

LearNet

Ștefan-Vladimir Sbârcea

Faculty of Computer Science, Alexandru Ioan Cuza University of Iasi
`stefan.sbarcea@info.uaic.ro`

Abstract. LearNet is an application created to help students or anyone who wants to learn Computer Networks. The application is designed so that users have quick access to a lot of information in the field and can interact with other enthusiasts at the same time.

1 Introduction

In this report we will discuss in detail about LearNet application. This application is a client-server application where clients can interact with each other through chat or find useful information about computer networks. Every client needs an existing account to enter the application or they can create one if they have an invite code. Every chat, account and informational data is saved in an SQLite database. The information about computer networks that is presented in the application is taken from Alboaie Lenuta and Panu Andrei Computer Networks courses from Faculty of Computer Science [3].

The structure of this paper is as follows: Section 2 describes the technologies used in the creation of the application. Section 4 describes the architecture of the application. In Section 5 we present some implementation details including parts of the source code. Finally we draw some conclusions.

2 Used technologies

The application source code is written in C++ programming language.

To achieve the client-server connection we used the concurrent TCP model (multi-processing), which allows the possibility of connecting several clients to the server and TCP's reliable data transfer service ensures that the data stream that a process reads out of its TCP receive buffer is uncorrupted, without gaps, without duplication, and in sequence; that is, the byte stream is exactly the same byte stream that was sent by the end system on the other side of the connection [2].

The client GUI was made with Qt Creator, a cross-platform integrated development environment (IDE) built for the maximum developer experience [4].

To save the information we used SQLite database because it is lightweight when it comes to setup complexity and resource usage [1].

3 Application architecture

As we said prior the connection between server and client is done with TCP protocol and we create different processes to be able to serve clients concurrently. The server gets requests from the client it processes them and send back the data resulted. We created a database to save every client account information (username, password, rank, chats and chat messages) and the information about computer networks. In figure 1 you can see how the connection between server-client was made and in figure 2 are presented the inside workings of the application.

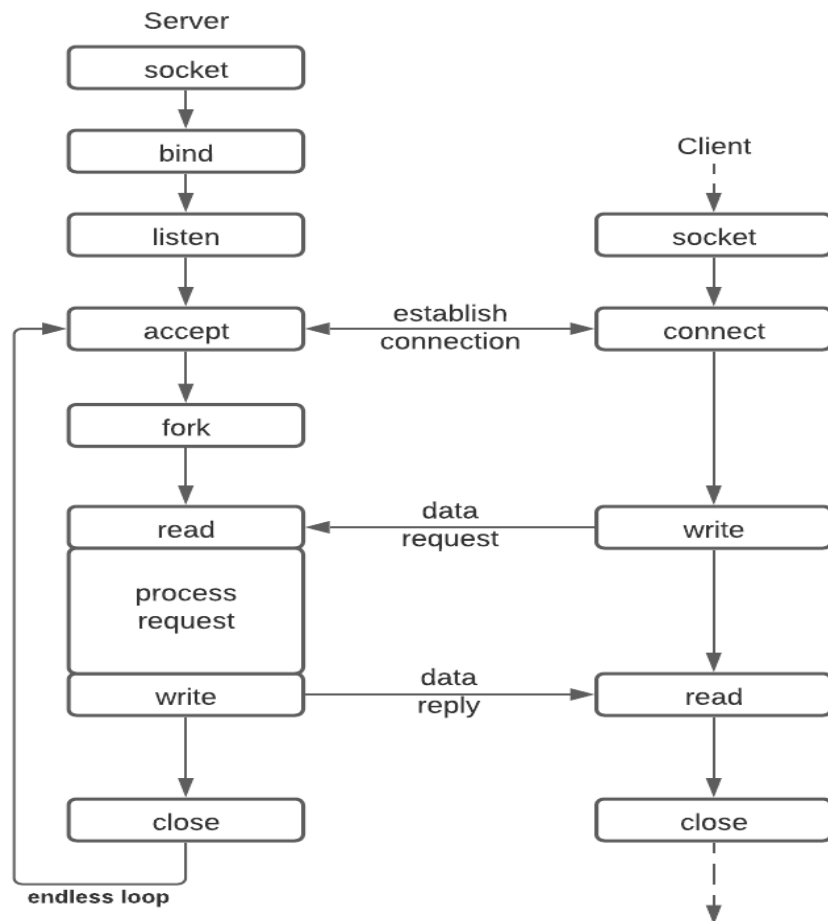
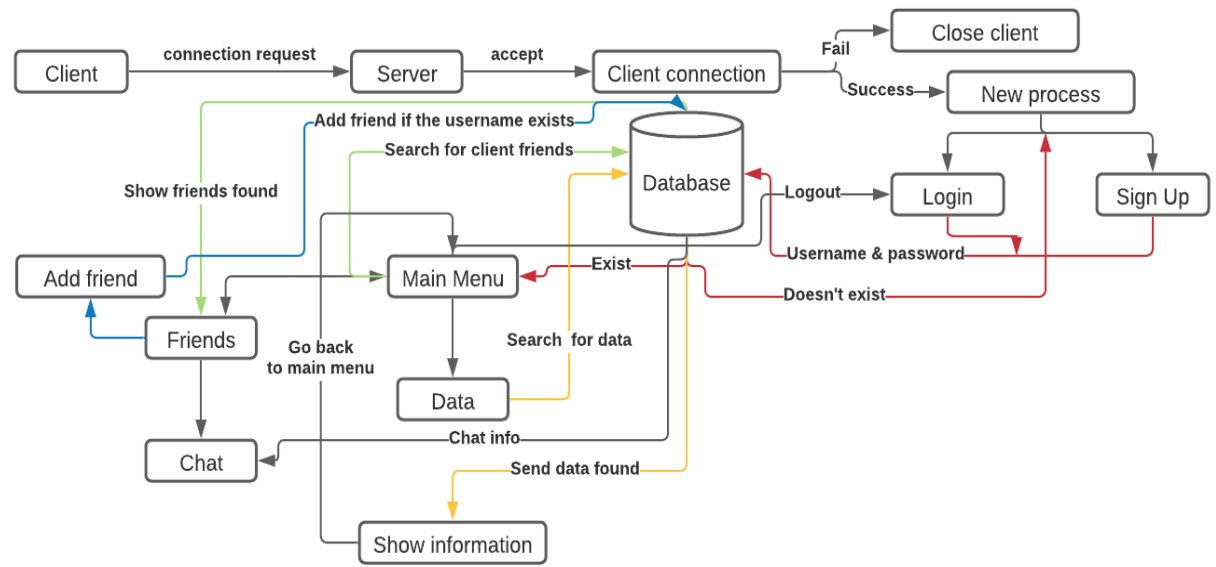


