APPENDIX 1

(A typical Specimen of Cover Page & Title Page)

MOVIE TICKET BOOKING SYSTEM

A PROJECT REPORT

Submitted by

MYTHREIY ANAND

LAKSHANYA D

in partial fulfillment for the award of the degree of

Bachelor of engineering

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2024

APPENDIX 2

(A typical Specimen of Bonafide Certificate)

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"MOVIE TICKET BOOKING SYSTEM"** is the bonafide work of "**MYTHREIY ANAND (220701177) AND LAKSHANYA D(220701140)"** who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Dr.R.SABITHA

Professor and II Year Academic Head,

Computer Science and Engineering,

Rajalakshmi Engineering College,

Thandalam,Chennai -602 105

SIGNATURE

Ms.D.KALPANA

Assistant Professor(SG),

Computer Science and Engineering,

Rajalakshmi Engineering College,

Thandalam,Chennai -602 105

ABSTRACT

The Python and SQL-based Movie Ticket Booking System is a comprehensive solution designed to facilitate the booking of movie tickets through a user-friendly interface. Leveraging the Python programming language for frontend development and SQL for backend database management, the system provides users with a seamless and efficient booking experience. The system allows users to browse through a wide range of movie listings, view showtimes, select seats, and securely purchase tickets. By utilizing SQL databases to store and manage movie, user, and booking information, it ensures data integrity, security, and scalability.

With its intuitive interface and robust backend architecture, the system offers cinema-goers a convenient and hassle-free way to book tickets while enabling cinema operators to efficiently manage bookings and optimize operations. Through its Python and SQL-based technical stack, the movie ticket booking system aims to enhance the movie-going experience and drive value for both users and cinema owners alike.

TABLE OF CONTENTS

CHAPTER- 1

1.1 INTRODUCTION

The Ticket Booking System project aims to revolutionize the way tickets are purchased and managed for various events, including concerts, movies, and transportation services. In an era where digital solutions are increasingly becoming the norm, this project seeks to provide a comprehensive, user-friendly platform that addresses the needs of both consumers and event organizers.

The primary objective of this system is to offer a seamless and efficient ticket booking experience. Users can easily search for events, check availability, select preferred seats, and complete transactions with minimal effort. The system is designed to be accessible via multiple devices, ensuring convenience and flexibility for users who prefer booking tickets on-the-go.

1.2 OBJECTIVES

- Develop an intuitive and responsive user interface that simplifies the ticket booking process.
- Employ robust security measures, including encryption and fraud detection, to safeguard user information.
- Provide real-time analytics and reporting features to help organizers make informed decisions.
- Develop backend tools for administrators to easily manage the platform, including user accounts, event details, and financial transactions.

1.3 MODULES

- Navigation Module
- Movie Details Module
- Movie Browsing and Search Module
- Ticket Booking Module
- Seat Selection Module
- Ticket Generating Module

CHAPTER-2

2.1 SOFTWARE DESCRIPTION

The mysql-connector-python library is a MySQL driver for Python that allows you to connect to a MySQL database, execute queries, and retrieve results. It provides a simple and efficient way to interact with MySQL databases using Python. To use it, install the library with pip install mysql-connector-python and establish a connection using mysql.connector.connect(). This library supports various MySQL operations including CRUD (Create, Read, Update, Delete) operations and transaction management.

2.2 LANGUAGES

1. Python:-

It is used for scripting the application's logic, managing database operations, and integrating different modules.

2. Tkinter:-

Tkinter is the standard Python interface to the Tk GUI toolkit. It is used for developing the graphical user interface (GUI) of the application.

3.MySQL:-

MySQL is used as the database management system for the project. It is an embedded SQL database engine that provides a lightweight and efficient way to store and manage data.

CHAPTER-3

REQUIREMENT AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

The Movie Ticket Booking System requires functionalities for user registration and authentication, profile management, movie browsing and search with filtering options, and detailed movie information display. It must include interactive seat selection with real-time updates, secure payment processing through gateways like Stripe or PayPal, and instant booking confirmations via email and SMS. Administrators need tools for managing movie listings, showtimes, and seat layouts, as well as access to detailed sales and user activity reports. Non-functional requirements include ensuring high performance, scalability, security, and compliance with data protection standards. The system should provide personalized user notifications, promotional alerts, and robust customer support to enhance the overall user experience and operational efficiency.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

