Bilkent University

Department of Computer Engineering

CS353 Term Project

Project name: Eventica

Project Design Report

Group Members:

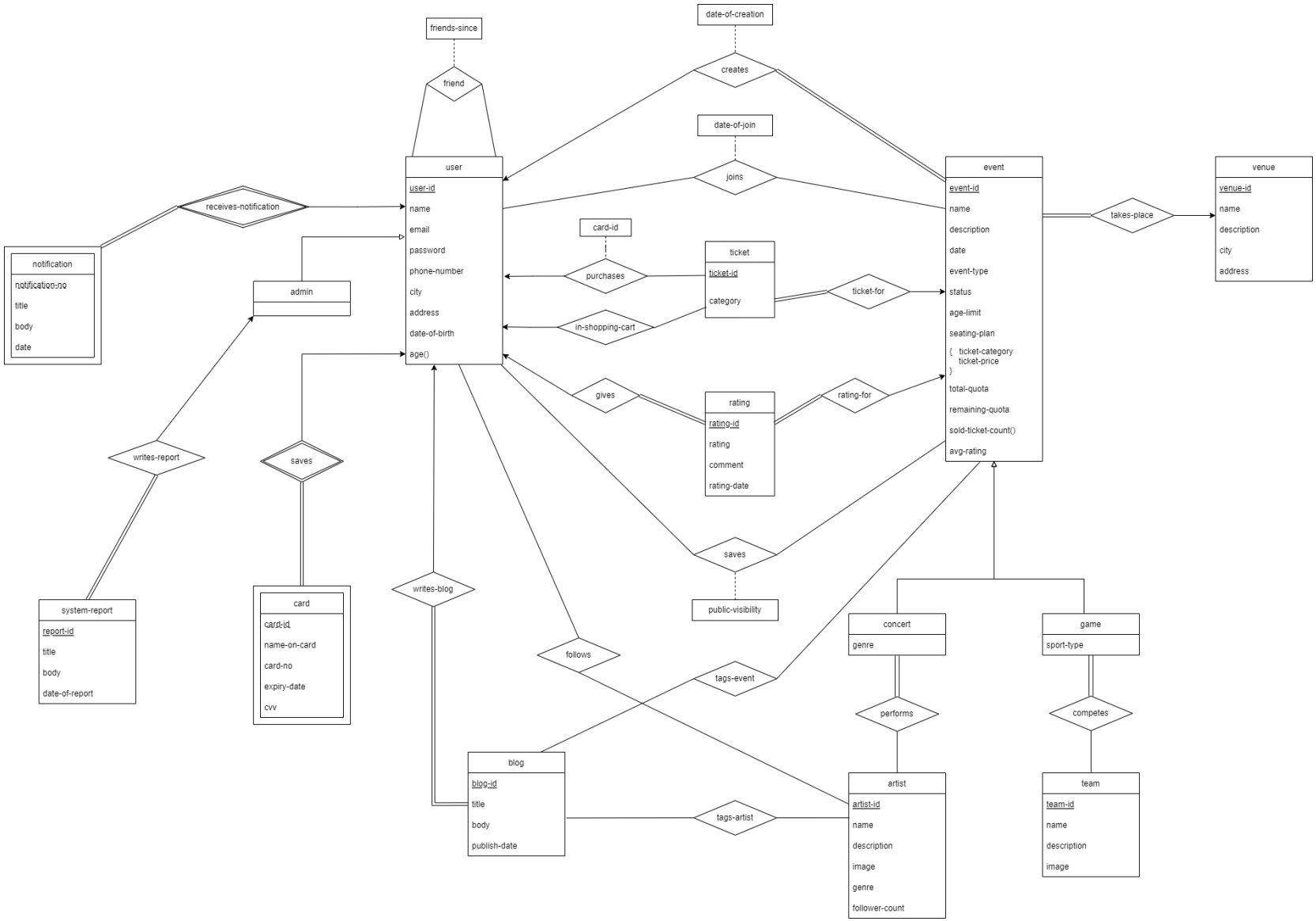
- Efe Beydoğan (21901548)
- Emir Melih Erdem (21903100)
- Eren Polat (21902615)
- Mehmet Berk Türkçapar (21902570)

Instructor: Özgür Ulusoy

TA: Zülal Bingöl

1. Design of the Database

1.1 Revised E/R Diagram



1.2 Table Schemas

User

Relational Model

user(<u>user_id,</u> name, email, password, phone_number, city, address, date_of_birth)

Functional Dependencies

user_id -> name, email, password, phone_number, city, address, date_of_birth
email -> name, email, password, phone_number, city, address, date_of_birth
phone_number -> name, email, password, phone_number, city, address, date_of_birth

Candidate Keys

{user_id}
{email}
{phone_number}

Normal Form

The table is in BCNF and therefore 3NF. All functional dependencies involve a superkey in the left-hand side (user_id, email, phone_number).

Table Definition

```
create table user(
    user_id int not null auto_increment,
    name varchar(50) not null,
    password varchar(40) not null,
    email varchar(40) not null,
    address varchar(100),
    city varchar(30),
    phone_number varchar(15),
    date_of_birth date,
    PRIMARY KEY (user_id));
```

Friend

Relational Model

friend(<u>user_id1, user_id2,</u> friends_since)

user_id1: FK to user(user_id), user_id2: FK to user(user_id)

Functional Dependencies

user_id1, user_id2 -> friends_since

Candidate Keys

{user_id1, user_id2}

Normal Form

The table is in BCNF and therefore 3NF. User_id1 and user_id2 together are a superkey.

Table Definition

```
create table friend(
        user_id1 int not null,
        user_id2 int not null,
        friends_since DATE not null,
        FOREIGN KEY user_id1 REFERENCES user(user_id)
                ON DELETE CASCADE,
        FOREIGN KEY user_id2 REFERENCES user(user_id)
                ON DELETE CASCADE,
        PRIMARY KEY  (user_id1, user_id2)
);
```

Notification

Relational Model

notification(<u>user_id, notification_no,</u> title, body, date)

user_id : FK to user

Functional Dependencies

user_id, notification_no -> title, body, date

Candidate Keys

{user_id, notification_no}

Normal Form

The table is in BCNF and therefore 3NF. User_id and notification_no together are a superkey.

Table Definition

```
create table notification(
    user_id int not null auto_increment,
    notification_no int not null,
    title varchar(100) not null,
    body text(300) not null,
    date DATE not null
    FOREIGN KEY user_id REFERENCES user(user_id)
        ON DELETE CASCADE,
    PRIMARY KEY (user_id, notification_no)
);
```

Admin

Relational Model

admin(<u>user_id</u>)

User_id: FK to user

Functional Dependencies

user_id -> user_id (trivial)

Candidate Keys

{user_id}

Normal Form

The table is in BCNF and therefore 3NF. User_id is a superkey.

Table Definition

```
create table admin(
    user_id int not null,
    FOREIGN KEY user_id REFERENCES user(user_id)
        ON DELETE CASCADE,
    PRIMARY KEY (user_id)
);
```

System Report

Relational Model

system_report(report_id, user_id, title, body, date_of_report)

user_id: FK to user

Functional Dependencies

report_id -> user_id, title, body, date_of_report

Candidate Keys

{report_id}

Normal Form

The table is in BCNF and therefore 3NF. Report_id is a superkey.