Fig. 1. Application diagram

**Fig. 2.** Application diagram

When the user starts the application it automatically connects to the server, if the server is not running it will throw an error, and the login/sign up menu will appear (figure 3)

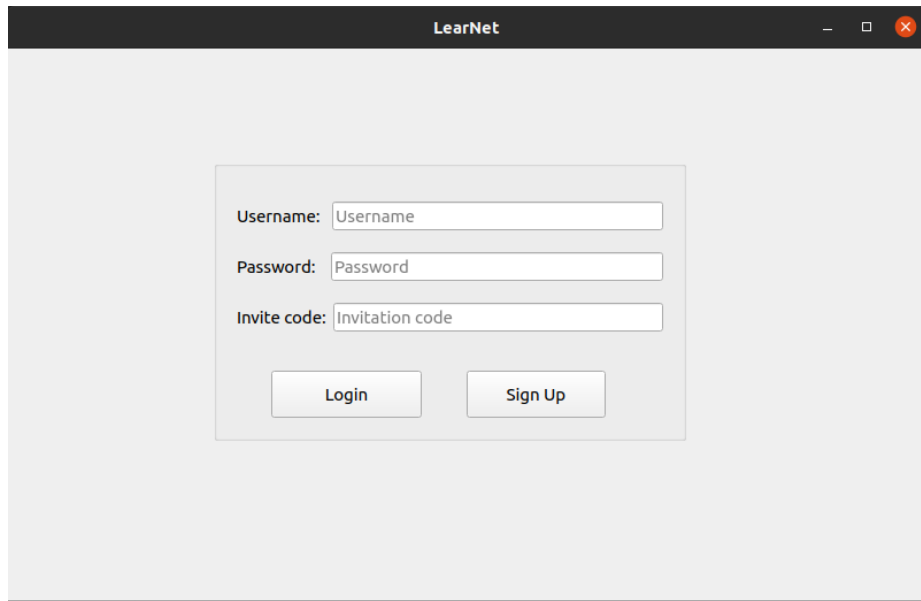
The image shows a screenshot of a web application window titled "LearNet". The window has a dark gray header bar with the title and standard window control buttons (minimize, maximize, close). The main content area is light gray and contains a centered white rectangular form. Inside the form, there are three input fields: "Username:" with a placeholder "Username", "Password:" with a placeholder "Password", and "Invite code:" with a placeholder "Invitation code". Below these fields are two buttons: "Login" and "Sign Up".

