

**PROBLEM STATEMENT**

**Balamurugan. S** – **2022103544**

**Shakith. A** – **2022103545**

**Team Number** : 7

**Semester** : 4

**Batch** : P

**Department**  : CSE

**Submitted On** : 23/05/2024

**Submitted To** : Dr.S.Renugadevi, DCSE, Anna University

**TicketsTrail**

**Online Movie Ticket Booking System**

**CS6106**

**Database Management Systems Report**

Buying movie tickets the old way can be a hassle: long lines, uncertain seat availability, and limited information. Our project aims to revolutionize the online movie ticket reservation experience by introducing innovative features and enhancing user experience.

We propose to offer advanced search options, seamless integration with nearby theaters, and a simplified yet powerful user interface. This will not only make booking movie tickets more convenient but also provide users with a richer experience. Additionally, our platform will support theater administrators by providing real-time statistics, such as revenue insights, peak hours and days, user preferences, and popular genres.

Our goal is to surpass existing platforms by delivering an enhanced user experience, making the process of booking movie tickets more efficient and enjoyable, while simultaneously equipping theater administrators with essential tools for managing theaters and movies, informed with decision-making and improved profitability.

**TECHNOLOGIES USED**

* HTML, CSS, Javascript, Bootstrap
* mySql - Xampp
* Flask and Python

**KEY FEATURES**

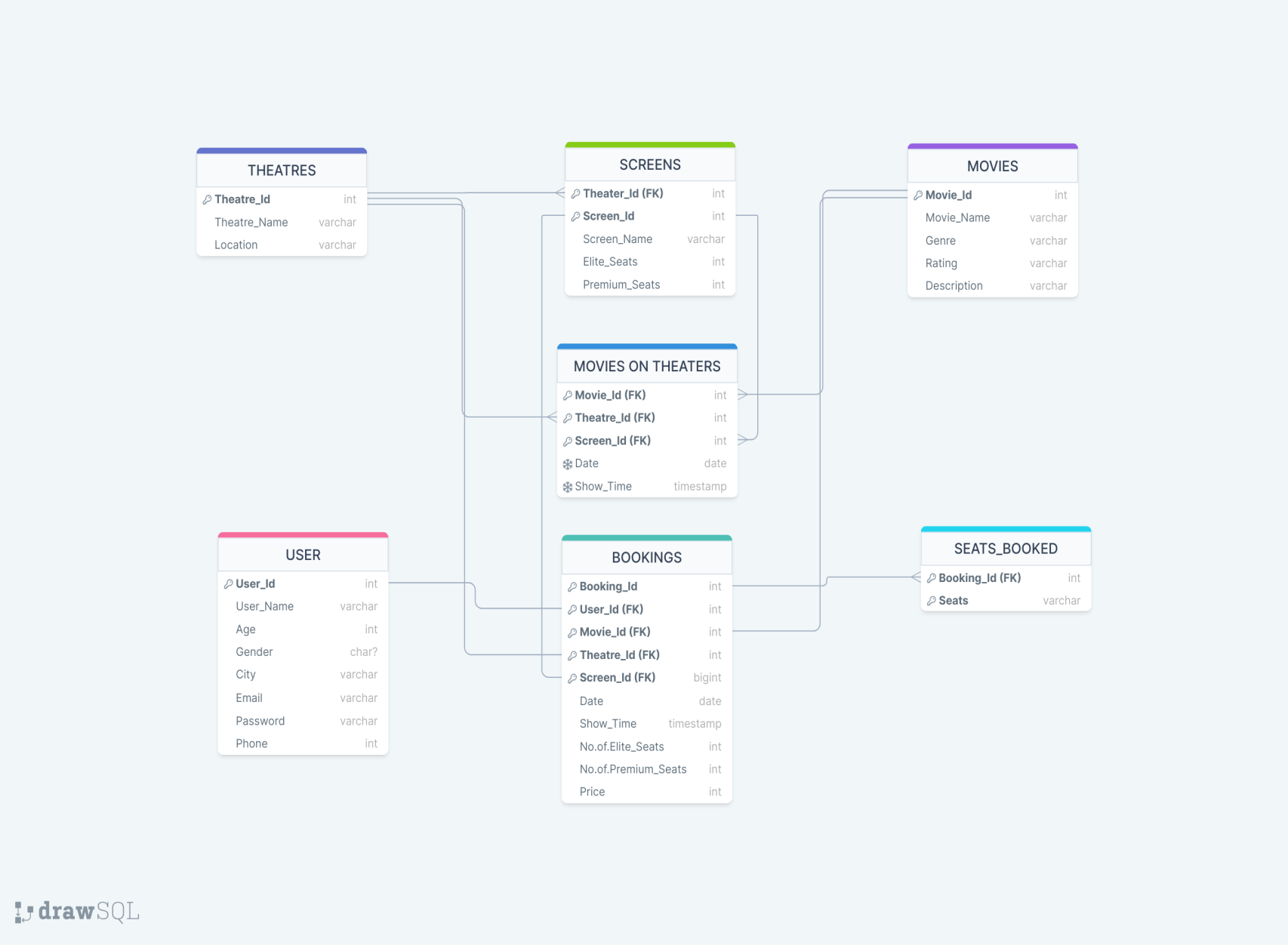
* Optimized Search for the movies.
* Theater suggestion based on user’s location.
* Enhanced and Simple UI/UX.
* Theater’s revenue**.**
* Movie’s revenue.
* Administrating Theaters, Movies and Screenings of Movies.

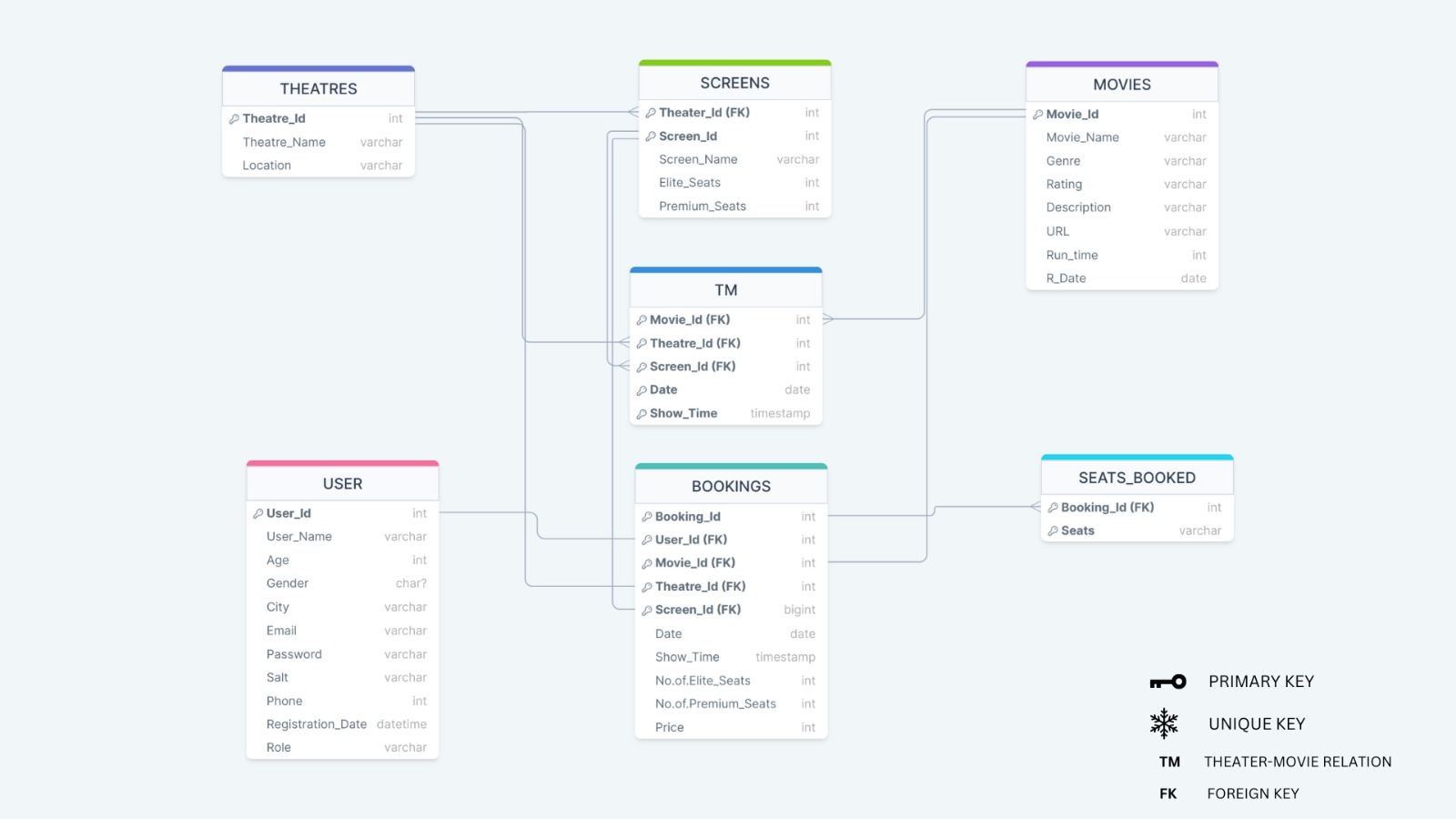
**ENTITIES**

* **Users -** User\_Id, User\_Name, Age, Gender, City, Email, Password, Phone, Registration date, salt, role
* **Theaters -** Theater\_Id, Theater\_Name, Location
* **Screens -** Theater\_Id, Screen\_id, Screen\_Name, Elite\_Seats, Premium\_Seats
* **Movies -** Movie\_Id, Movie\_Name, Genre, Rating, Description, URL, Run time, Run date
* **Bookings** - Booking\_Id, User\_Id, Movie\_Id, Theater\_Id, Screen\_id, Date, Show\_Time, No\_of\_Elite\_Seats, No\_of\_Premium\_Seats, Price
* **Seats\_Booked** - Booking\_Id, Seats

**RELATIONS**

* **Movies\_On\_Theaters** - Movie\_Id, Theater\_Id, Screen\_Id, Date, Show\_Time
* Theater is related to Screens
* Users, Theaters, Movies and Screens are related via Bookings.
* Bookings and Seats\_Booked are related to each other by the booking\_id.

**SCHEMA\_DIAGRAM**

****

URL varchar

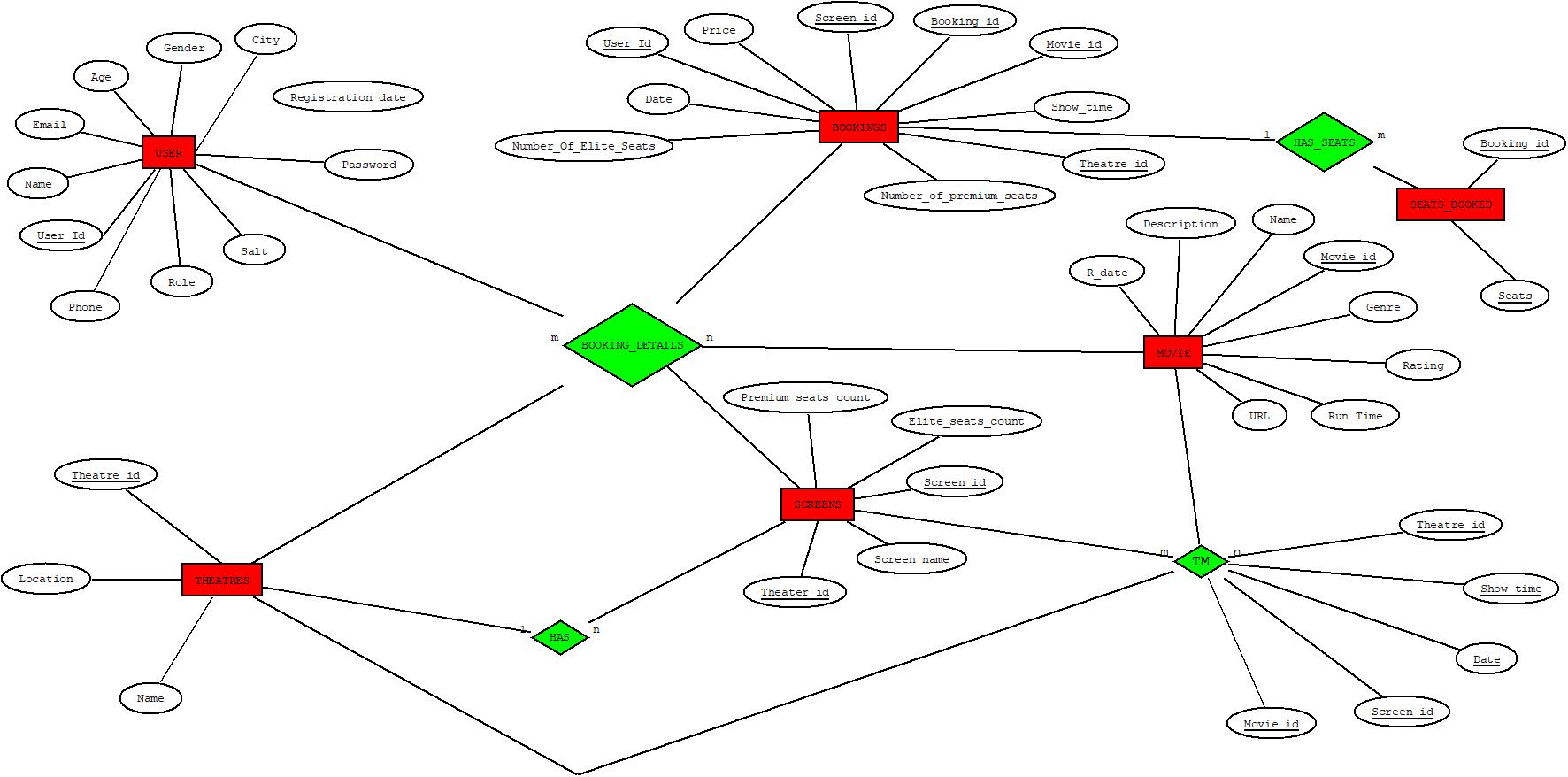
R\_time time

R\_date date

Reg\_date date

Role varchar

Salt varchar

** ER DIAGRAM**

**IMPLEMENTATION**

Front End

**USER SIDE**

**a. Login**

This consists of the login page, which serves as a gateway to enter the ticketing website. Users can either log in or sign up if they don’t have an account.

