

Project Design Report

CS 353 - Database Systems

Group 7
Figali Taho
Dias Alymbekov
Elena Cina
Erkam Berker Senol

October 2017



Department of Engineering
Faculty of Computer Engineering

Contents

1	Project Description	3
2	Revised ER-Diagram	4
2.1	Changes Made to the Model	4
2.2	Revised ER Diagram	5
3	Relational Schemas	6
3.1	User	6
3.2	Admin	6
3.3	Moderator	7
3.4	Content	7
3.5	Post	8
3.6	Comment	8
3.7	Category	9
3.8	Topic	9
3.9	Message	9
3.10	Category_Topic	10
3.11	Vote	10
3.12	Follows	11
4	Functional Components	11
4.1	Use Cases	11
4.2	Algorithms	14
4.3	Data Structures	14
5	User Interface Design and SQL Statements	15
5.1	Sign up	15
5.2	Sign in	16
5.3	Become admin	17
5.4	Post a post	19
5.5	View homepage	20
5.6	Comment on a post	21
5.7	Search	22
5.8	Follow user	23
5.9	Edit post	24
5.10	Upvote/Downvote post	25
5.11	Send Message	27
5.12	Delete Post	28
6	Advanced Database Components	29
6.1	Views	29
6.1.1	Default homepage for user and nonusers of system	29
6.2	Triggers	29
6.3	Stored Procedures	29
6.4	Constraints	29
7	Implementation Details	30

1 Project Description

The project is aimed to become a system for a social discussion website similar to Reddit. There will be many topics which the users can open themselves and on which every user can discuss with each other via commenting. Each topic belongs to a certain category. Categories will be predefined prior to being opened or users can create a new category. Users of this system can follow other users and should be able to comment others posts or comments on a particular discussion topic and also they can send messages to each other. There will be 3 user types: normal user, moderator and admin.

A database will be used to store the information regarding users, categories and topics, subtopics and comments. This database will allow the user to search information regarding other users and categories. For example, the user can select to see a list of topics that fall under a certain category via the web-page interface. This will result in a query being used to project only the topics under the category, sorted according to time. The user can also sort the list of topics by other attributes such as "popularity", derived from the number of people who have upvoted the topic, or "latest", derived from the date when the topic is posted.

Similarly, the users will also be stored in the database, and will be identified by their unique user name, which is the primary key. The user information will also consist of attributes such as joined date, password. Moreover, the various relationships that will be present, such as when the user opens a new category or creates a new topic will be supported by the entities of the relationships.

The database is going to be the integral part of the system and most of the idea is based around the database. It will be loaded onto the webpage which will be used as the interface by which the user access the database. However, the user will not be able to see the database structure and instead will only see relevant information displayed on the webpage. Hence the queries that will be used to retrieve the information and alter the tables will be automatically generated upon the selection of the user on the webpage interface.

2 Revised ER-Diagram

2.1 Changes Made to the Model

- New entity "Moderator" is created. The entity inherits attributes from "User" entity.
- New entity "Admin" is created. The entity inherits attributes from "User" entity.
- "Sub_Topic" transformed into weak entity.
- New relation "follows" is created between "User" and another "User" with cardinality many-to-many
- New relation "sends" is created between "User" and another "User" with attributes: timestamp, message with cardinality many-to-many.
- New relation "admins" is created between "Admin" and "User" with cardinality many-to-many, where "User" must be adminned by at least one user
- New relation "admins" is created between "Admin" and "Content" with cardinality many-to-many, where "Content" must be adminned by at least one user
- New relation "admins" is created between "Admin" and "Category" with cardinality many-to-many, where "Category" must be adminned by at least one user
- New relation "searches" is created between "User" and "Content" with cardinality many-to-many
- "comments" is transformed into ternary relation.
- The cardinality relation "contains" is transformed, add constraint of presence of minimum one topic for each post.
- The cardinality relation "moderates" is transformed, add constraint of presence of minimum one moderator for each category.

2.2 Revised ER Diagram

Below we provided the revised version of the design of the database for social discussion platform.

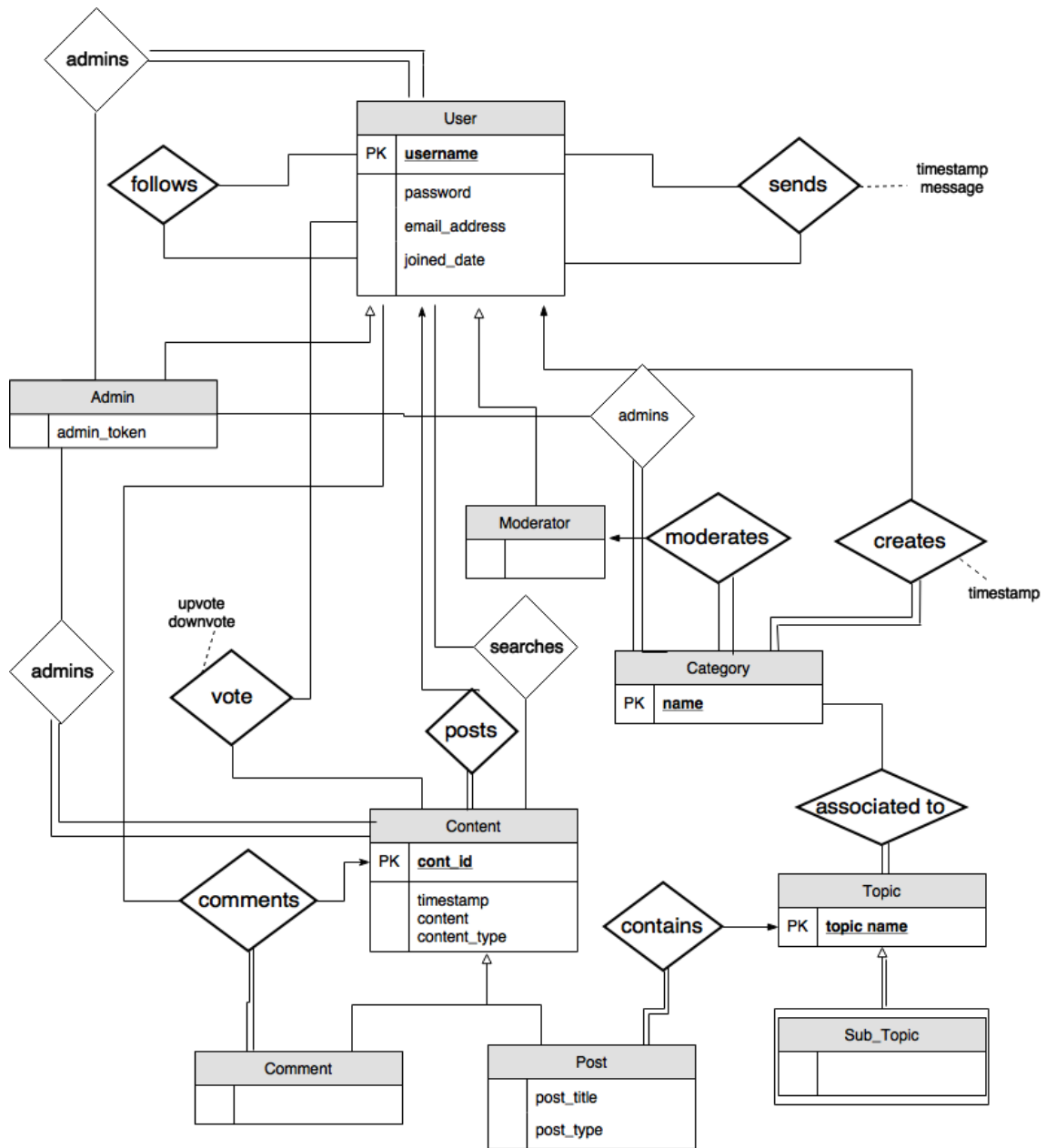


Figure 1: E/R diagram revisited.

