José Mariano Portilla Landa 163449 22/03/2022

Polyglot Data management on the Cloud

Social Networks Data and NF Guarantees

The social networks that are considered for this use case are: Facebook, Twitter, Instagram and Reddit. We choose these because they are some of the most popular ones and have singular properties and shared properties that are interesting to combine. Below are some attributes of each one, they are reduced in order to simplify the use case. We can see that most of them follow the same principles having users, posts, reposting, and relationships between users.

Facebook:

- Users: name, birthday, email, gender, location
- Posts: author, images, text, reactions, comments, date, reposts
- Relationships between users (friends, family, etc.)
- Groups

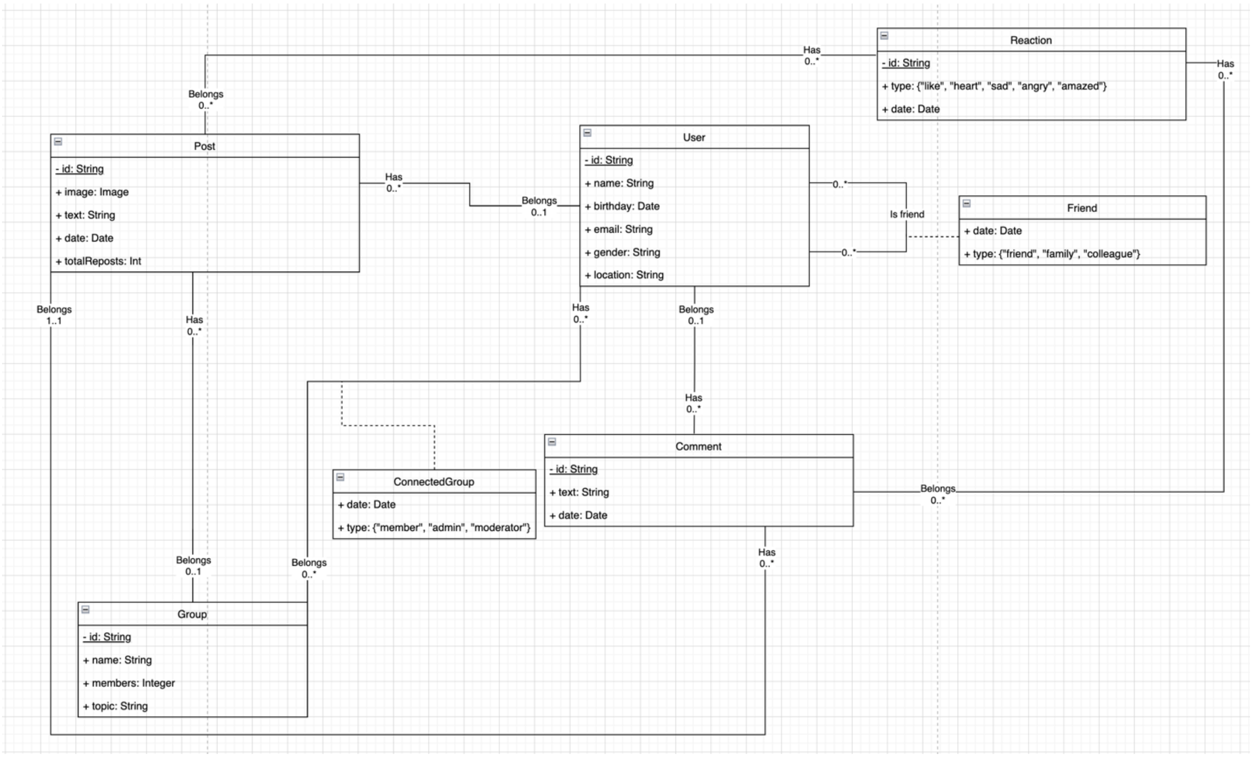

Figure 1: Facebook data UML class diagram.

Entity	NF Guarantees	Conclusion

User/Friend	• We want to have availability and consistency because we want that all clients see the same user data, and have reading and writing access at any time. • Attributes prone to change • May have “friends of friends queries” • We want it to comply with ACID properties, so that transactions (such as making a new friend) are atomic, consistent, isolated and durable.	A relational database management system may come handy in this situation, however, because we have “friends of friends queries”, a graph database solution will have a higher performance.
Post/Comment/Group	• We can allow them to have BASE properties because we value availability, for scaling, over consistency and replicated data • We want availability and partition tolerance given that we do not care if all clients see the same posts, comments or groups simultaneously	We can configure MongoDB to prioritize availability and partition tolerance and making it eventually consistent by implementing Replica-Sets. Other options include: dynamo, Voldemort, Cassandra, etc.
Reaction	• Reactions are dependent on a relationship either with posts or comments • We want it to have availability so that each client can always read and write reactions, and consistency so that all clients have the same view of the reactions of a given post or comment	Because reactions need to be linked to other entities and have relationships in order to matter, we find that modelling them with a graph database would give us enough flexibility and performance for our purposes.

