

CS353 DATABASE SYSTEMS

2018-2019 SPRING SEMESTER

**SCENEZ**

PROJECT PROPOSAL

GROUP 19

Dawood Muzammil **MALIK**

Masna **AHMED**

Muhammad Umair **AHMED**

Talha **ZEESHAN**

Contents

[**1.** **Revised ER Model** 3](#_Toc5350245)

[**2.** **Relational Schemas** 5](#_Toc5350246)

[**2.1** **Entities** 5](#_Toc5350247)

[**2.2** **Relationships** 9](#_Toc5350248)

[**3.** **Functional Dependencies and Normalization of Tables** 16](#_Toc5350249)

[**4.** **Functional Tables** 16](#_Toc5350250)

[**4.1 Use Case Scenarios** 16](#_Toc5350251)

[**4.2 Algorithms** 28](#_Toc5350252)

[**4.3 Data Structures** 29](#_Toc5350253)

[**5.** **User-Interface Design and corresponding SQL statements** 29](#_Toc5350254)

[**6.** **Advanced Database Components** 37](#_Toc5350255)

[**6.1** **Triggers** 37](#_Toc5350256)

[**6.2** **Views** 37](#_Toc5350257)

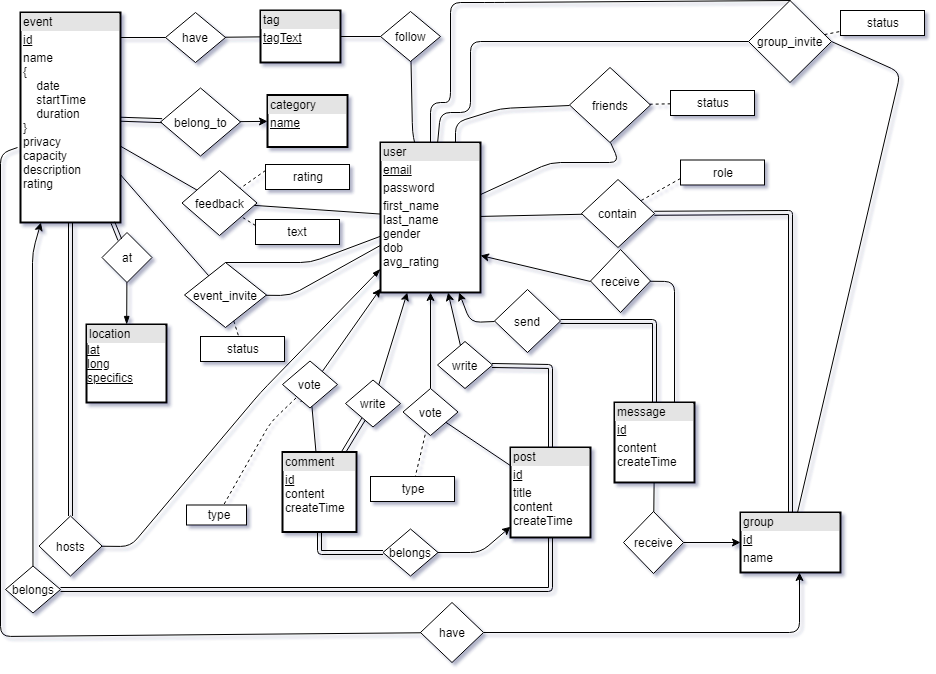
[**6.3** **Stored Procedures** 38](#_Toc5350258)

[**6.4** **Constraints** 38](#_Toc5350259)

[**7.** **Implementation Plan** 38](#_Toc5350260)

[**8.** **Website** 38](#_Toc5350261)

# **Revised ER Model**



The following changes have been made to the initial project ER diagram:

* Some new attributes have been added to the *users* entity to store more information about the user.
* A *follow* relationship has been added between *tags* and users to store the tags followed by specific users.
* The separate entities named *event\_comments* and *group\_comments* were deleted and a single entity *comment* was added. The rationale behind this was to eliminate all possible weak entities.
* For the purpose of reducing complexity, entity named *group\_posts* is now deleted. Posts now only belong to events (*post-belongs-event)*.
* Entities named *creates* and *contain* were merged into a single entity (*contain*) with a single attribute named role associated with it. This was done to avoid unnecessary duplication of data records.
* An extra attribute (*text*) was added on the *feedback* relationship between *user* and *event*. This will allow the users to give textual feedback on the events they attend.
* Deleted the ternary relationship *belongs\_to* as it was an incorrect representation of a ternary relationship.
* *Groups* are now linked to *events* through a *have* relationship. The rationale behind this change is that there could be some events that are hosted by the groups and not individual users.
* The relationship *is\_friends\_with* is now renamed to *friends* for simplicity purposes. An attribute *status* is also added to keep track of friend requests.
* An attribute *status* is added to relationship *group\_invite­* to keep track of invitations sent to friends to join groups.
* A mechanism to send message to a group is now enabled through a *receive* relationship.
* A *type* attribute is added on the *vote* relationship to keep track of the type of vote it is (upvote or downvote).
* An *id* attribute is added to *comment* to uniquely identify each comment.
* The relationships *attending* and *interested* are now merged in a new relationship named *event\_invite*. This also gives us the functionality of users inviting their friends to events.

# **Relational Schemas**

## **Entities**

1. ***User***

Relational Model

user(email, password, first\_name, last\_name, dob, gender, avg\_rating)

Functional Dependencies

email -> password, first\_name, last\_name, dob, gender, avg\_rating

Candidate Keys

{(email)}

Normal Form

BCNF

Table Definition

create table user(email varchar(50) PRIMARY KEY, password varchar(50) , first\_name varchar(50), last\_name varchar(50), dob date, gender ENUM('Male','Female','Other'), avg\_rating double check(avg\_rating >= 0 and avg\_rating <= 5))

1. ***Event***

Relational Model

event(id, name, date, start\_time, duration, privacy, capacity, description, email, cat\_name)

FK: ‘email’ references ‘email’ from ‘user’

FK: ‘cat\_name’ references ‘name’ from ‘category’

Functional Dependencies

id -> name, date, start\_time, duration, privacy, capacity, description

Candidate Keys

{(id)}

Normal Form

BCNF

Table Definition

create table event(id int PRIMARY KEY, name varchar(150), event\_date date, start\_time time, duration double, privacy char(7), capacity int, description varchar(5000), email varchar(50), cat\_name varchar(100), location\_id int,

FOREIGN KEY (email) REFERENCES user(email),

FOREIGN KEY (cat\_name) REFERENCES category(name),

FOREIGN KEY (location\_id) REFERENCES location(id))

1. ***Location***

Relational Model

location(id, lat, long, specifics)

Functional Dependencies

id -> lat, long, specifics

Candidate Keys

{(id)}

Normal Form

BCNF

Table Definition

CREATE TABLE location(id int, lat double, lng double, specifics VARCHAR(500), PRIMARY KEY(id))

1. ***Category***

Relational Model

category(name)

Candidate Keys

{(name)}

Normal Form

BCNF

Table Definition

create table category(name varchar(100) PRIMARY KEY)

1. *Group*

Relational Model

group(id, name, avg\_rating)

Functional Dependencies

id -> name, avg\_rating

Candidate Keys

{(id)}

Normal Form

BCNF

Table Definition

create table grp( id int PRIMARY KEY, name varchar(150) )

1. ***Post***

Relational Model

post(id, title, content, createTime, email, event\_id)

FK: ‘email’ references ‘email’ from ‘user’

FK: ‘event\_id’ references ‘id’ from ‘event’

