Network Assignment 1 pyChat Report

Nivan Poken (PKNNIV001), David Pullinger (PLLDAV013), Pfano Matsheketsheke (MTSPFA002)

PKNNIV001		
PLLDAAV013		
MTSPFA002		

Introduction

Having been tasked to implement a Python chat application that uses UDP and our own application layer protocol, we created a chat application named pyChat - Chatting made easy as pie. This application allows users to exchange messages to each other using the client-server architecture.

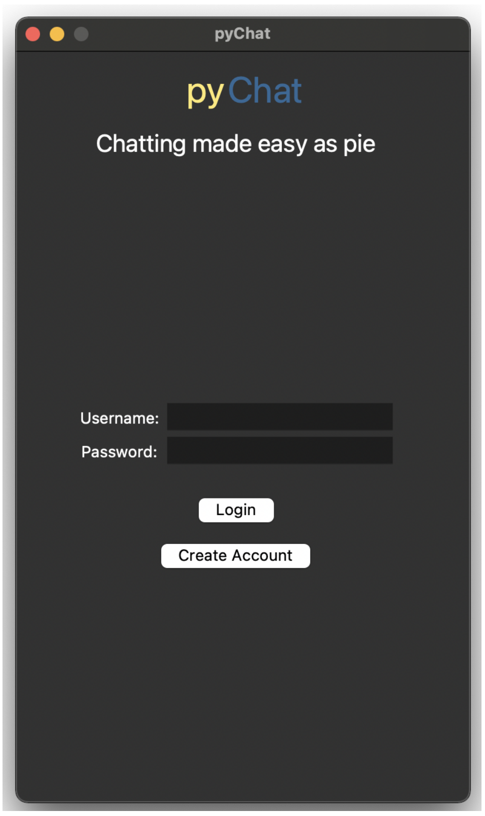


Figure 1:pyChat Landing page

In this report, we will detail the design and functionality of pyChat. We will first describe the system's functionality and features, followed by the system's specifications. We have also included screenshots of our application which will aid in revealing its features.

Functionality

pyChat is a Python chat application that allows users to exchange messages with other users on the application. To use pyChat, users will need to start the application by running chatUI.py, which has a registration/login screen as a landing page. Here first-time users will create a username (a unique name created by the user), and a password. This process only needs to be done once, as the account details are kept on the server and not on the client (device running the app).

There are two main functions that pyChat provides, namely:

1. Send and receive individual messages to and from users who are on the application - Private Chat.
2. Send and receive group messages to and from users who are on the application - Group Chat.