Here the password is checked with the hashed value in the table. If the password matches, user goes to the home page.

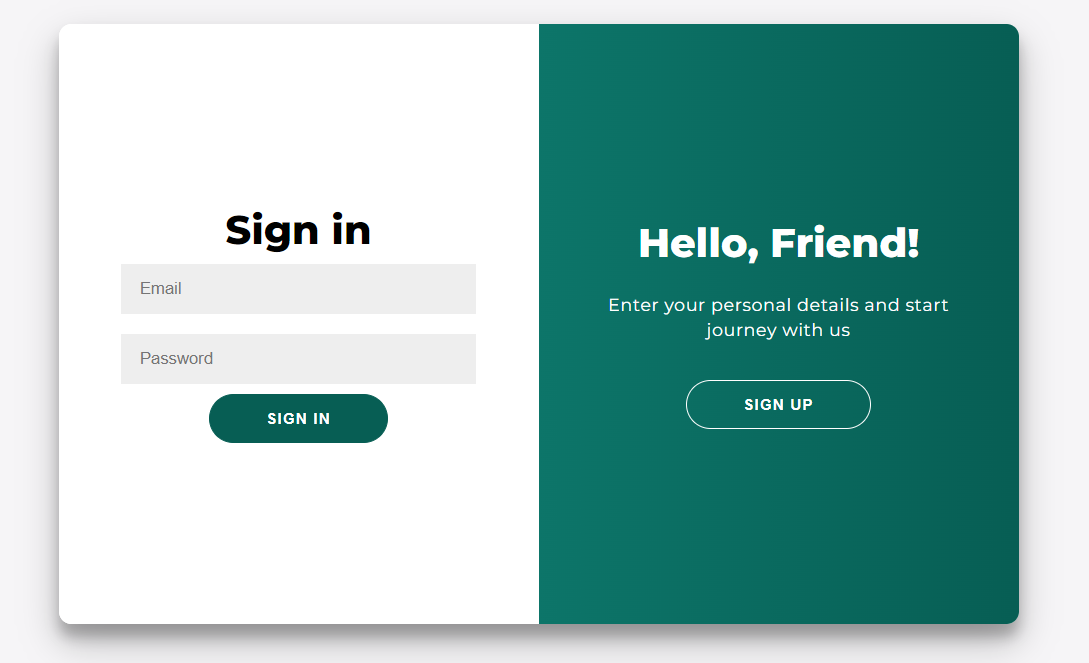


Figure 1: The website with login page

**b. Sign-Up**

New user details are stored in database for future reference and logins. Data is stored in the “users” table. This is used to store information about the new users.

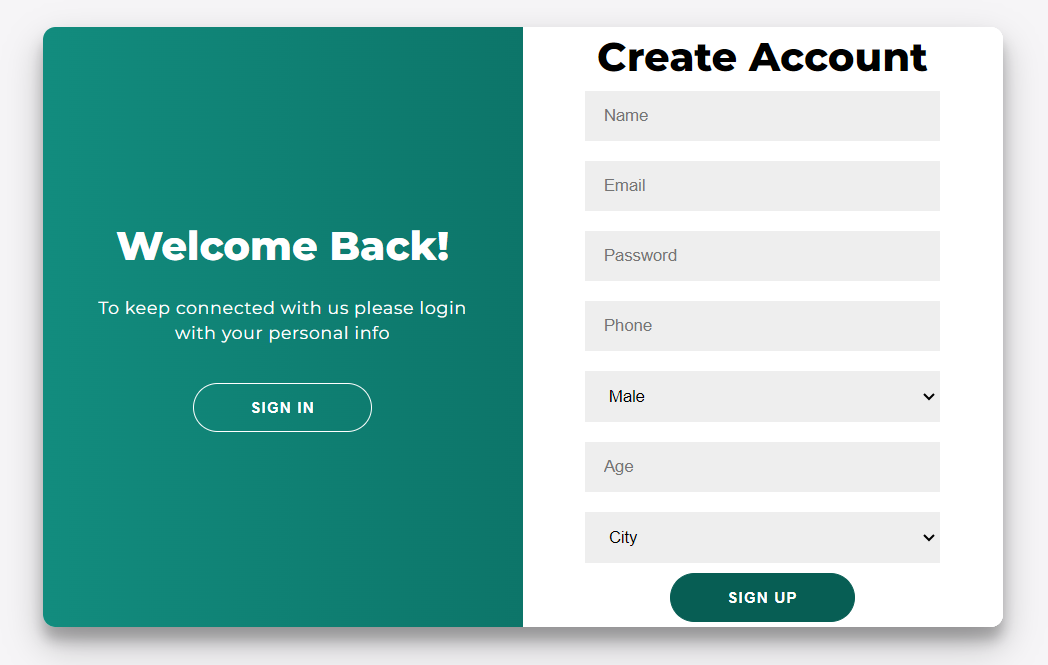
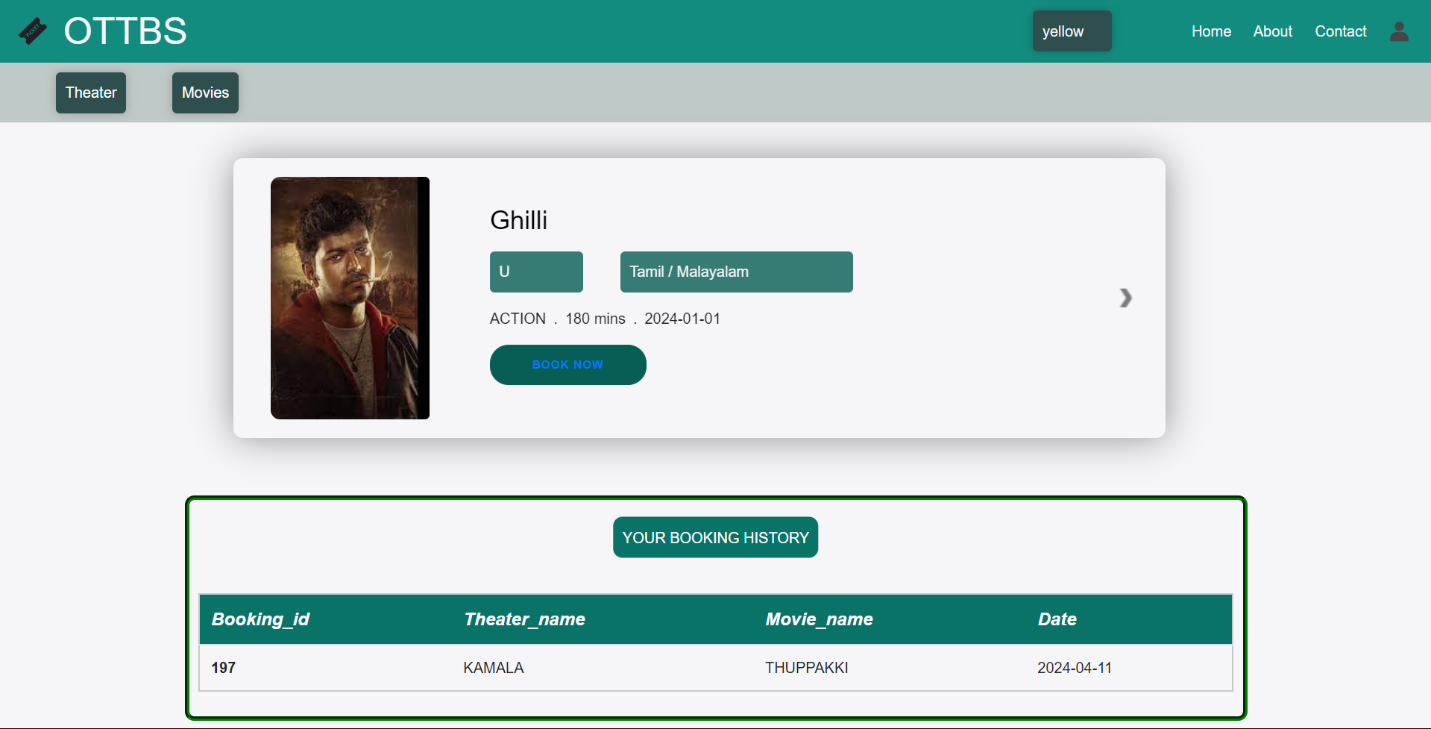
`

Figure 2: The website with signup page

**c. Homepage**

After logging in, shows the list of movies available to book from.

****

**d. Booking Options**

**1. Theater based selection.**

Allows user to select theatre to book.

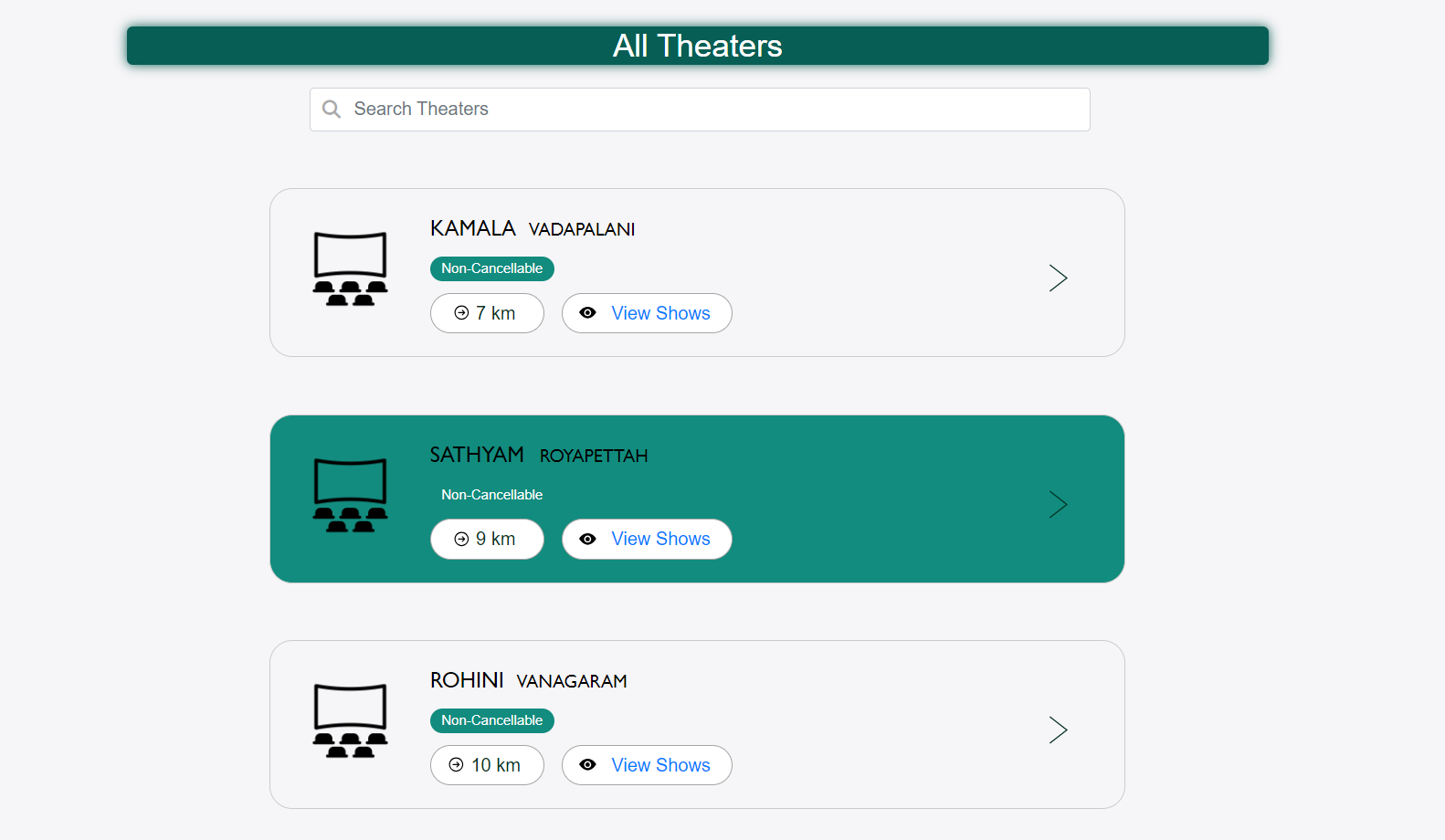
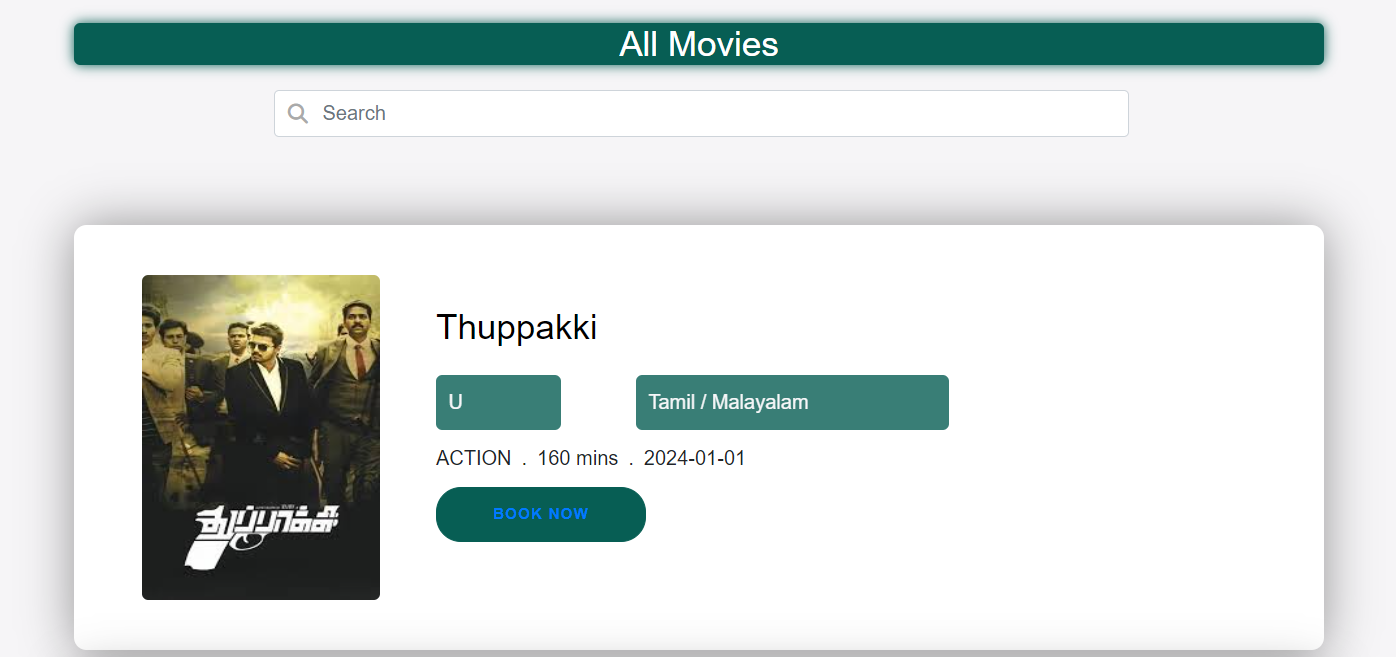
If the user clicks on the theaters on the home page, following theater page will be displayed according to the user’s nearby location.