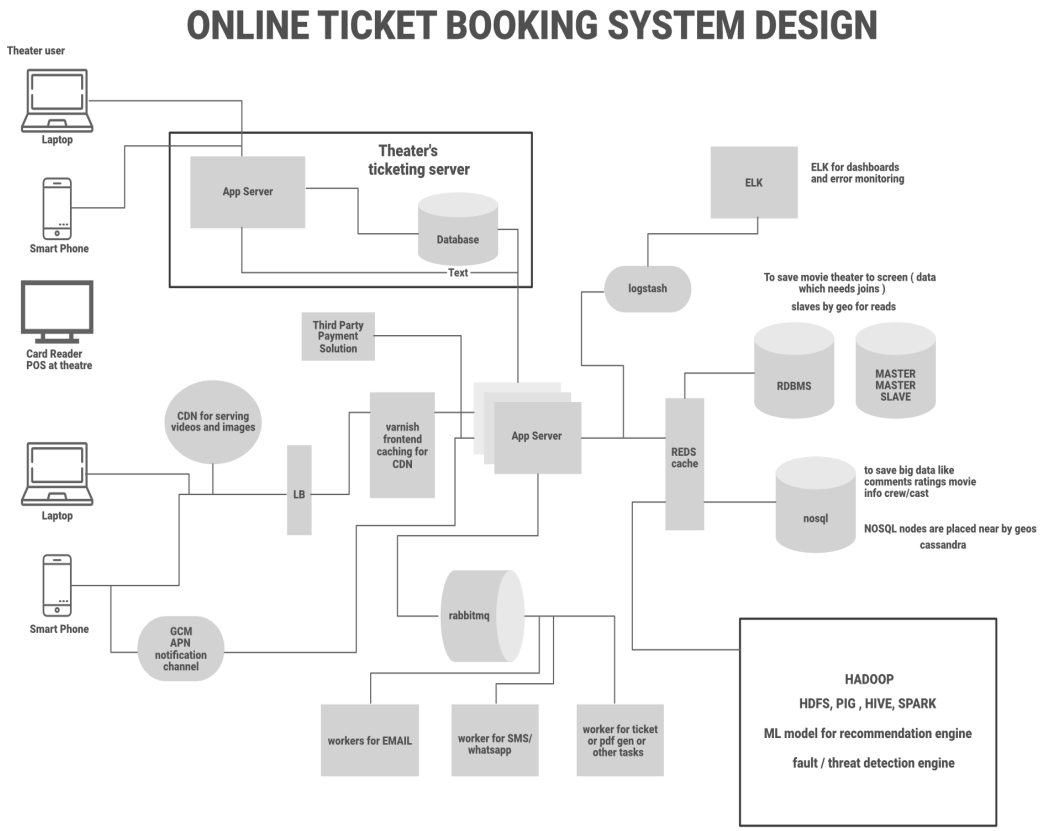
HARDWARE REQUIREMENT

- Processor: Quad-core CPU, 2.5 GHz or higher
- RAM: 16 GB or more
- Storage: SSD with 500 GB or more
- Network: High-speed internet connection (1 Gbps recommended)
- Backup: External backup drive or cloud storage for data redundancy
- Devices: Desktop computers, laptops, tablets, smartphones

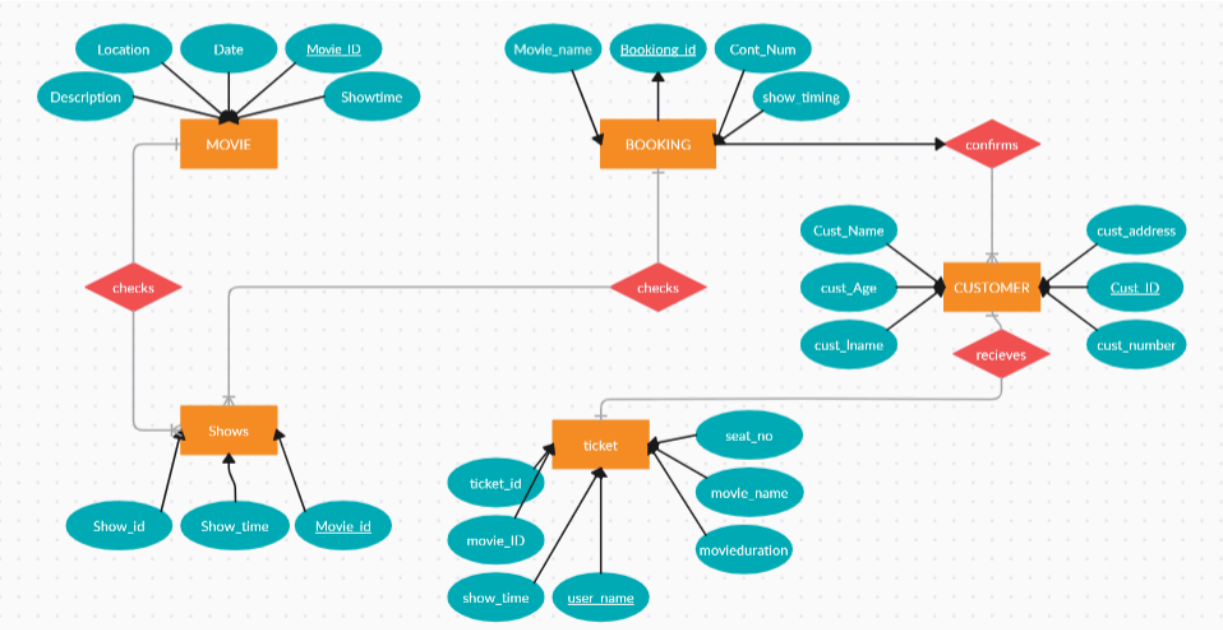
SOFTWARE REQUIREMENT

- Operating System: Linux (Ubuntu, CentOS) or Windows Server
- Database Management System: MySQL
- Programming Language: Python Version 3.6 or higher
- Web Browser: Latest versions of Chrome, Firefox, Safari, Edge
- Python Libraries:
  -‘tkinter’ for GUI development (included with Python)
  - ‘mysql ’ for database management (included with Python)

