Design and implementation of a multithreaded dictionary server using a client-server architecture allowing concurrent clients to search meaning(s) of a word, add new words, and remove an existing word.

DISTRIBUTED SYSTEMS

Dictionary Server

Abhinav Kumar Singh
LMS USERNAME: abhinavkumar
ROLL NUMBER: 1037729

COMP90015: DISTRIBUTED SYSTEMS

Semester 2,2019

Assignment 1

Name: Abhinav Kumar Singh
LMS Username: abhinavkumar
Roll Number: 1037729

Problem Statement

Design a multithreaded dictionary server using a client-server architecture allowing concurrent clients to search meaning(s) of a word, add new words, and remove an existing word. The solution must explicitly use the concept of **Threads and Sockets** for establishing a concurrent and multi-threaded dictionary server.

Overview

Designing and organizing software to run distributed systems involves separating the entire functionality into two major parts: clients and servers. A client programs facilitates and services the end user by using functionalities that other programs provide. Client program interacts with the server on user's behalf by sending requests and receiving subsequent responses. Server has the sole purpose of servicing all the end users by responding to their requests.
In the case of multi-threaded dictionary server, a client will provide graphical user interface onto which a user can enter their request. After capturing the information entered by the user, client sends this information to the server by using TCP mechanism. The server processes the information received and sends the response back to client using the same mechanism. Client receives the response and displays the response onto the GUI.

Component Description

Client

Client programs have two important obligations -

1. Interacting with the user through an interface(GUI to be precise)
2. Interacting with the server on behalf of the user by using either TCP or UDP mechanism

Client programs require an interface for interacting with the end users. User interaction involves capturing user's request and displaying server's response. Dictionary client's graphical user interface is shown below-

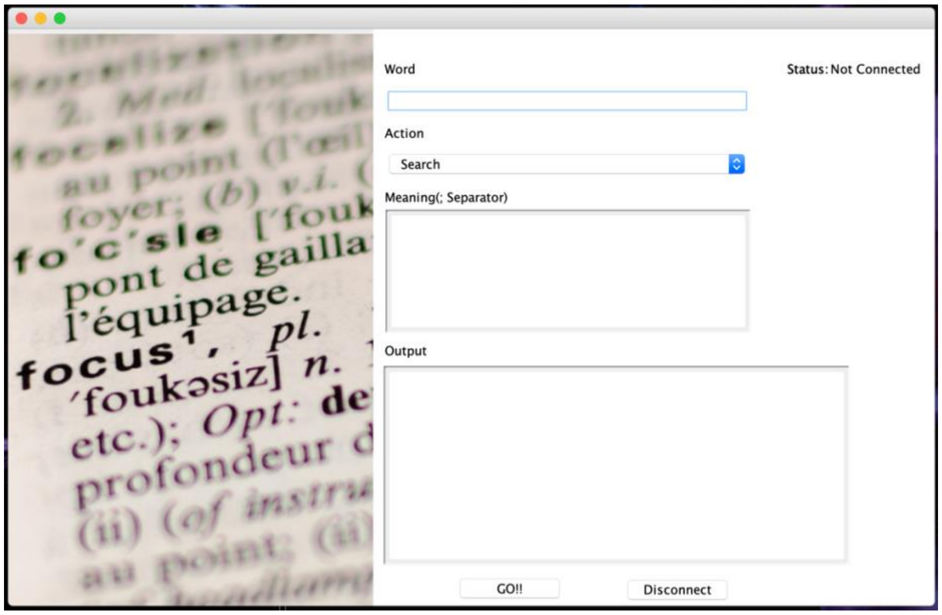
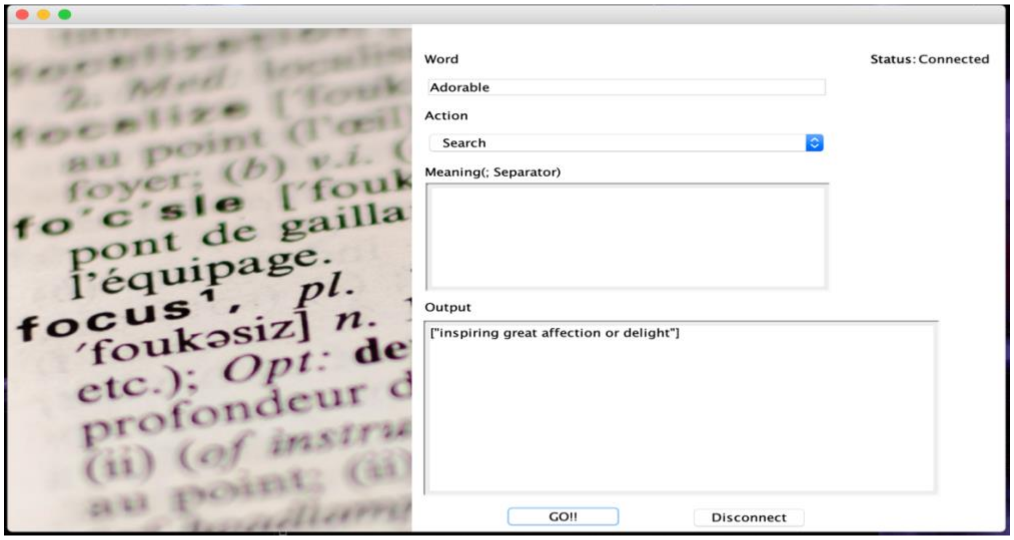


