CO567

[Document subtitle]

2022

Contents

[Section A - Design 2](#_Toc92731416)

[Use Cases and Descriptions 2](#_Toc92731417)

[Class Diagram 5](#_Toc92731418)

[Data Dictionary 6](#_Toc92731419)

[Report 9](#_Toc92731420)

[Section B - Implementation 11](#_Toc92731421)

[Implementation 11](#_Toc92731422)

[Statement of Requirements 11](#_Toc92731423)

[Class Diagram done Sequence Model Pseudo – Code Detailed Designs 11](#_Toc92731424)

[A) Login System: Sign up 11](#_Toc92731425)

[B) Login System: Login 12](#_Toc92731426)

[C) Login System: Account Selector 13](#_Toc92731427)

[D) Customer: Holding a ticket 14](#_Toc92731428)

[Implemented Code 15](#_Toc92731429)

[Testing Regime 15](#_Toc92731430)

[Report 15](#_Toc92731431)

# Section A - Design

Use Cases and Descriptions  
Diagram

Description automatically generated  
Consumer -The Consumer must be able to access the Online Ticket System buy a ticket for an event   
  
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 events present or future given a limited date range

Provide Details - To get into the ticket system the Consumer must Provide their Details such as an unique username and password that will authenticate that it is the specific customer who wishes to access the system. The Consumer 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.  
  
Select Show - In Order to purchase a ticket, the user must select an Event they wish to see. 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.  
  
Choose Seat - After choosing an event the consumer must choose a seat or seats but if they choose 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. 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.

Pay for Ticket- After the Seat or Seats have been chosen the Consumer must pay, the only form of payment is credit card.  
  
Pay for Multiple Tickets - When the Consumer chooses to pay for a multiple tickets the process is providing the credit card details and the billing address in addition the consumer could have a choice of choosing a promotion set by the Ticket Agent as if you buy certain amount of tickets you can get a certain percentage discount.

Diagram

Description automatically generated  
Venue Manager- The Venue Manager manages everything about the location of the event such as the tickets, prices, seats, and layout of the event  
  
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

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  
  
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 Consumer.  
  
Reschedule Event - A Venue Manager can Reschedule a 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  
  
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 world - wide 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 - There must be a price range for the seats this includes different prices for an adult, student, child, and senior citizen. Sometimes the seats could have promotions on such as if you buy 10 seats at once you will receive 10% off your total

Diagram

Description automatically generated  
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 Consumers 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 consumer  
  
Confirm Ticket Purchase - After Providing the Consumer Information the Ticket Agent will need to confirm the purchase of the seat/ticket that the Consumer has chosen, this will provide on the system that the Consumer has bought a ticket and has been confirmed/validated by a Ticket Agent

## Class Diagram

## Data Dictionary

Make tables on all classes attributes linked with class diagrams

Graphical user interface, text, application

Description automatically generated

|  |  |  |
| --- | --- | --- |
| 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.

In 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

## Implementation

## Statement of Requirements

Customer Will Be Able To Login Or Sign Up

Class Diagram  
done  
  
Sequence Model

Pseudo – Code Detailed Designs  
A) Login System: Sign up

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

function signup()

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*, usertype (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) Customer: Holding a ticket

String variables: *date time, event ID, Show ID, Ticket ID, Booking ID*

function holdTicket()

1. Select an event

Loop{

1i. Display current events table – execute SQL database query and display

2ii. Userinput choose *event ID* or filter table

2iii. ***if:*** user chooses *event ID*:

***else:***

- add error condition to *error message* string

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 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*, userType (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

Implemented Code  
not done

Testing Regime

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Objective** | **Aim of test** | **Input** | **Expected outcome** | **Outcome** | **Changes Made** |
| 1 | A1 | Get userinput for first name, last name, email, password | “Bob”, “Smith”, “bob@email.com”, “1234abcd” | firstName = “Bob”  lastName = “Smith”  email = “bob@email.com”  password = “1234abcd” | firstName = “Bob”  lastName = “Smith”  email = “bob@email.com”  password = “1234abcd” | N/A |
| 2 | A2i | Verify userinput for first/last name should be: not null, letters only |  |  |  |  |

Report

Not done