Rees Raphael
Y-quynh Nguyen

UML REPORT

Use Case Diagram

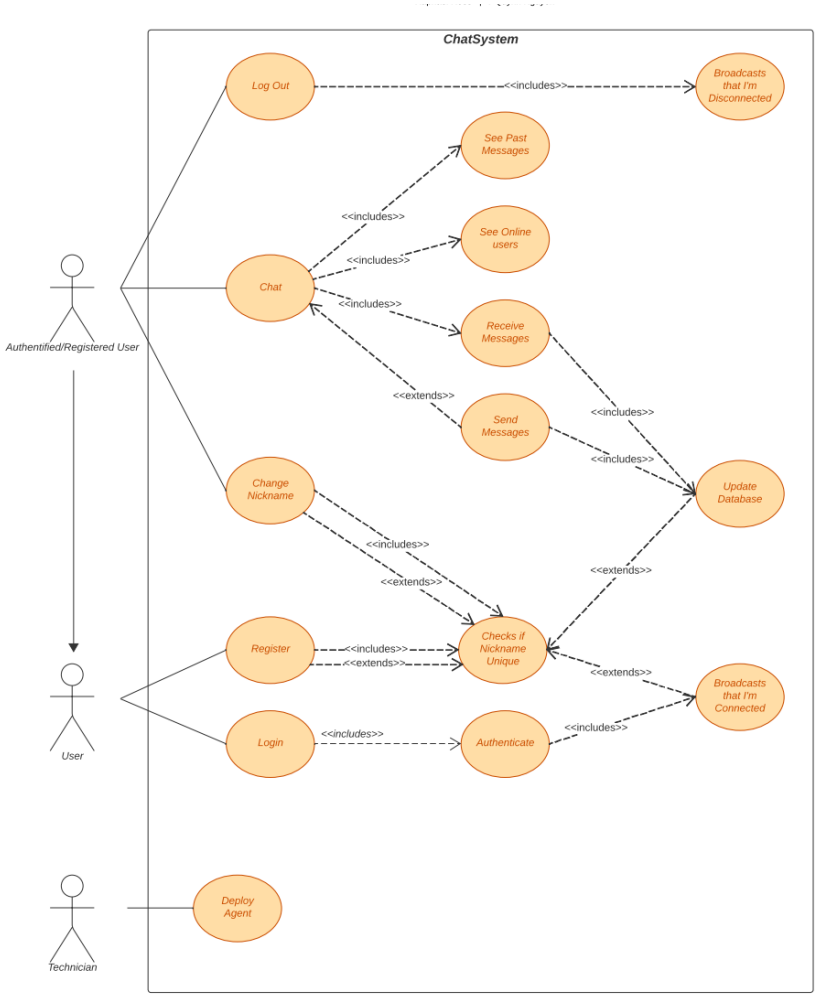
The actors in this case are the users and the technicians that deploy the agent.

Some assumptions that we are working with are:

a user always connects to the agent on his machine, that is where his database will be.

we can only talk to users that are connected right now

all machines of the company are all on the same local network



The use case diagram's goal is to help understand how our product works from the user's perspective. The actors are the users and the technicians. We decided to separate

authentified/registered users and users when they first get to the app to clarify the options the users get.

A technician can deploy an agent.

A user can login or register, which will lead to the agent checking the uniqueness of the nickname if they register or authenticating the user if they login. Depending on the input, it will broadcast that the user is connected if the input was in the correct format and passes some conditions or will go back to the previous round to get another input. That broadcast allows the other users to update their Contact List and know they can talk to the user. Otherwise, if they successfully logged in, it will update the database and save who the user is.

If they were able to sign in or up for the app, they will become an authentified/registered user. In that case, they will chat: meaning receive messages, see the list of users they can talk to. They can send messages and see past messages if they wish to. The messages sent and received are saved in the database in a Table for each contact we are talking to.

They can also log out, which broadcasts a message warning other users they can't talk to the user anymore since he is disconnected.

Class Diagram

We have decided to focus on the connection between the Controller and the Model. We have decided to not represent the ui package in the diagram. We have tried to keep the classes as separate as possible.

We have the interface MyObserver that has the list of possible observers. They are then implemented in the class Controller that is the acting link between most of the classes. It is notified by several classes when specific events happen.

