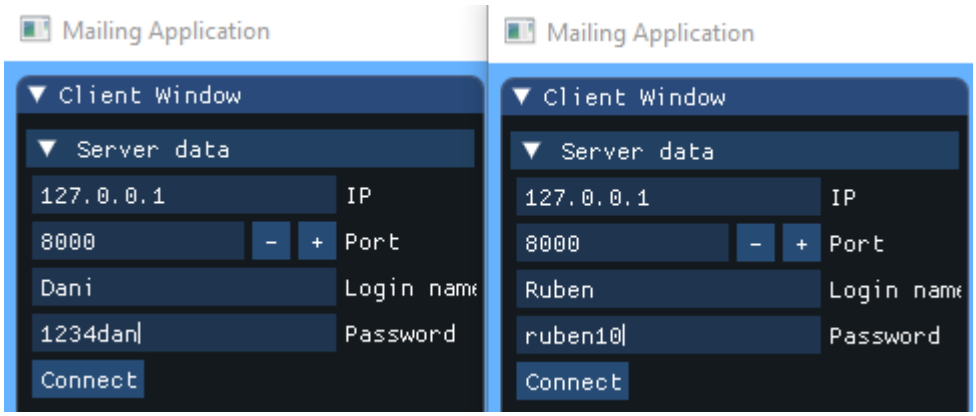
Práctica 1

Por Daniel Triviño y Rubén Sardón

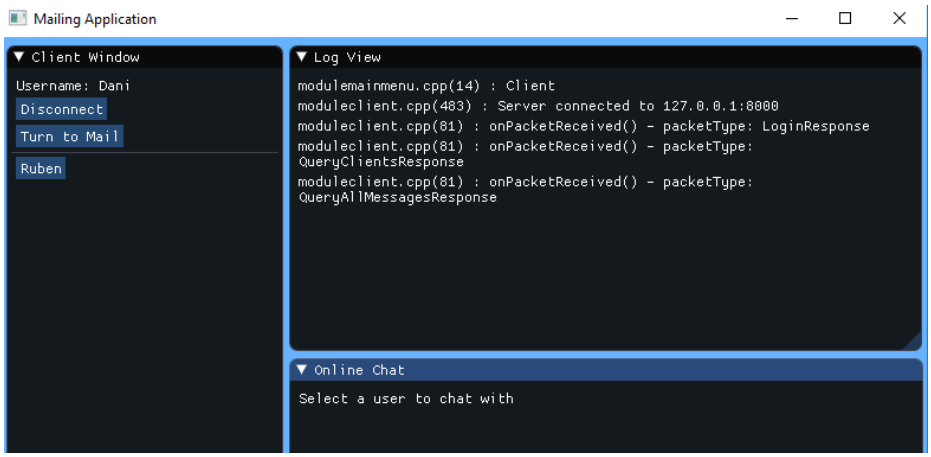
Contenido

Funcionalidades Extra Añadidas

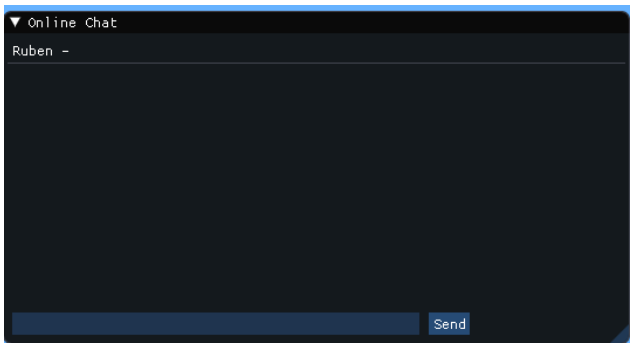
Hemos programado un sistema de chat online donde los usuarios se conectan a un servidor (simulado o usando SQL) usando contraseña. Se requiere contraseña para iniciar sesión y se manda la contraseña en hash usando md5 para no mandar la contraseña descubierta.



