MADRAS INSTITUTE OF TECHNOLOGY
ANNA UNIVERSITY
DEPARTMENT OF INFORMATION TECHNOLOGY

IT5412 ADVANCED DATA STRUCTURES LABORATORY

TITLE: SOCIAL MEDIA SIMULATOR

PROJECT TEAM MEMBERS:

1. PRASANNA KUMAR M(2022506063)
2. YUVASHRI AR P(2022506118)
3. HARINI MUTHULAKSHMI K(2022506119)
4. SOWNDHAR M(2022506127)
5. AADHARSH S(2022506132)
6. NAREN KARTHIKEYAN B(2022506133)
7. NAVIN SURGITH M(2022506317)

TABLE OF CONTENTS

INTRODUCTION:

The social media simulator is designed to encompass and showcase the various functionalities supported by social media platforms worldwide. This document aims to provide a comprehensive overview of the multitude of functionalities offered by the simulator, along with detailed information on the implementation of these features. Specifically, this simulator mirrors the usage and features of popular platforms like Instagram.

REQUIREMENT ANALYSIS:

Functional Requirements:

- User profiles are managed by the system, allowing personal information, settings, and preferences to be updated and maintained.
- Posts are created by users and shared within the platform, enabling content dissemination to followers or the public.
- Comments are made on posts by other users, facilitating discussion and feedback on the shared content.
- Stories are created and shared, providing users with a way to post temporary content that is visible for a limited time.
- Conversations are held between two users through direct messaging, allowing private communication.
- Group conversations are conducted within group chats, enabling multiple users to participate in discussions simultaneously.
- Communities are formed by users around shared interests, and polls are conducted within these communities to gather opinions and feedback.

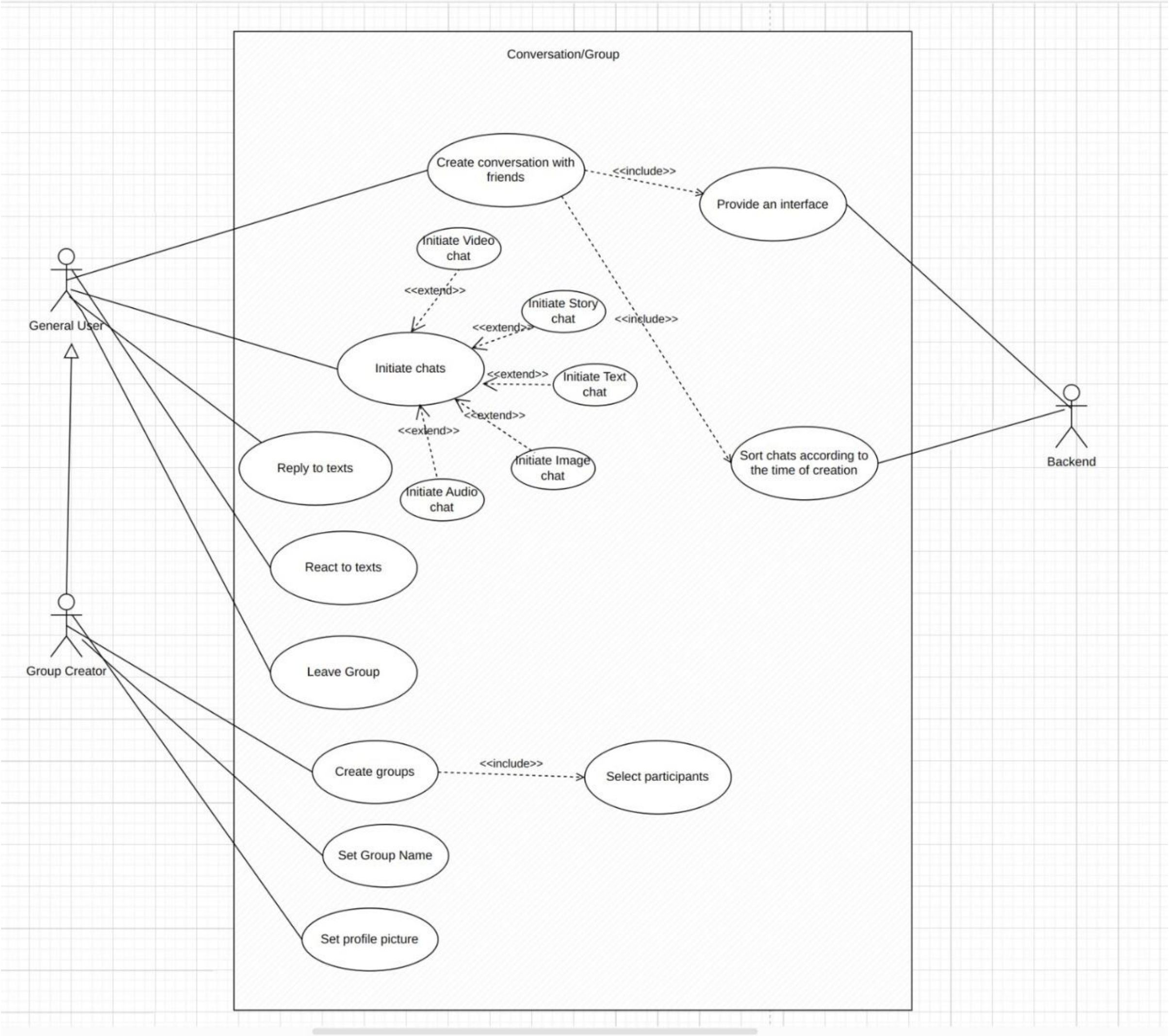
Non-Functional Requirements:

- The system handles high traffic and large volumes of data efficiently, ensuring quick response times and minimal latency.
- The platform provides an intuitive and easy-to-navigate interface, enhancing the overall user experience.