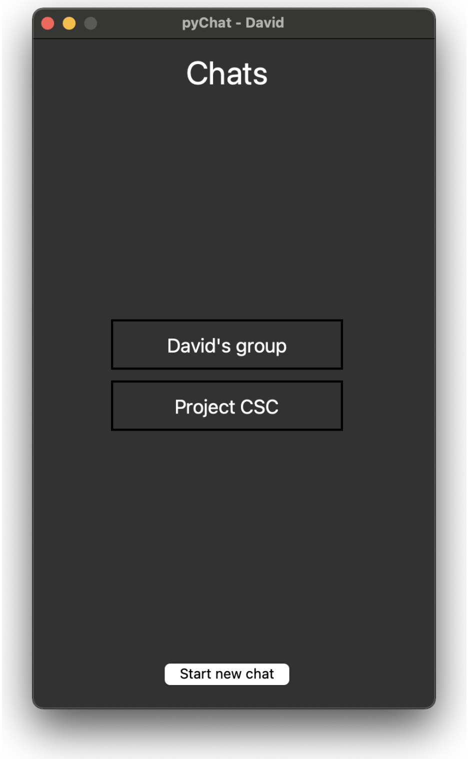
1. Private Chat

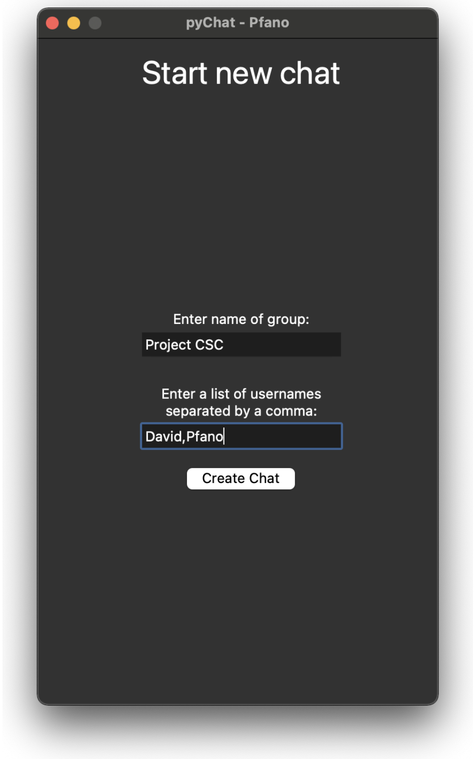
All users who are on pyChat have the ability to send private messages to individuals. We refer to this function as a Private Chat.

Messages that are sent within a private chat are only seen by the two individuals in the chat, this is because every message sent has an attached "SenderID" and a "groupID" which ensures that the messages with that "groupID" is only seen by the individuals in that "group" - which in this case is the private chat group.

After a user logs into pyChat, the application will prompt them to either start a "new chat" using the "create new group" button or click on an existing chat /group (if applicable). To initiate a private chat, the user must follow these steps:

1. Click on the "Start new chat" button - the user will be taken to a window where they can type the name of the individual they wish to start a chat with and type in the name of the chat. (See figure 2)
2. Click the "create chat" button after typing that individual's name - this will take the user to the private chat that has been created. (See figure 3)





All private chats that are created will be seen by the other user, once they log into pyChat, as a group chat with only two people inside that group.

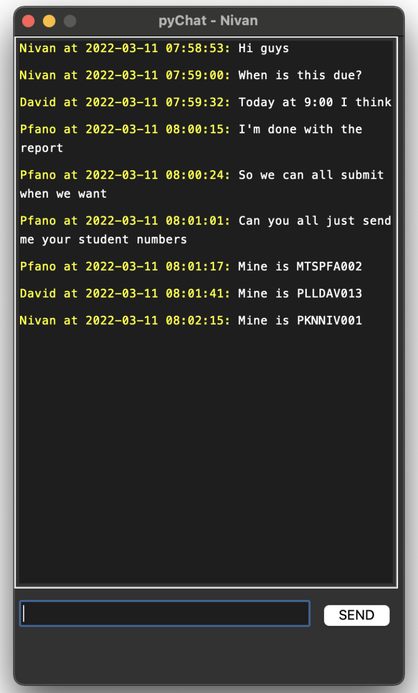
2. Group Messaging / Group Chat

Another function that comes with pyChat is the ability to exchange messages with multiple users at the same time. We refer to this function as a Group chat.

Similar to the private chat feature where all messages that are sent have an attached “messageID” and a “groupID” which ensures that messages are only seen by individuals in that group, all messages sent to a group are only visible to users in that group.

To create a group chat, the user would follow the same prompts that are used to create a private chat, but instead of typing out a single name, the user would type out multiple users' names. and click the “create chat” button. This will take the user to a chat room with the multiple users. All the users who have been added to the group will be able to see the group chat as well as all the participants of the group.

pyChat currently has no limit to the number of participants a group can have.



```
Nivan at 2022-03-11 07:58:53: Hi guys
Nivan at 2022-03-11 07:59:00: When is this due?
David at 2022-03-11 07:59:32: Today at 9:00 I think
Pfano at 2022-03-11 08:00:15: I'm done with the
report
Pfano at 2022-03-11 08:00:24: So we can all submit
when we want
Pfano at 2022-03-11 08:01:01: Can you all just send
me your student numbers
Pfano at 2022-03-11 08:01:17: Mine is MTSPFA002
David at 2022-03-11 08:01:41: Mine is PLLDAV013
Nivan at 2022-03-11 08:02:15: Mine is PKNNIV001
```

