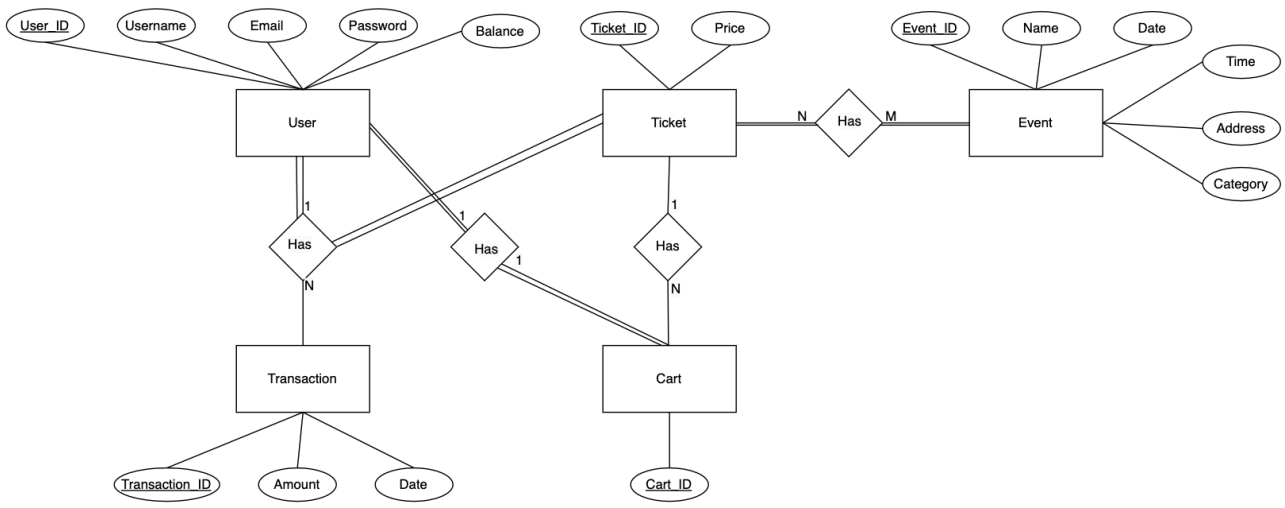
“EVENT TICKETING SYSTEM”

Begüm Ayten, Gülşah Arslan

1. Project Description

The main goal of our project is to create a database system for event tickets that resembles an actual-world scenario. This system includes several features, such as user management (data about registered users is stored in the User table); transaction handling (financial details of ticket purchases are recorded in the Transaction table); ticket management (the Ticket table shows individual ticket prices and cart associations); event organization (the Event table holds important data such as event names, dates, times, addresses, and categories); and lastly, the Event_Has_Tickets table links events to their corresponding tickets. By using a complete strategy, event organizers and users alike can benefit from the effective and well-organized management of ticketing.

2. Entity-Relationship Diagram



3. Relational Database Design

```
CREATE TABLE User(
    user_id INTEGER,
    user_name VARCHAR (255) NOT NULL,
    user_password VARCHAR (255) NOT NULL,
    user_email VARCHAR (255) NOT NULL,
    user_balance DOUBLE,
    cart_id INTEGER,
    PRIMARY KEY(user_id),
    INDEX (cart_id)
);
```

```
CREATE TABLE Event (
    event_id INTEGER,
    event_name VARCHAR (255) NOT NULL,
    event_date DATE,
    event_time TIME,
    event_adress VARCHAR (255) NOT NULL,
    event_category VARCHAR (255) NOT NULL,
    PRIMARY KEY(event_id)
);
```

```
CREATE TABLE Ticket (
    ticket_id INTEGER,
    ticket_price DOUBLE,
    cart_id INTEGER,
    PRIMARY KEY(ticket_id),
    FOREIGN KEY (cart_id) REFERENCES User(cart_id)
);
```

```sql
CREATE TABLE Transaction (
    transaction_id INTEGER,
    transaction_amount DOUBLE,
    transaction_date DATE,
    PRIMARY KEY(transaction_id),
    user_id INTEGER,
    ticket_id INTEGER,
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id)
);
```

```sql
CREATE TABLE Event_Has_Tickets (
    event_id INTEGER,
    ticket_id INTEGER,
    FOREIGN KEY (event_id) REFERENCES Event(event_id),
    FOREIGN KEY (ticket_id) REFERENCES Ticket(ticket_id),
    PRIMARY KEY(event_id, ticket_id)
);
```

