CS353 Database Systems

Project Design

Local Events Application

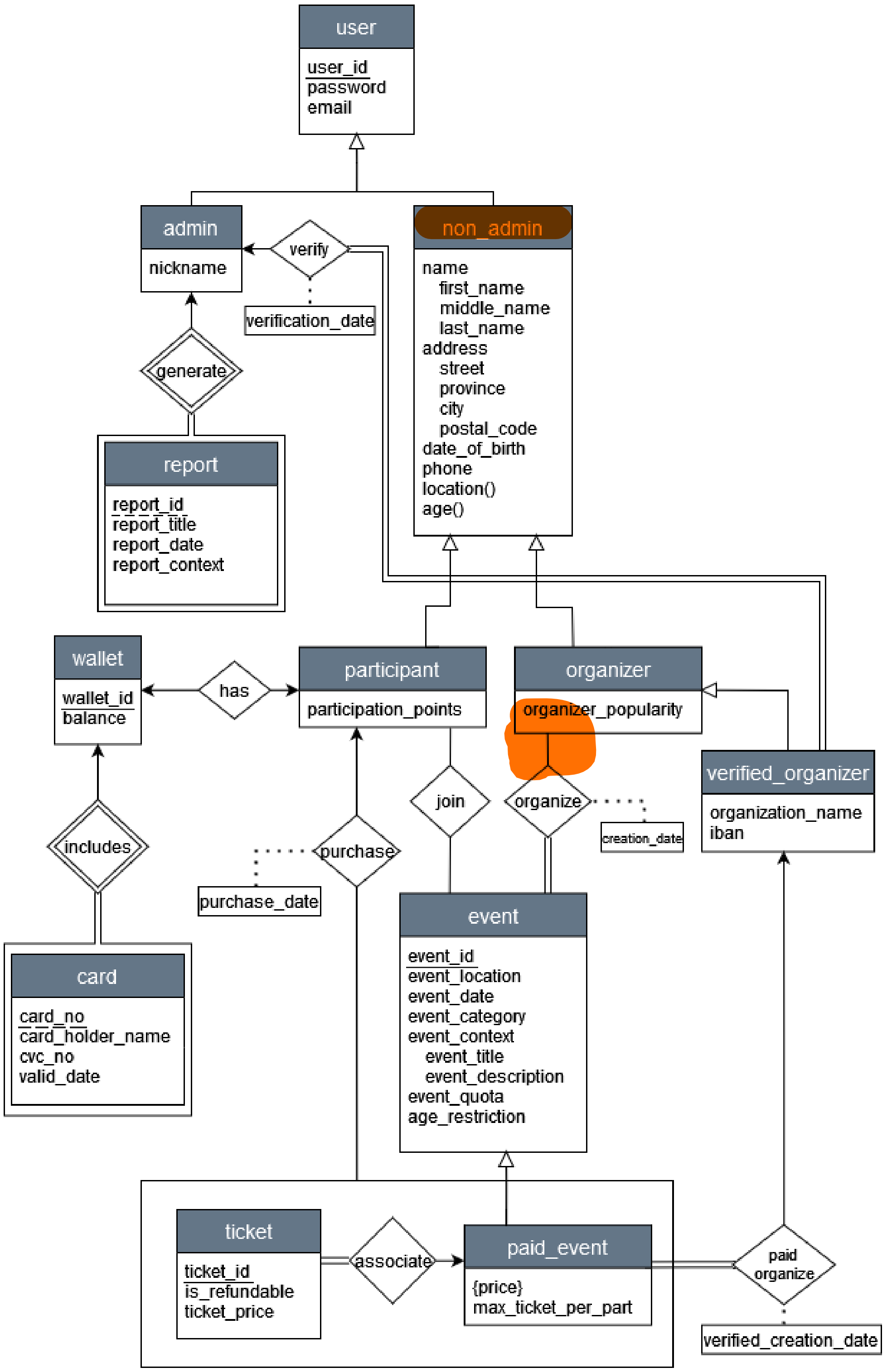
Group 1

Alper Mumcular - 21902740
Vesile İrem Aydın - 21902914
Ravan Aliyev - 21500405
Ece Kahraman - 21801879

Table of Contents

1. Design of the Database

1.1. Revised ER Diagram



1.2. Table Schemas

(Note: All SQL queries, triggers, reports, views have been tested on MySQL Workbench.)

1.2.1. User

Model: user(<u>user_id</u> , password , email)

Attribute Domains:

user_id → int

password → varchar(20)

email → varchar(45)

Functional Dependency: user_id → password, email

Candidate Keys: {user_id}, {email}

Foreign Key: none

Primary Key: {user_id}

Table Definition:

```
CREATE TABLE user (
  `user_id` INT NOT NULL AUTO_INCREMENT,
  `password` VARCHAR(20) NOT NULL,
  `email` VARCHAR(45) NOT NULL UNIQUE,
  PRIMARY KEY (`user_id`)
);
```

1.2.2. Admin

Model: admin(<u>user_id</u> , nickname)

Attribute Domains:

user_id → int

nickname → varchar(20)

Functional Dependency: user_id → nickname

Candidate Keys: {user_id}, {nickname}

Foreign Key: user_id references user(user_id)

Primary Key: {user_id}

Table Definition:

```
CREATE TABLE admin (
  `user_id` INT NOT NULL,
  `nickname` VARCHAR(20) NOT NULL UNIQUE,
  PRIMARY KEY (`user_id`),
  FOREIGN KEY (user_id) REFERENCES user(user_id)
);
```

1.2.3. Non_admin

Model: non_admin(user_id , first_name , middle_name , last_name , street , province , city , postal_code , date_of_birth , phone)

Attribute Domains:

user_id → int

first_name → varchar(20)

middle_name → varchar(20)

last_name → varchar(20)

street → varchar(20)

province → varchar(20)

city → varchar(20)

postal_code → int

date_of_birth → date

phone → varchar(11)

Candidate Keys: { user_id }, { phone }

Primary Key: { user_id }

Foreign Key: user_id references user(user_id)

Functional Dependency:

user_id → first_name, middle_name, last_name, street, province, city, postal_code, date_of_birth, phone

phone → user_id, first_name, middle_name, last_name, street, province, city, postal_code, date_of_birth

Table Definition:

```
CREATE TABLE non_admin (
  `user_id` INT NOT NULL,
  `first_name` VARCHAR(20) NOT NULL,
  `middle_name` VARCHAR(20),
  `last_name` VARCHAR(20) NOT NULL,
  `street` VARCHAR(20) NOT NULL,
  `province` VARCHAR(20) NOT NULL,
  `city` VARCHAR(20) NOT NULL,
  `postal_code` INT NOT NULL,
  `date_of_birth` DATE NOT NULL,
  `phone` VARCHAR(11) NOT NULL UNIQUE,
  PRIMARY KEY (`user_id`),
  FOREIGN KEY (user_id) REFERENCES user(user_id)
);
```

1.2.4. Participant

Model: participant(<u>user_id</u> , participation_points)

Attribute Domains:

user_id → int

participation_points → int

Candidate Keys: { user_id }

Foreign Key:

user_id references non_admin(user_id)

Primary Key: { user_id }

Functional Dependency: user_id → participation_points

Table Definition:

```
CREATE TABLE participant (
  `user_id` INT NOT NULL,
  `participation_points` INT DEFAULT 0,
  PRIMARY KEY (`user_id`),
  FOREIGN KEY (user_id) REFERENCES non_admin(user_id)
);
```

1.2.5. Organizer

Model: organizer(user_id , organizer_popularity)

Attribute Domains:

user_id → int

organizer_popularity → int

Candidate Keys: { user_id }

Foreign Key: user_id references non_admin(user_id)

Primary Key: { user_id }

Functional Dependency: user_id → organizer_popularity

Table Definition:

```
CREATE TABLE organizer (
  `user_id` INT NOT NULL,
  `organizer_popularity` INT DEFAULT 0,
  PRIMARY KEY (`user_id`),
  FOREIGN KEY (user_id) REFERENCES non_admin(user_id)
);
```

1.2.6. Verified Organizer

Model: verified_organizer(user_id , organization_name , iban , admin_id verification_date)

Attribute Domains:

user_id → int

organization_name → varchar(30)

iban → varchar(26)

admin_id → int

verification_date → date

Candidate Keys: { user_id }

Foreign Key:

user_id references organizer(user_id)

admin_id references admin(user_id)

Primary Key: { user_id }

Functional Dependency: user_id → organization_name, iban, admin_id, verification_date

Table Definition:

```
CREATE TABLE verified_organizer (
  `user_id` INT NOT NULL,
  `organization_name` VARCHAR(30),
  `iban` VARCHAR(26),
  `admin_id` INT NOT NULL,
  `verification_date` DATE NOT NULL,
  PRIMARY KEY (`user_id`),
  FOREIGN KEY (user_id) REFERENCES organizer(user_id),
  FOREIGN KEY (admin_id) REFERENCES admin(user_id)
);
```

1.2.7. Event

Model: event(event_id , user_id , creation_date , event_location , event_date , event_category , event_title , event_description , event_quota , age_restriction)

Attribute Domains:

event_id → int

user_id → int

creation_date → datetime

event_location → varchar(20)

event_date → date

event_category → varchar(20)

event_title → varchar(30)

event_description → varchar(144)

event_quota → int

age_restriction → int

Candidate Keys: { event_id }

Foreign Key: user_id references organizer(user_id)

Primary Key: { event_id }

Functional Dependency: event_id → user_id, creation_date, event_location, event_date, event_category, event_title, event_description, event_quota, age_restriction

Table Definition:

```
CREATE TABLE `event` (
  `event_id` INT NOT NULL AUTO_INCREMENT,
  `user_id` INT NOT NULL,
  `creation_date` DATETIME DEFAULT CURRENT_TIMESTAMP,
  `event_location` VARCHAR(20) NOT NULL,
  `event_date` DATE NOT NULL,
  `event_category` VARCHAR(20) NOT NULL,
  `event_title` VARCHAR(30) NOT NULL,
  `event_description` VARCHAR(144) NOT NULL,
  `event_quota` INT,
  `age_restriction` INT,
  PRIMARY KEY (`event_id`),
  FOREIGN KEY (user_id) REFERENCES organizer(user_id)
);
```

1.2.8. Paid Event

Model: paid_event(<u>event_id</u> , max_ticket_per_part , user_id)

Attribute Domains:

event_id → int

max_ticket_per_part → int

user_id → int

Candidate Keys: { event_id }

Foreign Key:

event_id references event(event_id)

user_id references verified_organizer(user_id)

Primary Key: { event_id }

Functional Dependency: event_id → max_ticket_per_part, user_id

Table Definition:

```
CREATE TABLE paid_event (
  `event_id` INT NOT NULL,
  `max_ticket_per_part` INT NOT NULL,
  `user_id` INT NOT NULL,
  PRIMARY KEY (`event_id`),
  FOREIGN KEY (event_id) REFERENCES event(event_id),
  FOREIGN KEY (user_id) REFERENCES verified_organizer(user_id)
);
```

1.2.9. Price

Model: price(<u>event_id , ticket_price</u>)

Attribute Domains:

event_id → int

ticket_price → numeric(7, 2)

Candidate Keys: { event_id , ticket_price }

Foreign Key: event_id references paid_event(event_id)

Primary Key: { event_id , ticket_price }

Functional Dependency: none

Table Definition:

```
CREATE TABLE price (
  `event_id` INT NOT NULL,
  `ticket_price` NUMERIC(7,2) NOT NULL,
  PRIMARY KEY (`event_id`, `ticket_price`),
  FOREIGN KEY (event_id) REFERENCES paid_event(event_id)
);
```