Figure 4: pyChat chat screen

Design

User Interface design

When designing the UI of pyChat, we went for a minimalistic approach ensuring that the main functions of the app are easily accessible to the users - easy to identify and easy to use. Some of the distinct design features include:

- Large buttons - These buttons give the affordance of a clickable interaction with a mouse. This allows for all types of users to interact with the application (from those who are not well acquainted with computers/desktops, to those who are seasoned users of computers)
- “{Action word and noun}” format as button labels - Combined with the large buttons, this labelling format gives clear mapping as to what the button does. This ensures that the user can make an

informed decision on which action they would like to take and not be surprised by the result of clicking the button. An example can be seen in figure 2.2 where the button is labelled “Start new chat”, when a user clicks that button, they can expect a response that allows them to create/ start a new chat.

- Few action buttons available - We have limited the UI to only have the necessary buttons needed for a successful chat. This helps keep the design “clean” and it does not complicate the user’s experience.

With this design pyChat is user friendly for the majority groups of users and because it has a simple minimalistic design, it can be used on most devices (windows, Mac and linux) that support Python 3.9 or later.

Application Architecture design

When designing the architecture of pyChat, a two-tier architecture was used – The Client-server architecture.

The client layer contains the graphical user interface, the application as well as the protocol to send messages while the server layer contains the server and the data source. Although this architecture is harder to maintain (tight couple) the main benefit of this structure is the speed at which the program runs and executes tasks.

Database design

When designing the database layer of pyChat, we separated the data into four tables, namely “message”, “user”, “users in group” and “group”. Figure 5 indicates the relationships between these tables.

1. Message: stores all the messages that have been sent through the pyChat server, every message has a
    - unique messageID – that identifies the message itself,
    - groupID – which identifies which group/individual the message is sent to
    - Sender – which identifies the individual who sent the message
    - Content – which is the content of the message
    - sentTime – a date time stamp of when the message was sent
2. User: This table stores all the users of pyChat, those who have created an account. Every user has a
    - Unique username – which identifies the user
    - A hashed password – a password that is hashed, this password is used by the user to login
3. Users in group: This table stores the usernames of members with the group they are linked to
4. Group: This table stores all the groups that are created in pyChat. Every group has a
    - GroupID – which is used to identify the group
    - Name – the name that is used by participants to identify the group

Features

pyChat has two distinct features that allow for a secure and convenient chatting experience, namely Password encrypting and Server stored data.

With password encryption, only the users have the knowledge of what their passwords are. Once the user submits their password in a human readable language, the program applies an encryption hashing key before it is sent to the server. This ensures that even if the data is intercepted, the users details will remain secure. The password is also saved in the database as the encrypted password for extra security.

Server stored data allows users to change devices freely without having to worry about backing up their chats. All the messages that are exchange on pyChat are not stored on the client but only on the server’s memory. The benefits of this also include the ability to read chat histories from any device and the ability to restore a conversation if it is cleared on the client side.

