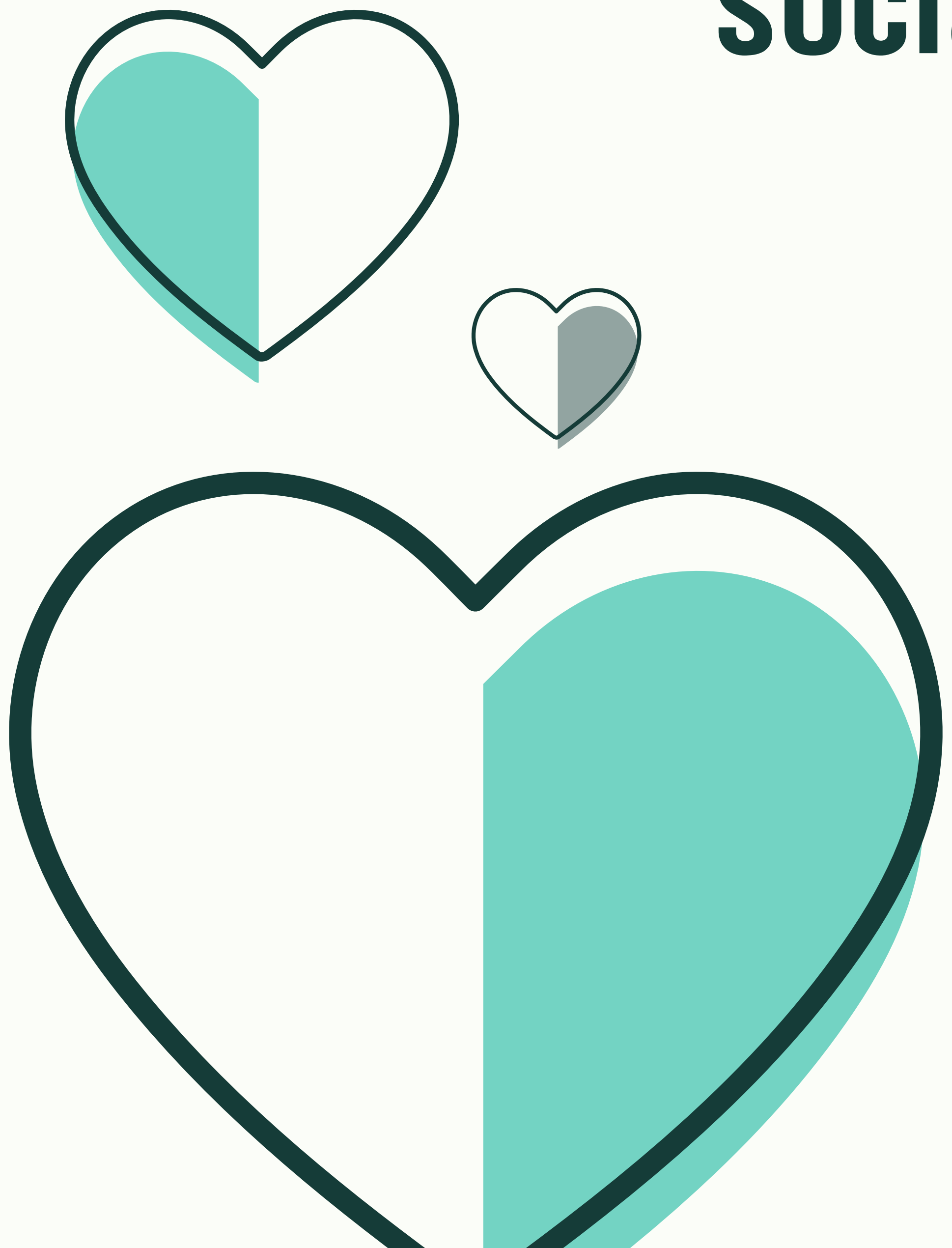
