Distributed System Assignment 1 Report

You should write a report describing your system and discussing your design choices. Your report should include:

* The problem context in which the assignment has been given.
* A brief description of the components of the system.
* An overall class design and an interaction diagram.
* A critical analysis of the work done followed by the conclusions.

Please mind that the report is a **WRITTEN** document, do not put only graphs. A report without any descriptive text addressing the problem, architecture, protocols, and the analysis of the work done will not be considered valid.

The length of the report is not fixed. A good report is auto-consistent and contains all the required information for understanding and evaluating the work done. Given the level of complexity of the assignment, a report in the range of 4 to 6 pages is reasonable. Please mind that the length of the report is simply a guideline to help you in avoiding writing an extremely long or incomplete report.

It is important to put your details (name, surname, student id) in:

* The head page of the report.
* As a header in each of the files of the software project.

This will help to avoid any mistakes in locating the assignment and its components on both sides.

**Introduction:**

In this assignment, we are asked to build a multi-threaded dictionary server. This multi-threaded dictionary server needs to be able to handle multiple clients’ connection at the same time. And it needs to allow clients to search the meaning of a word, add a new word, update the meaning of an existing word and remove an existing word.

**Architecture Used:**

I have implemented the multi-threaded dictionary server using client-server architecture. This means that multiple clients are able to be connected to the same multi-threaded dictionary server at the same time.

The thread-per-connection approach is used when designing the system. That is, whenever a new client connects to the server, a persistent connection is established between them. The client is able to send multiple requests through the same connection. The connection keeps alive even if the client is not sending any requests.

**Implementation:**

The system consists of two parts: the client part and the server part.

Client:

When running the client program, a GUI will pop up and asking for the dictionary server IP address and the server port. The client will send out a TCP connection to the server. Error message will be prompted if the server IP address or the server port is not correct. Once the TCP connection is established between client and server, an input stream and output stream will be created, and the dictionary GUI will show up. The client will be able to query the meaning of a word, add a new word, remove an existing word as well as update the meaning of an existing word. The client and server communicate by sending json files. Error messages will be prompted if any of the issues occur during any of the operations.

Server:

When running the server program, a GUI will pop up and asking for the port number to listen and the dictionary file. Error message will be prompted if the port number is in use or can’t find the dictionary file. Once a listening port has been established, a thread will be created to continuously listen for incoming connection requests on that port. The GUI will then display the port number that the server is currently listening. When a client sends a connection request, the server will automatically assign the client a new port for persistent connection, the input and the output stream will also be created. All of the future requests coming from that client will be handled by a new thread. This means that the server can handle multiple clients’ connections at the same time. Due to the fact that each client will occupy one port and there is a limited number of ports available, this means that the number of clients that can be connected to the server at the same time will be limited to the number of ports available.

**UML class diagram:**

**Interaction diagram:**