Twitter:

- Users: name, birthday, email, gender, location
- Followers
- tweets: author, text, images, likes, comments, retweets

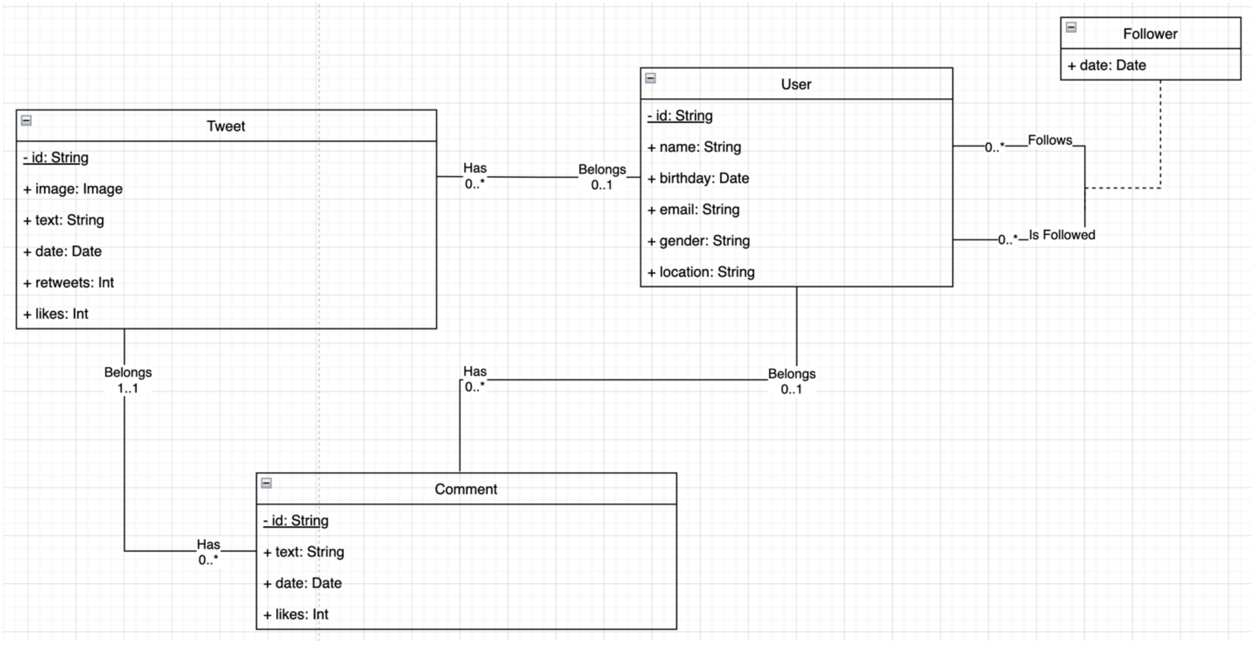

Figure 3: Twitter data UML class diagram.

Entity	NF Guarantees	Conclusion
User/Follower	• We want to have availability and consistency because we want that all clients see the same user data, and have reading and writing access at any time. • Attributes prone to change • May have "friends of friends queries" • We want it to comply with ACID properties, so that transactions (such as following a user) are atomic, consistent, isolated and durable.	A relational database management system may come handy in this situation, however, because we have "friends of friends queries", a graph database solution will have a higher performance.

Tweet/Comment	• We can allow them to have BASE properties because we value availability, for scaling, over consistency and replicated data • We want availability and partition tolerance given that we do not care if all clients see the same tweets and comments simultaneously.	We can configure MongoDB to prioritize availability and partition tolerance and making it eventually consistent by implementing Replica-Sets. Other options include: dynamo, Voldemort, Cassandra, etc.

Instagram:

- Users: name, birthday, email, gender, location
- Posts: author, images, text, comments, likes
- Followers