3 Relational Schemas

3.1 User

- **Relational Model:**

User(username , password , email_address , joined_date)

- **Functional Dependencies:**

username \Rightarrow password, email_address, joined_date

email \Rightarrow username

- **Candidate Keys:**

{username}, {email_address}

- **Normal Form:**

3NF

- **Table Definition:**

```
CREATE TABLE User (  
    username varchar(32) NOT NULL,  
    password varchar(32) NOT NULL,  
    email_address varchar(32) NOT NULL,  
    joined_date date,  
    PRIMARY KEY(username),  
    CHECK(email_address like '%@_%._%')) ENGINE = InnoDB;
```

3.2 Admin

- **Relational Model:**

Admin(username , password , email_address , joined_date, admin_token)

- **Functional Dependencies:**

username \Rightarrow password, email_address, joined_date, admin_token

email \Rightarrow username

- **Candidate Keys:**

{username}, {email_address}

- **Normal Form:**

3NF

- **Table Definition:**

```
CREATE TABLE Admin (  
    username varchar(32) NOT NULL,  
    password varchar(32) NOT NULL,  
    email_address varchar(32) NOT NULL,  
    joined_date date,  
    admin_token varchar(32) NOT NULL,  
    PRIMARY KEY(username),  
    CHECK(email_address like '%@_%._%')) ENGINE = InnoDB;
```

3.3 Moderator

- **Relational Model:**

Moderator(username , password , email_address , joined_date)

- **Functional Dependencies:**

username \Rightarrow password, email_address, joined_date

email \Rightarrow username

- **Candidate Keys:**

{username}, {email_address}

- **Normal Form:**

3NF

- **Table Definition:**

```
CREATE TABLE Moderator (  
    username varchar(32) NOT NULL,  
    password varchar(32) NOT NULL,  
    email_address varchar(32) NOT NULL,  
    joined_date date,  
    PRIMARY KEY(username),  
    CHECK(email_address like '%@_%._%')) ENGINE = InnoDB;
```

3.4 Content

- **Relational Model:**

Content(cont_id, timestamp, content, content_type, username, net_vote)

- **Functional Dependencies:**

cont_id \Rightarrow timestamp, content, content_type, username, net_vote

- **Candidate Keys:**

{cont_id}

- **Normal Form:**

3NF, BCNF

- **Table Definition:**

```
CREATE TABLE Content (  
    cont_id numeric(10,0) AUTO_INCREMENT,  
    timestamp date,  
    content varchar(800) NOT NULL,  
    content_type varchar(10) NOT NULL,  
    username varchar(32) NOT NULL,  
    net_vote numeric(10,0),  
    PRIMARY KEY(cont_id)) ENGINE = InnoDB;
```

3.5 Post

- **Relational Model:**

Post(cont_id, post_title, post_type, belongs)

- **Functional Dependencies:**

cont_id \Rightarrow post, post_title, post_type, belongs

- **Candidate Keys:**

{cont_id}

- **Normal Form:**

3NF, BCNF

- **Table Definition:**

```
CREATE TABLE Post (  
    cont_id numeric(10,0),  
    post_title varchar(50) NOT NULL,  
    post_type varchar(50) NOT NULL,  
    belongs varchar(50) NOT NULL,  
    PRIMARY KEY(cont_id),  
    FOREIGN KEY (belongs) REFERENCES Topic(topic_name),  
    FOREIGN KEY (cont_id) REFERENCES Content(cont_id))  
ENGINE = InnoDB;
```

3.6 Comment

- **Relational Model:**

Comment(cont_id, username, dst_cont_id)

- **Functional Dependencies:**

cont_id \Rightarrow username, dst_cont_id

- **Candidate Keys:**

{cont_id}

- **Normal Form:**

3NF, BCNF

- **Table Definition:**

```
CREATE TABLE Comment (  
    cont_id numeric(10,0),  
    username varchar(32) NOT NULL,  
    dst_cont_id numeric(10,0),  
    PRIMARY KEY(cont_id, username),  
    FOREIGN KEY (cont_id) REFERENCES Content(cont_id),  
    FOREIGN KEY (dst_cont_id) REFERENCES Content(cont_id),  
    FOREIGN KEY (username) REFERENCES Persons(username))  
ENGINE = InnoDB;
```


3.7 Category

- **Relational Model:**

Category(name)

- **Functional Dependencies:**

name \Rightarrow name

- **Candidate Keys:**

{name}

- **Normal Form:**

3NF, BCNF

- **Table Definition:**

```
CREATE TABLE Category (  
    name varchar(50) NOT NULL,  
    PRIMARY KEY(name)) ENGINE = InnoDB;
```

3.8 Topic

- **Relational Model:**

Topic(topic_name)

- **Functional Dependencies:**

topic_name \Rightarrow topic_name

- **Candidate Keys:**

{topic_name}

- **Normal Form:**

3NF, BCNF

- **Table Definition:**

```
CREATE TABLE Topic (  
    topic_name varchar(50) NOT NULL,  
    PRIMARY KEY(topic_name)) ENGINE = InnoDB;
```

3.9 Message

- **Relational Model:**

Topic(dst_name, rcv_name, timestamp, message)

- **Functional Dependencies:**

dst_name, rcv_name, timestamp \Rightarrow message

- **Candidate Keys:**

{dst_name, rcv_name, timestamp}

- **Normal Form:**

3NF, BCNF

- **Table Definition:**

```
CREATE TABLE Message (
    dst_name varchar(32) NOT NULL,
    rcv_name varchar(32) NOT NULL,
    timestamp date,
    message varchar(32) NOT NULL,c
    PRIMARY KEY(dst_name, rcv_name, timestamp),
    FOREIGN KEY (dst_name) REFERENCES User(username),
    FOREIGN KEY (rcv_name) REFERENCES User(username))
ENGINE = InnoDB;
```

3.10 Category_Topic

- **Relational Model:**

Category_Topic(category_name, topic_name)

- **Functional Dependencies:**

category_name, topic_name \Rightarrow category_name, topic_name

- **Candidate Keys:**

{category_name, topic_name}

- **Normal Form:**

3NF, BCNF

- **Table Definition:**

```
CREATE TABLE Category_Topic (
    category_name varchar(50) NOT NULL,
    topic_name varchar(50) NOT NULL,
    PRIMARY KEY(dst_name, rcv_name, timestamp),
    FOREIGN KEY (category_name) REFERENCES Category(name),
    FOREIGN KEY (topic_name) REFERENCES Topic(topic_name))
ENGINE = InnoDB;
```

3.11 Vote

- **Relational Model:**

Vote(username, cont_id, vote)

- **Functional Dependencies:**

username, cont_id, vote \Rightarrow username, cont_id, vote

- **Candidate Keys:**

{username, cont_id, vote}

- **Normal Form:**

3NF, BCNF

- **Table Definition:**

```
CREATE TABLE Vote (
    username varchar(32) NOT NULL,
```

```

    cont_id numeric(10,0) NOT NULL,
    vote boolean NOT NULL,
    PRIMARY KEY(username, content_id, vote),
    FOREIGN KEY (username) REFERENCES User(username),
    FOREIGN KEY (content_id) REFERENCES Content(content_id))
ENGINE = InnoDB;

```

3.12 Follows

- **Relational Model:**
Follows(follower, following)
- **Functional Dependencies:**
follower, following \Rightarrow follower, following
- **Candidate Keys:**
{follower, following}
- **Normal Form:**
3NF, BCNF
- **Table Definition:**

```

CREATE TABLE Message (
    follower varchar(32) NOT NULL,
    following varchar(32) NOT NULL,
    PRIMARY KEY(follower, following),
    FOREIGN KEY (follower) REFERENCES User(username),
    FOREIGN KEY (following) REFERENCES User(username))
ENGINE = InnoDB;

```

4 Functional Components

4.1 Use Cases

User:

- Users should be able to send messages to other users.
- Users should be able to follow other users.
- Users should be able to post content in the form of posts.
- Users should be able to post comments under posts and under another comments in the form of replies.
- Users should be able to upvote or downvote content.
- Users should be able to create new categories.
- Users should be able to edit their posted content.
- Users should be able to view content.

