Posts collection:

* postId/
  + authorId: userId
  + authorName: userName
  + createdAt: timestamp
  + ***text: String***
  + ***image: url***
  + ***video: url***
  + location: String
  + stayingWith: [{
    - userId
    - username

}]

* + groupId: groupId
  + groupName: groupName
  + Subcollection comment

1 doc gồm có:

* + - comId: id
    - userId: userId
    - username: String
    - text: String
    - avatar: url
    - updatedAt: timestamp
    - replies: [{
      * userId: userId
      * comId: generate uuid
      * username: String
      * text: String
      * avatar: url

}]

* + tags: [String]
  + priority: Int

Counters collection:

Khi tạo post, tạo thêm 1 doc trong này luôn

* postId/
  + num\_shards: Int
  + Subcollection shards

shardNumber/ (1,2,3,4,5,6,7,8,9,10)

* + - count: int

Users collection:

* userId/
  + avatar: url
  + block: [userId]
  + email: email
  + id: userId
  + Map location: {
    - denyExposingLocation: bool
    - Geohash
    - Latitude
    - Longitude

}

* + Username
  + Subcollection LikedPost: [postId]
  + Bio {
    - Biography: String
    - showBio: bool
    - Work: String
    - showWork: bool
    - Education: String:
    - showEducation: bool
    - Current Location: String
    - showLocation: bool
    - Birth Place: String
    - showBirthplace: bool

}

* + Favorite activities: [{
    - Activity: String
    - Image: url

}]

* + Group: [{
    - groupId
    - name
    - avatar

}]

* + pendingGroup: [groupId]
  + Subcollection ownPosts (added by Cloud Functions watch\_post)

Friends collection:

* userId/
  + Subcollection ownRequest

-friendId/

* + - userId
    - username
    - avatar
  + Subcollection friendRequest

-friendId/

* + - userId
    - username
    - avatar
  + Subcollection allFriends – replaced by relationship in Neo4j

Group collection:

* groupId/
  + avatar
  + name
  + members: [{
    - avatar
    - username
    - userId

}]

* pendingRequest: [

{

* + - username
    - avatar
    - userId

}]

* + chatId
  + Subcollection posts: …

Saved collection:

* userId/
  + postId
  + postContent
  + postMedia

Notifications collection:

* userId/
  + content
  + triggerImage
  + createdAt
  + isRead

Hashtag collection:

* content/
  + content
  + postCount (updated by cloud function watch\_post)
  + Subcollection posts (added by Cloud Functions watch\_post)

***UserChats collection:***

* ***userId/***
  + ***chatId***

***Chats collection:***

* ***chatId/***
  + ***Subcollection chat …***
    - ***Messages***
    - ***Messages:***
      * ***Share ? => url => <a href=”/posts/postId”/> tag***

User Bio + Liked activities -> create post + like counters -> share + saved -> comment -> Notify -> Friends

* Block?
* - Group?
* - Maps -> Hashtag -> Neo4j
* Algolia + Recommend algorithm

Cloud Functions:

* WatchPost:
  + Thêm data vào Hashtag, Users
  + Thêm data vào Neo4j

Algolia Full Text Search

[Đồng bộ data từ Cloud Firestore lên Algolia thông qua Firebase Functions (viblo.asia)](https://viblo.asia/p/dong-bo-data-tu-cloud-firestore-len-algolia-thong-qua-firebase-functions-3P0lP9Om5ox)

[Full-text search  |  Firestore  |  Firebase (google.com)](https://firebase.google.com/docs/firestore/solutions/search?provider=algolia)

Recommend system

Distributed counter

[Distributed counters  |  Firestore  |  Firebase (google.com)](https://firebase.google.com/docs/firestore/solutions/counters)

Cloud scheduler -> Cloud Function -> update hashtag count:

* [Scheduling Google Cloud Functions to Run Periodically | by Gaurav Singh Tanwar | Medium](https://gauravtanwar1.medium.com/scheduling-google-cloud-functions-to-run-periodically-4fd3e763e78f)
* [Schedule functions  |  Cloud Functions for Firebase (google.com)](https://firebase.google.com/docs/functions/schedule-functions?gen=2nd)

Message pagination:

* Realtime structure

Online status

* [Build presence in Cloud Firestore  |  Firebase (google.com)](https://firebase.google.com/docs/firestore/solutions/presence)

Location hash -> distance?

Friend Request….

***recommend***

So here is our data structure

posts(Collection) --postid(document)

Title.

Description.

Image.

timestamp.

priority

This is a simple post structure with basic details. You can see I have added a Priority field. This field will do the magic.

How to use Priority.

We should query the posts that start with the higher priority and ends with lower priority.

When a user posts a new Post. Assign the current timestamp as the default priority.

When the user upvotes (Likes) a post increase the priority by 1 minute(60000 milliseconds)

When the user downvotes (Dislike) a post decrease the priority by 1 minute (60000 ms)

You can reset the priority every 24 hours. If you start browsing the feed today morning you will see posts with the last 24 hours in past. Once the 24-hour duration reached you can reset the priority to the present time. The 24-hour limit can be changed according to your needs. You may want to reset the limit every 15 min. because in every 15 min 100s of new posts might have added. This limit will ensure the repetition of content in the feed.

So when you start scrolling the feed you will get all the trending posts first then lower priority posts later. If you post a post today and people start upvoting it. It will get an increased lifetime, thus overpowers the poor content and when you downvote it, it will push down the post as long as users will not reach it.

Using timestamp as a priority because the old posts should lose priority with time. Even the trending posts today should lose the priority tomorrow.

Things to consider:

The lifetime can vary according to your needs. The bigger the user base. You should lower the lifetime value. because if a post posted today is upvoted by 10,000 users it trends 6.9 days in the future. And if there are more than 100 posts that have been upvoted by more than 10,000 users then you will never get to see a new post in those 6.9 days. So a trending post should hardly last a day or two.

So in this case you can give 10 seconds lifetime, it will give 1.1 day lifetime for 10,000 upvotes.