Offline Messenger

Iosub Miruna-Elena

group E3

$2^{nd} year$

1 Introduction

The client/server application "Offline Messenger" will provide the ability to send messages between users. To be able to send a message, the user must be logged in, an operation performed by the server. After choosing an user from the database, either online or offline, the menu will have the following options left: "Log out", "Write message", "See new messages" "Reply" etc., the options are chosen by the user and sent to the server. Each user who receives a message can reply to the message (reply option) or ignore it. If an offline user receives a message, he/she will have the options to read the message once logged in. Each user that is logged in can write a message to another user, it doesn't matter if he/she is online or offline. A structure with Username and Message will be sent to the server. Thus, the server knows the content of the message. At the same time the server will save the conversation in a database file.
Each online user will be able to view the history of conversations with another user. The user's name will be sent to the server to see the history, the server will read from the database for a given user and send the conversation to the client.

2 Used technologies

I chose to use the TCP/IP connection-oriented protocol to have a permanent connection between users and the server. What really convinced me was the data sent from the client to the server and vice versa delivered without errors. At the same time, I also used the property of multiplexing, having several threads running on the same host. Concurrent TCP is used because it is easier to connect several users and to be able to transmit and receive information bidirectionally.

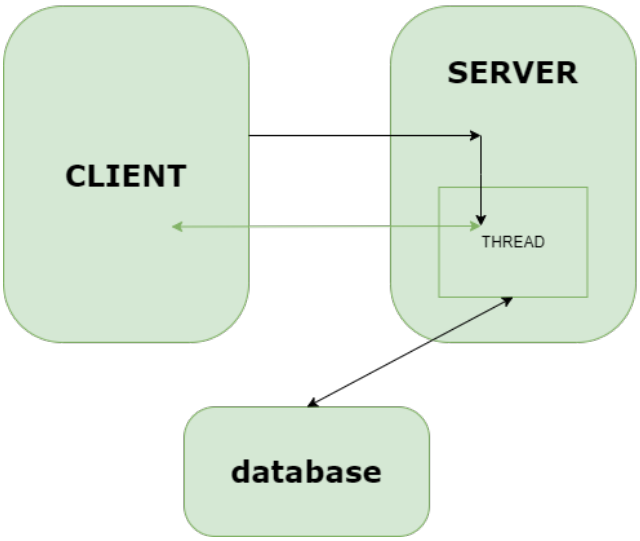


Fig. 1. Application's architecture.

3 Application architecture

3.1 Data Storage

The application will use a sqlite3 database that contains 2 specific tables called Users and Messages which are not depending on each other, both representing different information about the application.

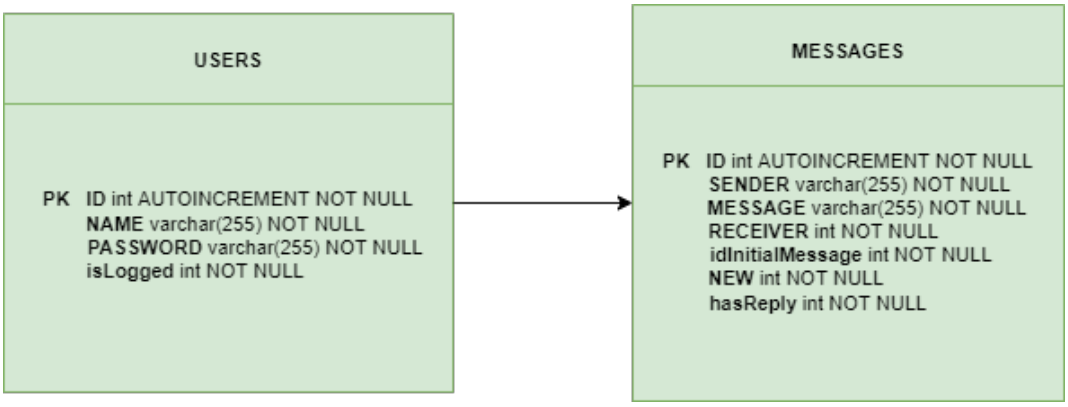


Fig. 2. Database architecture.

