2022

Muhie Al Haimus

Contents

[Investigation 2](#_Toc96940498)

[Desk based research 2](#_Toc96940499)

[Stakeholders 2](#_Toc96940500)

[Limitations of current system 2](#_Toc96940501)

[Design 3](#_Toc96940502)

[Sub programs 3](#_Toc96940503)

[Task List 3](#_Toc96940504)

[Algorithms 3](#_Toc96940505)

[User Interface 3](#_Toc96940506)

[UML 3](#_Toc96940507)

[Data Dictionary 3](#_Toc96940508)

[Software Development 5](#_Toc96940509)

[Developmental Testing 5](#_Toc96940510)

[Testing 0](#_Toc96940511)

[Evaluation 0](#_Toc96940512)

## Investigation

### Desk based research

Use this to look online for similar solutions, you should print screen your findings identifying any features you like and will be trying to incorporate into your solution as well as features you think don’t work.

### Stakeholders

What are the [stakeholders](https://www.bbc.co.uk/bitesize/guides/z4gcd2p/revision/1#:~:text=Stakeholders are individuals%2C groups or,They include%3A&text=Workers who want to earn,quality products at reasonable prices.) and what are their requirements for the solution, these should be as diverse as possible and will have different requirements.

### Limitations of current system

The current system is paper based with the performing arts teachers taking notes about any tickets sold.

Cineworld booking system



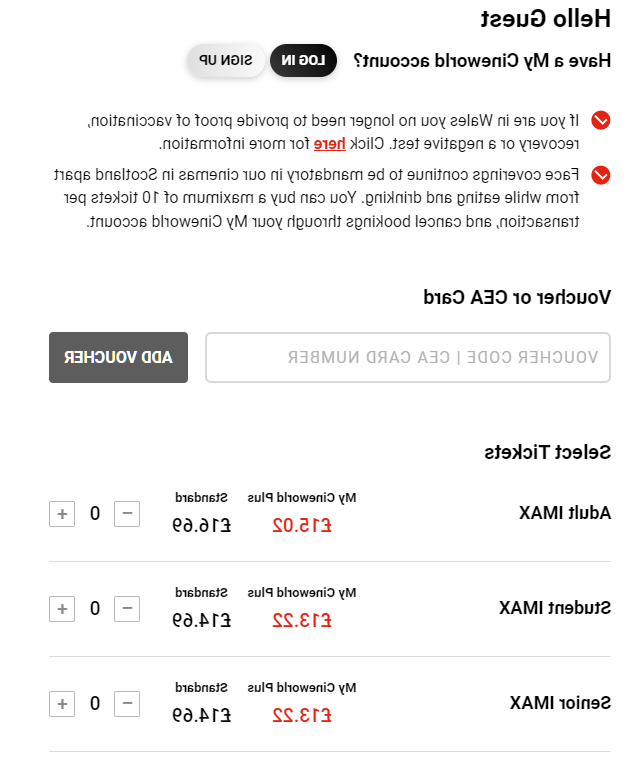


Figure 1Cineworld online booking

Figure 2Cineworld Ashton

You can select and see already booked seats. Check price bands and see where the screen is. However, the buttons for selecting seats are small and clunky. It would be hard to implement an image with clickable seats in PYQT5. On the Cineworld website you can add voucher codes, which I can use to allow for VIP’S to book for free. There are also different price tiers where Adults, students and seniors can book. With Cineworld you can only book a maximum of 10 tickets, Something I will add to my program. Although you must use your email to sign in to book any seats, something I do not like as you cannot see the available seats before signing in. with this booking system you cannot block seats. Which is something that I will allow for with my system, furthermore the customer is in control of the booking system. Which again, I will not allow for. You must be authorised to use my booking system. Also, the Cineworld website is not very optimised for mobile devices which is not good for modern times. Lastly, I like how you can pick any seat and not have to be sitting next to someone if you don’t want to.

British airways Booking system

A picture containing graphical user interface

Description automatically generated

British Airways Search Bar 1

[[1]](#footnote-1)

I really like how you can search for flights with the search bar, instead of flights I’d do films that are available to watch and what date there are. I’m going to add a search bar. I also like how the search bar is minimalistic. Furthermore, I like how it auto predicts where, you are going to go. I Don’t think I’d be able to add this to my system, but if I could it would be very intuitive for the user. Also, I like how you can choose all the passenger types in a drop-down menu, they allow you to book child seating, young adults, children, and infants. I like the amount of choice in how many different options there are but, I’d like to have OAP as and option and VIP/Staff as an option. Overall, I think that the British airways searching system is great design which if I could emulate would create a good user experience.

Everyman Cinema Booking System.

Graphical user interface, text

Description automatically generatedGraphical user interface, website

Description automatically generated

Everyman Cinema 1

I really like how you can see all the times for a specific performance and tells you a small description of the film before you buy your tickets. However, when choosing seats you have a time limit, and the user interface is a lot worse than Cineworld. Although the website is mobile friendly which is very practical today, as most if not everyone some sort of mobile device. It is also very convenient for the user as they can book a ticket online anywhere. Due to the limitations of PYQT5 this is not possible. If I use some sort of web framework or mobile app framework like flask for python or flask using dart. I could achieve this however due to the time constraints for this project I think I’ll stick with PYQT5. I do like with everyman you can book OAP and family tickets which I’d like to include in my project.

Research conclusion

For my project I’d like to take a design and feature queue from all the booking systems I referenced prior. From Cineworld I’d like to take the seating design and overall ease of use from the British airways design and from everyman I’d like to take the amount of ticket types available to the customer.

Why would the stakeholders need a computer-based system?

The stakeholders are a college, who need to keep track of the seats booked in an auditorium. There staff may not be able to keep track of all the seats being booked, they may make mistakes when booking the seats. This could cause the college to lose revenue. A computer-based system can eliminate some booking errors, leading the college increasing profits and improving customer satisfaction.

Stakeholders – Objectives

The system must allow for tickets being booked. For each ticket sold or allocated, it must record the name of the customer, phone number, price paid and performance date.

The system must prevent a seat, for a particular performance, being sold or allocated more than once.

The system must allow seats to be blocked so that they cannot be purchased or otherwise allocated.

The system must allow a staff member to search for a customer and display which ticket(s) they have booked for which performance(s)

The system must display a list of ticket holders (seat number, name, phone number), sorted by surname, for a selected performance

The system must save all relevant data to an external file (so that the system can be closed at the end of each day and used again the following day)

The system must produce management information for each performance

o Display how many tickets have been sold (or otherwise allocated)

o Display how many tickets remain available for sale

o Display the total revenue for each performance

My objectives:

All the objectives from the stakeholders and:

Creating an interactive seating view, with the ability to be able to click on the window and book multiple seats.

Creating a printable pdf that can be emailed to the customer and used for entry into the theatre

Creating a smart pricing algorithm that takes in the number of days and available seats into account to change pricing.

Limitations of current system.

The paper-based system can easily be misplaced, destroyed and details can be forgotten to be added. A computer-based solution cannot be misplaced as it is stored on a central server. data must be valid to be entered into the database, data can also not be duplicated as a relational database prevents this from happening. Computer-based solutions also allow for backing up however the user may choose for example the grandfather, father, son. This also prevents unauthorised users form looking at the details as only authorised users with the correct permission can use the computer-based system compared the paper-based counterpart. Computer based systems also prevents the wastage of space by not allowing duplicate bookings. The paper-based solution also would use up a lot of paper compared to a computer based one. However, a computer-based system may cost more to run. Another advantage of a computer-based system is that it can comply laws and regulations more easily compared to a paper-based counterpart.

