

ASSIGNMENT 1

SIMPLE KV STORE

ARCHITECTURE:

- 1 TCP/IP server built using python3 which stores the data and interacts with clients
- 1 Database, which is a collection of files (1 file per key) stored in the 'data' folder by the server
- Client processes that initiate set and get requests to the server

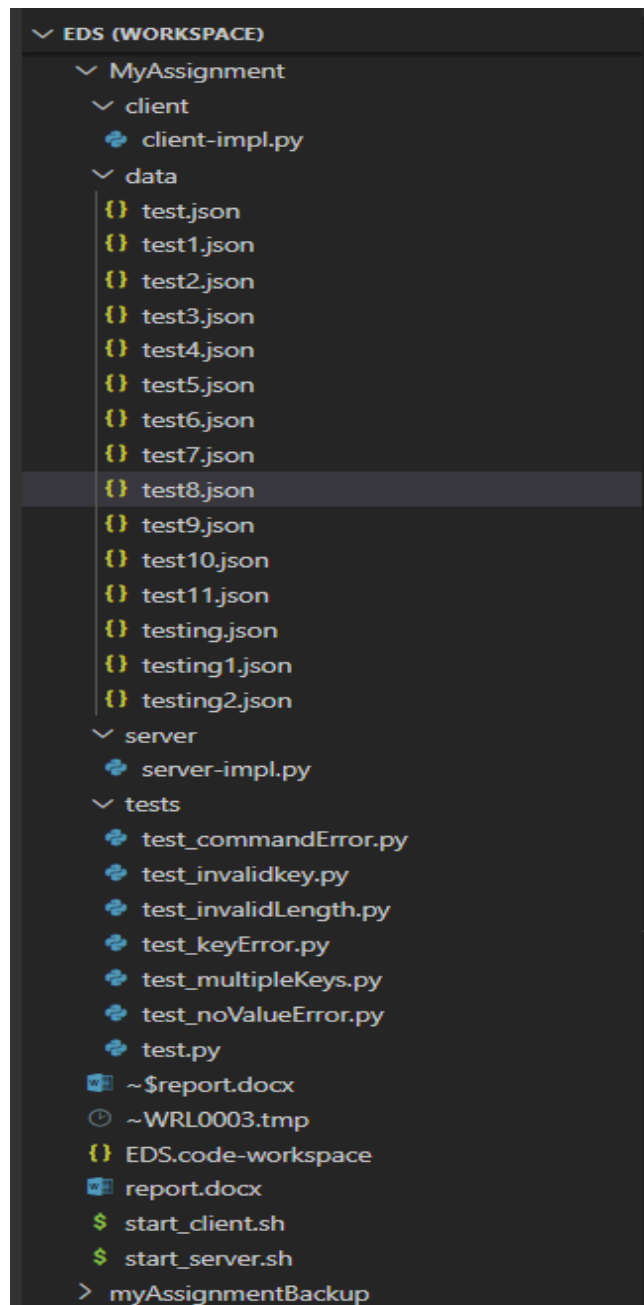


Figure 1 – Folder Structure

ASSIGNMENT 1

SIMPLE KV STORE

IMPLEMENTATION:

1. TCP- socket server:

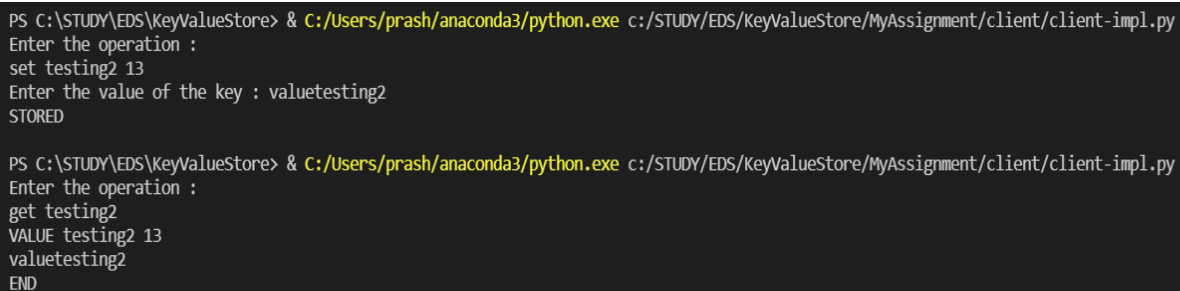
The server was implemented using the socket library offered by python. Upon creation of the server socket, it is associated with a specific network interface and port number. Once it accepts a connection from a client, it returns a new socket object representing the connection and a tuple holding the address of the client i.e (host, port). This socket is then used to communicate with the client. Based on the client request, the set or get operation is performed and a response for the request is sent back to the client.

