

PROJECT- LOG ANALYZER GROUP-8 (UI &BACKEND ROLE)

TEAM MEMBERS-
VANSI SHARMA
NAMAN GUPTA
MANISH GUPTA
SACHIN SHERAWAT

SUBMITTED TO- Mrs Shikha

Problem statement: -Different application have different logging strategies and logs are subsequently generated for various different reasons. While some logs are used to trace an error back to origin, there are logs that are generated that can help understand performance of different parts of applications.

Following is an extract from a log file that records User's activities on a web application:

```
2018-09-18 04:49:38,215 ERROR (default task-95)
IP-Address=157.49.141.133#,!User-Agent=Mozilla/5.0 (Windows NT 10.0; WOW64;
Trident/7.0; rv:11.0) like
Gecko#,!X-Request-From=UIX#,!Request-
Type=POST#,!API=/v1/admin/developers#,!U      ser-Login=test@demo.com#,!User-
Name=testUser#,!EnterpriseId=2#,!EnterpriseNam e=Enterprise-2#,!Auth-
Status=#,!Status-Code=200#,!Response-Time=346#,!Request
-Body=
```

```
2018-09-18 04:49:39,842 ERROR (default task-21)
IP-Address=157.49.141.133#,!User-Agent=Mozilla/5.0 (Windows NT 10.0; WOW64;
Trident/7.0; rv:11.0) like
Gecko#,!X-Request-From=UIX#,!Request-Type=GET#,!API=/v2/developers#,!User-
Log                               in=test@demo.com#,!User-
Name=testUser#,!EnterpriseId=2#,!EnterpriseName=Enter prise-2#,!Auth-
Status=#,!Status-Code=200#,!Response-Time=1240#,!Request-Body=
```

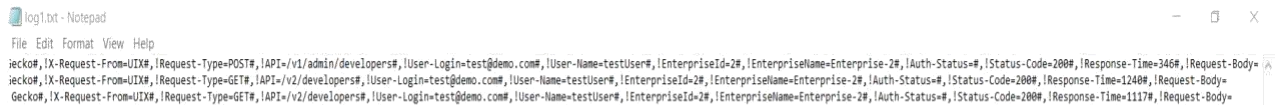
```
2018-09-18 04:49:41,946 ERROR (default task-127)
IP-Address=157.49.141.133#,!User-Agent=Mozilla/5.0 (Windows NT 10.0; WOW64;
Trident/7.0; rv:11.0) like
Gecko#,!X-Request-From=UIX#,!Request-Type=GET#,!API=/v2/developers#,!User-
Log                               in=test@demo.com#,!User-
Name=testUser#,!EnterpriseId=2#,!EnterpriseName=Enter prise-2#,!Auth-
Status=#,!Status-Code=200#,!Response-Time=1117#,!Request-Body=
```

To analyze these logs manually takes significant effort and time. It requires a developer to extract the logs, parse and review them, making it significantly time consuming.

Design a Log analyzer which takes a file as Input, extracts, parses and stores data in a format that can be used to easily filter out activities based on different parameters and the final result should contain at least following information

1. IP Address
2. User Agent
3. Status Code (200/401/500 etc).
4. Request Type (GET/POST/PUT etc)
5. API
6. User
7. Enterprise Id
8. Enterprise Name

Assumption made :- Since we weren't provided with any log file so we have assumed that the given file log has information in single rows i.e there is a new row after every string "!Request-Body=". For example in the log.txt file given below :-



```
log1.txt - Notepad
File Edit Format View Help
ieckof,IX-Request-From=UIX#,Request-Type=POST#,API=/v1/admin/developers#,User-Login=test@demo.com#,User-Name=testUser#,EnterpriseId=2#,EnterpriseName=Enterprise-2#,Auth-Status=#,Status-Code=200#,Response-Time=346#,Request-Body=
ieckof,IX-Request-From=UIX#,Request-Type=GET#,API=/v2/developers#,User-Login=test@demo.com#,User-Name=testUser#,EnterpriseId=2#,EnterpriseName=Enterprise-2#,Auth-Status=#,Status-Code=200#,Response-Time=1240#,Request-Body=
Gecko#,IX-Request-From=UIX#,Request-Type=GET#,API=/v2/developers#,User-Login=test@demo.com#,User-Name=testUser#,EnterpriseId=2#,EnterpriseName=Enterprise-2#,Auth-Status=#,Status-Code=200#,Response-Time=1117#,Request-Body=
```