Figure 1: GUI

GUI captures user's request with the help of fields like **Word, Action and Meaning** where word field is used for capturing a word, action field is used for capturing the function to be performed and meaning field is used for capturing the meaning associated with the word. The output field is used for displaying the response form the server. **GO!!** button initiates the process of sending user's request which includes the data captured form the input fields described above to the server. **Disconnect** button is used for gracefully disconnecting client and server. Status field is used for displaying the status for the connection between client and the server.


Word
Adorable

Status: Connected

Action
Search

Meaning(; Separator)

Output
```
["inspiring great affection or delight"]
```

Figure 2: GUI in action

Client manages interaction with the server on behalf of user

Server

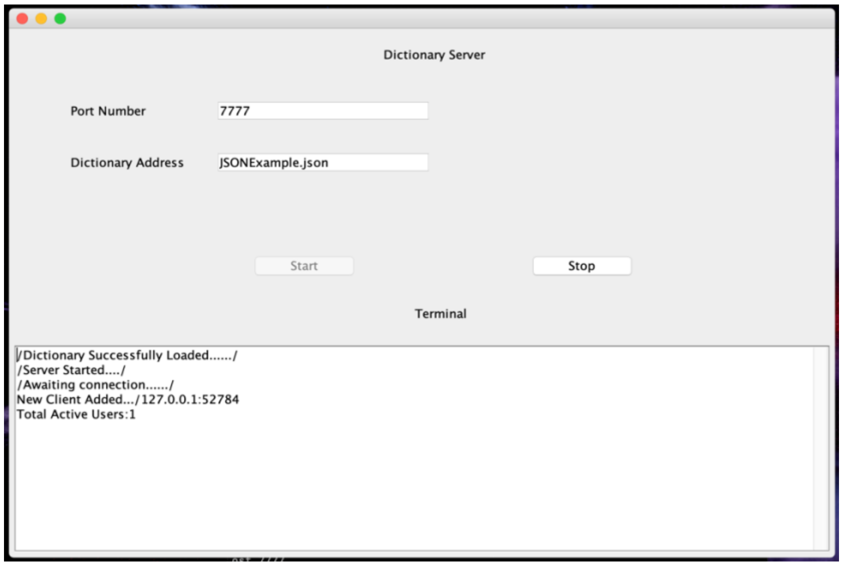
Server programs have following obligations-

1. Managing multiple client threads in a way that concurrency is achieved
2. Storing and extracting data from the data structure
3. Interacting with the server administrator using an interface

Dictionary sever uses **thread-per-connection** mechanism for supporting multiple clients and the reason for choosing this particular mechanism is explained in multithreading section below.

