Bilkent University

Department of Computer Engineering

CS 353-Database Management Systems

I-Venue

Group #27

Final Report

May. 14, 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.

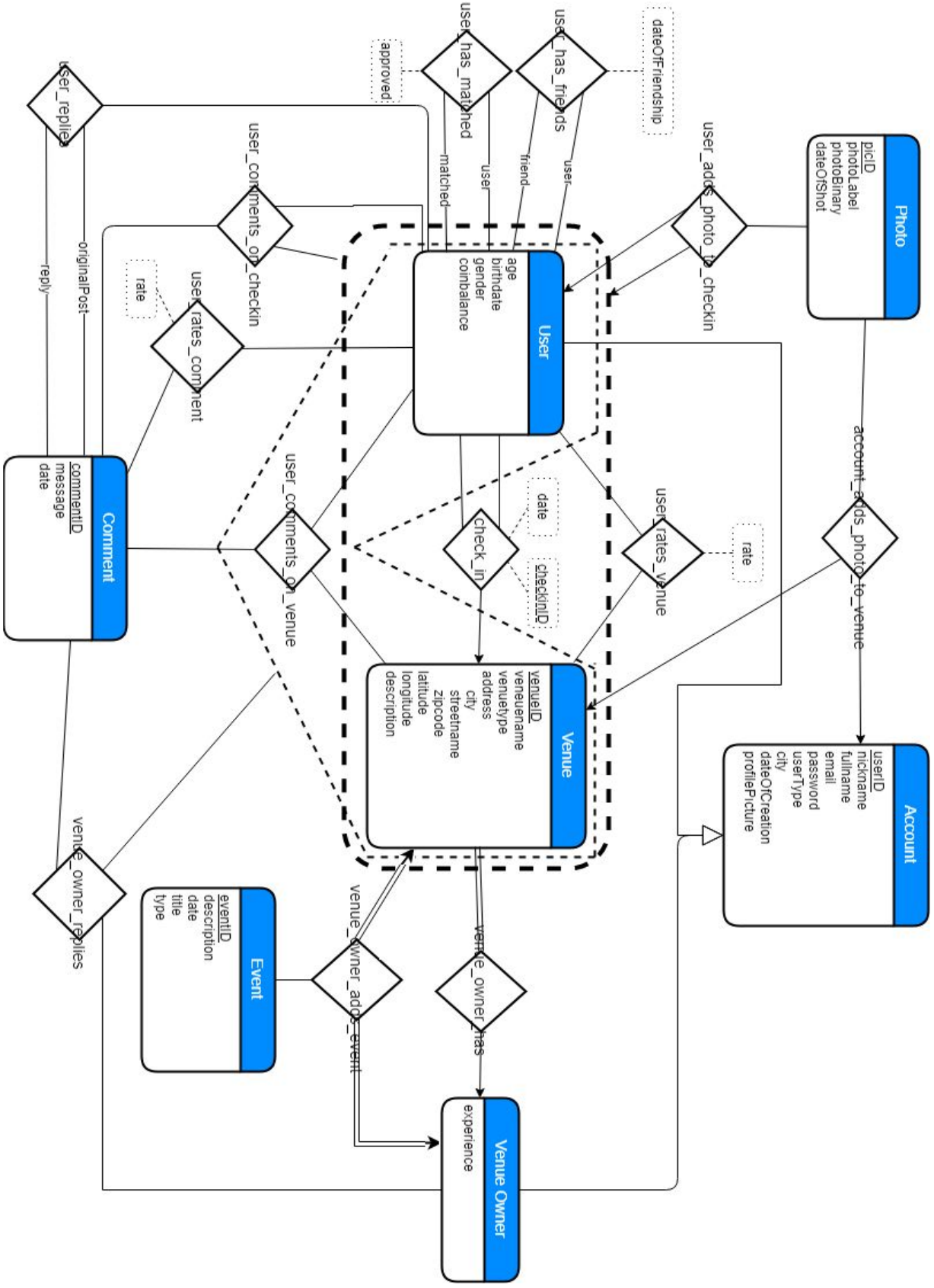
Project Description

I-venue is a web-based social platform for being in communication between friends and sharing online check in to a venue with those. There are two types of users in this platform that are distinguished with their roles in the system. The first type of users on the system is the venue owners who are able to create a venue profile (or multiple numbers of venues) in which the other type of the users, clients (we will simply call them users), are going to be able to check in to that venue. Therefore, it is crucial for the system that the venue owners should create their venue's profiles at first in order to make them checkable by the other users. With this system, the user is going to be able to check-in to a place so that their friends will be able to see his/her check-in, to comment on that place or check-in. Thus, our application allows you to see your friend's activities in general. Another function is that our application provides the users to see a venue's profile by means of making a search with the venue's name or, with querying the venues regarding their types such as breakfast, dinner, fast food etc.Our purpose of implementing this application is to establish an online platform in which the users will be able to share their activities, see their friends' activities, search for a particular venue to see its description and comments that have been shared by other users. This social platform will be a place for social people to share their locations with others to express their feeling about that venue and to get accompanies. Our implementation has two aspects: database and website. The workload is is aimed to be as fair as possible. Thus, each team member has implemented database and user interface part of the project. Below workload distributions can be found per team member throughout the project:

Kayacan Kaya: Implemented the necessary front-end and back-end functions for Traveller and Venue owner edit profile pages, and traveller's friends page.
Seda Gülkesen: Implemented the necessary database queries for Traveller Social Feed and Traveller Check-Ins pages by using MySQL. Also, the design of the interface for these pages are done. Website's general template, Traveller Social Feed, Traveller Check-Ins and some part of the Signup pages are implemented and enhanced by integrating Bootstrap Library.
Göktuğ Özdoğan: Implemented a java code for creating tables. Implemented the necessary database queries for Venue Owner such as My Venues, My Events, Social Feed and designed UI of these pages.
İlhami Özer: Implemented the necessary database queries for Login and SignUp by using MySQL. Necessary Php, javaScript and html code was provided. User interface part of login and signup has implemented.

Other than team member specific workload, everyone in the team individually did their self learning on implementing website and researched for extra materials (tutorials, documentation about Bootstrap etc.) for a better understanding.