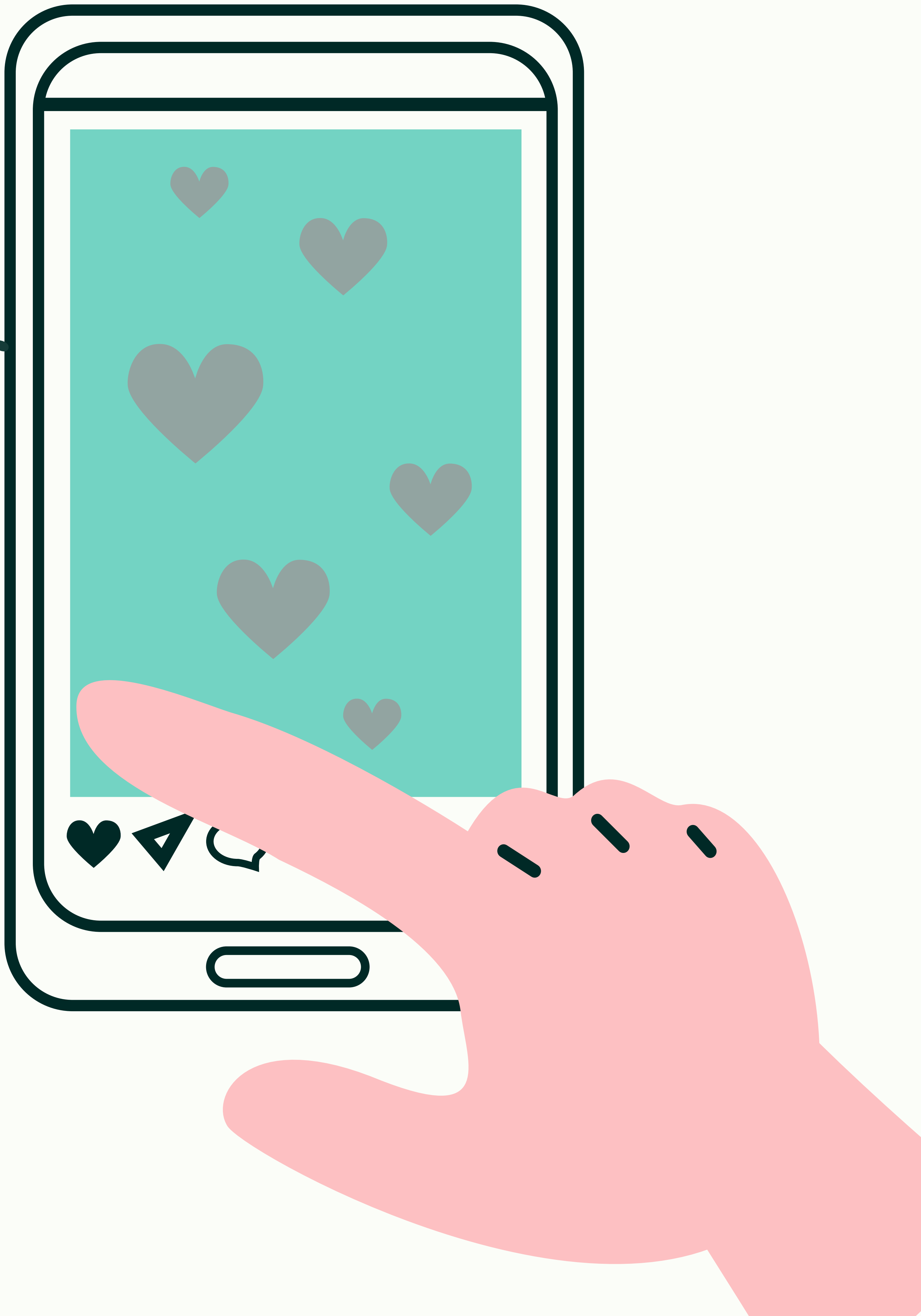
Yumelink