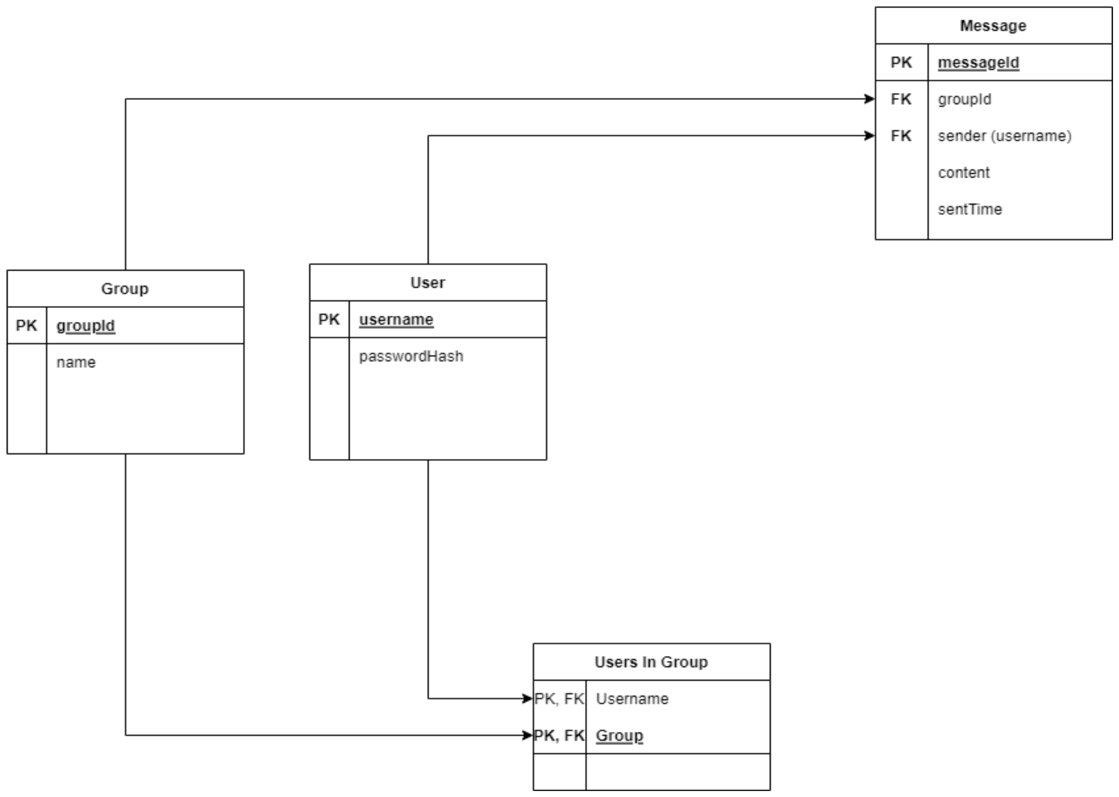


Figure 5: Database diagram

Protocol Design and Specifications

Message Format

The protocol has been implemented within a header and body system where the header contains all of the required protocol information, and the body contains the contents of the message that is to be sent. This has all been contained within a single string. To separate the header and body of the string, the following structure will be used:

{//	-*beginning of header*
CREATE_ACC	-*Action to be performed*
;	-*items separated by a semi-colon*
hu7dh20fh290dfhj48f8	-*Hash of message body to check for loss*
//}	-*End of header*
BODY TAGS	-*Tags that contain body content of message*

An example of this message being sent from the client to the server for a user who is sending message:

{//SEND_MSG;Hus456asdf782dasf4//}{“groupId”:“3”, “sender”:“JohnV01”, “content”:“Hi there guys!”}

Action types

Below is a list of the various actions our protocol implements, as well as canonical client requests and server responses.

Action Tag	Function	Body tags to be included
CREATE_ACC	Creates a new account	Request: username: JohnV01, password: sUpeRc001Passwd
		Response: (either) A. ACCOUNT CREATED B. USERNAME ALREADY EXISTS
LOGIN	Attempts to login to an existing account	Request: username: JohnV_01, password: sUpeRc001Passwd
		Response: (either) A. [messageObj1, messageObj2,...] B. INVALID PASSWORD OR USERNAME
UPDATE_MSGS (Each client sends this request at regular intervals while online)	Requests for all messages that have not yet been received by this specific client	Request: username: JohnV01, lastUpdate: 10 January 21:31:33
		Response: [messageObj1, messageObj2,...]
CREATE_GROUP	Requests to create a new group, which could be 1:1 or multiple users	Request: participants: [JohnV01, TracyL03, …] name: Cool temp group
		Response: (either) A. groupId B. PARTICIPANTS NOT VALID
SEND_MSG	Send a message to a specific group	Request: groupId: 3 sender: JohnV03 content: Hi all!
		Response: (either) A. MESSAGE SENT B. MESSAGE NOT SENT

pyChat's server is always on , this allows for the different clients to send and receive messages from a the same IP address. This means that the server is on a Listening state, waiting for client to send a message. Once the client creates a socket and sends a message, the Server changes states to either file transfer (see figure 7) or connected (see figure 6) depending on what the Action Tag on the message is. Action Tags that change the state from listening to connected are actions that do not need the server to transfer files back to the client, these are: CREATE_GROUP, CREATE_ACC and SEND_MSG. Action Tags that change the state from listening to file transfer are actions that require the server to send information back to the client, these are: UPDATE_MSGS and LOGIN.

