Yodelr - a micro-message communications platform

Backstory

The year is 3023. After digging for pirate treasure in your backyard, you have unearthed a time capsule more than a thousand years old. Peering through its contents, you find mystical mentions of what appears to be a service used for sharing messages of 140 characters with the world. You deduce that it featured some way of associating zero or more "topics" or "tags" with each such message by prefixing these with the character '#'. The system would then be able to list the most frequently used such topics for a given period of time, letting everyone know what truly mattered in the world at that particular instant.

Realizing that this could be a game changer, you fire up your code editor and set to work creating a first iteration of a similar system.

For the first iteration, you decide to keep things simple and to only reproduce a subset of the functionality found in the ancient tomes of knowledge:

- Users can register by name.
- A registered user may write posts of up to and including 140 characters. Such a post may contain zero or more topics, prefixed by "#" for the world to observe.
- All the posts a user has written may be retrieved.
- People curious about what's currently hot and trending can get a list of the most popular topics within a specified time interval.
- An already registered user may be deleted. In the name of privacy, this will also delete all their posts from the system.

You decide to name this system Yodelr.

The mission

Enclosed with this document, you should find a directory named yodelr, containing the files Yodelr.java, yodelr.h, and yodelr.py. These contain the barebones interface for the Yodelr service in Java, C++, and Python, respectively.

You are to create an implementation of the provided API interface that meets the functional requirements outlined as part of the API spec below. You can assume the following:

- Only a single thread is calling the provided API.
- The data set can fit entirely in memory on a single machine.
- It is fine to throw exceptions out of the API methods.

Before you get started, be aware of the following limitations of your chosen solution:

- Your implementation must be written in either Java (up to and including Java 17), C++ (up to and including C++20), or Python (version 3.6 and above).
- You may not alter the provided base Yodelr interface (your actual implementation may, of course, contain any superset of the interface's functionality).
- You may only use the standard library of your language of choice (classes starting with java.*for Java, STL for C++, modules in the Python distribution).
    - Exception to the above rule: you may use any testing/mocking library.
- State must be kept entirely in memory. No writing or reading to/from the disk or network, directly or indirectly, is allowed.
- Similarly, no external programs or services may be accessed.

Aside from this, you have full freedom in how you choose to approach this task.

We'll be looking in particular at the following:
- How you choose to model the solution
- Appropriate use of algorithms and data structures
- How do you test your code for correctness

Please submit your solution in a yodelr_yourlastname.zip file and include instructions on how we can compile the code and run the tests.

Have fun!

Yodelr API Spec

addUser(userName)
Adds a user with the given username to the system.

addPost(userName, postText, timestamp)
Adds a post for the given user with the given post text and timestamp. The post text can contain zero to many topics. A topic as part of the post text input is identified by the character '#' followed by one or more characters in the set [09azAZ_]. The topic name itself does not include the '#' prefix.

Example: the post text "#programming is #awesome!!" has two topics; "programming" and "awesome".

The following properties of the input can be assumed:

- Timestamps are unique and increase monotonically between each invocation of addPost.

- A timestamp is defined as the number of seconds elapsed since some arbitrary epoch. You do not need to care about what this epoch is, just the fact that higher timestamps represent events that have taken place later on the timeline than lower timestamps.

deleteUser(userName)
Deletes the given user and all posts made by the user from the system.

getPostsForUser(userName) > [postTexts]
Returns a list of all post texts made by the given user, sorted on descending timestamp.

getPostsForTopic(topic) > [postTexts]
Returns a list of all post texts containing the given topic sorted on descending timestamp.

getTrendingTopics(fromTimestamp, toTimestamp) > [topics]
Returns a list of all unique topics mentioned in posts made in the given time interval, sorted on descending mention count primarily, alphabetically on topic secondarily. The mention count for a topic is the number of posts that have mentioned this topic in the time interval. The time interval is inclusive. Remember that topics do not include the prefix “#”.

Yodelr Example

addUser(“john”)
addPost(“john”, “just #chilling today”, 1)
addPost(“john”, “eating #steak for dinner”, 2)
addPost(“john”, “ugh! this #steak tasted like dog food”, 3)
getPostsForUser(“john”) > [“ugh! this #steak tasted like dog food”, “eating #steak for dinner”, “just #chilling today”]
getPostsForTopic(“steak”) > [“ugh! this #steak tasted like dog food”, “eating #steak for dinner”]
getTrendingTopics(1, 3) > [“steak”, “chilling”]
getTrendingTopics(2, 3) > [“steak”]