Contents

Analysis

Objective:

The objective of the Cinema Ticketing System project is to develop a comprehensive databasedriven application that streamlines the process of movie ticket booking for customers while providing administrative functionality for cinema staff. The system aims to provide an intuitive user interface for customers to select movies, choose slots, and book tickets for their desired showtimes, as well as an admin interface for managing movie listings, scheduling, and ticketing operations.

Summary:

The Cinema Ticketing System is a robust and user-friendly platform designed to facilitate the reservation and purchasing of cinema tickets through an online portal. This system replaces the traditional manual ticketing process with an automated solution that enhances the efficiency and accuracy of cinema operations. On the user side, customers can browse current movie listings, select a showing, choose their seats, and make payments securely. The admin side allows cinema staff to update the movie database, including additions, modifications, and deletions, as well as manage show timings, halls, and seat arrangements. The system ensures data integrity and provides real-time updates to both users and admins, thus improving the overall movie-going experience and operational management.

Scope:

The scope of the Cinema Ticketing System project encompasses the following key features and functionalities:

User Module:

User Registration and Authentication: Allow users to create accounts and log in securely.

Movie Selection: Enable users to browse through a list of available movies, view details, and select preferred options.

Show Timings and Dates: Provide information about movie schedules, show timings, and available dates.

Ticket Booking: Allow users to choose specific slots, select seats, and complete the booking process.

Booking History: Maintain a record of users' booking history for reference.

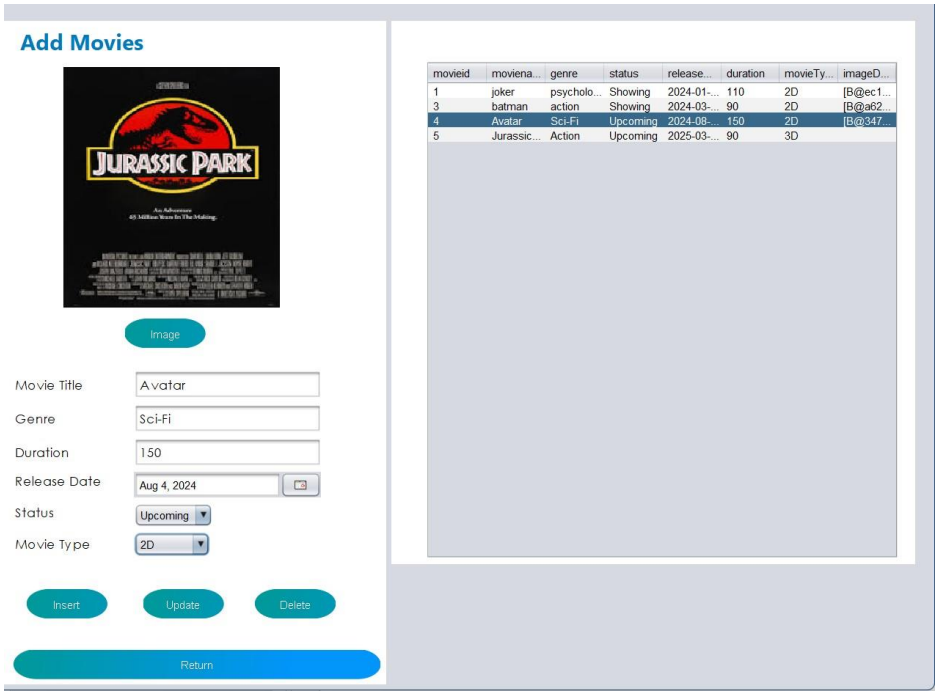
Admin Module:

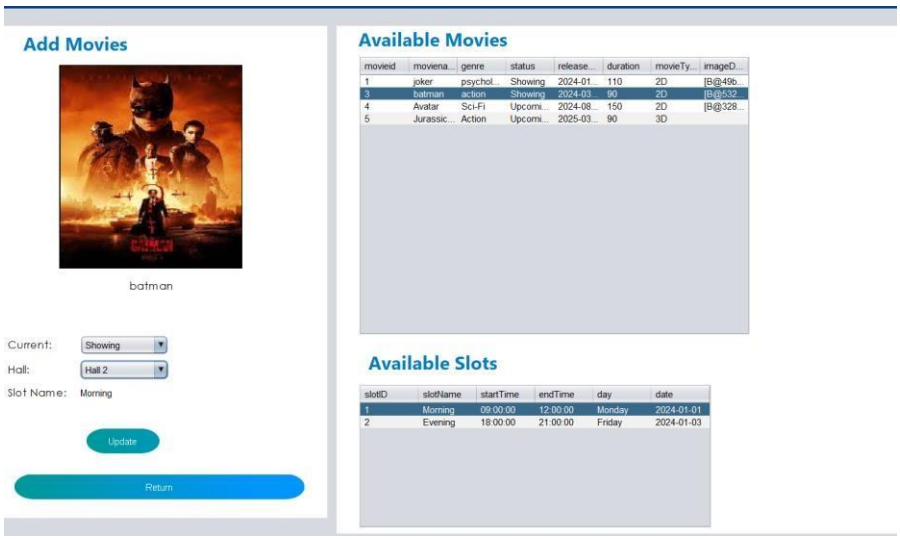
Movie Management: Allow administrators to add new movies, update existing movie details, and remove outdated entries.

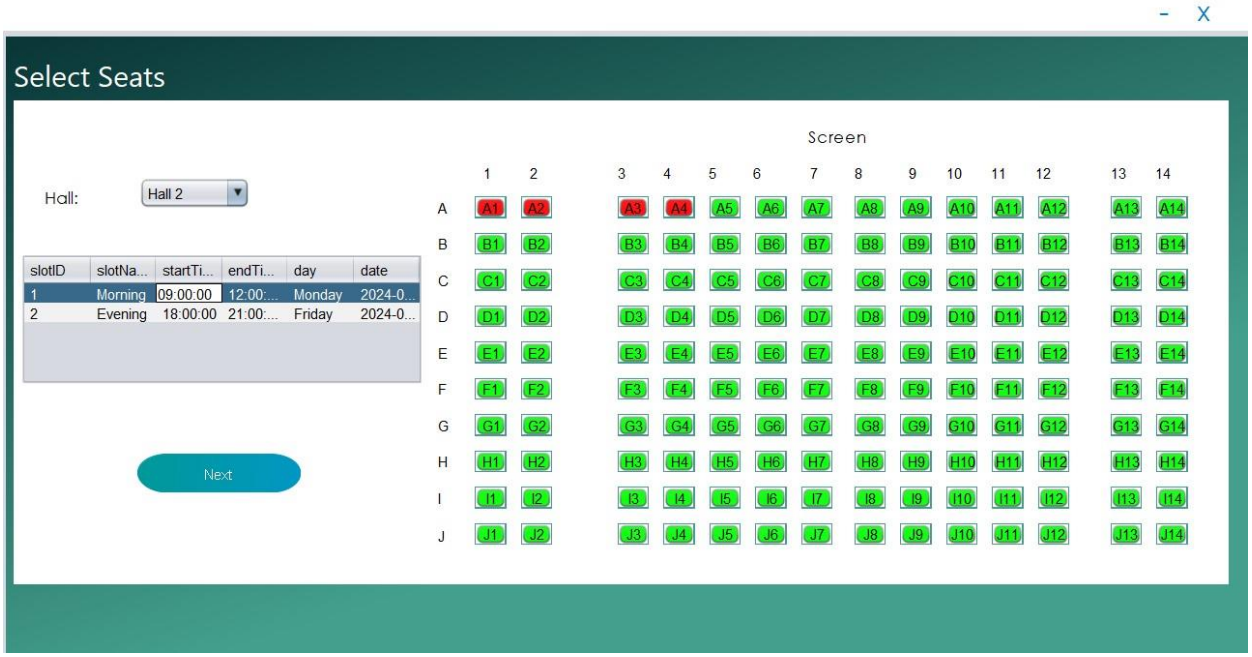
Schedule Management: Enable admins to set and modify movie schedules, including show timings and dates.

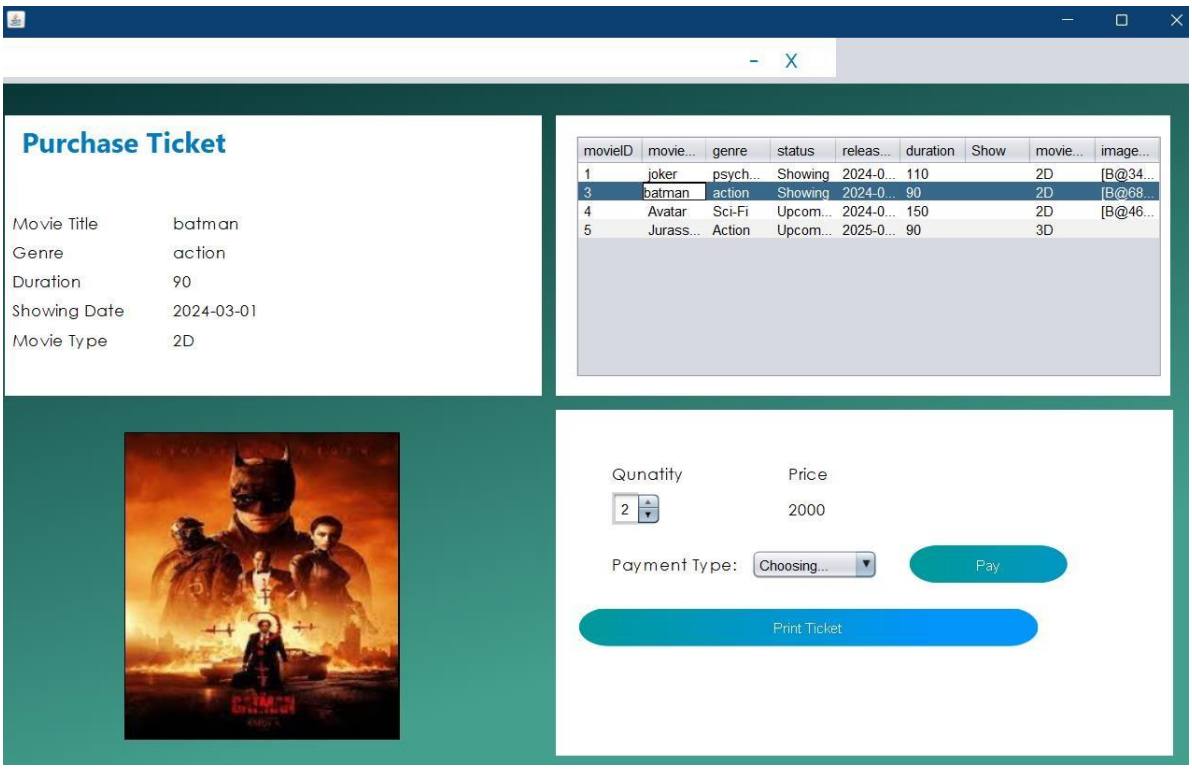
Booking Oversight: Allow administrators to view and manage user bookings, ensuring accurate records.