Initial Approach:-

We basically thought of using python to write the source code. So the initial approach was to check the type of data provided to us in each lines

```
In [19]: line=[]
li=[]
with open('C:/Users/vabss/Documents/log1.txt') as f:
    lines = f.readlines()
    for each_line in lines:
        print(type(each_line))

<class 'str'>
<class 'str'>
<class 'str'>
```

So the data is of string type so we knew we can perform string operations on each lines.

We thought of using regular expressions for parsing the log file. A regular expression (regex or regexp for short) is a special text string for describing a search pattern. Regex is present in almost every language and uses a regex processor that uses algorithms like backtracking etc to search for the patterns.

Now we wrote regular expressions for each keyword for eg:

For IP-address the regular expression is `r'(\d+.\d+.\d+.\d+)'`. Similarly, we wrote it for every keyword we want to extract.

Now for every line, we used the `The re.search()` method takes a regular expression pattern and a string and searches for that pattern within the string. If the search is successful, `search()` returns a match object

`Re.search(pattern,string).group()` This will return the complete matched string and stored it in a empty dictionary using `dict.append()` .So the output dictionary will have keys as the keywords and their respective values as parsed by the regular expression.

The implementation is as below:

```
In [2]: import re
IP=r'(\d+\.\d+\.\d+\.\d+)'
info=r'(INFO|ERROR|WARN|TRACE|DEBUG|FATAL)'
request=r'(?<!=Request-Type\=).*?[\^#]*'
userAgent=r'(?<!=User-Agent\=).*?[\^#]*'
api=r'(?<!=API\=).*?[\^#]*'
userName=r'(?<!=User-Name\=).*?[\^#]*'
userLogin=r'(?<!=User-Login\=).*?[\^#]*'
EnterpriseName=r'(?<!=EnterpriseName\=).*?[\^#]*'
EnterpriseId=r'(?<!=EnterpriseId\=).*?[\^#]*'
requestBody=r'(?<!=Request-Body\=).*?[\^#]*'
responseTime=r'(?<!=Response-Time\=).*?[\^#]*'
statusCode=r'(?<!=Status-Code\=).*?[\^#]*'
authStatus=r'(?<!=Auth-Status\=).*?[\^#]*'

line=[]
li=[]
dict1={}
dict1['ip_address']=[]
dict1['user_agent']=[]
dict1['status_code']=[]
dict1['Request_Type']=[]
dict1['Api']=[]
dict1['User']=[]
dict1['EnterpriseId']=[]
dict1['EnterpriseName']=[]
with open('C:/Users/vabss/Documents/log1.txt') as f:
    for each_line in f:
        li.append(each_line)
for l in li:

    dict1['ip_address'].append(re.search(IP,l).group())
    dict1['user_agent'].append(re.search(userAgent,l).group())
    dict1['status_code'].append(re.search(statusCode,l).group())
    dict1['Request_Type'].append(re.search(request,l).group())
    dict1['Api'].append(re.search(api,l).group())
    dict1['User'].append(re.search(userName,l).group())
    dict1['EnterpriseId'].append(re.search(EnterpriseId,l).group())
    dict1['EnterpriseName'].append(re.search(EnterpriseName,l).group())
```

This was the result when dictionary was printed:

```
In [3]: print(dict1)

{'ip_address': ['157.49.141.133', '157.49.141.133', '157.49.141.133'], 'user_agent': ['Mozilla/5.0 (Windows NT 10.0; WOW64;Trident/7.0; rv:11.0) like Gecko', 'Mozilla/5.0 (Windows NT 10.0; WOW64;Trident/7.0; rv:11.0) like Gecko', 'Mozilla/5.0 (Windows NT 10.0; WOW64;Trident/7.0; rv:11.0) like Gecko'], 'status_code': ['200', '200', '200'], 'Request_Type': ['POST', 'GET', 'GET'], 'Api': ['/v1/admin/developers', '/v2/developers', '/v2/developers'], 'User': ['testUser', 'testUser', 'testUser'], 'EnterpriseId': ['2', '2', '2'], 'EnterpriseName': ['Enterprise-2', 'Enterprise-2', 'Enterprise-2']}
```

Now we checked our approach for a large log file(approx. size 1.52Gbs) and the total time for parsing the log file was around 8 minutes.

