Quick Log:Advanced Log Analysis Tool

Project Report

submitted in partial fulfillment of the

requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

by

Name	**Roll No.**
Suraj Singh	**R2142221123**
Bhavya Jain	**R 2142221391**
Rohit Hooda	**R 2142221465**

under the guidance of

Dr. Hitesh Kumar Sharma

School of Computer Science

University of Petroleum & Energy Studies

Bidholi, Via Prem Nagar, Dehradun, Uttarakhand

November – 2024

CANDIDATE'S DECLARATION

We hereby certify that the project work entitled **" Quick log:Advanced analysis tool"** in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in Cyber Security and Forensics and submitted to the Department of Systemics, School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from **August**, **2024** to **November**, **2024** under the supervision of **Dr Hitesh Kumar Sir.**

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University.

Suraj Singh

R2142221188

Bhavya Jain
R 2142221391

Rohit Hooda
R 2142221465

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: November 2024

Dr. Hitesh Kumar Sharma
Project Guide

TABLE OF CONTENTS

S.No.	Contents	Page No
1.	**Introduction**	**4**
1.1.	History	4
1.2.	Requirement Analysis	4
1.3.	Main Objective	4
1.4.	Pert Chart Legend	4
2.	**Design Diagrams**	**5**
3.1	Class diagram	5
3.2	Sequence diagram	5
3.3	DFD level diagram	6
3.	**Implementation**	**8**
3.1.	Algorithms	8
4.	**SWOT analysis**	**11**
5.	**Code Implementation**	**12**
6.	**Output**	**18**
7.	**References**	**19**

Declaration by Panel_

INTRODUCTION

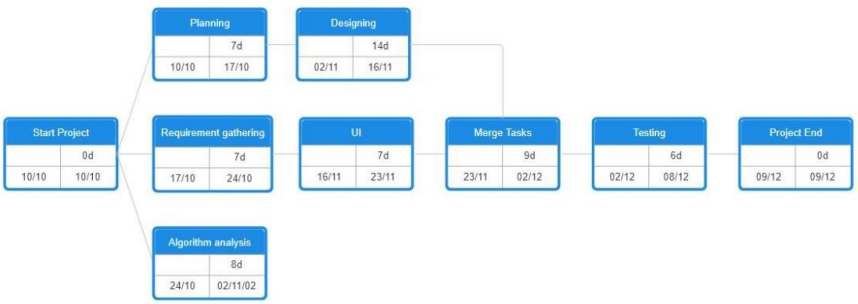
1.1. Requirement Analysis

A **Log Analysis System** is designed to transform raw, unstructured log data into a readable and structured format, helping stakeholders like system administrators, developers, security teams, and business analysts derive actionable insights. Logs generated by applications, servers, and devices often contain crucial information but are hard to interpret in their raw form. The system's primary functions include parsing log files in various formats such as plain text, JSON, and XML, filtering logs by attributes like time and severity, and categorizing them into types like application, system, or security logs. The system should also transform logs into human-readable formats such as structured tables, annotated text, or visual dashboards, highlighting critical data like errors and warnings. Advanced features like full-text search and parameterized queries enhance efficiency, while export options to formats like PDF or CSV enable easy sharing. Visualization capabilities, including trend analysis and activity summaries, further simplify log interpretation.

