WEB SERVER LOG ANALYSIS SYSTEM

by

VIJAY KRISHNAA S,

PRAVEEN K,

SANTHOSHKUMAR P.

Report submitted in partial fulfilment of
the requirements for the Degree of
Bachelor of Engineering in
Computer Science and Engineering

Sri Ramakrishna Institute of Technology

Coimbatore – 641010

November, 2020

ACKNOWLEDGEMENTS

We sincerely thank Mr. Jim Mathew Philip Assistant Professor (Sr. Gr.), CSE Dept., for guiding through our project for the successful completion. We also thank our beloved principal, Dr. M. Paulraj Ph.D, for providing us with the resources required for the successful completion.

We also thank our IDP Course Instructor, Mr. Devendra Kumar, Assistant Professor, CSE Dept., for properly conducting this course. Lastly, we would also like to thank our Reviewer Mr. Rajesh, Assistant Professor, CSE Dept., for reviewing our project sessions.

APPROVAL AND DECLARATION

This project report titled **Web Server Log Analysis System** was prepared and submitted by **Vijay Krishnaa S (1702128), Praveen K (1702082), Santhosh Kumar P (1702106)** and has been found satisfactory in terms of scope, quality and presentation as partial fulfilment of the requirement for the **Bachelor of Engineering (Computer Science and Engineering)** in Sri Ramakrishna Institute of Technology, Coimbatore (SRIT).

Checked and Approved by

_____

Mr. Jim Mathew Philip
Assisstant Professor

Department of Computer Science and Engineering
Sri Ramakrishna Institute of Technology, Coimbatore – 10
October 2020

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS OR NOMENCLATURE

1.	IP	Internet Protocol
2.	UA	User-Agent
3.	GUI	Graphical User Interface
4.	CSV	Comma Separated Values
5.	HTML	Hyper Text Mark-up Language
6.	DoS	Denial of Service
7.	SEO	Search Engine Optimization

WEB SERVER LOG ANALYSIS SYSTEM

ABSTRACT

This project analyzes the log files of the web server and generate visuals for better understanding of the information in the logs. Log files from the web server are fed as input to the system and it reads the entire log file line by line and stores the data in memory and performs analysis on the data and then the final part of the program is to generate the visuals based on the analysis of the data. The data that is visualized: IP address and the total number of requests from them, request methods like GET, POST and the count of each request methods, the User-Agents used the clients, finally the files that are requested by the clients and frequency of their access. This project can analyze logs from the Apache Web server. From the output provided by the project, we can perform a wide range of tasks. If we find a specific User-Agent requests are increasing, we may develop the features for that specific User-Agent. If we see some alarming rates of requests from a specific IP address, it may be an attempt of Denial-of-Service which may need further investigation, it is also useful for Incident response.

INTRODUCTION

The objective of this project is to ease the analysis of the log files and to infer as much as information from the log files. A user may input a web server log file to the program and wait for it to run the analysis of the data in the log file. After the analysis it returns visual data representing the data like IP address and the frequency of visit from the IP address. User-Agents used to visit the website and the method of requests made by the clients and frequency of the request methods. The output is not shown as traditional tables, but are visualized using graphs which makes it easier to consume the given information. The interface is a GUI in a windows based application which is a platform-independent because a platform-independent programming language is used to create the application. The application serves to be robust, which means that it could handle huge log files. The final product will have no more features that are specified in this document. This document describes the functional and non-functional requirements of the “Web Server Log Analysis System”. This application “Web Server Log Analysis System” is developed to help Security administrators, developers and Incident responders to remove their burden of looking up the logs manually.

OVERALL DESCRIPTION

The user launches the application, then by clicking on the Choose File button he could browse through the file system to find the log file that needs to be analyzed, then he clicks on the Get Results button to start the analysis. Then the application starts reading the file line by line and perform analysis on them. After it has done with the analysis, the graphs are generated based on the analysis performed and displayed to the user also several other output formats are available like the data is presented in the form of tables and we could export the results to a CSV or a HTML file for later analysis. From the displayed graphs and tables, the users may easily infer information from the log file. The CSV files makes it easier for the user to consume the data programmatically on a later time. The speed of the program depends on the size of the log file, the speed goes inversely proportional to the size of the log file.

SPECIFIC REQUIREMENTS

3.1 Functional Requirements

Easy to use – Provides a user-friendly GUI with very minimal input.

Analysis – Collects all data from the log file to provide statistics.

Output – Provide visual outputs in form of graphs and table, for efficient inference.

Looks – The application is visually appealing.

No Restrictions – One with access to log files can use this application.

3.2 Non-Functional Requirements

Performance – It generates statistics faster depending upon the file size.

Usability – People with no prior experience with log files can use the app.

Scalability – Process multiple log files in a single run of the application.

3.3 External Interface Requirements

3.3.1 User Interfaces

The application is user-friendly menu based interface. It has 2 simple screens as follows.

1. The Initial Screen

This is the screen where the log file that is to be analyzed is selected.

There is a button provided to browse the filesystem to select the log file.

Another button to generate the output when clicked.

2. The Output Screen

This screen shows the output of the analysis, this is filled with the graphs and tables from the analysis.