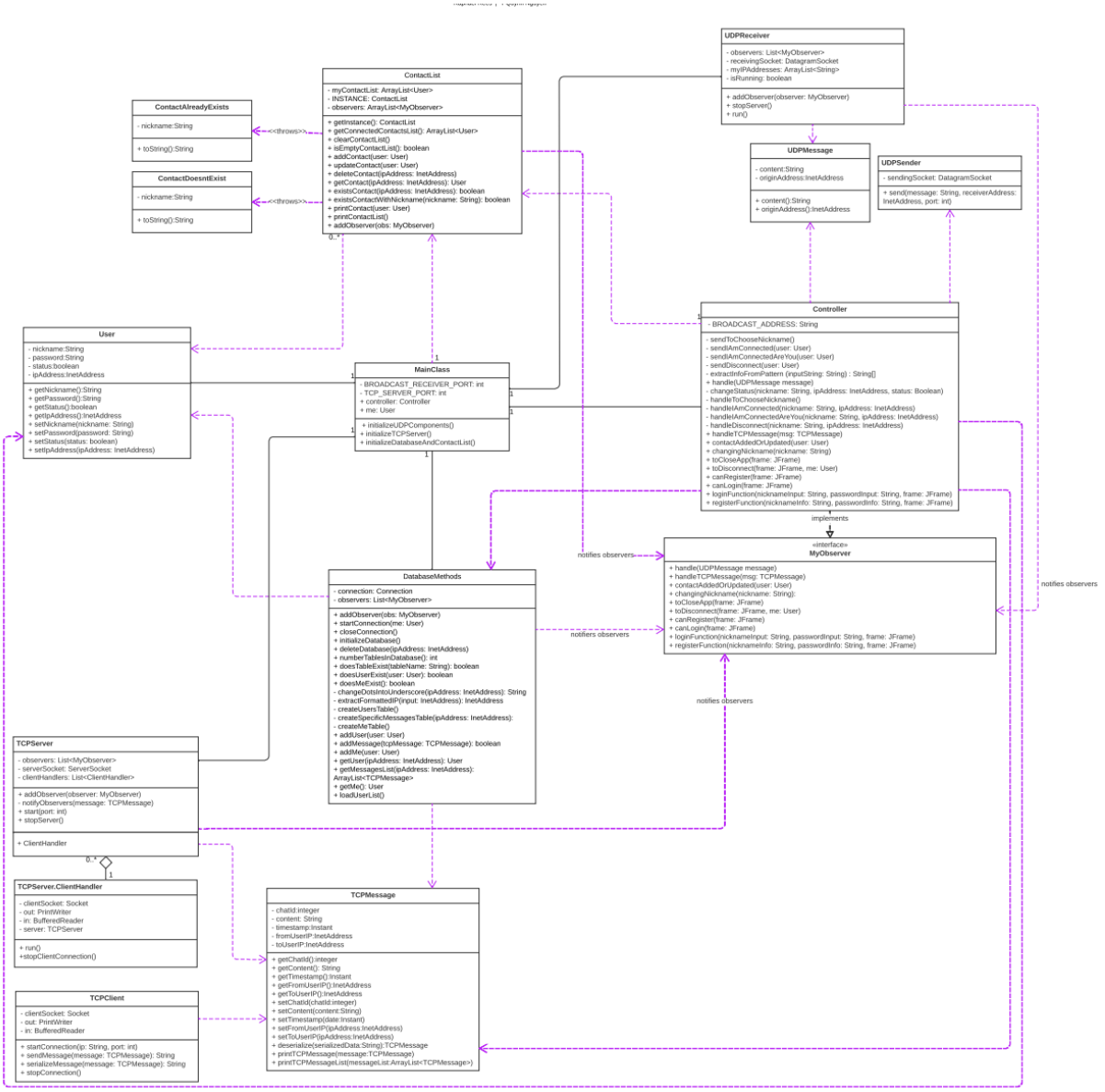
User, TCPMessage and UDPMessage are classes that represent the Model. They have private attributes that are accessed through getters and setters.

ContactAlreadyExists and ContactDoesntExist extend Exception and are thrown by ContactList.