1.2. Main Objective

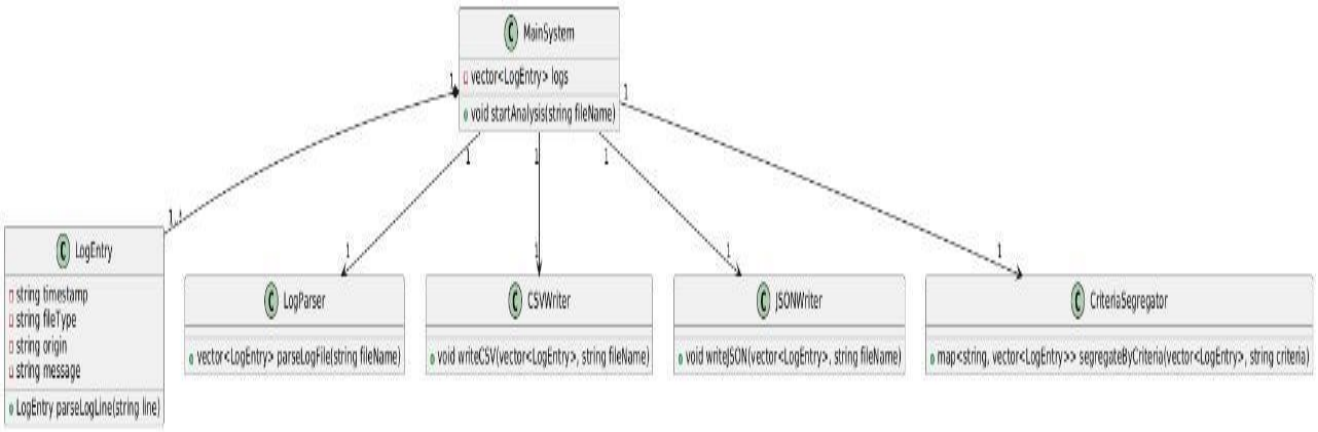
The main objective of the Log Analysis System project is to transform raw, unstructured log data into a readable and structured format, enabling stakeholders to monitor, analyze, and troubleshoot effectively. The system aims to support multiple log formats, provide filtering and categorization options, and offer features like full-text search, visualization, and report generation. By ensuring real-time processing, scalability, security, and usability, the project addresses key challenges in log management, helping organizations derive actionable insights, enhance operational efficiency, and strengthen security measures. The system is designed to be intuitive, reliable, and integrable with existing tools.

