**Achieve AI Asignment**

Picture

**Context**

At achieve.ai, generate logs at a very high rate. For example, our service creates over 100,000 log lines a second.

Usually, these logs are loaded inside databases to enable fast querying, but the cost of keeping all the logs becomes too high. For this reason, only the recent logs are kept in databases, and older logs are kept in file archives. For this problem, we should assume that we store our data in multiple files.We close a file and start a new file when the file size reaches 16GB. Our file names are of the format LogFile-######.log (e.g., LogFile-000008.log, or LogFile-000139.log).

Currently, we have over 10,000 log files with the last log file named LogFile-0018203.log and a total data size of 285TB.

**Problem Statement**

We usually use our log database to query our logs. But now and then, we may have to query older logs for customer support or debugging. In most of these cases, we know the time range for which we need to analyze the logs.We need a tool that could extract the log lines from a given time range and print it to the console in a time-effective manner. The command line (CLI) for the desired program is as below LogExtractor.exe -f "From Time" -t "To Time" -i "Log file directory location" The time format will be in "ISO 8601". The extraction process should complete in a few seconds, minimizing the engineer's wait

**Installation**

git clone ---URL

npm i --save-dev

# For starting the node server

npm start

# For Starting the nodemon server

npm run dev

**Log file format**

• The log file has one log per line

• Every log line will start with the timestamp in "ISO 8601" format followed by a comma (',')

• All the log lines will be separated by a single newline character '\n'

• Example logline:

◦ 2020-01-31T20:12:38.1234Z, some field, other field, & so on, till new line

**Language Used**

Picture

I chose this language because

• More Familiarity with Javascript than other languages like Python, Java etc.

• Javascript can be used for making both the frontend and the backend applications

• Easier read operation than other languages.

**My Approach**

My approach was very simple. In the problem statement were given that there is some data which is divided into multiple segments and we have to fetch data from those segments in very less time.

Iterating each and every segment will take a huge time for checking data. Instead we can create some sort of meta data for every segment which will gove use the information about the segment.

Those information can be

fromDate: "2022-07-11T20:10:09.357Z"

toDate: "XXXXXXXXXXXXXX"

path: "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

# and some more data

These details can be stored in database. For posting these data we can create a **CronJob** which will take snapshot of each metadata every minute.

This will be very helpfull when we fetch the data from the time to time.We can also use **Timeseries** **Database** in **Mongoose** for making it possible

Picture

Now when the user demands for the logs between a specific timeperiod we can easily process it by finding all the files which lies between the following dates. Accordingly we can give the output the user of the files regarding the same.

**Log File Format**

**Getting the last line function**

This will give us the first line from the log file and after that we can get the startDate of that log file by **Splitting the** **First Line**

Picture

**Similarly we can do it for getting the first Line**

Picture

**Format for Log File**

Picture

**We can call these API's in local Server**

# For posting Data in the Database

http://localhost:3000/api/v1/logs/postData

# For Retrieval we can use this API

http://localhost:3000/api/v1/logs/postData\

After the successFull Confirmation of the API we will get the following response

Picture

This Picture Confirms the demanded data lies between these 2 segments in the Log Files. Now the log Files can be searched accordingly.

**Project Working Video**

Picture

**Bottlenecks Involved**

This project doesn't give the exact value of data in the logs. Instead it is giving us the file where the data is present. This means it gives us some extra data which is not useFull