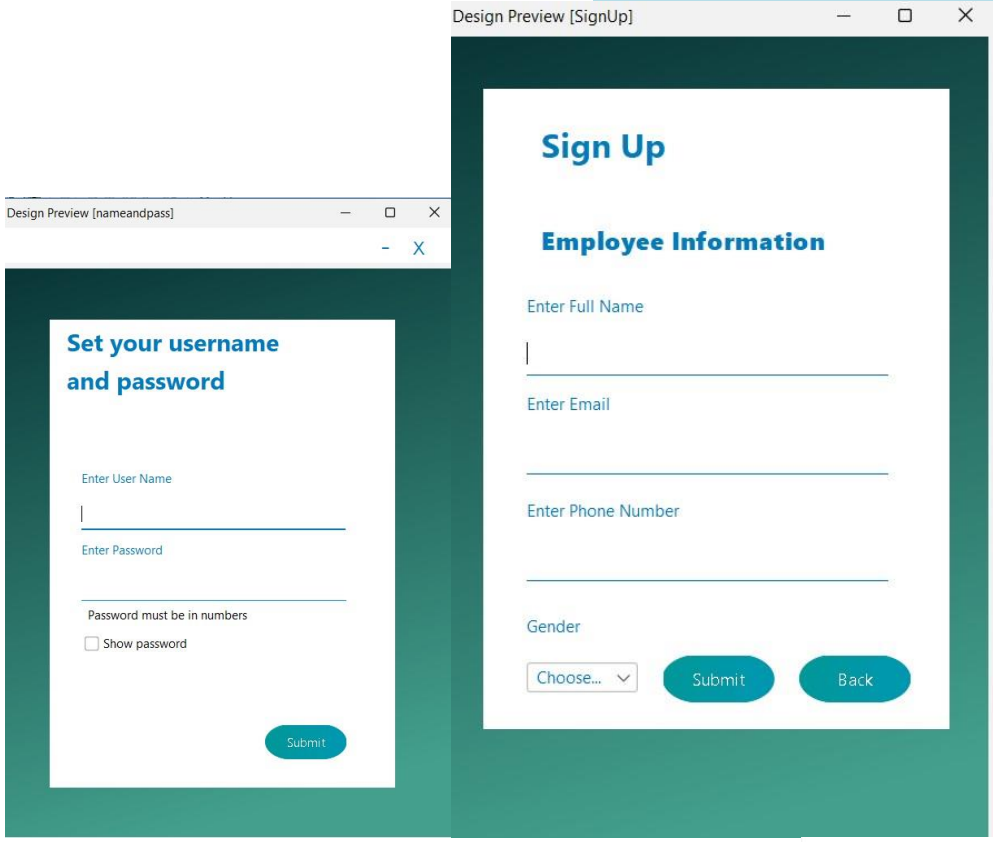
Tracks ticketing: Tracks the amount of ticket has booked **Screenshots:**

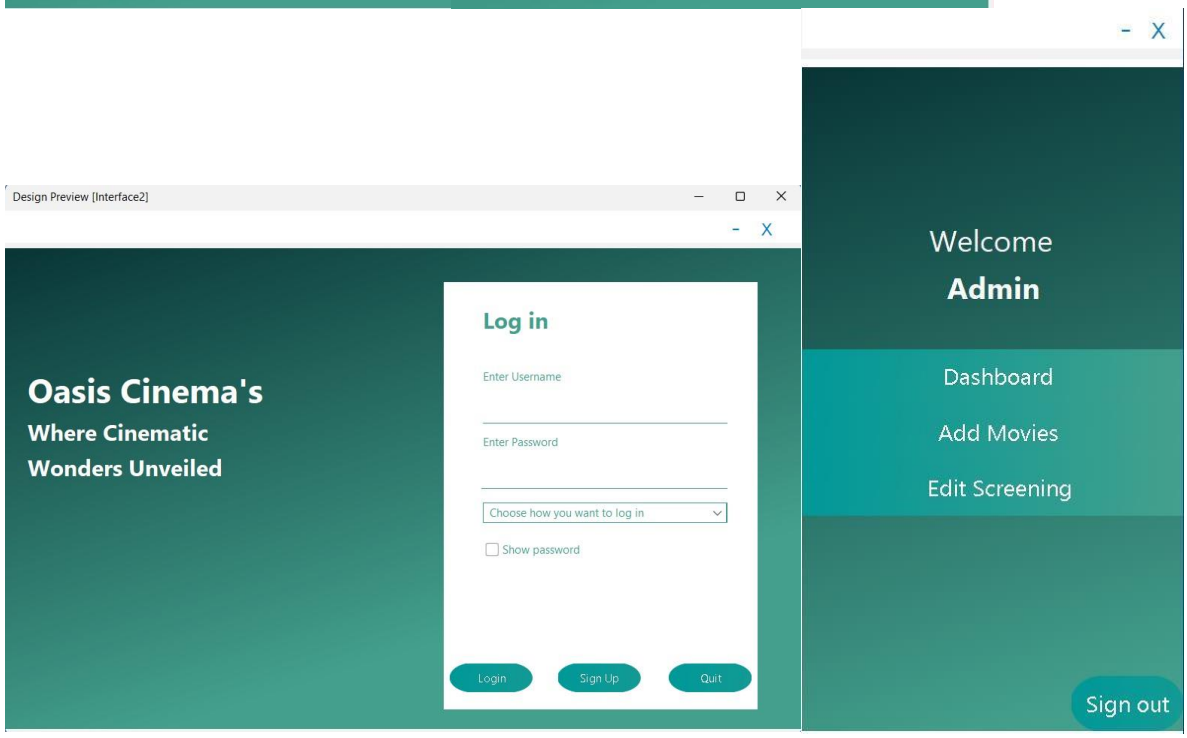





Select Seats
Hall:
Hall 2
slotID slotNa... startTi... endTi... day date
1 Morning 09:00:00 12:00:... Monday 2024-0...
2 Evening 18:00:00 21:00:... Friday 2024-0...
Next
Screen
1 2 3 4 5 6 7 8 9 10 11 12 13 14
A B C D E F G H I J