social media *project*

Borcelle Agency
2023





Project outline

Real world - business	UML
Translate to Relation	1 -3 NF
Django queries	DEMO

REAL WORLD - BUSINESS

Business selection: Social Media Platform

- **Objective:** A social media web application designed to showcase the process of implementing database knowledge into building a functional software product. The focus is on learning and practical experience.

List Business Activities/Processes.

User Profiles

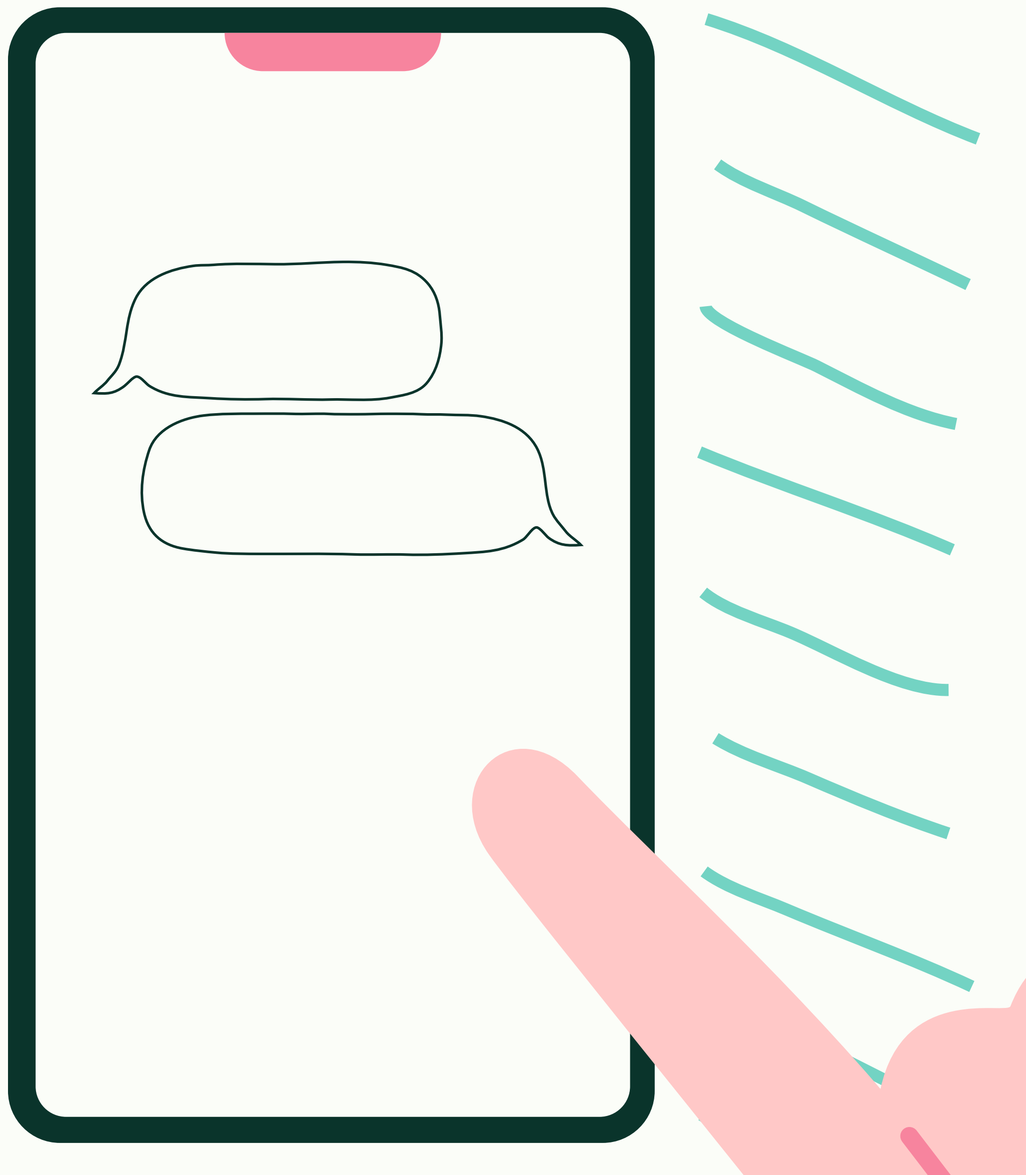
Activities/Processes:

- Users fill out personal information, create bios, and upload profile pictures.
- Users can edit their bios or change their profile pictures.

User Relationship

Activities/Processes:

- Users can follow other users to see their posts or block users to prevent interactions.



List Business Activities/Processes.

Content

Activities/Processes:

- Users post text and images

Message

Activities/Processes:

- Users can send private messages to each other, either individually or in groups

List Business Activities/Processes.

User Activity

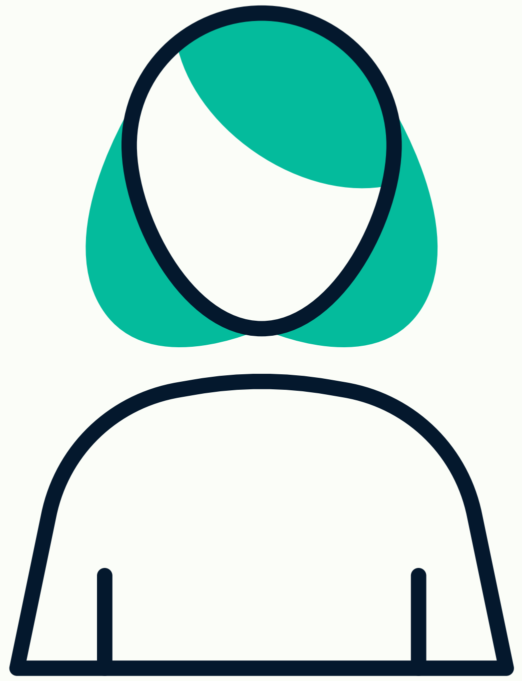
Activities/Processes:

- Users can like, share, or comment on post

Tags

Activities/Processes:

- Users can add tags (keywords) to their posts for categorization.



Processes Necessitate

User Profiles

Collect:

- Collect user details (e.g., name, email, bio, profile picture) during registration and updates.

Manage (Insert, Update, Delete):

- Add new user profile data when they sign up.
- Allow users to update their bio, profile picture, and other personal details.

Data Use:

- Display user profiles on the use interface.

Processes Necessitate

User Relationship

Collect:

- Collect and store data about who is following whom, and who has blocked whom.

Manage (Insert, Update, Delete):

- A new relationship (e.g., following or blocking)
- Users can change their relationships (e.g., unfollow someone or unblock them)

Data Use:

- Displaying the correct feed of posts for each user based on who they follow.

Processes Necessitate

Content

Collect:

- Collect data related to user posts (timestamps, post ID, etc.).

Manage (Insert, Update, Delete):

- New posts are inserted into the database when users upload content.
- Users can edit, delete or update their posts

Data Use:

- Displaying posts to users (on their feed or specific user pages).

Processes Necessitate

Message

Collect:

- Collect message data (sender, receiver, timestamp, message content).

Manage:

- Each new message sent between users is added to the database.

Data Use:

- Displaying messages and conversations on the user interface.