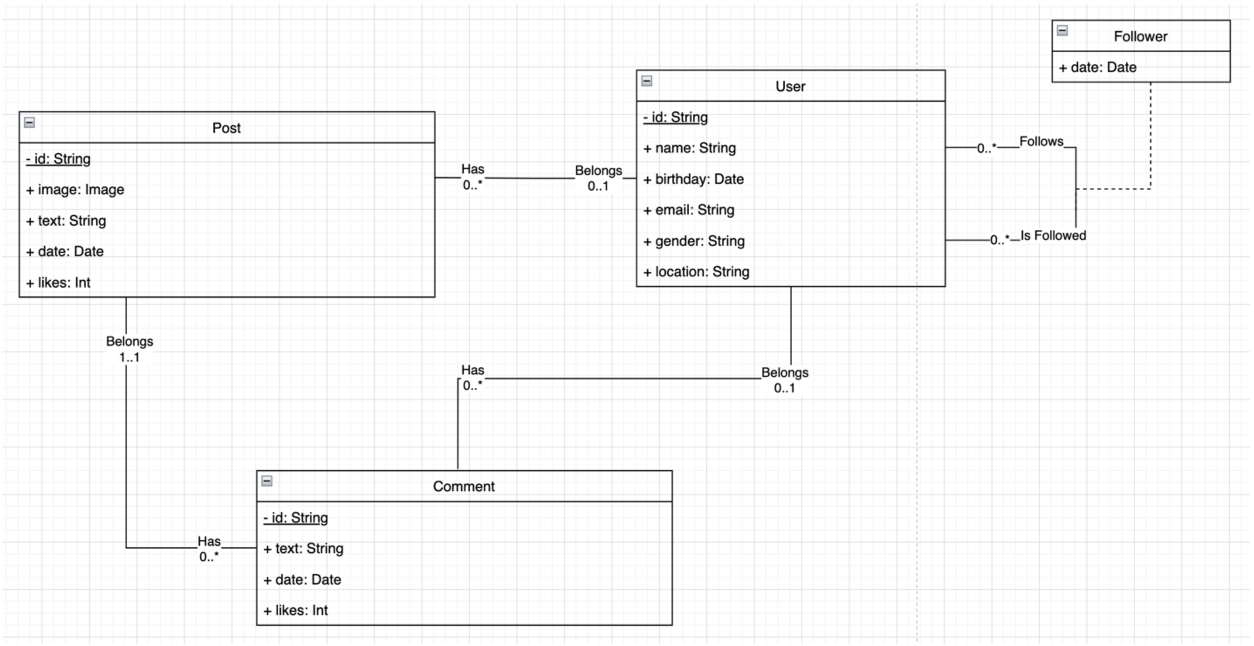

Figure 4: Instagram data UML class diagram.

Entity	NF Guarantees	Conclusion
User/Follower	• We want to have availability and consistency because we want that all	A relational database management system may

	clients see the same user data, and have reading and writing access at any time. • Attributes prone to change • May have “friends of friends queries” • We want it to comply with ACID properties, so that transactions (such as making a new friend) are atomic, consistent, isolated and durable.	come handy in this situation, however, because we have “friends of friends queries”, a graph database solution will have a higher performance.
Post/Comment	• We can allow them to have BASE properties because we value availability, for scaling, over consistency and replicated data • We want availability and partition tolerance given that we do not care if all clients see the same posts and comments simultaneously	We can configure MongoDB to prioritize availability and partition tolerance and making it eventually consistent by implementing Replica-Sets. Other options include: dynamo, Voldemort, Cassandra, etc.

Reddit:

- Users: name, birthday, email, gender, location
- Posts: author, images, text, comments, likes, dislikes, reposts
- Followers
- Communities

