UNIVERSITI TEKNOLOGI MARA

KEDAH BRANCH, SCHOOL OF INFORMATION SCIENCE

COLLEGE OF COMPUTING, INFORMATICS AND MATHEMATICS

DIPLOMA IN LIBRARY INFORMATIC (CDIM144)

IML208: PROGRAMMING FOR LIBRARIES

GROUP ASSIGNMENT:
"CINEMANIA"

PREPARED BY:

ALYA NATASHA BINTI SAMSURI (2022622842)

PUTERI NUR SYAZLEEN BINTI MOHD FAUDZY (2022877382)

WAN NURZULAIKHA BINTI MEGAT ZULKEFLI (2022812158)

ZAHIRATUL SYAZWINA BINTI ZAIDI (2022660998)

GROUP CDIM1443F

PREPARED FOR:

MR. AIRUL SHAZWAN BIN NORSHAHIMI

SUBMISSION DATE:

17th JANUARY 2024

GROUP ASSIGNMENT:
“CINEMANIA”

PREPARED BY:

ALYA NATASHA BINTI SAMSURI (2022622842)
PUTERI NUR SYAZLEEN BINTI MOHD FAUDZY (2022877382)
WAN NURZULAIKHA BINTI MEGAT ZULKEFLI (2022812158)
ZAHIRATUL SYAZWINA BINTI ZAIDI (2022660998)

GROUP CDIM1443F

CDIM144 – DIPLOMA IN LIBRARY INFORMATIC

SCHOOL OF INFORMATION SCIENCE
COLLEGE OF COMPUTING, INFORMATICS AND MATHEMATICS
UNIVERSITI TEKNOLOGI MARA (UITM)
KEDAH BRANCH

TABLE OF CONTENTS

1.0 INTRODUCTION

In this group projects, we decided to make a database about movie cinema and it called “CINEMANIA”. For this assignment, we choose to code a movie booking system in Python. The system is a simple Python programming application built with Tkinter for the graphical user interface (GUI) and MySQL Connector to link the Python programmed to the database, where the user must enter a few information in order to make a booking using the system.

The main window is the primary interface that users see upon launching the application. On the main window, there are three buttons: one for movies, one for customer details, and the last one for payment details.

For this movie booking system, we have made three submodules, which are the movie table, the customer details table, and the payment details table. Every table has its own functionality that the customer needs to use for the booking system. Firstly, the first submodule is the movie table; it is for customers to choose movies, showtimes, and seats. Next is the customer table. This table requires the customer to enter their data, such as their name, email, and phone number. The last one is a payment details table for customers to select methods of payment and enter the data needed.

Next, to interact with the data, we also use the CRUD (Create, Read, Update, Delete) operations in the database system. CRUD functions, explaining how they help with storing, getting, changing, and deleting data in a database. User interfaces and backend functionality must work together seamlessly for CRUD operations to be implemented in applications. Forms and buttons are examples of frontend elements that are intended to make it easier for users to interact with these functions.

In a GUI table interface, we have added buttons for “DELETE” and “UPDATE” to manage the records or entries within the table. In every table, we add a delete and update button so that customers can delete and update the data if they need to. The purpose of the “DELETE” button is to remove a selected record or entry from the table. Meanwhile, the “UPDATE” buttons allow customers to modify information in a selected record.

The process of developing a graphical user interface (GUI) that efficiently connects with a database. We use localhost as the hostname to connect to a database server. After all the data has been entered from customers, it will appear in the database, where it is stored through the table of the submodule.

2.0 PROBLEMS STATEMENT

2.1 INCORRECT INFORMATION

Sometimes the information accessible on movie booking websites is erroneous, such as showing times that are out of date or ticket prices that are incorrect. Customers may be confused and disappointed as a result of this. The consequences of incorrect information in the cinema industry are crucial. This may suffer the company from reputational damage if incorrect information increase, potentially impacting the future endeavours.

2.2 SECURITY CONCERNS

Online movie booking entails exchanging sensitive information with the booking website, such as credit card data. This may cause customers to be concerned about the security of their personal and financial information. (Ahasanul et al., 2009) observed that a problem which is noticeable is that there is no actual guarantee that a particular service can be sold online due to certain factors. From another important point of view, customers, as of now, do not feel fairly confident to engage in online transaction because of the insecurity associated with the disclosure of personal private information and data such as age, date of birth, nationality, and details of credit card on websites which are conditions often required by the vendors.

2.3 TECHNICAL ISSUES

Technical issues, such as sluggish loading times, website breakdowns, or difficulty processing payments, can occur with online movie booking systems at times. Customers may be irritated, and the movie theatre may lose sales as a result. One of the prominent technical issues encountered in online movie booking is slugging loading times. When users encounter delays in accessing the booking platform, navigating through different sections, or loading seat selection interfaces, this occurs. Slow loading times not only frustrate users but also contribute to a subpar booking experience.

3.0 OBJECTIVES

3.1 ACCESSABLE SERVICES

To automate the ticketing procedure for the cinema hall. In order to reduce the work required to complete this operation. This is an online web platform where users and theatre owners can utilise to update movies in theatres. Customers will come to the ticket window less frequently if services are conveniently accessible to them. This might also save the company money because they won't require as many people at the ticket booths. Employees will be able to focus their efforts elsewhere. Online platforms do not require a rest or a night off. They can operate continuously, giving every client access to information at any time of day. The initiative aims to provide movie information and ticketing services 24 hours a day, seven days a week. Unlike traditional ticket windows, which have set hours of operation, online platforms are open 24/7, allowing customers to browse movie options, check showtimes, and make reservations at any time. This ensures that a diverse range of customers with varying schedules can be reached and served.

Lubeck, Wittman and Battistella (2012) are able to examine these issues by tracing the evolution of e-tickets and efforts by the organization to improve efficiency in ticketing operations. According to these authors, e-tickets have evolved to address concerns associated with "inefficiency in information management and control of operations" (p. 18). E-tickets, as noted by Lubeck and co-workers, require the creation of a comprehensive technological platform that controls almost every aspect of the customer relationship within the organization. As such, the roots of e-ticketing go much further than the interface with the customer.

3.2 DATA INTEGRITY

Data integrity. It is the main concern of data outsourcing. The data in the DSP may be a risk of loss or corruption. For example, Amazon, Dropbox, and Tencent Cloud lost their data due to natural disasters, software vulnerabilities and human error. The use of incomplete data can cause economic damage for data consumers and reputation damage for data providers (Zhang et al., 2023).

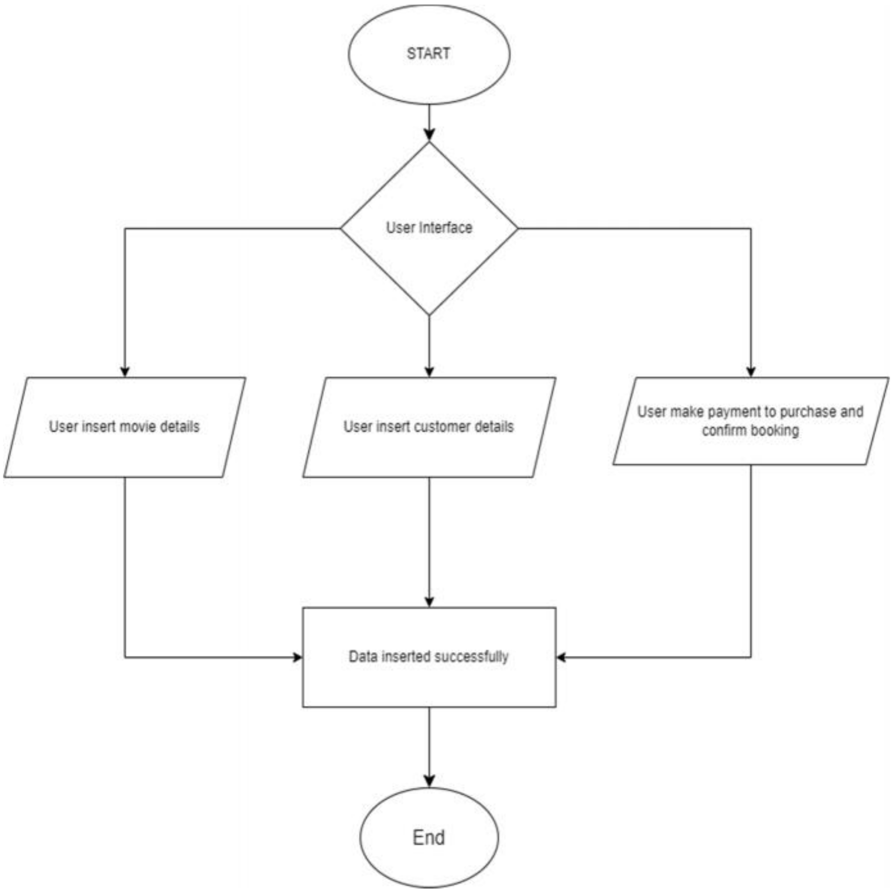
Particularly in this Application, Webpages and Forms, as well as controls in Webpages with which the user interacts. Application logic, often known as business rules, performs calculations and determines the application's flow. Business rules are the procedures and policies that govern how a company behaves. Business rules are self-imposed constraints that firms adopt to help them operate in their specific business environment. Business rules frequently establish and provide recommendations for application needs. The focus in on implementing and maintaining the integrity of webpages forms within the online ticketing platforms.

3.3 FACILE TO HANDLE

The system will be facile to use in that it will aid users in appropriately adding data without being confused about how to key in the data. Users will find it simple to use and manage the system thanks to an intuitive Graphical User Interface (GUI). The goal is to design a system that aim users in adding data without confusion. This includes developing clear and simple data input fields, validation prompts, and error feedback mechanisms.

The system's goal is not only to help users with data entry, but also to make overall system management simple and straightforward. This includes features like updating movie information, managing seat availability, and dealing with transaction records. Users, including cinema staff, can efficiently handle and maintain the online ticketing system by incorporating simplicity into these management tasks. Turner and Wilson (2010) integrated ticketing is expected to deliver greater flexibility and simplicity for passengers, prompting greater use of public transport, combined with smart ticketing to offer increased speed, convenience and security against loss and theft. Based on these statements, enough to prove that it is easy and also has many benefits if we use this system.

4.0 MAIN FLOWCHART

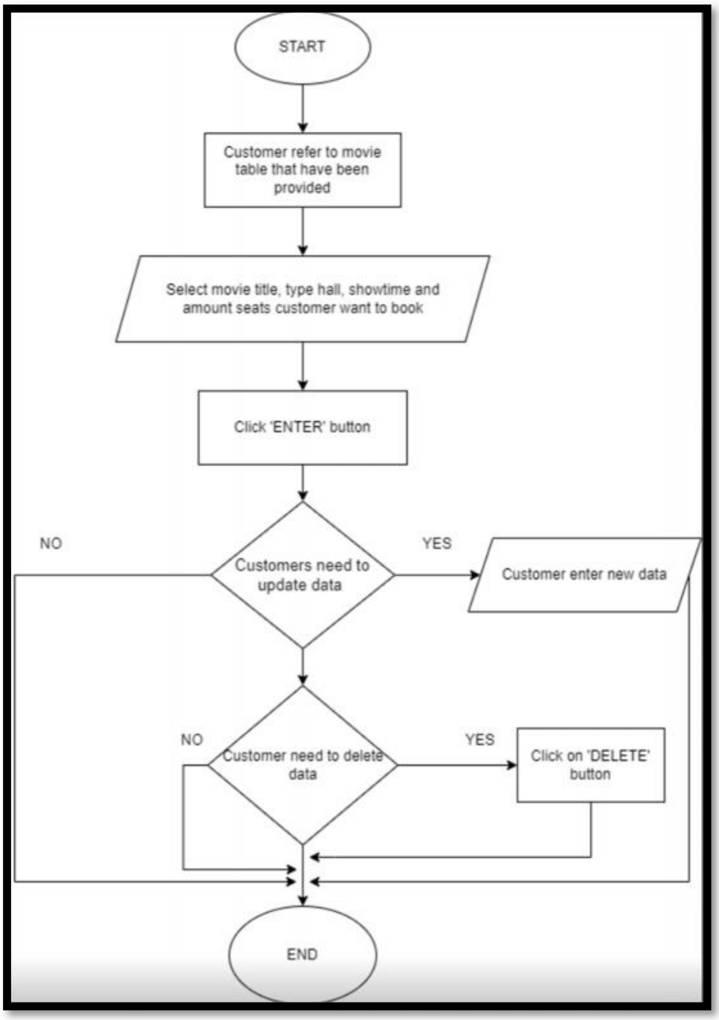

START
User Interface
User insert movie details
User insert customer details
User make payment to purchase and confirm booking
Data inserted successfully
End

EXPLANATION:

The main flowchart starts with “START”, the starting point of our flowchart. Next, “user interface”, this is the interaction between the user and the system. In the flowchart, it appears that the user will interact the system with three different actions. First, “user insert customer details”, the users need to insert customer details, this could include information such as name, email and phone number. Secondly, “user insert movie details”, this suggests user to insert movie details including selecting a movie, type hall, and seat number. Thirdly, “user make payment to purchase and confirm booking”, user needs to make a payment to purchase and confirm the booking. Users can choose what type of way they want to pay it. Lastly, “data insert successfully”, this is the endpoint of all three actions. When users successfully complete any of three actions, it will end to this section.

5.0 SUBMODULES FLOWCHART

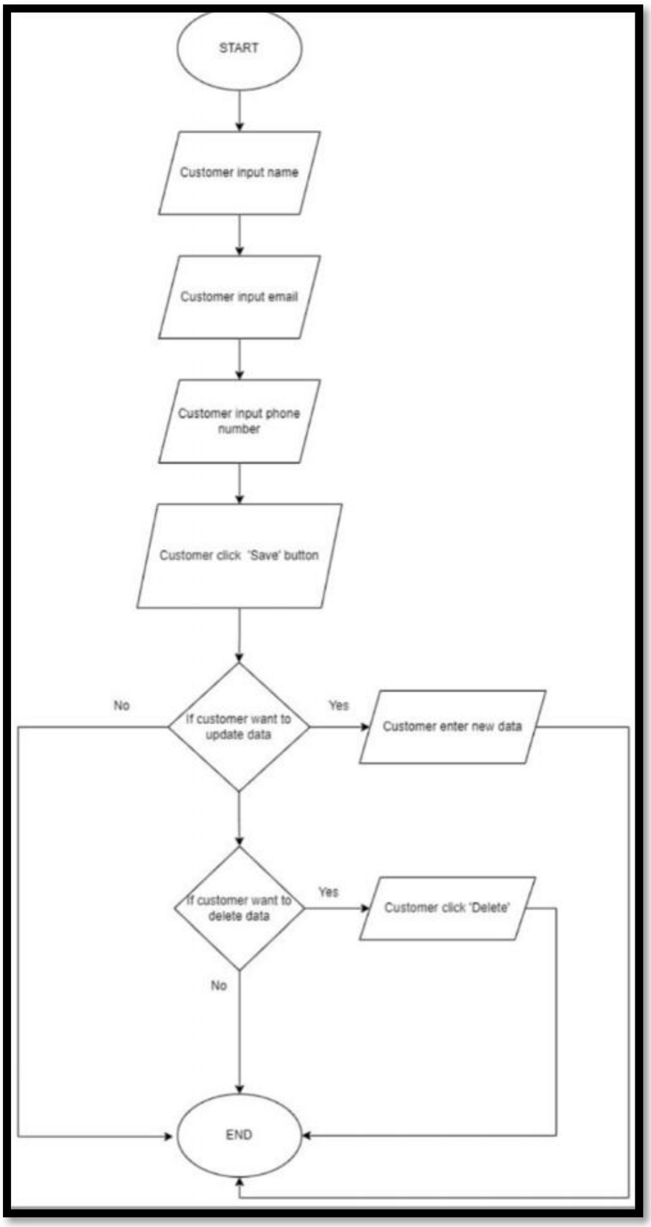
5.1 MOVIE


START
Customer refer to movie table that have been provided
Select movie title, type hall, showtime and amount seats customer want to book
Click 'ENTER' button
NO
Customers need to update data
YES
Customer enter new data
NO
Customer need to delete data
YES
Click on 'DELETE' button
END

EXPLANATION:

Movie flowchart starting with "customer refer to movie table that has been provided" that's mean, it has a lot of movie's choices so that customer can refer to it. If they already decide what movie they want to watch, they can move to other section which is "select movie title, type hall, showtime, and amount seats customer want to book". Then, "click 'ENTER' button". If "customer need to update data", it will proceed to "customer enter new data" if no, it will "END". Other than that, if "customer need to delete data", they need to "click on 'DELETE' button". If no, the process will "END".