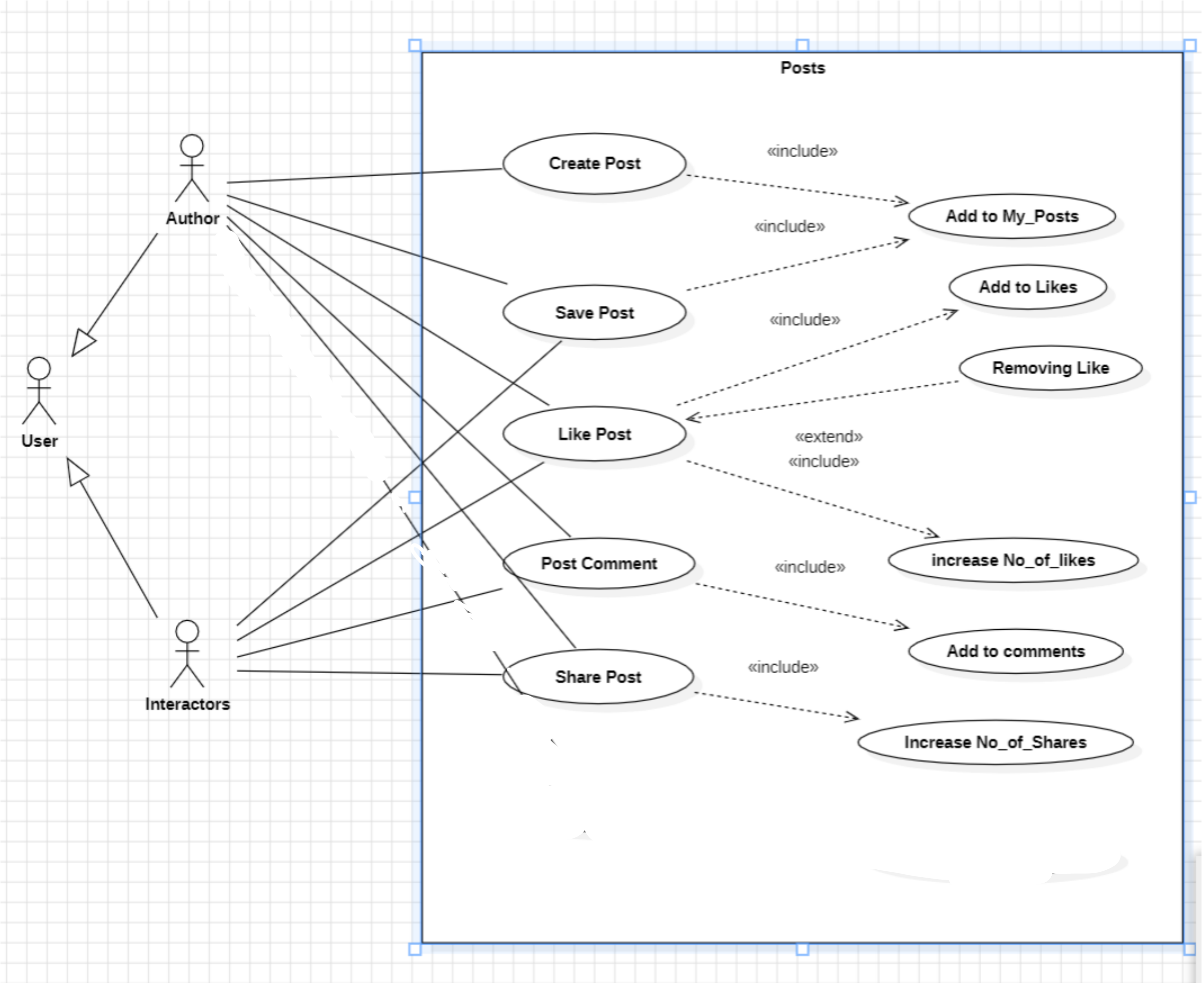
USE CASE DIAGRAMS:

1. **Functionality: CONVERSATION**

Handled by: NAREN KARTHIKEYAN B – 2022506133



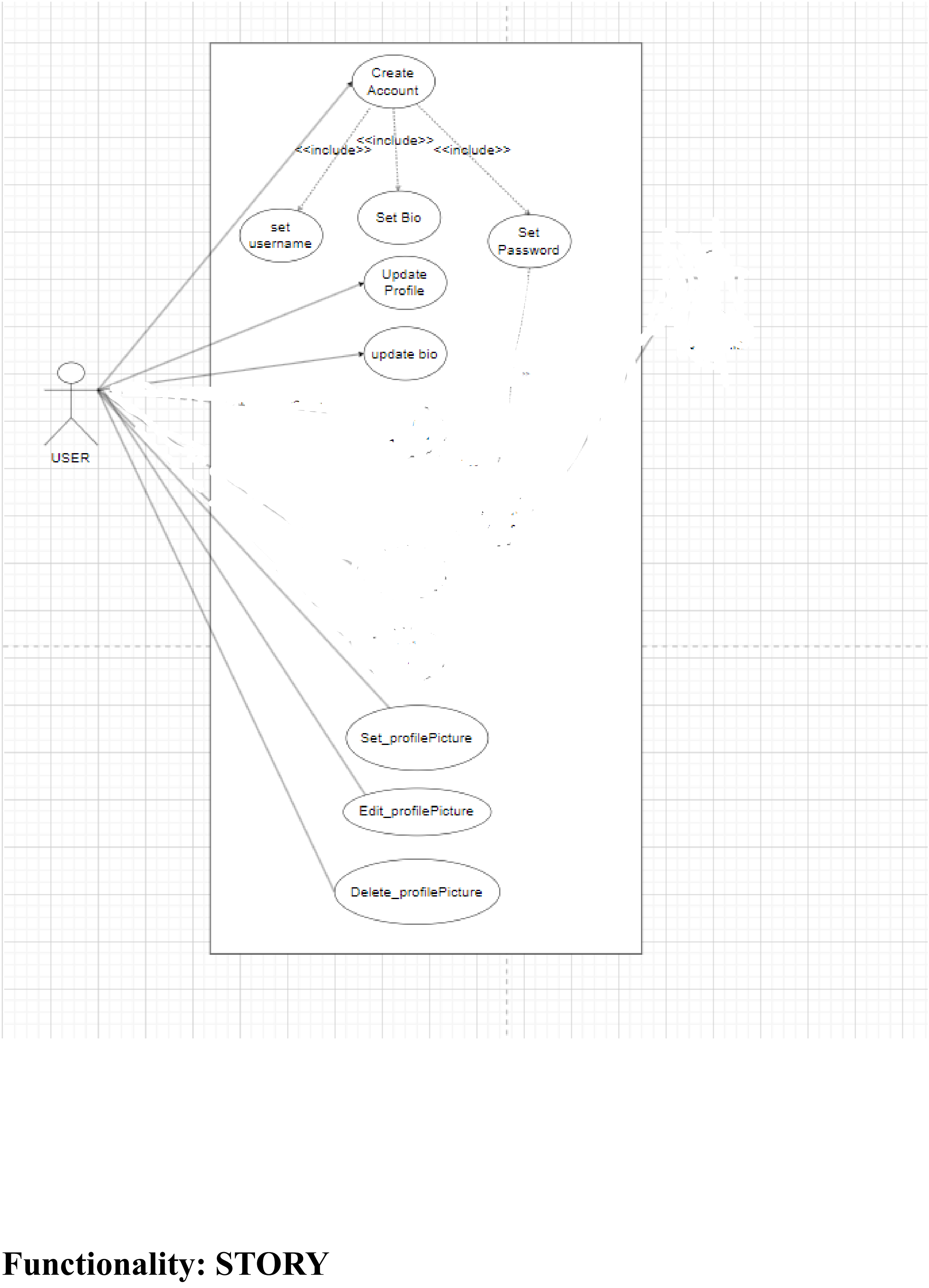
2. **Functionality: POST**



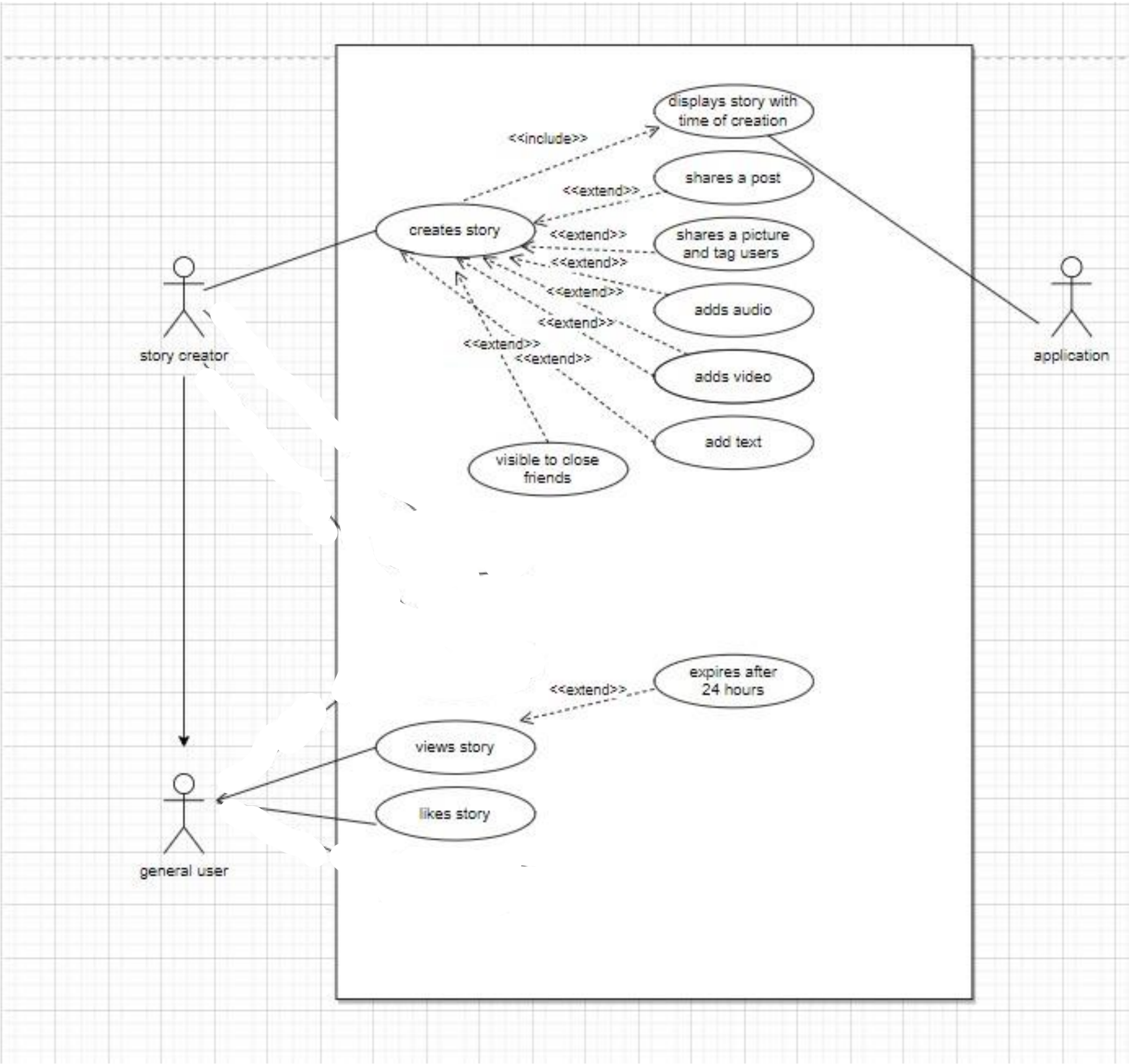
Handled by: AADHARSH S – 2022506132

3. **Functionality: USER**

   Handled by: NAVIN SURGITH M – 2022506317



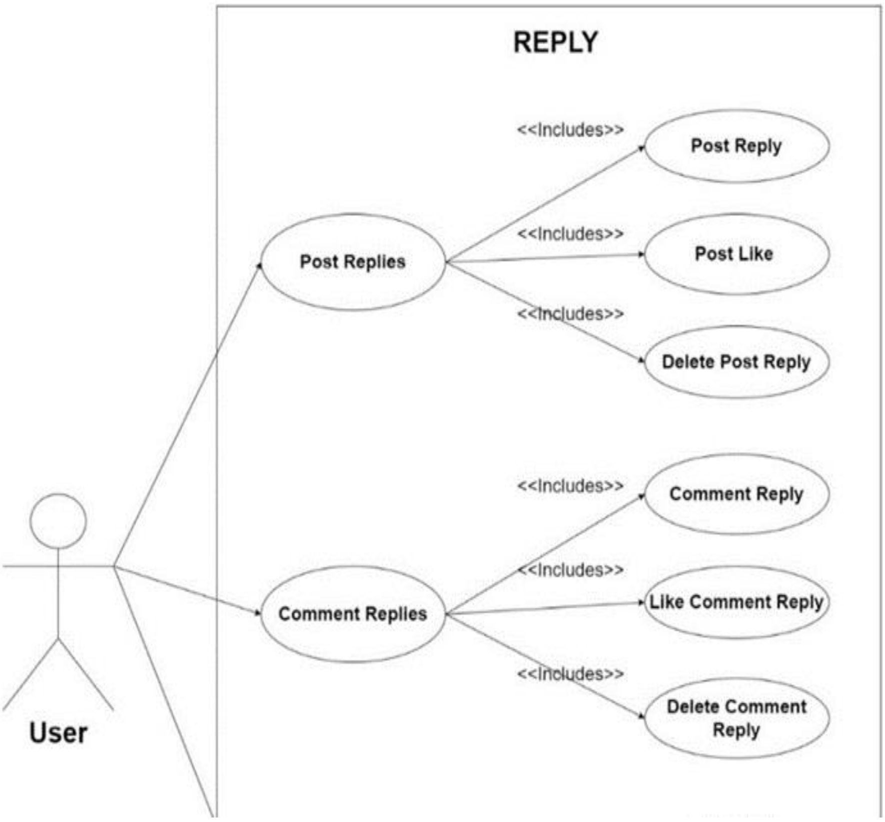
4. Functionality: STORY



Handled by: HARINI MUTHULAKSHMI K – 2022506119

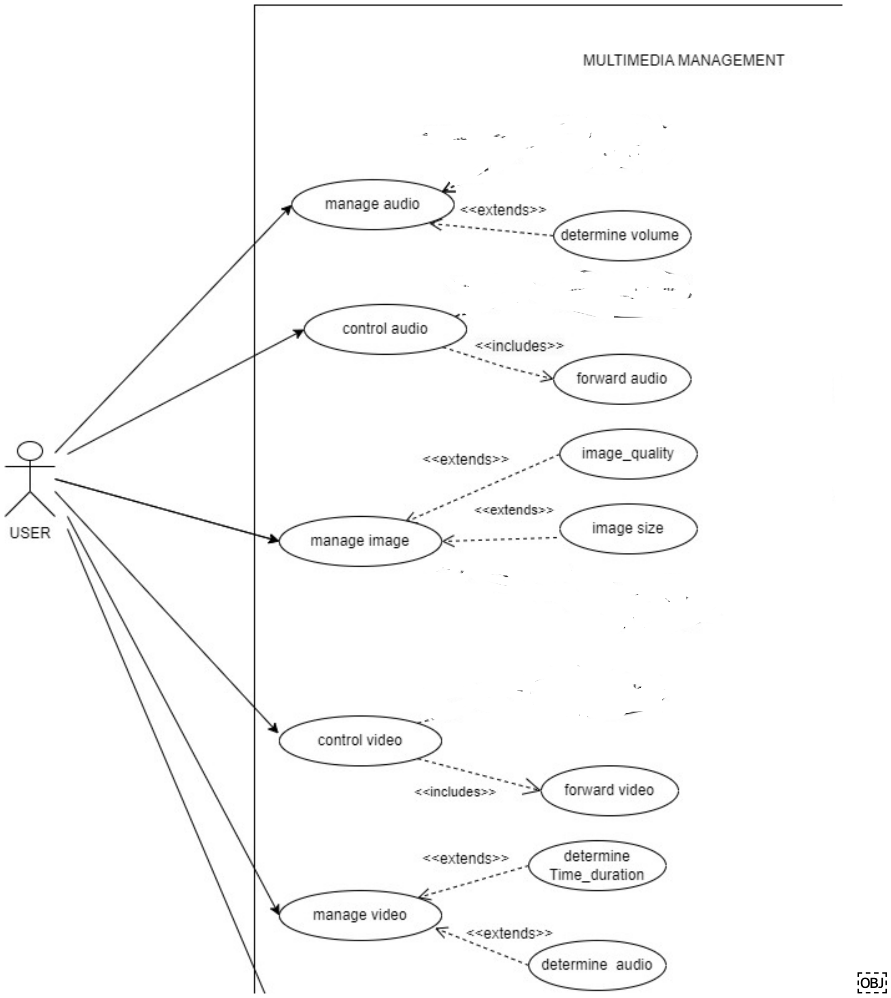
5. **Functionality: COMMENT**

   Handled by: SOWNDHAR M – 2022506127



6. **Functionality: MESSAGE**

   Handled by: YUVASHRI AR P – 2022506118



FUNCTIONALITY OVERVIEW: CLASSES AND DESCRIPTIONS

POST - (AADHARSH S – 2022506132)

CLASSES: POST

This class contains all the functionalities related to Post. Posts are the main medium of interaction between users in our social media

CONVERSATION - (NAREN KARTHIKEYAN B - 2022506133)

CLASSES: CONVERSATION, DMCONVERSATION, COMMUNITYCONVERSATION, FREEJOINCOMMUNITY, REQUESTBASEDCOMMUNITY, GROUPCONVERSATION

Used to maintain Direct Messaging (Conversation between two people), Community and Group conversation (Conversation between any number of people).

Community has two more divisions – free to join and request based communities depending on the joining nature (either free to join or wait for admin approval to join).

Community only allows admins to message and conduct polls. For everyone to message, group conversation can be created.

MESSAGE - (YUVASHRI AR P - 2022506118)

CLASSES: MESSAGE, MULTIMEDIA, IMAGE, VIDEO, AUDIO, TEXT<STRING>, TEXT<EMOJI>