## Design

### Sub programs

Since the college needs to maximise profits, a smart pricing algorithm might be required to sell more tickets.

-This would work by getting the current date and time form the operating system and subtracting that from the date of the performance.

-Then the algorithm would get the number of available seats form the SQL server.

-If the number of seats sold and or the date of the performance is coming soon, the prices will be adjusted accordingly.

Since the college wants a simple and easy to use user interface a seating layout is required.

-This would work by storing the seating layout in an SQL database, then adjusting it accordingly to the number of seats being booked, blocked, or otherwise purchased.

-The program would have to get the inputs of the user, when clicking the seats available and not allowing for already allocated seats to be clicked.

-These inputs are then sent to the database, adjusting information, where necessary

The college also wants staff to be able to search for customers in a database.

-This would work by getting all the inputs of the customer details and selecting the corresponding customer.

-This information would then be displayed in the user interface.

The college need to be able to calculate costs of the performance to be able to easily add it to there revenue splits.

-This work by asking the user how many adults, children or OAP’s buying a ticket.

-It would then calculate the costs for the user to pay.

Colouring the seats according to whether they have been booked or otherwise allocated (full) or empty where a seat has not been booked.

-figuring out where the seats are occupied then colouring them in red for FULL

-figuring out where seats are empty and and colouring them green for EMPTY

Inserting values into the SQL database to be able to be stored when the system is shut down.

-The SQL statement need to be properly formatted to insert values into the systems database.

-All inputs need to be in the right format to not produce an error when inserting values into the sol database.

-All invalid data will be notified to the user, prompting them to check the inputs.

-The final SQL statement will then be committed to the database.

### Task List

For each task (sub program) you need to identify the inputs, outputs and processing

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Input | Output | Processing |
| Smart Pricing algorithm. | Current date, the date of the performance | The calculated price, if it has more than 100 seats available and it less than five days remaining till the performance date. | -finding the current date, subtracting the current date from the date of the performance, finding how many seats remain. |
| Calculating the price of the performance. | Inputting the number of adults, children or OAP’s being booked into a certain performance | Outputting the total price to pay in pounds | Finding the number of adults, children, or OAP’s to be booked then multiplying the number of ticket holders by the price. |
| Searching for customers inside the program | Inputting customer details (Seats Numbers, Phone number or name.) | Output the credentials of the customer, with the price paid, name, seats selected and phone number | Using the right SQL statement for whatever details the staff member has inputted, then committing the SQL statement to the database, then returning the data to the user in the table widgets. |
| Quick sort algorithm to sort the top payees. When the top payees are > 10 stored in the database | Takes the input of all payees for a given performance then sorts them | Prints out the top to the command line payees as an array. | Pivot with recursion, is very fast and efficient has the big O notation of O(log n) (very fast) |
| Bubble sort Algorithm to sort the top payees when the number of payees are less than 10 people stored in the database | Takes the input of all payees for a given performance then sorts them | Prints out the top to the command line payees as an array. | Not very fast but is efficient with a small amount of people inside the database with a complexity of  O(n2) so it is the worst case scenario for sorting data |

### 

### Algorithms

Each subtask should have its own algorithm written in pseudocode, these should be clear enough for a third party (fellow students) to follow, consider if someone in the class could make your system from these designs.

These should be lots of separate algorithms, I don’t need to see any algorithms for the user interface as these are designed above.

Procedure Quick sort algorithm

/\* low  --> Starting index,  high  --> Ending index \*/

quickSort(arr[], low, high)

{

    if (low < high)

    {

        /\* pi is partitioning index, arr[p] is now

           at right place \*/

        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi

        quickSort(arr, pi + 1, high); // After pi

    }

}

// Procedure Bubble sort algorithm

void bubbleSort(int arr[], int n)

{

    int i, j;

    for (i = 0; i < n - 1; i++)

        // Last i elements are already in place

        for (j = 0; j < n - i - 1; j++)

            if (arr[j] > arr[j + 1])

                swap(arr[j], arr[j + 1]);

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

    int i;

    for (i = 0; i < size; i++)

        cout << arr[i] << " ";

    cout << endl;

}

// Driver code

int main()

{

    int arr[] = { 64, 34, 25, 12, 22, 11, 90 };

    int n = sizeof(arr) / sizeof(arr[0]);

    bubbleSort(arr, n);

    cout << "The sorted array is: \n";

    printArray(arr, n);

    return 0;

}

Using datetime import date

// Function smart pricing algorithm

current\_Date = date date.today

Performance\_Date = date date.getText(datewidget)

//Finding the difference between two dates

difference\_in\_Dates = Performance\_Date - current\_Date

//Finding the number of avalible seats by finding the amount of

if column is equal to "FULL":

    total\_FullSeats += 1

    seats\_Avalible = str(200-self.totalfullseats)

else:

    return

endif

//Calcuating the end price

if difference\_in\_Dates less than 10 and totalfullseats is greater than 100:

    set\_Price\_Multiplyer = 0.75

elseif difference\_in\_Dates less than 5 and totalfullseats is greater than 100:

    set\_Price\_Multiplyer = 0.5

elseif difference\_in\_Dates less than 3 and totalfullseats is greater than 100:

    set\_Price\_Multiplyer = 0.25

else:

    set\_Price\_Multiplyer = 1

endif

endproceedure

//Procedure Searching for customers

Searching\_staement = str "Select \* form table name"

function Searching:

    statementSQL = Searching\_statment

    try:

        cs = (

            "Driver={SQL Server};"

            "Server=RX-ORIGIN\MSSQLSERVER01;"

            "Database=Muhie;"

            "Trusted\_Connection=yes;"

            "UID=RX-ORIGIN\16mal;"

        )

        cnxn = pyodbc.connect(cs)

    endtry

        if cs is not None:

            cursor = cnxn.cursor()

            cursor.execute(statementSQL)

            records = cursor.fetchall()

            return cs

        endfor

endprocedure

### User Interface

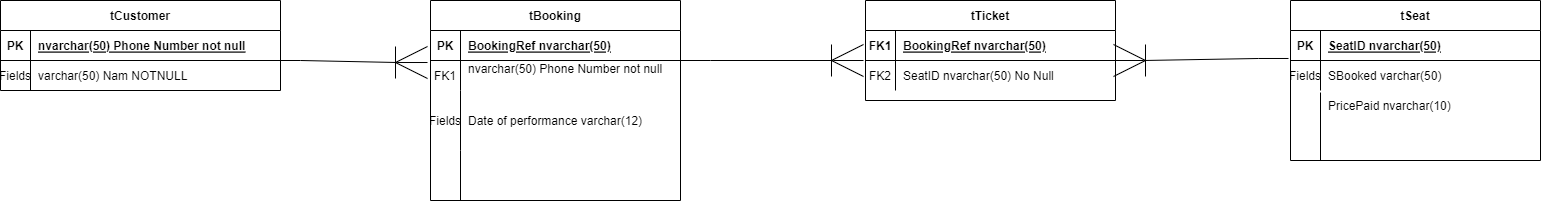
Draw annotated designs of your user interface. These should be detailed enough that another person could make your system and it will look as you envision.

A picture containing graphical user interface

Description automatically generated

### UML

My ERD for the theatre project, please note that the seats seen in the user interface are form a separate table containing the date and all the seats that are full/empty.



