**TRD/BRD Documentation**

**Cover Page**

* **Title:** Server Application Documentation
* **Author:** Kennedy Owiro
* **Version:** 3.0
* **Date:** January 13, 2025

**Business Requirements Document (BRD)**

**1. Introduction**

**Purpose of the Document**

This document outlines the business objectives, scope, and use cases for the server application. It is intended to align stakeholders on the goals and value of the project.

**Project Overview**

The server application provides secure, efficient, and scalable text search capabilities for client queries. It ensures high performance, robust security, and extensive configurability to meet diverse operational requirements.

**Intended Audience**

* Business stakeholders.
* Operations teams.
* Software evaluators.

**2. Objectives and Goals**

* Deliver a robust, multi-threaded server application capable of handling unlimited concurrent client connections.
* Provide secure communication channels using SSL/TLS.
* Implement efficient file searching with optional dynamic reloading.
* Achieve optimal performance, ensuring minimal query latency and resource usage.

**3. Scope**

**In-Scope**

* Unlimited concurrent client connections.
* Real-time and cached file search capabilities.
* Configurable features such as rate-limiting and secure communication.
* Logging of client activity and server performance.

**Out-of-Scope**

* GUI interfaces.
* Compatibility with non-Linux environments.

**4. Use Cases and Scenarios**

**Use Case 1: Real-Time Error Monitoring**

**Actor:** Operations Team  
**Scenario:** Monitor server logs in real-time for error patterns using client queries.

**Use Case 2: Secure Data Query**

**Actor:** Application Clients  
**Scenario:** Securely query sensitive data over encrypted channels.

**Use Case 3: Rate-Limiting Abusive Users**

**Actor:** Server  
**Scenario:** Restrict the frequency of requests from specific IPs to prevent abuse.

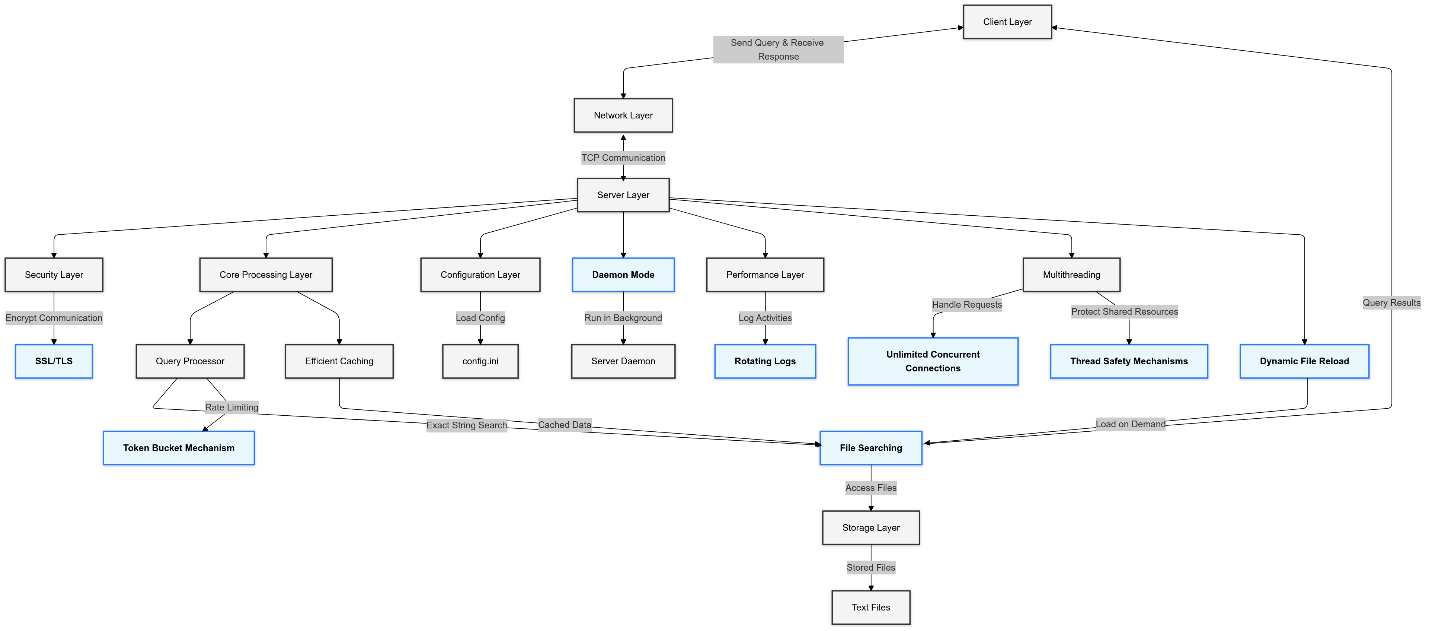
**Technical Requirements Document (TRD)**

**1. System Architecture**

**Overview**

The server application is built with a client-server model. Clients send text-based queries to the server, which searches a preloaded file and responds with match results.