4. Data Sources

Our event ticketing database has been populated with synthetic data that was manually created to simulate a realistic environment for development and testing purposes. The data inserted into the tables represents typical entities such as users, events, tickets, and transactions, ensuring a comprehensive testing ground for all functionalities of our ticketing system.

- The User table has been filled with 6 tuples, representing registered users, each with a unique identifier, name, password, email, balance, and associated cart.
- The Transaction table contains 6 tuples as well, each representing a financial record of a ticket purchase, linked to both a user and a ticket.
- In the Ticket table, there are 19 tuples, indicating individual tickets, their prices, and which cart they are associated with, if any.
- The Event table has been populated with 9 tuples, each detailing an event, including its name, date, time, address, and category.
- Lastly, the Event_Has_Tickets table holds 19 tuples that create a relationship between events and the tickets available for them.

```sql
INSERT INTO User (user_id, user_name, user_password, user_email, user_balance, cart_id)
VALUES (1, 'JohnDoe', 'password123', 'johndoe@example.com', '1000', 501),
       (2, 'JaneSmith', 'password456', 'janesmith@example.com', '2000', 502),
       (3, 'AliceJohnson', 'alicepass789', 'alicejohnson@example.com', '800',503),
       (4, 'MikeBrown', 'mikepass101', 'mikebrown@example.com', '1500', 504),
       (5, 'EmmaWilson', 'emmawilson321', 'emmawilson@example.com', '1900', 505),
       (6, 'RobertMiller', 'robertmiller654', 'robertmiller@example.com', '2500', 506);
```

```sql
INSERT INTO Ticket (ticket_id, ticket_price, cart_id)
VALUES (401, 100.00, 501),
       (402, 50.00, 502),
       (403, 120.00, 503),
       (404, 80.00, 504),
       (405, 60.00, 505),
       (406, 200.00, 506),
       (407, 200.00, NULL),
       (408, 200.00, NULL),
       (409, 100.00, NULL),
       (410, 100.00, 502),
       (411, 100.00, 503),
       (412, 50.00, 506),
       (413, 120.00, NULL),
       (414, 80 , NULL),
       (415, 80 , 501),
       (416, 60 , NULL),
       (417, 200 ,NULL),
       (418, 200, NULL),
       (419, 100, NULL);
```

```sql
INSERT INTO Transaction (transaction_id, transaction_amount, transaction_date, user_id, ticket_id)
VALUES (101, 100.00, '2024-01-15', 1, 401),
       (102, 50.00, '2024-01-20', 2, 402),
       (103, 120.00, '2024-01-25', 3, 403),
       (104, 80.00, '2024-02-05', 4, 404),
       (105, 60.00, '2024-02-15', 5, 405),
       (106, 200.00, '2024-02-20', 6, 406);
```

```
INSERT INTO Event (event_id, event_name, event_date, event_time, event_adress, event_category)
VALUES (301, 'Rock Concert', '2024-02-25', '19:00', '123 Music Ave', 'Concert'),
       (302, 'Shakespeare Play', '2024-03-10', '18:00', '456 Drama St', 'Theater'),
       (303, 'Jazz Festival', '2024-04-20', '17:00', '789 Jazz Blvd', 'Concert'),
       (304, 'Tech Conference', '2024-05-05', '09:00', '101 Tech Park', 'Conference'),
       (305, 'Art Exhibition', '2024-06-15', '14:00', '202 Art Lane', 'Art'),
       (306, 'Classical Music Concert', '2024-10-20', '19:30', '606 Symphony St', 'Concert'),
       (307, 'Film Festival', '2024-08-10', '17:00', '404 Cinema Road', 'Movie'),
       (308, 'New Year Gala', '2025-01-01', '20:00', '808 Celebration Hall', 'Festival'),
       (309, 'Marathon', '2024-11-15', '06:00', '606 Athletic Ave', 'Sports') ;
```

```
INSERT INTO Event_Has_Tickets (event_id, ticket_id)
VALUES (301, 401),
       (301, 410),
       (301, 411),
       (302, 402),
       (302, 412),
       (303, 403),
       (303, 413),
       (304, 404),
       (304 ,414),
       (304, 415),
       (305, 405),
       (305, 416),
       (306, 406),
       (307, 407),
       (307, 417),
       (308, 408),
       (308, 418),
       (309, 409),
       (309, 419);
```