Processes Necessitate

Tags

Collect:

- Collect data about the tags associated with posts

Manage:

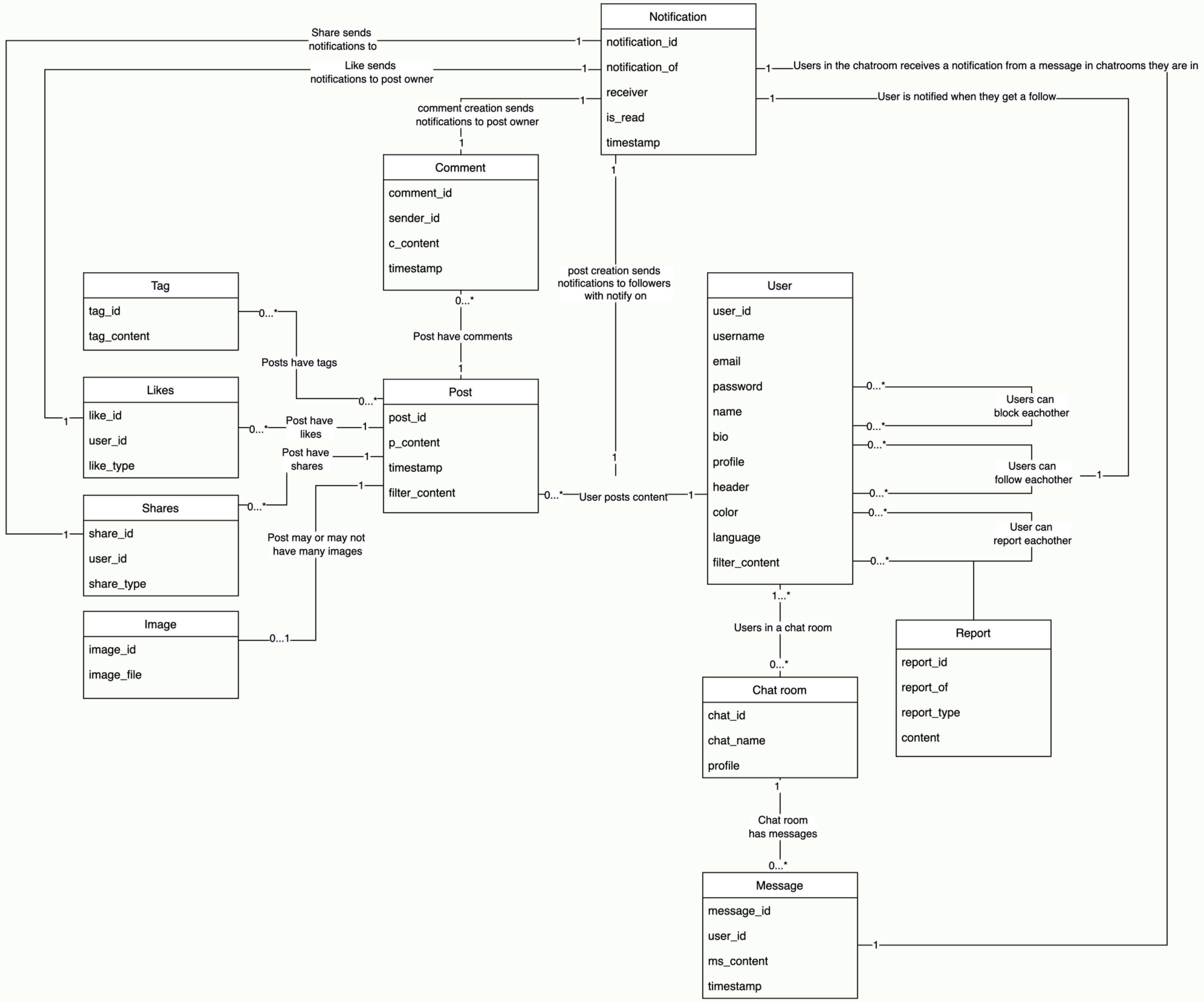
- Insert: Tags are added when users post content.
- Update: Tags may be modified (e.g., changing or adding new ones).
- Delete: If a tag becomes irrelevant or outdated, it can be removed from the post.

Data Use:

- Enabling users to filter and search for posts based on tags.

