Homework 4 starts on page 34

Software Design Specification Template

Theater Ticketing System

Software Design Specification

Version 3

November 2, 2023

Group #3
Jake Stonebraker
Jose Urrutia
Emre Yikilmaz

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

Revision History

Date	Description	Author	Comments
October 5th, 2023	Version 1	Group 3	<First Revision>
October 19, 2023	Version 2	Group 3	<Second Revision> <Added Verification Test Plan>
November 2, 2023	Version 3	Group 3	<Third Revision> <Added Database Management Strategy>

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

The introduction of the Software Design Specification (SDS) provides an overview of the SDS with purpose, product description, scope, and the objective of this document. The aim of this software design specification is to describe the software organization and development plan.

1.1 Purpose

The purpose of the document is to define the design for the overall software architecture and implementation of the theater ticketing system . The classes, data structures, together with all major functions and their parameters will be organized and specified. It should help the developers to adequately implement the software components as specified in this document.

1.2 Scope

This software design specification is to be used by our Software Engineers as a template definition of the design and architecture that is to be used in implementing the Theater Ticketing System.

1.3 Objective

The Theater Ticketing System is meant to provide the customer and theater employee the ability to view, select, reserve, purchase, and refund tickets. Customer accounts will also be created (if customer wishes to do so) to remember personal information, preferences, and payment methods.

2. System Overview

2.1 Product Perspective

The product enables users (customer/employee) to view, sort, select, reserve, purchase, and refund tickets for theaters the web browser and purchase tickets online. Employees of the cinema can see and manage bookings from ticket buyers. Besides that, cinema administrators can manage user accounts. Using the online ticketing system the ticket purchasing process becomes easier and more efficient for both ticket buyers and the cinemas. Enhanced features like account registration enable opportunities for loyalty programs and discounts for ticket buyers.

2.2 Product Functions

*2.2.1 **Theater class functions**: displayTheaterDetails()*

2.2.2 ***Person class functions****: n/a*
2.2.3 ***Account class functions****: resetPassword(), validate_Username()*
2.2.4 ***Admin class functions****: addMovie(), addShowtime(), deleteUser()*
2.2.5 ***Customer class functions****: getReservation(), initiateRefund()*
2.2.6 ***Employee class functions****: searchReservation(), initiateRefund()*
2.2.7 ***Showtime class functions****: displayShowtime()*
2.2.8 ***Movie class functions****: sortMovies(), getMovieRating()*
2.2.9 ***Seat class functions****: isOccupied(), occupy(), vacate()*
2.2.10 ***Reservation class functions****: calculatePrice(), addSeat(), removeSeat(), timeLimit(), cancelReservation(), createReservation()*
2.2.11 ***Ticket class functions****: getUser(), getShowTime(), getSeatInfo(), checkTicketType()*
2.2.12 ***Ticket type functions:*** *verifyDiscount(), setPrice()*
2.2.13 ***Payment class functions****: wipeCardInfo(), processPayment(), refund()*
2.2.14 ***Receipt class functions****: generateReceipt()*

2.3 User Characteristics

The intended user is any customer looking to reserve and purchase a ticket online or at a kiosk at any of the theater locations. Employees will be able to make reservations and purchases on customer's behalf when necessary.

2.4 General Constraints

This application will only run on a system that supports a GNU C++ compiler. Is meant to be run on the three most popular web browsers: Google Chrome, Firefox, Safari.

2.5 Assumptions and Dependencies

The user will have basic knowledge of how to navigate a website using a mainstream web browser, have a valid email address, locate their desired theater, select the movie of interest, enter personal and payment information, and present their ticket at the movie theater.

3. System Architecture

3.1 UML Class Diagram

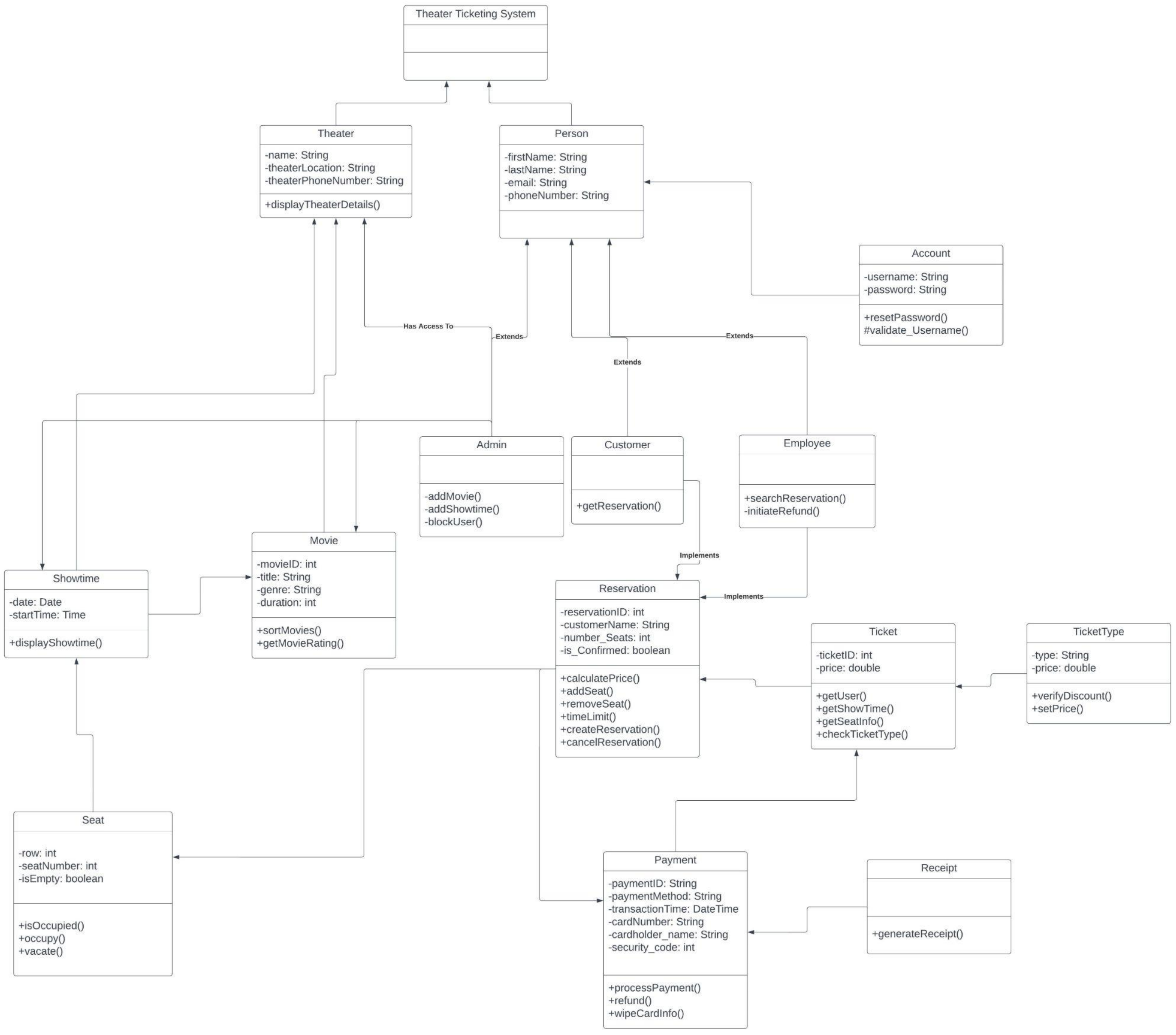

Figure 1: UML Class Diagram containing the class objects and their relationships

3.2 Individual Classes of System

3.2.1 Theater

the Theater class is will be the object class that can create instances of the individual theaters within the theater ticketing system