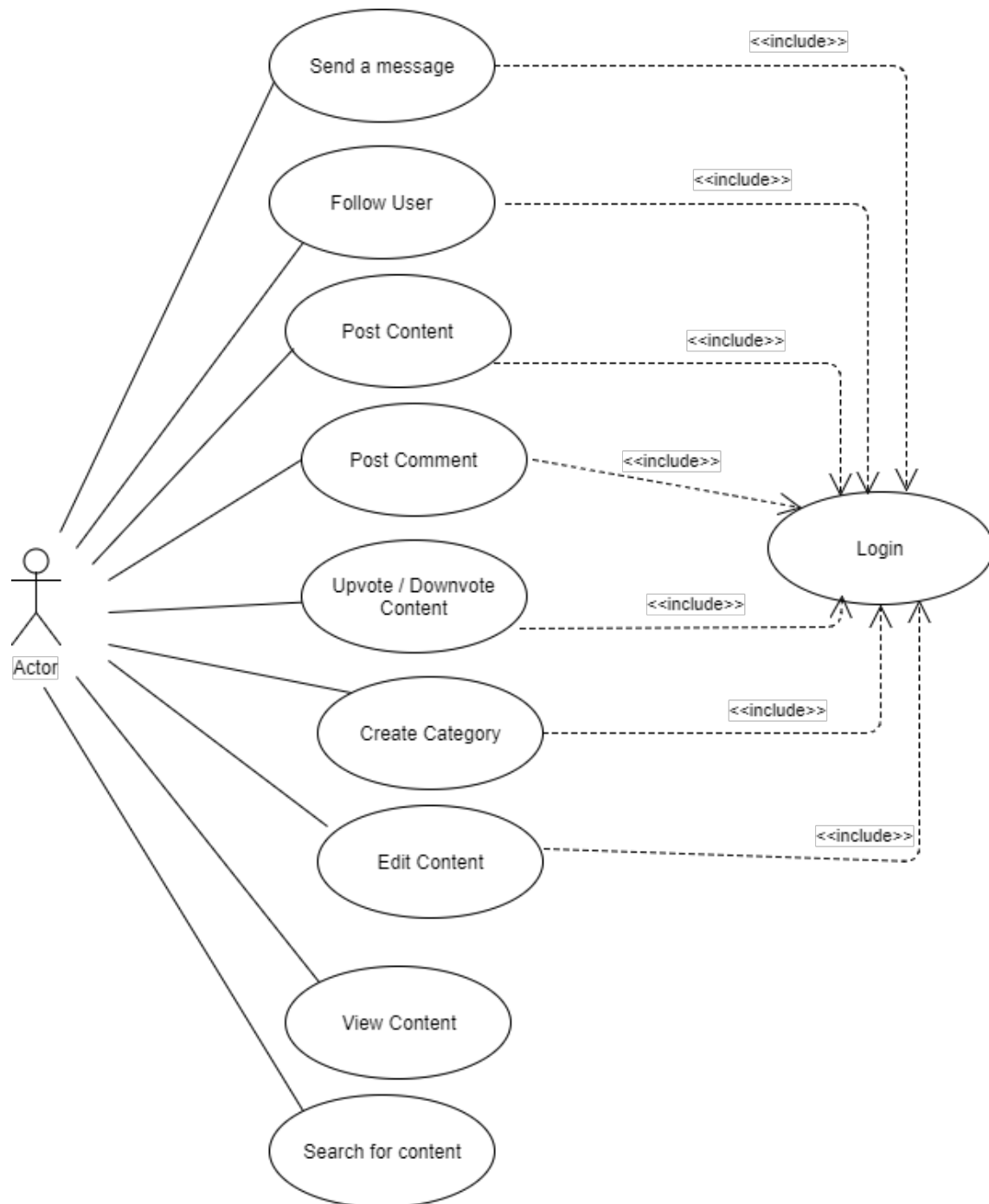


Figure 2: Use case diagram for user.

- Users should be able to search for content.

Moderator:

- Moderators should be able to delete existing posts in their categories.
- Moderators should be able to ban users in their categories.

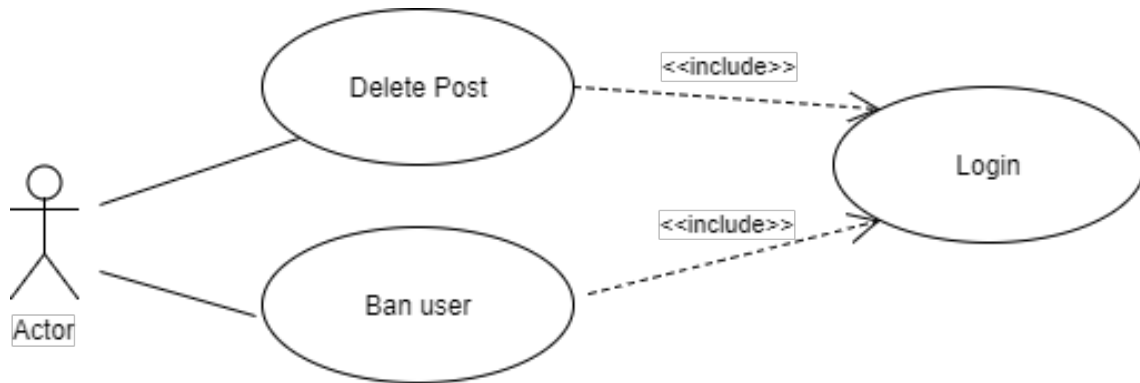


Figure 3: Use case for the moderator.

Admin:

- Admins should be able to delete users.
- Admins should be able to delete categories.
- Admins should be able to delete content.

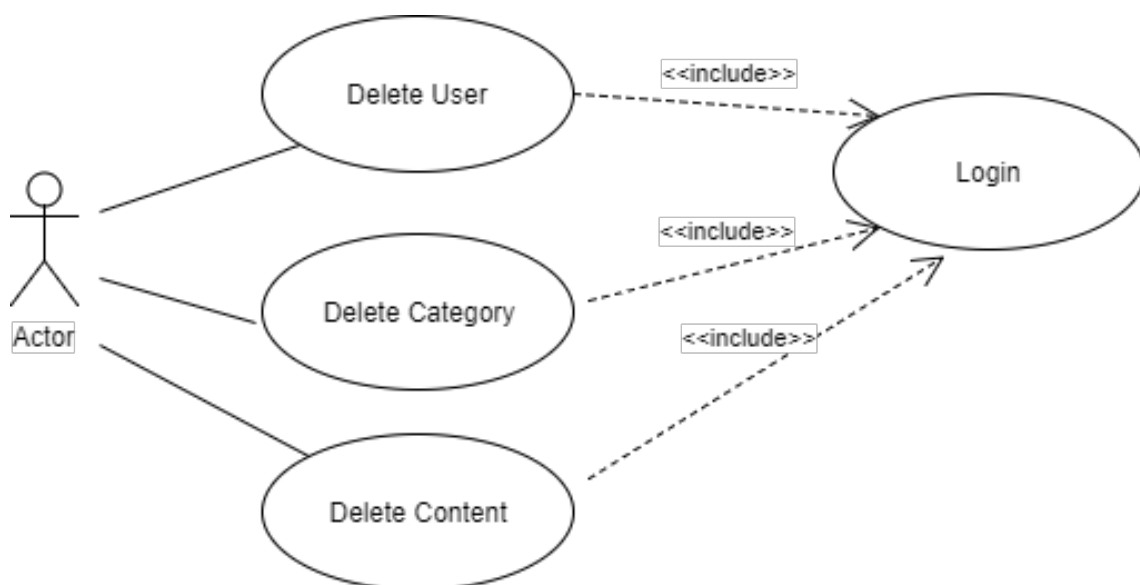


Figure 4: Use case diagram for the admin.

4.2 Algorithms

The main purpose of the platform is to store content posted by users and offer functionality that will allow users to navigate through that content. One of the main functionalities is the ability to search through the content. In order to facilitate this functionality we will offer refined search. We aim to implement a dynamic search in the data layer, that will use an inverted index table of the content to create a list with weighted ranking. The schema for the inverted index table is provided below:

```
words(id, tablename, what, sen, idx, word, pos, lemma);
```

In the table above id represents id of the content, tablename represents the origin table of the given word, what represents the origin of the word in the content(title or body), sen represents the ordered number of the sentence in the text, idx represents the index of the word in the sentence, word represents the word itself, pos represents part of speech and lemma represents the basic form of the word.

```
drop procedure if exists new;
delimiter //
CREATE PROCEDURE weighted_list(param VARCHAR(1000))
BEGIN
SET @s = 'select word, sum(rank) from words,
      (select id, sum(score) rank from
        (select distinct id, 1 score from words where word = “”;
        SET @s = concat(@s,replace (param,',',',') union all
          select distinct id, 1 score from words where word = “”) );
        SET @s = concat(@s,’’ ) t1
        group by id) t2
        where words.id = t2.id
      group by word;
’);
```

Using the procedure above we can build a weighted list of words in the content and further refine it and provide results ordered in descending order and limited by some threshold. What is more, words table gives us opportunity for experiments, using this table we can further refine our search to use more advanced computations such TF-IDF score instead of frequency. Also we might try to delete stop words before applying weighting techniques, to improve our frequency based results. For this purpose, we may use another stop_words table that will contain a set of stop words.

4.3 Data Structures

In our relational schemas we use String type, Numeric type and Time type. String type is required to store any character-composed attributes such as usernames, ids. Attributes with Numeric domain is used in order to store numeric data such as id primary keys. Time type will be used to keep time values such as timestamp.

A Random Token Generator will also be used to provide a token for when a user request to gain administrator rights.

All of the data will be indexed to ensure a higher performance for all our queries.