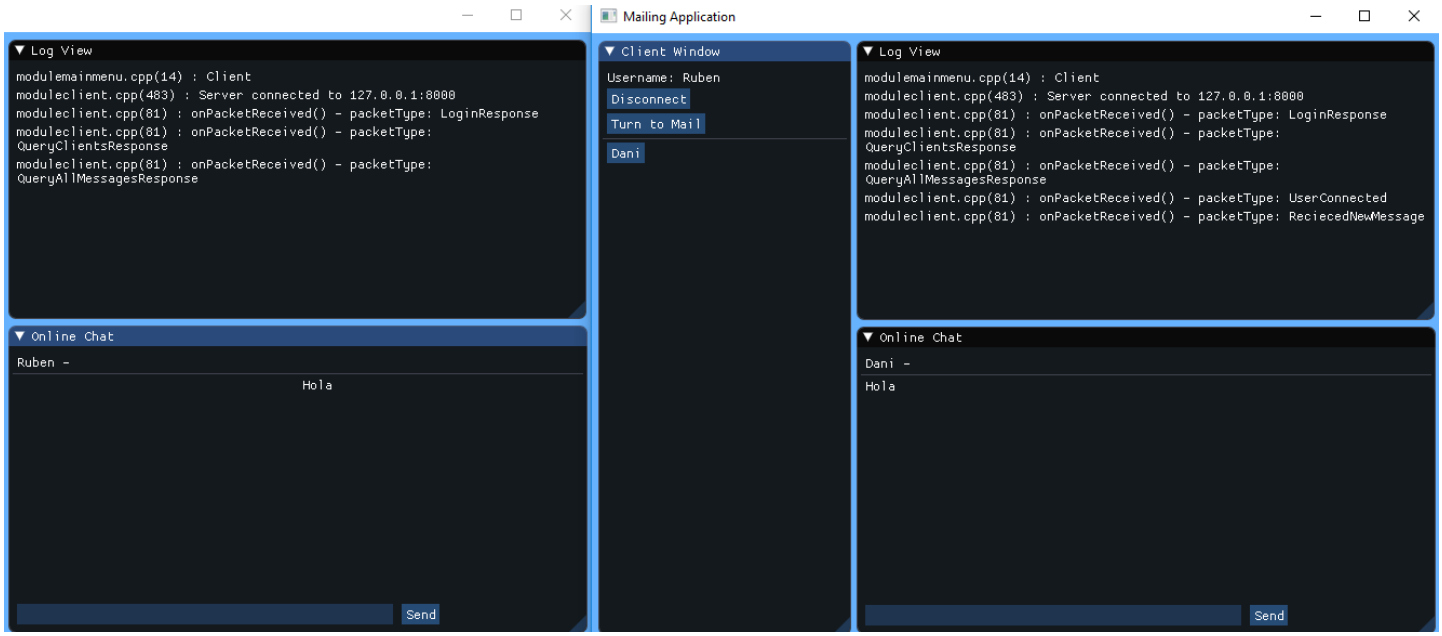
Si el nombre de usuario no ha sido registrado en la base de datos, el usuario se conecta al servidor y pasa a ver la vista de chat. Si hay usuarios conectados, se verán sus nombres en botones.

