VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER NETWORK LAB (CO3094)

Assignment 1

P2P CHAT APPLICATION

Advisor: Nguyễn Phương Duy
Students: Trần Công Huy Hoàng - 2052482.
Hoàng Đình Huy - 2053034
Nguyễn Đức Danh - 2052904

HO CHI MINH CITY, DECEMBER 2022

Contents

1 Introduction

1.1 Background

This chat application is a social-networking tool that enables communicating and sharing information between its users by taking advantage of technological advances. The clients are still centralized and managed, but it allows functionality like file sharing and conversation between two users without the need for a server like traditional chat programs.

This chat program aims to provide users more control over their devices and communications as well as increased privacy.

Developers need a firm grasp of the TCP protocol, P2P architecture, Client-Server architecture,... in order to construct this chat application.

Since the goal is to lessen reliance on servers but still maintaining central management of users, developers must implement a hybrid P2P and Client-Server architecture. In this architecture, the server only manages the accounts and addresses of users. The communication and sharing happens between two users directly by connecting their accounts through the TCP protocol.

Why is it necessary to reduce reliance on servers? The fact that users' conversations and files are kept in centralized databases increases the likelihood that their information may be exposed online and misused by malicious people. On the other hand, the users' concern for privacy is very high, since their information is stored somewhere else where the database administrator may have access to it.

The hybrid architecture is not a groundbreaking technology, and it is not a widely used solution in this field. However, despite some drawbacks, users could greatly benefit from this solution.

1.2 Objectives

- Allow users to communicate directly through TCP protocol.
- Allow users to share files using the TCP protocol directly.
- Only accounts, friends, and friend requests are managed by server.
- Only accounts and friends are stored in the database.

Comparing the app to conventional chat applications, more privacy and security should be improved. The messages and files sent and received by users only exist between them; they are not stored in the database.

The work that were conducted for this projects:

- Create file based database in SQLite3.
- Create database connection, server, client and user interface by using python programming language.

2 Methodology

2.1 TCP Protocols

Transmission Control Protocol (TCP) is a transport protocol that is used on top of IP to ensure reliable transmission of packets.

The TCP service model includes a connection-oriented service and a reliable data transfer service. When an application invokes TCP as its transport protocol, the application receives both of these services from TCP.

- Connection-oriented service. TCP has the client and server exchange transport-layer control information with each other before the application-level messages begin to flow. This so-called handshaking procedure alerts the client and server, allowing them to prepare for an onslaught of packets. After the handshaking phase, a TCP connection is said to exist between the sockets of the two processes. The connection is a full-duplex connection in that the two processes can send messages to each other over the connection at the same time. When the application finishes sending messages, it must tear down the connection.
- Reliable data transfer service. The communicating processes can rely on TCP to deliver all data sent without error and in the proper order. When one side of the application passes a stream of bytes into a socket, it can count on TCP to deliver the same stream of bytes to the receiving socket, with no missing or duplicate bytes.

2.2 Client-Server Paradigm

In a client-server architecture, there is an always-on host, called the ***server***, which services requests from many other hosts, called ***clients***.

With the client-server architecture, clients do not directly communicate with each other. Another characteristic of the client-server architecture is that the server has a fixed address, called an IP address. Because the server has a fixed address, and because the server is always on, a client can always contact the server by sending a packet to the server's IP address.

2.3 P2P Paradigm

In a P2P architecture, there is minimal (or no) reliance on dedicated servers. Instead the application exploits direct communication between pairs of intermittently connected hosts, called ***peers***.

The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices.

2.4 Hybrid Architecture Between P2P & Client-Server Paradigm

Hybrid Architecture are networks that are based on both P2P and Client-Server Paradigm. Hybrid networks incorporate the best features of workgroups in P2P networks with the performance, security and reliability of server-based networks. Hybrid networks still provide all of the centralized services of servers, but they also allow users to share and manage their own resources within the workgroup.

Advantages:

- Client Server application are still centrally located and managed.
- Users can assign local access to resources in their computers.
- Workgroups can manage resources without requiring assistance from network administrator.

Disadvantags:

- Users may need to remember multiple passwords.
- Files can be duplicated and changes overwritten between the computers with the shared folder and the Server.
- Files saved on the workstation are not backed up.

3 Project Activities

With all the resources, we implement all of the features in python

3.1 Database

There is a file user.sql will create two tables account and friend after execution and insert into each table the default value for the database.

	userid	password	email
	Filter	Filter	Filter
1	huyhoang	123456	huy@gmail.com
2	danh	123789	danh@gmail.com
3	huy	123	huy@gmail.com

	userid	friendid
	Filter	Filter
1	huyhoang	danh
2	danh	huyhoang

Table **account** has *userid*, *password*, *email* and all its type are text, this table represents the user's account information in the database.

Table **friend** has *userid*, *friendid* and all its type are text, this table respresents the friend list of the users in the database.