3.3.2 Hardware Interfaces

Screen resolution of at least 800×600 required for proper and complete rendering of the graphs. Higher resolutions would be better.

Standalone systems, no network connection required.

3.3.3 Software Interfaces

Any kind of Operating System(Linux, Unix, Windows, Mac OS).

Python3.8 Installed.

3.4 Performance Requirements

The application should be able to handle large log files within seconds.

Graph generation should be processed quickly.

The application must support multiple files in a single run.

The application is platform independent and consumes less space.

3.5 Design Constraints

Simple and easy to use with user-friendly experience.

The application is visually appealing.

The graphs generated allow us to infer information with ease.

The data provided as output is accurate.

The application runs on all standalone systems with python.

3.6 Software System Attributes

The application can handle multiple files in the single run.

The system is robust to analyze huge log files.

The application is portable and can run on any standalone system.

The application is easy to use and can be used without any knowledge of the system.

UML ANALYSIS MODEL

4.1 USE CASE DIAGRAM

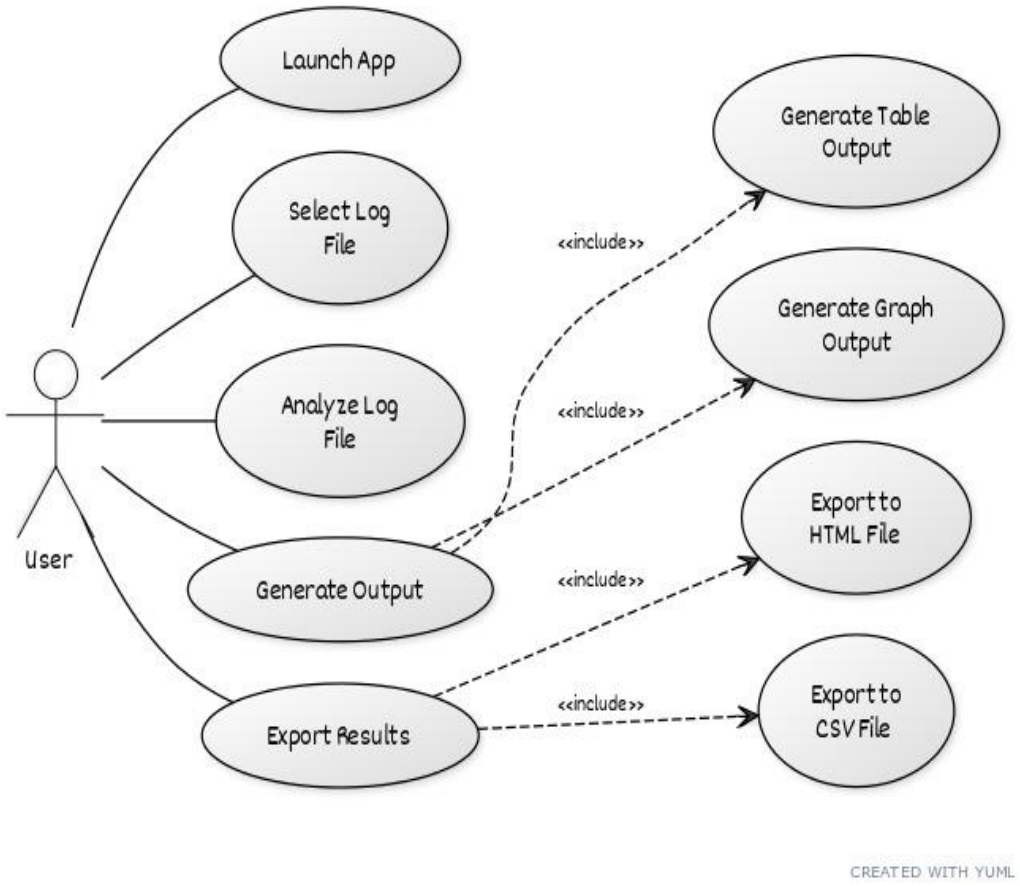


Figure 4.1 : Use Case Diagram

4.2 CLASS DIAGRAM

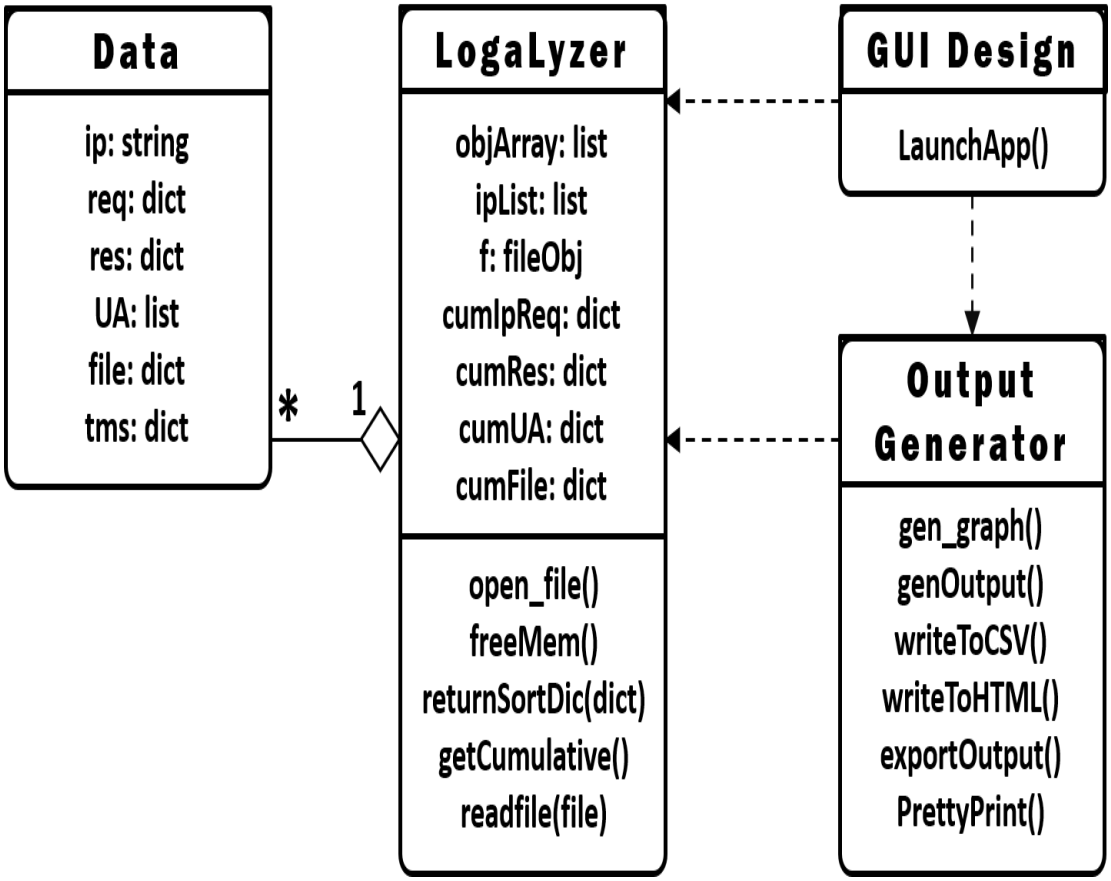


Figure 4.2 : Class Diagram

4.3 SEQUENCE DIAGRAM

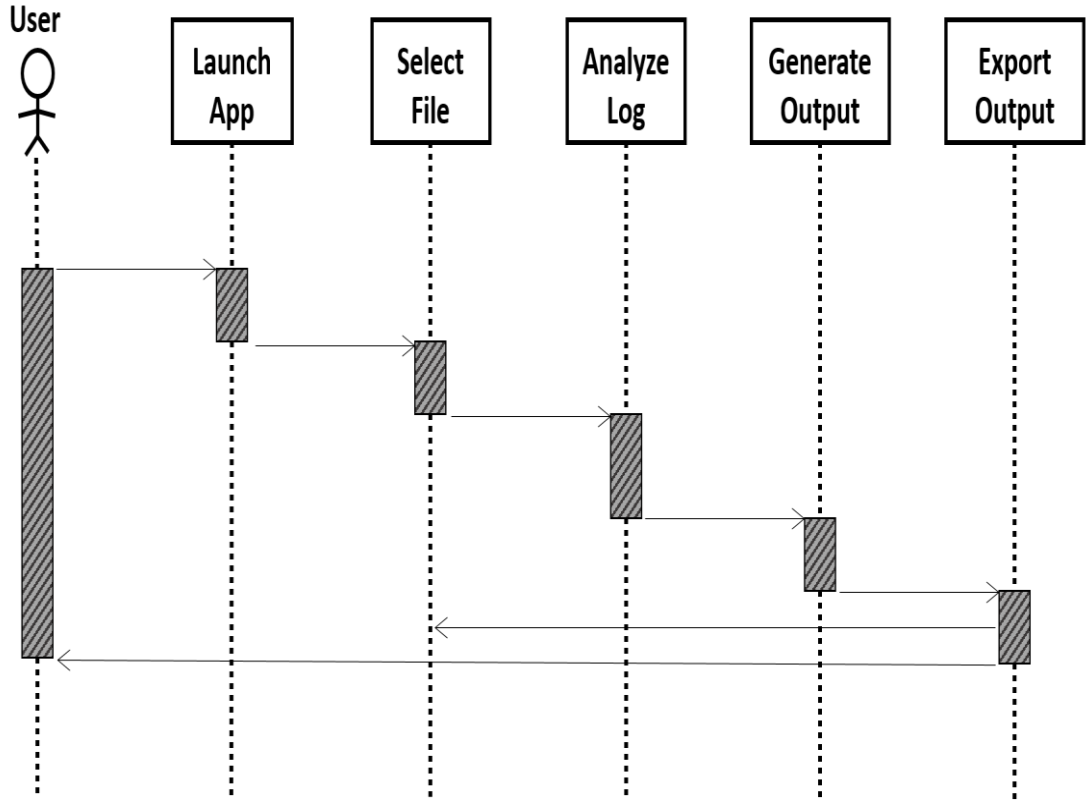


Figure 4.3 : Sequence Diagram

4.4 ACTIVITY DIAGRAM

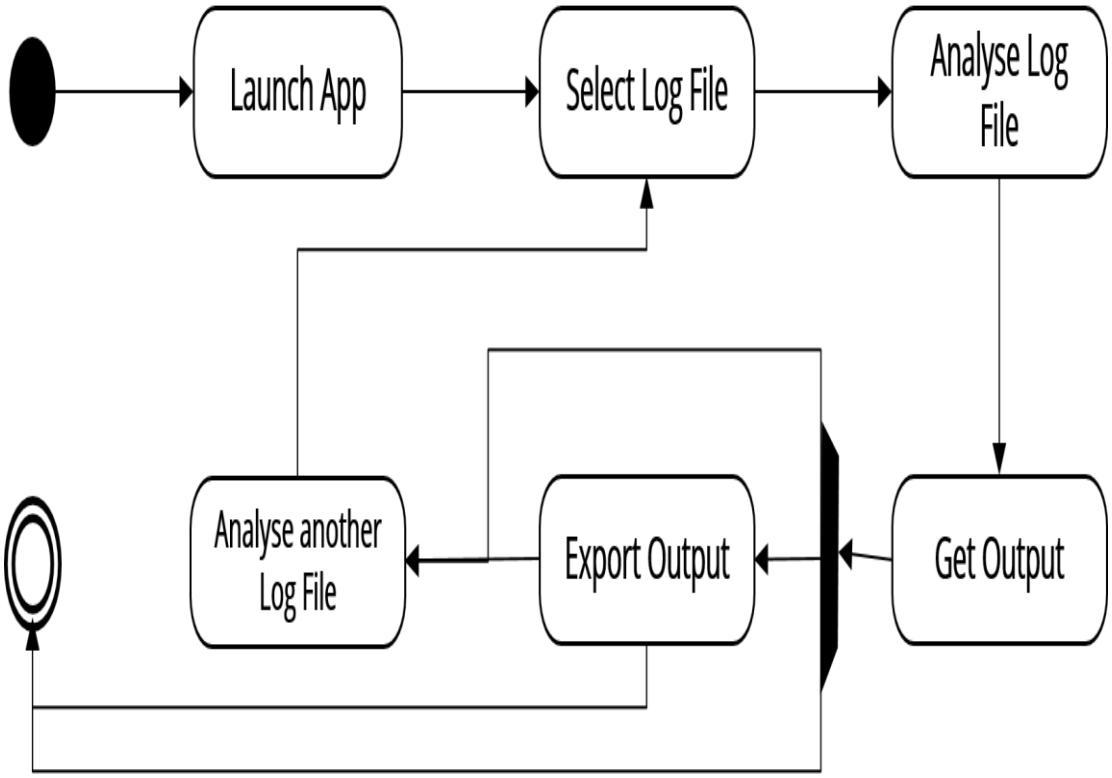


Figure 4.4 : Activity Diagram

