refund Processing**

- Refunds should be payable via theater gift cards.
- Gift cards will be applied by adding “Reward Points” to a user’s account in the database.
- Refunded points are non-transferable.

## **3.3 Use Cases**

Use Cases are a vital way to structure the functionality of the ticketing system and how it will proceed to interact with the users to reach the requirement goals. Below are examples of use cases that could give a clear blueprint of how the users will interact with the ticketing system. These examples will discuss the system’s actors including what each user has the authority to access and what relationships connect them with one another to ensure that the ultimate project will meet the intended expectations.

### **3.3.1 Use Case #1**

The Primary Actor in this case is the Customer. Customers have the ability to:

- create an Account by filling in their contact information including name, address, email, phone number, and credit card info
- log into the user portal or as a guest
- update their account info such as contact, payment, and manage membership subscription
- navigate through the Movies using the search engine or filtering them based on location, alphabetical order, and other categories mentioned
- choose a movie’s time and location
- choose types of ticket (regular or deluxe)
- choose seat and request accommodation
- add tickets to cart and choose the quantity of the tickets
- add any awarded promotional codes for discounts
- place the final purchase
- view purchase history
- view order details in the profile section

- cancel the order within 24 hours of placing it
- contact support
- leave feedback and reviews

### **3.3.2 Use Case #2**

The Secondary Actor in this case are the employees/ managers of the theater. They have the ability to:

- log in via the employee portal
- update movies by adding or removing them
- adjust the show times or room numbers
- view Customer's accounts including their orders and memberships
- process the purchase made by the customer
- adjust the prices and discounts offered
- adjust the ticket numbers left
- modify and upgrade memberships
- send confirmation to customers
- manually edit the purchase made by customers such as:
  - adding tickets
  - canceling the order after the 24 hour limit
  - changing the movie
  - etc.
- contact the IT admins for system issues/ updates