Functional Dependencies

id -> title, content, createTime, email, event\_id

Candidate Keys

{(id)}

Normal Form

BCNF

Table Definition

create table post(id int PRIMARY KEY, title varchar(150), content varchar(5000), timestamp datetime,

email varchar(50), event\_id int(11),

FOREIGN KEY (email) REFERENCES user(email),

FOREIGN KEY (event\_id) REFERENCES event(id))

1. ***Comment***

Relational Model

comment(id, content, createTime, email, post\_id)

FK: ‘email’ references ‘email’ from ‘user’

FK: ‘post\_id’ references ‘id’ from ‘post

Functional Dependencies

id -> content, createTime, email, post\_id

Candidate Keys

{(id)}

Normal Form

BCNF

Table Definition

create table comment(id int PRIMARY KEY, content varchar(1000), timestamp datetime, email varchar(50), post\_id int,

FOREIGN KEY(email) REFERENCES user(email),

FOREIGN KEY(post\_id) REFERENCES post(id))

1. ***Message***

Relational Model

message(id, content, createTime, sender)

FK: ‘sender’ references ‘email’ from ‘user’

Functional Dependencies

id -> content, createTime, sender

Candidate Keys

{(id)}

Normal Form

BCNF

Table Definition

create table message(id int, content varchar(1000), timestamp datetime, sender varchar(50),

FOREIGN KEY (sender) REFERENCES user(email))

1. Tag

Relational Model

tag(tagText)

Functional Dependencies

None

Candidate Keys

{(tagText)}

Normal Form

3NF

Table Definition

create table tag(tagText varchar(50) PRIMARY KEY)

## **Relationships**

1. ***Follow***

Relational Model

user\_tags(email, tagText)

FK: ‘email’ references ‘email’ from ‘user’

FK: ‘tagText’ references ‘tagText’ from ‘tag’

Functional Dependencies

None

Candidate Keys

{(email, tagText)}

Normal Form

3NF

Table Definition

create table user\_tags(email varchar(50), tagText varchar(50), PRIMARY KEY(email, tagText),

FOREIGN KEY(email) REFERENCES user(email),

FOREIGN KEY(tagText) REFERENCES tag(tagText))

1. ***Group Invite***

Relational Model

group\_invite(inviter, invitee, group\_id, status)

FK: ‘inviter’ references ‘email’ from ‘user’

FK: ‘invitee’ references ‘email’ from ‘user’

FK: ‘group\_id’ references ‘id’ from ‘group’

Functional Dependencies

None

Candidate Keys

{(inviter, invitee, group\_id)}

Normal Form

3NF

Table Definition

create table group\_invite(inviter varchar(50), invitee varchar(50), group\_id int, status ENUM

(‘Accepted’, ‘User\_Pending’, ‘Admin\_Pending’, ‘Rejected’),

PRIMARY KEY(inviter, invitee, group\_id),

FOREIGN KEY(inviter) REFERENCES user(email),

FOREIGN KEY(invitee) REFERENCES user(email),

FOREIGN KEY(group\_id) REFERENCES grp(id))

1. ***Event Invite***

Relational Model

event\_invite(inviter, invitee, event\_id, status)

FK: ‘inviter’ references ‘email’ from ‘user’

FK: ‘invitee’ references ‘email’ from ‘user’

FK: ‘event\_id’ references ‘id’ from ‘event’

Functional Dependencies

None

Candidate Keys

{(inviter, invitee, event\_id)}

Normal Form

3NF

Table Definition

create table event\_invite(inviter varchar(50), invitee varchar(50), event\_id int, status ENUM('User\_Pending','Admin\_Pending','Accepted','Rejected'),

FOREIGN KEY(inviter) REFERENCES user(email),

FOREIGN KEY(invitee) REFERENCES user(email),

FOREIGN KEY(event\_id) REFERENCES event(id),

PRIMARY KEY(inviter, invitee, event\_id))

1. ***Is Friends***

Relational Model

is\_friends\_with(user1, user2, status)

FK: ‘user1’ references ‘email’ from ‘user’

FK: ‘user2’ references ‘email’ from ‘user’

Functional Dependencies

None

Candidate Keys

{(user1, user2)}

Normal Form

3NF

Table Definition

create table is\_friends\_with(user1 varchar(50), user2 varchar(50),

status ENUM('Accepted','Pending','Rejected'), PRIMARY KEY(user1, user2),

FOREIGN KEY (user1) REFERENCES user(email),

FOREIGN KEY (user2) REFERENCES user(email))

1. ***Contains***

Relational Model

contains(group\_id, user, role)

FK: ‘group\_id’ references ‘id’ from group

FK: ‘user’ references ‘email’ from ‘user’

Functional Dependencies

None

Candidate Keys

{(group\_id, user)}

Normal Form

3NF

Table Definition

create table contains(group\_id int, user varchar(50), role ENUM('Admin','Creator','Member'),

PRIMARY KEY(group\_id, user),

FOREIGN KEY (group\_id) REFERENCES grp(id),

FOREIGN KEY (user) REFERENCES user(email))

1. ***Recepient User***

Relational Model

recipient\_user(message\_id, user\_id)

FK: ‘message\_id’ references ‘id’ from ‘message’

FK: ‘user\_id’ references ‘email’ from ‘user’

Functional Dependencies

message\_id -> user\_id

Candidate Keys

{(message\_id, user\_id)}

Normal Form

3NF

Table Definition

create table recipient\_user(message\_id int, user\_id varchar(50), PRIMARY KEY(message\_id, user\_id),

FOREIGN KEY (message\_id) REFERENCES message(id),

FOREIGN KEY (user\_id) REFERENCES user(email))

1. ***Recipient Group***

Relational Model

recipient\_group(message\_id, group\_id)

FK: ‘message\_id’ as ‘id’ from ‘message’

FK: ‘group\_id’ as ‘id’ from ‘group’

Functional Dependencies

message\_id -> group\_id

Candidate Keys

{(message\_id, group\_id)}

Normal Form

3NF

Table Definition

create table recipient\_grp(message\_id int, grp\_id int,

FOREIGN KEY(message\_id) REFERENCES message(id),

FOREIGN KEY(grp\_id) REFERENCES grp(id),

PRIMARY KEY(message\_id, grp\_id))

1. ***Vote Post***

Relational Model

vote\_post(post\_id, user\_id)

FK: ‘user\_id’ references ‘email’ from ‘user’

FK: ‘post\_id’ references ‘id’ from ‘post’

Functional Dependencies

None

Candidate Keys

{(post\_id, user\_id)}

Normal Form

3NF

Table Definition

create table vote\_post(post\_id int, user\_id varchar(50), vote\_type int check(type = 1 OR type = -1),

PRIMARY KEY(post\_id, user\_id),

FOREIGN KEY (post\_id) REFERENCES post(id),

FOREIGN KEY (user\_id) REFERENCES user(email))

1. ***Vote Comment***

Relational Model

vote\_comment(comment\_id, user\_id)

FK: ‘comment\_id’ as ‘id’ from ‘comment’

FK: ‘user\_id’ as ‘email’ from ‘user’

Functional Dependencies

None

Candidate Keys

{(comment\_id, user\_id)}

Normal Form

3NF

Table Definition

create table comment\_vote(comment\_id int, email varchar(50), vote\_type int check(type = 1 OR type = -1),

FOREIGN KEY (comment\_id) REFERENCES comment(id),

FOREIGN KEY (email) REFERENCES user(email),

PRIMARY KEY(comment\_id, email))

1. ***Group Event***

Relational Model

group\_event(group\_id, event\_id)

FK: ‘group\_id’ references ‘id’ from ‘group’

FK: ‘event\_id’ references ‘id’ from ‘event’