Table Definition

```
create table system_report(
    report_id int not null auto_increment,
    user_id int not null,
    title varchar(100) not null,
    body MEDIUMTEXT not null,
    date_of_report DATE not null,
    FOREIGN KEY user_id REFERENCES user(user_id)
        ON DELETE CASCADE,
    PRIMARY KEY (user_id)
);
```

Card

Relational Model

card(<u>user_id, card_id,</u> name_on_card, card_no, expiry_date, cvv)

user_id: FK to user

Functional Dependencies

user_id, card_id -> name_on_card, card_no, expiry_date, cvv
card_no, expiry_date, name_on_card -> user_id, card_id

Candidate Keys

{user_id, card_id}

Normal Form

The table is in BCNF and therefore 3NF. User_id and card_id together is a superkey, card_no and expiry_date and name_on_card together is a superkey.

Table Definition

```
create table card(
    user_id int not null,
    card_id int not null auto_increment,
    name_on_card varchar(50) not null,
    card_no int not null,
    expiry_date varchar(10) not null,
    cvv int not null,
    FOREIGN KEY user_id REFERENCES user(user_id)
        ON DELETE CASCADE,
    PRIMARY KEY (user_id, card_id)
);
```

Blog

Relational Model

blog(blog_id, user_id, title, body, publish_date)

user_id: FK to user

Functional Dependencies

blog_id, user_id -> title, body, publish_date

Candidate Keys

{blog_id, user_id}

Normal Form

The table is in BCNF and therefore 3NF. Blog_id and user_id together is a superkey.

Table Definition

```
create table blog(
    blog_id int not null auto_increment,
    user_id int not null,
    title varchar(100) not null,
    body MEDIUMTEXT not null,
    publish_date DATE not null,
    FOREIGN KEY user_id REFERENCES user(user_id)
        ON DELETE CASCADE,
    PRIMARY KEY (blog_id, user_id)
);
```

Ticket

Relational Model

ticket(<u>ticket_id,</u> event_id, ticket_category)

event_id: FK to event

Functional Dependencies

ticket_id -> event_id, ticket_category

Candidate Keys

{ticket_id}

Normal Form

The table is in BCNF and therefore 3NF. Ticket_id is a superkey.

Table Definition

```
create table ticket(
    ticket_id int not null auto_increment,
    event_id int not null,
    ticket_category varchar(50) not null,
    FOREIGN KEY event_id REFERENCES event(event_id),
    PRIMARY KEY (ticket_id)
);
```

Purchases

Relational Model

purchases(ticket_id, user_id, card_id)

ticket_id: FK to ticket

user_id: FK to card

card_id: FK to card

Functional Dependencies

ticket_id -> user_id, card_id

Candidate Keys

{ticket_id}

Normal Form

The table is in BCNF and therefore 3NF. Ticket_id is a superkey.

Table Definition

```
create table purchases(
    ticket_id int not null auto_increment,
    user_id int not null,
    card_id int not null,
    FOREIGN KEY user_id REFERENCES card(user_id)
        ON DELETE CASCADE,
    FOREIGN KEY card_id REFERENCES card(card_id)
        ON DELETE CASCADE,
    FOREIGN KEY ticket_id REFERENCES ticket(ticket_id)
        ON DELETE CASCADE,
    PRIMARY KEY (ticket_id)
);
```

In Shopping Cart

Relational Model

in_shopping_cart(ticket_id, user_id)

ticket_id: FK to ticket

user_id: FK to user

Functional Dependencies

ticket_id -> user_id

Candidate Keys

{ticket_id}

Normal Form

The table is in BCNF and therefore 3NF. Ticket_id is a superkey.

Table Definition

```
create table in_shopping_cart(
    ticket_id int not null auto_increment,
    user_id int not null,
    FOREIGN KEY user_id REFERENCES user(user_id)
        ON DELETE CASCADE,
    FOREIGN KEY ticket_id REFERENCES ticket(ticket_id)
        ON DELETE CASCADE,
    PRIMARY KEY (ticket_id)
);
```

Rating

Relational Model

rating(rating_id, rating, comment, rating_date, event_id, user_id)

user_id: FK to user

event_id: FK to event

Functional Dependencies

rating_id -> rating, comment, rating_date, event_id, user_id

Candidate Keys

{rating_id}

Normal Form

The table is in BCNF and therefore 3NF. Rating_id is a superkey.

Table Definition

```
create table rating(
        rating_id int not null auto_increment,
        user_id int not null,
        rating int not null,
        comment text(500),
        rating_date DATE not null,
        event_id int not null,
        FOREIGN KEY user_id REFERENCES user(user_id)
                ON DELETE CASCADE,
        FOREIGN KEY event_id REFERENCES event(event_id)
                ON DELETE CASCADE,
        PRIMARY KEY (rating_id)
);
```

Event

Relational Model

event(event_id, name, description, date, event_type, status, age_limit, total_quota, remaining_quota, seating_plan, venue_id, creator_id, date_of_creation, avg_rating)

venue_id: FK to venue, creator_id: FK to user(user_id)

Functional Dependencies

event_id -> name, description, date, event_type, status, age_limit, total_quota, remaining_quota, seating_plan, venue_id, creator_id, date_of_creation, sold_ticket_count, avg_rating

Candidate Keys

{event_id}

Normal Form

The table is in BCNF and therefore 3NF. Event_id is a superkey.

Table Definition

```
create table event(
    event_id int not null auto_increment,
    name varchar(50) not null,
    description MEDIUMTEXT not null,
    date DATE not null,
    event_type varchar(20) not null,
    status varchar(20),
    age_limit int,
    total_quota int not null,
    remaining_quota int not null,
    seating_plan varchar(100),
    venue_id int not null,
    creator_id int not null,
    date_of_creation DATE not null,
    avg_rating int,
    CHECK (event_type in ('Concert', 'Sports', 'Gathering', 'Art',
        'Other')),
```

```
        CHECK (total_quota = remaining_quota),
        CHECK (remaining_quota >= 0),
        FOREIGN KEY creator_id REFERENCES user(user_id)
            ON DELETE CASCADE,
        FOREIGN KEY venue_id REFERENCES venue(venue_id)
            ON DELETE CASCADE,
        PRIMARY KEY (event_id)
);
```

Event Category

Relational Model

event_category(event_id, ticket_category, ticket_price)

event_id: FK to event

Functional Dependencies

event_id, ticket_category -> ticket_price

Candidate Keys

{event_id, ticket_category}

Normal Form

The table is in BCNF and therefore 3NF. Event_id, ticket_category together is a superkey

Table Definition

```
create table event_category(
    event_id int not null,
    ticket_category VARCHAR(30),
    ticket_price FLOAT(20),
    FOREIGN KEY event_id REFERENCES event(event_id)
        ON DELETE CASCADE,
    PRIMARY KEY (event_id, ticket_category)
);
```

Joins

Relational Model

joins(event_id, user_id, date_of_join)

event_id: FK to event

user_id: FK to user

Functional Dependencies

event_id, user_id -> date_of_join

Candidate Keys

{event_id, user_id}

Normal Form

The table is in BCNF and therefore 3NF. Event_id, user_id together is a superkey

Table Definition

```
create table joins(
    event_id int not null,
    user_id int not null,
    date_of_join DATE not null,
    FOREIGN KEY event_id REFERENCES event(event_id)
        ON DELETE CASCADE,
    FOREIGN KEY user_id REFERENCES user(user_id)
        ON DELETE CASCADE,
    PRIMARY KEY (event_id, user_id)
);
```

