# **Key-Value (KV) database**

## Name: Ioanna Mourtzaki cs22200021

## Course: Big Data Management (M111)

## Professor: Alexandros Ntoulas

## Semester: Fall 2022

In this project we have created a simple version of a distributed, fault-tolerant, Key-Value (KV) database (or store), with a few tweaks. This database implementation was fully coded in Python and was run and tested on a Python environment using the Visual Code IDE. The “Mourtzaki\_Ioanna\_Project.zip” file also contains the source code, the input/output files (.txt) that are part of the project ana a README.md file with running details.

### **Data Creation (genData.py)**

In this of the project, we generate data that will be loaded to the key value store.

The assumptions made at this part of the project are the following:

* In the “keyFile” file there are different keys (name, age, height etc.), while the supported types are string, int and float.
* The output data are stored in a file called “dataToIndex” and their format is similar to the JSON format, as shown in the example below. Every key (high-level and the other ones) and also every value (string, int, float) are enclosed in ‘..’.
  + *'key0':{'name': {'age': '76', 'address': 'VM'}}*
* The high-level keys are named keyX, where X is an increasing number from 0 to n (number of lines input given from the user). So, in the “dataToIndex” file the high-level keys are key0, key1,.., keyN.
* In order to create the payload with the key-values data, we have used recursion. Specifically, we randomly decide the nesting level we will apply, based on the maximum level of nesting given from the user. In addition, we randomly decide the insides key number of the high-level key, based on the maximum number of keys given from the user. Then, for every inside key we apply recursion, as many times as the nesting level.
* We use the maximum string length given from the user to create the randomly created string value. All string variables consist of ASCII letters (both lower and upper case). Also, the maximum integer value is 100 and the maximum float value is 200 (with two decimal digits). These limits have been set based on the logic of the keys identified in the “keyFile” file.
* When the output data is successfully created, a “Data was successfully generated” message will be shown to the user.
* When an argument is missing from the user’s input, a message “Some arguments are missing. Please provide the required info in this order:….” will be shown to the user.

### **Key Value store**

In this part of the project, we create a KV Client that will be accepting queries and will be redirecting requests to the KV servers, collecting the results, and presenting them to the user and a KV Server that will be storing the actual data and will be handling the queries coming from the client.

### KV Client

The assumptions made at this part of the project are the following:

* The “serverFile” is a file containing server IPs and their respective ports that will be listening for queries and indexing commands. For example, “123.123.12.12 8000”. The k value is the replication factor, i.e., how many different servers will have the same replicated data. First, we check if the servers provided are less than the 'k' number. If so, the program must exit.
* We establish connection with the server using the socket connection, providing the IP address and the port given.
* We inform the user with suitable messages when the index of the servers is started and finished.
* Every time we check if the servers are down before answering the user query.
* When user is providing a new query, if their input is empty, then a suitable message is shown, and the user has to provide another query. The same happens when user provides only a command without key specified, for example just “GET”, or a command that is not supported. The supported ones are : GET, QUERY, COMPUTE, DELETE, EXIT.
* The “EXIT” command is used in order to exit the program.
* The commands given from the user can be either capitalize or not. For example, both “GET” and “get” formats are acceptable.
* After server indexing with the data, the user cannot perform any “PUT” requests. So, if they try so, a “PUT requests are not supported after server indexing” message will be shown.
* The user is not able to perform “DELETE” actions, if at least one server is down, because this will lead to differences between the servers. So, if this happens, a “DELETE request cannot be executed because at least one server is down” message will be shown.
* The “socket.recv” parameter has been set to 4098, so the socket will be able to read at most 4098 bytes of the incoming data.
* Every time the client communicates with the server from any reason (first indexing or responding to requests), the server response will be shown to the user. This response would be “OK” with some data, or “ERROR” with the error message. It is depended on each case.
* When an argument is missing from the user’s input, a message “Some arguments are missing. Please provide the required info in this order:….” will be shown to the user.

### KV Server

At this part of the project, we can run from different terminals the “connection” command, in order to set up multiple servers simultaneously.

The assumptions made at this part of the project are the following:

* We establish a server connection based on the IP address and port given from the user. We inform the user for this process, with messages like "Socket successfully created!" and "The socket is waiting..(for requests)".
* When the indexing of the server is done, a message like “OK” or “ERROR” will be shown to the user, based on the indexing status.
* The “socket.recv” parameter has been set to 4098, so the socket will be able to read at most 4098 bytes of the incoming data.
* The “socket.listen” parameter has been set to 5.