# Therapy Bot: Final Project Presented by Nicole Sood

**Contents:**

Compilation & Implementation

Project Description

Architecture One: Peer to Peer (unselected)

Architecture Two: Client Server

Rational of selection and difference between the choices.

**Compilation & Implementation:**

I implemented this project using Python, with PyCharm 2023.2.1 (Professional Edition). This IDE can be downloaded from JetBrains. The system I am running is: MacOS Catalina Version 10.15.7 (which is out of date in terms of MacOS).

To install my application, simply clone my GitHub repo onto your local machine.

Necessary Packages/imports:

* Sqlite3
* nltk.chat.util
* socket
* select
* sys
* \_thread\_

To compile & execute the system:

1. Set up the database.   
   - For this step, navigate to the file directory and run the command **python databasesetup.py**
2. Initiate the server:

* The server is set up to run locally. To get it running, you will need to provide an ip address as well as a port number. You will run the command: **python server.py [ip address] [port number]**

Example: python server.py 127.0.0.1 8081

1. Connect the client:

* Using a different terminal, navigate to the directory and then run the client.py file, then run the following command:

**python client.py [ip address] [port number]**

* Example: python client.py 127.0.0.1 8081
* Multiple clients can connect to the same server. Each will have their own thread and can hold their own conversation.

**Project Description:**

For this project, I created a Therapy Chatbot. The purpose behind this is so that a person could use this as a supportive tool to get advice on how to deal with anxiety, depression ect. Conditions such as these affects over 30% of the population, and others might experience symptoms but do not know how to seek help. This tool will provide a non-judgemental way for people to get connected to the resources that they need.

There are two main users of this project, each with different operational scenarios and capability’s:

1. Those seeking help:
   1. The user will be able to interact with a text-based bot and supply it with prompts and questions. The bot will return appropriate responses.
   2. The user will receive a dynamic response based on their input.
   3. The user will also get a response in real time.
2. Those maintaining the tool:
   1. Therapists/those who have access to the back end will be able to add more information on prompts and responses if they so choose.
   2. There is an ability to restore the database from a basic set of prompts/responses should data be lost.
   3. They system will create and store a backup of all data in the database.

Other operational scenarios and capability’s:

* Multithreading has been implemented, which allows for multiple clients to connect to one server.
* Database integration:
  + Although the chatbot does not directly use the database, the database does allow for maintaining predefined responses.

**Architecture Option One: Peer to Peer Architecture (unselected):**

Background:

A peer to peer (P2P) architecture models a distributed system where a node can act as a server or a client. As a result, each node can have the same functionality as a peer, it can also retrieve and transmit data as needed. Often this architecture will be used with things such as file sharing.

Some of the advantages of a P2P architecture is that there is no single point of failure. This means that if one of the clients goes down the rest of the system can maintain its integrity. Additionally, because data is distributed across multiple nodes – in the event of a failure there is a chance that not all data is lost. Additionally, new clients need to be added to the system, this means you have control over who can access the system and its database.

On the other hand, there are many issues surrounding peer to peer architectures. For example, there are security related risks. If there is a security risk on one of the clients, it can easily spread across the whole entire system. In addition, there is not necessarily a way to control how files and data are being shared. This means people might get access to parts of the system they should not be able to. Additionally, you risk a denial of service if all nodes end up offline.

**Sources: https://www.enjoyalgorithms.com/blog/peer-to-peer-networks**

Component diagram showing the components and connectors

A screenshot of a computer

Description automatically generated

Class Diagram showing the classes and associations

A diagram of a chatbot

Description automatically generated

Mapping from each component/connector to its implementing classes in class diagram

This was my **unselected** architecture style. As a result, I do not have any implementing classes for this part of the project.

**Architecture Option Two: Client/Server Architecture (implemented):**

Background:

A client server architecture is when multiple clients can access one server. As a result, the client will send a request to the server. Once the request is received it will be processed and a response will be provided back to the client.

Some advantages of using a client server architecture means that you have centralization. All the data for the system will be in one location, and because of the interaction with the server you can control who accesses it. Additionally, this architecture allows for scalability which means multiple people can connect to the server at one time all while being in their own thread of requests/replies. Moreover, if there is a security issue such as a denial of service or a virus it will be easy to restore service due to the centralised design.

On the other hand, some of the disadvantages of this architecture is that you may be at risk for a denial-of-service attack is too many people connect to the server at the same time. This can be seen in the form of either slower service with the whole server crashing. Additionally, it is costly to maintain a server especially as it grows larger.

