Computer Networks - Programming Assignment

Part 1: Client-side Program Operation Explanation Document

1. Source Code is located at `ARCHIVE_ROOT/src/`
2. 操作說明文件 PDF 檔 is the document you are reading right now
3. Binary 執行檔 is located at `ARCHIVE_ROOT/build/`
4. 用以編譯程式之 Makefile is located at `ARCHIVE_ROOT/Makefile`

1. compilation

Prerequisite:

`make`

Run the following command if `make` is not installed on your system.

```
sudo apt update && sudo apt install make
```

Install dependencies:

`gcc`, `build-essential`, `libgtkmm-3.0-dev`

Run the following command in the same directory as the `Makefile` (source code root) to install the required packages.
You may need to install the `libgtkmm-3.0-dev` package regardless of whether you are building the application yourself or are planning to run the precompiled binaries.

```
make install-deps
```

Compilation:

Run the following command in the same directory as the `Makefile` (source code root) to compile the client-side program to `build/client`.
The application is written in C++ for the Linux platform, using the `gtkmm` library.

```
make client
```

2. execution

Run `build/client` or

```
make run
```

to run the client GUI application on Linux.

3. program execution environment

This program currently only runs on Linux with GUI. The `libgtkmm-3.0-dev` package may be needed during runtime.
There is no option to build or run the application headless.

4. references and sources

- The socket programming part of this application (included in the `src/mySocket.h` file) is heavily inspired by the awesome book `Beej's Guide to Network Programming - Using Internet Sockets` by Brian "Beej Jorgensen" Hall, completely free to read at A4 direct link *<https://beej.us/guide/bgnet/pdf/bgnet_a4_c_2.pdf>* or the home page *<https://beej.us/guide/bgnet/>*.
- Some code may be AI generated by `GitHub Copilot`, but all the code are understood and reviewed by me.

5. project structure

You may skip reading the remainder if you'd like. Below briefly describes how the program is written, as well as the graphical user interface screenshots and error handling demonstrations, which is not mentioned in the submission requirements to be included in the operation explanation document.

All source files are included in the `src` folder.

The entry point `main` for the client program is located at `client.cpp`. This file contains the code to run the `loginWindow` and `mainWindow`, including showing the `mainWindow` after login, returning to the `loginWindow` if the `Log out` button is pressed on the `mainWindow`, or exit the program if the `loginWindow` or `mainWindow` is closed.

The custom class for basic socket interfacing is located in `mySocket.h`. This file uses the Unix/Linux Socket Programming libraries, and include functions to interface with a socket in a simpler manner. All functions include extensive error handling, debug information, and timeouts for connecting and receiving.
This class will be shared between the client and the server in the future.

```
// checks if a string port number is valid, and converts it to an integer. isServer = true
allows ports less than 1024
int MySocket::checkPort(const std::string &port, bool isServer = false);
// connect to the given hostname/IP address and port using TCP
bool MySocket::connect(const std::string &hostname, const std::string &serverPort, int timeout
= 5);
// bind the socket to the given port on the system
bool MySocket::bindSocket(const std::string &clientPort);
// listen for incoming TCP connections
bool MySocket::listen(int timeout_sec = 5);
// accept the incoming connection with the given socket
void MySocket::accept(MySocket &newSock);
// randomly find an available port on the system
int MySocket::findAvailablePort();
// send a message with the socket
bool MySocket::send(const std::string &message);
// receive a message from the socket
std::string MySocket::recv(int timeout_sec = 5);
// close the active TCP connection. This is also called when the MySocket object is destroyed
void MySocket::closeConnection();
```

All the logic required to run the client is located in `clientAction.h`. A global object `ClientAction clientAction` manages the state for the client program, and provides interfaces for performing client actions.