DATABASE

UML


Notification
notification_id
notification_of
receiver
is_read
timestamp
Share sends notifications to
Like sends notifications to post owner
comment creation sends notifications to post owner
Users in the chatroom receives a notification from a message in chatrooms they are in
User is notified when they get a follow
post creation sends notifications to followers with notify on
Comment
comment_id
sender_id
c_content
timestamp
Tag
tag_id
tag_content
Posts have tags
Post have comments
Likes
like_id
user_id
like_type
Post have likes
Post have shares
Shares
share_id
user_id
share_type
Post may or may not have many images
Image
image_id
image_file
Post
post_id
p_content
timestamp
filter_content
User posts content
User
user_id
username
email
password
name
bio
profile
header
color
language
filter_content
Users can block eachother
Users can follow eachother
User can report eachother
Report
report_id
report_of
report_type
content
Users in a chat room
Chat room
chat_id
chat_name
profile
Chat room has messages
Message
message_id
user_id
ms_content
timestamp
1
0...*
0...1
1...*

Translate to Relation

red- Foreign key
blue- Primary key that is not Foreign key
underline - Primary key
black- Mon-key attribute

Relation	Attribute	Foreign keys
Follow	• user_id • follower_id • notify	• user_id: User(user_id) • follower_id: User(user_id)
Blocker	• blocker_id • blocked_id	• blocker_id: User(user_id) • blocked_id: User(user_id)
Report	• report_id • report_of • reporter_id • report_type • content	• report_of : Any class object • reporter_id : User(user_id)

Relation	Attribute	Foreign keys
User	• <u>user_id</u> • username • email • password • name • bio • profile • header • color • language • filter_content	
UserChat	• <u>user_id</u> • <u>chat_id</u>	• user_id: User(user_id) • chat_id: ChatRoom(chat_id)

red- Foreign key
blue- Primary key that is not Foreign key
<u>underline</u> - <u>Primary key</u>
black- Mon-key attribute

Relation	Attribute	Foreign keys
Post	• <u>post_id</u> • user_id • p_content • timestamp • filter_content	• user_id: User(user_id)
PostImage	• <u>postimage_id</u> • post_id • image	• post_id : Post(post_id)
Tag	• <u>tag_id</u> • tag_content	
PostTag	• <u>tag_id</u> • <u>post_id</u>	• tag_id : Tag(tag_id) • post_id : Post(post_id)

red- Foreign key
blue- Primary key that is not Foreign key
<u>underline</u> - <u>Primary key</u>
black- Mon-key attribute

Relation	Attribute	Foreign keys
Comment	• comment_id • user_id • post_id • c_content • timestamp	• user_id : User(user_id) • post_id : Post(post_id)
ChatRoom	• chat_id • chat_name • profile	
Message	• message_id • user_id • chat_id • ms_content • timestamp	• user_id : User(user_id) • chat_id : ChatRoom(chat_id)

red- Foreign key
blue- Primary key that is not Foreign key
underline - Primary key
black- Mon-key attribute

Relation	Attribute	Foreign keys
Likes	• <u>like_id</u> • user_id • post_id • like_type	• user_id : User(user_id) • post_id : Post(post_id)
Shares	• <u>share_id</u> • user_id • post_id • share_type	• post_id : Post(post_id) • user_id : User(user_id)
Notification	• <u>notification_id</u> • notification_of • receiver_id • is_read • timestamp	• notification_of : Any Class object • receiver_id : User(user_id)

red- Foreign key
blue- Primary key that is not Foreign key
<u>underline</u> - <u>Primary key</u>
black- Mon-key attribute

1 - 3 nf

Notification(notification_id, notification_of, receiver_id, is_read, timestamp)

1NF: All attributes contain atomic values.

