**Internet Technology Report**

**Name** - Tuhin saha

**Roll** - 002110501087

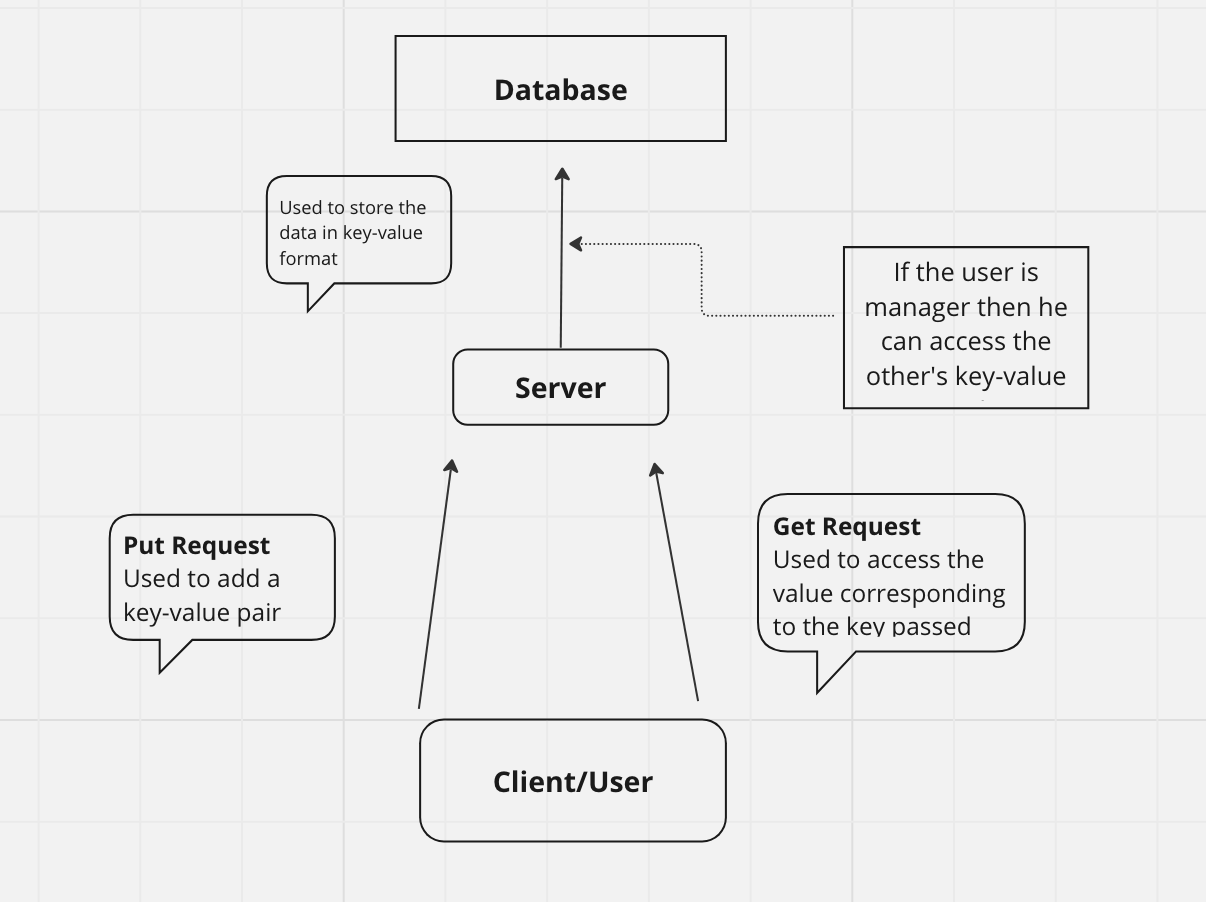
**Department** - JUBCSE UG-III (A3)

**Assignment -I**

**Problem Statement -** Implement a TCP-based key-value store. The server implements the key-value store and clients make use of it. The server must accept clients’ connections and serve their requests for ‘get’ and ‘put’ key value pairs. All key-value pairs should be stored by the server only in memory. Keys and values are strings. The client accepts a variable no of command line arguments where the first argument is the server hostname followed by port no. It should be followed by any sequence of “get <key>” and/or “put <key> <value>”. ./client 192.168.124.5 5555 put city Kolkata put country India get country get city get Institute India Kolkata <blank> The server should be running on a TCP port. The server should support multiple clients and maintain their key-value stores separately. Implement authorization so that only few clients having the role “manager” can access other’s key-valuestores. A user is assigned the “guest” role by default. The server can upgrade a “guest” user to a “manager” user.

