# Therapy Bot – CS7319 Project Proposal

**Project Team Information (individual):**

Nicole Sood

CS 7319

On-Campus

GitHub Repo Link: <https://github.com/nsood1/TherapyBot>

**Project Overview:**

For my project, I will be creating a chatbot. The intent behind the chatbot is to act as a ‘therapist’ and would allow people to ask the bot questions on how to access mental health resources, as well as provide a friendly conversation. For the scope of this project, I am aiming to create a basic command line program to model the architecture as a proof of concept that the architecture can support the need of the application.

*What are the main functions of your software system?*

Within my software system, I need to have the following functions:

1. A client should be able to connect to the chatroom.
2. A client should be able to send a message in the chatroom.
3. The system should be able to return a response.

Functional requirements:

* The chatbot should be able to handle a user input.
* The chatbot should be able to take in different questions and statements.
* The chatbot should be able to generate a response based on different prompts.
* The chatbot should be able to access a database to get responses.
* The user should be able to start and end a conversation.
* The server should log a message to keep track of when someone has accessed the therapy portal.

Non-functional requirements:

* The chatbot should provide a reasonable output.
* The chatbot should be operationally available.
* The chatbot should be scalable, in case more people begin accessing the system.

*Describe the expected outputs or results of your software system:*

The expected output of my software system should be a conversation. A person should be able to ask the chatbot a question, and an automatic computer-generated response should be returned.

Examples of an expected output can be seen below:

<user>: Hi

<TherapyBot>: Hello, how can I help?

<user>: “what is anxiety?”

<TherapyBot>: “Anxiety is a medical condition spawned from our natural flight and fight instincts”

*Prototyping interface:*

For the scope of this project, I will be creating a command line program. As a result, I am excepting the interface to look as followed:

User Side:

*A screenshot of a computer

Description automatically generated*

(Prototype image created using Figma)

**Project Design:**

1. Peer to Peer Architecture:

For this application, one of the first applicable architectures I thought of was a peer-to-peer architecture. This is because peer to peer architectures allows for each machine to act as both a server/client.

A peer-to-peer architecture models a distributed system where a node can act as a server of a client. As a result, workload and be shared and distributed. A node can have the same functionality as its peer and can transmit and receive data as needed [1]. This architecture as a result is best used when you need features such as file sharing.

Below illustrates how this architecture might work for my chatbot.

Client/server

Database

Client/server

Client/server

Client/server

Within this architecture:

* Each machine can act as either a server or a client.
* Any of these machines can be used to spawn or respond to an input.
* Any node can be used to send a response or give a response.
* Multiple people can access one singular ‘server’ at one time.
* They can all connect to one central database which will contain information regarding appropriate responses.

Some of the advantages of a peer-to-peer architectures is that they have no single point of failure. This means if one of the ‘servers’ go down, data might not necessarily be lost because it is distributed across multiple notes. Additionally, you can control who is accessing and sharing data because it needs to be added to the network before it can be used. [1]

However, there are many issues with peer-to-peer architectures. The main big issues are regarding security. If a security risk is found on one of the nodes, due to the nature of the architecture, all the other peers connected in the system can also be affected. Additionally, there is not necessarily a way to control how files are being shared, and you also risk access becoming unavailable if all the servers end up offline. [1]

1. Client – Server Architecture:

The second architecture I decided to investigate is the client server architecture. In this architecture, multiple clients can access on server. This architecture works in the following way: firstly, the client (who is the user) will send a request. The request is then received and processed, at which point the server will provide a response to a client. [2]

Bellow illustrates how this architecture might work:

Server

Database

Client

Client

Client

Within this architecture:

* We have on central server which can access a database.
* This database is going to contain a series of training response data.
* A client can then send a request to the server (in the form of a chat)
* The server will send generate an appropriate response.
* Server will return a response to the client, in the form of an chat

An additional way to model a Client Server Architecture is within three tiers as seen below.

request

User Interface (terminal)

Middleware

Database

response

Client Server

*Diagram modelled from source [1].*

Within the first tier, you will have the client tier which will be the user interface. In my case this would be the command line. In the second tier is a form of middleware, this is where you will do most of the processing. Lastly, you have the database/server tier where information and responses can be spawned. [2]

Some of the advantages of a client server architecture is that it provides centralisation. This means all data will be in one location, and it also means that you can control who accesses it. In addition, a client server architecture allows for scalability. Which means, many clients can connect to the server, and it will control to function as expected. IT also provides more security than a peer-to-peer network because you can enforce access control, and if there was something to be injected into the system it would be easy to restore services. [3]

On the other hand, some of the disadvantages of a client server architecture is you might risk denial of service if too many clients are trying to connect at the same time. In addition, it would be difficult to maintain the server, which could be costly. [3]

**Final Architecture Choice: Client Server Architecture**

After evaluating the pros and cons of both a peer-to-peer architecture style, and a client server architecture; I settled on a client server for my project. Overall, I deemed it the most appropriate for a chat bot.

Firstly, I want my chatbot to be scalable. Although both provided scalability, the middleware aspect contained in the client server architecture was more appropriate. This is because I do not necessarily want my database to be accessed by all nodes in a network. Moreover, within a peer-to-peer network all the client/servers need to already be in the network. A client server allows for clients to connect when needed.

Secondly, although the denial of service is a risk with a client server architecture, there is greater security with this style. You will need to be authorised to directly access the database. Move over, if there a virus injected into the system it would not spread to other clients in the network. Recovery is also easier on a client server.

Lastly, many current chatbots are hosted on a client server architecture. As a result, it is shown to be able to fit business needs. For example, discord runs off a client/server model and even allows people access to their servers to design their own bots.

**Project Implementation Plan:**

I will be implementing my project in Python (using PyCharm as my IDE). I have chosen Python due to its flexibility in terms of the programming language. It also provides a lot of useful Machine Learning, Chatbot, and Artificial Intelligence library’s which I can utilize for my project.

To successfully implement this project, I will be completing the following:

* Setting up a client
* Setting up a server to run locally
* Create a database, in the form of a text file, which will contain information on responses.
* Can read in the data
* Provide a middle ware to generate the response
* Create a user interact for the client to send a message
* Server should be able to send a response to the client.

Works Cited/External Sources:

[1] Easy Algorithm, Peer to Peer (P2P) Architecture: <https://www.enjoyalgorithms.com/blog/peer-to-peer-networks>

[2] Simlilearn, Client Server Architecture: <https://www.simplilearn.com/what-is-client-server-architecture-article#:~:text=The%20client%2Dserver%20architecture%20refers,model%20or%20client%20server%20network>.

[3] zenarmor, What is a client server network: <https://www.zenarmor.com/docs/network-basics/what-is-client-server-network>

Other sources: include materials given during lecture.