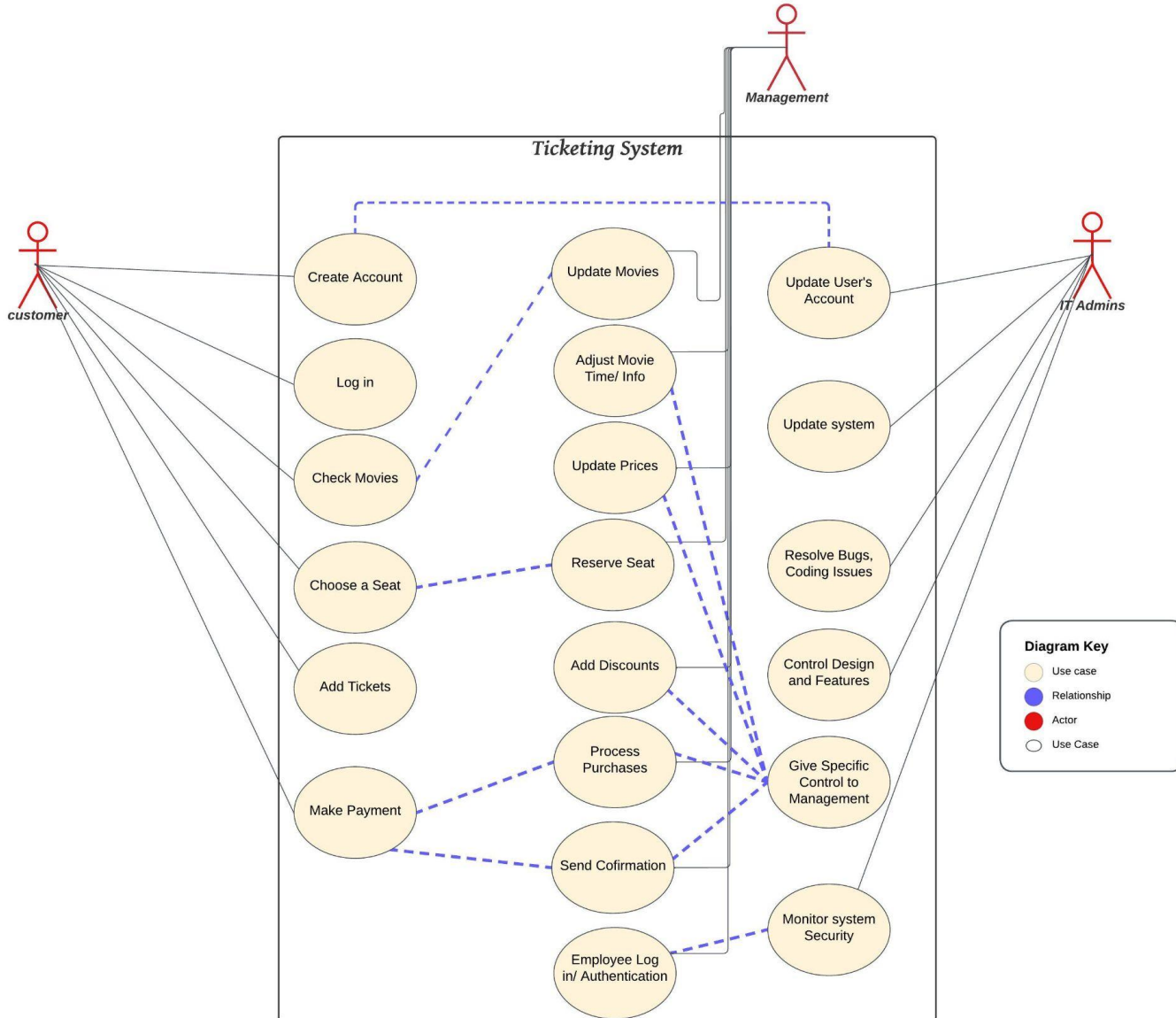
### **3.3.3 Use Case #3**

The Actor in this case are the IT Admins that monitor the overall system. They have the ability to:

- view the ticketing system from the customer's perspective and the management's perspective
- modify and control the layout of both portals
- update the overall design and features of the website
- adjust users' and employees' accounts by activating/ deactivating or permanently deleting them from the system
- shut down the system for updates or maintenance
- fix code or debug issues in the system
- give certain authority to management
  - confirming purchases
  - viewing orders
  - reserving seats
  - updating info
  - etc.
- monitor the system's security through login authentication for customers and employees

- ensure that each user has limitations to what they can view or access

## \*Ticketing System Use Cases Diagram



### 3.4 Classes / Objects

All classes and objects described in this document will exist in our SQL database structures, as well as in backend design using Java classes and Python data structures. The database will be transmitted to web clients from the web server in JSON format, to be parsed using integrated JavaScript data structures.

More information about specific classes, methods, and attributes can be found in Section 6 on pg. 27.

## 3.5 Functional Requirements

The non-functional aspect of developing the ticketing system is vital as it highlights the system's attributes. That includes elements such as but not limited to enhancing the performance, reliability, and security to offer the best experience for the users.

### 3.5.1 Performance

- The system should be able to quickly index the database and produce relevant results to any authorized query from a user
- Web servers shall have a response time of no more than 200ms on average

### 3.5.2 Reliability

- Databases containing information about available seats and showings should be updated constantly, and all servers should share identical information
- The database should be held on as few servers as possible, to eliminate discrepancies between them

### 3.5.3 Availability

The Theater Ticketing System will strive to maintain a percentage of 99.5% availability. Certain measures will be taken to ensure the system stays live in order to prevent substantial interruption.

- base formula to be determined to estimate unplanned downtime during the month
- set up service to notify System Admins of an outage
- ensure proper maintenance of servers

### 3.5.4 Security

It is crucial for the ticketing system to ensure the security and the integrity of the program and the user's information. Therefore, the project will uphold the following:

- Data Security
  - use of hashing and HTTPS for securing login information such as passwords
  - including two step authentication or OTP for login
  - use of verification codes for password or username resets
  - email confirmation for verifying the user's contact info and/ or securing a purchase
  - ensure the security of encrypting messages from client's server
  - requires specific authentication for management/ employees and IT admins such as scanning employee ID.
- Agreements
  - require users to sign user's agreement / terms of service
    - when signing up for an account
    - when making a purchase
  - include third party documentation and references to policies and federal acts.
- Backup and Recovery
  - backup and store data to prevent data loss during incidents such as system shut downs
  - allows users to permanently delete an account or restore a deactivated one

### **3.5.5 Maintainability**

- set maintenance downtime once a week for updates with no more than 4 hours downtime
- downtime conducted during off-peak hours for minimal interruptions
- unplanned downtime addressed as soon as possible with service to alert when system is down
- user reports sorted by priority and addressed accordingly
  - high priority: addressed asap
  - medium priority: once a week, off-peak hours
  - low priority: once a month, off-peak hours
- metrics to monitor health of system

### **3.5.6 Portability**

- Portability will be achieved by using Java for most backend operations and using HTML / CSS / JavaScript for frontend design. These languages are fairly universal, and can be used from almost any modern web browser.
- The website will be accessible from most web browsers, and will include dedicated support for Chrome, Safari, Firefox, Opera, and Edge. This will allow it to be run on basically any internet-connected device.
- The website will be adapted for mobile users by using larger text and more mobile-friendly GUI features, as well as accommodating a portrait orientation.
- Accessibility features will be provided to ensure that all users are capable of using the website.

## **3.6 Inverse Requirements**

The following is a compiled list of features that will not be available currently or in the foreseeable future. This is subject to change based on client's future needs and will be amended accordingly:

- The system will provide user feedback and email support. However, there are no plans to implement a live chat function for users to receive assistance.
- The system will not provide instruction on troubleshooting the Ticketing System. Any outages on our end will be announced, and it will be up to the user to solve any connection issues on their end.
- Until there are requests for the program to be scaled up to handle more traffic, the system is only designed to handle no more than 50 \* # of theater locations, but AWS will handle the minutiae.
- Concession purchases will not be available through the system.