### Data Dictionary

Fill in the table for each of your entities.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attribute/Field | Data type | Validation | Key | Comments |
| Phone Number | nvarchar(12)(50) is a mistake in the ERD | Range check, presence check, format check must start with 07. | Primary Key in the tCustomer Table/Foreign key in the tBooking Table | The Phone number of the customer. |
| Name | Varchar (50) | Presence check. | N/A | Name of the customer. |
| Booking Reference (BookingREF) | Varchar(62) | Range check must be less than 62 characters. Format Check as the booking reference is the name of the customer and the date of the performance. For example “(02-02-2022)”JohnSmith.  Also a presence check to check if the field is empty. | Primary key in the tBooking table/Foreign key in the tTicket (linking table) table. | Booking reference of the performance selected. |
| DateOfPerformance | Date | Presence check to check if field is empty, must be formatted correctly for SQL. | N/A | Date of selected performance |
| SeatID | Nvarchar(22) | Must be formatted correctly for example it must have the date then the phone number of the customer for example “(02-02-2022)” 7911 123456. Also a range check to check that the seatID doesn’t exceed 22 characters | Primary key in tSeats/ Foreign key in tTicket. | The SeatID of a selected performance sold. |
| Price Paid | Nvarchar(10) | Must not exceed the character limit of 10. | N/A | Price paid of the seats that they booked. |
| SBooked | Varchar(120) | Must not exceed the character limit of 120. The seats that have been booked are stored as a list of items i.e. A1,A2,E5,ect. | N/A | The seats that have been booked on a specific performance date. |

## Software Development

All your code should be annotated with suitable comments and doc strings, use <http://hilite.me/> to format your code as this retains the colour coding and lets you copy and paste the code in the correct format.

import random

from array import array

from datetime import date

import datetime

from datetime import datetime

from traceback import print\_exc

import pyodbc

import time

from PyQt5.QtWidgets import QDialog, QApplication, QTableWidgetItem

from BookingSytemV10 import \*

from PyQt5.QtGui import QPixmap

from PyQt5.QtCore import Qt

from PyQt5 import QtCore, QtGui, QtWidgets

class Theatre(QDialog):

    def \_\_init\_\_(self): #initialiatising all values that need to be set at the start of the program.

        super(QDialog, self).\_\_init\_\_()

        self.ui = Ui\_Cinema()

        self.ui.setupUi(self)

        self.statementsearch = ""

        self.ui.Buy\_PB.clicked.connect(self.GetInputs)

        self.show()

        self.ui.SeatE\_PB\_2.clicked.connect(self.choosefull)

        self.ui.SeatE\_PB.clicked.connect(self.chooseEmpty)

        self.ui.Seat\_PB.clicked.connect(self.chooseALL)

        self.ui.NewShow\_pb.clicked.connect(self.NewPerformances)

        self.ui.Customer\_PB.clicked.connect(self.FindCust)

        self.ui.Reset.clicked.connect(self.Reset)

        today = date.today()

        nowy = datetime.now().year

        nowm = now = datetime.now().month

        nowd = now = now = datetime.now().day # not allowing a date more than a year after the current date to prevent random tables being created like in 2040.

        self.ui.PERFORMANCE\_LE.setMinimumDate(QtCore.QDate(today))

        self.ui.PERFORMANCE\_LE.setMaximumDate(QtCore.QDate(nowy+1, nowm, nowd))

        self.ui.ADULT\_SB.valueChanged.connect(self.SBvaluechange)

        self.ui.CHILD\_SB.valueChanged.connect(self.SBvaluechange)

        self.ui.OAP\_SB.valueChanged.connect(self.SBvaluechange)

        self.ui.pushButton.clicked.connect(self.SearchCustData)

        self.totalprice = 0

        self.BuySeatSQL = ""

        self.StandardDate = '""'

        self.ui.Table\_Table.setEditTriggers(QtWidgets.QTableWidget.NoEditTriggers)

        self.ui.Table\_Table.cellClicked.connect(self.cell\_was\_clicked)

        self.total = []

        self.totalFormatSQL = []

        self.runingcount = 0

        self.SeatID = ''

        self.BookingRef = ''

        self.changeheaders = 0

        self.totalfullseats = 0

        self.adult = 10

        self.childprice = 5

        self.oap\_Price = 0

        self.value = 0

        self.length = 0

        self.setrowcount = 0

    def Reset(self):

        self.ui.CustType\_LE.setText("")

        self.totalFormatSQL.clear()

        self.total.clear()

        self.runingcount = 0

    def Seats\_Selected(self):

        self.ui.CustType\_LE.setText(self.selected\_Seat)

    def cell\_was\_clicked(self):

        current\_row = self.ui.Table\_Table.currentRow()

        current\_column = self.ui.Table\_Table.currentColumn()

        try:

            cell\_value = self.ui.Table\_Table.item(current\_row, current\_column).text()

            if cell\_value == "EMPTY":

                if self.ui.BlockSeats.isChecked() == True:

                    #Allowing for blocking of seats

                    self.runingcount = 0

                    self.totalseats = 1

                    self.seatRow = current\_row

                    self.seatColumn = current\_column

                    self.Seats\_Selected()

                else:

                    pass

                self.seatRow = current\_row

                self.seatColumn = current\_column

                self.Seats\_Selected()

            else:

                print("cell is not EMPTY, the seat must be empty to be added.")

        except:

            print("cell is not EMPTY, the seat must be empty to be added.")

    def Seats\_Selected(self):

        self.totalseats = self.ui.ADULT\_SB.value() + self.ui.CHILD\_SB.value() + self.ui.OAP\_SB.value() + self.ui.VIP\_SB.value()

        if self.ui.BlockSeats.isChecked() == True:

            self.totalseats = 1

        else:

            pass

        if self.runingcount >= self.totalseats: #not allowing more seats than the total number of seats selected to be booked

            print("ignor if blocked seats is checked, else please check that you have a enough seats booked.")

            return

        else:

            pass

        self.seatColumn += 1

        self.seatRow += 1

        if not self.total:

            pass

        else:

            self.total.append(",")

        if self.seatColumn == 1: #finding the seat selected, appending to the list on the gui wiget and not allowing for full seats to be booked

            self.seatColumn = "A"

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        elif self.seatColumn == 2:#finding the seat selected, appending to the list on the gui wiget and not allowing for full seats to be booked

            self.seatColumn = "B"

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        elif self.seatColumn == 3:

            self.seatColumn = "C"#finding the seat selected, appending to the list on the gui wiget and not allowing for full seats to be booked

            seat = self.seatRow,self.seatColumn

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        elif self.seatColumn == 4:#finding the seat selected, appending to the list on the gui wiget and not allowing for full seats to be booked

            self.seatColumn = "D"

            seat = self.seatRow,self.seatColumn

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        elif self.seatColumn == 5:#finding the seat selected, appending to the list on the gui wiget and not allowing for full seats to be booked

            self.seatColumn = "E"

            seat = self.seatRow,self.seatColumn

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            print(self.total)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        elif self.seatColumn == 6:#finding the seat selected, appending to the list on the gui wiget and not allowing for full seats to be booked

            self.seatColumn = "F"

            seat = self.seatRow,self.seatColumn

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            print(self.total)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        elif self.seatColumn == 7:#finding the seat selected, appending to the list on the gui wiget and not allowing for full seats to be booked

            self.seatColumn = "G"

            seat = self.seatRow,self.seatColumn

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            print(self.total)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        elif self.seatColumn == 8:#finding the seat selected, appending to the list on the gui wiget and not allowing for full seats to be booked

            self.seatColumn = "H"

            seat = self.seatRow,self.seatColumn

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            print(self.total)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        elif self.seatColumn == 9:#finding the seat selected, appending to the list on the gui wiget and not allowing for full seats to be booked

            self.seatColumn = "I"

            seat = self.seatRow,self.seatColumn

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            print(self.total)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        elif self.seatColumn == 10:

            self.seatColumn = "J"

            seat = self.seatRow,self.seatColumn

            self.total.append(self.seatColumn)

            self.total.append(self.seatRow)

            print(self.total)

            makeitastring = ''.join(map(str, self.total))

            self.ui.CustType\_LE.setText(makeitastring)

            self.runingcount += 1

            self.totalFormatSQL.append(self.seatColumn)

            self.totalFormatSQL.append(self.seatRow)

            self.Seat\_Row = self.totalFormatSQL[1::2]

            self.Seat\_Col = self.totalFormatSQL[0::2]

        else:

            print("If you would like to add more seats please, add more ticket holders.")