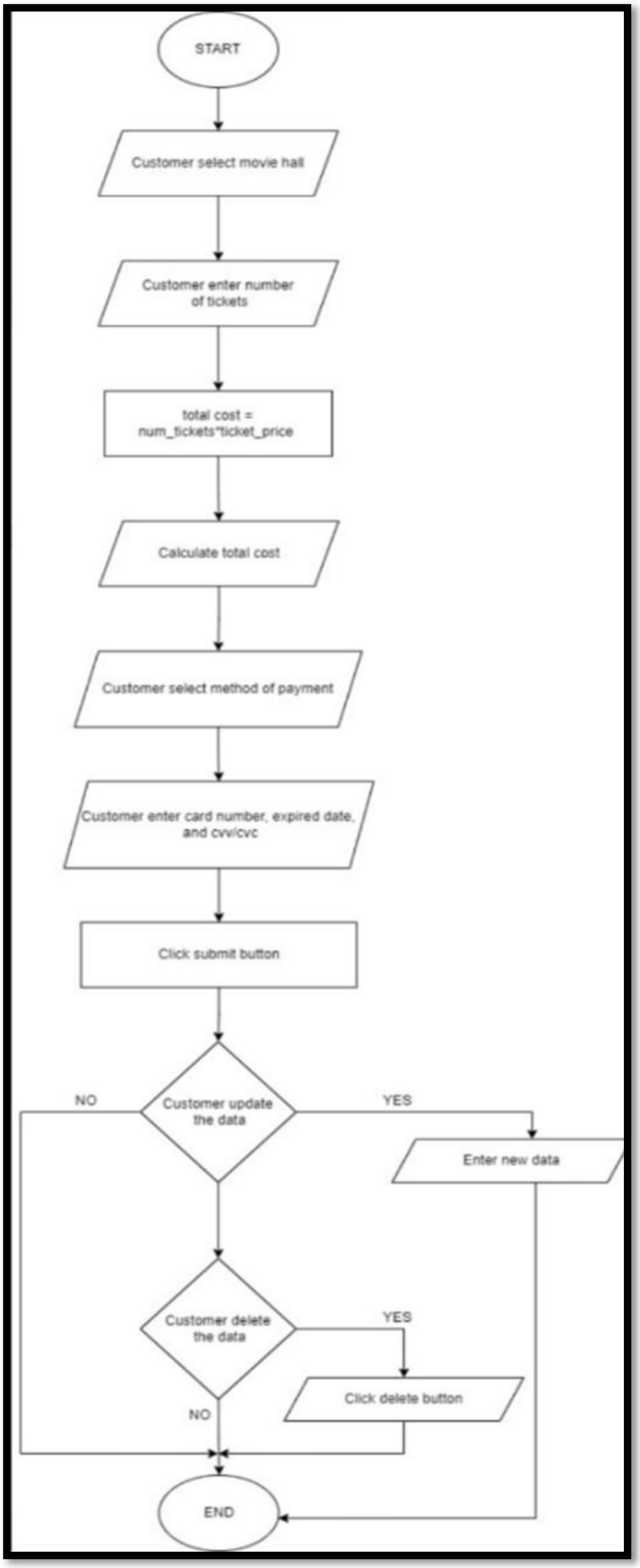
5.2 CUSTOMER


START
Customer input name
Customer input email
Customer input phone number
Customer click 'Save' button
No
If customer want to update data
Yes
Customer enter new data
If customer want to delete data
Yes
Customer click 'Delete'
No
END

EXPLANATION:

First off, “customer input name”, they need to input their name as it required to. After that, “customer input email”, they have to fill in their email. Next, “customer input phone number”, they need to insert their phone number as it asked to. After all the requirement have been fill in, “customer click ‘Save button’”. If “customer want to update data”, “customer enter new data”. If no, the process will “END”. Other than that, if “customer want to delete data”, “customer click ‘Delete’” and the process will “END”.

5.3 PAYMENT DETAILS


START
Customer select movie hall
Customer enter number of tickets
total cost = num_tickets*ticket_price
Calculate total cost
Customer select method of payment
Customer enter card number, expired date, and cvv/cvc
Click submit button
Customer update the data
NO
YES
Enter new data
Customer delete the data
YES
Click delete button
NO
END

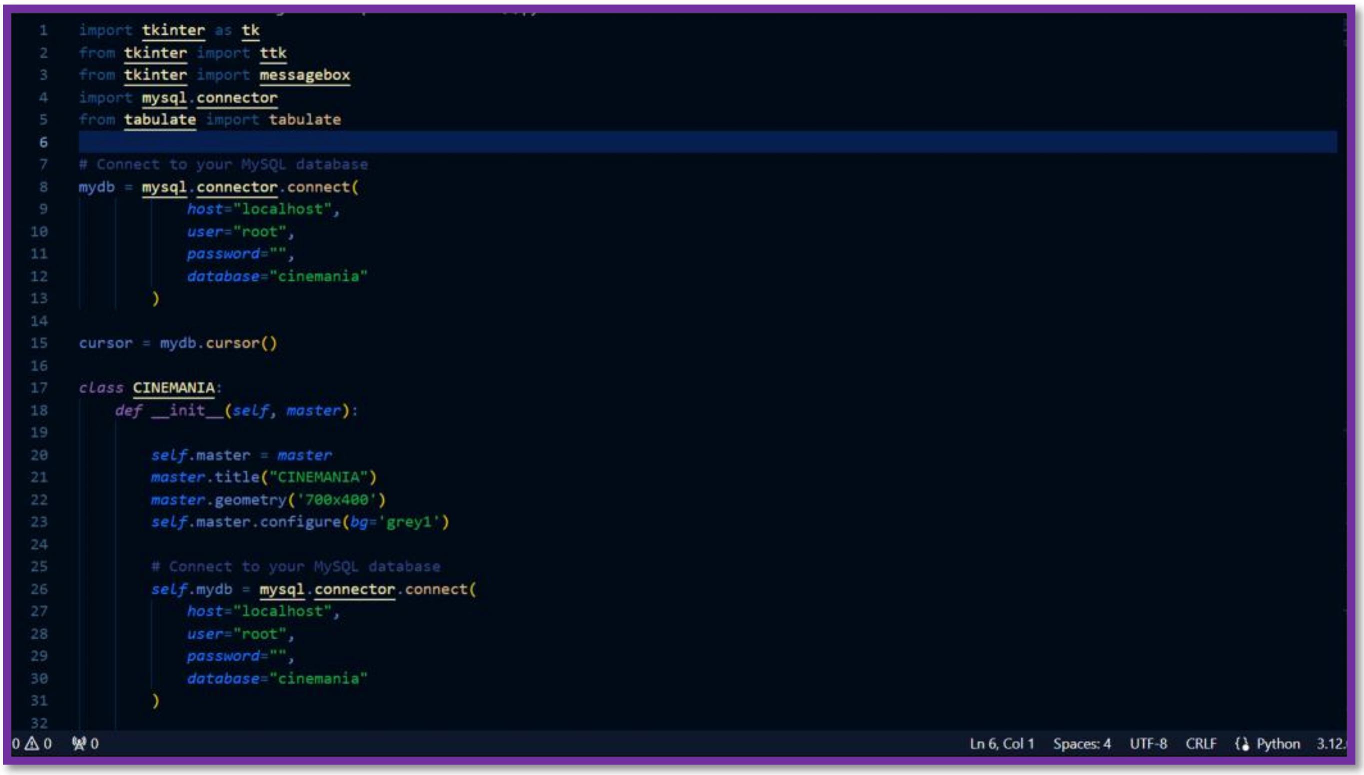
EXPLANATION:

Flowchart for payment details, start off with “customer select movie hall”, as we provide various type of hall, users can choose their desire type hall. After that, “customer enter number of tickets”, by that user can choose as many tickets they want. And then, our system will calculate “total cost= num_tickets*ticket_price” and it will show the “calculate total cost”. User will see the total cost and proceed to the next steps, “customer select method of payment”. It will be the choices of methods that user can choose. And, “customer enter card number, expired date, and cvv/cvc” and “click submit button”. If “customer update the data”, they need to “enter new data”. If no, the process will “END”. Other than that, if “customer delete the data”, they need to “click delete button”. If no, the process will “END”.

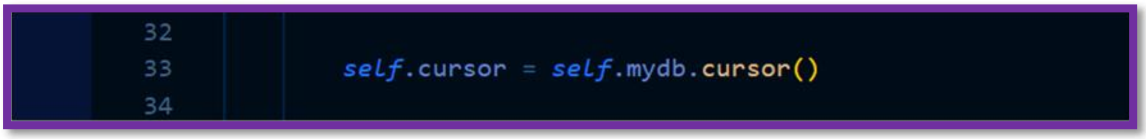
6.0 PYTHON CODE

6.1 CINEMANIA IMPORTING CODE TO DATABASE SQL

Firstly, we insert the SQL Connector to our database so that all the information that user key in, will successfully keep in there and protect it from missing. After that, create a new database under name “cinemania” to connect it to “localhost”.



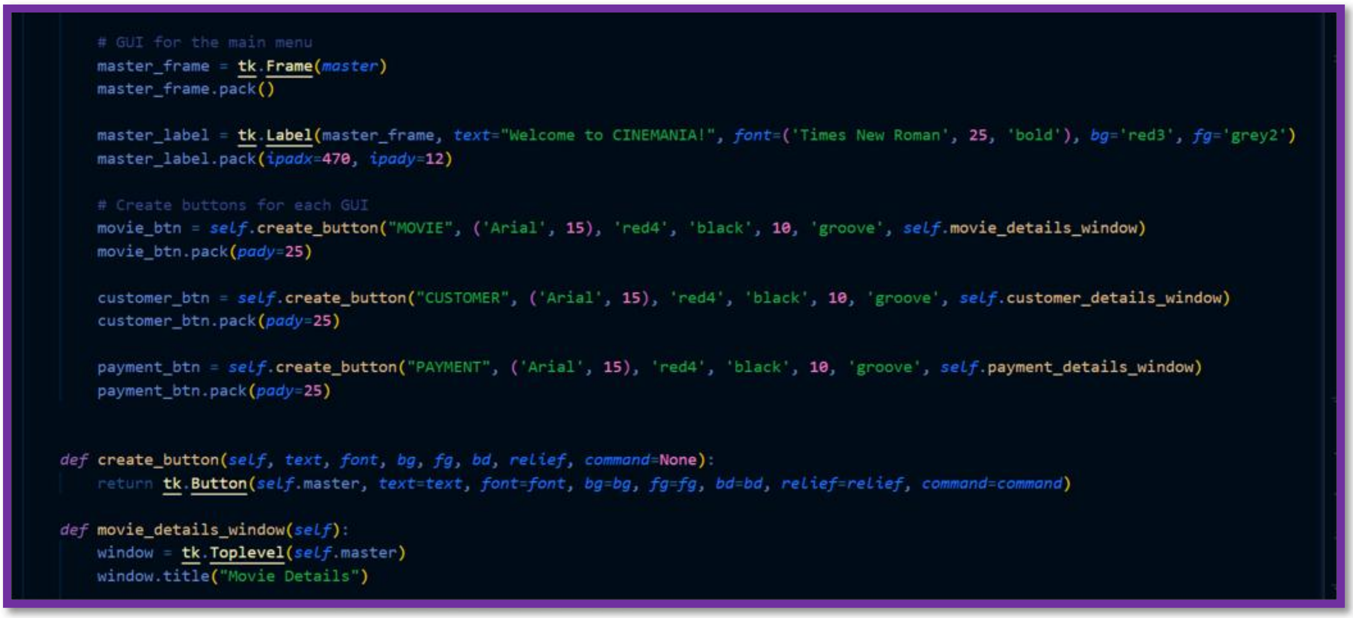
```python
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import mysql.connector
from tabulate import tabulate

# Connect to your MySQL database
mydb = mysql.connector.connect(
            host="localhost",
            user="root",
            password="",
            database="cinemania"
        )

cursor = mydb.cursor()

class CINEMANIA:
    def __init__(self, master):

        self.master = master
        master.title("CINEMANIA")
        master.geometry('700x400')
        self.master.configure(bg='grey1')

        # Connect to your MySQL database
        self.mydb = mysql.connector.connect(
            host="localhost",
            user="root",
            password="",
            database="cinemania"
        )

```



```python

        self.cursor = self.mydb.cursor()

```

6.2 GUI CODING FOR THE MAIN MENU AND BUTTON FOR EACH GUI

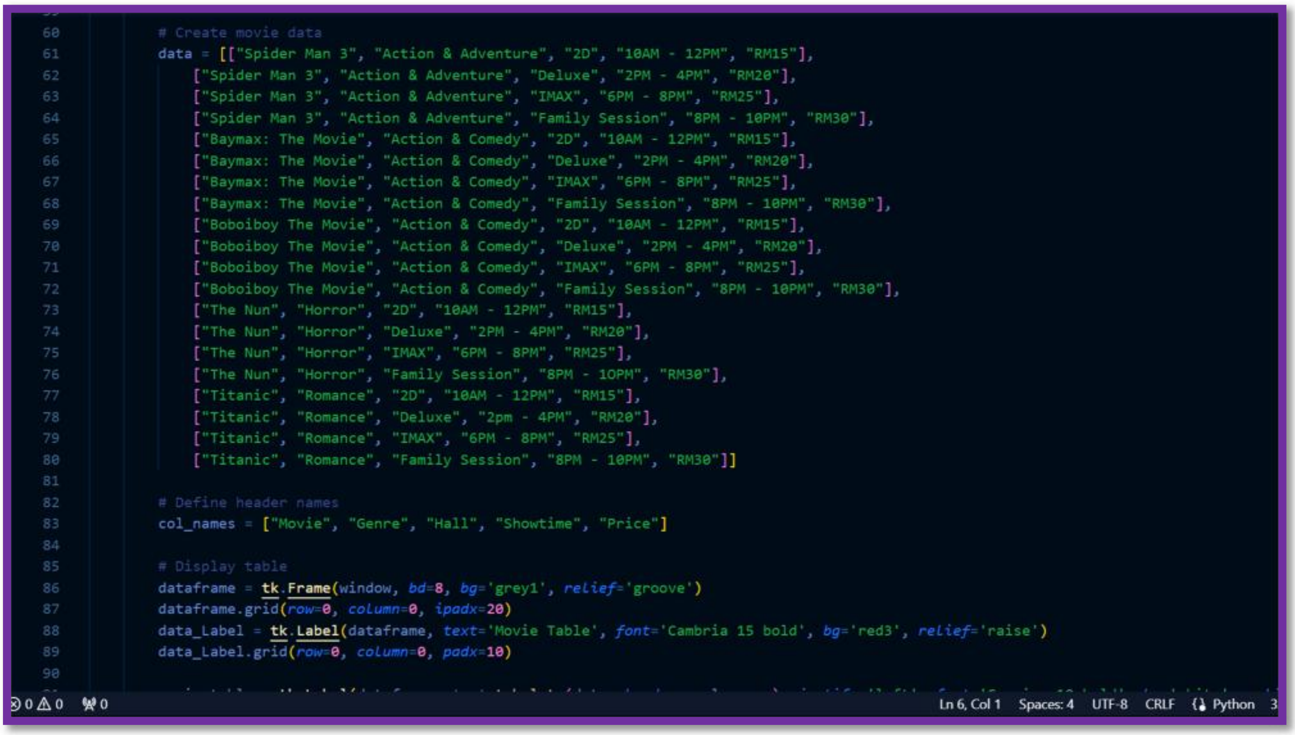
In this part, we creating the main menu with “Welcome to CINEMANIA!” name. Next, we create the interface for user to choose “movie”, “customer” and “payment”. They can click to each of these three buttons. Also, we used “Times New Roman” font to make it more interactive to the users.