Figure 4: Theater page

If the user clicks on the view shows on the movie page, following movie page will be displayed.

Figure 5: Movie page

Onclicking the Book Now button in the Movie page, the following page will page

will be displayed to select the screen and the timings.

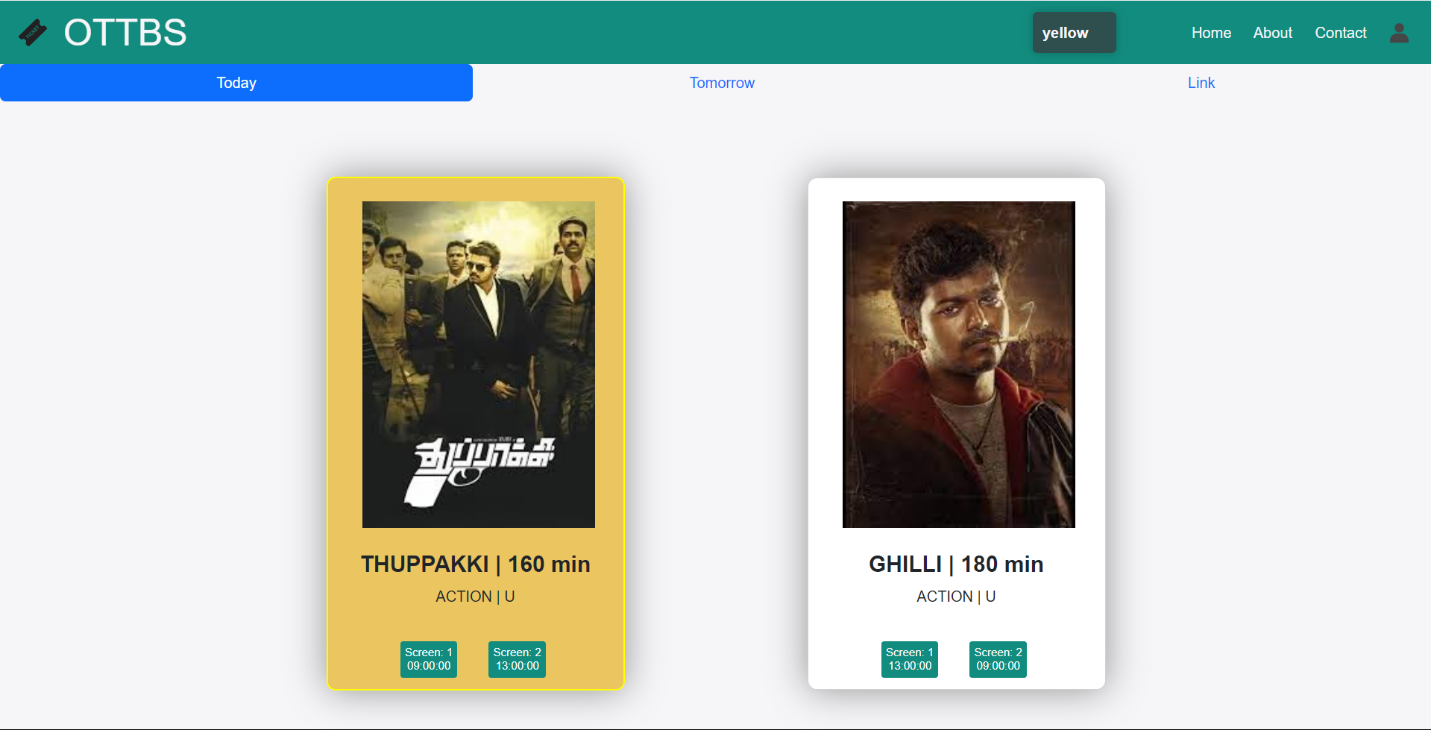


Figure 6: Screen Selection Page

**2. Movie based Selection:**

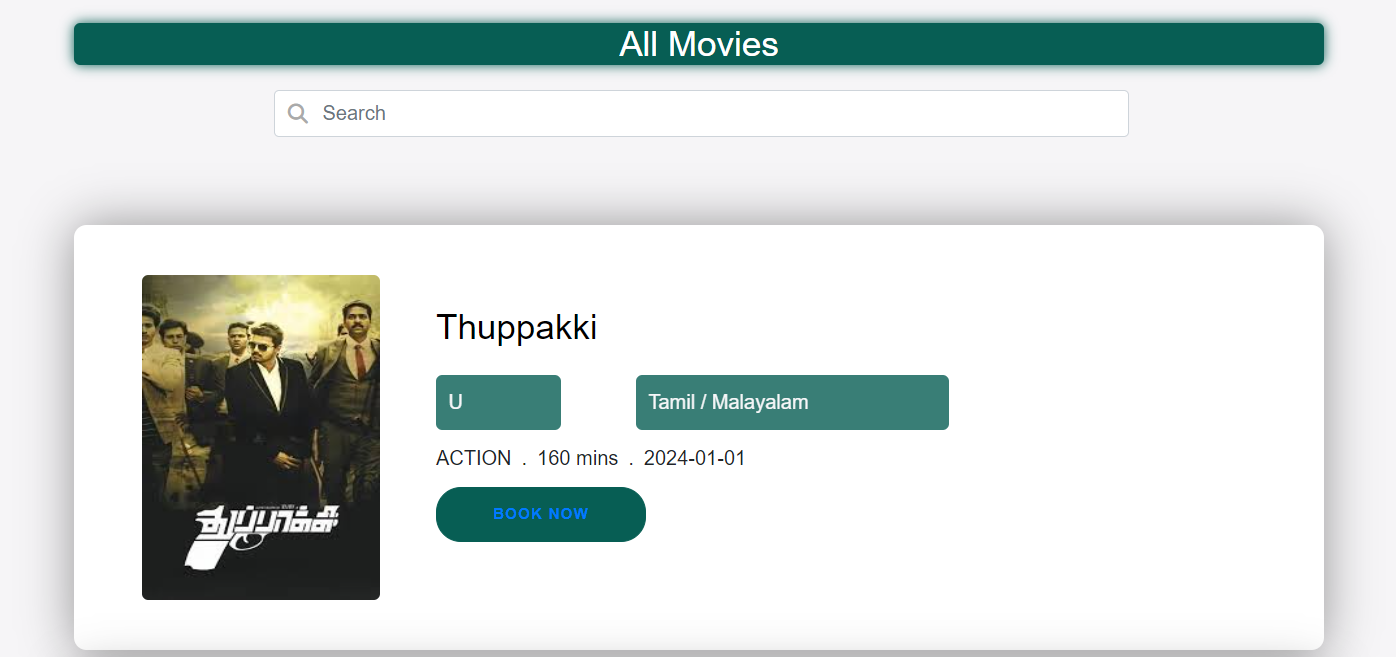
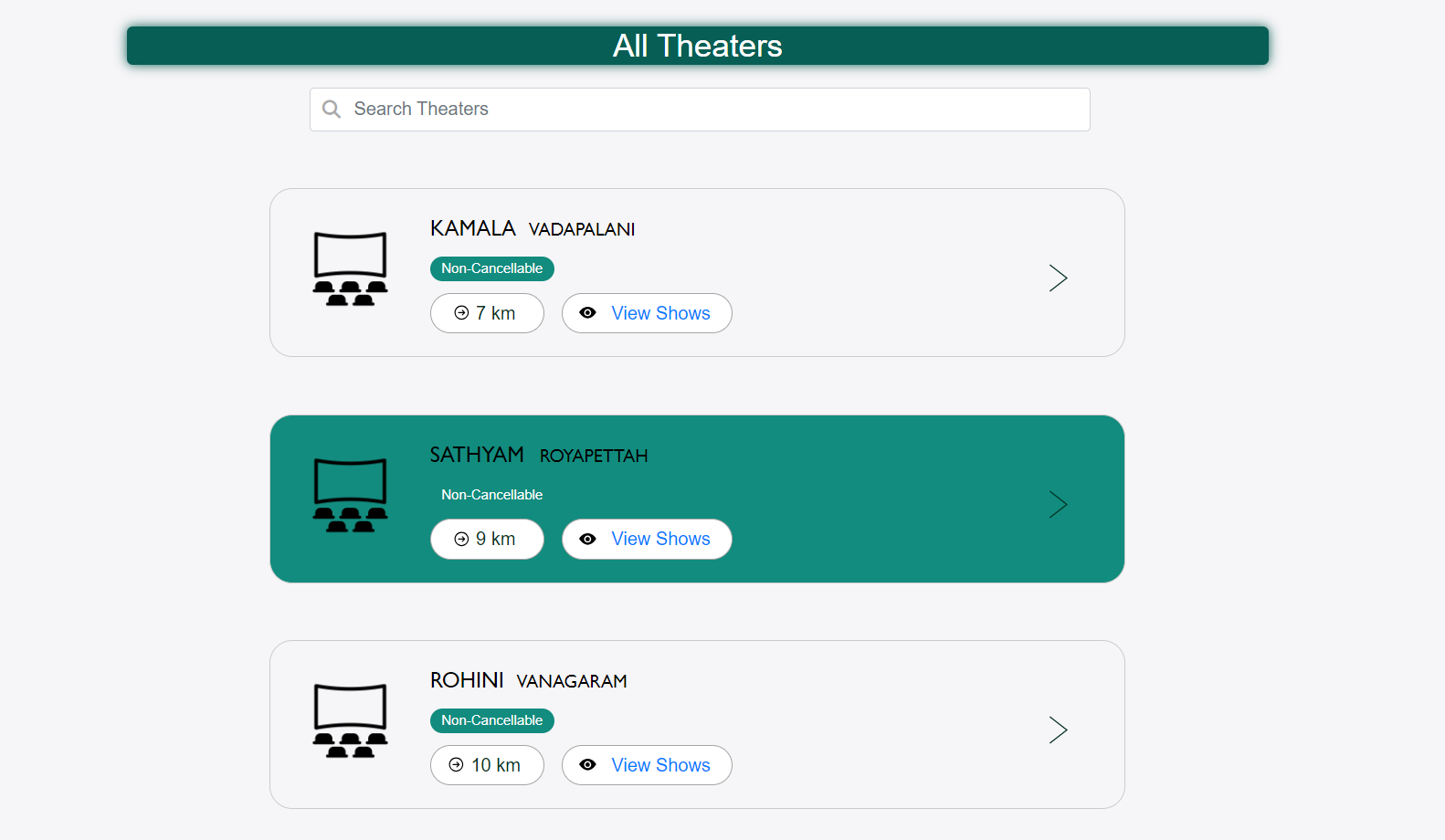
 Similarly by clicking on the Movie, the (figure5) Movie page will be displayed.

Figure 5: Movie page

After clicking on the book show button, the theaters page with

all theaters displaying the movies will be displayed.



Then, clicking on the view shows, the page with screen and time selection will be displayed.