Final E/R Model



Final List of Tables

```
Account (userID PRIMARY KEY, nickname, fullname, email, password, userType,
dateOfCreation, profilePicture)
VenueOwner (userID PRIMARY KEY, experience)
      FOREIGN KEY (userID) references Account(userID)
Venue (venueID,userID,venuename,venuetype,city,address,zipcode, description)
      PRIMARY KEY (venueID, venuename)
      FOREIGN KEY (userID) references VenueOwner(userID)
User(userID,birthdate,city,gender,coinbalance)
      FOREIGN KEY (userID) references Account(userID)
UserRatesVenue(userID,venueID,rate)
      PRIMARY KEY (userID, venueID)
      FOREIGN KEY (userID) references User(userID)
      FOREIGN KEY (venueID) references Venue(venueID)
CheckIn(checkInID,venueID,userID,friendID,date)
      PRIMARY KEY (checkInID)
      FOREIGN KEY (venueID) references Venue(venueID)
      FOREIGN KEY (userID) references User(userID)
      FOREIGN KEY (friendID) references User(userID)
Comment(commentID,messag,date)
UserCommentVenue (commentID,userID,venueID)
      PRIMARY KEY (commentID, userID, venueID)
      FOREIGN KEY (userID) references User(userID)
      FOREIGN KEY (commentID) references Comment(commentID)
      FOREIGN KEY (venueID) references Venue(venueID)
VenueOwnerReplies
(userCommentID,venueownerCommentID,userID,veneueownerID,venueID)
      PRIMARY KEY (userCommentID, venueownerCommentID, userID,
veneueownerID, venueID)
      FOREIGN KEY (userCommentID) references Comment(commentID)
      FOREIGN KEY (venueownerCommentID) references Comment(commentID)
      FOREIGN KEY (userID) references UserCommentVenue(userID)
      FOREIGN KEY (venueID) references UserCommentVenue(venueID)
      FOREIGN KEY (veneueownerID) references VenueOwner(userID)
UserCommentsCheckIn (commentID,userID,checkInID)
      PRIMARY KEY(commentID, userID, checkInID)
      FOREIGN KEY (userID) references User(userID)
      FOREIGN KEY (commentID) references Comment(commentID)
      FOREIGN KEY (checkInID) references CheckIn(checkInID)
UserRatesComment (commentID,userID,rate)
      PRIMARY KEY(commentID, userID)
      FOREIGN KEY (userID) references User(userID)
      FOREIGN KEY (commentID) references Comment(commentID)
UserReplies (orginalPostCommentID,replyCommentID,userID)
      PRIMARY KEY(orginalPostCommentID, userID, replyCommentID)
      FOREIGN KEY (userID) references User(userID)
      FOREIGN KEY (orginalPostCommentID) references Comment(commentID)
      FOREIGN KEY (replyCommentID) references Comment(CommentID)
```

```
UserHasFriends (useruserID,frienduserID,dateOfFriendShip)
      PRIMARY KEY(useruserID , frienduserID)
      FOREIGN KEY (useruserID) references User(userID)
      FOREIGN KEY (frienduserID) references User(userID)
UserHasMatched (useruserID,matcheduserID,approved)
      PRIMARY KEY(useruserID , matcheduserID)
      FOREIGN KEY (useruserID) references User(userID)
      FOREIGN KEY (matcheduserID) references User(userID)
Event(eventID,venueownerID,venueID,escription,date,title,type)
      PRIMARY KEY(eventID)
      FOREIGN KEY (venueID) references Venue(venueID)
      FOREIGN KEY (venueownerID) references VenueOwner(userID)
Photo(picID PRIMARY KEY, photoLabel, dataOfShot)
AccountAddsPhotoToVenue(picID PRIMARY KEY,userID)
      FOREIGN KEY (picID) references Photo(picID)
      FOREIGN KEY (userID) references Account(userID)
      FOREIGN KEY (venueID) references Venue(venueID)
UserAddPhotoToCheckin(picID,userID,checkInID)
      FOREIGN KEY (picID) references Photo(picID)
      FOREIGN KEY (userID) references Account(userID)
      FOREIGN KEY (checkInID) references CheckIn(checkInID)
```

Implementation Details

The implementation process of our project, can be divided into two parts which are the front-end and the back-end. For database implementation, we used MySQL as our database management system and in the process of connection of the database, creating tables and accessing our database we used Java, and by means of the Java connection class we access our database with use of JDBC driver. After the access is granted, all the necessary tables are created and deleted, if requested, by the Java code. The database and data manipulation are done by the use of PHP. The HTML codes which are used for the front-end part of the project are supported by external CSS, JavaScript(JS) files and PHP codes. Logical operations are done in mostly PHP and JS parts of the code. Database connection and data manipulation is done via PHP which is connected to the database by the use of MySQLi database driver, and manipulate data that it retrieves and writes to the database. Some sorts of input operations and pop up box operations are handled using JS.

The design of the user interface is done with HTML. In the process of design, Bootstrap framework has been used as an external source of CSS, and Javascript files. It eased the creation of user friendly, and good looking pages. We linked our HTML codes to the framework via CDN. In addition to Bootstrap CSS files, we added custom CSS scripts inside our HTML codes that are unique to the corresponding page.

In the process of DB connections in both Java and PHP, we did not face any serious problem. However, for the creation of the HTML codes and UI components, we encountered several issues and paid a lot of time to get what we really wanted to serve to the end user. IN PHP codes, while retrieval and insertion of the data, we faced with SQL mistakes and fixed them. Apart from that, there needed to be session handling for both the aims of moving necessity information between pages and encapsulation of data when the user is not currently online in the system. Handling those issues in PHP codes took a little bit of time either as we controlled the corresponding session variables for the related pages. Another time consuming issue was about the version of Bootstrap framework. The sources on the Internet are often in different versions in which the tags of both CSS and HTML have different names. Thus, though the code was correct, it was not featuring as we expected. To fix this, we have read a lot of parts of Bootstrap Document for our own version.

Advanced Database Features

1. Reports

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
```

1.2. People 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
```

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
```

2. Views

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
```

2.2. User's Friend User View

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
```

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
 group by V.totalVisit desc
```

3. Triggers

3.1. Coin balance system

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.

```
/* Coinbalance */
create trigger CheckIn_coin after insert on CheckIn
        for each row begin
        update User set coinbalance = coinbalance +5 where
CheckIn.userID = User.userID;
        end

/* Coinbalance */
create trigger Comment_coin after insert on Comment
        for each row begin
        update Comment set coinbalance = coinbalance +1 where
Comment.userID = User.userID;
        end

/* Coinbalance */
create trigger UserRatesVenue_coin after insert on UserRatesVenue
        for each row begin
        update UserRatesVenue set coinbalance = coinbalance +1 where
UserRatesVenue.userID = User.userID;
        end

/* Coinbalance */
create trigger UserRatesComment_coin after insert on
UserRatesComment
        for each row begin
        update UserRatesComment set coinbalance = coinbalance +1 where
UserRatesComment.userID = User.userID;
        end
```