5 User Interface Design and SQL Statements

5.1 Sign up

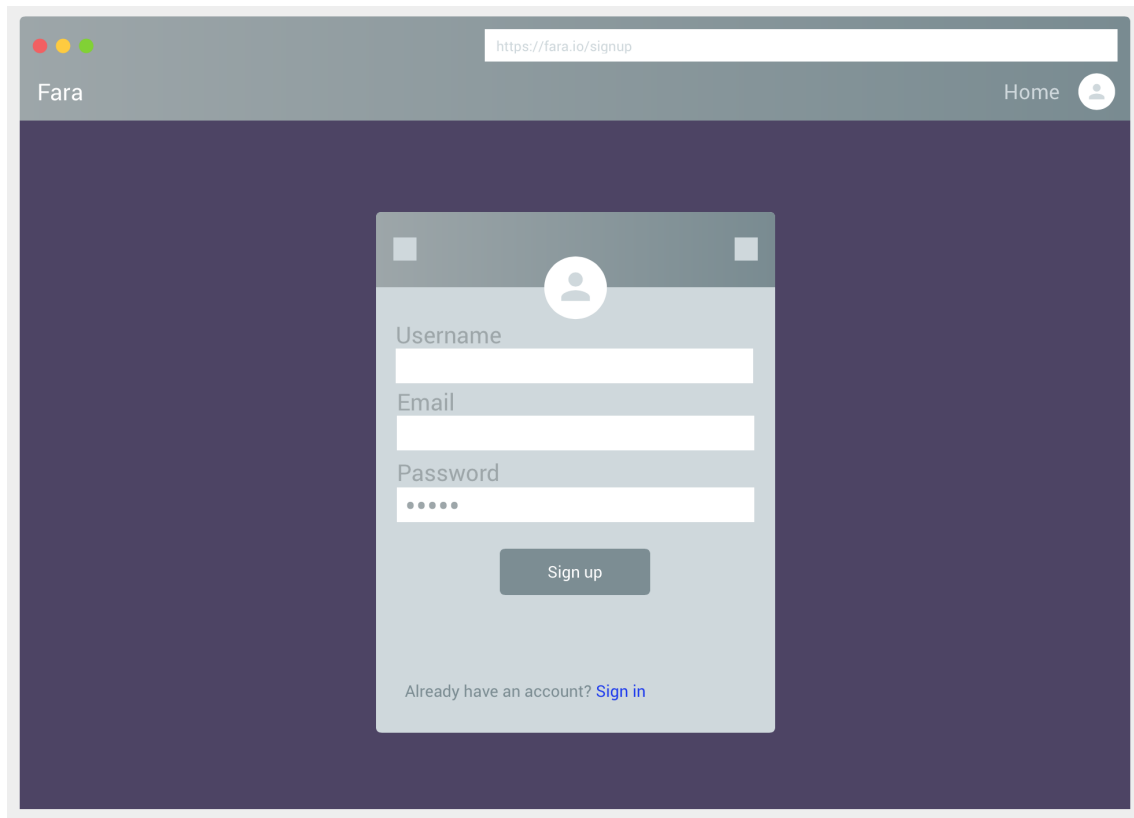


Figure 5: Signup page.

- **Inputs:**
@username, @email, @password
- **Process:**
User can create a new account using his/her email and specifying a user name and a password.
The new account is added to the database.
- **SQL Statements:**

```
INSERT INTO User USER (@username, @password, @email, now());
```

5.2 Sign in

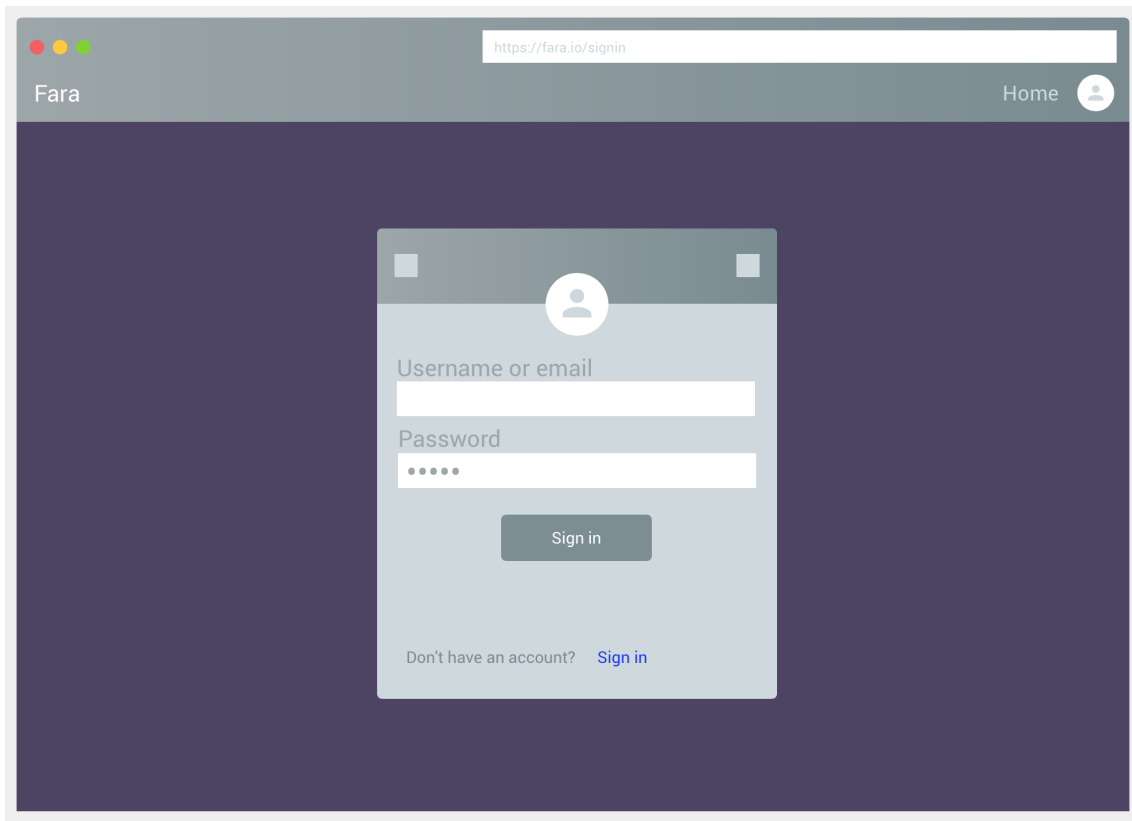


Figure 6: Sign in page.

- **Inputs:**

@username, @password

- **Process:**

User enters his/her username or email and password in order to log in.

- **SQL Statements:**

```
SELECT username
FROM User
WHERE (User.username = @username OR User.email = @email) AND User.password = @password;
```


5.3 Become admin

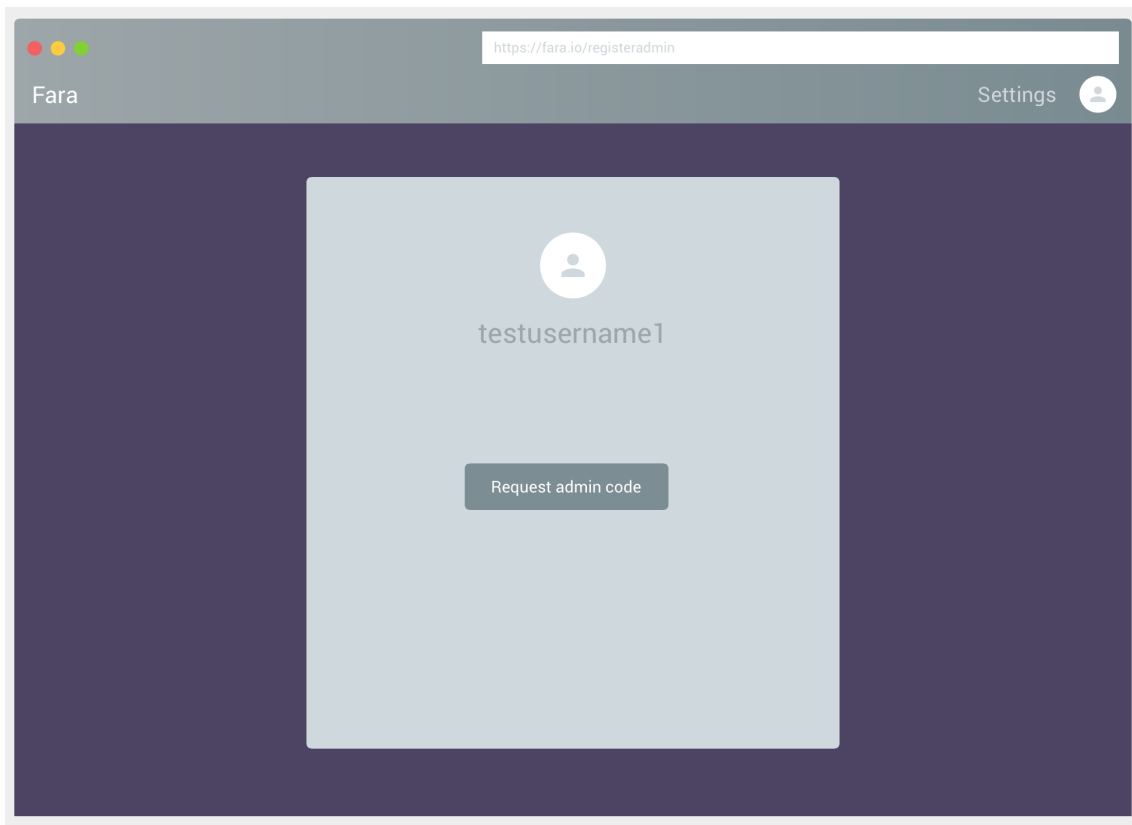


Figure 7: Admin requesting a admin code.

- **Inputs:**

@username, @email, @password, @admin_pass

- **Process:**

User requests to become admin by requesting a code, which will be sent to his/her email address. Then user will input this code, and if the user has been active in the system for the past 2 years, then he/she can become an admin and a page like the one on the right will be shown.