Maintains multimedia and text aspects of the system. Message is the abstract class for all of them. Our system allows images, videos, audio, gif and emojified texts.

USER - (NAVIN SURGITH M - 2022506317)

(PRASANNA KUMAR M - 2022506063)

CLASSES: USER

It is the focal point of our system.

Maintains user information such as username, email, profile picture and password. In addition to these, it maintains user notifications, posts and stories.

Notification class closely interact with users for notification management. Each notification is given a priority and is stored in a heaptree.

STORY - (HARINI MUTHULAKSHMI K - 2022506119)

CLASSES: STORY

It manages a temporary piece of content shared by a user. This content can include images, videos or text, and is designed to last for a limited period, typically 24 hours, before disappearing.

COMMENT - (SOWNDHAR M – 2022506127)

CLASSES: COMMENT

This class contains all the functionalities related to Post's Comments. Comments are the gateway for the users to express their opinions on social media's posts. Comment class also maintains the reply structure (comment to comments).

POLL - (PRASANNA KUMAR M - 2022506063)

CLASSES: POLL

This class helps in simulating a poll in the community tab.

A question is assigned for which options and the votes by the users are stored in vectors.

DATA STRUCTURES UTILIZED ACROSS FUNCTIONALITIES

POST:

- **List** is used to store info about the users who liked the post and to store the reported posts by a user in user class
- **Stack** is used to store the users who reported the specific post

- **Pair** is used to store the latitude and longitude of the location of a post
- **Graph** is used to store the Comments under each post