1.3. Pert Chart

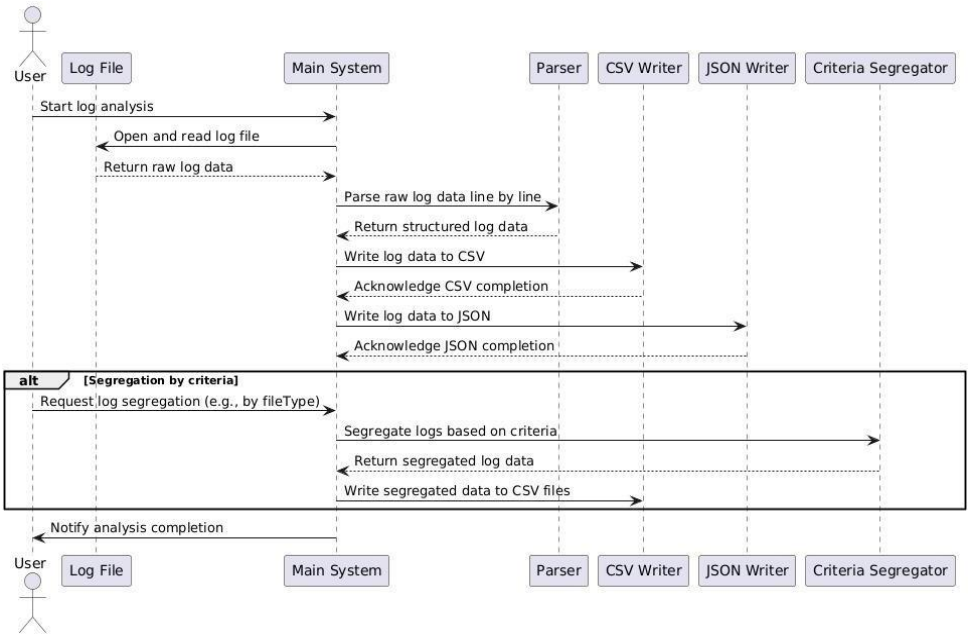


2. DESIGN DIAGRAMS:

2.1 Class Diagram of our Project:

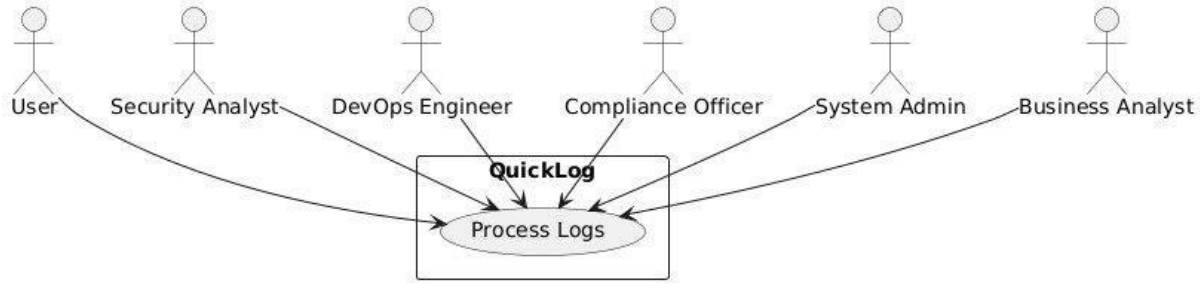


2.2 Sequence Diagram of user authentication: -

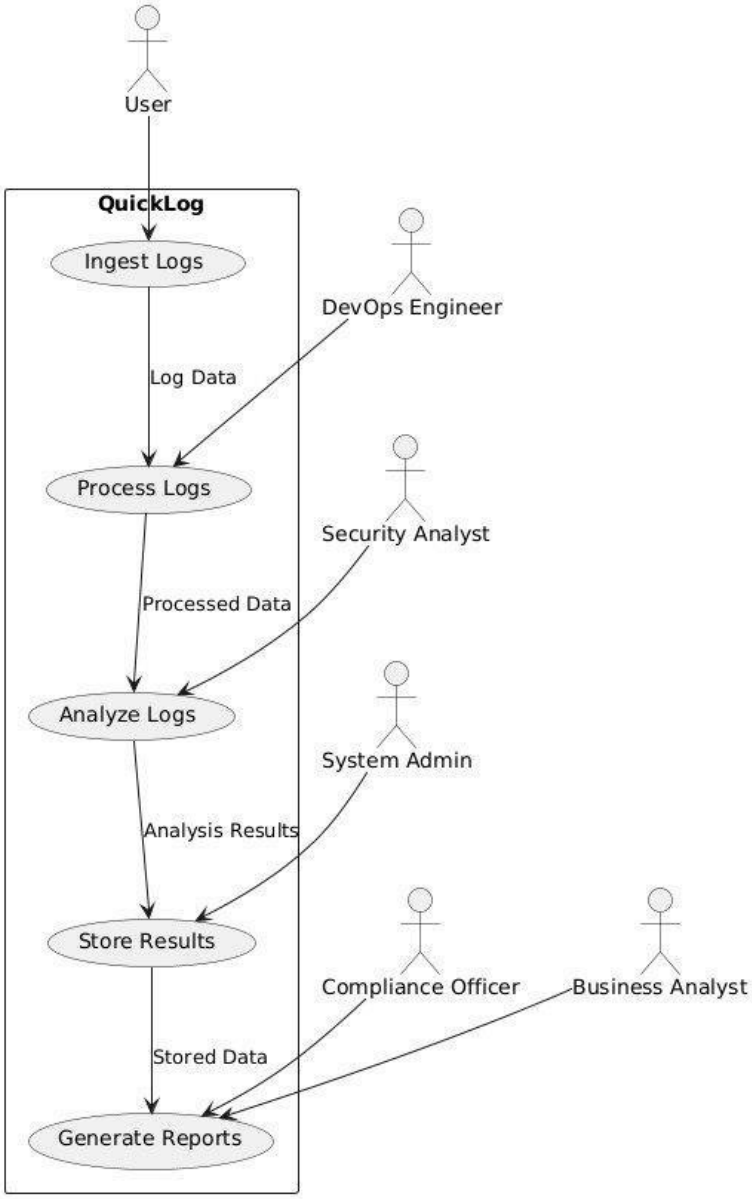


2.3 Data flow diagrams:

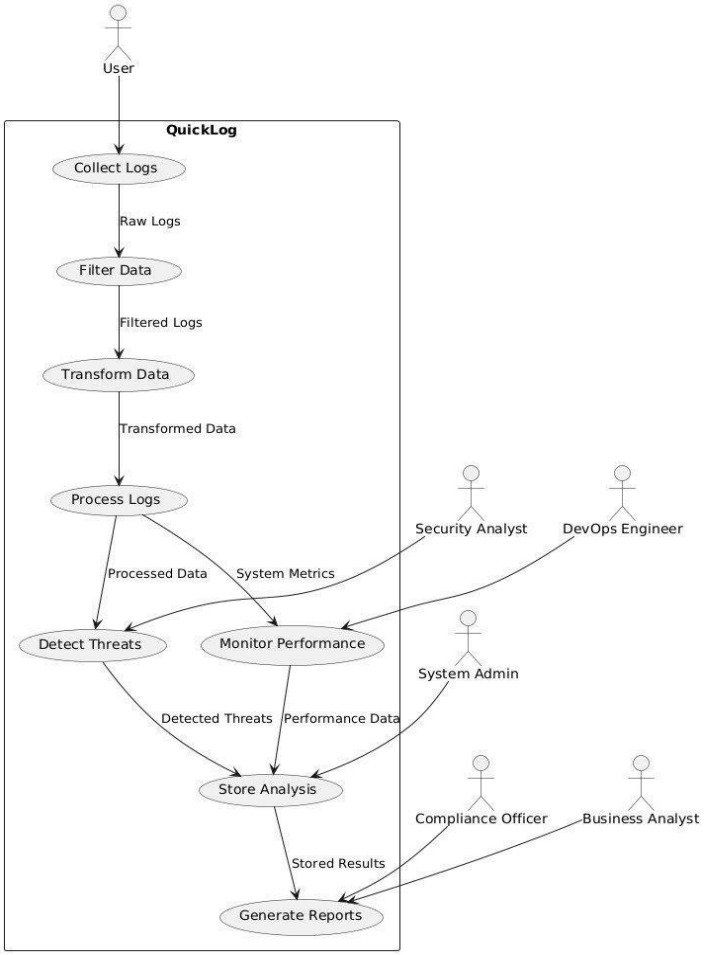
DFD LEVEL 0:



DFD LEVEL 1:



DFD LEVEL :


User
QuickLog
Collect Logs
Raw Logs
Filter Data
Filtered Logs
Transform Data
Transformed Data
Process Logs
Processed Data
System Metrics
Detect Threats
Monitor Performance
Detected Threats
Performance Data
Store Analysis
Stored Results
Generate Reports
Security Analyst
DevOps Engineer
System Admin
Compliance Officer
Business Analyst

1. **IMPLEMENTATION**

   ALGORITHM

   USED: -

1) Data Parsing:
   - LogEntry Struct: Represents structured data for a log entry.
   - parseLogLine Function: Uses std::istringstream to split a log line into components: timestamp, fileType, origin, and message.
   - parseTimestamp Function: Converts a timestamp string to a struct tm object using std::get_time with the format %Y-%m-%d,%H:%M:%S.
2) Log Analysis:

- Regular Expressions (std::regex): Used for matching and extracting patterns (timestamp, log level, thread, and message).