3.2 Menu

The menu will be displayed whenever the client wants to check all the options in order to use the application, using the "menu" command.

menu - The client can check the menu.

register - The client sends the data needed to register in the database.

login - The client sends the data needed to log in (login $< username >< password >$).

logout - The client sends the data needed to log out from the database (logout $< username >$).

message - The client sends a message to another user, no matter if he/she is online/offline, the user will still be able to read the messages once it connects (message $< usernameReceiver >< message >$).

check new - The client is able to see new messages from the database that he/she received before connecting to the application.

check conversation - The client will have the option to check the whole conversation with another user that is displayed from the database (check $< usernameOfTheOtherPerson >$).

delete - The client will eventually be able to delete a message only if it has not been replied to by the other user (delete $< contentMessage >$).

quit - Once the client is done, command $< quit >$ will help closing the connection.

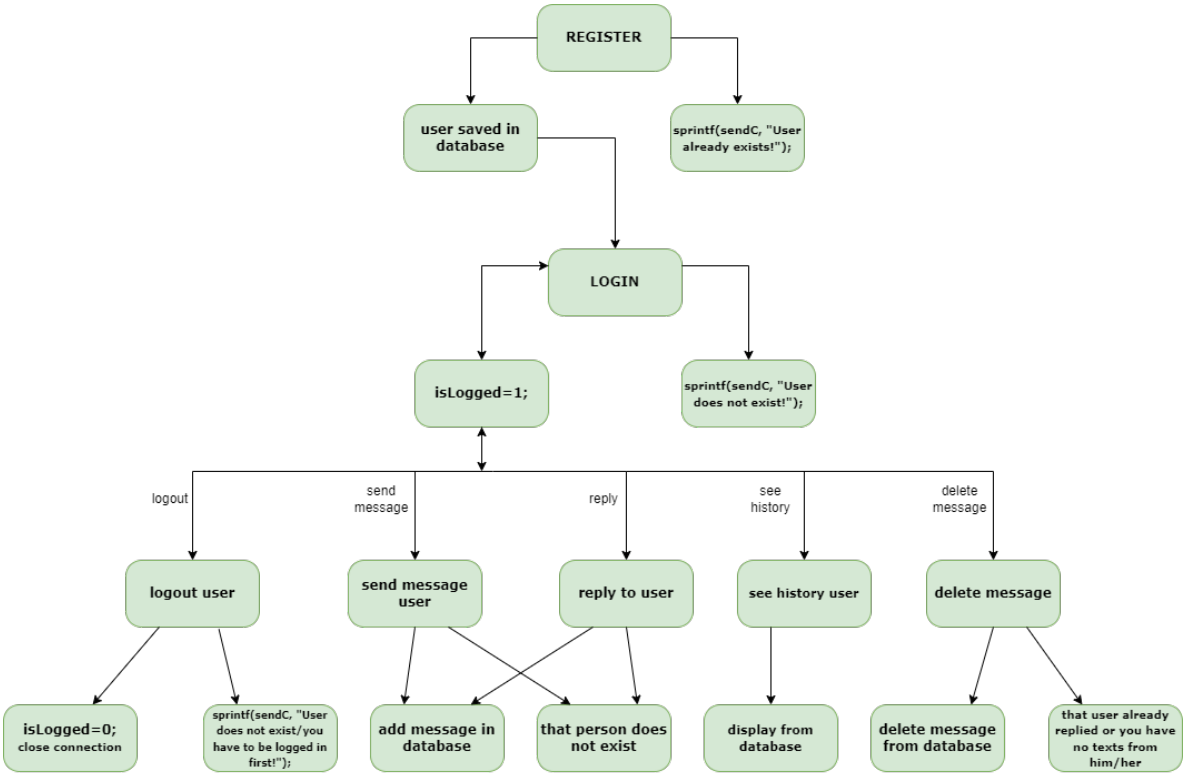


Fig. 3. Diagram of how the app works.