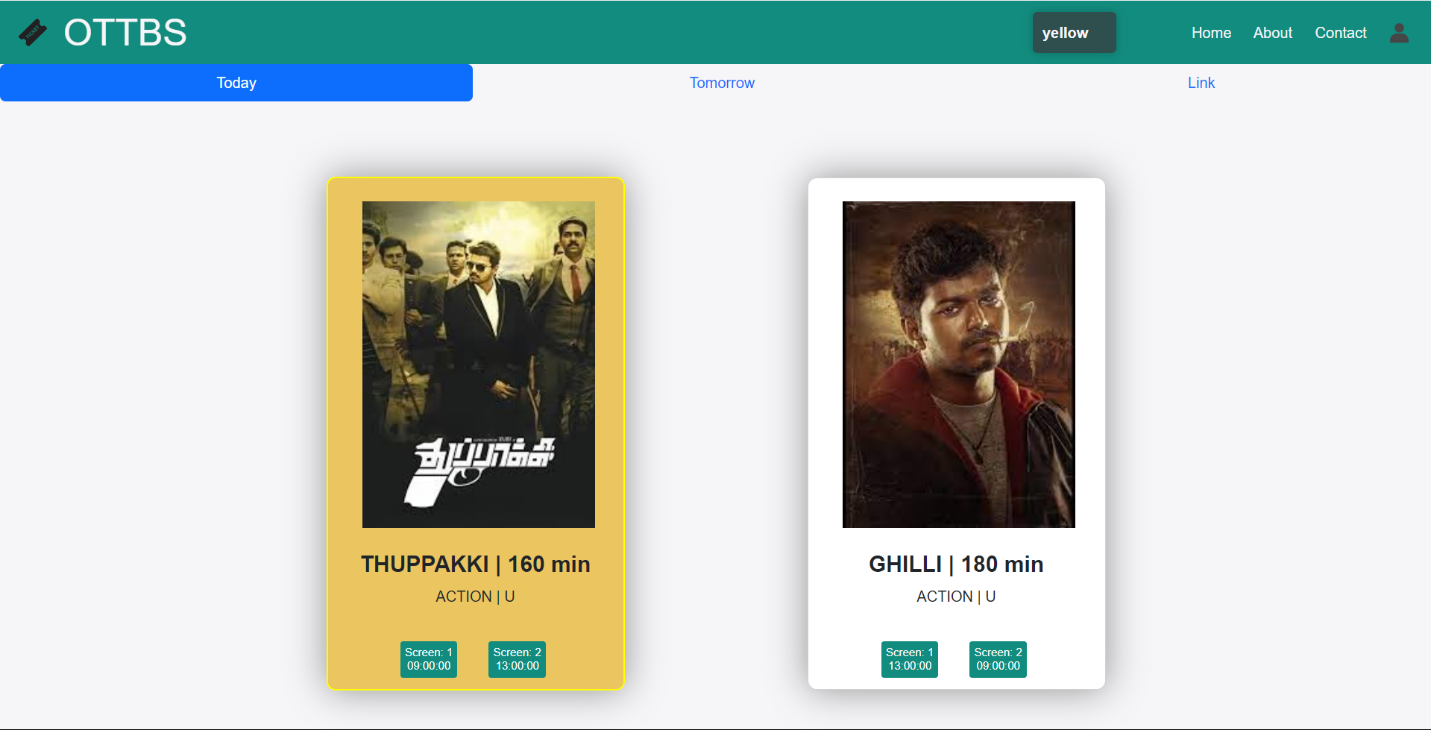
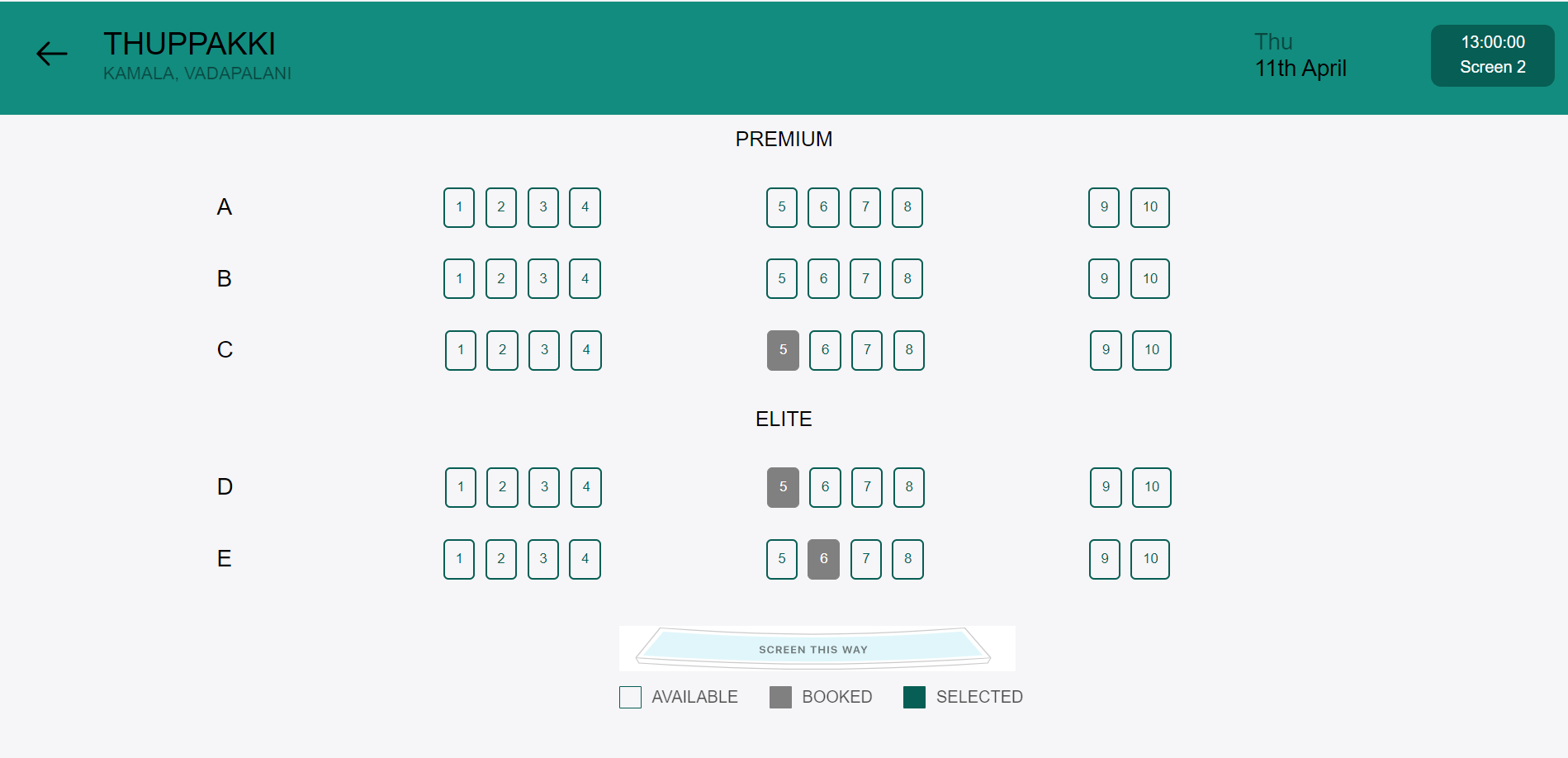


Figure 6: Screen Selection Page

**e. Seat selection :** Here after, the steps will be common to both theater and movie selection mode. By clicking on any one of the screen and time button, the following seat selection page will be displayed.



Already booked seats will be in grey color, If the user selects the seats,

a component with total seats and continue will toggled.

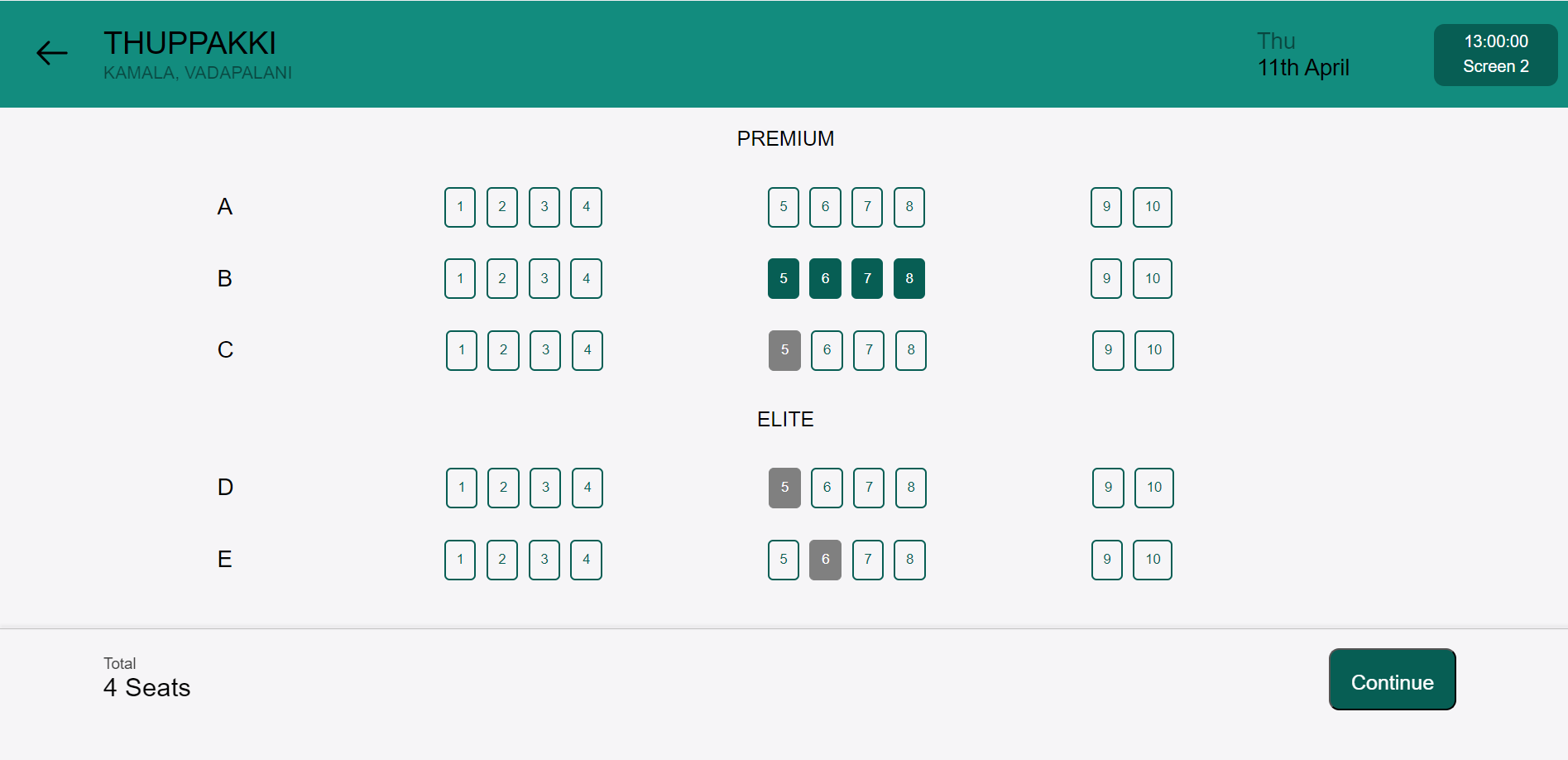
****

Figure 8: Seat Selection Page

**f. Payment :** By Selection the continue button, following page will be payment page with the ticket details on the left side will be displayed.

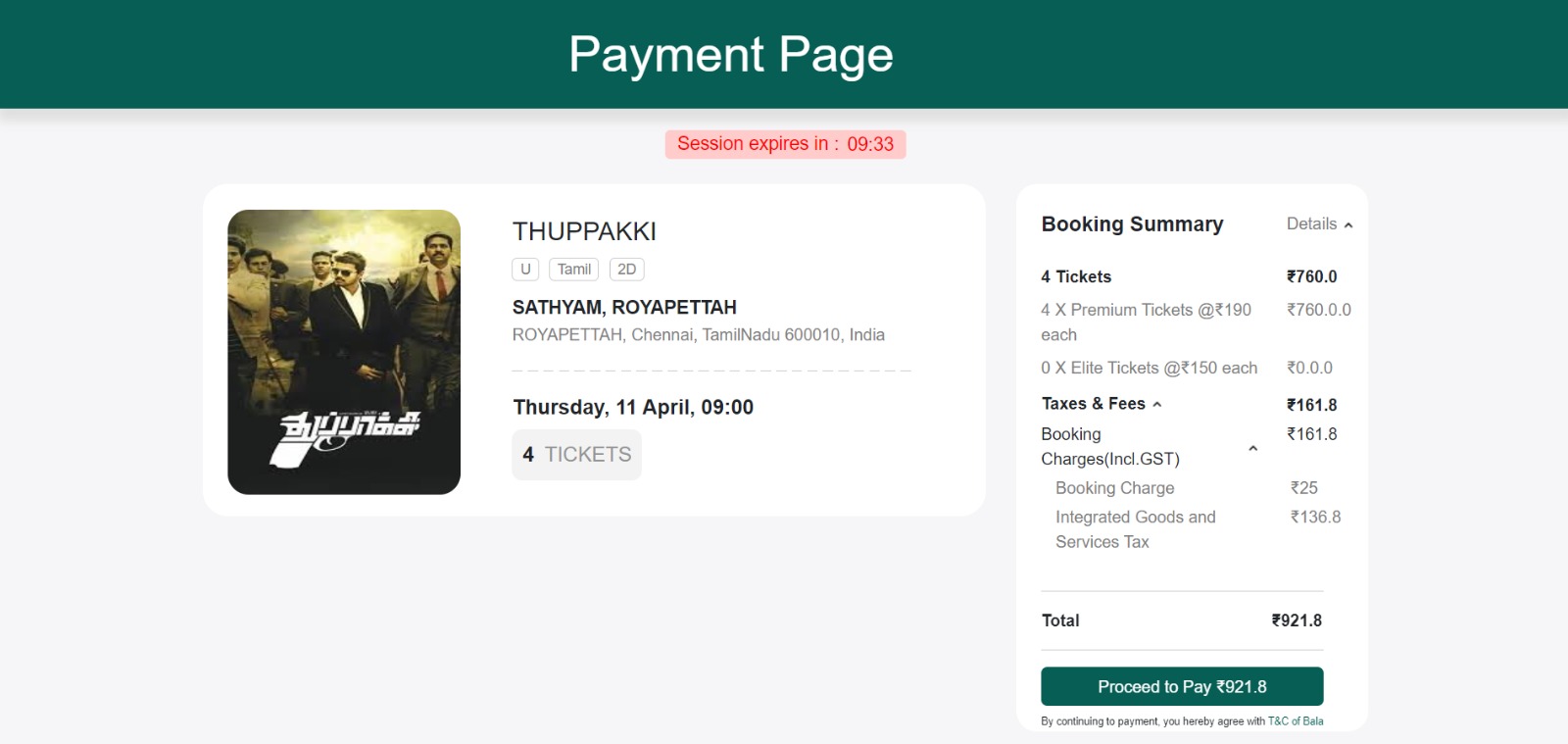


Figure 9: Payment Page

**g. Ticket :** By clicking on the proceed to pay, the user will pay the amount and the ticket will be generated along with QR code and that terminates the user side.