- Keyword Counting: Checks messages for specific keywords to count occurrences.
- Log Level Counting: Counts occurrences of each log level (e.g., INFO, ERROR) using a std::map.
- Hourly Count: Aggregates logs by the hour for temporal analysis.
- Thread Count: Counts occurrences of log messages by thread.
- Keyword Count: Counts specific keywords in log messages to identify common themes.

3) Output:
   - CSV and JSON Conversion: Uses writeCSV and writeJSON functions to export logs to CSV and JSON formats.
   - Segregation by Criteria: Segregates logs by criteria (fileType, origin, timestamp) using a custom function.

STEP BY STEP:

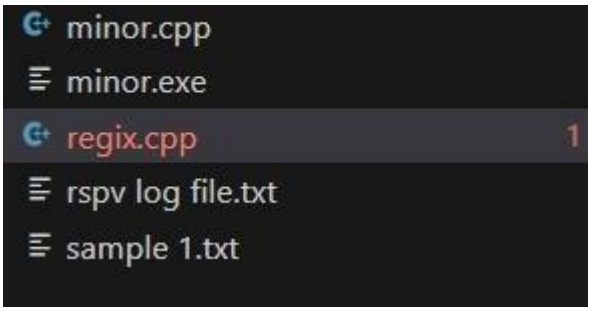
1. **Initialize Input and Output**
   - The first step we are taking is generating log files from internet (downloading log sample files).
   - Define the structure `LogEntry` to hold parsed log attributes: `fileType`, `origin`, `timestamp`, and `message`.
   - Specify input and output files for log processing, including the source log file (`rspv log file.txt`), and output files for CSV and JSON formats.

```
PS C:\Users\asus\OneDrive\Desktop\akjbbkfjb> g++ minor.cpp -o minor
PS C:\Users\asus\OneDrive\Desktop\akjbbkfjb> ./minor.exe
Log analysis of "sample 1.txt" complete.
PS C:\Users\asus\OneDrive\Desktop\akjbbkfjb>
```