## **3.7 Design Constraints**

- for better experience, recommend user upgrades browser to latest version
  - have detection system that alerts user
  - show list of supported browsers and versions
- system unable to handle more than 1,000 users at once



- no advanced system to reliably detect bot purchases
- future refactoring necessary to increase responsiveness of system
- minimal budget of 500k may not be suitable for project ambitions
- 4 months for viable prototype to user test with a total of 10 months to a year for release

### 3.8 Logical Database Requirements

- A database will be hosted using a Microsoft Azure “Business Critical” SQL Database (~\$1,000 / month)
- Data will be formatted in the most storage-efficient manner possible
- All data will be expressed numerically when possible
- Commonly-queried non-numerical data (i.e. movie title, genre) will be stored using hash tables for quick indexing
- Repetitive attributes (i.e. genre) will be represented using their own data structures and objects
- Class hierarchy will be used for cross-referencing between database entries
- Data integrity will be preserved
- Confidential data (i.e. passwords, credit cards) will be encrypted one-way using hash functions
- Data usage:
  - We expect approx. 120,000 ticket sales per day between 20 theater locations, with each sale corresponding to roughly 128 bytes of data. This adds up to approx. 5.6GB of ticket information per year.
  - We expect to use approx. 1.0GB per year to store information about user accounts, including administrators and authorized employees.
  - We expect to use no more than 20MB per year to store showtimes, theater locations, movies, and genres.
  - Therefore, we expect to use no more than 6.62GB of storage per year, which means that a 128GB central hard drive would last us ~19 years. Because of this, it is not necessary to use more than 128GB database, since we can purge old data after a few years.
  - To account for this, we will rent an Azure Business Critical SQL Database with 128GB of storage for approx. \$1,000/month
- Query volume:
  - Given the high expected volume of ticket sales (120,000 per day), we anticipate at least 500,000 database queries will be necessary. This is covered with the aforementioned Azure database, which does not directly limit our query volume except by the limitations of our dedicated CPU.
- Use TLS/SSL for added encryption