3.2.1.1 Attributes of Theater class

3.2.1.1.1 theaterName
the name of the theater will be held in data type: String

3.2.1.1.2 theaterLocation
the address of the theater will be held in data type: String

3.2.1.1.3 theaterPhoneNumber
the phone number of the specific theater will be held in data type: String

3.2.1.2 Functions available in Theater class

3.2.1.2.1 displayTheaterDetails()

this function will serve to display the attributes of the specific theater

Arguments:
this function will contain no arguments as it will directly handle the attributes within the class

Return Type:
this function will have a void return type as it will simply display the details of the theater in a neat manner

3.2.2 Person
the Person class will be the object class that will create instances of each individual actor within the Theater Ticketing System

3.2.2.1 Attributes of Person class

3.2.2.1.1 firstName
the first name of the person will be held in data type: String

3.2.2.1.2 lastName
the last name of the person will be held in data type: String

3.2.2.1.3 email
the email of the person will be held in data type: String

3.2.2.1.3 phoneNumber
the phone number of the person will be held in data type: String

3.2.3 Account
the Account class will have an associative relationship with the Person class as each instance of the person class will have an account corresponding to the type of "Person"

3.2.3.1 Attributes of Account class

3.2.3.1.1 username

the username of the instance of the person class will be held in data type: String

3.2.3.2.1 password

the password of the instance of the person class will be held in data type: String

3.2.3.2 Functions available in Account class

3.2.3.2.1 resetPassword()

this function will allow the instance of the person class (whether a customer, admin, or employee) to reset their password when they have forgotten it or wish to change it for security reasons

Arguments:

-new_password (str): the new password for the user

Return Type:

-boolean: Will return true if the password reset is successful, false if not successful

3.2.3.2.2 validate_Username()

this function will check whether or not the username given by the user is taken by another user

Arguments:

there are no arguments required for this function as it will directly deal with the username attribute of this class

Return Type:

-boolean: will return true if username is available, false if username is not available

3.2.4 Admin

the Admin class will be the specific object class that creates instances of each individual that has administrator privileges in the Theater Ticketing System. They will have access to the Showtime and Movie classes in order to add movies and showtimes for each Theater object

3.2.4.1 Attributes of Admin class

the admin class will inherit the attributes of the Person class

3.2.4.2 Functions available in Admin class

3.2.4.2.1 addMovie()

this function will allow the admin to create the instance of a Movie Object that will be added to the list of movies for the specific Theater object they want to add it to

Arguments:

this function has no arguments as the admin class will have access to the movie class and its attributes

Return Type:

-void: the function will return nothing

3.2.4.2.2 addShowtime()

this function will allow the admin to create the instance of a Showtime Object that will be added to the list of showtimes for the specific Movie object they want to add it to

Arguments:

this function has no arguments as the admin class will have access to the Showtime class and its attributes

Return Type:

-void: the function will return nothing

3.2.4.2.3 deleteUser()

this function will allow the admin to delete Customer Class instances in the person database

Arguments:

this function has no arguments as the admin class will have access to the person class and its attributes

Return Type:

-void: the function will return nothing

3.2.5 Customer

the Customer class will be the specific Object class that will create instances of each individual wanting to purchase tickets using the Theater Ticketing System

3.2.5.1 Attributes of Customer class

the Customer class will inherit the attributes of the Person class

3.2.5.2 Functions available in Customer class

3.2.5.2.1 getReservation()

this function will allow the Customer to search for their existing reservation

Arguments:

-reservationID (int): This function will use the reservation ID to use in a search algorithm in the Theater Ticketing System database

Return Type:

-boolean: the function will return true if the reservation is found or false if it is not found.

3.2.6 Employee

3.2.6.1 Attributes of Employee class

the Customer class will inherit the attributes of the Person class

3.2.6.2 Functions available in Employee class

3.2.6.2.1 searchReservation()

this function will allow the Employee to search for a customer's reservation in order to print tickets or initiate a refund.

Arguments:

-reservationID (int): This function will use the reservation ID to use in a search algorithm in the Theater Ticketing System database

Return Type:

-boolean: the function will return true if the reservation is found or false if it is not found.

3.2.6.2.2 initiateRefund()

this function will allow the Employee to initiate a refund to the Customer if a reservation is found and it abides by the refund policy as described in the functional requirements. If the argument captured in the parameters is true, the function will initiate the refund

Arguments:

-reservationFound (bool): This function will use the reservation ID to use in a search algorithm in the Theater Ticketing System database and capture the boolean from the searchReservation() function within the same Employee Class

Return Type:

-void: the function will return nothing.

3.2.7 Showtime

an instance of the Showtime class will be created by the administrator to be added under the movie database for the specific instance of the Theater class. It will contain the date and start time for the specific movie. It will also have access to the attributes of the Movie class that will be displayed to the user

3.2.7.1 Attributes of Showtime class

3.2.7.1.1 date

the date will contain the date of the instance of the Showtime object using data type: Date

3.2.7.1.2 startTime

the startTime will be initialized with the data type: Time

3.2.7.2 Functions available in Showtime class

3.2.7.2.1 displayShowtime()

this function will allow the Employee to initiate a refund to the Customer if a reservation is found and it abides by the refund policy as described in the functional requirements. If the argument captured in the parameters is true, the function will initiate the refund

Arguments:

-reservationFound (bool): This function will use the reservation ID to use in a search algorithm in the Theater Ticketing System database and capture the boolean from the searchReservation() function within the same Employee Class

Return Type:

-void: the function will return nothing.

3.2.8 Movie

the Movie class will hold the information necessary for the primary actor to view and search for the specific movie they want to see

3.2.8.1 Attributes of Movie class

3.2.8.1.1 movieID

each movie instance will have a unique ID that will have a data type: int

3.2.8.1.2 title

each movie instance will have a title that will use the data type: String

3.2.8.1.3 genre

each movie instance will have a genre classifier that will use the data type: String

3.2.8.1.4 duration

each movie instance will have a run time that will be in minutes using the data type: int

3.2.8.2 Functions available in Movie class

3.2.8.2.1 sortMovies()

this function will allow the user to filter movies based on genre or movie rating

Arguments:

-this function has no parameters as the logic inside the function will determine whether the movie will be filtered by genre or movie rating, which will call the getMovieRating() function within the same class.

Return Type:

-void: the function will return nothing.