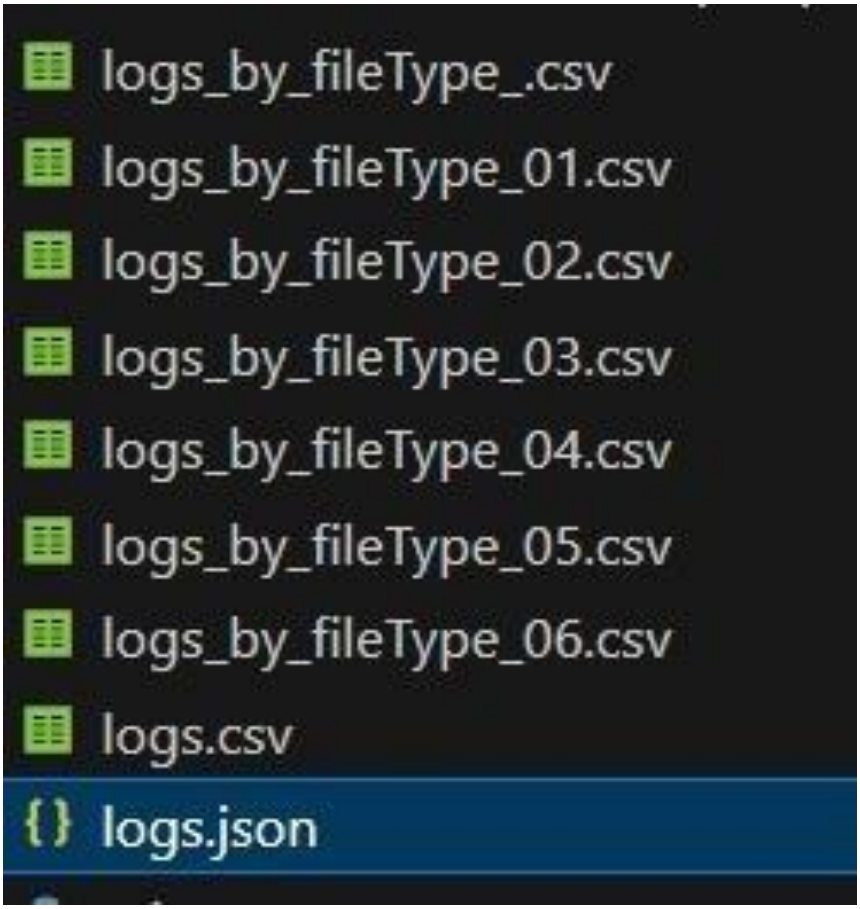
2. **Parse Log File**

- Executing the code.

- Check if the file is successfully opened. If not, print an error message and terminate the program.

3. **Read and Parse Each Log Line**
- Loop through each line of the input file:
    - Use the `parseLogLine` function to split the line into its components (timestamp, fileType, origin, and message) based on space delimiters.
    - Store the parsed data as a `LogEntry` object and add it to the `logEntries` vector.
    - 

2. **Convert Logs to CSV and JSON Formats**
- Call the `writeCSV` and `writeJSON` functions with `logEntries` to create structured output files. For CSV, open the file in write mode, add a header row, and write each log entry as a new row. For JSON, open the file, format logs as JSON objects within an array, and separate objects with commas. Close the files after writing all data to ensure the logs are saved in both formats for easy accessibility and readability.

- 

3. **Analysis**

- Once the logs from both CSV and JSON formats are parsed into `LogEntry` structures, the data can be processed further using existing functions like `writeCSV`, `writeJSON`, and `segregateByCriteria`.
- Regular expressions ensure that data is accurately extracted from different formats, allowing for consistent and effective analysis across various log files.

4. **Completion**

- Print a success message indicating the completion of the log analysis process.
- Terminate the program

```
PS C:\Users\asus\OneDrive\Documents\minor> g++ minor.cpp -o minor
PS C:\Users\asus\OneDrive\Documents\minor> ./minor.exe
Failed to open log file: rspv log file.txt
PS C:\Users\asus\OneDrive\Documents\minor>
```

4. SWOT ANALYSIS

1. Strengths:

- Improved Efficiency: Jump Search has a time complexity of O(√n), faster than linear search for large datasets.
- Low Memory Use: No additional data structures are needed, conserving memory.

