ATILIM UNIVERSITY

2024-2025 FALL SEMESTER

CMPE341-Database Design and Management

Movie - Ticket Reservation System

Group Members:

Reşat Berkut TURGUT – 23243510014

Mohamed Anies Awad SATTI – 22243510145

Ali Çağrı SEPET – 21243510096

Pelin GENÇ – 20243510018

Batuhan Taha YEŞİLYURT – 21243510013

Supervisor:

Aysu İrem ADEM

Table of Contents

1. Introduction

This report presents a database system solution to address some inefficiencies in the traditional cinema ticket booking processes. Our team who has been examining several current ticket booking websites has identified the underserved need for a streamlined system that simplifies operations and improves customer satisfaction.

With the principles of database design from [1], and insights gathered from additional resources including [2], [3], and [4], we decided to implement a Movie-Ticket Reservation System designed to manage key aspects of cinema operations effectively. The system addresses challenges such as the management of different movie genres, movie schedules and showtimes, the tracking of availability in all branches, and the smooth ticket booking for customers. It also meets the specific requirements of each branch, making cinema chains able to efficiently organize their operations in multiple locations.

Our main objective through this project is to deal with the problems of long queues, double booking, and communication delays. Instead, we plan to offer them a hassle-free booking experience, and a powerful tool around automation that will make the best use of resources.

2. Requirements

2.1. User's expectations

- The system should handle operations such as movie scheduling, hall assignments, and reserving tickets operations efficiently.
- The system should allow customers to browse available movies and their showtimes in several cinema branches.
- Customers should be able to book tickets for specific showtime.
- Admin should schedule showtimes for movies across different cinema branches by specifying movie's show date and time, and hall information for each branch.
- Admin should be able to display and analyse ticket sales, popular movies, and showtime performance.

2.2. Stored data

The following data (entities and their attributes) should be stored in the database:

- Branch: Unique ID, name, contact information, address (street number, district, city).
- Hall: Unique number (within branch), capacity, related branch ID.
- Movie: Unique ID, name, duration, release date, rating, age restrictions.
- Genre: Unique ID, name, related movies ID.
- Showtime: Unique ID, date, time, associated movie, associated hall.
- Ticket: Unique ID, price, booking date, booking time, seat number, associated showtime.
- Customer: Unique ID, name, date of birth, phone number, email address.

2.3. Operations needed on data

- Branch: Add, update, and view branch details.
- Hall: Add, update, and view hall details within a branch.
- Movie: Add, update, and view movie details.
- Genre: Add new genres and associate them with movies.

- Showtime: Schedule showtimes, update schedules, and search for schedules by branch, movie, or hall.
- Ticket: Reserve tickets, manage bookings, and retrieve ticket details.
- Customer: Add new customer records, update details, and retrieve booking history.

3. ER/EER Diagram of The System

For our conceptual design, shown in Fig. 1 & 2, we used software tools [5], and [6] to draw ER diagrams. We have 7 entities; one of them is a weak entity. The regular entities are *Customer*, *Movie*, *Ticket*, *Showtime*, *Branch*, and *Genre*, although it can be an attribute for *Movie*, we preferred to make it as a separate entity because a movie can have multiple genres which violates normalization rules [1]. And one weak entity which is *Hall*, since a hall can't stand alone without a *Branch*. Additionally, we have 6 relationships as follows:

- A *Customer* BOOKS a *Ticket/Tickets*
- A *Ticket/Tickets* BELONGS TO a *Showtime*
- *Showtime/Showtimes* is/are SCHEDULED FOR a *Movie*
- A *Movie* HAS a *Genre/Genres*
- *Showtime* OCCURS IN a *Hall*
- A cinema *Branch* CONTAINS a *Hall/Halls*