Saves

Relational Model

saves(<u>user_id, event_id,</u> public_visibility)

user_id. FK to user

event_id: FK to event

Functional Dependencies

user_id, event_id -> public_visibility

Candidate Keys

{user_id, event_id}

Normal Form

The table is in BCNF and therefore 3NF. Event_id, user_id together is a superkey

Table Definition

```
create table saves(
    user_id int not null,
    event_id int not null,
    public_visibility BOOL not null,
    FOREIGN KEY user_id REFERENCES user(user_id)
        ON DELETE CASCADE,
    FOREIGN KEY event_id REFERENCES event(event_id)
        ON DELETE CASCADE,
    PRIMARY KEY (user_id, event_id)
);
```

Venue

Relational Model

venue(venue_id, name, description, city, address)

Functional Dependencies

venue_id -> name, description, city, address

Candidate Keys

{venue_id}

Normal Form

The table is in BCNF and therefore 3NF. Venue_id is a superkey

Table Definition

```
create table venue(
    venue_id int not null auto_increment,
    name varchar(100) not null,
    description MEDIUMTEXT not null,
    city varchar(30) not null,
    address varchar(100) not null,
    PRIMARY KEY (venue_id)
);
```

Concert

Relational Model

concert(event_id, genre)

event_id: FK to event

Functional Dependencies

event_id -> genre

Candidate Keys

{event_id}

Normal Form

The table is in BCNF and therefore 3NF. Event_id is a superkey

Table Definition

```
create table concert(
        event_id int not null,
        genre VARCHAR(30),
        FOREIGN KEY event_id REFERENCES event(event_id)
                ON DELETE CASCADE,
        PRIMARY KEY (event_id)
);
```

Game

Relational Model

concert(event_id, sport_type)

event_id: FK to event

Functional Dependencies

event_id -> sport_type

Candidate Keys

{event_id}

Normal Form

The table is in BCNF and therefore 3NF. Event_id is a superkey

Table Definition

```
create table game(
        event_id int not null,
        sport_type VARCHAR(30),
        FOREIGN KEY event_id REFERENCES event(event_id)
                ON DELETE CASCADE,
        PRIMARY KEY (event_id)
);
```

Performs

Relational Model

performs(artist_id, event_id)

artist_id: FK to artist

event_id: FK to concert

Functional Dependencies

artist_id, event_id -> artist_id, event_id

Candidate Keys

{artist_id, event_id}

Normal Form

The table is in BCNF and therefore 3NF. Artist_id and event_id together is a superkey

Table Definition

```
create table performs(
    event_id int not null,
    artist_id int not null,
    FOREIGN KEY event_id REFERENCES event(event_id)
        ON DELETE CASCADE,
    FOREIGN KEY artist_id REFERENCES artist
        ON DELETE CASCADE,
    PRIMARY KEY (artist_id, event_id)
);
```

Artist

Relational Model

artist(artist_id, name, description, image, genre, follower_count)

Functional Dependencies

artist_id -> name, description, image, genre, follower_count

Candidate Keys

{artist_id}

Normal Form

The table is in BCNF and therefore 3NF. Artist_id is a superkey

Table Definition

```
create table artist(
    artist_id int not null auto_increment,
    name varchar(50) not null,
    description MEDIUMTEXT not null,
    image varchar(50),
    genre varchar(30) not null,
    follower_count int not null,
    PRIMARY KEY (artist_id)
);
```

Competes

Relational Model

competes(team_id, event_id)

team_id: FK to team

event_id: FK to event

Functional Dependencies

team_id, event_id -> team_id, event_id (trivial)

Candidate Keys

{team_id, event_id}

Normal Form

The table is in BCNF and therefore 3NF. Team_id and event_id together are a superkey.

Table Definition

```
create table competes(
    team_id int not null,
    event_id int not null,
    FOREIGN KEY team_id REFERENCES event(event_id)
        ON DELETE CASCADE,
    FOREIGN KEY event_id REFERENCES event(event_id)
        ON DELETE CASCADE,
    PRIMARY KEY (team_id, event_id)
);
```

Team

Relational Model

team(team_id, name, description, image)

Functional Dependencies

team_id -> name, description, image

Candidate Keys

{team_id}

Normal Form

The table is in BCNF and therefore 3NF. Team_id is a superkey

Table Definition

```
create table team(
    team_id int not null auto_increment,
    name varchar(50) not null,
    description MEDIUMTEXT not null,
    image varchar(50),
    PRIMARY KEY (team_id)
);
```

Blog

Relational Model

blog(blog_id, title, body, publish_date, user_id)

user_id: FK to user

Functional Dependencies

blog_id-> title, body, publish_date, user_id

Candidate Keys

{blog_id}

Normal Form

The table is in BCNF and therefore 3NF. Blog_id is a superkey

Table Definition

```
create table blog(
        blog_id int not null auto_increment,
        user_id int not null
        title varchar(50) not null,
        body MEDIUMTEXT not null,
        publish_date DATE,
        FOREIGN KEY user_id REFERENCES user
                ON DELETE CASCADE,
        PRIMARY KEY (blog_id)
);
```

Follows

Relational Model

follows(artist_id, user_id)

artist_id: FK to artist

user_id: FK to user

Functional Dependencies

artist_id, user_id -> artist_id, user_id (trivial)

Candidate Keys

{artist_id, user_id}

Normal Form

The table is in BCNF and therefore 3NF. Artist_id and user_id together are a superkey.

Table Definition

```
create table follows(
    artist_id int not null,
    user_id int not null,
    FOREIGN KEY artist_id REFERENCES artist
        ON DELETE CASCADE,
    FOREIGN KEY user_id REFERENCES user
        ON DELETE CASCADE,
    PRIMARY KEY (artist_id, user_id)
);
```

Tags Event

Relational Model

tags_event(event_id, blog_id)

event_id: FK to event

blog_id: FK to blog

Functional Dependencies

event_id, blog_id -> event_id, blog_id (trivial)

Candidate Keys

{event_id, blog_id}

Normal Form

The table is in BCNF and therefore 3NF. Event_id and blog_id together are a superkey.

Table Definition

```
create table tags_event(
    event_id int not null,
    blog_id int not null,
    FOREIGN KEY event_id REFERENCES event
        ON DELETE CASCADE,
    FOREIGN KEY blog_id REFERENCES blog
        ON DELETE CASCADE,
    PRIMARY KEY (event_id, blog_id)
);
```

Tags Artist

Relational Model

tags_artist(blog_id, artist_id)

blog_id: FK to artist

artist_id: FK to blog

Functional Dependencies

blog_id, artist_id -> blog_id, artist_id (trivial)

Candidate Keys

{blog_id, artist_id}

Normal Form

The table is in BCNF and therefore 3NF. Blog_id and artist_id together are a superkey.