3.2. When A User Deletes Account

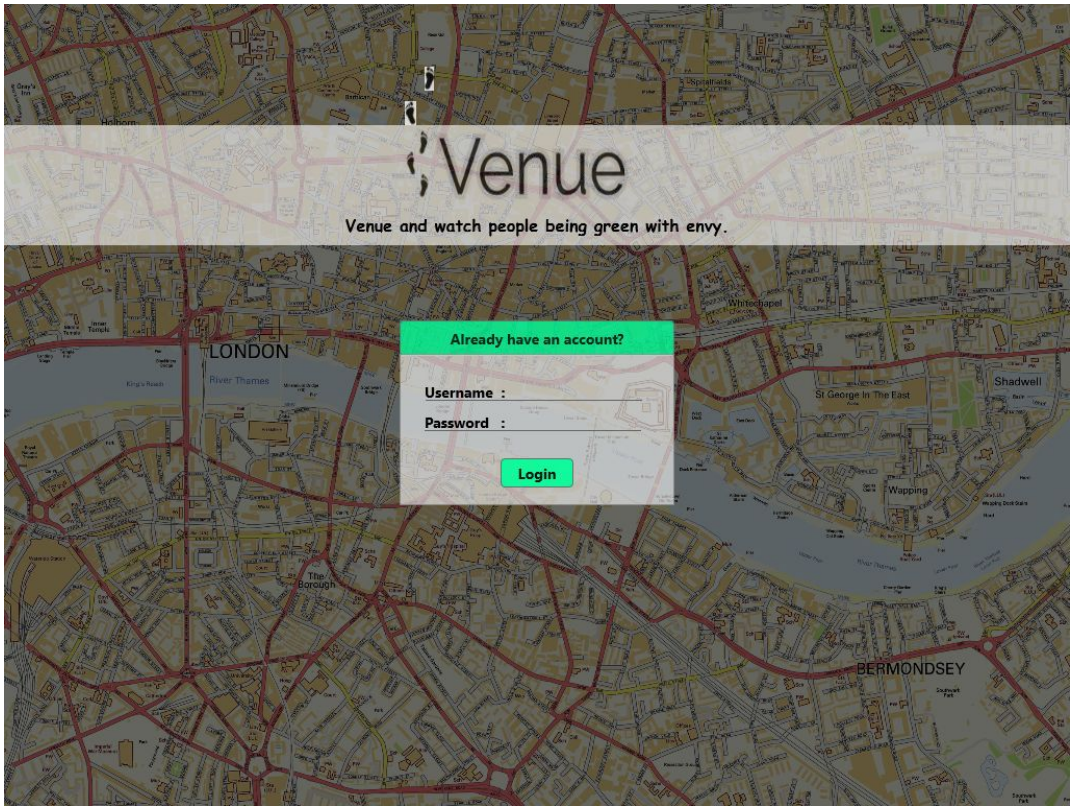
```
/* When Venue Owner delete Account */
create trigger delete_venueowner before delete on VenuOwner
      for each row begin
      delete from Event where Event.venueownerID = VenuOwner.userID;
      delete from VenueOwnerReplies where VenueOwnerReplies.venueownerID =
VenuOwner.userID;
      delete from AccountAddsPhotoToVenue where
AccountAddsPhotoToVenue.userID = VenuOwner.userID;
      delete from Venue where Venue.userID = VenuOwner.userID;
    end
/* When Venue Owner delete Account */
create trigger delete_venueowner after delete on VenuOwner
      for each row begin
      delete from Account where Account.userID = VenuOwner.userID;
      end

 /* When User delete Account */
create trigger delete_user before delete on User
      for each row begin
      delete from UserAddPhotoToCheckin where UserAddPhotoToCheckin.userID
= User.userID;
      delete from UserHasFriends where UserHasFriends.useruserID =
User.userID;
      delete from UserHasMatched where UserHasMatched.userID = User.userID;
      delete from UserReplies where UserReplies.userID = User.userID;
      delete from UserCommentsCheckIn where UserCommentsCheckIn.userID =
User.userID;
      delete from UserRatesComment where UserRatesComment.userID =
User.userID;
      delete from UserCommentVenue where UserCommentVenue.userID =
User.userID;
      delete from UserRatesVenue where UserRatesVenue.userID = User.userID;
      delete from CheckIn where CheckIn.userID = User.userID;
      end
/* When Venue Owner delete Account */
create trigger delete_venueowner after delete on User
```

```
for each row begin
delete from Account where Account.userID = User.userID;
end
```

