Software Design Specification

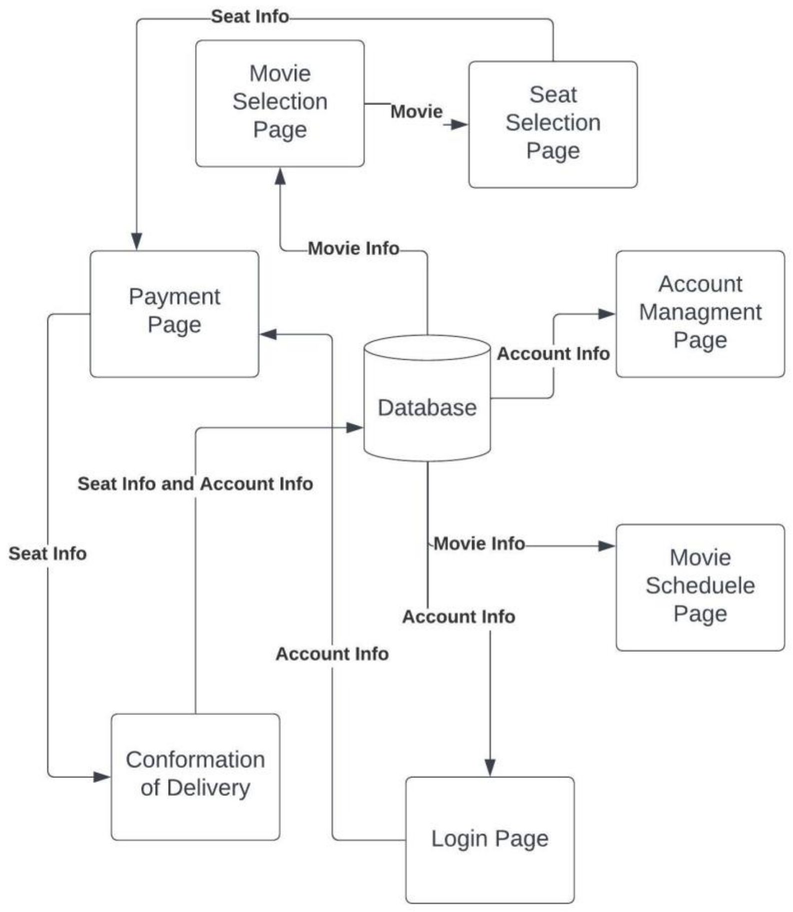
By: Luis Suarez

System Description

The system to be constructed will have the main function of being a ticketing service for a local San Diego cinema chain. It will also host a secondary function of being a customer feedback system, a user profile service, and an AMC subscription service.