5. Advanced SQL Queries

1)

```
// Get all events
app.get('/api/events', async (req, res) => {
    try {
        const connection = await mysql.createConnection(dbConfig);

        const [rows] = await connection.query(`
            SELECT DISTINCT E.*, T.ticket_price,
                (SELECT COUNT(*) FROM Event_Has_Tickets WHERE event_id = E.event_id) AS total_tickets,
                (SELECT COUNT(*) FROM Transaction TR
                    JOIN Ticket TK ON TR.ticket_id = TK.ticket_id
                    JOIN Event_Has_Tickets EHT ON TK.ticket_id = EHT.ticket_id
                    WHERE EHT.event_id = E.event_id) AS sold_tickets
            FROM Event E
            JOIN Event_Has_Tickets EHT ON E.event_id = EHT.event_id
            JOIN Ticket T ON EHT.ticket_id = T.ticket_id
        `);

        connection.end();

        const eventsWithTicketsLeft = rows.map(event => ({
            ...event,
            tickets_left: event.total_tickets - event.sold_tickets
        }));

        res.json(eventsWithTicketsLeft);
    } catch (error) {
        console.error('Error fetching events:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});
```

The /api/events endpoint fetches all events and their ticket prices, so that it can be displayed on the main screen.

2)

```
// Get the history of events a user has tickets for
app.get('/api/user-event-history', async (req, res) => {
    try {
        if (!req.session.userId) {
            return res.json([]);
        }

        const userId = req.session.userId;
        const connection = await mysql.createConnection(dbConfig);
        const [rows] = await connection.query(`
            SELECT E.event_name, E.event_date, E.event_time, E.event_adress, E.event_category
            FROM Event E
            JOIN Event_Has_Tickets EHT ON E.event_id = EHT.event_id
            JOIN Ticket T ON EHT.ticket_id = T.ticket_id
            WHERE T.cart_id = (
                SELECT U.cart_id
                FROM User U
                WHERE U.user_id = ?
            )
        `, [userId]);
        connection.end();
        res.json(rows);
    } catch (error) {
        console.error('Error fetching user event history:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});
```

The /api/user-event-history/:userId endpoint fetches a user's past event tickets by their user ID. On receiving a GET request, the system queries the database for event details such as name, date, time, address, and category. This allows users to view their event history through their dashboard. If there's a query error, the server logs the error and returns an 'Internal Server Error' message to the user.