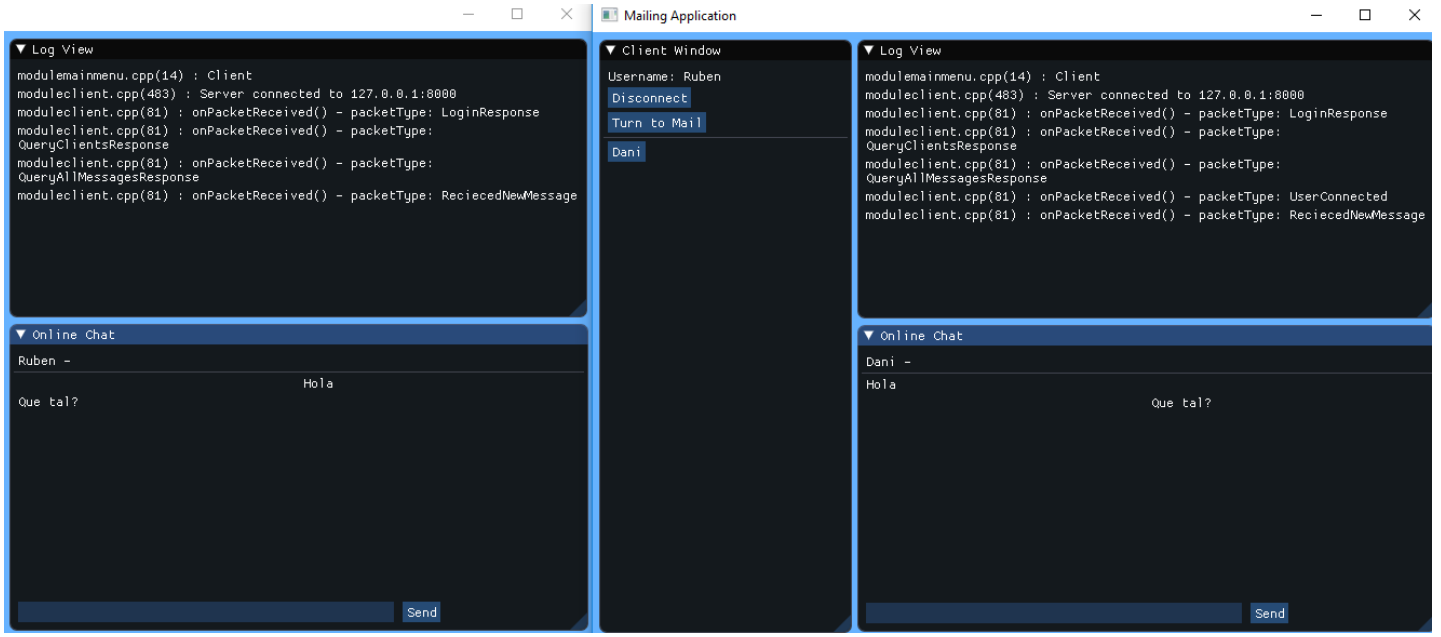


Si pulsas el botón del usuario con quien quieres chatear, la ventana de OnlineChat pasa a mostrar el chat con los mensajes entre el usuario y el otro cliente.

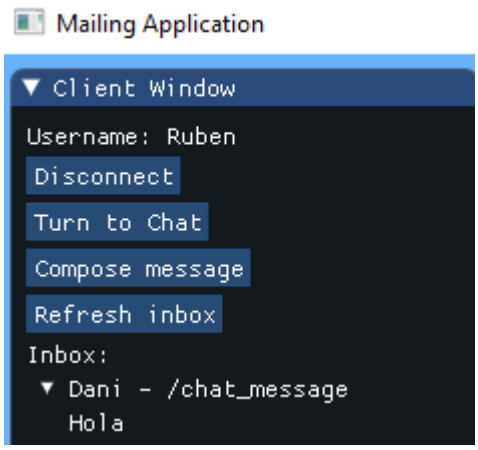


Cuando escribes el contenido del mensaje y pulsas Send, el cliente que ha de recibir el mensaje recibe su paquete y se actualiza el chat.