Software Architecture Overview

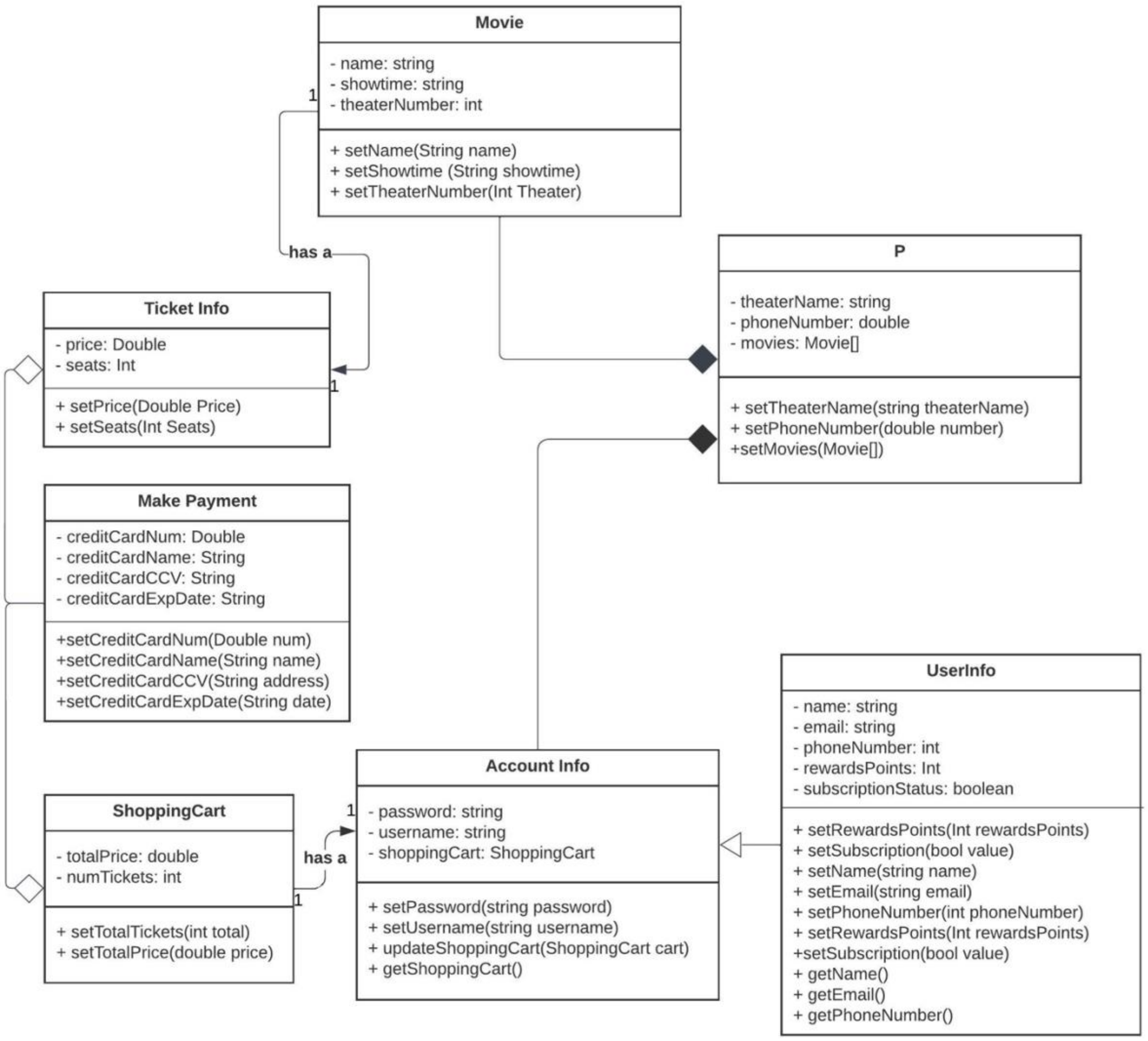
The architecture diagram of all major components is the following:



<showtimes, locations, seats, prices> = movie info

<username, password, authorization loyalty points, payment information, purchase history> = account info.

The UML Class Diagram for the system will be the following:


Movie
- name: string
- showtime: string
- theaterNumber: int
+ setName(String name)
+ setShowtime (String showtime)
+ setTheaterNumber(Int Theater)
1
has a
1
P
- theaterName: string
- phoneNumber: double
- movies: Movie[]
+ setTheaterName(string theaterName)
+ setPhoneNumber(double number)
+setMovies(Movie[])
Ticket Info
- price: Double
- seats: Int
+ setPrice(Double Price)
+ setSeats(Int Seats)
Make Payment
- creditCardNum: Double
- creditCardName: String
- creditCardCCV: String
- creditCardExpDate: String
+setCreditCardNum(Double num)
+setCreditCardName(String name)
+setCreditCardCCV(String address)
+setCreditCardExpDate(String date)
UserInfo
- name: string
- email: string
- phoneNumber: int
- rewardsPoints: Int
- subscriptionStatus: boolean
+ setRewardsPoints(Int rewardsPoints)
+ setSubscription(bool value)
+ setName(string name)
+ setEmail(string email)
+ setPhoneNumber(int phoneNumber)
+ setRewardsPoints(Int rewardsPoints)
+setSubscription(bool value)
+ getName()
+ getEmail()
+ getPhoneNumber()
Account Info
- password: string
- username: string
- shoppingCart: ShoppingCart
+ setPassword(string password)
+ setUsername(string username)
+ updateShoppingCart(ShoppingCart cart)
+ getShoppingCart()
1
has a
1
ShoppingCart
- totalPrice: double
- numTickets: int
+ setTotalTickets(int total)
+ setTotalPrice(double price)

The following are the more detailed descriptions of the classes, attributes, and operations of the system:

Movies [class]: This class defines the attributes of the films that will be screening in this cinema chain. They will have all the attributes that any film should have: a name, showtime, and screen number.

setName(string name): This operation will set the name by passing in a String as an argument.

setTime(string time): This operation will set the showtime by passing in a String as an argument.

setLocation(string location): This operation will set the cinema location by passing in a String as an argument.

setTheaterNumber(int number): This operation will set the screen number by passing in an integer as an argument.