3.2.8.2.2 getMovieRating()

this function will interact with a third-party website such as IMDB or Rotten Tomatoes to get information on a specific movie's rating

Arguments:

-title (Movie Object): This function will use the title from an instance of a Movie object to search for the movie rating from the third-party website

Return Type:

-the function will return the rating of the movie which will be displayed with a data type: String

3.2.9 Seat

the Seat class is responsible for creating an instance for each individual seat for a specific Showtime instance. It will interact with the Reservation class for when either a customer or employee wants create a reservation for a specific movie

3.2.9.1 Attributes of Seat class

3.2.9.1.1 row

the row for the specific seat will be initialized with the data type: int

3.2.9.1.2 seatNumber

the actual seat number will be initialized with the data type: int

3.2.9.1.3 isEmpty

this attribute will flag whether or not the seat is empty with data type: boolean

3.2.9.2 Functions available in Seat class

3.2.9.2.1 isOccupied()

this function will interact with the seating chart interface and will display the seat in black if it is already purchased or red if it is still available to purchase

Arguments:

-this function has no parameters since it will interact directly with the attributes within the class

Return Type:

-boolean: the function will return true if the seat is occupied or false if the seat is empty

3.2.9.2.2 occupy()

this function will interact with the seating chart interface to mark that is empty as occupied after a reservation is confirmed with payment confirmation

Arguments:

-isOccupied (bool): This function will call the isOccupied() function within the class to see if the seat is empty and ready for purchase.

-reservationConfirmed(bool): this function will also call the reservationConfirmed() function to make sure that payment was processed before marking the seat as occupied

Return Type:

-boolean: the function will return true if occupied successfully and false, if otherwise

3.2.9.2.3 vacate()

this function will interact with the seating chart interface and will turn an occupied seat back to red (previously black) after a successful reservation cancellation

Arguments:

-this function has no arguments as it will be called within the cancelReservation() method within the Reservation class

Return Type:

-void: the function will return nothing.

3.2.10 Reservation

the Reservation class will hold the attributes and functions of each single unique reservation instance created by either the Customer or the Employee. It will include a unique numerical ID, the name of the customer, and the number of seats they purchased

3.2.10.1 Attributes of Reservation class

3.2.10.1.1 reservationID

the reservation ID will be unique to the customer using a random number generator of type: int

3.2.10.1.2 customerName

the name of the Purchaser will be initialized by user input with data type: String

3.2.10.1.3 is_Confirmed

this will be a flag for when the reservation is confirmed: type boolean

3.2.10.1.4 number_seats

this will represent the number of seats the user intends to purchase with data type: int

3.2.10.2 Functions available in Reservation class

3.2.10.2.1 calculatePrice()

this function will calculate the total price of the reservation based on the number of seats chosen for each Ticket Type. It will also include the sales tax and include a reservation fee calculated in the total

Arguments:

-this function will have no parameters as it will interact directly with the variables within the class itself.

Return Type:

-double: the function will a floating point double containing the total price of the reservation

3.2.10.2.2 addSeat()

this function will see what Ticket Type the user wishes to purchase and will increment number_seats variable by one after each time the program takes user input of a mouse click on a plus sign on the website interface

Arguments:

-this function will take a user input of a single mouse click on the plus sign located on the website interface

Return Type:

-void: the function will return nothing.

3.2.10.2.3 removeSeat()

this function will see what Ticket Type the user wishes to purchase and will decrement number_seats variable by one after each time the program takes user input of a mouse click on a minus sign on the website interface

Arguments:

-this function will take a user input of a single mouse click on the minus sign located on the website interface

Return Type:

-void: the function will return nothing.

3.2.10.2.4 timeLimit()

this function will start a 10 minute timer once the user chooses to create a reservation. It will kick them out of the reservation window once time runs out

Arguments:

-This function contains no parameters

Return Type:

-void: the function will return nothing.

3.2.10.2.5 cancelReservation()

this function will allow the user to cancel a reservation by providing their unique reservation ID. The function will search the reservation database and will return a boolean notifying the user whether or not their reservation was found. The function will interact with the refund method to initiate the refund process

Arguments:

-reservationID (int): This function will use the reservation ID to use in a search algorithm in the Theater Ticketing System database.

Return Type:

-boolean: the function will return true if reservation is found and refund is processed

3.2.10.2.6 createReservation()

this function will initiate the process of creating a new reservation prompted by either the Customer or the Employee. It will interact with the Seat Class for them to be able to choose a Movie and Showtime. Once chosen, the function will interact with the functions within the Reservation class to add or remove seats and to calculate the total price. Once total price is calculated, the Payment class will be called where the processPayment() function will be invoked to process the payment.

Arguments:

-this function takes no arguments

Return Type:

void: the function will return nothing. It will change the is_Confirmed variable to true which will trigger the system to prompt the user if they want an email or print receipt

3.2.11 Ticket

The instance of a Ticket class will be generated once a reservation has been made in the Reservation class. The amount of instances created will depend on the number of seats purchased.

3.2.11.1 Attributes of Ticket class

3.2.11.1.1 ticketID

Each ticket will have a unique ID which has the data type: int

3.2.11.1.2 price

the price of each ticket displayed will be determined by the Ticket Type. This variable will have the data type: double

3.2.11.2 Functions available in Ticket class

3.2.11.2.1 getUser()

this function will retrieve the customer name from the Customer class to be displayed on the ticket

Arguments:

-this function has no arguments

Return Type:

-void: the function will return nothing.

3.2.11.2.2 getShowTime()

this function will retrieve the Showtime information from the Showtime class to be displayed on the ticket

Arguments:
-this function has no arguments

Return Type:
-void: the function will return nothing.

3.2.11.2.3 getSeatInfo()

this function will retrieve the seat information from the Seat class to be displayed on the ticket

Arguments:
-this function has no arguments

Return Type:
-void: the function will return nothing.

3.2.11.2.4 checkTicketType()

this function will verify the ticket type: adult, child, student, or senior. This will invoke the verifyDiscount() method in the TicketType subclass. If the ticket type is verified, it will apply the discount

Arguments:
This function has no arguments

Return Type:
-boolean: the function will return true if the ticket type is a discounted type and verified correctly

3.2.12 TicketType

the TicketType class is a subclass of the Ticket Class that will create an instance based on the type of ticket the user chooses. If a discounted ticket is chosen, the verifyDiscount() function will be called

3.2.12.1 Attributes of TicketType class

3.2.12.1.1 type

it could either be Adult, Child, Student, Military, or Senior. Will be data type: String

3.2.12.1.2 price

the price will be based on the ticket type. Will invoke setPrice() method that returns type: double

3.2.12.2 Functions available in TicketType class

3.2.12.2.1 verifyDiscount()

this function will interact with a third-party verification website such as ID.me to verify student, military, or senior status. If the verification fails, the user will be notified.

Arguments:

This function has no arguments

Return Type:

-boolean: the function will return true if the verification was returned as successful.

3.2.12.2.2 setPrice()

this function will set the price based on the type of ticket the user chooses. If it's either adult or child ticket type, no verification is needed and the price will be set accordingly. If the user requests a discounted ticket type, the verifyDiscount() method will be invoked and the discounted price will be set based on the discounted ticket type. if the method returns false, the adult ticket price will be set by default.