- **SQL Statements:**

```
INSERT INTO Admin VALUES(@username, @email, @password, now(), @admin_pass);
```

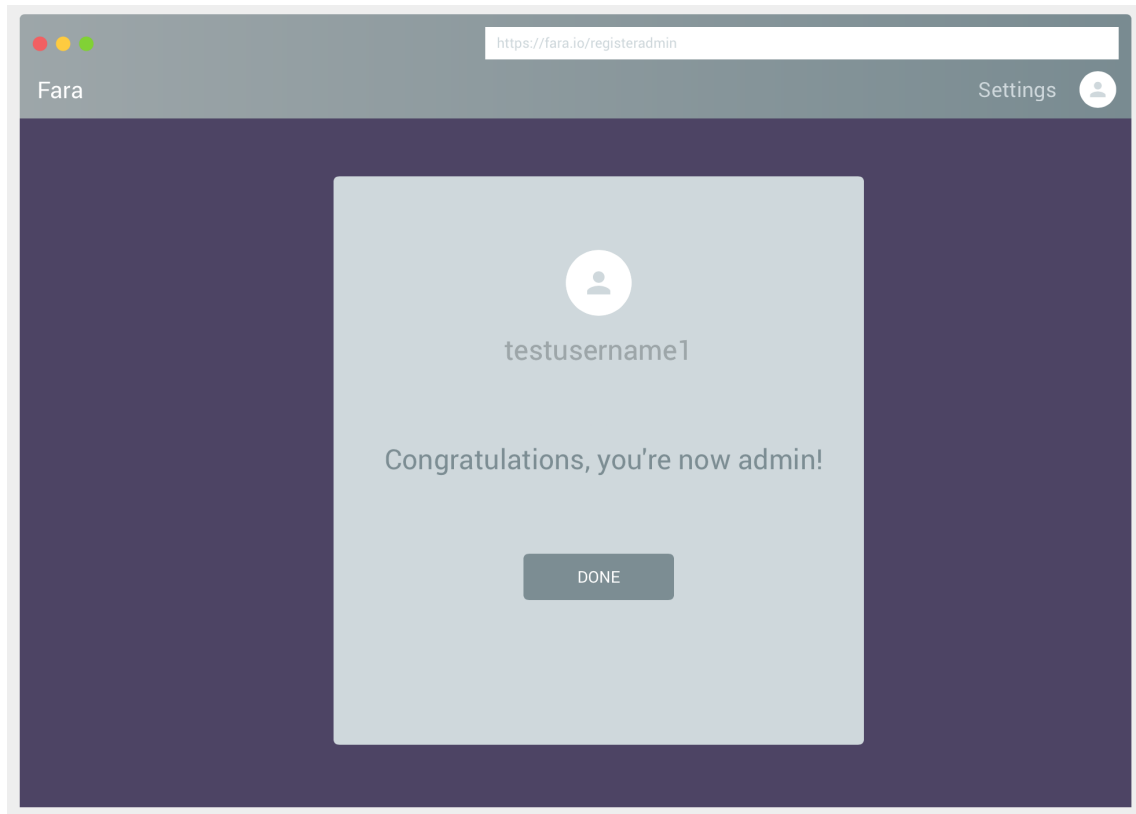
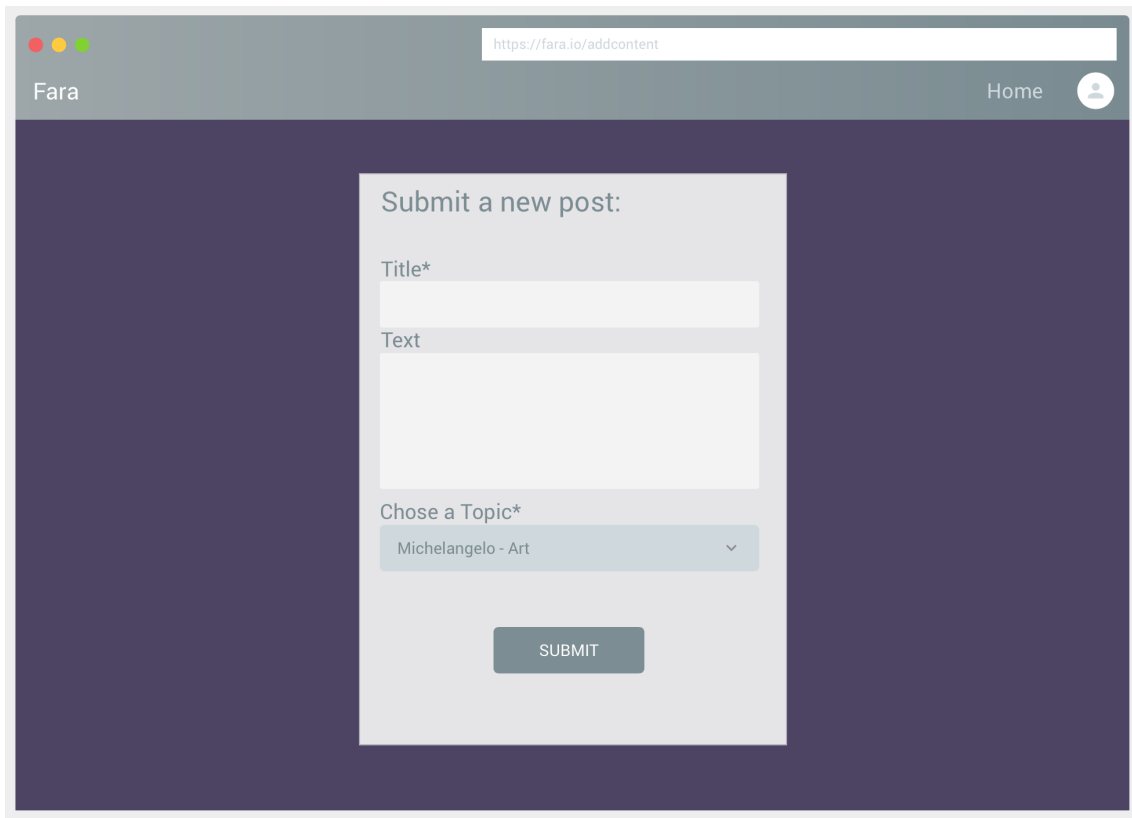


Figure 8: User is now admin.

5.4 Post a post



The screenshot shows a web browser window with the address bar displaying `https://fara.io/addcontent`. The browser's title bar shows 'Fara' on the left and 'Home' with a user profile icon on the right. The main content area has a dark purple background. In the center, there is a light gray rectangular form titled 'Submit a new post:'. The form contains three input fields: 'Title*' (a single-line text box), 'Text' (a multi-line text area), and 'Chose a Topic*' (a dropdown menu). The dropdown menu is currently open, showing 'Michelangelo - Art' with a downward arrow. Below these fields is a dark gray button labeled 'SUBMIT'.

Figure 9: Submitting a post.

- **Inputs:**

@title, @content, @topic, @username

- **Process:**

User can post a new post by choosing a relevant topic and providing a title to the post. The new post will be added to the database

- **SQL Statements**

```
INSERT INTO Content VALUES (NULL, now(), @content, POST, @username, 0);  
INSERT INTO Post VALUES (SELECT LAST_INSERT_ID(), @title, @topic);
```

5.5 View homepage

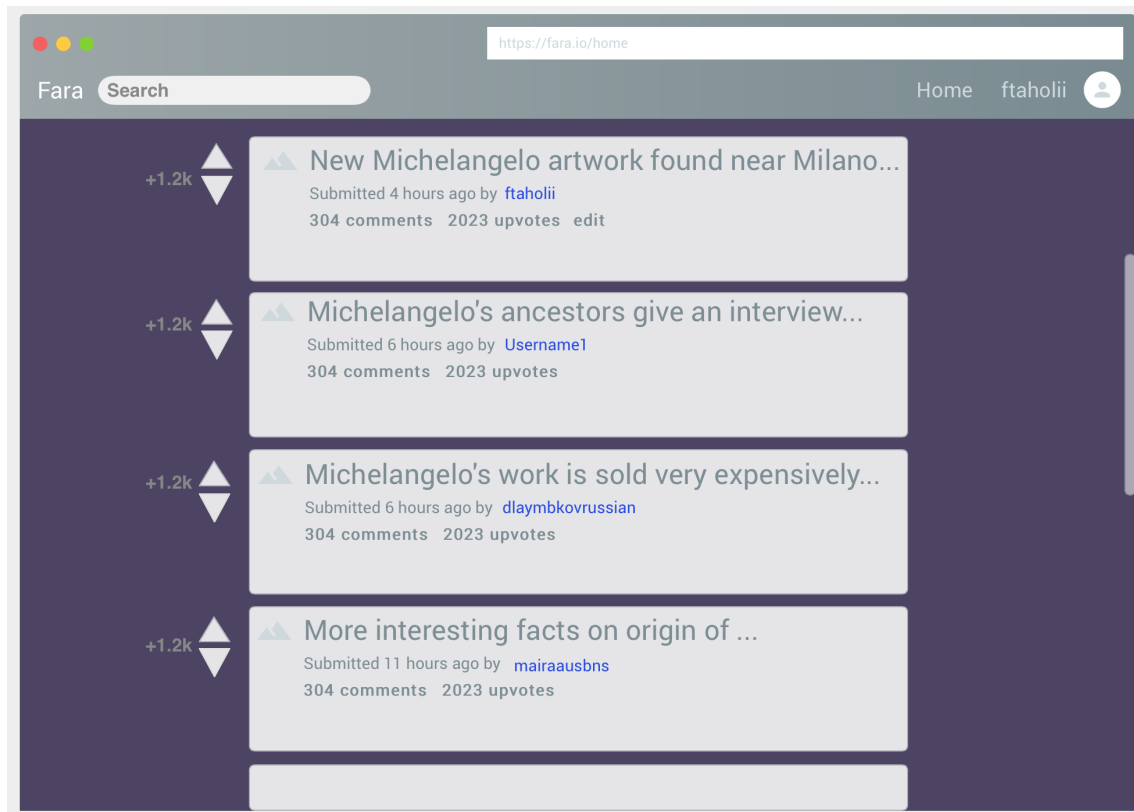


Figure 10: Homepage of a user.