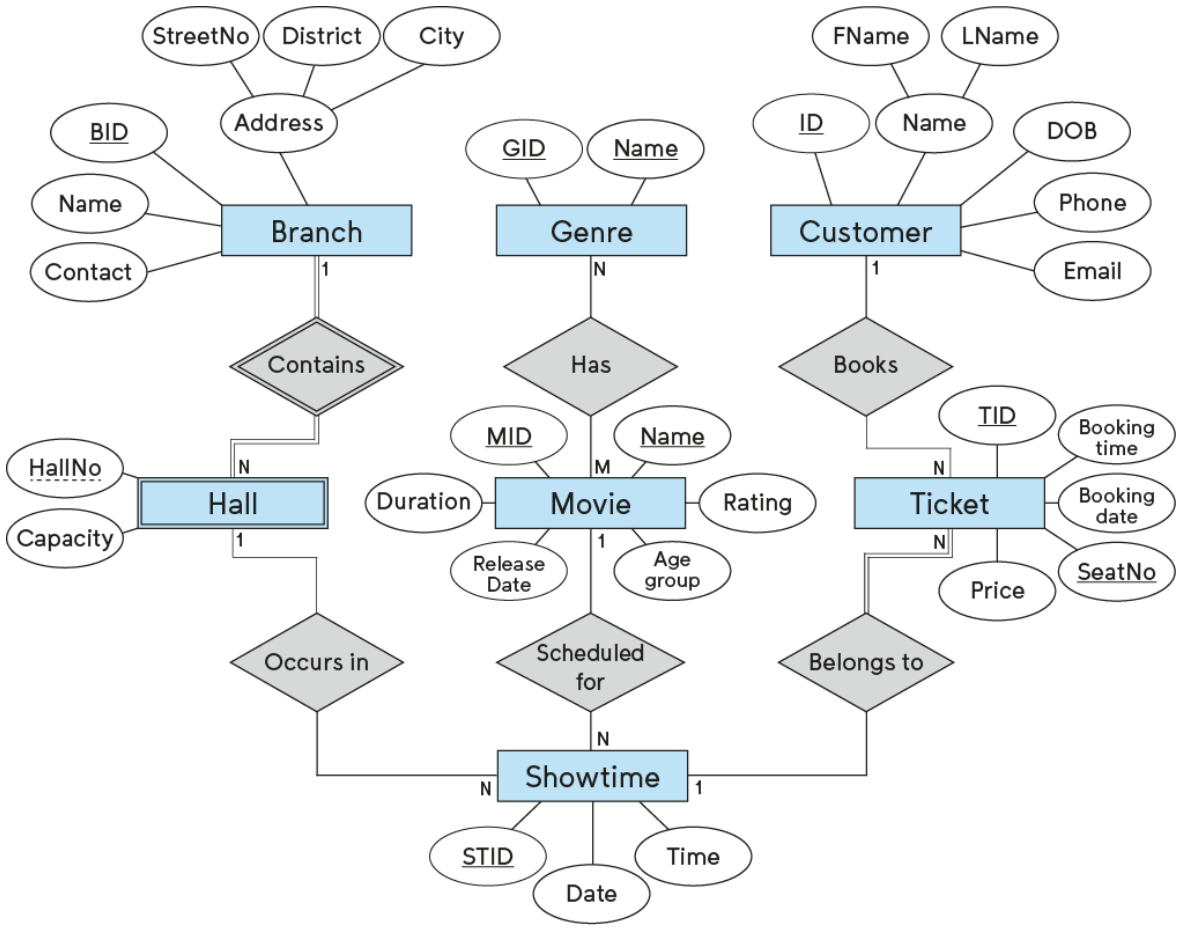

Figure 1. ER Diagram (Chen's Notation)