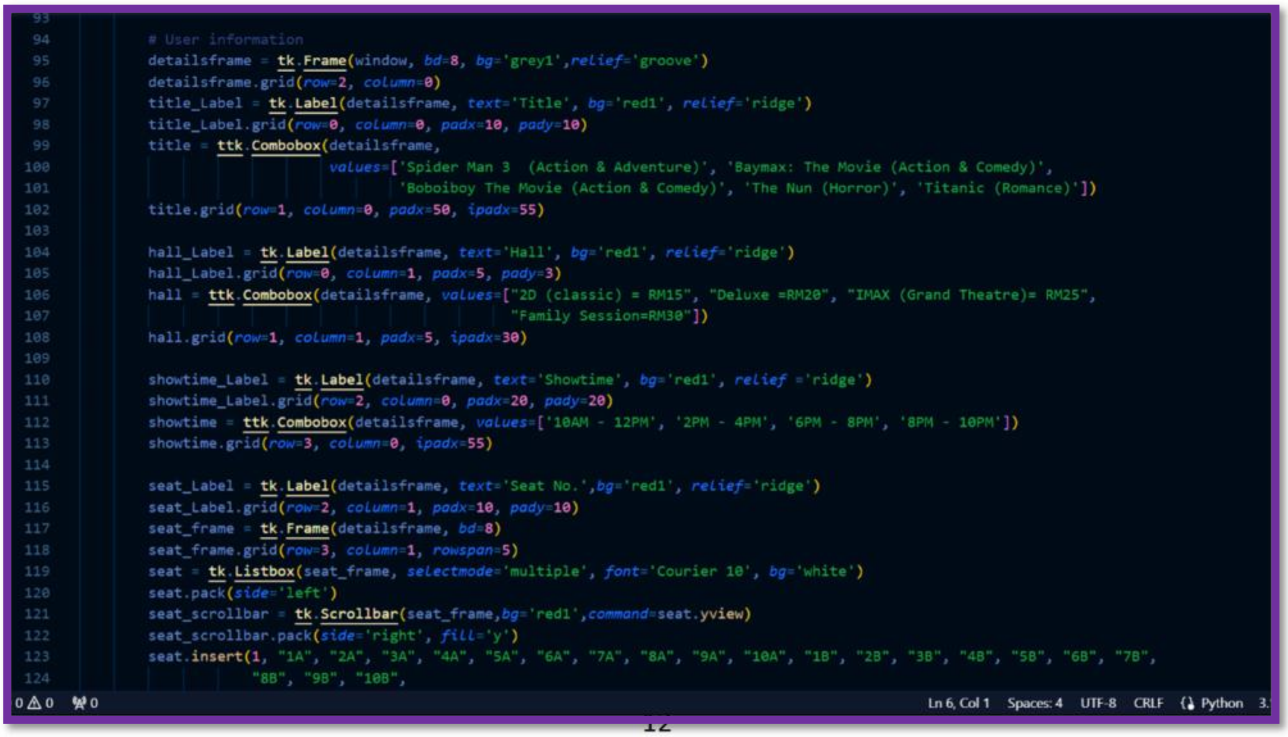
```python
        # GUI for the main menu
        master_frame = tk.Frame(master)
        master_frame.pack()

        master_label = tk.Label(master_frame, text="Welcome to CINEMANIA!", font=('Times New Roman', 25, 'bold'), bg='red3', fg='grey2')
        master_label.pack(ipadx=470, ipady=12)

        # Create buttons for each GUI
        movie_btn = self.create_button("MOVIE", ('Arial', 15), 'red4', 'black', 10, 'groove', self.movie_details_window)
        movie_btn.pack(pady=25)

        customer_btn = self.create_button("CUSTOMER", ('Arial', 15), 'red4', 'black', 10, 'groove', self.customer_details_window)
        customer_btn.pack(pady=25)

        payment_btn = self.create_button("PAYMENT", ('Arial', 15), 'red4', 'black', 10, 'groove', self.payment_details_window)
        payment_btn.pack(pady=25)


    def create_button(self, text, font, bg, fg, bd, relief, command=None):
        return tk.Button(self.master, text=text, font=font, bg=bg, fg=fg, bd=bd, relief=relief, command=command)

    def movie_details_window(self):
        window = tk.Toplevel(self.master)
        window.title("Movie Details")
```

6.3 CODING FOR MOVIE (FIRST SUBMODULES)

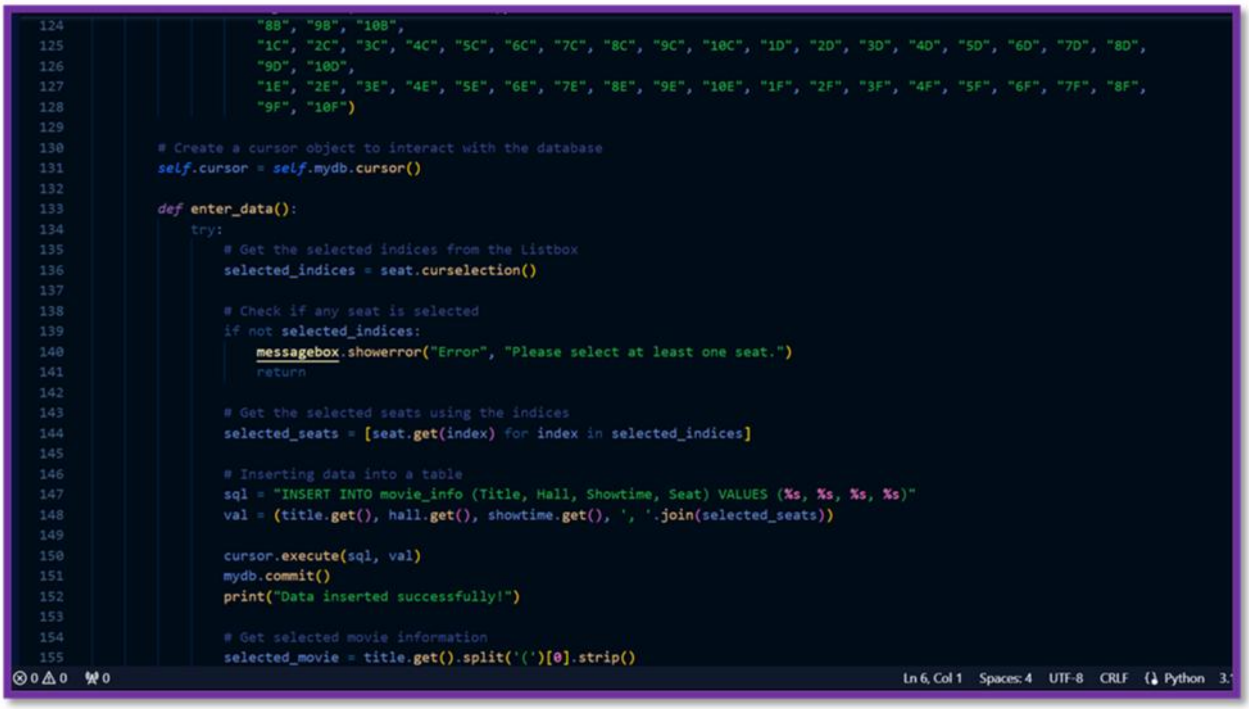
Our first submodules are Movie. In this part of coding, user can see the various choices of movie title, genre, hall, showtime and price. Users can choose their interest movie to watch based on the type hall that they want, each of the movie has their own showtimes that we already set up for it.



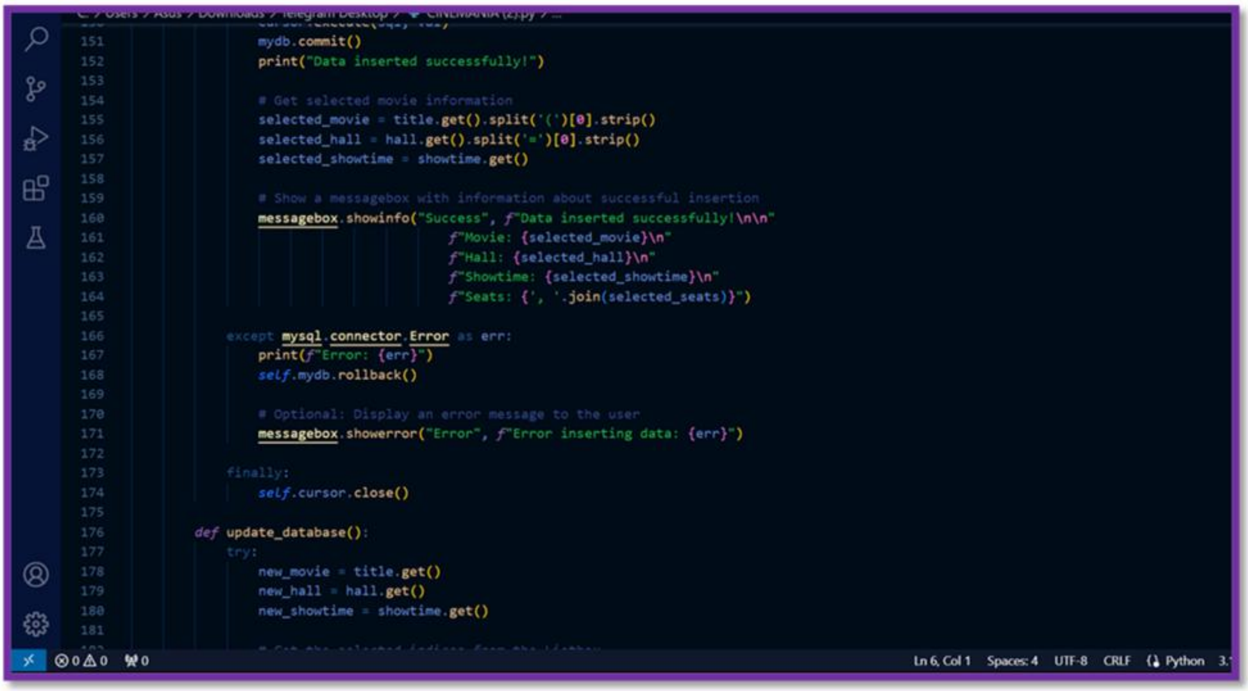
```
        # Create movie data
        data = [["Spider Man 3", "Action & Adventure", "2D", "10AM - 12PM", "RM15"],
            ["Spider Man 3", "Action & Adventure", "Deluxe", "2PM - 4PM", "RM20"],
            ["Spider Man 3", "Action & Adventure", "IMAX", "6PM - 8PM", "RM25"],
            ["Spider Man 3", "Action & Adventure", "Family Session", "8PM - 10PM", "RM30"],
            ["Baymax: The Movie", "Action & Comedy", "2D", "10AM - 12PM", "RM15"],
            ["Baymax: The Movie", "Action & Comedy", "Deluxe", "2PM - 4PM", "RM20"],
            ["Baymax: The Movie", "Action & Comedy", "IMAX", "6PM - 8PM", "RM25"],
            ["Baymax: The Movie", "Action & Comedy", "Family Session", "8PM - 10PM", "RM30"],
            ["Boboiboy The Movie", "Action & Comedy", "2D", "10AM - 12PM", "RM15"],
            ["Boboiboy The Movie", "Action & Comedy", "Deluxe", "2PM - 4PM", "RM20"],
            ["Boboiboy The Movie", "Action & Comedy", "IMAX", "6PM - 8PM", "RM25"],
            ["Boboiboy The Movie", "Action & Comedy", "Family Session", "8PM - 10PM", "RM30"],
            ["The Nun", "Horror", "2D", "10AM - 12PM", "RM15"],
            ["The Nun", "Horror", "Deluxe", "2PM - 4PM", "RM20"],
            ["The Nun", "Horror", "IMAX", "6PM - 8PM", "RM25"],
            ["The Nun", "Horror", "Family Session", "8PM - 1OPM", "RM30"],
            ["Titanic", "Romance", "2D", "10AM - 12PM", "RM15"],
            ["Titanic", "Romance", "Deluxe", "2pm - 4PM", "RM20"],
            ["Titanic", "Romance", "IMAX", "6PM - 8PM", "RM25"],
            ["Titanic", "Romance", "Family Session", "8PM - 10PM", "RM30"]]

        # Define header names
        col_names = ["Movie", "Genre", "Hall", "Showtime", "Price"]

        # Display table
        dataframe = tk.Frame(window, bd=8, bg='grey1', relief='groove')
        dataframe.grid(row=0, column=0, ipadx=20)
        data_Label = tk.Label(dataframe, text='Movie Table', font='Cambria 15 bold', bg='red3', relief='raise')
        data_Label.grid(row=0, column=0, padx=10)
```



```
        # User information
        detailsframe = tk.Frame(window, bd=8, bg='grey1',relief='groove')
        detailsframe.grid(row=2, column=0)
        title_Label = tk.Label(detailsframe, text='Title', bg='red1', relief='ridge')
        title_Label.grid(row=0, column=0, padx=10, pady=10)
        title = ttk.Combobox(detailsframe,
                             values=['Spider Man 3  (Action & Adventure)', 'Baymax: The Movie (Action & Comedy)',
                                     'Boboiboy The Movie (Action & Comedy)', 'The Nun (Horror)', 'Titanic (Romance)'])
        title.grid(row=1, column=0, padx=50, ipadx=55)

        hall_Label = tk.Label(detailsframe, text='Hall', bg='red1', relief='ridge')
        hall_Label.grid(row=0, column=1, padx=5, pady=3)
        hall = ttk.Combobox(detailsframe, values=["2D (classic) = RM15", "Deluxe =RM20", "IMAX (Grand Theatre)= RM25",
                                                  "Family Session=RM30"])
        hall.grid(row=1, column=1, padx=5, ipadx=30)

        showtime_Label = tk.Label(detailsframe, text='Showtime', bg='red1', relief ='ridge')
        showtime_Label.grid(row=2, column=0, padx=20, pady=20)
        showtime = ttk.Combobox(detailsframe, values=['10AM - 12PM', '2PM - 4PM', '6PM - 8PM', '8PM - 10PM'])
        showtime.grid(row=3, column=0, ipadx=55)

        seat_Label = tk.Label(detailsframe, text='Seat No.',bg='red1', relief='ridge')
        seat_Label.grid(row=2, column=1, padx=10, pady=10)
        seat_frame = tk.Frame(detailsframe, bd=8)
        seat_frame.grid(row=3, column=1, rowspan=5)
        seat = tk.Listbox(seat_frame, selectmode='multiple', font='Courier 10', bg='white')
        seat.pack(side='left')
        seat_scrollbar = tk.Scrollbar(seat_frame,bg='red1',command=seat.yview)
        seat_scrollbar.pack(side='right', fill='y')
        seat.insert(1, "1A", "2A", "3A", "4A", "5A", "6A", "7A", "8A", "9A", "10A", "1B", "2B", "3B", "4B", "5B", "6B", "7B",
                    "8B", "9B", "10B",
```

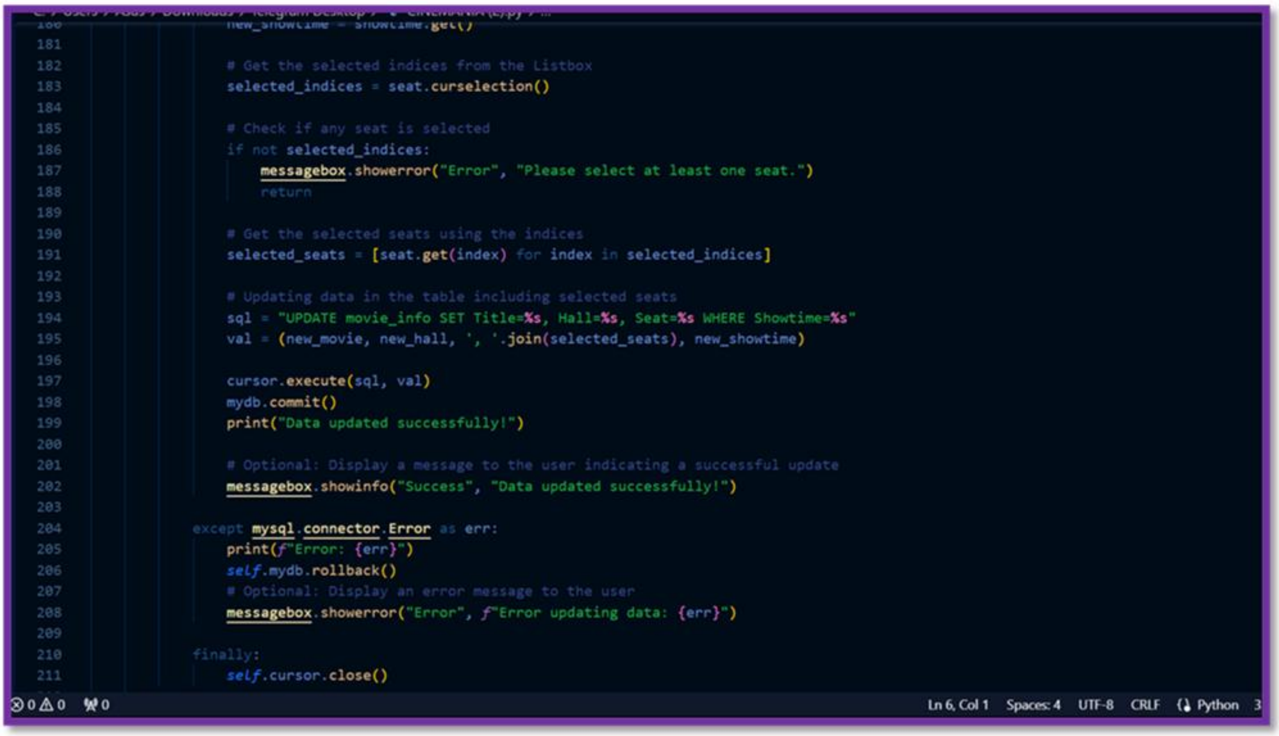


```python
                    "8B", "9B", "10B",
                    "1C", "2C", "3C", "4C", "5C", "6C", "7C", "8C", "9C", "10C", "1D", "2D", "3D", "4D", "5D", "6D", "7D", "8D",
                    "9D", "10D",
                    "1E", "2E", "3E", "4E", "5E", "6E", "7E", "8E", "9E", "10E", "1F", "2F", "3F", "4F", "5F", "6F", "7F", "8F",
                    "9F", "10F")

        # Create a cursor object to interact with the database
        self.cursor = self.mydb.cursor()

        def enter_data():
            try:
                # Get the selected indices from the Listbox
                selected_indices = seat.curselection()

                # Check if any seat is selected
                if not selected_indices:
                    messagebox.showerror("Error", "Please select at least one seat.")
                    return

                # Get the selected seats using the indices
                selected_seats = [seat.get(index) for index in selected_indices]

                # Inserting data into a table
                sql = "INSERT INTO movie_info (Title, Hall, Showtime, Seat) VALUES (%s, %s, %s, %s)"
                val = (title.get(), hall.get(), showtime.get(), ', '.join(selected_seats))

                cursor.execute(sql, val)
                mydb.commit()
                print("Data inserted successfully!")

                # Get selected movie information
                selected_movie = title.get().split('(')[0].strip()
```

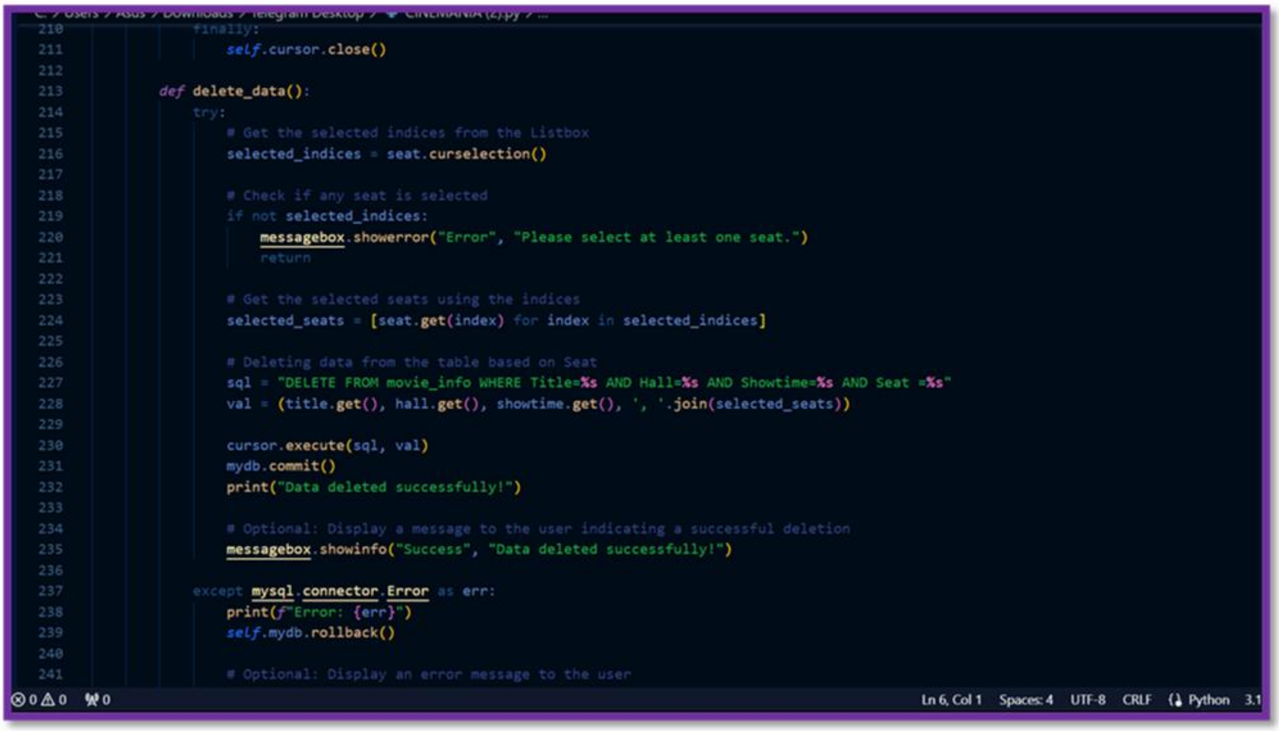


C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > ...
⊗ 0 ⚠ 0 0
Ln 6, Col 1 Spaces: 4 UTF-8 CRLF Python 3.

