## Executive Summary

While in the last project, we learnt about basics of TCP and UDP protocol and its implementation in Java, in the project, we were supposed to implement Remote Procedure Calls in Java to provide support for executing methods on remote systems. Also, we were required to provide support for the multithreaded server which can handle multiple client requests at the same time. Making a server multithreaded means the data store could be accessed by multiple threads at the same time and there should not be any error in that. For implementing multithreading, we can use thread management, such as using thread pools, and ensuring proper synchronization to handle mutual exclusion. By completing these tasks, the application will be capable of handling concurrent PUT, GET, and DELETE operations from multiple client instances simultaneously, thus enhancing its scalability and responsiveness.

While carrying out the project, there were multiple challenges and learning along the way. First, I chosed to use Java RMI to implement RPC since it was the most basic and easy to understand to provide support for distributed objects in Java, allowing objects to invoke methods on remote objects with exact same syntac as for local invocation. Using RMI, an object server exports a remote object and registers it with a directory service. The object provides remote methods, which can be invoked in client programs. For making server multithreaded, I used **ConcurrentHashMap** to ensure thread safety, preventing concurrent modification exception, and simplifying concurrent programming for handling multiple client requests simultaneously in a multi-threaded server. I also learned about various different ways through which we can implement RPC in Java, besides RMI, like Apache Thrift. And also, for I got to learn a lot about thread safe operations of ConcurrentHashmap and how does it actually handles concurrent requests without throwing error.

One thing that can be improved for future assignments is more clarification on testing part for the multithreading. Since, I implemented RMI for RPC, there was very little testing that I could do for multiple threads, since this API handles it internally.