There is also a db.py file which make a connection to the database and execute query that is request by the server.

In the db.py file we use sqlite3 library, sqlite is a light weight library that store data in only one *.db file, to connect to the sqlite database.

To connect to the database, just call sqlite3.connect('*.db') where *.db is the database, in case the database not exist then the command will create the database and save it as the name of the database we want to connect to.

After connection success then it return a connection object which you can use to execute query in the database.

3.2 Server

Communication from server to client is by sending message that use the same format on client. The message use 'utf-8' encoding then get encrypted by 'rsa'

encryption and only then, the server will send the encrypted message to the clients.

Server is multi-thread, which meant many clients can connect to the server at the same time. After a client connects to the server, the server will make a thread for that client.

After the thread started, client have 3 authentication options by sending to the server different messages:

- Login: Client sends messages with format "LOGIN userid:password".
- Register: Client sends messages with format "REGISTER userid:email:password".
- Forgot password: Client sends messages with format "FORGOTPASS userid:email:newpassword".

In case login, server will send back to client message "FAIL" if the user input the wrong login information, else it will send back to client message "SUCCESS".

In case register, server will send back to client message "FAIL_USERID" if the user input the userid that exist in the database or "FAIL_EMAIL" if the user input the email that exist in the database. Else the server will send to client message "SUCCESS".

In case forgot password, server will send back to client message "FAIL" if the user input the wrong email or userid information, else it will send back to client message "SUCCESS" and update the user password in the database.

After the authentication step, client is added to the online list, and server will collect the client's friend address if they are online and send message to client list of, list of ip, list of port of friends online.

While connecting to the server, client have different options:

- Unfriend: Client sends messages with format "UNFRIEND userid".
- Find: Client sends messages with format "FIND userid".
- Request: Client sends messages with format "REQUEST userid".
- Accept: Client sends messages with format "ACCPET_FRIEND userid".
- Refuse: Client sends messages with format "REFUSE_FRIEND userid".
- Request Time Out: Client sends messages with format "REQUEST_TIME_OUT".

In case unfriend, server will delete rows in friend table represent the friend list.

In case find, server will send back to client message "FOUND_ONLINE" if the userid is in the online list of the server, else server will send back to client message "FOUND_OFFLINE".

In case request, server will send the the client have the id requested message "FRIEND_REQUEST userid", userid here is the id of the client who requested.

In case accept, server will send the client message "FRIEND_LIST_UPDATE" and an updated list of information of friends and insert new friend into the database.

In case refuse, server will send the client message "REQUEST_DENIED" and "userid".

In case request, server will send the client message "REQUEST_TIMEOUT" and "userid".

If the user logout, the server disconnect the connection and then update the online list

3.3 Client

Communication from client to server is by sending message that use the same format on server. The message use 'utf-8' encoding then get encrypted by 'rsa' encryption and only then, the server will send the encrypted message to the clients.

Server is multi-thread, which meant client can chat with other clients and connect to the server at the same time. After a client logout, if friend request are not empty, client sends back to server message "FRIEND_REQUEST_TIMEOUT friend_requests".

If authentication return "SUCCESS" then clients are prompted to the main screen, else if the message is "FAIL" or "FAIL_USERID" or "FAIL_EMAIL" the program will prompt message according to the response.

If client receive "FRIEND_LIST_UPDATE", the program will update the friend list which is being display.

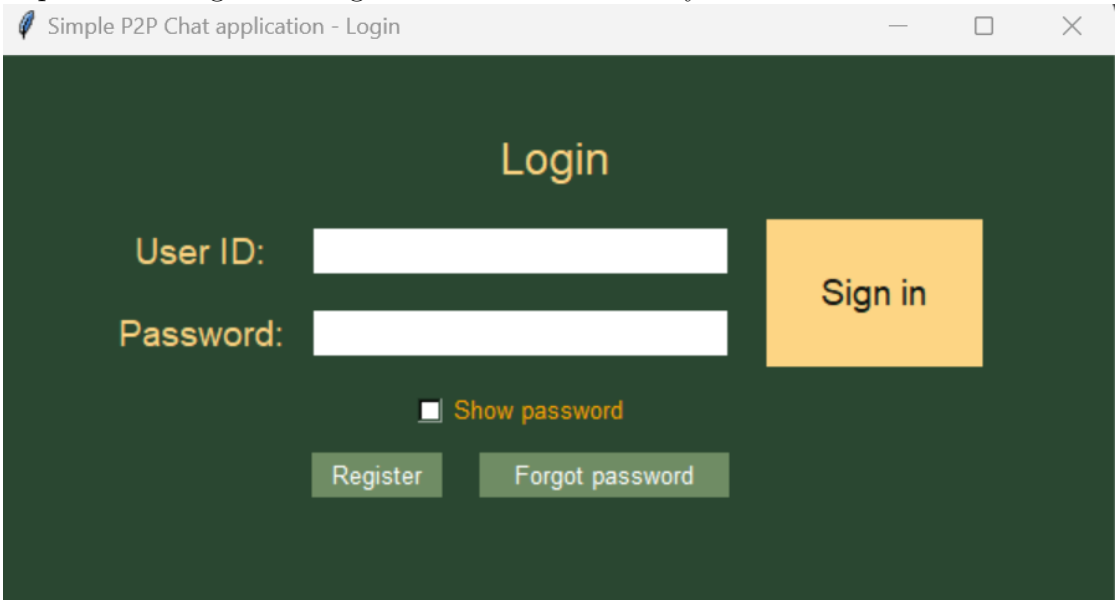
As for the rest of the service, they are already described on the server part. All else messages return from the server will prompted the warning according to the server.