- notification of

notification_of is a model object with id, type

Thus, we seperate it into 2 attributes:

id(Integer) and content_type(ContentType)

*ContentType is a Django model for model object class Types

2NF: No partial dependency on any subset of a primary key.
Since notification_id is the primary key, all non-key attributes (notification_id, notification_of, receiver_id, is_read, timestamp) are fully dependent on notification_id.
3NF: No transitive dependency on non-key attributes.
There are no dependencies between non-key attributes; therefore, no transitive dependency exists.

Identical process for Report(report_of)

DJANGO QUERIES

We will discuss Django Queries in DEMO, for example;

- current_user = User.objects.get(id=self.request.user.id)
- blocking_users = Block.objects.filter(blocked=current_user).values_list('blocker', flat=True)
- posts = Post.objects.all().exclude(user__id__in=excluded_users).order_by('-timestamp')
- posts = Post.objects.all().order_by('-timestamp')
- 'post_images': PostImage.objects.filter(post=post)
- is_blocked = Block.objects.filter(blocker=post_owner, blocked=current_user).exists()
- for comment in Comment.objects.filter(post=post).order_by('-timestamp')
- 'owns': User.objects.get(id=self.request.user.id) == comment.user
- context['tags'] = Tag.objects.filter(posttag__post=post)
- context['post_tags'] = PostTag.objects.filter(post=post)
- context['likes'] = Like.objects.filter(post=post, type=LikeType.like.name).count()
- context['is_block'] = Block.objects.filter(blocker=user, blocked=viewed_user).exists()
- ...

There’s much more than 9 Queries
but let’s discuss them here;

1. Get current User

```
current_user = User.objects.get(id=self.request.user.id)
```

Get current user from the id of Django’s request id

2. Filter posts for home page

```
blocking_users = Block.objects.filter(blocked=current_user).values_list('blocker', flat=True)
blocked_users = Block.objects.filter(blocker=current_user).values_list('blocked', flat=True)
excluded_users = set(blocking_users) | set(blocked_users)
```

Get blocked users and people who blocked the user

```
queryset =
Post.objects.all().exclude(user__id__in=excluded_users)
```

And get posts excluding theirs

3. Filter posts for home page with search

```
queryset = Post.objects.filter(
    Q(content__icontains=search_query) |
    Q(posttag__tag__content__icontains=search_query) |
    Q(user__username__icontains=search_query) |
    Q(user__name__icontains=search_query)
).exclude(
    user__id__in=excluded_users)
```

Further filter posts from home page to only post with things related in search

4. Get post images

```
context['post_images'] = PostImage.objects.filter(post=post)
```

Get all images of this post

5. Get post comments

```
context['comments'] = [
    {
        'comment': comment,
        'owns': User.objects.get(id=self.request.user.id) == comment.user
    }
    for comment in Comment.objects.filter(post=post).order_by('-timestamp')
]
```

Get all comments of this post

And keep track of whether the current user owns this comment

6.Get viewed user’s posts

```
posts = Post.objects.filter(user=user)
```

Get all posts of this user

```
context['posts_with_images'] = [
    {
        'post': post,
        'post_images': PostImage.objects.filter(post=post)
    }
    for post in posts.order_by('-timestamp')
]
```

and couple them with their post images ordered by timestamp

7.Get user’s notifications

```
user = User.objects.get(id=self.request.user.id)
```

Get user by self.request.user.id

```
Notification.objects.filter(receiver=user).order_by('-timestamp')
```

get notifications where receiver is the current user

8. Get chatrooms user is in

```
current_user = User.objects.get(id=self.request.user.id)
ChatRoom.objects.filter(userchat__user=current_user).order_by('chat_name')
```

Get ChatRooms user is a member of

9.Get messages in a chatroom

```
messages = Message.objects.filter(chat=chat_room).order_by('timestamp')
```

Get all messages of a room ordered by timestamp

DEMO

TEAM MEMBER

6510545276 Kantapon Hemmadhun

6510545535 Nantawan Paramapooti

6510545616 Phatthadon Suwanpattanawech

THANK YOU

for watching!

More Details of the Project on

YumeLink Github