Fig. 3. Application login/sign up

, as said prior, if the user doesn't have an account he can create one by providing an username that doesn't exists in the database with a password and a valid invite code, that can be obtained from a friend that has the permission to generate one, if the username already exists, the invite code is incorrect or the password doesn't respect the minimum requirements a specific warning will pop-up (figures 4-5). After successfully signing up the user will be automatically logged in the application. (figure 6)

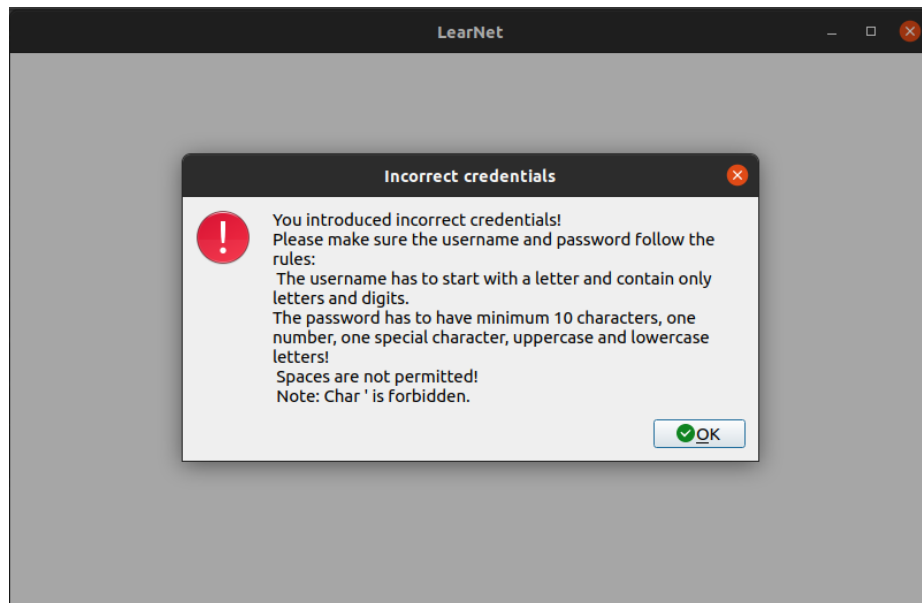


Fig. 4. Incorrect Credentials warning

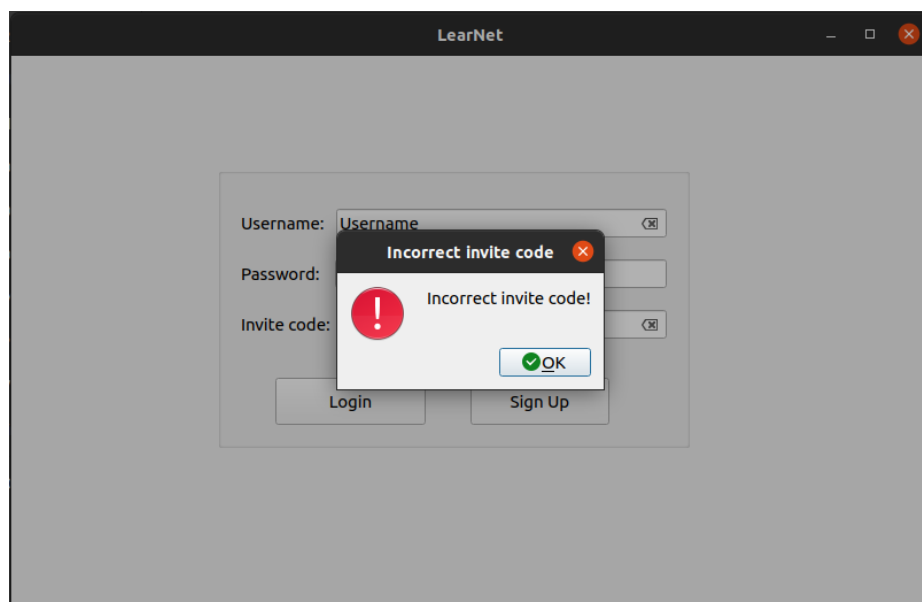


Fig. 5. Incorrect Invite code warning

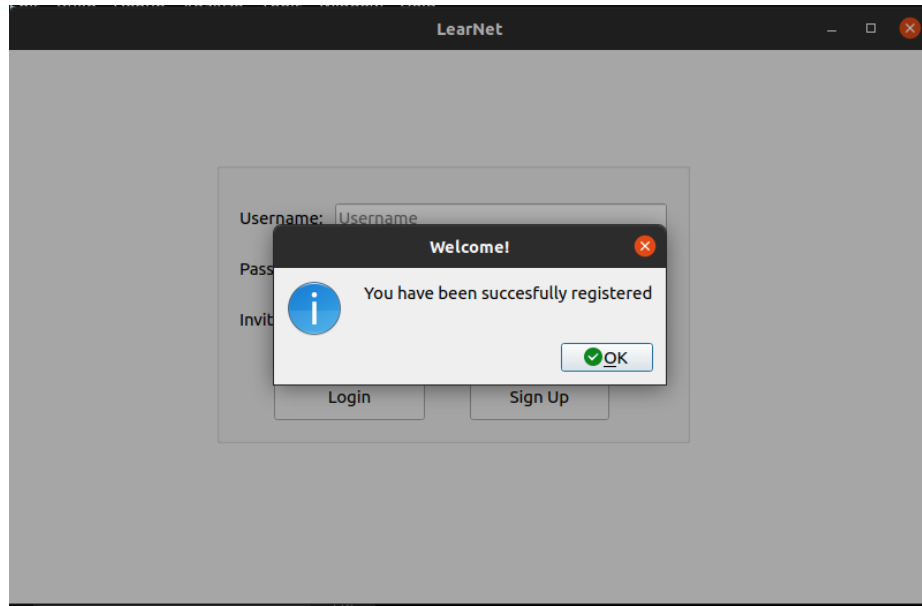


Fig. 6. Successfully registered message box

The users that already have an account can just press on "Login" after completing the "Username:" and "Password:" fields (the "Invite code:" field is ignored in this case), after pressing the "Login" button a pop-up will appear, if the username exists and the password is correct (is the same with the one introduced at the creation of the account) a "Welcome!" message box will appear (figure 7) and after closing it the user will be redirected to the application main menu, otherwise a warning will appear (figure 8)

