Cinema Reservation System

Programming language3 project

Abstract

The Cinema Ticket Reservation System is an online system that allows users to view movies, select showtimes, choose seats, and book tickets electronically. The system reduces manual work, minimizes errors, and improves customer convenience by providing real-time seat availability

Introduction

Traditional ticket booking causes long queues, Manual errors in seat allocation, Need for fast, error free online reservation system therefore, there is a strong need for a fast, reliable, and error-free online ticket reservation system that allows users to book their desired movies from anywhere and at any time with ease and efficiency.

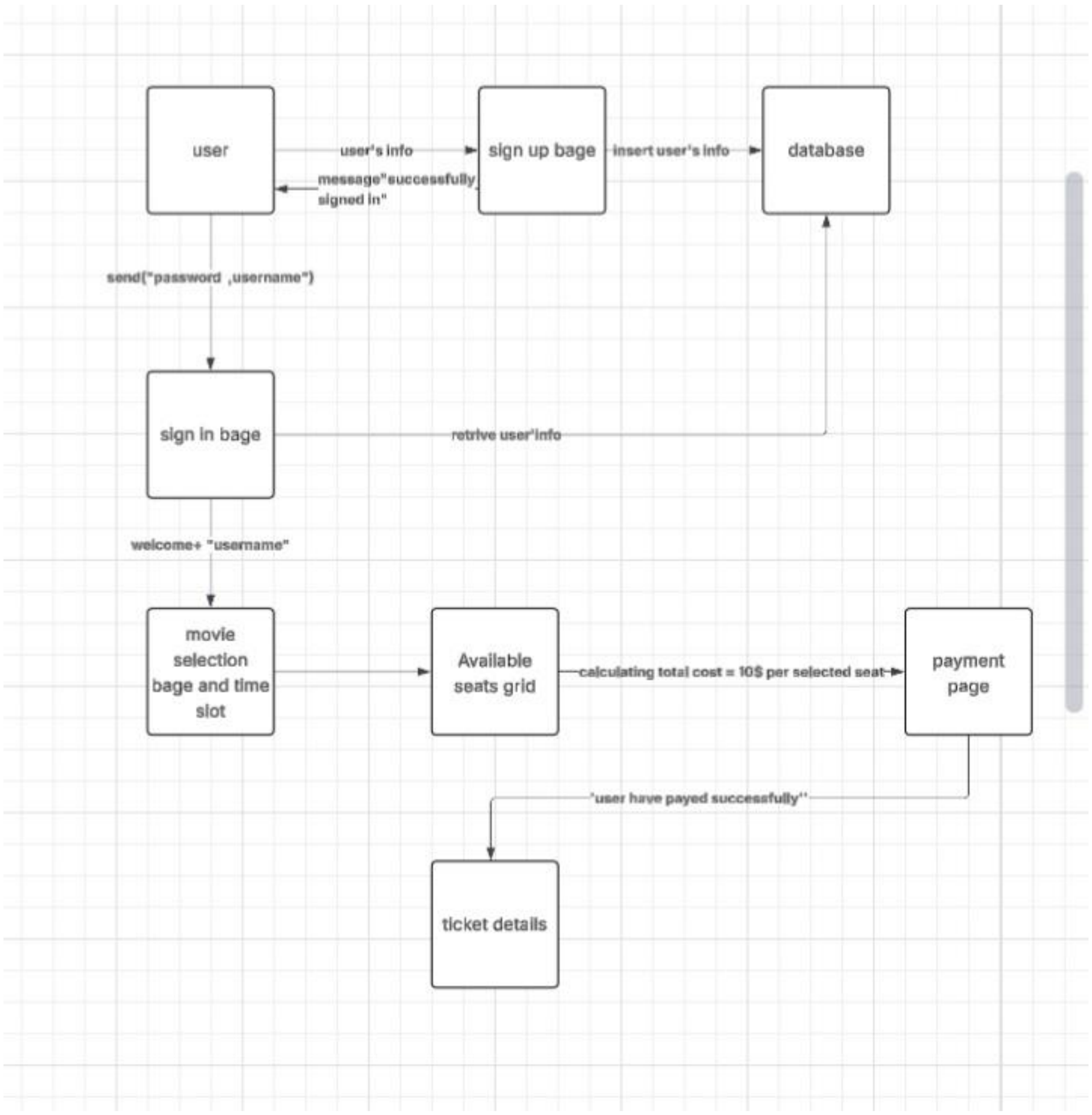
Objectives of the System

1. The signed in user choose a movies and select the time slot
2. select among the seats available in the room
3. show the total cost of the tickets
4. pay the tickets by choosed payment method
5. show the tickets details Electronically

Scope of the System

- ✓ Movie listing
- ✓ Showtime selection
- ✓ Seat booking
- ✓ Online payment
- ✓ Ticket confirmation

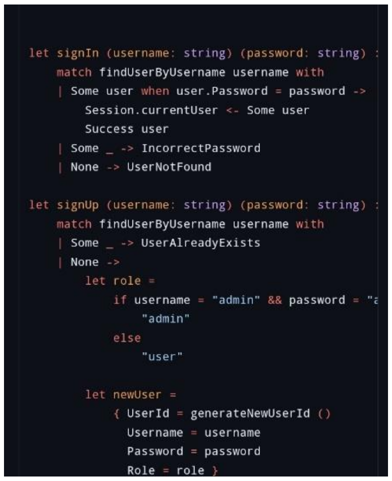
Block Diagram


user
user's info