- **Process:**
Homepage of a user.
- **SQL Statement:**

```
SELECT *  
FROM Show_Posts
```

(P.S: Show_Posts is the name of the view in 6.1.1)

5.6 Comment on a post

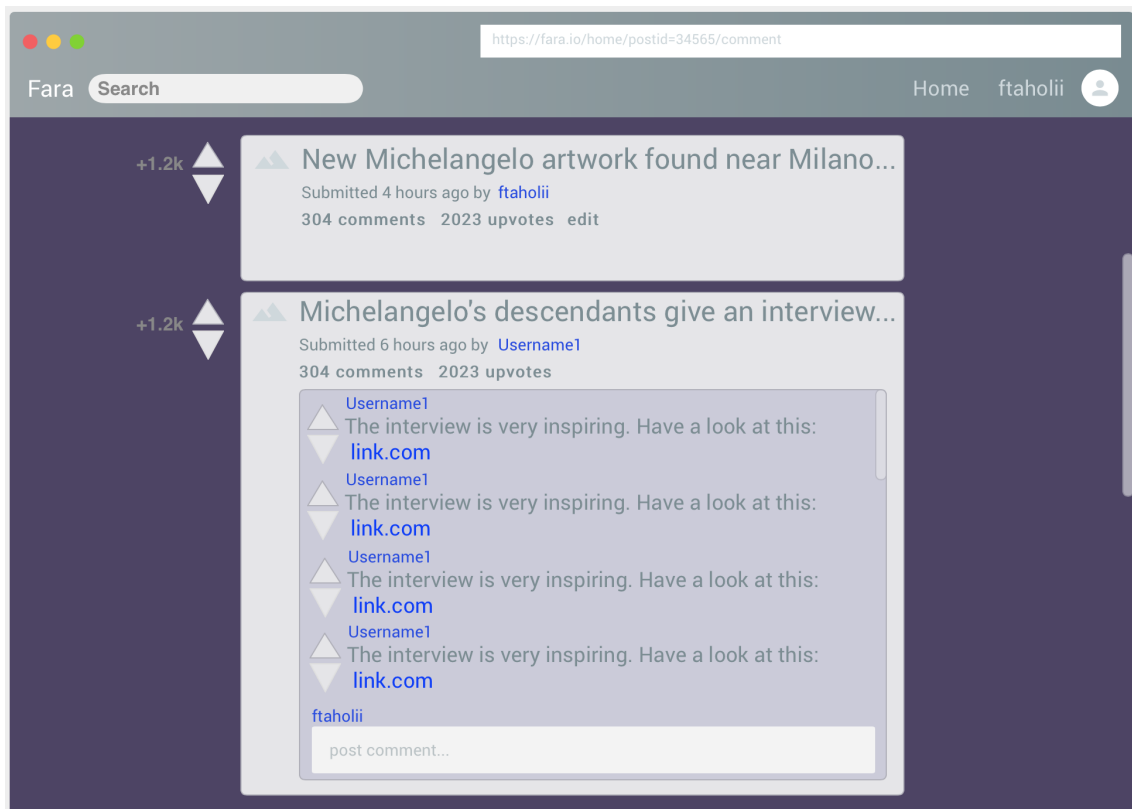


Figure 11: Commenting on a post.

- **Inputs:**

@comment, @dest_cont_id

- **Process:**

Any user can make a comment to a specific post

- **SQL Statement:**

```
INSERT INTO Content VALUES (NULL, now(), @comment, COMMENT, username);
INSERT INTO Comment VALUES (SELECT LAST_INSERT_ID(), username, @dest_cont_id);
```

5.7 Search

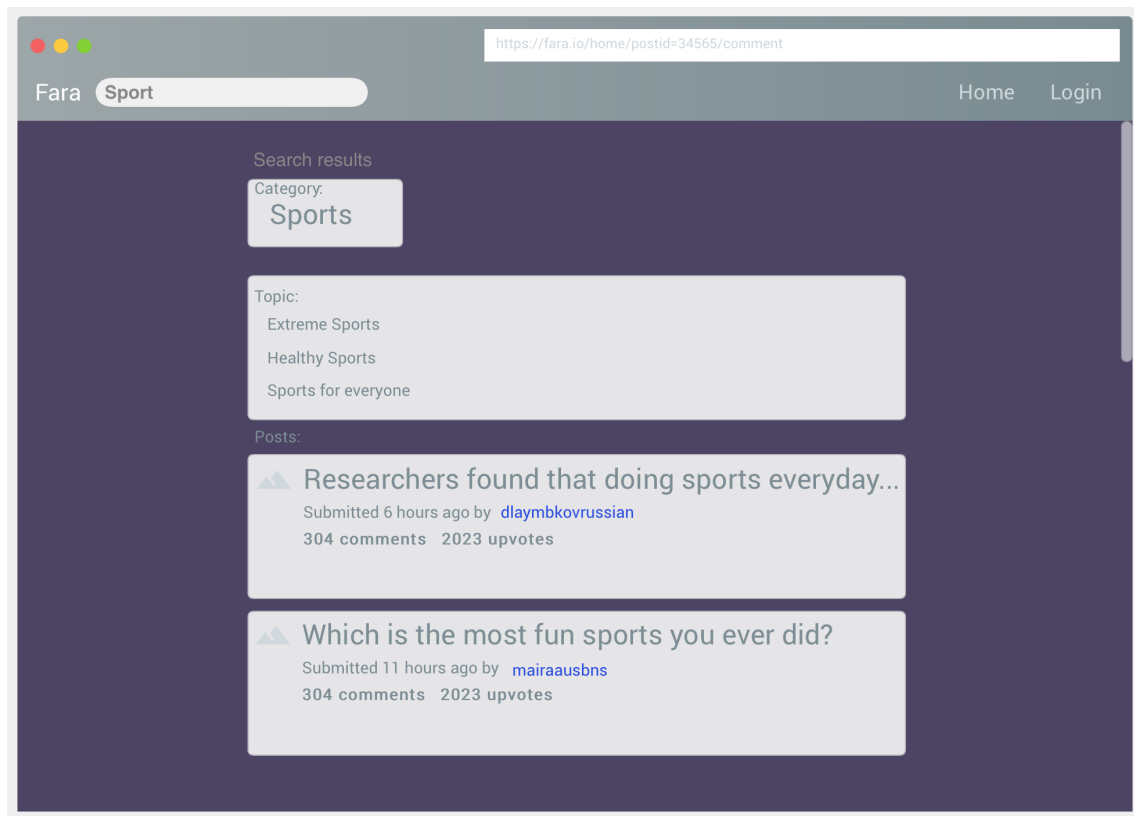


Figure 12: Searching on Fara.

- **Inputs:**

@searched_word

- **Process:**

User can search on the website to see relevant posts, topics or categories. The results of the search will be taken from database and shown to user in the order of Category, Topic and Post. Posta are ordered by their dates, most recent post is shown at top.

- **SQL Statements**

```
SELECT name
FROM Category
WHERE name LIKE %@searched_word%
ORDER BY name;
```

```
SELECT topic_name
FROM TOPIC
WHERE topic_name LIKE %@searched_word%
ORDER BY topic_name;
```

```
SELECT P.post_title, P.post_type, C.timestamp, C.content, C.username
```

```

FROM Post AS P, CONTENT AS C
WHERE C.cont_id = P.cont_id
AND P.post_title LIKE %@searched_word%
ORDER BY C.timestamp DESC;

```

(P.S: These queries will be merged in the application layer for better UI.)