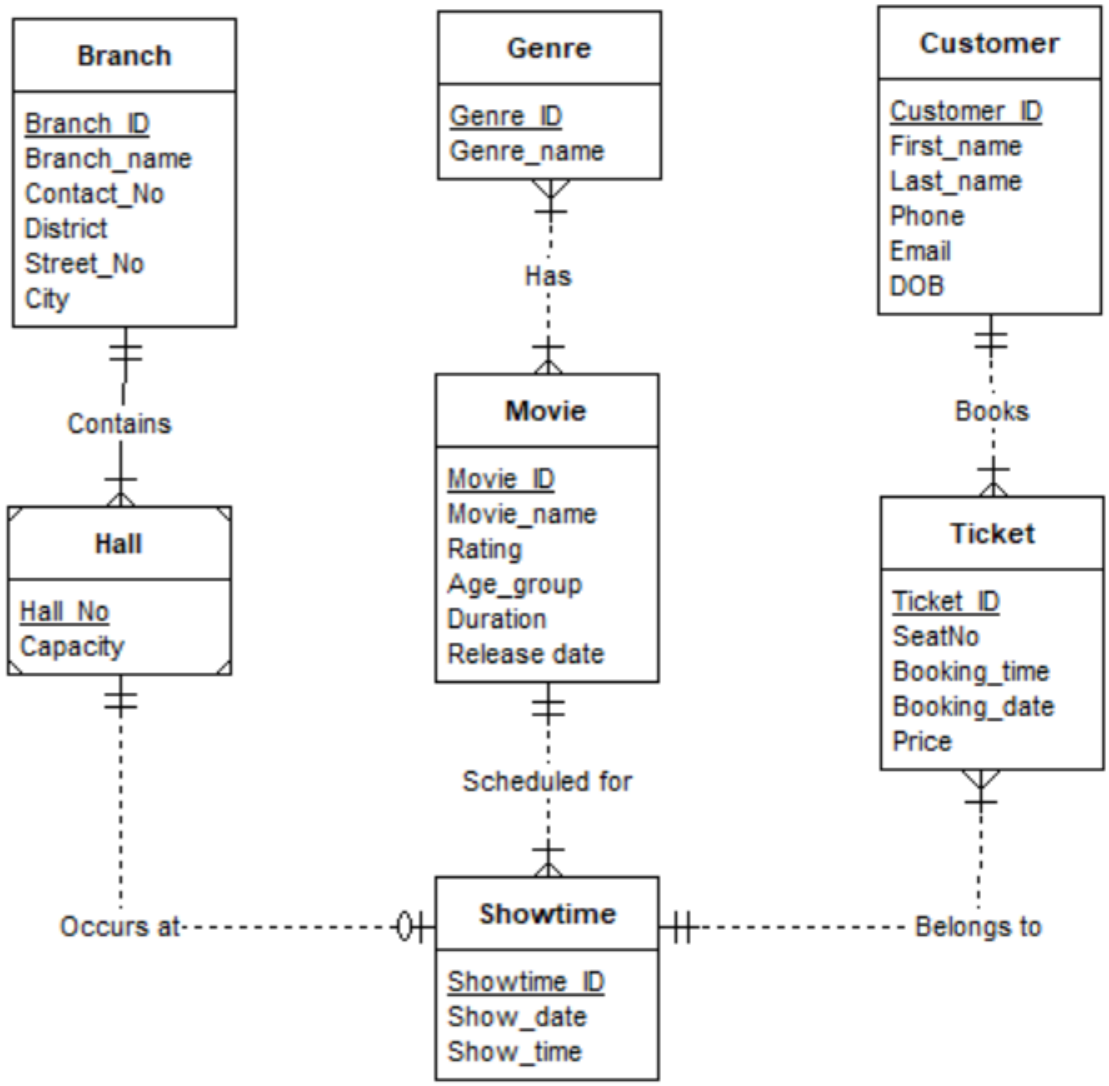

Figure 2. ER Diagram (Crow's Foot Notation)

4. Logical Design

In the logical design part, we mapped our ERD into a logical schema as follows:

- **Regular entities:**

Customer [Customer_ID, First_name, Last_name, DOB, Phone_no, Email]