            pass

    def GetInputs(self):

        if self.ui.BlockSeats.isChecked() == True:

            self.length = len(self.Seat\_Row)

            self.MakeSeatsFull()

        else:

            pass

        Name = self.ui.NAME\_LE\_2.text() #not allowing any empty inputs!

        if Name == "" or Name =="full" or Name == "empty" or Name =="FULL" or Name == "EMPTY":

            print("Name is empty, please add a value to this field")

            return

        else:

            self.NameFinal = Name

            print(self.NameFinal)

        Phone = self.ui.PHONE\_LE.text()

        if Phone == "07" or "": #not allowing any empty inputs!

            print("Phone is empty, please add a value to this field")

            return

        else:

            self.PhoneFinal = Phone

            print(self.PhoneFinal)

        RawDate = self.ui.PERFORMANCE\_LE.date()

        self.StandardDate = RawDate.toPyDate()

        print(self.StandardDate)

        Row = self.ui.CustType\_LE.text()

        #self.Col = self.ui.Col\_SB.value()

        self.length = len(self.Seat\_Row)

        if self.length != self.totalseats:

                print("please enter the correct amount of seats, for the number of people booked.")

        else:

            self.MakeSeatsFull()

    def MakeSeatsFull(self):

        print(self.length)

        for i in range(0,self.length):

            print(i)

            RawDate = self.ui.PERFORMANCE\_LE.date()

            self.StandardDate = RawDate.toPyDate()

            date = self.StandardDate

            self.dateformatted = (f'"{date}"')

            frow = self.Seat\_Row[i]

            fcol = self.Seat\_Col[i]

            print(frow)

            print(fcol)

            self.commitSQl = "UPDATE {} SET {} = 'FULL' WHERE Seatrow = {}".format(self.dateformatted,fcol,frow)

            print(self.commitSQl)

            #self.ui.BlockSeats.setChecked(False)

            self.Commiting()

            print("successfully added customers to the database.")

            i+=1

        if self.ui.BlockSeats.isChecked() == True:

            pass

            self.ui.CustType\_LE.setText("")

            self.runingcount = 0

            self.totalFormatSQL.clear()

            self.total.clear()

            self.chooseALL()

        else:

            print("inserting customer details to the databse")

            self.insert\_customers()

            self.ui.CustType\_LE.setText("")

            self.runingcount = 0

            self.totalFormatSQL.clear()

            self.total.clear()

    def SBvaluechange(self): #changing the running count of the spin box, with the smart pricing algorithm attached.

        now = date.today()

        RawDate = self.ui.PERFORMANCE\_LE.date()

        self.StandardDate = RawDate.toPyDate()

        date\_Difference = self.StandardDate - now

        if date\_Difference.days < 5:

            self.adult = 5

            self.childprice = 2.0

            self.oap\_Price = 2.5

        else:

            self.adult = 10

            self.childprice = 5

            self.oap\_Price = 5

        self.totalprice = self.ui.ADULT\_SB.value()\*self.adult + self.ui.CHILD\_SB.value()\*self.childprice + self.ui.OAP\_SB.value()\*self.oap\_Price

        self.ui.PAY\_LE.setText(str(self.totalprice))

    def chooseALL(self):

        self.ui.Table\_Table.setRowCount(20)

        self.ui.CustType\_LE.setText("")

        self.totalFormatSQL.clear()

        self.total.clear()

        self.runingcount = 0

        self.totalfullseats = 0

        self.ui.CustType\_LE.setText("")

        RawDate = self.ui.PERFORMANCE\_LE.date()

        self.StandardDate = RawDate.toPyDate()

        date = self.StandardDate

        self.dateformatted = (f'"{date}"')

        self.FindSQL = "Select A,B,C,D,E,F,G,H,I,J from {}".format(self.dateformatted)

        self.conformation = "ALL" #selecting all seats form the date selected!

        self.changeheaders = 0

        self.totalfullseats = 0

        self.Show\_Search\_Data()

        self.Searching()

        self.value = 1

        self.FindSQL = "SELECT tCustomer.PhoneNumber, tCustomer.Name, tBooking.BookingRef, tBooking.date, tSeat.SBooked, tSeat.PricePaid FROM tCustomer INNER JOIN tBooking ON  tCustomer.PhoneNumber = tBooking.PhoneNumber INNER JOIN tTicket on tBooking.BookingRef = tTicket.BookingRef INNER JOIN tSeat on tSeat.SeatID = tTicket.SeatID WHERE tBooking.date = '{}'".format(self.dateformatted)

        self.Searching()

        self.value = 0

        self.show()

    def choosefull(self):

        self.ui.Table\_Table.setRowCount(20)

        self.ui.CustType\_LE.setText("")

        RawDate = self.ui.PERFORMANCE\_LE.date()

        self.StandardDate = RawDate.toPyDate()

        date = self.StandardDate

        self.dateformatted = (f'"{date}"')

        self.FindSQL = "Select A,B,C,D,E,F,G,H,I,J from {}".format(self.dateformatted)

        self.conformation = "FULL" # only choosing full seats for the pb full!

        self.changeheaders = 0

        self.Show\_Search\_Data()

        self.Searching()

        self.show()

    def chooseEmpty(self):

        self.ui.Table\_Table.setRowCount(20)

        print(self.ui.SearchSeats.text())

        self.totalfullseats = 0

        self.ui.CustType\_LE.setText("")

        RawDate = self.ui.PERFORMANCE\_LE.date()

        self.StandardDate = RawDate.toPyDate()

        date = self.StandardDate

        self.dateformatted = (f'"{date}"')

        self.FindSQL = "Select A,B,C,D,E,F,G,H,I,J from {}".format(self.dateformatted)

        self.conformation = "EMPTY" #choosing only empty seats for the pb empty seats!

        self.changeheaders = 0

        self.Show\_Search\_Data()

        self.Searching()

        self.Searching()

    def NewPerformances(self):

        RawDate = self.ui.PERFORMANCE\_LE.date()

        self.StandardDate = RawDate.toPyDate()

        print(self.StandardDate)

        date = self.StandardDate

        self.dateformatted = (f'"{date}"')

        self.customerTable = "tCust" + self.dateformatted

        self.BookingTable = "tBooking" + self.dateformatted

        self.TicketTable = "tTicket" + self.dateformatted

        self.SeatTable = "tSeats" + self.dateformatted

        print(self.SeatTable)

        self.commitSQl = "CREATE TABLE {}(Seatrow int PRIMARY KEY NOT NULL, DATEOP DATE NOT NULL, A VARCHAR(8) NOT NULL, B VARCHAR(8) NOT NULL, C VARCHAR(8) NOT NULL, D VARCHAR(8) NOT NULL, E VARCHAR(8) NOT NULL, F VARCHAR(8) NOT NULL, G VARCHAR(8) NOT NULL, H VARCHAR(8) NOT NULL, I VARCHAR(8) NOT NULL, J VARCHAR(8) NOT NULL)".format(self.dateformatted)

        self.Commiting() #call a function to create a table!