5.8 Follow user

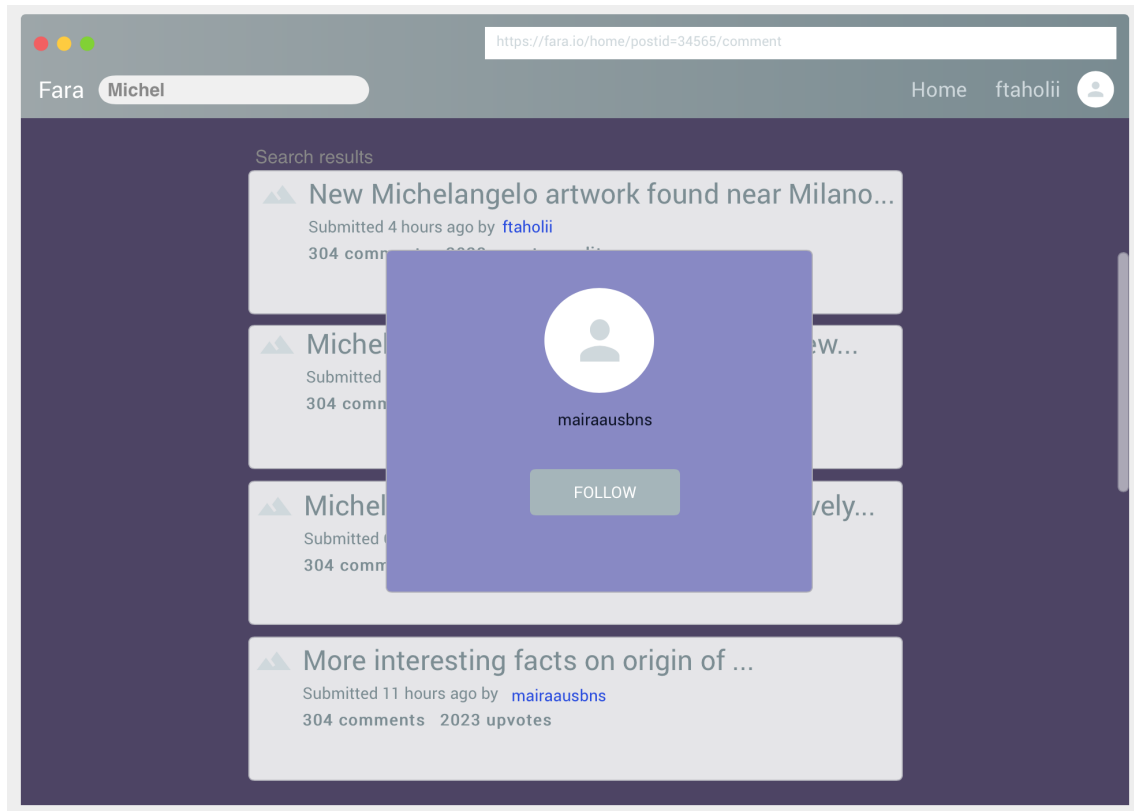


Figure 13: Following a user.

- **Inputs:**
@following
- **Process:**
Users can follow other users.
- **SQL Statements**

```

INSERT INTO Follows (Follower, Following)
VALUES (@username, @following);

```

5.9 Edit post

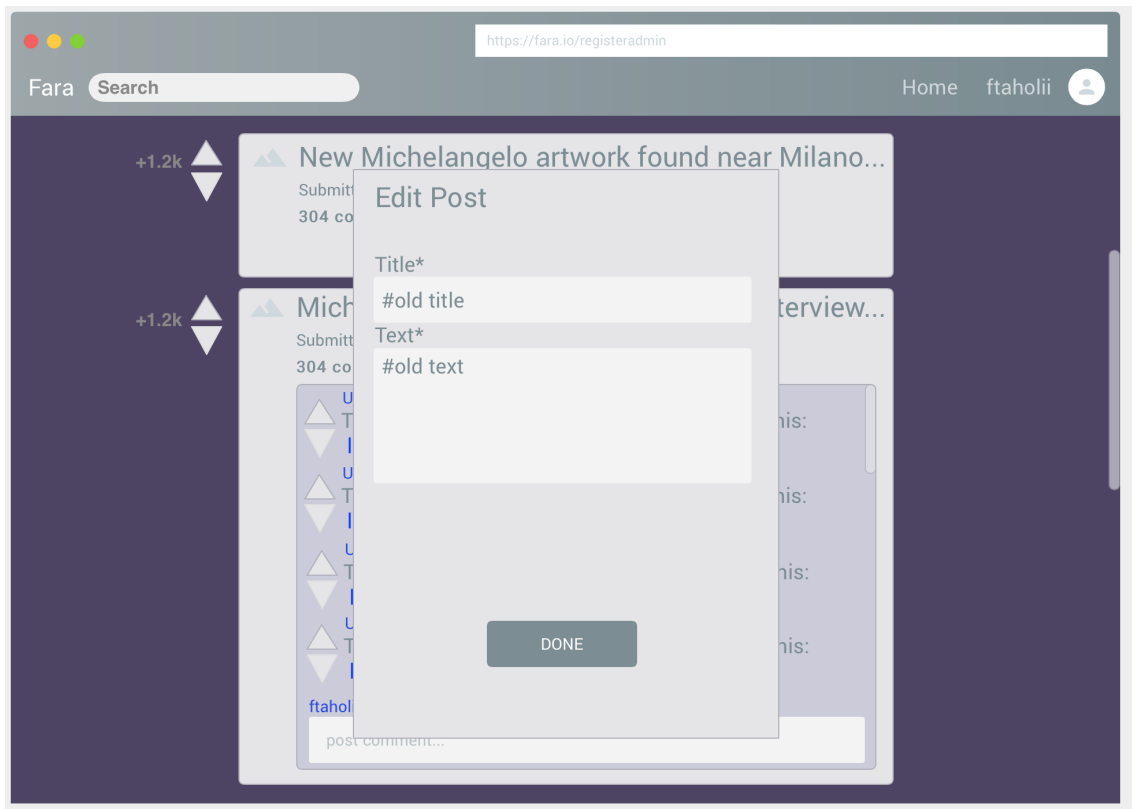


Figure 14: Editing a post.

- **Inputs:**
@cont_id, @post_title, @post_content
- **Process:**
Users can edit their post content or post title.
- **SQL Statements**

```
UPDATE Content, Post
SET Post.post_title = @post_title, Content.content = @post_content
WHERE Content.cont_id = @cont_id AND Post.cont_id = @cont_id;
```


5.10 Upvote/Downvote post

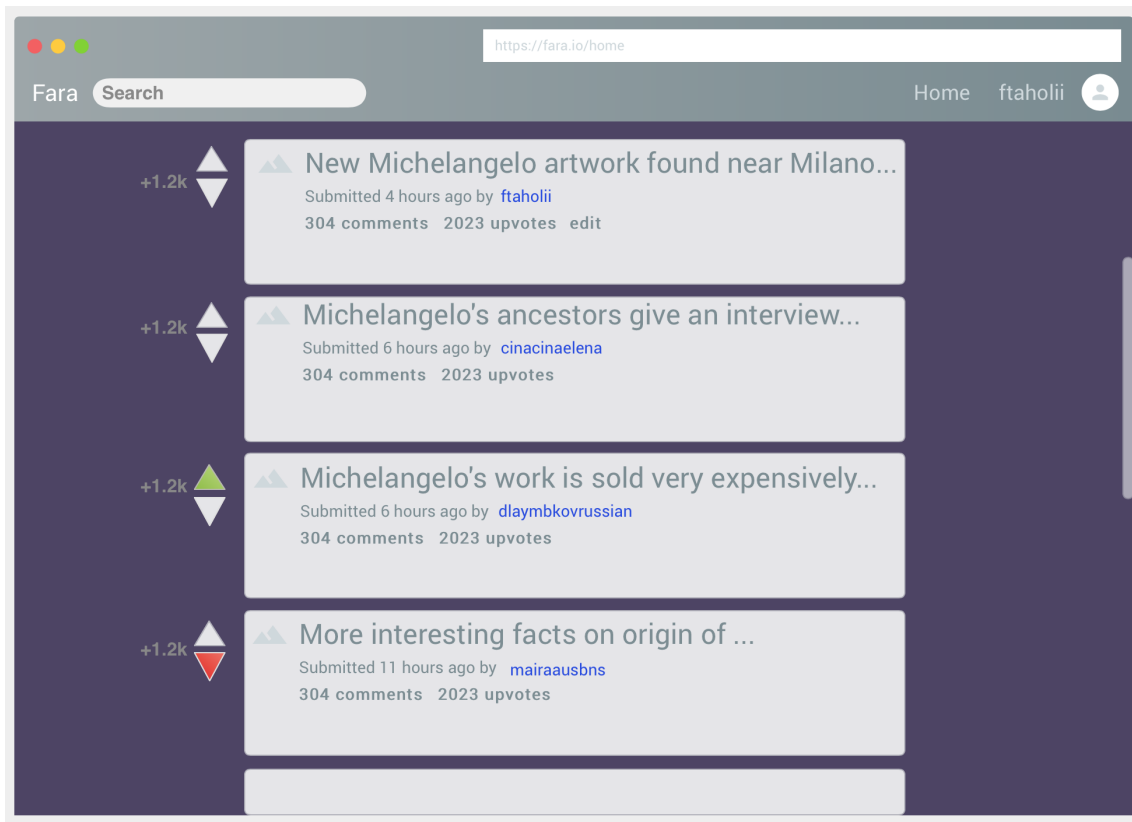


Figure 15: Upvote/Downvote a post.

- **Inputs:**

@cont_id, @username, @vote

- **Process:**

Users can upvote or downvote the posts.