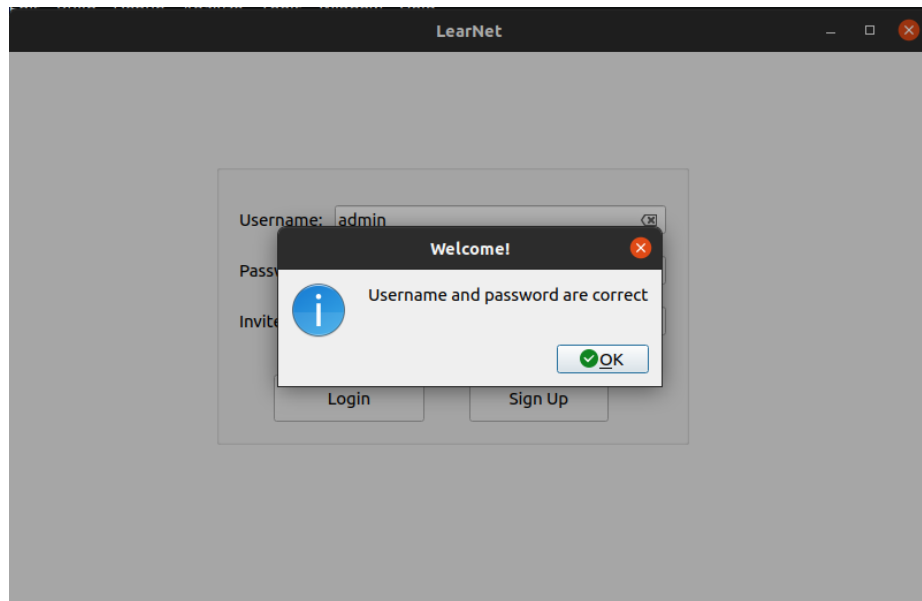


Fig. 7. Welcome! message box

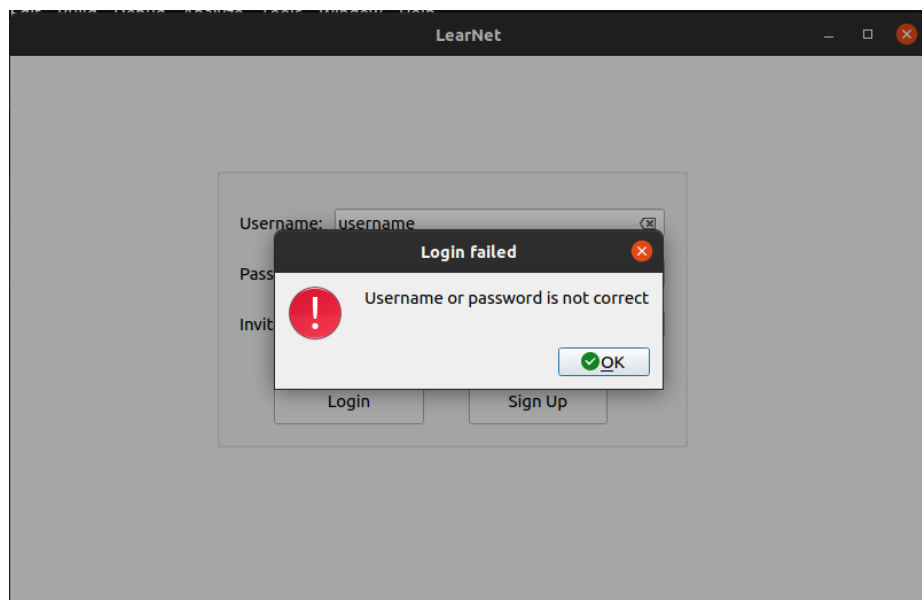


Fig. 8. Login failed warning

The main menu (figure 9) has 4 different buttons, the "Logout" button redirect the user to the login/sign up menu, the "Friends" button redirect the user to the friend list where it can add, remove or message a friend and generate an invite code if he has the permission (figure 10), the "Search" button helps the user find the information he wants, of course if it exists in the database, and finally the "Open chat" button will open the all chat selected on the list located to the left of the "Open chat" button (figure 11)