Dictionary server uses **JSON data format** for storing dictionary content permanently and **hash map** for storing dictionary contents dynamically.

Server's interaction with server administrator is managed through a graphical user interface. Server's GUI is shown below-


Dictionary Server

Port Number 7777

Dictionary Address JSONExample.json

```
/Dictionary Successfully Loaded....../
/Server Started..../
/Awaiting connection....../
New Client Added.../127.0.0.1:52784
Total Active Users:1
```

GUI captures port number and dictionary path using **Port Number** and **Dictionary Address** field respectively. Start button initiates the server program which starts listening on the port number provided by the administrator in the field above and establishes link with the dictionary stored in the address provide above for exchanging data.

Multithreading

Server manages multiple client requests by using multithreaded server architecture. The advantages of using multi-threaded server compared to a single-threaded server are mentioned below-

1. Server program spends minimum time outside the accept() call
2. Long client requests will not block the entire server
3. Concurrency is guaranteed

There are multiple mechanisms for supporting multithreaded server. Dictionary server uses thread per connection model and the reasons for using the same is mentioned below-

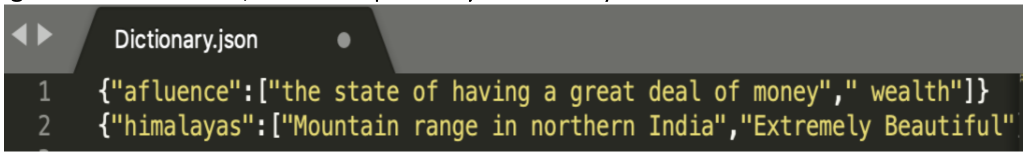
- It amortizes the cost of spawning the thread across multiple request generated from the same client process
- It is simple to implement and performs efficiently for small number of users performing multiple requests. As the number of users using the dictionary servers are small thread per connections suits the premise
- Main disadvantage of the thread per connection architecture is that threads remain active during inactivity on client's end, but this condition is mitigated as the users can disconnect by using disconnect button if they no longer require services provided by the server thereby killing the thread spawned for that particular user(connection) on the server side

Thread per connection architecture is only suited for the current premise where the number of users is limited as this architecture doesn't scale well. Its limitation is its inability to balance load under extreme conditions creating performance bottle necks.

Data Structures

Data structures used by the dictionary server are mentioned below-

- JSON dictionary file – Dictionary server uses JSON file structure for storing words and meanings permanently. JSON structure is preferred over other structures because JSON ensures fast data parsing on the server side, wide compatibility and is easy to handle.



Dictionary.json

```
{"afluence":["the state of having a great deal of money"," wealth"]}
{"himalayas":["Mountain range in northern India","Extremely Beautiful"
```

- Hash Map – Dictionary server uses hash map for storing the contents stored in JSON dictionary file dynamically. Hash map is preferred over other data structures because it guarantees fast information retrieval based on key and it doesn't allow two entries with the same key.

CLASS DESIGN

The entire class design can be separated into two parts –

1. Client class design – Client's class design includes two main classes clientGUI and TCPInteractiveClient. clientGUI class is responsible for GUI interface which helps in capturing user's request and displaying server's response.