Arguments:

This function has no arguments

Return Type:

-void: this function returns nothing.

3.2.13 Payment

The Payment Class will create an instance of the class whenever a reservation is created or a refund is requested in terms of canceling the reservation. The functions within this class will interact with a external payment gateway connected to the banking systems that will process payments and refunds. Security is a top priority to protect the information of the user

3.2.13.1 Attributes of Payment class

3.2.13.1.1 paymentID

each transaction will have a unique ID that will be logged and tracked using the data type: int. It will utilize a random number generator to create the unique ID

3.2.13.1.2 paymentMethod

payment method type will be initialized based on what type of card they use for the transaction. The name of the card will be of data type: String

3.2.13.1.3 transactionTime

date and time of transaction will be recorded with type: DateTime

3.2.13.1.4 cardNumber

cardNumber will be sent to payment gateway with type: String

3.2.13.1.5 cardholder_name

will be sent to payment gateway with type: String

3.2.13.1.6 security_code

will be sent to payment gateway with type: int

3.2.13.2 Functions available in Payment class

3.2.13.2.1 wipeCardInfo()

this function will set all of the attributes to null after the transaction is successfully completed, the attributes in the class will be set to null. The information will be recorded in the database beforehand. The Payment object can then be deleted from memory to protect user information.

Arguments:

-the function contains no arguments

Return Type:

-void: the function will return nothing.

3.2.13.2.2 processPayment()

this function is responsible for handling the payment transaction when a customer makes a purchase in a theater ticketing system. It will interact with the payment gateway to communicate with the banks to process the transaction

Arguments:

-customer (Customer): the customer making the payment

-totalPrice (double): the total amount to be charged for the ticket booking including taxes

-paymentMethod (String): payment method used for the transaction

-paymentDetails (String, String, int): the cardholder name, card number, and security code

Return Type:

-boolean: returns true if the payment was successful, false otherwise

3.2.13.2.3 refund()

this function is responsible for processing refunds for a payment transaction. It will interact with the payment gateway that will communicate with the banks to process the refund.

Arguments:

-Transaction ID (String): the unique identifier of the payment transaction

-refundAmount (double): the amount to be refunded to the customer

Return Type:

-boolean: the function will return true if refund is successful, false otherwise

3.2.14 Receipt

3.2.14.1 Attributes of Receipt class

the Receipt class will inherit all of the attributes of the Payment and will also have access to the variables in the Ticket class containing reservation ID

3.2.14.2 Functions available in Receipt class

3.2.14.2.1 generateReceipt()

this function will generate either a printed or emailed receipt based off of the user input. If asking for an email receipt, it will prompt the user to enter their email address to be sent the receipt with all of the information. Otherwise, it will trigger the print function on their desktop to print a physical receipt

Arguments:

-paymentSuccessful(boolean): if true, will send a receipt, if false, will return a message saying payment was not processed correctly

-deliveryType (String): this function will take in an argument of type String that the user will input as being either an email or printed receipt.

Return Type:

-void: the function will return nothing.

3.3 Software Architecture

3.3.1 Software Architecture Diagram

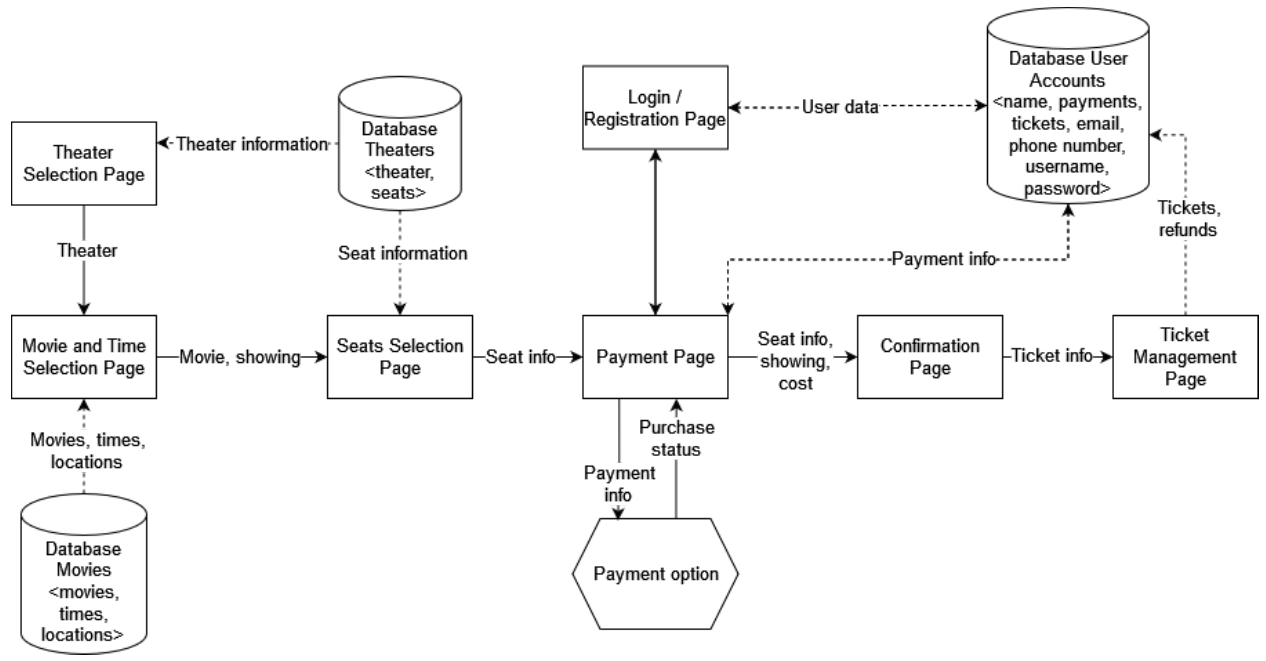

Figure 2: Software Architecture Diagram of the Theater Ticketing System including components and connectors

3.3.2 Software Architecture Description

The next section will provide a complete explanation of the entire SWA diagram, including an overview of components and their connectors.

3.3.2.1 Theater Selection Page

Description: On this page the user is selecting one of the theaters of the client from the San Diego Area.

Functionality: Selecting the theater enables the user to go further in the process of ticket buying. Once a theater is selected it will be the default theater shown in the drop-down selection.

Connectors: Theater Selection Page gets its theater information from the theater database which stores information about the theaters that can be selected. Once the theater is selected the user is enabled to select the movies and showtimes that are available in the theater.

3.3.2.2 Movie and Time Selection Page

Description: On this page the movie and showtime for the selected movie will be selected.

Functionality: Every movie currently available in the selected theater has different showtimes available. Showtimes that are booked out can’t be selected and should be using a disabled secondary button.

Connectors: The movie database provides the Movie and Time Selection Page with data about available movies, the showtimes and the location where the movie is shown inside the theater that was selected on the Theater Selection Page.

