Cinema Seat Reservation System

Overview

The Cinema Seat Reservation System is an interactive application developed using **F#** and **Windows Forms** to manage cinema seat reservations. It adheres to functional programming principles like immutability, first-class functions, and pattern matching. Users can view available seats, book specific seats, and generate unique tickets for each reservation. The project structure is modular and organized, ensuring clarity and scalability.

Features

1. Seat Layout:

- The seating chart is represented as a list of immutable `Seat` records.
- The application dynamically displays available and reserved seats in a GUI.
- Reserved seats are visually distinct (colored red).

2. Booking System:

- Users can select seats using a GUI.
- Ensures seats cannot be double-booked.

3. **Ticket Management:**

- Generates a unique ticket ID for each reservation.
- Saves booking details (seat, customer name, showtime) to a file.—

4. Functional Programming Principles:

- **Immutability**: Data structures like `Seat` and layouts are immutable.
- **First-Class Functions**: Functions are passed and used in event handlers.
- **Pattern Matching**: Used for validating input and managing optional data.

5. Windows Forms GUI:

- Provides an interactive user interface.
- Supports dropdown selection for showtimes and seat booking via buttons.

6. Modularity:

- Clearly separated components: `SeatManager`, `TicketManager`, `BookingForm`, and `CinemaForm`.

Modules

1. SeatManager

Manages seat layouts and reservations.

Code Features:

- Seat Record:

```
type Seat = { Row: int; Col: int; IsReserved: bool }
```

Represents individual seats in the cinema. Immutable by design.

- Methods:

-InitializeShowtime:

```
member this.InitializeShowtime(showtime: string, rows: int, cols: int)
```

- Initializes a seating layout for a given showtime.

- IsSeatAvailable:

```
member this.IsSeatAvailable(showtime: string, row: int, col: int)
```

- Checks if a specific seat is available.

- ReserveSeat:

```
member this.ReserveSeat(showtime: string, row: int, col: int)
```

- Updates a seat to be reserved by creating a new layout with updated seat data.

- GetSeatLayout:

```
member this.GetSeatLayout(showtime: string)
```

- `Retrieves the layout for a given showtime.

Functional Programming Principles:

- **Immutability**: Seats are updated by creating a new list instead of modifying existing data.

-Pattern Matching: Used to handle optional layouts.

---

2. TicketManager

- Handles ticket generation and persistence.

Code Features:

- Ticket Record:

```
  type Ticket = { TicketID: string; CustomerName: string; Seat: string; Showtime: string }
```

- Represents ticket details. Immutable.

- Methods:

- GenerateTicketID:

```
member this.GenerateTicketID() = Guid.NewGuid().ToString()
```

- Creates a unique ticket ID.

- SaveTicket:

```
member this.SaveTicket(ticket: Ticket)
```

- Appends ticket details to a file (`bookings.txt`).

Functional Programming Principles:

- **Immutability**: Ticket data is represented using immutable records.

3. BookingForm

- Manages the booking process for individual seats.

Code Features:

- UI Components:

- `Label`, `TextBox`, and `Button` are used to gather customer details and confirm bookings.

- Validation:

```
let validateCustomerName name =
    match name with
    | "" | null -> false
    | _ -> true
```

- Ensures customer name input is not empty or null.

- Booking Handling*:*

```
  let handleBooking () =
      let customerName = nameTextBox.Text
      if validateCustomerName customerName then
          let ticketID = ticketManager.GenerateTicketID()
          let seat = $"Row {row + 1}, Col {col + 1}"
          let ticket = { TicketID = ticketID; CustomerName =
customerName; Seat = seat; Showtime = showtime }
          ticketManager.SaveTicket(ticket)
          MessageBox.Show($"Booking Confirmed!\nTicket ID:
{ticketID}") |> ignore
          bookingForm.Close()
      else
          MessageBox.Show("Customer name cannot be empty.",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error) |>
ignore
```

- Handles input validation, ticket creation, and success/error messages.

Functional Programming Principles:

- Pattern Matching: Used to validate input.

- **First-Class Functions**: `handleBooking` is passed to the button click event handler.

4. CinemaForm

Manages the main user interface and seat display.

Code Features:

- UI Components:

- "Form": Main application window.

- "Panel": Contains dynamically generated seat buttons.

- "ComboBox": Allows users to select showtimes.

- Dynamic Seat Layout:

```
  let redrawSeats showtime =
      panel.Controls.Clear()
      let layout = seatManager.GetSeatLayout(showtime)
      layout
      |> List.iter (fun seat ->
          let button = new Button(Text = $"R{seat.Row + 1}C{seat.Col + 1}")
          button.BackColor <- if seat.IsReserved then Color.Red else Color.Green
          button.Click.Add(fun _ ->
              if not seat.IsReserved then
                  let bookingForm = new BookingForm(ticketManager, showtime, seat.Row, seat.Col)
                  bookingForm.ShowBookingForm()
          )
          panel.Controls.Add(button)
      )
```

- **Dynamically generates buttons for seats and updates the panel.**

Functional Programming Principles:

- **First-Class Functions**: `redrawSeats` and seat button click handlers are passed around.

- **Immutability**: Seat states are updated through the `SeatManager`.

Functional Programming Principles in Practice

Immutability

- Data structures (`Seat`, `Ticket`) are immutable.

- Seat reservations generate new layouts without mutating the existing ones.

First-Class Functions

- Functions are passed as arguments (e.g., event handlers).

- Higher-order functions like `List.map` and `List.iter` are used extensively.

Pattern Matching

- Input validation uses pattern matching.

- Optional data (`Map.TryFind`) is handled via `match` expressions.

File Structure

- `**SeatManager.fs**`: Manages seat layouts and reservations.

- `**TicketManager.fs**`: Handles ticket creation and storage.

- `**BookingForm.fs**`: Defines the booking UI and processes bookings.

- `**CinemaForm.fs**`: Builds the main application UI.

- `**Program.fs**`: Entry point for the application.

- `**bookings.txt**`: Stores ticket details.

Conclusion

This system demonstrates a robust application of functional programming principles in F#. It provides an intuitive user interface, modular design, and a clear separation of concerns, making it maintainable and scalable for future development.