- **SQL Statements**

If user downvoted before and upvotes:

```
UPDATE Vote, Content
SET Vote.vote = 'true', Content.net_vote = Content.net_vote + 2
WHERE Content.cont_id = @cont_id
AND Vote.cont_id = @cont_id;
```

If user upvoted before and downvotes:

```
UPDATE Vote, Content
SET Vote.vote = 'false', Content.net_vote = Content.net_vote - 2
WHERE Content.cont_id = @cont_id
AND Vote.cont_id = @cont_id;
```

If user didn't vote before and upvotes:

```
INSERT INTO Vote (cont_id, username, vote)
VALUES (@cont_id, @username, @vote);
```

```
UPDATE Content
SET Content.content = Content.content + 1
WHERE Content.cont_id = @cont_id;
```

If user didn't vote before and downvotes:

```
INSERT INTO Vote (cont_id, username, vote)
VALUES (@cont_id, @username, @vote);
```

```
UPDATE Content
SET Content.content = Content.content - 1
WHERE Content.cont_id = @cont_id;
```

5.11 Send Message

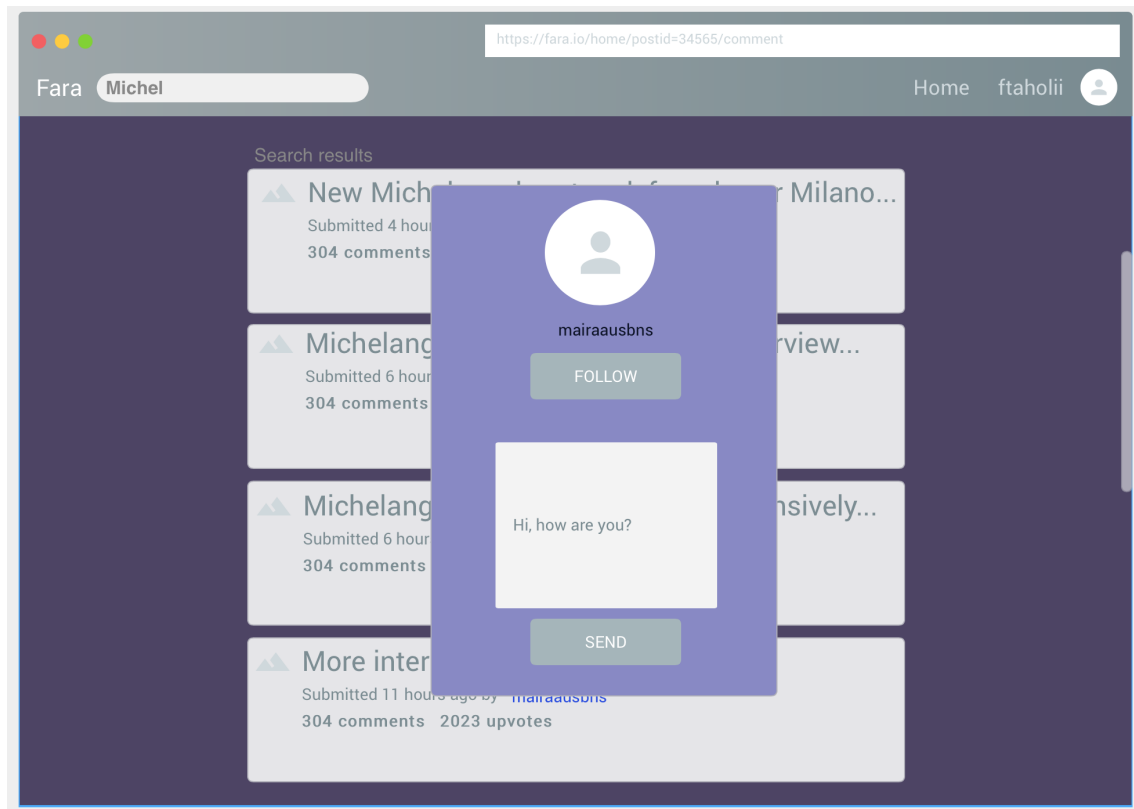


Figure 16: Sending a message to another user.

- **Inputs:**
@username, @dest_user, @message_text
- **Process:**
A user can send a message to another user. t
- **SQL Statements**

```
INSERT INTO (dst_name, rcv_name, timestamp, message)
VALUES (@dest_user, @username, now(), @message_text);
```

5.12 Delete Post

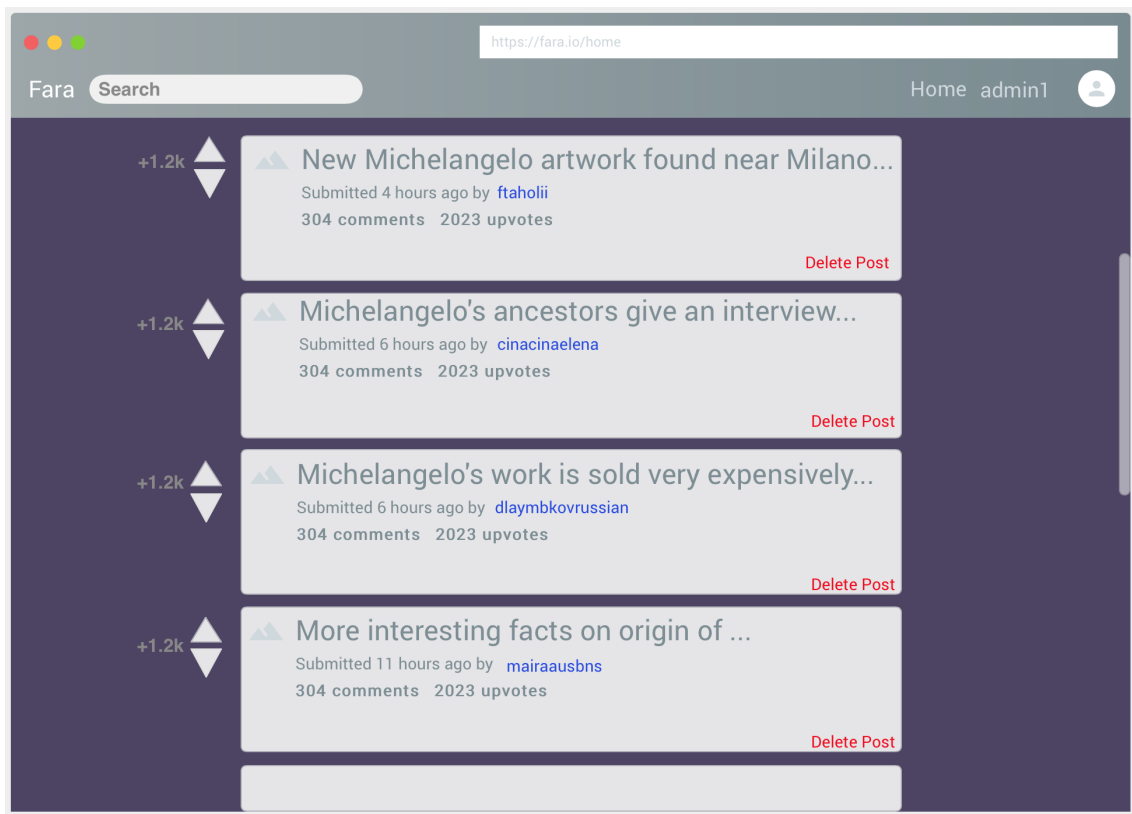


Figure 17: Admin can delete a post when delete post clicked.

- **Inputs:**
@cont_id
- **Process:**
Admin can delete posts. t
- **SQL Statements**

```
DELETE FROM Post
WHERE cont_id = @cont_id;
DELIMITER $$
CREATE TRIGGER remove_content AFTER DELETE ON
FOR EACH ROW BEGIN
DELETE FROM Content
    WHERE cont_id = @cont_id;
    END $$
DELIMITER ;
```

6 Advanced Database Components

6.1 Views

6.1.1 Default homepage for user and nonusers of system

Both users and nonusers of a system will face a default view when opening the homepage of Fara, what will only show the most popular posts of the past day of all categories. This can also be something like 'trending'.

```
CREATE VIEW Show_Posts AS
(SELECT Post.post_title, Content.content, Content.username, Content.timestamp, Content.net_vote
FROM Post, Content
WHERE Post.cont_id = Content.cont_id
AND Content.timestamp > DATEADD(day, -1, now())
ORDER BY Content.net_vote)
```

6.2 Triggers

- When a user follows another user, the corresponding users followers and people he/she follows are being updated.
- When a certain post is upvoted/downvoted the corresponding post's votes are updated accordingly.
- When a new comment is added to a certain post the corresponding posts's comments are updated accordingly.
- When a user posts a content, his/her followers will be notified.

6.3 Stored Procedures

- A procedure will be used to notify users when any of their posts is been upvoted/downvoted.
- A procedure will be used to notify users when they get a new follower
- A procedure will be used to notify users when a new comment is added on any of their posts.
- A procedure will be used to search for content in the database using ranking based on counting
- A procedure will be used to search for content in the database using ranking based on TF-IDF score.

6.4 Constraints

- Users must login in in order to be able to post a new posts, comment or downvote/upvote a post, otherwise the
- The maximum length of the content of a post or of a comment must not exceed 800 characters.
- A specific user cannot post more than 50 post in one day.
- A specific user cannot make more than 50 comments on the same post.

7 Implementation Details

For implementation of this project we are considering three different layers: data layer, service layer and presentation layer. We will use different stacks of technologies for each layer. In the data layer, we will use standard MySQL, and MySQL workbench for local testing. In the service layer we will use PHP. In the presentation layer we are considering HTML, CSS and Bootstrap for responsiveness. For data-binding and modularity we are considering frameworks such as Knockout.