If client wants to chat to friends, select a green name on the display list and start the chat. The connection is directly between client, no need for server.

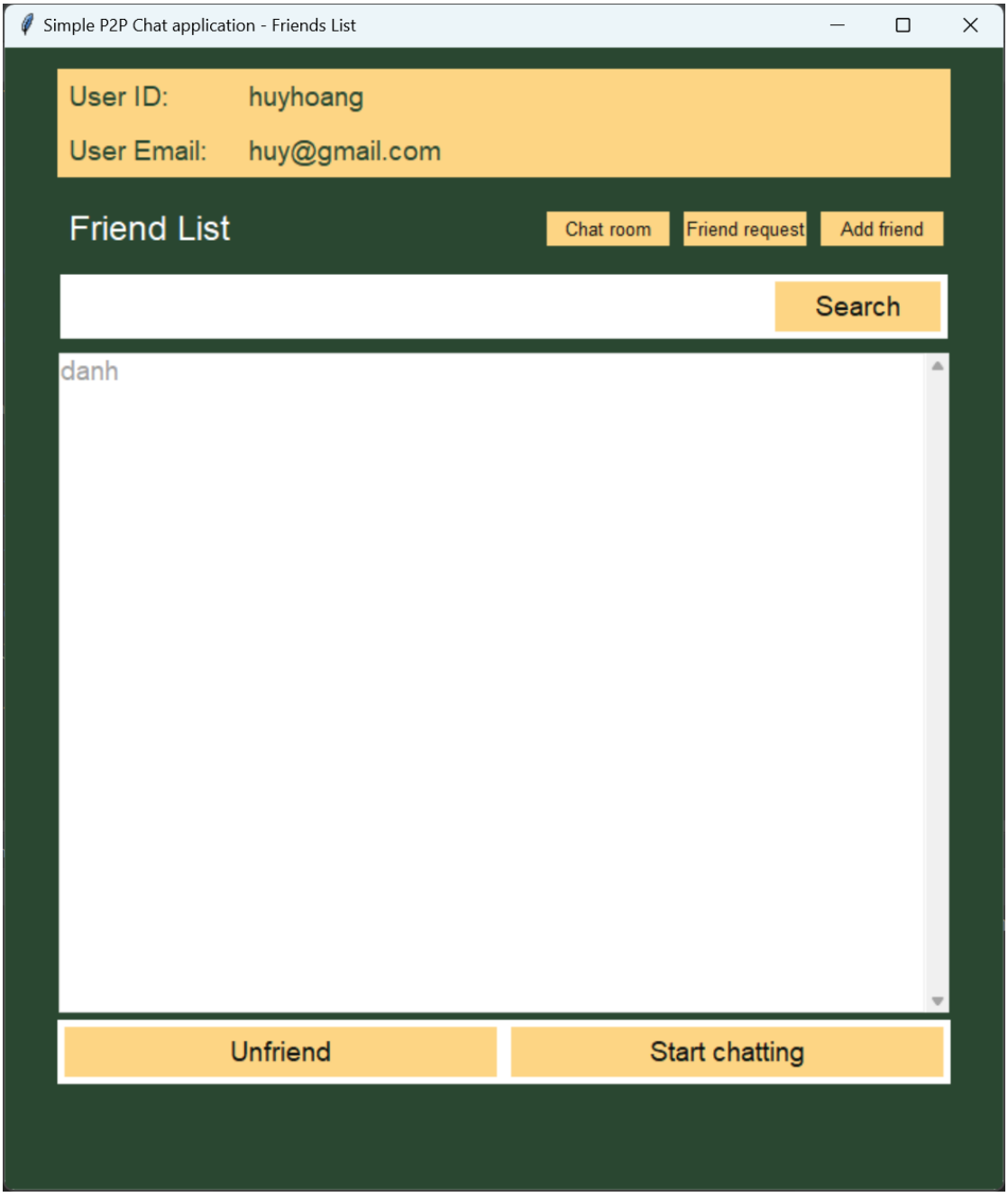
The connection between clients is TCP.