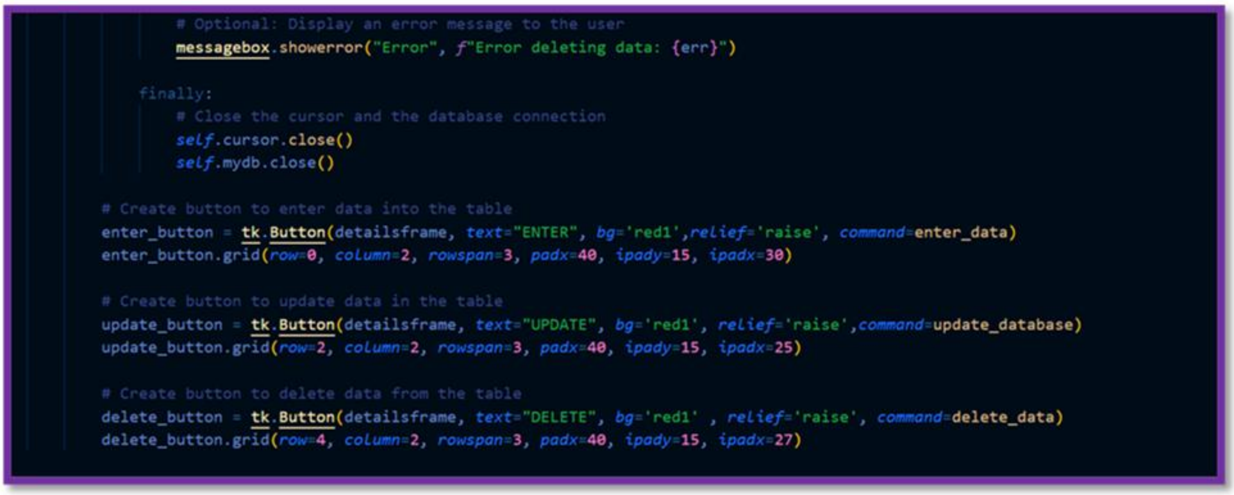
```python
                mydb.commit()
                print("Data inserted successfully!")

                # Get selected movie information
                selected_movie = title.get().split('(')[0].strip()
                selected_hall = hall.get().split('=')[0].strip()
                selected_showtime = showtime.get()

                # Show a messagebox with information about successful insertion
                messagebox.showinfo("Success", f"Data inserted successfully!\n\n"
                                        f"Movie: {selected_movie}\n"
                                        f"Hall: {selected_hall}\n"
                                        f"Showtime: {selected_showtime}\n"
                                        f"Seats: {', '.join(selected_seats)}")

            except mysql.connector.Error as err:
                print(f"Error: {err}")
                self.mydb.rollback()

                # Optional: Display an error message to the user
                messagebox.showerror("Error", f"Error inserting data: {err}")

            finally:
                self.cursor.close()

        def update_database():
            try:
                new_movie = title.get()
                new_hall = hall.get()
                new_showtime = showtime.get()
```



```
                new_showtime = showtime.get()

                # Get the selected indices from the Listbox
                selected_indices = seat.curselection()

                # Check if any seat is selected
                if not selected_indices:
                    messagebox.showerror("Error", "Please select at least one seat.")
                    return

                # Get the selected seats using the indices
                selected_seats = [seat.get(index) for index in selected_indices]

                # Updating data in the table including selected seats
                sql = "UPDATE movie_info SET Title=%s, Hall=%s, Seat=%s WHERE Showtime=%s"
                val = (new_movie, new_hall, ', '.join(selected_seats), new_showtime)

                cursor.execute(sql, val)
                mydb.commit()
                print("Data updated successfully!")

                # Optional: Display a message to the user indicating a successful update
                messagebox.showinfo("Success", "Data updated successfully!")

            except mysql.connector.Error as err:
                print(f"Error: {err}")
                self.mydb.rollback()
                # Optional: Display an error message to the user
                messagebox.showerror("Error", f"Error updating data: {err}")

            finally:
                self.cursor.close()
```



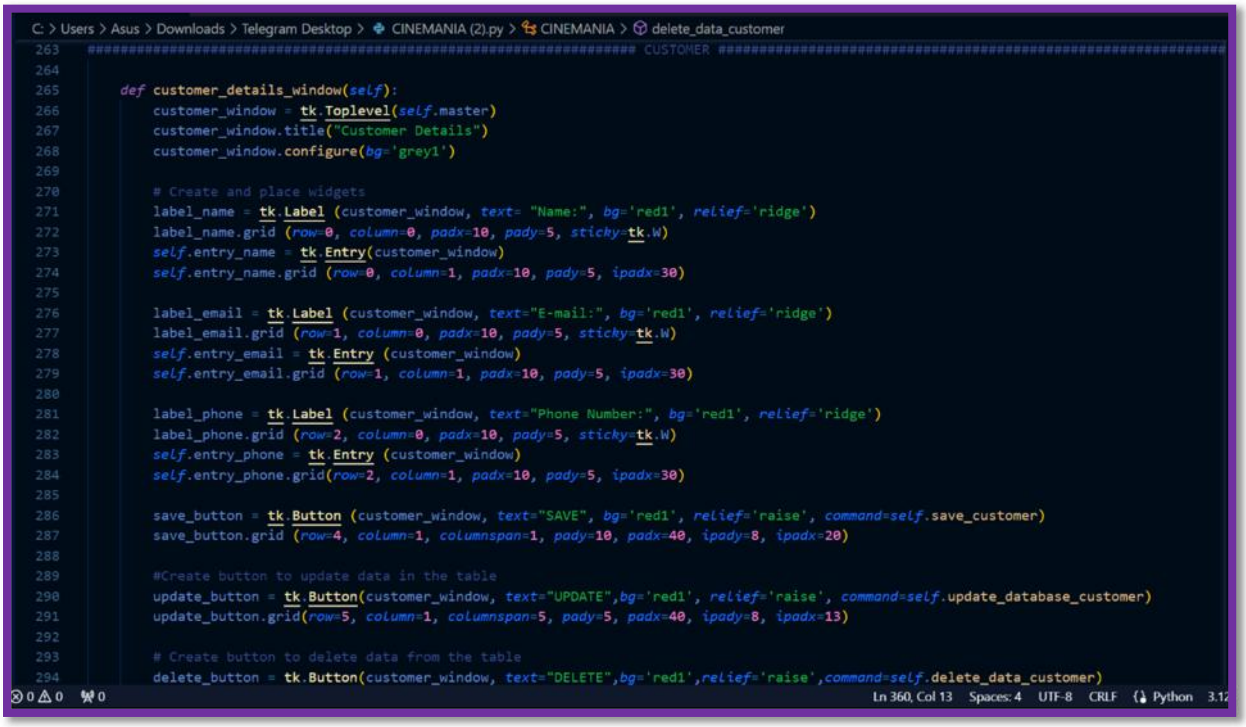
```
            finally:
                self.cursor.close()

        def delete_data():
            try:
                # Get the selected indices from the Listbox
                selected_indices = seat.curselection()

                # Check if any seat is selected
                if not selected_indices:
                    messagebox.showerror("Error", "Please select at least one seat.")
                    return

                # Get the selected seats using the indices
                selected_seats = [seat.get(index) for index in selected_indices]

                # Deleting data from the table based on Seat
                sql = "DELETE FROM movie_info WHERE Title=%s AND Hall=%s AND Showtime=%s AND Seat =%s"
                val = (title.get(), hall.get(), showtime.get(), ', '.join(selected_seats))

                cursor.execute(sql, val)
                mydb.commit()
                print("Data deleted successfully!")

                # Optional: Display a message to the user indicating a successful deletion
                messagebox.showinfo("Success", "Data deleted successfully!")

            except mysql.connector.Error as err:
                print(f"Error: {err}")
                self.mydb.rollback()

                # Optional: Display an error message to the user
```



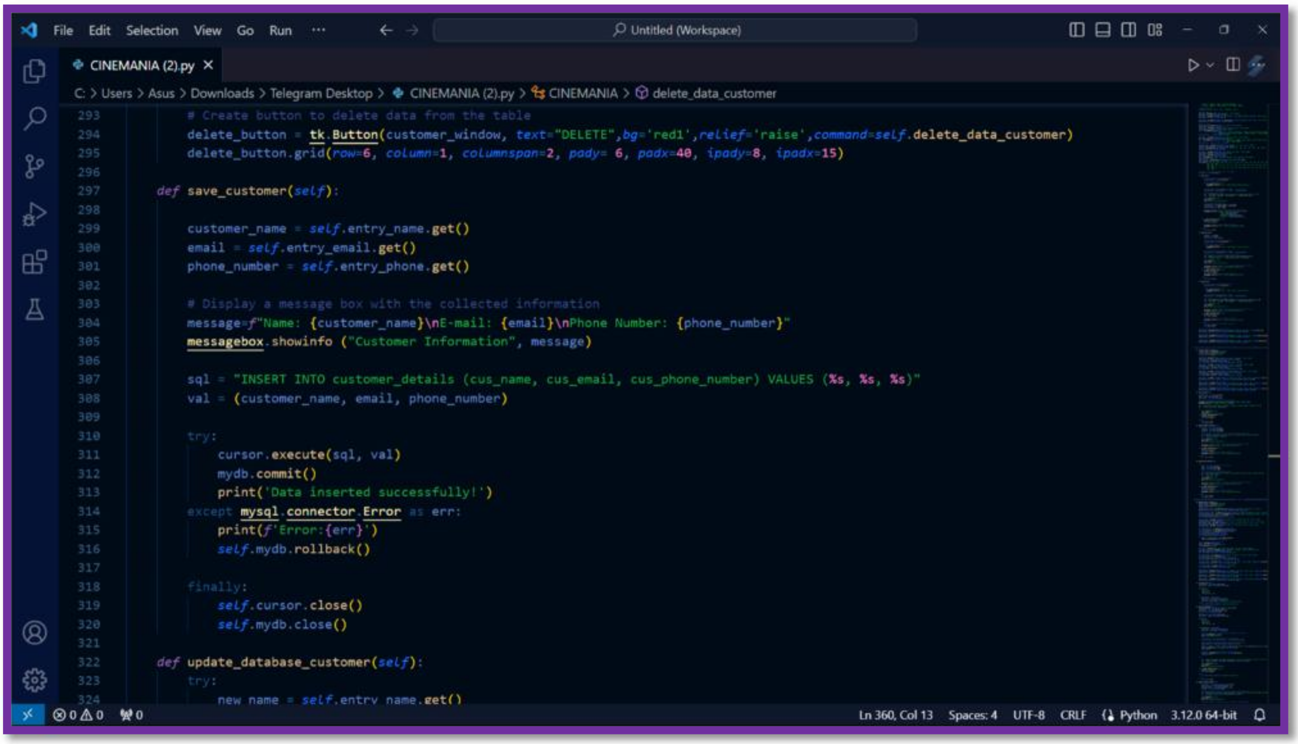
```python
                # Optional: Display an error message to the user
                messagebox.showerror("Error", f"Error deleting data: {err}")

            finally:
                # Close the cursor and the database connection
                self.cursor.close()
                self.mydb.close()

        # Create button to enter data into the table
        enter_button = tk.Button(detailsframe, text="ENTER", bg='red1',relief='raise', command=enter_data)
        enter_button.grid(row=0, column=2, rowspan=3, padx=40, ipady=15, ipadx=30)

        # Create button to update data in the table
        update_button = tk.Button(detailsframe, text="UPDATE", bg='red1', relief='raise',command=update_database)
        update_button.grid(row=2, column=2, rowspan=3, padx=40, ipady=15, ipadx=25)

        # Create button to delete data from the table
        delete_button = tk.Button(detailsframe, text="DELETE", bg='red1' , relief='raise', command=delete_data)
        delete_button.grid(row=4, column=2, rowspan=3, padx=40, ipady=15, ipadx=27)
```

6.4 CUSTOMER INFO CODING

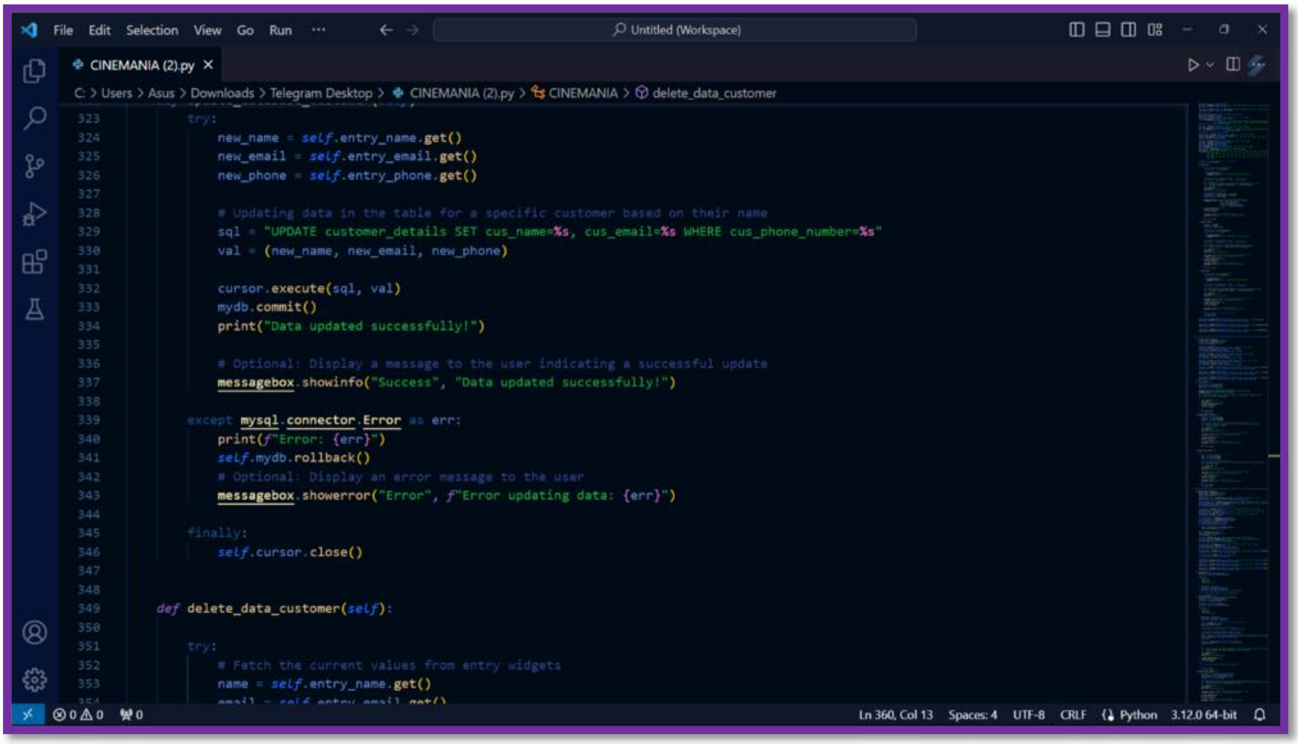
For customer info, user need to fill in the requirement boxes that include “Name”, “Email”, “Phone number”. For phone number, user need to fill in their phone number and start with the number “6”. For example, like, 60136789543. Users also can save, update and delete their data if they want to. Once they already fill in the requirement, it will show the “Data inserted successfully” message. If user need to update the data, they can press the update button, and if successfully it will show the “Data updated successfully!” message. If users need to delete the data, they can press the delete button and if successfully it will show the “Data deleted successfully!”.