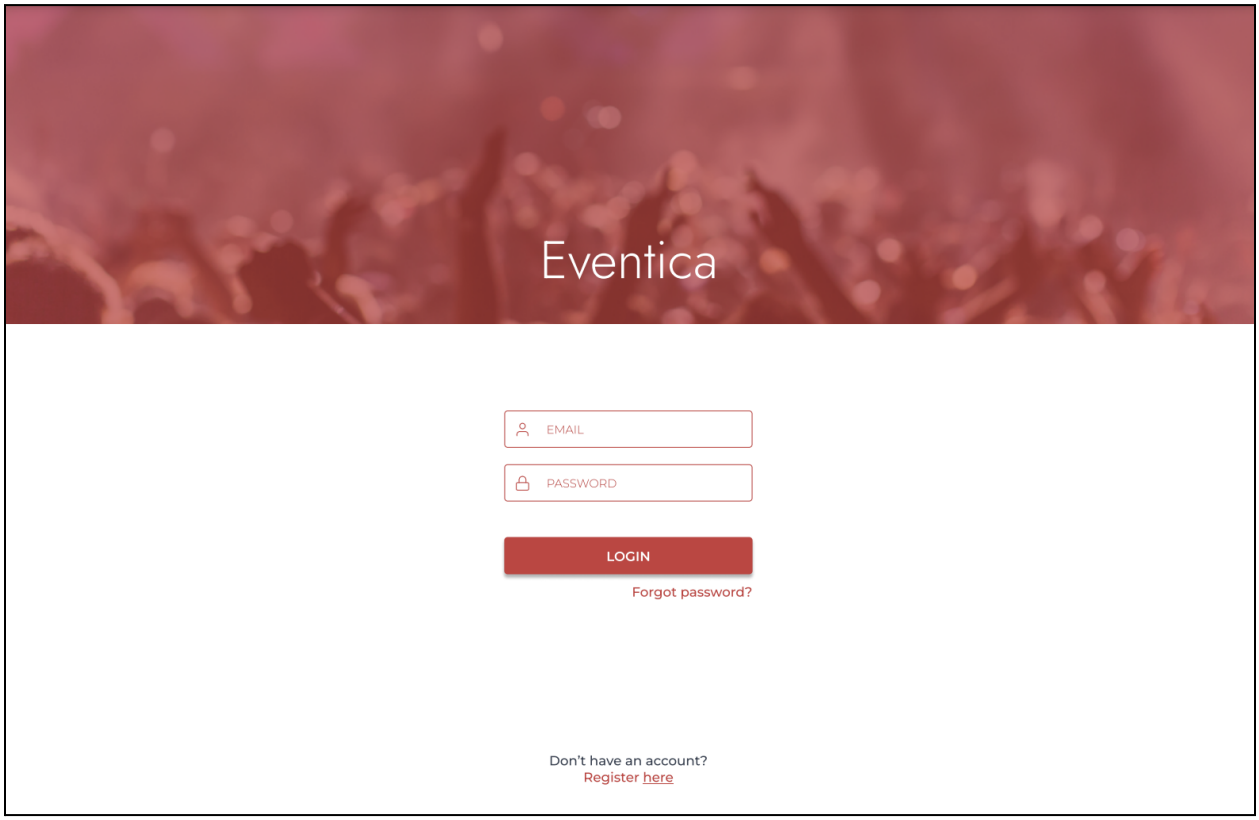
Table Definition

```
create table tags_artist(
        artist_id int not null,
        blog_id int not null,
        FOREIGN KEY artist_id REFERENCES artist
                ON DELETE CASCADE,
        FOREIGN KEY blog_id REFERENCES blog
                ON DELETE CASCADE,
        PRIMARY KEY (blog_id, artist_id)
);
```

2. User Interface Design and Related SQL Statements

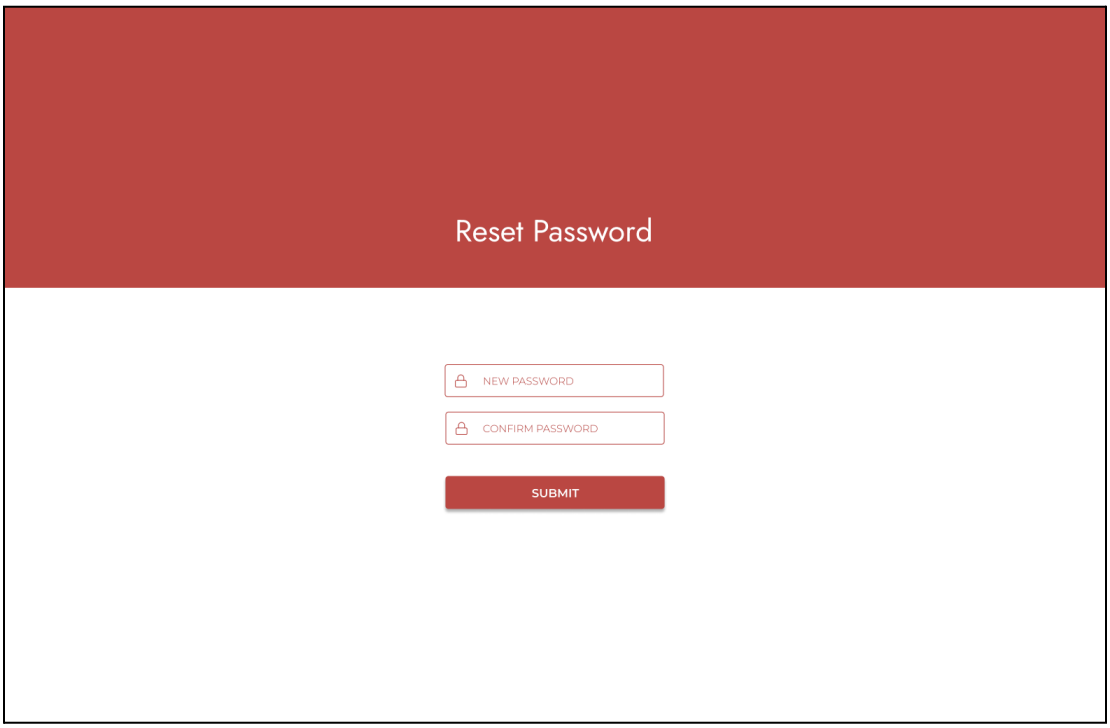
2.1 Common Functionalities

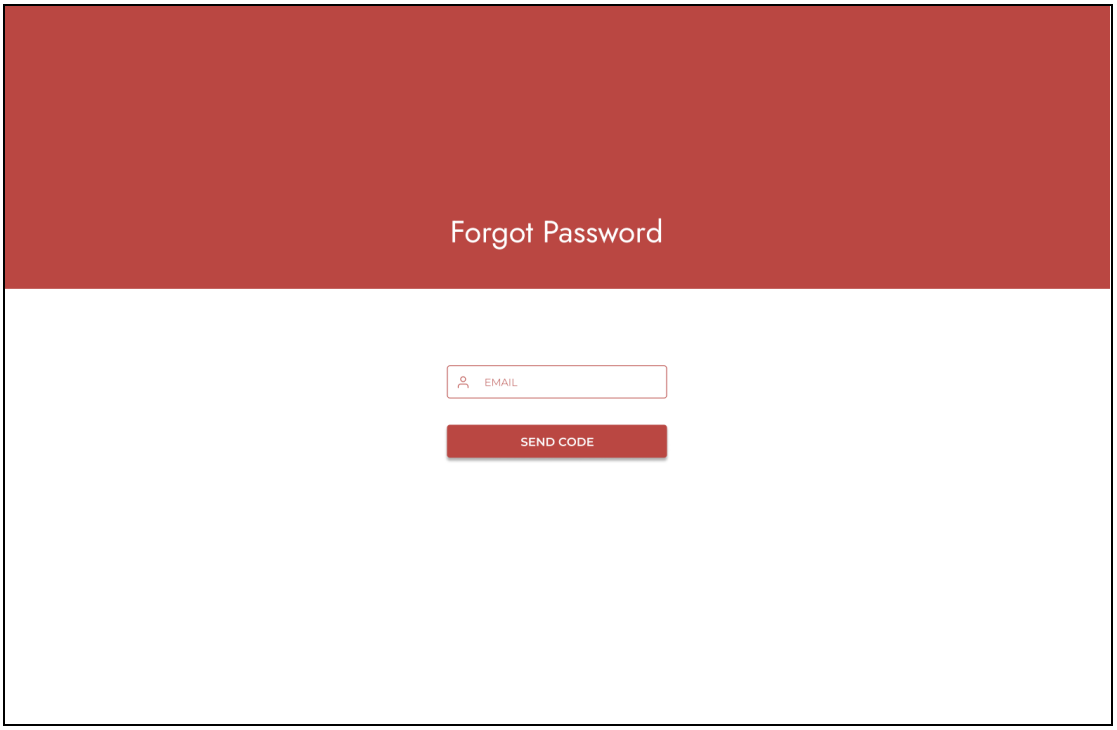
Login



SQL Query:
SELECT *
FROM user
WHERE email = @email AND password = @password;