Final Approach: -

So in our next approach we thought of using string manipulations for parsing since we now know that that data provided to us is in form of strings.

To begin with we used the file object as an iterator to read the file line by line (This is faster because we can iterate the returned object itself)

Now we use the split(',') function to split every string by a comma (,) and append it in a list.

So after analyzing the whole log file we found fixed patterns in obtaining IP, User-agent etc. from particular indexes. So we chose those indexes and provided the sufficient result. For various strings like Request type, api etc. where number of characters may change so we found that the string will be between "=" and "#".

Implementation is given below: -

```
In [25]: line=[]
li=[]
with open('C:/Users/vabss/Documents/log1.txt') as f:

    for each_line in f:

        li.append(each_line.split(','))
print(li)
```

```
[['2018-09-18 04:49:38', '215 ERROR (default task-95)IP-Address=157.49.141.133#', '!User-Agent=Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko#', '!X-Request-From=UIX#', '!Request-Type=POST#', '!API=/v1/admin/developers#', '!User-Login=test@demo.com#', '!User-Name=testUser#', '!EnterpriseId=2#', '!EnterpriseName=Enterprise-2#', '!Auth-Status=#', '!Status-Code=200#', '!Response-Time=346#', '!Request-Body=\n'], ['2018-09-18 04:49:39', '842 ERROR (default task-21)IP-Address=157.49.141.133#', '!User-Agent=Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko#', '!X-Request-From=UIX#', '!Request-Type=GET#', '!API=/v2/developers#', '!User-Login=test@demo.com#', '!User-Name=testUser#', '!EnterpriseId=2#', '!EnterpriseName=Enterprise-2#', '!Auth-Status=#', '!Status-Code=200#', '!Response-Time=1240#', '!Request-Body=\n'], ['2018-09-18 04:49:41', '946 ERROR (default task-127)IP-Address=157.49.141.133#', '!User-Agent=Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko#', '!X-Request-From=UIX#', '!Request-Type=GET#', '!API=/v2/developers#', '!User-Login=test@demo.com#', '!User-Name=testUser#', '!EnterpriseId=2#', '!EnterpriseName=Enterprise-2#', '!Auth-Status=#', '!Status-Code=200#', '!Response-Time=1117#', '!Request-Body=\n']]
```

In [26]:

```
dict={}
dict['ip_address']=[]
dict['user_agent']=[]
dict['status_code']=[]
dict['Request_Type']=[]
dict['Api']=[]
dict['User']=[]
dict['EnterpriseId']=[]
dict['EnterpriseName']=[]
for l in li:
    ip_address=l[1][-15:-1]
    user_agent=l[2][12:23]
    status_code=l[11][13:16]
    Request_Type=''
    for i in l[4][14:]:
        if i=='#':
            break
        Request_Type+=i

    Api=''
    for i in l[5][5:]:
        if i=='#':
            break
        Api+=i

    User=''
    for i in l[7][11:]:
        if i=='#':
            break
        User+=i

    EnterpriseId=''
    for i in l[8][14:]:
        if i=='#':
            break
        EnterpriseId+=i

    EnterpriseName=''
    for i in l[9][16:]:
        if i=='#':
            break
        EnterpriseName+=i
```

```
dict['ip_address'].append(ip_address)
dict['user_agent'].append(user_agent)
dict['status_code'].append(status_code)
dict['Request_Type'].append(Request_Type)
dict['Api'].append(Api)
dict['User'].append(User)
dict['EnterpriseId'].append(EnterpriseId)
dict['EnterpriseName'].append(EnterpriseName)
```

```
print(dict)
```

```
{'ip_address': ['157.49.141.133', '157.49.141.133', '157.49.141.133'], 'user_agent': ['Mozilla/5.0', 'Mozilla/5.0', 'Mozilla/5.0'], 'status_code': ['200', '200', '200'], 'Request_Type': ['POST', 'GET', 'GET'], 'Api': ['/v1/admin/developers', '/v2/developers', '/v2/developers'], 'User': ['testUser', 'testUser', 'testUser'], 'EnterpriseId': ['2', '2', '2'], 'EnterpriseName': ['Enterprise-2', 'Enterprise-2', 'Enterprise-2']}
```

Now we checked our approach for a large log file(approx. size 1.52Gbs) and the total time for parsing the log file was around 3-4 minutes(better than the previous approach).