**Architecture Diagram**



**2. Design Details**

**Multithreading**

The server uses Python’s threading module to handle concurrent connections efficiently.

**File Handling**

* **Dynamic Reloading:** If REREAD\_ON\_QUERY=True, the server re-reads the file for every query.
* **Caching:** If REREAD\_ON\_QUERY=False, the file contents are cached in memory to minimize disk I/O.

**Rate-Limiting**

Implements a Token Bucket algorithm to control client request rates.

**Security Features**

* SSL/TLS for encrypted communication.
* Payload validation to prevent buffer overflows.

**3. API Documentation**

**Endpoint**

* **Path:** /search
* **Method:** TCP-based communication.
* **Request Payload:** Plain text (UTF-8).
* **Response Payload:**
  + STRING EXISTS\n if a full line matches the query.
  + STRING NOT FOUND\n otherwise.

**Example Communication Flow**

1. Client sends a query string.
2. Server searches the file.
3. Server responds with match results.

**4. Configuration**

**Default/Example config.ini**

[server]

host = 127.0.0.1

port = 44444

ssl = false

cert\_file = server.crt

key\_file = server.key

REREAD\_ON\_QUERY = false

linuxpath = /mnt/d/Algorithmic\_Sciences/Revised\_Intro\_Task\_v3/src/data/200k.txt

max\_payload = 1024

token\_bucket\_capacity = 10000

token\_bucket\_fill\_rate = 1000

pid\_file = server\_daemon.pid

[paths]

config\_dir = config

log\_dir = logs

cert\_dir = certs

data\_dir = data

file\_path = src/data/200k.txt

config\_path = ${config\_dir}/config.ini

pid\_file = ${log\_dir}/server.pid

log\_file = ${log\_dir}/server.log

**5. Performance Summary**

**1. Execution Time Summary (SSL=True)**

* **Dynamic Files (REREAD\_ON\_QUERY=True):**
  + Average execution time: ~16-26ms per query for 250,000-row files.
  + Round-trip execution time: ~20-32ms.
* **Static Files (REREAD\_ON\_QUERY=False):**
  + Average execution time: ~0.001–0.002ms per query for 250,000-row files.
  + Round-trip execution time: ~0.3–0.6ms.

**2. Query Per Second (QPS) Performance**

* **Dynamic Files (REREAD\_ON\_QUERY=True):**
  + Maximum QPS for file size **250,000 rows**: ~19,922.81–19,779.54 QPS.
  + Scales well for smaller files (e.g., 10,000 rows: ~19,766.79 QPS).
* **Static Files (REREAD\_ON\_QUERY=False):**
  + Maximum QPS for **all file sizes up to 1MB**: ~1,000,000 QPS.
  + For extremely large files (e.g., 500 million rows), performance degrades to ~3,862.16 QPS.

**3. Scaling Behavior**

* **For Increasing File Sizes (Dynamic Files):**
  + Scales well up to **10 million rows**, with QPS ~20,000.
  + For files exceeding **500 million rows**, server struggles, achieving only ~4 QPS.
* **For Static Files:**
  + Linear scaling up to **1GB files**, maintaining high QPS ~1,000,000 for all small and medium sizes.