3.3.2.3 Seats Selection Page

Description: Seats for the selected movie and showtime can be selected on this page

Functionality: This page allows the user to select how many seats he wants to purchase. The default value is 1 and can be increased and decreased with a plus and minus sign UI element or through clicking and writing an integer between 1 and 20 into the input field where the default value of 1 is shown. The maximum number of seats available per purchase are limited to 20.

Connectors: With the input from the Movie and Time Selection Page with the movie and showtime and the seat information from the Theater Database the seat selection is enabled. The seat information from this page will be directed to the next page for the payment.

3.3.2.4 Payment Page

Description: On the Payment Page the user can see his previous selections in the process and pay for the selected tickets.

Functionality: The Payment Page enables the user to see the selected theater, movie, location, showtime and seat information. Different payment options are available to choose from. Once a payment is confirmed the payment provider checks the validity of the payment and returns the payment status. If there is any complication the payment will be declined with an error message, if everything is correct the payment will be confirmed. Besides the option of proceeding as a guest the user can register or log in with an account.

Connectors: Through the seat selection all information about the purchase is displayed. The Registration / Login Page is connected to the Payment page as well. The payment option gets the payment request with the payment information and returns the payment status as confirmed or declined. The purchase information with the costs and tickets will be forwarded to the Confirmation Page. If an user account exists the payment info will be stored in the User Account Database.

3.3.2.5 Login / Registration Page

Description: The Login / Registration Page enables the user to sign up or log in with an user account.

Functionality: When proceeding with the payment process there is the option to sign up or log in with a user account. The user data is stored in the User Account Database. Name, username and password are required fields but phone number is an optional input. The username will be an email address and the password has to be at least eight characters including minimum one number, uppercase letter, lowercase letter and a special character.

Connectors: There is a connection between the Login / Registration page and the Payment Page. When logging in with the user account the previously used payment option will be set as a default. All the user data from the registration will be stored in the User Account Database.

3.3.2.6 Confirmation Page

Description: After a successful purchase of tickets the Confirmation Page will be displayed summing up the purchase information.

Functionality: The Confirmation Page will display the theater, movie, showtime, seats and the location inside the selected theater where the movie is shown. Also the seat information and the costs will be shown on this page.

Connectors: The information of the purchase and tickets comes from the connection to the Payment Page. The ticket information will be passed on to the Ticket Management Page.

3.3.2.7 Ticket Management Page

Description: The Ticket Management Page will display the purchased tickets with the information about the purchased tickets. Also the option of refunding will be found on this page.

Functionality: This page enables the user to view his tickets with the ticket information (seats, movie, showtime and location) and can be used to show the purchased tickets at the theater for entry. Also there is a button that enables the user to refund the tickets.

Connectors: The purchase information is retrieved from the Confirmation Page. The connection to the User Account Database facilitates the process of refunding as the ticket information from the user is stored in the database as well.

3.3.2.8 Payment Option

Description: After the request for the payment the payment provider checks the payments validity and returns a purchase status.

Functionality: The request for the payment from the Payment Page will be checked from the payment provider and will return a confirmation or decline the payment. The status of confirmation or declinement will be returned to the Payment Page with a corresponding message on the UI of the Payment Page.

Connectors: The Payment Option is only connected to the Payment Page getting the payment request and returning the payment status from the payment provider.

3.3.2.9 Database Theaters

Description: The Database for theaters stores information about theaters of the client together with seat information.

Functionality: The theater database provides data about the theaters for the Theater Selection Page and seat information to the Seats Selection Page.

Connectors: see 3.3.2.9 Functionality.

3.3.2.10 Database Movies

Description: The movie database stored data about the movies, showtimes and locations (halls) where the movie is shown.

Functionality: The movie database provides the Movie and Time Selection Page with data about availability of movies in the selected theater together with showtimes for each movie and locations inside the theater where the movie is shown.

Connectors: see 3.3.2.10 Functionality.

3.3.2.11 Database User Accounts

Description: The user accounts database stores information about user accounts and related activities such as payments.

Functionality: From the registration page the user data like name, username, password, email address and phone number is retrieved. Payment information comes from the connection to the Payment Page. Previously stored payment information can also be retrieved from the database to the payment page when the user logs into the user account. Refunds that are requested in the Ticket Management Page are checked with the information in the user account database.

Connectors: see 3.3.2.11 Functionality.

4. Development Plan and Timeline and Partitioning of Tasks

4.1 Timeline

4.1.1 Phase 1: Project Initiation (Project Manager, Business Analyst)

Timeline: Weeks 1-2

1. **Project Start (Week 1)**

- Define scope and objectives
- Hire project manager and create development team
- Obtain deliverables
- Resource Allocation

2. **Requirements Gathering (1-2)**

- Document requirements for theater ticketing system
- Create features and functionalities

4.1.2 Phase 2: Design Planning (System Architect, Database Designer, UI/UX Designer)

Timeline: Weeks 3-6

1. **Software Architecture (3-4)**

- Create software architecture design
- Create software architecture diagram

2. **Database (4-6)**

- Define UML Class Design
- Create UML Class Diagram
- Plan for securing data

3. **User Interface (4-6)**

- Plan for visual interface and layout
- Gather feedback on UI

4.1.3 Phase 3: Development (Frontend/Backend Developers, Quality Assurance/Test Engineers, DevOps Engineers)

Timeline: Weeks 7-23

1.**Frontend Development (Weeks 7-13)**

- Develop the user interface based on UI designs.
- Implement user authentication and authorization.

2. **Backend Development (Weeks 13-19)**

- Build the server-side application.

- Implement payment processing.

3. **Database Implementation (Weeks 19-21)**

- Create the database using the planned design
- Implement data storage and retrieval functionalities.

4. **Testing and Quality Assurance (Weeks 21-23)**

- Conduct testing
- Fix bugs experienced during testing
- Make sure the system is secure

4.1.4 Phase 4: Deployment (Training Specialist, Support & Maintenance Team, Marketing & Promotion team)

Timeline: Weeks 24-27

1. **Deployment and System Integration (Weeks 24-26)**

- Deploy the ticketing system to production servers
- Integrate with third-party services (IMDB, Rotten Tomatoes)
- Initiate performance testing

2. **Training and Documentation (Week 26)**

- Train the theater staff on how to use the system
- Create user manuals and documentation

3. **Soft Launch and Feedback Gathering (Week 27)**

- Perform a soft launch with audience being stockholders/family
- Gather feedback

4.1.5 Phase 5: Launch (Support/Maintenance Team, Marketing & Promotion Team, Product Manager)

Timeline: Weeks 27-30

1. **Full-Scale Launch (Week 27)**

- Launch the Theater Ticketing System to the general public
- Promote the new ticketing system through advertising

2. **Monitoring and Maintenance (Weeks 28-30)**

- Regularly monitor the system performance and keep obtaining feedback
- Address any issues and release regular system updates (security, bug fixes, etc.)
- Plan for weekly to bi-weekly maintenance and support for the system

5. Verification Test Plan

5.1 Introduction

5.1.1 Purpose

The purpose of this test plan is to outline the overall strategies, objectives and approaches to verifying the functionality, performance, security, and usability, and compatibility of the theater ticketing system.