- **Splay tree** is used for viewed posts in user

CONVERSATION:

- **Avl tree** is used to maintain messages in the chat. It is keyed on time of creation, so it is naturally expected to have right skewed structure. To combat it, avltree is used.
- **Vector** is used for maintaining participants and admins as they may be accessed in a random order.
- **Stack** is used to maintain history of deleted messages. Deletion activity can be rolled back by popping a message out from the stack.

USER:

- **Heaptree** (Maxheap) for maintaining notifications. Priority is based on two conditions – time of creation and conversation priority.
- **Stack** is maintained for my_stories as viewed stories go the end until expiry date passes (mimics LIFO).
- **Vector** is used for storing my_posts as its access can be random.

STORY:

- **Vector** is utilized to store the user's friends.
- **Graph** employs the BFS algorithm to determine friends, indicating that those in the first layer are considered friends.
- **Stack** is used to simulate the story viewing experience of users.

POLL :

- **Vector** is used for maintaining options and the number of votes provided to those options, which can be accessed in a random order based on our utility.

COMMENT:

- **Graph** is used to store the Comments under each post.

  Each comment is represented as a node in the graph.

  Edges between nodes represent the "reply" relationship between comments.

  The DFS algorithm will start from the root comments (those that are not replies to any other comments) and recursively visit each reply, ensuring all nested replies are visited in sequence.

