DSA PROJECT REPORT ON

TOPIC: QUEUE BASED MOVIE TICKET BOOKING SYSTEM

SUBMITTED BY:

WARIHA ASIM (65914)

AQDAS NADEEM (65905)

KHALIL (65887)

UNDER THE GUIDANCE OF

MISS NIDA-E-RUB

DEPARTMENT OF BACHELOR OF SCIENCE IN COMPUTER SCIENCE (BSCS)

SUBMITTED TO THE FACULTY OF COMPUTING,KARACHI INSTITUTE OF ECONOMICS AND TECHNOLOGY

1. Project Title:

Queue based Movie Ticket Booking System

2. Purpose and Objective:

The **Movie Ticket Booking System** is designed to streamline the process of booking movie tickets. The system consists of two user roles: **customers and administrators**. Customers can join the queue, book tickets, view queue status, and more, while administrators can manage the queue, view booking history, and clear the queue. The system is developed using Python with the Tkinter library for a graphical interface, making it user-friendly and easy to interact with.

3. Features Overview:

Customer Features:

1. **Join Queue:**
    - Customers must be above 18 years old and provide a valid email to join the queue. Invalid input results in an error message.
2. **Book Tickets:**
    - Only the customer at the front of the queue can book tickets.
    - If a customer tries to book tickets out of turn, they are informed of their position in the queue, ensuring the queue operates on the FIFO principle.
3. **Seat Availability and Booking Confirmation:**
    - The system ensures that only available seats can be booked. Initially, all seats are displayed as available.
    - When a customer books tickets, the seats selected will turn **gray** to indicate they are booked.
    - **Payment Gateway:** Payment is integrated into the booking process. Only after the customer completes the payment can the seat be officially confirmed and booked.
    - The number of seats booked is directly linked to the quantity of tickets selected by the customer. For instance, if the customer selects 3 tickets, 3 seats will be reserved from the available 100 seats.
    - Once the payment is confirmed, the seats are marked as booked and unavailable for other customers.
4. **Booking History:**
    - Customers can view a history of their past bookings.
5. **Exit:**
    - Upon exiting, customers receive a "Thank You" message.

Admin Features:

1. **Admin Login**
   - Admins log in using the **default password**: `admin123`.
2. **View Booking History**
   - Admins can view all booking details, including customer information.
3. **Clear Queue**
   - Admins can clear selected customers or the entire queue.
4. **Exit**
   - When the admin clicks **Exit**, it will close the **admin panel window**

4. Project's Relevance to Data Structures & Queue Implementation:

This project effectively demonstrates the use of data structures, specifically the **queue**, in a real-world scenario like a movie ticket booking system. The queue follows the **FIFO (First In First Out)** principle, ensuring that customers are processed in the order they join. The system uses **Numpy arrays** to store customer details and efficiently manage the queue operations.

Queue Structure:

The queue structure is implemented using **Numpy arrays**, which store customer details such as ID, name, email, age, and status. Customers are processed in the order they join the queue, ensuring a fair and systematic booking process.

Queue Operations:

1. **Enqueue Operation**: When a customer joins the queue, their details are added to the end of the queue. This operation ensures that the system maintains the order in which customers arrive. The customer's details are appended to the queue array using:

   ```
   queue = np.append(queue, customer_details)  # Adding customer to the queue
   ```

2. **Dequeue Operation**:Once a customer has been processed (i.e., their tickets have been booked or they have exited the queue), they are removed from the front of the queue using:

   ```
   queue = np.delete(queue, index)  # Removing a customer from the queue
   ```

FIFO Implementation in Booking Tickets:

The **Booking Tickets** functionality adheres to the FIFO principle. Only the customer at the front of the queue (the first to join) is allowed to book tickets. If any customer tries to book tickets out of order, an error message is displayed. This ensures that the sequence is maintained and that customers are processed in the exact order they joined the queue.

Example Error Handling: If a customer at the back of the queue tries to book a ticket, they are shown an error message, and the system displays the name of the customer at the front of the queue:

```
if queue[0] != current_customer:

    error_message = "You are not at the front of the queue. Please wait your turn."
    # Show error and display the front of the queue customer
```

Efficiency: Using a queue structure allows the operations to be highly efficient, with a time complexity of **O(1)** for both enqueue and dequeue operations. This ensures that the system can handle a large number of customers without significant performance degradation.

By using a queue for this system, we effectively maintain the order in which customers join the queue, ensuring a smooth and fair booking experience. The system is not only functional but also responsive and efficient, making it an excellent use case for data structures in real-world applications.

5. Technical Details:

Technologies Used:

- **Frontend**: Tkinter (Python library) for GUI development.
- **Backend**: Numpy arrays for efficient queue management and customer data storage.
- **Image Processing**: Pillow for handling background images and resizing them dynamically based on window size.
- **Persistent Storage**: Text files to save queue data (e.g., customer details) and last ID values.

Key Components:

1. ## Queue Management
    - The queue is represented as a Numpy array that stores customer details (ID, Name, Email, Age, Status).
    - **Enqueue**: When customers join the queue, their details are added to the queue.
    - **Dequeue**: When customers are processed or removed from the queue, their details are deleted from the queue array.

- The queue is updated dynamically as customers join or exit the queue, and changes are saved to a file for persistence.

2. **Dynamic Backgrounds**
   - Background images are dynamically updated based on the window configuration using the Pillow library. This enhances the user interface's visual appeal and makes the system more interactive.
3. **Treeview for Data Display**
   - A Treeview widget is used to display the queue status in a structured manner, allowing the admin to easily view and manage customer details in the queue.
4. **Secure Admin Login**
   - The admin panel is password-protected to prevent unauthorized access and ensure data security. The default admin password is set to `admin123`.
5. **Persistent Data**
   - Customer queue data and last ID values are saved to text files to ensure that the system maintains data even after restarting.

6. System Workflow:

1. **Customer Panel Workflow**
   - Customers start by accessing the **Customer Panel** from the main menu.
   - They can join the queue, book tickets, view the queue status, check seat availability, and review their booking history.
   - After completing actions, customers can exit with a "Thank You" message.
2. **Admin Panel Workflow**
   - Admins log in using the default password `admin123` to access the **Admin Panel**.
   - Admins can view booking history, manage the queue (clear selected or all customers), and exit after completing their operations.

8. Detailed Conditions and Error Handling:

1. **Join Queue Conditions**:
   - Age must be greater than 18.
   - The email must be in the correct format (valid email address). If these conditions are not met, the system will display a message informing the user about the issue.
2. **Ticket Booking Conditions**:
   - Only the customer at the front of the queue can book tickets.
   - If a customer tries to book tickets but is not at the front of the queue, they will be shown an error message, and the system will display the name of the customer at the front of the queue.

3.Error Handling:

- Try/except blocks are used to handle potential errors during the booking process (e.g., invalid customer data or an issue with booking).

9. Conclusion:

The **Movie Ticket Booking System** provides a simple, effective solution for booking movie tickets and managing customer queues. With secure admin features and a user-friendly interface, the system can be used by both customers and administrators efficiently. The use of the queue data structure in this project is a great example of how data structures can be applied in real-world scenarios like ticket booking systems. The system is both functional and efficient, handling enqueue and dequeue operations effectively to ensure smooth user interaction.