CLIENT:

Channel and Stub Creation:

- The communication between server and client happens through the GRPC channel and stub that is created in the connectTo() method of client.
- The RPCs defined in the proto file uses this stub to send and receive messages, which are also defined in the proto file, from the server.
- The connectTo() also adds the user to the users database and timeline database via the add_user() RPC. This method checks for users who are currently active and users who connected to the server but may or may not be active currently, before creating an entry for the requesting user.

Commands:

- ***Follow*** : This command uses the stub created previously and calls the rpc method addTo(). Depending on the response of the server, the client knows if any user with the same name had previously registered with the user or if any active users have the same name.

- ***Unfollow*** : This command uses the stub created previously and calls the rpc method removeFrom(). Depending on the response from the server, the client knows if the rpc call was successful.

- ***List*** : This command uses the stub created previously and calls the rpc method getFollowersUsers(). Depending on the response from the server, the client knows if the rpc call was successful and also the users registered with the server as well as the users who are following the requesting user.

- ***Timeline*** : This command prompts the user to go into the timeline mode.

- Once the user in in timeline mode, he can post updates which are sent to the server through the rpc method updates(). At the same time, the user can also receive updates from other users who are being followed by the current user. These functionalities is achieved by a write thread and a read thread respectively.

SERVER:

Server uses the two son files : users.json and timeline.json as the database and modifies the files when it receives a request from a user.

- In ***addUser***() rpc mehod, the server checks if the username is existing in the database of registered users, if so, it just returns to the client after adding the username to the "*active_members"*. If the username is a existing in active_members, the server will not let the client to create the same username once again.

- In ***addTo***() rpc method, the server modifies the users.json file by adding a new key for the username of the requesting client. The database contains entries for followers and following for every user.

- ***removeFrom***() rpc method modifies the following and followers details of the users in the user.json file.

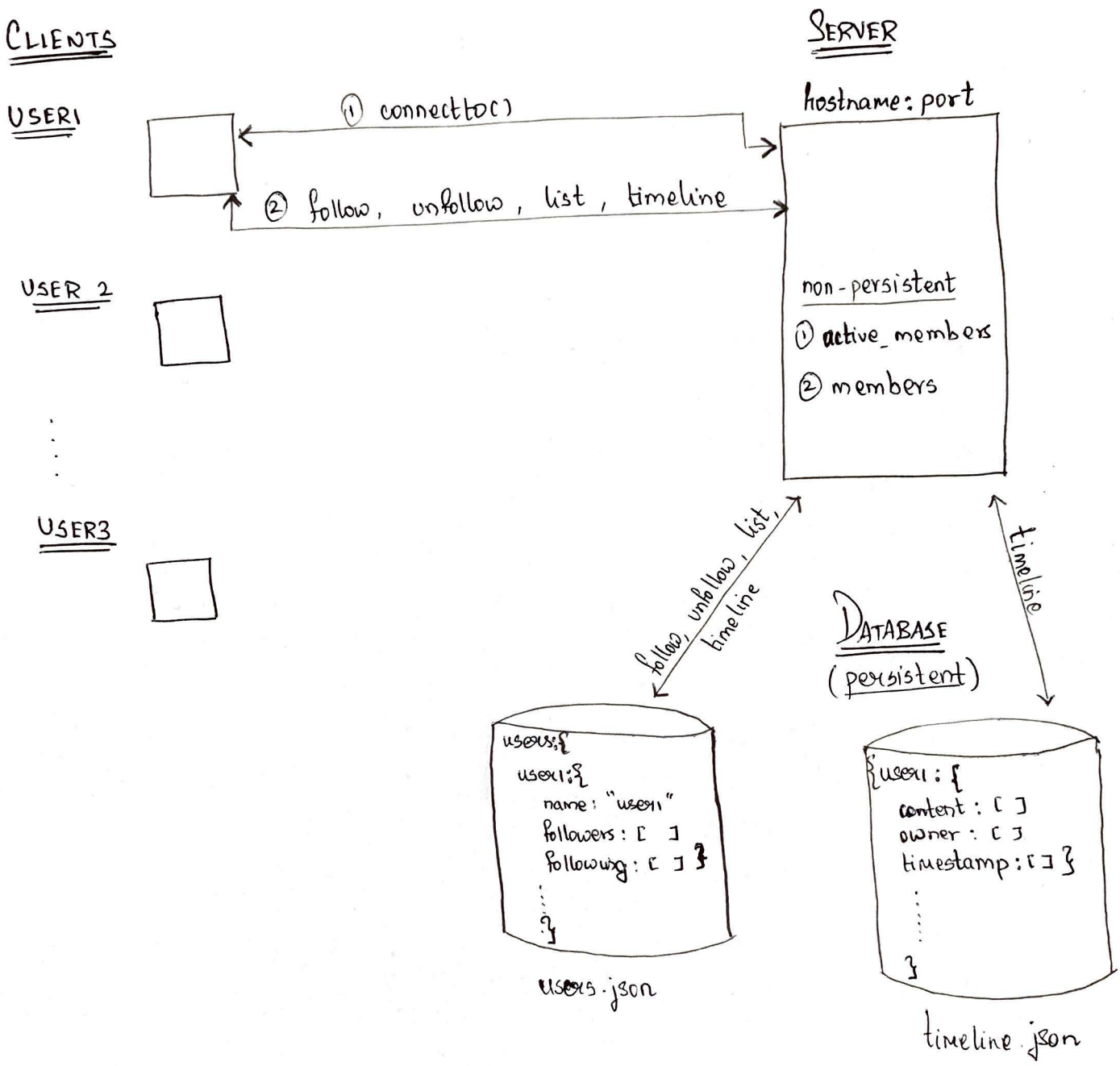
- The ***getFollowersUsers***() rpc method, lists the registered users and the users who are following the current user by accessing the users.json file.

- ***updates***() rpc method is invoked when the user inputs the command "timeline". The user then enters the timeline mode which he/she can exit only by using ctrl+c. This method takes care of the update that a user in a timeline mode is trying to post and also the writing of new posts to the timeline of the users who are in timeline mode. This involves receiving the user's updates, reading the user's followers from the users.json file and accessing their stream objects from the non-persistent object "*members*", that stores the stream object as the value for the username key. The stream objects are then used by a streaming writer to write the updates to the users who are a part of the user's followers and who are a part of "*active_members*". This process verifies that the user to whom the server is writing to is active, is a follower and is in timeline mode.

Persistent and Non-Persistent Information:

- ***active_members*** : This is an unordered map which is used to identify the users who are currently active. The active users' information is used when a new user is trying to register with the server and when the server is trying to send the new post.

- ***members*** : This is an unordered map which is used to store the stream object information when any user gets into the timeline mode. Stream object is used when the server tries to send new updates to a user who is following the user posting updates.


CLIENTS
SERVER
USER1
① connectto()
hostname: port
② follow, unfollow, list, timeline
USER 2
non-persistent
① active_members
② members
USER3
follow, unfollow, list, timeline
timeline
DATABASE
(persistent)
users:{
user1:{
name: "user1"
followers: [ ]
following: [ ] }
}
users.json
{user1: {
content: [ ]
owner: [ ]
timestamp: [ ] }
}
timeline.json

Acknowledgements:

- GRPC route guide server and client code : referenced the usage of stubs and rpc methods.
- GRPC helloworld server and client code : referenced the usage of stubs and rpc methods.
- Janvi Palan : discussed design ideas.