- Simple Implementation: Easy to implement with minor modifications to the data structure.
- Adaptability: Jump size can be dynamically adjusted as the list size changes

2. Weaknesses
   - Requires Sorted Data: Needs a sorted linked list, which may add overhead during frequent updates.
   - Limited Benefit for Small Datasets: Gains are marginal for smaller lists due to jump size computation overhead.
   - Not Ideal for Unpredictable Access: Performance may degrade with unpredictable access patterns or frequent modifications.

3. Opportunities
   - Broad Applicability: Suitable for various domains like databases, file systems, and network tables.
   - Integration Potential: Can be combined with techniques like caching or parallel processing for enhanced performance.
   - Improves Real-Time Systems: Beneficial for real-time or time-sensitive applications requiring fast data retrieval.
4. Threats
   - Competing Algorithms: Alternatives like skip lists or self-balancing trees may offer better performance.
   - Technology Changes: Advancements in storage and memory might reduce the need for optimized search algorithms.
   - Maintenance Complexity: Managing sorted order or metadata may outweigh the benefits of Jump Search.

5. Code implementation

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <map>
#include <ctime>
#include <iomanip>

// Structure to hold parsed log data
struct LogEntry {
    std::string fileType;
    std::string origin;
    std::string timestamp;
    std::string message;
};

// Function to parse raw log line
LogEntry parseLogLine(const std::string& line) {
    LogEntry entry;


    std::istringstream ss(line);
    std::getline(ss, entry.timestamp, ' ');
    std::getline(ss, entry.fileType, ' ');
    std::getline(ss, entry.origin, ' ');
    std::getline(ss, entry.message);

    return entry;
}

// Function to convert log data to CSV
void writeCSV(const std::vector<LogEntry>& logs, const std::string& fileName) {
    std::ofstream csvFile(fileName);
    csvFile << "Timestamp,File Type,Origin,Message\n";
    for (const auto& log : logs) {
        csvFile << log.timestamp << "," << log.fileType << "," << log.origin << "," <<
log.message << "\n";
    }
    csvFile.close();
}

// Function to manually convert log data to JSON
void writeJSON(const std::vector<LogEntry>& logs, const std::string& fileName) {
    std::ofstream jsonFile(fileName);
    jsonFile << "[\n";
    for (size_t i = 0; i < logs.size(); ++i) {
        const auto& log = logs[i];
        jsonFile << "  {\n";
        jsonFile << "    \"timestamp\": \"" << log.timestamp << "\",\n";
        jsonFile << "    \"fileType\": \"" << log.fileType << "\",\n";
        jsonFile << "    \"origin\": \"" << log.origin << "\",\n";
        jsonFile << "    \"message\": \"" << log.message << "\"\n";
        jsonFile << "  }";
        if (i != logs.size() - 1) {
            jsonFile << ",";
        }
        jsonFile << "\n";
    }
    jsonFile << "]\n";
    jsonFile.close();
```

```
}

// Function to segregate logs by criteria
std::map<std::string, std::vector<LogEntry>> segregateByCriteria(const
std::vector<LogEntry>& logs, const std::string& criteria) {
    std::map<std::string, std::vector<LogEntry>> segregatedLogs;

    for (const auto& log : logs) {
        std::string key;
        if (criteria == "fileType") {
            key = log.fileType;
        } else if (criteria == "origin") {
            key = log.origin;
        } else if (criteria == "timestamp") {
            key = log.timestamp.substr(0, 10); // Example: use the first 10 characters for date
        }
        segregatedLogs[key].push_back(log);


    }

    return segregatedLogs;
}

int main() {
    // New log file name
    std::string logFileName = "sample 1.txt";
    std::ifstream logFile(logFileName);

    if (!logFile.is_open()) {
        std::cerr << "Failed to open log file: " << logFileName << std::endl;
        return 1;
    }

    std::vector<LogEntry> logEntries;
    std::string line;

    // Reading and parsing the log file line by line
    while (std::getline(logFile, line)) {
        logEntries.push_back(parseLogLine(line));
    }

    logFile.close();