Some Additional Information – We also wrote the above code in c++ using the same approach . (File will be attached along with the documentation)

Now we know that our log data is parsed. Next step will be storage

Storage

Initially we thought , of using pandas dataframes to store the output .DataFrames is a 2 - Dimensional labeled Data Structure with index for rows and columns, where each cell is used to store a value of any type. Basically, DataFrames are Dictionary based out of NumPy Arrays.

Implementation:

```
In [27]: import pandas as pd
df=pd.DataFrame.from_dict(dict)
df
```

```
Out[27]:
```

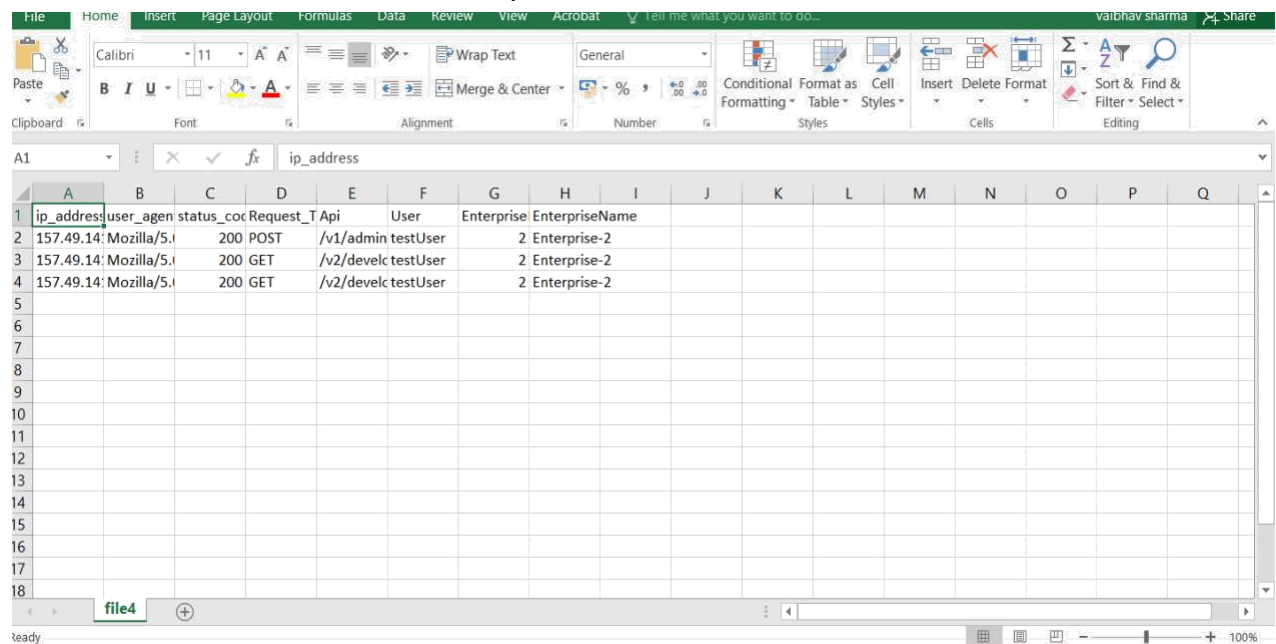
	ip_address	user_agent	status_code	Request_Type	Api	User	EnterpriseId	EnterpriseName
0	157.49.141.133	Mozilla/5.0	200	POST	/v1/admin/developers	testUser	2	Enterprise-2
1	157.49.141.133	Mozilla/5.0	200	GET	/v2/developers	testUser	2	Enterprise-2
2	157.49.141.133	Mozilla/5.0	200	GET	/v2/developers	testUser	2	Enterprise-2