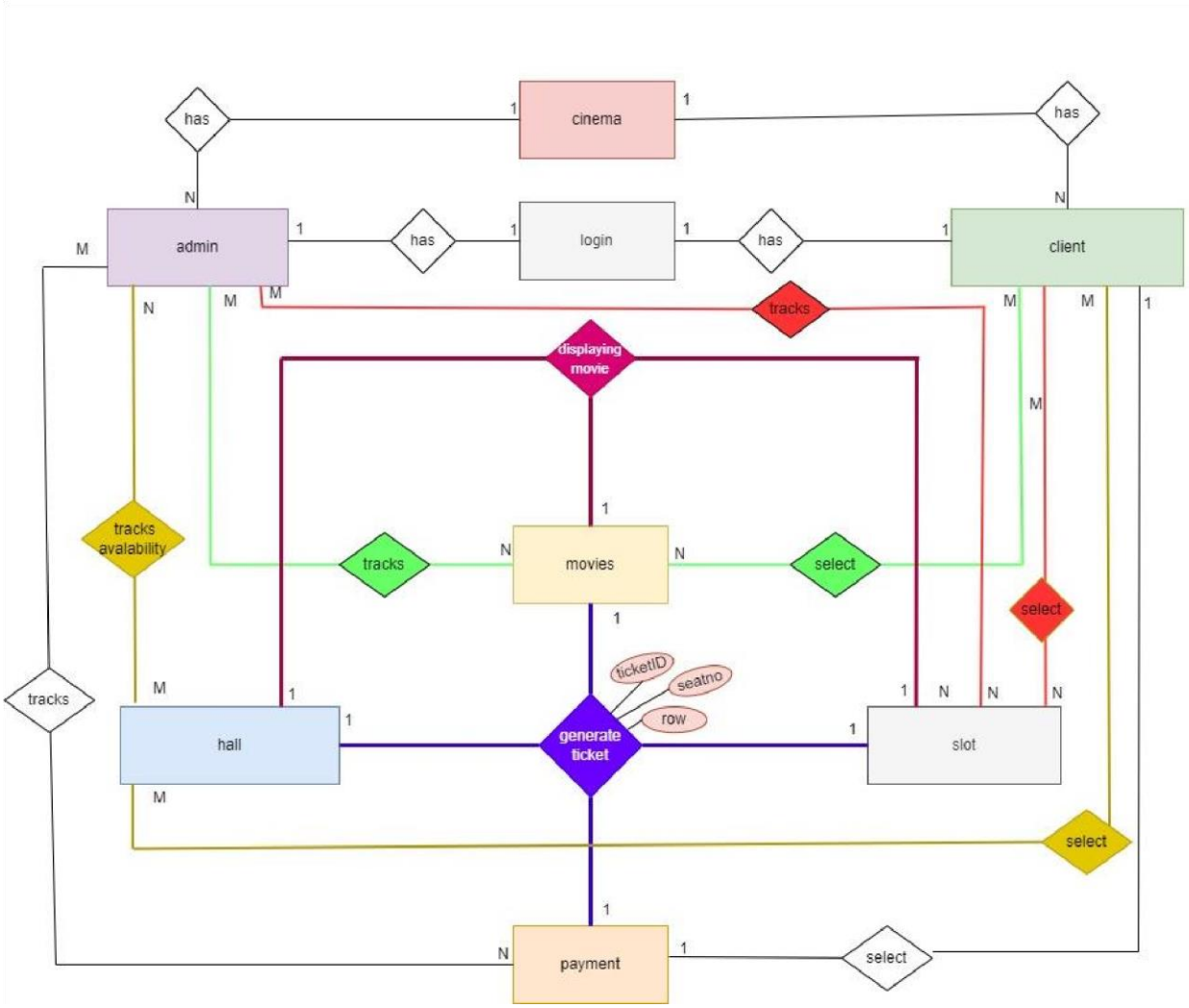

Purchase Ticket
Movie Title batman
Genre action
Duration 90
Showing Date 2024-03-01
Movie Type 2D
movieID movie... genre status releas... duration Show movie... image...
1 joker psych... Showing 2024-0... 110 2D [B@34...
3 batman action Showing 2024-0... 90 2D [B@68...
4 Avatar Sci-Fi Upcom... 2024-0... 150 2D [B@46...
5 Jurass... Action Upcom... 2025-0... 90 3D
Qunatity
Price
2
2000
Payment Type:
Choosing...
Pay
Print Ticket





Methodology:

Entity Relationship Diagram:



Business Rules:

- Each cinema has one or more admin users.
- Each cinema has one or more clients.
- A cinema can have multiple halls.
- Each admin and client must have a single login id.
- Each login id is associated with a single admin or a single client.
- An admin is responsible for tracking the availability of movies and halls.

- An admin can manage multiple movies and halls.
- A client can pay for multiple tickets.
- A client's transaction tracks a payment type.
- Movies are displayed within a cinema and can be tracked by admins.
- Movies have multiple slots in which they are shown.
- A movie can be selected by multiple clients.
- Each hall belongs to a single cinema.
- A hall has multiple slots for movie showings.
- A slot is a specific time period within a hall for showing a movie.
- Clients can select from multiple slots for booking tickets.
- Each payment is made by a client.
- A payment is associated with the booking of a movie slot.
- Each client can make multiple payments for different bookings.
- A ticket is generated for a client when they select a movie slot, movie name and payment.
- If a seat is booked, no other client is able to take that seat
- A hall consist of multiple seats
- Each ticket has specific hall name and seat number following by movie details
- Each ticket corresponds to a specific seat within a hall.
- Each client can have multiple tickets for different movies and slots.

Implementation:

- **Conceptual to Logical Mapping**

loginID ❼ username, password username
❼ password
ClientID ❼ clientName, contact, email, gender, loginID adminID
❼ adminName, contact,email, gender,loginID
movieID ❼ movieName, genre, status, releasedate, movieType, duration, show
paymentID ❼ paymenytAmount, paymentType, ClientID
slotID ❼ slotName, startTime, endTime, day, date hallID
❼ hallName, floor bookingID ❼ date, day, time,
ClientID
ticketID ❼ seatNumber,seatRow, hallID, slotID, movieID, bookingID, paymentID, clientID
movieID, slotID, hallID ❼ x

- **Normalized Tables up to BCNF (SQL Server Schema Diagram)**

1st NF:

All values are atomic ,Keys are identified

loginID ❼ username, password username

❼ password

ClientID ❼ clientName, contact, email, gender, loginID adminID ❼

adminName, contact,email, gender,loginID movieID ❼ movieName, genre,

status, releasedate, , movieType, duration, show

paymentID ❼ paymenytAmount, paymentType, ClientID

slotID ❼ slotName, startTime, endTime, day, date

hallID ❼ hallName, floor

bookingID ❼ date, day, time, ClientID

ticketID ❼ seatNumber,seatRow, hallID, slotID, movieID, bookingID, paymentID, ClientID

movieID, slotID, hallID ❼

Table:

loginInformation (loginID, username, password)

AdminLogin (loginID, username, password) username

(username, password)

Client (ClientID, clientName, contact, email, gender, **loginID**)

Admin (adminID, adminName, contact, email, gender, **loginID**)

Movie (movieID, movieName, genre, status, releasedate, , movieType, duration, show)

Payment (paymentID, paymenytAmount, paymentType, **ClientID)**

Slot (slotID, slotName, startTime, endTime, day, date)

Hall (hallID, hallName, floor)

bookingInfo (bookingID, date, day, time, ClientID)

Ticket (ticketID, seatNumber, seatRow, **hallID, slotID, movieID, bookingID, paymentID, ClientID)**

movieCinema (**movieID, slotID, hallID**)

2nd NF:

-All nonkey attributes should be fully functionally dependent on key

loginInformation (loginID, username, password) AdminLogin

(loginID, username, password) username (username, password)

Client (ClientID, clientName, contact, email, gender, **loginID**)

Admin (adminID, adminName, contact, email, gender, **loginID**)

Movie (movieID, movieName, genre, status, releasedate, , movieType, duration, show)

Payment (paymentID, paymenytAmount, paymentType, **ClientID)**

Slot (slotID, slotName, startTime, endTime, day, date)

Hall (hallID, hallName, floor)

bookingInfo (bookingID, date, day, time, ClientID)

Ticket (ticketID, seatNumber, seatRow, **hallID, slotID, movieID, bookingID, paymentID, ClientID)**

movieCinema (**movieID, slotID, hallID**)

3rd NF:

No non key attribute should determine another non key attribute

New tables are:

loginInformation (loginID, username, password)

Client (ClientID, clientName, contact, email, gender, **loginID**)

Admin (adminID, adminName, contact, email, gender, **loginID**)

Movie (movieID, movieName, genre, status, releasedate, , movieType, duration, show)

Payment (paymentID, paymenytAmount, paymentType, **ClientID)**

Slot (slotID, slotName, startTime, endTime, day, date)

Hall (hallID, hallName, floor)

bookingInfo (bookingID, date, day, time, ClientID)

Ticket (ticketID, seatNumber, seatRow, **hallID, slotID, movieID, bookingID, paymentID, ClientID)**

movieCinema (**movieID, slotID, hallID**)

BCNF:

Already in BCNF

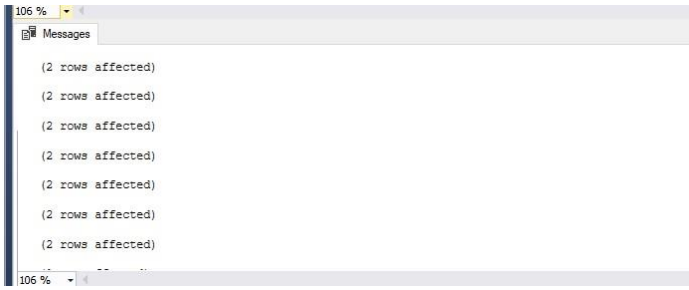
- **Code snippets**

DDL:

```
CREATE TABLE loginInformation (

    loginID INT PRIMARY KEY IDENTITY(1,1),
username VARCHAR(50),     password
VARCHAR(50) );
Create Table AdminLogin(

    loginID INT PRIMARY KEY IDENTITY(1,1),
username VARCHAR(50),     password
VARCHAR(50)
);
CREATE TABLE client (
    clientID INT PRIMARY KEY identity (1,1),
clientName VARCHAR(255),     contact
VARCHAR(20),     email VARCHAR(50),
gender CHAR(10),     loginID INT,
    CONSTRAINT fk_client_login FOREIGN KEY (loginID) REFERENCES loginInformation
(loginID)
);
CREATE TABLE admin (
    adminID INT PRIMARY KEY identity (1,1),
adminName VARCHAR(50),     contact
VARCHAR(20),     gender CHAR(10),
loginID INT,
    CONSTRAINT fk_admin_login FOREIGN KEY (loginID) REFERENCES loginInformation (loginID)
);

CREATE TABLE movie (
    movieID INT PRIMARY KEY identity(1,1),
movieName VARCHAR(80),     genre
VARCHAR(50),     status VARCHAR(20),
    releaseDate DATE,
duration int,        Show
VARCHAR(50),
movieType VARCHAR(50),
imageData IMAGE
);
```

```
CREATE TABLE
payment (
    paymentID INT PRIMARY KEY identity (1,1),
paymentAmount int,     paymentType
VARCHAR(50),     clientID INT,
    CONSTRAINT fk_payment_client FOREIGN KEY (clientID) REFERENCES client(clientID)
);

CREATE TABLE slot (
    slotID INT PRIMARY KEY Identity (1,1),
       slotName
VARCHAR(30),     startTime
TIME,     endTime TIME,
day VARCHAR(20),     date
DATE,
    movieID INT,
       status VARCHAR(30),
    CONSTRAINT fk_slot_movie FOREIGN KEY (movieID) REFERENCES movie(movieID)
);

CREATE TABLE hall (
    hallID INT PRIMARY KEY identity (1,1),
hallName VARCHAR(50),     floor INT
);

CREATE TABLE bookingInfo (
    bookingID INT PRIMARY KEY identity(1,1),
date DATE,     day VARCHAR(20),     time
TIME, paymentid int,
  CONSTRAINT fk_bo_info FOREIGN KEY (paymentid) REFERENCES payment(paymentid),

);
create table ticket
(
ticketid  int primary key identity (1,1),
seatnumber int, seatRow varchar (50),
hallid int, slotid int, movieid int,
bookingid int, paymentid int, clientid
int,
 CONSTRAINT fk_ct_client FOREIGN KEY (clientID) REFERENCES client(clientID),
    CONSTRAINT fk_ct_booking FOREIGN KEY (bookingID) REFERENCES bookingInfo(bookingID),
    CONSTRAINT fk_ct_movie FOREIGN KEY (movieID) REFERENCES movie(movieID),
    CONSTRAINT fk_ct_payment FOREIGN KEY (paymentID) REFERENCES payment(paymentID),
    CONSTRAINT fk_ct_slot FOREIGN KEY (slotID) REFERENCES slot(slotID),
    CONSTRAINT fk_ct_hall FOREIGN KEY (hallID) REFERENCES hall(hallID)
)

CREATE TABLE cinemamovie(
movieID INT,     slotID
INT,     hallID INT,
    CONSTRAINT fk_cinema_movie FOREIGN KEY (movieID) REFERENCES movie(movieID),
    CONSTRAINT fk_cinema_slot FOREIGN KEY (slotID) REFERENCES slot(slotID),
CONSTRAINT fk_cinema_hall FOREIGN KEY (hallID) REFERENCES hall(hallID),
);
INSERT INTO loginInformation (username, password)
VALUES ('hania', '123'),
       ('abdullah', '123');
```

```
INSERT
INTO loginInformation (username, password)
VALUES
 ('faiz', '123'),
  ('zainab', '123');

INSERT INTO client (clientName, contact, email, gender)
VALUES ('hania', '1234567890', 'hania@example.com', 'feMale'),
       ('abdullah', '0987654321', 'abdullah@example.com', 'male');

INSERT INTO admin (adminName, contact, gender, loginID)
VALUES ('faiz', '1231231234', 'Male', 3),
       ('zainab', '3213214321', 'Female', 4);


INSERT INTO movie (movieName, genre, status, releaseDate,movieType,duration, show)
VALUES ('avengers', 'Action', 'Now showing', '2024-01-01', ‘2D’ , ‘3 hours’,),
       ('nun', 'horror', 'coming soon', '2024-04-04');
INSERT INTO payment (paymentAmount, paymentType, clientID)
VALUES (1200, 'Card', 1),
       (1200, 'cash', 2);

INSERT INTO slot (slotName, startTime, endTime, day, date, movieID,status) VALUES
('Morning', '09:00:00', '12:00:00', 'Monday', '2024-01-01', 1,'Booked'),
('Evening', '18:00:00', '21:00:00', 'Friday', '2024-01-03', 2,'Not Booked')
INSERT INTO hall (hallName, floor)
VALUES ('Main Hall', 1);

INSERT INTO bookingInfo (date, day, time, clientid)
VALUES ('2024-01-01', 'mnday', '14:00:00', 1);

INSERT INTO ticket (seatnumber, seatRow, hallid, slotid, movieid, bookingid, paymentid,
clientid)
VALUES (12, 'B', 1, 1, 1, 1, 1, 1);

INSERT INTO cinemamovie (movieID, slotID, hallID)
VALUES (1, 1, 1);
```