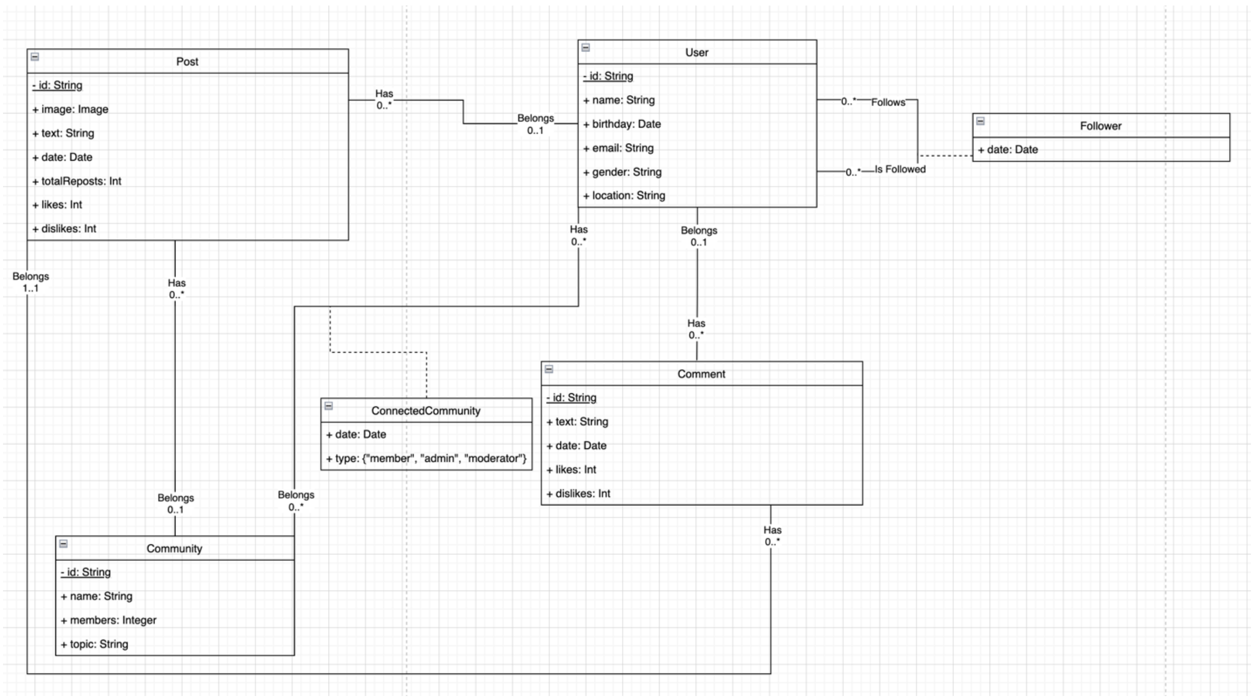

Figure 5: Reddit data UML class diagram.

Entity	NF Guarantees	Conclusion
User/Follower	• We want to have availability and consistency because we want that all clients see the same user data, and have reading and writing access at any time. • Attributes prone to change • May have "friends of friends queries" • We want it to comply with ACID properties, so that transactions (such as making a new friend) are atomic, consistent, isolated and durable	A relational database management system may come handy in this situation, however, because we have "friends of friends queries", a graph database solution will have a higher performance.
Post/Comment/ Community	• We can allow them to have BASE properties because we value availability, for scaling, over consistency and replicated data	We can configure MongoDB to prioritize availability and partition tolerance and making it eventually consistent by

	• We want availability and partition tolerance given that we do not care if all clients see the same posts, comments or communities simultaneously	implementing Replica-Sets. Other options include: dynamo, Voldemort, Cassandra, etc.

Finally, we created a general data schema (Fig. 6), this schema follows the properties of each social network in a generalized way, meaning that we do not care if in Twitter a friend is called a follower or if in Reddit a group is a community, each class has its own string id and attributes related to their purpose, some of them have association classes and their correspondent relationship to other classes. The mapping that we implemented was the following:

- User: contains all attributes that are present in Facebook, Instagram, Reddit and Twitter. Relationship “follows” is the same as “is friend”.
- ConnectedUser: represents a follower, friend or colleague depending on the social network.
- Post: contains all the attributes that all social networks shared, likes and dislikes were omitted to have a general schema and an entity called reaction was created to contain them
- Reaction: contains all kinds of reactions from Facebook (like, heart, sad, angry, amazed), Twitter (like), Instagram (like) and Reddit (like and dislike).
- Comment: contains all shared attributes of all social networks, similar to Post, reactions are contained in the entity Reaction.
- Community: a community is only present in Facebook and Reddit, followers are called members in Facebook.
- ConnectedCommunity: stores data about the relationship between a user and a community, such as the date in which the user joined and the type of the relationship (follower, admin or moderator) depending on the needs each social network.

Although we used a schema to define the data that we have, it is worth noticing that some of this data may not be present in all our records, that is why we are using a Non-SQL solution.