C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer

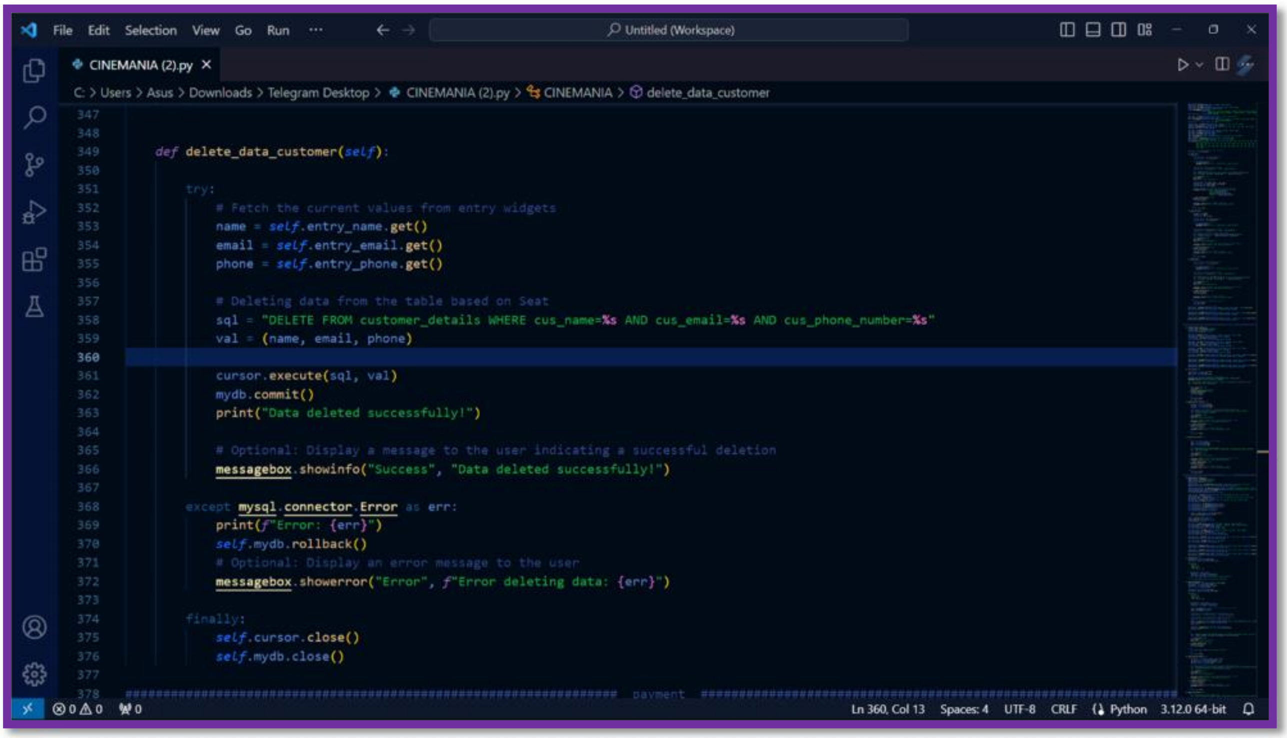
```python
################################################################## CUSTOMER ##################################################################

    def customer_details_window(self):
        customer_window = tk.Toplevel(self.master)
        customer_window.title("Customer Details")
        customer_window.configure(bg='grey1')

        # Create and place widgets
        label_name = tk.Label (customer_window, text= "Name:", bg='red1', relief='ridge')
        label_name.grid (row=0, column=0, padx=10, pady=5, sticky=tk.W)
        self.entry_name = tk.Entry(customer_window)
        self.entry_name.grid (row=0, column=1, padx=10, pady=5, ipadx=30)

        label_email = tk.Label (customer_window, text="E-mail:", bg='red1', relief='ridge')
        label_email.grid (row=1, column=0, padx=10, pady=5, sticky=tk.W)
        self.entry_email = tk.Entry (customer_window)
        self.entry_email.grid (row=1, column=1, padx=10, pady=5, ipadx=30)

        label_phone = tk.Label (customer_window, text="Phone Number:", bg='red1', relief='ridge')
        label_phone.grid (row=2, column=0, padx=10, pady=5, sticky=tk.W)
        self.entry_phone = tk.Entry (customer_window)
        self.entry_phone.grid(row=2, column=1, padx=10, pady=5, ipadx=30)

        save_button = tk.Button (customer_window, text="SAVE", bg='red1', relief='raise', command=self.save_customer)
        save_button.grid (row=4, column=1, columnspan=1, pady=10, padx=40, ipady=8, ipadx=20)

        #Create button to update data in the table
        update_button = tk.Button(customer_window, text="UPDATE",bg='red1', relief='raise', command=self.update_database_customer)
        update_button.grid(row=5, column=1, columnspan=5, pady=5, padx=40, ipady=8, ipadx=13)

        # Create button to delete data from the table
        delete_button = tk.Button(customer_window, text="DELETE",bg='red1',relief='raise',command=self.delete_data_customer)
```


File Edit Selection View Go Run ···
Untitled (Workspace)
CINEMANIA (2).py
C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer
⊗ 0 ⚠ 0 0
Ln 360, Col 13 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit

```python
        # Create button to delete data from the table
        delete_button = tk.Button(customer_window, text="DELETE",bg='red1',relief='raise',command=self.delete_data_customer)
        delete_button.grid(row=6, column=1, columnspan=2, pady= 6, padx=40, ipady=8, ipadx=15)

    def save_customer(self):

        customer_name = self.entry_name.get()
        email = self.entry_email.get()
        phone_number = self.entry_phone.get()

        # Display a message box with the collected information
        message=f"Name: {customer_name}\nE-mail: {email}\nPhone Number: {phone_number}"
        messagebox.showinfo ("Customer Information", message)

        sql = "INSERT INTO customer_details (cus_name, cus_email, cus_phone_number) VALUES (%s, %s, %s)"
        val = (customer_name, email, phone_number)

        try:
            cursor.execute(sql, val)
            mydb.commit()
            print('Data inserted successfully!')
        except mysql.connector.Error as err:
            print(f'Error:{err}')
            self.mydb.rollback()

        finally:
            self.cursor.close()
            self.mydb.close()

    def update_database_customer(self):
        try:
            new_name = self.entry_name.get()
```


File Edit Selection View Go Run ···
Untitled (Workspace)
CINEMANIA (2).py
C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer
⊗ 0 ⚠ 0 0
Ln 360, Col 13 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit

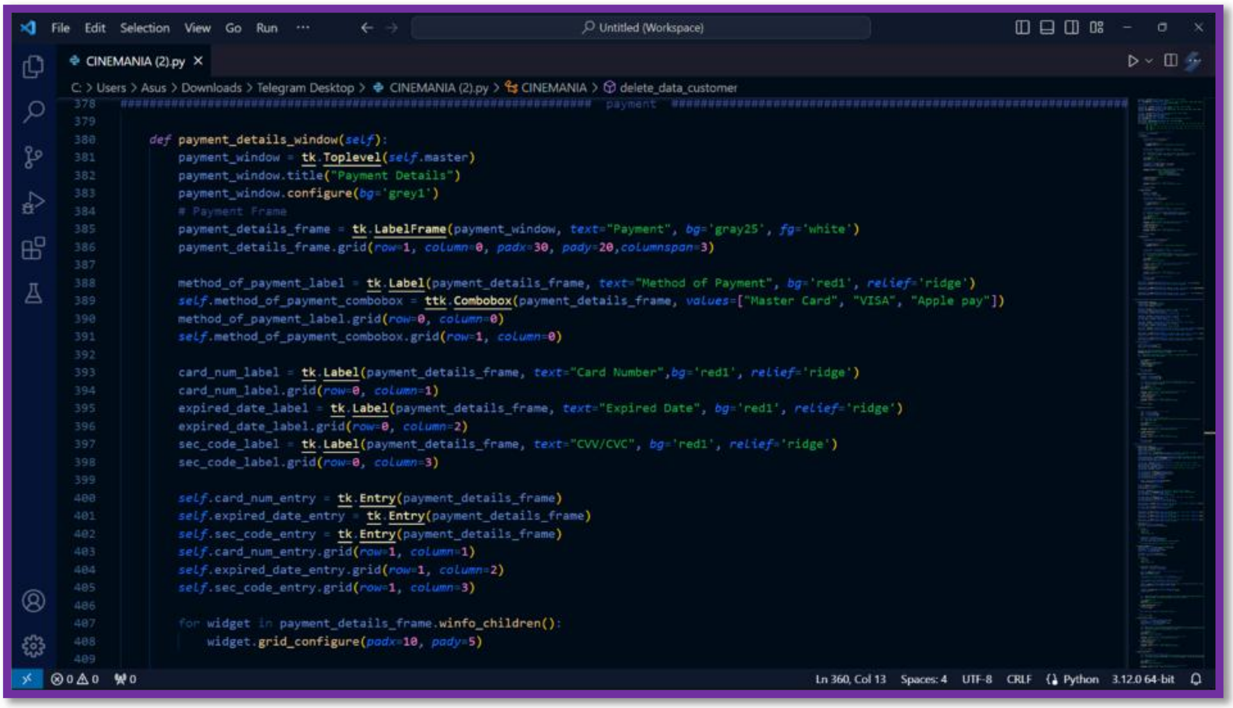
```python
        try:
            new_name = self.entry_name.get()
            new_email = self.entry_email.get()
            new_phone = self.entry_phone.get()

            # Updating data in the table for a specific customer based on their name
            sql = "UPDATE customer_details SET cus_name=%s, cus_email=%s WHERE cus_phone_number=%s"
            val = (new_name, new_email, new_phone)

            cursor.execute(sql, val)
            mydb.commit()
            print("Data updated successfully!")

            # Optional: Display a message to the user indicating a successful update
            messagebox.showinfo("Success", "Data updated successfully!")

        except mysql.connector.Error as err:
            print(f"Error: {err}")
            self.mydb.rollback()
            # Optional: Display an error message to the user
            messagebox.showerror("Error", f"Error updating data: {err}")

        finally:
            self.cursor.close()


    def delete_data_customer(self):

        try:
            # Fetch the current values from entry widgets
            name = self.entry_name.get()
            email = self.entry_email.get()
```



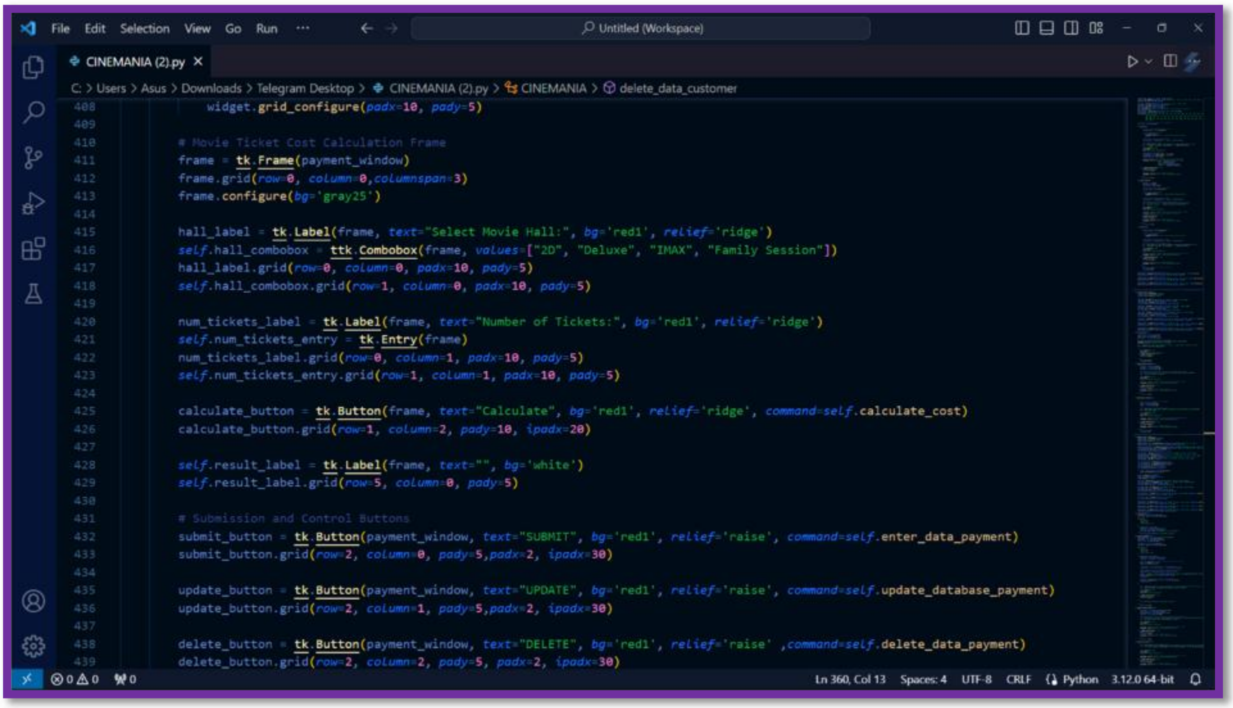
```python


    def delete_data_customer(self):

        try:
            # Fetch the current values from entry widgets
            name = self.entry_name.get()
            email = self.entry_email.get()
            phone = self.entry_phone.get()

            # Deleting data from the table based on Seat
            sql = "DELETE FROM customer_details WHERE cus_name=%s AND cus_email=%s AND cus_phone_number=%s"
            val = (name, email, phone)

            cursor.execute(sql, val)
            mydb.commit()
            print("Data deleted successfully!")

            # Optional: Display a message to the user indicating a successful deletion
            messagebox.showinfo("Success", "Data deleted successfully!")

        except mysql.connector.Error as err:
            print(f"Error: {err}")
            self.mydb.rollback()
            # Optional: Display an error message to the user
            messagebox.showerror("Error", f"Error deleting data: {err}")

        finally:
            self.cursor.close()
            self.mydb.close()

################################################################  payment  ##############################################################
```

6.5 PAYMENT DETAILS

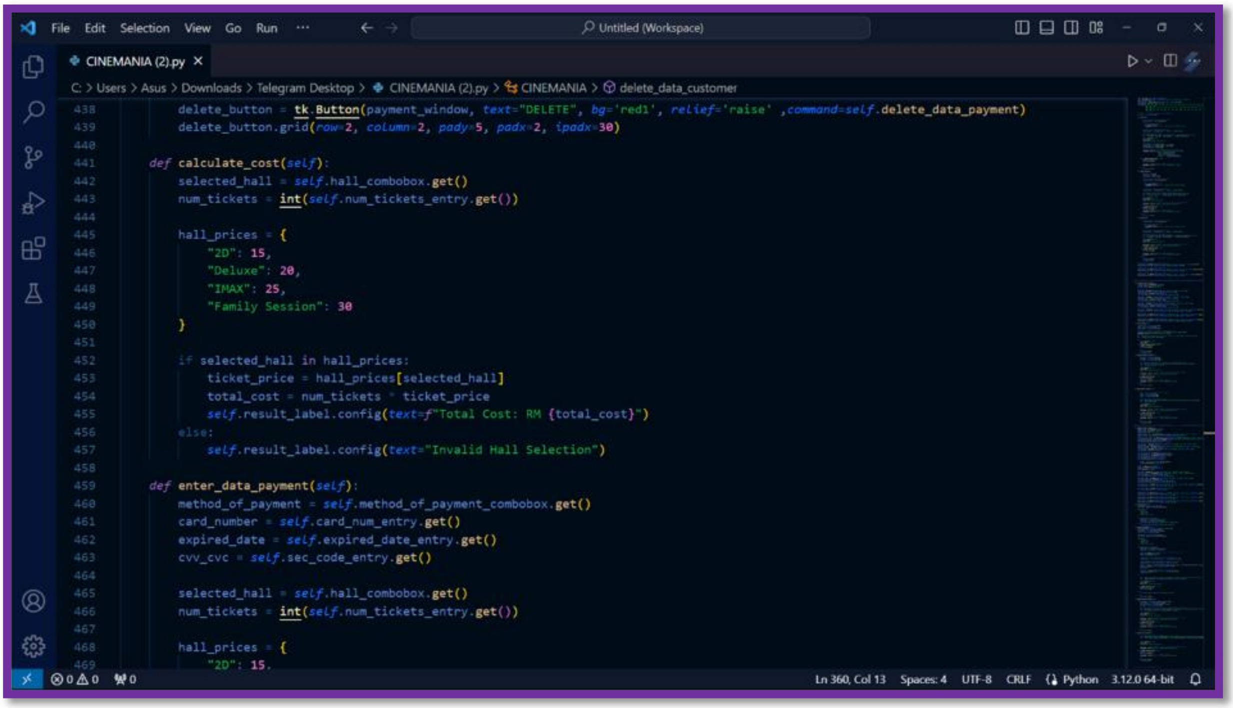
In this payment details part, the users need to select their movie hall that hey choose earlier and also the number of tickets. Because in this part it will calculates the total cost and the user need to make a payment. After that, users need to make a payment by choosing methods of payment. This including, MasterCard, Visa and Apple pay. After choosing the type of credit card, the users need to fill in the requirement boxes such as, card number, expired date and cvv/cvc to successfully booking the tickets.