Ticket Info [class]: This class defines the attributes of the tickets for a particular film at a cinema. They will hold a particular price as well as how many tickets are left for sale.

setPrice(Double price): This operation will set the price of a ticket for a particular film by passing a double as an argument

setSeats(int seats): This operation will set the number of seats left for sale for a particular film by passing an int as an argument.

Make Payment [class]: This class defines the attributes of the films that will be screening in this cinema chain. They will have all the attributes that any payment should have: credit card number, name of the credit card holder, address of the credit card holder.

setCreditCardNum(double num): This operation will set the credit card number of the holder when making a payment by passing in a double as an argument

setCreditCardName(string name): This operation will set the name of the credit card holder when making a payment by passing in a string as an argument
setCreditCardCCV(int num): This operation will set the credit card CCV number (the 3 digits on the back of a credit card) when making a payment by passing in an integer as an argument

setCreditCardExpDate(String date): This operation will set the credit card expiration date when making a payment by passing in a string as an argument
----
Account Info [class]: This class defines the attributes any general account for the cinema chain will have. They will have the attributes that any generic account, regardless of the user, should have including: a password, username, name, email, phone number, and an online shopping cart for the website.

setPassword(string password): This operation will set the password to a particular account made by passing in a string as an argument

setUsername(string username); This operation will set the username to a particular account made by passing in a string as an argument

updateShoppingCart(ShoppingCart cart): This operation will update the shopping cart of a particular user by passing in an object of the class type ShoppingCart (explained later) as an argument

getShoppingCart(): This operation will retrieve the online shopping cart of a particular account
----

User Info [class]: This class defines the attributes that an account should have in reference to a particular user. They will have attributes such as: name, email, and a phone number for the website.

setRewardsPoints(int rewardsPoints): This operation will set the num of rewards points a particular user has accumulated thus far by passing in an integer as an argument

setSubscription(bool value): This operation will set whether a particular user has the AMC subscription for the cinema chain by passing in a boolean as an argument

setName(string name): This operation will set the name of a particular user by passing in a string as an argument

setEmail(string email): This operation will set the email of a particular user by passing in a string as an argument

setPhoneNumber(double phoneNumber): This operation will set the phone number of a particular user by passing in a double as an argument

getName(): This operation will retrieve the name of a particular user

getEmail(): This operation will retrieve the email of a particular user

getPhoneNumber(): This operation will retrieve the phone number of a particular user
----

Shopping Cart [class]: This class defines the attributes that the online shopping cart would have of a particular user for a cinema chain. They will include attributes such as: the total price of the tickets, and the number of tickets purchased.

setTotalTickets(int total): This operation sets the current total number of tickets that the shopping cart for a particular user holds by passing in an integer as an argument setTotalPrice(double price): This operation sets the current total price of the tickets that the shopping cart for a particular user holds by passing in a double as an argument

----

Movie Theater [class]: This class defines the attributes that a particular cinema in the cinema chain should have. They will include attributes such as: the name of the particular theater, the phone number to the particular theater, and the movies screening at the particular theater.