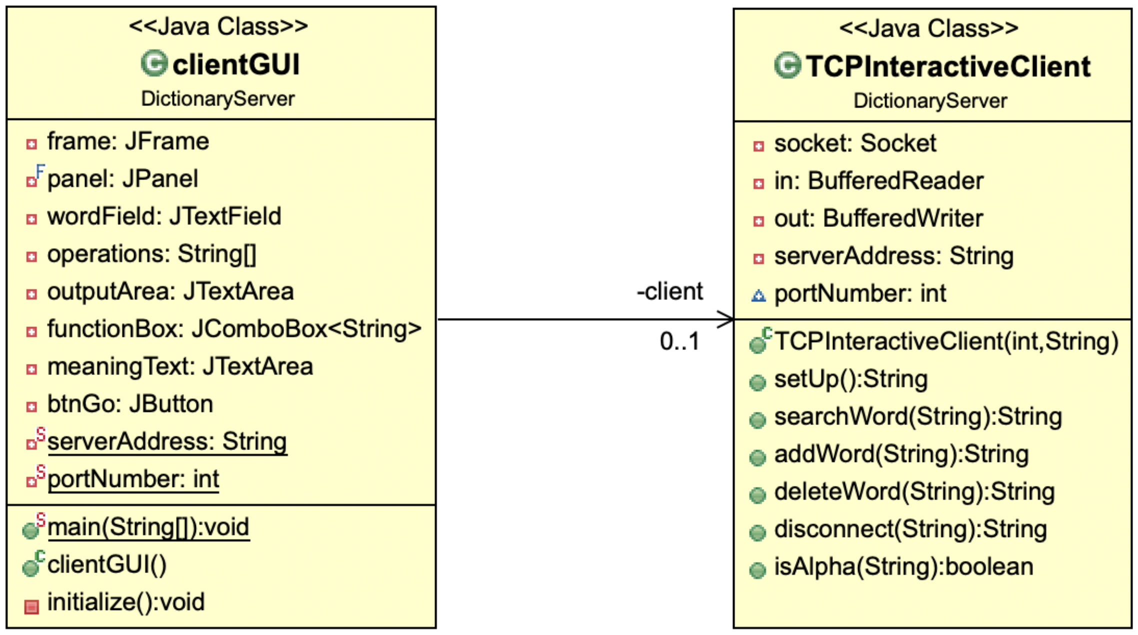


Figure 3: Client's UML Diagram

2. Server class design – Server's class design includes serverGUI class which is responsible for server's GUI, TCPInteractiveServer class for establish server socket and accepting incoming client's request, ClientHandler for multithreading clients and message list for controlling hash map operations.