1.2.10. Ticket

Model: ticket(<u>ticket_id</u> , is_refundable , ticket_price , event_id)

Attribute Domains:

ticket_id → int

is_refundable → tinyint

ticket_price → numeric(7, 2)

event_id → int

Candidate Keys: { ticket_id }

Foreign Key: event_id, ticket_price references price(event_id, ticket_price)

Primary Key: { ticket_id }

Functional Dependency: ticket_id → is_refundable, ticket_price, event_id

Table Definition:

```
CREATE TABLE ticket (
  `ticket_id` INT NOT NULL,
  `is_refundable` TINYINT NOT NULL,
  `ticket_price` NUMERIC(7,2) NOT NULL,
  `event_id` INT NOT NULL,
  PRIMARY KEY (`ticket_id`),
  FOREIGN KEY (event_id, ticket_price) REFERENCES price(event_id, ticket_price)
);
```

1.2.11. Wallet

Model: wallet(<u>wallet_id</u> , balance)

Attribute Domains:

wallet_id → int

balance → numeric(7, 2)

Candidate Keys: { wallet_id }

Foreign Key: none

Primary Key: { wallet_id }

Functional Dependency: wallet_id → balance

Table Definition:

```
CREATE TABLE wallet (
  `wallet_id` INT NOT NULL,
  `balance` NUMERIC(7,2) DEFAULT 0,
  PRIMARY KEY (`wallet_id`)
);
```

1.2.12. Card

Model: card(wallet_id , card_no , card_holder_name , cvc_no , valid_date)

Attribute Domains:

wallet_id → int

card_no → numeric(16, 0)

card_holder_name → varchar(40)

cvc_no → numeric(3, 0)

valid_date → date

Candidate Keys: { wallet_id, card_no }

Foreign Key: wallet_id references wallet(wallet_id)

Primary Key: { wallet_id, card_no }

Functional Dependency: wallet_id, card_no → card_holder_name, cvc_no, valid_date

Table Definition:

```
CREATE TABLE card (
  `wallet_id` INT NOT NULL,
  `card_no` NUMERIC(16,0) NOT NULL,
  `card_holder_name` VARCHAR(40) NOT NULL,
  `cvc_no` NUMERIC(3,0) NOT NULL,
  `valid_date` DATE NOT NULL,
  PRIMARY KEY (`wallet_id`, `card_no`),
  FOREIGN KEY (wallet_id) REFERENCES wallet(wallet_id)
);
```

1.2.13. Report

Model: report(user_id , report_id , report_title , report_date , report_context)

Attribute Domains:

user_id → int

report_id → int

report_title → varchar(40)

report_date → datetime

report_context → varchar(144)

Candidate Keys: { user_id, report_id }

Foreign Key: user_id references admin(user_id)

Primary Key: { user_id, report_id }

Functional Dependency: user_id, report_id → report_title, report_date, report_context

Table Definition:

```
CREATE TABLE report (
  `user_id` INT NOT NULL,
  `report_id` INT NOT NULL,
  `report_title` VARCHAR(40) NOT NULL,
  `report_date` DATETIME DEFAULT CURRENT_TIMESTAMP,
  `report_context` VARCHAR(144) NOT NULL,
  PRIMARY KEY (`user_id`, `report_id`),
  FOREIGN KEY (user_id) REFERENCES admin(user_id)
);
```

1.2.14. Has

Model: has(user_id , wallet_id)

Attribute Domains:

user_id → int

wallet_id → int

Candidate Keys: { user_id }

Foreign Key:

user_id references participant(user_id)

wallet_id references wallet(wallet_id)

Primary Key: { user_id }

Functional Dependency: user_id → wallet_id

Table Definition:

```
CREATE TABLE has (
  `user_id` INT NOT NULL,
  `wallet_id` INT NOT NULL,
  PRIMARY KEY (`user_id`),
  FOREIGN KEY (user_id) REFERENCES participant(user_id),
  FOREIGN KEY (wallet_id) REFERENCES wallet(wallet_id)
);
```

1.2.15. Joins

Model: joins(<u>user_id , event_id</u>)

Attribute Domains:

user_id → int

event_id → int

Candidate Keys: { user_id , event_id }

Foreign Key:

user_id references participant(user_id)

event_id references event(event_id)

Primary Key: { user_id , event_id }

Functional Dependency: none

Table Definition:

```
CREATE TABLE joins (
  `user_id` INT NOT NULL,
  `event_id` INT NOT NULL,
  PRIMARY KEY (`user_id`, `event_id`),
  FOREIGN KEY (user_id) REFERENCES participant(user_id),
  FOREIGN KEY (event_id) REFERENCES event(event_id)
);
```

1.2.16. Purchase

Model: purchase(<u>ticket_id</u> , user_id , purchase_date)

Attribute Domains:

ticket_id → int

user_id → int

purchase_date → datetime

Candidate Keys: { ticket_id }

Foreign Key:

ticket_id references ticket(ticket_id)

user_id references participant(user_id)

Primary Key: { ticket_id }

Functional Dependency: ticket_id → user_id, purchase_date

Table Definition:

```
CREATE TABLE purchase (
  `ticket_id` INT NOT NULL,
  `user_id` INT NOT NULL,
  `purchase_date` DATETIME DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`ticket_id`),
  FOREIGN KEY (ticket_id) REFERENCES ticket(ticket_id),
  FOREIGN KEY (user_id) REFERENCES participant(user_id)
);
```

2. UI Design and Their Corresponding SQL Statements

2.1. Login

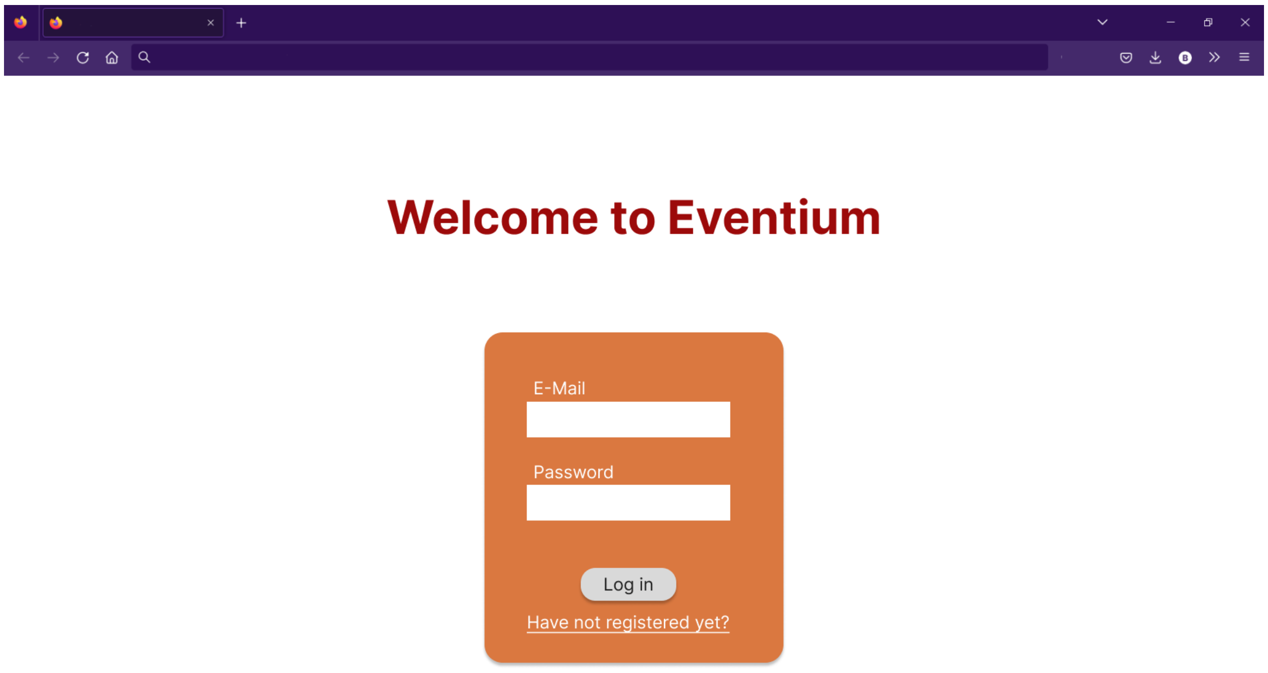


Figure 1: View of the login screen

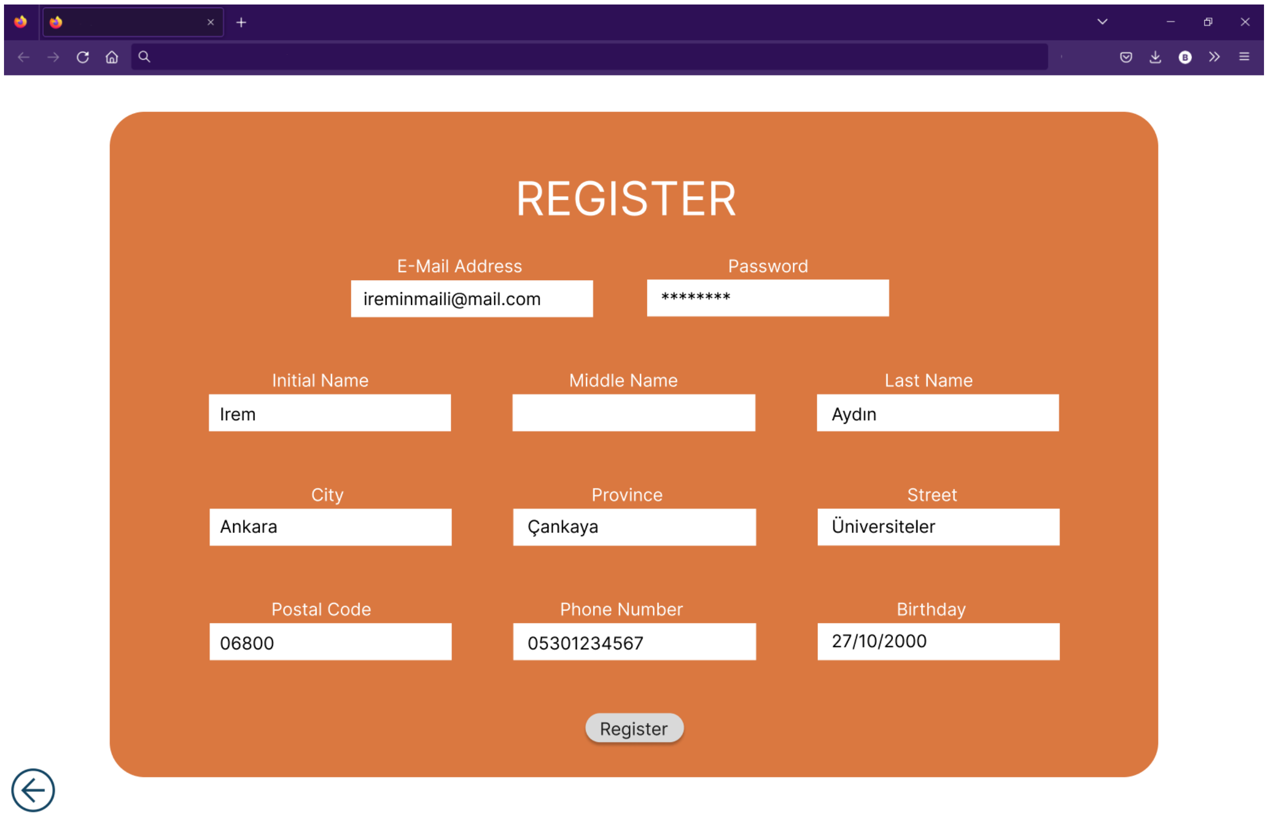
Participants and organizers (both verified and unverified) log in to the site using their registered emails and password. On the other hand, admins use their nickname and password to log in to their home pages.