4 Project Outputs

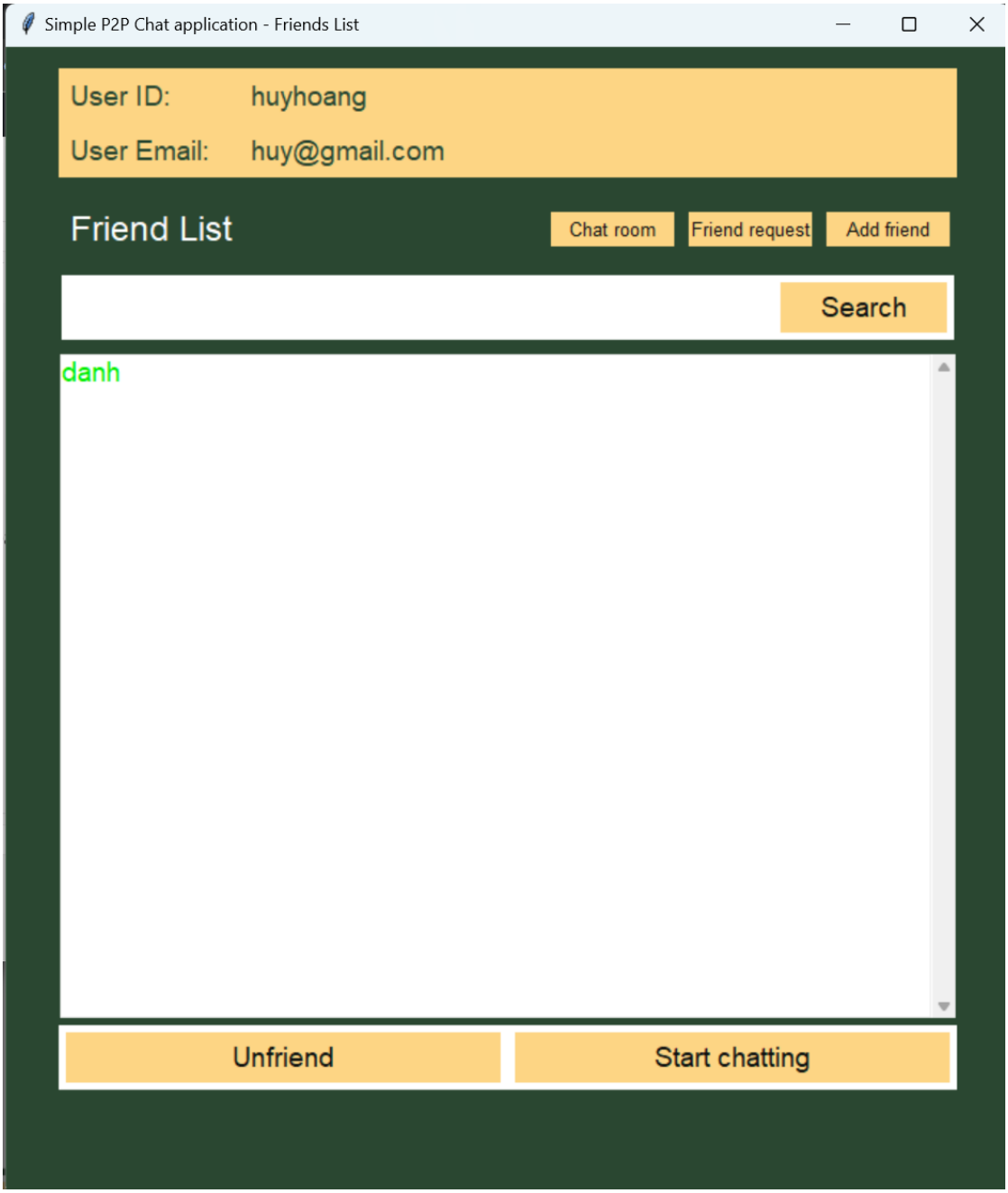
First, when we open the app, a login screen will be displayed and you will be required to log in using a valid account or you can create new account.



