2022

CO567

Haroon, Hassan, Mauro

Contents

Section A - Design 2

Use Cases and Descriptions 2

Class Diagram 7

Data Dictionary 11

Report 15

Section B - Implementation 16

Statement of Requirements 16

Class Diagram 17

Sequence Model 18

Pseudo – Code Detailed Designs 19

A) Login System: Sign up 19

B) Login System: Login 20

C) Login System: Account Selector 20

D) Booking System: View Events 21

E) Booking System: View Shows 21

F) Venue Manager: Add Event 22

G) Venue Manager: Add Show 22

H) Venue Manager: Add Ticket Agent 23

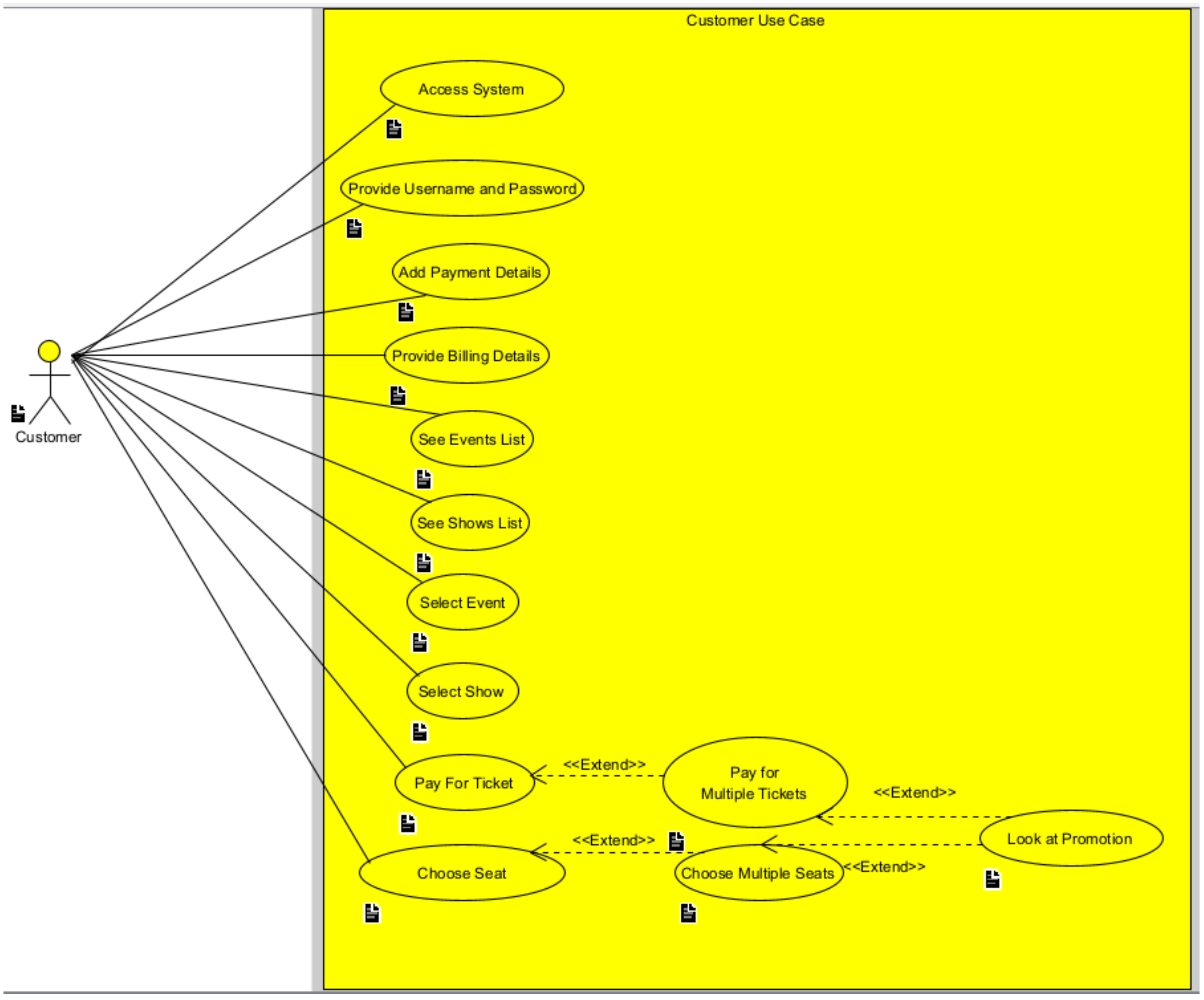
Implemented Code 24

Testing Regime 25

Report 28

Section A - Design

Use Cases and Descriptions



Customer - The Customer will be able to access the Online Ticket System to buy a ticket for an event or show

Access System - The consumer must be able to access the ticket system via the world wide web from any device that can access the Internet, in this ticket system they will be allowed to buy tickets for any shows present or future given a limited date range

Provide Username and Password - To get into the ticket system the Consumer must Provide their Details such as a unique username and password that will authenticate that it is the specific customer who wishes to access the system.

Add Payment Details - The Customer must Add their Payment Details such as their credit or debit card details, as this will be the only form of payment for the purchases on the system

Provide Billing Details - The Customer must provide a billing address when they first access the online ticket system, this address will be used to deliver the tickets to the correct address.

See Events List - When the Consumer chooses an Event they will be given a list that is compiled of Events from random dates they can choose from or The Consumer can give a specific date range such as next week Tuesday and a new list will be complied giving the events going from the present day to the data given in order.

See Shows List - When the Customer chooses a show they will be given a list that is compiled of shows from random dates they can choose from or The Customer can give a specific date range such as next week Tuesday and a new list will be complied giving the shows going from the present day to the data given in order.

Select Event - The Customer will be able to Select an Event they wish to see given from the lists

Select Show - The Customer will be able to Select a Show they wish to see given from the lists

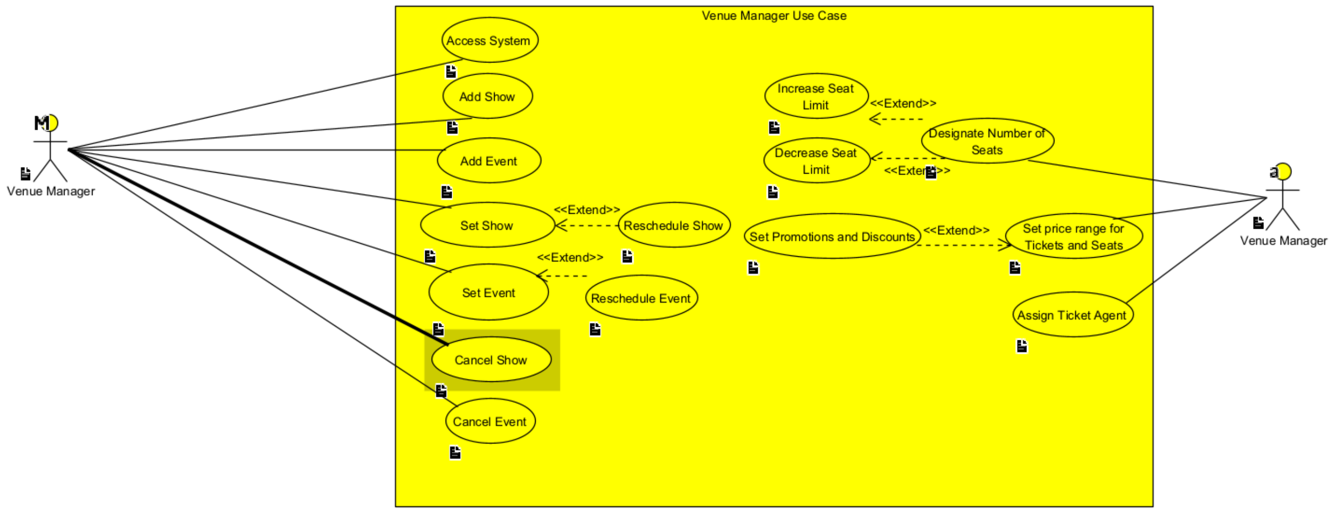
Pay For Ticket - After the Show or Event has been chosen, the Customer must Pay for the Ticket

Pay for Multiple Tickets - If the Customer chooses to buy multiple Tickets they must first identify the limit set by the venue manager for how many Tickets can be bought by one Customer.

Choose Seat - After Paying for the Tickets the customer must choose a seat that they will be able to see from a list that will show which seats are available. The Consumer can choose their seat manually this means being able to see what seats are on hold, have been sold and which ones are available. When the seats have been chosen by the Consumer they will be on hold so no other Consumer could choose that exact seat or seats. The Consumer can choose to allow the Online Ticket System to choose a seat for them, if they wish for this process to take place, they must first input a price range such as what is the minimum they can spend and the maximum they can spend.

Choose Multiple Seats - If the Customer chooses to buy multiple seats, they must first identify the limit set by the venue manager for how many seats can be chosen by one consumer.

Look at Promotion - On the Tickets and Seats there could be a Promotion, as if the Customer buys this many Tickets they could get a certain amount off, and sometimes the Seats will have price differences such as for Child, Student or Adult



Venue Manager - The Venue Manager manages everything about the location of the event or show such as the tickets, prices, seats and layout of the event

Access System - The Venue Manager will be able to Access the System to make sure everything is ready for the Events and Shows.

Add Event - The Venue Manager will be allowed to add an event to the Venue such as what the event, the time of the event, who is performing and how many people are allowed in the venue

Add Show - The Venue Manager will be allowed to add an Show to the Venue such as what is the show, the time of the show, who is performing and how many people are allowed in the venue

Set Show - A Venue Manager can Set a Show this means to be able to Set a date and time for a performer to be seen by the Customer.

Set Event - A Venue Manager can Set an Event this means to be able to Set a date and time for a performer to be seen by the Customer.