4 Implementation details

When the server is opened, it will permanently wait for the connection with the clients.
At this moment, a thread is created, and the communication between the respective client and server will be done only on the newly created thread.
Whenever a new client connects, he will be permanently listened to for receiving messages.
All tasks requested by the client (example: register, login, logout, check new messages, check conversation, delete message etc.) are done in a thread.
At the end of processing the requested data, the client will be permanently listened to by the server for receiving the next command until the thread is closed. The server will continuously wait for the commands received from each client.

menu - First option in case the user never registered is to enter the command for displaying the menu, this will help in understanding the commands.

register - Every time we have to create a new user it must be registered to the database. In order to do that, we will only need the command "register" followed by the username and its password.

login - Each user must be registered in order to log in. First, we check if it exists in the database. If so, we set the variable isLogged = 1 to let the database know that the user has been logged in.

logout - To log out an user we need to make sure the database shows it exists and by that we must check if it isn't already logged out. If not, we set the variable back to isLogged = 0.

message - In order to message someone (receiver), either online or offline, the user itself (sender) must be logged in. After checking it in the database, the command with the following syntax will be accomplished: message $< usernameReceiver >< message >$, updating the table $Messages$ from database.

check new - Each user is able to execute this command only if it is logged in. In case the database does not return any texts, the program will display the message "You don't have any new messages."

check conversation - Every online in user can check a conversation with a certain person following the syntax: check $< username >$. If there is no conversation to be displayed, the application will show the client "You don't have any conversations with that person or the user does not exist!".

delete - The users are also able to delete a message from any conversation, but only as long as the receiver hasn't replied yet. The message will be deleted from the database.

quit - The users are also able to disconnect from the server using $< quit >$ that will close the connection.

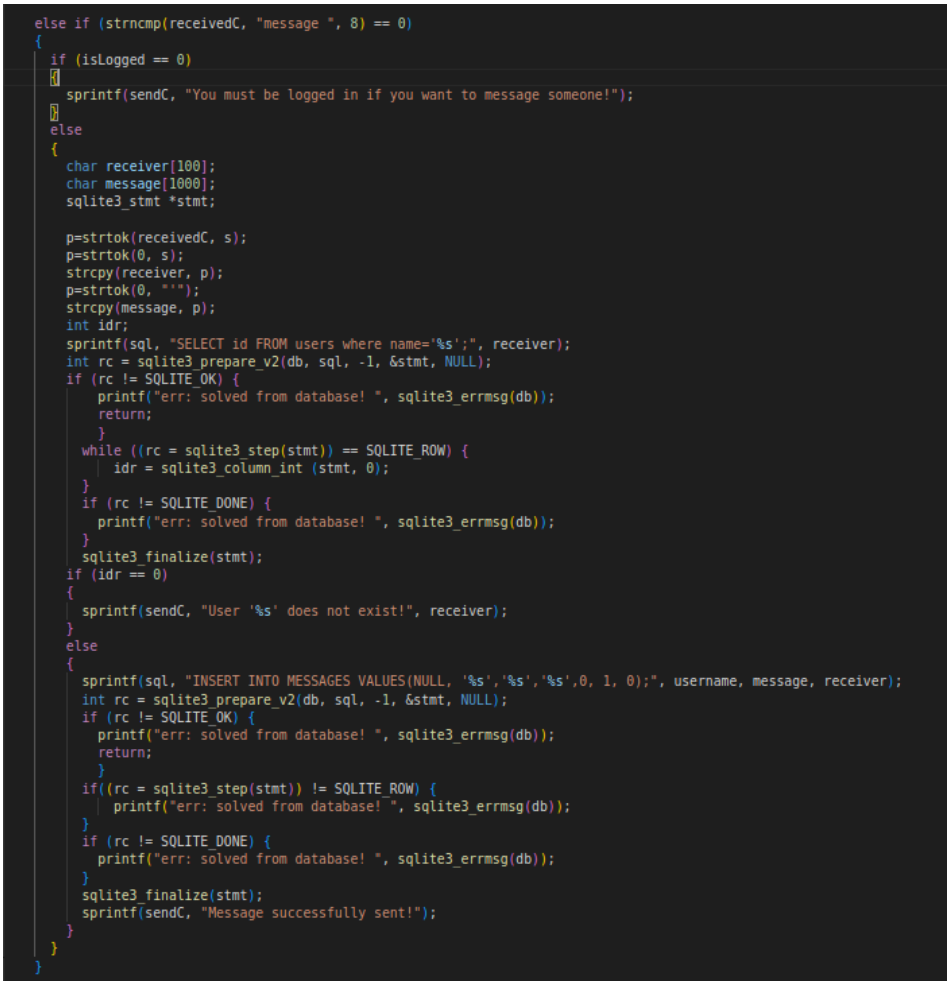
```
else if (strncmp(receivedC, "register ", 9) == 0)
{
    if(isLogged == 0)
    {
      int id = 0;
      p=strtok(receivedC, s);
      p=strtok(0,s);
      strcpy(username, p);
      p=strtok(0,s);
      strcpy(password, p);


      sqlite3_stmt *stmt;

      sprintf(sql, "SELECT ID FROM users where name='%s';", username);

      int rc = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
      if (rc != SQLITE_OK) {
        printf("err: solved from database! ", sqlite3_errmsg(db));
        return;
        }
      while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
          id = sqlite3_column_int (stmt, 0);
      }
      if (rc != SQLITE_DONE) {
        printf("err: solved from database! ", sqlite3_errmsg(db));
      }
      sqlite3_finalize(stmt);
      if (id == 0)
      {
        sprintf(sql, "INSERT INTO users values(NULL,'%s','%s',0);", username, password);
        int rc = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
        if (rc != SQLITE_OK) {
          printf("err: solved from database! ", sqlite3_errmsg(db));
          return;
        }
        while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
            id = sqlite3_column_int (stmt, 0);
          }
        if (rc != SQLITE_DONE) {
            printf("err: solved from database! ", sqlite3_errmsg(db));
        }
        sqlite3_finalize(stmt);
        sprintf(sendC, "User '%s' successfully created, now you can log in!", username);
      }
      else
        sprintf(sendC, "User %s already exists!", username);
    }
    else
        sprintf(sendC, "You have to be logged out in order to register!");
}
```