Functional Dependencies

event\_id -> group\_id

Candidate Keys

{(group\_id, event\_id)}

Normal Form

3NF

Table Definition

create table grp\_event(grp\_id int, event\_id int,

FOREIGN KEY(grp\_id) REFERENCES grp(id),

FOREIGN KEY(event\_id) REFERENCES event(id),

PRIMARY KEY(grp\_id, event\_id))

1. ***Event Tag***

Relational Model

event\_tag(event\_id, tag)

FK: ‘event\_id’ references ‘id’ as ‘event’

FK: ‘tag’ references ‘tagText’ from ‘tag’

Functional Dependencies

None

Candidate Keys

{(event\_id, tag)}

Normal Form

3NF

Table Definition

create table event\_tag(event\_id int, tag varchar(50), PRIMARY KEY(event\_id, tag),

FOREIGN KEY (event\_id) REFERENCES event(id),

FOREIGN KEY (tag) REFERENCES tag(tagText))

1. ***Event\_feedback***

Relational Model

event\_feedback(user\_id, event\_id, rating, text)

FK: ‘user\_id’ references ‘email’ from ‘user’

FK: ‘event\_id’ references ‘id’ from ‘event’

Functional Dependencies

None

Candidate Key

{(user\_id, event\_id)}

Normal Form

3NF

Table Definition

create table event\_feedback(user\_id varchar(50), event\_id int, rating int check(rating >= 0 and rating <= 5)

, text varchar(500), PRIMARY KEY (user\_id, event\_id),

FOREIGN KEY (user\_id) REFERENCES user(email),

FOREIGN KEY (event\_id) REFERENCES event(id))

# **Functional Dependencies and Normalization of Tables**

# **Functional Tables**

## **4.1 Use Case Scenarios**

The Scenez Application system includes three main types of users; User, Group\_Admin and Event\_Host. A normal user will require added functionality depending on if they create a group or an event. This new added functionality is described using the two mentioned user types. A regular user in the event of creating an event or group will acquire administrative/Host privileges for that specific group or event. For clarity purposes the use case scenarios of all three are types of users are given and corresponding use case scenarios described below. While each type of user possess unique functionality, there are a few that overlap among the users. For clarity purposes, common functionalities between the three types of users have not been removed. All three user types share the same entry condition which is that the user must have an account on the application.

**i) User**

The user represents the average, regular user of the application without any administrative or host privileges due to them not having created an event or group. Described below are various use case scenarios for such a user of Scenez. The user may or maynot be a member of a group or event. For clarity purposes, the use cases provided include all possible scenarios for a regular user.

* **Searching the Application**

Use Case Name: *Search*

Description:

The user may use the search bar to look for desired events or groups or friend profiles on the application. Depending on the setting chosen, the system will display the result of the search query which will include either the specific event mentioned or similar events, groups matching the search criteria or the profile of user searched. All three search types extend the “search” use case scenario. The “View” scenario views details related to resulting event,group or searched profile.

* **View My Profile**

Use Case Name: *View Profile*

Description:

The “View Profile” use case is invoked if the user wishes to view their Scenez profile. The system displays the user profile if this use case is invoked. The “View Profile” use case is further extended by the use case scenario “Update Profile” where the user may or may not choose to update their profile settings.

* **Create an Event**

Use Case Name: *Create Event*

Description:

The “Create Event” use case is invoked when the user chooses to create an event themselves. When invoked, the system provides an interface in which the user enters relevant details relating to the event (Title, location, description, Date of Event, Theme etc) and chooses whether the event will be public or invite only. Once created, the user is granted host privileges/functionality for their created event (detailed further in the report).

* **Join Event**

Use Case Name: *Join Event*

Condition: Event is Public

Description:

The user has the functionality to join public event. In such a case, if the user chooses to attend the event, the user may select the *Going* option in which this use case is invoked. The system confirms the user choice by displaying a small text message.

* **Create a Group**

Use Case Name: *Create Group*

Description:

The “Create Group” use case is invoked when the user chooses to create a group themselves. When invoked, the system provides an interface in which the user enters relevant details relating to the group(Title, description, Themes etc) and chooses whether the event will be public or invite only. Once created, the user is granted group administrator privileges/functionality for their created group (detailed further in the report).

* **Create a Post**

Use Case Name: *Create Post*

Entry Condition: User is part of or host of an event

Description:

The “Create Post” use case is invoked when the user chooses to post to a specific event. The user has to be a part of or host of an event in which the post is to be made. The post content is limited to simple text.

* **Make a Comment**

Use Case Name: *Make Comment*

Description:

The “Make Comment” use case is invoked when the user chooses to comment on specific post. The comment is limited to include simple text.

* **Leave a Vote**

Use Case Name: *Vote*

Description:

The “Vote Comment” use case is invoked when the user chooses to Upvote or DownVote a comment or post.

* **Handle Requests**

Use Case Name: *Handle Requests*

Description:

The “Handle Requests” use case is invoked when the user chooses to accept or reject requests they have been sent. The requests may be related requests to join a group, requests to join an event or friend requests from other users. The user may accept or reject requests sent.

* **Send Invite’s**

Use Case Name: *Send Invite*

Condition: User part of group/event for group/event invites

Description:

The “Send Invite” use case is invoked when the user chooses to send invitations to their friends to join a specific group (only if they are already part of said group) or event or a friendship invite to another user.

* **Send Message**

Use Case Name: *Send Message*

Description:

The user has the ability to send messages to their friends, group administrators or event hosts. In such a scenario, the “Send Message” use case is invoked. The messages are limited to simple text.

* **View Messages**

Use Case Name: *View Message*

Description:

This use case is invoked in the case when the user reads/views a received message from either friends, group administrator of group they are a member of or event host of event they are attending.

* **Leave Feedback**

Use Case Name: *Give Feedback*

Description:

This use case invoked when the user chooses to give feedback for an event they have attended. the feedback can only be given after the date of event. the feedback consists of textual description and a rating system out of 5.

Figure 1: User Use Case Diagram

**ii) Group\_Admin**

The Group\_Admin user type represents the a user of the application who has acquired group administrator privileges by creating their own group. Described below are various additional use case scenarios for such a user of Scenez. The described use case scenarios below for group administrator are limited to within the group of which they are the administrator of.

* **Invite Approvals**

Use Case Name: Handle Requested Invites

Description:

This use case is invoked when the group administrator approves or rejects requested invites of friends of group members. The group administrator is alerted of these requests after a group member successfully invites a friend of theirs to the group. All users not invited by the administrator are subjected to administrator approval (use case invoked).

* **Remove Members**

Use Case Name: Remove Members

Description:

The use case is invoked when the group administrator chooses to remove a group member from the group. The group administrator has the authority to remove any member of the group they see fit. The invoking of this use case results in system revoking membership of said member.

* **View Members**

Use Case Name: View Members

Description:

The group administrator has the authority to view all members that are part of the group. The invoking of this use case results in the system displaying all current members of the group.

* **Message Members**

Use Case Name: Message Members

Description:

The group administrator has the ability to send messages to entire group. The invoking of this use case results in a message being sent to all current members of the group. The message is limited to simple text form.

* **Remove Posts**

Use Case Name: Remove Posts

Description:

The group administrator has the authority to remove any posts made on group events. Upon invoking this use case, the respective post is removed from the group event.

* **Remove Comments**

Use Case Name: Remove Comments

Description:

The group administrator has the authority to remove any comments made on group event posts. Upon invoking this use case, the respective comment is removed from the group event post.

* **Make Comment**

Use Case Name: Make Comment

Description:

The “Make Comment” use case is invoked when the group administrator chooses to comment on specific group event post. The comment is limited to include simple text.

* **Leave a Vote**

Use Case Name: *Vote*

Description:

The “Vote Comment” use case is invoked when the group admin chooses to Upvote or DownVote a comment or post made within said group events.