## [Assignment #2 – Design Specification] begins here

### 4. System Description

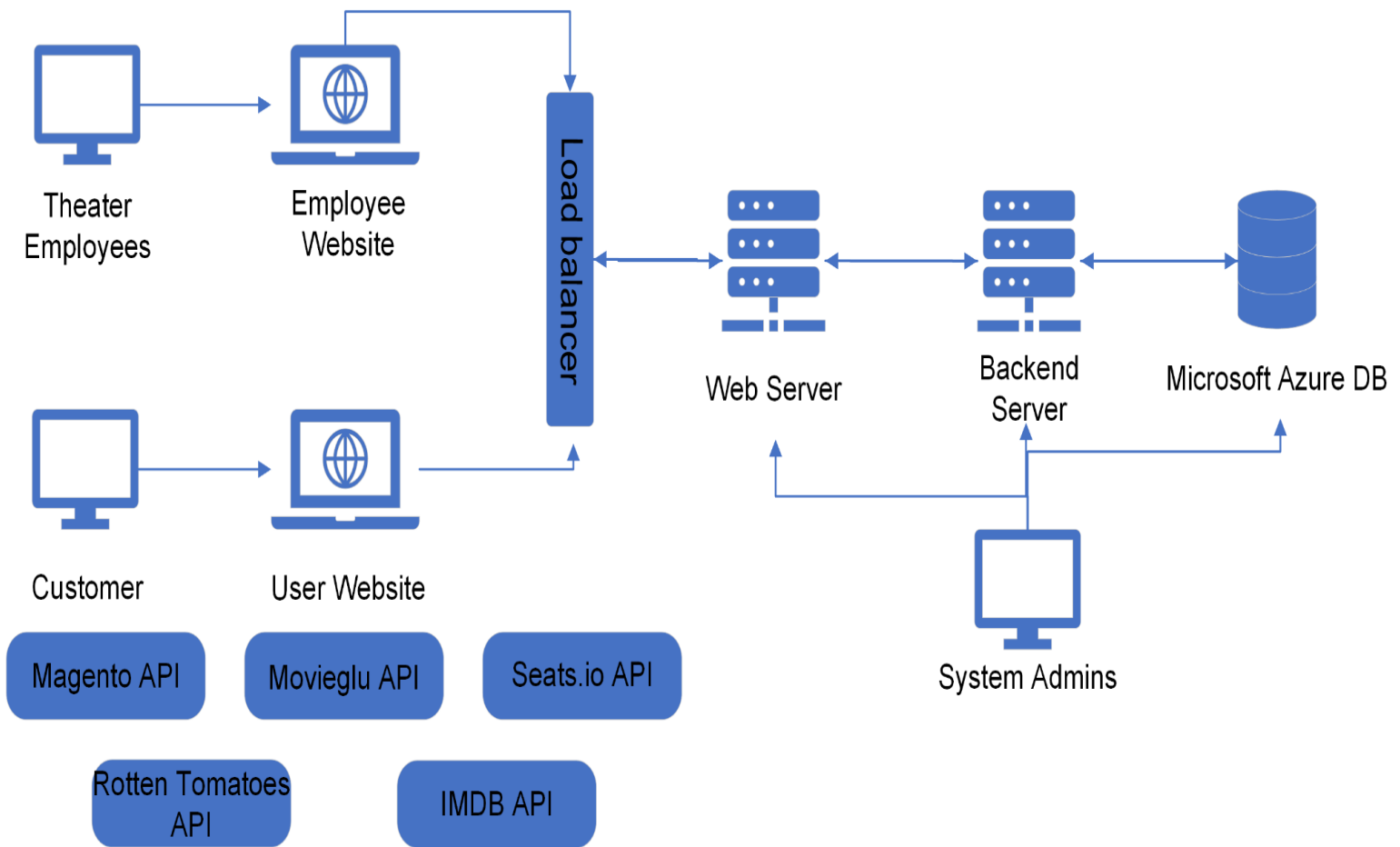
#### 4.1 Brief overview of system

The following Modules detail the Software Design Specification for the ticketing system and offer a meticulous outline for the architectural component of the project to prepare for the implementation phase. That includes analyzing the relationships between the system's components and how the data flows between them to the safe processing of tickets, purchases, and other measures of user interactions.

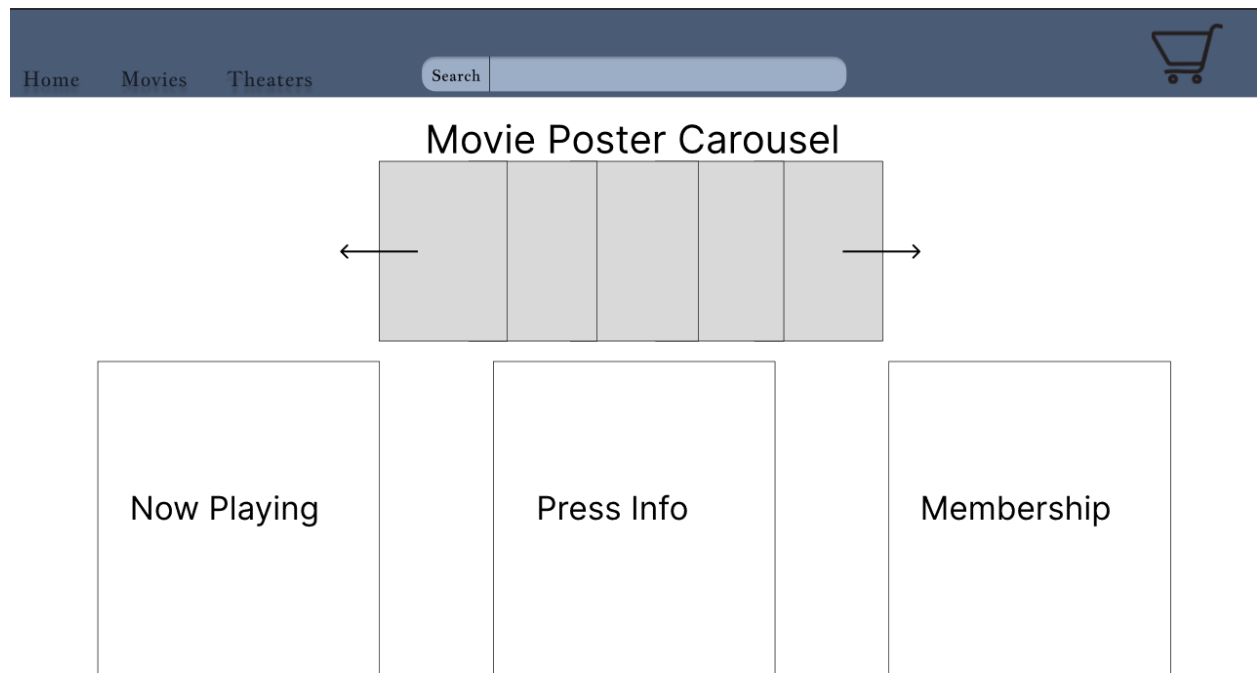
### 5. Software Architecture Overview

To better understand the functionality of the ticketing system, presenting a visual diagram of the system's architecture and design is crucial. This module delves into a comprehensive review of these various diagrams.