Now we use these data frames to store the output in a csv file

Implementation

```
In [28]: df.to_csv('file3.csv',index=False)
```

A .csv file is stored at our local memory location.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	ip_address	user_agent	status_code	Request_Type	Api	User	Enterprise	EnterpriseName									
2	157.49.141.133	Mozilla/5.0	200	POST	/v1/admin/developers	testUser	2	Enterprise-2									
3	157.49.141.133	Mozilla/5.0	200	GET	/v2/developers	testUser	2	Enterprise-2									
4	157.49.141.133	Mozilla/5.0	200	GET	/v2/developers	testUser	2	Enterprise-2									
5																	
6																	
7																	
8																	
9																	
10																	
11																	
12																	
13																	
14																	
15																	
16																	
17																	
18																	

It was found that the for large log files say 1-2gbs the size of csv file was atleast 250mb. So we further researched to optimize this storage

Our approach- We thought about storing the data in Parquet format. It is a comparatively highly efficient data storage format. It is open source and licensed under Apache. CSVs are row-orientated, which means they're slow to query and difficult to store efficiently. That's not the case with Parquet, which is a column-orientated storage option. The size difference between those two is enormous for identical datasets. Parquet files take much less disk space than CSVs and are faster to scan. As a result, the identical dataset is 16 times cheaper to store in Parquet format. **It was found that the for large log files say 1-2gbs the size of csv file was atleast 250mb and for parquet format file size was merely few kbs.**

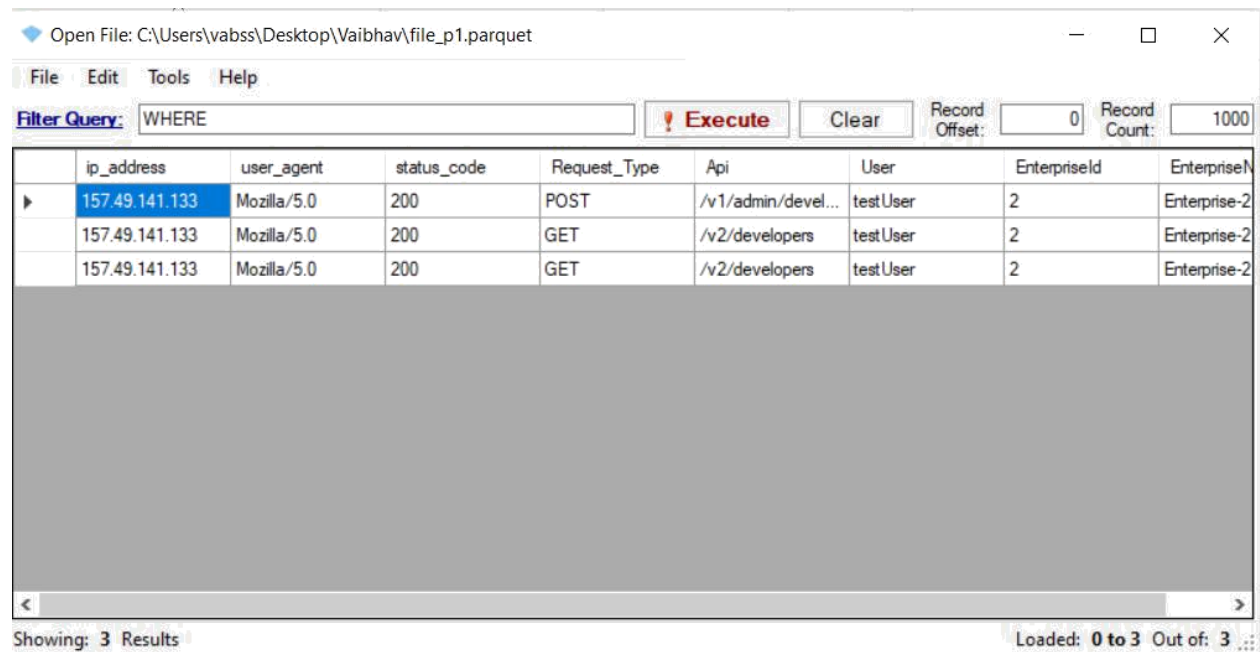
Implementation:

```
In [10]: import pandas as pd
         df=pd.DataFrame.from_dict(dict)

In [11]: import pyarrow as pa
         import pyarrow.parquet as pq
         table = pa.Table.from_pandas(df)
         pq.write_table(table, 'file_p.parquet')

In [ ]:
```