* **Make Post**

Use Case Name: *Make Post*

Description:

The “Create Post” use case is invoked when the administrator chooses to post to a specific group event. The post content is limited to simple text.

* **Send Invites**

Use Case Name: *Send Invite*

Description:

The “Send Invite” use case is invoked when the group admin chooses to send invitations to his/her friends to join the group. Invitees Invited by the group administrator are exempt from approvals and these invitees can immediately join the group if they reply positively.

* **Update Settings**

Use Case Name: *Update Settings*

Description:

The group administrator has the authority to change group settings(changing name, themes, rules etc). In such a case, this use case is invoked. The system updates the group settings upon successful invoke.

* **Create Group Event**

Use Case Name: *Create Group Event*

Description:

The group administrator has the sole authority to create group events. When invoked, the system provides an interface in which the admin enters relevant details relating to the event (Title, location, description, Date of Event, Theme etc) and chooses whether the event will be public or limited to group members. Furthermore, members of said group are automatically invited to the event upon its creation.

* **Delete Group Event**

Use Case Name: *Delete Group Event*

Description:

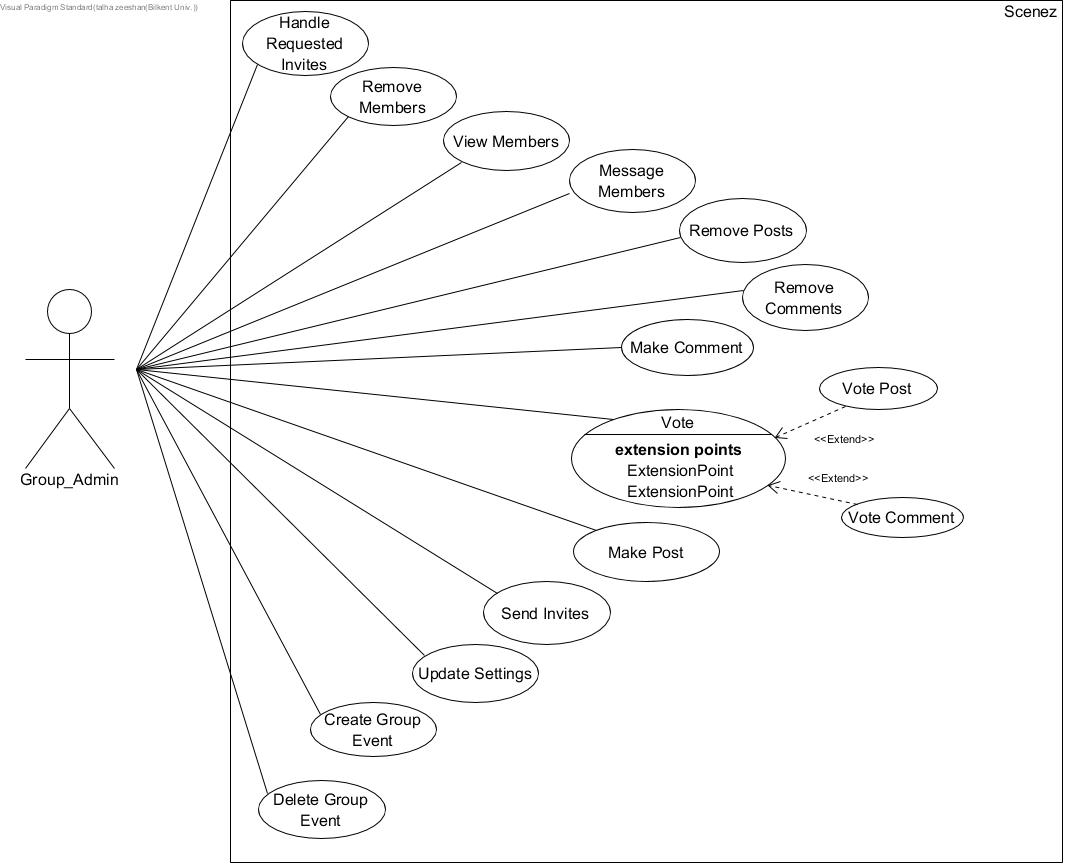
The group administrator has the authority to delete any group event. Upon such a request by the group administrator, this use case is invoked in which the event is dedeleted from the group.

Figure 2: Group Administrator Use Case Scenarios

**iii) Event\_Host**

The Event\_Host user type represents the a user of the application who has acquired event host privileges by creating their own event. Described below are various additional use case scenarios for such a user of Scenez. The described use case scenarios below for event hosts are limited to within the event of which they are the host of.

* **Send Invites**

Use Case Name: *Send Invite*

Description:

The “Send Invite” use case is invoked when the event host chooses to send invitations to his/her friends to join the group. Invitees invited by the host are exempt from approvals and invitees can immediately join the group if they reply positively (in the case of Events being invite only).

* **Handle Requests**

Use Case Name: Handle Requested Invites

Condition: Event is Invite Only

Description:

This use case is invoked when the event host approves or rejects requested invites of friends of event participants. The event host is alerted of these requests after a attendant successfully invites a friend of theirs to the event. All users not invited by the host are subjected to host approval (use case invoked).

* **View Participants**

Use Case Name: View Members

Description:

The event host has the authority to view all participants of the event. The invoking of this use case results in the system displaying all current participants of the event.

* **Update Settings**

Use Case Name: *Update Settings*

Description:

The event host has the authority to change event settings(changing name, themes, location, start time etc). In such a case, this use case is invoked. The system updates the event settings upon successful invocation.

* **View Feedback**

Use Case Name: *View Feedback*

Description:

By invoking this use case, the event host can view feedback related to their event. Upon invocation, the system will display ratings and feedbacks given by attendants of the event.

* **Remove Posts**

Use Case Name: Remove Posts

Description:

The event host has the authority to remove any posts made on their event. Upon invoking this use case, the respective post is removed from the event.

* **Remove Comments**

Use Case Name: Remove Comments

Description:

The event host has the authority to remove any comments made on event posts. Upon invoking this use case, the respective comment is removed from the event post.

* **Make Comment**

Use Case Name: Make Comment

Description:

The “Make Comment” use case is invoked when the event host chooses to comment on specific post in their event. The comment is limited to include simple text.

* **Vote**

Use Case Name: *Vote*

Description:

The “Vote Comment” use case is invoked when the host chooses to Upvote or DownVote a comment or post made within their event(s).

* **Make Post**

Use Case Name: *Make Post*

Description:

The “Create Post” use case is invoked when the host chooses to post to their event(s). The post content is limited to simple text.

* **Delete Event**

Use Case Name: *Delete Group Event*

Description:

The host has the authority to delete any of their event(s). Upon such a request by the host, this use case is invoked in which the event is deleted.