Inputs: given_email, given_password

Corresponding SQL Statements:

```
SELECT user_id
FROM user U NATURAL JOIN admin A
WHERE ( (U.email = given_email OR A.nickname = given_email) AND
                                            U.password = given_password);
```

2.2. Register



E-Mail Address	Password	
ireminmaili@mail.com	********	
Initial Name	Middle Name	Last Name
Irem		Aydın
City	Province	Street
Ankara	Çankaya	Üniversiteler
Postal Code	Phone Number	Birthday
06800	05301234567	27/10/2000

Figure 2: View of the register screen

Registration is specific to the non_admin type users. They fill in their information, middle name is optional so it can be left blank, and the given information is inserted into user and non_admin tables. The user ID is generated during the insertion, and the participation _points are initialized to zero.

Inputs: given_email, given_password, given_first_name, given_middle_name, given_last_name, given_street, given_province, given_city, given_postal_code, given_birthday, given_phone

Corresponding SQL Statements:

INSERT INTO user(email, password) VALUES (given_email, given_password);

INSERT INTO non_admin(user_id, first_name, middle_name, last_name, street, province, city, postal_code, date_of_birth, phone) VALUES ((SELECT user_id
FROM user U
WHERE U.email = ‘given_email’), 'given_first_name', 'given_middle_name' ,'given_last_name', 'given_street', 'given_province', 'given_city', ‘given_postal_code’, 'given_birthday', ‘given_phone’);

INSERT INTO organizer(user_id) SELECT user_id FROM user WHERE user.email = “given_email”;

INSERT INTO participant(user_id) SELECT user_id FROM user WHERE user.email = “given_email”;

2.3. Participant Home Page

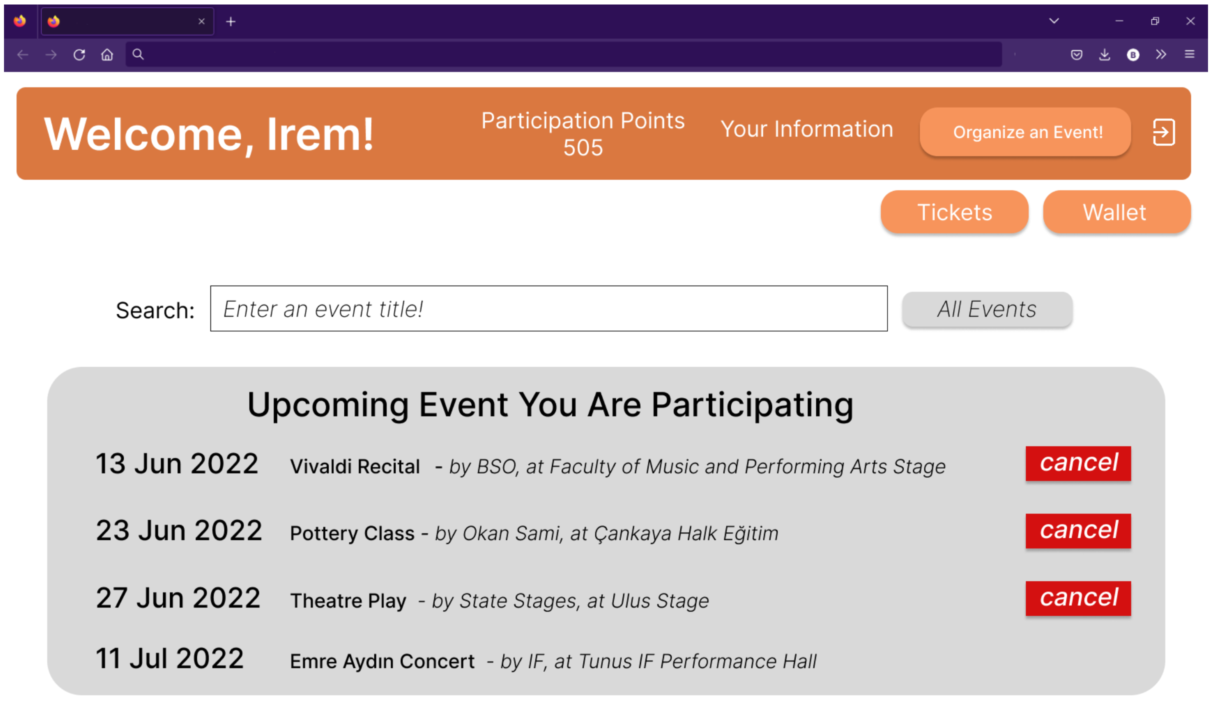

Welcome, Irem!
Participation Points 505
Your Information
Organize an Event!
Tickets
Wallet
Search:
Enter an event title!
All Events
Upcoming Event You Are Participating
13 Jun 2022 Vivaldi Recital - by BSO, at Faculty of Music and Performing Arts Stage
cancel
23 Jun 2022 Pottery Class - by Okan Sami, at Çankaya Halk Eğitim
cancel
27 Jun 2022 Theatre Play - by State Stages, at Ulus Stage
cancel
11 Jul 2022 Emre Aydın Concert - by IF, at Tunus IF Performance Hall

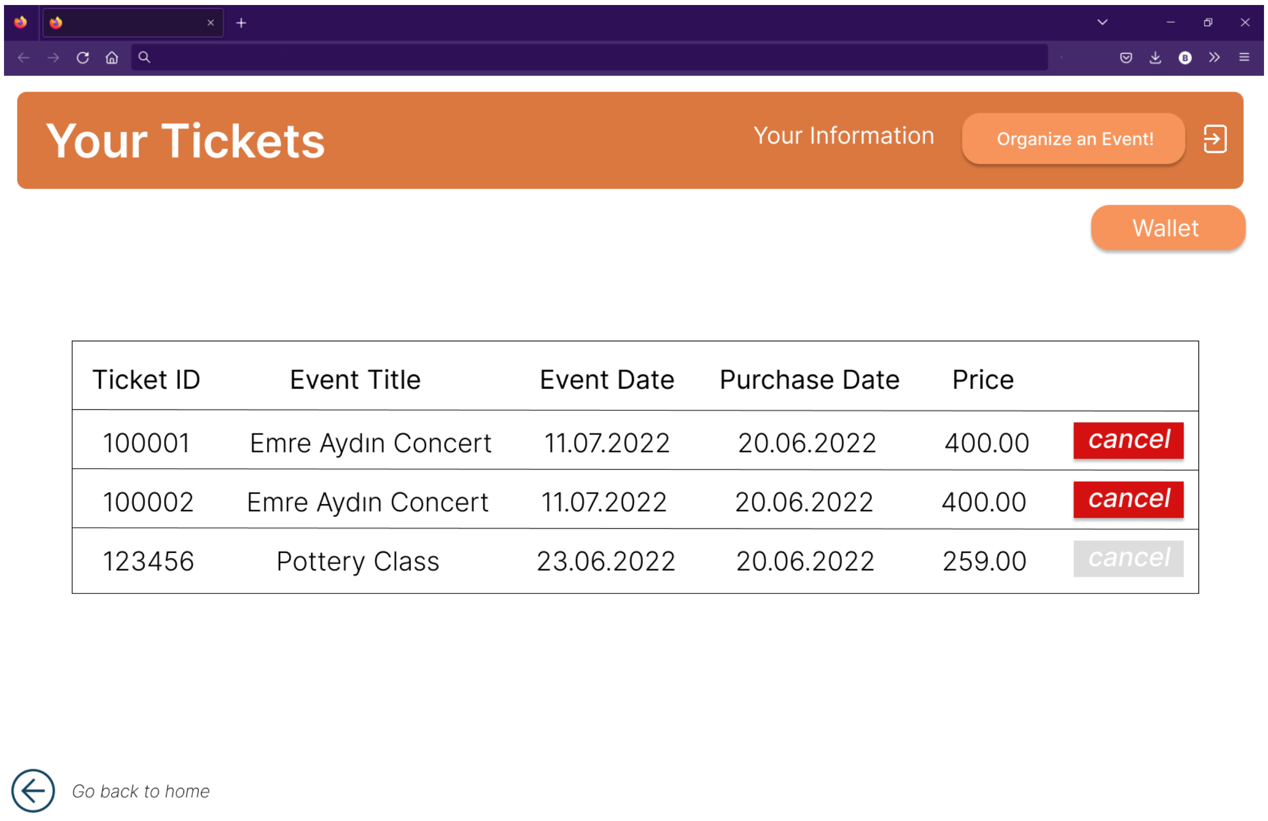
Figure 3: View of the participant home page

A participant's home page displays their name, participation points, and a list of the upcoming events they have joined and paid for. There is a search bar for event titles. The user can either search for a specific event or can view a list of all events. Either way, they are moved to a different event filtering page. They can cancel their participation, view their wallet, view their paid tickets, update their personal information, or switch to the unverified organizer mode by their designated buttons. The user can log out by clicking the logout icon (upper rightmost icon).

Inputs: uid (user id), event_title

```
// For name and participation point of the user
SELECT first_name, participation_points
FROM non_admin NATURAL JOIN participant
WHERE non_admin.user_id = uid AND participant.user_id = uid;

// For the upcoming events that the user are participating
SELECT  event_date, event_title, N.first_name, event_description
FROM non_admin N, joins J, event E
WHERE J.user_id = uid AND J.event_id = E.event_id AND E.user_id = N.user_id;

// For searching event by event title
SELECT event_date, event_title, event_category
FROM event E
WHERE E.event_title LIKE 'event_title%'
```

2.4. Participant Ticket Page



Ticket ID	Event Title	Event Date	Purchase Date	Price	
100001	Emre Aydın Concert	11.07.2022	20.06.2022	400.00	cancel
100002	Emre Aydın Concert	11.07.2022	20.06.2022	400.00	cancel
123456	Pottery Class	23.06.2022	20.06.2022	259.00	cancel

Figure 4: View of the participant's ticket list

A participant can purchase multiple tickets to one event if the verified organizer enables the feature and sets an upper limit. Some tickets are refundable and some are not, so the button to be refunded may be grayed out for some tickets. The tickets are automatically removed, and refunded if possible, if the participation is canceled on the home page.