Stored Procedure:

```
CREATE PROCEDURE InsertMovie
    @Title NVARCHAR(255),
    @Genre NVARCHAR(255),
    @Status NVARCHAR(255),
    @ReleaseDate Date,
       @Duration INT,
```

```
        @Show NVARCHAR(50),
        @movieType NVARCHAR(20),
    @ImageData VARBINARY(MAX)
AS
BEGIN
    IF @movieType = 'Both'
BEGIN
        -- Insert 2D movie
        INSERT INTO movie (movieName, genre, status, releaseDate, duration, Show,
movieType, imageData)
        VALUES (@Title, @Genre, @Status, FORMAT(@ReleaseDate, 'dd/MM/yyyy'), @Duration,
@Show, '2D', @ImageData);

        -- Insert 3D movie
        INSERT INTO movie (movieName, genre, status, releaseDate, duration, Show,
movieType, imageData)
        VALUES (@Title, @Genre, @Status, FORMAT(@ReleaseDate, 'dd/MM/yyyy'), @Duration,
@Show, '3D', @ImageData);
    END
    ELSE
    BEGIN
        -- Insert a single movie
        INSERT INTO movie (movieName, genre, status, releaseDate, duration, Show,
movieType, imageData)
        VALUES (@Title, @Genre, @Status, FORMAT(@ReleaseDate, 'dd/MM/yyyy'), @Duration,
@Show, @movieType, @ImageData);
    END
END;
```

```
CREATE PROCEDURE UpdateMovie
    @MovieId INT,
    @Title NVARCHAR(255),
    @Genre NVARCHAR(255),
    @Status NVARCHAR(255),
    @ReleaseDate Date,
    @Duration INT,
    @Show NVARCHAR(50),
    @MovieType NVARCHAR(20),
    @ImageData VARBINARY(MAX)
AS
BEGIN
    UPDATE movie
SET
        movieName = @Title,
genre = @Genre,         status =
@Status,         releaseDate =
@releaseDate ,         duration =
@Duration,         Show = @Show,
movieType = @MovieType,
imageData = @ImageData
    WHERE movieId = @MovieId;
END;
```

```
s
CREATE
PROCEDURE GetAllSlots
AS
BEGIN
    SELECT * FROM slot;
END;
```

```
CREATE TABLE slot (
    slotID INT PRIMARY KEY Identity (1,1),
       slotName
VARCHAR(30),     startTime
TIME,     endTime TIME,
day VARCHAR(20),     date
DATE,
   );


CREATE TABLE loginInformation (
loginID INT PRIMARY KEY IDENTITY(1,1),
username VARCHAR(50),     password
VARCHAR(50)
);

CREATE TABLE client (
    clientID INT PRIMARY KEY IDENTITY(1,1) ,
clientName VARCHAR(255),     contact
VARCHAR(20),     email VARCHAR(50),
gender CHAR(10),     loginID INT,
    CONSTRAINT fk_client_login FOREIGN KEY (loginID) REFERENCES loginInformation
(loginID)
);
```

```
CREATE PROCEDURE InsertLogin
    @Username NVARCHAR(255),
    @Password NVARCHAR(255)
AS
BEGIN
    INSERT INTO loginInformation(username, password)
    VALUES (@Username, @Password);
END;

CREATE PROCEDURE GetLoginIdByUsername
    @Username NVARCHAR(255),
    @LoginId INT OUTPUT
AS
BEGIN
    SELECT @LoginId = LoginId
    FROM loginInformation
    WHERE Username = @Username;
END;
```

```
CREATE PROCEDURE InsertClient
    @ClientName VARCHAR(255),
    @Contact VARCHAR(20),
    @Email VARCHAR(50),
    @Gender CHAR(10),
    @LoginID INT
AS
BEGIN
    INSERT INTO client (clientName, contact, email, gender, loginID)
    VALUES (@ClientName, @Contact, @Email, @Gender, @LoginID);
END; delete client

exec InsertClient 'abdullah khan','03101211','email','male',1
```

```
CREATE PROCEDURE GetClientid
    @Username NVARCHAR(255),
    @ClientidId INT OUTPUT
AS
BEGIN
    SELECT @ClientidId = clientId
    FROM client
    WHERE clientName = @Username;
END;
```

```
CREATE PROCEDURE InsertPayment
    @PaymentAmount INT,
    @PaymentType VARCHAR(50),
    @ClientID INT
AS
BEGIN
       DECLARE @CurrentDate DATE = GETDATE();
    DECLARE @CurrentDay VARCHAR(20) = FORMAT(GETDATE(), 'dddd', 'en-US');
    DECLARE @CurrentTime TIME = FORMAT(GETDATE(), 'hh:mm:ss');
       DECLARE @paymentid int;

    INSERT INTO payment (paymentAmount, paymentType, clientID)
    VALUES (@PaymentAmount, @PaymentType, @ClientID);
       select @paymentid= paymentid from payment where clientid=
@ClientID
    INSERT INTO bookingInfo (date, day, time, paymentid)
    VALUES (@CurrentDate, @CurrentDay, @CurrentTime, @PaymentID);

END;
```

```
CREATE PROCEDURE InsertBookingInfo
    @PaymentID INT
AS
```

```
s
BEGIN
DECLARE @CurrentDate DATE = GETDATE();
    DECLARE @CurrentDay VARCHAR(20) = FORMAT(GETDATE(), 'dddd', 'en-US');
    DECLARE @CurrentTime TIME = FORMAT(GETDATE(), 'hh:mm:ss');

    INSERT INTO bookingInfo (date, day, time, paymentid)
    VALUES (@CurrentDate, @CurrentDay, @CurrentTime, @PaymentID);
END;
```

```
CREATE PROCEDURE GetHallID
    @HallName NVARCHAR(255),
    @HallID INT OUTPUT
AS
BEGIN
    SELECT @HallID = hallID
    FROM hall
    WHERE hallName = @hallName;
END;
```

```
CREATE PROCEDURE InsertCinemaMovie
    @movieID INT,
    @slotID INT,
    @hallID INT
AS
BEGIN
    INSERT INTO CinemaMovie (movieID, slotID, hallID)
    VALUES (@movieID, @slotID, @hallID);
END;
```

```
CREATE PROCEDURE GetSeatStatus
@movieid int,
    @hallID INT,
    @slotID INT,
    @seatID INT,
    @Seatrow VARCHAR(50),
    @Status VARCHAR(50) OUTPUT
AS
BEGIN
    SELECT @Status = stats
    FROM clientTicket
    WHERE hallID = @hallID
        AND slotID = @slotID
        AND seatID = @seatID
        AND Seatrow = @Seatrow
              AND movieid = @movieid;
END;
```

```
CREATE PROCEDURE GetSlotInfo
```

```
@SlotID INT,
    @SlotName NVARCHAR(50) OUTPUT,
    @Date DATE OUTPUT,
    @startTime TIME OUTPUT,
       @endTime TIME OUTPUT,
    @Day NVARCHAR(20) OUTPUT
AS
BEGIN
    SELECT @SlotName = slotName,
       @endTime = endtime,
           @Date = date,
           @startTime = startTime,
           @Day = day
    FROM slot
    WHERE slotID = @SlotID;
END;
```

```
CREATE PROCEDURE InsertClientTicket
    @ClientID INT,
    @MovieID INT,
    @SlotID INT,
    @SeatRow VARCHAR(50),
    @SeatID INT,
    @HallID INT
AS
BEGIN
    DECLARE @PaymentID INT;
    DECLARE @BookingID INT;

    -- Retrieve payment ID for the client
    SELECT @PaymentID = paymentid FROM payment WHERE clientID = @ClientID;
    -- Retrieve booking ID based on the payment ID
    SELECT @BookingID = bookingid FROM bookingInfo WHERE paymentid = @PaymentID;

        INSERT INTO clientTicket (clientID, bookingID, movieID, paymentID, slotID,
Seatrow, seatID, hallID, stats)
        VALUES (@ClientID, @BookingID, @MovieID, @PaymentID, @SlotID, @SeatRow, @SeatID,
@HallID, 'Booked');

END;
```

```
CREATE PROCEDURE CheckUserLogin
    @Username NVARCHAR(255),
    @Password NVARCHAR(255),
    @IsValid BIT OUTPUT
AS
BEGIN
    SET @IsValid = 0; -- Initialize the output parameter to false
    IF EXISTS (
        SELECT 1
        FROM loginInformation
        WHERE Username = @Username AND Password = @Password
    )
```

```
BEGIN
SET @IsValid = 1; -- Set to true if the user exists with the given credentials     END
END;
```

```
CREATE PROCEDURE CheckUserLogin
    @Username NVARCHAR(255),
    @Password NVARCHAR(255),
    @IsValid BIT OUTPUT
AS
BEGIN
    SET @IsValid = 0; -- Initialize the output parameter to false

    IF EXISTS (
        SELECT 1
        FROM loginInformation
        WHERE Username = @Username AND Password = @Password
    )
    BEGIN
        SET @IsValid = 1; -- Set to true if the user exists with the given credentials
END
END;
```

```
CREATE PROCEDURE CheckAdminLogin
    @Username NVARCHAR(255),
    @Password NVARCHAR(255),
    @IsValid BIT OUTPUT
AS
BEGIN
    SET @IsValid = 0; -- Initialize the output parameter to false

    IF EXISTS (
        SELECT 1
        FROM adminlogin
        WHERE Username = @Username AND Password = @Password
)
    BEGIN
        SET @IsValid = 1; -- Set to true if the user exists with the given credentials
END
END;
```

```
CREATE PROCEDURE CountTickets
    @TicketCount INT OUTPUT
AS
BEGIN
    SELECT @TicketCount = COUNT(*)
    FROM YourTicketTable;
END;
```

```
CREATE PROCEDURE GetPaymentID
    @ClientID INT,
    @BookingID INT
AS
BEGIN
    SELECT paymentID
    FROM clientTicket
    WHERE clientID = @ClientID AND bookingID = @BookingID;
END;
```

```
CREATE PROCEDURE InsertBookingInfo

AS
BEGIN
    DECLARE @LastPaymentID INT;

    -- Fetch the last payment ID
    SELECT TOP 1 @LastPaymentID = paymentID
    FROM payment
    ORDER BY paymentID DESC;

    -- Insert into bookingInfo table
    INSERT INTO bookingInfo (date, day, time, paymentID)
    SELECT GETDATE(), FORMAT(GETDATE(), 'dddd'), FORMAT(GETDATE(), 'HH:mm:ss'),
@LastPaymentID
    WHERE @LastPaymentID IS NOT NULL;
END;
```

```
CREATE PROCEDURE GetStudnfo
    @StudentID INT,
  @studentname NVARCHAR(50) OUTPUT,
    @DateofBirth DATE OUTPUT,
   @gender NVARCHAR(50) OUTPUT,
       @contact NVARCHAR(50) OUTPUT,
  @emailaddress NVARCHAR(20) OUTPUT,
       @address NVARCHAR(20) OUTPUT,
       @admission NVARCHAR(20) OUTPUT,
       @classid int,
       @guardianName NVARCHAR(20) OUTPUT,
       @guardianAddress NVARCHAR(20) OUTPUT,
       @guardianContact NVARCHAR(20) OUTPUT
AS
BEGIN
SELECT
       studentID = @studentid,
@studentname= name,
       @DateofBirth= dateOfBirth,
       @gender = gender,
        @contact=contact,
```

```
@emailaddress=   emailAddress,
      @address=  address,
       @admission= admissionDate,
     @classid=   classID,
       @guardianName =  guardianName,
      @guardianAddress=  guardianAddress,
      @guardianContact=  guardianContact
    FROM
        Student
WHERE
        studentID = @StudentID;

END;
```

```
CREATE PROCEDURE GetClientIDByUsername
    @Username NVARCHAR(255),
    @ClientID INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    -- Declare a variable to store loginID
    DECLARE @LoginID INT;

    -- Get loginID using the provided username
    SELECT @LoginID = loginID
    FROM loginInformation
    WHERE username = @Username;

    -- Fetch clientID using the obtained loginID
    SELECT @ClientID = clientId
    FROM client
    WHERE loginid = @LoginID;
END;
```

```
create procedure GetMovies
as Begin
select movieid,moviename,genre,status,releaseDate,duration,movieType,imageData from Movie
End
```

