Bilkent University
Department of Computer Engineering

CS 353-Database Management Systems

I-Venue
Group #27

Design Report

April 02, 2018

Seda Gülkesen - 21302403
İlhami Kayacan Kaya -21502875
Göktuğ Özdoğan -21301042
İlhami Özer -21300828

Course Instructor: Çağrı Toraman

This report is submitted to the GitHub in partial fulfilment of the requirements of the Database Management Systems Project, course CS353.

Table of Contents:

1. REVISED E/R DIAGRAM

1.1. Changes Made in the E/R Diagram

We had changed our E/R model according to feedback we got from assistant. Changes are as follows:

- Primary key of Account has been deleted from User and Venue Owner since there is no need to show in specialization.
- *profilePciture* attribute has been added to Account.
- *date* attribute has been added to *check_in* relation.
- *checkInID* attribute has been added to *check_in* relation.
- *coinbalance* attribute has been added to User.
- *numberOfComments* has been deleted from Account.
- *totalCheckin* attribute has been deleted from User.
- hereNow, likeCount, dislikeCount, totalVisit and totalCount attributes have been deleted from Venue.
- *numberOfVenues* attribute has been deleted from VenueOwner.
- *venuetype, venuename* attributes have been added to Venue.
- *address* attribute has been changed into the composite attribute. It consists of *streetname*, *city* and *zipcode*.
- Venue is no longer a weak entity.

- Photo entity has been added. Account(both User and Venue Owner) can add photos to venue. User can now add photos to check-in.

- *user_has_friends* relation has been added. User can now add friends.

- A new feature, *user_has_matched* relation has been added. Users will be matched with each other if both checked-in in same place at least 5 times (number subject to change).

- *user_rates_comment* Rand *user_rates_venue* relations has been added. User can rate Venue and Comment. Both relations have an attribute called *rate.* In a tuple, if rate is 1, it means like, if it is -1 then it means dislike.

- *venue_owner_replies* relation has been added. Venue Owner can now reply comments, if they are made by User to Venue.

1.2. Revised E/R Diagram

Figure 1: E/R Diagram

2. RELATIONAL SCHEMAS

2.1. Photo

Relational Model:

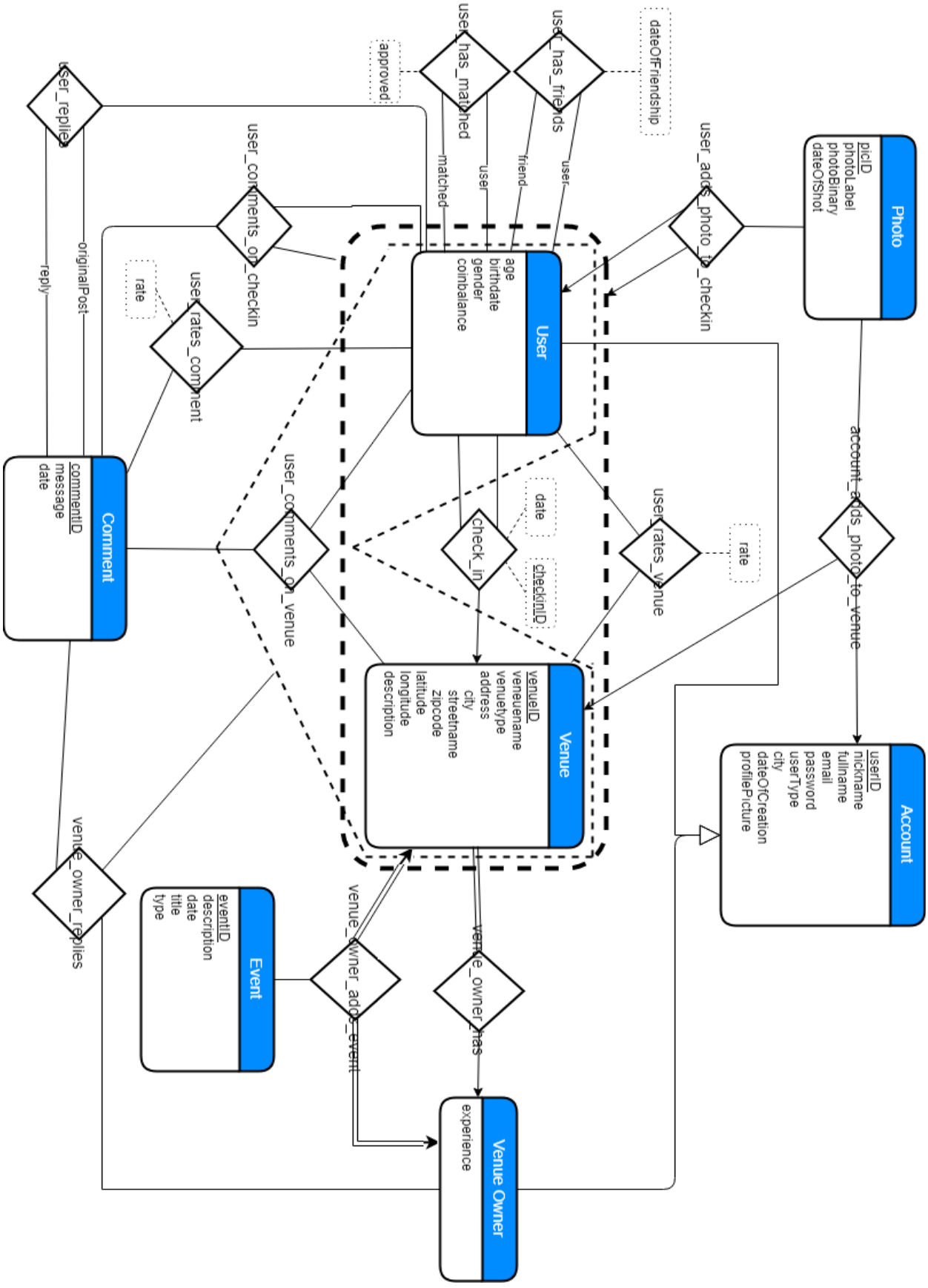
Photo(picID,photoLabel,photoBinary,dateOfShot,userID,venueID,checkInID)

Functional Dependencies:
picID - > photoLabel photoBinary dateOfShot userID venueID checkInID

Candidate Keys:
{ (picID) }

Normal Form
BCNF
Table Definition

```
CREATE TABLE photo(
        picID           INT PRIMARY KEY ,
        userID          INT NOT NULL,
        checkInID       INT NOT NULL,
        venueID         INT NOT NULL,
        photoLabel      VARCHAR(40) NOT NULL,
        photoBinary     BLOB,
        dataOfShot      DATE,
        FOREIGN KEY (userID) references account,
        FOREIGN KEY (checkInID) references check_in,
```



FOREIGN KEY (venueID) references venue) ENGINE=InnoDB;

2.2. Account

Relational Model:
Account(userID, nickname, fullname, email, password, userType, city, dateOfCreation, profilePicture)

Functional Dependencies:
userID - >nickname fullname email password userType city dateOfCreation profilePicture
nickname -> userID fullname email password userType city dateOfCreation profilePicture

Candidate Keys:
{ (userId, nickname) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Account (
        userId              INT PRIMARY KEY ,
        nickname            VARCHAR(40)  NOT NULL,
        fullname            VARCHAR ( 40) NOT NULL,
        email               VARCHAR ( 40) NOT NULL,
        password            VARCHAR ( 40) NOT NULL,
        userType            VARCHAR ( 40) NOT NULL,
        city                VARCHAR ( 40) NOT NULL,
        dateOfCreation      DATE( 40),
        profilePicture      BLOB ) ENGINE=InnoDB;
```

2.3. VenueOwner

Relational Model

VenueOwner(userID,experience)

Functional Dependencies:

userID - > experience

Candidate Keys:

{ (userID) }

Normal Form:

BCNF

Table Definition

```
CREATE TABLE VeneueOwner (     userId          INT PRIMARY KEY ,
                               experince       INT  ) ENGINE=InnoDB;
```

2.4. Venue

Relational Model:

Venue(venueID,venuename,venuetype,city,streetname,zipcode,latitude,longitude,description,userID)

Functional Dependencies:

venueID -> venuename venuetype city streetname zipcode latitude longitude description userID

Candidate Keys:

{ (venueID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Venue (
        venueId             INT PRIMARY KEY ,
        userId              INT NOT NULL,
        venuename           VARCHAR(40) NOT NULL,
        venuetype           VARCHAR ( 40) NOT NULL,
        latitude            INT NOT NULL,
        longitude           INT NOT NULL,
        city                VARCHAR(40) NOT NULL,
        streetname          VARCHAR(40) NOT NULL
        zipcode             INT NOT NULL
        description         VARCHAR(40) NOT NULL,
        FOREIGN KEY (userID) references venue ) ENGINE=InnoDB;
```

2.5. User

Relational Model:

User(userID,age,birthdate,gender,coinbalance)

Functional Dependencies:

userID -> age birthdate gender totalChekIn coinbalance

Candidate Keys:

{ (userID) }

Normal Form:

BCNF

Table Definition

```
CREATE TABLE User(
        userId              INT PRIMARY KEY ,
        age                 INT NOT NULL,
        birthdate           INT NOT NULL,
        gender              VARCHAR(40) NOT NULL,
        coinbalance         INT ) ENGINE=InnoDB;
```

2.6. UserRatesVenue

Relational Model:

User_rates_venue (userID,venueId,rate)

Functional Dependencies:

No dependencies .

Candidate Keys:

{ (userID, venueId) }

Normal Form

BCNF

Table Definition

```
CREATE TABLE user_rates_venue(
        userId              INT NOT NULL,
        venueId             INT NOT NULL,
        rate                INT NOT NULL,
        FOREIGN KEY (userID) references User,
FOREIGN KEY (venueID) references Venue) ENGINE=InnoDB;
```

2.7. CheckIn

Relational Model:

check_in (userID,friendID,venueID,checkInID,date)

Functional Dependencies:

checkInID - > userID venueID checkInID date

Candidate Keys:

{ (userID, venueID,photoID) }

Normal Form

BCNF

Table Definition

```
CREATE TABLE check_in(
        userId                          INT NOT NULL,
        venueId                         INT NOT NULL,
        rate                            INT NOT NULL,
        checkInID                       INT NOT NULL,
        PRIMARY KEY(userID, venueID,checkInID),
        FOREIGN KEY (userID) references User,
        FOREIGN KEY (venueID) references Venue) ENGINE=InnoDB;
```

2.8. Comment

Relational Model:

Comment (commentID,message,date)

Functional Dependencies:

commentID - > message date

Candidate Keys:

{ (commentID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Comment(
                commentID               INT PRIMERY KEY,
                message                 VARCHAR(40) NOT NULL,
                date                    DATE NOT NULL) ENGINE=InnoDB;
```

2.9. UserCommentVenue

Relational Model:

user_comments_on_venue(commentID,userID,venueID)

Functional Dependencies:

commentID - > userID venueID

Candidate Keys:

{ (commentID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user_comments_on_venue (
        commentID                   INT NOT NULL,
        userID                          INT NOT NULL,
        venueID                     INT NOT NULL,
       PRIMARY KEY(commentID, userID, venueID),
       FOREIGN KEY (userID) references User,
       FOREIGN KEY (commentID) references Comment,
       FOREIGN KEY (venueID) references Venue,) ENGINE=InnoDB;
```

2.10. VenueOwnerReplies

Relational Model:

venue_owner_replies (commentID,commentUserID,venueID,venueownerUserID)

Functional Dependencies:

commentID -> commentID commentUserID venueID venueownerUserID

Candidate Keys:

{ (commentID) }

Normal Form

BCNF

Table Definition

CREATE TABLE venue_owner_replies (

commentID INT,

commentUserID INT,

venueID INT,

veneueOwnerUserID INT NOT NULL,

PRIMARY KEY(commentID,commentUserID,venueID,venueownerUserID),

FOREIGN KEY (commentUserID) references User,

FOREIGN KEY (venueownerUserID) references VenueOwner,

FOREIGN KEY (commentID) references Comment,

FOREIGN KEY (venueID) references Venue) ENGINE=InnoDB;

2.11. UserCommentsCheckIn

Relational Model:

user_comments_on_checkin (commentID,userID,chekInID)

Functional Dependencies:

commentID - > userID checkInID

Candidate Keys:

{ (commentID) }

Normal Form

BCNF

Table Definition

CREATE TABLE user_comments_on_checkin (

```
commentID       INT,
userID          INT,
checkInID       INT ,
 PRIMARY KEY(commentID, userID,checkInID),
 FOREIGN KEY (userID) references User,
 FOREIGN KEY (commentID) references Comment,
 FOREIGN KEY (checkInID) references check_in) ENGINE=InnoDB;
```

2.12. UserRatesComment
Relational Model:

User_rates_comment (commentID,userID,rate)

Functional Dependencies:

commentID - > userID rate

Candidate Keys:

{ (commentID,userID) }

Normal Form

BCNF

Table Definition

```
CREATE TABLE user_rates_comment (
        commentID           INT NOT NULL,
        userID              INT NOT NULL,
        rate                INT NOT NULL,
        PRIMARY KEY(commentID, userID),
```

```
        FOREIGN KEY (userID) references User,
        FOREIGN KEY (commentID) references Comment) ENGINE=InnoDB;
```

2.13. UserReplies

Relational Model:

user_replies (userID,originalPostCommentID,replyCommentID)

Functional Dependencies:

replyCommentID -> userID originalPostCommentID

Candidate Keys:

{ (replyCommentID) }

Normal Form

BCNF

Table Definition

```
CREATE TABLE user_replies  (
        orginalPostCommentID    INT,
        replyCommentID          INT,
        userID                  INT ,
        PRIMARY KEY(orginalPostCommentID, userID,replyCommentID),
        FOREIGN KEY (userID) references User,
        FOREIGN KEY (orginalPostCommentID) references Comment,
        FOREIGN KEY (replyCommentID ) references Comment) ENGINE=InnoDB;
```

2.14. UserHasFriends

Relational Model:

user_has_friends(useruserID,frienduserID, dateOfFriendShip)

Functional Dependencies:

No functional dependencies

Candidate Keys:

{ (useruserID,frienduserID) }

Normal Form

BCNF

Table Definition

```
CREATE TABLE userFriends (
        useruserID              INT,
        frienduserID            INT,
        dateOfFriendShip    DATE,
        PRIMARY KEY(useruserID , frienduserID ),
        FOREIGN KEY (userID) references User,
        FOREIGN KEY (user, friend) references User ) ENGINE=InnoDB;
```

2.15. UserHasMatched

user_has_matched(useruserID,matcheduserID,approved)

Functional Dependencies:

No functional dependencies.

Candidate Keys:

{ (useruserID, matcheduserID) }

Normal Form

BCNF

Table Definition

```
CREATE TABLE userMatched (
        useruserID                  INT,
        matcheduserID               INT,
        approved                    INT ,
        PRIMARY KEY(useruserID , matcheduserID),
        FOREIGN KEY (useruserID,matcheduserID) references User  ) ENGINE=InnoDB;
```

2.16. Event

Event(<u>eventID</u>, description, date, title, type, <u>venueID, userID</u>)

Functional Dependencies:

eventID -> description, date, title, type, venueID, userID

Candidate Keys:

{ (eventID) }

Normal Form

BCNF

Table Definition

```
CREATE TABLE Event(
        eventID                     INT,
        description                 VARCHAR(40) NOT NULL,
        date                        DATE,
        title                       VARCHAR(40) NOT NULL
        type                        VARCHAR(40) NOT NULL
        PRIMARY KEY(useruserID),
```

FOREIGN KEY (venueID,) references Venue,
FOREIGN KEY (userID,) references User,) ENGINE=InnoDB;

3. FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES

In Relational Schemas part, the normal form of the tables are indicated. Since all relations are in BCNF form, no composition needed.

4. FUNCTIONAL COMPONENTS

4.1. Use Cases/Scenarios

In I-Venue, there are two main end-users: user and venue owner. Each end user has some roles in the system. Even though the system provides common features to all user types, according to the user's type, the system puts some limitations for the corresponding user. This part is to clarify the differences and similarities of the user's roles. First of all, in order to start using the system, both user and venue owner should register and login to the system.

4.1.1 User Use Cases

- Users should be able to register the system.
- User should be able to login with using their username and password or log out the system.
- Users should be able to add profile picture to their profiles.
- Users should be able to add their friends.
- Users should be able to see their friends' profile and see their activities.
- Users should be able to search a place (cafe, restaurants etc.)
- Users should be able to like or dislike a venue.
- Users should be able to comment on a venue.
- Users should be able to reply to the comments.
- Users should be able to check-in to the venue.
- Users should be able to check-in with their friends. (tagging)
- Users should be able to see who else is in the same spot.
- Users should be able to see who is the mayor of the venue.
- Users should be able to see their rank on the leaderboard.
- Users should be able to read reviews about the place.
- Users should be able to add pictures to added places
- Users should be able to rate the venues.

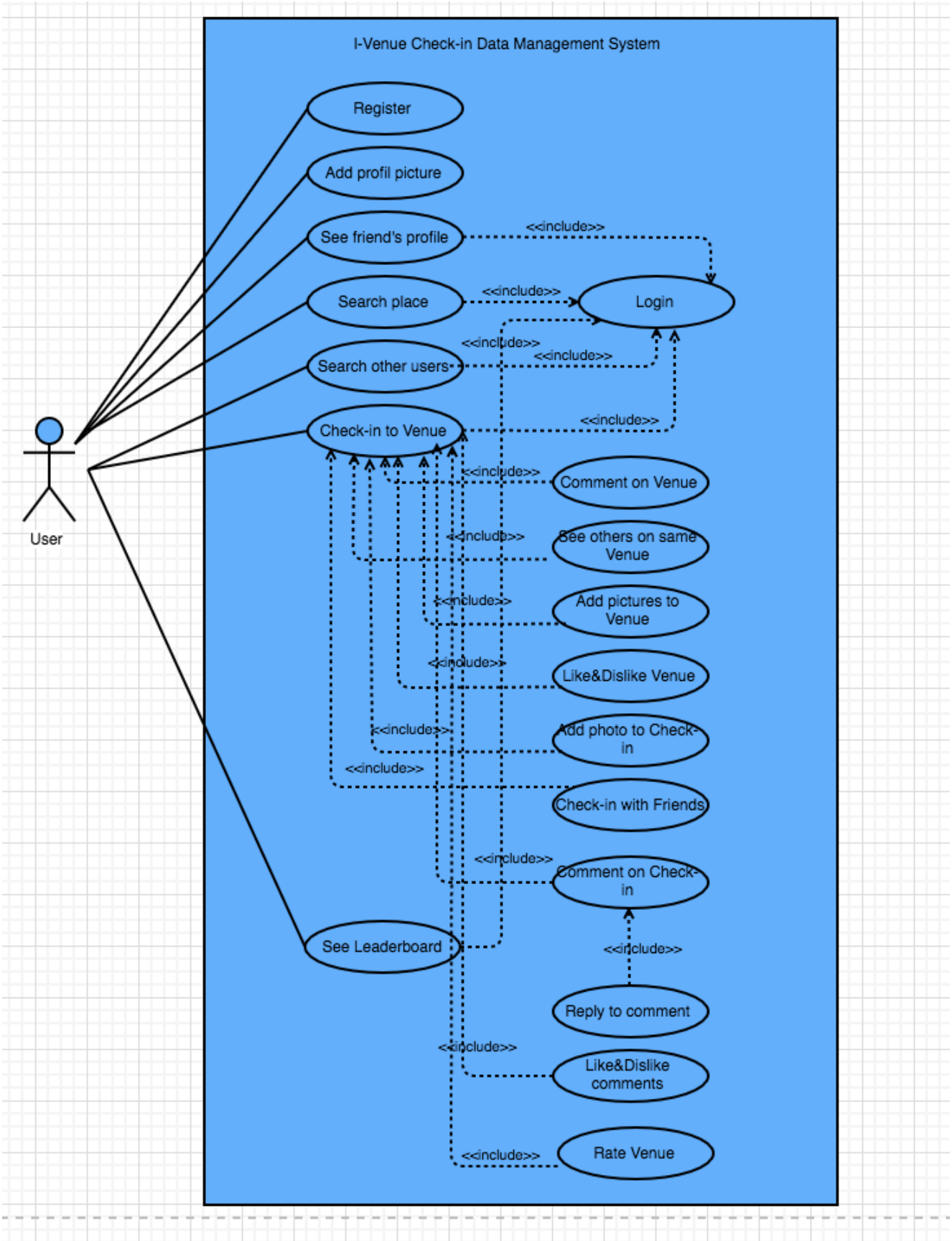

I-Venue Check-in Data Management System
Register
Add profil picture
See friend's profile
Search place
Search other users
Check-in to Venue
See Leaderboard
Login
Comment on Venue
See others on same Venue
Add pictures to Venue
Like&Dislike Venue
Add photo to Check-in
Check-in with Friends
Comment on Check-in
Reply to comment
Like&Dislike comments
Rate Venue
<<include>>
User

Figure 2: User Use Case Diagram

Register: The user can register to the system by specifying his/her unique username, password, name, surname, birthdate, age and gender. The user can use this information to login to the system. The number of check-in and coin balance is set to 0 by default.

Login: The user can login to the system using his/her username and password. Login is required to start using the system.

See Friend's Profile: The user can see his/her friends profile. The user can see their friend's activation and the number of total check-in. The user can see that where their friends check-in. The user can see with whom their friends check-in to the venue.

Search Place: The user can search the places to read comments about the venue. When the user search the place, he/she can only see the check-ins that are done by their friends. The user cannot see the people who are not his friends. The user can see the number of likes and dislikes for the searched place.

Search Other Users: The user can search other users who are registered to system. The user should add the other users to check-in with them.

Check-in to Venue: The user can check-in to the venue. When the user check-in to the venue, his/her total number of check-in increments by one. Also, he/she gets 5 points for each check-in.

Comment on Venue: The user can comment on the venue after she/he checks-in to the venue.

See others on same Venue: The user can see the people who are on the same spot with him/her when she/he checks-in there. Otherwise, the user can only see the people who are their friends before.

Add pictures to Venue: The user can add the pictures to the venue where she/he has check-in there.

Like&Dislike Venue: The user can like or dislike the venue. She/he does not need to comment on the venue to like or dislike there.The number of like and dislike are hold in venue's profile.

Add photo to check-in: The user can add the photos to his/her check-in.

Comment on check-in : The user can comment on his/her friend's check-in. The only requirement is that the user has already checked-in there before.

Check-in with friends: The user can check-in with his/her friends. For each check-in with friends brings bonus coins.

Like&Dislike comment: The user can like or dislike their friend's comments.

Reply to the comment: The user can reply the comments which have done by his/her friends to the user's comment. And, the user can reply the comments which have done to venue.

Rate Venue: The user can rate the venue according to his/her experience. To rate venue, the user has already checked-in there before.

See Leaderboard: The user can see his/her status and his friends' status on the leaderboard. The coins he/she collects help to climb the top of the leaderboard. Also, if the user check-in with his/her friends, or has a mayorship on the venue, he/she will get bonus coins.