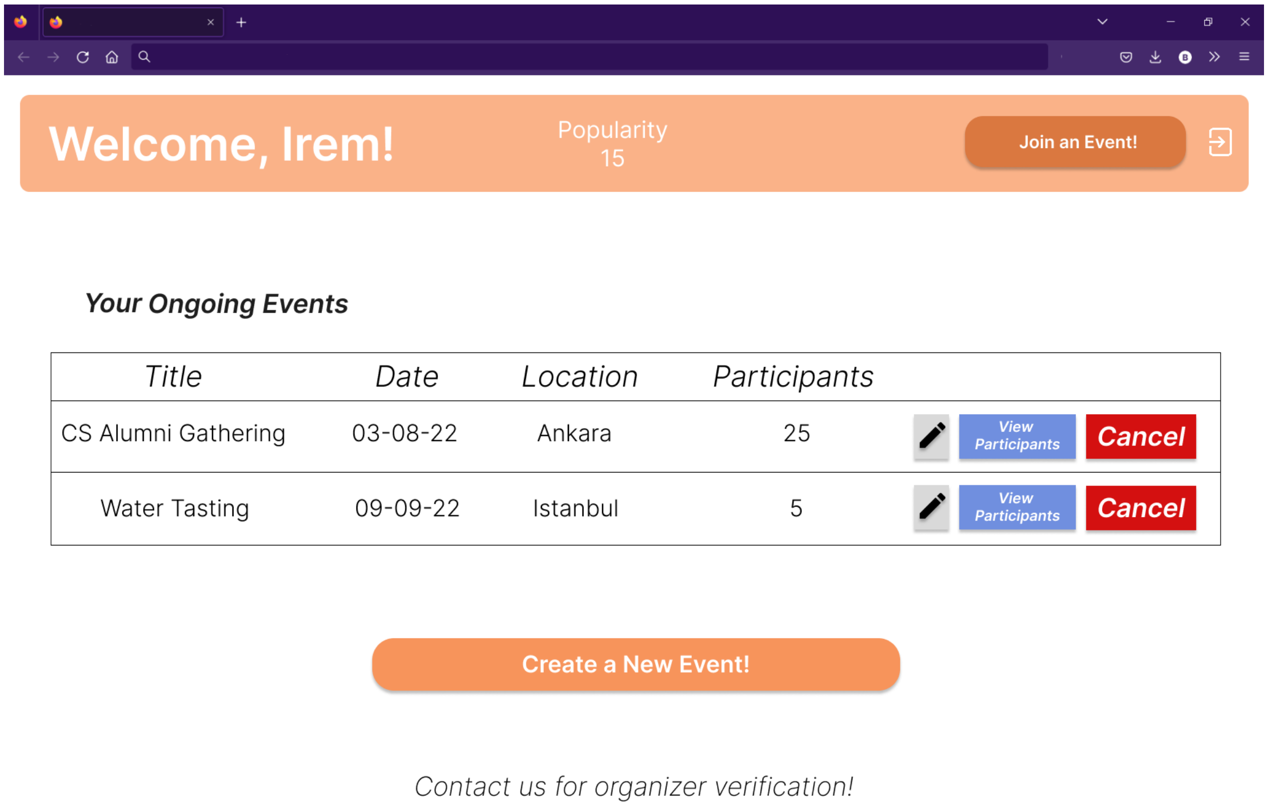
Inputs: uid (user id)

Corresponding SQL Statements:

// For listing all tickets that is purchased by the user

```
SELECT T.ticket_id, E.event_title, E.event_date, P.purchase_date, T.ticket_price, T.is_refundable
FROM ticket T NATURAL JOIN purchase P NATURAL JOIN participant PA, event E
WHERE T.event_id = E.event_id AND P.user_id =  PA.user_id AND
                                        ticket_id in ( SELECT ticket_id
                                                   FROM purchase
                                                   WHERE purchase.user_id = uid );
```

2.5. Organizer Home Page



Title	Date	Location	Participants	
CS Alumni Gathering	03-08-22	Ankara	25	View Participants Cancel
Water Tasting	09-09-22	Istanbul	5	View Participants Cancel

Figure 5: View of the unverified organizer's home page (organizer mode)

An organizer's home page displays the user's name, their popularity rating, and a list of their ongoing events they have created, listed by their title, date, location, and remaining quota. An organizer can edit or cancel their own events, view participants, create a new event, or switch to the participant mode by the designated buttons.

Inputs: uid (user id)

Corresponding SQL Statements:

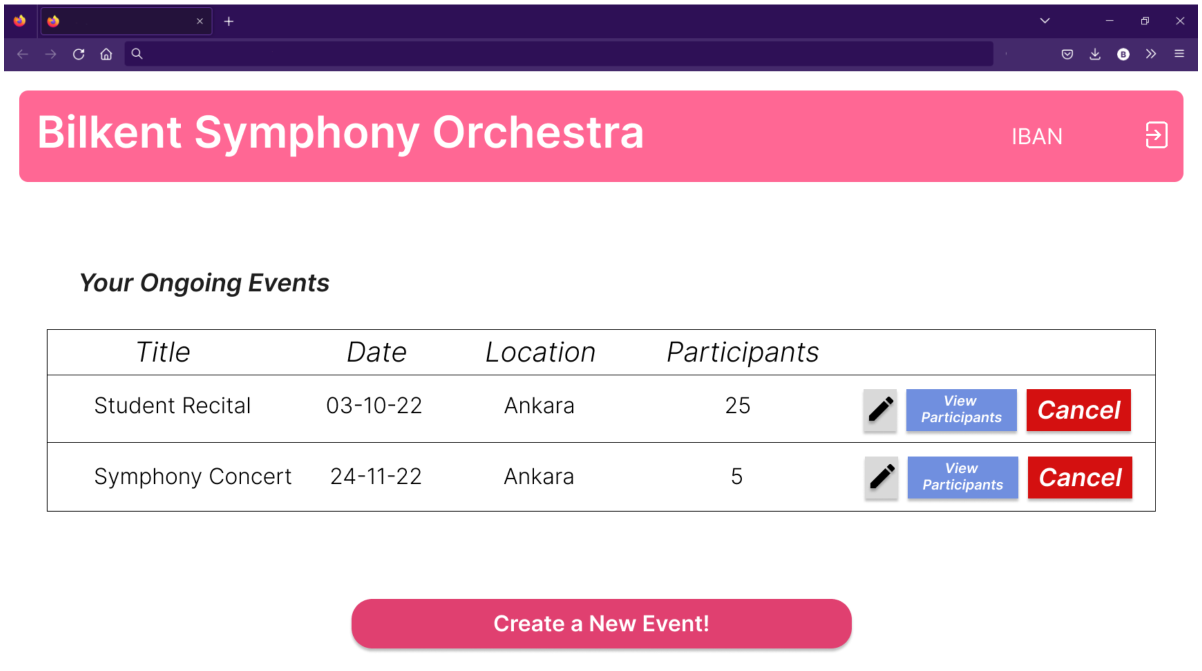
// For the name and popularity of the user

```
SELECT first_name, organizer_popularity
FROM non_admin NATURAL JOIN organizer
WHERE non_admin.user_id = uid
```

// For listing all events that is going to be organized by the user

```
SELECT event_title, event_date, event_location, event_quota
FROM event E
WHERE E.user_id = uid and E.event_date > CURRENT_TIMESTAMP
```

2.6. Verified Organizer Home Page



Your Ongoing Events

Title	Date	Location	Participants
Student Recital	03-10-22	Ankara	25
Symphony Concert	24-11-22	Ankara	5

Figure 6: View of the verified organizer's home page

Verified organizers can do all the things an organizer can do, plus, they can create paid events and sell tickets to those events. Also, a verified organizer can change their name to an organization name. A verified organizer can enter their International Bank Account Number (IBAN) to get a payment for a paid event. Verification can be received by applying to the website admins.

Inputs: uid (user id)

Corresponding SQL Statements:

// For showing the organization name at the top of the page

```
SELECT organization_name
FROM verified_organizer V
WHERE V.user_id = uid
```

// For showing all free events all paid events that are going to be organized by the user

```
SELECT event_title, event_date, event_location, event_quota
FROM event E
WHERE E.user_id = uid and E.event_date > CURRENT_TIMESTAMP
```

2.7. Web Administrator Home Page

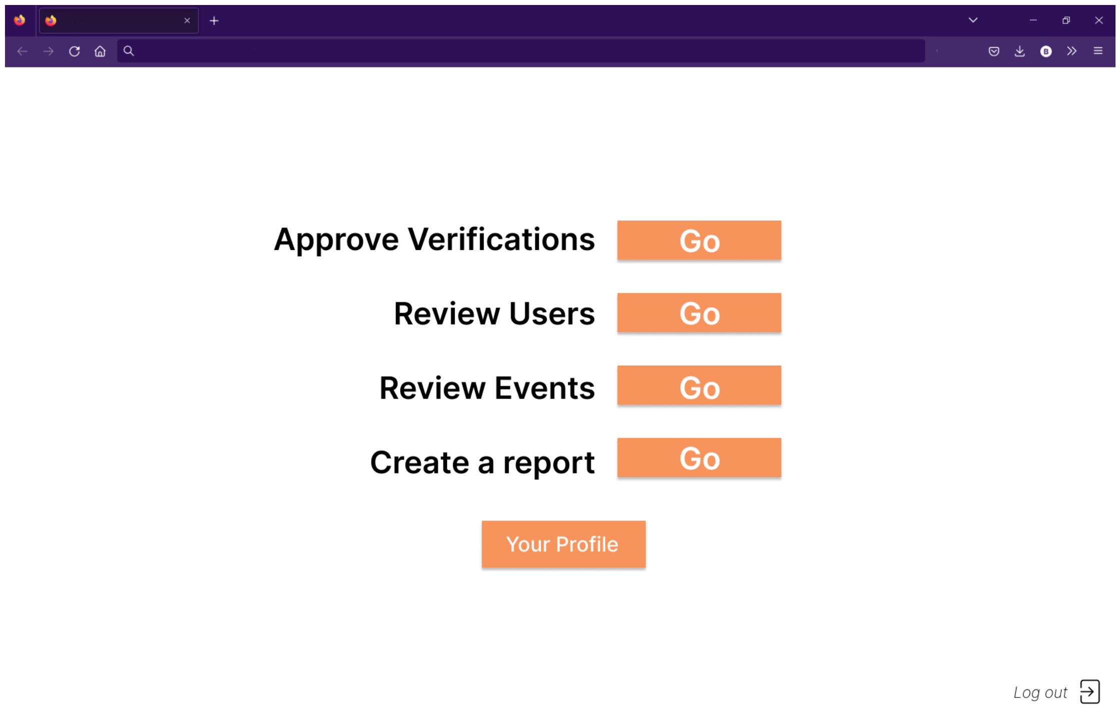


Figure 7: View of the web administrator's home page

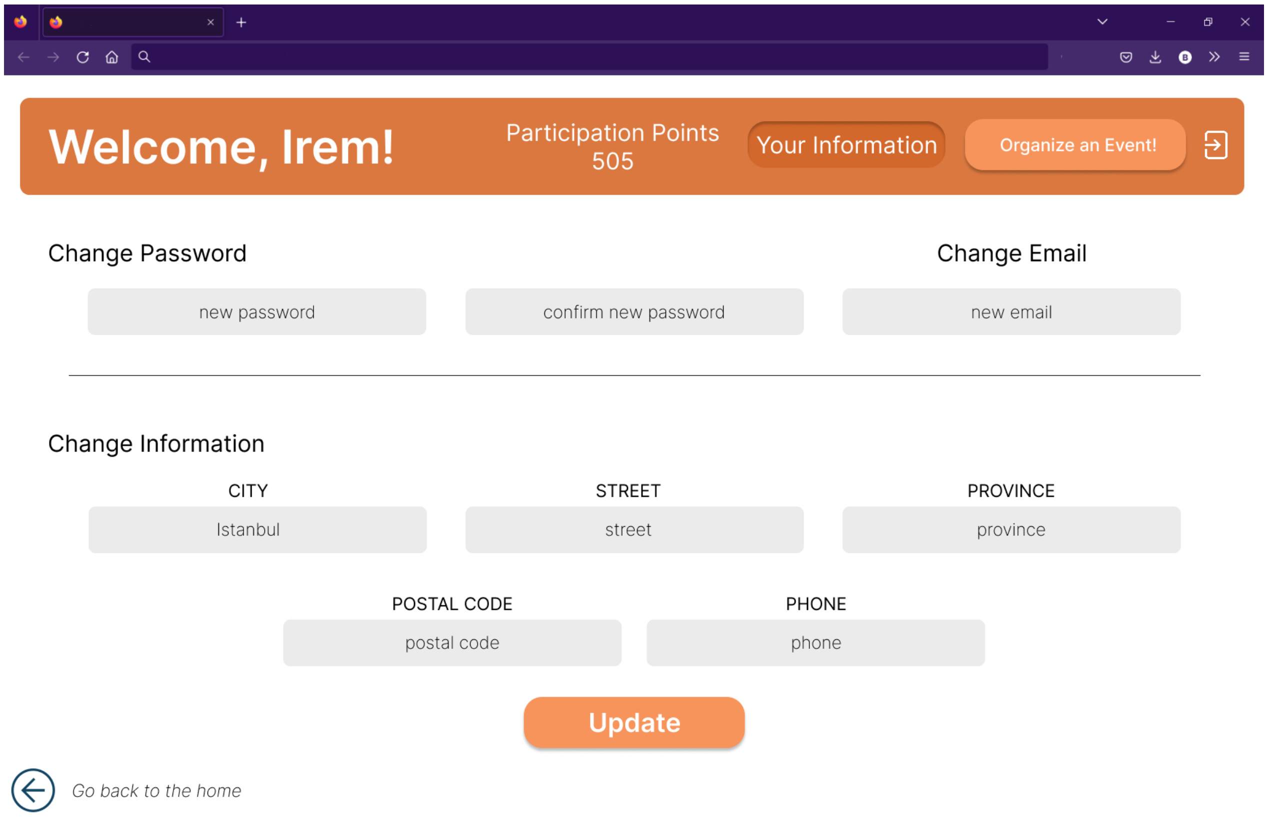
Web admin is the person who is able to manage the website itself. They are able to verify organizers so they can host paid events. They can see all users and events. Also admins are able to ban user accounts and cancel events. Admins can create system reports. For instance a report can retrieve the free event which has the maximum number of participants for each category located in Ankara.

