Rees Raphael
Y-quynh Nguyen

UML REPORT

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)

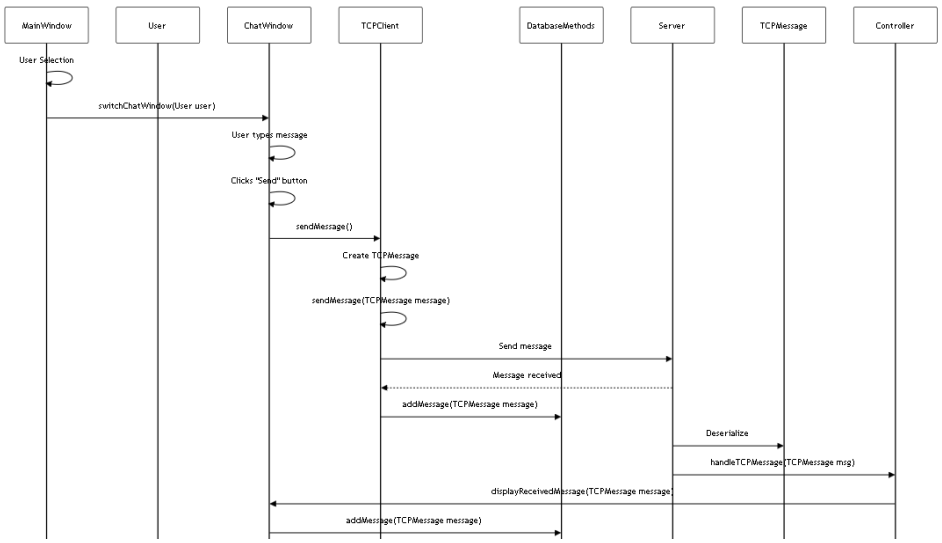
There are separate messages tables for each user we have in our Users Table, named dynamically based on the user's IP address (e.g., Messages_192_168_1_1).

- 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.