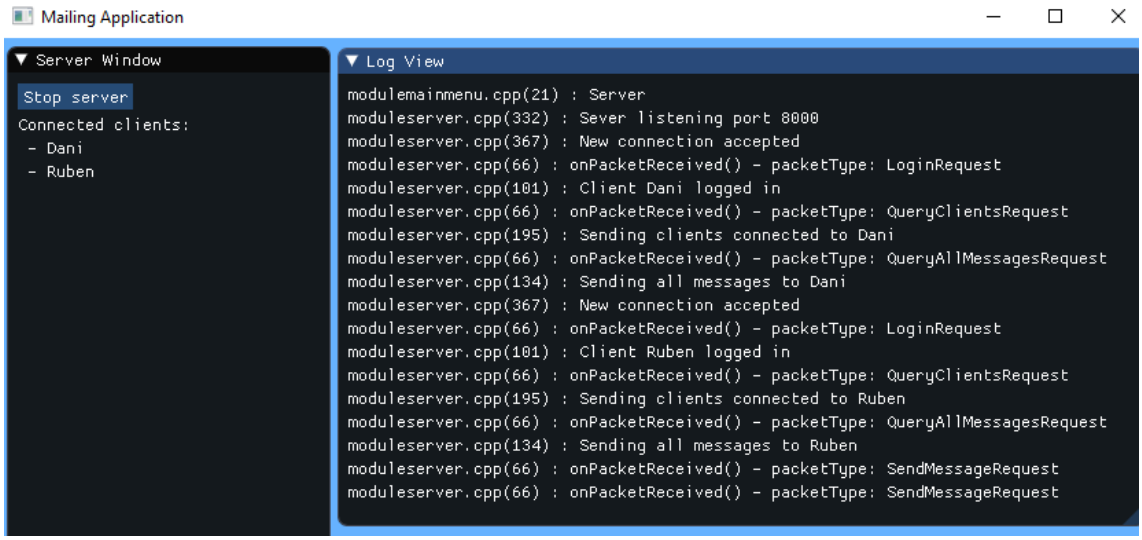




Pulsando “Turn to Mail” pasas a ver la vista antigua de correo. El sujeto de los mensajes mandados a través del chat es por defecto “/chat_message”.



Aquí se ve la consola del servidor simulado y la lista de usuarios conectados. Después de que el servidor reciba la conexión de un nuevo cliente, el cliente manda su paquete de LoginRequest con sus iniciales y contraseña. Si el cliente no se desconecta por no tener un usuario y contraseña válidos, el cliente pide los clientes que hay conectados y luego los mensajes registrados. Cuando un cliente manda un mensaje, primero se registra en la base de datos y luego se comprueba si el receptor del mensaje está conectado para nadárselo.



Nuevos Paquetes

```
enum class PacketType : int8_t
{
    Empty,
    LoginRequest,
    LoginResponse,
    QueryClientsRequest,
    QueryClientsResponse,
    QueryAllMessagesRequest,
    QueryAllMessagesResponse,
    RecievedNewMessage,
    UserConnected,
    SendMessageRequest
};
```

- **LoginRequest** – El cliente manda su usuario y contraseña
    - Tipo + (string)usuario + (string)contraseña en md5

    ```
    ModuleClient::sendPacketLogin(const char * username, const char * password)
    ```

- **LoginResponse** – El cliente manda la respuesta. Si estaba registrado y la contraseña es errónea, se devuelve falso y hará que el cliente se desconecte. Si la contraseña es correcta o no estaba registrado, se devuelve verdadero.
    - Tipo + (bool)conectado

    ```
    ModuleServer::sendPacketLoginResponse(SOCKET socket, const bool connected)
    ```

- **QueryClientsRequest** – El cliente pide los usuarios conectados.
    - Tipo

    ```
    ModuleClient::sendPacketQueryClients()
    ```

- **QueryClientsResponse** – El servidor manda los usuarios conectados.
    - Tipo + (int)numero de clientes + (string)usuario + (string)usuario + (string)usuario…

    ```
    ModuleServer::sendPacketQueryClientsResponse(SOCKET socket)
    ```

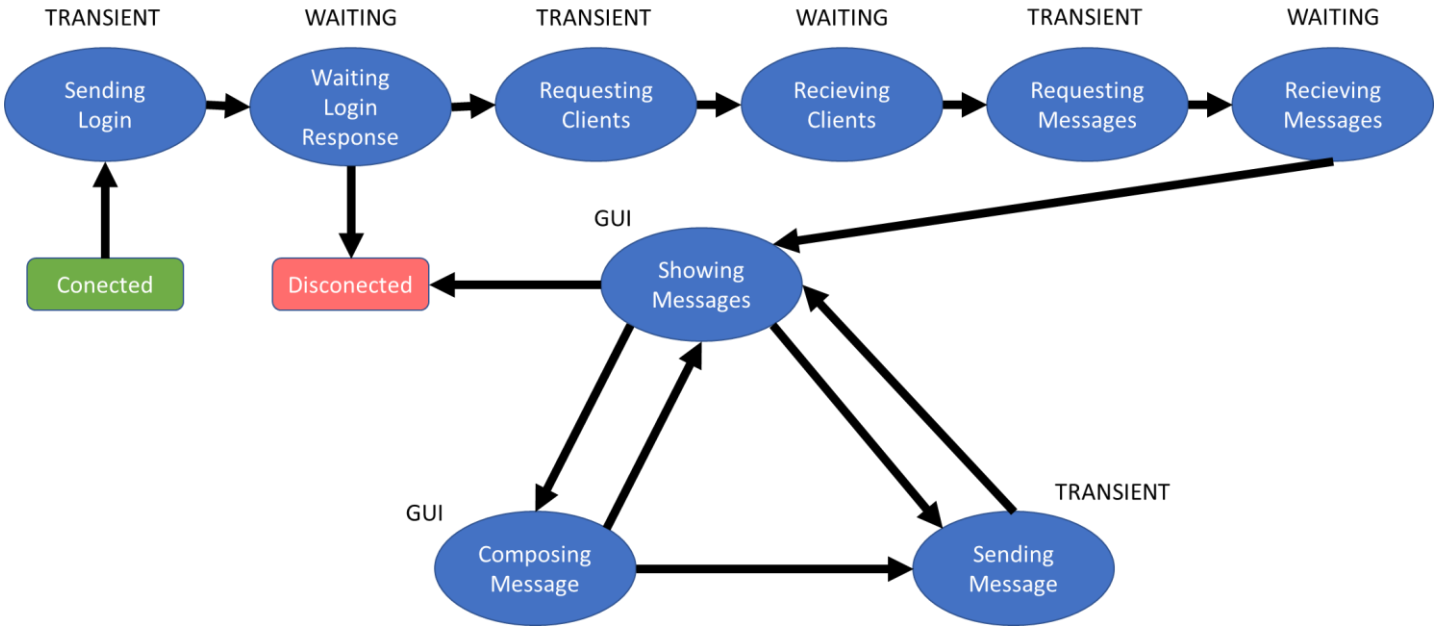
- **RecievedNewMessage** – Al mandar un mensaje a un usuario, si el receptor está conectado, el servidor le manda el mensaje.
    - Tipo + (string)cliente mandador + (string)sujeto mensaje + (string)cuerpo de mensaje

