<u>Executive Summary</u>

For Project1, I had to learn about TCP and UDP and understand how they work differently. The assignment also taught me how servers and clients communicate, no matter which protocol is used. By organizing and simplifying the code, I figured out how to write clear and easy-to-understand code. Since the project needed a single-threaded server, I focused on dealing with one request at a time. I also handled different situations like dealing with a request that's not quite right and making sure the client knows when to stop waiting for the server. Keeping logs for both the client and server made sure all messages were saved in a way that's easy for people to read and understand the message status.

While carrying out this project, I faced multiple challenges, for example, while implementing timeout, earlier we tried implementing a thread for each request, but later I realized that java provides “setSoConnect” functionality to assign timeout to the socket. Other than that, I learned a lot about java.net and java.io packages and exceptions that can be raised in certain scenarios. Also, handling packets in TCP and UDP was very different, and we had to handle malformed/out of order packets in UDP using checksums as request IDs which was not required in TCP, so handling the requests in different protocols was a challenge for me. Regarding logging, initially we thought of using external logging API like log4j, but then we thought of creating our own logs manually, but I eventually came across java's own logging package and used that. Also, while processing getAll request in UDP, I couldn't transmit all the data in single packet, so I divided the data in key,value pairs, and added as many pairs possible in 1 packet and sent all data in multiple packets. I also maintained a list of negative edge cases to make sure our client and server are robust to all major failures. Also, I added documentation and modularized the code to make it readable and scalable.

One use case of such a single threaded server client application could be a “web server” which can be used to handle HTTP requests. One application where such a web server can be used is a “single user gaming systems” where there is one server to handle the user requests to ensure timely and accurate responses.