Reschedule Show - A Venue Manager can Reschedule a Show this could be caused by shows clashing such as the next booking will be overlapped by the previous performer so they will be rescheduled to the next available time or maybe the weather is not so good so the performer will want to perform in better conditions

Reschedule Event - A Venue Manager can Reschedule an Event this could be caused by events clashing such as the next booking will be overlapped by the previous performer so they will be rescheduled to the next available time or maybe the weather is not so good so the performer will want to perform in better conditions

Cancel Show - The Venue Manager will be allowed to Cancel a Show this could be caused by payment issues, not enough audience, weather or disagreements

Cancel Event - The Venue Manager will be allowed to Cancel an Event this could be caused by payment issues, not enough audience, weather or disagreements

Designate Number Of Seats - A Venue manager must designate the number of seats in an event this must be done so the tickets will have a sale limit as if the tickets sold have gone over the capacity for the number of people allowed in the event this will cause issues and the venue manager will have to sort out refunds

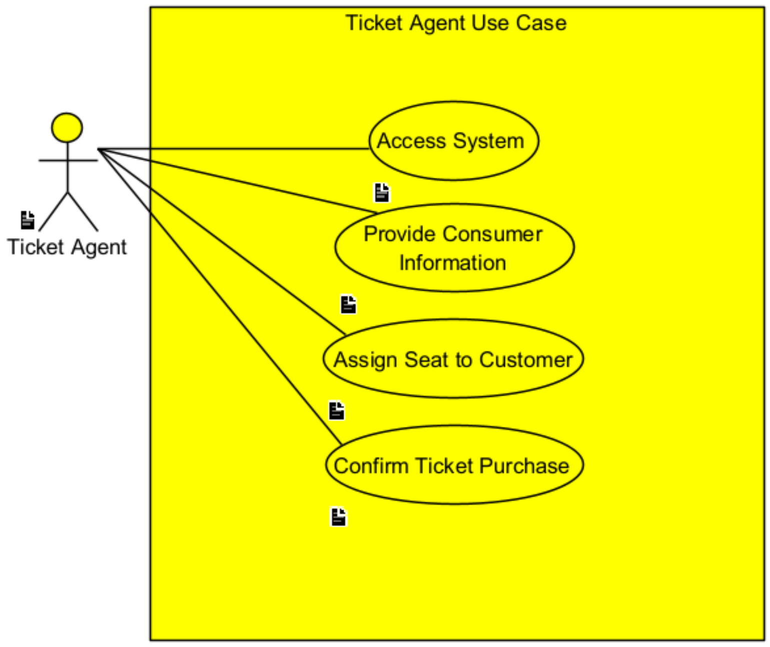
Increase Seat Limit - The Venue manager could choose to increase the seat limit for the event this could be due to the reason that the performer is worldwide known and the Venue Manager knows that many people would come to see this person. In addition, the seat limit for one customer could be increased to allow promotions as one consumer may buy tickets for family and friends.

Decrease Seat Limit - The venue manager could choose to decrease the number of seats this would be done to allow 2 meters distance between the audience in case of a pandemic. In addition, the seat limit for one consumer could be decreased as sometimes consumers could buy many tickets to sell outside and gain profit.

Set Price Range For Ticket And Seats - There has to be a Price Range for the Seats and Tickets this includes, different prices for an adult, student, child and senior citizen.

Set Promotions And Discounts - Seats and Tickets can have promotions on such as if you buy 10 Tickets at once you will receive 10% off your total. For the Seats there will usually be different prices for a Child, Student or Adult

Assign Ticket Agent - The Venue Manager must Assign a Ticket Agent when adding an Event or Show, this individual will help with promotions, discounts, seat and ticket pricing and much more.



Ticket Agent - The Ticket Agent is the individual who is in charge of the buying and selling of the tickets as when the consumers buy a ticket, the agent must verify the purchase by using the OTS(Online Ticket System)

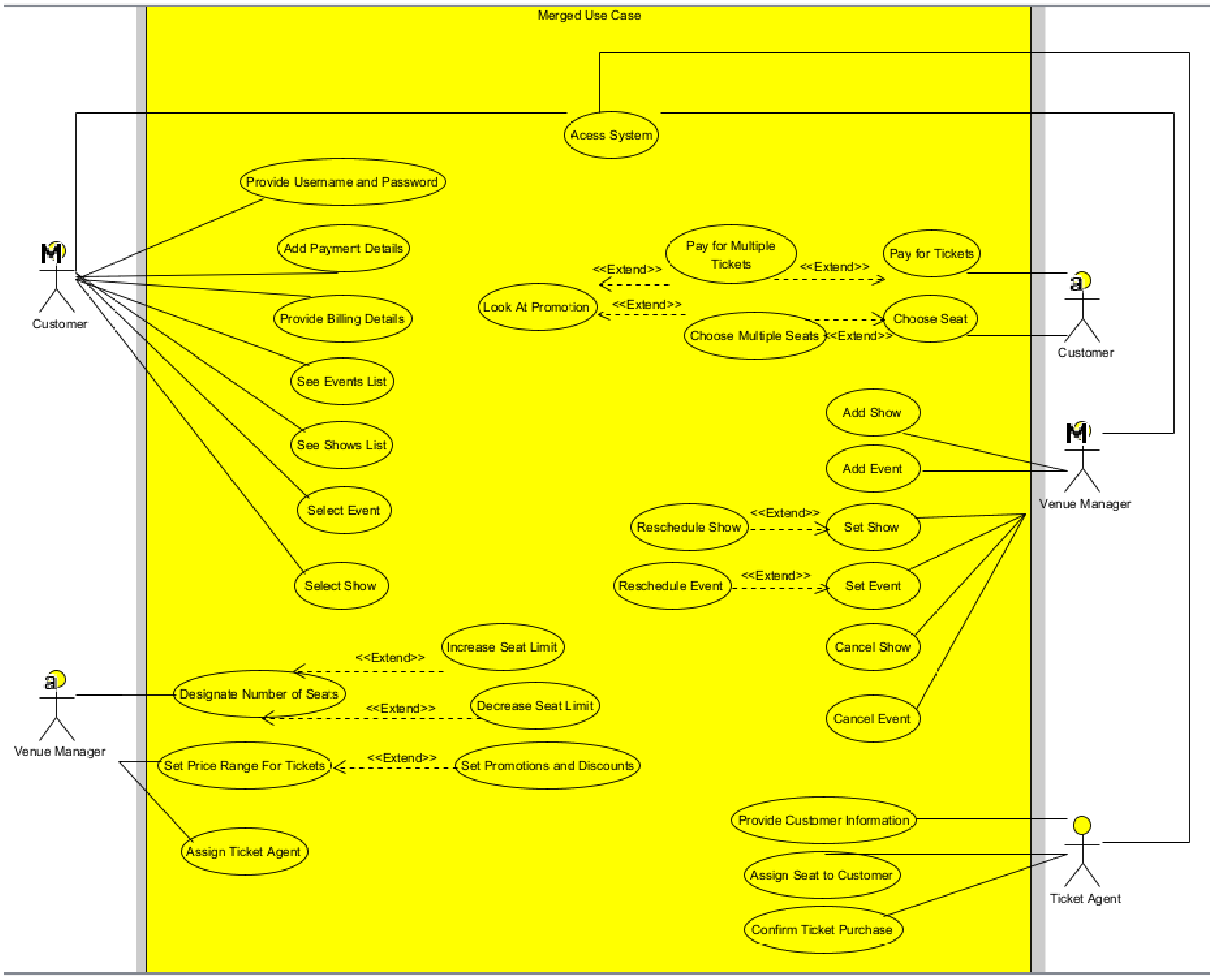
Access System - Once the Customers have purchased a ticket(s) the ticket agent must verify this by using the Online Ticket System.

Provide Consumer Information - Providing the customer information will able the ticket agent to verify the tickets on the correct Customer

Assign Seat to Customer - After the Customer has bought the Ticket or Tickets the Ticket Agent must assign that Ticket(s) to the Customer who has bought it

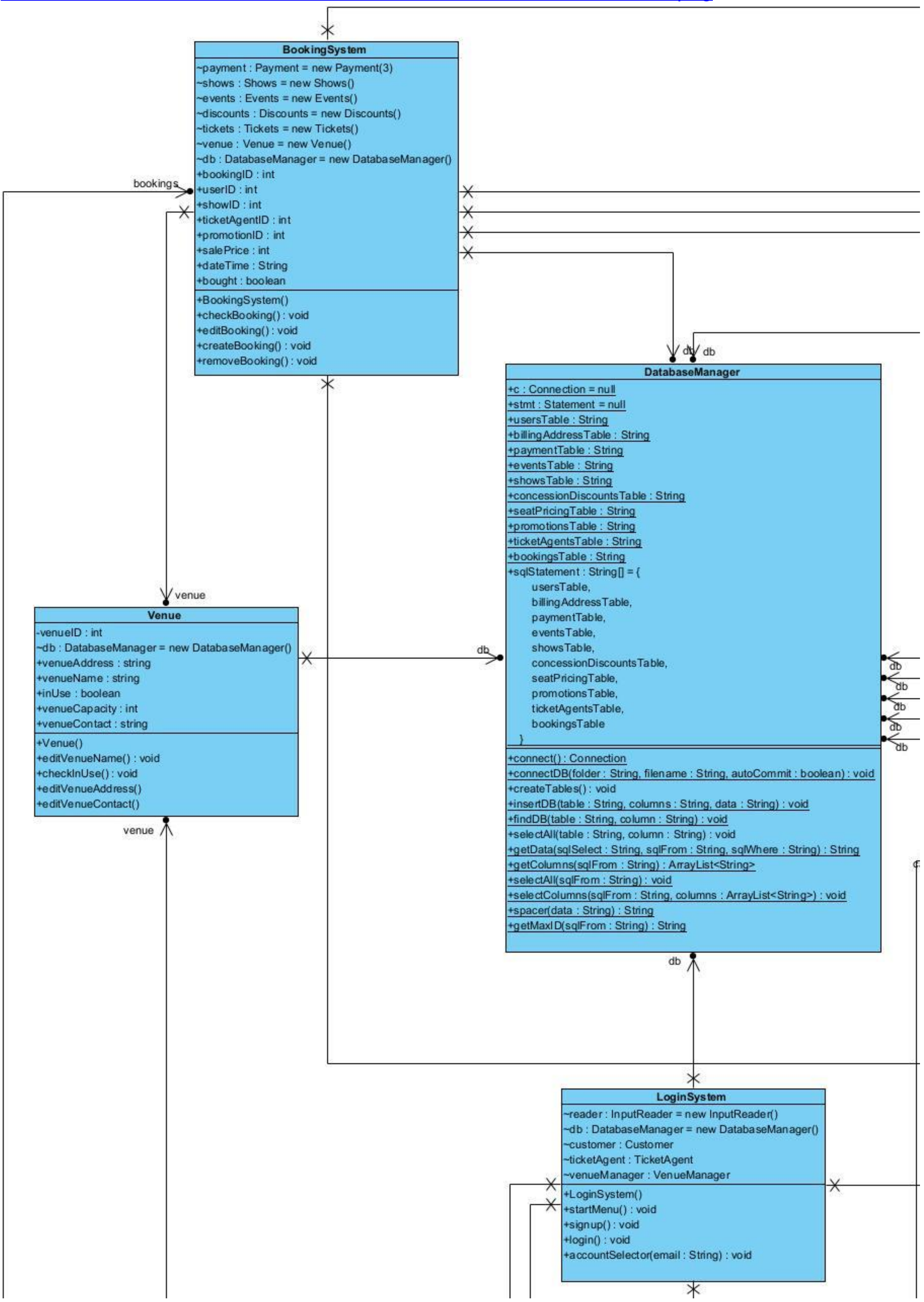
Confirm Ticket Purchase - After assigning the Ticket(s) to the Customer the Ticket Agent will need to confirm the purchase of the Ticket(s) that the Customer has chosen, this will provide on the system that the Customer has bought a ticket and has been confirmed/validated by a Ticket Agent