File Edit Selection View Go Run ···
Untitled (Workspace)
CINEMANIA (2).py
C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer
⊗ 0 ⚠ 0 0
Ln 360, Col 13 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit

```
    ################################################################### payment ###################################################################

    def payment_details_window(self):
        payment_window = tk.Toplevel(self.master)
        payment_window.title("Payment Details")
        payment_window.configure(bg='grey1')
        # Payment Frame
        payment_details_frame = tk.LabelFrame(payment_window, text="Payment", bg='gray25', fg='white')
        payment_details_frame.grid(row=1, column=0, padx=30, pady=20,columnspan=3)

        method_of_payment_label = tk.Label(payment_details_frame, text="Method of Payment", bg='red1', relief='ridge')
        self.method_of_payment_combobox = ttk.Combobox(payment_details_frame, values=["Master Card", "VISA", "Apple pay"])
        method_of_payment_label.grid(row=0, column=0)
        self.method_of_payment_combobox.grid(row=1, column=0)

        card_num_label = tk.Label(payment_details_frame, text="Card Number",bg='red1', relief='ridge')
        card_num_label.grid(row=0, column=1)
        expired_date_label = tk.Label(payment_details_frame, text="Expired Date", bg='red1', relief='ridge')
        expired_date_label.grid(row=0, column=2)
        sec_code_label = tk.Label(payment_details_frame, text="CVV/CVC", bg='red1', relief='ridge')
        sec_code_label.grid(row=0, column=3)

        self.card_num_entry = tk.Entry(payment_details_frame)
        self.expired_date_entry = tk.Entry(payment_details_frame)
        self.sec_code_entry = tk.Entry(payment_details_frame)
        self.card_num_entry.grid(row=1, column=1)
        self.expired_date_entry.grid(row=1, column=2)
        self.sec_code_entry.grid(row=1, column=3)

        for widget in payment_details_frame.winfo_children():
            widget.grid_configure(padx=10, pady=5)

```

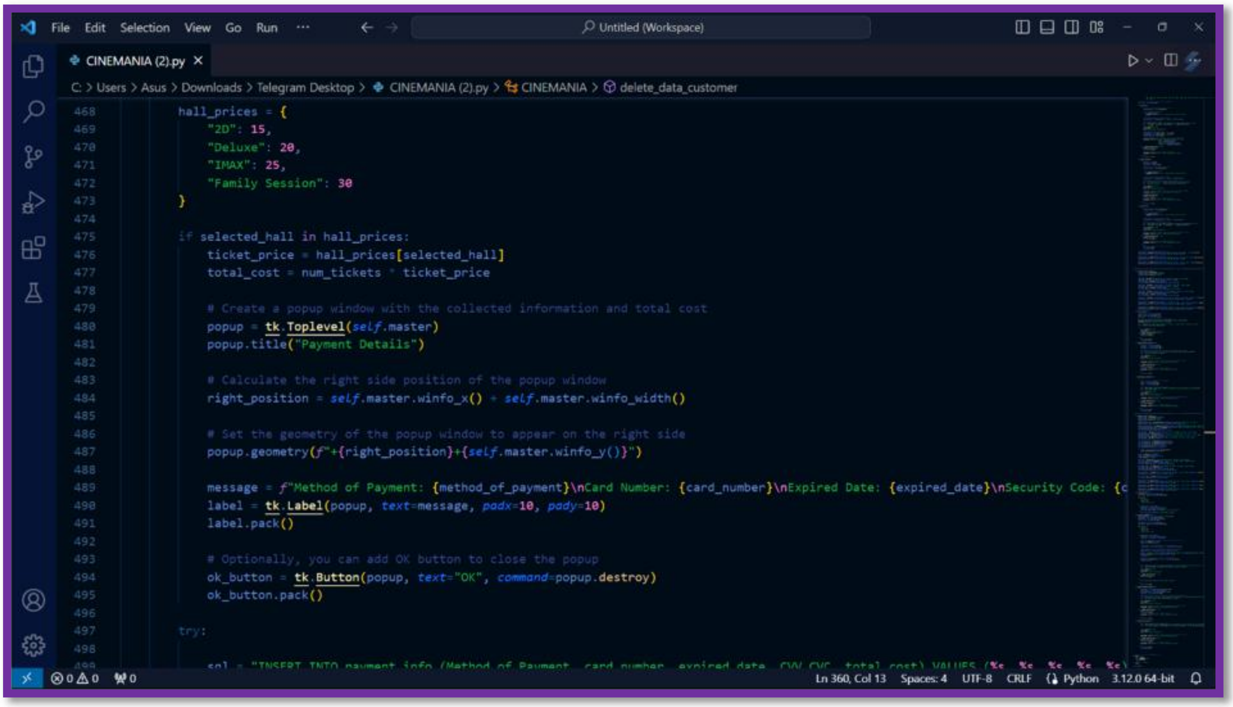

File Edit Selection View Go Run ···
Untitled (Workspace)
CINEMANIA (2).py
C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer
⊗ 0 ⚠ 0 0
Ln 360, Col 13 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit

```
            widget.grid_configure(padx=10, pady=5)

        # Movie Ticket Cost Calculation Frame
        frame = tk.Frame(payment_window)
        frame.grid(row=0, column=0,columnspan=3)
        frame.configure(bg='gray25')

        hall_label = tk.Label(frame, text="Select Movie Hall:", bg='red1', relief='ridge')
        self.hall_combobox = ttk.Combobox(frame, values=["2D", "Deluxe", "IMAX", "Family Session"])
        hall_label.grid(row=0, column=0, padx=10, pady=5)
        self.hall_combobox.grid(row=1, column=0, padx=10, pady=5)

        num_tickets_label = tk.Label(frame, text="Number of Tickets:", bg='red1', relief='ridge')
        self.num_tickets_entry = tk.Entry(frame)
        num_tickets_label.grid(row=0, column=1, padx=10, pady=5)
        self.num_tickets_entry.grid(row=1, column=1, padx=10, pady=5)

        calculate_button = tk.Button(frame, text="Calculate", bg='red1', relief='ridge', command=self.calculate_cost)
        calculate_button.grid(row=1, column=2, pady=10, ipadx=20)

        self.result_label = tk.Label(frame, text="", bg='white')
        self.result_label.grid(row=5, column=0, pady=5)

        # Submission and Control Buttons
        submit_button = tk.Button(payment_window, text="SUBMIT", bg='red1', relief='raise', command=self.enter_data_payment)
        submit_button.grid(row=2, column=0, pady=5,padx=2, ipadx=30)

        update_button = tk.Button(payment_window, text="UPDATE", bg='red1', relief='raise', command=self.update_database_payment)
        update_button.grid(row=2, column=1, pady=5,padx=2, ipadx=30)

        delete_button = tk.Button(payment_window, text="DELETE", bg='red1', relief='raise' ,command=self.delete_data_payment)
        delete_button.grid(row=2, column=2, pady=5, padx=2, ipadx=30)
```



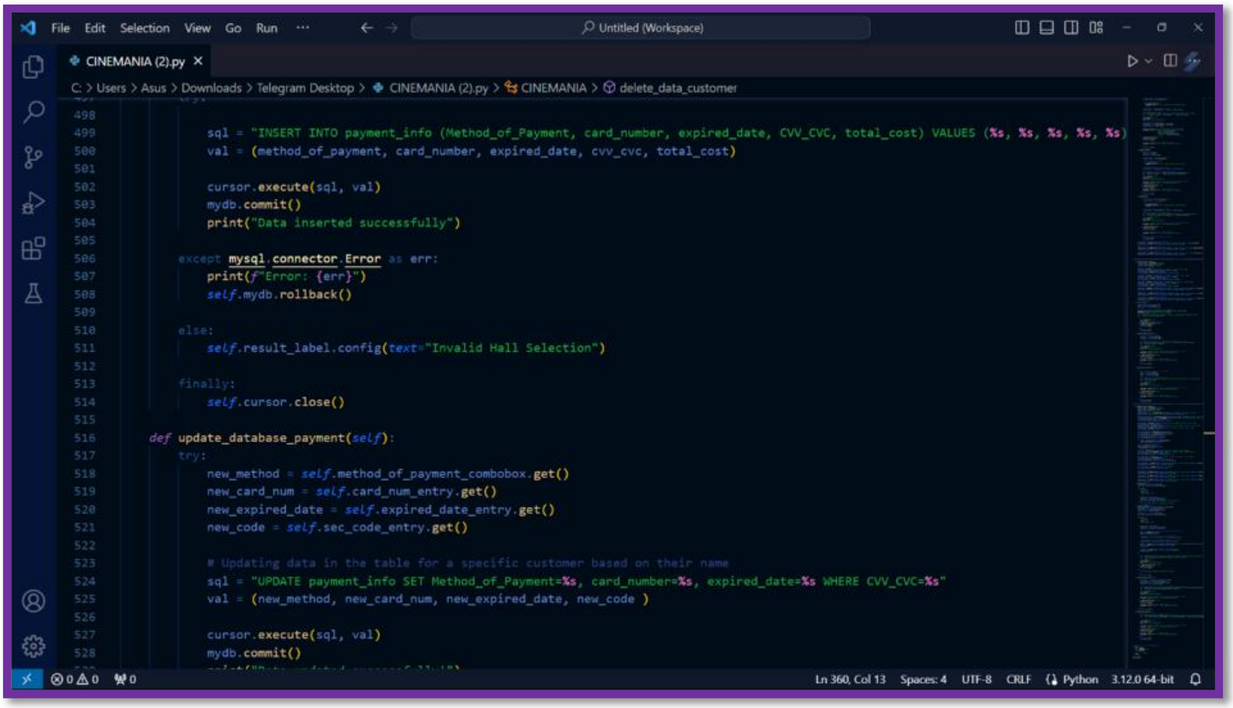
File Edit Selection View Go Run ···
Untitled (Workspace)
CINEMANIA (2).py
C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer
⊗ 0 ⚠ 0 0
Ln 360, Col 13 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit

```
        delete_button = tk.Button(payment_window, text="DELETE", bg='red1', relief='raise' ,command=self.delete_data_payment)
        delete_button.grid(row=2, column=2, pady=5, padx=2, ipadx=30)

    def calculate_cost(self):
        selected_hall = self.hall_combobox.get()
        num_tickets = int(self.num_tickets_entry.get())

        hall_prices = {
            "2D": 15,
            "Deluxe": 20,
            "IMAX": 25,
            "Family Session": 30
        }

        if selected_hall in hall_prices:
            ticket_price = hall_prices[selected_hall]
            total_cost = num_tickets * ticket_price
            self.result_label.config(text=f"Total Cost: RM {total_cost}")
        else:
            self.result_label.config(text="Invalid Hall Selection")

    def enter_data_payment(self):
        method_of_payment = self.method_of_payment_combobox.get()
        card_number = self.card_num_entry.get()
        expired_date = self.expired_date_entry.get()
        cvv_cvc = self.sec_code_entry.get()

        selected_hall = self.hall_combobox.get()
        num_tickets = int(self.num_tickets_entry.get())

        hall_prices = {
            "2D": 15,
```

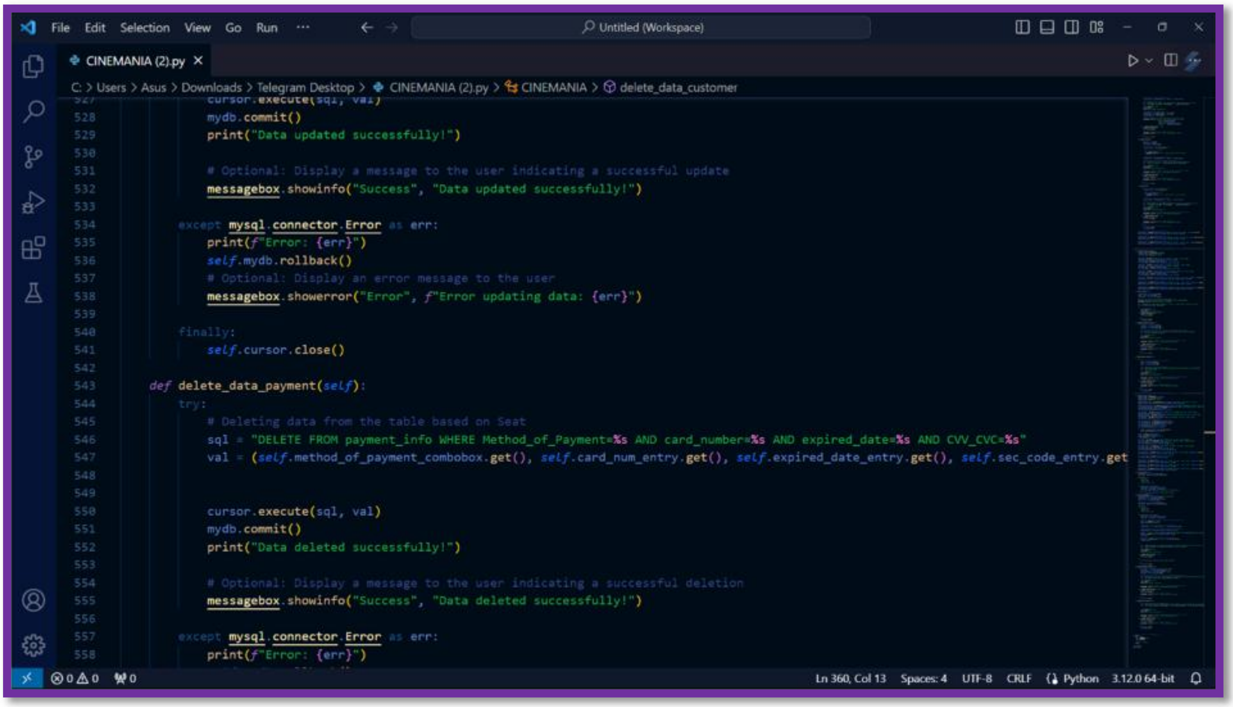


File Edit Selection View Go Run ···
Untitled (Workspace)
CINEMANIA (2).py
C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer
⊗ 0 ⚠ 0 0
Ln 360, Col 13 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit

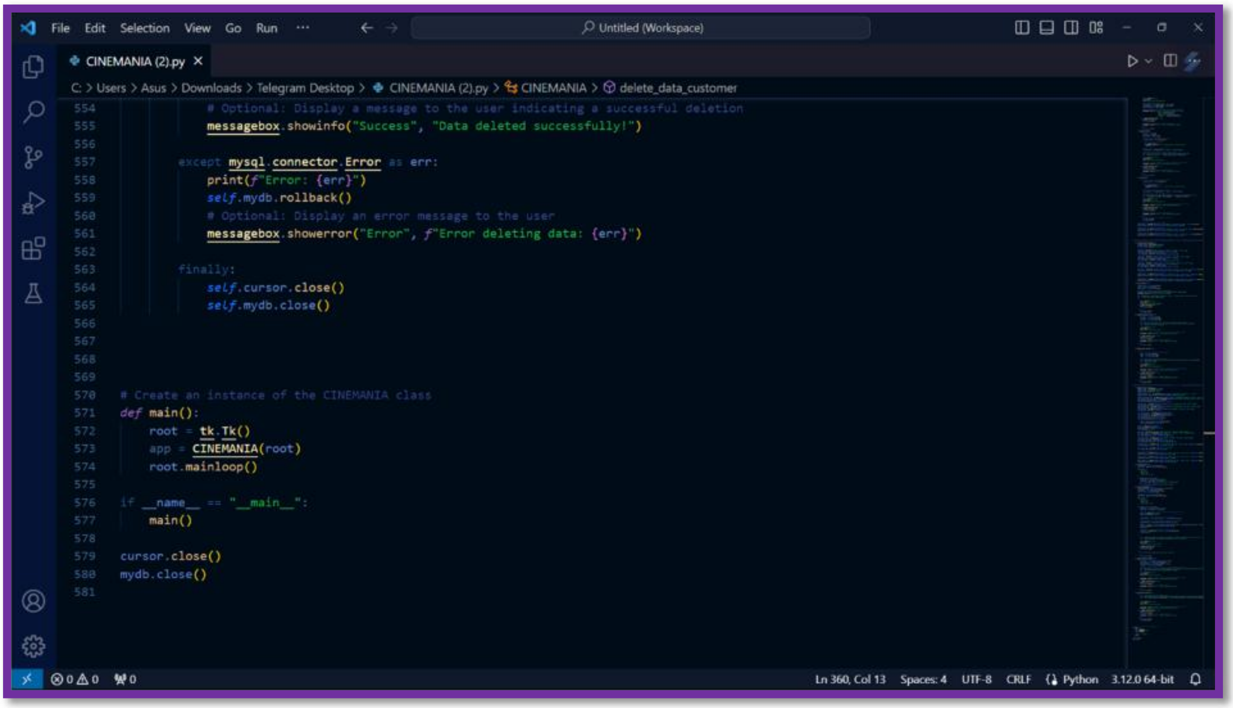
```
        hall_prices = {
            "2D": 15,
            "Deluxe": 20,
            "IMAX": 25,
            "Family Session": 30
        }

        if selected_hall in hall_prices:
            ticket_price = hall_prices[selected_hall]
            total_cost = num_tickets * ticket_price

            # Create a popup window with the collected information and total cost
            popup = tk.Toplevel(self.master)
            popup.title("Payment Details")

            # Calculate the right side position of the popup window
            right_position = self.master.winfo_x() + self.master.winfo_width()

            # Set the geometry of the popup window to appear on the right side
            popup.geometry(f"+{right_position}+{self.master.winfo_y()}")

            message = f"Method of Payment: {method_of_payment}\nCard Number: {card_number}\nExpired Date: {expired_date}\nSecurity Code: {c
            label = tk.Label(popup, text=message, padx=10, pady=10)
            label.pack()

            # Optionally, you can add OK button to close the popup
            ok_button = tk.Button(popup, text="OK", command=popup.destroy)
            ok_button.pack()

        try:
            sql = "INSERT INTO payment_info (Method_of_Payment, card_number, expired_date, CVV_CVC, total_cost) VALUES (%s, %s, %s, %s, %s)
```