**Theory -** This is a TCP-based key-value store system where clients can connect to a server to store and retrieve string key-value pairs. The server, running on a specified port, supports multiple clients and maintains separate key-value stores for each client. Authorization is implemented to allow only "manager" clients to access other clients' stores, while "guest" clients have limited access, with the server able to upgrade "guest" clients to "manager" status.

**Flowchart**

****

**Code with Explanation**

Client.py

import socket

import sys

print("Command line args")

print(sys.argv)

in\_put=sys.argv

password="1234"

client\_socket = socket.socket()

client\_socket.connect((f"{in\_put[1]}", int(in\_put[2])))

print("Connected to the server")

n=len(in\_put)

i=3

isManager=False

while i<n:

if(in\_put[i]=="put"):

query="put"+"."+in\_put[i+1]+"."+in\_put[i+2]

if(isManager):

query=query+".manager"

client\_socket.send(query.encode())

# print("Send put request")

client\_socket.recv(1024).decode()

i=i+3

elif in\_put[i]=='get':

query="get"+"."+in\_put[i+1]

if(isManager):

query=query+".manager"

client\_socket.send(query.encode())

# print("Send get request")

message=client\_socket.recv(1024).decode()

print(message)

i=i+2

elif in\_put[i]=='promote':

recvPass=input("Enter the password:\n")

if(recvPass==password):

isManager=True

print("Promoted to manager")

else:

print("Wrong password")

i=i+1

client\_socket.close()

TCP-based key-value store system where clients can connect to a server to store and retrieve string key-value pairs with socket.

**Put -** When the 'put' command is encountered, the script constructs a query with the provided key-value pair. If the client is promoted to a manager, '.manager' is appended to the query. The query is then sent to the server, and the script awaits a response.

**Get -**  For the 'get' command, the script forms a query with the specified key. If the client is a manager, '.manager' is added to the query. This query is sent to the server, and the script prints the received response.

**Promote -**  When 'promote' is invoked, the script prompts the user to enter a password. If the password matches the predefined one, the client is promoted to a manager. Otherwise, an error message is displayed.

Server.py

import socket

import threading

server\_socket=socket.socket()

server\_socket.bind(("localhost", 8000))

server\_socket.listen(10)

sem=threading.Semaphore(1)

db=dict()#key=token value=collection of key-value pair

def handleClient(client\_socket,token):

print(f"Thread start for {token}")

if(token not in db):

sem.acquire()

db[token]=dict()

sem.release()

while True:

query=client\_socket.recv(1024).decode()

# print(f"query received {query}")

if not query:

#connection closed

client\_socket.close()

break

query=query.split(".")

if(query[0]=="put"):

key=query[1]

val=query[2]

# if (key not in db[token]):

sem.acquire()

db[token][key]=val

sem.release()

client\_socket.send("ack".encode())

elif (query[0]=="get"):

key=query[1]

if(len(query)==2):

#guest

if key in db[token]:

client\_socket.send(db[token][key].encode())

else:

client\_socket.send("<blank>".encode())

else:

#manage

data=list()

for it in db:

if(key in db[it]):

data.append((it,db[it][key]))

if len(data)!=0:

client\_socket.send(str(data).encode())

else:

client\_socket.send("<blank>".encode())

while True:

print("\n\nWaiting for connections....\n\n")

client\_socket,client\_address=server\_socket.accept()

print(f"connected to client {client\_address}")

#unique toke for each connection

token=client\_address[0]+":"+str(client\_address[1])

# print(f"token {token}")

threading.Thread(target=handleClient,args=(client\_socket,token)).start()

The socket module provides access to the BSD socket interface for network communication.

The threading module is used for creating and managing threads for concurrent execution. It creates a socket object (server\_socket) and binds it to the localhost on port 8000, then starts listening for incoming connections with a maximum backlog of 10 connections. It initializes a semaphore (sem) to control access to the shared data structure and a dictionary (DB) to store key-value pairs associated with client tokens.