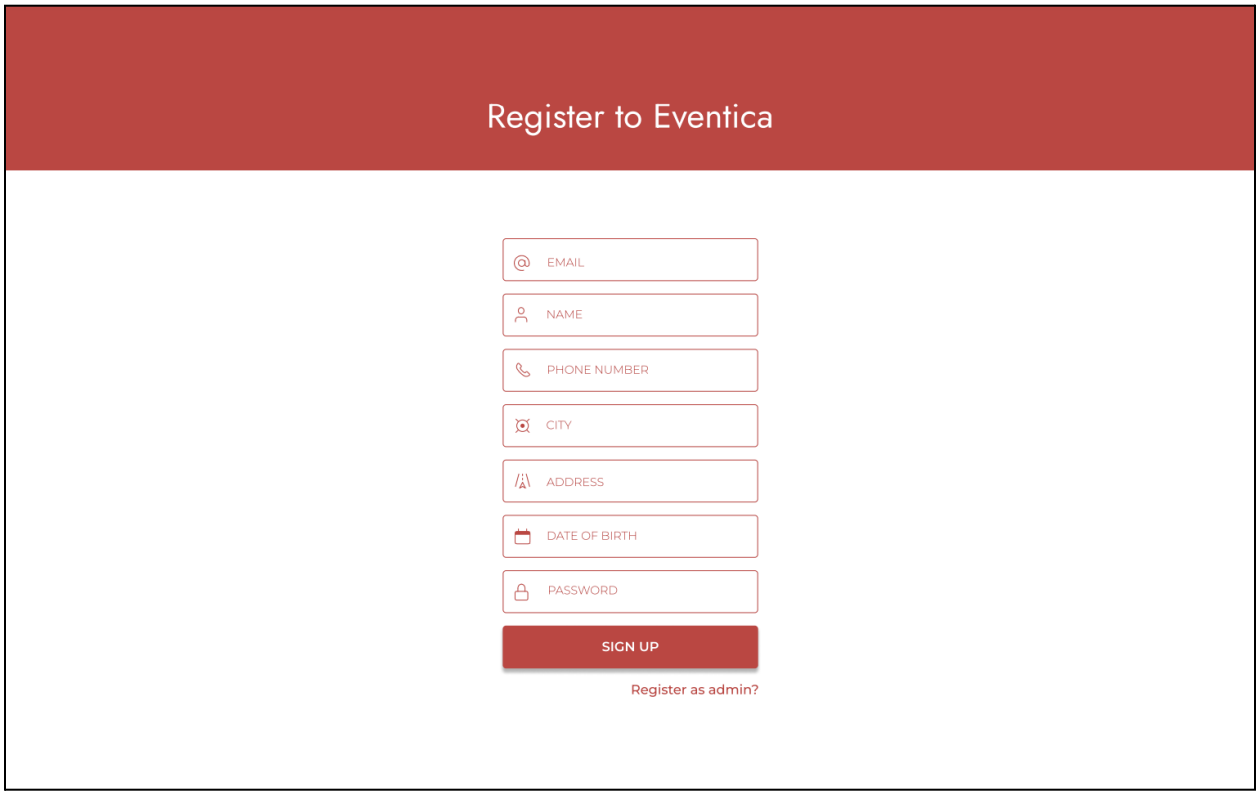
–Reset Password:






```
UPDATE user
SET user.password = @new_password
WHERE email = @email;
```

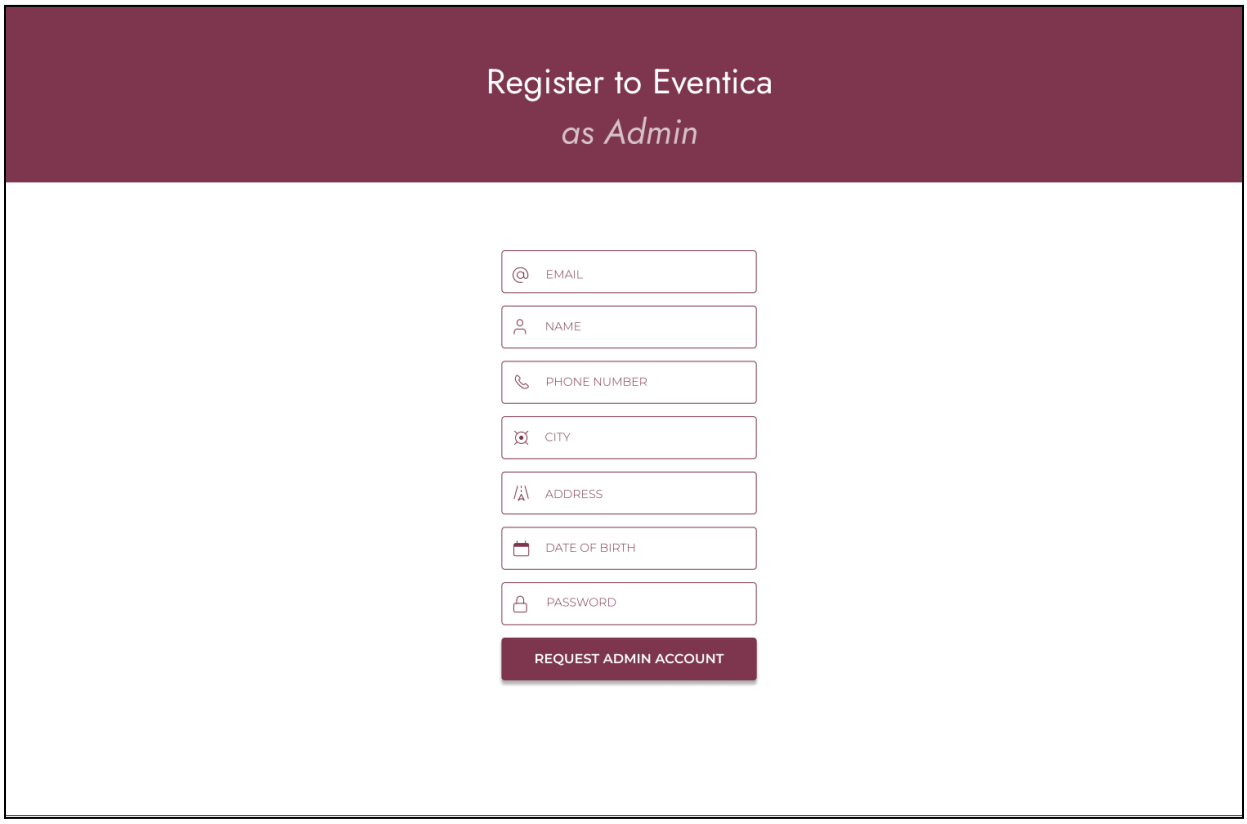
Sign-up for Users



SQL Query:

```
INSERT INTO user VALUES(
        0,
        @name,
        @email,
        @password,
        @phone_number,
        @city,
        @address,
        @date_of_birth
);
```

Sign-up for Admins



SQL Queries:

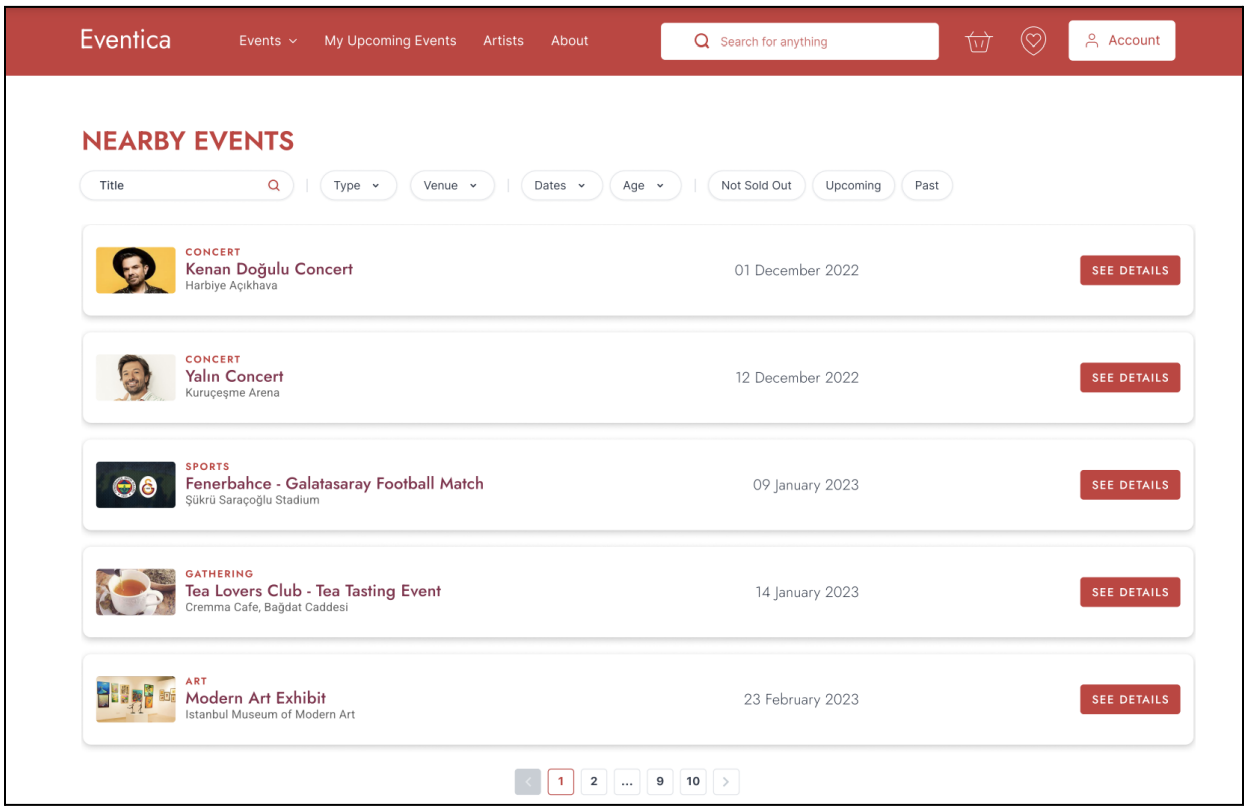
```
INSERT INTO user VALUES(
        0,
        @name,
        @email,
        @password,
        @phone_number,
        @city,
        @address,
        @date_of_birth
);
INSERT INTO admin VALUES(0);
```

2.2 Topic-Specific Functionality

Join Event

SQL Queries:

a. List all possible events by applying necessary filters



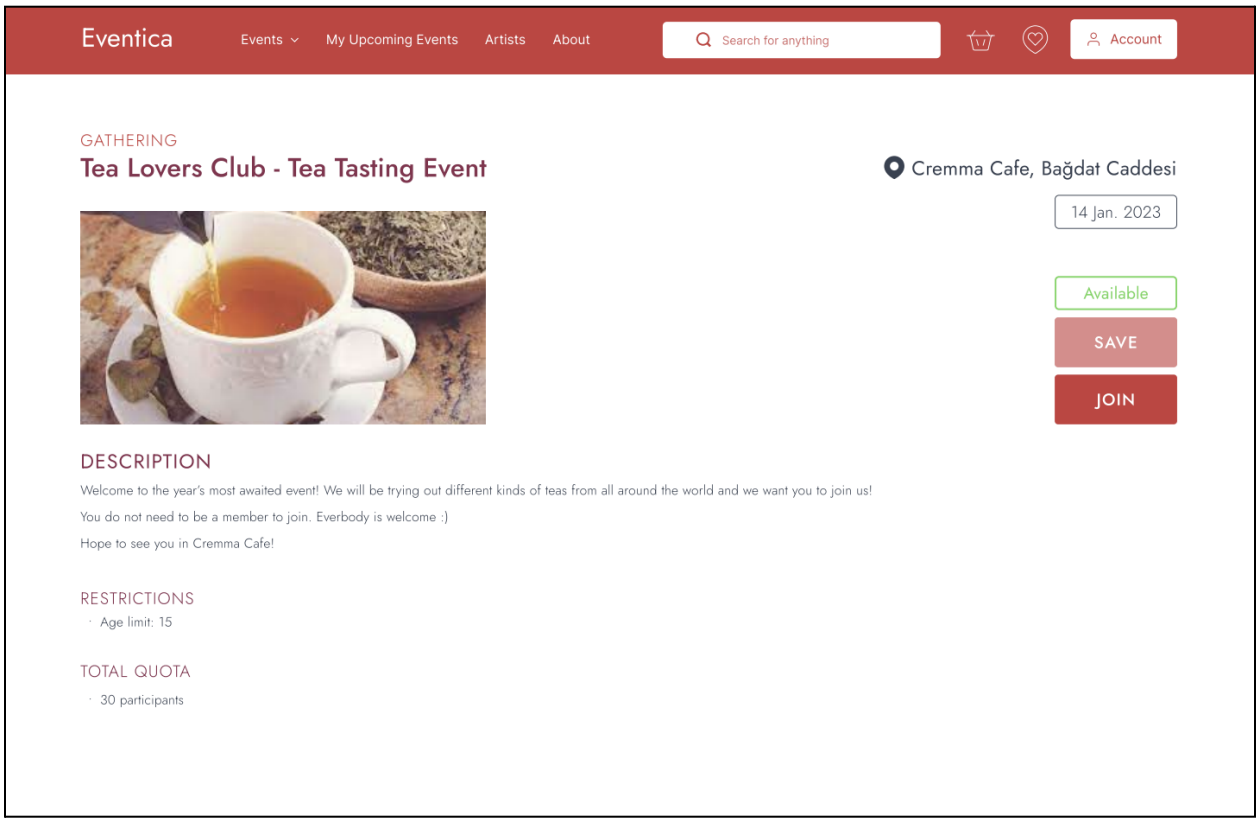
<u>–If selected event type is not ‘Concert’:</u>

```
SELECT event_id, E.name, date, event_type, V.name
FROM event E JOIN venue V USING (venue_id)
WHERE E.name LIKE ‘%@title%’ AND event_type=@type AND
        city=@my_city AND V.venue_id=@venue AND
        (date BETWEEN @start_date AND @end_date) AND
        age_limit <= @age AND
        (@not_sold_out = 0 OR remaining_quota > 0) AND
        (@upcoming = 0 OR date > @today) AND
        (@past = 0 OR date < @today);
```