        self.PopulateTable()

    def PopulateTable(self):

        date = self.StandardDate

        for i in range(1,21):

            self.commitSQl = "insert into ""{}"" values('{}','{}','EMPTY','EMPTY','EMPTY','EMPTY','EMPTY','EMPTY','EMPTY','EMPTY','EMPTY','EMPTY')".format(self.dateformatted,i,date)

            self.Commiting() #call a function to insert the rows into the table!

    def insert\_customers(self):

        self.SeatID = self.dateformatted + self.NameFinal

        self.BookingRef = self.dateformatted + self.PhoneFinal

        self.formmatted\_Seats = self.ui.CustType\_LE.text()

        self.commitSQl = "insert into tCustomer values('{}','{}') insert into tBooking values('{}','{}','{}') insert into tTicket values('{}','{}') insert into tSeat Values('{}','{}','{}')".format(self.PhoneFinal,self.NameFinal,self.BookingRef,self.dateformatted,self.PhoneFinal,self.BookingRef,self.SeatID,self.SeatID,self.formmatted\_Seats,self.totalprice)

        print(self.commitSQl)

        self.Commiting()

    def SearchCustData(self):

        if self.ui.NAME\_LE\_2.text() != "":

             self.FindSQL = "SELECT tCustomer.PhoneNumber, tCustomer.Name, tBooking.BookingRef, tBooking.date, tSeat.SBooked, tSeat.PricePaid FROM tCustomer INNER JOIN tBooking ON  tCustomer.PhoneNumber = tBooking.PhoneNumber INNER JOIN tTicket on tBooking.BookingRef = tTicket.BookingRef INNER JOIN tSeat on tSeat.SeatID = tTicket.SeatID WHERE tCustomer.Name = '{}'".format(self.ui.NAME\_LE\_2.text())

             self.changeheaders = 1

             self.conformation = ""

             self.Searching()

             self.Show\_Search\_Data()

        elif self.ui.SearchSeats.text() != "":

             temp = self.ui.SearchSeats.text()

             self.FindSQL = "SELECT tCustomer.PhoneNumber, tCustomer.Name, tBooking.BookingRef, tBooking.date, tSeat.SBooked, tSeat.PricePaid FROM tCustomer INNER JOIN tBooking ON  tCustomer.PhoneNumber = tBooking.PhoneNumber INNER JOIN tTicket on tBooking.BookingRef = tTicket.BookingRef INNER JOIN tSeat on tSeat.SeatID = tTicket.SeatID WHERE tSeat.SBooked like '%{}%'".format(temp)

             self.changeheaders = 1

             self.conformation = ""

             self.Searching()

             self.Show\_Search\_Data()

        elif self.ui.PHONE\_LE.text() != "":

             self.FindSQL = "SELECT tCustomer.PhoneNumber, tCustomer.Name, tBooking.BookingRef, tBooking.date, tSeat.SBooked, tSeat.PricePaid FROM tCustomer INNER JOIN tBooking ON  tCustomer.PhoneNumber = tBooking.PhoneNumber INNER JOIN tTicket on tBooking.BookingRef = tTicket.BookingRef INNER JOIN tSeat on tSeat.SeatID = tTicket.SeatID WHERE tCustomer.PhoneNumber = '{}'".format(self.ui.PHONE\_LE.text())

             self.changeheaders = 1

             self.conformation = ""

             self.Searching()

             self.Show\_Search\_Data()

    def FindCust(self):

        RawDate = self.ui.PERFORMANCE\_LE.date()

        self.StandardDate = RawDate.toPyDate()

        date = self.StandardDate

        self.dateformatted = (f'"{date}"')

        print(self.dateformatted)

        self.value = 1

        self.setrowcount = 1

        self.FindSQL = "SELECT tCustomer.PhoneNumber, tCustomer.Name, tBooking.BookingRef, tBooking.date, tSeat.SBooked, tSeat.PricePaid FROM tCustomer INNER JOIN tBooking ON  tCustomer.PhoneNumber = tBooking.PhoneNumber INNER JOIN tTicket on tBooking.BookingRef = tTicket.BookingRef INNER JOIN tSeat on tSeat.SeatID = tTicket.SeatID WHERE tBooking.date = '{}'".format(self.dateformatted)

        self.value = 1

        self.changeheaders = 1

        self.conformation = ""

        self.Searching()

        self.Show\_Search\_Data()

        self.value = 0

        self.setrowcount = 0

    def Commiting(self):#inserting, updating and deleting values from the tables!

        statementSQL=self.commitSQl

        try:

            cs = (

                "Driver={SQL Server};"

                "Server=RX-ORIGIN\MSSQLSERVER01;"

                "Database=Muhie;"

                "Trusted\_Connection=yes;"

                "UID=RX-ORIGIN\16mal;"

            )

            cnxn = pyodbc.connect(cs)

            print("Connected")

            if cs is not None:

                cursor = cnxn.cursor()

                cursor.execute(statementSQL)

                cursor.commit()

        except pyodbc.DatabaseError as err:

            print("SQL Error, Plese check you have everything formatted correctly: {}".format(err))

        finally:

            cnxn.close()

            print("Connection Closed")

    def Searching(self):#Searching for values!

        totalRev = 0

        statementSQL = self.FindSQL

        try:

            cs = (

                "Driver={SQL Server};"

                "Server=RX-ORIGIN\MSSQLSERVER01;"

                "Database=Muhie;"

                "Trusted\_Connection=yes;"

                "UID=RX-ORIGIN\16mal;"

            )

            cnxn = pyodbc.connect(cs)

            print("Connected")

            if cs is not None:

                cursor = cnxn.cursor()

                cursor.execute(statementSQL)

                records = cursor.fetchall()

                if self.value == 1:

                    self.arraytobesortedaytobesorted = []

                    for row in records:

                        total = total =+ 1

                        total = row[5]

                        value = float(row[5])

                        self.arraytobesortedaytobesorted.append(value)

                        rawValue = total.strip("")

                        revenueMake = float(rawValue)

                        totalRev = totalRev + revenueMake

                    self.ui.AvalibleSeats\_2.setText(str(totalRev))

                    def QuickSort(arr):

                        elements = len(arr)

                        #Base case

                        if elements < 2:

                            return arr

                        current\_position = 0 #Position of the partitioning element

                        for i in range(1, elements): #Partitioning loop

                            if arr[i] <= arr[0]:

                                current\_position += 1

                                temp = arr[i]

                                arr[i] = arr[current\_position]

                                arr[current\_position] = temp

                        temp = arr[0]

                        arr[0] = arr[current\_position]

                        arr[current\_position] = temp #Brings pivot to it's appropriate position

                        left = QuickSort(arr[0:current\_position]) #Sorts the elements to the left of pivot

                        right = QuickSort(arr[current\_position+1:elements]) #sorts the elements to the right of pivot

                        arr = left + [arr[current\_position]] + right #Merging everything together

                        return arr

                    if len(self.arraytobesortedaytobesorted) > 10:

                        array\_to\_be\_sorted = self.arraytobesortedaytobesorted

                        print("The Most Paying customers in using quick sort order are: ",QuickSort(array\_to\_be\_sorted))

                    else:

                        pass

                    def bubbleSort(arr):

                        n = len(arr)

                        # Traverse through all array elements

                        for i in range(n-1):

                        # range(n) also work but outer loop will

                        # repeat one time more than needed.

                            # Last i elements are already in place

                            for j in range(0, n-i-1):

                                # traverse the array from 0 to n-i-1

                                # Swap if the element found is greater

                                # than the next element

                                if arr[j] > arr[j + 1] :