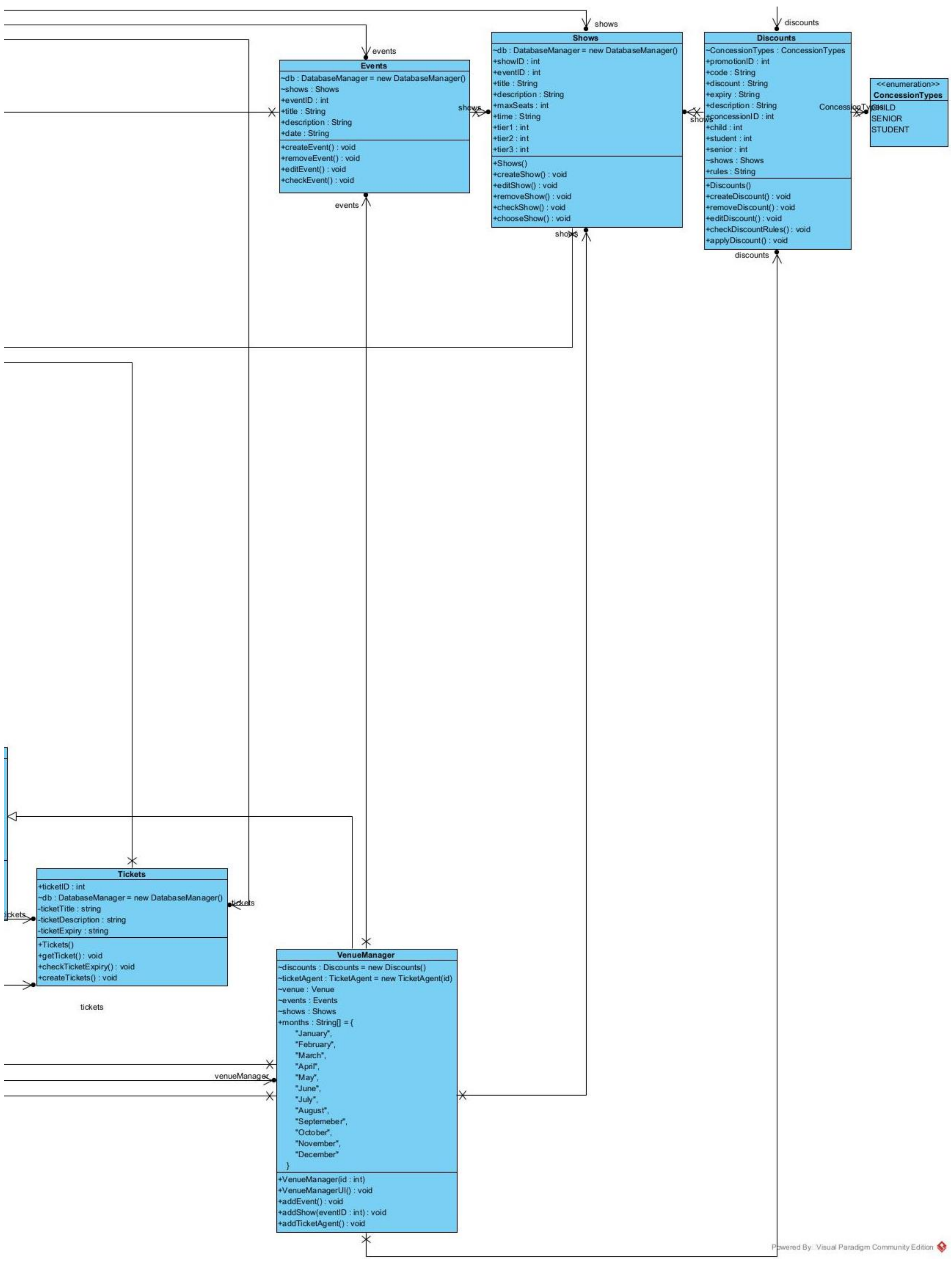
Merged Use Case:

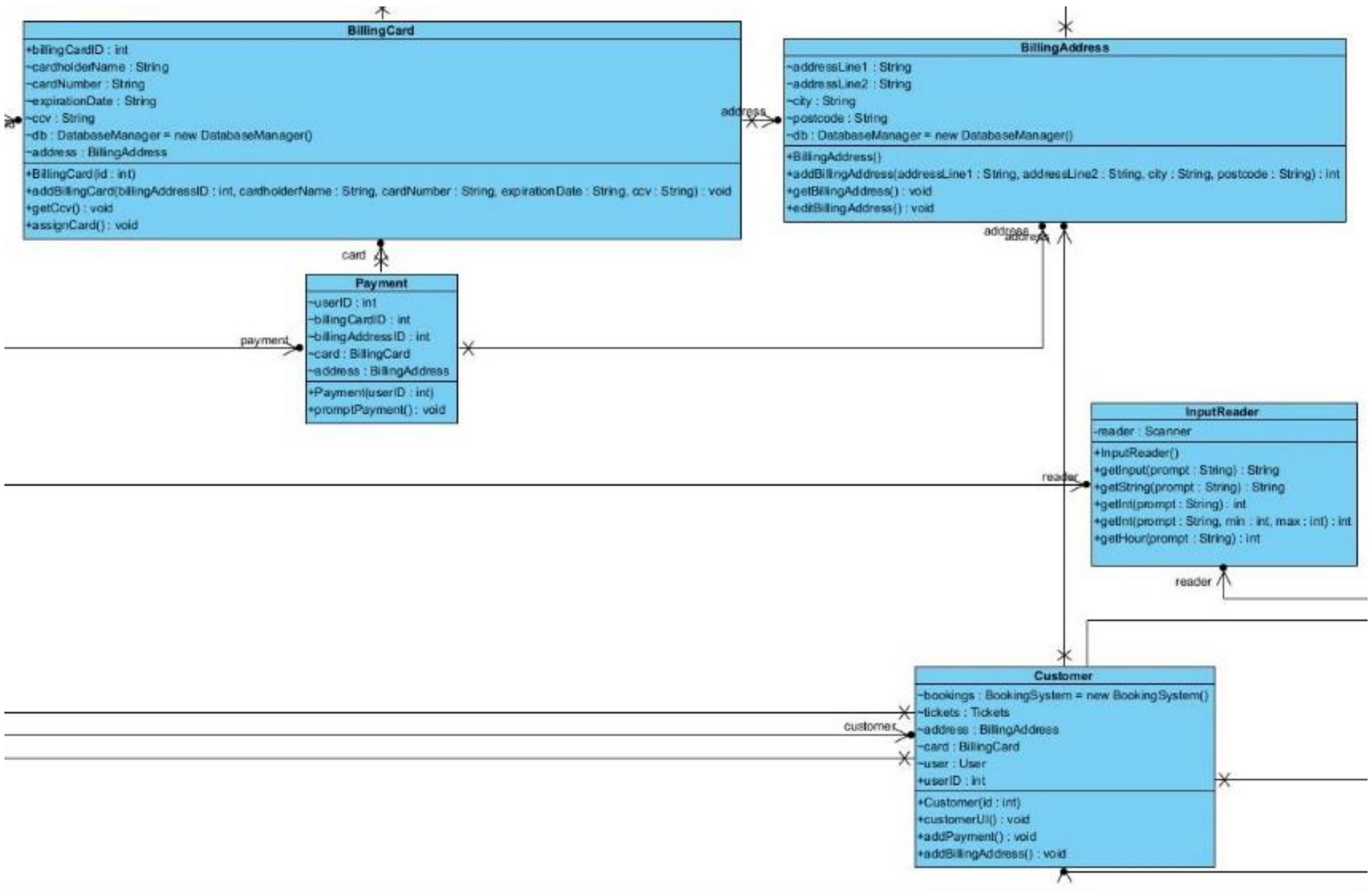


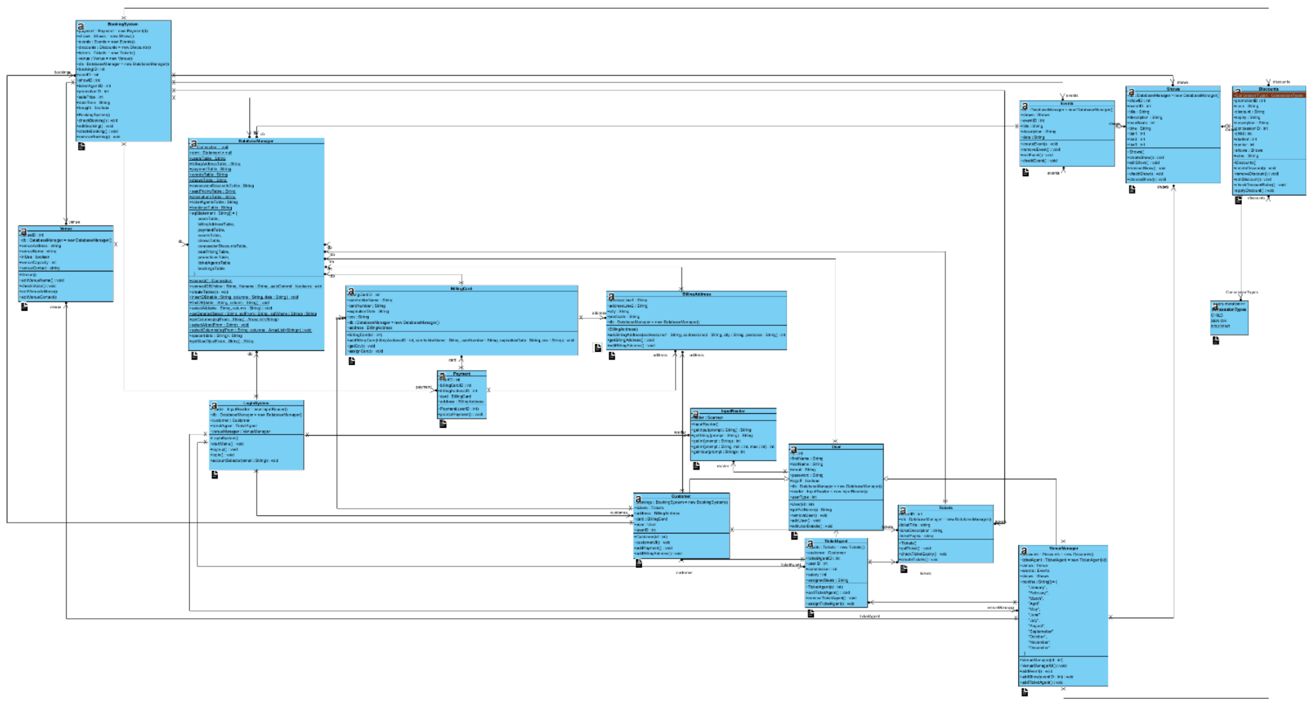
Class Diagram

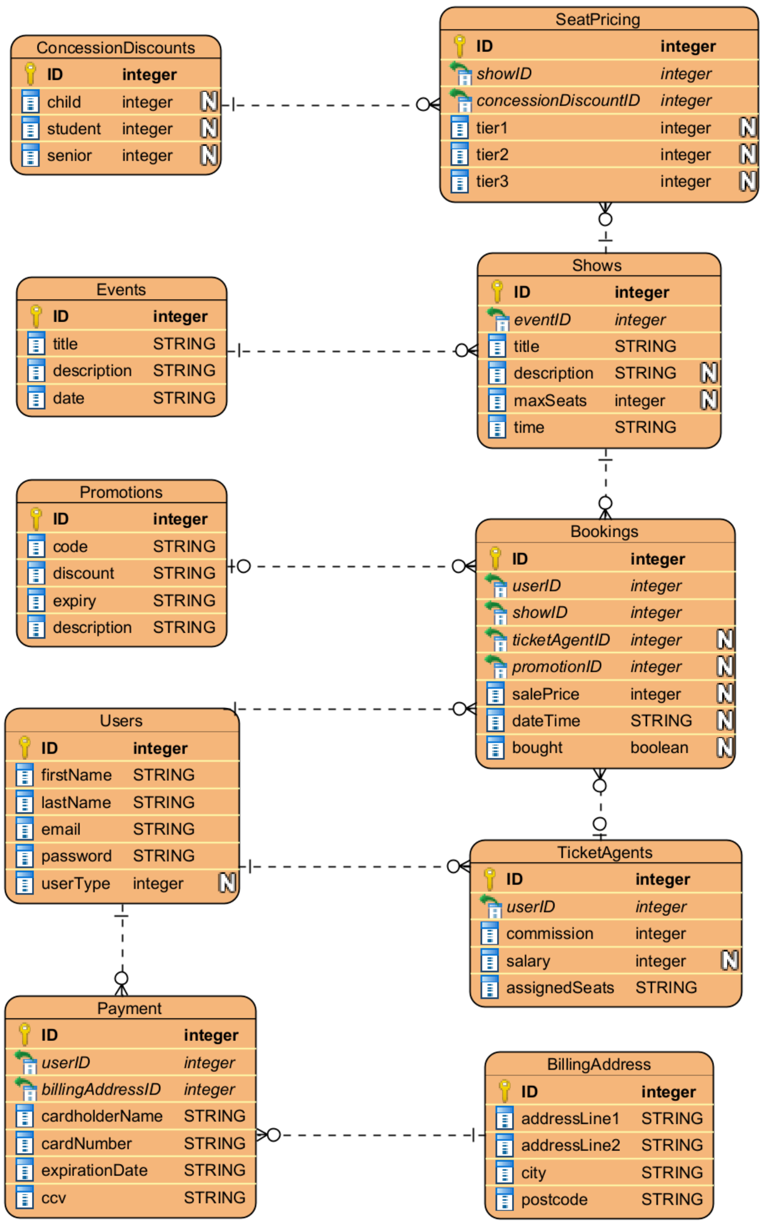
Full Diagram Link: https://github.com/hsadiq32/CO567-OOSD/blob/main/Screenshots/Screenshot%202022-01-13%20200343.png


BookingSystem
~payment : Payment = new Payment(3)
~shows : Shows = new Shows()
~events : Events = new Events()
~discounts : Discounts = new Discounts()
~tickets : Tickets = new Tickets()
~venue : Venue = new Venue()
~db : DatabaseManager = new DatabaseManager()
+bookingID : int
+userID : int
+showID : int
+ticketAgentID : int
+promotionID : int
+salePrice : int
+dateTime : String
+bought : boolean
+BookingSystem()
+checkBooking() : void
+editBooking() : void
+createBooking() : void
+removeBooking() : void
bookings
venue
db
db
db
Venue
-venueID : int
~db : DatabaseManager = new DatabaseManager()
+venueAddress : string
+venueName : string
+inUse : boolean
+venueCapacity : int
+venueContact : string
+Venue()
+editVenueName() : void
+checkInUse() : void
+editVenueAddress()
+editVenueContact()
venue
db
DatabaseManager
+c : Connection = null
+stmt : Statement = null
+usersTable : String
+billingAddressTable : String
+paymentTable : String
+eventsTable : String
+showsTable : String
+concessionDiscountsTable : String
+seatPricingTable : String
+promotionsTable : String
+ticketAgentsTable : String
+bookingsTable : String
+sqlStatement : String[] = {
usersTable,
billingAddressTable,
paymentTable,
eventsTable,
showsTable,
concessionDiscountsTable,
seatPricingTable,
promotionsTable,
ticketAgentsTable,
bookingsTable
}
+connect() : Connection
+connectDB(folder : String, filename : String, autoCommit : boolean) : void
+createTables() : void
+insertDB(table : String, columns : String, data : String) : void
+findDB(table : String, column : String) : void
+selectAll(table : String, column : String) : void
+getData(sqlSelect : String, sqlFrom : String, sqlWhere : String) : String
+getColumns(sqlFrom : String) : ArrayList<String>
+selectAll(sqlFrom : String) : void
+selectColumns(sqlFrom : String, columns : ArrayList<String>) : void
+spacer(data : String) : String
+getMaxID(sqlFrom : String) : String
db
db
db
db
db
db
LoginSystem
~reader : InputReader = new InputReader()
~db : DatabaseManager = new DatabaseManager()
~customer : Customer
~ticketAgent : TicketAgent
~venueManager : VenueManager
+LoginSystem()
+startMenu() : void
+signup() : void
+login() : void
+accountSelector(email : String) : void


shows
discounts
events
Events
~db : DatabaseManager = new DatabaseManager()
~shows : Shows
+eventID : int
+title : String
+description : String
+date : String
+createEvent() : void
+removeEvent() : void
+editEvent() : void
+checkEvent() : void
events
shows
Shows
~db : DatabaseManager = new DatabaseManager()
+showID : int
+eventID : int
+title : String
+description : String
+maxSeats : int
+time : String
+tier1 : int
+tier2 : int
+tier3 : int
+Shows()
+createShow() : void
+editShow() : void
+removeShow() : void
+checkShow() : void
+chooseShow() : void
shows
Discounts
~ConcessionTypes : ConcessionTypes
+promotionID : int
+code : String
+discount : String
+expiry : String
+description : String
+concessionID : int
+child : int
+student : int
+senior : int
~shows : Shows
+rules : String
+Discounts()
+createDiscount() : void
+removeDiscount() : void
+editDiscount() : void
+checkDiscountRules() : void
+applyDiscount() : void
discounts
shows
ConcessionTypes
<<enumeration>>
ConcessionTypes
CHILD
SENIOR
STUDENT
Tickets
+ticketID : int
~db : DatabaseManager = new DatabaseManager()
-ticketTitle : string
-ticketDescription : string
-ticketExpiry : string
+Tickets()
+getTicket() : void
+checkTicketExpiry() : void
+createTickets() : void
tickets
tickets
tickets
VenueManager
~discounts : Discounts = new Discounts()
~ticketAgent : TicketAgent = new TicketAgent(id)
~venue : Venue
~events : Events
~shows : Shows
+months : String[] = {
"January",
"February",
"March",
"April",
"May",
"June",
"July",
"August",
"Septemeber",
"October",
"November",
"December"
}
+VenueManager(id : int)
+VenueManagerUI() : void
+addEvent() : void
+addShow(eventID : int) : void
+addTicketAgent() : void
venueManager
Powered By: Visual Paradigm Community Edition


BillingCard
+billingCardID : int
~cardholderName : String
~cardNumber : String
~expirationDate : String
~ccv : String
~db : DatabaseManager = new DatabaseManager()
~address : BillingAddress
+BillingCard(id : int)
+addBillingCard(billingAddressID : int, cardholderName : String, cardNumber : String, expirationDate : String, ccv : String) : void
+getCcv() : void
+assignCard() : void
address
BillingAddress
~addressLine1 : String
~addressLine2 : String
~city : String
~postcode : String
~db : DatabaseManager = new DatabaseManager()
+BillingAddress()
+addBillingAddress(addressLine1 : String, addressLine2 : String, city : String, postcode : String) : int
+getBillingAddress() : void
+editBillingAddress() : void
address
address
card
Payment
~userID : int
~billingCardID : int
~billingAddressID : int
~card : BillingCard
~address : BillingAddress
+Payment(userID : int)
+promptPayment() : void
payment
InputReader
~reader : Scanner
+InputReader()
+getInput(prompt : String) : String
+getString(prompt : String) : String
+getInt(prompt : String) : int
+getInt(prompt : String, min : int, max : int) : int
+getHour(prompt : String) : int
reader
reader
Customer
~bookings : BookingSystem = new BookingSystem()
~tickets : Tickets
~address : BillingAddress
~card : BillingCard
~user : User
+userID : int
+Customer(id : int)
+customerUI() : void
+addPayment() : void
+addBillingAddress() : void
customer




ConcessionDiscounts
ID integer
child integer
student integer
senior integer
SeatPricing
ID integer
showID integer
concessionDiscountID integer
tier1 integer
tier2 integer
tier3 integer
Events
ID integer
title STRING
description STRING
date STRING
Shows
ID integer
eventID integer
title STRING
description STRING
maxSeats integer
time STRING
Promotions
ID integer
code STRING
discount STRING
expiry STRING
description STRING
Bookings
ID integer
userID integer
showID integer
ticketAgentID integer
promotionID integer
salePrice integer
dateTime STRING
bought boolean
Users
ID integer
firstName STRING
lastName STRING
email STRING
password STRING
userType integer
TicketAgents
ID integer
userID integer
commission integer
salary integer
assignedSeats STRING
Payment
ID integer
userID integer
billingAddressID integer
cardholderName STRING
cardNumber STRING
expirationDate STRING
ccv STRING
BillingAddress
ID integer
addressLine1 STRING
addressLine2 STRING
city STRING
postcode STRING

Data Dictionary

ConcessionTypes		
Attribute	**Data Type**	**Description**
ID	integer	Unique discount ID
child	integer	Child discount ID
student	integer	Student discount ID
senior	integer	Senior discount ID

Events		
ID	integer	Unique event ID
title	string	Name of event
description	string	Description of what a given event entails
date	string	Date and time of a given event

Shows		
ID	integer	Unique Show ID
showID	integer	Foreign Key
eventID	integer	Foreign Key
title	string	Show’s title identifier
description	string	Brief description of a show’s plot
maxSeats	integer	Maximum number of seats for a given viewing of a show
time	string	Show’s runtime
tier1	integer	First tier pricing
tier2	integer	Second tier pricing
tier3	integer	Third tier pricing