ContactList and DatabaseMethods are classes that allow accessing, handling and modifying their instance/database. They have methods to better manipulate them.

Then there are the classes TCPServer/TCPClient and UDPSender/UDPReceiver that allow handling the network connections.

Rees Raphael
Y-quynh Nguyen



Database schema

Each person who connects has a local SQLITE3 database, organised as follows :

1. Users Table

- nickname (TEXT): The nickname of the user (unique)
- ipAddress (TEXT): The IP address of the user, serving as the primary key.
- status (INT): The user's status (0 for offline, 1 for online).
- password (TEXT): The user's password.

2. Messages Table (for each user)

Rees Raphael
Y-quynh Nguyen

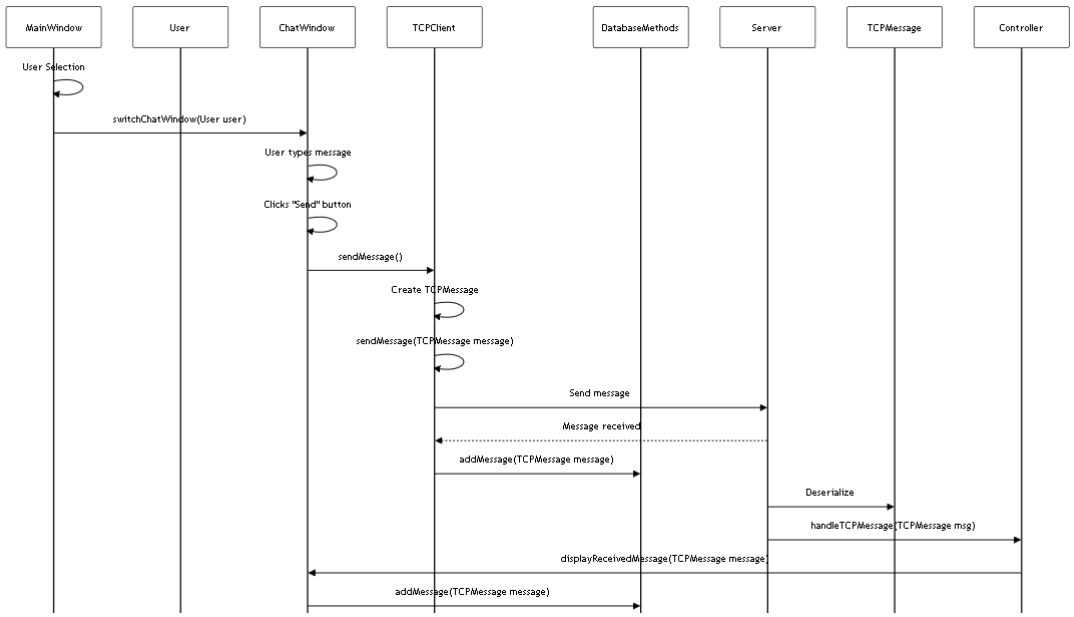
There are separate messages tables for each user we have in our Users Table, named dynamically based on the user's IP address (e.g., Messages_192_168_1_1).

- chatID (INTEGER): A unique identifier for each message, primary key, auto-increment.
- content (TEXT): The content of the message.
- date (TEXT): The date and time when the message was sent (using timestamp).
- fromUser (TEXT): The IP address of the user who sent the message.
- toUser (TEXT): The IP address of the recipient user.

3. Me Table

- nickname (TEXT): My nickname.
- ipAddress (TEXT): My IP address (primary key).
- password (TEXT): My password.