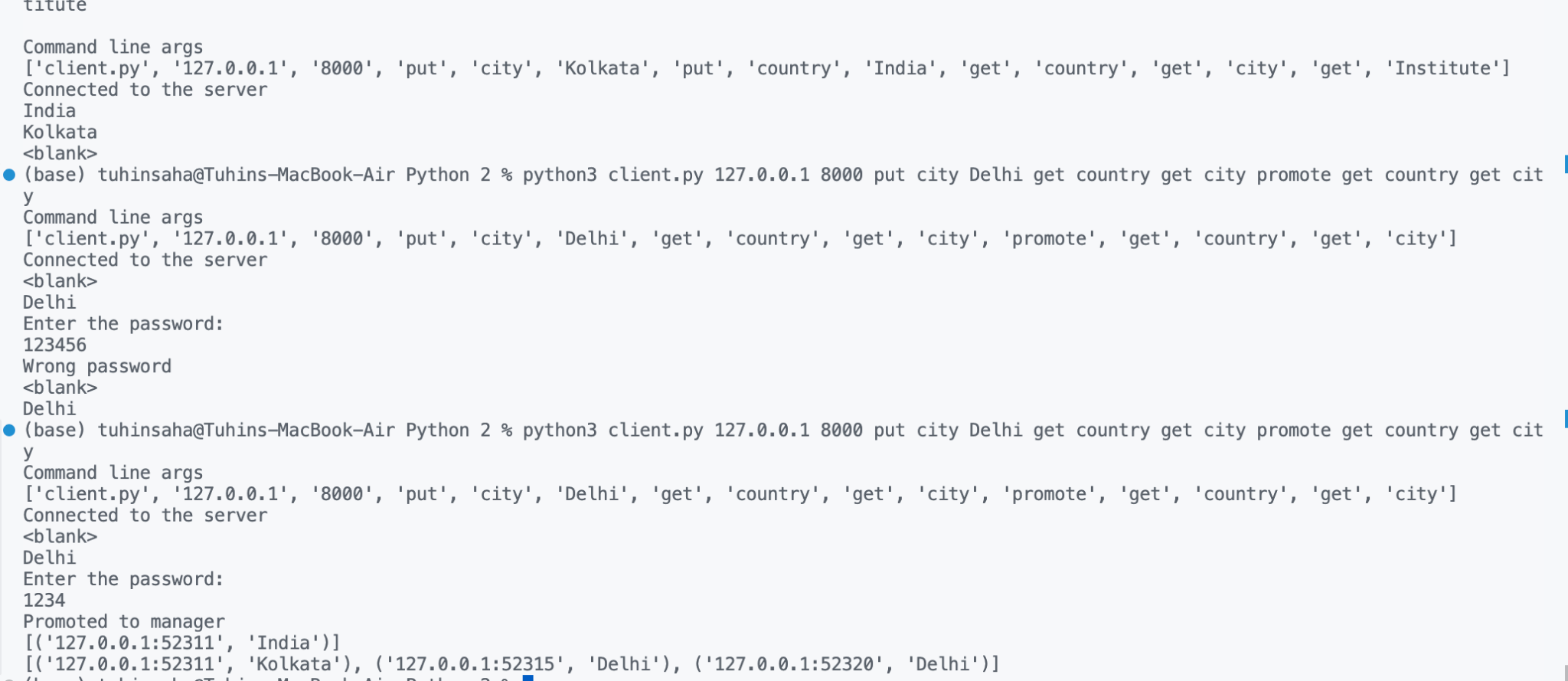
**Loop -** It continuously listens for incoming client connections. Once a connection is established, it retrieves the client socket object and client address.

* It generates a unique token for the client based on its address.
* It spawns a new thread to handle the client connection using the handleClient function.

**Handling Client requests -** The handleClient function starts by checking if the client token exists in the database (db). If not, it creates an entry for the client.

* It then enters a loop to continuously receive queries from the client.
* Upon receiving a query, it splits it into components to determine the action (e.g., 'put' or 'get').
* For 'put' queries, it updates the database with the provided key-value pair.
* For 'get' queries, it retrieves data from the database based on the key and sends it back to the client.

**Output Screenshot**

****