Movie [Movie_ID, Movie_name, Release_date, Age_group, Duration, Rating]

Ticket [Ticket_ID, Booking_date, Booking_time, Price, Seat_no, Customer_ID (FK refers to *Customer*), Showtime_ID (FK refers to *Showtime*)]

Showtime [Showtime_ID, Show_date, Show_time, Movie_ID (FK refers to *Movie*), Branch_ID, Hall_No ({Branch_ID, Hall_No} FK refers to *Hall*)]

Branch [Branch_ID, Branch_name, Street_no, District, City, Contact_no]

Genre [Genre_ID, Genre_name]

- **Weak entities:**

Hall [Branch_ID (FK refers to *Branch*), Hall_No, Capacity]

- **Many-to-many relations:**

Has_Genre [Movie_ID (FK refers to *Movie*), Genre_ID (FK refers to *Genre*)]

5. Physical Design

5.1. Tables

For the physical design, we are using Oracle SQL Developer. Firstly, we should create the tables based on the logical schema, then we have to load them with data. All of the movies' data were collected from Internet Movie Database (IMDb) [7].

```
CREATE TABLE Customer (
        customer_id NUMBER(6),
        first_name VARCHAR2(30),
        last_name VARCHAR2(30),
        dob DATE,
        phone_no NUMBER(11) NOT NULL,
        email VARCHAR2(30) NOT NULL UNIQUE,
        PRIMARY KEY(customer_id)
);
```

```
INSERT INTO Customer
VALUES (1001, 'Berkut', 'Turgut', '01-JAN-1996', 5554443322, 'turgut@cmpe341.com');
```

```
INSERT INTO Customer
VALUES (1002, 'Ali', 'Sepet', '01-JAN-1998', 5554443333, 'sepet@cmpe341.com');
```

```
INSERT INTO Customer
VALUES (1003, 'Pelin', 'Genc', '01-JAN-1999', 5554443344, 'genc@cmpe341.com');
```

```
INSERT INTO Customer
VALUES (1004, 'Batuhan', 'Yesilyurt', NULL, 5554443355, 'yesilyurt@cmpe341.com');
```

```
INSERT INTO Customer
VALUES (1005, 'Mohamed', 'Satti', '01-JAN-1998', 5554443366, 'satti@cmpe341.com');
```

	CUSTOMER_ID	FIRST_NAME	LAST_NAME	DOB	PHONE_NO	EMAIL
1	1001	Berkut	Turgut	01-JAN-96	5554443322	turgut@cmpe341.com
2	1002	Ali	Sepet	01-JAN-98	5554443333	sepet@cmpe341.com
3	1003	Pelin	Genc	01-JAN-99	5554443344	genc@cmpe341.com
4	1004	Batuhan	Yesilyurt	(null)	5554443355	yesilyurt@cmpe341.com
5	1005	Mohamed	(null)	01-JAN-98	5554443366	satti@cmpe341.com

Figure 3. Customer table

```
CREATE TABLE Movie (
        movie_id NUMBER(6),
        movie_name VARCHAR2(40) NOT NULL,
        release_date DATE,
        age_group VARCHAR2(10),
        duration_in_minutes NUMBER(3),
        rating NUMBER(2,1),
        PRIMARY KEY(movie_id) UNIQUE
);
```

```
INSERT INTO Movie
VALUES (1, 'The Man from Toronto', '24-JUN-2022', 'PG-13', 110, 5.8);
INSERT INTO Movie
VALUES (2, 'Home Alone', '10-NOV-1990', 'PG', 103, 7.7);
INSERT INTO Movie
VALUES (3, 'The Dark Knight', '18-JUL-2008', 'PG-13', 152, 9.0);
INSERT INTO Movie
VALUES (4, 'Up', '23-JUL-2009', 'PG', 96, 8.3);
```