sign up bage
insert user's info
database
message"successfully signed in"
send("password ,username")
sign in bage
retrive user'info
welcome+ "username"
movie selection bage and time slot
Available seats grid
calculating total cost = 10$ per selected seat
payment page
"user have payed successfully"
ticket details

Functional Requirements

- **User registration and sign in**

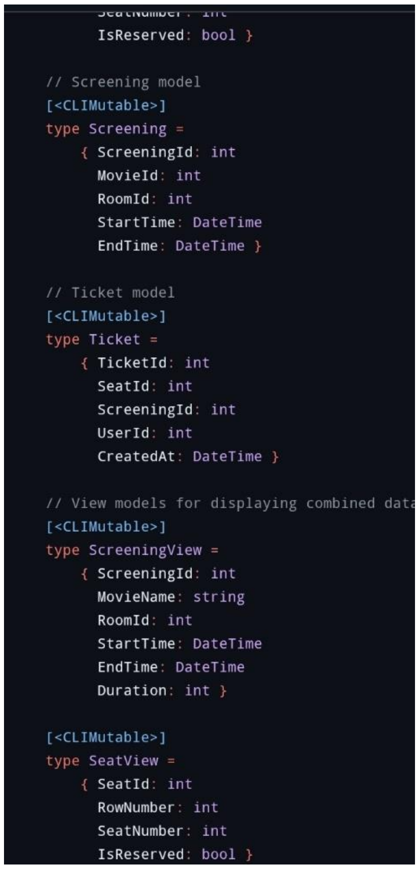
User enters info to sign up then sign in by its username and password



```
let signIn (username: string) (password: string) :
    match findUserByUsername username with
    | Some user when user.Password = password ->
        Session.currentUser <- Some user
        Success user
    | Some _ -> IncorrectPassword
    | None -> UserNotFound

let signUp (username: string) (password: string) :
    match findUserByUsername username with
    | Some _ -> UserAlreadyExists
    | None ->
        let role =
            if username = "admin" && password = "
                "admin"
            else
                "user"

        let newUser =
            { UserId = generateNewUserId ()
              Username = username
              Password = password
              Role = role }
```