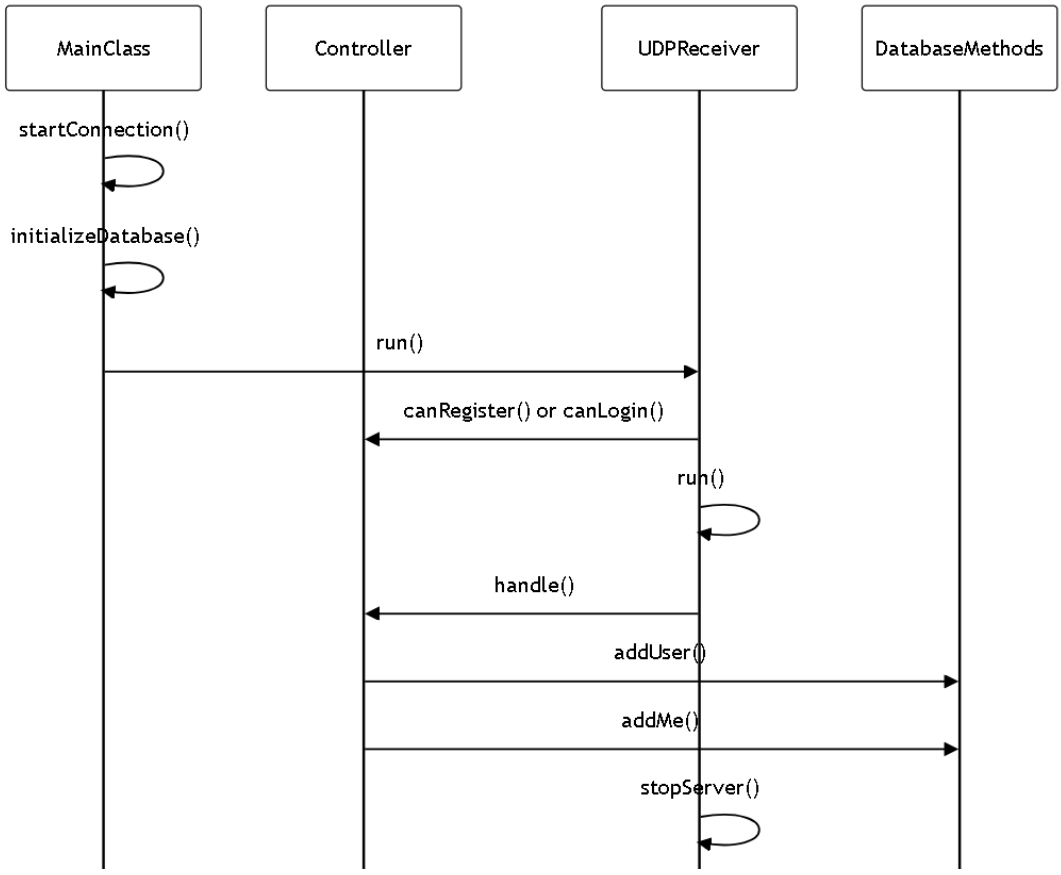
Sequence Diagrams



Sequence Diagram for sending/receiving messages

Rees Raphael
Y-quynh Nguyen

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

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.

JIRA

https://projetgeoqueen.atlassian.net/jira/software/projects/CHAT/boards/2?atlOrigin=eyJpIjoiMjM4MDZlZGI1MjQzNDNlMzlmMGVlYjczYjRlZTUyMGIiLCJwIjoiaiJ9

Class Diagram of ChatSystem

Raphael Rees | Y-Quynh Nguyen

UDPReceiver

- observers: List<MyObserver>
- receivingSocket: DatagramSocket
- myIPAddresses: ArrayList<String>
- isRunning: boolean

+ addObserver(observer: MyObserver)
+ stopServer()
+ run()

ContactList

- myContactList: ArrayList<User>
- INSTANCE: ContactList
- observers: ArrayList<MyObserver>

+ getInstance(): ContactList
+ getConnectedContactsList(): ArrayList<User>
+ clearContactList()
+ isEmptyContactList(): boolean
+ addContact(user: User)
+ updateContact(user: User)
+ deleteContact(ipAddress: InetAddress)
+ getContact(ipAddress: InetAddress): User
+ existsContact(ipAddress: InetAddress): boolean
+ existsContactWithNickname(nickname: String): boolean
+ printContact(user: User)
+ printContactList()
+ addObserver(obs: MyObserver)

ContactAlreadyExists

- nickname:String

+ toString():String

<<throws>>

ContactDoesntExist

- nickname:String

+ toString():String

<<throws>>

0..*

UDPMessage

- content:String
- originAddress:InetAddress

+ content():String
+ originAddress():InetAddress

UDPSender

- sendingSocket: DatagramSocket

+ send(message: String, receiverAddress: InetAddress, port: int)

User

- nickname:String
- password:String
- status:boolean
- ipAddress:InetAddress

+ getNickname():String
+ getPassword():String
+ getStatus():boolean
+ getIpAddress():InetAddress
+ setNickname(nickname: String)
+ setPassword(password: String)
+ setStatus(status: boolean)
+ setIpAddress(ipAddress: InetAddress)

MainClass

- BROADCAST_RECEIVER_PORT: int
- TCP_SERVER_PORT: int
+ controller: Controller
+ me: User

+ initializeUDPComponents()
+ initializeTCPServer()
+ initializeDatabaseAndContactList()