Fig. 4. How registering a new user works.



```
else if (strncmp(receivedC, "message ", 8) == 0)
{
  if (isLogged == 0)
  {
    sprintf(sendC, "You must be logged in if you want to message someone!");
  }
  else
  {
    char receiver[100];
    char message[1000];
    sqlite3_stmt *stmt;

    p=strtok(receivedC, s);
    p=strtok(0, s);
    strcpy(receiver, p);
    p=strtok(0, "'");
    strcpy(message, p);
    int idr;
    sprintf(sql, "SELECT id FROM users where name='%s';", receiver);
    int rc = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
    if (rc != SQLITE_OK) {
        printf("err: solved from database! ", sqlite3_errmsg(db));
        return;
        }
      while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
          idr = sqlite3_column_int (stmt, 0);
      }
      if (rc != SQLITE_DONE) {
        printf("err: solved from database! ", sqlite3_errmsg(db));
      }
      sqlite3_finalize(stmt);
    if (idr == 0)
    {
      sprintf(sendC, "User '%s' does not exist!", receiver);
    }
    else
    {
      sprintf(sql, "INSERT INTO MESSAGES VALUES(NULL, '%s','%s','%s',0, 1, 0);", username, message, receiver);
      int rc = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
      if (rc != SQLITE_OK) {
        printf("err: solved from database! ", sqlite3_errmsg(db));
        return;
        }
      if((rc = sqlite3_step(stmt)) != SQLITE_ROW) {
          printf("err: solved from database! ", sqlite3_errmsg(db));
      }
      if (rc != SQLITE_DONE) {
        printf("err: solved from database! ", sqlite3_errmsg(db));
      }
      sqlite3_finalize(stmt);
      sprintf(sendC, "Message successfully sent!");
    }
  }
}
```

Fig. 5. How texting another user works.

5 Conclusions

The project could be improved by implementing the option of creating a group chat, with clients that could be admins (they will have a separate menu that allows them to add/delete people from the chat) and normal participants of the group.

References

https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
https://profs.info.uaic.ro/ andrei.panu/