3)

```
// Calculates the total revenue generated from ticket sales for all events
app.get('/api/all-event-total-revenue', async (req, res) => {
    try {
        const connection = await mysql.createConnection(dbConfig);
        const [rows] = await connection.query(`
            SELECT E.event_id, E.event_name, SUM(T.ticket_price) AS Total_Revenue
            FROM Event E
            JOIN Event_Has_Tickets EHT ON E.event_id = EHT.event_id
            JOIN Ticket T ON EHT.ticket_id = T.ticket_id
            GROUP BY E.event_id, E.event_name
        `);
        connection.end();
        res.json(rows);
    } catch (error) {
        console.error('Error fetching all events total revenue:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});
```

The /api/event-total-revenue/:eventId endpoint allows for the calculation of total revenue from tickets sold for a specific event. When a GET request is made with an event ID, the server queries the database, summing up the prices of all tickets linked to that event. The result returned includes the event name and its total revenue. If there's an error during this process, the server records the error and notifies the user with an error message.

4)

```
// Gets events that have at least one ticket priced above the average price of all tickets
app.get('/api/events-above-average-price', async (req, res) => {
    try {
        const connection = await mysql.createConnection(dbConfig);
        const [rows] = await connection.query(`
            SELECT E.event_id, E.event_name, E.event_date, E.event_time, E.event_adress, E.event_category
            FROM Event E
            WHERE EXISTS (
                SELECT 1
                FROM Event_Has_Tickets EHT
                JOIN Ticket T ON EHT.ticket_id = T.ticket_id
                WHERE EHT.event_id = E.event_id AND T.ticket_price > (
                    SELECT AVG(ticket_price)
                    FROM Ticket
                )
            )
        `);
        connection.end();
        res.json(rows);
    } catch (error) {
        console.error('Error fetching events:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});
```

The /api/events-above-average-price endpoint retrieves a list of events where at least one ticket's price is higher than the average ticket price across all events. Upon a GET request, it executes a database query to find these events. If successful, the server responds with the event IDs and names. In case of a query failure, it logs the error and issues an 'Internal Server Error' to the user.

5)

```
// Get events that have no tickets sold
app.get('/api/events-with-no-tickets-sold', async (req, res) => {
    try {
        const connection = await mysql.createConnection(dbConfig);
        const [rows] = await connection.query(`
            SELECT E.event_id, E.event_name, E.event_date, E.event_time, E.event_adress, E.event_category
            FROM Event E
            WHERE NOT EXISTS (
                SELECT 1
                FROM Event_Has_Tickets EHT
                JOIN Ticket T ON EHT.ticket_id = T.ticket_id
                JOIN Transaction TR ON T.ticket_id = TR.ticket_id
                WHERE EHT.event_id = E.event_id
            )
        `);
        connection.end();
        res.json(rows);
    } catch (error) {
        console.error('Error fetching events with no tickets sold:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});
```