- **Map** associates each Comments* (a pointer to a comment object) with a unique integer. The integer serves as an identifier to quickly access and manage comments within our graph structure.

GLOBAL:

- **AVLTree** to maintain available users. It is keyed on the username. Chosen because of o(logn) insertion and search time.
- **Splay tree** is used to maintain available conversations of a user so accessing the recently accessed conversation is cheap.
- **Graph:**

  1. Used similarity graph is used to recommend posts. Each vertex is a user. Weight of the edge is determined by the number of communities they are both part of together. Used Kruskal algorithm find clusters. Users belonging to the same clusters are similar than the those belonging to different clusters. Using that logic, post is recommended based on similarity.

  2. Used community graph to store similarity between communities. Each vertex is a community and weight of the edge is determined by the similarity (alpha*Lichenstein distance + beta*string edit distance). Community suggestions are found by running prims algorithm on any of the communities that the user is a part of. Length of the prims tree is controlled, and that incomplete tree is displayed as suggestions.

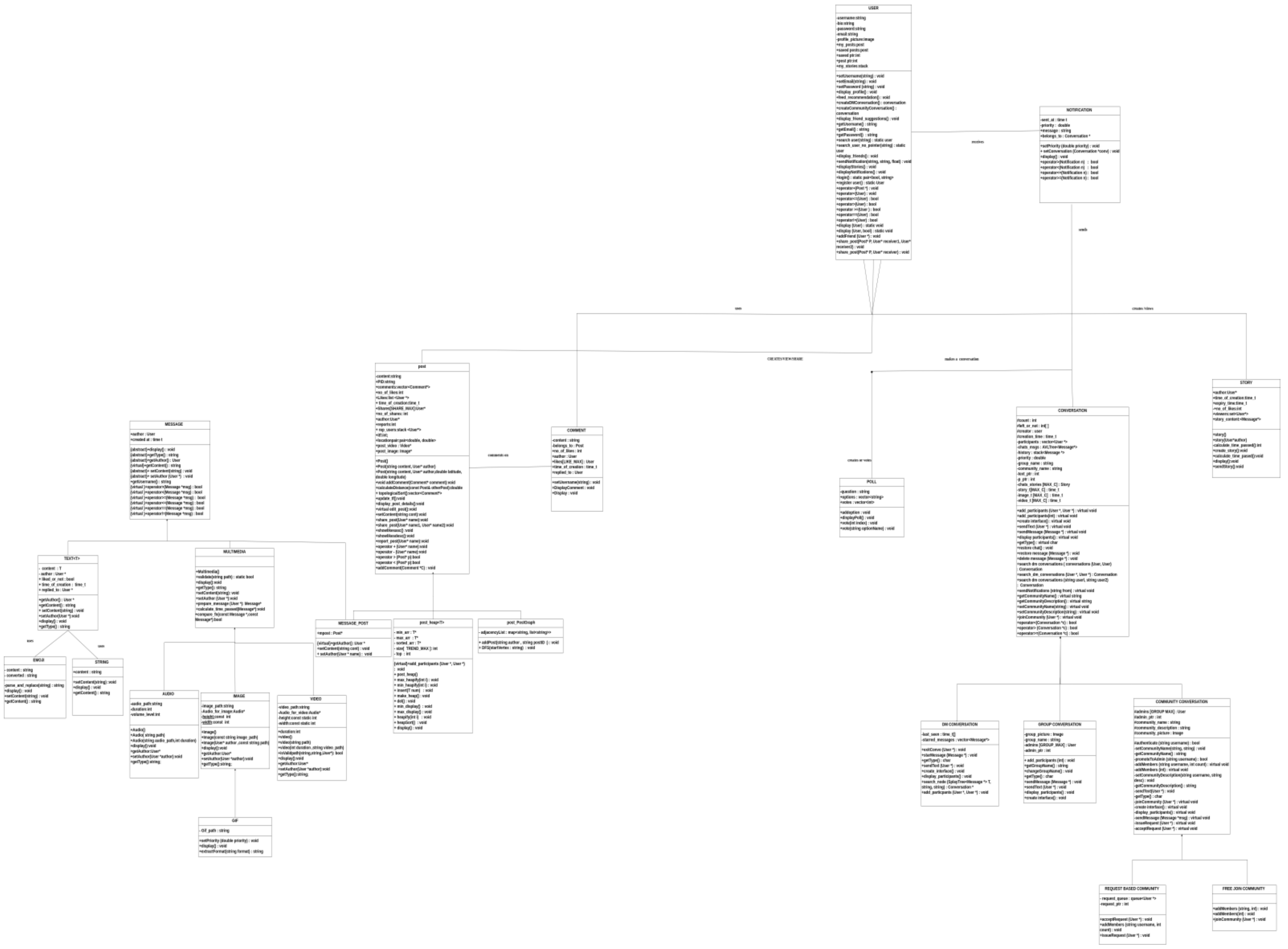
  3. Community graph is again used to drive suggestions based on description given by the user. User’s description becomes a virtual node and a single source shortest path finding algorithm is run from it. Top 5 results are shown. Here, we have chosen Floyd Warshall algorithm. Upon adding a vertex, Floyd Warshall is run to compute the distance matrix. So, satisfying a query becomes a O(V) operation on the precomputed distance matrix. This is chosen because the number of queries is far larger than the number of communities realistically.

4. Global users graph is maintained to store all the users. It is a directed graph with edges pointing from a user to his/her friend. Running breadth first search from the user up to the first level fetches the direct friends of the user and up to the second level fetches the friend suggestions.