File Edit Selection View Go Run ···
Untitled (Workspace)
CINEMANIA (2).py
C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer
⊗ 0 ⚠ 0 0
Ln 360, Col 13 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit

```python

            sql = "INSERT INTO payment_info (Method_of_Payment, card_number, expired_date, CVV_CVC, total_cost) VALUES (%s, %s, %s, %s, %s)
            val = (method_of_payment, card_number, expired_date, cvv_cvc, total_cost)

            cursor.execute(sql, val)
            mydb.commit()
            print("Data inserted successfully")

        except mysql.connector.Error as err:
            print(f"Error: {err}")
            self.mydb.rollback()

        else:
            self.result_label.config(text="Invalid Hall Selection")

        finally:
            self.cursor.close()

    def update_database_payment(self):
        try:
            new_method = self.method_of_payment_combobox.get()
            new_card_num = self.card_num_entry.get()
            new_expired_date = self.expired_date_entry.get()
            new_code = self.sec_code_entry.get()

            # Updating data in the table for a specific customer based on their name
            sql = "UPDATE payment_info SET Method_of_Payment=%s, card_number=%s, expired_date=%s WHERE CVV_CVC=%s"
            val = (new_method, new_card_num, new_expired_date, new_code )

            cursor.execute(sql, val)
            mydb.commit()
```


File Edit Selection View Go Run ···
Untitled (Workspace)
CINEMANIA (2).py
C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer
⊗ 0 ⚠ 0 0
Ln 360, Col 13 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit

```python
            mydb.commit()
            print("Data updated successfully!")

            # Optional: Display a message to the user indicating a successful update
            messagebox.showinfo("Success", "Data updated successfully!")

        except mysql.connector.Error as err:
            print(f"Error: {err}")
            self.mydb.rollback()
            # Optional: Display an error message to the user
            messagebox.showerror("Error", f"Error updating data: {err}")

        finally:
            self.cursor.close()

    def delete_data_payment(self):
        try:
            # Deleting data from the table based on Seat
            sql = "DELETE FROM payment_info WHERE Method_of_Payment=%s AND card_number=%s AND expired_date=%s AND CVV_CVC=%s"
            val = (self.method_of_payment_combobox.get(), self.card_num_entry.get(), self.expired_date_entry.get(), self.sec_code_entry.get


            cursor.execute(sql, val)
            mydb.commit()
            print("Data deleted successfully!")

            # Optional: Display a message to the user indicating a successful deletion
            messagebox.showinfo("Success", "Data deleted successfully!")

        except mysql.connector.Error as err:
            print(f"Error: {err}")
```

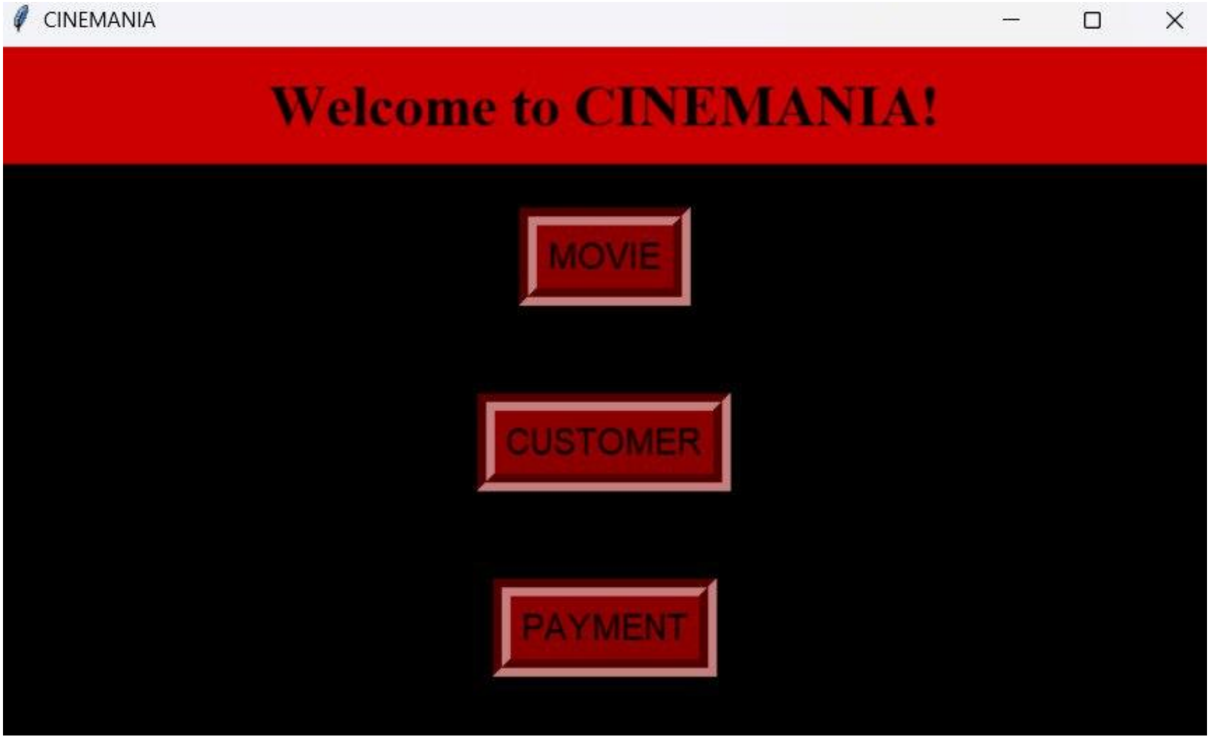

File Edit Selection View Go Run ···
Untitled (Workspace)
CINEMANIA (2).py
C: > Users > Asus > Downloads > Telegram Desktop > CINEMANIA (2).py > CINEMANIA > delete_data_customer
⊗ 0 ⚠ 0 0
Ln 360, Col 13 Spaces: 4 UTF-8 CRLF { } Python 3.12.0 64-bit

```
            # Optional: Display a message to the user indicating a successful deletion
            messagebox.showinfo("Success", "Data deleted successfully!")

        except mysql.connector.Error as err:
            print(f"Error: {err}")
            self.mydb.rollback()
            # Optional: Display an error message to the user
            messagebox.showerror("Error", f"Error deleting data: {err}")

        finally:
            self.cursor.close()
            self.mydb.close()




# Create an instance of the CINEMANIA class
def main():
    root = tk.Tk()
    app = CINEMANIA(root)
    root.mainloop()

if __name__ == "__main__":
    main()

cursor.close()
mydb.close()
```

7.0 GRAPHICAL USER INTERFACES (GUI)

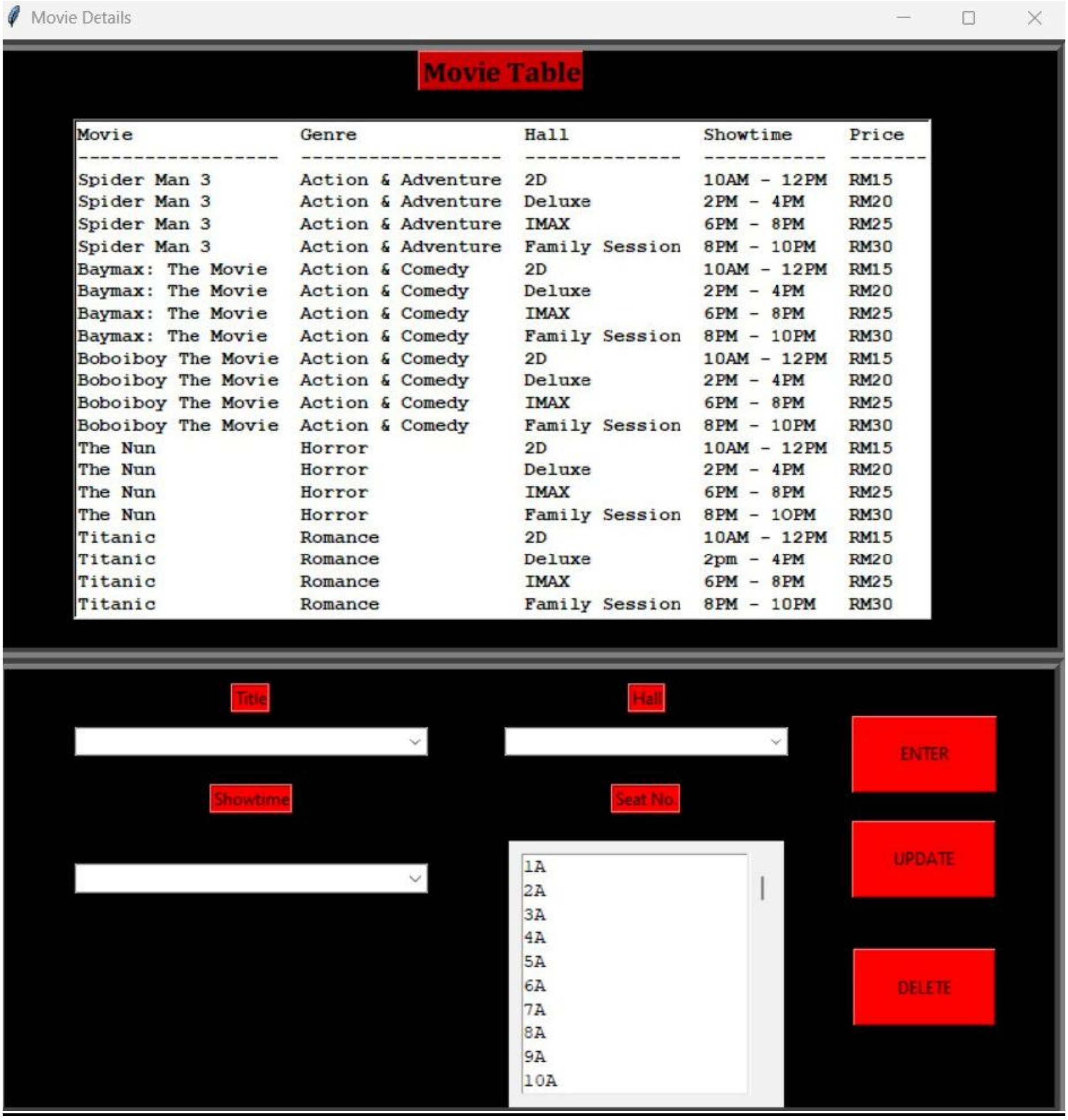
7.1 MAIN WINDOW

This interface let user press the button either “MOVIE”, “CUSTOMER” or “PAYMENT”.



7.2 MOVIE

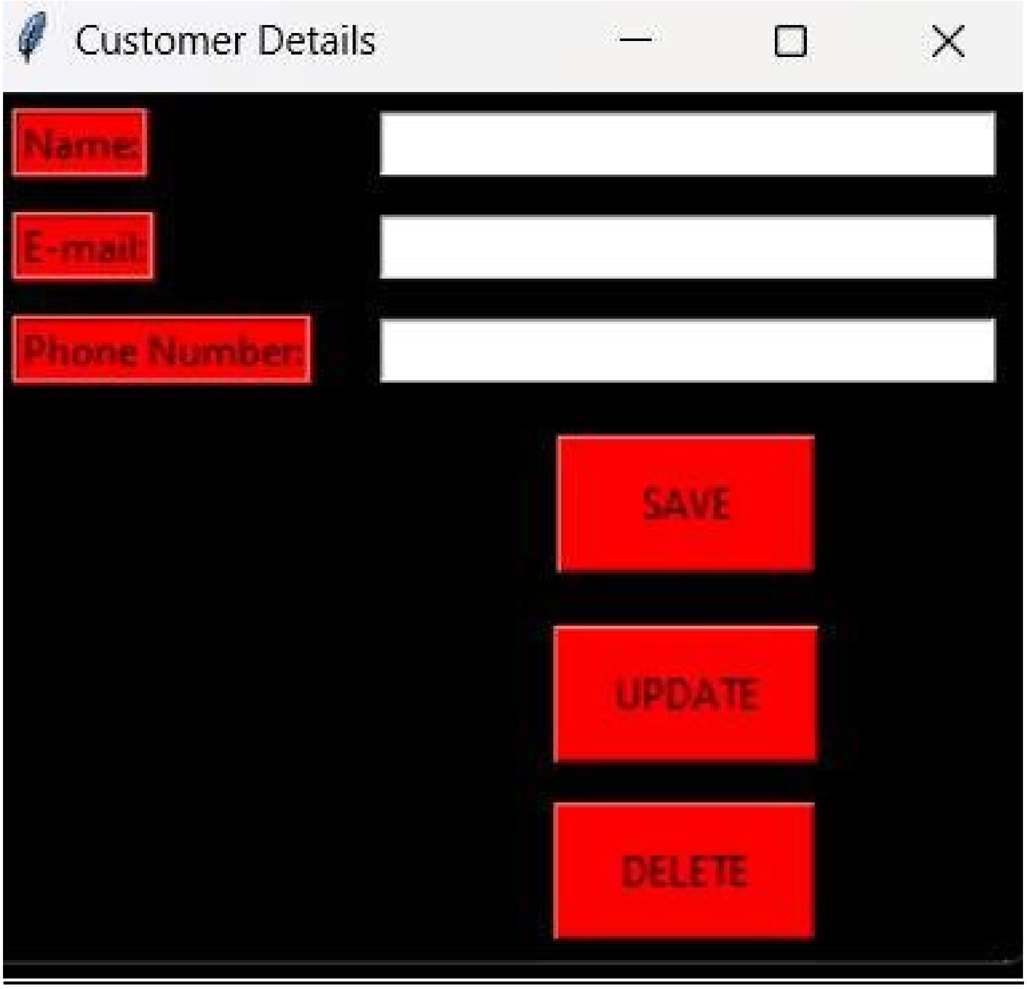
This interface let user decide which movie they want to watch with various type of hall, genre and also the showtime. This interface also include price that based on type of hall.



```
Movie                 Genre                 Hall              Showtime      Price
------------------    ------------------    --------------    -----------   -------
Spider Man 3          Action & Adventure    2D                10AM - 12PM   RM15
Spider Man 3          Action & Adventure    Deluxe            2PM - 4PM     RM20
Spider Man 3          Action & Adventure    IMAX              6PM - 8PM     RM25
Spider Man 3          Action & Adventure    Family Session    8PM - 10PM    RM30
Baymax: The Movie     Action & Comedy       2D                10AM - 12PM   RM15
Baymax: The Movie     Action & Comedy       Deluxe            2PM - 4PM     RM20
Baymax: The Movie     Action & Comedy       IMAX              6PM - 8PM     RM25
Baymax: The Movie     Action & Comedy       Family Session    8PM - 10PM    RM30
Boboiboy The Movie    Action & Comedy       2D                10AM - 12PM   RM15
Boboiboy The Movie    Action & Comedy       Deluxe            2PM - 4PM     RM20
Boboiboy The Movie    Action & Comedy       IMAX              6PM - 8PM     RM25
Boboiboy The Movie    Action & Comedy       Family Session    8PM - 10PM    RM30
The Nun               Horror                2D                10AM - 12PM   RM15
The Nun               Horror                Deluxe            2PM - 4PM     RM20
The Nun               Horror                IMAX              6PM - 8PM     RM25
The Nun               Horror                Family Session    8PM - 10PM    RM30
Titanic               Romance               2D                10AM - 12PM   RM15
Titanic               Romance               Deluxe            2pm - 4PM     RM20
Titanic               Romance               IMAX              6PM - 8PM     RM25
Titanic               Romance               Family Session    8PM - 10PM    RM30
```

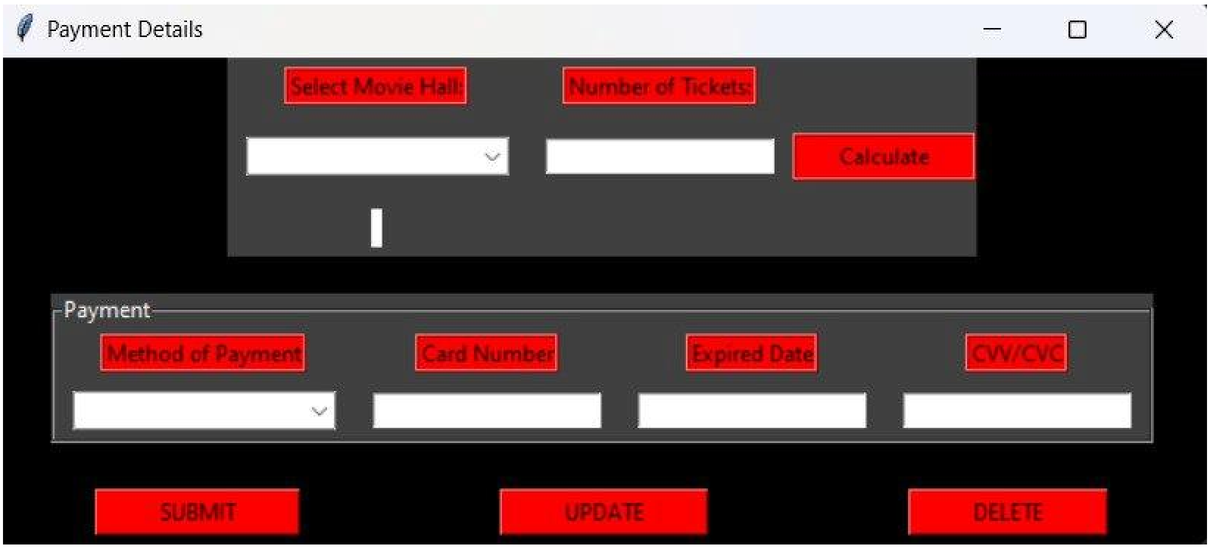
7.3 CUSTOMER

After user already fill in the name, e-mail and phone number box, they can choose either “SAVE”, “UPDATE” OR “DELETE”.