PROJECT DESIGN

The objective of this project is to develop an application that analyzes the log file from Web servers. This application could analyze logs from the Apache Web server.

The Web Server Log

The log file generated by default installation of Apache is as follows,

```
78.172.44.50 - - [19/Jan/2015:12:08:08 +0000] "GET /run
HTTP/1.1" 200 83 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS
X 10.10; rv:35.0) Gecko/20100101 Firefox/35.0"
```

Breaking down it into parts,

"*78.172.44.50*" – It is the IP address that is used to access the web server.

"-" - Identity check when authentication is enabled on the server.

"-" - The username used to access the authenticated content on the server.

"*[19/Jan/2015:12:08:08 +0000]*" – Timestamp, the time which the request made.

"*GET / HTTP/1.1*" – The request method used to access the resource.

"*200*" – The status cide returned by the server while accessing a resource.

"*83*" – The size of the response returned to the user by the server.

"-" - The Referer, the site which referred the user to our site.

***"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:35.0) Gecko/20100101 Firefox/35.0"* –** The User-Agent used by the user to access the server.

Our project reads a log file with lot of lines similar to this. Then save all the data in variables and generate graphs from all the data obtained from the file.

The File Selection Phase

First the program is launched, it opens the user-friendly GUI of the application. In the GUI there is a Choose File button to select the log file for analysis. After the button is pressed, the filesystem browser opens allowing us to select the log file to proceed for the analysis. Then the path is stored in a variable and the analysis on the file starts leading us to the next phase.

The Analysis Phase

This phase of the application follows up after selecting the file to analyze and proceeds to the analysis. The file is read line by line to avoid memory overhead. First a line is read and is split by the spaces. Then IP address(String), Timestamp (dictionar -y), Request method(dictionary), Status code(dictionary), the Accessed file (dictionar -y) and User-Agent(list) are obtained and stored in seperate variables after the split. First the IP address is appended to a list and a Data Object is created which stores all the above said variables as class variables. Then this data object is appended to another seperate list. While reading the next line, first the IP address list is checked to see if the IP is present, if it is present, the data according to the object is updated(the main reason why the variables are initialized as dictionaries). If it is not present, then the process of initialization occurs. The reason to initialize the User-Agent as a list is that one user may use the same browser, when he uses a large number of User-Agents within a time, it is an alarm of a DoS attack, also may serve as a work for the development team to tailor pages for that specific UA. This process is repeated until the whole file is read. Then the application moves on to the next phase.