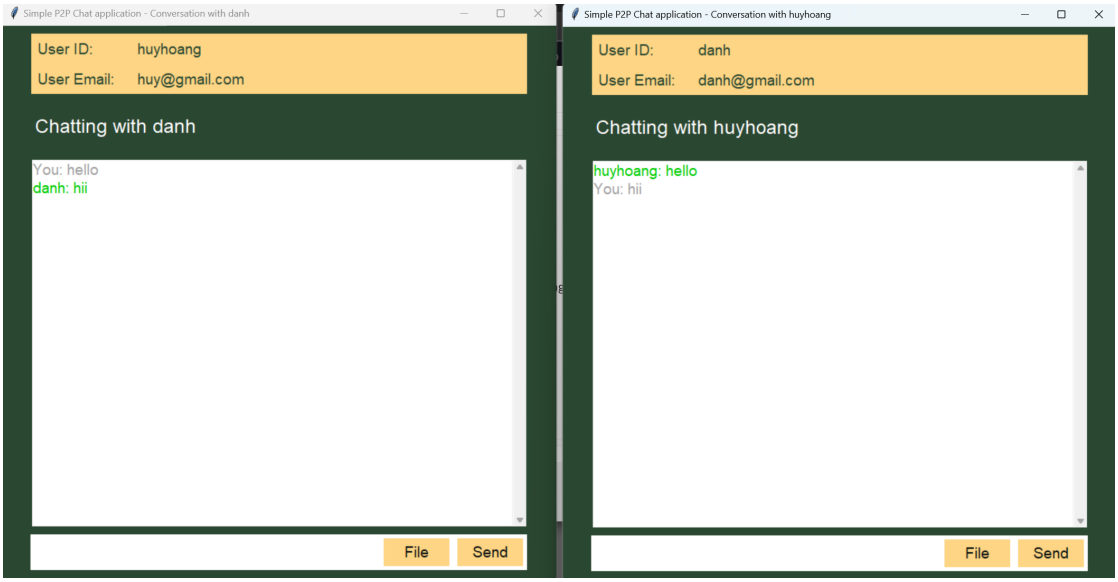
After log in using a valid account, a friend list window will alter the login window.



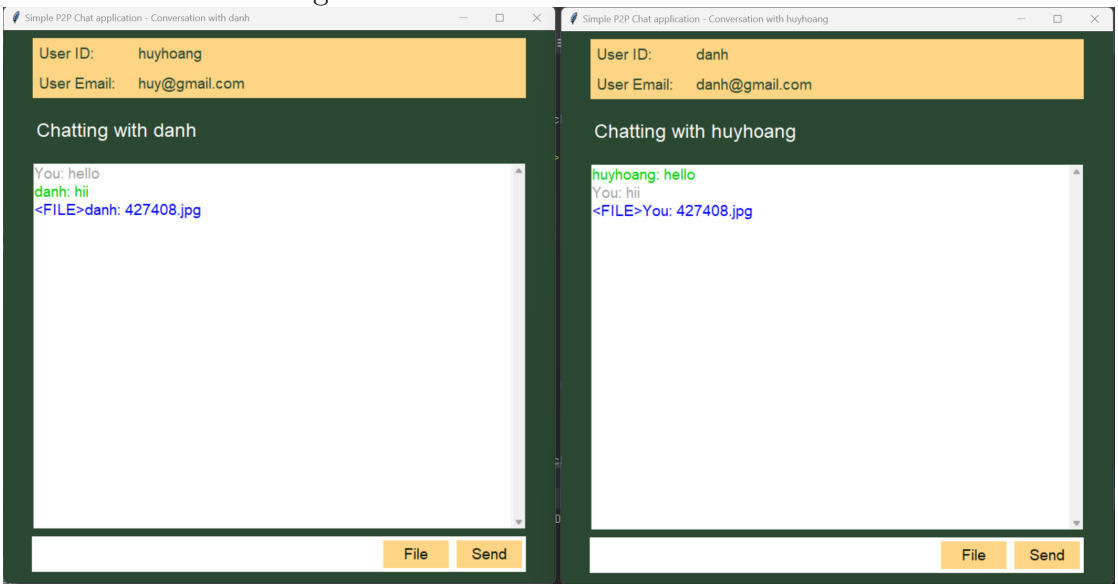
If your friends are not online, their name will be shown in grey. Otherwise, it will be displayed in green.