Inputs: event_location (Example for creating a report that retrieves the free event which has the maximum number of participants for each category located in Ankara*)*

The corresponding SQL statement for the example report (Check Reports section for more example reports):

```
WITH temp(event_id, cat, cnt)  as
        ( SELECT E.event_id, E.event_category,COUNT(J.user_id)
          FROM joins J, event E
          WHERE J.event_id = E.event_id AND E.event_location = current_location
          GROUP BY E.event_id, E.event_category )
, temp2( cat, cnt) as (SELECT cat, MAX(cnt)
                       FROM temp
                       GROUP BY cat)
SELECT cat, event_id, cnt
FROM temp NATURAL JOIN temp2;
```

2.8. Update Information



Welcome, Irem!

Participation Points 505

Your Information

Organize an Event!

Change Password

new password

confirm new password

Change Email

new email

Change Information

CITY: Istanbul

STREET: street

PROVINCE: province

POSTAL CODE: postal code

PHONE: phone

Update

Go back to the home

Figure 8: View of the participant's information update page

A participant is able to change their information by filling in the form which opens when the corresponding button is clicked.

Inputs: new_street, new_province, new_city, new_postal_code, new_phone, new_password, new_email, uid (user id)

Corresponding SQL Statements:

// For updating the street:

```
UPDATE non_admin
SET non_admin.street = new_street
WHERE non_admin.user_id = uid;
```

// For updating the province:

```
UPDATE non_admin
SET non_admin.province = new_province
WHERE non_admin.user_id = uid;
```

// For updating the city:

```
UPDATE non_admin
SET non_admin.city = new_city
WHERE non_admin.user_id = uid;
```

// For updating the postal code:

```
UPDATE non_admin
SET non_admin.postal_code = new_postal_code
WHERE non_admin.user_id = uid;
```

// For updating the phone:

```
UPDATE non_admin
SET phone = new_phone
WHERE non_admin.user_id = uid;
```

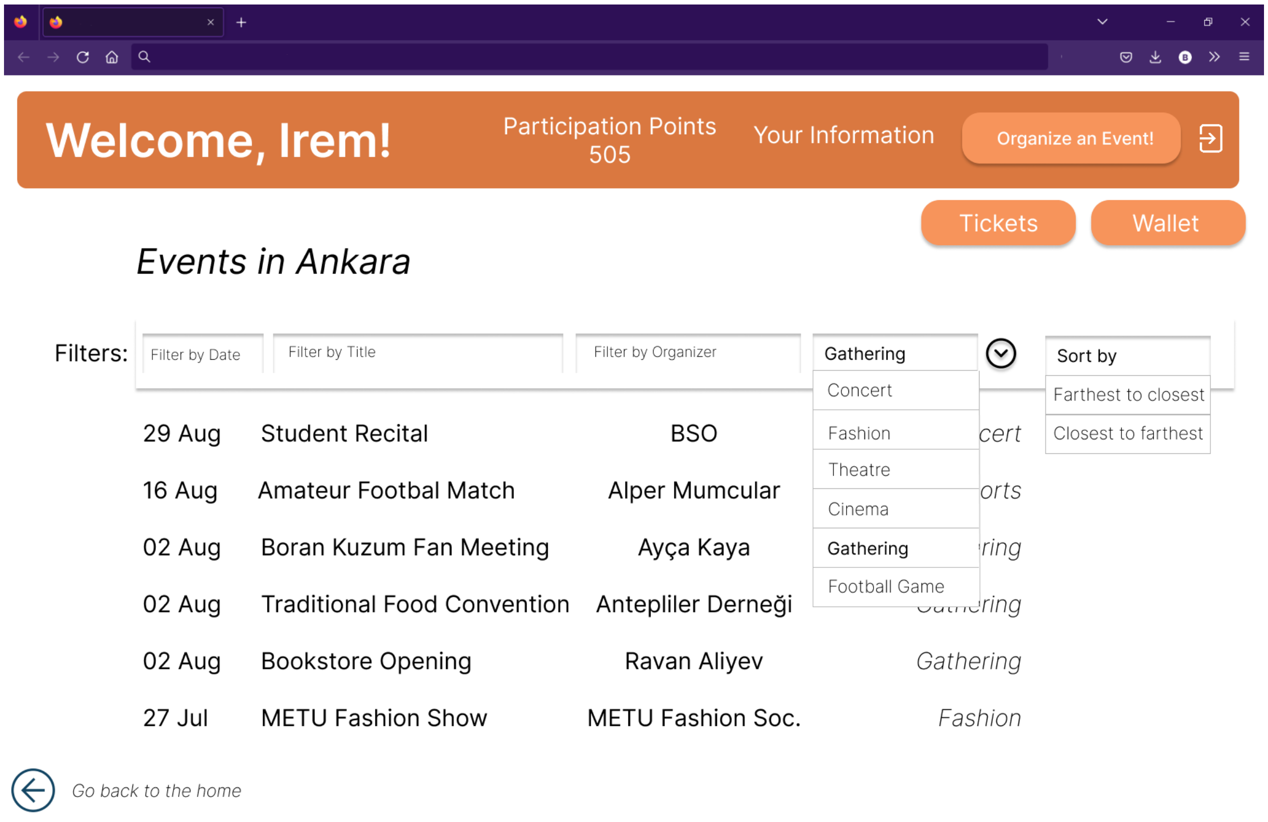
// For updating the password:

```
UPDATE user
SET password = new_password
WHERE user.user_id = uid;
```

// For updating the email:

```
UPDATE user
SET email = new_email
WHERE user.user_id = uid;
```

2.9. Filtering Events



Welcome, Irem! Participation Points 505 Your Information Organize an Event!

Tickets Wallet

Events in Ankara

Filters: Filter by Date | Filter by Title | Filter by Organizer | Gathering | Sort by

Concert / Fashion / Theatre / Cinema / Gathering / Football Game

Farthest to closest / Closest to farthest

29 Aug	Student Recital	BSO	*cert*
16 Aug	Amateur Footbal Match	Alper Mumcular	*orts*
02 Aug	Boran Kuzum Fan Meeting	Ayça Kaya	*ring*
02 Aug	Traditional Food Convention	Antepliler Derneği	*Gathering*
02 Aug	Bookstore Opening	Ravan Aliyev	*Gathering*
27 Jul	METU Fashion Show	METU Fashion Soc.	*Fashion*

Go back to the home

Figure 9: View for event filtering

Participants can view the list of all events in their city (if the user does not satisfy the age restriction of the event, the event won't be listed.) without applying any filtration. They can filter the events by entering a specific date, title, or organizer name. They can choose an event category from the drop-down menu. They can choose to list their events as the farthest from today to the closest or as the closest to today to the farthest in the rightmost dropdown menu.

Inputs: uid, filter_date, filter_title, filter_organizer, filter_category, sort_by (farthest to closest or closest to farthest)

Corresponding SQL Statements:
// For listing all the events in the city where the user lives

```
with temp(user_id, age) as (SELECT user_id, 2022 - YEAR(N.date_of_birth)
                                                    from non_admin N )
SELECT E.event_date, E.event_title, N.first_name, N.last_name, V.organization_name,
                                                                    E.event_category
FROM event E NATURAL JOIN non_admin N  NATURAL JOIN verified_organizer V NATURAL
                                                                          JOIN temp T
```

```
WHERE T.age > E.age_restriction AND E.event_date > CURRENT_TIMESTAMP AND
                                                     event_location in (select NA.city
                                                                        from non_admin NA
                                                                        where NA.user_id = uid)
```

// For listing all the events in the city for a specific date

```
with temp(user_id, age) as (SELECT user_id, 2022 - YEAR(N.date_of_birth)
                                                     from non_admin N )

SELECT E.event_date, E.event_title, N.first_name, N.last_name, V.organization_name,
                                                                        E.event_category
FROM event E NATURAL JOIN non_admin N  NATURAL JOIN verified_organizer V NATURAL
                                                                              JOIN temp T
WHERE T.age > E.age_restriction AND E.event_date = filter_date AND event_location in (select
NA.city
                                                                       from non_admin NA
                                                                       where NA.user_id = uid)
```

// Filter by title

```
with temp(user_id, age) as (SELECT user_id, 2022 - YEAR(N.date_of_birth)
                                                     from non_admin N )

SELECT E.event_date, E.event_title, N.first_name, N.last_name, V.organization_name,
                                                                        E.event_category
FROM event E NATURAL JOIN non_admin N  NATURAL JOIN verified_organizer V NATURAL
                                                                              JOIN temp T
WHERE T.age > E.age_restriction AND E.event_date > CURRENT_TIMESTAMP AND
             E.event_title LIKE ‘filter_title%’ AND event_location in (select NA.city
                                                                        from non_admin NA
                                                                        where NA.user_id = uid)
```

// Filter by organizer

```
with temp2(user_id, age) as (SELECT user_id, 2022 - YEAR(N.date_of_birth)
                                                     from non_admin N )

, temp(full_name) as (SELECT CONCAT(N.first_name, N.last_name, V.organization_name) as
full_name
                      FROM non_admin N  NATURAL JOIN verified_organizer V )

SELECT E.event_date, E.event_title, N.first_name, N.last_name, V.organization_name,
E.event_category

FROM event E NATURAL JOIN non_admin N  NATURAL JOIN verified_organizer V NATURAL
JOIN temp T NATURAL JOIN temp2 T2

WHERE T2.age > E.age_restriction AND E.event_date > CURRENT_TIMESTAMP AND
T.full_name LIKE 'filter_organizer%'  AND event_location in (select NA.city
                                                             from non_admin NA
                                                             where NA.user_id = uid)
```

// Filter by category

```
with temp(user_id, age) as (SELECT user_id, 2022 - YEAR(N.date_of_birth)
                                                          from non_admin N )

SELECT E.event_date, E.event_title, N.first_name, N.last_name, V.organization_name,
E.event_category

FROM event E NATURAL JOIN non_admin N  NATURAL JOIN verified_organizer V NATURAL
                                                                                         JOIN temp T

WHERE T.age > E.age_restriction AND E.event_category = filter_category AND E.event_date >
CURRENT_TIMESTAMP AND event_location in (select NA.city
                    from non_admin NA
                    where NA.user_id = uid)
```