	MOVIE_ID	MOVIE_NAME	RELEASE_DATE	AGE_GROUP	DURATION_IN_MINUTES	RATING
1	1	The Man from Toronto	24-JUN-24	PG-13	110	5.8
2	2	Home Alone	10-NOV-90	PG	103	7.7
3	3	The Dark Knight	18-JUL-08	PG-13	152	9
4	4	Up	23-JUL-09	PG	96	8.3

Figure 4. Movie table

CREATE TABLE Branch (

branch_id NUMBER(6),

branch_name VARCHAR2(30) NOT NULL,

street_no NUMBER(8) NOT NULL,

district NUMBER(20) NOT NULL,

city VARCHAR2(20) NOT NULL,

contact_no NUMBER(11) NOT NULL UNIQUE,

PRIMARY KEY(branch_id)

);

INSERT INTO Branch
VALUES (060010, 'FutureVision-Golbasi', 1176, 'Incek', 'Ankara', 03124440999**);**

INSERT INTO Branch
VALUES (060011, 'FutureVision-Cankaya', 450, 'Kizilay', 'Ankara', 03124440998**);**

INSERT INTO Branch
VALUES (340010, 'FutureVision-Taksim', 999, 'Beyoglu', 'Istanbul', 02124440997**);**

	BRANCH_ID	BRANCH_NAME	STREET_NO	DISTRICT	CITY	CONTACT_NO
1	60010	FutureVision-Golbasi	1176	Incek	Ankara	3124440999
2	60011	FutureVision-Cankaya	450	Kizilay	Ankara	3124440998
3	340010	FutureVision-Taksim	999	Beyoglu	Istanbul	2124440997

Figure 5. Branch table

CREATE TABLE Hall (

hall_no NUMBER(10),

capacity NUMBER(4),

branch_id NUMBER(6),

PRIMARY KEY(branch_id, hall_no),

FOREIGN KEY(branch_id) REFERENCES Branch(branch_id)

);

INSERT INTO Hall **VALUES (**1, 100, 060010**);**

INSERT INTO Hall **VALUES (**1, 250, 060011**);**

INSERT INTO Hall **VALUES (**2, 200, 060011**);**

INSERT INTO Hall **VALUES (**1, 300, 340010**);**

INSERT INTO Hall **VALUES (**2, 250, 340010**);**

INSERT INTO Hall **VALUES (**3, 250, 340010**);**

	HALL_NO	CAPACITY	BRANCH_ID
1	1	100	60010
2	1	250	60011
3	2	200	60011
4	1	300	340010
5	2	250	340010
6	3	250	340010

Figure 6. Hall table

CREATE TABLE Showtime (

showtime_id NUMBER6),

show_date DATE NOT NULL,

show_time VARCHAR2(5) NOT NULL,

movie_id NUMBER(6) NOT NULL,

branch_id NUMBER(6) NOT NULL,

hall_no NUMBER(10),

PRIMARY KEY(showtime_id),

FOREIGN KEY(movie_id) REFERENCES Movie(movie_id),

FOREIGN KEY(branch_id, hall_no) REFERENCES Hall(branch_id, hall_no)

);

INSERT INTO Showtime **VALUES (**1, '29-DEC-2024', '20:00', 1, 060011, 1**);**

INSERT INTO Showtime **VALUES (**2, '31-DEC-2024', '18:30', 2, 060011, 1**);**

INSERT INTO Showtime **VALUES (**3, '31-DEC-2024', '18:30', 2, 340010, 3**);**

INSERT INTO Showtime **VALUES (**4, '30-DEC-2024', '19:00', 3, 060010, 1**);**

	SHOWTIME_ID	SHOW_DATE	SHOW_TIME	MOVIE_ID	BRANCH_ID	HALL_NO
1	1	29-DEC-24	20:00	1	60011	1
2	2	31-DEC-24	18:30	2	60011	1
3	3	31-DEC-24	18:30	2	340010	3
4	4	30-DEC-24	19:00	3	60010	1