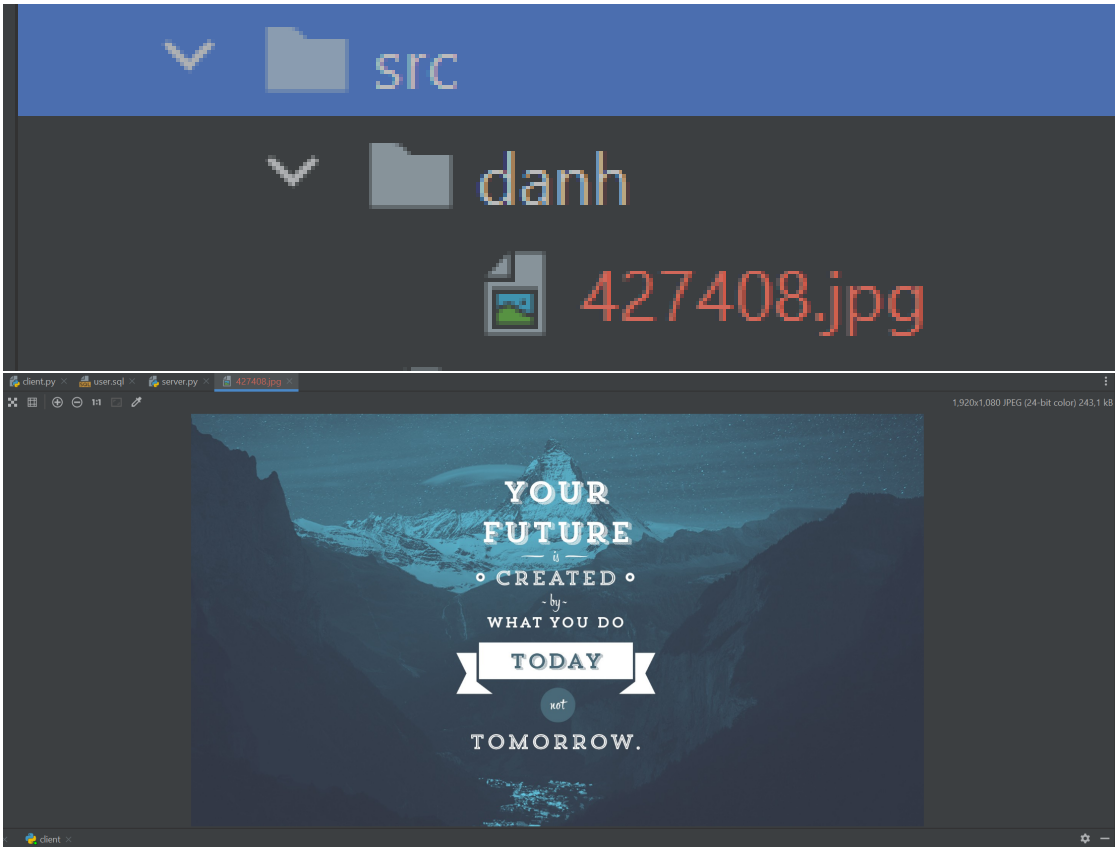
Our app has two main function, sending text messages and sending any sort of files.



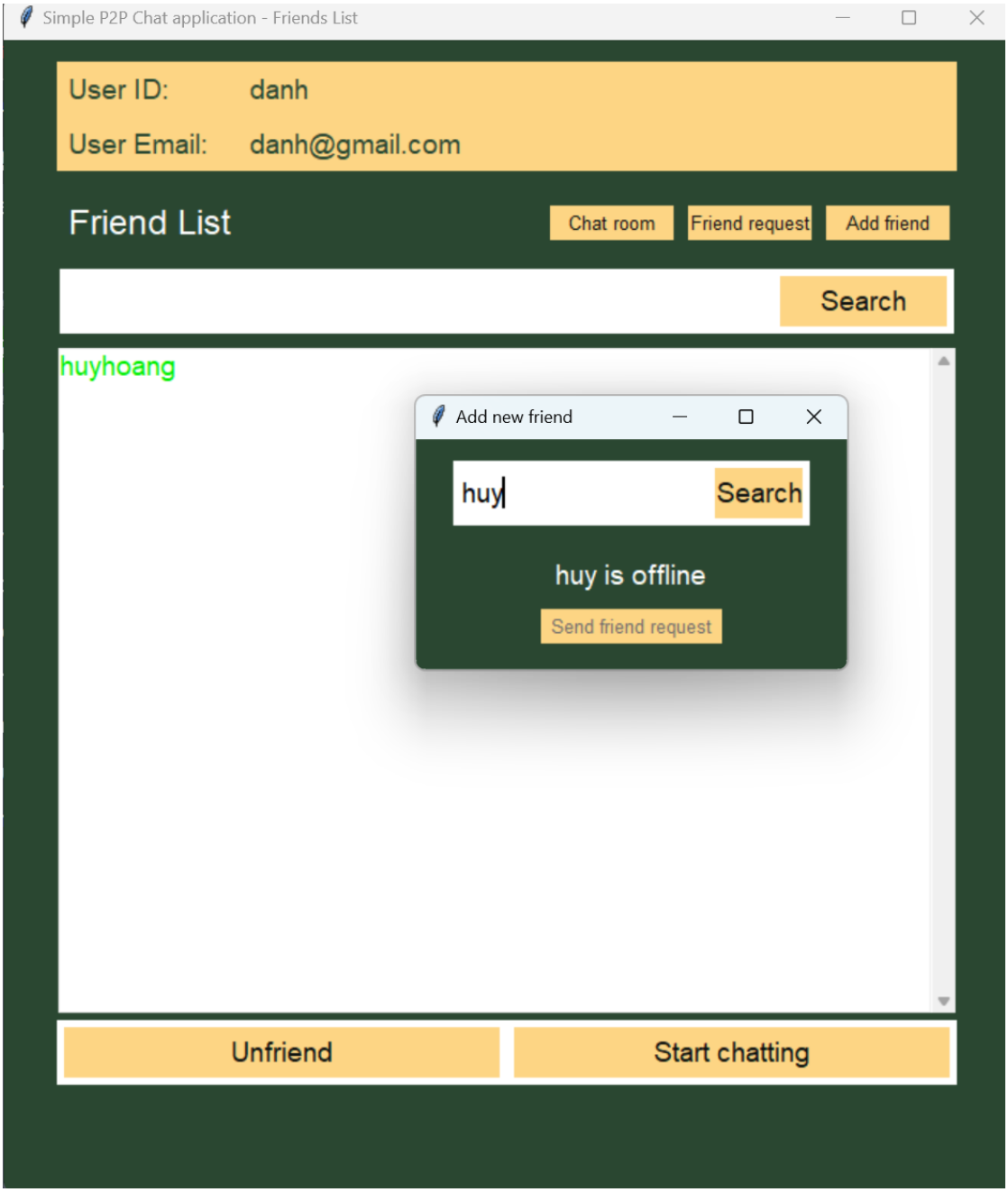
Since our user connect directly to each other using P2P TCP protocol, our users can continue chatting with each other even when the server is down.