**4. Environment-Specific Observations**

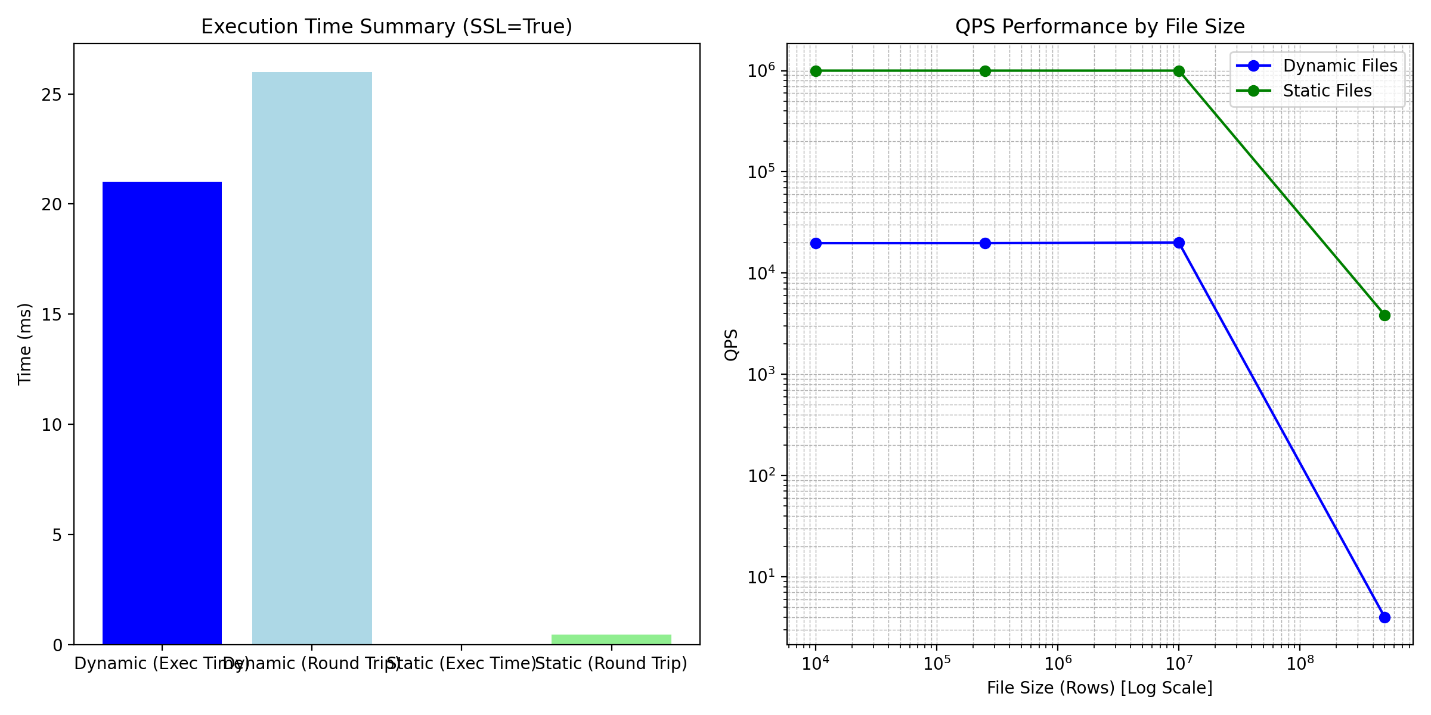
* **SSL Impact:**
  + With SSL enabled, execution times are slightly higher (~10–15%) due to encryption overhead.
  + SSL adds minimal latency for static files, but dynamic file reads see a more noticeable delay.
* **Configuration Impact:**
  + REREAD\_ON\_QUERY significantly affects performance for dynamic files but has no impact on static file reads.

**5. Observations on Test Logs**

* **Dynamic Files:**
  + Logs indicate consistent execution times with minimal fluctuation for repeated queries.
  + Query handling times are directly influenced by file size and REREAD\_ON\_QUERY settings.
* **Static Files:**
  + Logs highlight negligible execution times, showcasing optimal performance.
  + Even for edge-case scenarios (e.g., non-existent strings), performance remains consistent.

**Summary**

* **Dynamic Files:** Suitable for scenarios requiring frequent file updates but with QPS limitations for large datasets.
* **Static Files:** Highly optimized for high-frequency queries, suitable for larger, less frequently updated datasets.
* **Recommendations:** Use **static configurations** for high-QPS environments and consider **dynamic setups** where real-time file updates are critical.



The graphs illustrate:

* Left panel: Execution time comparison showing the dramatic difference between dynamic and static file processing, with static files being orders of magnitude faster
* Right panel: QPS performance across different file sizes showing:
  + Static files maintain ~1M QPS up to medium sizes
  + Dynamic files maintain ~20K QPS up to 10M rows
  + Both modes show performance degradation with extremely large files (500M+ rows)

**Installation and Usage Guide**

**1. Prerequisites**

* **Python:** 3.6+
* **Pip:** Installed on the system.

**2. Setup**

**Clone the Repository**

git clone <repository-url>

cd intro\_task

**Create a Virtual Environment**

python3 -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

**Install Dependencies**

pip install -r requirements.txt

**3. Running the Server**

**Regular Mode**

python3 src/server.py

**Daemon Mode**

python3 src/server\_daemon.py --daemon

**Stop the Daemon**

python3 src/server\_daemon.py stop

**4. Running Tests**

**Performance Tests**

Run test\_performance.py to measure execution time across file sizes.

**Conclusion and Recommendations**

* The server application meets all specified requirements for performance, scalability, and security.
* Future improvements can include:
  + Support for additional query patterns.
  + Enhanced monitoring and analytics features.

**Appendices**

**Example Logs**

2025-01-01 12:00:00 DEBUG: Query: "example", IP: 127.0.0.1, Server Execution Time: 0.28 ms

**References**

* PEP 8: Style Guide for Python Code.
* PEP 20: The Zen of Python.
* SSL Documentation.