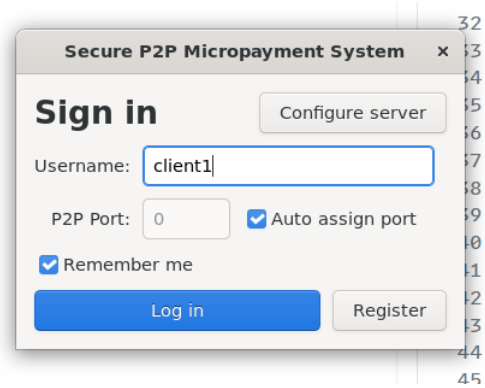
```
// establish a TCP connection to the server
bool ClientAction::connectToServer(const std::string &hostname, const std::string
&serverPort);
// sends "REGISTER#<USERNAME>" to register an account from the server
bool ClientAction::registerAccount(const std::string &username);
// sends "<USERNAME>#<P2P_PORT>" to the server to log in
bool ClientAction::logIn(const std::string &username, const std::string &p2pPort);
// parses the results from ClientAction::fetchServerInfo() and save the results to
std::vector<UserAccount> ClientAction::userAccounts
bool ClientAction::parseOnlineUsers(const std::string &response);
```

```
// sends "List" to the server to obtain the online users and the user information, like the balance
bool ClientAction::fetchServerInfo();
// send a payment to a given user, by sending "<MY_USERNAME>#<AMOUNT>#<PAYEE_USERNAME>" to the payee directly using P2P
bool ClientAction::sendMicropaymentTransaction(int amount, const std::string &payeeUsername);
// wait for the "Transfer OK!" message from the server
bool ClientAction::verifyMicropaymentTransaction();