5. Many maps are used for integer to desired data type (the type of the vertex) translation for graphs. For instance, two maps are used for each post. Those maps are used for translating from integer to Comment* and vice versa.

- **Queue** is used to store all the posts that have been reported and are under verification. A queue is used since the order of verification should be in a FIFO basis.
- **Heap tree** is used to store all the trending posts, the degree of trend of a post is calculated using the number of likes and no of shares.

CLASS DIAGRAM:


USER
-username:string
-bio:string
-password:string
-email:string
-profile_picture:image
+my_posts:post
+saved posts:post
+saved ptr:int
+post ptr:int
+my_stories:stack
+setUsername(string) : void
+setEmail(string) : void
+setPassword (string) : void
+display_profile() : void
+feed_recommendation() : void
+createDMConversation() : conversation
+createCommunityConversation() : conversation
+display_friend_suggestions() : void
+getUsername() : string
+getEmail() : string
+getPassword() : string
+search user(string) : static user
+search_user_no_pointer(string) : static user
+display_friends() : void
+sendNotification(string, string, float) : void
+displayStories() : void
+displayNotifications() : void
+login() : static pair<bool, string>
+register user() : static User
+operator+(Post *) : void
+operator=(User) : void
+operator<=(User) : bool
+operator>(User) : bool
+operator >=(User) : bool
+operator==(User) : bool
+operator!=(User) : bool
+display (User) : static void
+display (User, bool) : static void
+addFriend (User *) : void
+share_post(Post* P, User* receiver1, User* receiver2) : void
+share_post(Post* P, User* receiver) : void
receives
NOTIFICATION
-sent_at : time t
-priority : double
+message : string
+belongs_to : Conversation *
+setPriority (double priority) : void
+ setConversation (Conversation *conv) : void
+display() : void
+operator>(Notification n) : bool
+operator<(Notification n) : bool
+operator<=(Notification n) : bool
+operator>=(Notification n) : bool
sends
user
CREATES/VIEW/SHARE
makes a conversation
creates /views
post
-content:string
+PID:string
+comments:vector<Comment*>
+no_of_likes:int
+Likes:list <User *>
+ time_of_creation:time_t
+Shares[SHARE_MAX]:User*
+no_of_shares: int
+author:User*
+reports:int
+ rep_users:stack <User*>
+tf:int;
+locationpair:pair<double, double>
+post_video : Video*
+post_image: Image*
+Post()
+Post(string content, User* author)
+Post(string content, User* author,double latitude, double longitude)
+void addComment(Comment* comment):void
+calculateDistance(const Post& otherPost):double
+ topologicalSort():vector<Comment*>
+update_tf():void
+display_post_details():void
+virtual edit_post():void
+setContent(string cont):void
+share_post(User* name):void
+share_post(User* name1, User* name2):void
+showlikesasc() :void
+showlikesdesc():void
+report_post(User* name):void
+operator + (User* name):void
+operator - (User* name):void
+operator > (Post* p):bool
+operator < (Post* p):bool
+addComment(Comment *C) : void
comments on
COMMENT
-content : string
-belongs_to : Post
+no_of_likes : int
+author : User
+likes[LIKE_MAX] : User
+time_of_creation : time_t
+replied_to : User
+setUsername(string) : void
+DisplayComment : void
+Display : void
MESSAGE
+author : User
+created at : time t
{abstract}+display() : void
{abstract}+getType() : string
{abstract}+getAuthor() : User
{virtual}+getContent() : string
{abstract}+ setContent(string) : void
{abstract}+ setAuthor (User *) : void
+getUsername() : string
{virtual }+operator>(Message *msg) : bool
{virtual }+operator<(Message *msg) : bool
{virtual }+operator>=(Message *msg) : bool
{virtual }+operator<=(Message *msg) : bool
{virtual }+operator==(Message *msg) : bool
{virtual }+operator!=(Message *msg) : bool
TEXT<T>
- content : T
- author : User *
+ liked_or_not : bool
+ time_of_creation : time_t
+ replied_to : User *
+getAuthor() : User *
+getContent() : string
+ setContent(string) : void
+setAuthor(User *):void
+display() : void
+getType() : string
uses
uses
EMOJI
- content : string
- converted : string
-parse_and_replace(string) : string
+display() : void
+setContent(string) : void
+getContent() : string
STRING
+content : string
+setContent(string) :void
+display() : void
+getContent() : string
MULTIMEDIA
+Multimedia()
+validate(string path) : static bool
+display():void
+getType(): string
+setContent(string): void
+setAuthor (User *):void
+prepare_message (User *): Message*
+calculate_time_passed(Message*):void
+compare_fn(const Message *,const Message*):bool
MESSAGE_POST
+mpost : Post*
(virtual)+getAuthor() :User *
+setContent(string cont) : void
+ setAuthor(User * name) : void
post_heap<T>
- min_arr : T*
- max_arr : T*
- sorted_arr : T*
- size[TREND_MAX]: int
- top : int
{virtual}+add_participants (User *, User *) : void
+ post_heap()
+ max_heapify(int i) : void
+ min_heapify(int i) : void
+ insert(T num) : void
+ make_heap() : void
+ del() : void
+ min_display() : void
+ max_display() : void
+ heapify(int i) : void
+ heapSort() : void
+ display() : void
post_PostGraph
- adjacencyList : map<string, list<string>>
+ addPost(string author , string postID) : void
+ DFS(startVertex : string) : void
AUDIO
-audio_path:string
-duration:int
-volume_level:int
+Audio()
+Audio(string path)
+Audio(string audio_path,int duration)
+display():void
+getAuthor:User*
+setAuthor(User *author):void
+getType():string;