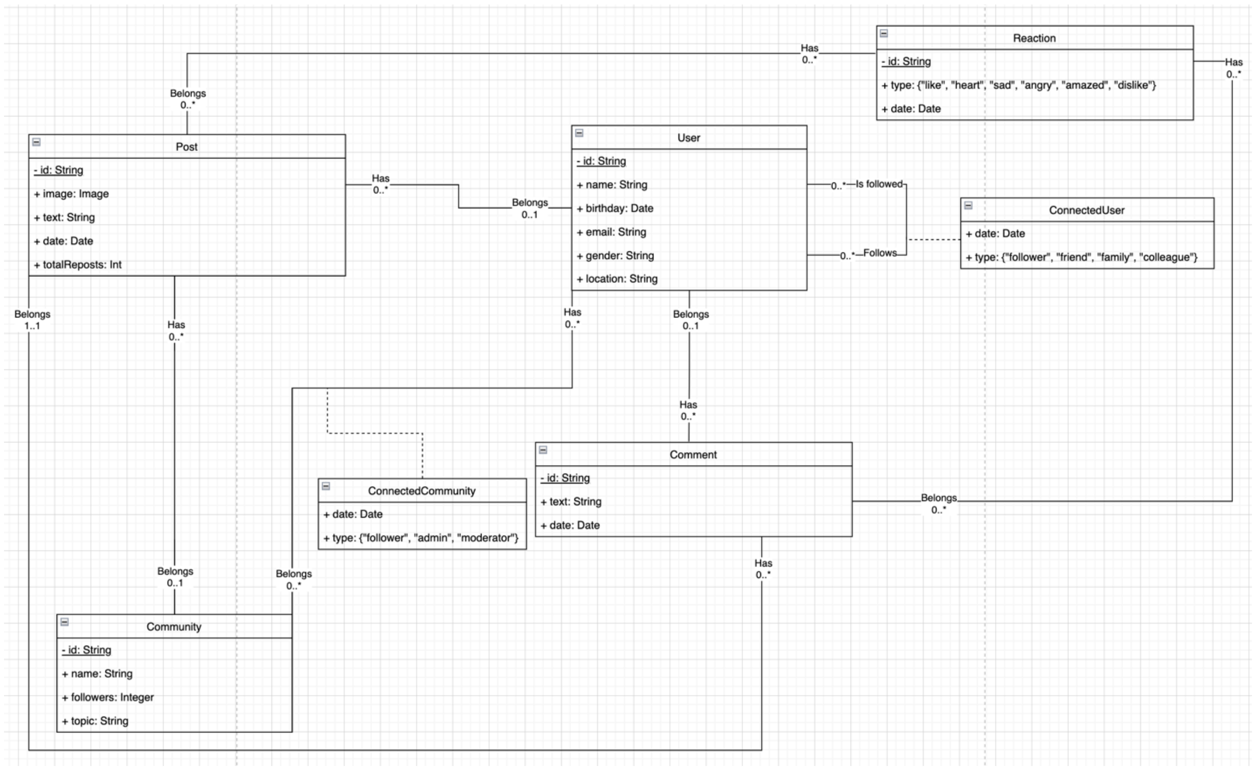

Figure 6: UML data schema of general data model.

Once we created the general schema we decided to use non-relational databases for the following reasons:

- The schema is prone to change
- Friends or followers are well suited for graph databases because we may have queries that traverse multiple levels through graph data such as friends of friends queries
- Data variety and possibly huge amounts of data
- We do not require all the safety that ACID guarantees provide to us, we can do fine with BASE because we value availability over consistency
- Following the CAP theorem, we would sacrifice consistency to get availability and partition tolerance
- Even though we can be fine without ACID it is important to note that most graph databases use an ACID consistency model

Storing data

Fig. 7 shows us the schema that was done for the document base, the rules that were followed in order to produce it were based on the ones that we had on the relational model, however they were adjusted and some of them were not taken into account. If we were constructing a relational database, then we would create a new table for the relationships that contain 0..1 if there is not a 1..1 relationship, however because in NoSQL we do not really care for null values, we just treated a 0..1 relationship as a 1..1 relationship and include the primary key in the correspondent collection (attribute id from Community is inserted in collection Comment). This was really the only rule that we follow, besides making each class a collection of course. Furthermore we thought that relationships with 0..* in both sides were naturally fit for a graph schema, so we omitted them from the document schema.