User's Manual

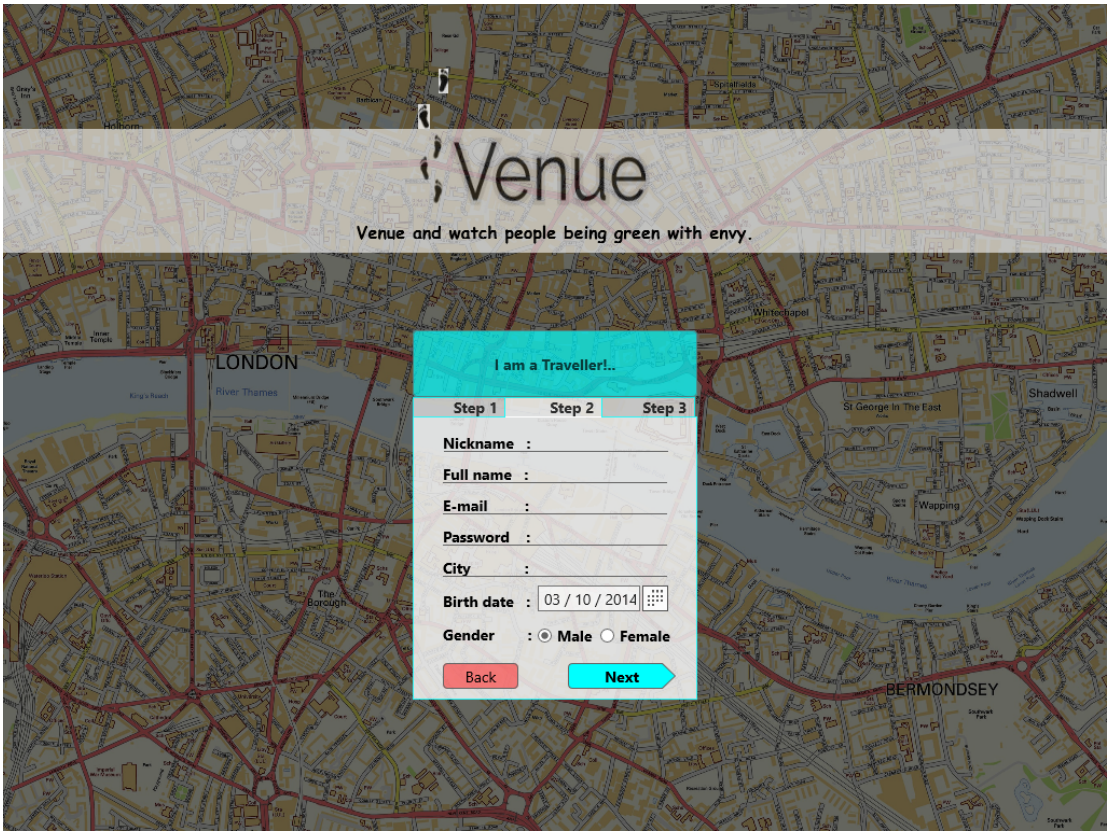
Common Pages

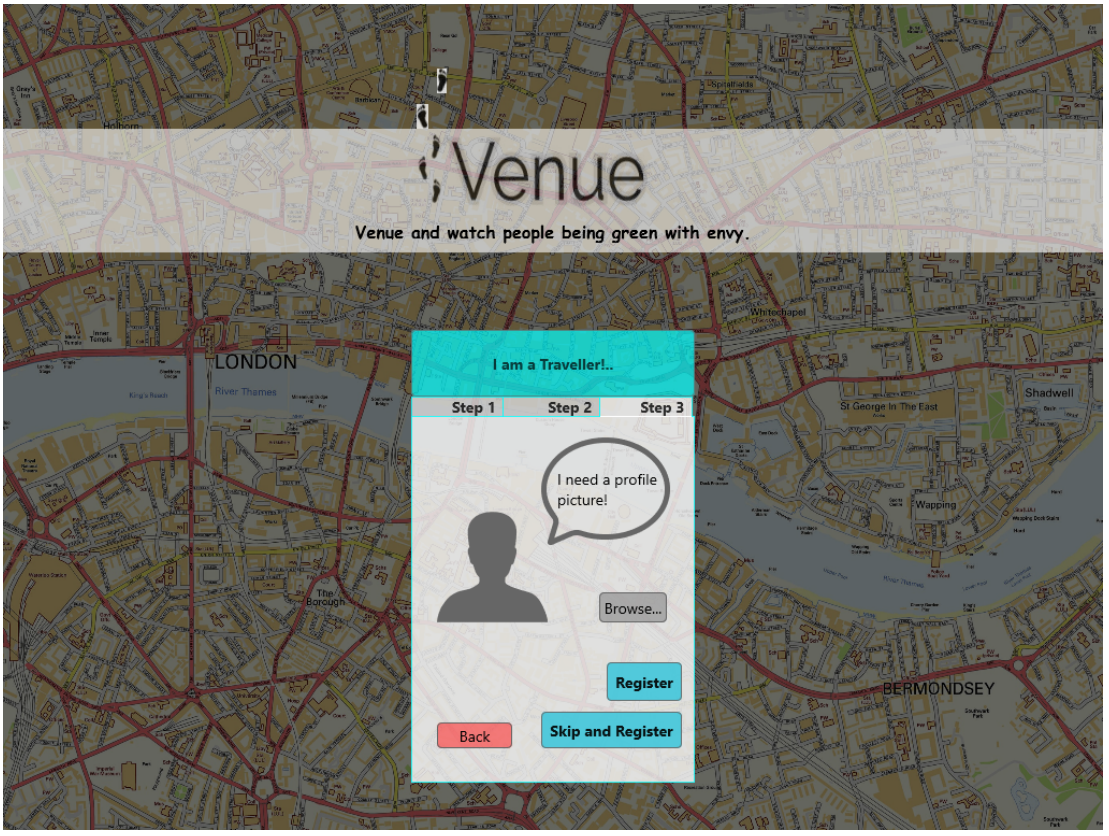
Login

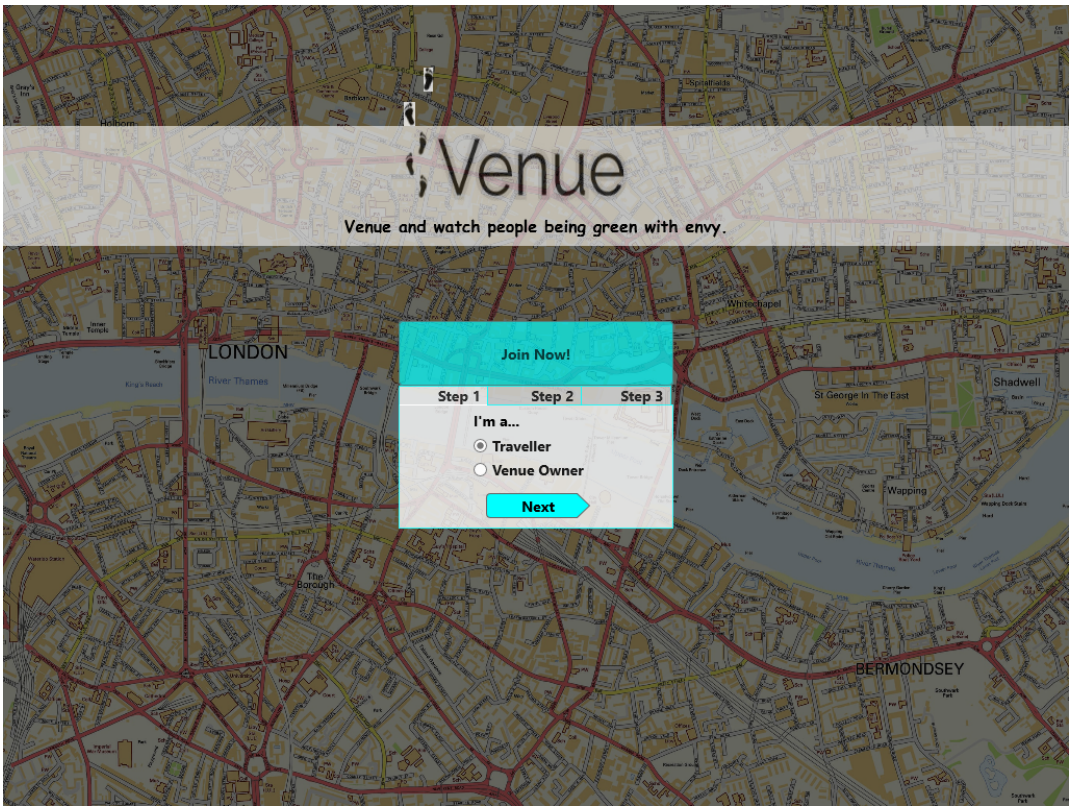


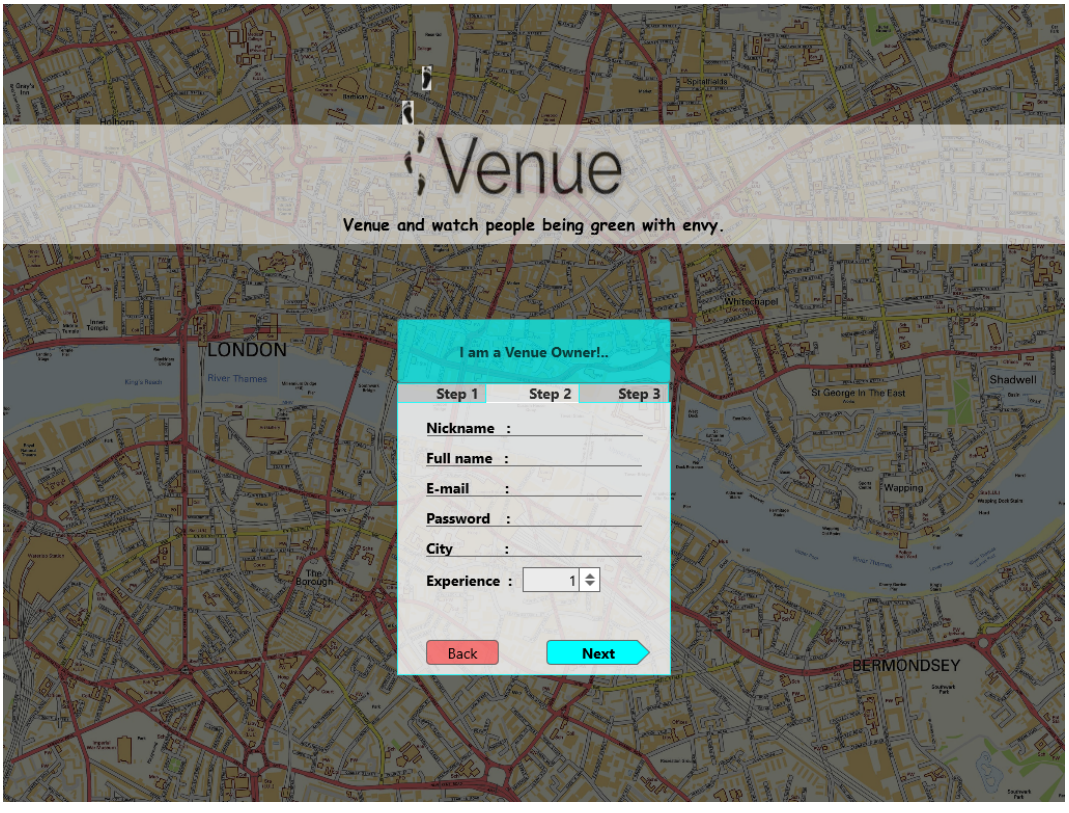
In order to login to the system, as a user or venue owner you can click on Login button in the homepage screen. Username and password are required to log on to the system. If your variables are matched with the databases values, you are directed to the social feed screen. If you do not have account by clicking "Create an Account" you will directed to sign up page.