IMAGE
-image_path:string
-Audio_for_image:Audio*
-height:const int
-width:const int
+image()
+image(const string image_path)
+image(User* author ,const string path)
+display():void
+getAuthor:User*
+setAuthor(User *author):void
+getType():string;
VIDEO
-video_path:string
-Audio_for_video:Audio*
-height:const static int
-width:const static int
+duration:int
+video()
+video(string path)
+video(int duration ,string video_path)
+isValidpath(string,string,User*): bool
+display():void
+getAuthor:User*
+setAuthor(User *author):void
+getType():string;
GIF
- Gif_path : string
+setPriority (double priority) : void
+display() : void
+extractFormat(string format) : string
creates or votes
POLL
-question : string
+options : vector<string>
+votes : vector<int>
+addoption : void
+displayPoll() : void
+vote(int index) : void
+vote(string optionName) : void
CONVERSATION
#count : int
#left_or_not : int[ ]
#creator : user
#creation_time : time_t
-participants : vector<User *>
-chats_msgs : AVLTree<Message*>
-history : stack<Message *>
-priority : double
-group_name : string
-community_name : string
-text_ptr : int
-p_ptr : int
-chats_stories [MAX_C] : Story
-story_t[MAX_C] : time_t
-image_t [MAX_C] : time_t
-video_t [MAX_C] : time_t
+add_participants (User *, User *) : virtual void
+add_participants(int) : virtual void
+create interface() : virtual void
+sendText (User *) : virtual void
+sendMessage (Message *) : virtual void
+display participants() : virtual void
+getType() : virtual char
+restore chat() : void
+restore message (Message *) : void
+delete message (Message *) : void
+search dm conversations (conversations (User, User) : Conversation
+search_dm_conversations (User *, User *) : Conversation
+search dm conversations (string user1, string user2) : Conversation
+sendNotifications (string from) : virtual void
+getCommunityName() : virtual string
+getCommunityDescription() : virtual string
+setCommunityName(string) : virtual void
+setCommunityDescription(string) : virtual void
+joinCommunity (User *) : virtual void
+operator<(Conversation *c) : bool
+operator> (Conversation *c) : bool
+operator>=(Conversation *c) : bool
STORY
+author:User*
+time_of_creation:time_t
+expiry_time:time_t
-+no_of_likes:int
+viewers:set<User*>
+story_content:<Message*>
+story()
+story(User*author)
-calculate_time_passed():int
+create_story():void
+calculate_time_pased():void
+display():void
+sendStory():void
DM CONVERSATION
-last_seen : time_t[]
-starred_messages : vector<Message*>
+exitConvo (User *) : void
+starMessage (Message *) : void
+getType() : char
+sendText (User *) : void
+create_interface() : void
+display_participants() : void
+search_node (SplayTree<Message *> T, string, string) : Conversation *
+add_particpants (User *, User *) : void
GROUP CONVERSATION
-group_picture : Image
-group_name : string
-admins [GROUP_MAX] : User
-admin_ptr : int
+ add_participants (int) : void
+getGroupName() : string
+changeGroupName() : void
+getType() : char
+sendMessage (Message *) : void
+sendText (User *) : void
+display_participants() : void
+create interface() : void
COMMUNITY CONVERSATION
#admins [GROUP MAX] : User
#admin_ptr : int
#community_name : string
#community_description : string
#community_picture : Image
#authenticate (string username) : bool
-setCommunityName(string, string) : void
-getCommunityName() : string
-promoteToAdmin (string username) : bool
-addMembers (string username, int count) : virtual void
-addMembers (int) : virtual void
-setCommunityDescription(string username, string desc) : void
-getCommunityDescription() : string
-sendText(User *) : void
-getType() : char
-joinCommunity (User *) : virtual void
-create interface() : virtual void
-display_participants() : virtual void
-sendMessage (Message *msg) : virtual void
-issueRequest (User *) : virtual void
-acceptRequest (User *) : virtual void
REQUEST BASED COMMUNITY
- request_queue : queue<User *>
-request_ptr : int
+acceptRequest (User *) : void
+addMembers (string username, int count) : void
+issueRequest (User *) : void
FREE JOIN COMMUNITY
+addMembers (string, int) : void
+addMembers(int) : void
+joinCommunity (User *) : void

SCOPE FOR FUTURE:

- Allow users to use different kinds of reactions in chat.
- For the convos that aim to boost productivity, separate functionalities such as note keeping, tasks management can be brought.
- Automate comment moderation with sentiment analysis for flagging inappropriate content.
- Use machine learning to offer personalized content recommendations based on user behaviour.
- Enable real-time collaboration in group discussions for simultaneous interaction.
- Implement advanced machine learning for improved identification of inappropriate content.
- Introduce privacy settings for chat replies, giving users control over message visibility.
- Expand reaction options beyond likes, incorporating emojis or custom reactions for user expression.
- Allow for users to set filters to multimedia objects especially images.
- Automatically enhance the quality of uploaded or utilized multimedia to a higher standard.
- Biometrics like fingerprints and face recognition can be used a replacement for the username and password as authentication requirements
- To store all the details, NOSQL dB such as MongoDB can be used instead of relying on a file system.
- Creation of anonymous accounts for anonymous interaction.
- Recommendation of Stories based on the Feed of the User.