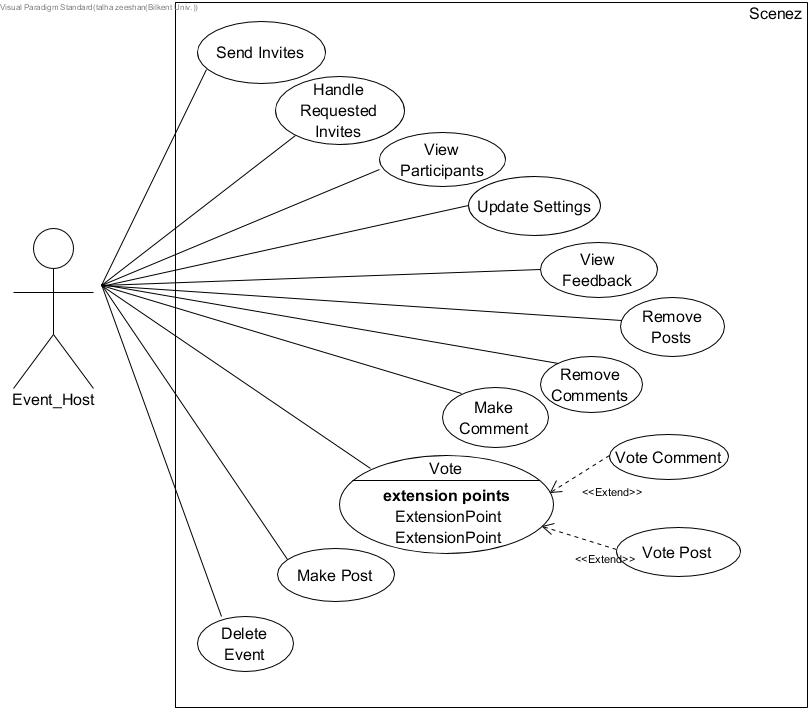
****

Figure 3: Event Host Use Case Scenarios

## **4.2 Algorithms**

Our application provides three different search criterias for the user; Search for Group, Search for Event and Search for friend. Hence, the search algorithm is of significant importance to the application. MySQL provides significant support for search related algorithms with two algorithms; Optimization and Indexing and Full Text Search. For our purposes, we will make use of the Optimization and Indexing MySQL built in algorithm. The algorithm improves the performance of the select operations by creating indexes on one or more columns tested in the query. These indexes act like pointers to the table rows hence allowing the query to quickly determine, match rows to the where clause and fetch the relate column values. MySQL allows all data types to be indexed hence providing us the ability to search our three criterias with one algorithm.

## **4.3 Data Structures**

Our database system design uses three basic types of data structures:

* Tables idenified by a unique ID will have ID’s of type int with the exception of the user table in which the ID is of type varchar, since users are uniquely identified by their email addresses.
* Textual values such as passwords, descriptions, titles and other such attributes will have types VARCHAR.
* In this database system design, many tables rely on status values in relationships between entities. Such status values have ENUM types with each type indicating a status having a certain implication.

# **User-Interface Design and corresponding SQL statements**



Figure 4: Landing Page

User Login Query

SELECT email

FROM user

WHERE email = @email and password = @password

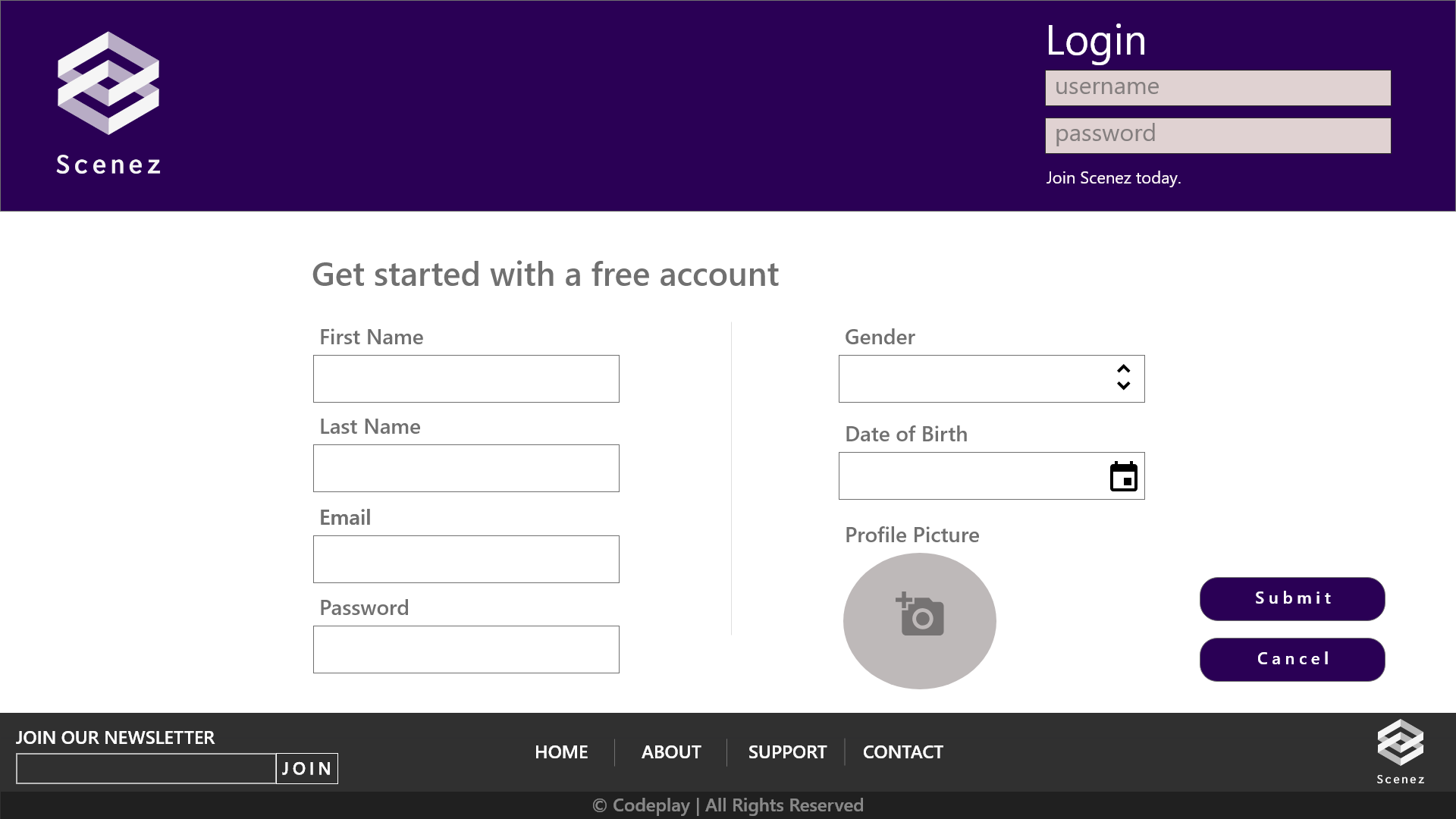


Figure 5: Sign up Page

Sign Up for new user

INSERT INTO user

VALUES (@email,@password,@first\_name,@last\_name,@dob,@gender,@-1)



Figure 6: Main Feed

Query for listing user’s joined groups

SELECT g1.name

FROM grp g1

WHERE g1.id in (SELECT group\_id

FROM grp\_invite

WHERE invitee = @email AND status =

"Accepted")

Query for listing user’s char friends

SELECT u1.name

FROM user u1

WHERE u1.id in ((SELECT user1

FROM is\_friends\_with

WHERE user2 = @email)

UNION

(SELECT user2

FROM is\_friends\_with

WHERE user1 = @email))