1 1 1 1 1 1

Controller

- BROADCAST_ADDRESS: String

- sendToChooseNickname()
- sendIAmConnected(user: User)
- sendIAmConnectedAreYou(user: User)
- sendDisconnect(user: User)
- extractInfoFromPattern (inputString: String) : String[]
+ handle(UDPMessage message)
- changeStatus(nickname: String, ipAddress: InetAddress, status: Boolean)
- handleToChooseNickname()
- handleIAmConnected(nickname: String, ipAddress: InetAddress)
- handleIAmConnectedAreYou(nickname: String, ipAddress: InetAddress)
- handleDisconnect(nickname: String, ipAddress: InetAddress)
+ handleTCPMessage(msg: TCPMessage)
+ contactAddedOrUpdated(user: User)
+ changingNickname(nickname: String)
+ toCloseApp(frame: JFrame)
+ toDisconnect(frame: JFrame, me: User)
+ canRegister(frame: JFrame)
+ canLogin(frame: JFrame)
+ loginFunction(nicknameInput: String, passwordInput: String, frame: JFrame)
+ registerFunction(nicknameInfo: String, passwordInfo: String, frame: JFrame)

implements

«interface»
MyObserver

+ handle(UDPMessage message)
+ handleTCPMessage(msg: TCPMessage)
+ contactAddedOrUpdated(user: User)
+ changingNickname(nickname: String):
+ toCloseApp(frame: JFrame)
+ toDisconnect(frame: JFrame, me: User)
+ canRegister(frame: JFrame)
+ canLogin(frame: JFrame)
+ loginFunction(nicknameInput: String, passwordInput: String, frame: JFrame)
+ registerFunction(nicknameInfo: String, passwordInfo: String, frame: JFrame)

notifies observers

notifies observers

notifiers observers

notifies observers

DatabaseMethods

- connection: Connection
- observers: List<MyObserver>

+ addObserver(obs: MyObserver)
+ startConnection(me: User)
+ closeConnection()
+ initializeDatabase()
+ deleteDatabase(ipAddress: InetAddress)
+ numberTablesInDatabase(): int
+ doesTableExist(tableName: String): boolean
+ doesUserExist(user: User): boolean
+ doesMeExist(): boolean
- changeDotsIntoUnderscore(ipAddress: InetAddress): String
- extractFormattedIP(input: InetAddress): InetAddress
- createUsersTable()
- createSpecificMessagesTable(ipAddress: InetAddress):
- createMeTable()
+ addUser(user: User)
+ addMessage(tcpMessage: TCPMessage): boolean
+ addMe(user: User)
+ getUser(ipAddress: InetAddress): User
+ getMessagesList(ipAddress: InetAddress):
ArrayList<TCPMessage>
+ getMe(): User
+ loadUserList()

TCPServer

- observers: List<MyObserver>
- serverSocket: ServerSocket
- clientHandlers: List<ClientHandler>

+ addObserver(observer: MyObserver)
- notifyObservers(message: TCPMessage)
+ start(port: int)
+ stopServer()

+ ClientHandler

0..* 1

TCPServer.ClientHandler

- clientSocket: Socket
- out: PrintWriter
- in: BufferedReader
- server: TCPServer

+ run()
+stopClientConnection()

TCPClient

- clientSocket: Socket
- out: PrintWriter
- in: BufferedReader

+ startConnection(ip: String, port: int)
+ sendMessage(message: TCPMessage): String
+ serializeMessage(message: TCPMessage): String
+ stopConnection()

TCPMessage

- chatId:integer
- content: String
- timestamp:Instant
- fromUserIP:InetAddress
- toUserIP:InetAddress

+ getChatId():integer
+ getContent(): String
+ getTimestamp():Instant
+ getFromUserIP():InetAddress
+ getToUserIP():InetAddress
+ setChatId(chatId:integer)
+ setContent(content:String)
+ setTimestamp(date:Instant)
+ setFromUserIP(ipAddress:InetAddress)
+ setToUserIP(ipAddress:InetAddress)
+ deserialize(serializedData:String):TCPMessage
+ printTCPMessage(message:TCPMessage)
+ printTCPMessageList(messageList:ArrayList<TCPMessage>)

Use Case Diagram of our ChatSystem

Raphael Rees | Y-Quynh Nguyen