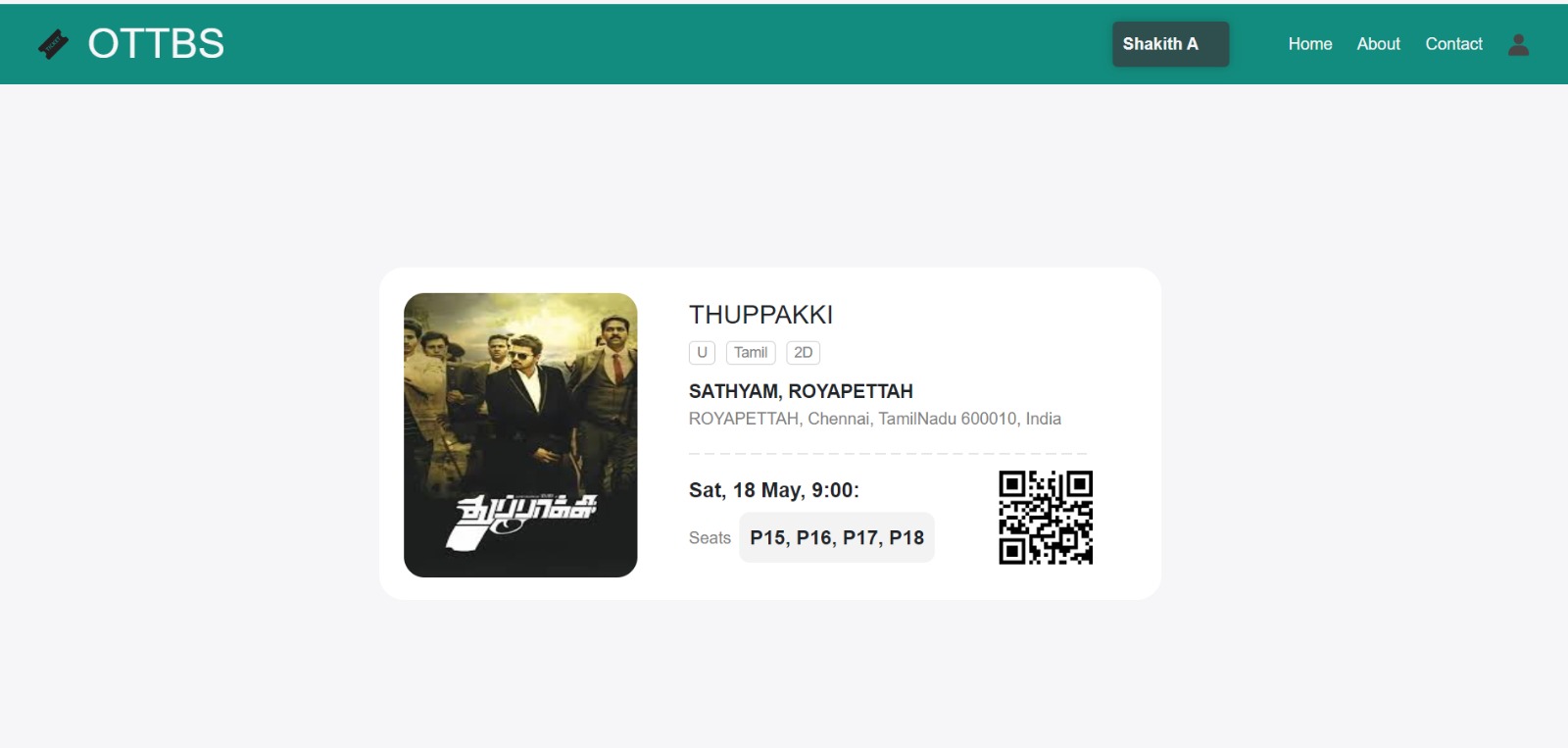
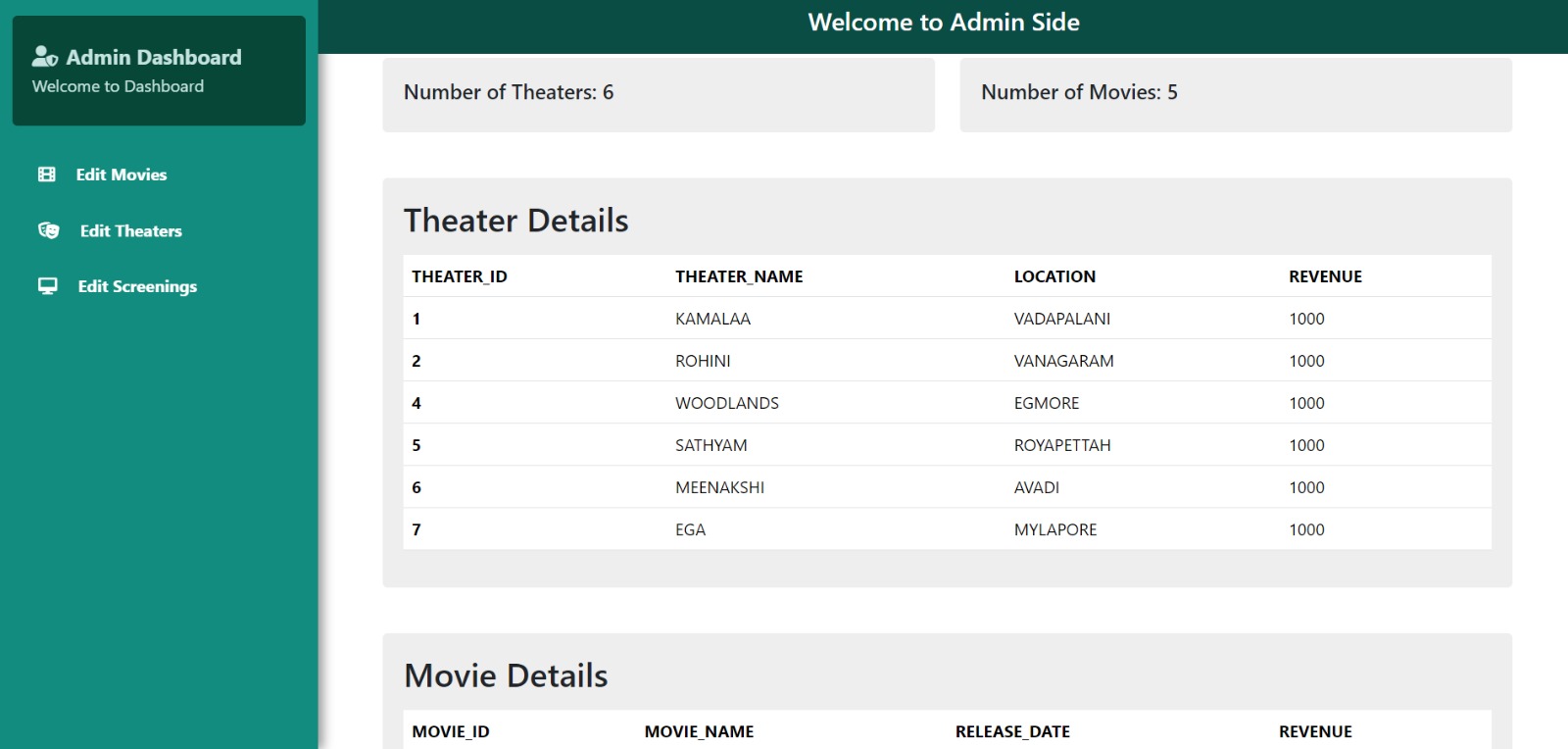