5.1.2 Scope

This test plan incorporates comprehensive testing of all aspects of the theater ticketing system, traversing all aspects of software verification.

5.1.3 Objectives

The main objectives of the verification test plan are to ensure the theater ticketing system:

- Meets the standards of both functional and non-functional requirements
- Performs under a varying number of loads to ensure reliability
- Maintains the highest level of security and confidentiality of data
- Provides the user an exceptional experience accessing the theater ticketing system
- Is able to function across all varying devices and platforms

5.2 Test Objectives

5.2.1 Unit Testing

- Focus on verifying individual code components that include: methods and classes
- Test cases will ensure that each unit of code performs to standard

5.2.2 Functional Testing

- Verify that all functional requirements: user registration, show selection, booking a reservation, and payment processing are fully satisfied

- Validate user and system interactions along with the handling of error exceptions

5.2.3 Performance Testing

- Evaluate the performance and response times of the theater ticketing system under normal and stress conditions

5.2.4 Security Testing

- Identify and mitigate security vulnerabilities, such as authentication issues, data encryption, and protection against common security threats.
- Ensure that sensitive user data, including payment information, is handled securely.

5.2.5 Compatibility Testing

- Ensure that the system functions correctly on different web browsers (e.g., Chrome, Firefox, Safari, Edge)
- Verify that the system adapts to different screen sizes

5.3 Testing Approach

5.3.1 Unit Testing

5.3.1.1 Seat Reservation Functionality

Objective: We want to verify that the "reserveSeat" method accurately reserves the seats while updating the availability of the seat in real-time

Features Covered:
- Seat Reservation Functionality
- Functionality of seat availability

Test Description:

1. **Test Vector 1:** Reserve a single seat and confirm it is marked as occupied
    - Input: Valid Seat Identifier

- Expected Output: The seat is successfully reserved

5.3.1.2 Calculating the Total Price Functionality

Objective: Ensure that the calculatePrice() function correctly calculates the ticket price based on ticket type and movie.

Features Covered:
- Ticket Price Calculation
- Handling of the different ticket types

Test Description:

1. Test Vector 1: Calculate the ticket price for a single adult seat
- Input: Movie Type, Ticket Type: standard adult ticket
- Expected Output: The function returns the correct ticket price for a single adult ticket

2. Test Vector 2: Calculate the ticket price for a single discounted seat for a student and veteran
- Input: Movie Type, Ticket Type: student discounted ticket / veteran discounted ticket
- Expected Output: The function returns the correct ticket price for a discounted student / veteran ticket

5.3.1.3 Find Reservation Functionality

Objective: Ensure that the getReservation() function in the Customer Class correctly searches the database for a user's successfully booked reservation

Features Covered:
- Efficient Search Algorithm for the Database

Test Description:

1. Test Vector 1: Search for a valid reservation that was previously booked
- Input: Ticket ID: valid and unique identifier, Customer last name

- Expected Output: The function correctly returns the information for the valid booking including: date purchased, total price, movie and showtime information, name of purchaser

2. Test Vector 2: Search for an invalid reservation

- Input: Ticket ID: invalid identifier, Customer last name
- Expected Output: The function returns an error message to the user saying the reservation was not found. It will prompt the user to input a different reservation ID

5.3.2 Functional Testing

5.3.2.1 Booking a Reservation

Objective: We will test the end-to-end booking process from: movie selection, showtime selection, seat reservation, and payment processing.

Features Covered:

- User and system interactions including workflows
- End-to-end booking process

Test Description:

1. Test Vector 1: Select a movie, choose showtime, reserve seats, and complete payment

- Input: Valid movie, showtime, and seat selection and successful payment
- Expected Output: The reservation is successfully booked, marked by a confirmation email being sent by the system

2. Test Vector 2: Attempt to reserve seats that are already occupied

- Input: User attempts to book a seat that is already reserved
- Expected Output: The system prevents a double-booking scenario and outputs an error message to the user.

2. **Test Vector 3:** User cancels a booking and receives a refund

- Input: User requests to cancel a reservation

- Expected Output: The system successfully cancels the reservation, seats are released and marked as available, and a refund is issued

5.3.2.2 Filtering Showtimes

Objective: Ensure that the system correctly filters and displays movies along with their showtimes based on the preferences set by the user

Features Covered:
- Efficient Search Algorithm for the movie database
- Filtering of movies

Test Description:

1. **Test Vector 1:** Filter movies by movie rating (G, PG, PG-13 etc.)
    - Input: Movie rating selection by the user
    - Expected Output: Only movies with the specified rating are displayed along with the showtimes for that specific date.

2. **Test Vector 2:** Filter movies by both movie rating AND genre
    - Input: Movie rating selection and genre selection by the user
    - Expected Output: Only movies with the specified rating and genre are displayed

5.3.3 Performance Testing

5.3.3.1 Multiple Users Try Booking at the Same Time

Objective: We will simulate multiple users attempting to book the same seat simultaneously

Features Covered:
- Simultaneous Booking
- Reservation Handling

Test Description:

1. Test Vector 1: Have multiple users attempt to book the same seat for the same movie and showtime

- Input: Multiple users online trying to book the same seat
- Expected Output: The system is able to handle concurrent bookings and won't allow for double booking. It should have users choose a different seat if the seat is already in the process of being purchased

2. Test Vector 2: Stress Test the system with multiple concurrent booking attempts

- Input: Very high load of concurrent booking requests
- Expected Output: The system is still responsive and avoids crashing or system degradation

5.3.3.2 System Integration with Payment Gateway

Objective: We will validate the system's integration with the external payment gateway system by conducting real sample payment transactions

Features Covered:

- External payment gateway integration
- Payment processing and confirmation

Test Description:

1. Test Vector 1: Perform real sample payment transactions with real payment methods customers will actually use once system is launched

- Input: Book a movie ticket with credit card transaction
- Expected Output:The payment is processed successfully, and the sample user receives confirmation receipt of payment

2. Test Vector 2: Test for payment failures and error handling

- Input: Book a movie ticket with debit card with zero balance
- Expected Output: The system handles the payment failure successfully and provides the user with the appropriate error message

5.4 Test Cases

see attached file “Theater Ticketing System Test Cases”

Homework 4 start

6. Data Management Strategy for Theater Ticketing System

6.1 Data Collection

6.1.1 Show Information

- Display data that includes movie titles, descriptions, dates, showtimes, and locations from theater management

6.1.2 Seat Information

- Seating arrangements and seat availability data are provided by the theater management and updated regularly to reflect any changes in the seating arrangements

6.1.1 User Data

- User profiles, which include registration details, email addresses, and encrypted passwords, are collected when users sign up for accounts

6.2 Data Storage

6.2.1 Database Management

- Data is stored in a secure and scalable relational database management system (RDBMS), SQL. The database will undergo regular back-ups to prevent any data loss

6.2.2 Data Partitioning

- This database will utilize data partitioning in order to manage large datasets efficiently and improve database performance.

6.3 Data Access and Security

6.3.1 User Authentication

- This database will implement secure user authentication mechanisms, multi-factor authentication and encrypted password storage, to protect user accounts from any unauthorized data breaches.