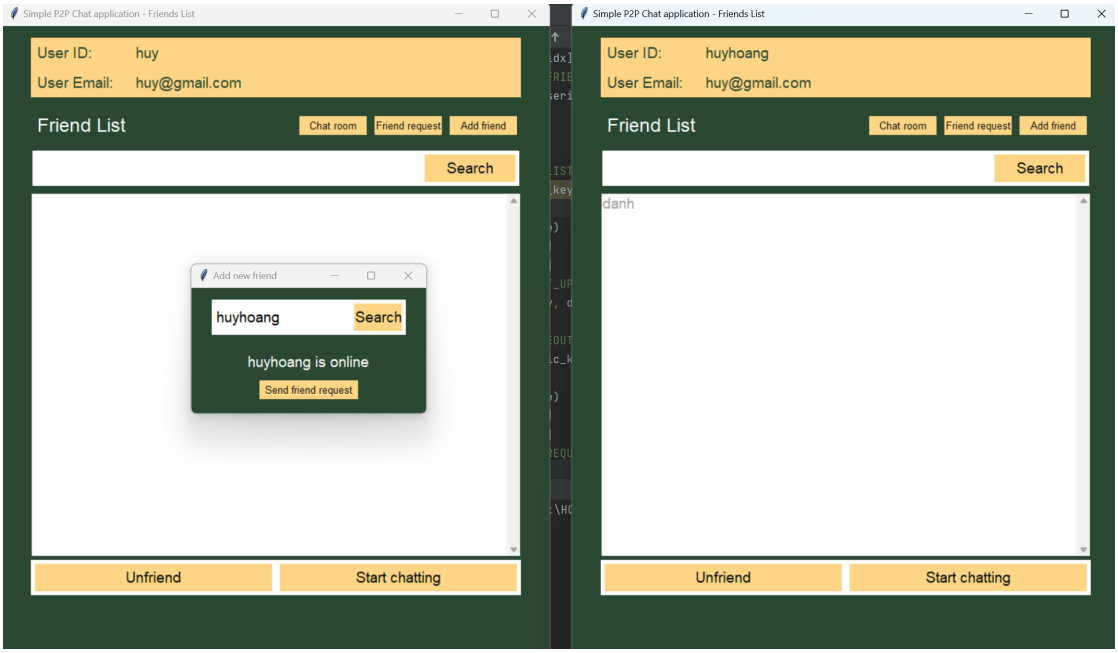
When a file is sent, a line with format "<File>"user_id": 'filename'" will be displayed into the list box. The file is saved locally in the user's device, within the folder with the same name with the sender.



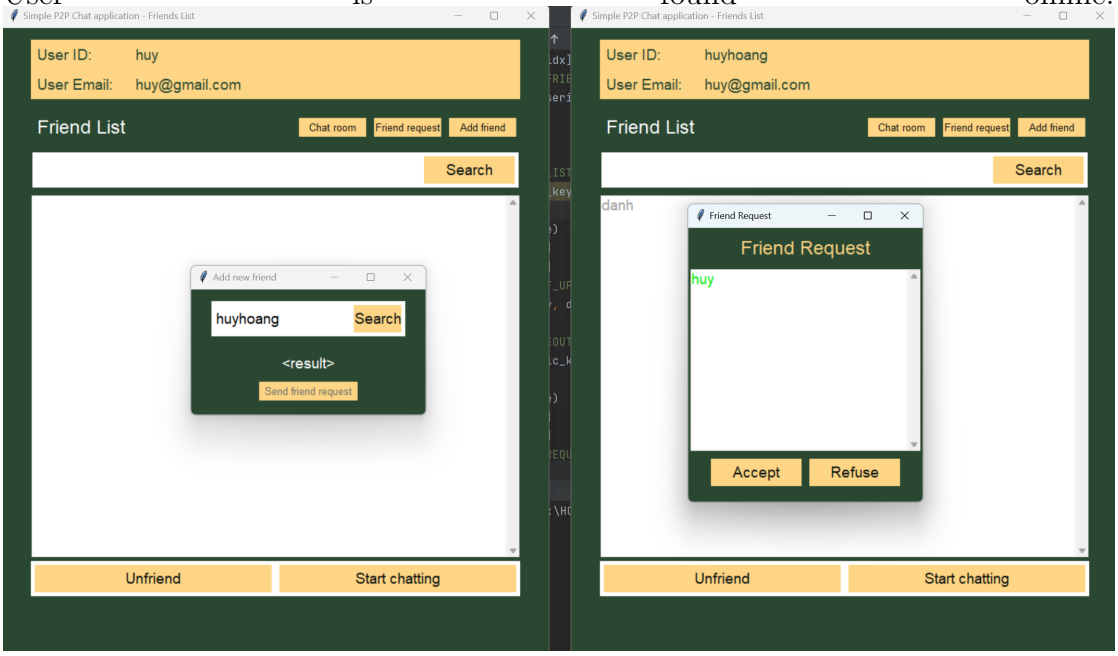
We can also add new friends to our account by searching the user_id.