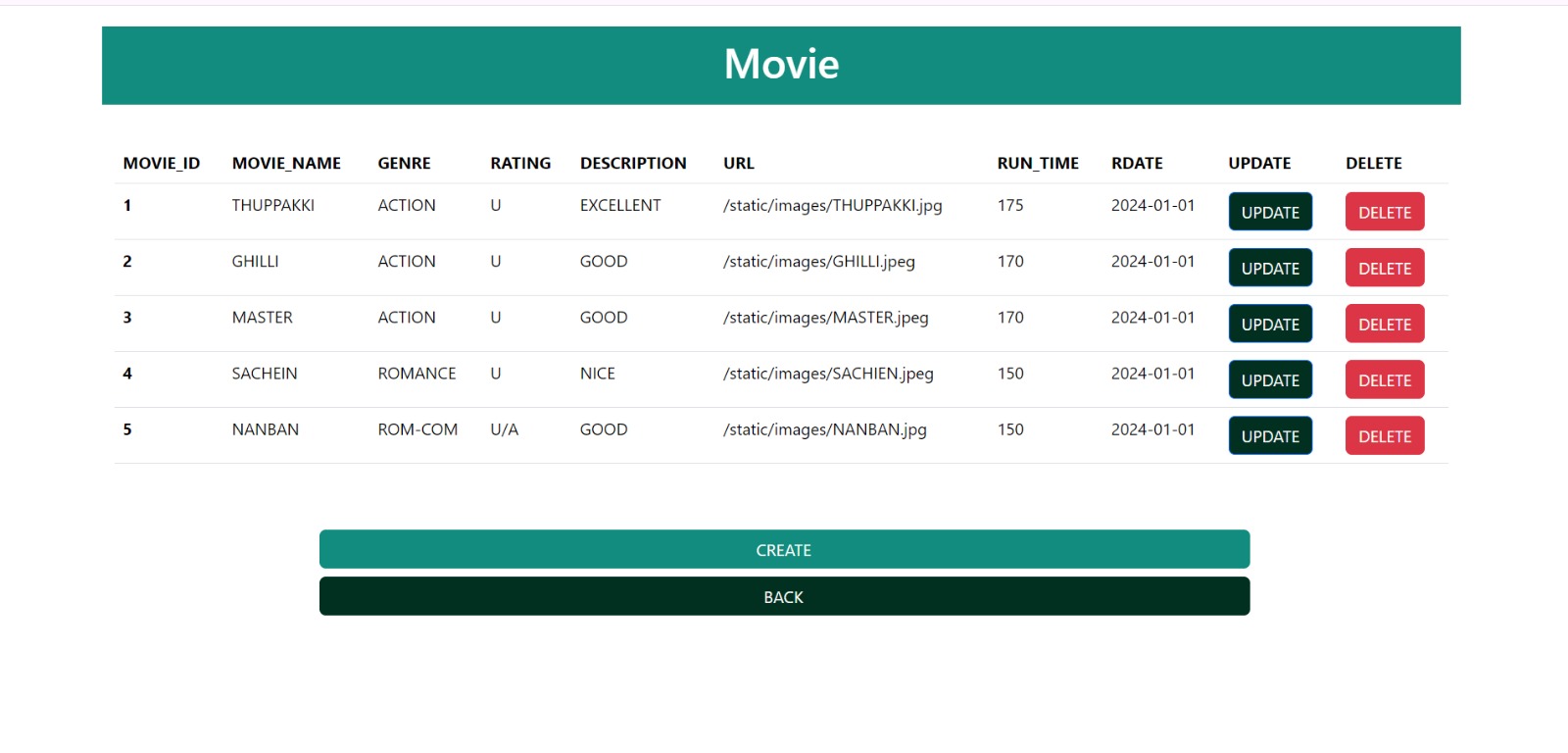
Figure 10: ticket generation Page

**ADMIN SIDE:**

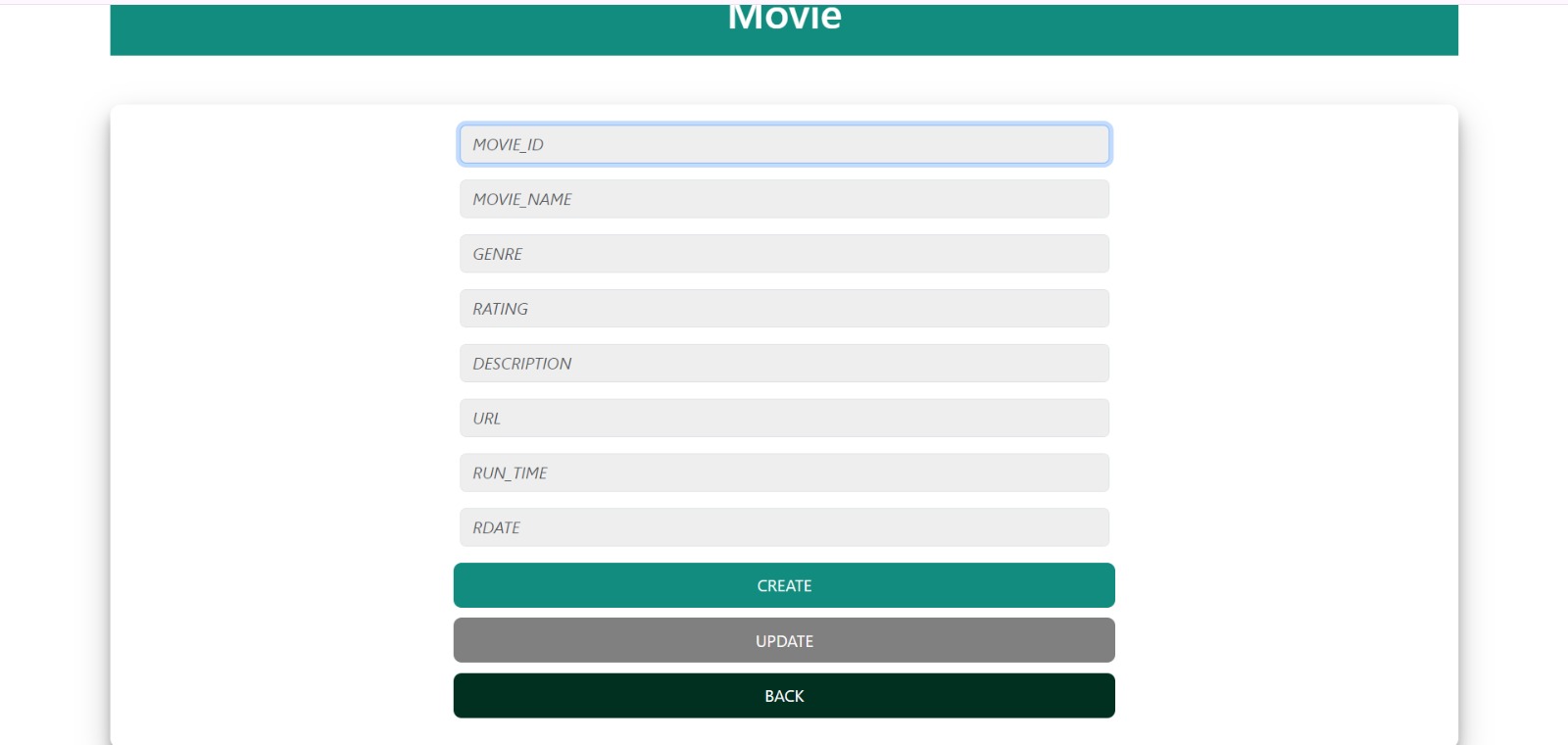
1. ****The admin landing page.

The dashboard page will show theater details and movie details.

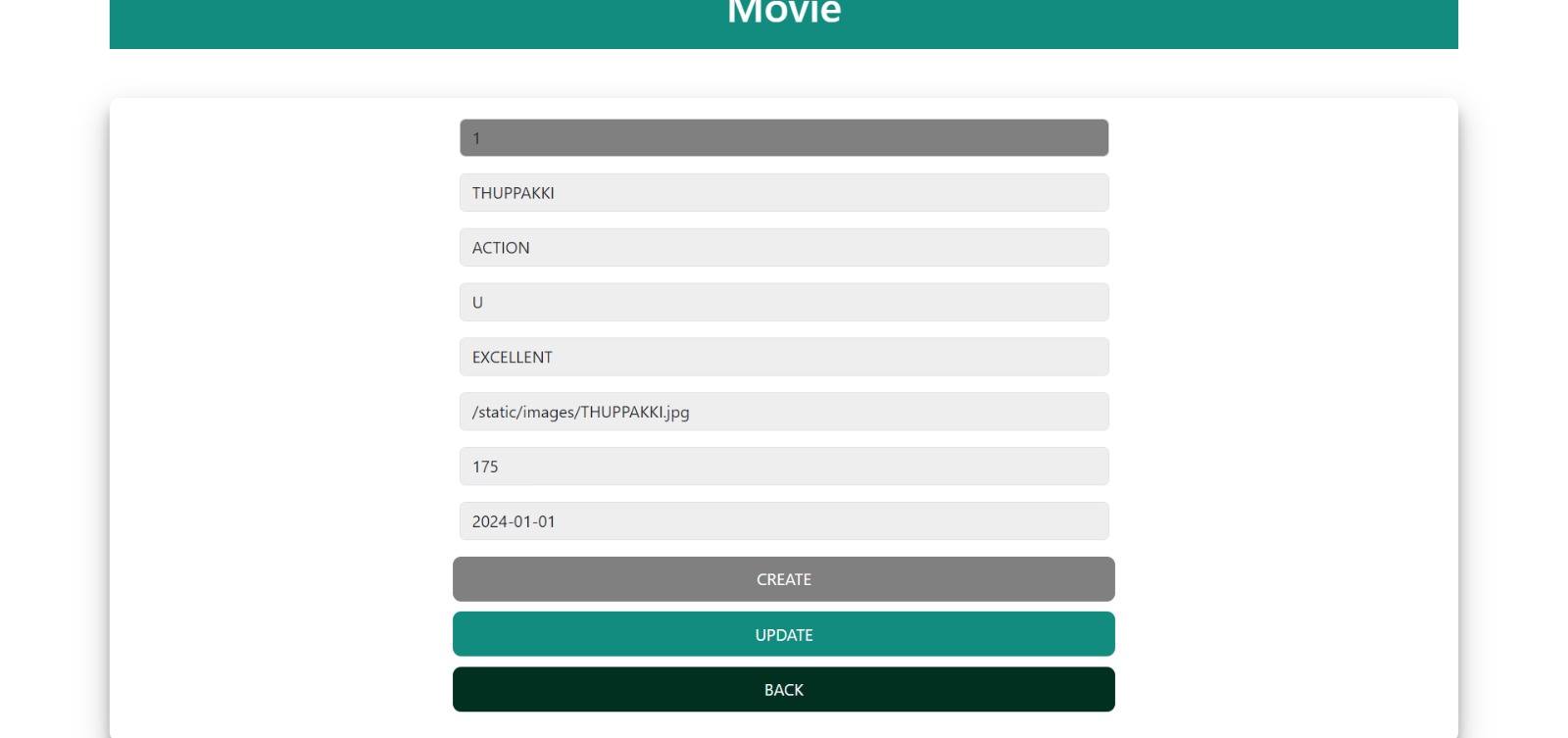
1)By clicking on the Edit movies, a page with create, update and delete movies will be displayed.



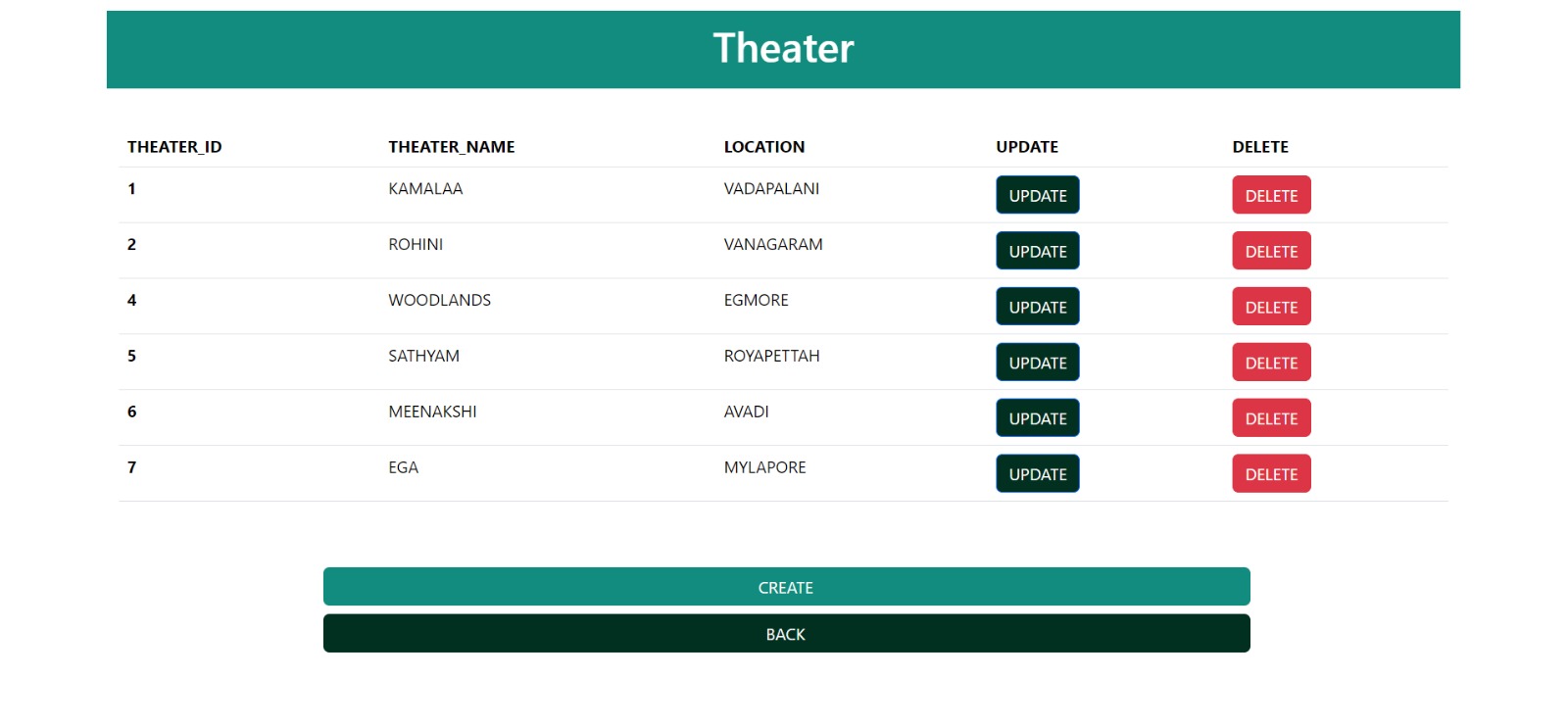
2) Inserting into the movie table from admin side.