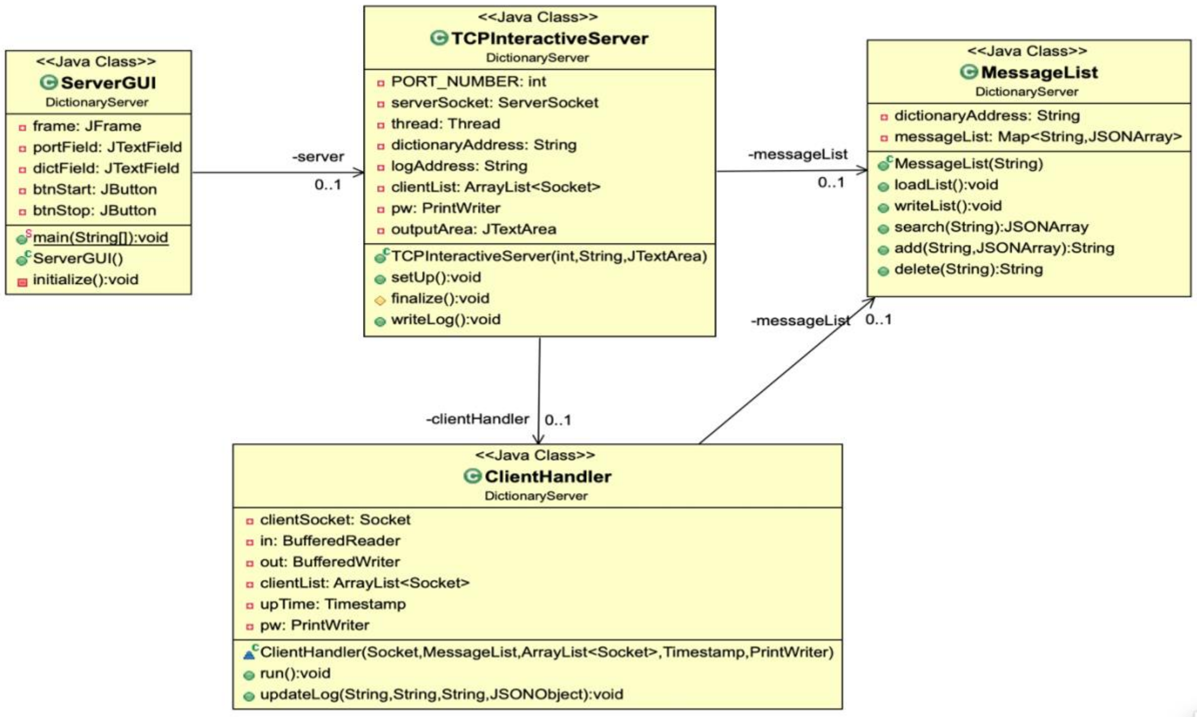


Figure 4: Server's UML Diagram

INTERACTION

TCP

TCP mechanism is preferred over UDP because of the following reasons –

1. TCP ensures reliability while sending messages whereas there is no reliability guaranteed while using UDP
2. TCP ensures ordering of messages as they are delivered therefore eliminating chances of receiving an out of order packets

3. TCP is connection oriented and suited for point to point data transmission where as UDP is connectionless and is generally used for broadcasting purpose.

Overall Flow

Server starts by capturing the information in form of port number and dictionary address. Server starts listening on the given port number and extracts the content stored in the dictionary and loads it into the hash map for dynamic processing of data. As client sends a connection request to the server on the same port, three-way handshake takes place and both server and client are connected. On the sever side, server creates a new thread for the client connected and starts awaiting for more clients.
Once connected client-server use request response cycle for data exchange.