// send out "Exit" to the server and close the connection gracefully
void ClientAction::logOut();
// start the P2P listening port, by opening a separate thread and continuously listen for an incoming connection every second
void ClientAction::p2pStartListening();
// stop the P2P listening port, when logging out or quitting the app
void ClientAction::p2pStopListening();
// a callback function to accept the P2P payment message from a peer, and reemit to the server
void ClientAction::handleIncomingMessage(const std::string &message);
// stop all the connections gracefully before closing the app
void ClientAction::quitApp();
```

All the GTK window classes for the client GUI is in `client_windows/*.h`.

A brief look at the client windows and the general usage is shown below:

loginWindow



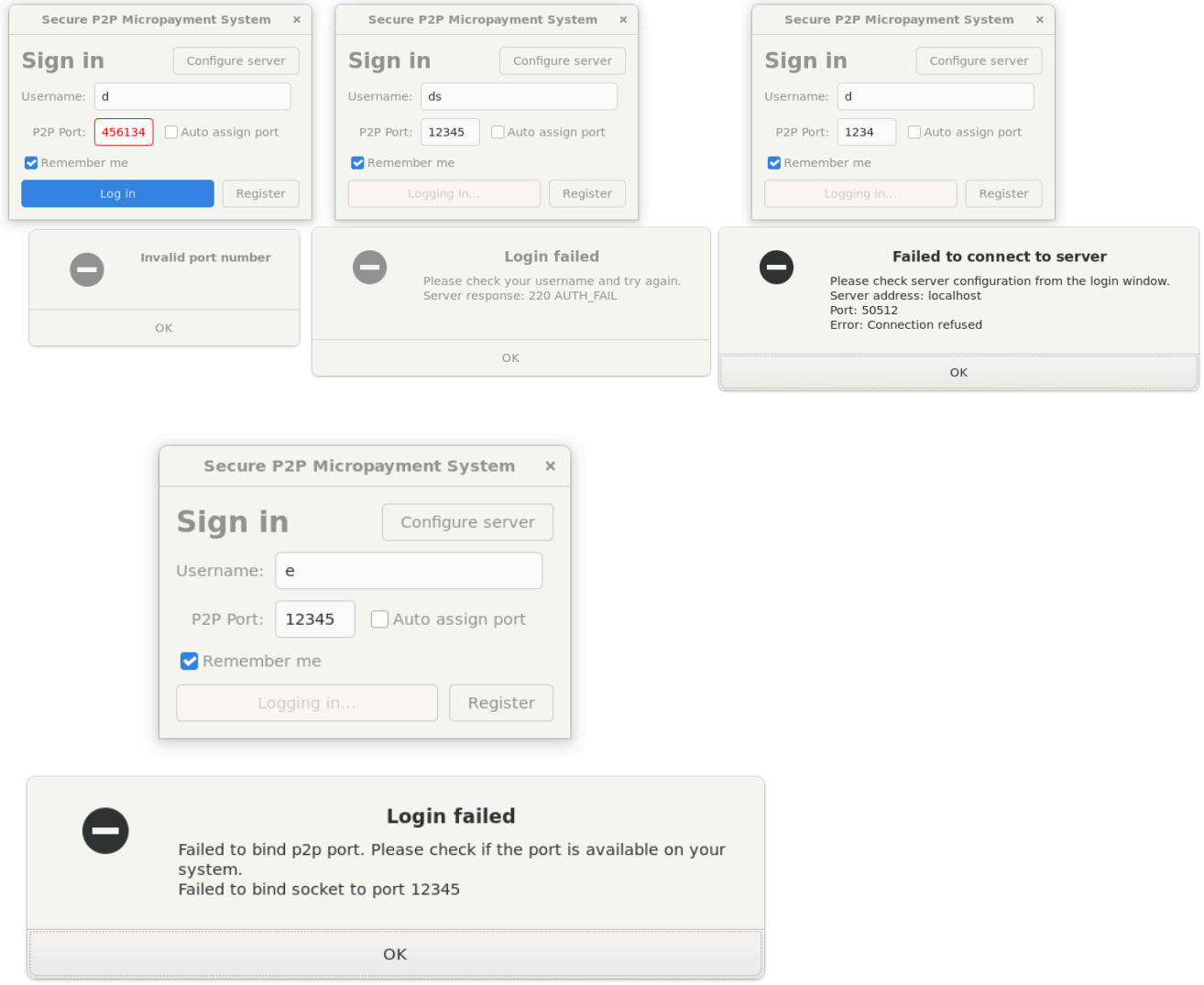
The user may choose to save the username and port settings for future logins. The server settings are saved automatically. The save file can be found in the working directory, and the code responsible for loading and saving the configuration file is in `clientConfig.h`.