Discounts		
ID	integer	Unique promotion ID
code	string	Activation code
rules	string	Set of rules for promotions
discount	string	Percentage of promotion discount
expiry	string	Final date permitted for use of promotion
description	string	Informative text about promotion
concessionID	integer	Unique concession ID
child	integer	Child discount ID
student	integer	Student discount ID
senior	integer	Senior discount ID

BookingSystem		
ID	integer	Unique booking ID
userID	Integer	Foreign Key
showID	Integer	Foreign Key
ticketAgentID	Integer	Foreign Key
promotionID	integer	Foreign Key
salePrice	integer	Baseline cost of a booking
dateTime	string	Date and time of a given booking
bought	boolean	Status of a given booking

User		
ID	integer	Unique user ID
firstName	string	User’s first name
lastName	string	User’s last name
email	string	Email on user’s account
password	string	Secret user login password
userType	integer	Identifier for recognizing what the user likes
logoff	boolean	Status of user presence

Customer		
userID	integer	ForeignKey
tickets	Tickets	Data from class Tickets
address	BillingAddress	Data from class BillingAddress
card	BillingCard	Data from class Card
user	User	Data from class User

TicketAgent		
ID	integer	Unique ticket agent ID
userID	integer	Foreign Key
commission	integer	Monetary gain per sale
salary	integer	Baseline ticket agent wage
assignedSeats	string	Assigned seat range from Venue Manager

BillingCard		
ID	integer	Unique payment ID
cardholderName	string	Name on a customer’s payment card
cardNumber	string	Payment card number identifier
expirationDate	string	Expiration date of customer’s payment card
ccv	string	3 digit number on back of customer’s payment card

Payment		
userID	integer	Foreign Key
billingCardID	integer	Foreign Key
billingAddressID	integer	Foreign Key

BillingAddress		
ID	integer	Unique billing address ID
addressLine1	string	First line of customer’s billing address
addressLine2	string	Second line of customer’s billing address
city	string	City of billing address
postcode	string	Postcode of billing address

Venue		
ID	integer	Unique venue ID
venueAddress	string	Location of venue
venueName	string	Venue’s current name
inUse	boolean	Status of venue usage
venueCapacity	integer	Maximum capacity of venue
venueContact	string	Preferred contact form of venue

DatabaseManager		
usersTable	string	Data table for Users
billingAddressTable	string	Data table for BillingAddress
paymentTable	string	Data table for Payment
eventsTable	string	Data table for Events
showsTable	string	Data table for Shows
concessionDiscountsTable	string	Data table for ConcessionDiscounts
seatPricingTable	string	Data table for SeatPricing
promotionsTable	string	Data table for Promotions
ticketAgentsTable	string	Data table for TicketAgents
bookingsTable	string	Data table for Bookings

InputReader		
reader	Scanner	Reader for searching details

Tickets		
ticketID	integer	Unique ticket ID
ticketTitle	string	Name identifier for purpose of ticket
ticketDescription	string	Descriptive information for a given ticket
ticketExpiry	string	Expiration date of a given ticket

LoginSystem		
customer	Customer	Data from class Customer
ticketAgent	TicketAgent	Data from class TicketAgent
venueManager	VenueManager	Data from class VenueManager

VenueManager		
venue	Venue	Data from class Venue
events	Events	Data from class Events
shows	Shows	Data from class Shows
months	string	String of months of the year

Report

In this report will contain information of how we designed the start of the BUCKS Centre for the Performing Arts (Design & Implementation). The first section we designed was the three use cases which were designed by Hassan Nisar (22011971). He used Visual Paradigm Community Edition, the reason he used the Community Edition was because he used the 16.3 Evaluation Edition for another module and the 30-day trial had run out also, the apps anywhere Visual Paradigm was not working so he chose to use the Community Edition. He started making the Use Cases by looking at the assignment brief and listing down what each Actor does such as the Consumer, Venue Manager, and the Ticket Agent. After he made the list for each Actor, he done implemented every action they had done into three different Use Cases. After each Use Case he had opened the specification for each one and wrote a description as the Use Cases contain 5 or less words, another individual looking at the Use Case will find it hard to understand so he made a description for each Use Case which he also put into a Word Document.

The next segment that was looked and was a part of the Use Cases was the merging of the Use Cases, this was done by Hassan Nisar. Hassan first took all three Use Cases and laid them all out on a new Diagram, he came across an issue when merging the Use Cases as he could not simply import and export the already made Use Cases on to another diagram, he had to remake all the Use Cases again on to one diagram, this was not his fault as the software the team was using was not very helpful for this task. After, laying out all the Use Cases he then looked at each Use Case and found which Use Cases were similar, could be changed to be similar and the ones that were completely different. The ones that were similar was all Actors had to access the Online Ticket System this could be normalized and instead of three Use Cases for accessing the Online Ticket System it was normalized into one where all three Actors were related to. Some use cases could be seen as similar as the Ticket Agent had to confirm the seat as purchased and the Consumer had to purchase the seat, Hassan tried to make the Use Cases as similar as they could to allow normalization but was unable, so he normalized as many Use Cases as he could.

The Second feature which Haroon Sadiq took responsibility for was the Class diagrams, this task had a bit of everyone's hand in it as the classes found in the study case was done by Hassan, the diagrams were generated by Haroon and the data inside the classes was described by Mauro which is another task in itself. To start of making the classes Hassan done a process called Noun Identifying this allowed Hassan to look through the Case Study and Identify the Main Nouns that were used to construct the BUCKS Centre for the Performing Arts. He ended up with nine nouns that were made into classes to begin with and was handed onto Haroon. Haroon generated the classes through the noun identifier that Hassan had completed, with this Haroon then created a full database schema in Visual Paradigm to augment the way the classes will communicate with each other and bring to light the multiplicity of such entities. Using this, Haroon then created a class diagram incorporating all of the client's needs, with the by-product of this being having a full database design which would be used in the implementation for a robust and professional execution of the case study concerned.

At the halfway point of making the Class Diagrams, Mauro Nunes took the responsibility of generating a Data Dictionary for the Attributes in each class, he had the idea of doing it in between as if he had taken the responsibility at the end, it would have taken him a long time to complete so if he had started it all he needed to be doing was updating it with the new attributes. He decided to generate the Data Dictionary in Word as making a Table is simple in Word and he included the attribute, the type such as integer or string and lastly, the actual definition of the Attribute.

The team met up with each other every Monday from 2pm to 3pm and then every Friday from 3pm to 4pm. Those timings would be affected depending on availability. But, varying on work consumption as if less work was being done more meetings would be set up, if the work was being completed on time the team would have 30-minute meetings just letting each other know what has been completed and what must be completed next. Hassan Nisar took the role as Team Leader, so he oversaw roles and deadlines which were mostly met. The team would mostly keep communications via social media such as letting the group know what has been done and what needs to be done.