3.3 ARCHITECTURE DIAGRAM



3.4 ER DIAGRAM



CHAPTER-4

PROGRAM CODE

```
import tkinter as tk

fromnnector


# Database connection setup

def connect_to_db():

    return mysql.connector.connect(

        host="localhost",

        user="root",

        password="Zaynmalik@4",

        database="MovieTicketBooking"

    )


class MovieTicketBookingSystem(tk.Tk):

    def __init__(self):

        super().__init__()

        self.title("Movie Ticket Booking System")

        self.geometry("1000x430")
```

```
        self.resizable(False, False)


        self.frames = {}
        for F in (HomePage, MoviesPage, BookTicketPage, SelectSeatsPage,
ViewBookingsPage, TicketPage):
            page_name = F.__name__
            frame = F(parent=self, controller=self)
            self.frames[page_name] = frame
            frame.grid(row=0, column=0, sticky="nsew")


        self.show_frame("HomePage")


    def show_frame(self, page_name):
        frame = self.frames[page_name]
        frame.tkraise()


class HomePage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.configure(bg="#e0f7fa")
```

label = tk.Label(self, text="Welcome to Movie Ticket Booking System", font=("Arial", 20), bg="#e0f7fa", fg="#004d40")

label.pack(pady=20)

view_movies_button = tk.Button(self, text="View Movies", command=lambda: controller.show_frame("MoviesPage"), bg="#4dd0e1", fg="white", font=("Arial", 12), width=20)

view_movies_button.pack(pady=10)

book_ticket_button = tk.Button(self, text="Book Ticket", command=lambda: controller.show_frame("BookTicketPage"), bg="#26c6da", fg="white", font=("Arial", 12), width=20)

book_ticket_button.pack(pady=10)

view_bookings_button = tk.Button(self, text="View Bookings", command=lambda: controller.show_frame("ViewBookingsPage"), bg="#00acc1", fg="white", font=("Arial", 12), width=20)

view_bookings_button.pack(pady=10)

class MoviesPage(tk.Frame):

def __init__(self, parent, controller):