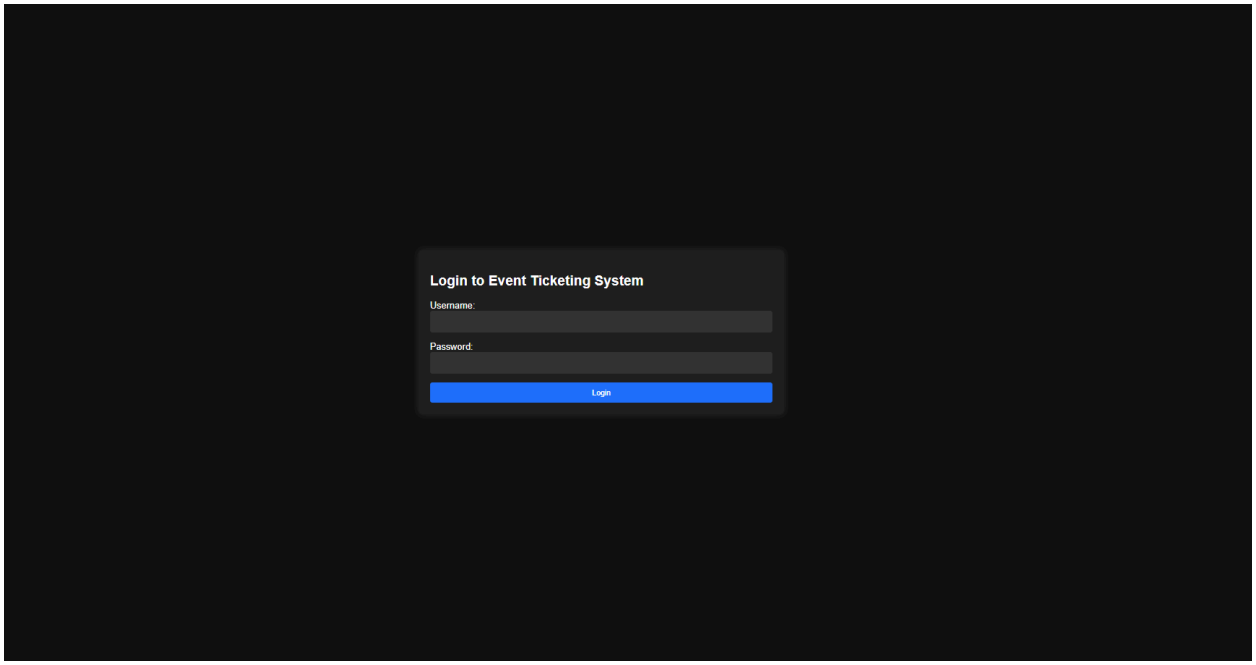
The endpoint /api/events-with-no-tickets-sold is set up to find events for which no tickets have been sold yet. When a request is made to this endpoint, the server checks the database to identify such events by looking for tickets that have not been linked to any transactions. It returns the IDs and names of these events. If an error occurs while running this query, the server logs the error and informs the user with an 'Internal Server Error' message.

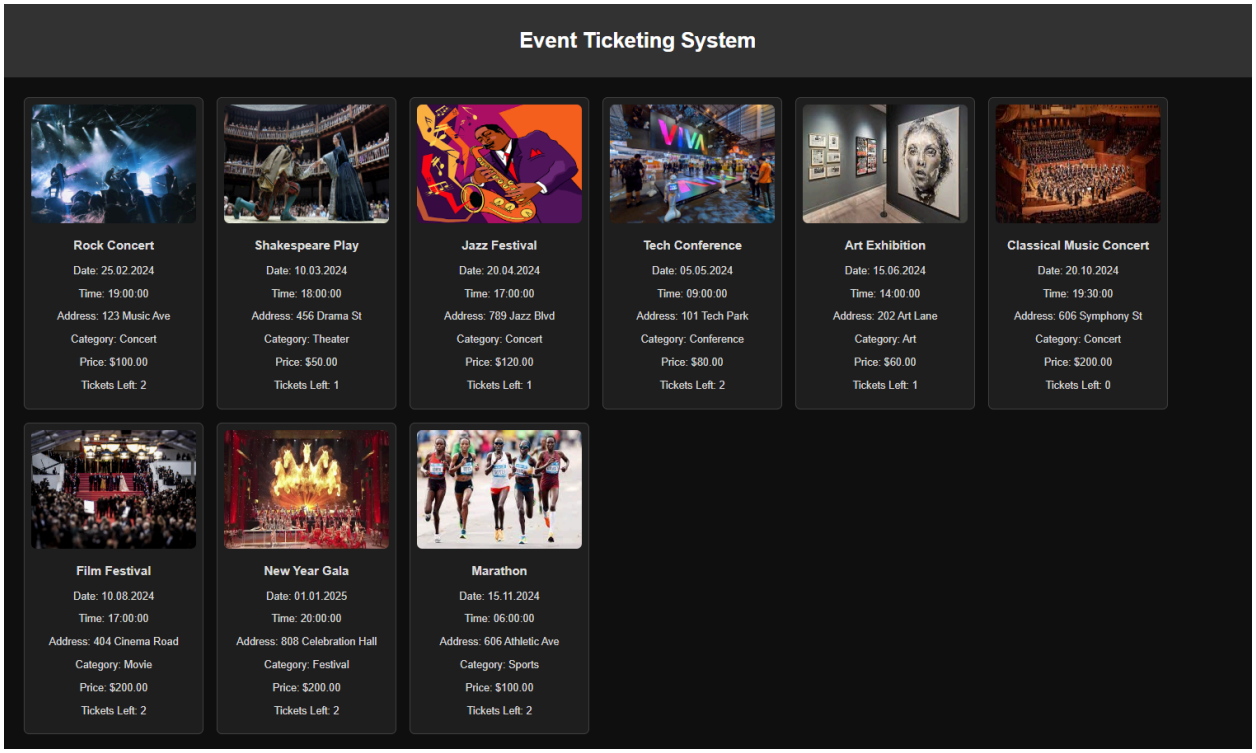
6)

```
// Get users who have bought tickets for 'Concert' category events
app.get('/api/users-who-bought-concert-tickets', async (req, res) => {
    try {
        const connection = await mysql.createConnection(dbConfig);
        const [rows] = await connection.query(`
            SELECT DISTINCT U.user_id, U.user_name
            FROM User U
            JOIN Transaction T ON U.user_id = T.user_id
            WHERE T.transaction_id IN (
                SELECT T.transaction_id
                FROM Ticket TK
                JOIN Event_Has_Tickets EHT ON TK.ticket_id = EHT.ticket_id
                JOIN Event E ON EHT.event_id = E.event_id
                WHERE E.event_category = 'Concert'
            )
        `);
        connection.end();
        res.json(rows);
    } catch (error) {
        console.error('Error fetching users who bought concert tickets:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
});
```