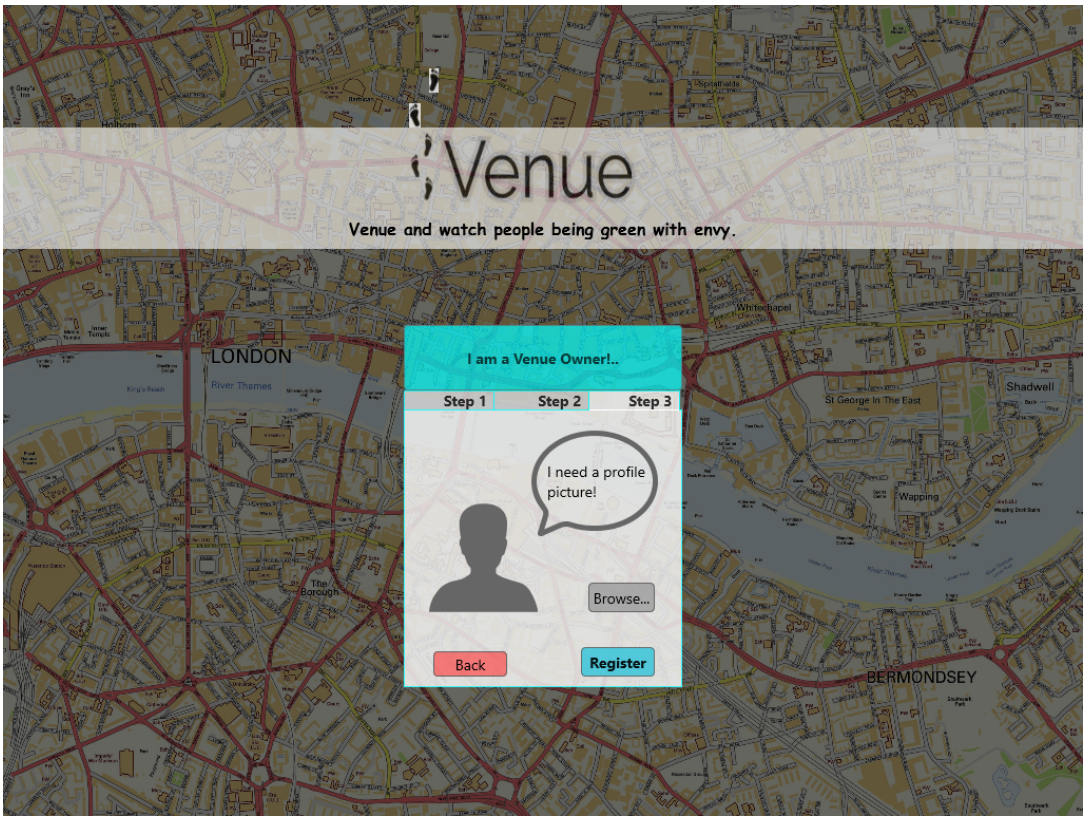
Sign Up


Venue
Venue and watch people being green with envy.
I am a Traveller!..
Step 1
Step 2
Step 3
Nickname :
Full name :
E-mail :
Password :
City :
Birth date : 03 / 10 / 2014
Gender : Male Female
Back
Next
LONDON
BERMONDSEY


Venue
Venue and watch people being green with envy.
I am a Traveller!..
Step 1
Step 2
Step 3
I need a profile picture!
Browse...
Register
Back
Skip and Register
LONDON
BERMONDSEY


Venue
Venue and watch people being green with envy.
LONDON
Join Now!
Step 1
Step 2
Step 3
I'm a...
Traveller
Venue Owner
Next
BERMONDSEY


Venue
Venue and watch people being green with envy.
LONDON
I am a Venue Owner!..
Step 1
Step 2
Step 3
Nickname :
Full name :
E-mail :
Password :
City :
Experience :
1
Back
Next
BERMONDSEY