                                    arr[j], arr[j + 1] = arr[j + 1], arr[j]

                    # Driver code to test above

                    if len(self.arraytobesortedaytobesorted) == 1 or len(self.arraytobesortedaytobesorted) == 0:

                        pass

                    elif len(self.arraytobesortedaytobesorted) < 10:

                        arr = self.arraytobesortedaytobesorted

                        bubbleSort(arr)

                        print ("Most paying customers using buble sort is:")

                        for i in range(len(arr)):

                            print("% d" % arr[i],end=" ")

                    else:

                        pass

            if self.setrowcount == 1:

                self.ui.Table\_Table.setRowCount(len(self.arraytobesortedaytobesorted))

            else:

                self.ui.Table\_Table.setRowCount(20)

        except pyodbc.DatabaseError as err:

            print("SQL Error, Plese check you have everything formatted correctly: {}".format(err))

            self.ui.Table\_Table.clearContents()

        finally:

            cnxn.close()

    def Show\_Search\_Data(self):#showing the search values in the table

        # clean table

        self.ui.Table\_Table.clearContents()

        statementSQL = self.FindSQL

        # populate table headers

        if self.changeheaders == 1:

            self.ui.Table\_Table.setColumnCount(6)

            col = ["Phone","Customer","BookRef","Date","Seats Booked","PricePaid"]

            self.ui.Table\_Table.horizontalHeader().setSectionResizeMode(1, QtWidgets.QHeaderView.Stretch)

            self.ui.Table\_Table.horizontalHeader().setSectionResizeMode(2, QtWidgets.QHeaderView.Stretch)

            self.ui.Table\_Table.horizontalHeader().setSectionResizeMode(3, QtWidgets.QHeaderView.Stretch)

            self.ui.Table\_Table.horizontalHeader().setSectionResizeMode(4, QtWidgets.QHeaderView.Stretch)

            self.ui.Table\_Table.horizontalHeader().setSectionResizeMode(0, QtWidgets.QHeaderView.Stretch)

            self.ui.Table\_Table.horizontalHeader().setSectionResizeMode(5, QtWidgets.QHeaderView.Stretch)

        else:

            col = ["A","B", "C", \

               "D", "E","F","G" , "H","I","J"]

            self.ui.Table\_Table.setColumnCount(10)

        self.ui.Table\_Table.setHorizontalHeaderLabels(col)

        #statementSQL = self.FindSQL

        try:

            cs = (

                "Driver={SQL Server};"

                "Server=RX-ORIGIN\MSSQLSERVER01;"

                "Database=Muhie;"

                "Trusted\_Connection=yes;"

                "UID=RX-ORIGIN\16mal;"

            )

            cnxn = pyodbc.connect(cs)

            if cs is not None:

                cursor = cnxn.cursor()

                cursor.execute(statementSQL)

                rows = cursor.fetchall()

                noRow = 0

                for tuple in rows:

                    noCol = 0

                    for column in tuple:

                        satuKolum = QTableWidgetItem(str(column))

                        if column == self.conformation:

                            self.ui.Table\_Table.setItem(noRow, noCol, satuKolum)

                            if column == "FULL":

                                self.ui.Table\_Table.item(noRow, noCol).setBackground(QtGui.QColor(255,0,0))

                            elif column == "EMPTY":

                                self.ui.Table\_Table.item(noRow, noCol).setBackground(QtGui.QColor(0,100,0))

                        elif self.conformation == "":

                            self.ui.Table\_Table.setItem(noRow, noCol, satuKolum)

                        elif self.conformation == "ALL":

                            self.ui.Table\_Table.setItem(noRow, noCol, satuKolum)

                            if column == "FULL":

                                self.totalfullseats += 1

                                self.ui.Table\_Table.item(noRow, noCol).setBackground(QtGui.QColor(255,0,0))

                                formmated = str(200-self.totalfullseats)

                                self.ui.AvalibleSeats.setText(formmated)

                            elif column == "EMPTY":

                                if self.totalfullseats == 0:

                                    self.ui.AvalibleSeats.setText("200")

                                    self.ui.AvalibleSeats\_2.setText("0")

                                self.ui.Table\_Table.item(noRow, noCol).setBackground(QtGui.QColor(0,100,0))

                        noCol += 1

                    noRow += 1

        except pyodbc.DatabaseError as err:

            print("SQL Error, Plese check you have everything formatted correctly: {}".format(err))

            self.ui.Table\_Table.clearContents()

        finally:

            cnxn.close()

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QApplication(sys.argv)

    Start = Theatre()

    Start.show()

    sys.exit(app.exec())

## Developmental Testing

Identify at least 5 different problems you had and your solutions, there should be before and after images.

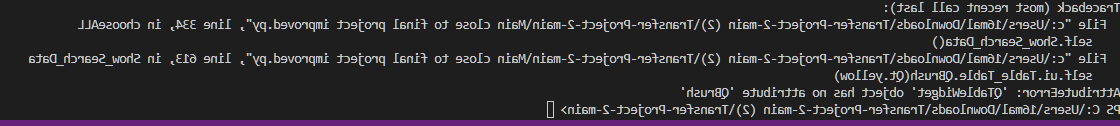
Error 1.

Syntax Error

Using PYQT5 I needed to figure out how to colour code certain table widgets, I tried many different methods to do this. Every time I got a syntax error as I did not know the right way how to colour in table widgets. Then I found the right way of colouring in QT5 table widget cells, resulting in the cells that had were “empty” colour coded to green.

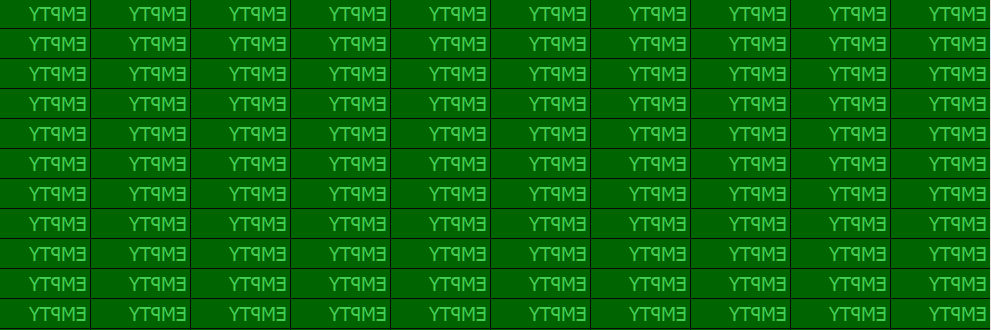
self.ui.Table\_Table.QBrush(Qt.green)#first approach.

This resulted in the error below



Figuring out the right syntax gave me this result:

 self.ui.Table\_Table.item(noRow, noCol).setBackground(QtGui.QColor(255,0,0))



Error 2:

Logic error

Every time I buy tickets, I need to select them form the available ones, the number of seats selected must correspond to the number of tickets purchased in the ticket section if not it will not allow any seats to be selected at-all, breaking the whole program entirely. When doing this I could not book multiple people in at the same time, so say if I booked one person in the next person would not be able to be booked in this was because I forgot to reset the running count of the total seats selected and I forgot to clear the list which the seats that are selected are displayed as an list inside of PYQT5. Once I had cleared the list of data after purchasing tickets and wiped the running count setting it back to zero, the problem no longer occurred.