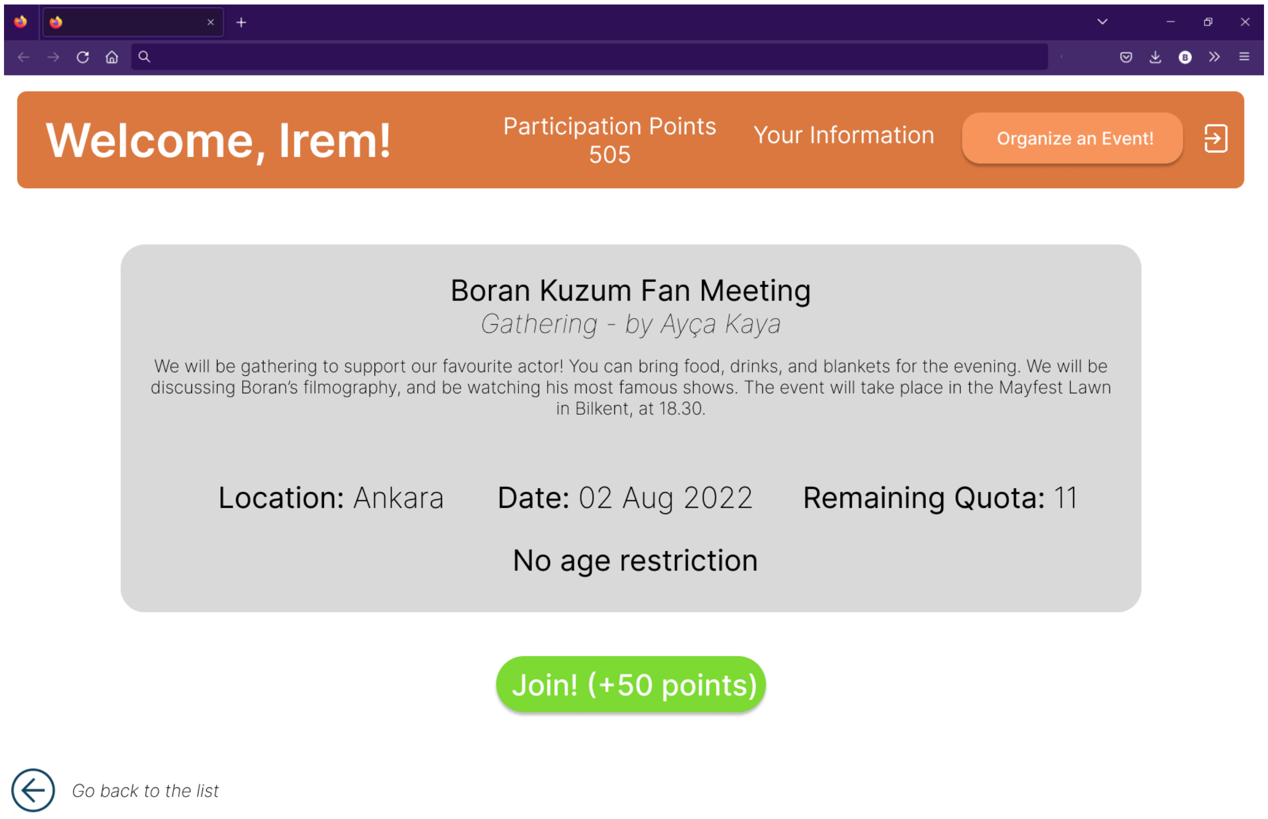
// Filter farthest to closest

```
with temp(user_id, age) as (SELECT user_id, 2022 - YEAR(N.date_of_birth)
                                                          from non_admin N )

SELECT E.event_date, E.event_title, N.first_name, N.last_name, V.organization_name,
E.event_category
FROM event E NATURAL JOIN non_admin N  NATURAL JOIN verified_organizer V NATURAL
                                                                                         JOIN temp T
WHERE T.age > E.age_restriction AND E.event_date > CURRENT_TIMESTAMP AND
event_location in (select NA.city
                    from  non_admin NA
                    where NA.user_id = uid)
ORDER BY E.event_date desc
```

// Filter closest to farthest

```
with temp(user_id, age) as (SELECT user_id, 2022 - YEAR(N.date_of_birth)
                                                          from non_admin N )

SELECT E.event_date, E.event_title, N.first_name, N.last_name, V.organization_name,
E.event_category
FROM event E NATURAL JOIN non_admin N  NATURAL JOIN verified_organizer V NATURAL
                                                                                         JOIN temp T
WHERE T.age > E.age_restriction AND E.event_date > CURRENT_TIMESTAMP AND
event_location in (select NA.city
                    from non_admin NA
                    where NA.user_id = uid)
ORDER BY E.event_date
```

2.10. Join an Event as a Participant


Welcome, Irem!

Participation Points
505

Your Information

Organize an Event!

Boran Kuzum Fan Meeting

Gathering - by Ayça Kaya

We will be gathering to support our favourite actor! You can bring food, drinks, and blankets for the evening. We will be discussing Boran's filmography, and be watching his most famous shows. The event will take place in the Mayfest Lawn in Bilkent, at 18.30.

Location: Ankara Date: 02 Aug 2022 Remaining Quota: 11

No age restriction

Join! (+50 points)

Go back to the list

Figure 10: View of the event details and joining

An event can be viewed in more detail when they are clicked on, in the filtering list. All details related to the event are displayed: its title, category, organizer, description, location, date, remaining quota, and age restriction. The user can join the event by clicking on the green button, and it will increment their participation points by 50 (Check the necessary Trigger below). The user can reach their own personal data from the top bar, they can switch to the organizer mode, and log out by clicking the icon on the rightmost side of the bar. The user can move to the previous filtering list by clicking the arrow on the lower left corner.

Inputs: eid (id of the event)

Corresponding SQL Statements:

// **Show event details**

SELECT event_location, event_date, event_category, event_title, event_description, event_quota, age_restriction
FROM event E
WHERE E.event_id = eid

2.11. Wallet

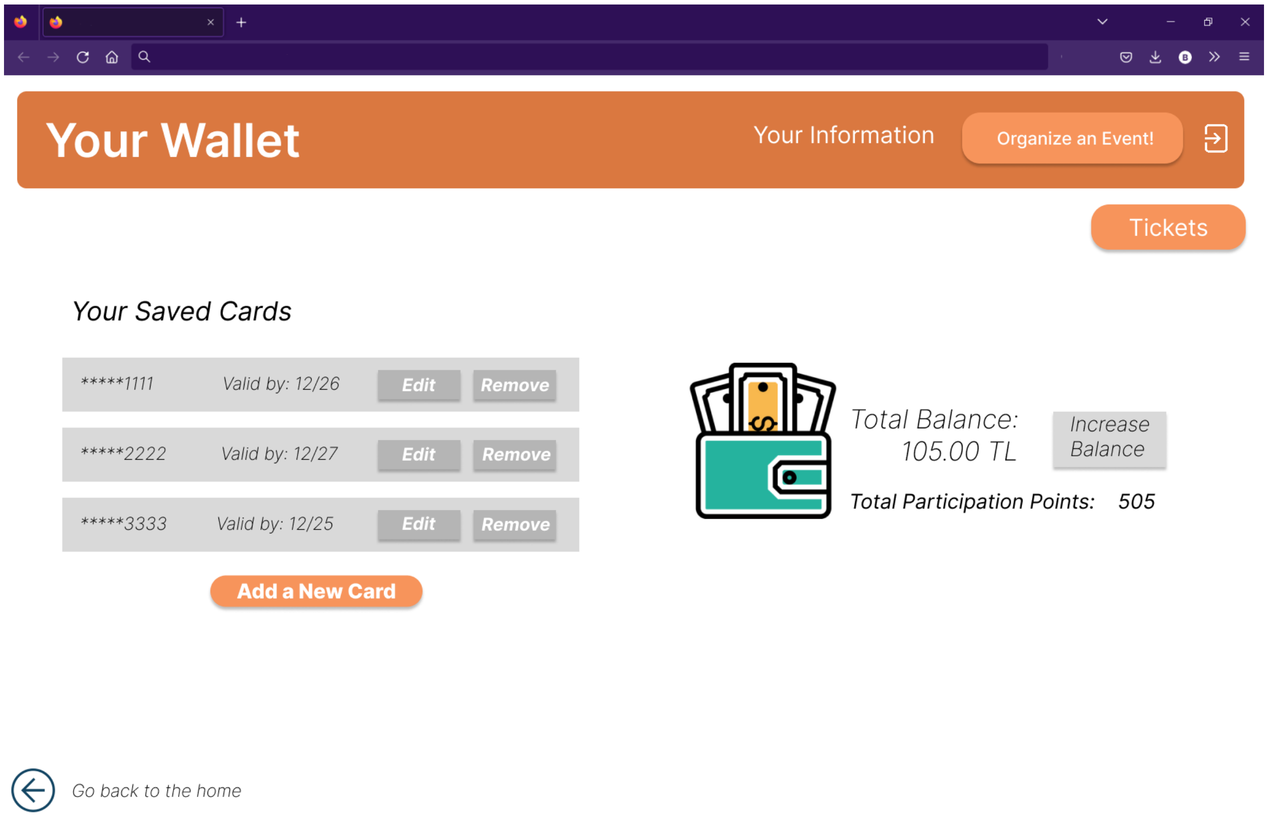

Your Wallet
Your Information
Organize an Event!
Tickets
Your Saved Cards
*****1111 Valid by: 12/26 Edit Remove
*****2222 Valid by: 12/27 Edit Remove
*****3333 Valid by: 12/25 Edit Remove
Add a New Card
Total Balance: 105.00 TL
Increase Balance
Total Participation Points: 505
Go back to the home

Figure 11: View of the participant's wallet

Wallet displays the amount of balance the user has. Next to the balance, the button prompts the user to increase their balance. By clicking the button, the user is redirected to that page to increase the balance from one of the saved cards. It lists the saved credit cards, these cards can be edited or completely removed from the database. The user can also add a new card by clicking the orange button below the list. It redirects to the card addition page. The user can reach their own personal data from the top bar, they can switch to the organizer mode, and log out by clicking the icon on the rightmost side of the bar. The user can move to the previous filtering list by clicking the arrow on the lower left corner.