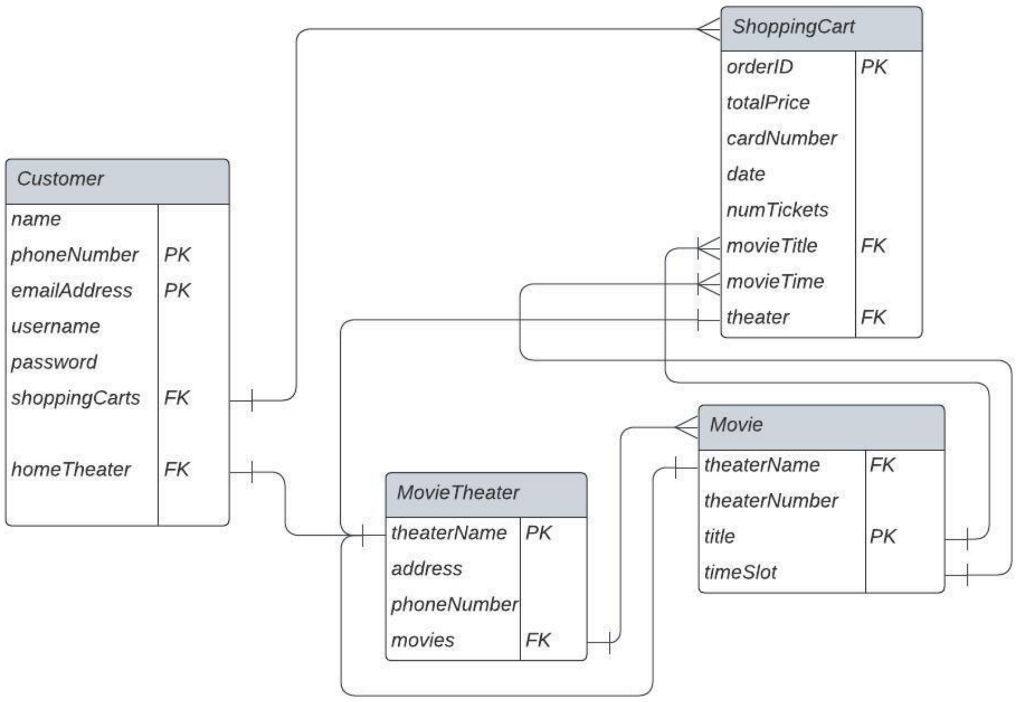
setTheaterName(string name): This operation sets the name of a particular theater in the cinema chain by passing in a string as an argument

setPhoneNumber(double number): This operation sets the phone number of a particular theater in the cinema chain by passing in a double as an argument

setMovies(Movies[]): This operation sets the list of films being screened at a particular theater by passing in an array of objects of the class type Movies (explained previously) as an argument

Data Management Strategy

Data Management (Overview)



Data Management: Customers

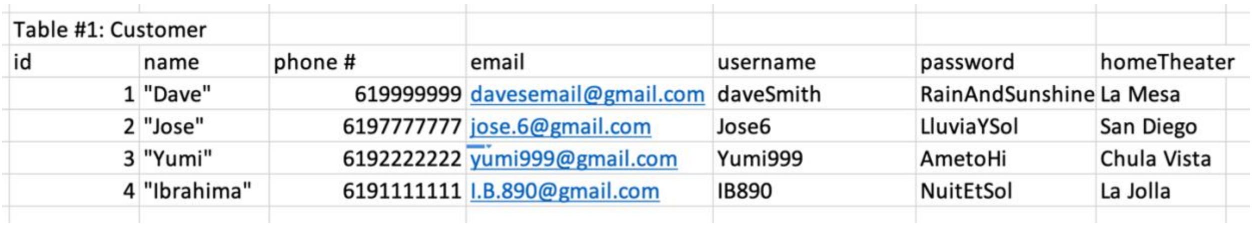


Table #1: Customer						
id	name	phone #	email	username	password	homeTheater
1	"Dave"	619999999	davesemail@gmail.com	daveSmith	RainAndSunshine	La Mesa
2	"Jose"	6197777777	jose.6@gmail.com	Jose6	LluviaYSol	San Diego
3	"Yumi"	6192222222	yumi999@gmail.com	Yumi999	AmetoHi	Chula Vista
4	"Ibrahima"	6191111111	I.B.890@gmail.com	IB890	NuitEtSol	La Jolla