    ```
    ModuleServer::sendPacketMessageSent(SOCKET socket, const Message & message)
    ```

- **UserConnected** – Cuando un usuario se conecta, el servidor le manda a cada cliente conectado un paquete con el nombre del cliente nuevo.
    - Tipo + (string)cliente

    ```
    ModuleServer::sendPacketUserConnected(SOCKET socket, const char * client)
    ```

Máquina de Estados del Cliente




```
enum class MessengerState
{
    SendingLogin,
    WaitingLoginResopnse,
    RequestingClients,
    ReceivingClients,
    RequestingMessages,
    ReceivingMessages,
    ShowingMessages,
    ComposingMessage,
    SendingMessage
};
```

Cuando el usuario se CONECTA:

1. Manda al servidor usuario y contraseña.
2. Espera a recibir respuesta.
    a. Si la respuesta es válida, pasa al siguiente estado.
    b. Si la respuesta no es válida, el cliente se DESCONECTA.
3. Pide al servidor los clientes
4. Espera a recibir los clientes
5. Pide al servidor los mensajes (los enviados y recibidos por el usuario)
6. Espera a recibir los mensajes
7. Ver los mensajes
    a. En vista de Mail se muestran los mensajes recibidos
    b. En vista de Chat se muestran los usuarios conectados y puedes seleccionar la conversación a mostrar. No es necesario pasar a componer mensaje en vista de chat.
8. Componer mensaje – solo se puede ver en vista de Mail
9. Mandar mensaje

Base de Datos

Tablas

User
name – VARCHAR(64)
Password – VARCHAR(64)

Message
sender – VARCHAR(64)
receiver – VARCHAR(64)
subject – VARCHAR(256)
body – VARCHAR(4096)

Llamadas de MySqlDatabaseGateway

```
bool CheckPasswordForClient(const std::string & username, const std::string & password)
```

Check if user is registered:

```
sqlStatement = stringFormat("SELECT * FROM user WHERE name = '%s'", username.c_str());
```

Register a user:

```
sqlStatement = stringFormat("INSERT INTO user VALUES ('%s','%s')", username.c_str(), password.c_str());
```

```
void insertMessage(const Message & message)
```

Insert message to ‘message’ table with sender, receiver, subject and body.

```
sqlStatement = stringFormat("INSERT INTO message VALUES ('%s','%s','%s','%s')",
message.senderUsername.c_str(), message.receiverUsername.c_str(), message.subject.c_str(),
message.body.c_str());
```

```
std::vector<Message> getAllMessagesFromUser(const std::string & username)
```

Get sent and received messages from a user:

```
sqlStatement = stringFormat("SELECT * FROM message WHERE receiver = '%s' OR sender = '%s'",
username.c_str(), username.c_str());
```