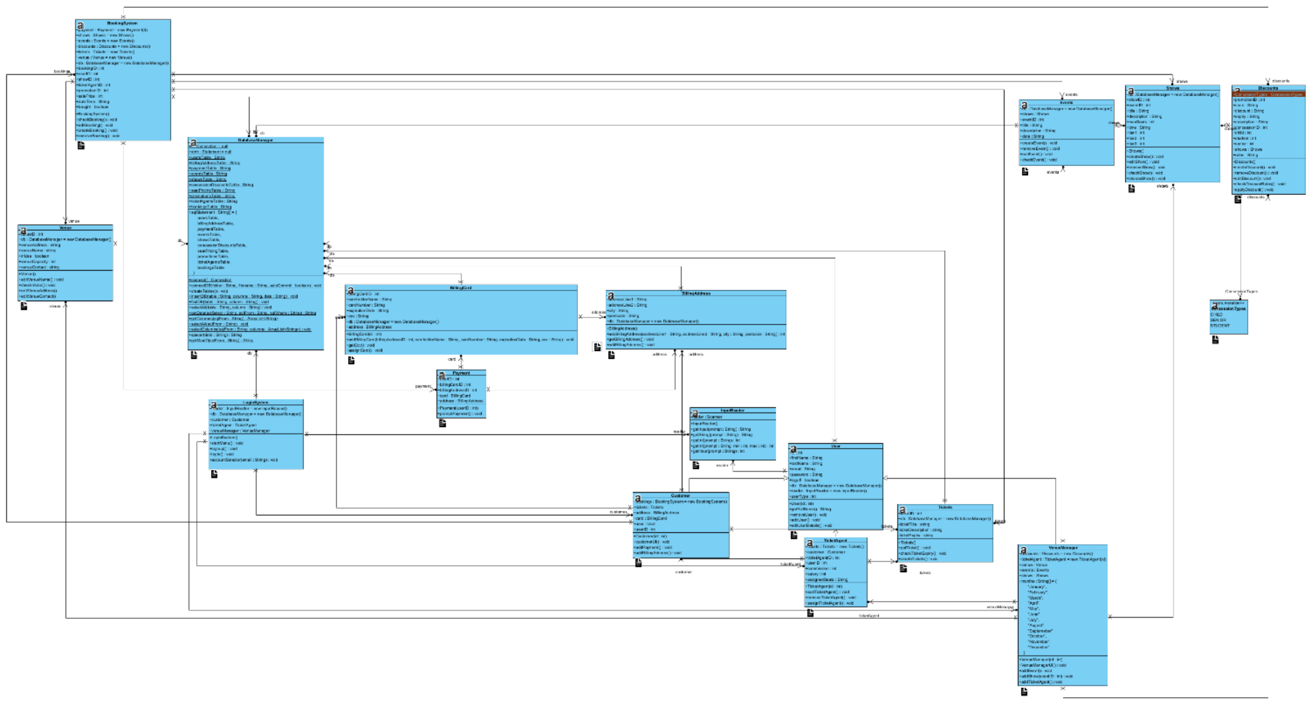
Section B - Implementation

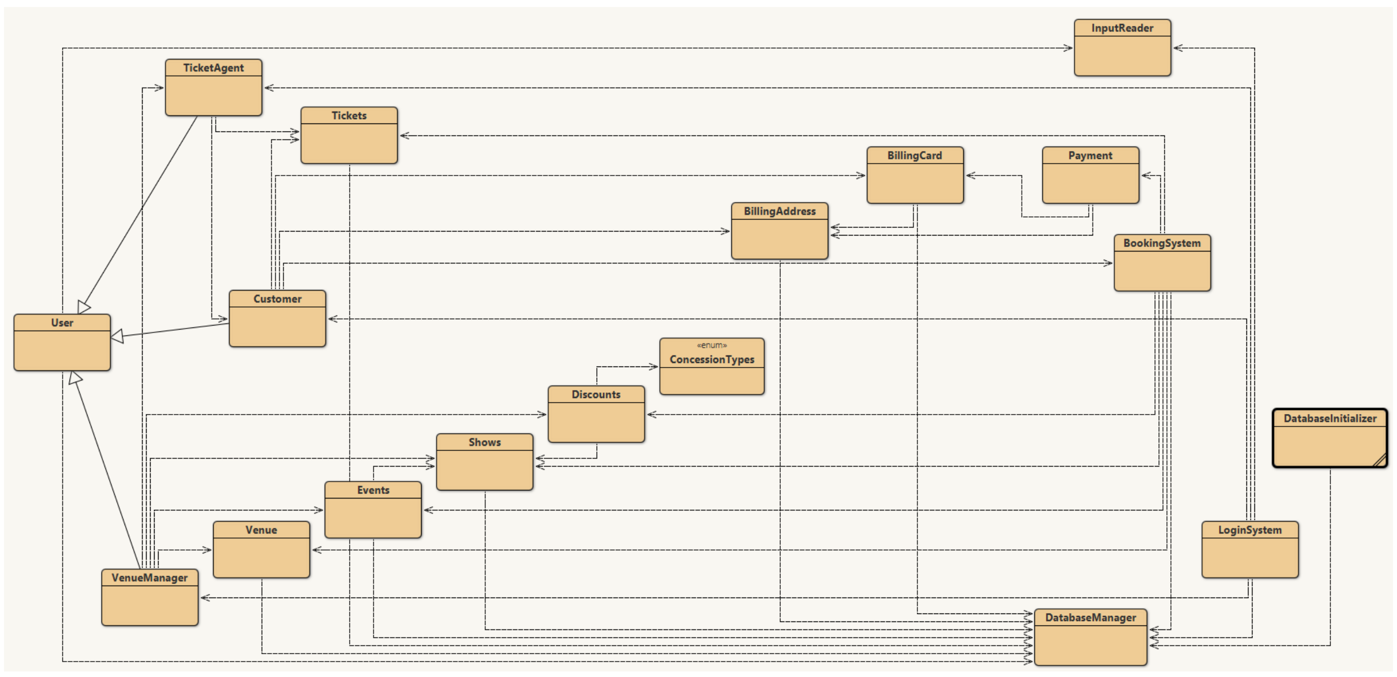
Statement of Requirements

1. Sign up system will not allow new accounts with the same email to avoid conflicts
2. Verify if email is correct with “@” symbol and a TLD (e.g. .com or .co.uk)
3. Customer will be able to sign up on the login system
4. Ticket Agents will be able to sign up via the ticket manager
5. Venue Manager will be able to login successfully with a user type dependant UI
6. Ticket Agent will be able to login successfully with a user type dependant UI
7. Venue Manager will be able to log in successfully with a user type dependant UI
8. Venue Manager will be able to add events
9. Venue Manager will be able to add shows corresponding to events
10. Customer will be able to view events
11. Customer will be able to view shows corresponding to events
12. Customers will be able to hold tickets
13. Customers will require first time billing signup on login, all other user types do not.
14. Input reader will catch invalid data types
15. Database manager will create a fully capable database allowing all data to be stored and read

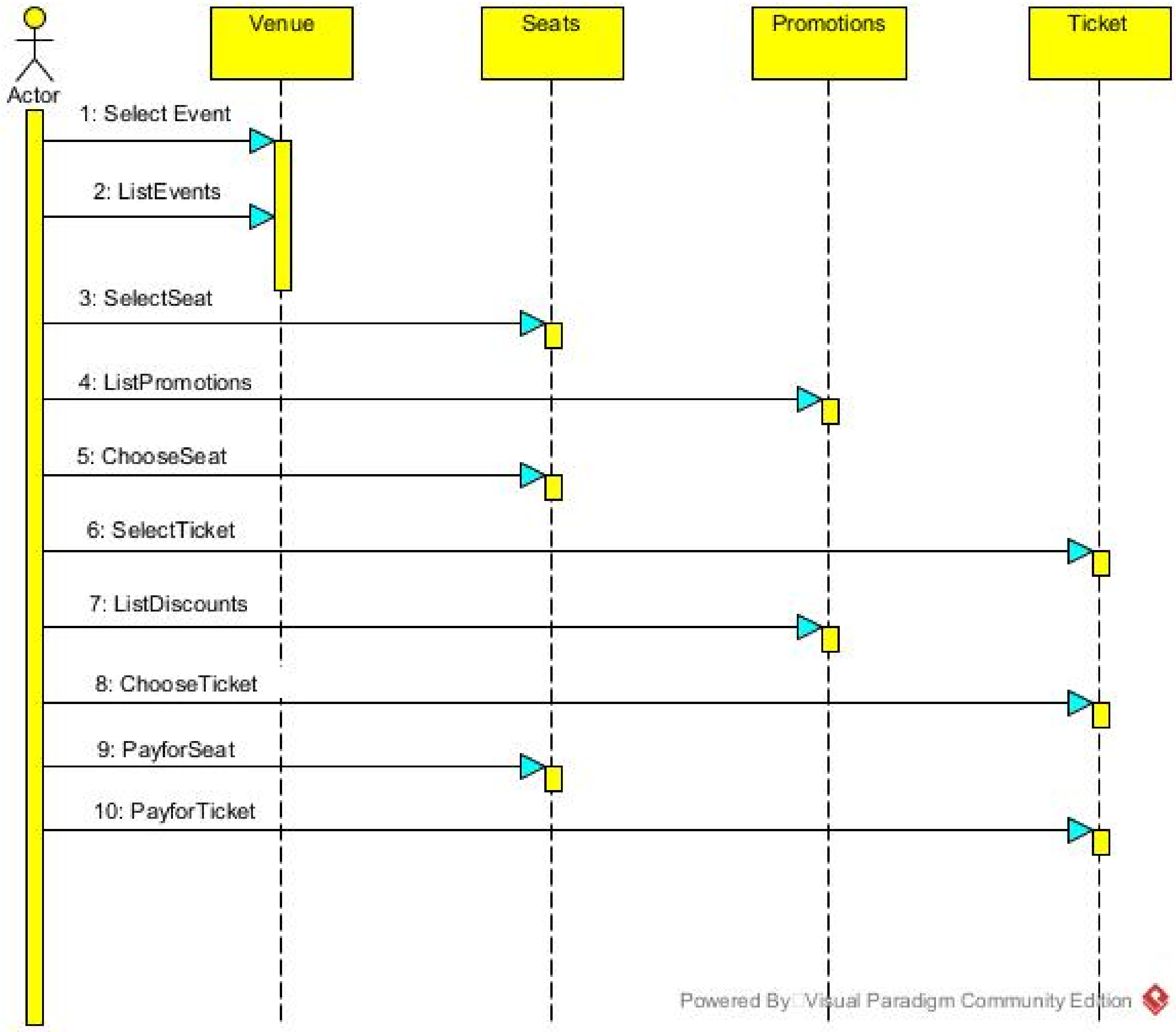
Class Diagram

Full Diagram Link: https://github.com/hsadiq32/CO567-OOSD/blob/main/Screenshots/Screenshot%202022-01-13%20200343.png





Sequence Model



Pseudo – Code Detailed Designs

A) Login System: Sign up

String variables: *first name, last name, email, password, error message*

Integer Variables: *user type*

function signup(*user type*)

```
1. Get userinput for first name, last name, email, password
2. Check if userinput is valid data
    2i. if: first name and last name is: ascii letters only, not null, 20-character max length
        else:
            - add error condition to error message string
    2ii. if: email contains "@" and contains a known TLD (e.g. .com or .co.uk)
            if: email is taken in database – execute SQL database query
            else:
                - add error condition to error message string
        else:
            - add error condition to error message string
    2iii. if: password is not null and 50 characters max length
        else:
            - add error condition to error message string
3. Attempt signup
    3i. if: error message string is null
            - Get Database Connection
            - Create new row of data with ID (auto-increment, unique primary key), first name,
              last name, email, password, user type (default is 0 which is a customer account)
            - Log into customer UI with details
        else:
            - Display error message string
            - error message string set to null for next signup attempt
```

B) Login System: Login

String variables: *email, password, error message*

Integer variables: *user ID*

```
function login()
  1. Get userinput for email, password
  2. Attempt login
    2i. if: email is found in email row in database – execute SQL database query
          if: password == password column in email row in database – execute SQL database query
              - AccountSelector(get user ID - execute SQL database query using email)
          else:
              - add error condition to error message string
        else:
          - add error condition to error message string
     error message string set to null for next login attempt
```

C) Login System: Account Selector

Integer variables: *user ID, user type*

```
function AccountSelector(integer user ID)
  1.  Use user ID to find usertype in the database – execute SQL database query
   1i. get usertype variable - execute SQL database query using email
  2. Select Account type
    2i. if: usertype == 0
          Customer()
        else if: usertype == 1
          TicketAgent()
        else if: usertype == 2
          VenueManager()
        else:
          display error
```

D) Booking System: View Events

```
Integer variables: event ID

function viewEvents()

loop{

   1. Display all available events

        1i. Get all events – execute SQL database query

   2. Let user choose event

     16. 2ii. Userinput choose event ID,

        2iii. if: event ID is found in database – execute SQL database query:

                    - viewShows()

                    - break loop

            else:

                     - Display error message – "Invalid event ID"

} end loop
```

E) Booking System: View Shows

```
Integer variables: show ID

function viewShows()

loop {

   1. Display all available shows

        1i. Display table with eventID conditional factor – execute SQL database query

   2. Let user choose Show

     17. 2ii. Userinput choose show ID,

        2iii. if: show ID is found in database where eventID matches – execute SQL database query:

                    - viewShows()

                    - break loop

            else:

                     - Display error message – "Invalid show ID"

} end loop
```

F) Venue Manager: Add Event

Integer variables: *event ID*

String variables: *title, description, date*

```
function addEvent()
    1.  Get userinput for title, description and date of event
        1i. check if title variable is a valid string
        1ii. check if description variable is a valid string
        1iii. check if date variable is a valid date format
loop{
    2. Insert event into database
        2i. if: title, description, date does not match database row – execute SQL database query
                - break loop
            else:
                - Display “Duplicate data error”
} end loop
```