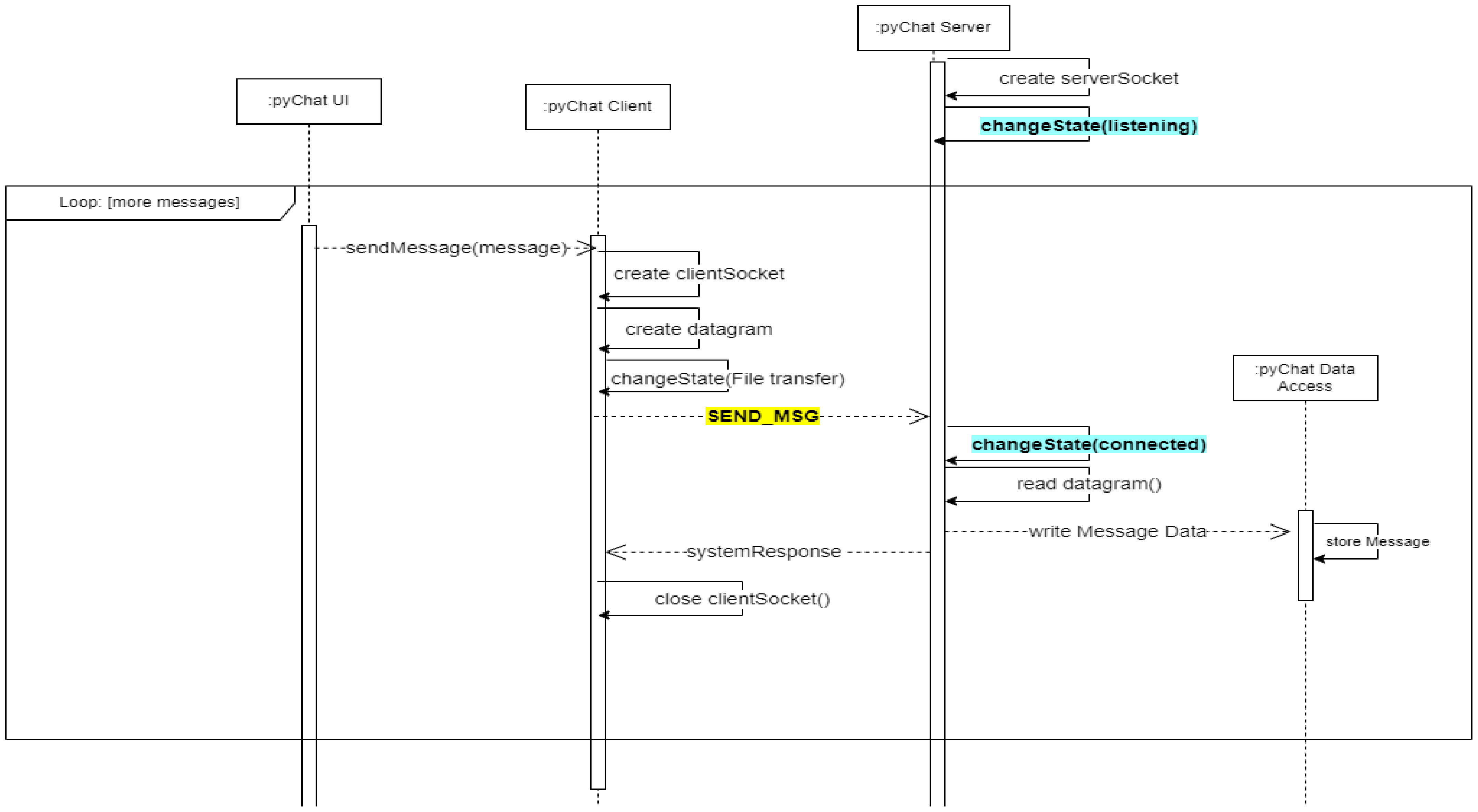


Figure 6: Detailed Sequence Diagram of a client sending a message