Inputs: uid (user id)

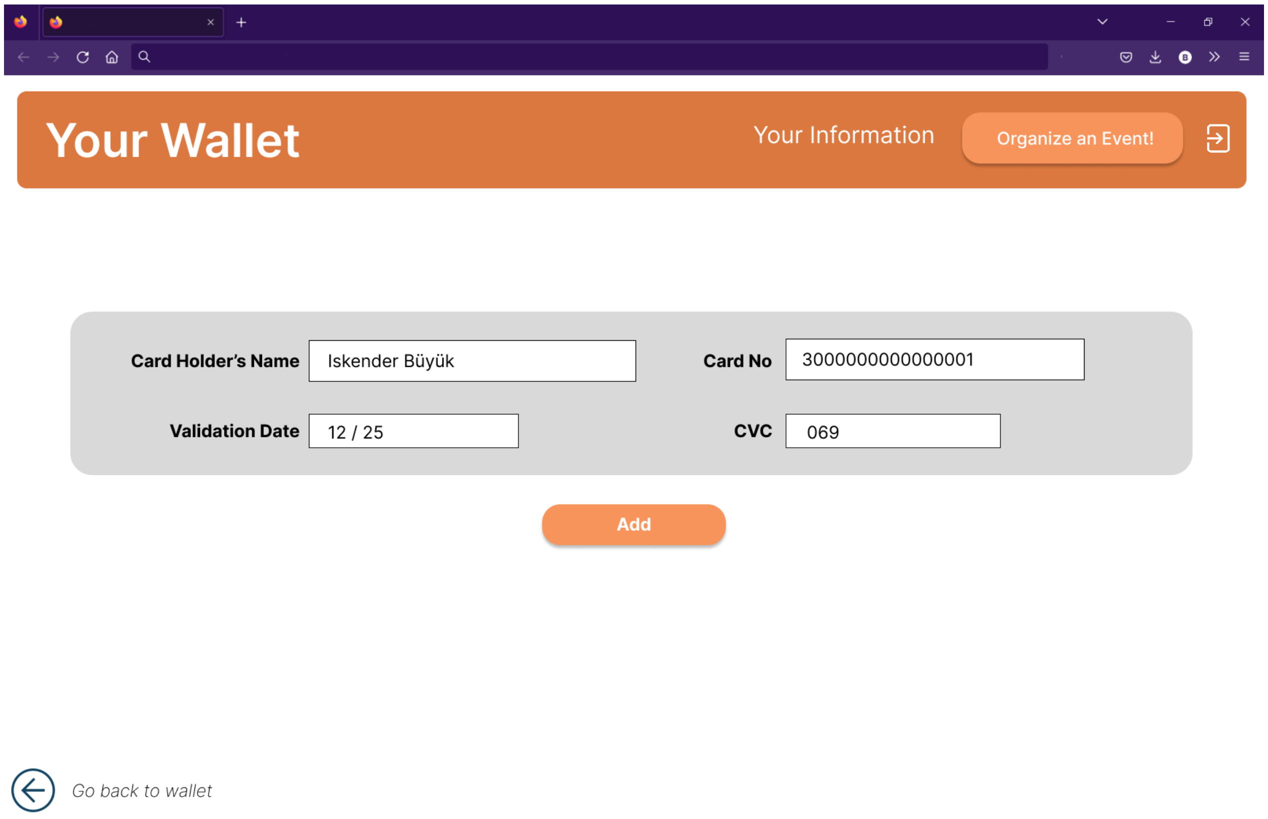
Corresponding SQL Statements:

// For displaying the wallet balance

```
SELECT balance
FROM wallet
WHERE wallet.wallet_id = ( SELECT wallet_id
                           FROM has
                            WHERE has.user_id = uid );
```

// For listing the added cards

```
SELECT card_no, valid_date
FROM card
WHERE card.wallet_id = (   SELECT wallet_id
                           FROM has
                           WHERE has.user_id = uid );
```



Card Holder's Name	Iskender Büyük	Card No	3000000000000001
Validation Date	12 / 25	CVC	069

Figure 12: View of adding a new card to the wallet

Inputs: wid (id of the wallet), new_no, new_holder, new_cvc, new_valid

```
INSERT INTO card( wallet_id , card_no , card_holder_name , cvc_no , valid_date )
VALUES ( wid, new_no, new_holder, new_cvc, new_valid );
```

3. Advanced Database Components

3.1. Report

- List all events in Ankara whose organizer has a popularity rate higher that 90

```
        SELECT organizer .user_id, organizer.organizer_popularity
        FROM organizer, event
        WHERE organizer.organizer_popularity > 90 AND event.event_location =
‘Ankara’ AND organizer.user_id = event.user_id;
```

- Retrieve the free event which has the maximum number of participants for each category located in Ankara

```
WITH temp(event_id, cat, cnt)  as (
                SELECT E.event_id, E.event_category, COUNT(J.user_id)
                FROM joins J, event E
                WHERE J.event_id = E.event_id AND E.event_location = 'Ankara'
                GROUP BY E.event_id, E.event_category )
        , temp2( cat, cnt) as (SELECT cat, MAX(cnt)
                        FROM temp
                        GROUP BY cat)
SELECT cat, event_id, cnt
FROM temp NATURAL JOIN temp2
```

- Retrieve the participants who attended the “sport” events last year

```
SELECT DISTINCT P.user_id
FROM participant P, event E, joins J
WHERE P.user_id = J.user_.id AND E.event_id = J.event_id AND E.category =
‘sport’ AND E.date > ‘2021-11-16’;
```

- List all the concerts which will take place in the next month in Ankara

```
SELECT E.event_id, E.context
FROM event E
```

WHERE E.category = ‘concert’ AND E.location = ‘Ankara’ AND E.date > ‘2022-11-16’;

- Find the average number of participants of events for each category

```
SELECT E.event_id, E.category, avg(total)
FROM event E, participant P, joins J
WHERE E.event_id = J.event_id AND J.user_id = P.user_id
GROUP BY E.category
HAVING count(P.user_id) as total
```

3.2. Views

3.2.1. Participant View for Organizer

Organizers may need to know some name, age and phone number of participants joining an event they created. Some attributes are kept private to provide data privacy. Corresponding SQL query:

```
CREATE VIEW AS view_participants
SELECT P.name, P.age, P.phone
FROM participant P, event E, joins J
WHERE J.event_id = E.event_id AND P.user_id = J.user_id;
```

3.2.2. Wallet View for Admin

Participants’ credit card information should not be visible to admin users. However, admins may need some wallet id or balance value. Corresponding SQL query:

```
CREATE VIEW AS view_wallet
SELECT P.name, P.user_id, P.email, W.wallet_id, W.balance
FROM  participant P, wallet W, has H
WHERE P.user_id = H.user_id AND W.wallet_id = H.wallet_id;
```

3.3. Triggers

3.3.1. Check date of birth of non_admins before insertion, if the date is invalid (greater than todays' date), gives an error

```
delimiter //
CREATE TRIGGER check_age BEFORE INSERT
ON non_admin
FOR EACH ROW
IF NEW.date_of_birth >= CURRENT_TIMESTAMP THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'invalid birth date.';
END IF; //
delimiter ;
```

3.3.2. Check date of events before insertion, if the date has passed, gives an error

```
delimiter //
CREATE TRIGGER check_event_date BEFORE INSERT
ON event
FOR EACH ROW
IF NEW.event_date < CURRENT_TIMESTAMP THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'invalid event date.';
END IF; //
delimiter ;
```

3.3.3. Increment the event quota after deleting row from joins table

```
delimiter //
CREATE TRIGGER inc_quota AFTER DELETE ON joins
FOR EACH ROW
BEGIN
    UPDATE event
    SET event_quota = event_quota + 1
    WHERE event_id = OLD.event_id;
END;
delimiter;
```

3.3.4. Decrement the event quota after inserting a row into joins table

```
delimiter //
CREATE TRIGGER dec_quota AFTER INSERT ON joins
FOR EACH ROW
BEGIN
    UPDATE event
    SET event_quota = event_quota - 1
    WHERE event_id = OLD.event_id;
END;
delimiter;
```

3.3.5. Increment participation points of user after canceling an event

```
delimiter //
CREATE TRIGGER inc_points AFTER INSERT ON joins
FOR EACH ROW
BEGIN
    UPDATE participant
    SET participation_points = participation_points + 50
    WHERE user_id = NEW..user_id;
END;
delimiter;
```

3.3.6. Decrement participation points of user after canceling an event

```
delimiter //
CREATE TRIGGER dec_points AFTER DELETE ON joins
FOR EACH ROW
BEGIN
    UPDATE participant
    SET participation_points = participation_points - 50
    WHERE user_id = OLD.user_id;
END;
delimiter;
```

3.3.7. Checks if the event’s date/time collides with any of the events on the user’s events list.

```
delimiter //
CREATE TRIGGER date_check AFTER INSERT ON joins
FOR EACH ROW
IF  (SELECT COUNT(*)
        FROM event E, joins J
        WHERE J.user_id = NEW.user_id AND J.event_id = E.event_id
AND E.event_date in (SELECT event_date
                            FROM event EV
                            WHERE NEW.event_id = EV.event_id ) ) > 1 THEN
  SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'date conflict.';
END IF;
END;
delimiter;
```

3.4. Constraints

Integrity constraints during the creation of tables are listed in the following formats.

- not null
- primary key(A1, ..., An)
- foreign key (A1, ..., An) references r
- unique
- default
- auto_increment

Therefore, there was no need for renaming.

4. Functional Components

4.1. Use Case Diagram

The use case diagram is given on the next page.

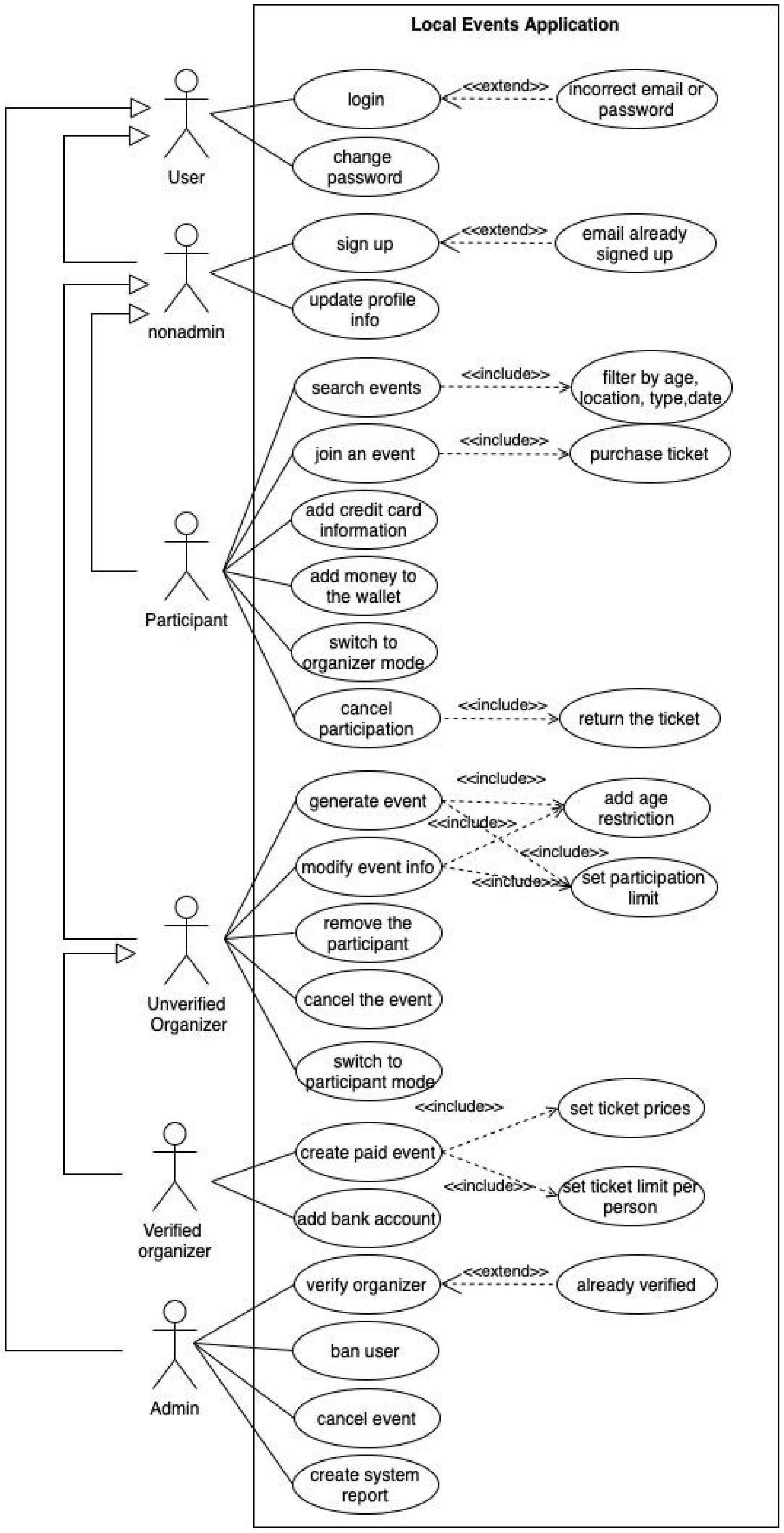


Figure 13: Use Case Diagram for Local Events Application

4.2. Use Case Scenarios

4.2.1. Login

Use Case Name:	Login
Participating Actor:	User
Entry Condition:	User is on the login screen
Exit Condition:	User has logged in to the system
Flow Of Events:	
1. Basic Flow	
1.1 User inputs the email address for their account	
1.2 User inputs the password for their account	
	1.3 System checks the credentials
1.4 User logs in to the system	
	1.5 The System displays the main screen according to the user type
2 Exceptional Cases:	
2.1 User inputs the wrong email address or password	
	2.1.1 System shows an error message saying wrong email address or password
Special/Quality Requirements: ● Extends the incorrect email or password	