The Output Visualization Phase

The list of data objects is iterated through one-by-one and for a given IP address, the total number of requests made is found out by adding all the values in the Requests dictionary. The variables for which the graphs are requested in the first phase is stored in variables and using those variables the graphs are generated.

When the Get Results button is pressed, the graph gets displayed to the screen and the tables from the data is also displayed to the console. There is an Export Results button provisioned at the GUI, if that button is clicked, the program creates CSV and HTML files of the results. This may be consumed later.

The final thing is left to the user, he may infer the necessary information from the graphs. It may either be used for Incident Response, SEO or for further development of the deployed application depending upon the type of user running the application.

CODE

1. The GUI Module

```
#!/usr/bin/env python3

"""
FrontEnd Module
"""

import tkinter as tk

class GuiDesign:
     """
    GUI Design Class gets in input as the Log Object and
Output Object.
    It ties all three modules of the application together.
    """

     def __init__(self, logObj, graphObj):
          self.log = logObj
          self.op  = graphObj

     def launchApp(self):
          """
         This function Launches the GUI Window for the
application.

         Returns
         -------
         None.

         """
```

```
        window = tk.Tk()  # Starting tkinter window
        window.title('Welcome!')  # Setting window title
        window.configure(bg='#293d3d')
        label_head = tk.Label(window, text='LogaLyzer',
width=15, underline=4, padx=8, pady=8, borderwidth=10,
highlightthickness=3, anchor='center', relief='ridge',
fg='#003d66', bg='#80ccff', font=('Russo One',
20)).grid(row=0,column=0, padx=20, pady=25, columnspan=3)
# Displaying heading label
        label_getfile = tk.Label(window, text='Kindly
choose the log file:', font=('verdana', 9,
'bold'),fg='#ffffff', bg='#293d3d',
anchor='w').grid(row=1,column=0, padx=15)

        fileNameSetButton = tk.Button(window,
text='Choose File', font=("Open Sans", 9, "bold"),
borderwidth=5, bg='#b3e0ff', command=lambda:
self.log.open_file(), cursor='hand2', padx=5, pady=5,
anchor='w').grid(row=1, column=1, padx=5, pady=0)  #
Setting the filename to variable

        label_getfile = tk.Label(window, text='Click for
the results:', font=('verdana', 9, 'bold'),fg='#ffffff',
bg='#293d3d', anchor='w').grid(row=2,column=0, padx=15)
        graphGenButton = tk.Button(window, text='Get
Results', font=("Open Sans", 9, "bold"), borderwidth=5,
bg='#b3e0ff', command=self.op.genOutput, cursor='rtl_logo',
padx=5, pady=5).grid(row=2, column=1, padx=20, pady=20)

        window.mainloop()
```