Figure 7. Showtime table

CREATE TABLE Ticket (

ticket_id NUMBER(6),

booking_date DATE DEFAULT SYSDATE,

booking_time TIMESTAMP DEFAULT SYSTIMESTAMP,

price NUMBER(5,2),

seat_no NUMBER (3),

customer_id NUMBER(6),

showtime_id NUMBER(6),

PRIMARY KEY(ticket_id),

FOREIGN KEY(customer_id) REFERENCES Customer(customer_id),

FOREIGN KEY(showtime_id) REFERENCES Showtime(showtime_id),

CONSTRAINT unique_showtime_seat UNIQUE (showtime_id, seat_no)

);

INSERT INTO Ticket **VALUES (**1, default, default, 120.0, 12, 1001, 1**);**

INSERT INTO Ticket **VALUES (**2, default, default, 120.0, 25, 1005, 2**);**

INSERT INTO Ticket **VALUES (**3, default, default, 120.0, 10, 1004, 1**);**

	TICKET_ID	BOOKING_DATE	BOOKING_TIME	PRICE	SEAT_NO	CUSTOMER_ID	SHOWTIME_ID
1	1	25-DEC-24	25-DEC-24 19.46.48.103000000	120	12	1001	1
2	2	25-DEC-24	25-DEC-24 19.52.02.149000000	120	25	1005	2
3	3	25-DEC-24	25-DEC-24 19.51.53.359000000	120	10	1004	1

Figure 8. Ticket table

```
CREATE TABLE Genre (
        genre_id NUMBER(6),
        genre_name VARCHAR2(20),
        PRIMARY KEY(genre_id)
);
```

```
INSERT INTO Genre VALUES (1, 'Action');
INSERT INTO Genre VALUES (2, 'Comedy');
INSERT INTO Genre VALUES (3, 'Family');
INSERT INTO Genre VALUES (4, 'Drama');
INSERT INTO Genre VALUES (5, 'Adventure');
INSERT INTO Genre VALUES (6, 'Animation');
INSERT INTO Genre VALUES (7, 'Crime');
INSERT INTO Genre VALUES (8, 'Thriller');
```

	GENRE_ID	GENRE_NAME
1	1	Action
2	2	Comedy
3	3	Family
4	4	Drama
5	5	Adventure
6	6	Animation
7	7	Crime
8	8	Thriller

Figure 9. Genre table

```
CREATE TABLE Has_genre (
        movie_id NUMBER(6),
        genre_id NUMBER(6),
        PRIMARY KEY (movie_id, genre_id),
        FOREIGN KEY(movie_id) REFERENCES Movie(movie_id),
        FOREIGN KEY(genre_id) REFERENCES Genre(genre_id)
);

INSERT INTO Has_genre VALUES (1, 1);
INSERT INTO Has_genre VALUES (1, 2);
INSERT INTO Has_genre VALUES (1, 5);
INSERT INTO Has_genre VALUES (1, 8);
INSERT INTO Has_genre VALUES (2, 2);
INSERT INTO Has_genre VALUES (2, 3);
INSERT INTO Has_genre VALUES (3, 1);
INSERT INTO Has_genre VALUES (3, 4);
INSERT INTO Has_genre VALUES (3, 7);
INSERT INTO Has_genre VALUES (3, 8);
INSERT INTO Has_genre VALUES (4, 2);
INSERT INTO Has_genre VALUES (4, 3);
INSERT INTO Has_genre VALUES (4, 4);
INSERT INTO Has_genre VALUES (4, 5);
INSERT INTO Has_genre VALUES (4, 6);
```

	MOVIE_ID	GENRE_ID
1	1	1
2	1	2
3	1	5
4	1	8
5	2	2
6	2	3
7	3	1
8	3	4
9	3	7
10	3	8
11	4	2
12	4	3
13	4	4
14	4	5
15	4	6

Figure 10. Has_genre table

5.2. Queries

- **Displaying the tickets sold and the total income of the cinema during 2024:**

```
SELECT count(ticket_id) "Tickets sold", sum(price) "Total income"
FROM ticket WHERE booking_date BETWEEN '01-JAN-24' AND '31-DEC-24';
```

	Tickets sold	Total income
1	3	360

Figure 11. Sold tickets and total income.