Sample of data stored in parquet format:



	ip_address	user_agent	status_code	Request_Type	Api	User	EnterpriseId	EnterpriseN
▶	157.49.141.133	Mozilla/5.0	200	POST	/v1/admin/devel...	testUser	2	Enterprise-2
	157.49.141.133	Mozilla/5.0	200	GET	/v2/developers	testUser	2	Enterprise-2
	157.49.141.133	Mozilla/5.0	200	GET	/v2/developers	testUser	2	Enterprise-2

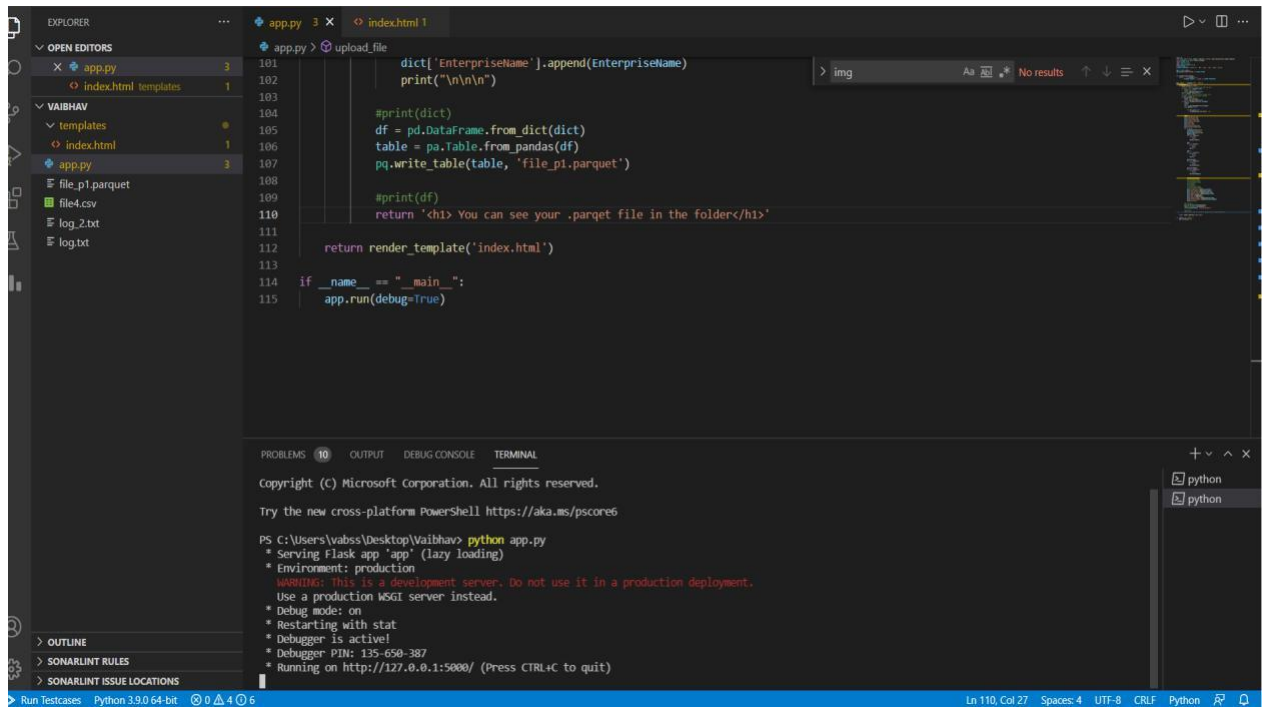
Showing: 3 Results Loaded: 0 to 3 Out of: 3

Query results are also faster in this data format.