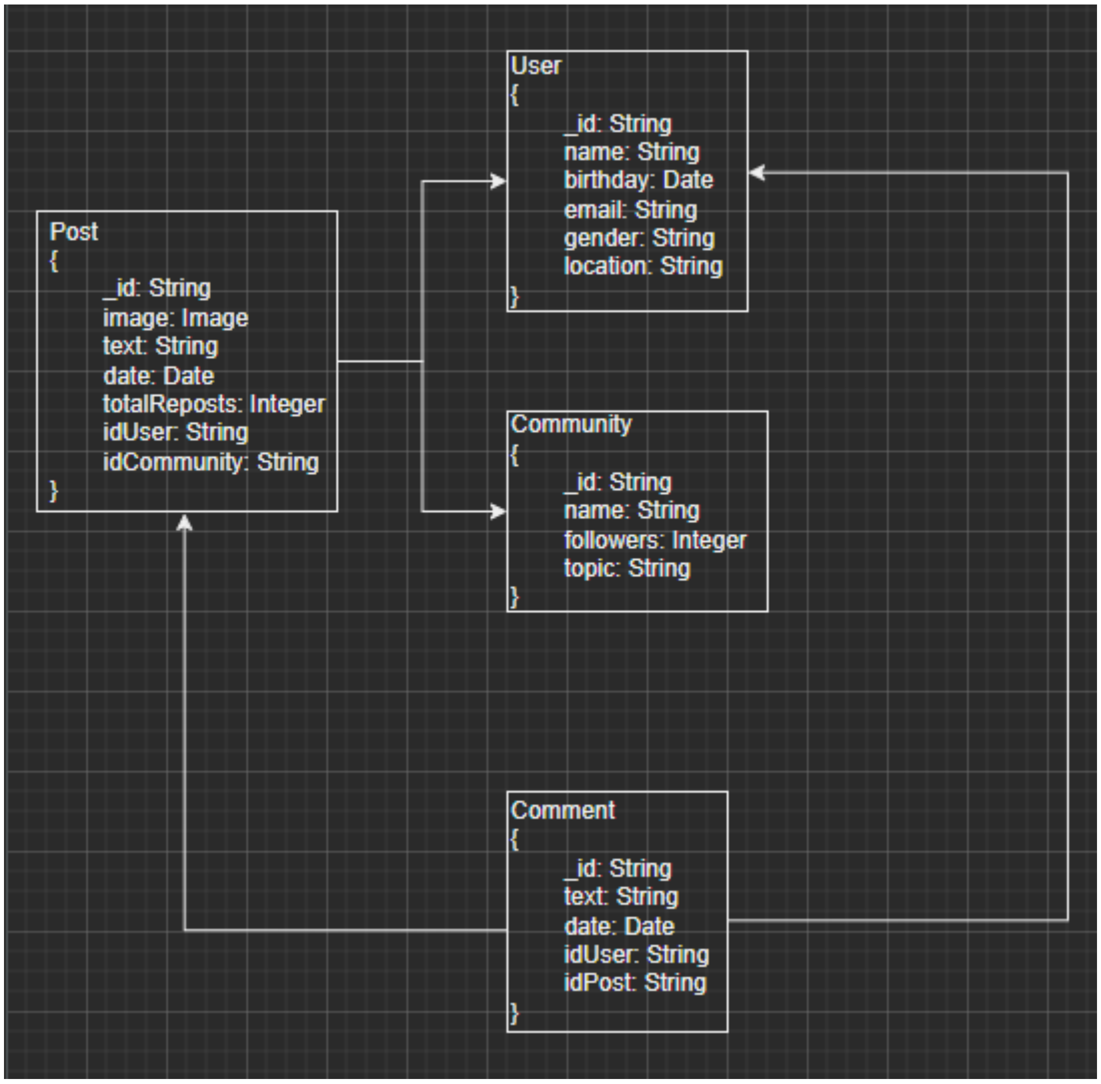

Figure 7: Document schema of the data stored in document database.

As said before, in Fig. 8 we declared the relationships and classes that were missing. A user can follow either a community or another user, the type of the relationship dictates the role that the user takes, this is to address the different types that we can have in different social networks. Furthermore, a reaction can be in a comment or in a post, with the type of the reaction as an attribute of the reaction node itself (to address the different reactions in different social media). An important observation is that we are only specifying the essential fields of data in order to save resources and avoid having two types of databases with the same information, we just need the id to identify each node and the attributes that were not considered for the document schema, any other data can be looked up in the document database.