Triggers:

```
CREATE TRIGGER trg_Client_Deletion
ON client
INSTEAD OF DELETE
```

```
s
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM bookingInfo WHERE clientid IN (SELECT clientID FROM deleted);
    DELETE FROM ticket WHERE clientid IN (SELECT clientID FROM deleted);
    DELETE FROM payment WHERE clientID IN (SELECT clientID FROM deleted);
    DELETE FROM client WHERE clientID IN (SELECT clientID FROM deleted); END;
```

```
CREATE TRIGGER trg_Movie_Deletion
ON movie
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM cinemamovie WHERE movieID IN (SELECT movieID FROM deleted);
    DELETE FROM ticket WHERE movieid IN (SELECT movieID FROM deleted);
    DELETE FROM movie WHERE movieID IN (SELECT movieID FROM deleted); END;
```

```
CREATE TRIGGER trg_Prevent_Invalid_Payments
ON payment
INSTEAD OF INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE i.paymentAmount <= 0
    )
    BEGIN
        RAISERROR('Payment amount must be greater than 0.', 16, 1);
END;
    ELSE
    BEGIN
        INSERT INTO payment (paymentAmount, paymentType, clientID)
        SELECT paymentAmount, paymentType, clientID
        FROM inserted;
    END;
END;
```

```
Msg 50000, Level 16, State 1, Procedure trg_Prevent_Invalid_Payments, Line 14 [Batch Start Line 0]
Payment amount must be greater than 0.

(1 row affected)

Completion time: 2024-01-07T17:26:35.8541582+05:00
```

```
CREATE TRIGGER cascade_delete_tickets
ON slot
AFTER DELETE
AS
BEGIN
    DELETE FROM ticket
    WHERE slotID IN (SELECT slotID FROM deleted); END;
```

```
s
CREATE
TRIGGER cascade_delete_hall_relations
ON hall
AFTER DELETE
AS
BEGIN
    DELETE FROM cinemamovie WHERE hallID IN (SELECT hallID FROM deleted);
    DELETE FROM ticket WHERE hallID IN (SELECT hallID FROM deleted);
END;
```

```
CREATE TRIGGER trg_Check_Overlapping_Slots
ON slot
INSTEAD OF INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE EXISTS (
            SELECT 1
            FROM slot s
            WHERE s.day = i.day
              AND (
                  (s.startTime <= i.endTime AND s.endTime >= i.startTime)
                  OR (s.startTime >= i.startTime AND s.startTime <= i.endTime)
              )
              AND s.slotID <> i.slotID -- Exclude the current slot for updates
        )
    )
    BEGIN
        RAISERROR('Overlapping time slots are not allowed.', 16, 1);
    END;
    ELSE
    BEGIN
        UPDATE s
        SET
            s.slotName = i.slotName,
            s.startTime = i.startTime,
            s.endTime = i.endTime,
            s.day = i.day,
            s.date = i.date
        FROM inserted i
        INNER JOIN slot s ON i.slotID = s.slotID;

        INSERT INTO slot (slotName, startTime, endTime, day, date)
        SELECT slotName, startTime, endTime, day, date
        FROM inserted
        WHERE NOT EXISTS (SELECT 1 FROM slot s WHERE s.slotID = inserted.slotID);
END;
END;
```

```
CREATE TRIGGER trg_PreventSlotAndDuplicateUsage
ON cinemamovie
```

```
INSTEAD OF INSERT
AS
BEGIN
    -- Check if any new slotID being inserted has been used 5 or more times
    IF EXISTS (
        SELECT 1
        FROM (
            SELECT slotID, COUNT(*) AS UsageCount
            FROM inserted
            GROUP BY slotID
        ) AS SlotUsageInfo
        WHERE EXISTS (
            SELECT 1
            FROM cinemamovie cm
            WHERE cm.slotID = SlotUsageInfo.slotID
            GROUP BY cm.slotID
            HAVING COUNT(*) + SlotUsageInfo.UsageCount > 5
        )
)
    BEGIN
        RAISERROR('Cannot insert slotID more than 5 times in cinemamovie table.', 16, 1);
END
    ELSE
    BEGIN
        -- Logic to ensure foreign key constraints
        IF NOT EXISTS (SELECT 1 FROM inserted i
                       JOIN movie m ON i.movieID = m.movieID
                       JOIN slot s ON i.slotID = s.slotID
                       JOIN hall h ON i.hallID = h.hallID)
        BEGIN
            RAISERROR('Invalid movie, slot, or hall ID.', 16, 1);
ROLLBACK;
        END
        ELSE
        BEGIN
            -- Insert valid data into cinemamovie
            INSERT INTO cinemamovie (movieID, slotID, hallID)
            SELECT movieID, slotID, hallID FROM inserted;

            -- Logic to prevent duplicate movie schedules
            IF EXISTS (
                SELECT 1
                FROM cinemamovie
                WHERE movieID = (SELECT movieID FROM inserted)
                    AND slotID IN (SELECT slotID FROM inserted)
                    AND hallID IN (SELECT hallID FROM inserted)
)
            BEGIN
                RAISERROR('Cannot schedule the same movie in the same hall at the same
time.', 16, 1);
                ROLLBACK TRANSACTION;
            END;
        END
    END
END;
```

```
s
CREATE
TRIGGER trg_ticket_before_insert
ON ticket
INSTEAD OF INSERT
AS
BEGIN
    -- Logic to ensure foreign key constraints and unique combinations
    IF NOT EXISTS (SELECT 1 FROM inserted i
                   JOIN client c ON i.clientID = c.clientID
                   JOIN bookingInfo b ON i.bookingID = b.bookingID
                   JOIN movie m ON i.movieID = m.movieID
                   JOIN payment p ON i.paymentID = p.paymentID
                   JOIN slot s ON i.slotID = s.slotID
                   JOIN hall h ON i.hallID = h.hallID
                   WHERE NOT EXISTS (SELECT 1 FROM ticket t
                                     WHERE t.hallID = i.hallID
                                       AND t.slotID = i.slotID
                                       AND t.seatnumber = i.seatnumber))
    BEGIN
        RAISERROR('Invalid client, booking, movie, payment, slot, or hall ID, or duplicate
seat booking.', 16, 1);
        ROLLBACK;
    END
    ELSE
    BEGIN
        -- Insert valid data into ticket
        INSERT INTO ticket (seatnumber, seatRow, hallid, slotid, movieid, bookingid,
paymentid, clientid)
        SELECT seatnumber, seatRow, hallid, slotid, movieid, bookingid, paymentid,
clientid FROM inserted;
    END
END;
```

```
CREATE TABLE auditlog (
    logid INT PRIMARY KEY IDENTITY(1,1),
adminid INT,     adminName VARCHAR(50),
tablename VARCHAR(50),     action
VARCHAR(10),     manipulationtime
DATETIME
);

-- Table 2: Activitylog CREATE TABLE
activitylog (     logid INT PRIMARY KEY
IDENTITY(1,1),     bookingid INT,
ticketid INT,
    clientid INT,
clientname VARCHAR(255),
movieid INT,     moviename
VARCHAR(80),     slotid INT,
slotname VARCHAR(50),
seatid INT,     seatnumber
INT,     seatrow
VARCHAR(50),     paymentid
INT,     paymentamount INT,
```

```
hallid INT,     hallname
VARCHAR(50),     floor INT,
    manipulationtime DATETIME,
action VARCHAR(10)
);


CREATE TRIGGER client_trigger
ON client
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @action VARCHAR(10);

    IF EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted)
        SET @action = 'UPDATE';
    ELSE IF EXISTS(SELECT * FROM inserted)
        SET @action = 'INSERT';
    ELSE
        SET @action = 'DELETE';

    INSERT INTO auditlog (adminid, adminName, tablename, action, manipulationtime)
    SELECT
        ISNULL((SELECT adminID FROM admin WHERE loginID = USER_ID()), 0),
        ISNULL((SELECT adminName FROM admin WHERE loginID = USER_ID()), 'Unknown'),
'client',          @action,
        GETDATE();

    -- Insert into activitylog for specific details
    INSERT INTO activitylog (clientid, clientname, manipulationtime, action, ticketid)
SELECT
        ISNULL(i.clientID, d.clientID),
        ISNULL(i.clientName, d.clientName),
        GETDATE(),
        @action,
        NULL
    FROM (SELECT * FROM inserted) i
    FULL JOIN (SELECT * FROM deleted) d ON i.clientID = d.clientID; END;



ALTER TRIGGER loginInformation_trigger
ON loginInformation
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @action VARCHAR(10);

    IF EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted)
        SET @action = 'UPDATE';
    ELSE IF EXISTS(SELECT * FROM inserted)
        SET @action = 'INSERT';
    ELSE IF EXISTS(SELECT * FROM deleted)
        SET @action = 'DELETE';

    INSERT INTO auditlog (adminid, adminName, tablename, action, manipulationtime)
```

```
VALUES
(
        ISNULL((SELECT adminID FROM admin WHERE loginID = USER_ID()), 0),
        ISNULL((SELECT adminName FROM admin WHERE loginID = USER_ID()), 'Unknown'),
        'loginInformation',
        @action,
        GETDATE()
    );
END;
```

```
CREATE TRIGGER movie_trigger ON
movie
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @action VARCHAR(10);

    IF EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted)
        SET @action = 'UPDATE';
    ELSE IF EXISTS(SELECT * FROM inserted)
        SET @action = 'INSERT';
    ELSE IF EXISTS(SELECT * FROM deleted)
        SET @action = 'DELETE';

    INSERT INTO auditlog (adminid, adminName, tablename, action, manipulationtime)
VALUES (
        ISNULL((SELECT adminID FROM admin WHERE loginID = USER_ID()), 0),
        ISNULL((SELECT adminName FROM admin WHERE loginID = USER_ID()), 'Unknown'),
'movie',
        @action,
        GETDATE()
    );

    -- Insert into activitylog for specific details
    INSERT INTO activitylog (movieid, moviename, manipulationtime, action)
SELECT
        COALESCE(i.movieID, d.movieID),
        COALESCE(i.movieName, d.movieName),
        GETDATE(),
        @action
    FROM (SELECT movieID, movieName FROM inserted) i
    FULL JOIN (SELECT movieID, movieName FROM deleted) d ON i.movieID = d.movieID; END;
```

```
CREATE TRIGGER payment_trigger
ON payment
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @action VARCHAR(10);

    IF EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted)
        SET @action = 'UPDATE';
    ELSE IF EXISTS(SELECT * FROM inserted)
        SET @action = 'INSERT';
ELSE
```

```
SET
@action = 'DELETE';

    INSERT INTO auditlog (adminid, adminName, tablename, action, manipulationtime)
    VALUES (ISNULL((SELECT adminID FROM admin WHERE loginID = USER_ID()), 0),
            ISNULL((SELECT adminName FROM admin WHERE loginID = USER_ID()), 'Unknown'),
            'payment',
            @action,
            GETDATE());

    -- Insert into activitylog for specific details
    INSERT INTO activitylog (paymentid, paymentamount, manipulationtime, action,
ticketid)     SELECT
        ISNULL(i.paymentID, d.paymentID),
        ISNULL(CONVERT(VARCHAR, i.paymentAmount), CONVERT(VARCHAR, d.paymentAmount)),
        GETDATE(),
        @action,
        NULL
    FROM (SELECT paymentID, paymentAmount FROM inserted) i
    FULL JOIN (SELECT paymentID, paymentAmount FROM deleted) d ON i.paymentID =
d.paymentID; END;
```

```
CREATE TRIGGER slot_trigger
ON slot
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @action VARCHAR(10);

    IF EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted)
        SET @action = 'UPDATE';
    ELSE IF EXISTS(SELECT * FROM inserted)
        SET @action = 'INSERT';
ELSE
        SET @action = 'DELETE';

    INSERT INTO auditlog (adminid, adminName, tablename, action, manipulationtime)
    VALUES (ISNULL((SELECT adminID FROM admin WHERE loginID = USER_ID()), 0),
            ISNULL((SELECT adminName FROM admin WHERE loginID = USER_ID()), 'Unknown'),
'slot',
            @action,
            GETDATE());

    -- Insert into activitylog for specific details
    INSERT INTO activitylog (slotid, slotname, manipulationtime, action, ticketid)
SELECT
        ISNULL(i.slotID, d.slotID),
        ISNULL(i.slotName, d.slotName),
        GETDATE(),
        @action,
        NULL
    FROM (SELECT slotID, slotName FROM inserted) i
    FULL JOIN (SELECT slotID, slotName FROM deleted) d ON i.slotID = d.slotID; END;
```

```
CREATE TRIGGER hall_trigger
```

```
s
ON
hall
AFTER
INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @action VARCHAR(10);

    IF EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted)
        SET @action = 'UPDATE';
    ELSE IF EXISTS(SELECT * FROM inserted)
        SET @action = 'INSERT';
    ELSE
        SET @action = 'DELETE';

    INSERT INTO auditlog (adminid, adminName, tablename, action, manipulationtime)
SELECT
        ISNULL(i.hallID, d.hallID),
        ISNULL(i.hallName, d.hallName),
        'hall',
        @action,
        GETDATE()
    FROM (SELECT * FROM inserted) i
    FULL JOIN (SELECT * FROM deleted) d ON i.hallID = d.hallID;
    -- Insert into activitylog for specific details
    INSERT INTO activitylog (hallid, hallname, floor, manipulationtime, action, ticketid)
SELECT
        ISNULL(i.hallID, d.hallID),
        ISNULL(i.hallName, d.hallName),
        ISNULL(i.floor, d.floor),
        GETDATE(),
        @action,
        NULL
    FROM (SELECT * FROM inserted) i
    FULL JOIN (SELECT * FROM deleted) d ON i.hallID = d.hallID; END;
```

```
CREATE TRIGGER bookingInfo_trigger
ON bookingInfo
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @action VARCHAR(10);

    IF EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted)
        SET @action = 'UPDATE';
    ELSE IF EXISTS(SELECT * FROM inserted)
        SET @action = 'INSERT';
    ELSE
        SET @action = 'DELETE';

    INSERT INTO auditlog (adminid, adminName, tablename, action, manipulationtime)
SELECT
        ISNULL(i.bookingID, d.bookingID),
        'N/A',
        'bookingInfo',
        @action,
        GETDATE()
```

```
FROM
(SELECT * FROM inserted) i
    FULL JOIN (SELECT * FROM deleted) d ON i.bookingID = d.bookingID;

    -- Insert into activitylog for specific details
    INSERT INTO activitylog (bookingid, clientid, manipulationtime, action, ticketid)
SELECT
        ISNULL(i.bookingID, d.bookingID),
        ISNULL(i.clientID, d.clientID),
        GETDATE(),
        @action,
        NULL
    FROM (SELECT * FROM inserted) i
    FULL JOIN (SELECT * FROM deleted) d ON i.bookingID = d.bookingID; END;
```

```
CREATE TRIGGER ticket_trigger
ON ticket
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @action VARCHAR(10);

    IF EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted)
        SET @action = 'UPDATE';
    ELSE IF EXISTS(SELECT * FROM inserted)
        SET @action = 'INSERT';
    ELSE IF EXISTS(SELECT * FROM deleted)
        SET @action = 'DELETE';

    INSERT INTO auditlog (adminid, adminName, tablename, action, manipulationtime)
    VALUES (USER_ID(), (SELECT adminName FROM admin WHERE loginID = USER_ID()), 'ticket',
@action, GETDATE());

    -- Insert into activitylog for specific details
    INSERT INTO activitylog (seatnumber, seatrow, hallid, slotid, movieid, bookingid,
paymentid, clientid, manipulationtime, action, ticketid)
SELECT
        COALESCE(i.seatnumber, d.seatnumber),
        COALESCE(i.seatrow, d.seatrow),
        COALESCE(i.hallid, d.hallid),
        COALESCE(i.slotid, d.slotid),
        COALESCE(i.movieid, d.movieid),
        COALESCE(i.bookingid, d.bookingid),
        COALESCE(i.paymentid, d.paymentid),
        COALESCE(i.clientid, d.clientid),
        GETDATE(),
        @action,
        COALESCE(i.ticketid, d.ticketid)
    FROM (SELECT * FROM inserted) i
    FULL JOIN (SELECT * FROM deleted) d ON i.ticketid = d.ticketid; END;
```

```
CREATE TRIGGER admin_trigger ON
admin
AFTER INSERT, UPDATE, DELETE
AS
```

```
s
BEGIN
DECLARE @action VARCHAR(10);

    IF EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT * FROM deleted)
        SET @action = 'UPDATE';
    ELSE IF EXISTS(SELECT * FROM inserted)
        SET @action = 'INSERT';
    ELSE IF EXISTS(SELECT * FROM deleted)
        SET @action = 'DELETE';

    INSERT INTO auditlog (adminid, adminName, tablename, action, manipulationtime)
    VALUES (
        ISNULL((SELECT adminID FROM admin WHERE loginID = USER_ID()), 0),
        ISNULL((SELECT adminName FROM admin WHERE loginID = USER_ID()), 'Unknown'),
        'admin',
        @action,
        GETDATE()
    );
END;
```