- SET COMMAND:

This stores the value for later retrieval.

```
set <key> <value-size-bytes> \r\n
<value> \r\n
```

The set command is whitespace delimited. It takes the key to be stored, the length of the value associated with the key and the value itself as the arguments. The server responds with either "STORED\r\n", or "NOT-STORED\r\n" depending on whether the key was stored or not.



```
PS C:\STUDY\EDS\KeyValueStore> & C:/Users/prash/anaconda3/python.exe c:/STUDY/EDS/KeyValueStore/MyAssignment/client/client-impl.py
Enter the operation :
set testing2 13
Enter the value of the key : valuetesting2
STORED

PS C:\STUDY\EDS\KeyValueStore> & C:/Users/prash/anaconda3/python.exe c:/STUDY/EDS/KeyValueStore/MyAssignment/client/client-impl.py
Enter the operation :
get testing2
VALUE testing2 13
valuetesting2
END
```

Figure 2 – Successful set operation

- GET COMMAND:

This will fetch the data corresponding to the key and return it to the client

```
get <key>\r\n
```

The get command gets the value associated with the given key from KV store. It accepts the key name as the argument.

ASSIGNMENT 1

SIMPLE KV STORE

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\STUDY\EDS\KeyValueStore> & C:/Users/prash/anaconda3/python.exe c:/STUDY/EDS/KeyValueStore/MyAssignment/client/client-impl.py
Enter the operation :
get testing
VALUE testing 12
valuetesting
END
```

Figure 3 – Successful get operation

```
yS C:\STUDY\EDS\KeyValueStore\MyAssignment>
get test testing
VALUE test 4
test

VALUE testing 12
valuetesting

END
```

Figure 4 – Getting values of multiple keys

2. Client:

The client program connects to the server and makes get or set requests. It sends a command line/data block and receives a response from the server which indicates the success or failure of request made.

PROTOCOL:

The data stored by the server is identified by the key. For the set operation, there are few rules that should be followed for setting the key in the KV store. According to the Memcached protocol, a key should be a text string with a limit on its length at 250 characters. The key should not include any whitespace or any special characters. The below function has been implemented to check if the key is valid or not:

```
PS C:\STUDY\EDS\KeyValueStore> & C:/Users/prash/anaconda3/python.exe c:/STUDY/EDS/KeyValueStore/MyAssignment/client/client-impl.py
Enter the operation :
set key-Error 5
Enter the value of the key : value
CLIENT_ERROR INVALID KEY
```

Figure 5 – Invalid Key Error

ASSIGNMENT 1

SIMPLE KV STORE

TEST CASES:

These are the associated test files with the KV store:

1. test_commandError.py – This test file makes an incorrect client request with a wrong command protocol. In the test file, a client request with keyword edit is made to the server, which is not included in the protocol. The server returns an error.

```
PS C:\STUDY\EDS\KeyValueStore> & C:/Users/prash/anaconda3/python.exe c:/STUDY/EDS/KeyValueStore/MyAssignment/client/client-impl.py
Enter the operation :
edit commandError 12
ERROR
```

Figure 6 – Command Error

2. test_keyError.py – This test file makes a client request with an invalid key as it includes a special character in the key and throws an error as it does not follow the protocol.

```
PS C:\STUDY\EDS\KeyValueStore> & C:/Users/prash/anaconda3/python.exe c:/STUDY/EDS/KeyValueStore/MyAssignment/client/client-impl.py
Enter the operation :
set key-Error 5
Enter the value of the key : value
CLIENT_ERROR INVALID KEY
```

Figure 7 – Invalid Key Error

3. test_invalidLength.py – This test file makes a client request with the length of value not matching with the specified length.

```
PS C:\STUDY\EDS\KeyValueStore> & C:/Users/prash/anaconda3/python.exe c:/STUDY/EDS/KeyValueStore/MyAssignment/client/client-impl.py
Enter the operation :
set invalidLength 20
Enter the value of the key : valueinvalidLength
CLIENT_ERROR : THE LENGTH OF THE VALUE DO NOT MATCH.
```

Figure 8 – Invalid Value Length Error

4. test_noValueError.py – This test file makes a client request with just 2 parameters passed with the set command. Here the value of the key is not passed hence an error from the server is received.

ASSIGNMENT 1

SIMPLE KV STORE

```
PS C:\STUDY\EDS\KeyValueStore> & C:/Users/prash/anaconda3/python.exe c:/STUDY/EDS/KeyValueStore/MyAssignment/client/client-impl.py
Enter the operation :
set testing3 13
Enter the value of the key :
CLIENT_ERROR PROTOCOL WAS NOT FOLLOWED
```

Figure 9 – No Value Error in set operation

LIMITATIONS:

1. The current implementation of the server has a single process concurrency which allows easy caching and database access but has potential disadvantages like blockage of services when busy.
2. In-memory data structure store like Redis can be implemented for better performance and improve the scalability.
3. Input files from the system can be accepted
4. Better error handling and validation process
5. Memcached protocols like flags and expiration time can be implemented to better enhance the functionalities of the KV store