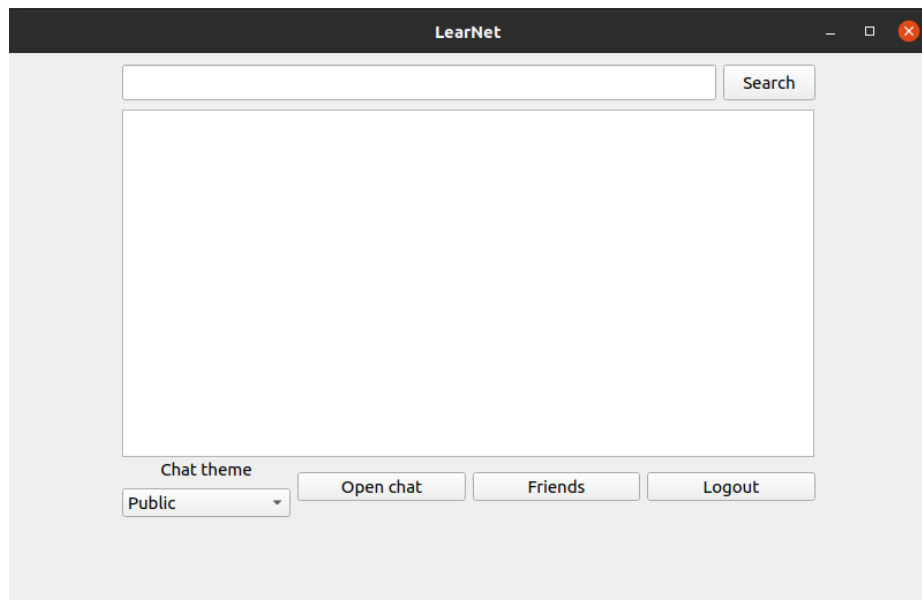


Fig. 9. Main menu

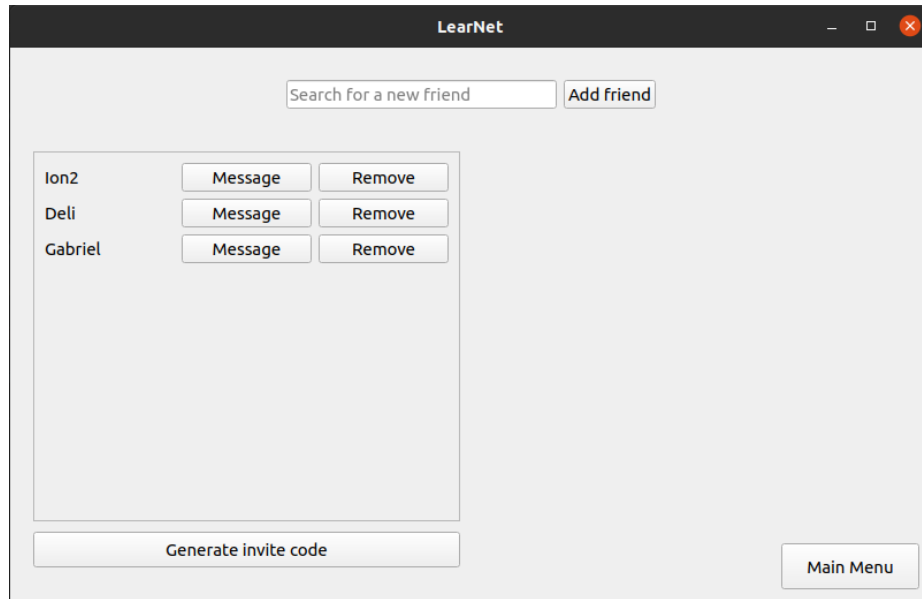


Fig. 10. Friends menu

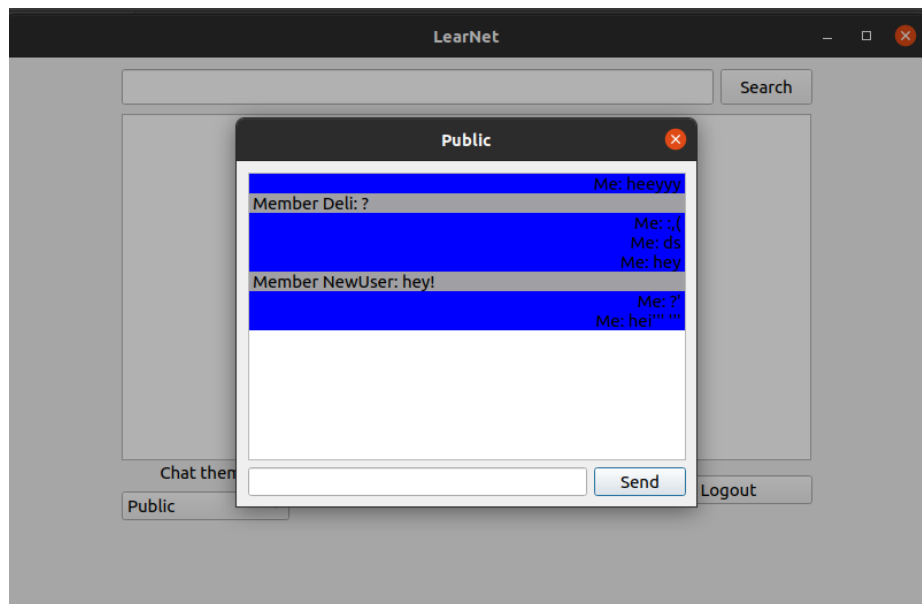


Fig. 11. All chat

As you have seen in the application diagram (figure 2) the database stands in the center of the implementation, in figure 12 we show how the database was designed. The chats between individual users are saved as different tables for every two users, that's why only the table **chats** appears in the diagram, which is used to save messages from every global chat. The **id** is unique and the **username** and **type** are primary keys.

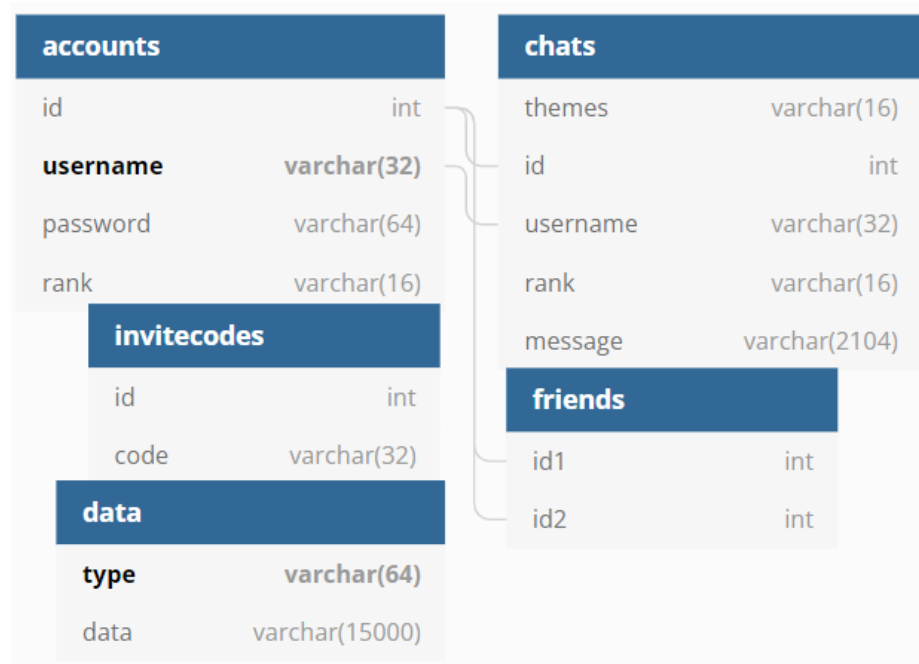


Fig. 12. Database diagram

The **invitecodes** table is used for sign up, only clients with an invite code are able to create a new account and every invite code is generated by an user with the specific rank.

4 Implementation details

4.1 Implementation use cases

- **Use case 1:** Search for data

Actor: Student

Basic Flow: A student decides to use the application to search for a piece of information he couldn't find elsewhere. He starts the application, login into his account, searches for the information and finds it.

Alternative Flow 1: He notices that he doesn't have an account so he needs an invite code to create one.