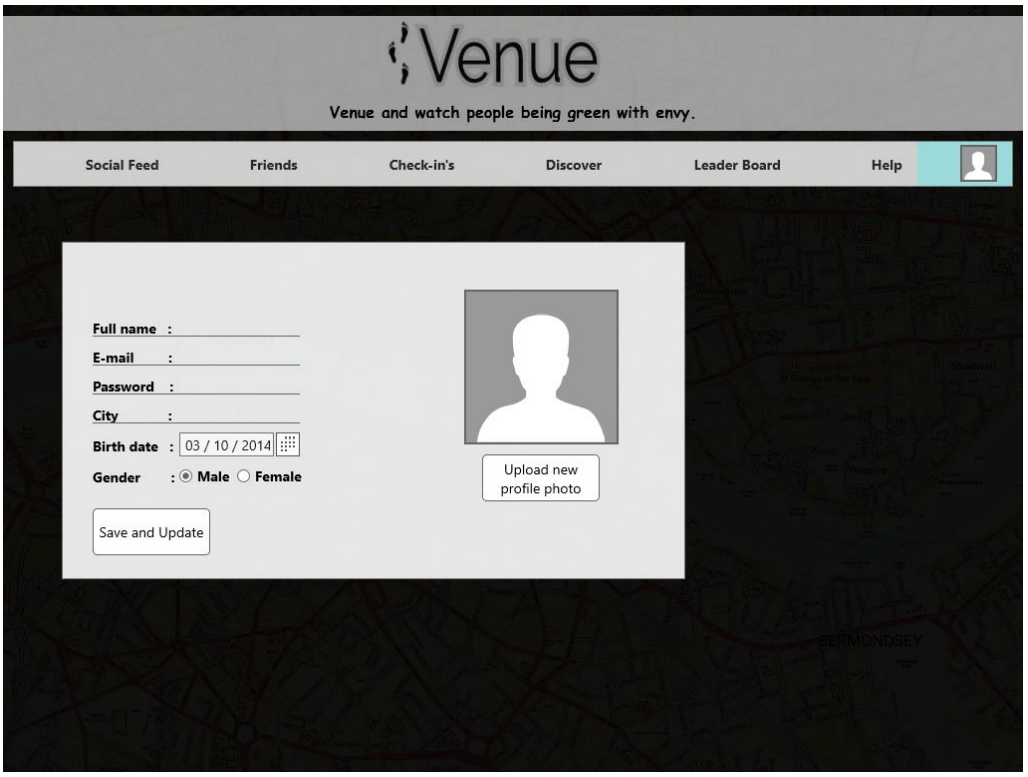


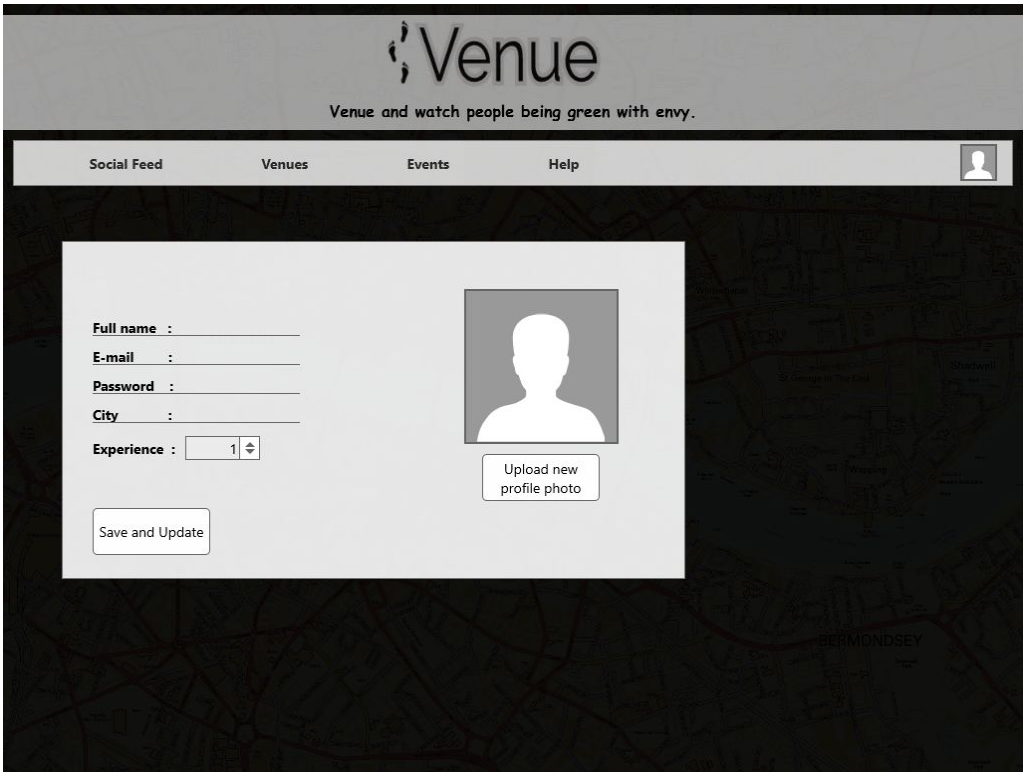
To sign-up as user, user type should be checked (traveller or venue owner), after that , username, full name, email, city,password gender must be provided. If you want you can add photo. Choose Files buttons enables you to choose photo from your computer. If sign up is successful you are directed to the salesperson screen.

Help

FAQ will be added in help web pages of both traveller and the venue owner.

Edit Profile

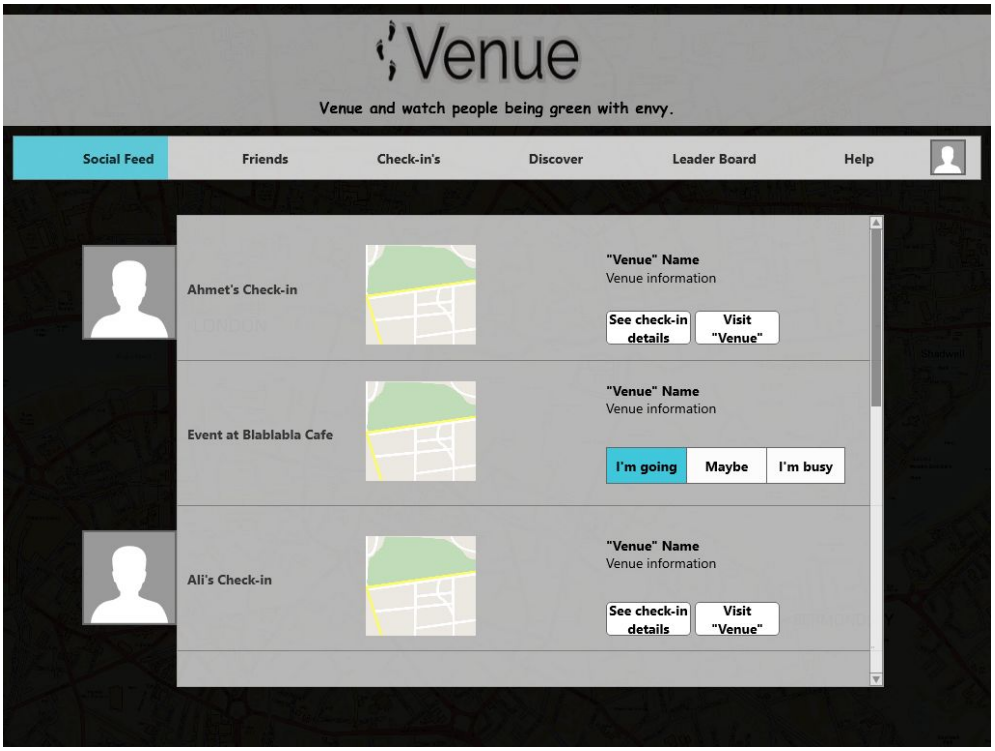




For both of the users, their edit profile web pages are for editing the existing profile information regarding to the current user in the session. If user updates any information regarding to its profile, this information is/are updated in the database, too.