client.conf

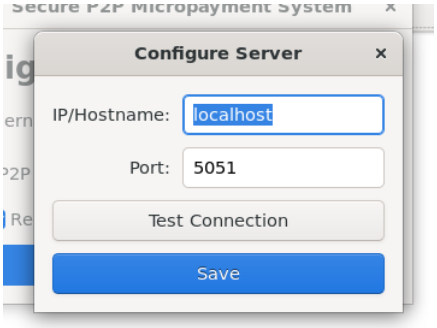
```
serverAddress=localhost
serverPort=5051
username=d
p2pPort=1234

```

Basic error handling are in place, to prevent the program from crashing unexpectedly, or provide information to the user.

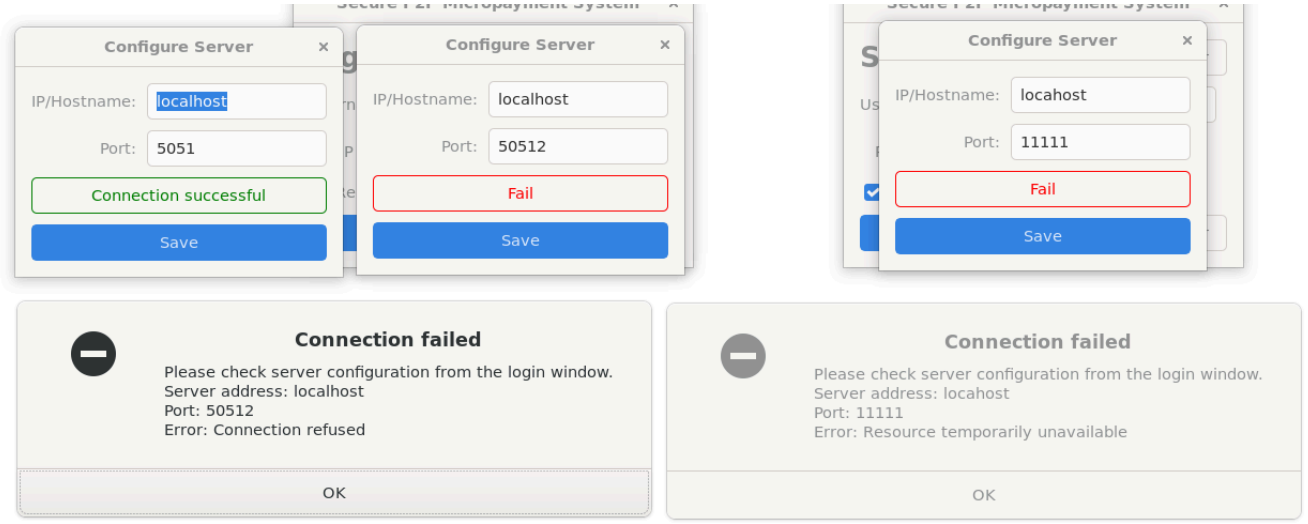


configureServerWindow

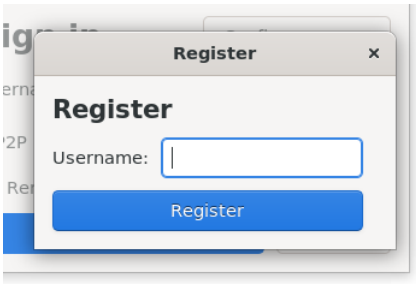


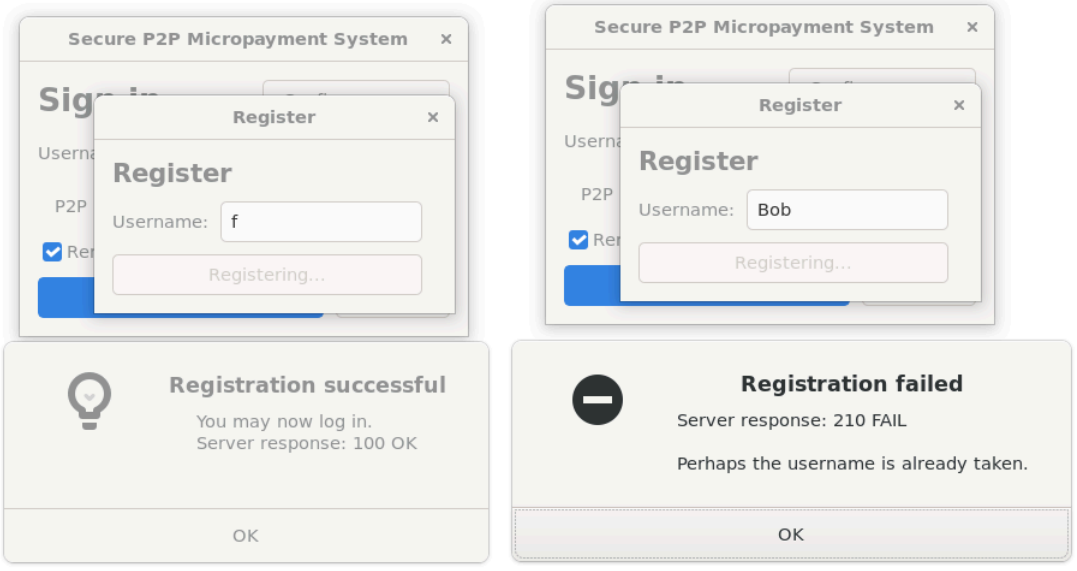
Test connection button is included to let the user check for their input before committing and saving the

configuration. Input field error handling are also in place, as well as connection timeouts.