- **View available movies and showtimes**

Show available movies to the user and the available rooms and time slots



```
      SeatNumber: int
      IsReserved: bool }

// Screening model
[<CLIMutable>]
type Screening =
    { ScreeningId: int
      MovieId: int
      RoomId: int
      StartTime: DateTime
      EndTime: DateTime }

// Ticket model
[<CLIMutable>]
type Ticket =
    { TicketId: int
      SeatId: int
      ScreeningId: int
      UserId: int
      CreatedAt: DateTime }

// View models for displaying combined data
[<CLIMutable>]
type ScreeningView =
    { ScreeningId: int
      MovieName: string
      RoomId: int
      StartTime: DateTime
      EndTime: DateTime
      Duration: int }

[<CLIMutable>]
type SeatView =
    { SeatId: int
      RowNumber: int
      SeatNumber: int
      IsReserved: bool }
```

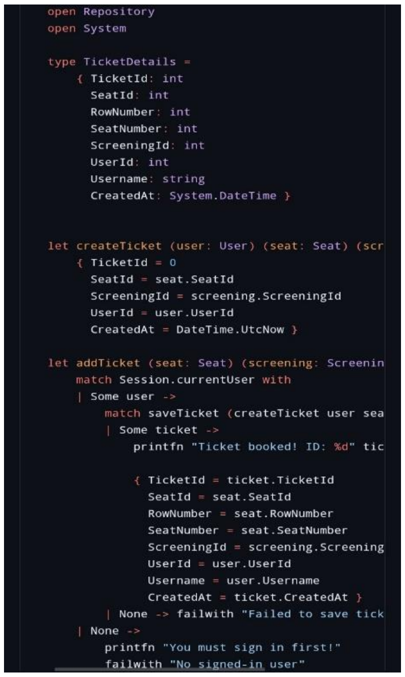
- **Select seats**

  Selecting seats green: not selected, blue: selected, red: not available

- **Calculate ticket price**

  Automatically calculate the cost = selected seats * 10$

- **Generate tickets**

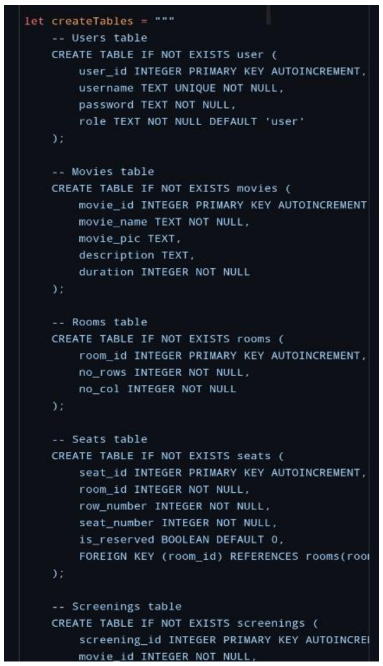


```
open Repository
open System

type TicketDetails =
    { TicketId: int
      SeatId: int
      RowNumber: int
      SeatNumber: int
      ScreeningId: int
      UserId: int
      Username: string
      CreatedAt: System.DateTime }


let createTicket (user: User) (seat: Seat) (scr
    { TicketId = 0
      SeatId = seat.SeatId
      ScreeningId = screening.ScreeningId
      UserId = user.UserId
      CreatedAt = DateTime.UtcNow }

let addTicket (seat: Seat) (screening: Screenin
    match Session.currentUser with
    | Some user ->
        match saveTicket (createTicket user sea
        | Some ticket ->
            printfn "Ticket booked! ID: %d" tic

            { TicketId = ticket.TicketId
              SeatId = seat.SeatId
              RowNumber = seat.RowNumber
              SeatNumber = seat.SeatNumber
              ScreeningId = screening.Screening
              UserId = user.UserId
              Username = user.Username
              CreatedAt = ticket.CreatedAt }
        | None -> failwith "Failed to save tick
    | None ->
        printfn "You must sign in first!"
        failwith "No signed-in user"
```