7.4 PAYMENT DETAILS

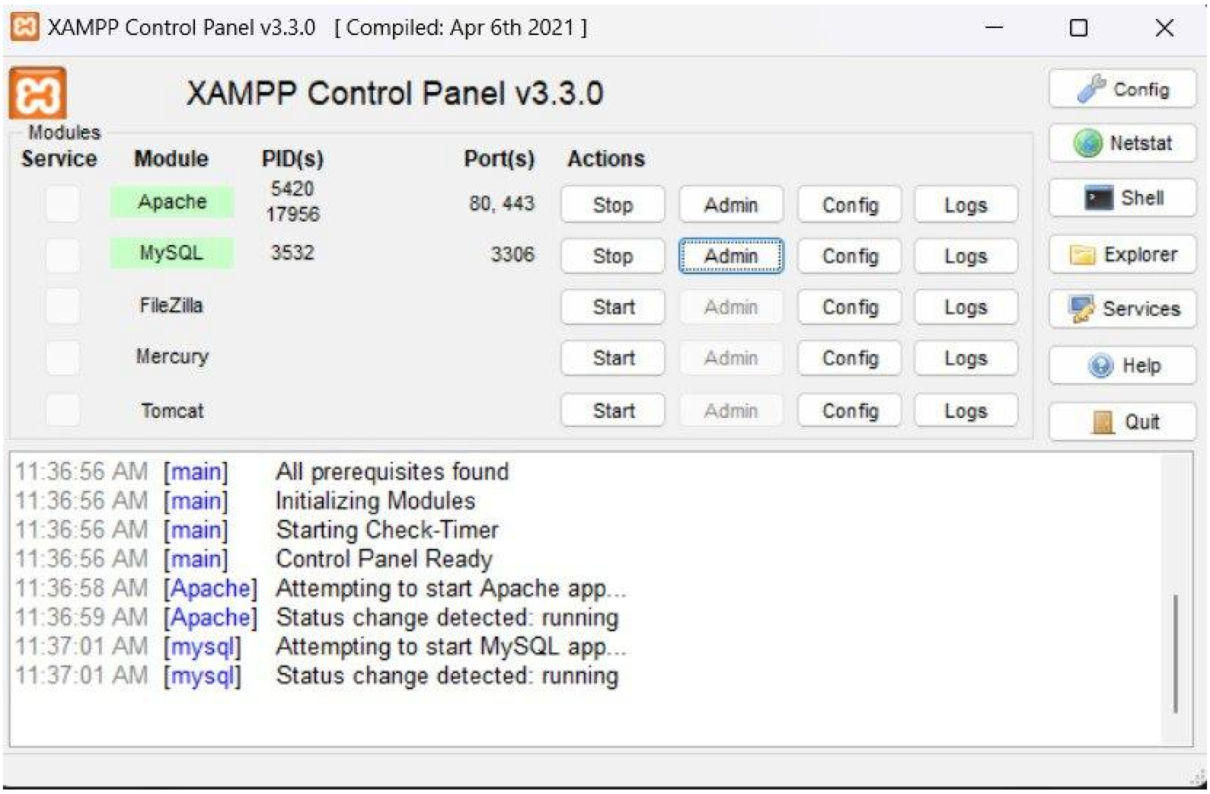
Users have to select movie hall that they choose earlier and number of tickets if they need the tickets for 3pax and so on. It will calculate the total cost that customer need to pay. And they have to choose the payment methods along with card number, expired date and CVV/CVC.



8.0 DATABASE

8.1 XAMPP

This part of XAMPP Control Panel, we must to start the Apache andMySQL also with Admin.

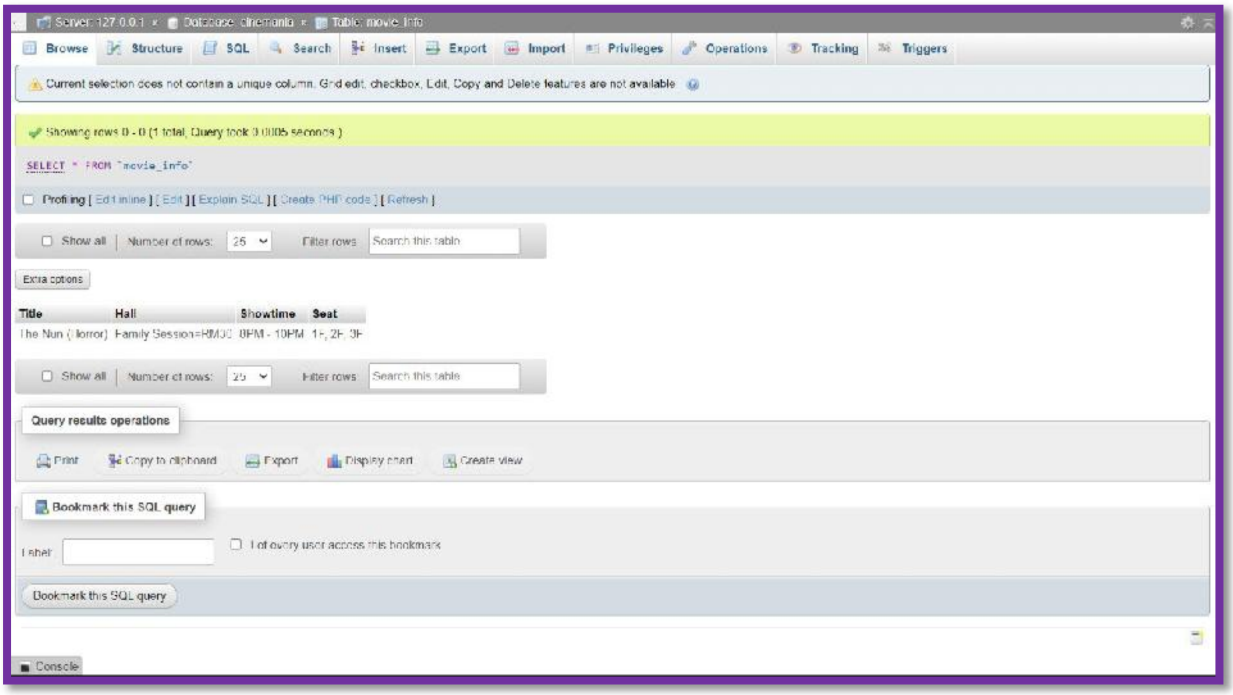

XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]
XAMPP Control Panel v3.3.0
Modules
Config
Netstat
Shell
Explorer
Services
Help
Quit

Service	Module	PID(s)	Port(s)	Actions			
□	Apache	5420 17956	80, 443	Stop	Admin	Config	Logs
□	MySQL	3532	3306	Stop	Admin	Config	Logs
□	FileZilla			Start	Admin	Config	Logs
□	Mercury			Start	Admin	Config	Logs
□	Tomcat			Start	Admin	Config	Logs

```
11:36:56 AM  [main]     All prerequisites found
11:36:56 AM  [main]     Initializing Modules
11:36:56 AM  [main]     Starting Check-Timer
11:36:56 AM  [main]     Control Panel Ready
11:36:58 AM  [Apache]   Attempting to start Apache app...
11:36:59 AM  [Apache]   Status change detected: running
11:37:01 AM  [mysql]    Attempting to start MySQL app...
11:37:01 AM  [mysql]    Status change detected: running
```

8.2 DATABASE FOR MOVIE

The example of the database that when user key in their data, it will automatically insert here in the “cinemania” database. This is the example when users already choose the title of the movie along with hall, showtime and seat.



Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available

Showing rows 0 - 0 (1 total, Query took 0.0005 seconds.)

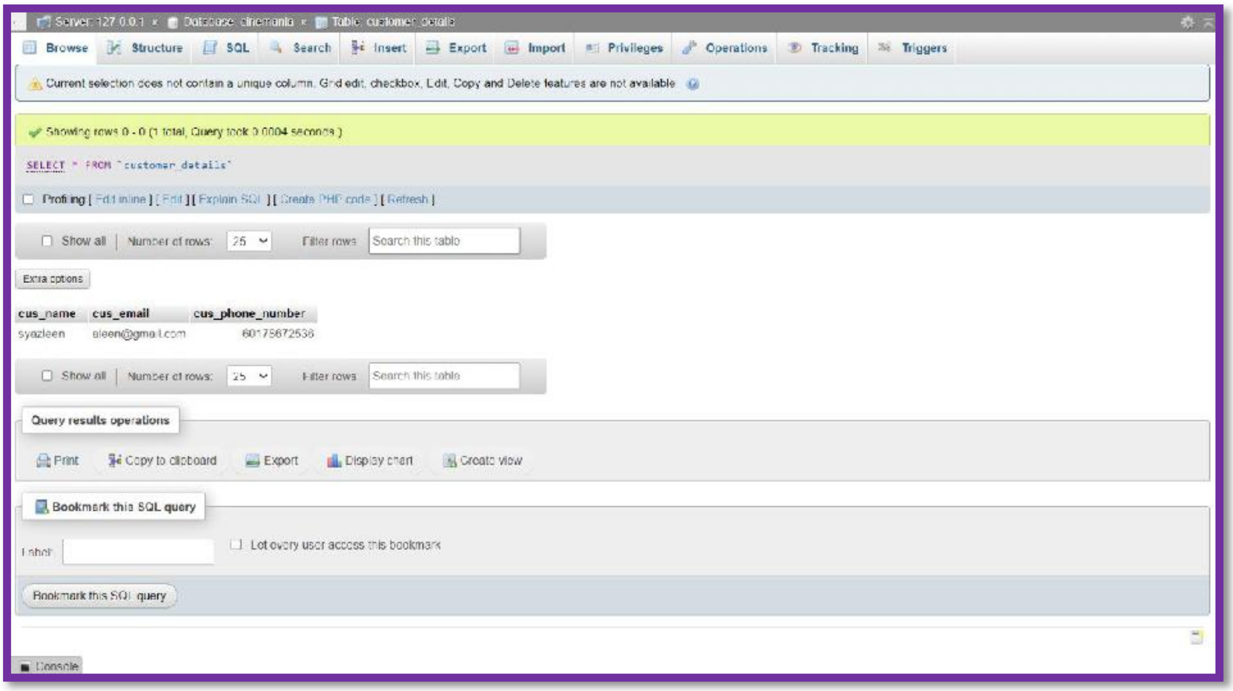
```
SELECT * FROM `movie_info`
```

□ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
□ Show all
Number of rows: 25
Filter rows
Search this table
Extra options

Title	Hall	Showtime	Seat
The Nun (Horror)	Family Session=RM30	8PM - 10PM	1F, 2F, 3F

8.3 DATABASE FOR CUSTOMER

This is an example after the users already fil in their name, email and phone number.



Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available

✓ Showing rows 0 - 0 (1 total, Query took 0.0004 seconds.)

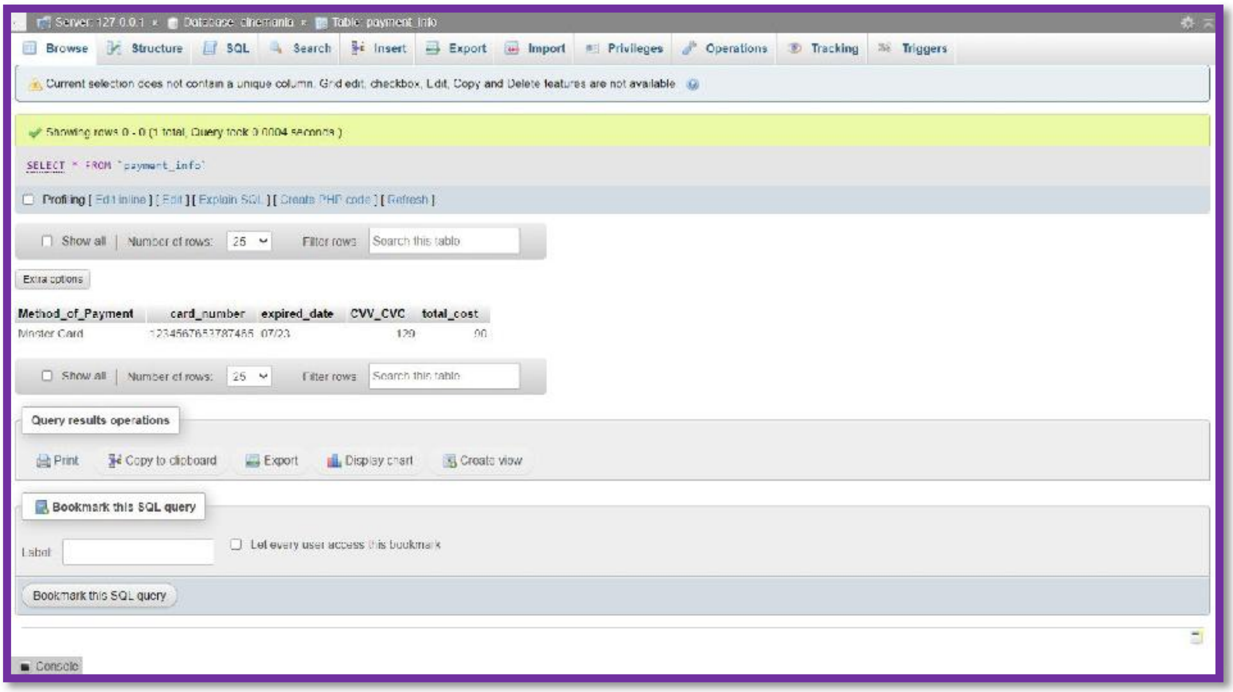
```
SELECT * FROM `customer_details`
```

□ Show all | Number of rows: 25 | Filter rows: Search this table

cus_name	cus_email	cus_phone_number
syazleen	aleen@gmail.com	60175672536

8.4 DATABASE FOR METHOD OF PAYMENT

This is an example, if users choose master card for their method of payment and also have the card number, expired date and cvv/cvc. It also has the total cost of payment that users need to pay.



Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available

Showing rows 0 - 0 (1 total, Query took 0.0004 seconds.)

```
SELECT * FROM `payment_info`
```

□ Show all | Number of rows: 25 Filter rows: Search this table

Method_of_Payment	card_number	expired_date	CVV_CVC	total_cost
Master Card	1234567653787465	07/23	129	90

Label: □ Let every user access this bookmark

9.0 CONCLUSION

Finally, the Movie Booking System is a well-designed Python software that includes a graphical user interface (GUI) built with Tkinter and is linked to a MySQL database via the MySQL Connector. The system effectively accepts user inputs for user booking data, calculates total costs in real time, and stores the data in a MySQL database table entitled "name Cinemania" To ensure proper data entry into the database, the system performs adequate error handling during database interactions.

System dependability is improved by feedback strategies such as the "Data Entered Successfully" message. this message not only informs users about the successful completion of transactions, but as well contributes to the overall dependability of the Movie Booking system.

This project has been a valuable learning experience, providing us with insights into Python coding techniques and best practices. The exploration of various command structures and their integration into the system has expanded our coding proficiency. As we reflect on the development process, it becomes clear that continuous research and exploration are required to stay abreast of the ever-changing programming landscape.

Special thanks to our classmates and our esteemed lecturer, Sir Airul, for his unwavering support and guidance throughout this endeavour; his insights, feedback, and collaborative spirit were instrumental in overcoming obstacles and achieving our goals. As we conclude this project, we carry forward not only a functional Movie Booking System, but also a wealth of knowledge and skills that will undoubtly benefit our future endeavours in the realm of soft computing.

REFERENCES

Ahasanul Haque, Ali Khatibi and Shameem Al Mahmud, (2009). ‘Factors determinates customer shopping behavior through internet: the Malaysian case’, Australian journal of Basic and Applied Science, 3(4): 3452-3463.

Lunny, O. (2019, June 24). Battle for $15.19 billion secondary ticket market heats up with first Europe-wide anti touting law. Forbes. https://www.forbes.com/sites/oisinlunny/2019/06/24/the-battle-for-15-19b-secondary-ticket-market-heats-up-with-first-europe-wide-anti-touting-law/?sh=6aac29352e02

Lubeck, T.M., Wittmann, M.L., & Battistella, L.F. (2012). Electronic ticketing system as a process of innovation. Journal of Technology Management & Innovation, 7(1), 17-29.

Zhang, Y., Bao, Z., Wang, Q., Lü, N., Shi, W., Chen, B., & Lei, H. (2023). OWL: A data sharing scheme with controllable anonymity and integrity for group users. *Computer Communications*, *209*, 455–468. https://doi.org/10.1016/j.comcom.2023.07.022

Turner, M., & Wilson, R. C. (2010). Smart and integrated ticketing in the UK: Piecing together the jigsaw. *Computer Law & Security Review*, *26*(2), 170–177. https://doi.org/10.1016/j.clsr.2010.01.015