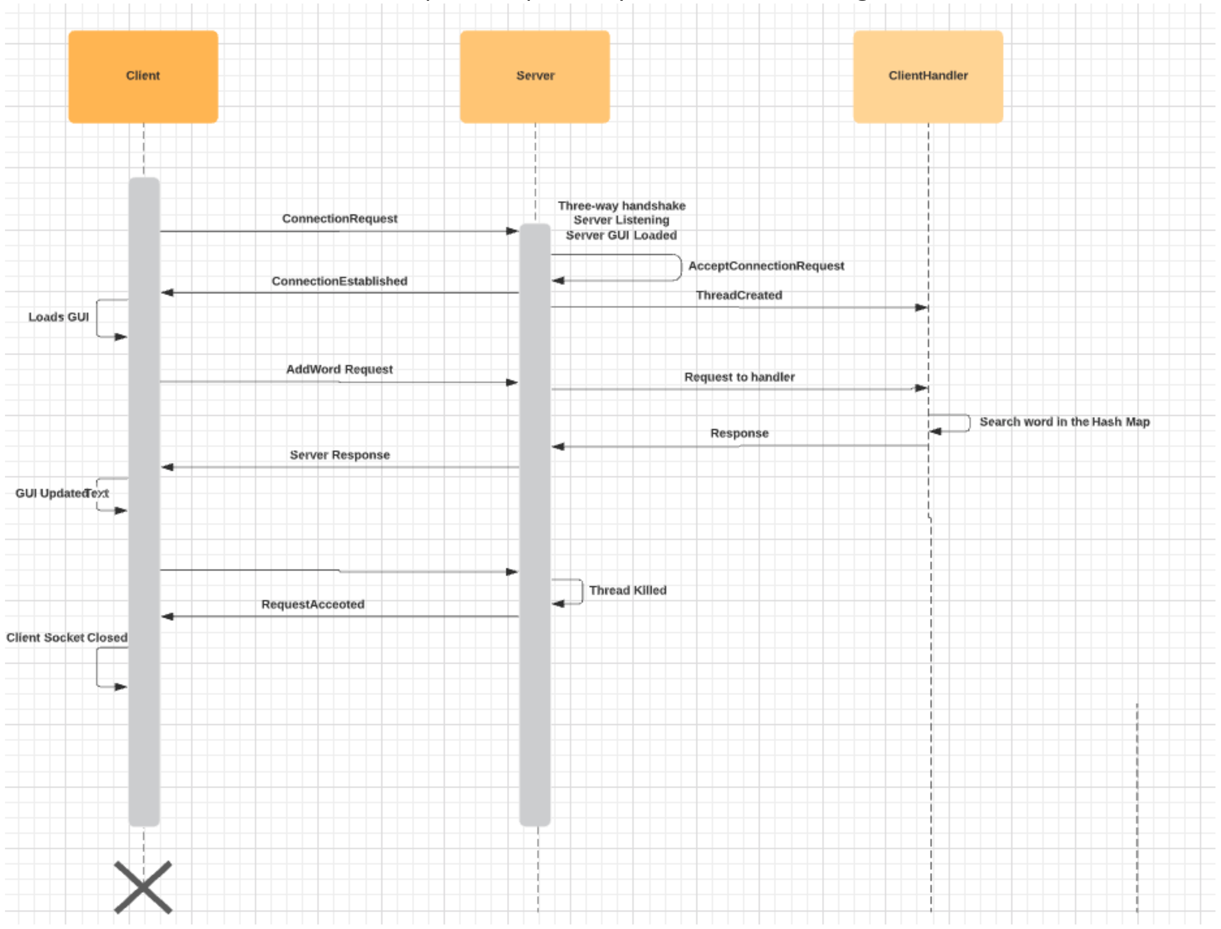
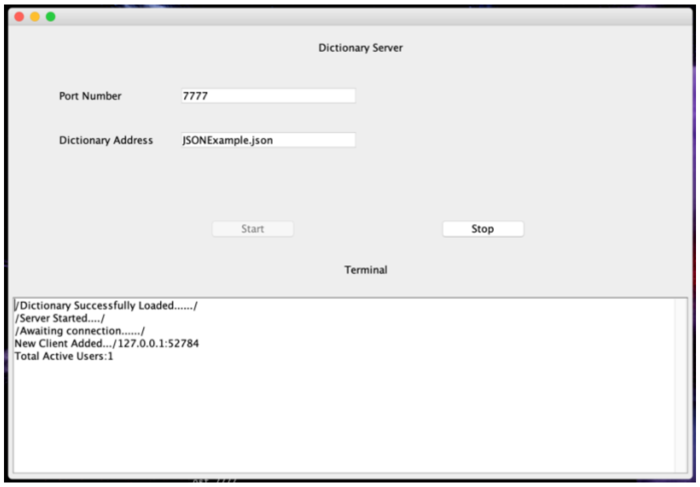


Figure 5: Sequence Diagram

Creativity

Server GUI

Server's interaction with server administrator is managed through a graphical user interface. GUI captures port number and dictionary path using **Port Number** and **Dictionary Address** field respectively. Start button initiates the server program which starts listening on the port number provided by the administrator in the field above and establishes link with the dictionary stored in the address provide above for exchanging data. Server GUI provides greater control to the server administrator by creating an interaction interface.