- **Showing availability of seats in a specific Showtime (using JOIN & GROUP BY statements):**

```
SELECT capacity "Total capacity", (capacity - COUNT(T.seat_no)) "Available seats"
FROM Hall h
JOIN Showtime s ON h.hall_no = s.hall_no AND h.branch_id = s.branch_id
LEFT JOIN Ticket t ON s.showtime_id = t.showtime_id
WHERE s.showtime_id = 1
GROUP BY capacity;
```

	Total capacity	Available seats
1	250	248

Figure 12. Available seats at showtime (1)

- **Displaying movies in a specific city (using Subqueries):**

```
SELECT movie_name FROM Movie
WHERE movie_id IN (SELECT movie_id
                   FROM Showtime
                   WHERE branch_id IN (SELECT branch_id
                                       FROM Branch
                                       WHERE city = 'Ankara'));
```

	MOVIE_NAME
1	The Man from Toronto
2	Home Alone
3	The Dark Knight

Figure 13. Movies shown in Ankara

6. Interface

Our web-based user interface for the Movie-Ticket Reservation System (https://cinema.mosatti.site/) provides a friendly way to interact with the database, which allows customers to perform transactions such as booking, updating, and cancelling a reservation. The interface is built using tools like cPanel, MySQL, PHP, and HTML. It has a simple design for efficient ticket booking and management experience.

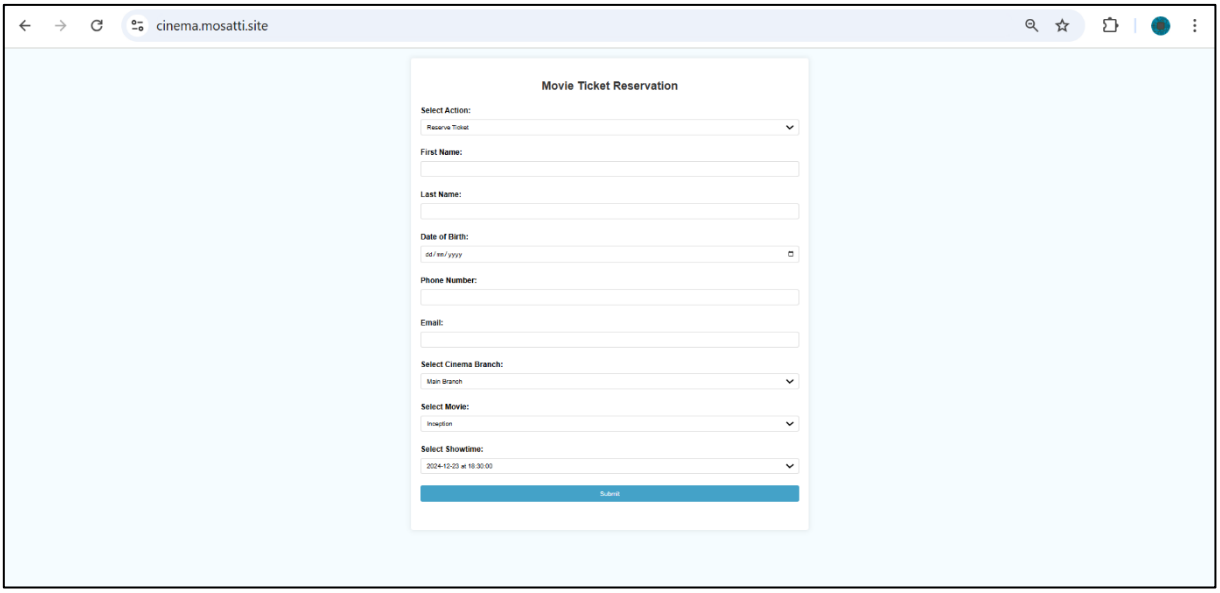

Figure 14. The web-based interface

Users can reserve tickets by selecting a branch, movie, and showtime. To edit the ticket, they must verify their reservation using the Ticket ID and the used email address. Once a transaction is made database gets updated in real-time.