Views:

CREATE VIEW vw_TicketDetails AS

SELECT t.ticketid, c.clientName, m.movieName, s.slotName, h.hallName, t.seatRow, t.seatnumber

FROM ticket t

INNER JOIN client c ON t.clientid = c.clientID

INNER JOIN movie m ON t.movieid = m.movieID

INNER JOIN slot s ON t.slotid = s.slotID

INNER JOIN hall h ON t.hallid = h.hallID;

CREATE VIEW vw_HallSchedules AS

SELECT h.hallName, h.floor, s.slotName, s.date, s.startTime, s.endTime, m.movieName

FROM hall h

INNER JOIN cinemamovie cm ON h.hallID = cm.hallID

INNER JOIN slot s ON cm.slotID = s.slotID

```
INNER JOIN
movie m ON cm.movieID = m.movieID;
```

```
CREATE VIEW vw_ClientBookings AS
SELECT c.clientName, b.date, b.day, b.time, p.paymentAmount, p.paymentType
FROM client c
INNER JOIN payment p ON c.clientID = p.clientID
INNER JOIN bookingInfo b ON p.paymentID = b.paymentid;
```

```
CREATE VIEW vw_MovieSlots AS
SELECT m.movieName, m.genre, m.movieType, s.slotName, s.startTime, s.endTime, s.day,
s.date, s.status
FROM movie m
INNER JOIN slot s ON m.movieID = s.movieID;
```

```
CREATE VIEW vw_UpcomingMovies AS
SELECT movieID, movieName, releaseDate, duration, Show, movieType
FROM movie
WHERE releaseDate > GETDATE();
```

```
CREATE VIEW vw_CurrentMovies AS
SELECT movieID, movieName, genre, status, releaseDate, duration, Show, movieType
FROM movie
WHERE status = 'Now Showing';
```

```
CREATE VIEW vw_AdminAccountDetails AS
```

```
SELECT a.adminID, a.adminName, a.contact,
a.gender, l.username, l.password

FROM admin a

INNER JOIN loginInformation l ON a.loginID = l.loginID;
```

```
CREATE VIEW vw_UserAccountDetails AS

SELECT c.clientID, c.clientName, c.contact, c.email, c.gender, l.username, l.password

FROM client c

INNER JOIN loginInformation l ON c.loginID = l.loginID;
```

Conclusion:

- **Evaluation of the project's success in meeting its objectives**

Our cinema ticketing system project has proven to be a resounding success, meeting and in many instances, surpassing its initial objectives. Here's how the project proves to be a success:

Ease of Navigation:

From the get-go, users have been greeted with a clean and straightforward interface. Clear callto-action buttons and a logical flow from movie selection to final payment have ensured that even the least tech-savvy users find the process hassle-free.

Quick Movie Selection:

With an intelligent search and filter system, users can quickly find movies based on title, genre, release date, or even actor names. This has cut down the time it takes to find a desired movie, increasing the speed of the overall booking process.

Real-Time Seat Selection:

Our real-time seat selection feature, displaying an accurate map of the cinema hall, allows customers to choose their preferred seats with a few clicks. This visual representation has been a game-changer, eliminating the uncertainty and frustration of blindly selecting seats.

Responsive Design:

Just like the user interface, the admin side is fully responsive, ensuring that admins can manage the cinema operations on the go, without losing functionality or experiencing a drop in performance.

Streamlined Movie Management:

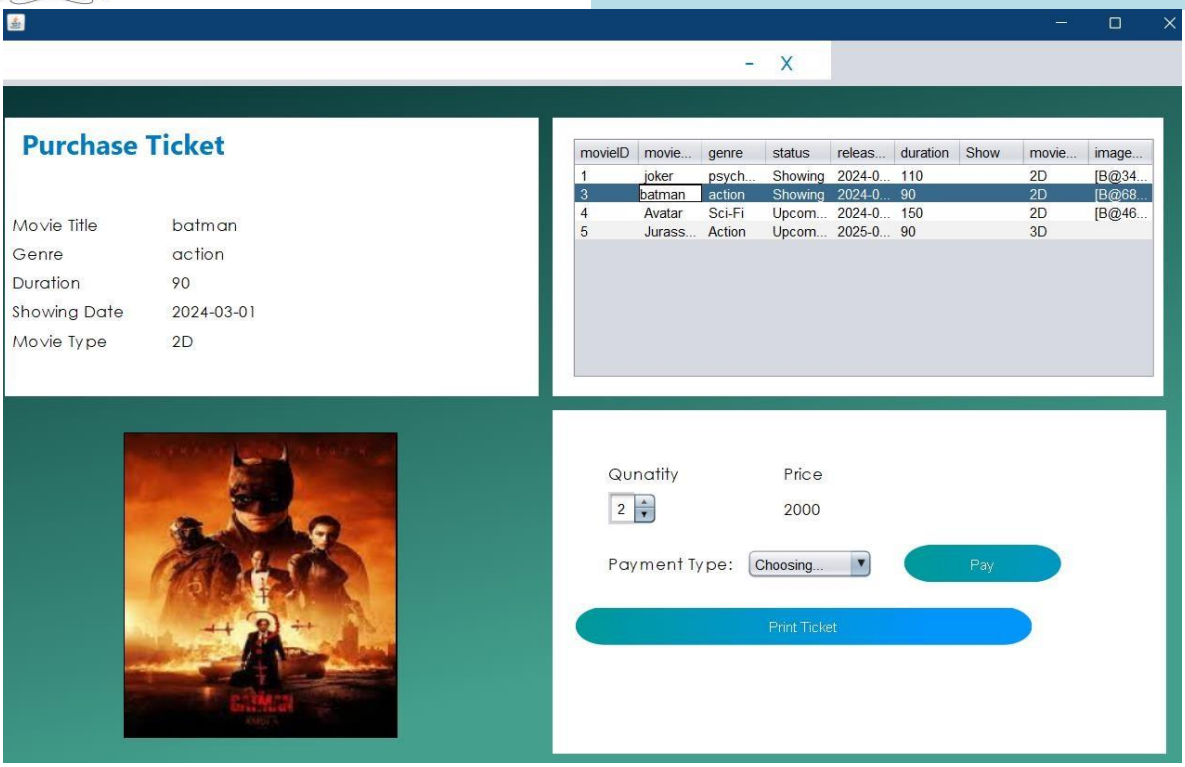
Admins can add new movie listings, update details, and remove showings with a few clicks. The intuitive interface ensures that changes are reflected in real time, allowing for dynamic management of movie schedules.