#### 5.1 Architectural diagram of all major components

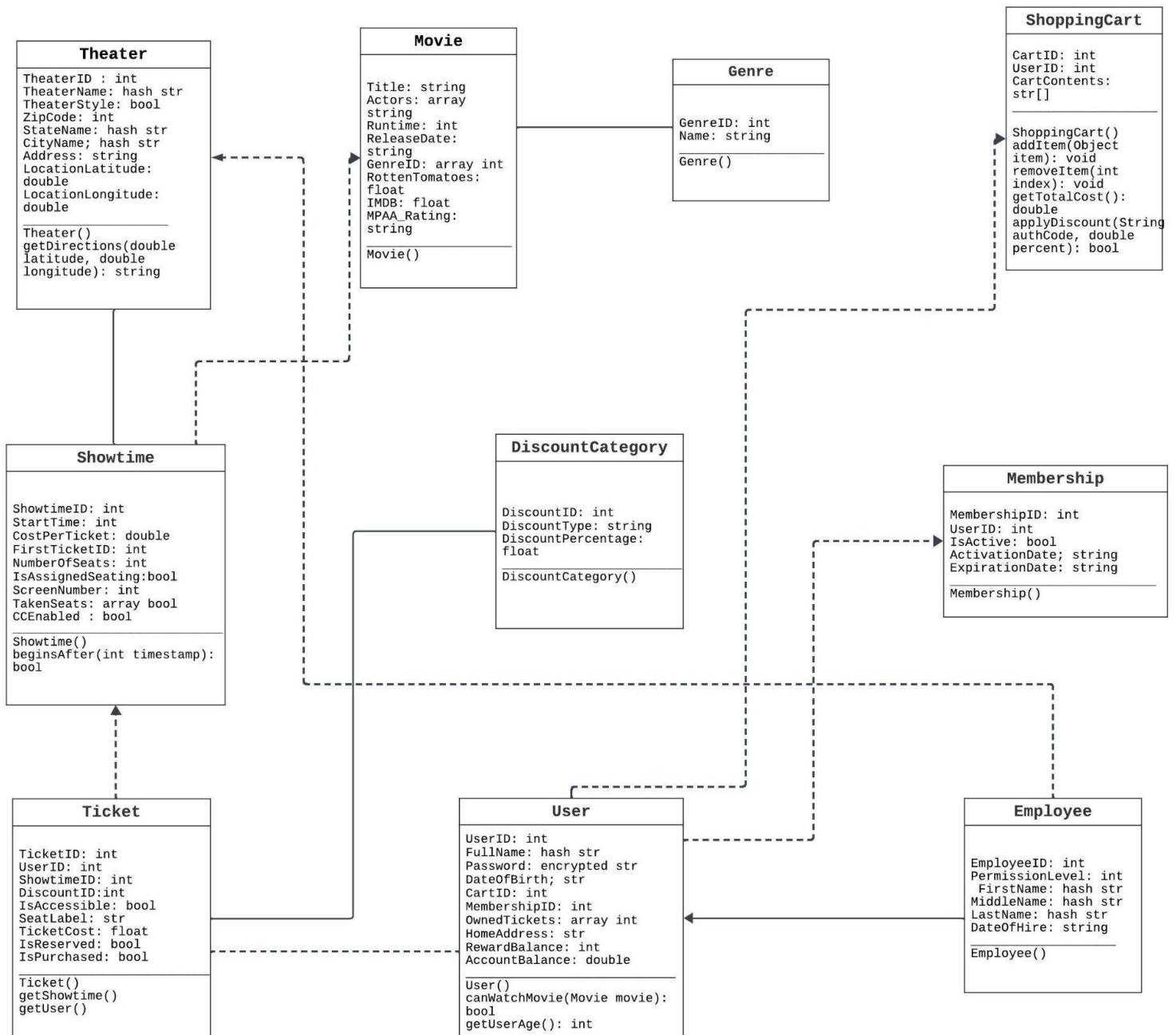


## 5.2 UI Landing Page Mockup



### **5.3 UML Class Diagram**

### UML Class Diagram



## 6. Classes, Objects, & Attributes

Described below are the classes, objects, and attributes that will be needed for our Theater Ticketing System. These objects will exist within our database, as well as within our backend and frontend processing as Java classes and JavaScript data structures.

The database will be structured using SQL and will be updated by authorized users, such as system administrators. The backend Java and frontend JavaScript programs will execute automatically according to the system's design specifications and relevant data in the database. Methods will only be executable by the client or web server, as specified, and will not exist within the database.

Objects in the database will interact with each other through specified SQL relationships, typically by referencing the unique numerical identifier(s) of the parent class(es). Objects in the internal software will interact through polymorphism and hierarchical class structures.

The prefix (HASH STR) refers to a String that is serialized using a two-way hash function when stored in the database, to allow for quicker indexing. In OOP, these will be represented with traditional String instances.