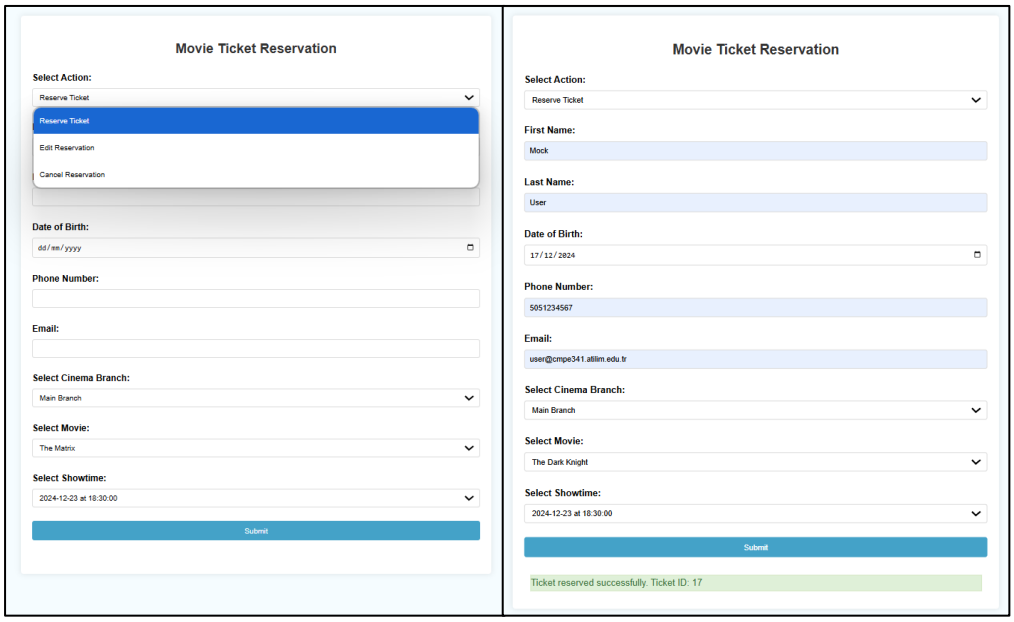

Figure 15. Reserving a ticket

Figure 16. Editing a previously reserved ticket

Additionally, customers can cancel bookings, using their Ticket ID and the email used at reservation process.

Figure 17. Cancelling a reservation

In future updates, we can develop our interface by adding more features including user authentication, online payments, enhanced reporting tools for admins, and notifications via email or as a SMS for better user experience.

References

[1] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, *7th ed*. USA: Pearson, 2017.

[2] Database Star, *Movie Theatre Database Design: Step-by-Step.* (Jun. 11, 2024). Accessed: Nov. 12, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=HYq3IL1ii70.

[3] H. Sathish, "Movie-Ticket-Booking-System," GitHub repository, 2020. Accessed: Nov. 12, 2024. [Online]. Available: https://github.com/hemantsathish/Movie-Ticket-Booking-System.

[4] IEEE Publication, *Reference Guide*. USA: IEEE, 2023.

[5] Adobe Illustrator (2021). Adobe Inc.

[6] ER Assistant 2.10. (2002). Mosor Inc.

[7] IMDb.com. [Online]. Available: https://www.imdb.com/. [Accessed: Dec. 10, 2024].