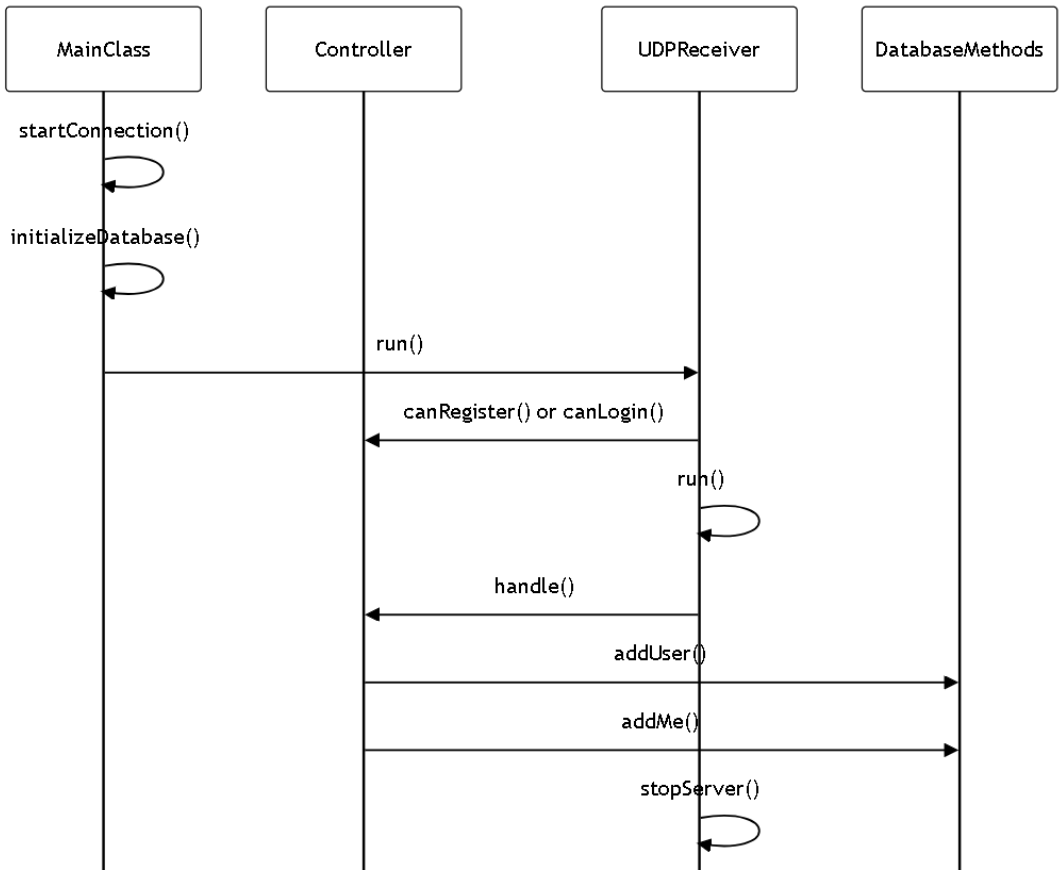
Sequence Diagrams



Sequence Diagram for sending/receiving messages

When a user selects a contact in the MainWindow, the switchChatWindow(User user) method is called to open a chat session with the chosen user. For sending messages, the ChatWindow class's sendMessage() method is invoked upon user interaction. This method creates a TCPMessage object, which is then sent through the TCPClient class's sendMessage(TCPMessage message) method. After the message is sent, it's stored in the database using DatabaseMethods.addMessage(TCPMessage message). On the receiving end, the Controller class's handleTCPMessage(TCPMessage msg) method is triggered when a message arrives. It retrieves the appropriate ChatWindow using MainWindow.getChatWindowForUser(String

userKey) and displays the message using displayReceivedMessage(TCPMessage message), also storing it in the database.



Sequence Diagram for contact discovery phase

In the contact discovery phase of the chat system, the application begins by establishing a connection to its database and starting the UDPReceiver to listen for incoming UDP packets. When a user either registers or logs in, the system sends a UDP broadcast message to discover active users on the network. This message prompts responses from other users, which are then received and processed by the UDPReceiver. The Controller class handles these messages, updating the contact list with new or modified user information. This updated contact list is then loaded from the database for use within the application. Finally, when the application is closed or the user logs out, the UDPReceiver is shut down, and all connections are closed.

PDLA

Rees Raphael
Y-quynh Nguyen

Agile methodology was combined with PDLA (Processus de Développement Logiciel Automatisé) to facilitate an automated development process by focusing on iterative development. Agile practices like short sprints and regular reviews made for easier planning and adapting.

By using GitHub's Continuous Integration tools, every push to the main branch launched automated builds and tests.

A simplified GitHub workflow was adopted, involving direct pushes to the main branch. We decided to do this because, with only two members on the project, this avoided the complexity and associated with branch management.