super().__init__(parent)

```
        self.controller = controller
        self.configure(bg="#e0f7fa")

        label = tk.Label(self, text="Available Movies", font=("Arial", 20), bg="#e0f7fa", fg="#004d40")
        label.pack(pady=20)

        self.tree = ttk.Treeview(self, columns=("ID", "Title", "Genre", "Duration", "Rating"), show='headings')
        self.tree.heading("ID", text="ID")
        self.tree.heading("Title", text="Title")
        self.tree.heading("Genre", text="Genre")
        self.tree.heading("Duration", text="Duration")
        self.tree.heading("Rating", text="Rating")
        self.tree.pack(pady=20)

        self.load_movies()

        back_button = tk.Button(self, text="Back to Home", command=lambda: controller.show_frame("HomePage"), bg="#4dd0e1", fg="white", font=("Arial", 12), width=20)
        back_button.pack(pady=10)
```

```
    def load_movies(self):
        db = connect_to_db()
        cursor = db.cursor()
        cursor.execute("SELECT * FROM Movies")
        movies = cursor.fetchall()
        for movie in movies:
            self.tree.insert("", "end", values=movie)
        db.close()


class BookTicketPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.configure(bg="#e0f7fa")

        label = tk.Label(self, text="Book Ticket", font=("Arial", 20), bg="#e0f7fa",
fg="#004d40")
        label.pack(pady=20)

        tk.Label(self, text="Movie ID:", bg="#e0f7fa", fg="#004d40").pack(pady=5)
```

```
        self.movie_id_entry = tk.Entry(self)
        self.movie_id_entry.pack(pady=5)

        tk.Label(self, text="Show Timing:", bg="#e0f7fa", fg="#004d40").pack(pady=5)
        self.show_time_combobox = ttk.Combobox(self)
        self.show_time_combobox.pack(pady=5)

        tk.Label(self, text="Your Name:", bg="#e0f7fa", fg="#004d40").pack(pady=5)
        self.user_name_entry = tk.Entry(self)
        self.user_name_entry.pack(pady=5)

        tk.Label(self, text="Number of Seats:", bg="#e0f7fa", fg="#004d40").pack(pady=5)
        self.seats_entry = tk.Entry(self)
        self.seats_entry.pack(pady=5)

        book_button = tk.Button(self, text="Select Seats", command=self.select_seats, bg="#26c6da", fg="white", font=("Arial", 12), width=20)
        book_button.pack(pady=10)

        back_button = tk.Button(self, text="Back to Home", command=lambda: controller.show_frame("HomePage"), bg="#4dd0e1", fg="white", font=("Arial", 12), width=20)
```

```
        back_button.pack(pady=10)

        self.movie_id_entry.bind("<FocusOut>", self.load_show_timings)

    def load_show_timings(self, event):
        movie_id = self.movie_id_entry.get()
        if not movie_id:
            return
        db = connect_to_db()
        cursor = db.cursor()
        cursor.execute("SELECT show_id, show_time FROM ShowTimings WHERE movie_id=%s", (movie_id,))
        show_timings = cursor.fetchall()
        db.close()

        self.show_time_combobox['values'] = [f"{show[1]}" for show in show_timings]
        self.show_time_combobox.set('')

    def select_seats(self):
        movie_id = self.movie_id_entry.get()
        show_time = self.show_time_combobox.get()
```

```
        user_name = self.user_name_entry.get()
        seats = self.seats_entry.get()

        if not movie_id or not show_time or not user_name or not seats:
            messagebox.showerror("Error", "All fields are required")
            return

        show_id = self.get_show_id(movie_id, show_time)
        if not show_id:
            messagebox.showerror("Error", "Invalid show timing selected")
            return

        self.controller.frames["SelectSeatsPage"].configure(show_id=show_id,
user_name=user_name, seats=int(seats))
        self.controller.show_frame("SelectSeatsPage")

    def get_show_id(self, movie_id, show_time):
        db = connect_to_db()
        cursor = db.cursor()
        cursor.execute("SELECT show_id FROM ShowTimings WHERE movie_id=%s AND
show_time=%s", (movie_id, show_time))
```

```
        show_id = cursor.fetchone()
        db.close()
        return show_id[0] if show_id else None

class SelectSeatsPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.selected_seats = []

        label = tk.Label(self, text="Select Seats", font=("Arial", 20), bg="#e0f7fa",
fg="#004d40")
        label.pack(pady=20)

        self.seats_frame = tk.Frame(self, bg="#e0f7fa")
        self.seats_frame.pack(pady=20)

        self.book_button = tk.Button(self, text="Book", command=self.book_seats,
bg="#26c6da", fg="white", font=("Arial", 12), width=20)
        self.book_button.pack(pady=10)
```

```
        self.back_button = tk.Button(self, text="Back to Book Ticket", command=lambda: controller.show_frame("BookTicketPage"), bg="#4dd0e1", fg="white", font=("Arial", 12), width=20)
        self.back_button.pack(pady=10)

    def configure(self, show_id, user_name, seats):
        self.show_id = show_id