–Else (it's 'Concert'):

```
SELECT DISTINCT event_id, E.name, date, event_type, V.name
FROM (event NATURAL JOIN concert NATURAL JOIN performs) E, venue V
WHERE E.venue_id=V.venue_id AND E.name LIKE '%@title%' AND
        event_type='Concert' AND
        city=@my_city AND V.venue_id=@venue AND
        artist_id=@artist AND genre=@genre AND
        (date BETWEEN @start_date AND @end_date) AND
        age_limit <= @age AND
        (@not_sold_out = 0 OR remaining_quota > 0) AND
        (@upcoming = 0 OR date > @today) AND
        (@past = 0 OR date < @today);
```

b. The user selects the event



The user view for events:

```
CREATE VIEW event_view(event_id, event_name, description, date,
                       event_type, status, age_limit, total_quota,
                       seating_plan, avg_rating, venue_id, event_city,
```

```
                                    event_adress) AS
        SELECT event_id, event.name, event.description, date, event_type,
                status, age_limit, total_quota, seating_plan, avg_rating,
                venue.venue_id, venue.city, venue.adress
        FROM event JOIN venue USING (venue_id);
```

<u>–When the detailed event page is opened for a specific event:</u>

```
SELECT *
FROM event_view
WHERE event_id=@event_id;
```

c. The event restrictions quota and age are checked by the system in the backend.

```
SELECT age_limit, remaining_quota
FROM event
WHERE event_id=@event_id;
```

d. The system checks if the event's date/time collides with any of the events on the user's events list.

<u>–User view for 'my upcoming events':</u>

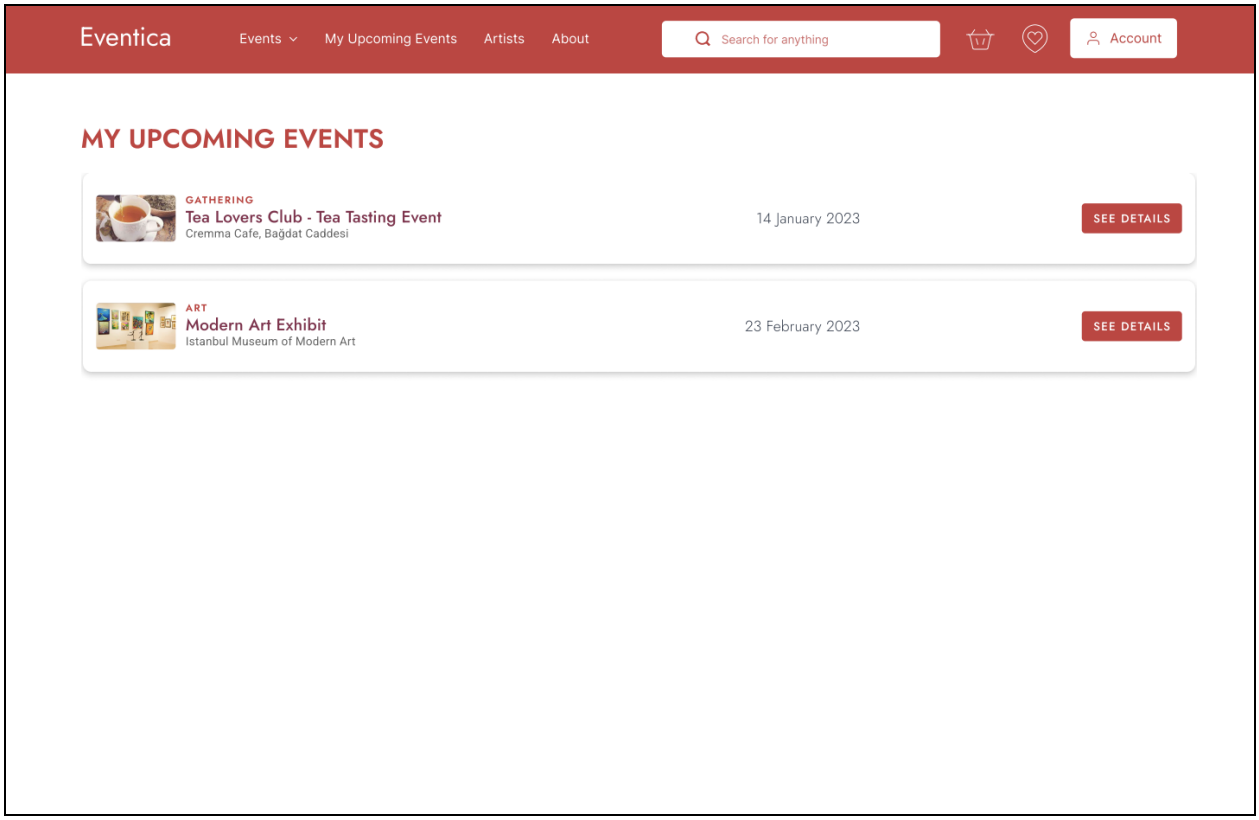
```
CREATE VIEW my_upcoming_events AS
        SELECT *
        FROM event_view
        WHERE (event_id, @user_id) IN (SELECT * FROM joins);
```

<u>–Two events in 'my upcoming events' cannot collide:</u>

```
CREATE ASSERTION dates_not_colliding CHECK (
        NOT EXISTS (SELECT *
                        FROM my_upcoming_events E1, my_upcoming_events E2
                        WHERE E1.event_id <> E2.event_id AND
                        E1.date = E2.date)
);
```

e. The event is added to the user's (upcoming) events list.

```
INSERT INTO joins VALUES (@event_id, @user_id);
```