G) Venue Manager: Add Show

Integer variables: *show ID, event ID*

String variables: *title, description, max seats, time range*

```
function addShow(event ID)
    1.  Get userinput for title, description and date of event
        1i. check if title variable is a valid string
        1ii. check if description variable is a valid string
        1iii. check if max seats variable is a valid integer
        1iv. check if time range variable is a valid time range format (start and end time)
```

continued on next page

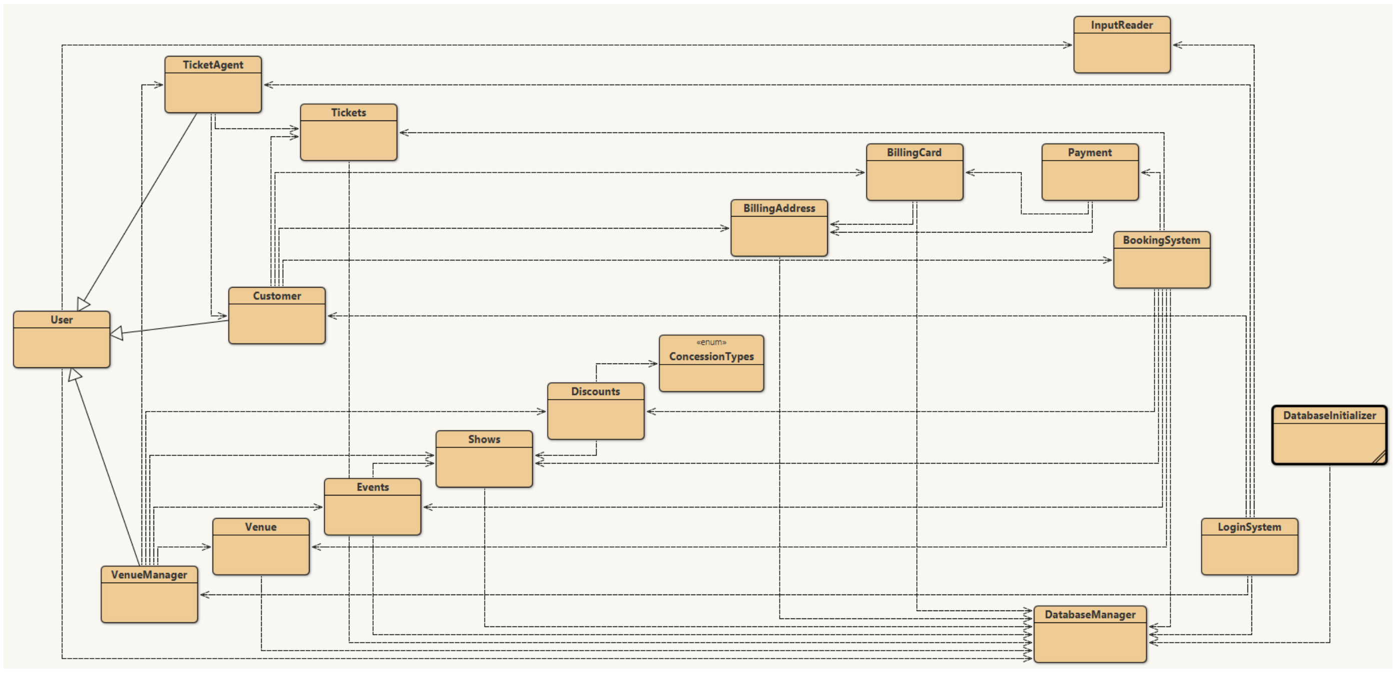
```
loop {
    2. Insert event into database
        2i. if: title, description, max seats, time range, event ID does not match database row where –
            execute SQL database query
                - Create new row of data with ID (auto-increment, unique primary key), title, description, max
                  seats, time range, eventID (foreign key)
                - break loop
            else:
                - Display “Duplicate data error”
} end loop
```

H) Venue Manager: Add Ticket Agent

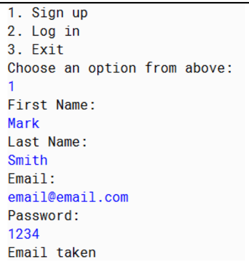
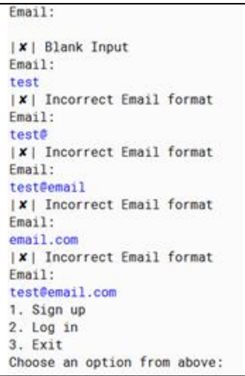
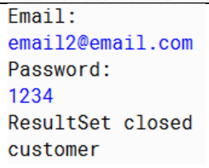
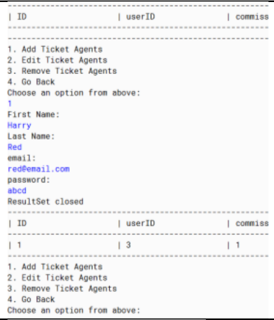
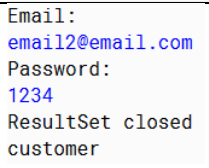
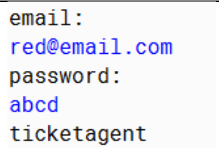
```
Integer variables: user type, user ID, commission, seat start, seat end, salary
String variables: assigned seats
function addTicketAgent()
    1. Add basic user account
        1i. signup(user type = 1)
        1ii. Get user ID
    2. Verify ticket agent data
        2i. check userinput if commission variable is a valid string
        2ii. check userinput if seat start variable is a valid string
        2iii. check userinput if seat end variable is a valid integer
        2iv. assigned seats equals: seat start combined with “-“ combined with seat end
    3. Add ticket agent account linked to user ID
        3i. Create new row of data in TicketAgent table with ID (auto-increment, unique primary key),
    commission, seat start, seat end, salary (default 0), user ID (foreign key)
```

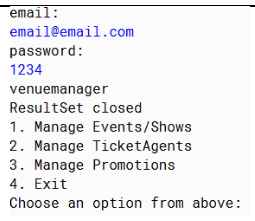
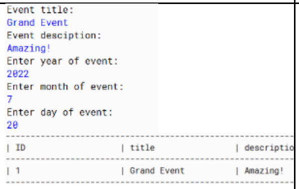
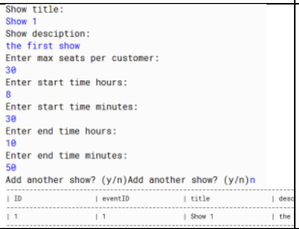
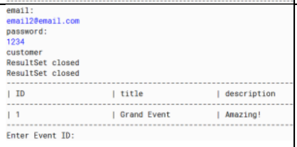
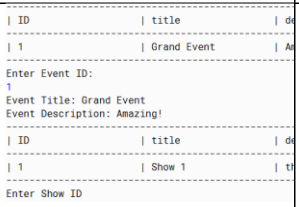
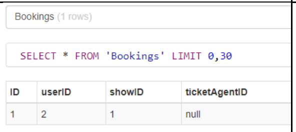
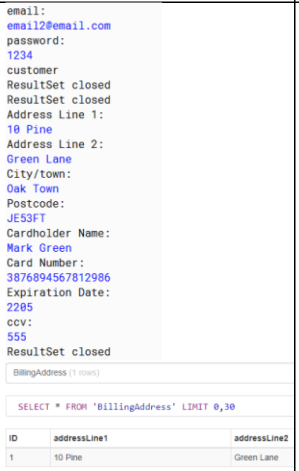
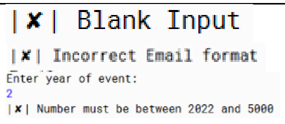
Implemented Code

https://github.com/hsadiq32/CO567-OOSD/tree/main/Code/CO567



Testing Regime

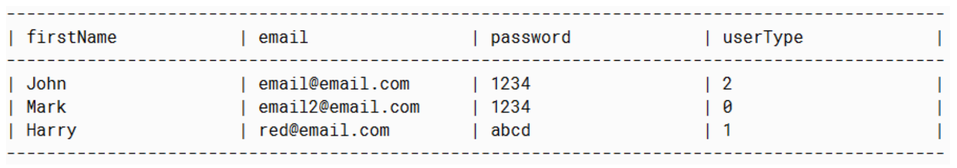
Requirement	Test	Input	Expected outcome	Outcome	Changes Made	Pass
1	Sign up system will not allow new accounts with the same email to avoid conflicts	Stored Email: "email@email.com" Inputted email: "email@email.com"	Email taken error message	 1. Sign up 2. Log in 3. Exit Choose an option from above: 1 First Name: Mark Last Name: Smith Email: email@email.com Password: 1234 Email taken	N/A	✓
2	Verify if email is correct with "@" symbol and a TLD (e.g. .com or .co.uk)	Email: "test" Email: "test@" Email: "test@email" Email: "email.com" Email = "test@email.com"	Reject string without an @ symbol or TLD	 Email: |✗| Blank Input Email: test |✗| Incorrect Email format Email: test@ |✗| Incorrect Email format Email: test@email |✗| Incorrect Email format Email: email.com |✗| Incorrect Email format Email: test@email.com 1. Sign up 2. Log in 3. Exit Choose an option from above:	N/A	✓
3	Customer will be able to sign up on the login system	Stored Email: email2@gmail.com Stored Password : "1234" Email: email2@gmail.com Password : "1234"	Login with customer string, identifying account type	 Email: email2@email.com Password: 1234 ResultSet closed customer	N/A	✓
4	Ticket Agents will be able to sign up via the ticket manager	Firstname: Harry Lastname: Red Email: red@email.com Password: abcd	Create a user account in user table, then reference userID for ticket agent account and create new row of data properly referenced.	 | ID | userID | commiss 1. Add Ticket Agents 2. Edit Ticket Agents 3. Remove Ticket Agents 4. Go Back Choose an option from above: 1 First Name: Harry Last Name: Red email: red@email.com password: abcd ResultSet closed | ID | userID | commiss | 1 | 3 | 1 1. Add Ticket Agents 2. Edit Ticket Agents 3. Remove Ticket Agents 4. Go Back Choose an option from above:	N/A	✓
5	Customer will be able to login successfully with a user type dependant UI	Stored Email: email2@gmail.com Stored Password : "1234"	Login with customer string, identifying account type	 Email: email2@email.com Password: 1234 ResultSet closed customer	Remove user dependant methods from constructor avoiding subclass references - fixed	✓
6	Ticket Agent will be able to login successfully with a user type dependant UI	Stored Email: email2@gmail.com Stored Password : "abcd"	Login with ticket agent string, identifying account type	 email: red@email.com password: abcd ticketagent	Remove user dependant methods from constructor avoiding subclass references - fixed	✓