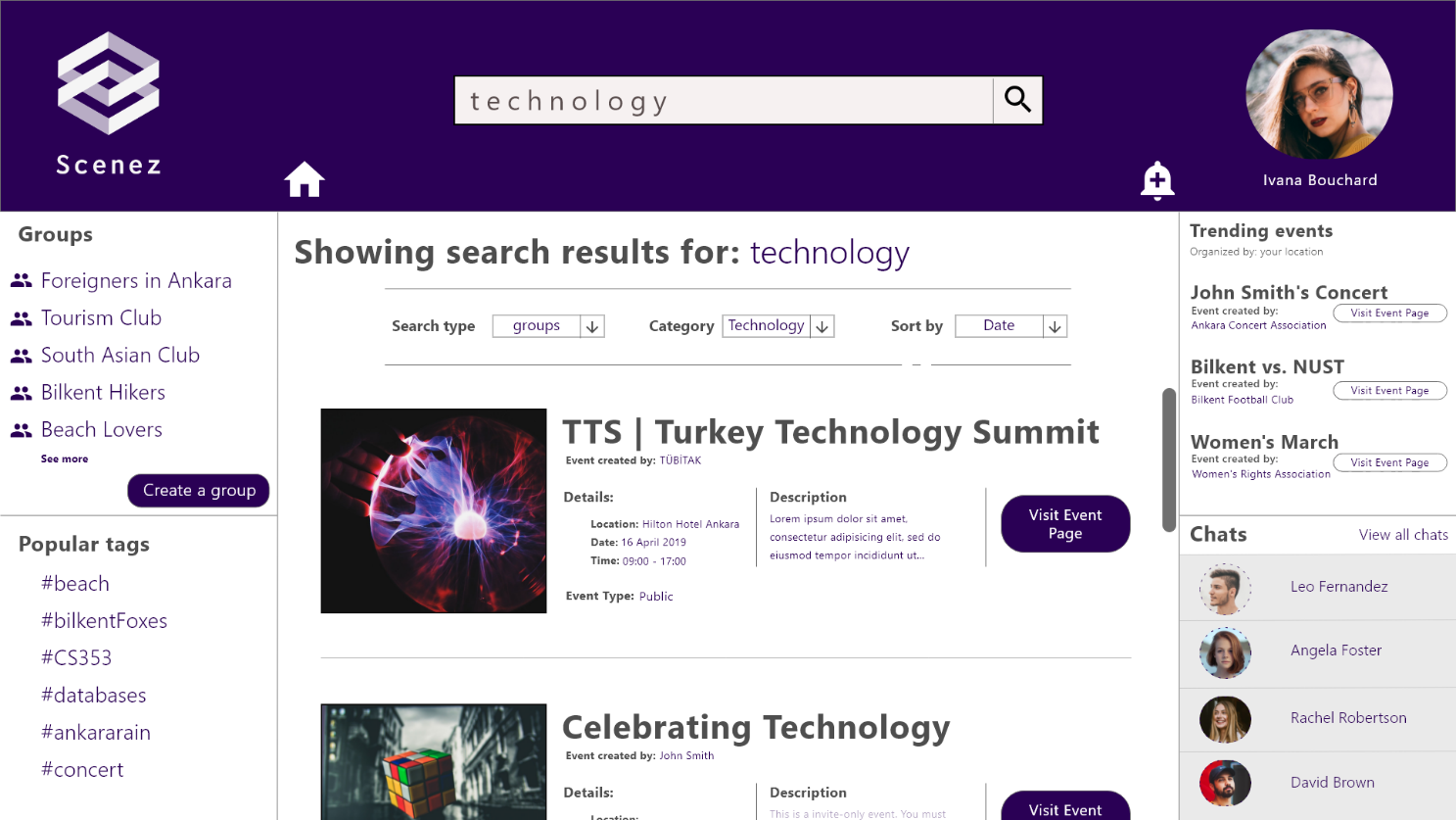


Figure 7: Search Results

Query for Search Results

SELECT ev1.name

FROM event ev1

WHERE ev1.id in ((SELECT event\_id

FROM event\_tag

WHERE tag like "%@search\_item%")

UNION

(SELECT ev2.id

FROM event\_tag ev2

WHERE ev2.name LIKE "%@searcg\_item%"

OR ev2.description LIKE "%search\_item%"))

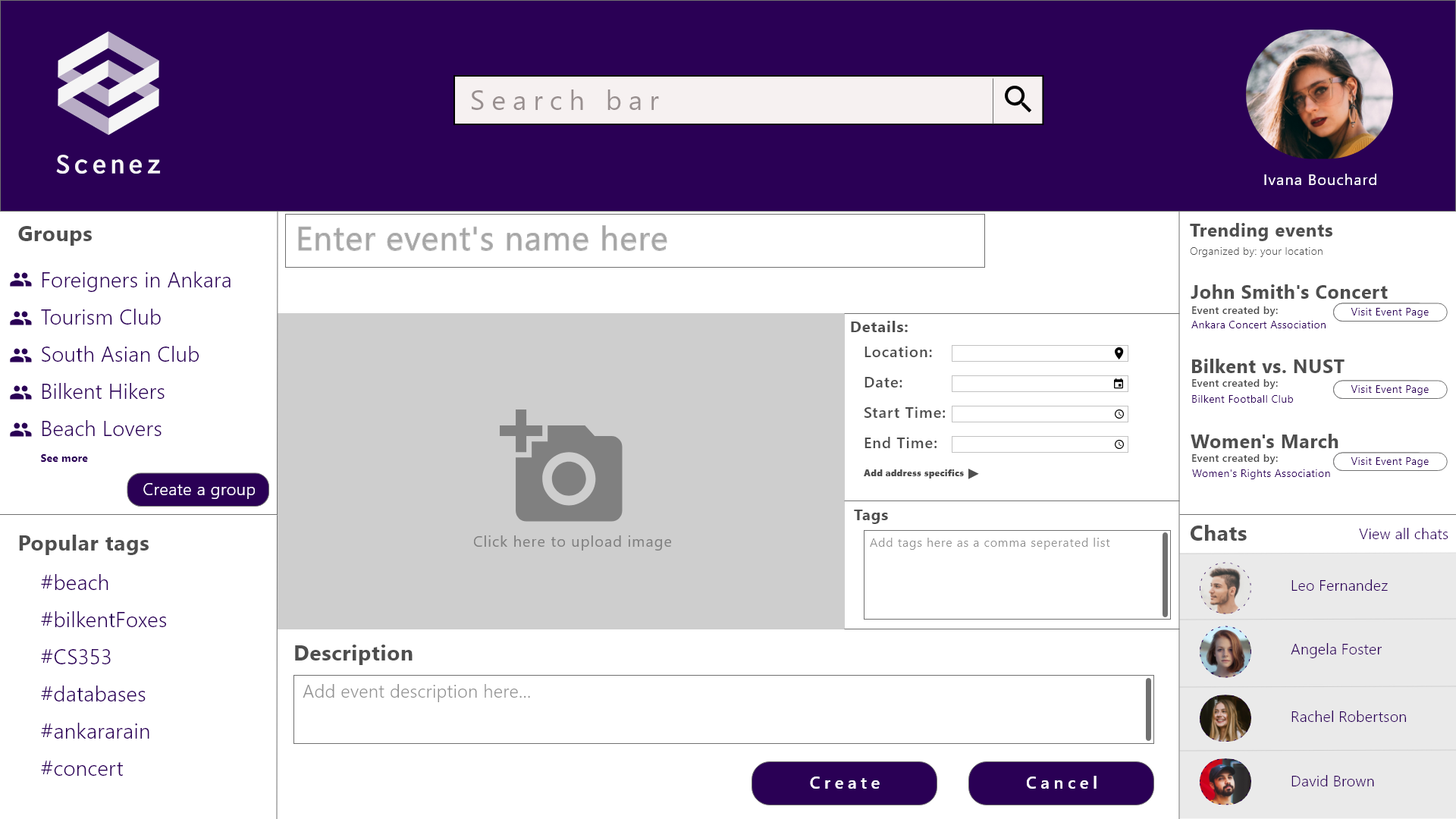


Figure 8: Create Event

Query for creating a new event

INSERT INTO event

values (1, "@Event Name", "@date.day", "@date.month", "@date.year", "@start\_time", "@duration", "TTTTTTTF", "@capacity", "@host\_email", "@cat\_name", "@location\_id")

/\*The privacy field takes eight characters which should be either ‘T’ or ‘F’ indicating whether a certain attribute of the event should be viewable to an uninvited person\*/