4.1.2 Venue Owner Use Cases

- Venue owner should be able to add a place.
- Venue owner should be able to add or edit address and description of the venue.
- Venue owner should be able to reply to user’s comment.
- Venue owner should be able to add pictures to the corresponding venue.
- Venue Owner can delete a Venue to the system.

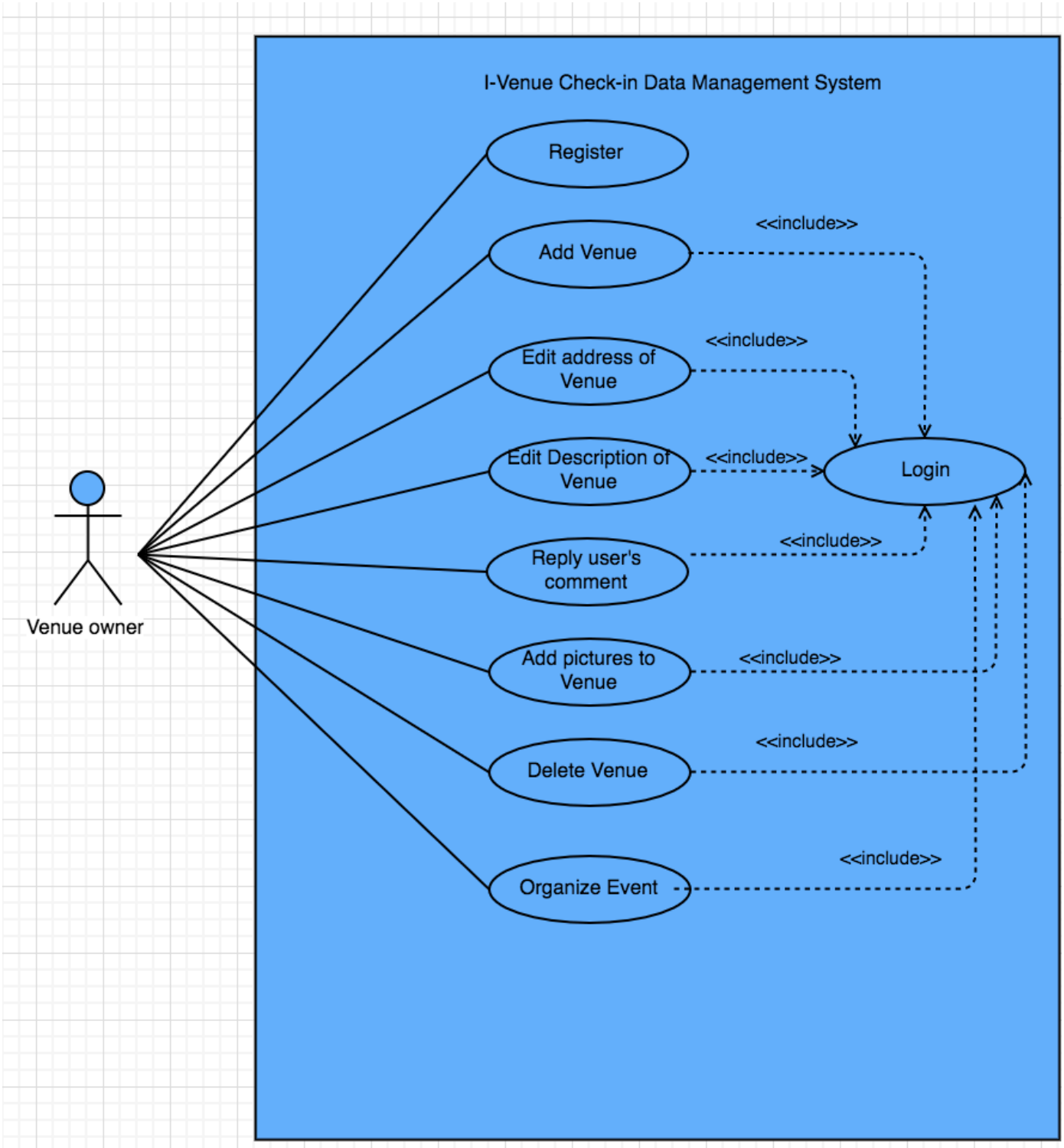

I-Venue Check-in Data Management System
Register
Add Venue
<<include>>
Edit address of Venue
<<include>>
Edit Description of Venue
<<include>>
Login
Reply user's comment
<<include>>
Add pictures to Venue
<<include>>
Delete Venue
<<include>>
Organize Event
<<include>>
Venue owner

Figure 3: Venue Owner Use Case Diagram

Login : Venue owner can login to the system with his/her username and password. The venue owner will be given access to venue owner functions when the account is approved. The venue owner needs to login to the system to be able to perform other operations.

Add Venue: Only the venue owner can add a venue into the system.

Edit the address of Venue: Only the venue owner can edit the address of venue.

Edit the description of Venue: Only the venue owner can edit the description of venue. Description of venue consists of information about the venue.

Reply user's comment: Venue Owner cannot reply a comment if it is not made for his/her Venue. Otherwise, she/he can reply the user's comment.

Add pictures to Venue: Venue Owner can add pictures to the venue.

Delete Venue: Only the venue owner can delete his/her venues to the system. If venue is deleted by venue owner, the check in will be deleted from user's profile.

Organize Event: The venue owner can organize the events in his/her venue. And, venue owner can send invitation message to the users to join his/her event.

4.2. Algorithms

4.2.1. Customer Related Algorithms

The user interacts with the system in order to check-in to the venue. In our system, the balance coin information is kept under User table and this entity is important while determining the top user on the leaderboard.

Each user has coin balance, and it is initialized as 0 coin by default. Each check-in helps the user to earn 5 coins. If the user checks-in with his/her friends, for each friend the user can earn 2 more points. For instance, let's assume that user's coin balance is 100 coins. If the user checks-in to the venue, he/she will earn 5 coins. The coin balance will increase to 105 coins. In other scenario, if the

user checks-in to the venue with his/her 4 more friends, he/she will earn 5 coins by default and 8 more coins. It means that the user's balance will increase to 113 coins.
Likes and dislikes to the comment or venue do not affect coin balance. However, comments on venue and comments on the friend's check-in brings 1 more coins.

For each venue, the system should be able to give mayorship when the user has visited somewhere more often than anyone else, s/he can become the mayor of that place. When she/he becomes the mayor of that place, he/she will earn 50 coins.

According to user's preference, the system can suggest the user to add the friends who have same preference with him/her. In the user_has_matched table, approved attribute is going to be decided according to the calculations involving age, gender parameters of the users. In the final step, the evaluation of the result of the calculation is going to decide whether the other user is going to be suggested as a matched for the corresponding user. If user accepts this match, approved attribute stays same. Otherwise, it becomes -1 and the suggested user is removed from the matched user list of the corresponding user.

4.2.2. Venue Related Algorithms

For each venue, fame information is hold. According to this fame rating, the user can prefer to go to that venue or not.The fame rate depends on total number of check-in.For each 100 checks-in, the venue will earn 0.1 fame rate. Also, each user can rate the venue according to his/her experiences. The user can give maximum 5, minimum 1 to the venue.

4.3. Data Structures

In our relational schemas, String type, Date type and Numeric type of MySQL are used.
String type is used to store character composed information such as nickname, full name, email, message, gender etc.Date type is used to specify birthdates, date of friendship and date of check-in.Attributes with Numeric domain is used to hold numeric data such as age, total number of check-in, balance, total number of visit, total number of comment etc.

5. USER INTERFACE DESIGN AND CORRESPONDING SQL STATEMENTS

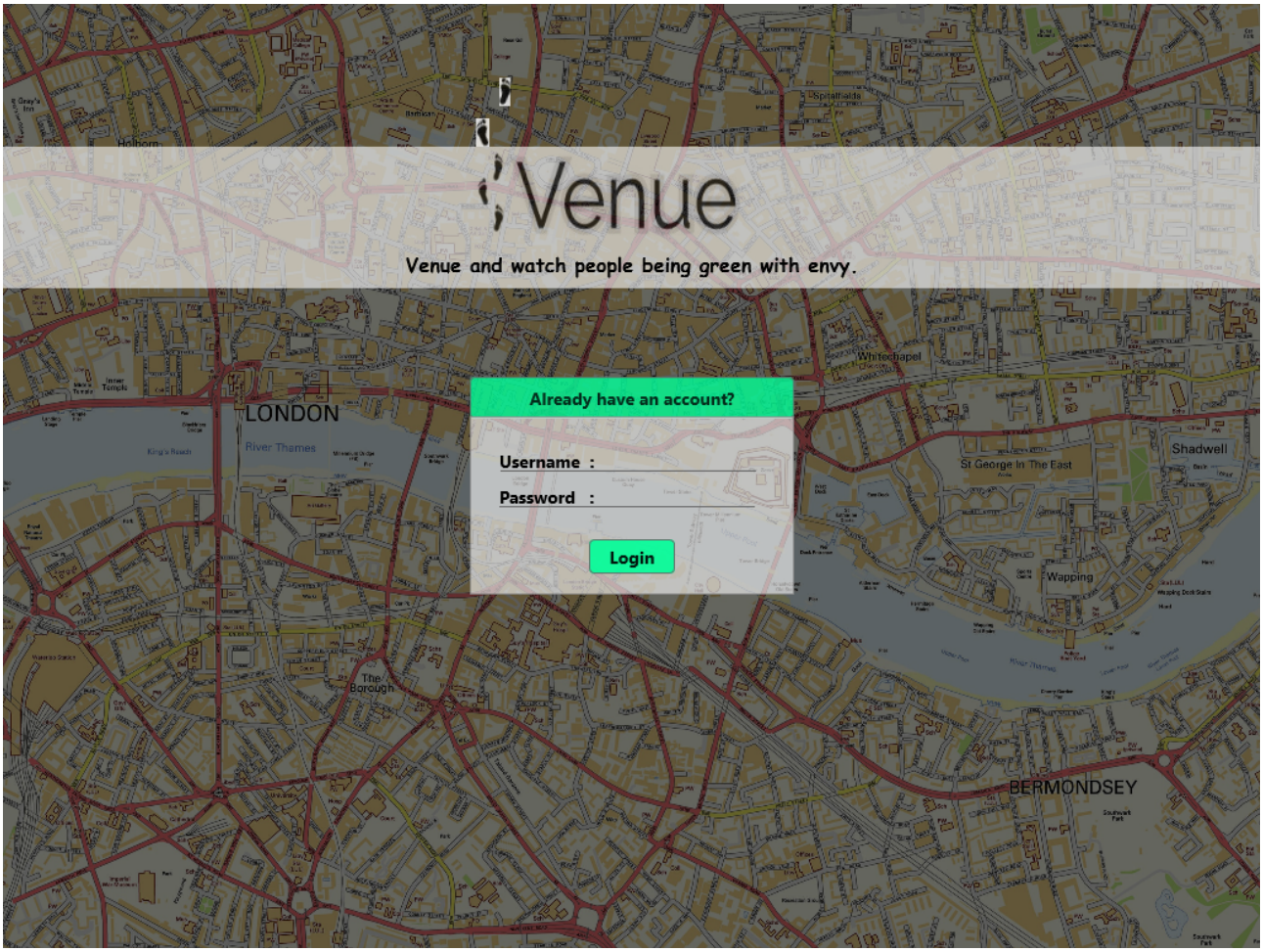
5.1. Login

Inputs: @nickname, @password

Process: User logs into the system by means of entering his/her username and password.