Every non-static class attribute will be `private` unless otherwise specified, and will be readable / writable using "get"/"set" methods, respectively. For example, the get/set methods for the "Theater" class are outlined below:

- `[void] set*Attribute*(*type* value) – e.g. setZipCode(int zip_code)`
  - Assigns the value of `*Attribute*` to `value`
  - Valid attributes: (INT) TheaterID, (STR) TheaterName, (BOOL) TheaterStyle, (INT) ZipCode, (STR) StateName, (STR) CityName, (STR) Address
- `[*type*] get*Attribute*() – e.g. getZipCode()`
  - Returns the value of `*Attribute*`
  - Valid attributes: (INT) TheaterID, (STR) TheaterName, (BOOL) TheaterStyle, (INT) ZipCode, (STR) StateName, (STR) CityName, (STR) Address, (DOUBLE) LocationLatitude, (DOUBLE) LocationLongitude

### 6.1 Theater

#### 6.1.1 Attributes

- `*(INT) TheaterID` – Unique sequential numerical identifier for a theater, starting at 1
- `(HASH STR) TheaterName` – Unique string representation of a theater's name, i.e. "AMC Mission Valley"
- `(BOOL) TheaterStyle` – The "style" of theater. 0 = standard, 1 = dine-in
- `(INT) ZipCode` – The geographic ZIP code in which the theater exists

- (HASH STR) StateName – The name of the state in which the theater exists
- (HASH STR) CityName – The name of the municipality in which the theater exists
- (STRING) Address – The physical address of the movie theater, i.e. “1234 Hanna Road”
- (DOUBLE) LocationLatitude – The geographical latitude of the theater’s physical address
- (DOUBLE) LocationLongitude – The geographical longitude of the theater’s physical address
- (SHOWTIME[])

### 6.1.2 Functions

- Used to identify a specific “showtime” object based on its physical location
- Used to search for nearby theaters based on various criteria, such as state, city, and ZIP

### 6.1.3 Methods

- [Theater] Theater() – Constructor. Creates a blank “Theater” instance.
- [String] getDirections(double latitude, double longitude) – e.g. getDirections(49.2842, -10.329)
  - Returns a hyperlink, in String format, to the device’s native navigation application (Google Maps in a web browser by default), set up to provide directions from the user’s current location to the theater’s location
    - If the user’s current location is not specified, the map application will direct the user to enter their location

## 6.2 Showtime

### 6.2.1 Attributes

- \*(INT) ShowtimeID – Unique sequential numerical identifier for a showtime, starting at 1
- (INT) StartTime – UNIX time stamp for the showtime’s beginning.
- (DOUBLE) CostPerTicket – Undiscounted cost (in \$) for an adult ticket for this showing
- (INT) FirstTicketID – The ID # of the first ticket associated with this showtime.
- (INT) NumberOfSeats – The number of seats in the theater where this Showtime is being held
- (BOOL) IsAssignedSeating – 0 = seating is first-come-first-serve, 1 = seating is reserved
- (INT) ScreenNumber – The number of the actual theater screening room within a Theater
- (ARRAY BOOL) TakenSeats – ordered list of boolean values representing whether each seat has been purchased already or not
- (BOOL) CCEnabled – whether or not Closed Captions are enabled for this showing

### 6.2.2 Functions

- “Belongs” to a Movie
- “Belongs” to a Theater
- Used to organize Tickets based on the specific showtime they refer to

- i.e. if *FirstTicketID* is 950, and *NumberOfSeats* is 100, then Tickets 950-1049 will belong to this Showtime.
- Used to query nearby showings of a certain movie, so that users can search for showings by time and date within a specified geographic area

### 6.2.3 Methods

- [Showtime] Showtime() – Constructor. Creates a blank “Showtime” instance.
- [Bool] beginsAfter(int timestamp) – e.g. beginsAfter(1604360704)
  - Returns True if the showtime begins after the specified Unix timestamp. Returns False otherwise.
- [Int] getNumberOfTicketsSold()
  - Returns the total number of tickets sold by indexing TakenSeats[].
- [String] getStartDate()
  - Returns the showtime’s date in Weekday, MM/DD/YYYY format.
- [String] getStartTime()
  - Returns the showtime’s start time in HH:MM (AM/PM) format.

## 6.3 Movie

### 6.3.1 Attributes

- \*(INT) MovieID – Unique sequential numerical identifier of the movie
- (HASH STR) Title – The name of the movie, hashed for more efficient indexing
- (ARRAY STR) Actors – CSV list of main actors appearing in the movie
- (INT) Runtime – The duration of the movie in minutes
- (STR) ReleaseDate – MM/DD/YYYY, the date the movie was first released in theaters
- (ARRAY INT) GenreID – The ID #(s) of the movie’s Genre(s)
- (FLOAT) RottenTomatoes – Consensus rating for the movie on Rotten Tomatoes. Updates daily based on continuous feedback.
- (FLOAT) IMDB – Consensus rating for the movie on IMDB. Updates daily based on continuous feedback.
- (STR) MPAA\_Rating – The movie’s rating by the MPAA, such as PG-13 or R.