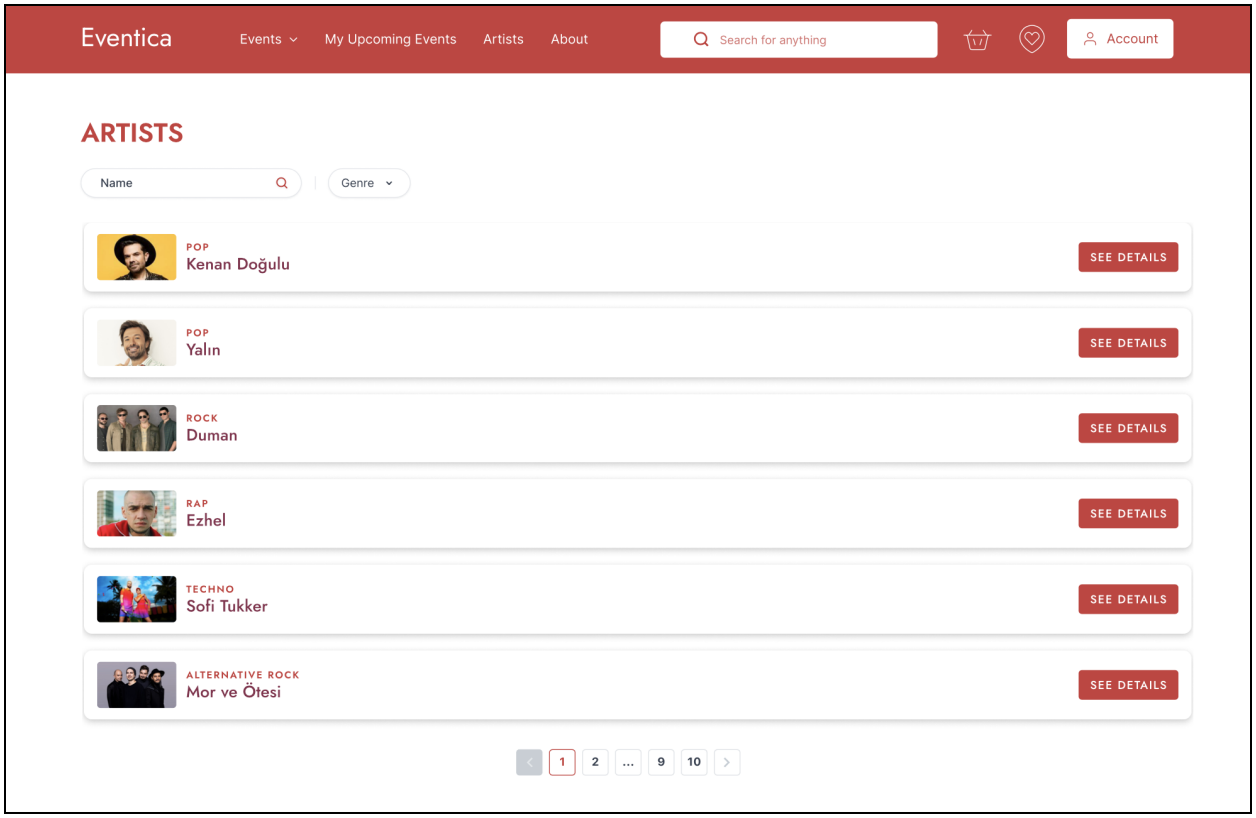


–When the My Upcoming Events page is opened

```
SELECT *
FROM my_upcoming_events
```

2.3 Additional Functionality

Artists

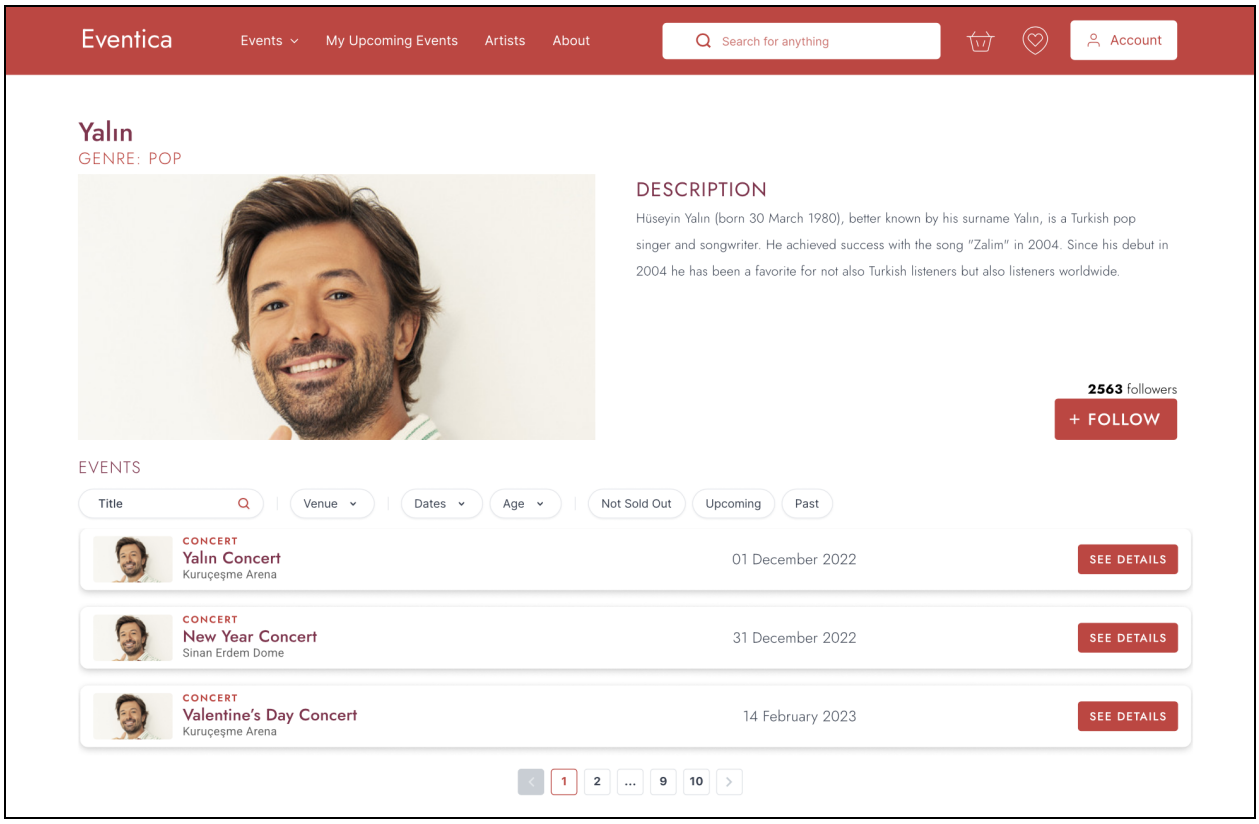


SQL Queries:

<u>–To get all artists' information:</u>

```
CREATE VIEW artist_view(artist_id, artist_name, artist_image, artist_genre)
AS ( SELECT artist_id, name, image, genre FROM artist);

SELECT *
FROM artist_view
WHERE artist_name LIKE '%@name% AND artist_genre = @genre;
```



<u>–To get selected artist's information:</u>

```
SELECT *
FROM artist,
WHERE artist_id = @selected_artist;
```

<u>–To get artist's events:</u>

```
SELECT e.name, e.date, e.event_type
FROM event e, performs p
WHERE p.artist_id = @selected_artist AND e.event_id = p.event_id AND
e.venue-id=@venue AND
        (e.date BETWEEN @start_date AND @end-date) AND
        e.age_limit <= @age AND name LIKE '%@title%'
        (@not_sold_out = 0 OR e.remaining_quota > 0) AND
        (@upcoming = 0 OR e.date > @today) AND
        (@past = 0 OR e.date < @today);
```

<u>–When user follows an artist:</u>

```
INSERT INTO follows VALUES (@selected_artist, @current_user);
```

3. Triggers

Update Remaining Quota After Insert on Joins

```
create trigger ticket_count_update
after insert on joins
referencing new row as nrow
begin atomic
	UPDATE event
	SET event.remaining_quota = event.remaining_quota - 1
	WHERE event.event_id = nrow.event_id
end;
```

Update Average Rating After New Rating Is Added

```
create trigger new_rating
after insert on rating
referencing new row as nrow
begin atomic
	UPDATE event
	SET event.avg_rating = (SELECT avg(rating)
				FROM rating
				WHERE event_id = nrow.event_id)
	WHERE event.event_id = nrow.event_id
end;
```

Update Average Rating After Rating Is Updated

```
create trigger rating_update
after update of rating on rating
referencing new row as nrow
begin atomic
	UPDATE event
	SET event.avg_rating = (SELECT avg(rating)
				FROM rating
				WHERE event_id = nrow.event_id)
```

```
        WHERE event.event_id = nrow.event_id
end;
```

Update Average Rating After Rating is Deleted

```
create trigger rating_delete
after delete of rating
referencing old row as orow
begin atomic
        UPDATE event
        SET event.avg_rating = (SELECT avg(rating)
                                FROM rating
                                WHERE event_id = orow.event_id)
        WHERE event.event_id = orow.event_id
end;
```

Update Follower Count After Insert on Follows

```
create trigger follower_count_update
after insert on follows
referencing new row as nrow
begin atomic
        UPDATE artist
        SET artist.follower_count = artist.follower_count + 1
        WHERE artist.artist_id = nrow.artist_id
end;
```

4. Implementation Plan

We will use the Django framework for implementing our project. We have chosen this framework specifically because it is versatile and fast, and it will also allow us to write raw SQL queries as required for this project. We will be writing the user interface in HTML and CSS, and merging frontend and backend via Django. We will utilize MySQL as the DBMS in our project, as it supports modern features like views, triggers, constraints, etc.