**Sources:** [**https://www.simplilearn.com/what-is-client-server-architecture-article**](https://www.simplilearn.com/what-is-client-server-architecture-article)

Component Diagram showing the components and connectors

A screenshot of a computer screen

Description automatically generated

Class diagram showing the classes and their associations

A diagram of a server

Description automatically generated

Mapping from each component/connector to its implementing classes in class diagram

|  |  |  |
| --- | --- | --- |
| **Component** | **Implemented Class** | **Functions** |
| Server | server.py | menuoptions()  clientthread(conn, addr, clients)()  broadcast(message, connection)()  remove (connection)() |
| Client | client.py |  |
| TherapyBot Logic | main.py | load\_pairs\_from\_file()  initalize\_chatbot()  main() |
| Database Handler | readindatabase.py | generaterecent()  restore()  add() |

Connectors:

* The client is going to get a user input. This input is then going to be passed into the therapybot logic on the server.
* The therapybot logic is going to take the input and then reference the database for an appropriate response.
* The therapybot logic once it gets a response on the server side, will send it to the client to be displayed to the user.
* Those working on the server side, also have an option to maintain the database. All input to update the database is done server side as well.

**Rational of Architecture Choice and Differences in Design Styles:**

Summary of Pro’s and Con’s for each style:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pro’s | Con’s | My system |
| Peer to Peer | * No single point of failure * Data is distributed across multiple nodes * Control who is accessing and sharing data | * High potential for security risks * Cannot control how files are shared * Potential for denial of service | Although we want to control access, we do not want to be adding people individually.  We do not intend to be file sharing which is the most common us for a peer-to-peer architecture. |
| Client Server | * Provides centralisation * Scalability * Enforced Access Controls * Easy to restore services | * Potential for denial of service * Costly to maintain server | We want the system to be scalable  We want to be able to enforce access control  Current design means we do not need to worry about cost to maintain server  Want to be able to restore in case of failure. |

Differences in architecture styles:

Overall, some of the main differences of the architecture styles include:

1. Resource Distribution
   1. A client server provides clear distinction between a client and a server meaning clearer access controls
   2. A P2P considers all nodes equal therefore they all have access to the same data.
2. Centralisation:
   1. A Client server uses a centralised model meaning that one server is responsible for allocating resources
   2. A P2P is decentralised meaning that all peers have the same controls and access to data.
3. Scalability:
   1. A client server will require more hardware to deal with more demand.
   2. A P2P can scale as large as needed.

Rational of Architecture Selection:

Overall, I settled on a client server architecture for my project. This is because I deemed it the most appropriate for my needs.

Firstly, I wanted my chatbot to be **scalable** without the need to individually add users to the system. This was a property that came with a client server. Additionally, I wanted a **centralised** system. This is because I did not want a client to have direct access to my database. The centralisation with a server allows for clients to connect, however requests will need to be accepted to retrieve a response.

Secondly, although denial of service is something which is a risk for both architecture styles. For this project, I deemed it as less of a risk with the client server because of the added security that this architecture style provides. Within a P2P network, if a virus is introduced it can easily spread across the whole system causing a denial of service. In a client server on the other hand, denial of service can present itself differently such as slow response times. If the system were to go down, it would also be **easier to restart the server**.

Lastly, many industry chatbots such as chatGPT or Discord run off a client server architecture style. As a result, I thought it would be better in terms of an industry standard practice to follow what is currently established.

Additional Design Decisions:

Although I started laying the groundworks of using a database for this system, the therapy bot logic is still running off a text file.

I decided on this for two reasons:

* Firstly, I wanted a be able to provide some way to add more responses or restore a database if it were to go down. Currently, whenever the system is running as an administrator on the server end has the option of add more responses and prompts to the database. This gets stored, and once a client connects the most recent data is pulled. This file is what is directly used by the chatbot logic.
* Secondly, I didn’t want the chatbot logic to directly access the database (only the server). One of the limits with SQLite came with multithreading because although the response was generated on the server side it all had to be in the same thread. Using a text file allows the therapy bot to access to data which is still the most recent version of the database without having to directly query it.

Work’s Cited:

**Project Proposal 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

**Coding Sources:**

[1] https://www.geeksforgeeks.org/simple-chat-room-using-python/

[2] https://www.biob.in/2018/04/simple-server-and-client-chat-using.html

[3] <https://www.tutorialspoint.com/creating-a-basic-hardcoded-chatbot-using-python-nltk>

[4] <https://realpython.com/build-a-chatbot-python-chatterbot/>