6.3.2 Role-Based Access Control

- This database will implement a role-based access control to restrict access to sensitive data and system functionality based on user roles (admin, employee, customer etc.)

6.3.3 Data Encryption

- This system will encrypt sensitive data, such as user passwords and payment information using strong encryption algorithms during transmission and storage.

6.3.4 Data Privacy Compliance

- Ensure compliance with data privacy regulations by providing users with control over their data. The database will obtain explicit consent from the user for data processing procedures.

6.3.5 Security Monitoring

- This system will implement around the clock security monitoring to detect and respond to any unauthorized access or security threats.

6.4 Data Integrity

6.4.1 Validation Rules

- This database will implement data validation rules to ensure that data entered in by the user adheres to a specific format and integrity specifications.

6.4.2 Consistency Checks

- This database will perform regular consistency checks to verify data such as seat availability and show information, and make sure that it matches the actual state of the theater and its events.

6.5 Data Backup and Recovery

6.5.1 Perform Regular Backups

- This database will schedule automated backups of the database to capture critical data at regular intervals. The backups will be stored in a separate location for ease of accessibility and security.

6.6 Data Retention

6.3.1 Retention Policies

- This database will create data retention policies that determine how long various types of data: reservation records, user profiles, and show information, are retained in the database.

6.7 Data Reporting and Analytics

6.7.1 Reporting Tools

- This database will implement reporting and analytics tools that will gain insight from data from the theater ticketing system. This will allow for improved performance analysis, revenue tracking, and customer behavior analysis for showtime analytics.

6.8 Legal Requirements

6.8.1 Data Compliance Officer

- Designate a data compliance officer responsible for making sure the system complies with relevant data protection laws and regulations.

6.9 Disaster Recovery Plan

6.9.1 Data Recovery

- Establish concrete procedures for data recovery in the chance of data breaches. It will include the identification and notification of affected parties.

7. Design Specification

7.1 Updated Software Architecture Diagram

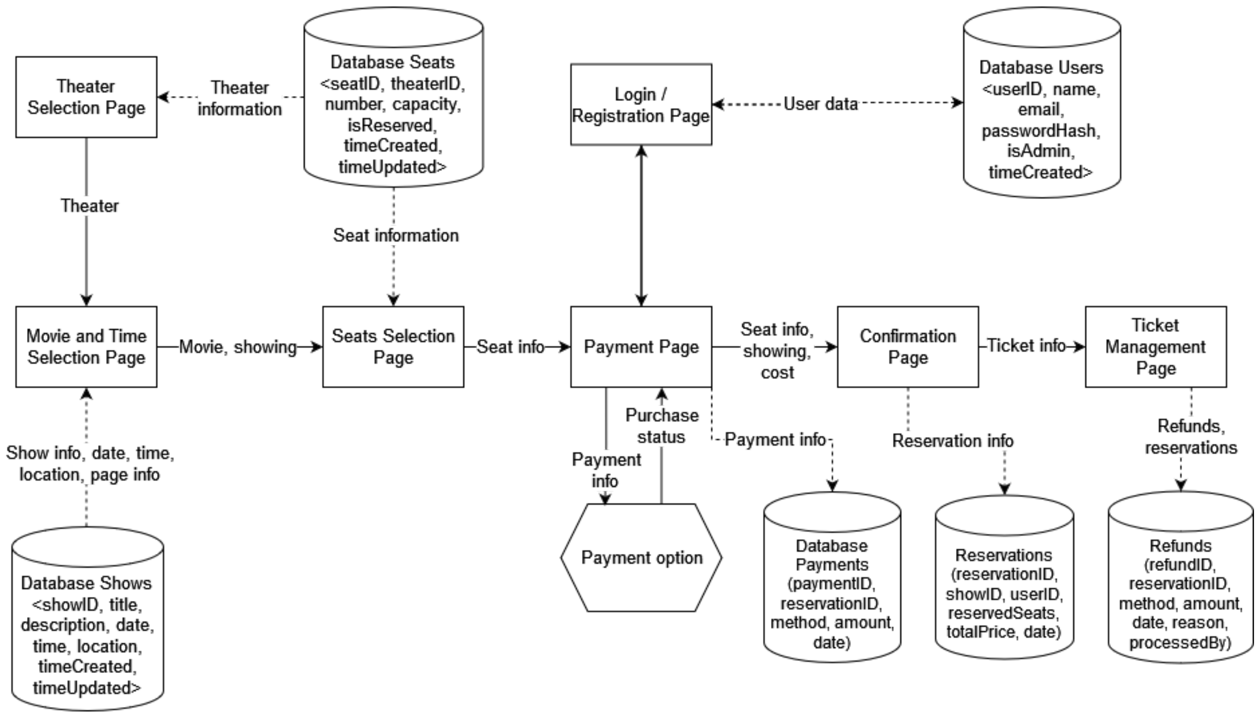
Theater Selection Page
Theater information
Database Seats <seatID, theaterID, number, capacity, isReserved, timeCreated, timeUpdated>
Login / Registration Page
User data
Database Users <userID, name, email, passwordHash, isAdmin, timeCreated>
Theater
Seat information
Movie and Time Selection Page
Movie, showing
Seats Selection Page
Seat info
Payment Page
Seat info, showing, cost
Confirmation Page
Ticket info
Ticket Management Page
Show info, date, time, location, page info
Purchase status
Payment info
Payment info
Reservation info
Refunds, reservations
Payment option
Database Payments (paymentID, reservationID, method, amount, date)
Reservations (reservationID, showID, userID, reservedSeats, totalPrice, date)
Refunds (refundID, reservationID, method, amount, date, reason, processedBy)
Database Shows <showID, title, description, date, time, location, timeCreated, timeUpdated>

Figure 3: Updated Software Architecture Diagram

7.2 Explanation of Modification

While advancing with the Data Management Strategy for the Theater Ticketing System the Design Specification had to be adjusted. This resulted in an updated Software Architecture Diagram. Initially the SWA had three databases: user accounts, movies and theaters. The updated version of the SWA includes six databases: shows, seats, users, payments, reservations and refunds. Newly added data requirements and easier handling with more modularity of the databases were crucial factors for the revision.