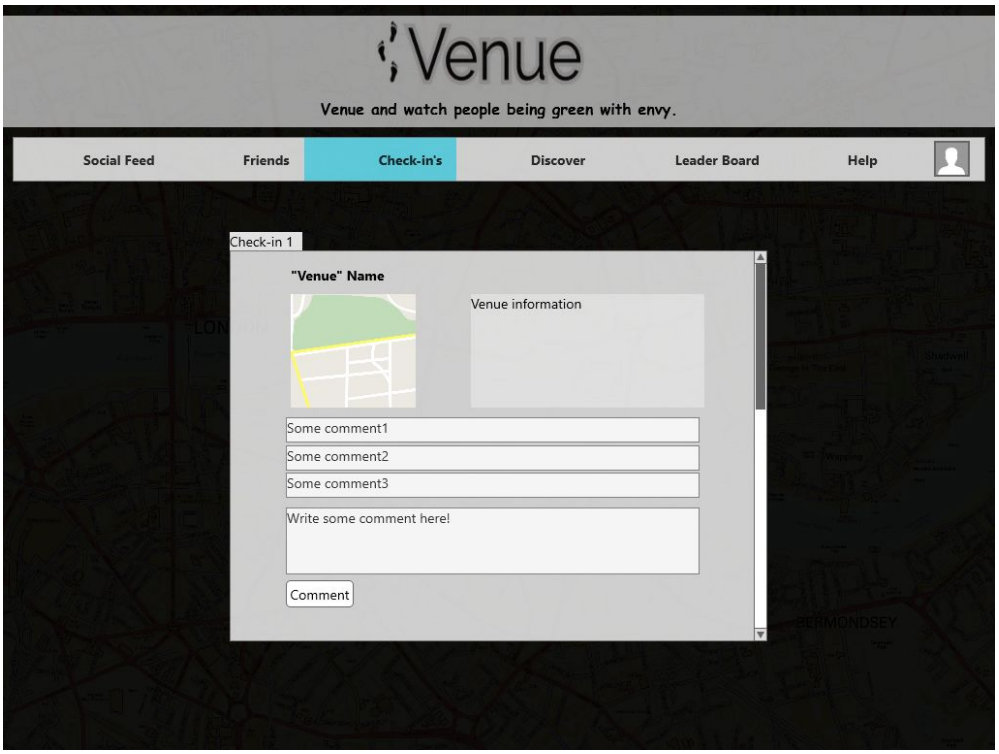
Traveller

Social Feed



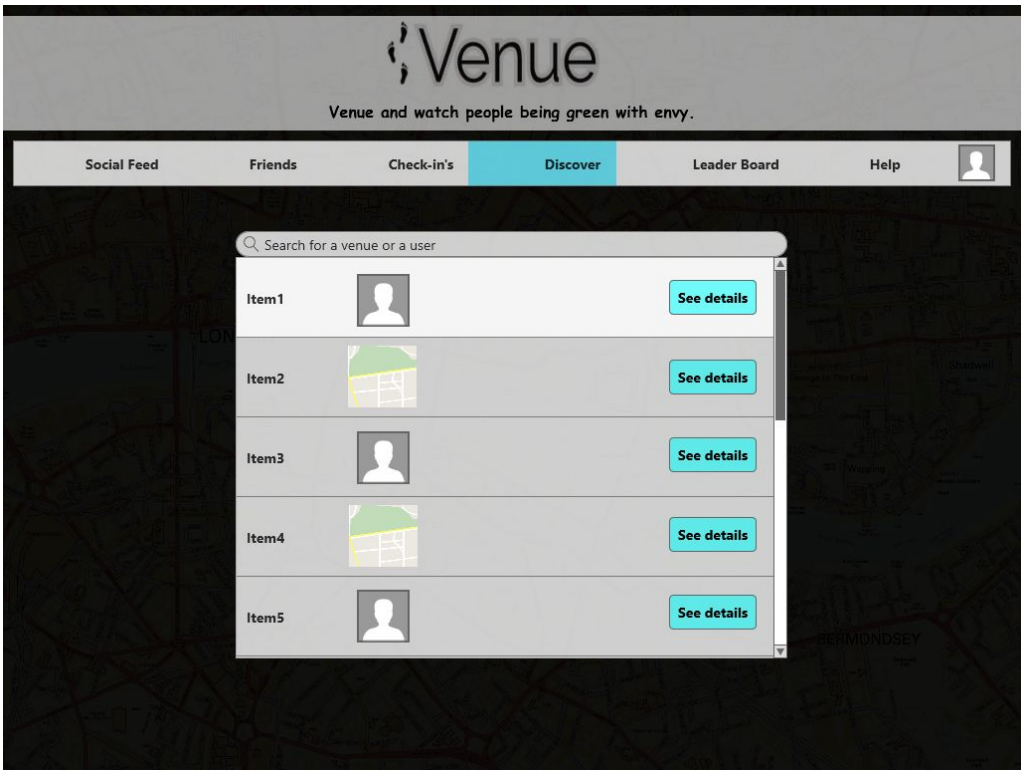
After user has signed up, or had a profile and logged in from the login page, it is directed to the Social Feed web page which can be considered as the home page for any type of user, i.e. traveller or venue owner. In this web page, traveller will be able to see the check-ins of its friends, and the events that are shot by the venue owners. In venue owner's social feed, venue owner will see the events shot by the other venue owner and the general information about its profile such as total visits to its venues will be shown. From this point, user can move between different pages through clicking the buttons on the navigation bar.

Check-in's



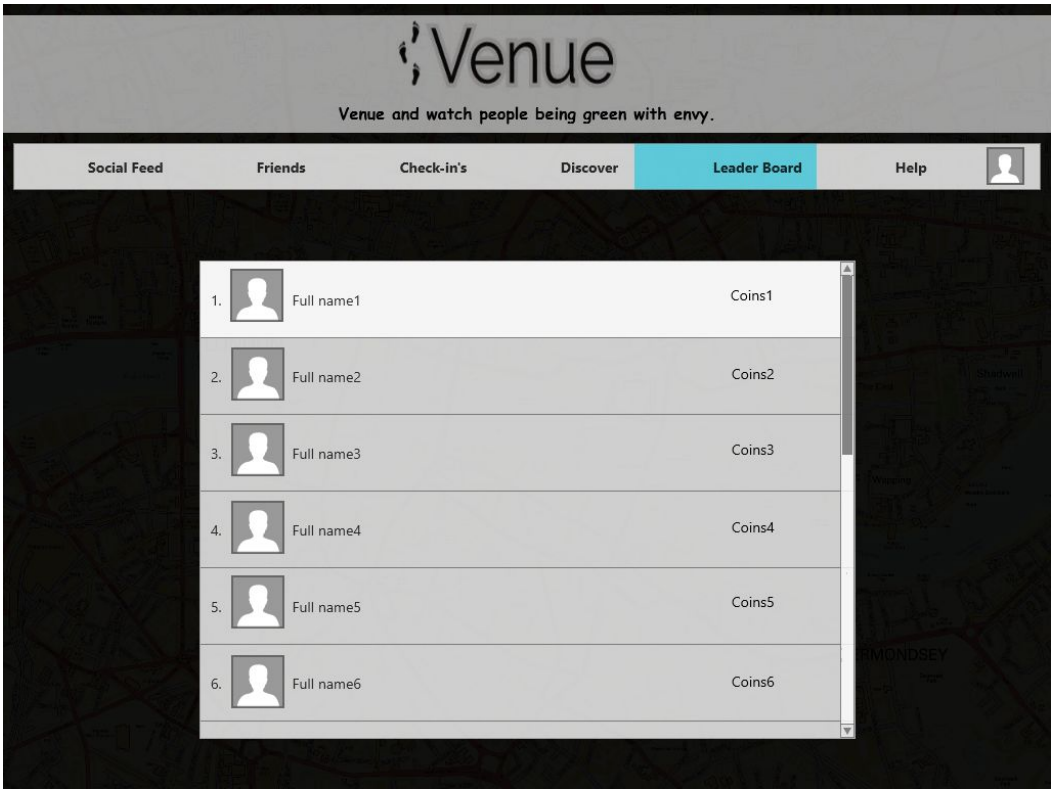
Traveller can move to this page via the button called Check-in's on the navigation bar. In this page, traveller will be able to see its previous check-ins to the venues and will be able to do a check-in to a venue from this web page.