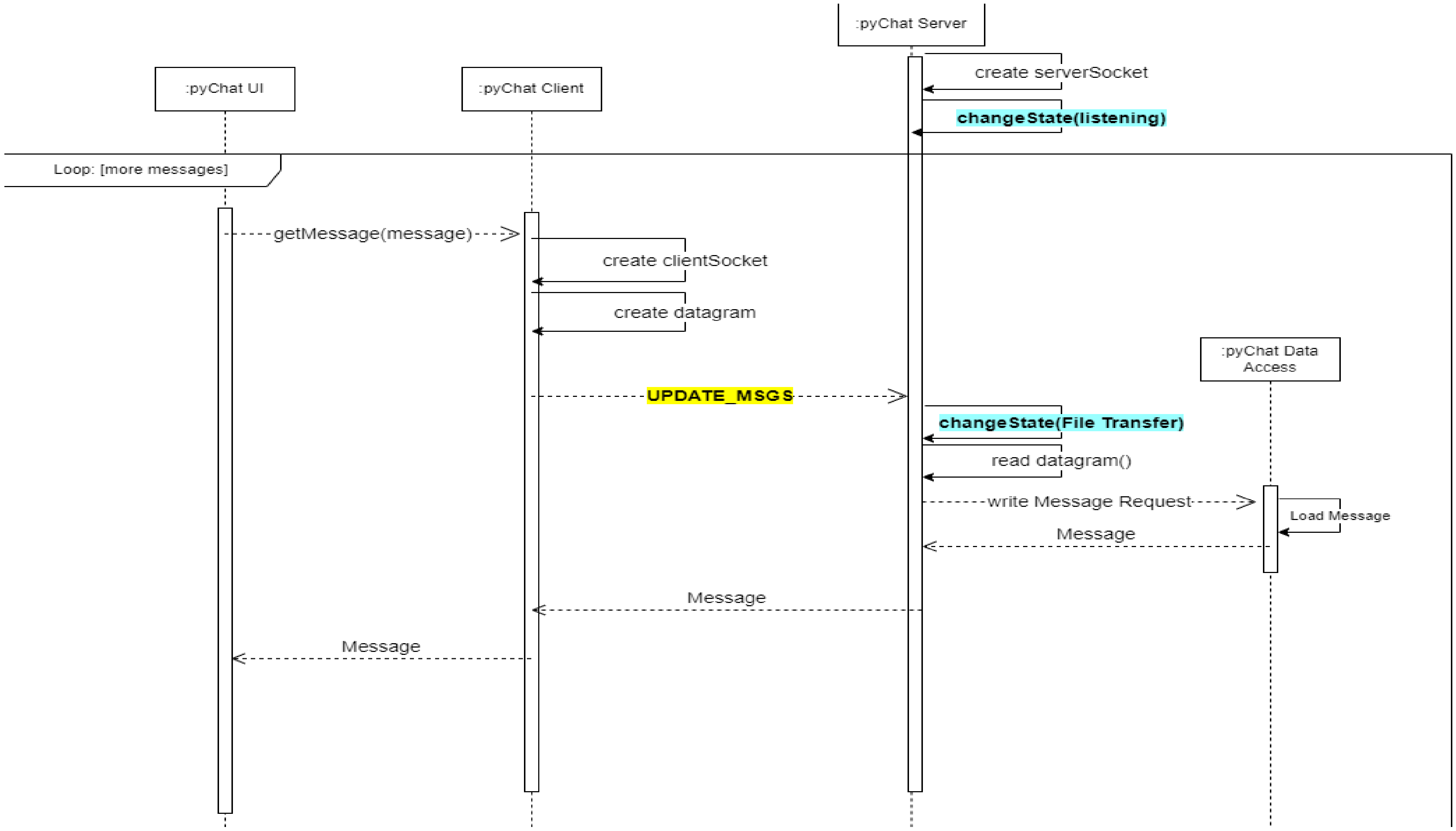


Figure 7: Detailed Sequence Diagram of client requesting to receive messages