SQL Statements:

Logging in



select nickname, password

from account

where nickname = @nickname AND password = @password

Figure 4: Login Page

5.2. Join

Inputs: @usertype

Process: User first starts with choosing who he/she is in the profile creation process.

SQL Statements:

Logging in

insert into account(usertype)

values (@usertype)

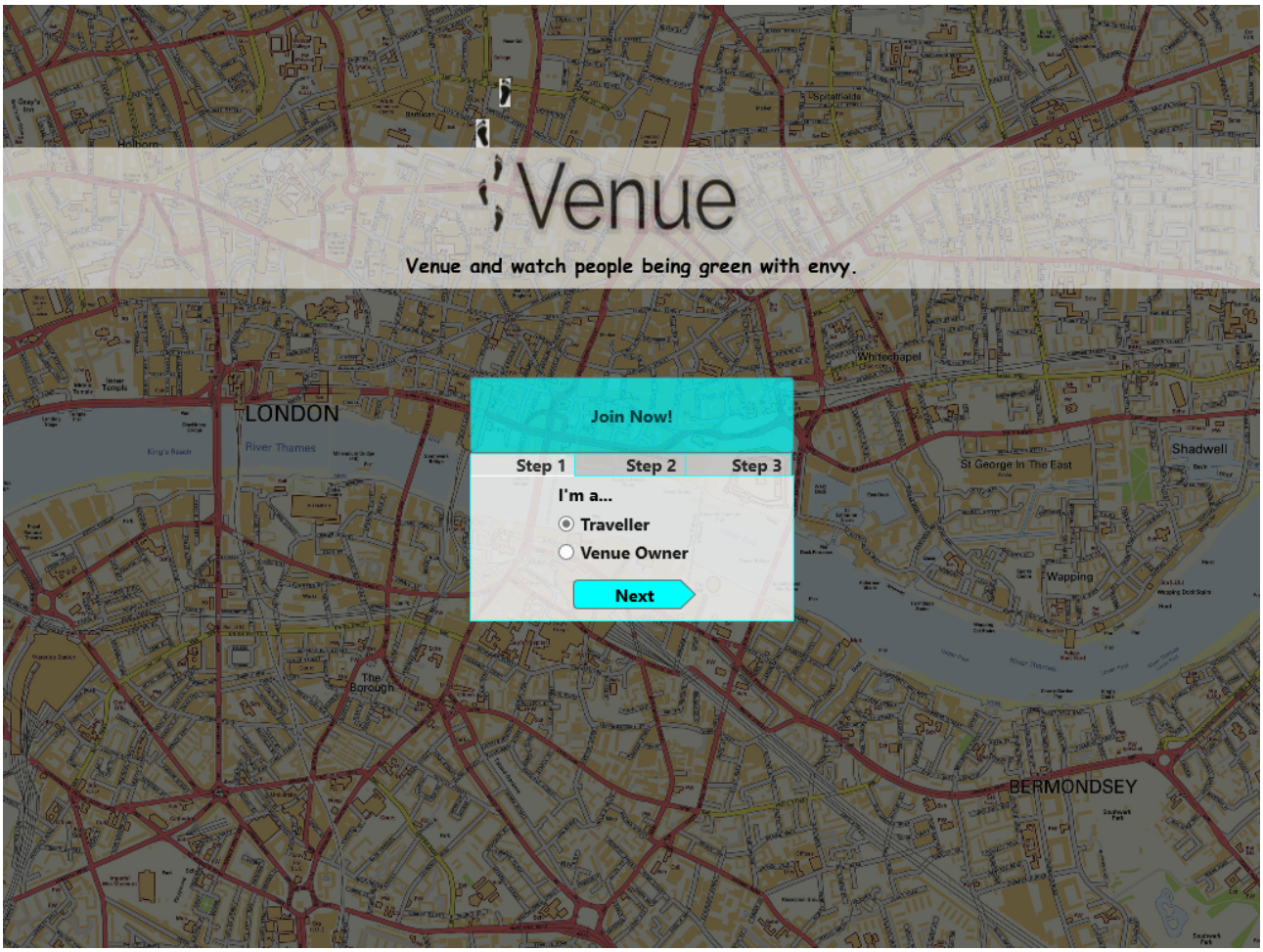


Figure 5: Join Page

5.3. User's Profile Creation

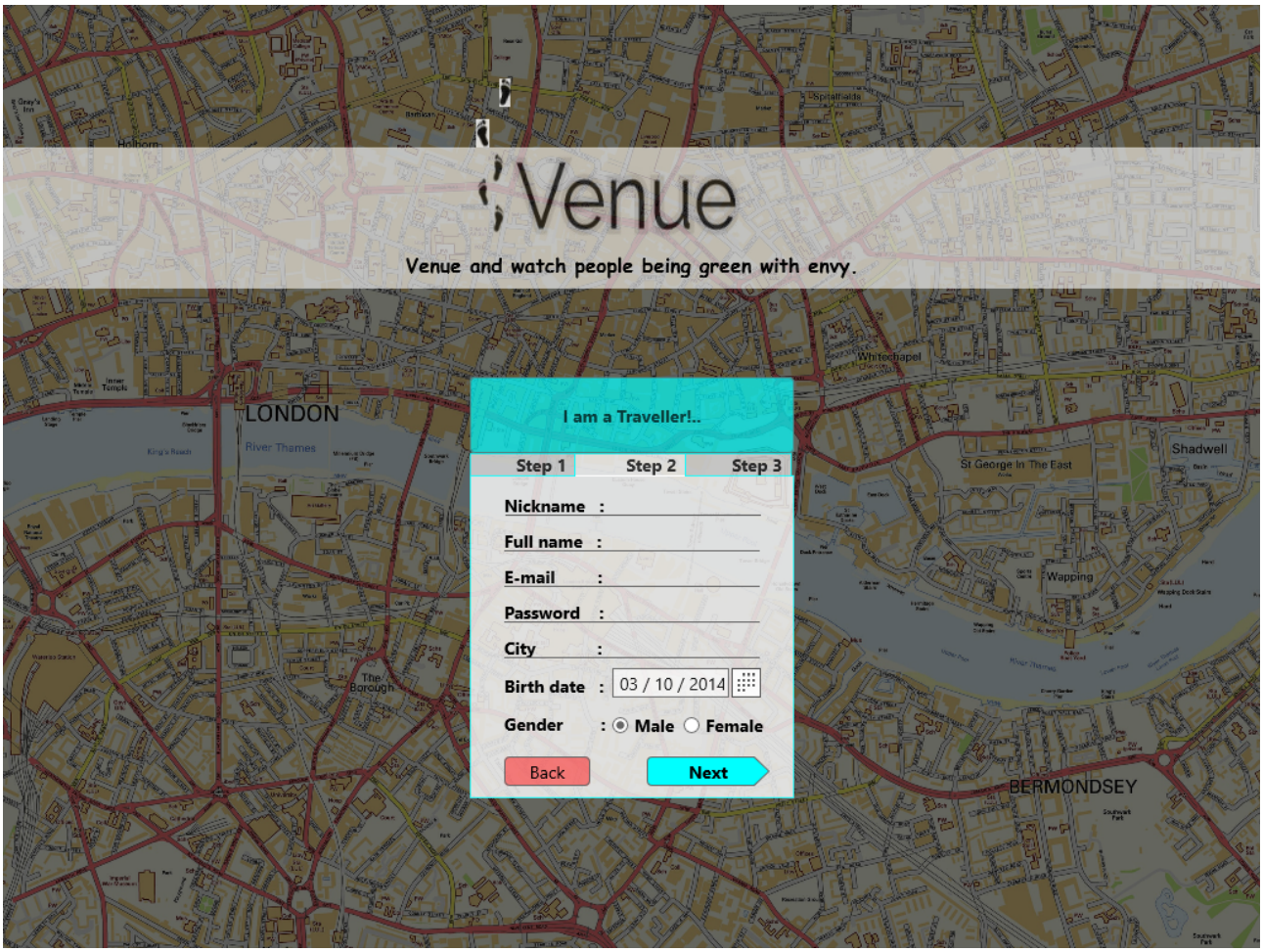
Inputs: @nickname, @fullname, @email, @password, @city, @birthdate, @gender

Process: User continues creating his/her profile by means of entering his/her personal information.

SQL Statements:

Entering information into Account

```
insert into Account( nickname, fullname, email, password, city)
values (@nickname, @fullname, @email, @password, @city)
```



Entering information into User

insert into User(nickname, fullname, email, password, city, birthdate, gender)
values (@nickname, @fullname, @email, @password, @city, @birthdate, @gender)

Figure 6: User's Profile Creation

5.4. Venue Owner's Profile Creation

Inputs: @nickname, @fullname, @email, @password, @city, @experience

Process: Venue owner continues creating his/her profile by means of entering his/her personal information.

SQL Statements:

Entering information into Account

insert into Account(nickname, fullname, email, password, city)

values (@nickname, @fullname, @email, @password, @city)

Entering information into VenueOwner

insert into Venue_Owner(nickname, fullname, email, password, city, experience)
values (@nickname, @fullname, @email, @password, @city, @experience)