2. The Analysis Module

```
#!/usr/bin/env python3

"""
Analysis Module
"""
from tkinter import filedialog

class Data:
    """
    Data class stores the data from the log file
    """
    def __init__(self, ip):
        self.ip   = ip
        self.req  = {}
        self.res  = {}
        self.UA   = []
        self.file = {}
        self.tms  = {}

    def __repr__(self):
        return self.ip

class LogLyzer:
    """
    LogLyzer class reads the log file and saves all the data
in easily
    consumable form
    """

    objArray = []
    ipList   = []
```

```
        f        = None
        ipReq    = {}
        cumIpReq = {}
        cumReq   = {}
        cumRes   = {}
        cumUA    = {}
        cumFile  = {}
        cumTms   = {}


        def open_file(self):
            """
            This function prompts the user to select the file
    that is to be analyzed
            in the GUI window.

            Returns
            -------
            Data stored in class variables

            """
            self.f = filedialog.askopenfile(mode='r')
            self.readFile(self.f)
            self.getCumulative()

        def freeMem(self):
            """
            This functions clears all variables makes the
    application ready for the
            next run.

            Returns
            -------
            None.
```

```
        """
        self.objArray.clear()
        self.ipList.clear()
        self.f = None
        self.ipReq.clear()
        self.cumIpReq.clear()
        self.cumReq.clear()
        self.cumRes.clear()
        self.cumUA.clear()
        self.cumFile.clear()
        self.cumTms.clear()

    def returnSortDic(self, dic):
        """
        This functions sorts a given dictionary in
descending order based on values.

        Parameters
        ----------
        dic : Dictionary
            Unsorted dictionary

        Returns
        -------
        dic : Dictionary
            A dictionary sorted in descending order based on
values.

        """
        dic = sorted(dic.items(), key = lambda
kv:(kv[1],kv[0]))
        dic.reverse()
        return dict(dic)
```

```
    def getCumulative(self):
        """
        This function reads all the Data Objects and
cumulates it, total requests
        responses, File access and User-Agents are
calculated and stored in class
        variables in descending order of frequencies.

        Returns
        -------
        Data is stored in class variables

        """
        for obj in self.objArray:
            self.cumIpReq[obj.ip]                          =
sum(list(obj.req.values()))
            for k,v in obj.req.items():
                self.cumReq[k] = self.cumReq.get(k,0) + v
            for k,v in obj.res.items():
                self.cumRes[k] = self.cumRes.get(k,0) + v
            for ua in obj.UA:
                self.cumUA[ua] = self.cumUA.get(ua,0) + 1
            for k,v in obj.file.items():
                self.cumFile[k] = self.cumFile.get(k,0) + v
            #for k,v in obj.tms.items():
            #    self.cumTms[k] = self.cumTms.get(k,0) + v
        self.ipReq = self.cumIpReq
        self.cumIpReq = self.returnSortDic(self.cumIpReq)
        self.cumReq   = self.returnSortDic(self.cumReq)
        self.cumRes   = self.returnSortDic(self.cumRes)
        self.cumUA    = self.returnSortDic(self.cumUA)
        self.cumFile  = self.returnSortDic(self.cumFile)
```

```
    def readFile(self, f):
        """
        This function reads the file line by line and groups
data based on IP address
        on different variables

        Parameters
        ----------
        f : file object
            The file object to read data for analysis

        Returns
        -------
        Data stored in Class variables.

        """
        while True:
            line = f.readline().replace("\"","").strip()
            if(not line):
                break
            else:
                temp = line.split()
                if( temp[0] not in self.ipList ):
                    self.ipList.append(temp[0])
                    obj = Data(temp[0])
                    obj.req[temp[5]]                        =
obj.req.get(temp[5],0) + 1
                    obj.res[temp[8]]                        =
obj.res.get(temp[8],0) + 1
                    obj.file[temp[6]]                       =
obj.file.get(temp[6],0) + 1
                    obj.UA.append(' '.join(temp[11:]))
```

```
                    date                                    =
temp[3].replace("[","").split(":")[0]
                    hr   = temp[3].split(":")[1]
                    obj.tms[date] = {hr:1}

#obj.tms[temp[3].replace("[","").split("/")[0]]           =
obj.tms.get(temp[3].replace("[",""),0) + 1
                    self.objArray.append(obj)
                else:
                    index = self.ipList.index(temp[0])
                    self.objArray[index].req[temp[5]]     =
self.objArray[index].req.get(temp[5], 0) + 1
                    self.objArray[index].res[temp[8]]     =
self.objArray[index].res.get(temp[8], 0) + 1
                    self.objArray[index].file[temp[6]]    =
self.objArray[index].file.get(temp[6], 0) + 1
                    if(   '   '.join(temp[11:])   not   in
self.objArray[index].UA ):
                        self.objArray[index].UA.append('
'.join(temp[11:]))
                    date                                    =
temp[3].replace("[","").split(":")[0]
                    hr   = temp[3].split(":")[1]
                    if(date                                in
self.objArray[index].tms.keys()):
                        self.objArray[index].tms[date][hr]
= self.objArray[index].tms[date].get(hr,1)


#self.objArray[index].tms[temp[3].replace("[","").split(":"
)[0]]                                                       =
self.objArray[index].tms.get(temp[3].replace("[",""),  0)  +
1
```

3. The Output Generator Module

```
#!/usr/bin/env python3

"""
Output Module
"""
import csv
import os
import time
import matplotlib.pyplot as plt
from tabulate import tabulate
#plt.rcParams["figure.figsize"] = [4,6]

class outputGenerator:

     def __init__(self, logObj):
          self.log = logObj

     def gen_graph(self):
          """
          This function generates graphs based on the
statistics presented from the
          logalyzer module

          Returns
          -------
          None.

          """
          fig,ax = plt.subplots()
```

```
            ax.bar(self.log.ipReq.keys(),
self.log.ipReq.values())
            plt.xticks(rotation=75)
            #fig.set_figheight(6)
            #fig.set_figwidth(5)
            fig.show()
            fig.tight_layout(pad=2.5)

        def genOutput(self):
            """
            This function calls all other functions based on the
output requested by
            the user.

            Returns
            -------
            None.

            """
              self.prettyPrint()
              self.gen_graph()
              self.exportOutput()
              self.log.freeMem()


        def writeToCsv(self, filename, data, header):
            """
             This function writes the data from the logalyzer
module to seperate CSV files

             Parameters
             ----------
             filename : str
                 The name of the file to write.
```

```
        data : dict
            A dictionary of the data that is to be written
to the CSV file.
        header : list
            The data that is to be written as the header of
the CSV file.

        Returns
        -------
        None.

        """
         with open(filename,"w") as f:
              csv_w = csv.writer(f)
              csv_w.writerow(header)
              zipObj = zip(data.keys(),data.values())
              for row in zipObj:
                   csv_w.writerow(list(row))


    def writeToHTML(self):
         """
         This function writes the data from the Logalyzer
module to a HTML file

        Returns
        -------
        None.

        """
         htmlPage = """
    <!DOCTYPE html>
    <html>
    <head>
```

```
<title>
Logalyzer - Results
</title>
</head>
<body>
<center>
<i> <h3> IP Addresses and The Frequency </h3> </i>
"""
     # Code to add rows
     htmlPage += """
<i> <h3> Request Method and The Frequency </h3> </i>
"""
     # Code to add rows
     htmlPage += """
<i> <h3> Status Code and The Frequency </h3> </i>
"""
     # Code to add rows
     htmlPage += """
<i> <h3> User Agent and The Frequency </h3> </i>
"""
     # Code to add rows
     htmlPage += """
<i> <h3> Accessed File and The Frequency </h3> </i>
"""
     # Code to add rows
     htmlPage += """
</center>
</body>
</html>
"""
     with open("output.html","w") as f:
          f.write(htmlPage)
```

```
    def exportOutput(self):
        """
        This function creates a directory based on
timestamp and calls other functions to
       write data from Logalyzer module to HTML and CSV
files.

        Returns
        -------
        None.

        """
        dirName = time.asctime().replace(" ","-
").replace(":","-")
        if(not os.path.isdir(dirName)):
            os.mkdir(dirName)
        os.chdir(dirName)
        self.writeToCsv("IP-and-Req.csv",
self.log.cumIpReq, ["IP Address", "Req Count"])
        self.writeToCsv("Request-Method-and-
Count.csv",self.log.cumReq, ["Request Method","Frequency"])
        self.writeToCsv("Response-Status-and-Count.csv",
self.log.cumRes, ["Status Code", "Frequency"])
        self.writeToCsv("UA-and-
Count.csv",self.log.cumUA, ["User-Agent","Frequency"])
        self.writeToCsv("File-Access-and-Count.csv",
self.log.cumFile, ["File Accessed", "Frequency"])
        self.writeToHTML()


    def prettyPrint(self):
        """
```

```
        This function pretty prints the data from
Logalyzer module on Terminal.

        Returns
        -------
        None.

        """
         print("\n==============    IP,    Request    Count
=======================")
         print(tabulate(zip(self.log.cumIpReq.keys(),
self.log.cumIpReq.values()),    headers=["IP",    "Request
Count"], tablefmt="fancy_grid"))
         print("\n==============  Request  Method,  Count
====================")

    print(tabulate(zip(self.log.cumReq.keys(),self.log.cum
Req.values()),    headers=["Request    Method",    "Count"],
tablefmt="fancy_grid"))
         print("\n==============    Status    Code,    Count
======================")
         print(tabulate(zip(self.log.cumRes.keys(),
self.log.cumRes.values()),     headers=["Status     Code",
"Count"], tablefmt="fancy_grid"))
         print("\n==============    User    Agent,    Count
=======================")

    print(tabulate(zip(self.log.cumUA.keys(),self.log.cumU
A.values()),     headers=["User     Agent",     "Count"],
tablefmt="fancy_grid"))
         print("\n==============  File  Accessed,  Count
====================")
```