Alternative Flow 2: He doesn't find the information, so he asks for that information in the chat.

- **Use case 2:** Need help

Actor: Student

Basic Flow: A student needs help for a project related to computer networks, he asks on the LearNet application chat for help and someone helps him.

Alternative Flow 1: He doesn't want everyone to see what he doesn't know because it might make him look stupid so he adds the user to his friend list and speaks with him in private.

4.2 Implementation

The server source code is split in 8 files:

1. **utilities.h** - used as an umbrella, containing every library needed and 5 utility functions: `handle_error(char*)`, `int readBytes(int, char *, unsigned int)`, `void recvMsg(int, char *)`, `void sendMsg(int, char *)`, `replaceAll(std::string &, const std::string &, const std::string &)`;
2. **utilities.cpp** - contains the implementation of the functions mentioned above;
3. **dbutilities.h** - contains the definition of the functions needed to save or take data from database;
4. **dbutilities.cpp** - contains the implementation of the functions from point 4;
5. **server.h** - contains the **Server** class;
6. **server.cpp** - contains the implementation of the **Server** class constructor and the following functions: `void Server::acceptClients()`, `void Server::executeClient(int, char *)`, `void Server::acceptClientsThreads()`;
7. **commands.cpp** - contains the implementation of the **Server** class functions that analyze the data received from clients;;
8. **main.cpp** - contains the main function.

The client source code is split in 13 files:

1. **client.h** - contains the **Client** class used for the connection between server and client;

2. **client.cpp** - contains the implementation of the **Client** class;
3. **mainwindow.h** - contains the **MainWindow** class;
4. **mainwindow.cpp** - contains the implementation of the **MainWindow** class;
5. **mainwindow.ui** - stores the forms of the application login/sign up window in XML format;
6. **application.h** - contains the **Application** class;
7. **application.cpp** - contains the implementation of the **Application** class;
8. **application.ui** - stores the forms of the application main window in XML format;
9. **chat.h** - contains the **Chat** class
10. **chat.cpp** - contains the implementation of the **Chat** class;
11. **chat.ui** - stores the forms of the application chat window in XML format;
12. **interface.pro** - it's a multiplatform project file which qmake turns into platform-specific makefiles;
13. **main.cpp** - contains the main function.

Every byte stream, sent between clients and server, is prefixed by his length as shown in figure 13, where **readBytes(int,char*,unsigned int)** function is used to read byte by byte the information so as not to lose data.

```

Server

void sendMsg(int client, char *str)
{
    int size = strlen(str);
    if (write(client, &size, sizeof(int)) == -1)
        handle_error("[server]Error writeBufferSize(int).\n");
    if (write(client, str, size) == -1)
        handle_error("[server]Error writeBuffer(char*).\n");
}

void recvMsg(int client, char *str)
{
    int size = 0;
    if (read(client, &size, sizeof(int)) == -1)
        handle_error("[server]Error readBufferSize(int).\n");
    if (readBytes(client, str, size) == -1)
    {
        perror("[server]Error read().\n");
        close(client);
    }
    str[size] = '\0';
}

int readBytes(int socket, char *buffer, unsigned int x)
{
    unsigned int bytesRead = 0;
    int result;
    while (bytesRead < x)
    {
        result = read(socket, buffer + bytesRead, x - bytesRead);
        if (result < 1)
            return -1;
        bytesRead += result;
    }
    return 1;
}

Client

bool Client::sendBufferSize(int size)
{
    if (write(this->socketClient, &size, sizeof(int)) == -1)
        handle_error("[client]Error sendBufferSize(int).\n");
    return true;
}

bool Client::sendBufferChar(char* msg)
{
    if (write(this->socketClient, msg, strlen(msg)) == -1)
        handle_error("[client]Error sendBuffer(char*).\n");
    return true;
}

// Receive
int Client::receiveBufferSize()
{
    int size = 0;
    if (read(this->socketClient, &size, sizeof(int)) == -1)
        handle_error("[client]Error receiveBufferSize(int).\n");
    return size;
}

int Client::receiveBufferChar(char *str)
{
    int size = 0;
    if (read(this->socketClient, &size, sizeof(int)) == -1)
        handle_error("[client]Error readBufferSize(int).\n");
    if (size == -1)
        return -1;
    if (readBytes(this->socketClient, str, size) == -1)
        handle_error("[client]Error read().\n");
    str[size] = '\0';
    return 1;
}