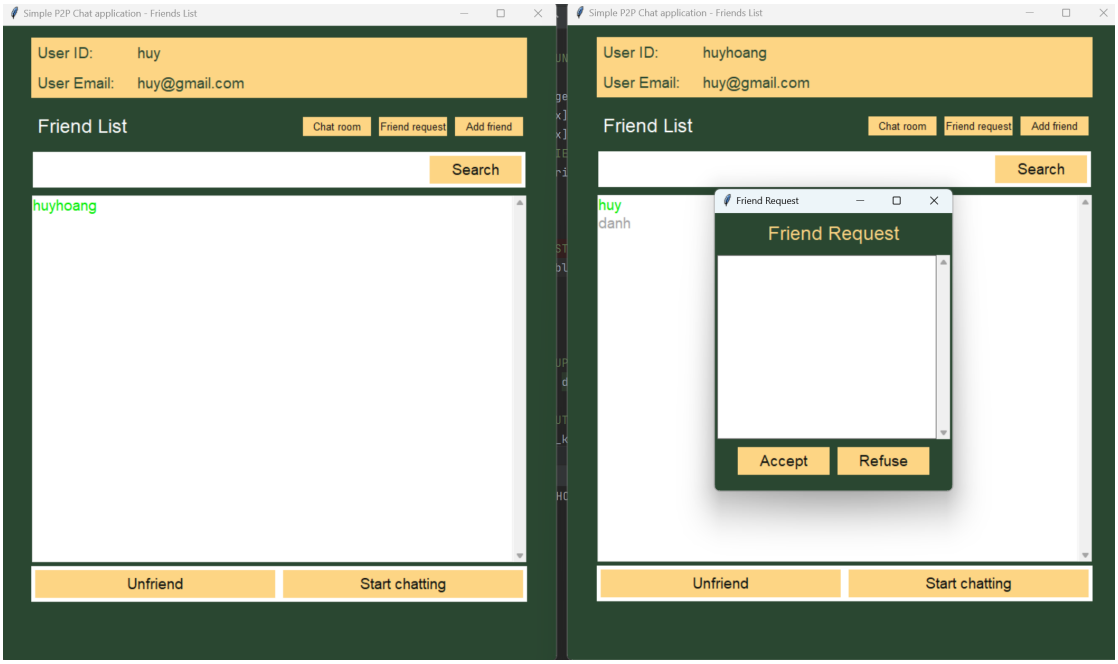
User is found but offline.



User is found online.



Whenever a user send a friend request to another user, a friend request will be added into the friend request list. And when a user accept a friend request, their user_id will be automatically add into friend list.


Simple P2P Chat application - Friends List
User ID: huy
User Email: huy@gmail.com
Friend List
Chat room
Friend request
Add friend
Search
huyhoang
Unfriend
Start chatting
Simple P2P Chat application - Friends List
User ID: huyhoang
User Email: huy@gmail.com
Friend List
Chat room
Friend request
Add friend
Search
huy
danh
Friend Request
Friend Request
Accept
Refuse
Unfriend
Start chatting

5 Project Outcomes

The project is currently relatively rudimentary, therefore it doesn't significantly advance knowledge, research, or science.

The initiative improved the team's ability to work together, communicate among developers, and operate more efficiently.

Though the project is basic, it promotes a different solution to a wider range of audiences and has a significant impact on the adoption and adaptation of technology

The project has taught us that we should break down and comprehend the project better before beginning it, start as soon as possible to give ourselves more time to evaluate and repair bugs, and we may have time to add more features.

6 Overall Assessment

After achieving the projects objectives, now users' can be more comfortable with their conversations and sharing information using this chat application.

Although not novel or ground-breaking, this effort can serve as a guide for those beginning out in this subject in the future. As a result, this project serves as both an introduction for newcomers and a minor addition to learning resources.

This project is merely a modest foundation upon which many further functions, such voice mail, video sharing, chat rooms, video calls,... can be added. It is recommended that these functionalities be included in the app.

In terms of publications, we have no plans to make this project public as is.