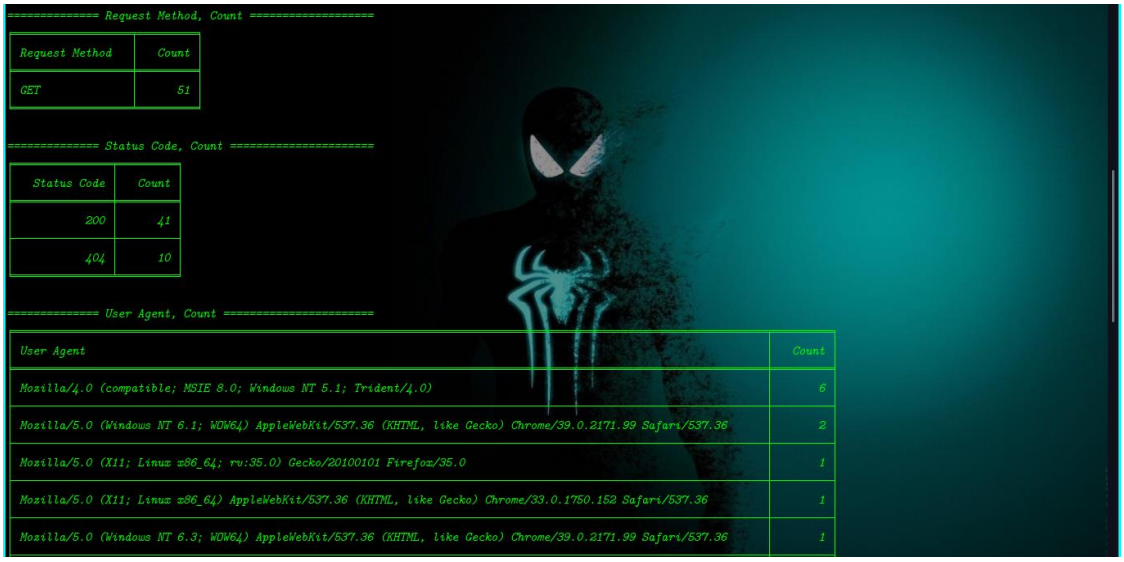
```
          print(tabulate(zip(self.log.cumFile.keys(),
self.log.cumFile.values()),    headers=["File    Accessed",
"Count"], tablefmt="fancy_grid"))
```

4. The Driver Module

```
#!/usr/bin/env python3

"""
Main Function #Driving Code
"""

import logaLyzer as ll
import outputGenerator as og
import frontEndGUI as fg

log   = ll.LogLyzer()
graph = og.outputGenerator(log)
gui   = fg.GuiDesign(log, graph)

gui.launchApp()
```