Before:

After the first customer is added to the database
Graphical user interface

Description automatically generated

Error 3:

Runtime error:

When sorting the top paying customers, I need to have an array with the values being used, however with my searching algorithm into the database I don’t return the values of the top paying customers, for example I may need to search for customers, which doesn’t include the top paying ones. This caused a runtime error where the program would look for indexes in an array that doesn’t exist. To solve this, I made a variable that only allows for the sorting

algorithm to be ran if the value of the variable is true that is passed before the searching method is called, and then subsequently set to false after the sorting is ran. After the searching algorithm has taken place, this prevents wasting resources on sorting an array when not necessary and prevents crashes when there is no array to be indexed.

Before:

The program would start, but when searching for data my sorting algorithm would run at the same time, indexing an array that doesn’t exist causing this error below

Text

Description automatically generated

After:

Graphical user interface

Description automatically generated

Error 4:

Data type error/ runtime error:

When trying to implement my smart pricing algorithm, I tried get the current date form the python library date time. For the date of the performance I’d get the date of the date widget from pyqt5, the issue with this method would result in the performance date being held as a string rather than an date.datetime, to solve this I converted the string into a date.datetime. which eliminated the problem.

Before:

A picture containing graphical user interface

Description automatically generated

After:

Text

Description automatically generated

Graphical user interface

Description automatically generated

Error 5:

Logic error – not setting the row counts when necessary:

Since I am only using one table widget for the whole system, I need to adapt the row counts in PYQT5, for example when I click the customer information, want to set the number of customers booked of a specific performance or, the search results of a performance it will adapt the row count accordingly for the ‘seat view’ I want the number of rows to fixed to 20 when clicking back on to it

Before:

Graphical user interface

Description automatically generated`

After:

A screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generated

## Testing

Complete the testing table, you must comprehensively test your solution using a range of testing criteria. Testing should include:

* Valid data – data with expected outcomes that should be accepted by any validation you have included
* Invalid data – data that should be rejected with an error message
* Erroneous data – symbols etc in text fields that could break everything
* Extreme data – only used for range checks, data that should be accepted but is on the edges of acceptable.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | Test type | Test data | Expected result | Actual result | Commentary |
| Creating a new performance | Valid data | “06/05/2022” in performance date and then clicking the button “new show” | creates a table in MSSQL server called performance “06/05/2022” | Yes, creates a table in MSSQL server called performance “06/05/2022” | Creates the table in MSSQL server. |
| Trying to create a new performance on the same date | Invalid data | “06/05/2022” in performance date and then clicking the button “new show” | an error occurs, preventing the creation of a duplicate table. | Yes, an error occurs, preventing the creation of a duplicate table. | Prevents shows being created on the same night |
| Searching for customers by name | Valid data | ‘John Cena’ in the name line edit | Shows, all the details of John Cena | Shows, all the details of John Cena | Searches for the customer entered in the search box. |
| Searching for a customer who hasn’t been booked in. | Invalid | ‘Joe Biden’ in the name field. | Returns nothing in the PYQT5 table widget | Returns nothing in the PYQT5 table widget | return no values as ‘Joe Biden’ has not booked a ticket with us. |
| Phone number that has 10 digits instead of the required 11 | Invalid data | ‘0734343443’ in the phone number line edit | Returns the error message ‘Phone is empty, please add a value to this field, or Phone is not 11 digits’ | Returns the error message ‘Phone is empty, please add a value to this field, or Phone is not 11 digits’ | The system doesn’t allow any phone number to be entered that is less than 11 digits |
| Trying to add a person as a number as a name | Extreme data | ‘445454454’ as the name in the Name line edit | Rejected by the table, providing an error message about how the wrong data type is trying to be added to the database | Accepted by the database and is showing as a booked seat | To fix this from happening I could change the data type where the name is being stored form nvarchar to varchar, this would eliminate the issue. |
| Trying to enter a invalid price for a customer | Extreme data | ‘100’ in the price to pay, line edit and ‘Joe John’ in the name field and ‘07232321234' for the phone number | The actual price to pay should be added, which is 5 pounds in this case as ‘John Joe’ is a child | The function is called to check the price paid is valid, correcting the error and entering the right price for ‘John Joe’ which is 5 pounds | My program calls the final price function before, committing the statement to the database to prevent any misuse. |
| Blocking seats | Valid data | Clicking ‘block seats’ then selecting A11,B11,C11,D11,E11 | Should block all seats selected, updating them inside the database to ‘full’ rather than ‘empty’ | All seats selected are blocked. |  |
| Searching customers via there seats that have been booked | Valid data | Entering ‘H4’ into the search seats field and then clicking the search button | The customer with the seat of ‘h4’ is returned to the table widget. | John cena is returned to the table widget. |  |
| Clicking the customer button on a performance data that does exist | Valid data | Clicking the customer button on 05/05/2022 | All customers are from the selected date returned into the qt5 table widget | All customers are from the selected date returned into the qt5 table widget |  |
| Clicking the full seats button on a day with a performance | Valid data | Clicking the ‘full seats’ on 05/05/2022 | All seats that are full are selected. | All seats that are full are selected. |  |
| Clicking the empty seats button on a day with a performance | Valid data | Clicking the ‘empty seats’ on 05/05/2022 | All seats that are empty are selected. | All seats that are empty are selected. |  |
| Checking that the Total revenue is correct. | Valid data | Checking all the customers booked into the performance add up to the displayed amount | All customers booked add up to ‘55’ pounds | All customers booked add up to ‘55’ pounds |  |
| Trying to book a person with the name of an emoji | Erroneous data | 😀  In the name column | Rejects the booking of the person as it is not recognised in the character set | Allows the person to be booked with the name ‘??’ but doesn’t crash the program |  |
| Is the number of available seats, correct? | Valid data | All he full seats in the seat view | Should total to 173 available seats out of 200 | Totals to 173 seats out of 200 |  |
| VIP/Staff can book seats | Valid data | Name staff member, 5 tickets are booked, on the 05/05/2022 | Staff pay zero, to book and their information is stored in the database | Staff pay zero, to book and their information is stored in the database |  |
| Searching via a phone number that a has been booked into the performance | Valid data | 00306565642 for the phone number | Staff member information returned | Staff member information returned |  |
| Searching for a person that has booked on a different date via phone number | Erroneous data | 00306565642 on the performance date of ‘06/05/2022’ | No-one is returned into the table widget | No-one is returned into the table widget |  |
| Checking that the Total revenue and available seats change on different performance dates | Valid data | Changing the date from ‘05/05/2022’ to ’06/05/2022 | The values change to the correct amount168 for seats, 60.0 for revenue in 05-05  And 195, 25.0 for 06/05 | The values change to the correct amount168 for seats, 60.0 for revenue in 05-05  And 195, 25.0 for 06/05 |  |
| Trying to add a seat that is already booked. | Erroneous data | Changing the date from ‘05/05/2022’ to ’06/05/2022 and selecting a seat that has been booked on ’05-06-2022’ | The seats selected get wiped when trying to change dates. | The seats selected get wiped when trying to change dates. | It’s hard to show this but it does work you can try it for your self |
| Trying to add a variation of ‘EMPTY to try to book seats that are already booked as a customer’ | Erroneous data | ‘empTY’ in the name field | The name is rejected as it is a variation of Empty with different amounts of capitalisation | The name is rejected as it is a variation of Empty with different amounts of capitalisation |  |
| Trying to add a variation of ‘full’ to try to book seats that are already booked as a customer’ | Erroneous data | full in the name field | The name is rejected as it is a variation of full with different amounts of capitalisation | The name is rejected as it is a variation of full with different amounts of capitalisation |  |
| Trying to add a 12 digit phone number | Invalid data | Impossible to add a 12 digit phone number as it is hard coded to only allow a maximum of 11 characters | Impossible to add a 12-digit phone number as it is hard coded to only allow a maximum of 11 characters | Impossible to add a 12-digit phone number as it is hard coded to only allow a maximum of 11 characters |  |
| Trying to add a phone number as a string | Erroneous data | ‘thisisatest’  In the phone number field | The name should be rejected as it has characters in it | The phone number is not rejected and is entered into the database | This could quite easily be fixed with a for loop checking each time if it is a digit, if not, it will be rejected, I will fix it now. 06/06/22  Adding this line of code fixed this:              for i in range(0,11):                  if Phone[i].isdigit():                      pass                  else:                      return |
| Trying to click ‘full seats on a day which has no performance’ | Extreme data | clicking the ‘full seats’ button on ’20-05-22’ | An SQL error occurs saying that the table doesn’t exist | The database connects but no values are returned into the table widget | I’m not sure why it does this but as long as it doesn’t crash that is fine. Or cause any bugs and or glitches |
| Trying to create a performance on a date that already happened | Invalid data | Trying to select ‘04/04/2022’ | Impossible to add a data less than the current date | Impossible to add a data less than the current date |  |
| Trying to click ‘empty seats on a day which has no performance’ | Extreme data | clicking the ‘empty seats’ button on ’20-05-22’ | An SQL error occurs saying that the table doesn’t exist | An SQL error occurs saying that the table doesn’t exist, returning no data into the table |  |
| Reset seats button when seats are selected. | Valid data | Clicking the reset seats button | Clears the amount of seats selected |  |  |
| Adding a performance, a year from the current date | Extreme data | Adding 07/05/2022 to the database as a performance | Should be valid as only a year as the program only rejects if it is more than a year and a day. | Yes, the performance is created successfully, ready for use. |  |
| Trying to buy a ticket with an OAP | Valid data | John doe sitting in A1  In an OAP seat on 15/05/2022 | Should cost 5 pounds and be added to the databases | John doe is added to the database, buying the seat A1, it cannot be purchased again |  |
| Trying to book an in a person with the same name as someone already in the database on the same day | Extreme data | John doe on 15/05/2022 with a different phone number | Causes the program to crash as the booking reference is the name of the custom r and the date of the performance | Causes the program to crash as the booking reference is the name of the custom r and the date of the performance, and returns an SQL error | This could be fixed if I made totally unique booking refernces but the chances of this are quite slim and it would take a long time to implement |
| Trying to buy a ticket with an Adult | Valid data | Jeff brooks on 15/05/2022 | Should allow the person to be booked with the price of 10 pounds | Allows the person to be booked with the price of 10 pounds |  |
| Trying to book a child into the database | Valid data | Lisa bob on 15/05/2022 | Should allow the person to be booked with the price of 5 pounds | Allows the person to be booked with the price of 5 pounds |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Evaluation