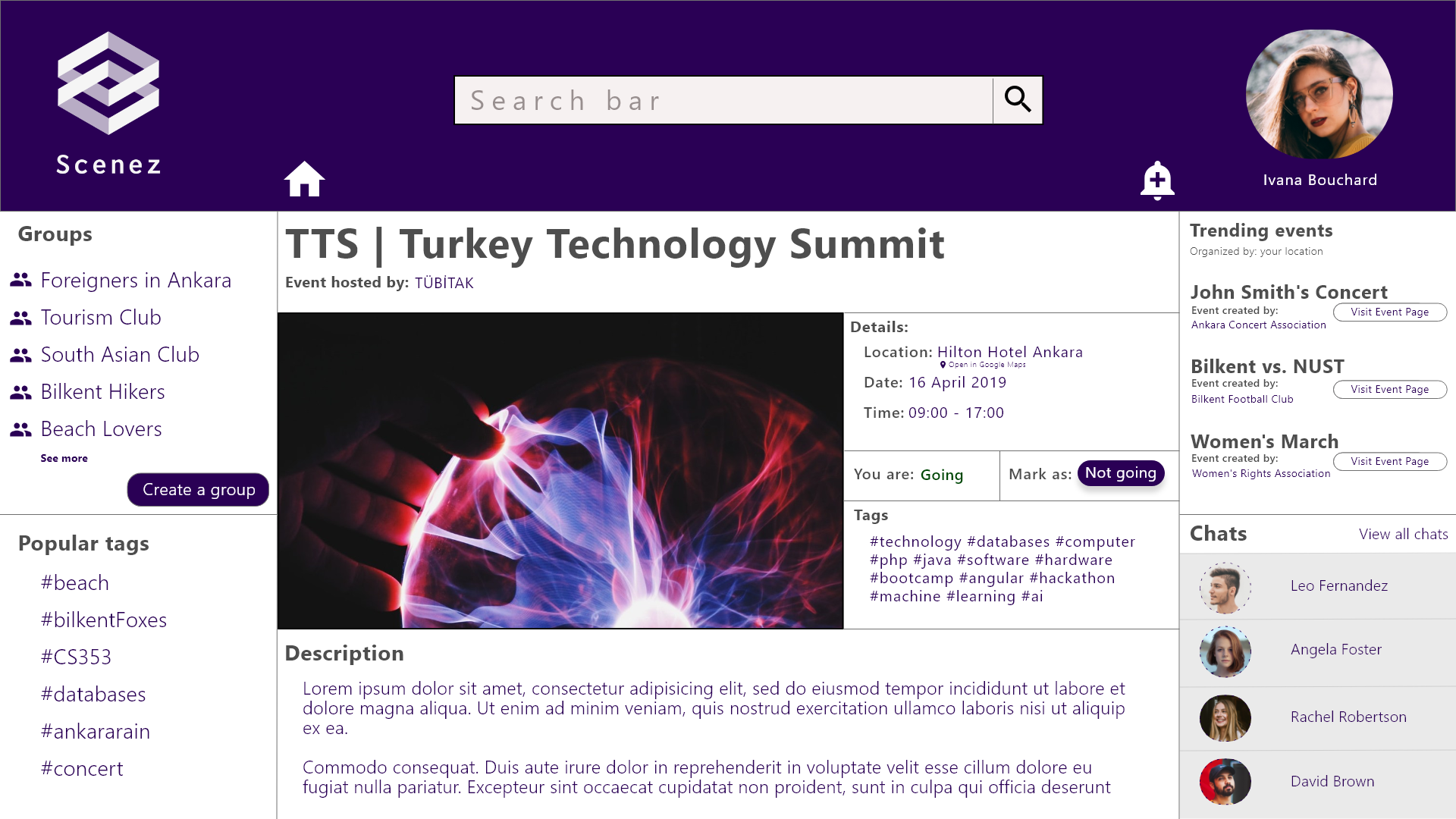


Figure 9: Event View

Query for displaying event data

CREATE VIEW event\_data AS

(SELECT id, name, day, month, year, start\_time, duration, capacity, description, email, cat\_name, location\_id

FROM event

WHERE id = "@link\_event\_id"

);

SELECT \* FROM event\_data

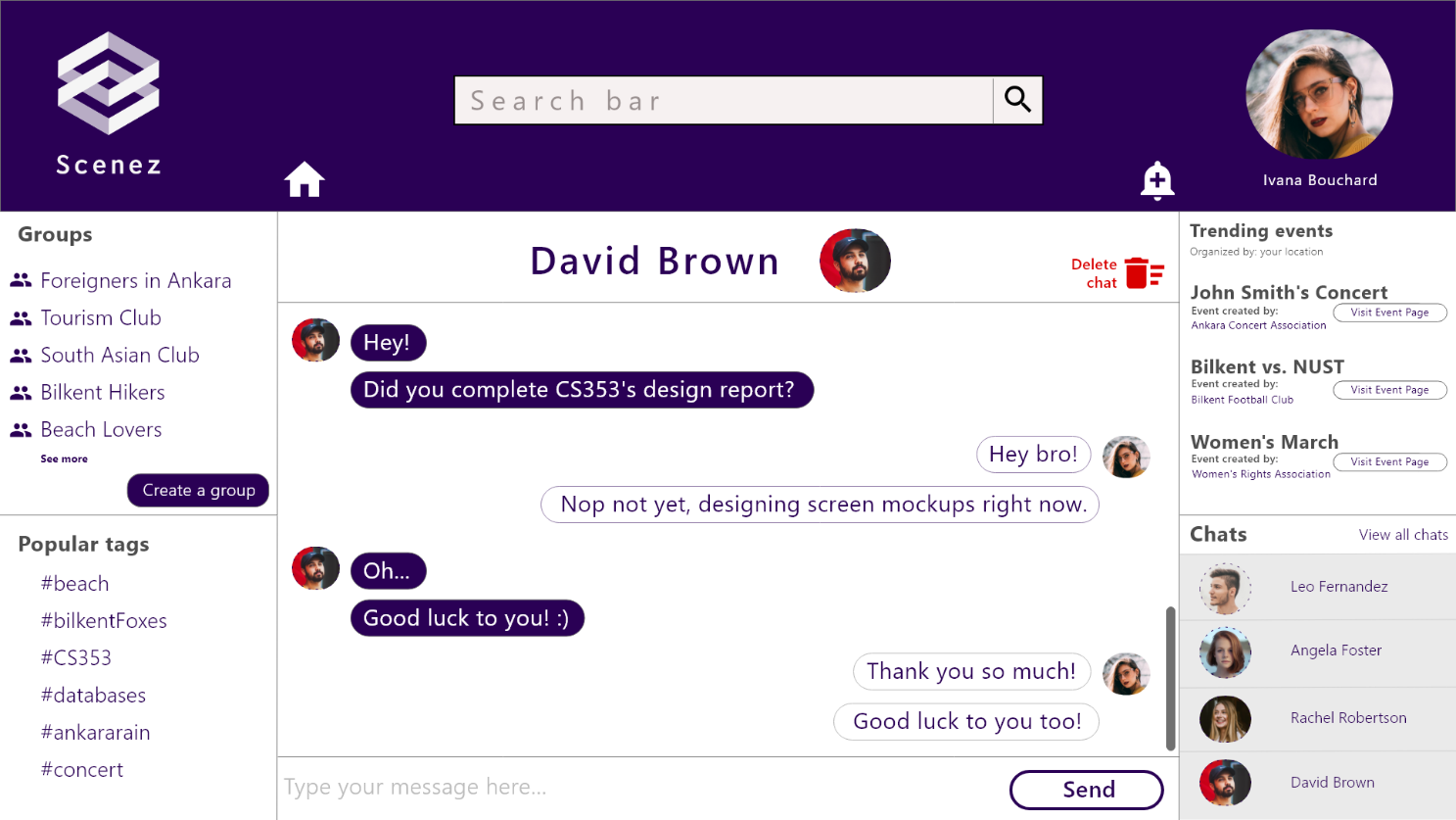


Figure 10: Chat

Query to view chat history between two users

SELECT content, sender, timestamp

FROM message, recipient\_user

WHERE (message.sender = @curr\_user AND recipient\_user.message\_id = id

AND recipient\_user.user\_id = @selected\_chat\_user) OR

(message.sender = @selected\_chat\_user AND recipient\_user.message\_id = id

AND recipient\_user.user\_id = @curr\_user)