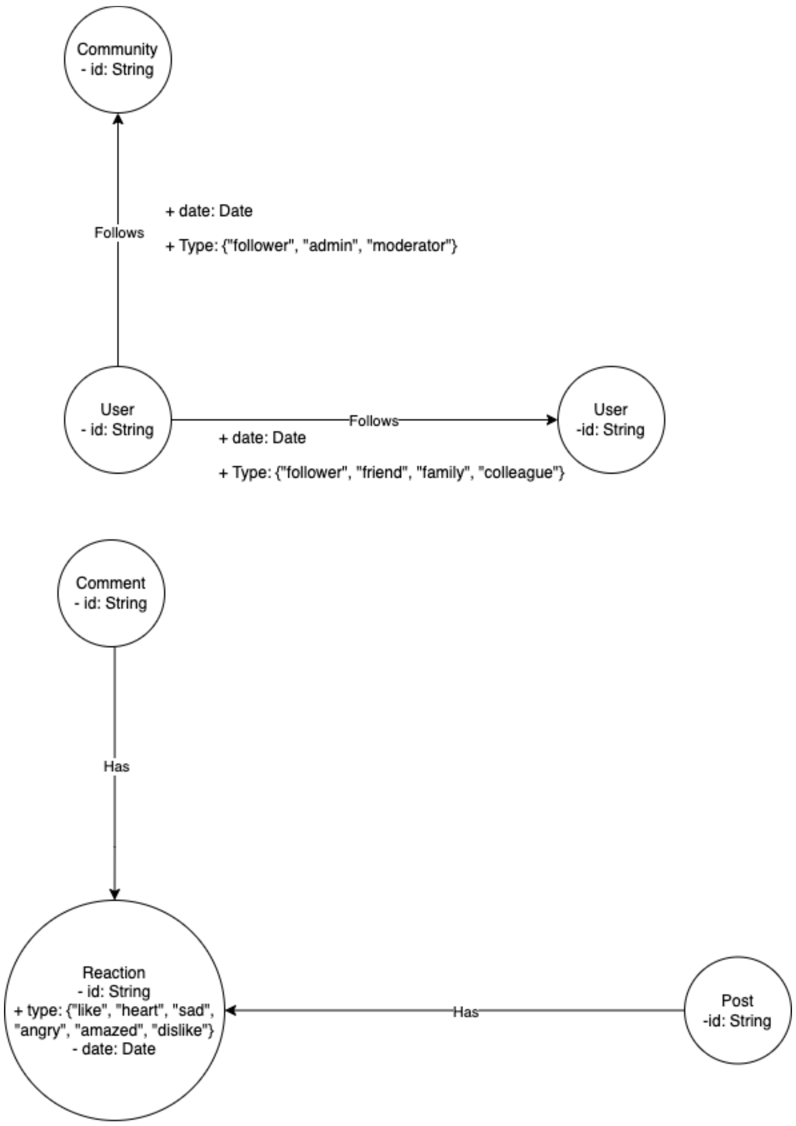

Figure 8: Graph schema of data stored in graph database.

Queries

For the document base the following queries are proposed.

- Information about the user: name, birthday, email, gender, location
- Deletion of user account (dangerous because it would erase all profiles from all social networks)
- Number of posts published by a user
- Total number of comments of a user
- Total number of reposts from all the posts that the user has
- Information about the community: name, followers, topic
- Number of posts published by a community
- Total number of reposts from all the posts that the community has
- Total number of comments of a community
- Creation of new posts
- Creation of new comments
- Update post
- Update comment
- Deletion of comment
- Deletion of post

For the graph base the queries that are proposed are the following.

- Number of friends/followers of users
- List of friends/followers of users sort by alphabetical order or date in which they established a relationship
- Number of communities followed by a user
- List of communities followed by a user sort by alphabetical order or date in which the user joined
- Most used reaction in comments
- Most used reaction in posts
- Unfollow communities
- Unfollow users
- Creation of reactions to posts and comments
- Deletion of reactions to posts and comments

For global queries, using both the graph and the document base, the following queries.

- Joined data to create a view of user profile with general information as well as the users or the communities that they follow
- Joined data to create a view of community profile with general information as well as information about the users that follow them
- Joined data to create a view of information about individual comments (including reactions) sort by date and with a reference to the post they belong to
- Joined data to create a view of information about each published post according to date, including reactions
- Joined data to create a view of information about posts of other users filtered by date or type of relationship
- Joined data to create a view of information about posts of communities filtered by date

Architecture

In Fig. 9 we can see the general architecture for the solution, while the user is interacting with the application, the backend recollects data from users via the API's that these platforms have and makes CRUD operations in order to reflect the information that we have in these social networks. Replicas of data are periodically managed in order to ensure availability, queries are requested and processed by our application whenever the user changes the view.