You should write a detailed evaluation covering the following points

* How effective the language including libraries etc you chose is for solving this and what features you used.
* Compared the system you have made with those you researched for your desk based research.
* Identified strengths and weaknesses of your system with potential improvements, you don’t have to program these so you can be imaginative
* Your own strengths and weaknesses, consider each stage of the process
* Identify any personal changes you would make for the future

My evaluation.

For this project, I researched, three different websites Cineworld, British Airways and Everyman. I think that I made a semi-accurate emulation of these systems. For example, I created a way for people to be booked into a performance on any given day, similarly to all the systems. Also, I allow users to choose their price bands depending on their age. Furthermore, just like in the British Airways system, all be it basic, I created a smart pricing system. My system only takes two factors on when to change prices, and these prices are fixed between the two bands, either 10,5,5 (pounds) for Adults, OAPs, and Children respectively when the performance is greater than 5 days away and. Whereas the cheaper price when the performance day is less than 5 days away is 5,2.50,2 (pounds) for Adults, OAPs, and Children respectively. I’m sure the British Airways system is far more advanced, taking in hundreds of factors into account.

Compared to the other systems I think my design is not as attractive and can only use on the desktop with a python installed, this is because I was using PYQT5 to design the user interface, I think that more time I would have used flask with HTML, CSS, and JavaScript to design the system, this would allow for portability to be used on any device with a web browser. Also, If I wanted to make an application rather than a website, I’d use flutter which has great cross-platform compatibility and is extremely fast.

To create my smart pricing algorithm, I used python date-time to calculate the difference between two dates, it was extremely fast and simple to implement. Originally, I wanted to give the days, months, and years a weighting to calculate the number of days left till a performance, but with this library it made it simple as subtracting two dates from each other.

I was going to also use SMTP.lib to send emails to the customer about the tickets and the performance that they had selected. However, it would have been pointless as google are shutting down the ability to send emails via this service in the near future. Due to it not being very secure. If I was using another provider, it would have some cost attached to it.

I also used PYQT5 core which was fundamental for my seat booking system with the seat display. PYQT5 core allowed me to find out if a specific table cell was clicked in the system, which then I checked if it was full or not to allow for the user to click seats and that got displayed on the user interface to show what seats had been selected. It was also vital to allow me to colour in the cells of the table depending on if a seat was ‘empty’ for green or ‘Full’ for read to make it easy for users to select full seats. It also completely prevented users selecting occupied seats as it would return an error message. Again, this somewhat emulated the systems that I researched at the start. Although using this system meant that I had to stick to the layout of the table, meaning I could not custom layouts. If I were to do this again, I’d probably make 200 buttons which could be clickable so I could use a custom seat image. I think this would make the UI look much more ascetically pleasing.

My database is simple, to create a new show it creates a specific table for that date, in the brief It said that the performance would operate only 3 nights, which is good as in a company that may do more shows would have to delete the old tables otherwise it would take up a lot of data in my program. Think that I could do this quite easily and would need to happen for that circumstance. Furthermore, my seat searching works off a SQL like statement so say I was searching for seat A13, it would also return seats A12 or A14 for example.

My program meets all the criteria of the stakeholders and goes beyond that with the implementation of the smart Pricing algorithm, the seating display which is colour coded and is able to be click multiple times for the staff member to add multiple seats to a performance. Although I did not create a system to prevent seats single left in middle rows or produce the emailed ticket. As for the single seats, I though it was extremely unfair how you could not choose your own seats with that system. For the electronic ticket I’d have to create a template for a ticket in a program like photoshop. Then overlay text for the performance details in HTML. This could then be printed by the end user.

I think that my biggest strength is my coding as I completed it within the time constraints with a full-featured program, my program allows you to book seats quickly and efficiently, it is robust as it doesn’t crash, quick and easy to use. The only details of the customer you need to add in is the date of the performance, the Phone number of the customer and the name and the number of tickets, then all you have to do is select the seats of the theatre. You can block seats by checking the option in the menu and then all you need to do is click the seats that you want to block that’s it.

Next time I think I should add more images and a more modern UI design with rounded corners, animations, dark mode and light mode, shadows, and gradients. Also, I’d like to add a way of creating season tickets, where passholders upload their photos and can pay a one-time fee to watch unlimited showings from the theatre, to increase the number of options to the user. Maximising profits. Another thing I’d like to add is the ability to refund seats, if the customer would like to cancel before the performance happens, I’d only allow is there were 5 days’ notice to be able to do this, this would work by deleting records form the database and updating the seats from ‘full’ to ‘empty’

Compared to the paper-based system, my design is fast, easy to use and prevent duplication of seats being booked, with the SQL database, it also allows for the searching of customers with one click of a button. Hopefully it resolves any booking errors. It also does use paper, that can easily be lost or destroyed. With the SQL server, it means that customer can easily backed up or archived.

1. [↑](#footnote-ref-1)