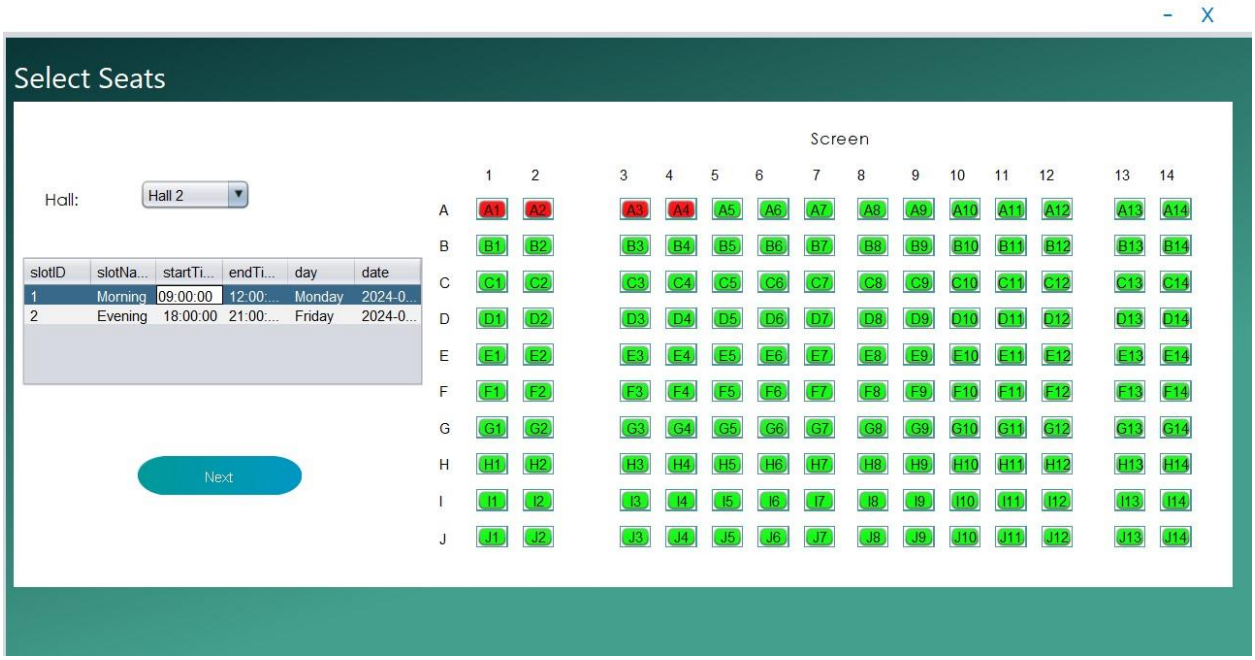
Scalability:

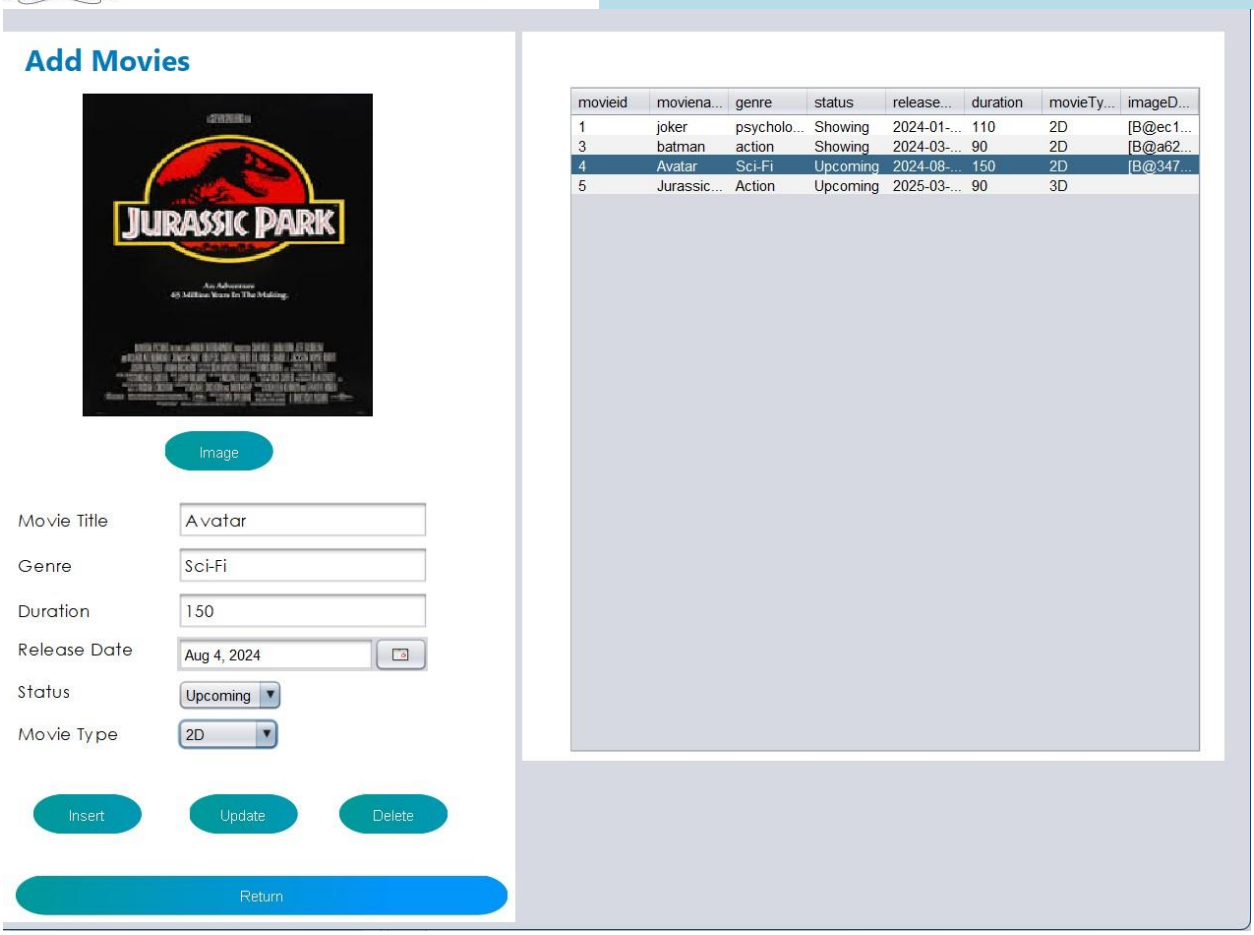
The system is designed to scale with the business, accommodating increases in customer numbers and additional cinemas without a loss of performance.

The cinema ticketing system project encapsulates a shift toward a more technologically savvy, customer-centric approach to movie watching, bringing the cinema into the modern age and setting a new standard for the entertainment industry.

- **Screenshots of major modules' outputs**


Purchase Ticket
Movie Title batman
Genre action
Duration 90
Showing Date 2024-03-01
Movie Type 2D
movieID movie... genre status releas... duration Show movie... image...
1 joker psych... Showing 2024-0... 110 2D [B@34...
3 batman action Showing 2024-0... 90 2D [B@68...
4 Avatar Sci-Fi Upcom... 2024-0... 150 2D [B@46...
5 Jurass... Action Upcom... 2025-0... 90 3D
Qunatity
Price
2
2000
Payment Type:
Choosing...
Pay
Print Ticket


Select Seats
Screen
Hall:
Hall 2
slotID slotNa... startTi... endTi... day date
1 Morning 09:00:00 12:00:... Monday 2024-0...
2 Evening 18:00:00 21:00:... Friday 2024-0...
Next


Add Movies
JURASSIC PARK
Image
Movie Title
Avatar
Genre
Sci-Fi
Duration
150
Release Date
Aug 4, 2024
Status
Upcoming
Movie Type
2D
Insert
Update
Delete
Return
movieid moviena... genre status release... duration movieTy... imageD...
1 joker psycholo... Showing 2024-01-... 110 2D [B@ec1...
3 batman action Showing 2024-03-... 90 2D [B@a62...
4 Avatar Sci-Fi Upcoming 2024-08-... 150 2D [B@347...
5 Jurassic... Action Upcoming 2025-03-... 90 3D