7	Venue Manager will be able to log in successfully with a user type dependant UI	Stored Email: email@gmail.com Stored Password : “1234”	Login with venue manager string, identifying account type	 email: email@email.com password: 1234 venuemanager ResultSet closed 1. Manage Events/Shows 2. Manage TicketAgents 3. Manage Promotions 4. Exit Choose an option from above:	Remove user dependant methods from constructor avoiding subclass references - fixed	✓
8	Venue Manager will be able to add events	Event title: “Grand Event” Event Description: “Amazing!” Event year: 2022 Event month: 7 Event day: 20	Add event to database for global viewing	 Event title: Grand Event Event desciption: Amazing! Enter year of event: 2022 Enter month of event: 7 Enter day of event: 20 --- | ID | title | descriptio --- | 1 | Grand Event | Amazing! ---	SQL error or missing database (table Events has no column named dateTime) – fixed changed “dateTime” to “date”	✓
9	Venue Manager will be able to add shows corresponding to events	Title: “Show1” Description:”the first show” Max seats: 30 Start hour: 8 Start Hour: 30 End hour: 10 End minutes: 50	Add shows to database for global viewing	 Show title: Show 1 Show desciption: the first show Enter max seats per customer: 30 Enter start time hours: 8 Enter start time minutes: 30 Enter end time hours: 10 Enter end time minutes: 50 Add another show? (y/n)Add another show? (y/n)n --- | ID | eventID | title | desc --- | 1 | 1 | Show 1 | the ---	N/A	✓
10	Customer will be able to view events	Email: email2@email.com Password:1234	Display events created by the venue manager	 email: email2@email.com password: 1234 customer ResultSet closed ResultSet closed --- | ID | title | description --- | 1 | Grand Event | Amazing! --- Enter Event ID:	N/A	✓
11	Customer will be able to view shows corresponding to events	Event ID: 1	Display shows corresponding to the event ID	 --- | ID | title | de --- | 1 | Grand Event | Am --- Enter Event ID: 1 Event Title: Grand Event Event Description: Amazing! --- | ID | title | de --- | 1 | Show 1 | th --- Enter Show ID	N/A	✓
12	Customers will be able to hold tickets	Event ID: 1 Show ID: 1	Tickets are stored but the descriptor bought should be false to indicate the ticket is on hold	 Bookings (1 rows) SELECT * FROM 'Bookings' LIMIT 0,30 ID | userID | showID | ticketAgentID 1 | 2 | 1 | null	N/A	✓
13	Customers will require first time billing signup on login, all other user types do not.	Addressline1: “10 Pine” Addressline2: “Green Lane” City: ”Oak Town” Postcode: “JE53FT” Cardholder Name: “Mark Green” Card Number: “3876894567812986” Expiration Date: 2205 ccv: 555	Customer billing data should be stored and inputted on the first login	 email: email2@email.com password: 1234 customer ResultSet closed ResultSet closed Address Line 1: 10 Pine Address Line 2: Green Lane City/town: Oak Town Postcode: JE53FT Cardholder Name: Mark Green Card Number: 3876894567812986 Expiration Date: 2205 ccv: 555 ResultSet closed BillingAddress (1 rows) SELECT * FROM 'BillingAddress' LIMIT 0,30 ID | addressLine1 | addressLine2 1 | 10 Pine | Green Lane	N/A	✓
14	Input reader will catch invalid data types	Invalid integer, email formats and nulls	Stop null inputs, wrong email formats, and out of bound integer ranges	 |✘| Blank Input |✘| Incorrect Email format Enter year of event: 2 |✘| Number must be between 2022 and 5000	N/A	✓

15

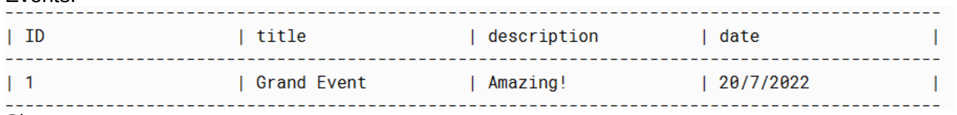
Database manager will create a fully capable database allowing all data to be stored and read

User Accounts:



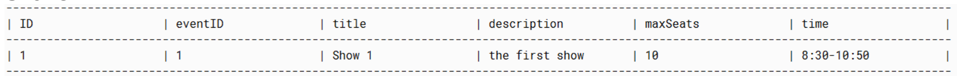
```
----------------------------------------------------------------------------------------------
| firstName             | email                 | password              | userType           |
----------------------------------------------------------------------------------------------
| John                  | email@email.com       | 1234                  | 2                  |
| Mark                  | email2@email.com      | 1234                  | 0                  |
| Harry                 | red@email.com         | abcd                  | 1                  |
----------------------------------------------------------------------------------------------
```

Events:



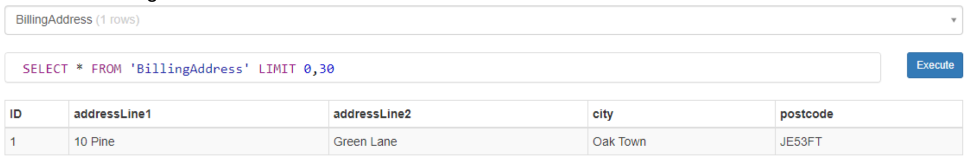
```
----------------------------------------------------------------------------------------------
| ID                    | title                 | description           | date               |
----------------------------------------------------------------------------------------------
| 1                     | Grand Event           | Amazing!              | 20/7/2022          |
----------------------------------------------------------------------------------------------
```

Shows:



```
-----------------------------------------------------------------------------------------------------------------------------
| ID                 | eventID            | title              | description        | maxSeats           | time               |
-----------------------------------------------------------------------------------------------------------------------------
| 1                  | 1                  | Show 1             | the first show     | 10                 | 8:30-10:50         |
-----------------------------------------------------------------------------------------------------------------------------
```

Customer billing details



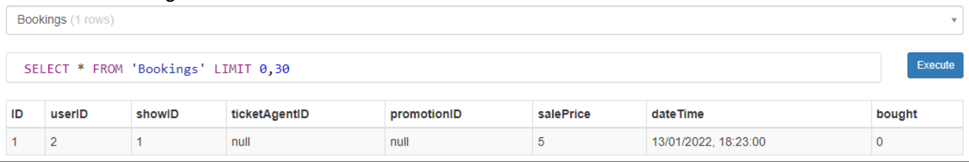
BillingAddress (1 rows)

```
SELECT * FROM 'BillingAddress' LIMIT 0,30
```

Execute

ID	addressLine1	addressLine2	city	postcode
1	10 Pine	Green Lane	Oak Town	JE53FT

Customer booking ticket details



Bookings (1 rows)

```
SELECT * FROM 'Bookings' LIMIT 0,30
```

Execute

ID	userID	showID	ticketAgentID	promotionID	salePrice	dateTime	bought
1	2	1	null	null	5	13/01/2022, 18:23:00	0

✓

Report

In this report I will be acknowledging the process which the team consisting of Haroon Sadiq, Hassan Nisar, and Mauro Nunes took to complete the Implementation Segment of the Assignment. The Implementation is built up of 6 different portions from diagrams, coding and detailed annotations and the team had spent days and nights completing these portions.

The team started off with the Statement of Requirements as we knew if we had a base of requirements and needs of the Customer it would be easier to focus on what the program would need to do, as if we did not have any Customer requirements we would not have any boundaries for the program it could either not fulfil the needs of the Customer or go totally over and beyond what they need. The team got a lot of the fundamentals of the Pseudo - Code from the Use Cases as this entailed what everyone (Customer, Venue Manager and Ticket Agent) would need doing so this let us have around enough requirements to get started as we knew everyone needed to login with a password and username, the Customer needed to buy a ticket and we could simply see from the Use Cases what we needed to have for everyone. The Statement of Requirements was generated in a word document

The Class Diagram was first created as draft deriving from a database model created from SQL query commands using JDBC java which was then converted using Visual Paradigm although this lacked many objects orientated features which could be used in a functional system. Elements of generalisation, inheritance and much more where then implemented using iterative development led to the final design which would then be used in the functional implementation. From its database orientated background the overall design works around using primary and foreign keys for classes to communicate efficiently with each other. A major optimization was of the user accounts where basic details such as first/last name, email and password was stored in a parent class while user type specific content was stored in sub classes encapsulating and inheriting data.

The Sequence Model was complicated for the Team as they did not fully understand what needed to go into the model, but once they had sat down and understood that a sequence model was a model/diagram which showed a sequence/process taking place they knew what they had to do. They first thought about what individual would the sequence be about and at the end they all decided on choosing the Customer purchasing a ticket, they listed down the methods needed to complete the process and the classes linked to each process this made it easier when it came to making the diagram. They opened Visual Paradigm and chose the option to make a sequence model and generated a diagram based on The Customer purchasing a ticket. The team started on the method Selecting an Event, Listing Events, Selecting a Seat, Listing the Promotions, Selecting a Ticket, choosing a Discount and then Paying for the Ticket.

The Pseudo – Code was mainly taken from the Statement of Requirements and as you can see, we kept focusing on the requirements as we knew our program would be based on the requirements we first listed. As a result, after looking at the Requirements we knew what we wanted the code to do but we had to not make code but a translated version and detailed. We first looked at the Customer functions such as Logging in, Holding a Ticket, Purchasing a Ticket and Inputting Details, then the Venue Manager functions and lastly the Ticket Agent, the reason for that order was because we went from the Individual who we thought had the most functions required to the least. They were also labelled as functions to then be referenced in other pseudocode designs. The pseudocode was developed in Microsoft word document.

Having a full database design made implementing a full class design more structured and streamlined. With a full database manager containing SQL logic for creating tables, creating folders and files if they don't exist, inserting data, updating data, printing tables (with full table syntax generator) and much more the programs classes could easily communicate with one another while staying structured and true to nature. Although before any data could even be stored, the input reader was required, which contains an array of functions which catch out: invalid emails, null values, out of bound integer ranges and more ensuring the data entering the program is valid and consistent, putting a stop to syntax errors when methods are communicating with each other. The Venue manager was the most developed class allowing control over events, shows, ticket agents and promotions – with most functions implemented allowing to view, add, delete, and modify key aspects of the database. This being a privileged user, the database needed a usertype system to identify different accounts which was implemented into the login system to actively select different account classes depending on the SQL save data.

The Testing was the last thing to be completed, the reason for this was because we needed fully completed and working code for this to work. We used Black Box testing layout to do the Testing as we started off with the process or task we were testing and a screenshot if the test worked. We got our Black Box testing for the State of Requirements and Pseudo – Code, we needed the State of Requirements as have we fulfilled the actual purpose of the program or not, and the Pseudo – Code was needed as did we get the Code to what we expected it to be. While there were a few shortcomings in the initial development, as noted in the testing, these were ironed out staying true to the target requirements. This was generated in a word document.

The team had many more meetings with the Implementation than the Design this was due to the fact that they had found the Implementation to be much more challenging than the Design and also, they had to balance with the other assignments. The team usually had one-hour meetings for three days a week (Monday, Thursday, and Friday)

STUDENT	TASK	MARKS
HASSAN NISAR	USE CASES	30
HASSAN NISAR	MERGING USE CASES	
MAURO NUNES	DATA DICTIONARY	20
HAROON SADIQ/MAURO NUNES	CLASS DIAGRAM	30
HASSAN NISAR	DESIGN REPORT	20

STUDENT	TASK	MARKS
MAURO NUNES	STATEMENT OF REQUIREMENTS	10
HAROON SADIQ	CLASS DIAGRAM	15
HASSAN NISAR	SEQUENCE MODEL	15
HAROON SADIQ	PSEUDO – CODE	20
HAROON SADIQ	IMPLEMENTED CODE	15
HARRON SADIQ	TESTING	10
HASSAN NISAR/HAROON SADIQ	IMPLEMENTATION REPORT	15