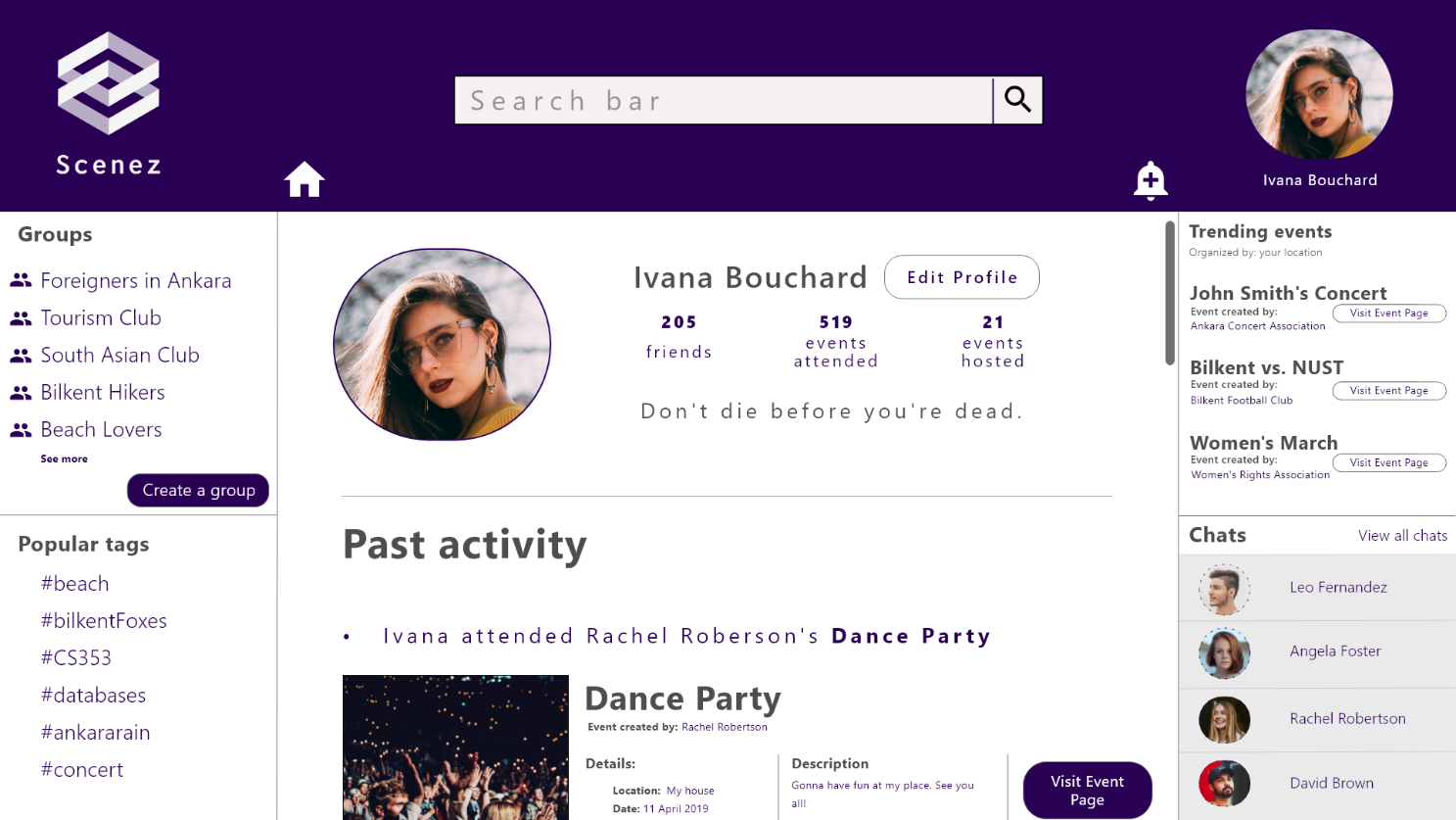


Figure 11: User Profile

Query for obtaining number of friends

SELECT count(\*)

FROM (SELECT \*

FROM is\_friends\_with

WHERE (user1 = @curr\_user OR user2 = @curr\_user) AND status = "Accepted")

Query for obtaining number of attended events

SELECT count(\*) FROM event

WHERE (event\_date < CURRENT\_DATE()

OR ( event\_date = CURRENT\_DATE()

AND start\_time <= CURRENT\_TIME()))

AND event.id in (SELECT \* FROM event\_invite

WHERE invitee = '@curr\_user'

AND status = 'Accepted')

Query for obtaining number of hosted events

SELECT count(\*)

FROM event

WHERE email = @curr\_user

Query for obtaining past events

SELECT \* FROM event

WHERE (event\_date < CURRENT\_DATE()

OR ( event\_date = CURRENT\_DATE()

AND start\_time <= CURRENT\_TIME()))

AND event.id in (SELECT \* FROM event\_invite

WHERE invitee = '@curr\_user'

AND status = 'Accepted')

# **Advanced Database Components**

## **Triggers**

* Temporarily stored net vote count is updated for that post every time a user votes on that post.
* Temporarily stored net vote count is updated for that count every time a user votes on that count.
* Whenever a user gives feedback on a past event, the average rating of that event will be updated to reflect the new value
* Whenever a host receives feedback on his/her hosted event, his/her average event rating will be updated to reflect the new values
* The previous two triggers can be implemented to act in a transitive manner in that every time a user gives feedback on an event, the average rating of the host of that event will be updated
* When a post will is deleted, all comments associated with that post will be deleted

## **Views**

* A view will be used to display events on his/her feed based on followed tags
* An 8-byte system has been implemented for events that allow a host to customize event privacies based on his/her choosing. A view will be used to hide this privacy mechanisms from other non-host users
* Separate views will be used to display data to users not members of a group and members of a group
* Trending tags and trending events views will be used to display topics and events that are most relevant to the user or are very popular

## **Stored Procedures**

* A stored procedure will be used to obtain a list of the user’s friends sorted in order of most recent messages send. This procedure will be useful since the friend list panel will be viewable by the user on all pages.
* A stored procedure will be used to obtain all chat messages of the current user with another user. The only varying aspect of this procedure will be the user with whom the chat took place

## **Constraints**

* All ratings will range between 0 and 5.
* Status values have a limited number of values having values “Accepted”, “Pending” and “Rejected” for friend request statues. Group and Event join requests have a specialized set of status values “User-Pending” and “Admin-Pending”.

# **Implementation Plan**

We hope to implement out project using a front-end, back-end and database layered approach. On the front end we will use HTML, JavaScript, and BootStrap to provide the user with an aesthetically pleasing experience and visually appealing features along with an easy-to-navigate interface.

We hope to use JSP (Java Server Pages) and a MySQL database backend along with MySQL Workbench 8.0 and local MySQL servers to easily manipulate its structure and contents.

# **Website**

All documentation regarding the project will be uploaded at the following repository:

<https://github.com/ahmed-umair/scenez>