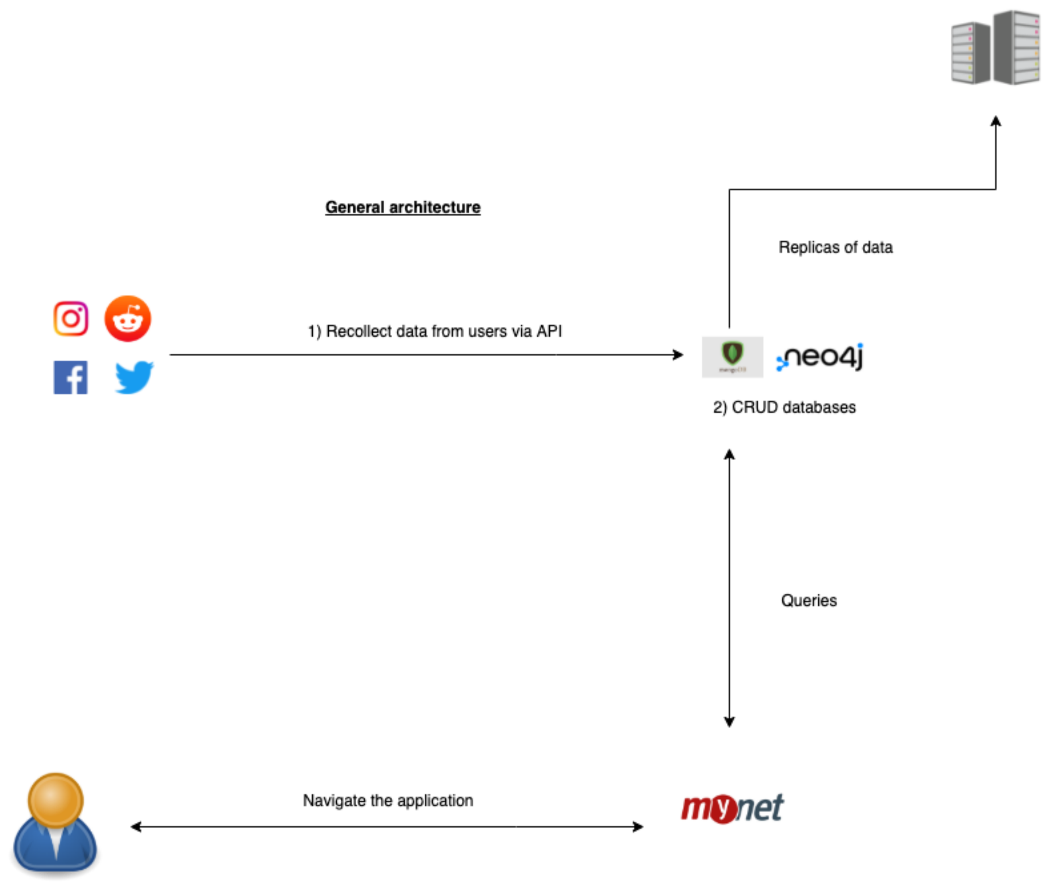

Figure 9: General architecture of application

Fig. 10 introduces the ETL diagram when requesting data from Facebook, Twitter, Reddit and Instagram. The extraction process is via API and then the backend handles where the data is going to go, either MongoDB (document database) or Neo4j (graph database), databases to process queries are selected depending on the nature of the data that the query requires. Finally, the data is loaded into MyNet application.

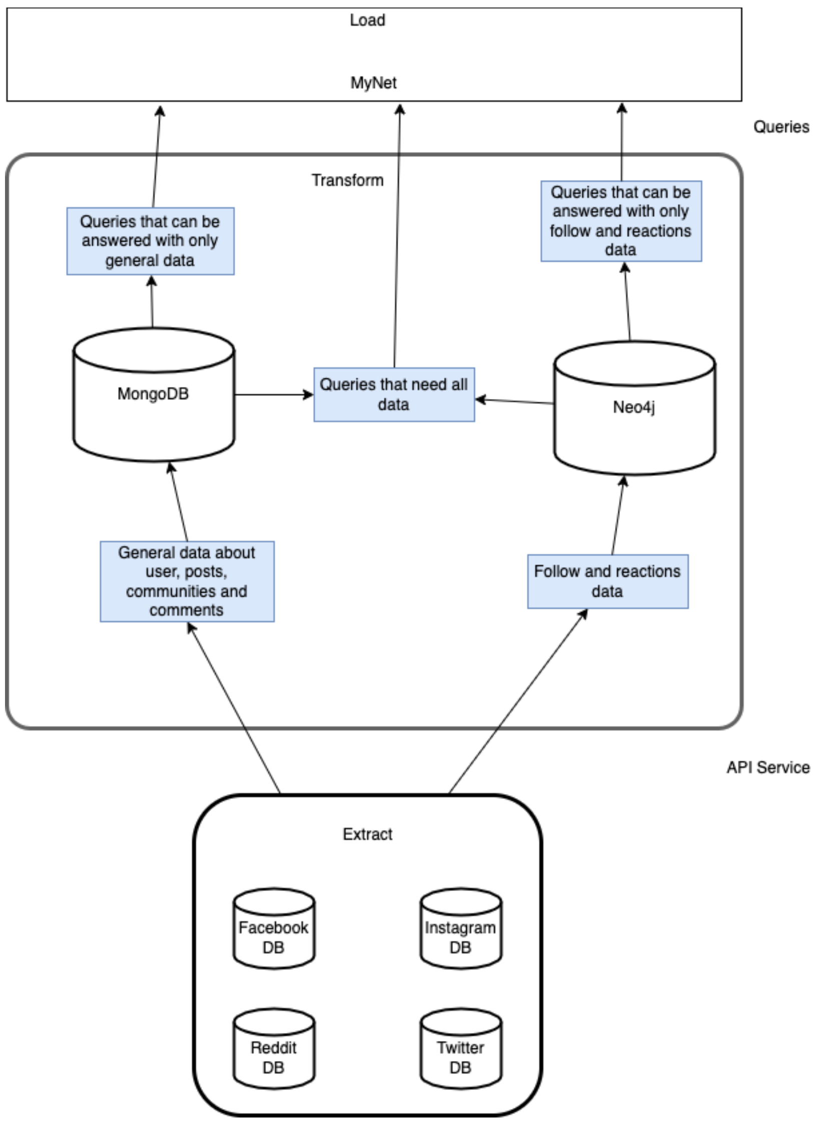
Load
MyNet
Queries
Transform
Queries that can be answered with only general data
Queries that can be answered with only follow and reactions data
MongoDB
Queries that need all data
Neo4j
General data about user, posts, communities and comments
Follow and reactions data
API Service
Extract
Facebook DB
Instagram DB
Reddit DB
Twitter DB

Figure 10: ETL process used for feeding application.