OUTPUT

Figure 7.1 : Logalyzer Launching Screen



========== Request Method, Count ==========

Request Method	Count
GET	51

========== Status Code, Count ==========

Status Code	Count
200	41
404	10

========== User Agent, Count ==========

User Agent	Count
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)	6
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.99 Safari/537.36	2
Mozilla/5.0 (X11; Linux x86_64; rv:35.0) Gecko/20100101 Firefox/35.0	1
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.152 Safari/537.36	1
Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.99 Safari/537.36	1

Figure 7.2 : Table Output from Logalyzer

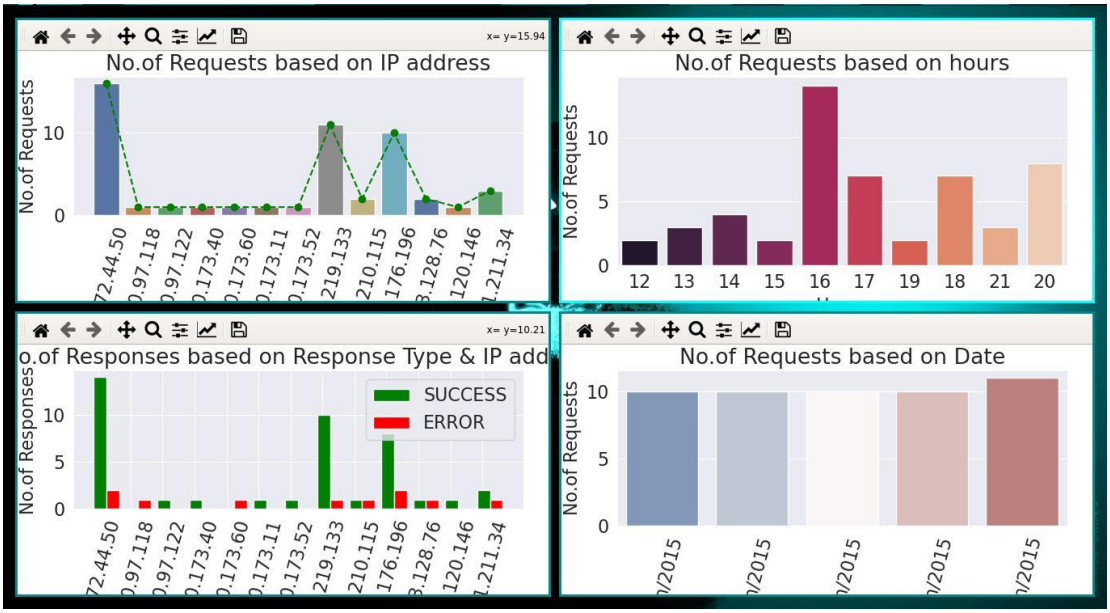


Figure 7.3 : Graph Output from Logalyzer

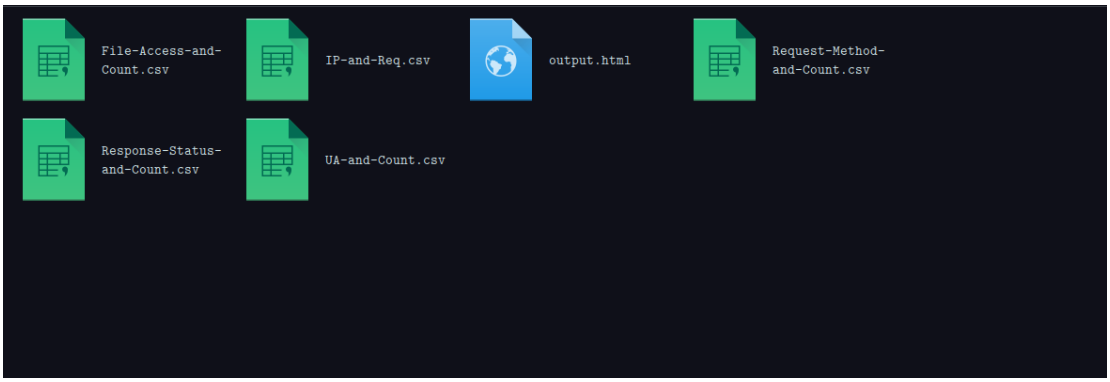


Figure 7.4 : Files Output from Logalyzer

CONCLUSION

Log analysis is an important task for System administrators, Security administrators and even for developers at times. Our project “Web Server Log Analysis System” makes the tasks of these administrators simpler by just getting in the log file as input and visualizing things present in the log file. This application would reduce the burden of those administrators task of manually going through the log files. All these presented using a simple to use and user-friendly GUI.

REFERENCES

1. Satya Prakash Singh, Meenu (2017), Analysis of Web Site Using Web Log Expert Tool Based on Web Data Mining, International Conference of Innovations in Information Embedded and Communication Systems(ICIIECS).

2. Dilip Singh Sisodia, Shrish Verma (2012), Web Usage Pattern Analysis Through Weblogs: A Review, Ninth International Joint Conference on Computer Science and Software Engineering(JCSSE).

3. AscToHTM(2005), A Web Server Log File Explained, https://www.jafsoft.com/searchengines/log_sample.html, 24 Oct 2020.

4. Nihuo Software Inc, Log File Sample Explained, https://www.loganalyzer.net/log-analysis-tutorial/log-file-sample-explain.html, 24 Oct 2020.