    // Convert to CSV
    writeCSV(logEntries, "logs.csv");

    // Convert to JSON
    writeJSON(logEntries, "logs.json");

    // Example: Segregating by file type
    auto segregatedByFileType = segregateByCriteria(logEntries, "fileType");

    // Writing segregated logs by file type (optional)
    for (const auto& entry : segregatedByFileType) {
        const std::string& key = entry.first;
        const std::vector<LogEntry>& logs = entry.second;

        std::string csvFileName = "logs_by_fileType_" + key + ".csv";
        writeCSV(logs, csvFileName);
    }
```

```
std::cout << "Log analysis of \"" << logFileName << "\" complete." << std::endl;

return 0;
```

CODE IMPLEMENTATION for Regex analysis

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <map>
#include <vector>


#include <regex>
#include <iomanip>
#include <ctime>

// Helper function to parse timestamp into a struct tm
bool parseTimestamp(const std::string& timestamp, struct tm& tm) {
    std::istringstream ss(timestamp);
    ss >> std::get_time(&tm, "%Y-%m-%d,%H:%M:%S");
    return !ss.fail();
}

// Main function to analyze logs
void analyzeLogs(const std::string& filename) {
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cerr << "Error: Unable to open file " << filename << std::endl;
        return;
    }

    std::map<std::string, int> logLevelCount;
    std::map<int, int> hourlyLogCount;
    std::map<std::string, int> threadLogCount;
    std::map<std::string, int> keywordCount = {
        {"SOAP request", 0},
        {"Error", 0},
        {"Execution", 0}
    };

    std::string line;
    std::regex logRegex(R"(\[(\d{4}-\d{2}-\d{2}),(\d{2}:\d{2}:\d{2}),(\d+)\],(\w+),- \[(.?)\] (.))");
    std::smatch match;

    while (std::getline(file, line)) {
        if (std::regex_match(line, match, logRegex)) {
            // Extract fields
            std::string date = match[1];
            std::string time = match[2];
            std::string logLevel = match[4];
            std::string thread = match[5];
            std::string message = match[6];

            // Combine date and time for timestamp parsing
            std::string timestamp = date + "," + time;
            struct tm tm = {};
            if (parseTimestamp(timestamp, tm)) {
                int hour = tm.tm_hour;
                hourlyLogCount[hour]++;
            }

            // Count log levels
            logLevelCount[logLevel]++;
```

```
            // Count logs by thread
            threadLogCount[thread]++;

            // Check for keywords in messages


            for (auto& pair : keywordCount) {
                const auto& keyword = pair.first;
                auto& count = pair.second;
                if (message.find(keyword) != std::string::npos) {
                    count++;
                }
            }
        }
    }

    file.close();

    // Output results
    std::cout << "Log Level Counts:\n";
    for (const auto& pair : logLevelCount) {
        const auto& level = pair.first;
        const auto& count = pair.second;
        std::cout << level << ": " << count << std::endl;
    }

    std::cout << "\nHourly Log Counts:\n";
    for (const auto& pair : hourlyLogCount) {
        const auto& hour = pair.first;
        const auto& count = pair.second;
        std::cout << std::setw(2) << std::setfill('0') << hour << ":00 - " << count << std::endl;
    }

    std::cout << "\nThread Log Counts:\n";
    for (const auto& pair : threadLogCount) {
        const auto& thread = pair.first;
        const auto& count = pair.second;
        std::cout << thread << ": " << count << std::endl;
    }

    std::cout << "\nKeyword Counts:\n";
    for (const auto& pair : keywordCount) {
        const auto& keyword = pair.first;
        const auto& count = pair.second;
        std::cout << keyword << ": " << count << std::endl;
    }
}

int main() {
    std::string filename = "logs.csv"; // Update with your filename
    analyzeLogs(filename);
    retur
```