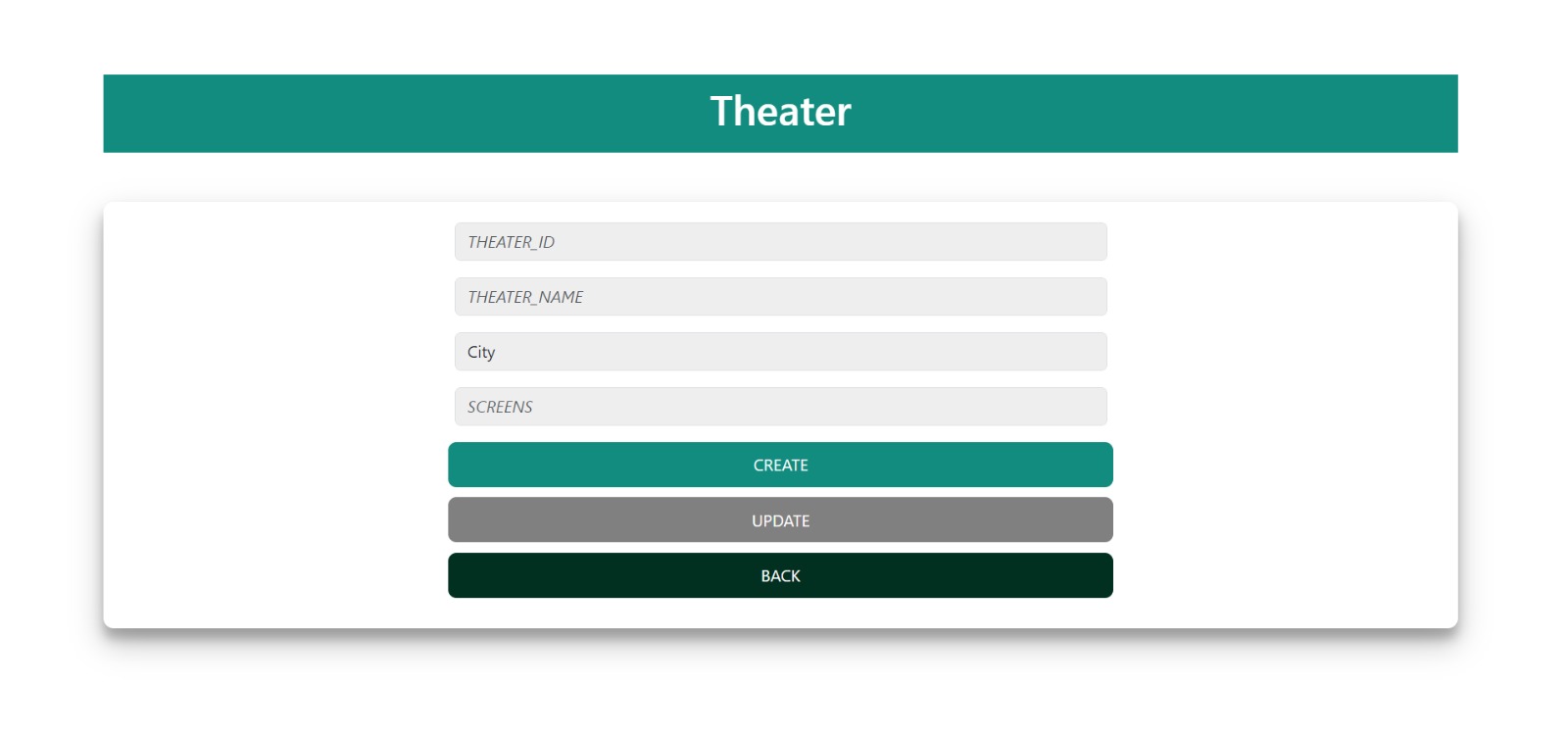


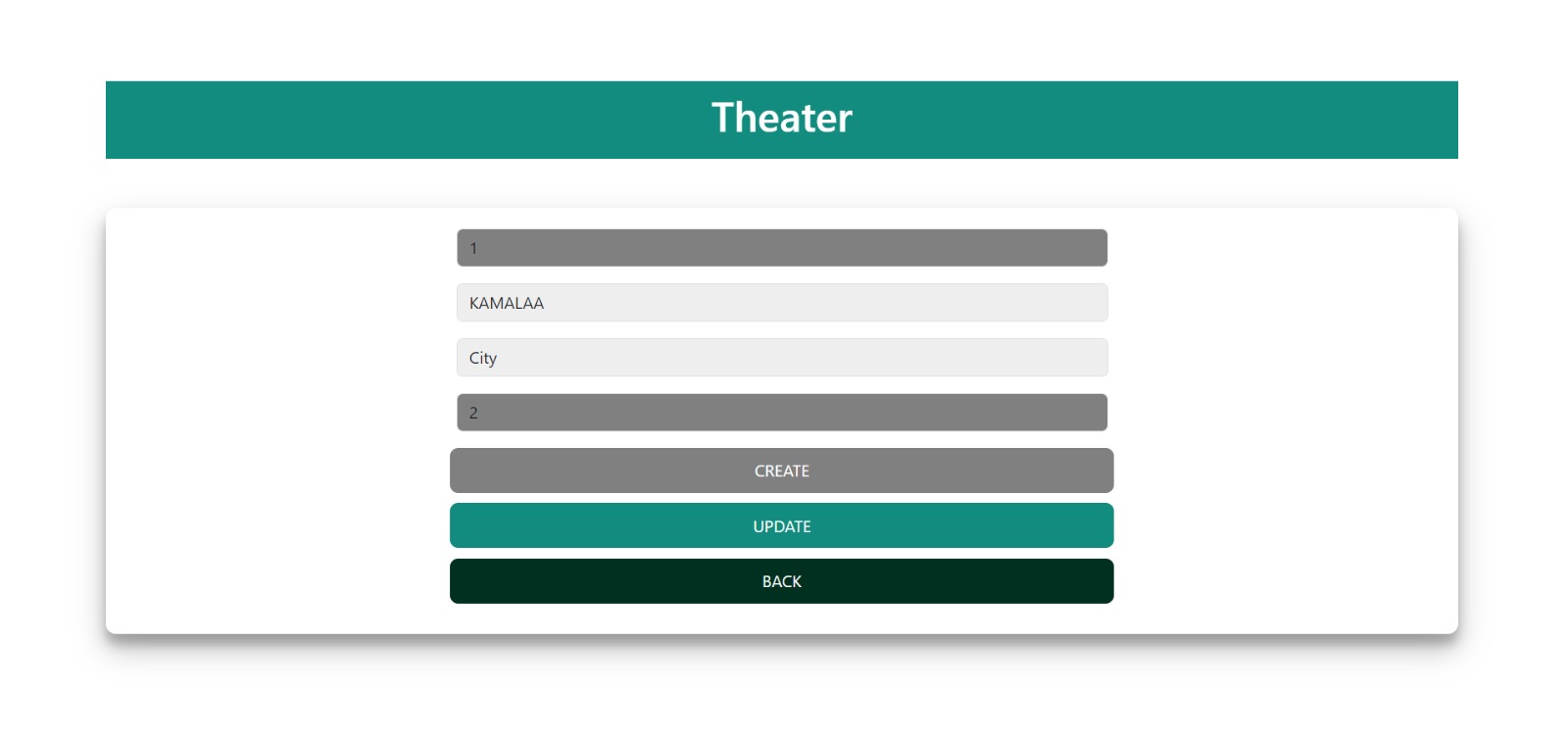
3) Updating the movie table

****

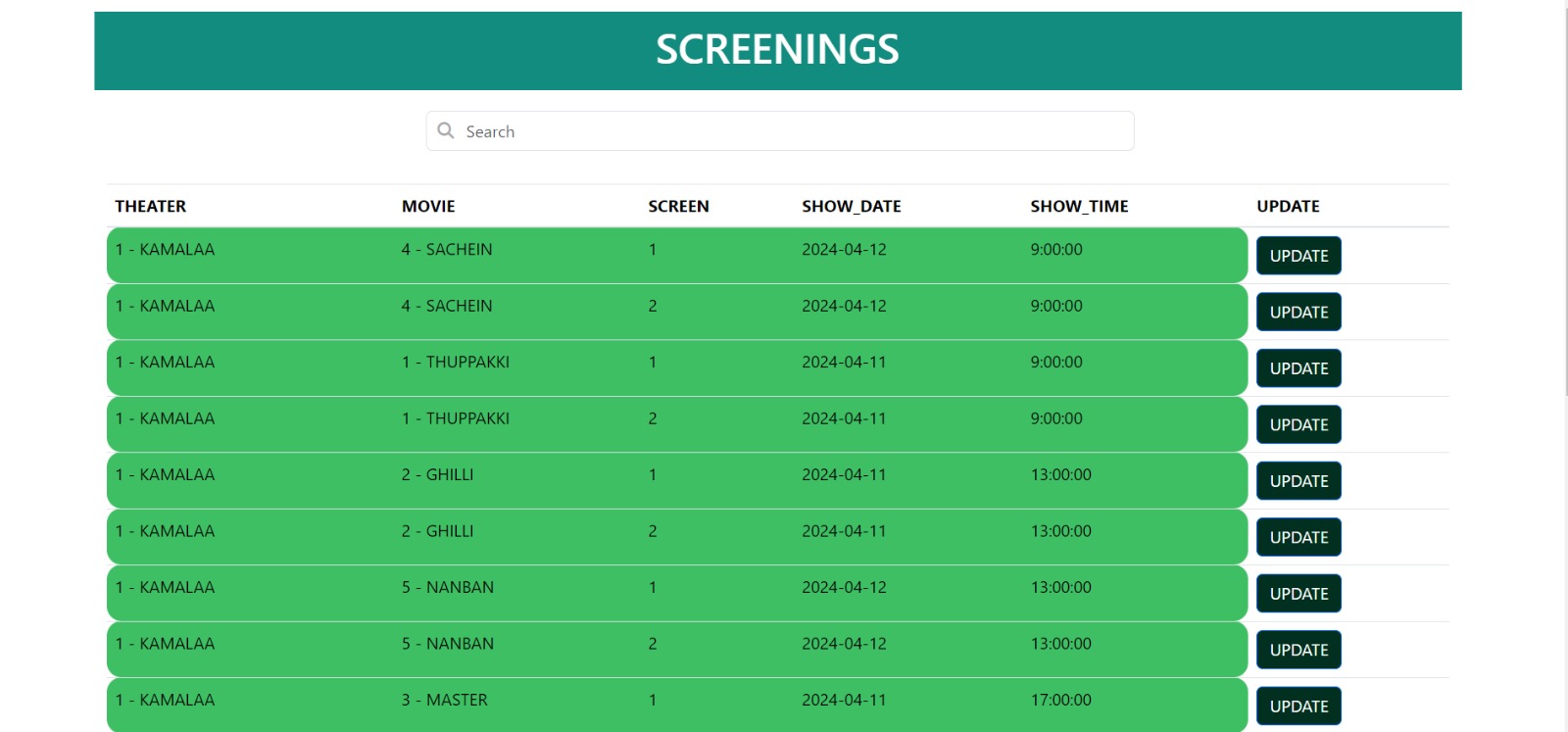
1. Theater details page

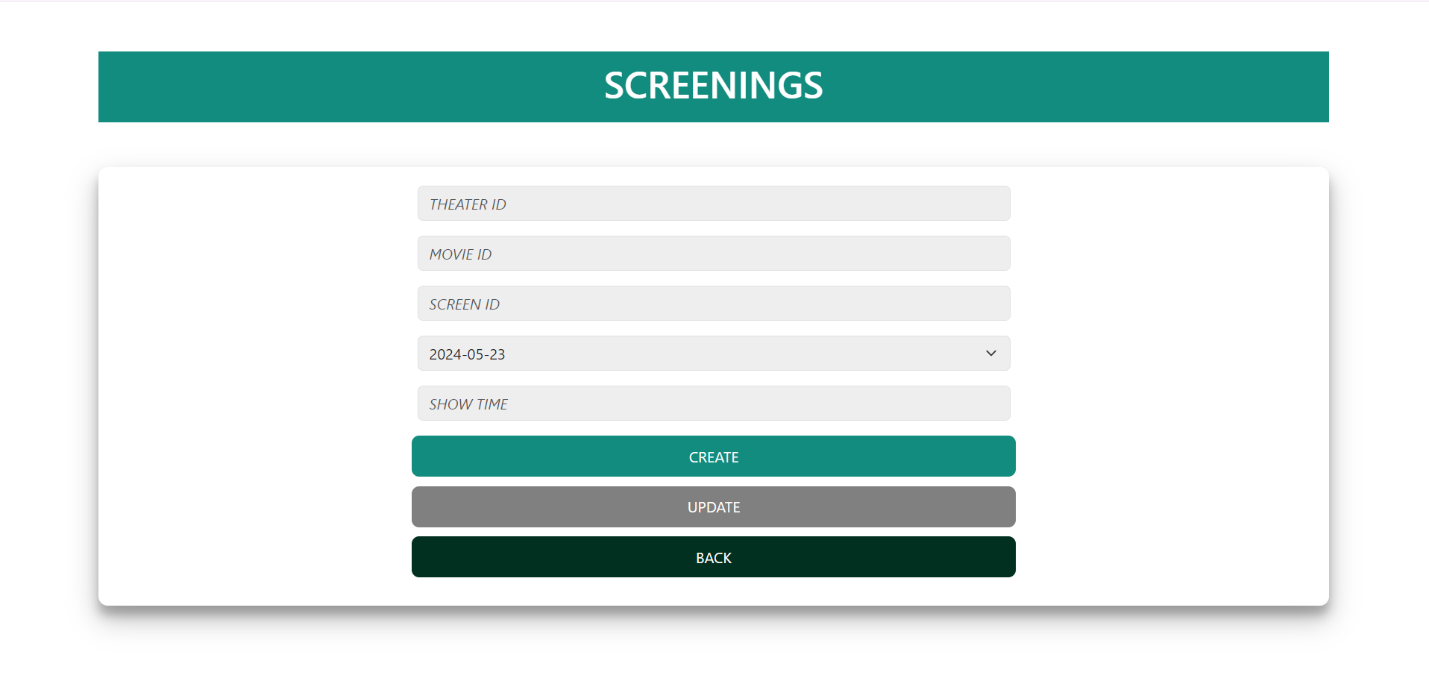


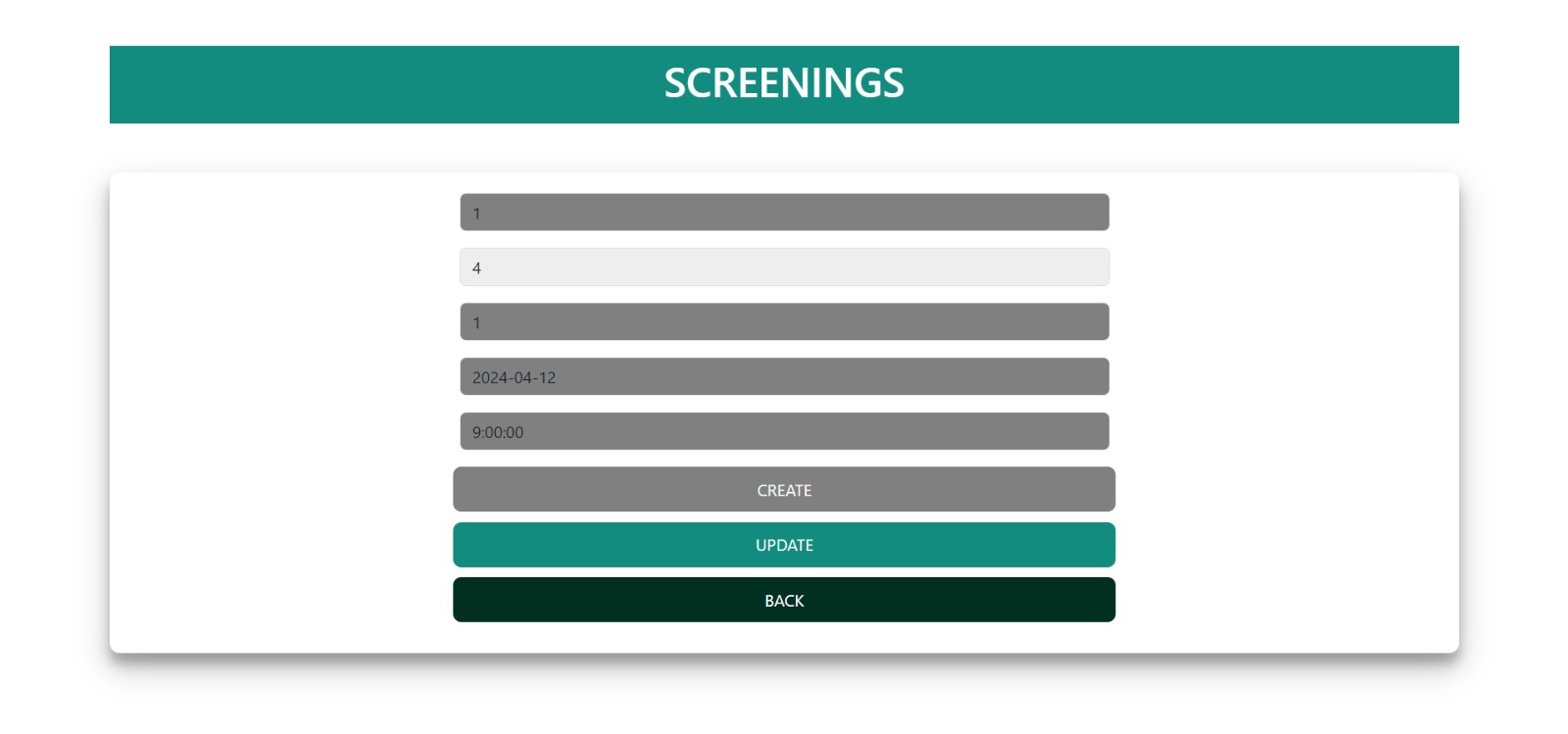
1. Theater table insertion from the admin side,
2. Theater table updation page,



1) The movies on theaters table.