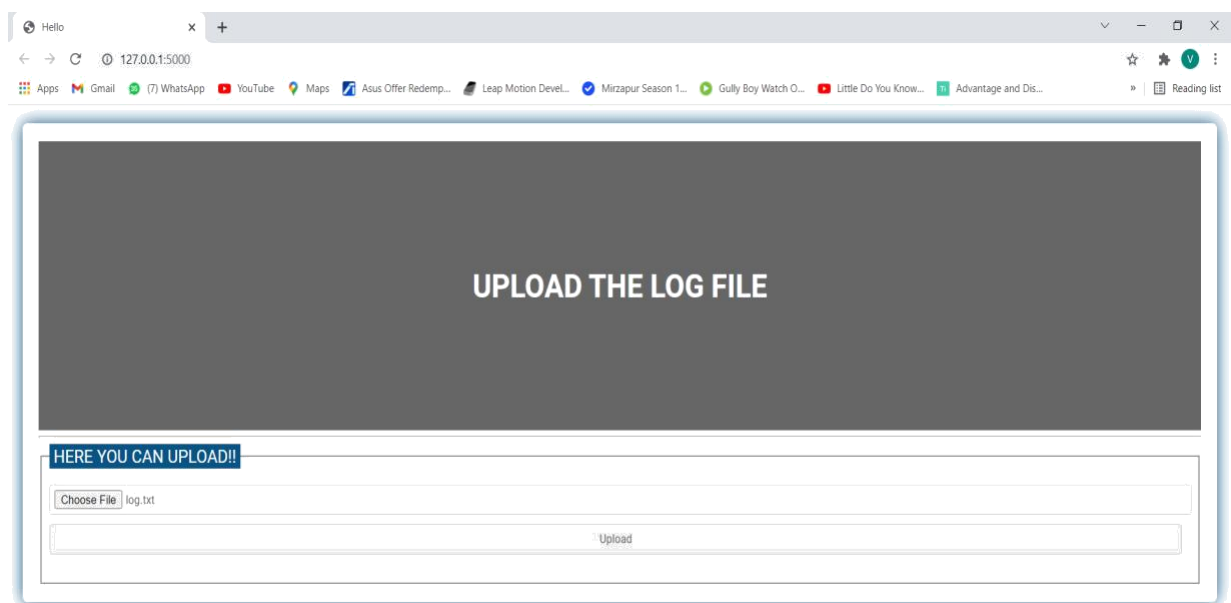
So finally we designed a simple fronted website that runs on local host and takes a log file as input and gives the result as output in parquet format. We basically used flask to connect the front end and the database. Everything was implemented on Microsoft studio.

Sample of our Implementation:

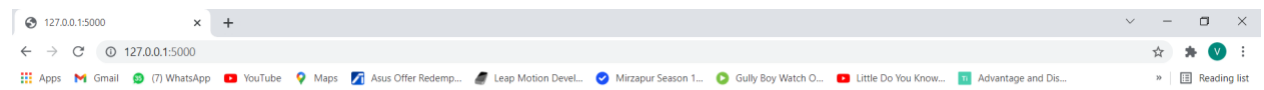
1. Run the local host using ctrl+right click on the link



- 2) Upload the log file using the upload button



3) Get the output in the folder



You can see your .parquet file in the folder

Name	Date modified	Type	Size
templates	17-10-2021 00:08	File folder	
app.py	18-10-2021 22:25	Python Source File	4 KB
file_p1.parquet	18-10-2021 23:12	PARQUET File	6 KB
log.txt	18-10-2021 19:31	Text Document	2 KB
log_2.txt	17-10-2021 01:42	Text Document	14,33,702 ...

Dependencies:-

- 1.Visual Studio (download from- <https://visualstudio.microsoft.com/downloads/>)
- 2.Python (download from <https://www.python.org/downloads/release/python-390/>)
- 3.Flask(installed in Microsoft visual studio. Go through the readme file for more details)
- 4.Parquet Viewer(download from - <https://github.com/mukunku/ParquetViewer/releases>)

Scope of improvements:

- 1.Validation of IP can be done using a validation function. If found invalid, we can directly store value as invalid.
- 2.Using logs to investigate SQL- injection attacks
3. Can store the parsed info into a cloud storage for query based searching.
4. Introducing a multi-tier profiling system, allowing users to choose a default rate limit.