8. Theater Ticketing System Database Schema

Table: Shows

- **ShowID (Primary Key):** a unique identifier for each show
- **Title:** The title or name of the movie
- **Description:** A textual description of the show, which includes a brief summary, cast, genre, and rating.
- **RottenTomatoesReview:** The external third-party reviews of the movie
- **Date:** The date the movie was scheduled
- **Time:** The movie showtime
- **Location:** The specific theater and auditorium the movie is playing at
- **timeCreated:** Timestamp showing when the movie was created by the administrator
- **timeUpdated:** Timestamp showing the movie record was last updated

Table: Seats

- **SeatID:** A unique identifier for each seat
- **ShowID:** A reference to the movie for which the seat is allocated
- **SeatNumber:** An identifier for the seat, including row and seat number
- **isReserved:** Flag indicating whether the seat is reserved or available
- **timeCreated:** Timestamp showing when the seat record was created
- **timeUpdated:** Timestamp showing when the seat record was updated
- **Capacity:** The total seating capacity of the auditorium
- **Theater:** The specific auditorium the movie is playing in

Table: Users

- **UserID (Primary Key):** A unique identifier for each user
- **UserName:** The username chosen by the user for their account

- **Email:** The email address associated with the user account
- **PasswordHash:** A secure has for the user's password
- **isAdmin:** A flag indicating whether the user has administrative privileges
- **timeCreated:** Timestamp showing when the user account was created

Table: Reservations

- **ReservationID (Primary Key):** A unique identifier for each reservation
- **ShowID (Foreign Key):** A reference to the movie for which the reservation was made
- **UserID (Foreign Key):** A reference to the user who made the reservation
- **ReservedSeats:** The number of seats reserved in the reservation
- **TotalPrice:** The total price of the reserved seats
- **ReservationDate:** Timestamp showing when the reservation was made

Table: Payments

- **Payment ID (Primary Key):** A unique identifier for each payment transaction
- **Reservation ID (Foreign Key):** A reference to the reservation associated with the payment
- **PaymentMethod:** The method used for payment (credit, debit, paypal)
- **PaymentAmount:** The total payment amount for the reservation
- **PaymentDate:** Timestamp showing when the payment was processed

Table: Refunds

- **Refund ID (Primary Key):** A unique identifier for each refund transaction
- **Reservation ID (Foreign Key):** A reference to the reservation for which the refund is issued, establishing a link between the refund and the original reservation
- **Refund Method:** The method used to issue the refund (credit, debit, paypal)
- **RefundAmount:** The amount refunded to the customer. Decimal value to handle currency accurately
- **Refund Date:** Timestamp indicating when the refund was processed
- **Reason:** A textual description of the reason for the refund, which can aid in tracking and analyzing refund requests
- **RefundProcessedBy (Foreign Key):** A reference to the user who processed the refund. This establishes a link between the refund and the user who initiated the refund transaction

Foreign Key: a column or set of columns in a table that establishes a link to the primary key in a separate table. It enforces referential integrity by ensuring that relationships between tables are maintained
Primary Key: a column or set of columns that uniquely identify each row in the database. No two rows in the same table can have the same primary key value.

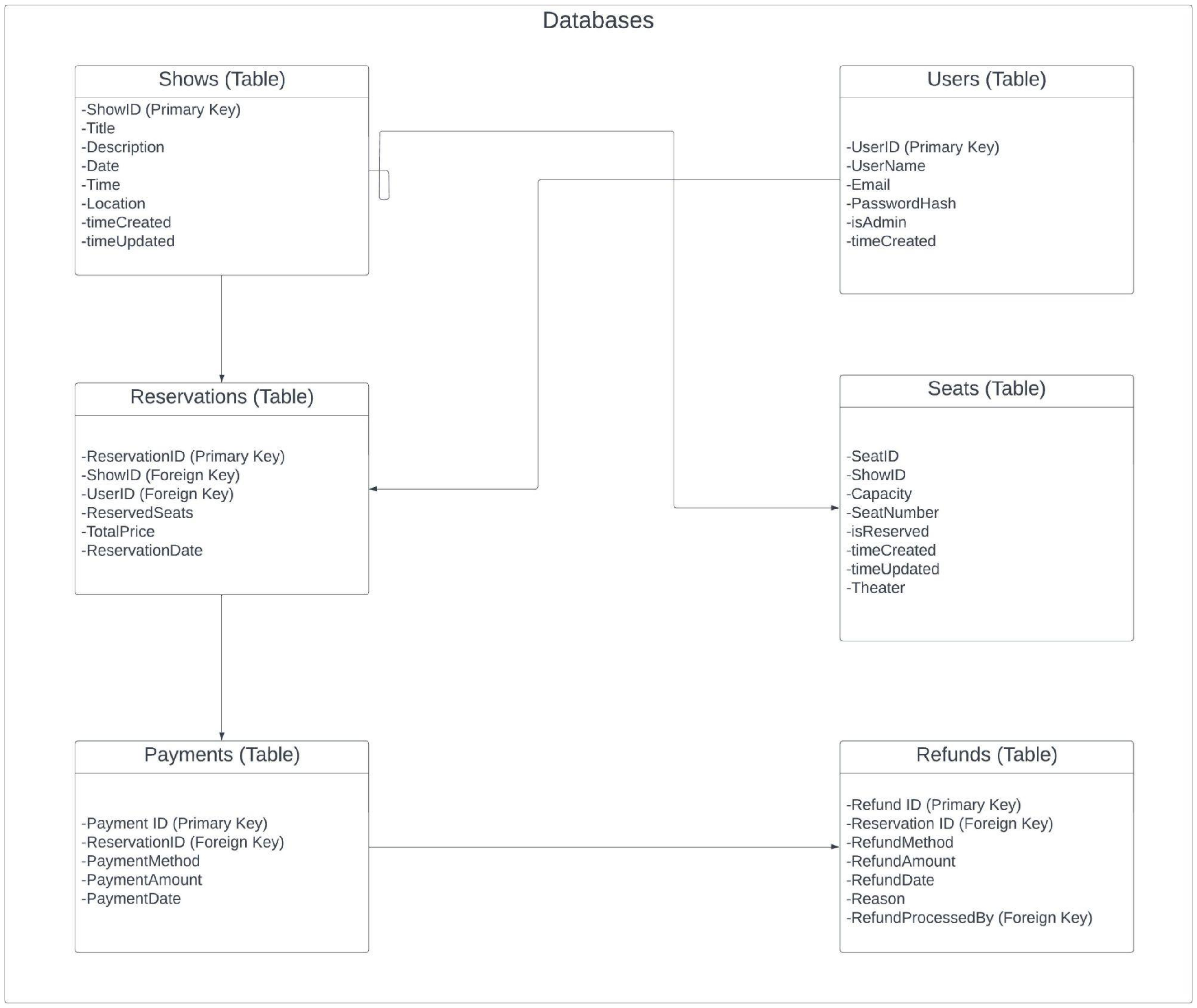

Figure 4: Theater Ticketing System Database Schema

9. Trade Off Discussion

- A major difference between PostgreSQL and SQL Server is that the first one is open source and the latter is owned by microsoft. Using SQL is more ideal for very large businesses, especially if they require the use of microsoft products. PostgreSQL is more suitable for smaller businesses that could use maximum functionality.

- Using a NoSQL database instead of PostgreSQL could also yield some benefits. It would allow a more flexible database model where you can store different types of data like JSON documents for fast retrieval, replication, analysis, etc.

- A big benefit of using PostgreSQL is that it is free to use, but since it is an open source database, it could also be slightly more vulnerable security-wise. You also have to make sure you do not have a version that will not lose support, The PostgreSQL Global Development Group will only support any major version for 5 years after release, after which that version will no longer be supported.

- Comparing PostgreSQL to another major SQL by Oracle we can also see some tradeoffs. Oracle will have better security, replication, and availability, while PostgreSQL has better API compatibility, more affordable support, and better scalability.