The endpoint /api/users-who-bought-concert-tickets fetches a list of users who have purchased tickets for events in the 'Concert' category. It runs a database query that looks for users with transactions linked to concert tickets. The response includes unique user IDs and names. If there's a problem with the query, the error is logged, and the user receives an 'Internal Server Error' notification.

6. Screenshots





User Event History

Rock Concert
Date: 25.02.2024
Time: 19:00:00
Address: 123 Music Ave
Category: Concert

Tech Conference
Date: 05.05.2024
Time: 09:00:00
Address: 101 Tech Park
Category: Conference

Event Total Revenue

Event: Rock Concert
Total Revenue: $300.00

Event: Shakespeare Play
Total Revenue: $100.00

Event: Jazz Festival
Total Revenue: $240.00

Event: Tech Conference
Total Revenue: $240.00

Event: Art Exhibition
Total Revenue: $120.00

Event: Classical Music Concert
Total Revenue: $200.00

Event: Film Festival
Total Revenue: $400.00

Event: New Year Gala
Total Revenue: $400.00

Event: Marathon
Total Revenue: $200.00

Events Above Average Price

Jazz Festival
Date: 20.04.2024
Time: 17:00:00
Address: 789 Jazz Blvd
Category: Concert

Classical Music Concert
Date: 20.10.2024
Time: 19:30:00
Address: 606 Symphony St
Category: Concert

Film Festival
Date: 10.08.2024
Time: 17:00:00
Address: 404 Cinema Road
Category: Movie

New Year Gala
Date: 01.01.2025
Time: 20:00:00
Address: 808 Celebration Hall
Category: Festival

Events With No Tickets Sold

Film Festival
Date: 10.08.2024
Time: 17:00:00
Address: 404 Cinema Road
Category: Movie

New Year Gala
Date: 01.01.2025
Time: 20:00:00
Address: 808 Celebration Hall
Category: Festival

Marathon
Date: 15.11.2024
Time: 06:00:00
Address: 606 Athletic Ave
Category: Sports

Users Who Bought Concert Tickets

JohnDoe

JaneSmith

AliceJohnson

MikeBrown

EmmaWilson

RobertMiller