### 6.3.2 Functions

- Allows several Showtimes to point to the same Movie in memory, to share common attributes
- Allows users to search for a movie before determining which theater/showtime to select

### 6.3.3 Methods

- [Movie] Movie() – Constructor. Creates a blank “Movie” instance.



## 6.4 User

### 6.4.1 Attributes

- \*(INT) UserID – Unique identifier for a user with an account
- (HASH STR) FullName – The full name of the user, hashed for easy access
- (ENCRYPTED STR) Password – The user’s password, with one-way hash encryption
- (STR) DateOfBirth – MM/DD/YYYY, the user’s birth date (used for age verification based on movie rating)
- (INT) CartID – Reference to the user’s cart
- (INT) MembershipID – The unique identifier of the user’s Premium Membership. 0 if no membership.
- (ARRAY INT) OwnedTickets – List of ticket ID’s owned/purchased by the user
- (STR) HomeAddress – The user’s full address, such as 1234 Hanna Blvd. San Diego, CA 92182.
- (INT) RewardBalance – The number of reward points the user has. 1 point = \$0.01
- (DOUBLE) AccountBalance – The user’s current account balance (\$), i.e. from gift cards and returns/refunds

### 6.4.2 Functions

- Facilitates the purchase of tickets and the handling of transactions
- Allows for administrators to review an individual’s order history and premium membership

### 6.4.3 Methods

- [User] User() – Constructor. Creates a blank “User” instance.
- [Bool] canWatchMovie(Movie movie) – e.g. canWatchMovie(StarWars)
  - Returns True if the user is old enough to purchase tickets for the specified movie, based on its MPAA rating (R, PG, PG-13, etc.). Returns False otherwise.
- [Int] getUserAge()
  - Returns the user’s age in years, based on their account’s DateOfBirth.

## 6.5 ShoppingCart

### 6.5.1 Attributes

- \*(INT) CartID
- (INT) UserID – The ID of the user to which the cart belongs
- (STR[]) CartContents – Array of all items in cart. Ex:  
{Tix.305,Tix.306,Tix.307,PremMbrsp}

### 6.5.2 Functions

- Allows a user to add several tickets / other items to their “cart” and check out all at once

### 6.5.3 Methods

- [ShoppingCart] ShoppingCart () – Constructor. Creates a blank “ShoppingCart” instance.
- [void] addItem(Object item)
  - Adds an item of type Object (any subclass) to the cart’s list of contents.
- [void] removeItem(int index)
  - Removes the item in the shopping cart at index [index]
- [double] getTotalCost()
  - Returns the total cost of everything in the cart.
- [Bool] applyDiscount(String authCode, double percent)
  - Applies a [percent]% discount to the ShoppingCart, with authorization code [authCode] to represent the discount’s identifier (i.e. coupon code, military discount).
  - Returns True if the discount is successfully applied. Returns False otherwise.

## 6.6 Membership

### 6.6.1 Attributes

- \*(INT) MembershipID – The unique ID # of the Membership
- (INT) UserID – The ID # of the user owning the membership
- (BOOL) IsActive – Whether or not the membership is currently active (0=false, 1=true)
- (STR) ActivationDate – MM/DD/YYYY
- (STR) ExpirationDate – MM/DD/YYYY, when the membership did/will expire
  - Can be modified by renewing the membership (can be automatic)

### 6.6.2 Functions

- Contains basic information about a user’s premium membership, if they have one
- Exists as a distinct object to handle expired memberships and enable transferral of membership between users, with administrator approval

### 6.6.3 Methods

- [Membership] `Membership()` – Constructor. Creates a blank “Membership” instance.

## 6.7 Ticket

### 6.7.1 Attributes

- \*(INT) `TicketID` – Unique numerical sequential identifier
- (INT) `UserID` – The ID of the user who owns the ticket
- (INT) `ShowtimeID` – The ID of the showtime that the ticket refers to
- (INT) `DiscountID` – The ID of the discount applied to this ticket. 0 for no discount or ticket not yet purchased.
- (BOOL) `IsAccessible` – Whether the seat is wheelchair accessible (will display as such when purchasing ticket). 0 = not wheelchair accessible, 1 = wheelchair accessible
- (STR) `SeatLabel` – The labeled letter/number of a seat, such as “A6” or “F12”
  - Used by the system to display the seat’s location within the theater, when purchasing tickets
- (FLOAT) `TicketCost` – Default, undiscounted cost of the ticket. May vary within a certain showing, based on variation in seating, i.e. luxury vs. standard seats
- (BOOL) `IsReserved` – Whether or not the ticket is temporarily “reserved”, meaning that a user has selected it and added it to their cart (but hasn’t purchased it yet). The user gets 5 minutes to purchase the ticket before it becomes available to other users.
- (BOOL) `IsPurchased` – Whether or not the ticket has been purchased by a user, thereby making it unavailable.