****2) Insertion of the movies on theaters table

1. Updation of the movies on theaters table

**Triggers, Procedures, Functions and Cursors**

--TO NOT UPDATE RUNTIME AFTER SCHEDULING

DELIMITER //(TRIGGER 1)

CREATE OR REPLACE TRIGGER before\_runtime\_update\_on\_scheduled\_movies

BEFORE UPDATE ON movies FOR EACH ROW

BEGIN

DECLARE movie\_exists INT;

IF NEW.run\_time != OLD.run\_time THEN

-- Check if the movie is present in the TM table

SELECT COUNT(\*) INTO movie\_exists

FROM TM

WHERE MOVIE\_ID = NEW.MOVIE\_ID;

-- If the movie is present in the TM table, prevent the update

IF movie\_exists > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot update run time of a movie scheduled for screening.';

END IF;

END IF;

END //

DELIMITER ;

--ENSURES THAT IF NEW MOVIE IS INSERTED IT IS ATLEAST INSERTED AFTER 1 HOUR AFTER ITS PREVIOUS MOVIE GETS OVER (SHOW\_TIME BASED)

--AND IT IS ATLEAST INSERTED AFTER 1 HOUR BEFORE THE NEXT MOVIE STARTS.

DELIMITER //(TRIGGER 2)

CREATE OR REPLACE TRIGGER before\_tm\_insert\_1hour\_hibernation

BEFORE INSERT ON TM FOR EACH ROW

BEGIN

DECLARE prev\_run\_time TIME;

DECLARE prev\_movie\_start\_time TIME;

DECLARE prev\_movie\_end\_time TIME;

DECLARE new\_run\_time TIME;

DECLARE new\_movie\_start\_time TIME;

DECLARE new\_movie\_end\_time TIME;

DECLARE next\_run\_time TIME;

DECLARE next\_movie\_start\_time TIME;

DECLARE next\_movie\_end\_time TIME;

DECLARE prev\_movieid INT;

DECLARE next\_movieid INT;

DECLARE time\_diff\_before TIME;

DECLARE time\_diff\_after TIME;

SELECT TM.MOVIE\_ID, SHOW\_TIME

INTO prev\_movieid, prev\_movie\_start\_time

FROM TM

WHERE THEATER\_ID = NEW.THEATER\_ID

AND SCREEN\_ID = NEW.SCREEN\_ID

AND SHOW\_DATE = NEW.SHOW\_DATE

AND SHOW\_TIME < NEW.SHOW\_TIME

ORDER BY SHOW\_TIME DESC

LIMIT 1;

SELECT TM.MOVIE\_ID, SHOW\_TIME

INTO next\_movieid, next\_movie\_start\_time

FROM TM

WHERE THEATER\_ID = NEW.THEATER\_ID

AND SCREEN\_ID = NEW.SCREEN\_ID

AND SHOW\_DATE = NEW.SHOW\_DATE

AND SHOW\_TIME > NEW.SHOW\_TIME

ORDER BY SHOW\_TIME ASC

LIMIT 1;

SET prev\_run\_time = MAKETIME(CEIL((SELECT RUN\_TIME FROM MOVIES WHERE MOVIE\_ID = prev\_movieid) / 60), 0, 0);

SET prev\_movie\_end\_time = ADDTIME(prev\_movie\_start\_time, prev\_run\_time);

SET new\_run\_time = MAKETIME(CEIL((SELECT RUN\_TIME FROM MOVIES WHERE MOVIE\_ID = NEW.MOVIE\_ID) / 60), 0, 0);

SET new\_movie\_start\_time = NEW.SHOW\_TIME;

SET new\_movie\_end\_time = ADDTIME(new\_movie\_start\_time, new\_run\_time);

SET next\_run\_time = MAKETIME(CEIL((SELECT RUN\_TIME FROM MOVIES WHERE MOVIE\_ID = next\_movieid) / 60), 0, 0);

SET next\_movie\_end\_time = ADDTIME(next\_movie\_start\_time, next\_run\_time);

SET time\_diff\_before = TIMEDIFF(new\_movie\_start\_time, prev\_movie\_end\_time);

SET time\_diff\_after = TIMEDIFF(next\_movie\_start\_time, new\_movie\_end\_time);

IF (time\_diff\_before < '01:00:00') THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'TIME GAP BETWEEN 2 SCREENINGS SHOULD BE ATLEAST 1 HOUR AFTER THE MOVIE ENDS.';

END IF;

IF (time\_diff\_after < '01:00:00') THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'TIME GAP BETWEEN 2 SCREENINGS SHOULD BE ATLEAST 1 HOUR AFTER THE MOVIE ENDS.';

END IF;

IF (time\_diff\_before = '00:00:00') THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'ANOTHER MOVIE IS SCREENED ENDS.';

END IF;

IF (time\_diff\_after = '00:00:00') THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'ANOTHER MOVIE IS SCREENED.';

END IF;

END//

DELIMITER ;

--TO NOT ADD 2 MOVIES AT SAME SCREEN, TIME, DATE, THEATER

DELIMITER //(TRIGGER 3)

CREATE TRIGGER prevent\_duplicate\_movie

BEFORE INSERT ON TM

FOR EACH ROW

BEGIN

DECLARE existing\_movie\_count INT;

SELECT COUNT(\*)

INTO existing\_movie\_count

FROM TM

WHERE THEATER\_ID = NEW.THEATER\_ID

AND SCREEN\_ID = NEW.SCREEN\_ID

AND SHOW\_DATE = NEW.SHOW\_DATE

AND SHOW\_TIME = NEW.SHOW\_TIME;

IF existing\_movie\_count > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot insert. Another movie is scheduled for the same theater, screen, date, and time.';

END IF;

END //

DELIMITER ;

--TO AUTO INCREMENT BOOKING ID (DOESN'T INCREASES EVEN IF ERROR OCCURS)(TRIGGER 4)

DELIMITER //

CREATE TRIGGER set\_booking\_id BEFORE INSERT ON bookings

FOR EACH ROW

BEGIN

DECLARE last\_id INT;

SELECT MAX(booking\_id) INTO last\_id FROM bookings;

IF last\_id IS NULL THEN

SET NEW.booking\_id = 1;

ELSE

SET NEW.booking\_id = last\_id + 1;

END IF;

END //

DELIMITER ;

--TO INCREMENT USER ID(TRIGGER 5)

DELIMITER //

CREATE TRIGGER set\_user\_id BEFORE INSERT ON users

FOR EACH ROW

BEGIN

DECLARE last\_id INT;

SELECT MAX(user\_id) INTO last\_id FROM users;

IF last\_id IS NULL THEN

SET NEW.user\_id = 1;

ELSE

SET NEW.user\_id = last\_id + 1;

END IF;

END //

DELIMITER ;

--PROCEDURE TO DELETE THEATER IF MOVIE IS NOT SCREENED

DELIMITER //(PROCEDURE)

CREATE OR REPLACE PROCEDURE DELETE\_THEATER (IN theater\_id\_param INT)

BEGIN

DECLARE theaters\_exist INT;

-- Check if the theater exists in the TM table

SELECT COUNT(\*) INTO theaters\_exist FROM TM WHERE THEATER\_ID = theater\_id\_param;

-- If the theater does not exist in the TM table, delete its screens and then delete the theater

IF theaters\_exist = 0 THEN

BEGIN

DELETE FROM SCREENS WHERE THEATER\_ID = theater\_id\_param;

DELETE FROM THEATERS WHERE THEATER\_ID = theater\_id\_param;

END;

ELSE

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot insert. Another movie is scheduled for the same theater, screen, date, and time.';

END IF;

END //

DELIMITER ;

--FUNCTION TO RETURN REVENUE OF EACH THEATER

DELIMITER //(FUNCTION 1)

CREATE OR REPLACE FUNCTION calculate\_theater\_revenue(theater\_id\_param INT)

RETURNS DECIMAL(10, 2)

BEGIN

DECLARE elite\_total DECIMAL(10, 2);

DECLARE premium\_total DECIMAL(10, 2);

DECLARE total DECIMAL(10, 2);

-- Calculate total price for elite seats

SELECT COALESCE(SUM(no\_of\_elite\_seats) \* 150, 0) INTO elite\_total

FROM BOOKINGS

WHERE theater\_id = theater\_id\_param;

-- Calculate total price for premium seats

SELECT COALESCE(SUM(no\_of\_premium\_seats) \* 190, 0) INTO premium\_total

FROM BOOKINGS

WHERE theater\_id = theater\_id\_param;

-- Add total price for elite and premium seats

SET total = elite\_total + premium\_total;

-- Return the total price

RETURN total;

END //

DELIMITER ;

--FUNCTION TO RETURN REVENUE OF EACH MOVIE

DELIMITER //(FUNCTION 2)

CREATE OR REPLACE FUNCTION calculate\_movie\_revenue(movie\_id\_param INT) RETURNS DECIMAL(10, 2)

BEGIN

DECLARE total\_revenue\_1 DECIMAL(10, 2);

DECLARE total\_revenue\_2 DECIMAL(10, 2);

DECLARE total\_revenue DECIMAL(10, 2);

-- Calculate revenue for elite seats

SELECT COALESCE(SUM(no\_of\_elite\_seats) \* 150, 0) INTO total\_revenue\_1

FROM BOOKINGS

WHERE movie\_id = movie\_id\_param;

-- Add revenue for premium seats

SELECT COALESCE(SUM(no\_of\_premium\_seats) \* 190, 0) INTO total\_revenue\_2

FROM BOOKINGS

WHERE movie\_id = movie\_id\_param;

-- Calculate total revenue

SET total\_revenue = total\_revenue\_1 + total\_revenue\_2;

-- Return the total revenue for the movie

RETURN total\_revenue;

END //

DELIMITER ;

--CURSOR TO CALCULATE ACTIVE USERS

DELIMITER //(CURSOR 1)

CREATE OR REPLACE PROCEDURE active\_users()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE user\_count INT DEFAULT 0;

DECLARE user\_id\_current INT;

DECLARE user\_cursor CURSOR FOR

SELECT DISTINCT user\_id FROM BOOKINGS;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN user\_cursor;

read\_loop: LOOP

FETCH user\_cursor INTO user\_id\_current;

IF done THEN

LEAVE read\_loop;

END IF;

SET user\_count = user\_count + 1;

END LOOP;

CLOSE user\_cursor;

SELECT CONCAT('Total active users: ', user\_count) AS user\_count;

END//

DELIMITER ;

--CURSOR TO CALCULATE TOTAL MOVIES

DELIMITER //(CURSOR 2)

CREATE OR REPLACE PROCEDURE calculate\_movie\_count()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE movie\_count INT DEFAULT 0;

DECLARE movie\_id\_current INT;

DECLARE movie\_cursor CURSOR FOR

SELECT movie\_id FROM MOVIES;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN movie\_cursor;

read\_movie\_loop: LOOP

FETCH movie\_cursor INTO movie\_id\_current;

IF done THEN

LEAVE read\_movie\_loop;

END IF;

SET movie\_count = movie\_count + 1;

END LOOP;

CLOSE movie\_cursor;

SELECT movie\_count;

END//

DELIMITER ;

--CURSOR TO CALCULATE TOTAL THEATERS

DELIMITER //(CURSOR 3)

CREATE OR REPLACE PROCEDURE calculate\_theater\_count()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE theater\_count INT DEFAULT 0;

DECLARE theater\_id\_current INT;

DECLARE theater\_cursor CURSOR FOR

SELECT theater\_id FROM THEATERS;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN theater\_cursor;

read\_theater\_loop: LOOP

FETCH theater\_cursor INTO theater\_id\_current;

IF done THEN

LEAVE read\_theater\_loop;

END IF;

SET theater\_count = theater\_count + 1;

END LOOP;

CLOSE theater\_cursor;

SELECT theater\_count;

END//

DELIMITER ;

**LIMITATIONS :**

* The screenings in the admin side can only updated, insertion is done directly via the database.
* Realtime payment is not done.
* Theaters is the Chennai region is alone present in the database.

We have done this project as a prototype for a real time project, in the future these limitations will be corrected.

**CONCLUSION :**

In conclusion, our project revolutionizes the movie ticket reservation experience by prioritizing user convenience, personalization, and efficiency. With advanced features such as personalized recommendations, real-time updates, and seamless ticket reservation, we aim to elevate the movie-going experience for audiences worldwide.

This project also aims to provide admin a convenient way for the proper oversight and management of the database ensuring no redundancy and preventing conflicts by providing flawless and smooth interface for editing movies, theaters and screenings.

By prioritizing user satisfaction and accessibility, we envision our platform becoming the go-to choice for movie enthusiasts, driving growth and modernization in the entertainment industry.