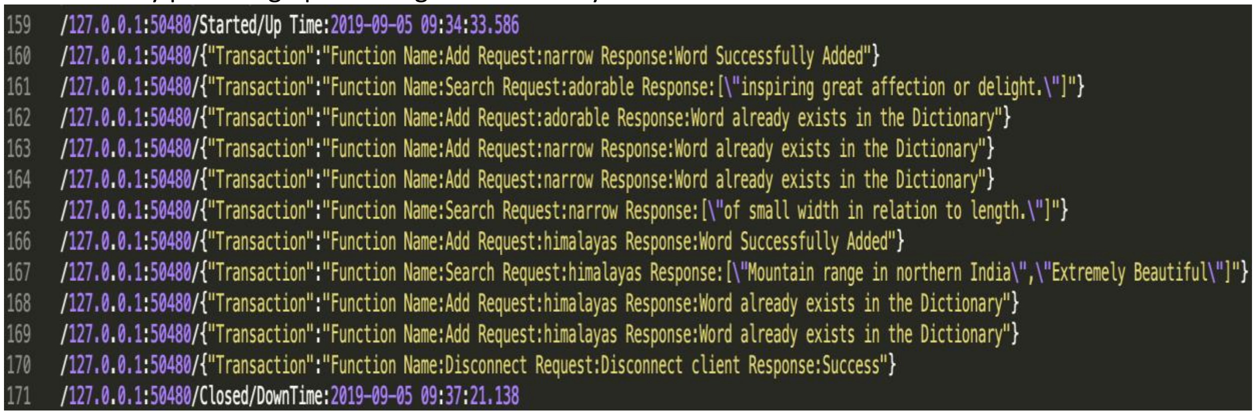


```
/Dictionary Successfully Loaded....../
/Server Started..../
/Awaiting connection....../
New Client Added.../127.0.0.1:52784
Total Active Users:1
```

Figure 6: Server GUI

Transaction Logs

All the transactions performed by multiple clients are logged in a transaction log on the server side in order to ensure greater compliance and data security. The server logs are generated on per request basis thereby providing option for greater scrutiny.

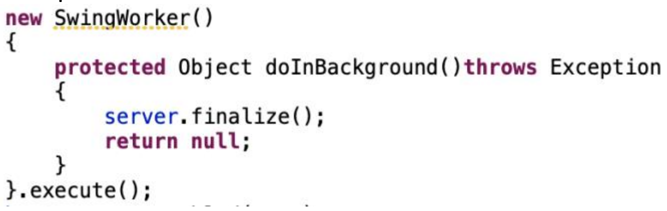


```
/127.0.0.1:50480/Started/Up Time:2019-09-05 09:34:33.586
/127.0.0.1:50480/{"Transaction":"Function Name:Add Request:narrow Response:Word Successfully Added"}
/127.0.0.1:50480/{"Transaction":"Function Name:Search Request:adorable Response:[\"inspiring great affection or delight.\"]"}
/127.0.0.1:50480/{"Transaction":"Function Name:Add Request:adorable Response:Word already exists in the Dictionary"}
/127.0.0.1:50480/{"Transaction":"Function Name:Add Request:narrow Response:Word already exists in the Dictionary"}
/127.0.0.1:50480/{"Transaction":"Function Name:Add Request:narrow Response:Word already exists in the Dictionary"}
/127.0.0.1:50480/{"Transaction":"Function Name:Search Request:narrow Response:[\"of small width in relation to length.\"]"}
/127.0.0.1:50480/{"Transaction":"Function Name:Add Request:himalayas Response:Word Successfully Added"}
/127.0.0.1:50480/{"Transaction":"Function Name:Search Request:himalayas Response:[\"Mountain range in northern India\",\"Extremely Beautiful\"]"}
/127.0.0.1:50480/{"Transaction":"Function Name:Add Request:himalayas Response:Word already exists in the Dictionary"}
/127.0.0.1:50480/{"Transaction":"Function Name:Add Request:himalayas Response:Word already exists in the Dictionary"}
/127.0.0.1:50480/{"Transaction":"Function Name:Disconnect Request:Disconnect client Response:Success"}
/127.0.0.1:50480/Closed/DownTime:2019-09-05 09:37:21.138
```

Figure 7: Transaction Logs

Swing workers

Swing workers are used while designing both server and client GUI. Swing workers prevent other program threads from interrupting or freezing the GUI thread by creating separate threads for any non-GUI operation performed in the GUI thread. Separate threads ensure that GUI will never get stuck even in case of errors in non-GUI operations.



```
new SwingWorker()
{
    protected Object doInBackground()throws Exception
    {
        server.finalize();
        return null;
    }
}.execute();
```

Figure 8: Swing Worker Stub

Conclusion

Multithreaded dictionary server was established with features and components mentioned above. Dictionary server satisfies all the requirements mentioned in the problem statement and performs as expected.