Figure 14: Textual use case for Login

4.2.2. Change Password

Use Case Name: Change Password
Participating Actor: User
Entry Condition: User clicks on the change password button
Exit Condition: User successfully changes the password
Flow Of Events:
1.Basic Flow
1.1 User clicks on the change password button
1.2 System displays the change password screen
1.3 User enters the new password
1.4 System checks the validity of the password
1.5 System shows the message indicating password has been changed successfully

Figure 15: Textual use case for Change Password

4.2.3. Sign up

Use Case Name: Sign Up
Participating Actor: Non admin
Entry Condition: Non admin is on the sign up screen
Exit Condition: Non admin successfully completes sign up
Flow Of Events:
1.Basic Flow
1.1 Non Admin fills the required fields in the sign up screen
1.2 System checks input information for validation
1.3 System redirects the non admin to the main screen
2. Exceptional Cases:
2.1 Non Admin inputs already taken email address
2.1.1 System displays an error message indicating the email is already taken
Special/Quality Requirements: ● Extends the ”email already signed up” exceptional use case

Figure 16: Textual use case for Sign Up

4.2.4. Update profile info

Use Case Name:	Update Profile Info
Participating Actor:	Non Admin
Entry Condition:	Non admin is on the update profile info screen
Exit Condition:	Non Admin completes updating
Flow Of Events:	
1.Basic Flow	
1.1 Non admin changes the desired info	
	1.2 System checks the validation of the inputs

Figure 17: Textual use case for Update Profile Info

4.2.5. Search events

Use Case Name:	Search Events
Participating Actor:	Participant
Entry Condition:	Participant is in the main screen
Exit Condition:	Participant has received the list of the events
Flow Of Events:	
1.Basic Flow	
1.1 Participant inputs the search filters and clicks search button	
	1.2 System displays the events fitting to the filters

Figure 18: Textual use case for Search Events

4.2.6. Join an event

Use Case Name: Join an Event
Participating Actor: Participant
Entry Condition: Participant clicks on the join button
Exit Condition: Participant has successfully joined the event
Flow Of Events:
1.Basic Flow
1.1 Participant clicks on one of the listed events which is free
1.2 System displays the detailed information about the event
1.3 Participant clicks on the join button
1.4 System displays the “joined to the event successfully ” message
2 Alternative flow
2.1 Participant clicks on one of the listed events which is not free
2.2 System displays the detailed information about the event
2.3 Participant chooses one of the payment types
2.4 If Participant chooses the “Wallet” option then the system takes the payment from “Wallet” otherwise it takes from the credit card.
2.5 System displays the “joined to the event successfully ” message

Figure 19: Textual use case for Join an Event

4.2.7. Add credit card info

Use Case Name: Add Credit card Information
Participating Actor: Participant
Entry Condition: Participant is in the “Credit Card Information” screen
Exit Condition: Participant successfully adds the credit card information
Flow Of Events:
1.Basic Flow
1.1 Participant fills the required details for the credit card and clicks save button
1.2 System checks the validation of the credit card details and displays the “successfully saved” message

Figure 20: Textual use case for Add Credit Card Information

4.2.8. Add money to wallet

Use Case Name:	Add Money to the Wallet
Participating Actor:	Participant
Entry Condition:	Participant clicks on the “increase balance” button on the Your Wallet screen
Exit Condition:	Participant increases wallet balance successfully
Flow Of Events:	
1.Basic Flow	
1.1 Participant clicks on the “increase balance” button on the Your Wallet screen	
	1.2 System displays the “Increase balance” screen
1.3 Participant chooses the credit card and amount to increase the balance	
	1.4 System takes the payment from the chosen Credit card, increases the balance accordingly and shows the “Balance Increased successfully” message

Figure 21: Textual use case for Add Money to the Wallet

4.2.9. Switch to organizer mode

Use Case Name:	Switch to Organizer Mode
Participating Actor:	Participant
Entry Condition:	Participant clicks “Organize an event” button
Exit Condition:	Participant is directed to the Organizer screed
Flow Of Events:	
1.Basic Flow	
1.1 Participant clicks on the “Organize an event” button on the Participant screen	
	1.2 System redirects the participant to the organizer screen

Figure 22: Textual use case for Switch to Organizer Mode

4.2.10. Cancel participation

Use Case Name: Cancel Participation
Participating Actor: Participant
Entry Condition: Participant clicks on the “cancel” button of the desired event on the “Your tickets” screen
Exit Condition: Event has been canceled successfully
Flow Of Events:
1.Basic Flow
1.1 Participant clicks on the cancel button of the free event
1.2 System cancels the participation of the Participant in that event and removes the event from your events list
2. Alternative Flow
2.1 Participant clicks on the cancel button of the event which needs a ticket
2.2 System refunds the money of the Participant and removes the event from your events list

Figure 23: Textual use case for Cancel Participation

4.2.11. Generate an event

Use Case Name: Generate an Event
Participating Actor: Unverified Organizer
Entry Condition: Unverified Organizer is on the “Organizer Home Page” and clicks “Create a new event!” button
Exit Condition: Unverified Organizer creates a new event successfully
Flow Of Events:
1.Basic Flow
1.1 Unverified Organizer is on the “Organizer Home Page” and clicks “Create a new event!” button
1.2 System redirects the Unverified Organizer to the “Event Creation Page”
1.3 Unverified Organizer inputs the detailed information for the event and clicks “Create” button
1.4 System checks the validation of the inputs, redirects the unverified organizer to the “Organizer Home Page” and adds the event to the events list

Figure 24: Textual use case for Generate an Event

4.2.12. Modify the event info

Use Case Name:	Modify The Event Info
Participating Actor:	Unverified Organizer
Entry Condition:	Unverified Organizer clicks the “Edit” button of the desired event
Exit Condition:	Event details are modified successfully
Flow Of Events:	
1.Basic Flow	
1.1 Unverified Organizer clicks the “Edit” button of the desired event	
	1.2 System redirects the unverified organizer to the “Edit Event Details Page”
1.3 Unverified Organizer changes the required details and clicks the save button	
	1.4 System saves the changes and displays the “Changes saved successfully!” message and redirects the unverified organizer to the “Unverified Organizer Home Page”

Figure 25: Textual use case for Modify The Event Information

4.2.13. Remove the participant

Use Case Name:	Remove The Participant
Participating Actor:	Unverified Organizer
Entry Condition:	Unverified Organizer clicks the “Remove Participant” button in the “Participants” page
Exit Condition:	Participant has been removed successfully
Flow Of Events:	
1.Basic Flow	
1.1 Unverified Organizer clicks the “Remove Participant” button in the “Participants” page	
	1.2 System removes the corresponding participant from the event and updates the participants list

Figure 26: Textual use case for Remove the Participant

4.2.14. Cancel the event

Use Case Name:	Cancel the Event
Participating Actor:	Unverified Organizer
Entry Condition:	Unverified Organizer clicks the “Cancel” button of the desired event
Exit Condition:	Event has been canceled successfully
Flow Of Events:	
1.Basic Flow	
1.1 Unverified Organizer clicks the “Cancel” button of the desired event	
	1.2 System cancels the event and removes it from the events list

Figure 27: Textual use case for Cancel the Event

4.2.15. Switch to the participant mode

Use Case Name:	Switch to the Participant Mode
Participating Actor:	Unverified Organizer
Entry Condition:	Unverified Organizer clicks the “Join an Event!” button
Exit Condition:	Unverified organizer is directed to the “Participants Home Page”
Flow Of Events:	
1.Basic Flow	
1.1 Unverified Organizer clicks the “Join an Event!” button	
	1.2 System directs the unverified organizer to the “Participants Home Page”

Figure 28: Textual use case for Switch to the Participation Mode

4.2.16. Create paid event

Use Case Name: Create Paid Event:
Participating Actor: Verified Organizer
Entry Condition: Verified Organizer clicks on the “Create a New Event”
Exit Condition: New event is created successfully
Flow Of Events:
1.Basic Flow
1.1 Verified Organizer clicks on the “Create a New Event” button on “Organizers Home Page” screen
1.2 System directs the Verified Organizer to the “Event Creation Page”
1.3 Verified Organizer inputs the detailed information for the event,sets the ticket price, ticket limit per participant and clicks “Create” button
1.4 System checks the validation of the inputs, redirects the verified organizer to the “Organizer Home Page” and adds the event to the events list

Figure 29: Textual use case for Create Paid Event

4.2.17. Add bank account

Use Case Name: Add Bank Account
Participating Actor: Verified Organizer
Entry Condition: Verified Organizer clicks the IBAN button
Exit Condition: IBAN address has been added successfully
Flow Of Events:
1.Basic Flow
1.1 Verified Organizer clicks the IBAN button on the Organizer Home Page screen
1.2 System directs the verified organizer to the “Add Bank Account Page”
1.3 Verified Organizer inputs the required information for the Bank Account and clicks the save button
1.4 System checks the inputs for the validation and displays the “Bank account information has been saved successfully”

Figure 30: Textual use case for Add Bank Account

4.2.18. Verify organizer

Use Case Name: Verify Organizer
Participating Actor: Admin
Entry Condition: Admin clicks the “Verify the Organizer” button
Exit Condition: Organizer has been verified successfully
Flow Of Events:
1.Basic Flow
1.1 Admin clicks on the “Verify the Organizer” button
1.2 System verifies the organizer and displays the “Organizer has been verified successfully!” message
2 Exceptional cases
2.1 Admin tries to verify the already verified organizer
2.2 System displays an error message saying ”organizer is already verified”
Special/Quality Requirements: ● Extends already verified exceptional use case

Figure 31: Textual use case for Verify Organizer

4.2.19. Ban user

Use Case Name: Ban User
Participating Actor: Admin
Entry Condition: Admin clicks on the “Ban” button for the desired user
Exit Condition: User is banned successfully
Flow Of Events:
1.Basic Flow
1.1 Admin clicks on the “Ban” button for the desired user
1.2 System bans the user and displays the “User has been banned successfully” message

Figure 32: Textual use case for Ban User

4.2.20. Cancel event

Use Case Name: Cancel Event
Participating Actor: Admin
Entry Condition: Admin clicks on the “Cancel” button for the desired event
Exit Condition: Event is banned successfully
Flow Of Events:
1.Basic Flow
1.1 Admin clicks on the “Cancel” button for the desired event from the events list
1.2 System cancels the chosen event and displays the “Event has been canceled successfully!” message

Figure 33: Textual use case for Cancel Event

4.2.21. Create the system report

Use Case Name: Create The System Report
Participating Actor: Admin
Entry Condition: Admin is on the “Create the System Report” page
Exit Condition: Report is created
Flow Of Events:
1.Basic Flow
1.1 Admin sets the desired specifications for the report and clicks “Create The System Report” button
1.2 System generates the report according to the specifications and displays it

Figure 34: Textual use case for Create the System Report

5. Implementation Plan

- MySQL will be used for the database implementation.
- React framework will be used for the front end.
- PHP will be used for the back end.