### 6.7.2 Functions

- The system creates “Tickets” for each showtime according to the number of seats required for that showtime. Tickets may then be purchased by users. A ticket represents an individual seat in an individual showing at a theater.
- Allows a user to keep track of the tickets they own
- Allows the theater to keep track of available seating and to track statistics

### 6.7.3 Methods

- [Ticket] `Ticket()` – Constructor. Creates a blank “Ticket” instance.
- [Showtime] `getShowtime()`
  - Returns the Showtime associated with this Ticket
- [User] `getUser()`
  - Returns the User who purchased this Ticket

## **6.8 DiscountCategory**

### **6.8.1 Attributes**

- \*(INT) DiscountID – Unique numerical sequential identifier
- (STR) DiscountType – i.e. “Military”, “Youth”, “Student”
- (FLOAT) DiscountPercentage – Number from 0 to 100, i.e. 36.3

### **6.8.2 Functions**

- Allows for consistent application of discounts, which are applied to each ticket according to the characteristics of the user who the ticket is for.

### **6.8.3 Methods**

- [DiscountCategory] DiscountCategory() – Constructor. Creates a blank “DiscountCategory” instance.

## **6.9 Genre**

### **6.9.1 Attributes**

- \*(INT) GenreID – Unique numerical sequential identifier
- (STR) Name – The name of the genre, i.e. “Comedy”, “Drama”, “Horror”

### **6.9.2 Functions**

- Allows a movie to “belong” to one or more Genre

### **6.9.3 Methods**

- [Genre] Genre() – Constructor. Creates a blank “Genre” instance.

## **6.10 Employee**

### **6.10.1 Attributes**

- \*(INT) EmployeeID – Unique numerical sequential identifier
- (INT) PermissionLevel – Refers to the employee’s level of permission. 0 = no permissions (i.e. suspended / terminated), 1 = low-level employee (i.e. cashier), 2 = theater manager, 3 = system administrator, 4 = corporate executive
- (HASH STR) First Name
- (HASH STR) Middle Name
- (HASH STR) Last Name
- (STR) DateOfHire

### 6.10.2 Functions

- Allows employees to handle certain tasks that are unavailable to customers
- Allows for permissions to be delegated among employees according to the requirements of their job duties
- Allows for secure entry to and administration of databases and web servers when necessary

### 6.10.3 Methods

- [Employee] Employee () – Constructor. Creates a blank “Employee” instance.

## 7. Task Management and Development Timeline

It is necessary to partition tasks among different development teams and individual developers to ensure that adequate progress is made in a timely manner. (Note: Pretend we have a fake development team)

### 7.1.1 Beginning Development

- Includes unit tests
- Consultations with the UX/UI team to establish a visual mockup and functional relationship between different parts of the system. After review, our Frontend and Backend team will collaborate to execute this vision. Design is subject to change throughout the process
- Each module and the interaction between them will be tested by QA to ensure satisfactory final product
- Security will be thoroughly tested to ensure final product is safe for consumers to use
- Live customer service agents will be available 9 AM - 5 PM to handle reports with appropriate response time

### 7.1.2 Full System Tests

- Functional:
  - Unit testing
  - Integration Testing
- Non-Functional:
  - Performance testing
  - Usability testing
  - Load testing
  - Security Testing
  - Scalability testing
  - Functionality testing
  - Recovery testing
- Maintenance

### **7.1.3 Completion of Minimum Viable Product**

In order to deliver a Minimal Viable Product that achieves our vision, the following will be taken into account to complete it:

- User research to find what features of similar systems work/don't work
- Small launch with a handful of theaters to gain real-time user feedback
- Estimated final system completion is 1 month after feedback; subject to change based on unknown risks
- Analysis of costs necessary to construct the system with our current team

### **7.1.4 Final Product Due**

- The project's timeline incorporates building a viable prototype that is estimated to take 4 months to examine user testing
- Including the testing interval, the expected deadline for release will be anywhere from 10 months to a year from now

## **7.2 Delegation of Tasks**

### **7.2.1 UX/UI**

- Research and analyze user requirements
- Responsible for making GUI to meet those requirements
- Collaborate with rest of team to ensure seamlessness

### **7.2.2 Frontend Team**

- Responsible for developing frontend code that will be rendered in a user's web browser
- Integration of APIs from the front end, such as the shopping cart and email system, to fit visual expectations
- Responsible for executing the final vision from consultations with UX/UI team
- Implement proper user authentication and require security checks and cool-downs for bolstered security; collaborate with Back end for seamlessness

### **7.2.3 QA Testers**

- Oversees all testing
- Collaborate with appropriate teams to provide feedback for bug and QoL fixes

### **7.2.4 Backend Team**

- Responsible for developing software architecture that operates on the web server
- Responsible for ensuring the security of the system as the intermediary between the database and the web client

### **7.2.5 Database Team**

- Responsible for setting up database infrastructure and working with the Backend Team to achieve parity between internal classes and database classes

### **7.2.6 Customer Service**

- Responsible for establishing communication with users/theater employees
- Mediator for resolution of issues