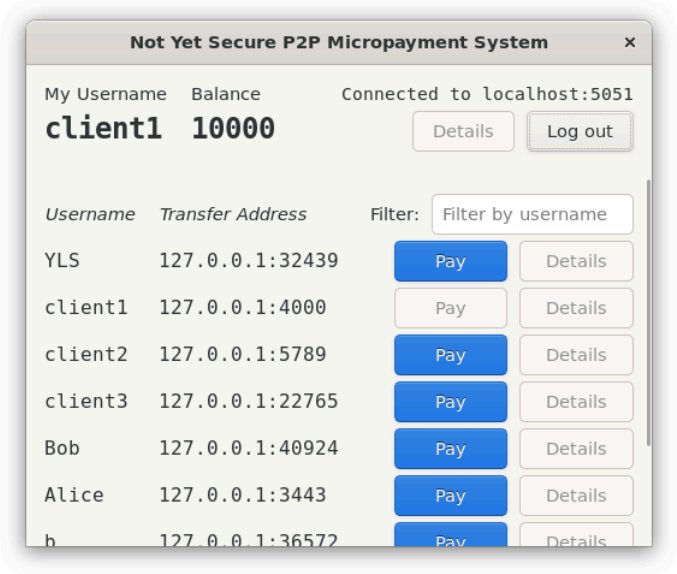


registerWindow

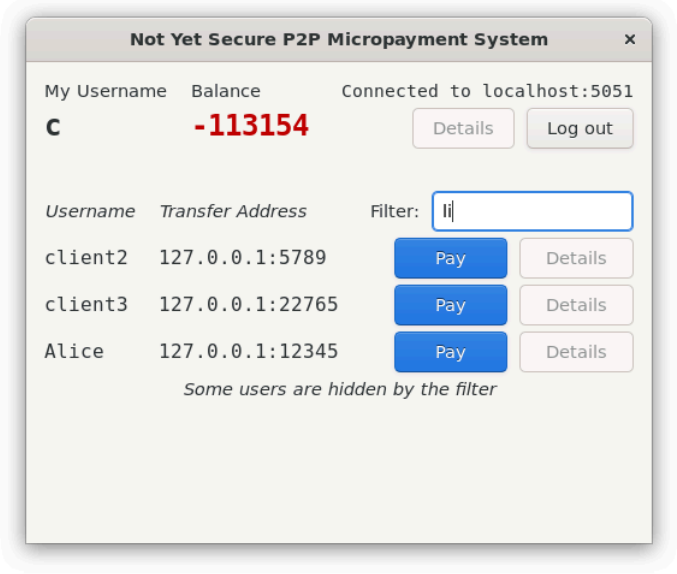




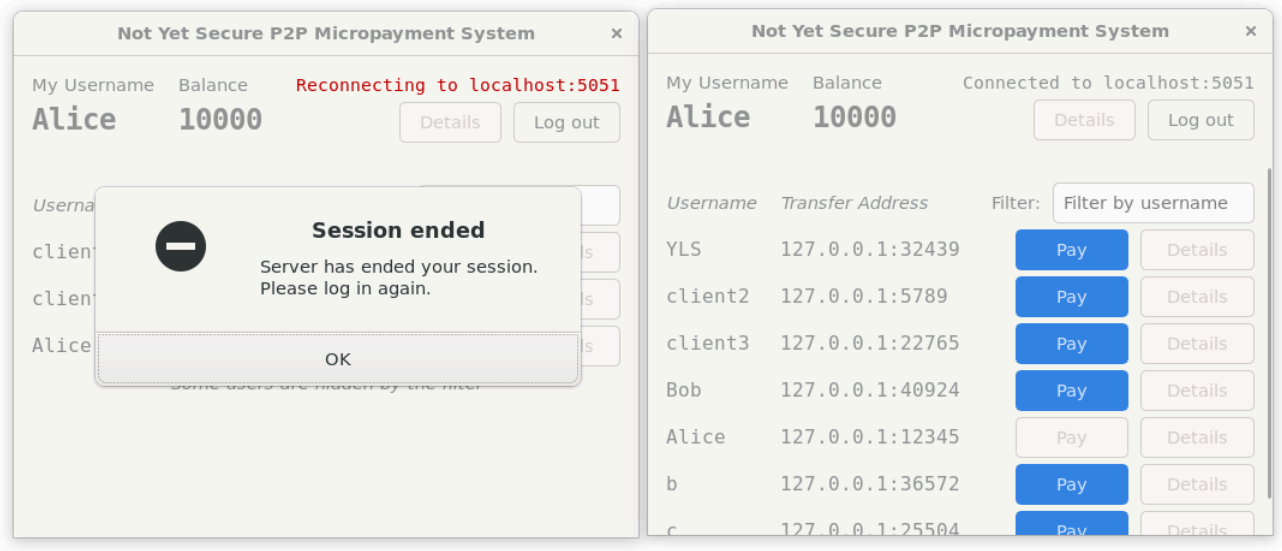
mainWindow



The filter feature is in place, so when there are too many users online, you can find the proper user quickly.