        self.user_name = user_name
        self.seats = seats
        self.load_seats()

    def load_seats(self):
        for widget in self.seats_frame.winfo_children():
            widget.destroy()

        db = connect_to_db()
        cursor = db.cursor()
        cursor.execute("SELECT seat_number, status FROM Seats WHERE show_id=%s", (self.show_id,))
        seats = cursor.fetchall()

        row = 0
```

```
        col = 0
        self.seat_buttons = {}
        for seat in seats:
            seat_number, status = seat
            button = tk.Button(self.seats_frame, text=seat_number, width=5, height=2,
bg="#4dd0e1" if status == 'available' else "#b0bec5",
                               state=tk.NORMAL if status == 'available' else tk.DISABLED,
                               command=lambda sn=seat_number: self.toggle_seat(sn))
            button.grid(row=row, column=col, padx=5, pady=5)
            self.seat_buttons[seat_number] = button
            col += 1
            if col >= 10:
                col = 0
                row += 1
        db.close()

    def toggle_seat(self, seat_number):
        if seat_number in self.selected_seats:
            self.selected_seats.remove(seat_number)
            self.seat_buttons[seat_number].configure(bg="#4dd0e1")
        else:
```

```
            if len(self.selected_seats) < self.seats:
                self.selected_seats.append(seat_number)
                self.seat_buttons[seat_number].configure(bg="#26c6da")
            else:
                messagebox.showwarning("Warning", f"You can only select {self.seats} seats")

    def book_seats(self):
        if len(self.selected_seats) != self.seats:
            messagebox.showwarning("Warning", f"You must select exactly {self.seats} seats")
            return

        db = connect_to_db()
        cursor = db.cursor()

        # Update seats status to 'booked'
        for seat in self.selected_seats:
            cursor.execute("UPDATE Seats SET status='booked' WHERE show_id=%s AND
seat_number=%s", (self.show_id, seat))

        # Insert booking record
        cursor.execute("INSERT INTO Bookings (show_id, user_name, seats) VALUES (%s,
```

```
%s, %s)",
                    (self.show_id, self.user_name, len(self.selected_seats)))

        db.commit()
        db.close()

        ticket_data = {
            "user_name": self.user_name,
            "show_id": self.show_id,
            "seats": self.selected_seats
        }
        self.controller.frames["TicketPage"].set_ticket_data(ticket_data)
        self.controller.show_frame("TicketPage")



class ViewBookingsPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.configure(bg="#e0f7fa")
```

```
        label = tk.Label(self, text="Bookings", font=("Arial", 20), bg="#e0f7fa",
fg="#004d40")

        label.pack(pady=20)


        self.tree = ttk.Treeview(self, columns=("ID", "Show ID", "User Name", "Seats"),
show='headings')

        self.tree.heading("ID", text="ID")

        self.tree.heading("Show ID", text="Show ID")

        self.tree.heading("User Name", text="User Name")

        self.tree.heading("Seats", text="Seats")

        self.tree.pack(pady=20)


        self.load_bookings()


        back_button = tk.Button(self, text="Back to Home", command=lambda:
controller.show_frame("HomePage"), bg="#4dd0e1", fg="white", font=("Arial", 12),
width=20)

        back_button.pack(pady=10)


    def load_bookings(self):

        db = connect_to_db()

        cursor = db.cursor()
```

```
        cursor.execute("SELECT * FROM Bookings")
        bookings = cursor.fetchall()
        for booking in bookings:
            self.tree.insert("", "end", values=booking)
        db.close()


class TicketPage(tk.Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller
        self.configure(bg="#e0f7fa")


        self.label = tk.Label(self, text="Your Ticket", font=("Arial", 20), bg="#e0f7fa",
fg="#004d40")
        self.label.pack(pady=20)


        self.ticket_text = tk.Text(self, width=60, height=10, font=("Arial", 12), bg="#e0f7fa",
fg="#004d40", bd=0)
        self.ticket_text.pack(pady=20)


        back_button = tk.Button(self, text="Back to Home", command=lambda:
controller.show_frame("HomePage"), bg="#4dd0e1", fg="white", font=("Arial", 12),
```

```
width=20)

        back_button.pack(pady=10)


    def set_ticket_data(self, ticket_data):

        db = connect_to_db()

        cursor = db.cursor()


        cursor.execute("SELECT movie_id FROM ShowTimings WHERE show_id = %s",
(ticket_data["show_id"],))

        movie_id = cursor.fetchone()[0]


        cursor.execute("SELECT title FROM Movies WHERE movie_id = %s", (movie_id,))

        movie_title = cursor.fetchone()[0]


        cursor.execute("SELECT show_time FROM ShowTimings WHERE show_id = %s",
(ticket_data["show_id"],))

        show_time = cursor.fetchone()[0]


        db.close()


        ticket_info = (
```