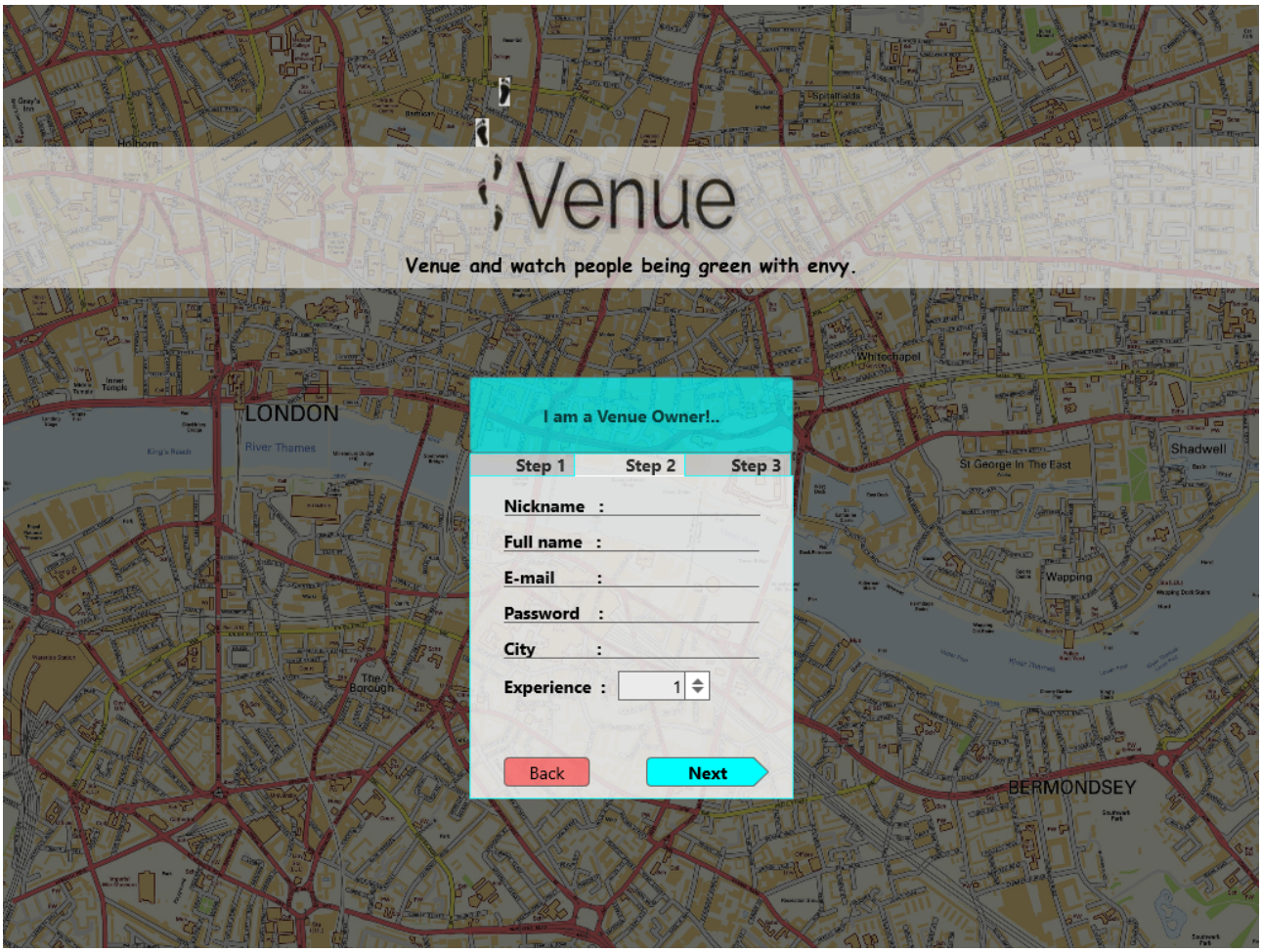


Figure 7: Venue Owner's Profile

5.5. Adding Profile Photo to Account

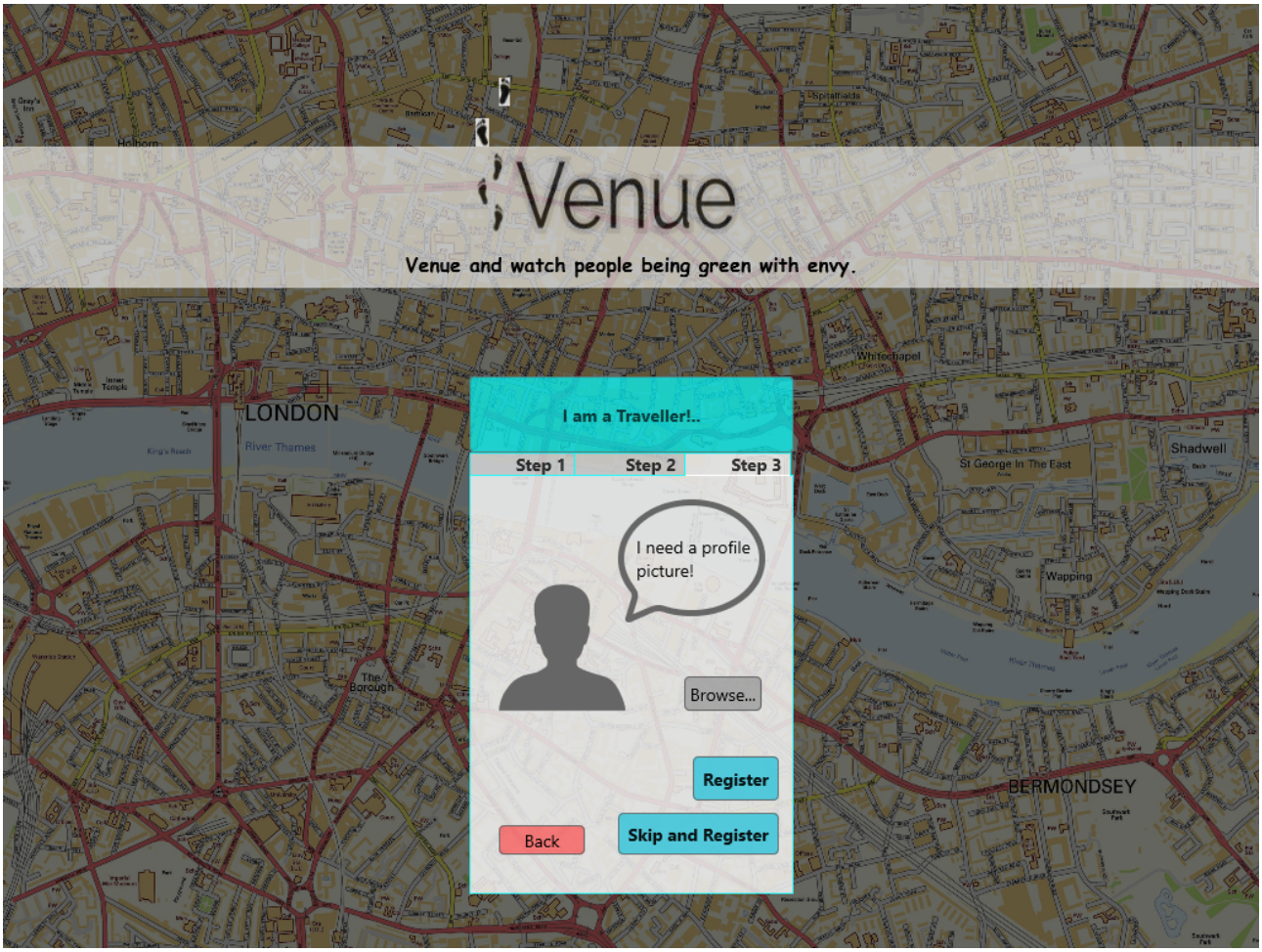
Inputs: @photo

Process: Account owners are adding profile photos to their profiles.

SQL Statements:

Adding binary information into Photo

insert into Photo(photoBinary)



values (@photo)

Figure 8: Adding Profile Photo to Account

5.6. User's Social Feed
Inputs: @userID

Process: User is able to see his/her social feed which shows his/her friends' activities.

SQL Statements:

Showing User's friends' activities

select nickname, checkinID, venuename, description
from (user_has_friends joins check_in using(useruserID)) natural join Venue
where userID = @userID

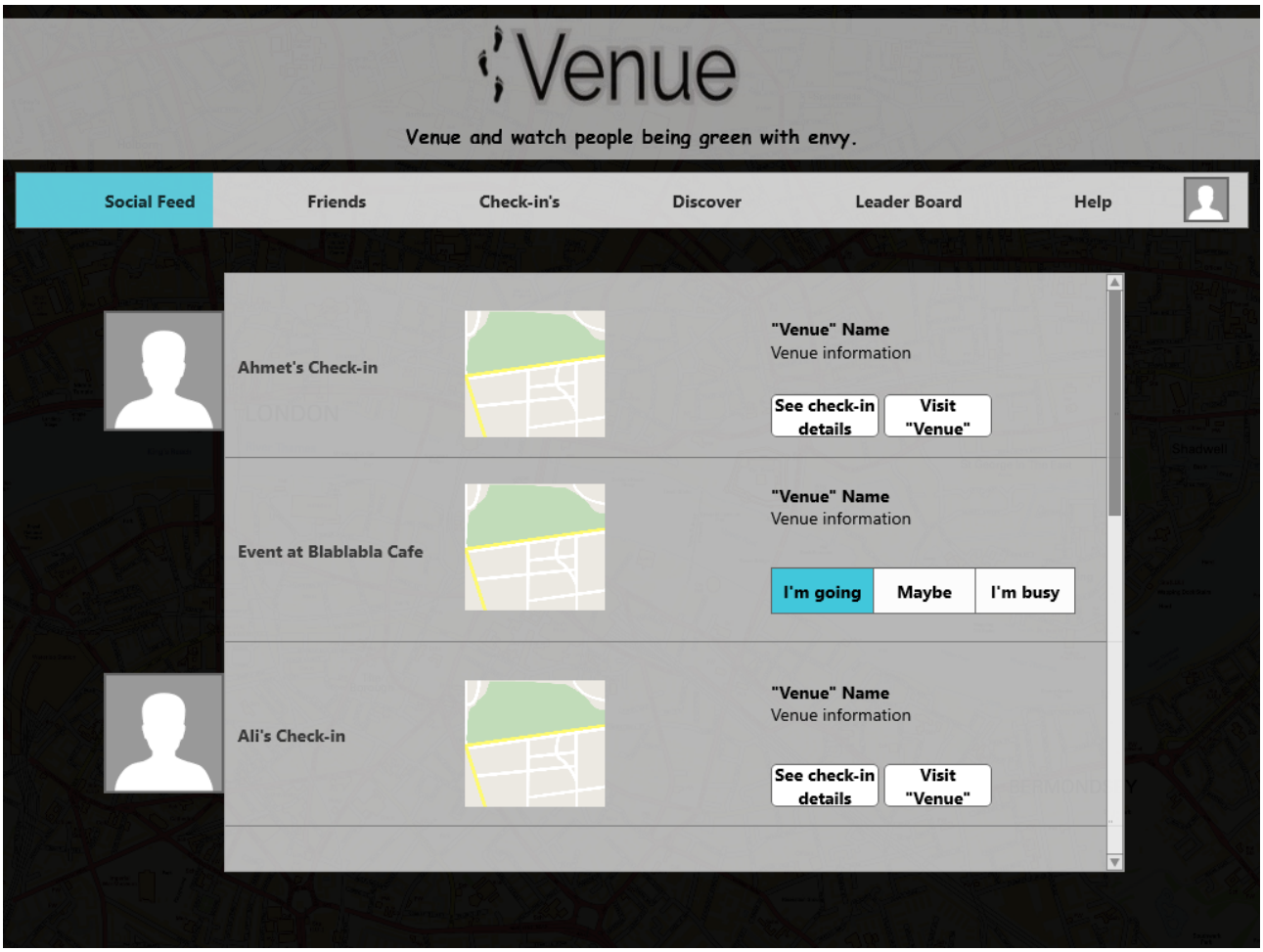


Figure 9: Social Feed Page

5.7. User Edit Profile

Inputs: @fullname, @email, @password, @city, @birthdate, @gender

Process: User edits his/her account settings. nickname cannot be changed.

SQL Statements:

User Edit Profile

update User
set fullname = @fullname, email = @email,

password = @password, city= @city, birthdate = @birthdate, gender = @gender
where userID = @userID and (@fullname is not null or @email is not null or @password is not null,
@city is not null or @birthdate is not null or @gender is not null)