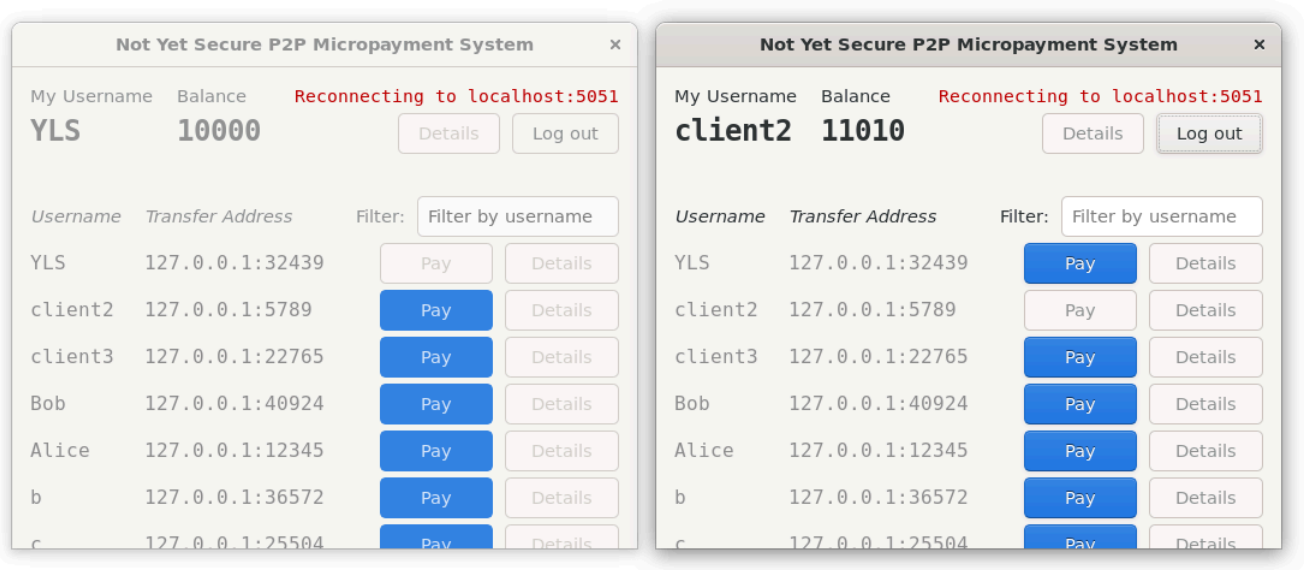


If the user attempts to log in from another window, the existing login session will be ended by the server. A popup dialog will show up, and redirects the user back to the login window.

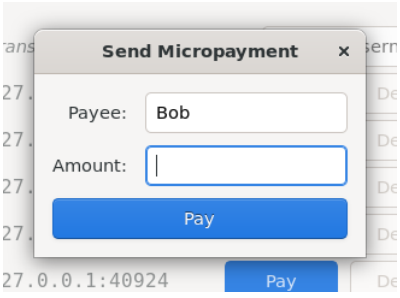


The user can also check for the server connection status with the server information text on the top right. In this

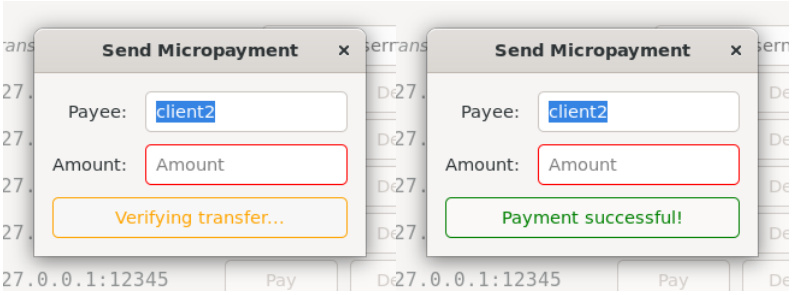
case, the server process is terminated, and the clients are then disconnected.