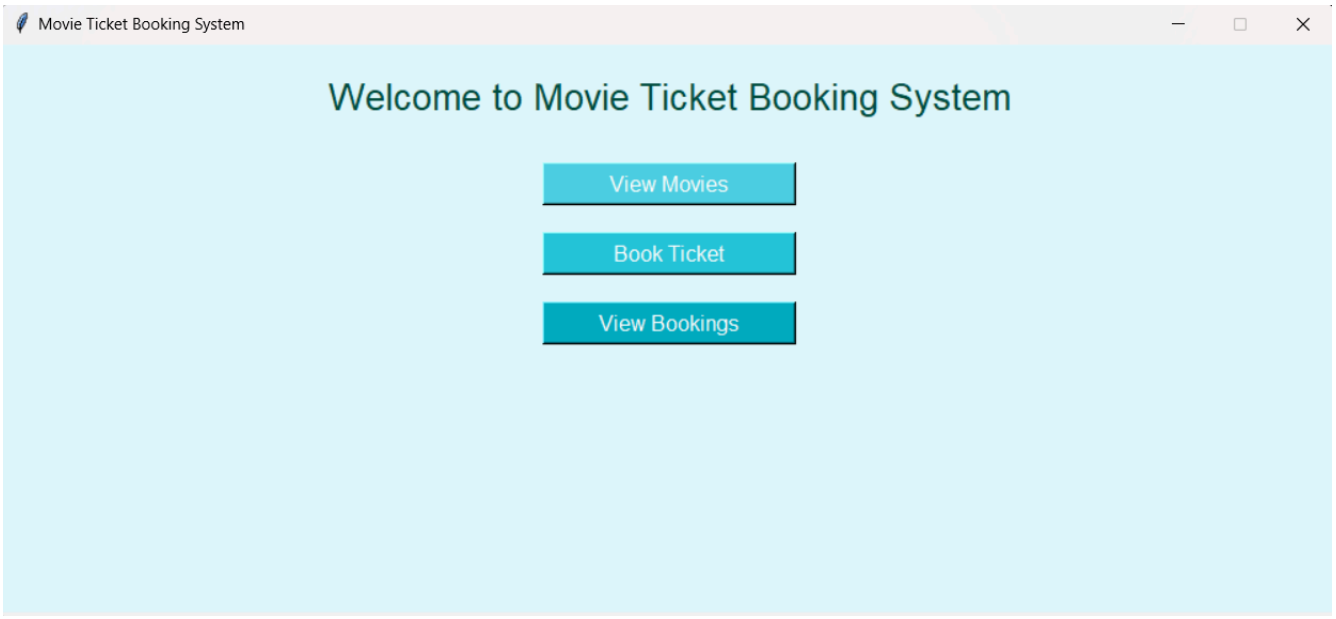
```
            f"User Name: {ticket_data['user_name']}\n"
            f"Movie Title: {movie_title}\n"
            f"Show Time: {show_time}\n"
            f"Seat Numbers: {', '.join(ticket_data['seats'])}\n"
        )

        self.ticket_text.delete(1.0, tk.END)
        self.ticket_text.insert(tk.END, ticket_info)

if __name__ == "__main__":
    app = MovieTicketBookingSystem()
    app.mainloop()
 tkinter import ttk, messagebox
import mysql.co
```