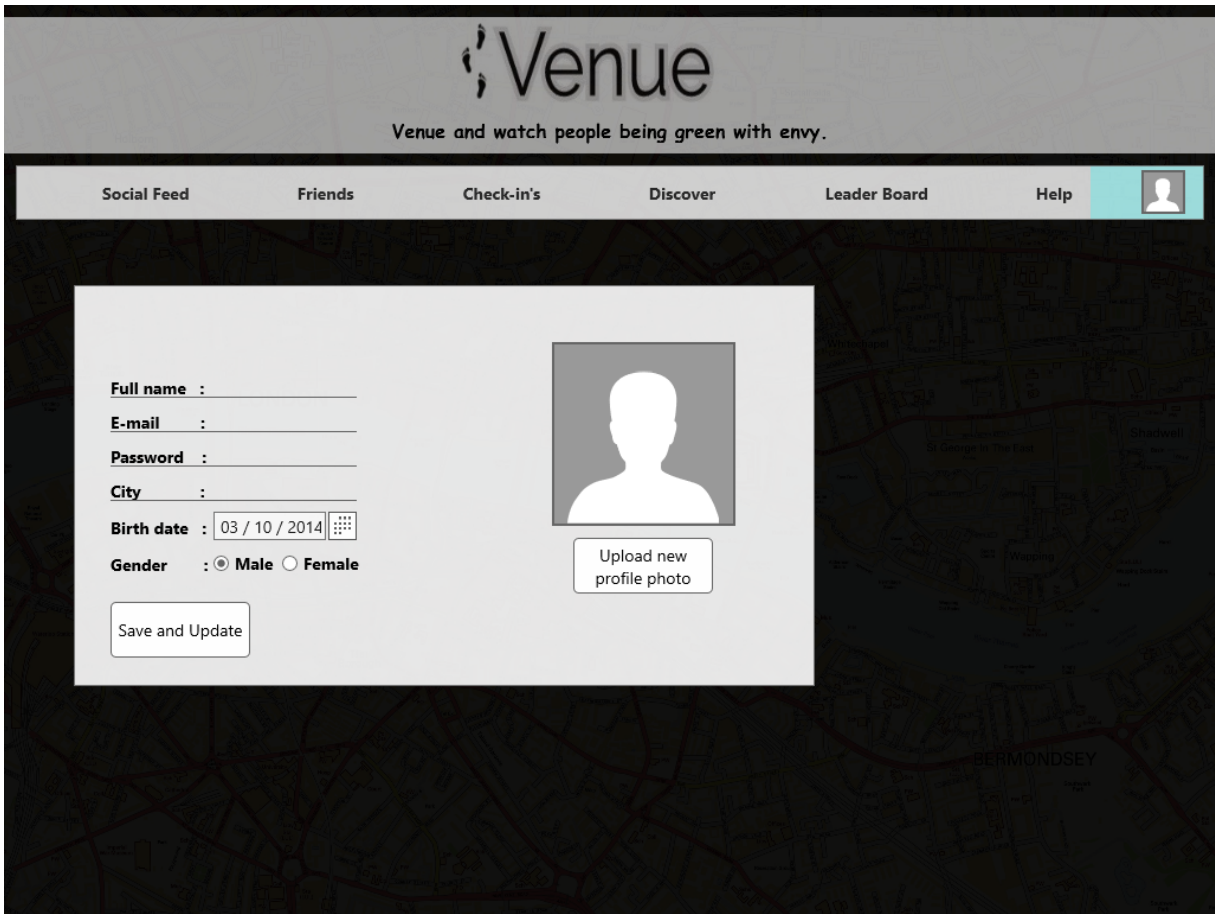


Figure 10: User Edit Profile

5.8. Venue Owner Edit Profile

Inputs: @fullname, @email, @password, @city, @experience,

Process: VenueOwner edits his/her account settings. nickname cannot be changed.

SQL Statements:

User Edit Profile

update VenueOwner
set fullname = @fullname, email = @email,
password = @password, city= @city, experience = @experience

where userID = @userID and (@fullname is not null or @email is not null or @password is not null,

@city is not null or @experience is not null)

5.9. User's Discovery

Inputs: @venuename OR @nickname

Process: User is able to query a venue or a user.

SQL Statements:
Querying a venue
select venuename, picID
from account_adds_photo_to_venue natural join Venue
where venuename = @venuename

Querying a user
select nickname, profilePicture
from Account
where nickname = @nickname

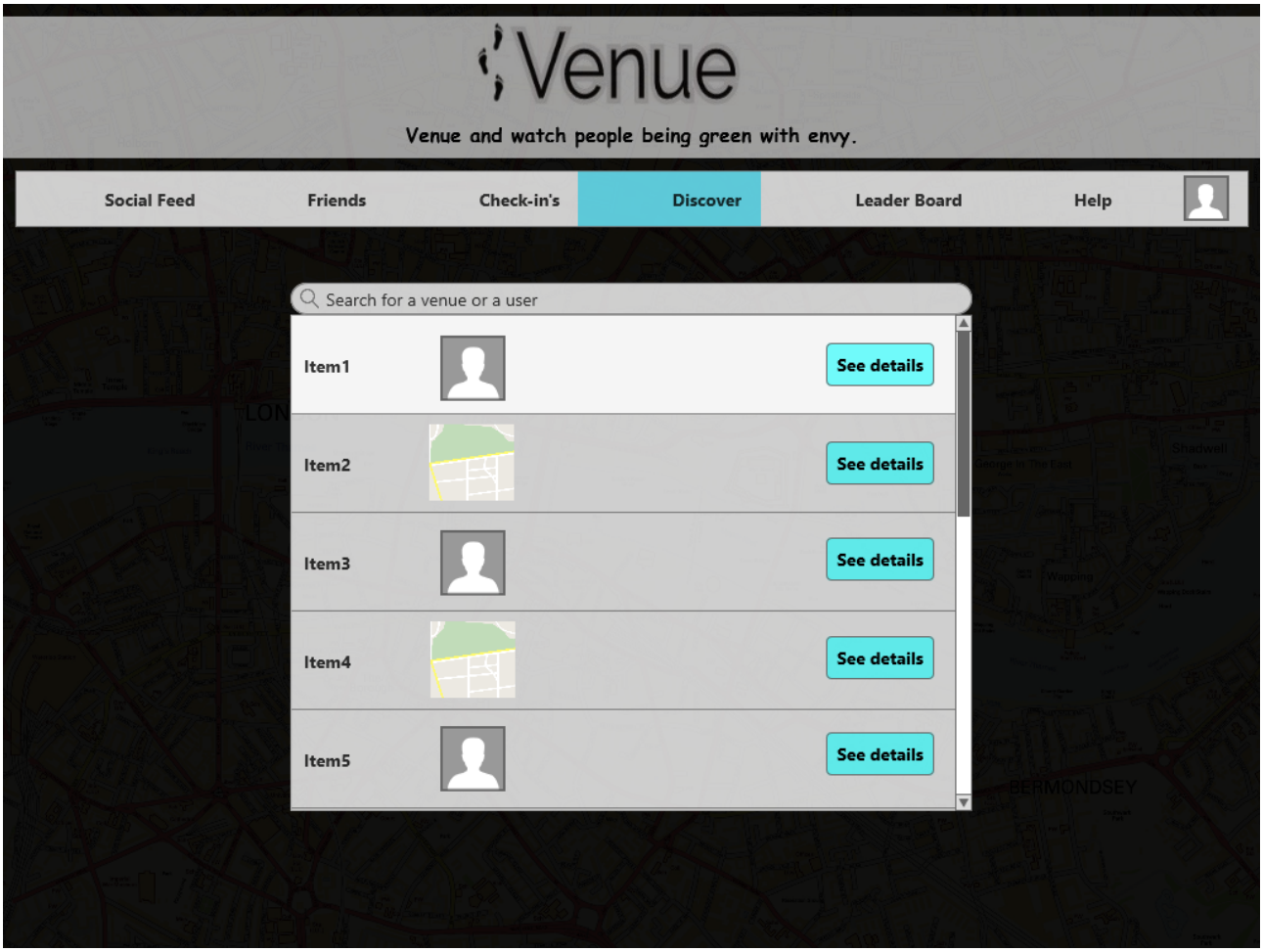


Figure 11: User's Discovery

5.10. Venue Owner adds Venue

Inputs: @venuename, @venuetype, @city, @streetname, @zipcode, @latitude, @longitude, @description

Process: Venue owner can add another venue of him/herself.

SQL Statements:

Adding a new venue

insert into Venue(venuename, venuetype, city, streetname, zipcode, latitude, longitude, description) values (@venuename, @venuetype, @city, @streetname, @zipcode, @latitude, @longitude, @description)

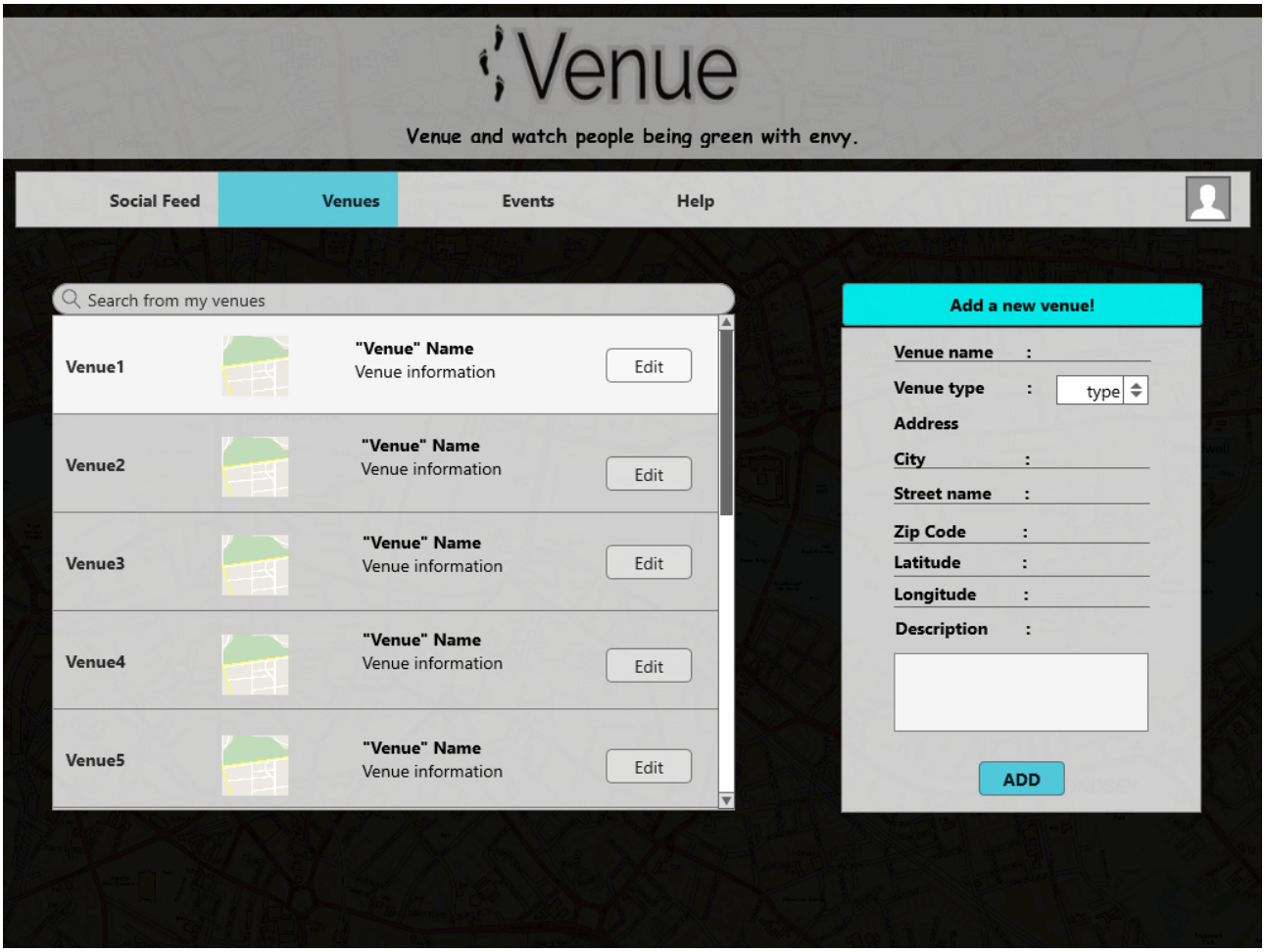


Figure 12 : Venue Owner Adds Venue

5.11. Venue Owner Adds Event

Inputs: @description, @date, @title, @type, @userID, @venueID

Process: Venue owner can add a Event into Venue.