Discover



In this webpage, traveller will be able to search for a venue, a friend or a non-friend user. According to what type of search it has done, some views will be shown to the traveller, i.e. if the searched user is not a friend, then it will show limited information about the user. When a venue is searched, it will show the information of the venue which has been specified by its venue owner in its creation process.

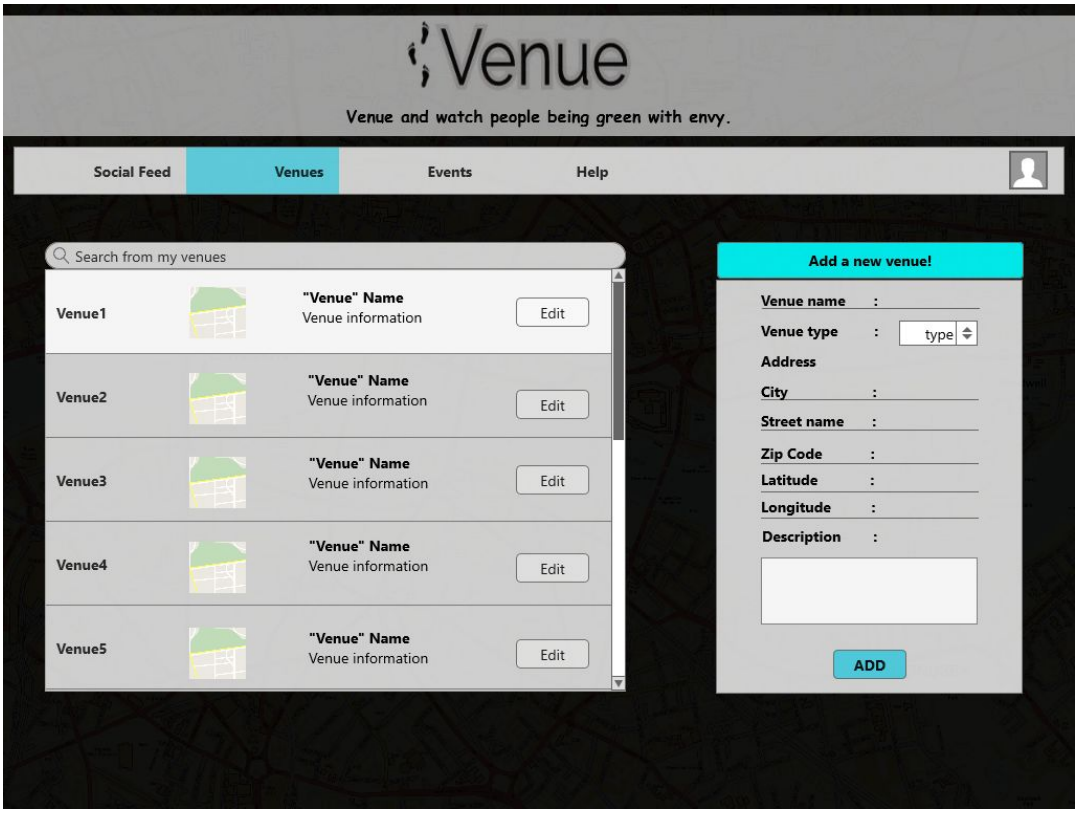
Leaderboard



This webpage show the leaderboard of the travellers who are located in the same city. The leaderboard is constructed by means of counting all the check-in numbers of the travellers and sorting them in the descending order.

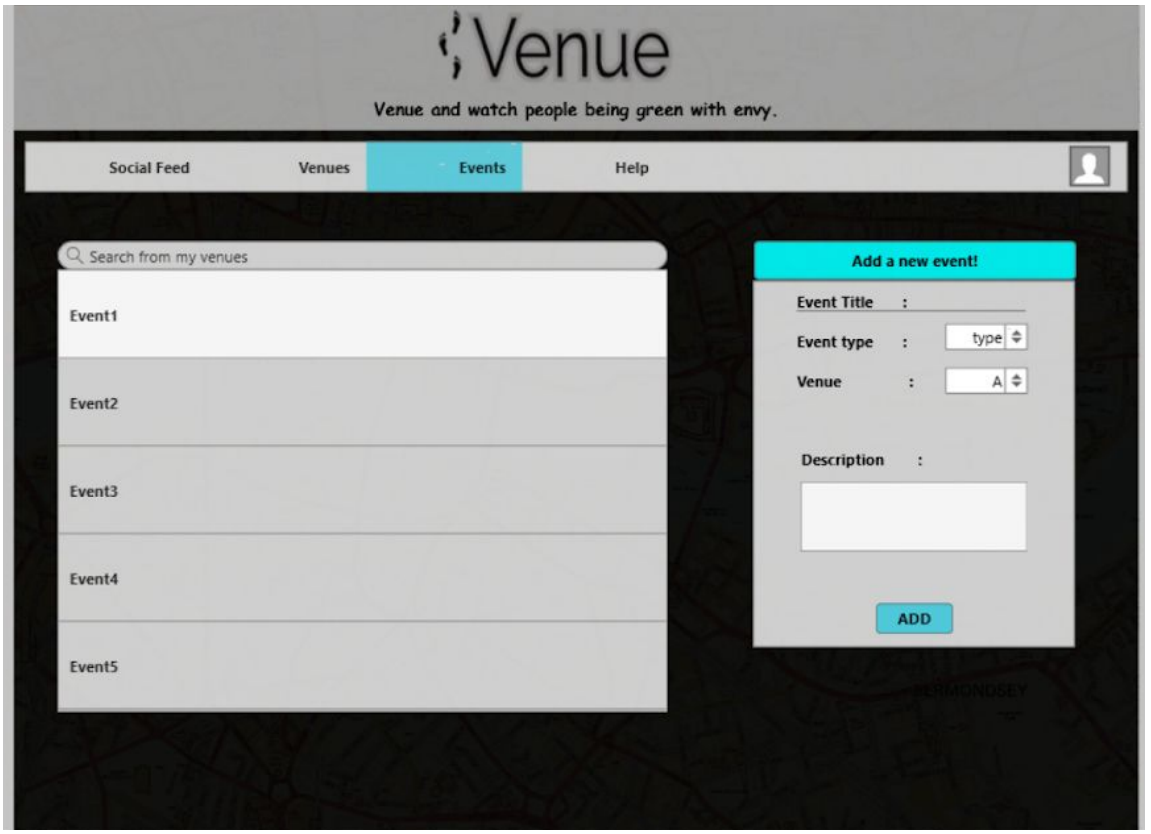
Venue Owner

Venues



In this webpage, venue owner is able to see its venues and detailed information of them. In addition to that venue owner will be able to add a new venue by means of a form located in the container of this webpage. To add a new venue, venue owner enters the information regarding to the new venue in the form and creates the venue by means of pressing the button.

Events



This webpage is for venue owners to see its previously created events and to add new events which will be shown to all the other users of the system. The working principle of this webpage is same with the one for the venues' webpage. In order to add a new event, venue owner fills the form that corresponds to the information regarding to the newly desired event, and then creates the event by pressing to the button.