```

Fig. 13. Send and receive functions server-client

We decided to explain and show how the login works. The client sends the command "login", the username and the password to the server and waits for the server to respond (figure 14). The server checks the command and calls function **login(int)** where it reads the username and password and searches for them in the database to see if there is a match in **accounts** table the function **searchUsrAndPwd(database, username, password)** returns -1 on failure and the user id on success ($0 \leq id$) (figure 15).

```

this->client->sendBufferSize(5);
this->client->sendBufferChar("login");

this->client->sendBufferSize(username.length());
this->client->sendBufferChar(usr.data());

this->client->sendBufferSize(password.length());
this->client->sendBufferChar(pwd.data());

int size = this->client->receiveBufferSize();

if(size >= 0)
{
    QMessageBox::information(this,"Login","Username and password are correct");

    ui->lineEdit_username->clear();
    ui->lineEdit_password->clear();
    ui->lineEdit_inviteCode->clear();

    ui->stackedWidget->setCurrentIndex(1);
}
else
    QMessageBox::warning(this,"Login","Username or password is not correct");

```

Fig. 14. Client-login

```

int Server::login(int client)
{
    int found = -1;
    char username[32], password[64];
    recvMsg(client, username);
    recvMsg(client, password);

    printf("Username: %s, Password: %s \n", username, password);
    found = db::searchUsrAndPwd(database, username, password);

    if (write(client, &found, sizeof(int)) == -1)
    |   handle_error("[client]Error sendBufferSize(int).\n");
    return found;
}

```

Fig. 15. Server-login

Another important part of the source code is the generation of the chats between users that is shown in the following figures (figure 6 and 7). The client creates a new object **Chat** and fills it with the messages the server provides from the database, if there are any.

```

void Application::openChat(const char* str)
{
    Chat * chat = new Chat(this);
    chat->setClient(client);
    chat->setUserId(this->userId);

    this->client->sendBufferSize(10);
    this->client->sendBufferChar((char*)"chatFriend");

    this->client->sendBufferSize(strlen(str));
    this->client->sendBufferChar((char*)str);
    chat->setFriendId(this->client->receiveBufferSize());

    chat->receiveMessages();

    chat->setWindowTitle(str);
    chat->exec();
}

```

Fig. 16. Client-chat

```

void Server::createChatFriend(int client, int id1)
{
    int finish = -1;
    char username[32], table_name[30], message[1000];
    recvMsg(client, username);
    int id2 = db::getUsrId(database, username);
    if (id2 == -1)
    {
        if (write(client, &finish, sizeof(int)) == -1)
            handle_error("[server]Error write().\n");
        perror("[server]Error createChat()");
        return;
    }
    if (write(client, &id2, sizeof(int)) == -1)
        handle_error("[server]Error write().\n");
    if (db::createChatTable(database, id1, id2, table_name) == -1)
    {
        if (write(client, &finish, sizeof(int)) == -1)
            handle_error("[server]Error write().\n");
        perror("[server]Error createChat()");
        return;
    }
    sqlite3 *db;
    if (sqlite3_open(database, &db))
    {
        perror("Error sqlite3_open()");
        return;
    }
    sqlite3_stmt *stmt;
    char sql[256];
    sprintf(sql, "SELECT id,message FROM %s;", table_name);
    if (sqlite3_prepare_v2(db, sql, -1, &stmt, NULL) == SQLITE_OK)
    {
        while (sqlite3_step(stmt) == SQLITE_ROW)
        {
            sprintf(message, "%s", sqlite3_column_text(stmt, 1));
            int id = sqlite3_column_int(stmt, 0);
            printf("User: %d Sending message: %s\n", id, message);
            sendMsg(client, message); // Send messages
            if (write(client, &id, sizeof(int)) == -1)
                handle_error("[server]Error write().\n");
        }
    }
    sqlite3_finalize(stmt);
    sqlite3_close(db);

    if (write(client, &finish, sizeof(int)) == -1)
        handle_error("[server]Error write().\n");
}

```

Fig. 17. Server-chat

5 Conclusions

This application can be very useful for people passionate about computer networks, because it offers a decent amount of information about the subject and it gives you the option to discuss about the subject with other people interested in this topic. An upgrade that can be done to this application would be the possibility to up vote useful messages or user profiles to show how involved they are in the community or how trusted they can be when asking for an advice or help in global chats.

References

1. Kreibich, J.: Using SQLite. ” O’Reilly Media, Inc.” (2010)
2. Kurose, J.F., Ross, K.W.: Computer networking. A Top-Down (1986), <https://bit.ly/31oJF1q>
3. Lenuta, A., Andrei, P.: Computer networks (2021), <https://profs.info.uaic.ro/~computernetworks/>, [Online; accessed 1-December-2021]
4. Nord, H., Chambe-Eng, E.: Embedded software development tools — cross platform ide — qt creator (2021), <https://www.qt.io/product/development-tools>, [Online; accessed 1-December-2021]