Data Management: Movie Theater

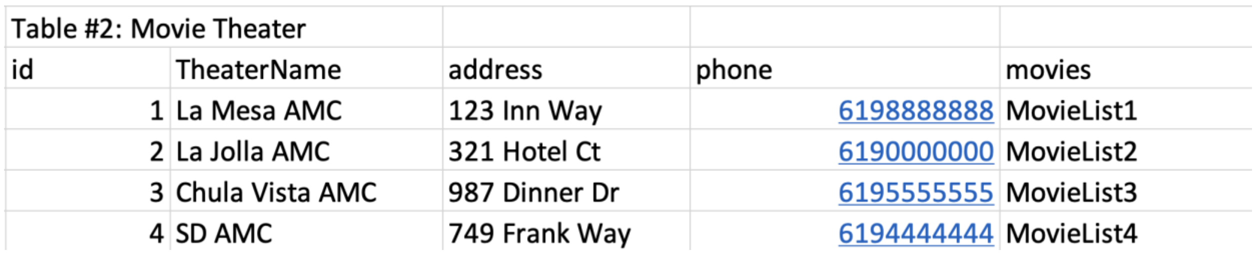


Table #2: Movie Theater				
id	TheaterName	address	phone	movies
1	La Mesa AMC	123 Inn Way	6198888888	MovieList1
2	La Jolla AMC	321 Hotel Ct	6190000000	MovieList2
3	Chula Vista AMC	987 Dinner Dr	6195555555	MovieList3
4	SD AMC	749 Frank Way	6194444444	MovieList4

The goal of this database layout is to make it easy to figure out what movies are currently playing at the theater, and to go back and figure out a customer's order history. The program already somewhat resembles a database because of the heavy use of arrays to store data, so the ER diagram will look and function similarly to the UML diagram. There are four entities: Customer, ShoppingCart, MovieTheater, and Movie. Because of the small volume of data being tracked, we only only need one database.
The above graphics show, in order, the broad layout of the database, a close up look of the data containing the customers, and finally a close up look of the data containing the movie theaters within the chain.

- Customer:
    - attributes:
        - name: the person's name. because names can be shared by people, it won't be used as an identifier.
        - phoneNumber: their number, which can be used as a primary key to identify the customer. However it's more likely to change than the email address, so the email address is usually used instead.
        - emailAddress: their email address is the preferred primary key for identifying the customer.

        - username/password: because the email address is the main primary key, the user can change their username and password as often as they'd like.
        - shoppingCarts: users have an array of ShoppingCart(s), most of which are to record past orders, so it has a one-to-many relationship with the ShoppingCart entity.
        - homeTheater: this is the movie theater designated by the customer as their usual theater, or the one they have seen the most movies at.
- ShoppingCart
    - attributes:
        - orderID: this is the primary key used to identify a customer's order. It is simply the index of an order in the shoppingCarts array. The index increments from zero, so their first order is at index zero and the most recent order is going to be the highest index.
        - totalPrice: The total amount billed for this order.
        - cardNumber: the last 4 digits of the debit or credit card used to complete the order.
        - date: the date and time the order was made
        - numTickets: the number of tickets ordered
        - movieTitle: the title(s) of the movies in the order. Since a customer can choose to see multiple films, it is a one-to-many relationship.
        - movieTime: the time slots filled by the order. Since a customer can choose to see multiple films, it is a one-to-many relationship.
        - theater: the theater the movies were being shown at.

- MovieTheater

        - attributes:
            - theaterName: a primary key used to identify the theater, since no two cinemas can have the same name.
            - address: the theater’s address
            - phoneNumber: the theater’s phone number
            - movies: each theater has an array of Movie objects, so this attribute will be a one-to-many relationship with the Movie Entity.
    - Movie
        - attributes:
            - theaterName: the name of the theater, taken from the parent class.
            - theaterNumber: the specific theater within the building that the movie is being shown in.
            - title: the title of the movie. A movie’s title is unique and can be used as a primary key.
            - timeSlot: the times at which a movie is being shown.

Tradeoff Discussion:
For this project, we will move forward with SQL as our technology, and we will utilize a single database.

We have decided to move forward with SQL technology for the following:

- SQL has a well-designed pre-defined schema for structured data.
- SQL databases are table based in the form of row & columns and must strictly adhere to standard schema definition. With the nature of the software (a cinema application) this seems most suitable.
- SQL is good for complex data with clear relationships between data sets.
- SQL is the best fit for high transaction-based applications – A necessity in this software.

Furthermore, we have decided to not use a non-SQL technology for the following:

- Non-SQL lacks in the ability to perform dynamic operations. It can't guarantee ACID properties.
- Consistency being a must, and with no expected large-scale changes in data to occur. Non-SQL will suffer.
- Non-SQL is worse for run-time flexibility in comparison to SQL.