CHAPTER-5

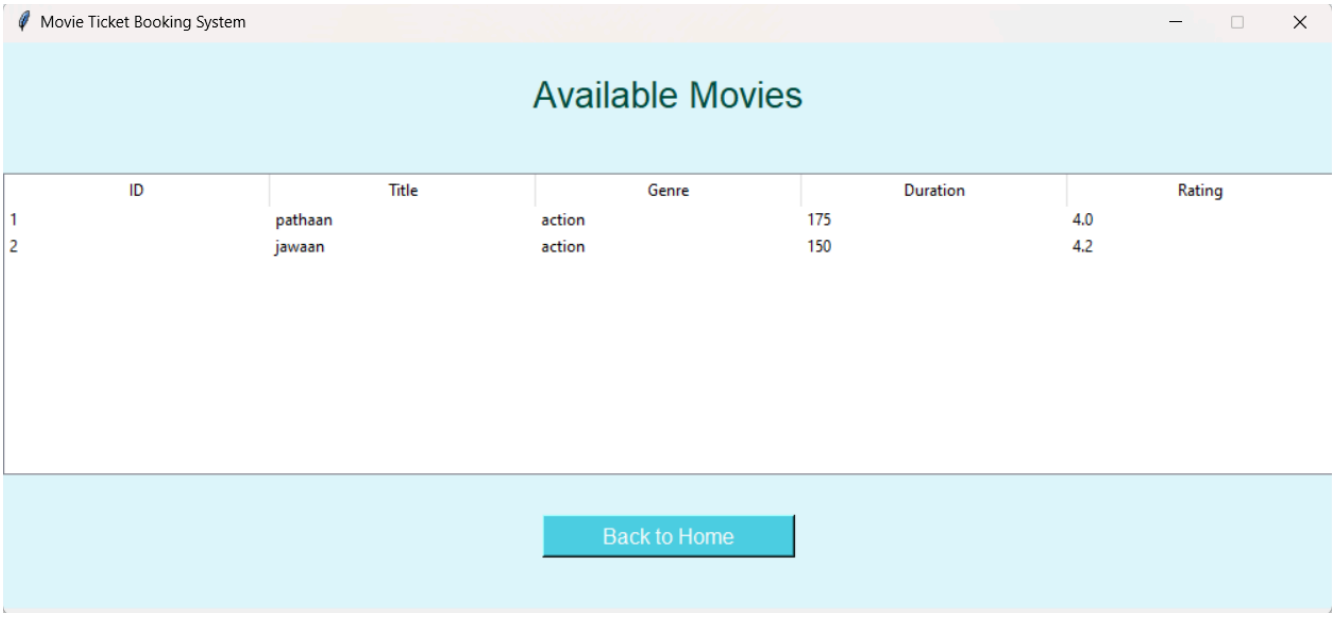
RESULT AND DISCUSSION

5.1 USER DOCUMENTATION

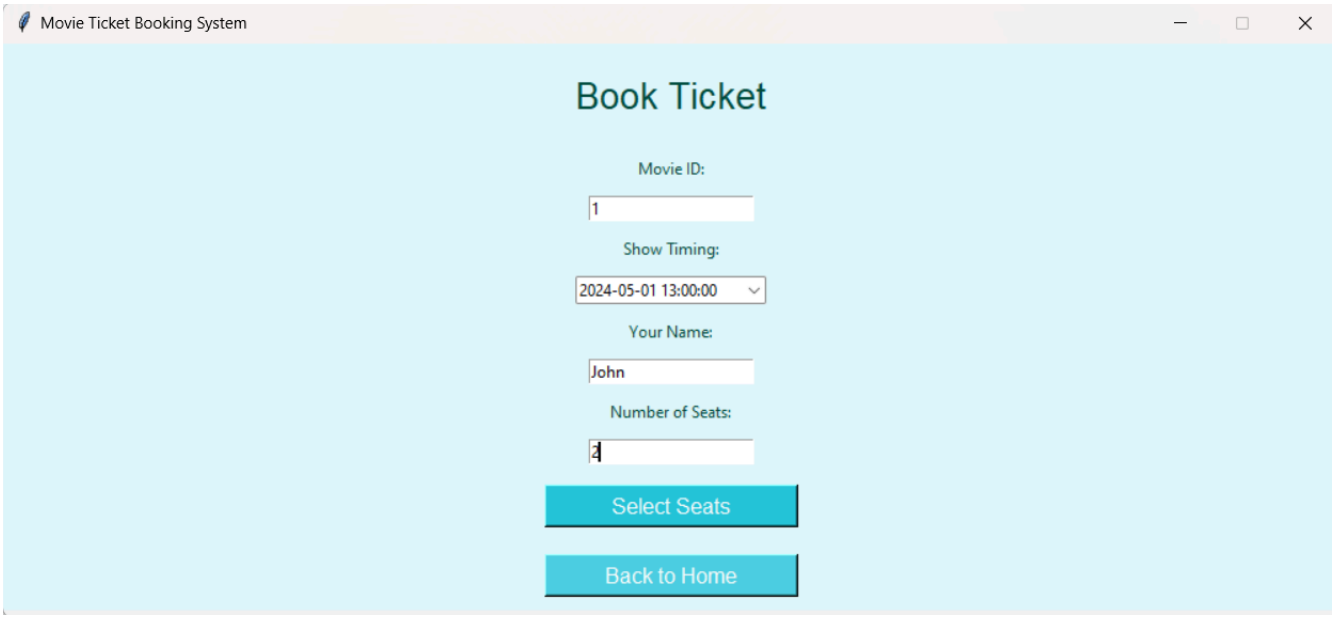
Navigation Module



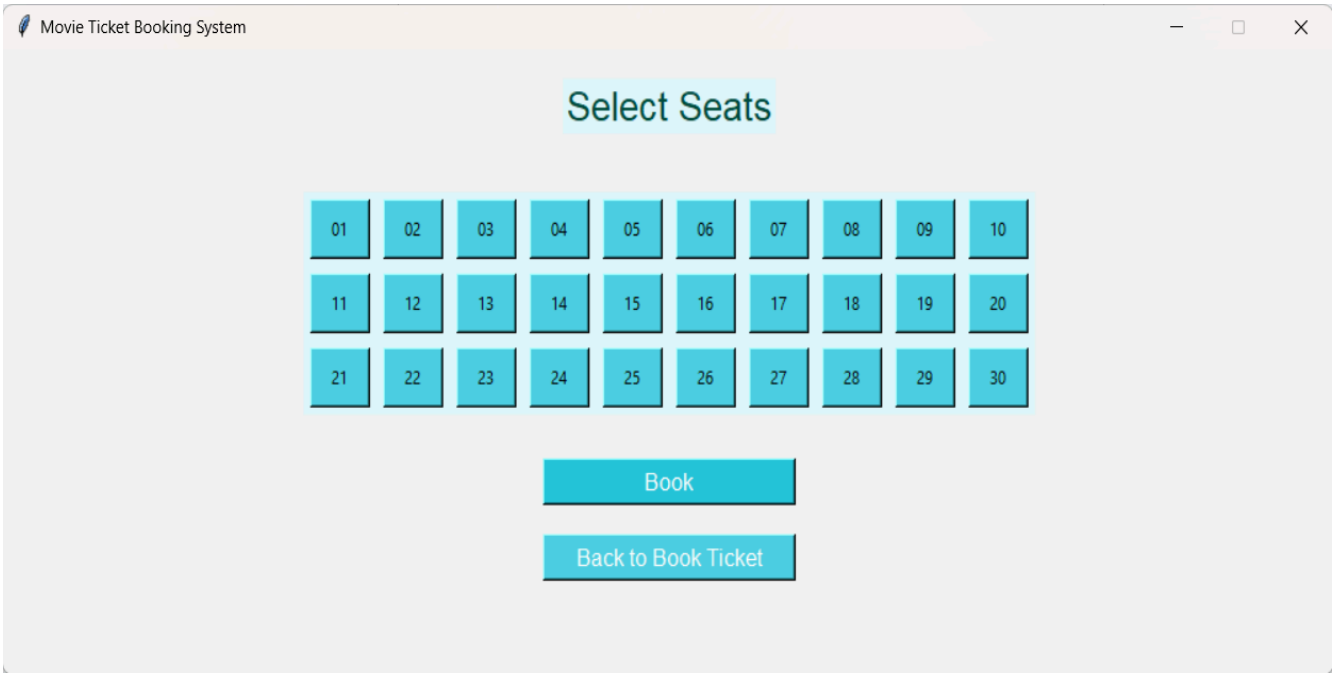
Movie Details Module



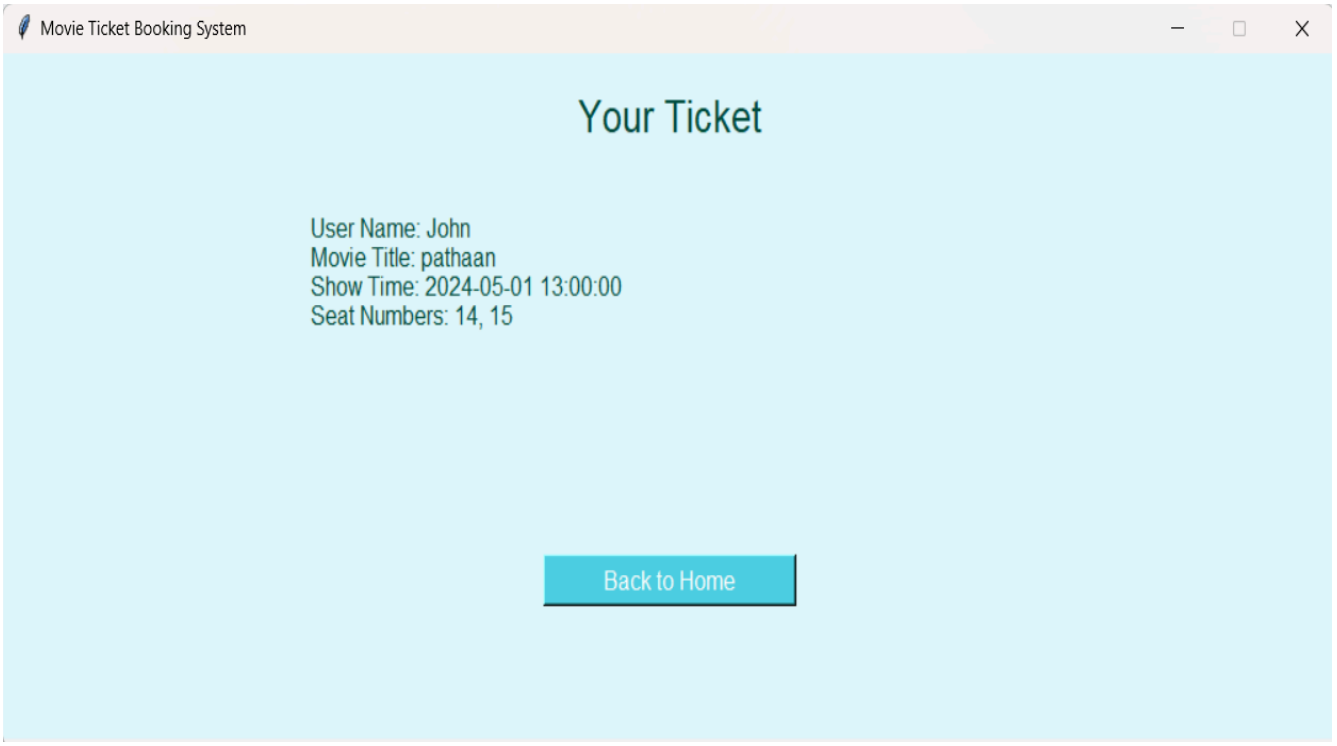
Ticket Booking Module



Seat Selection Module



Ticket Generating Module



CHAPTER-6

6.1 CONCLUSION

In conclusion, the Movie Ticket Booking System represents a comprehensive solution designed to modernize and streamline the process of booking movie tickets, catering to the evolving needs of users and cinema operators alike. Through the development of this system, several key objectives have been achieved:

Firstly, the system offers users a seamless and convenient platform for browsing movies, selecting seats, and completing transactions, enhancing the overall movie-going experience. With features such as real-time seat availability updates, interactive seat selection, and secure payment processing, users can easily find and book tickets for their desired movies with confidence.

Secondly, the system provides powerful management tools for cinema administrators,

allowing them to efficiently manage movie listings, showtimes, seat layouts, and user accounts. Detailed reporting and analytics features enable administrators to gain valuable insights into user behavior, ticket sales, and system performance, facilitating data-driven decision-making and optimization of operations.

Furthermore, the Movie Ticket Booking System prioritizes security and reliability, implementing measures such as data encryption, secure authentication, and regular backups to safeguard user information and ensure uninterrupted service delivery.

CHAPTER-7

7.1 REFERENCES

1. Python Documentation: Python Software Foundation. (n.d.). Python Documentation. Retrieved from https://docs.python.org/3/

2. Tkinter Documentation: TkDocs. (n.d.). Tkinter Tutorial. Retrieved from https://tkdocs.com/tutorial/

3. MySQL Documentation: MySQL (n.d.). MySQL Documentation. Retrieved from [https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html](https://www.sqlite.org/docs.html)

4. Python GUI Programming with Tkinter: Grayson, J. E. (2000). Python and Tkinter Programming. Manning Publications.

5.MySQL Employees Sample Database

@http://dev.mysql.com/doc/employee/en/index.html.

6. Database System Concepts: Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). Database System Concepts (6th ed.). McGraw-Hill.

7. GeeksforGeeks: Various authors. (n.d.). GeeksforGeeks. Retrieved

from https://www.geeksforgeeks.org/

8. W3Schools: W3Schools. (n.d.). SQL Tutorial. Retrieved from

[https://www.w3schools.com/sql/](https://www.w3schools.com/sql