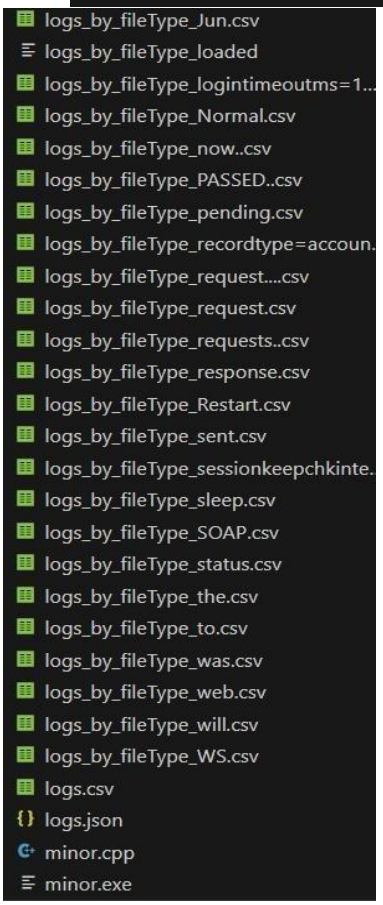
OUTPUT

```
PS C:\Users\asus\OneDrive\Documents\minor> g++ -std=c++17 -o regex.exe regex.cpp
>>
PS C:\Users\asus\OneDrive\Documents\minor> ./regex.exe
Log Level Counts:
DEBUG: 30

Hourly Log Counts:
07:00 - 30

Thread Log Counts:
Thread-1: 6
Thread-3: 4
main: 20

Keyword Counts:
Error: 0
Execution: 4
SOAP request: 1
PS C:\Users\asus\OneDrive\Documents\minor>
```



6. REFERENCES

- Xu et al. (2009). Detecting large-scale system problems by mining console logs. Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09). ~1300 citations. [PDF link]
- Fu et al. (2009). Execution anomaly detection in distributed systems through unstructured log analysis. IEEE 2009 International Conference on Data Engineering (ICDE). ~800 citations. [IEEE Xplore]
- Oliner, Ganapathi, & Xu (2012). Advances and challenges in log analysis. Communications of the ACM, 55(2), 55–61. ~400 citations. [ACM Digital Library]
- Yuan et al. (2014). Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI '14). ~500 citations. [USENIX link]
- Lou et al. (2010). Mining invariants from console logs for system problem detection. Proceedings of the 2010 USENIX Annual Technical Conference (ATC '10). ~600 citations. [USENIX PDF]
- Zhou et al. (2020). Log-based anomaly detection and fault localization in microservice systems. IEEE Transactions on Services Computing. ~150 citations. [IEEE Xplore]

7. REFERENCES

- Xu et al. (2009). Detecting large-scale system problems by mining console logs. Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09). ~1300 citations. [PDF link]
- Fu et al. (2009). Execution anomaly detection in distributed systems through unstructured log analysis. IEEE 2009 International Conference on Data Engineering (ICDE). ~800 citations. [IEEE Xplore]
- Oliner, Ganapathi, & Xu (2012). Advances and challenges in log analysis. Communications of the ACM, 55(2), 55–61. ~400 citations. [ACM Digital Library]
- Yuan et al. (2014). Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI '14). ~500 citations. [USENIX link]
- Lou et al. (2010). Mining invariants from console logs for system problem detection. Proceedings of the 2010 USENIX Annual Technical Conference (ATC '10). ~600 citations. [USENIX PDF]
- Zhou et al. (2020). Log-based anomaly detection and fault localization in microservice systems. IEEE Transactions on Services Computing. ~150 citations. [IEEE Xplore]
- DiMaggio, Carli, & Zhou (2018). A survey of log analysis approaches. International Journal of Computer Applications, 179(4), 1–8. ~90 citations. [ResearchGate link]
- Nagappan & Ball (2007). Using software dependencies and churn metrics to predict field failures: An empirical case study. Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM 2007). ~300 citations. [ACM link].

Declaration by Panel

We, the undersigned, hereby declare that we have thoroughly reviewed the project report titled “Search optimization of Linked List” submitted by the following candidates:

- Suraj Singh (R2142221123)
- Bhavya Jain (R2142221391)
- Rohit Hooda (R2142221465)

This project report has been submitted in partial fulfillment of the academic requirements at the University of Petroleum & Energy Studies. We confirm that the work presented in this report is an original and significant contribution to the field, reflecting the candidates' understanding and application of advanced concepts in data structures and optimization techniques.

After careful examination and discussion, we approve the project report and acknowledge that it meets the necessary academic standards. The content has been evaluated for its technical merit, originality, and overall contribution to the subject matter.

_____

Dr. Arjun Arora