Database overview

Data stored in the database includes: -

Users



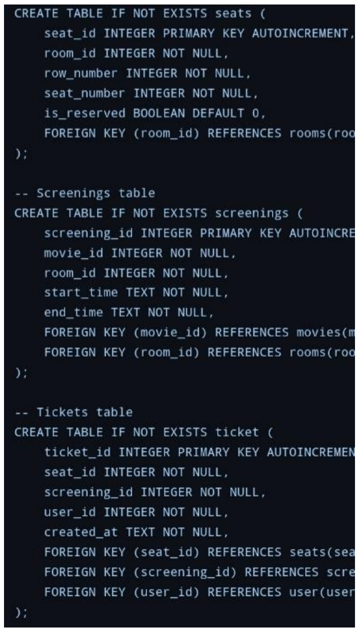
```
let createTables = """
    -- Users table
    CREATE TABLE IF NOT EXISTS user (
        user_id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        role TEXT NOT NULL DEFAULT 'user'
    );

    -- Movies table
    CREATE TABLE IF NOT EXISTS movies (
        movie_id INTEGER PRIMARY KEY AUTOINCREMENT
        movie_name TEXT NOT NULL,
        movie_pic TEXT,
        description TEXT,
        duration INTEGER NOT NULL
    );

    -- Rooms table
    CREATE TABLE IF NOT EXISTS rooms (
        room_id INTEGER PRIMARY KEY AUTOINCREMENT,
        no_rows INTEGER NOT NULL,
        no_col INTEGER NOT NULL
    );

    -- Seats table
    CREATE TABLE IF NOT EXISTS seats (
        seat_id INTEGER PRIMARY KEY AUTOINCREMENT,
        room_id INTEGER NOT NULL,
        row_number INTEGER NOT NULL,
        seat_number INTEGER NOT NULL,
        is_reserved BOOLEAN DEFAULT 0,
        FOREIGN KEY (room_id) REFERENCES rooms(roo
    );

    -- Screenings table
    CREATE TABLE IF NOT EXISTS screenings (
        screening_id INTEGER PRIMARY KEY AUTOINCRE
        movie_id INTEGER NOT NULL,
```

Tickets

Seats

Screenings

Movies



```sql
CREATE TABLE IF NOT EXISTS seats (
    seat_id INTEGER PRIMARY KEY AUTOINCREMENT,
    room_id INTEGER NOT NULL,
    row_number INTEGER NOT NULL,
    seat_number INTEGER NOT NULL,
    is_reserved BOOLEAN DEFAULT 0,
    FOREIGN KEY (room_id) REFERENCES rooms(roo
);

-- Screenings table
CREATE TABLE IF NOT EXISTS screenings (
    screening_id INTEGER PRIMARY KEY AUTOINCRE
    movie_id INTEGER NOT NULL,
    room_id INTEGER NOT NULL,
    start_time TEXT NOT NULL,
    end_time TEXT NOT NULL,
    FOREIGN KEY (movie_id) REFERENCES movies(m
    FOREIGN KEY (room_id) REFERENCES rooms(roo
);

-- Tickets table
CREATE TABLE IF NOT EXISTS ticket (
    ticket_id INTEGER PRIMARY KEY AUTOINCREMEN
    seat_id INTEGER NOT NULL,
    screening_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TEXT NOT NULL,
    FOREIGN KEY (seat_id) REFERENCES seats(sea
    FOREIGN KEY (screening_id) REFERENCES scre
    FOREIGN KEY (user_id) REFERENCES user(user
);
```

Rooms

Test Strategy

Scope of Testing

In Scope

- User registration and login
- Admin login
- Movie management (add, update, delete movies)
- Room and seat management
- Screening creation and scheduling
- Seat availability checking
- Ticket booking and confirmation
- Database operations (insert, update, retrieve)

Out of Scope

- External payment gateway failures
- Cinema hardware systems
- Network infrastructure issues

Unit Testing

- User registration function
- Seat booking validation function
- Movie creation function

Tools: Manual testing / developer testing

Integration Testing

- Booking process with seat availability
- Screening creation with movie duration
- Ticket generation after booking

System Testing

Tests full booking flow:

1. User login
2. Movie selection
3. Seat selection
4. Ticket booking
5. Confirmation

Acceptance Testing

- Ensures system meets user and admin requirements
- Conducted using real-life scenarios