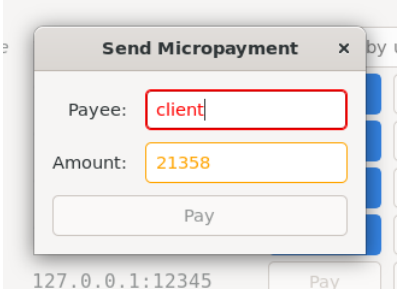
payWindow

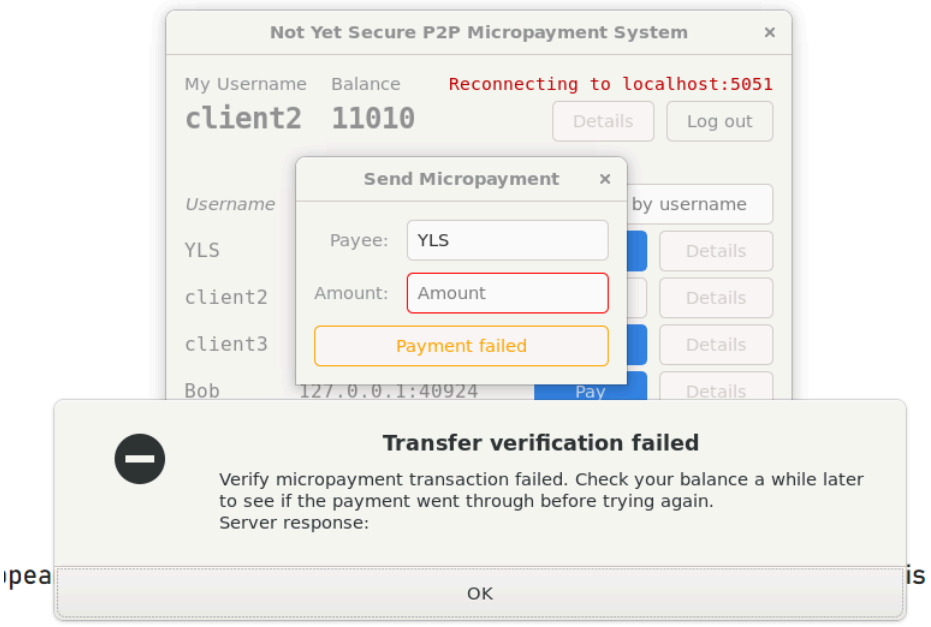


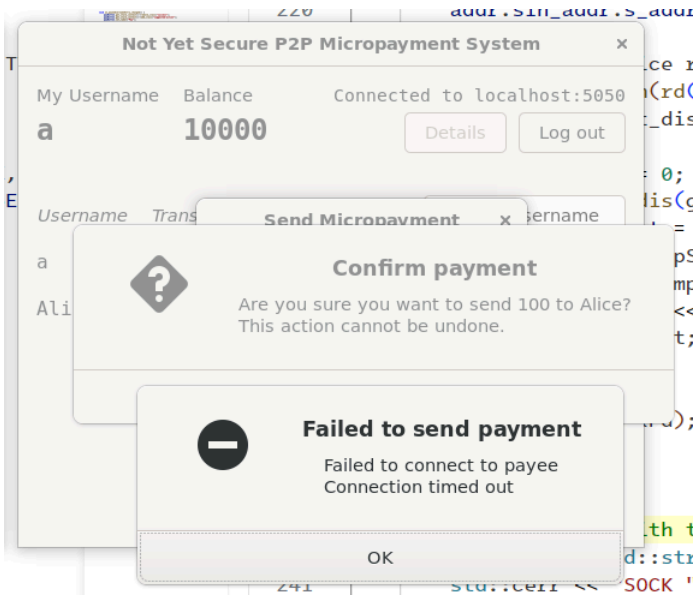
The application will check for the `Transfer OK!` message from the server, and indicates the transfer verification process to the user.



Basic error handling are in place as well, such as input errors and connection errors. The payee textbox can be entered manually, if the user knows precisely who to send the payment to, but appears red when the entered username is not online, and the `Pay` button is deactivated. The amount turns yellow when greater than the balance.




Not Yet Secure P2P Micropayment System
My Username
Balance
Reconnecting to localhost:5051
client2
11010
Details
Log out
Send Micropayment
Username
by username
Payee:
YLS
Amount:
Amount
Payment failed
YLS
client2
client3
Bob
127.0.0.1:40924
Pay
Transfer verification failed
Verify micropayment transaction failed. Check your balance a while later to see if the payment went through before trying again.
Server response:
OK


Not Yet Secure P2P Micropayment System
My Username
Balance
Connected to localhost:5050
a
10000
Details
Log out
Username
Send Micropayment
Confirm payment
Are you sure you want to send 100 to Alice?
This action cannot be undone.
Failed to send payment
Failed to connect to payee
Connection timed out
OK