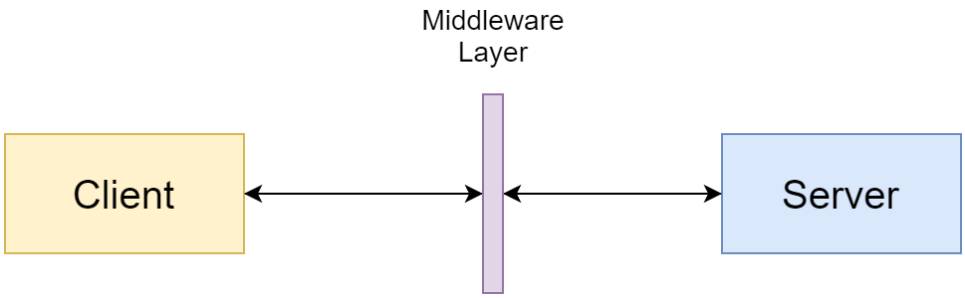
CS433: Project Report - Mini-Tweet

Members

Dyavarashetty Peeyush	17110053
Saumitra Sharma	17110135
Rushil Shah	17110140

Design Document



Middleware-Layer: We have implemented a middleware layer which contains shared functionality for both *Client* and the *Server*. It acts as a link between client and server. To be able to send requests to the server, the client has to be authorized(registered/logged-in) first. Middleware has two broad classes:

- **Interaction:** This class has methods to present the interface used by client
- **Authenticate:** This class has methods which are used by the server in order to check if a user is registered, or if the user is trying to log in, then we check if the credentials are correct.

Client: Out client uses the *Interaction* class of the middleware to access the options present at hand. As we proceed, and get responses the options change depending upon the state that we are presently in.

Server: Our Server waits for any new connection. Once any new connection is made, it allocates a new thread to handle the requests from that client. All the threads use a shared database. Any changes made to the database by any thread are reflected to the other threads too.

Communication: We are using the reliable **TCP** protocol to send out our packets. We have also applied a global timeout of 1 minute, after which the server leaves the waiting state and proceeds to the next state. Moreover, we are using a persistent multithreaded connection to avoid the delay caused due to the re-opening and re-closing of connection sockets.

detailing the technical aspects of your design. Communication paradigm, list of commands, associated messaging structure/data structures, state machine/model for client/server if any followed.

FSM: Please check out the Github repository for the **server FSM pdf file**, the client has a similar FSM with just one difference, i.e., it will be the one providing options while the server FSM will receive these options.

Register/Login → Interactive CLI → Select Option → Execute function → Interactive CLI → ….

Database Schema

We have used SQLite database (with concurrent setup) for our project
Has four relations :

- ➔ ***Users*** *with attributes:*
  - handle (primary key) : username of the user (is unique)
  - Name : name of the user
  - Password
- ➔ ***Tweets*** *Table with attributes:*
  - Tweet_id (primary key) : unique id to identify a tweet
  - Tweet_text : the tweet itself
  - Author(foreign key: handle(Users)) : handle of the person who made the tweet
- ➔ ***Follows*** Table *with attributes:*
  - follower(foreign key: handle(Users)) : the one who follows (let's say *A*)
  - gawd(foreign key: handle(Users)) : the one who is being followed (by *A*)
- ➔ ***Hashtags*** Table *with attributes:*
  - Tag: the text present after "#" in the tweet, if any.
  - t_id (foreign key: tweet_id(Tweets)) : the corresponding tweet_id of the tag

Feature Checklist

- ✓ Register New Users
  - After the user provides the credentials, we store them in a table “Users”
- ✓ Log-In existing Users (option 1)
  - We verify the credentials supplied by the user by matching them against the ones that are stored in our database
- ✓ Logging Out of a Session (option 8)
  - Save the changes made during the session to the database and log out
- ✓ Get Updates (option 3)
  - Fetches the tweets of the people the user follows (stateless)
- ✓ Search Registered Users (option 2)
  - Follow them, unfollow them and get their tweets
- ✓ Post Tweet (option 1)
  - The user is prompted to input the tweet, which is then stored in the *Tweets* Table. If there are any words with “#” at the start of it, then the Hashtag is stored with “tweet_id” in the *Hashtags* Table.
- ✓ Search Tweets with Specific Hashtags (Categorization) (option 1)
  - User is prompted for a hashtag, if there are any tweets with the hashtag, we send the tweets to the client and print them on the client-side, otherwise, we do not print anything.
- ✓ Threaded Server which can handle multiple clients at the same time
  - The server is constantly listening for new connections, once found it waits for the client to get authorized and then as the user requests something, it proceeds according to the FSM
- ✓ Features are accessible only after the user is authenticated
- ✓ The prompt for password hides the input
- ✓ Follow User (option 4)
  - The user is asked to input the handle he wants to follow. The entry is stored in the *Follows* table.
- ✓ Unfollow User (option 5)
  - The user is asked to input the handle he wants to unfollow. The entry is deleted from the *Follows* table if it exists.

Project Code

You can find out implementation of the code at the main branch of: **github/darkyed/mini-tweet**

For the detailed instructions and the information about the project structure please refer to the README.md file at the repository linked above

Dependency list

- ➔ Mininet VM
- ➔ Python 3.6
- ➔ **sqlite3** Library for Python3.6(Builtin)
- ➔ **threading** Library for Python3.6(Builtin)
- ➔ **getpass** Library for Python3.6(Builtin)
- ➔ **logging** library for Python3.6(Builtin)

References

- ➔ Concurrent operations on sqlite in python: StackOverflow
- ➔ Mininet Walkthrough: http://mininet.org/walkthrough/