SQL Statements:

insert into Event
values(eventID, @description, @date, @title, @type, @userID, @venueID)

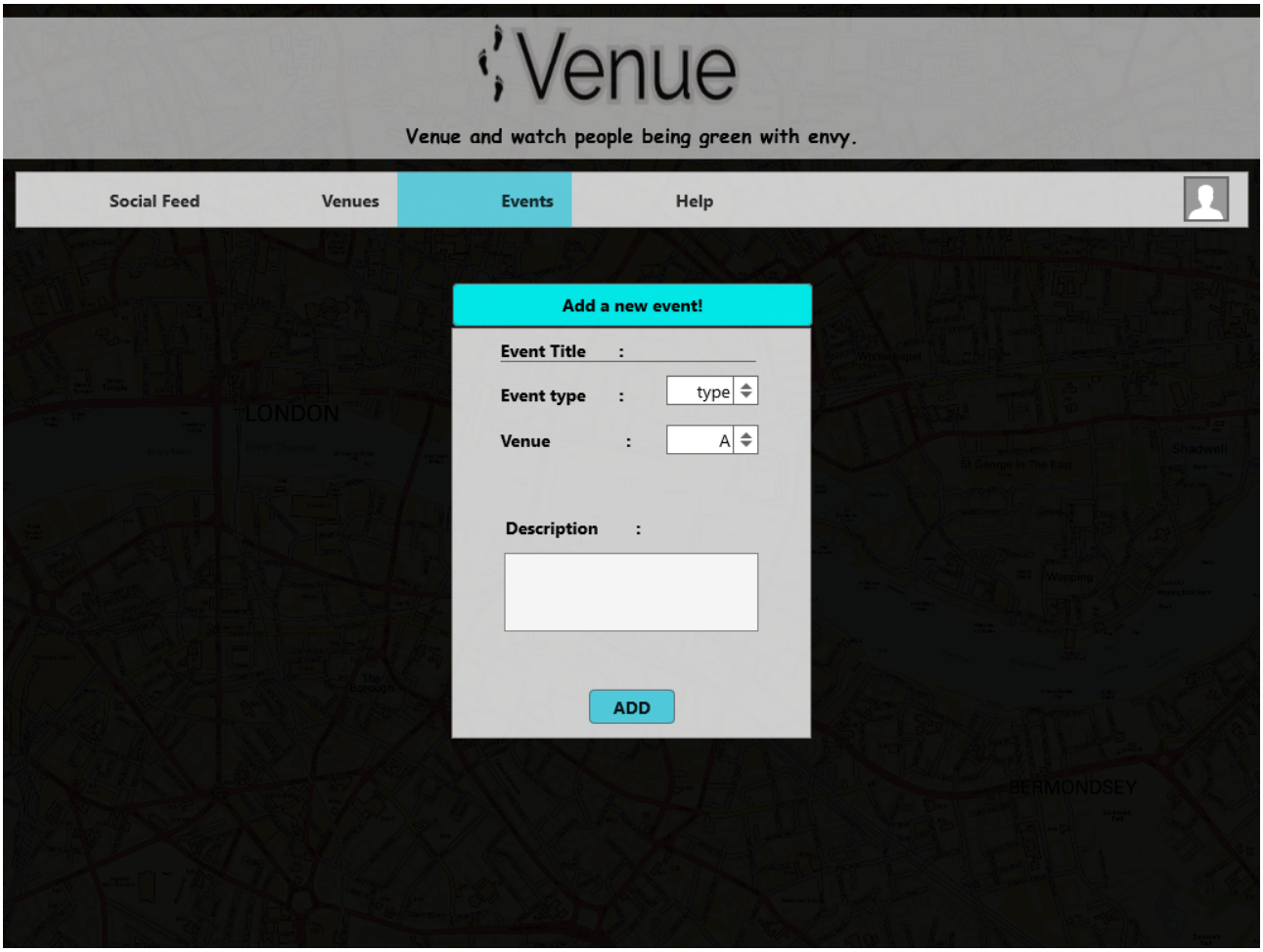


Figure 13: Venue Owner Adds Event

5.12. User's Leader Board

Inputs: @userID

Process: User is able to see a leader board indicating the coin balance of him/herself with his/her friends.

SQL Statements:

Listing leader board

```
select fullname, coinbalance
from User joins user_has_friends using(frienduserID)
where useruserID = @userID
group by fullname, coinbalance desc
```

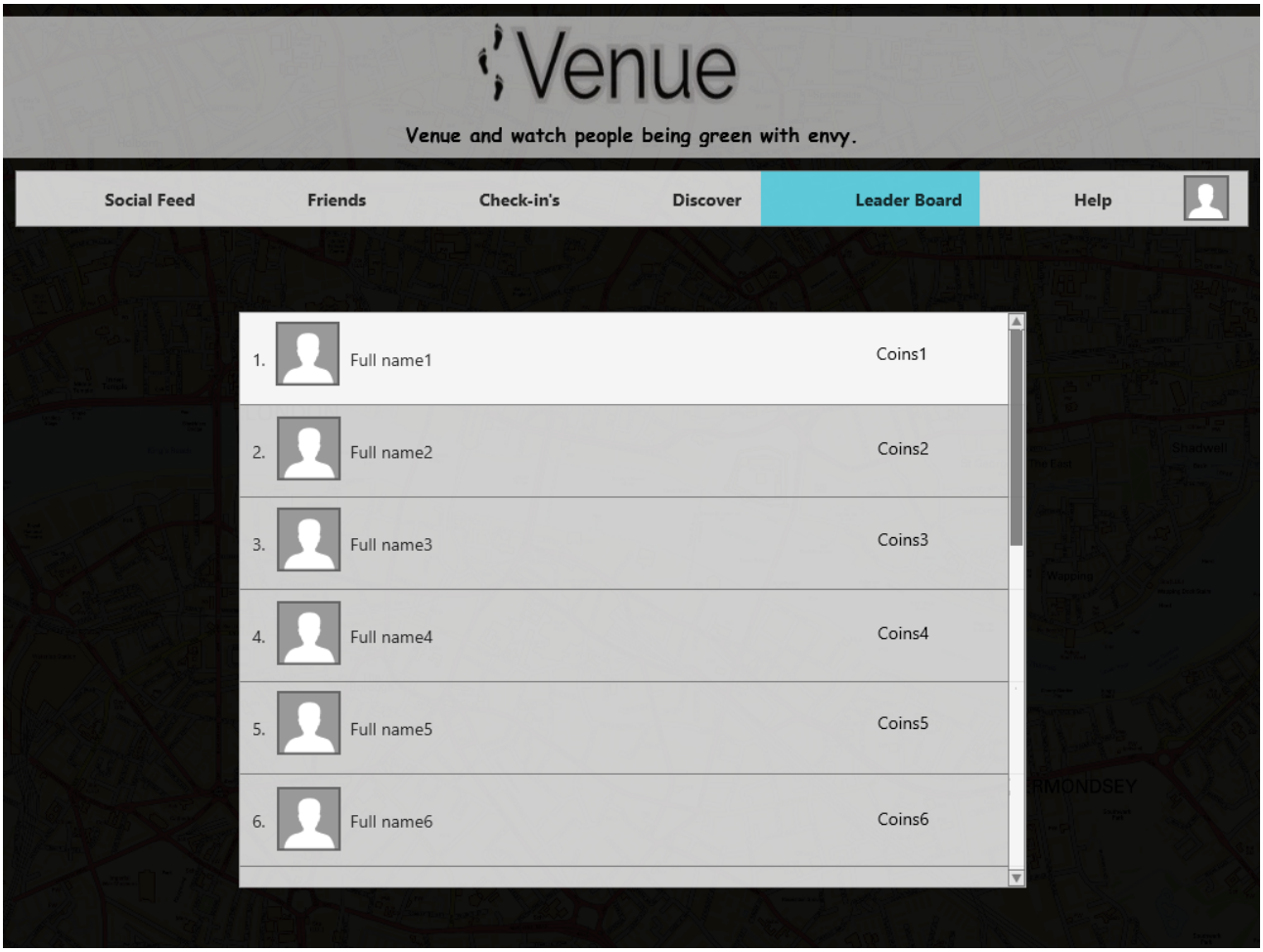


Figure 14: User's Leaderboard Page

5.13. User Comments on Check-in

Inputs: @message, @userID, @checkinID, @venueID

Process: User is able to comment on a check in which he/she has already been the place that check-in is referring to.

SQL Statements:

Comments on check-in

insert into Comment(message)
values (@message)

insert into comment_on_check_in(userID, checkinID, venueID)
values (@userID, @checkinID, @venueID)

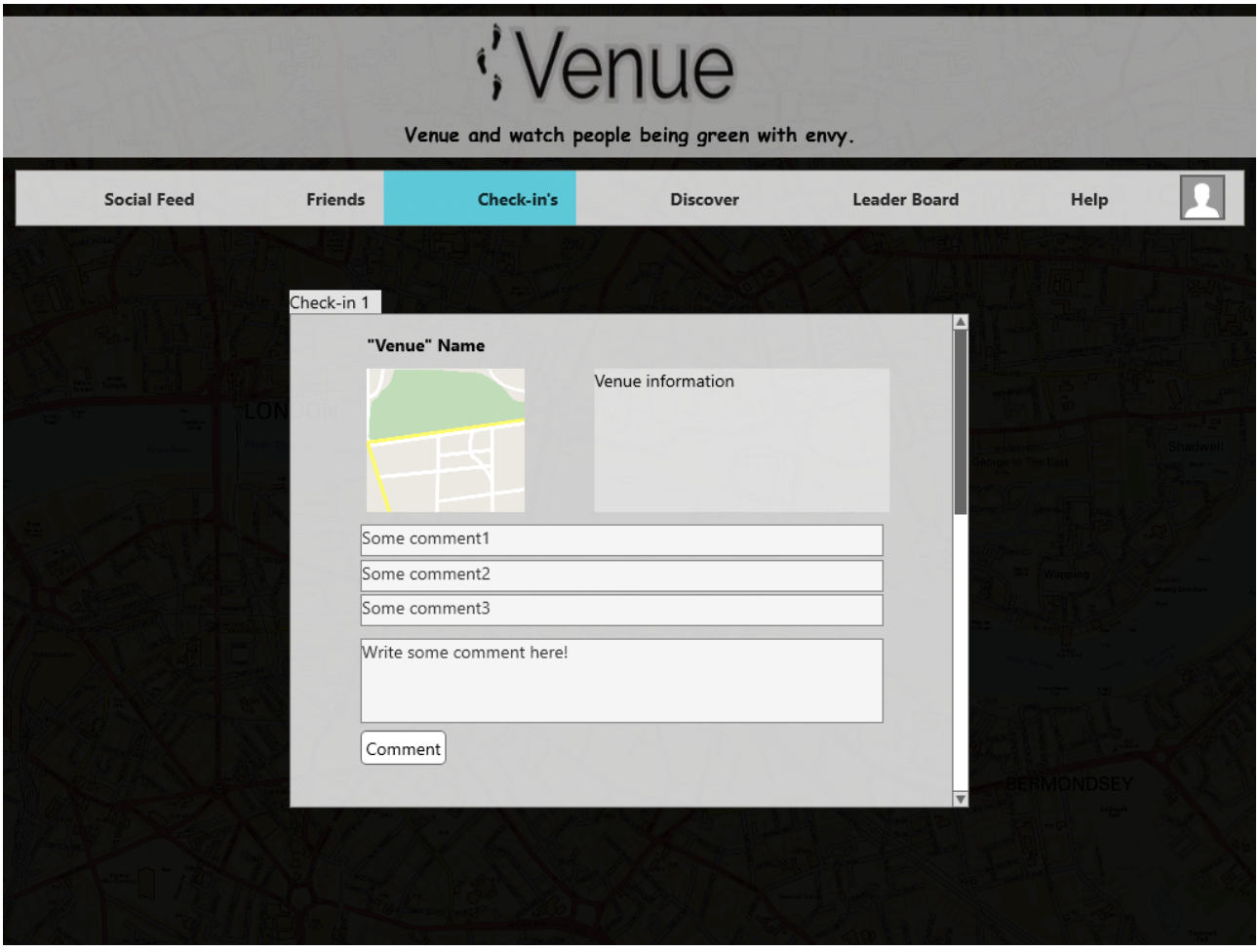


Figure 15: User Comments on Check-in

5.14.Venue Owner's Venues

Inputs: @userID

Process: Venue owner is able to see his/her venues as a list.

SQL Statements:
Listing all venues of a venue owner
select venuename, picID
from Venue joins account_adds_photo_to_venue using(venueID)
where userID = @userID

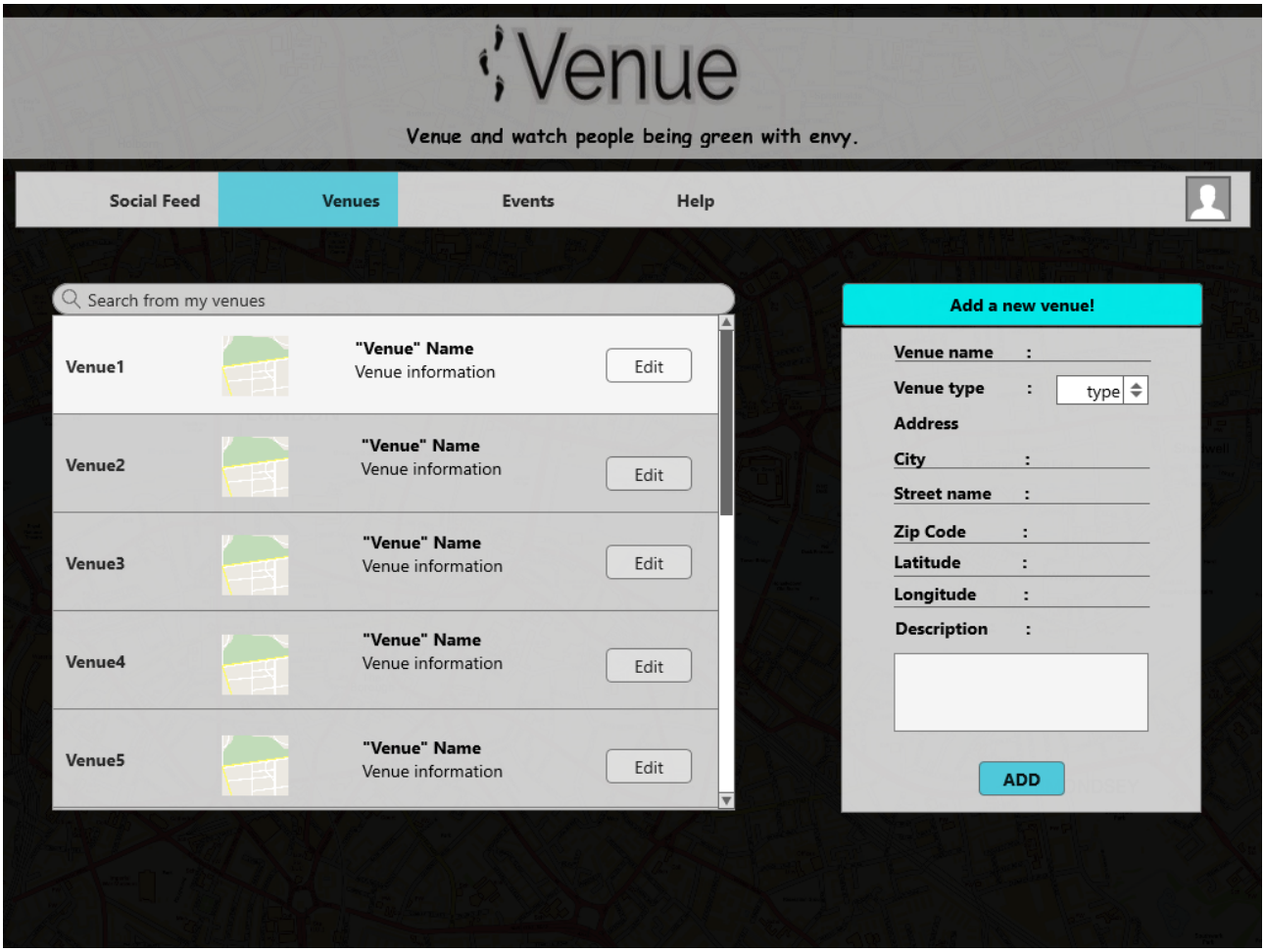


Figure 16: Venue Owner's Venue

6. ADVANCED DATABASE COMPONENTS

6.1. Reports

6.1.1. User Searchs Another User

```
/*  User can search by nickname or fullname.
        @nickname and @fullname are entered by User who are searching.
        */
with searched_users(nickname, fullname, profilePicture) as
        (select *
        from User
        where
        case
        when @nickname is not null then nickname = @nickname
        else fullname = @fullname
        end case
        )
select nickname, fullname, profilePicture
from searched_users
group by nickname
```

6.1.2. People who are in certain age group, in certain gender and specific Venue

People who are in age between x to y and gender z and checked-in in specific Venue. This kind of things can be used when we give recommendations or do matching. As an example; if user age of 21 and a male then he might get a matching like; x = 18, y = 25, z = “female” and venue is visited by both of them at least 5 times. *user_has_matched* will work similar to this.

```
/*  @x, @y, @z, @m are parameters we decide.
        @myID is userID of current User.
        */
with venues_visited_m_times(venueID) as
        (select venueID
```

```
        from check_in
        where userID = @myID
        group by userID, venueID
        having  count(*) > @m
        )
with user_been_in_venue_m_times(userID) as
        (select userID
        from check_in C, venues_visited_m_times VV
        where C.venueID = VV.venueID
        group by userID
        having count(*) > @m
        )
select UB.nickname, UB.fullname, UB.profilePicture
from user_been_in_venue_m_times UB, User U
where UB.userID = U.userID and (@x <= UB.birtdate)
  and (UB.birtdate <= @y)
  and gender = @z
```

6.1.3. Venues Are Categorised According to Age Group

Venues visited by User's age group. This might be helpful when we advise new places for User.

```
/*  @mybirthdate is birthdate of current User.
        */
with avg_age_of_venue(*) as
        (select venueID, avg(birthdate)
        from (select C.venueID, U.birthdate
        from check_in C, User U, Venue V
        where C.venueID = V.venueID and C.userID = U.userID
        group by C.venueID, U.birthdate
        )
        group by venueID
        )
```

```
select V.venuename, V.address, V.likeCount, V.dislikeCount,
                V.description, V.totalVisit, V.totalComment
from Venue V, avg_age_of_venue A
where V.venueID = A.venueID and (A.birthdate - @mybirthdate) between 5 and -5
```

6.2. Views

6.2.1. User's Non-friend User View

User cannot see full profile of a non-friend User. Thus, when a User search another User, if they are not friend only *nickname, fullname* and *profilePicture* will be displayed.

```
/*  User can search by nickname or fullname.
        @nickname and @fullname are entered by User who are searching.
        @myID is userID of User who are searching.
        */
with searched_users (*) as
        (select *
        from User
        where
        case
        when @nickname is not null then nickname = @nickname
        else fullname = @fullname
        end case
        )

with my_friends (*) as
        (select *
        from user_has_friends
        where useruserID = @myID
        )
/* If a non-friend has been searched. */
create view show_non_friend_profile as
        select nickname, fullname, profilePicture
```

```
        from my_friends M, searched_users S
        where M.frienduserID <> S.userID
```

6.2.2. User's Friend User View6.3. Triggers

When User search another User, if they are friends then user can be able to see all of the information about his/her friend, however, *userID, password,* will be hidden as well.

```
/*  User can search by nickname or fullname.
        @nickname and @fullname are entered by User who are searching.
        @myID is userID of User who are searching.
        */
with searched_users (*) as
        (select *
        from User
        where
        case
        when @nickname is not null then nickname = @nickname
        else fullname = @fullname
        end case
        )

with my_friends (*) as
        (select *
        from user_has_friends
        where useruserID = @myID
        )

/* If a friend has been searched. */
create view show_friend_profile as
        select nickname, fullname, email, userType, city, commentNumber, balance
        profilePicture, age, birthdate, gender, totalCheckIn, dateOfCreation
        from my_friends M, searched_users S
```

```
        where M.frienduserID = S.userID
```

6.2.3. User's Venue View

When User try to see popular Venues, he/she should be able to only ones in his/her city.

```
/*  User can search popular Venues.
        @mycity is city of User who are searching.
        */
create view show_popular_venues as
        select V.venuename, V.address, V.likeCount, V.dislikeCount,
        V.description, V.totalVisit, V.totalComment
        from Venue V
        where V.city = @mycity
            order by V.totalVisit desc
```

6.3. Triggers

- *coinbalance* is a point system. Each User can gain points when they check-in, rate or make a comment. Each action have its own multiplier and different weight. *coinbalance* will be changed when User do stated actions.

- When two user had been matched, *approved* will be automatically set to true.

experience has been increased according to *dateOfCreation* of Venue Owner. It will be affected number of venues and total check-in in Venue Owner's places and some other possible parameters. A trigger will be implemented to deal with this.

6.4. Constraints

- An account cannot use site without logging in.
- An account cannot take a nickname which is taken earlier by other account.
- An account can only add a picture in its profile one at a time.
- Only User can check-in.

- Only User can like or dislike.
- User can only check-in with his/her friends.
- User can only add photo a Venue if user check-in before in that Venue.
- User can only make comment a Venue if user check-in before in that Venue.
- User can only like or dislike a Venue if user check-in before in that Venue.
- User can only see his/her friends full profile(addition to normal info; *city, birthdate, gender* etc.) and activities(check-in).
- User can either like or dislike a Venue once.
- User can either like or dislike a Comment once
- Venue Owner cannot reply a comment if it is not made for his/her Venue.
- Only Venue Owner can add or delete a Venue to the system.
- Only Venue Owner can change Venue address and description.
- If the user deletes his/her account, his/her check-in will be displayed in venues' profile. However, if venue is deleted by venue owner, the check in will be deleted from user's profile.

6.5. Stored Procedures

- When a User comments on Venue, multiple tasks are happening at the same time. A new tuple is added to Comment table. User gain some points, thus, *balance* is increased. *experience* of Venue Owner is increased. This produce is repeated whenever a new comment has been made to Venue, which is why we prefer this to be a stored procedure.

- When User rate a Venue, multiple tasks are happening at the same time. A new tuple is added to *user_rates_venue* table. User gain some points, thus, *coinbalance* is increased. *experience* of Venue Owner is increased. This produce is repeated whenever User rates a Venue, which is why we prefer this to be a stored procedure.

- When User rate a Comment, multiple tasks are happening at the same time. A new tuple is added to *user_rates_comment* table. User gain some points, thus, coin*balance* is increased. This produce is repeated whenever User rates a Comment, which is why we prefer this to be a stored procedure.

- When User check-in in a Venue, multiple tasks are happening at the same time. A new tuple is added to *check_in* table. User gain some points, thus, *coinbalance* is increased. *totalCheckIn* of User is increased. If User check-in with some friends, other friends *coinbalance* areincreased as well. *experience* of Venue Owner is increased according to how many people check-in with. This produce is repeated whenever User check-in in a Venue, which is why we prefer this to be a stored procedure.

- When User search another User, same queries are used over and over again. A procedure will be used instead of writing multiple times. User can search among his/her friends or non-friends or can search among all User. Necessary procedures will be used.

7.0 WebSite

The following is the link to our CS353 Database Systems Project:

https://github.com/sedagulkesen/I-Venue-CS353.git

8.0 Implementation Plan

At data layer we use MySQL server in our project as database management system. For backend purposes we will code in JSP and small amount of HTML.