Project Report: Apache Log Analyzer and Visualizer

Parth Vaity

April 29, 2025

1 Introduction

This project is a web-based Apache log analyzer and visualizer. It provides users with the ability to upload raw Apache log files, convert them to structured CSV format, and visualize various statistics such as event distributions, level states, and time-based occurrences for any selected range of time. A self-plot feature is also implemented, allowing users to write their own Python code for custom visualizations directly in the web interface. Users also have the ability to sort and filter the files in many ways and then download them.

2 Requirements

- Python 3.10+
  To install python, run the following command on your terminal

  ```
  sudo apt install python3
  ```

- Libraries and Modules:
  - Flask
  - Matplotlib
  - NumPy
  - os
  - subprocesses

  To install the required modules, run the following command on your terminal:

  ```
  pip install flask matplotlib numpy
  ```

3 Running Instructions

1. Ensure that all required modules are installed.
2. Place your Apache log file in a known location.
3. Start the Flask server by running the following command in your terminal:

   ```
   python3 webp.py
   ```

4. Open a browser and navigate to
   http://127.0.0.1:5000
   Or simply click on the link that appears in your terminal
5. Upload a log file and explore the graph generation features.

4 Website Layout

- **Upload Log File Page:** Upload Apache `.log` files.
- **Upload Successful Page:** Provides links to generated CSV file and graph options after successful parsing of the file.
  - **View Log File Page:** View the uploaded log file as a table, with the ability to sort, filter and download its CSV file in different ways.
  - **Events Logged with Time Graph Page:** View line graphs for events logged with time for different events and different ranges of time.
  - **Level State Distribution graph Page:** View pie charts for level state distribution for different ranges of time.
  - **Event Code Distribution Graph Page:** View bar graphs for event code distributions for different ranges of time.
  - **Self Plotting of Graph Page:** Enter your personal Python code and view the output graphs for the initial data for different ranges of time.

5 Modules Used

- **Flask:** Used for the Web server framework.
- **Matplotlib:** Used for generating graphs and plots.
- **NumPy:** Efficient handling of timestamps and array-based calculations.
- **Subprocess:** For calling the log-to-CSV converter shell script.
- **Os:** To create directories, construct file paths, saving, and accessing files.

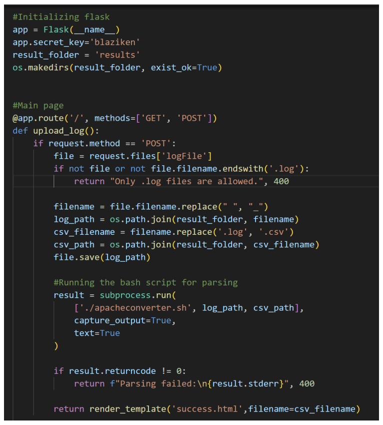
6 Directory Structure

- `webp.py`: Main Flask server file.
- `selfplot.py`: Contains user-editable plotting function.
- `eventcode.py`, `levelgraph.py`, `eventvstimegraph.py`: Logic for generating various graphs.
- `templates/`: All HTML templates.
  * **eventcodegraphs.html:** Page showing Event code distribution graphs.
  * **eventvstimegraphs.html:** Page showing Event logged with Time graphs.
  * **levelgraphs.html:** Page showing Level State Distribution Charts.
  * **logdisplay.html:** Initial page for uploading the log file.
  * **selfplot.html:** Page for Self plotting the graph based on python input.
  * **success.html:** Page afer successful parsing of file.
- `static/`: Stores generated plots, CSS and Javascript files.
  * **datetimeinit.js:** Initialises date and time for input in the graph pages and logupload.html page.
  * **displaylogs.js:** Handles all the filtering, sorting and downloading on the logdisplay.html page.
  * **display.css:** Handles CSS for logdisplay.html page.
  * **graphformat.css:** Handles CSS for all the pages containing graphs.
  * **success.css** Handles CSS for the success.html page.
- `results/`: Stores uploaded log files and converted CSVs.
- `timeparser.py`: Helper for parsing csv time into numpy default timeformat.
- `apacheconverter.sh`: Bash script to convert logs to CSV.

7 Basic and Advanced Features

7.1 Flask server file

This file runs in the background at all times managing the workflow of all the webpages, interconnecting them to each other for proper flow of data from one to other.

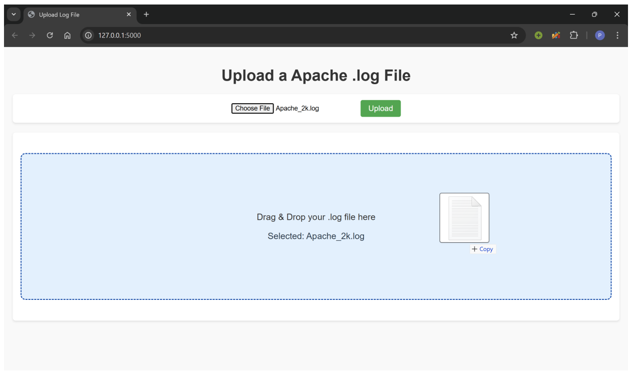


```python
#Initializing flask
app = Flask(__name__)
app.secret_key='blaziken'
result_folder = 'results'
os.makedirs(result_folder, exist_ok=True)


#Main page
@app.route('/', methods=['GET', 'POST'])
def upload_log():
    if request.method == 'POST':
        file = request.files['logFile']
        if not file or not file.filename.endswith('.log'):
            return "Only .log files are allowed.", 400

        filename = file.filename.replace(" ", "_")
        log_path = os.path.join(result_folder, filename)
        csv_filename = filename.replace('.log', '.csv')
        csv_path = os.path.join(result_folder, csv_filename)
        file.save(log_path)

        #Running the bash script for parsing
        result = subprocess.run(
            ['./apacheconverter.sh', log_path, csv_path],
            capture_output=True,
            text=True
        )

        if result.returncode != 0:
            return f"Parsing failed:\n{result.stderr}", 400

        return render_template('success.html',filename=csv_filename)
```

Figure 1: Flask python script

7.2 Upload Apache log files

1. **Basic features:** The page provides the basic option of uploading a file by clicking the "Choose file" button and selecting the file from your computer directories.
2. **Advanced features:** The page provides with the option to drag and drop the file directly from your computer to the designated box area to upload it.



Upload a Apache .log File

Apache_2k.log

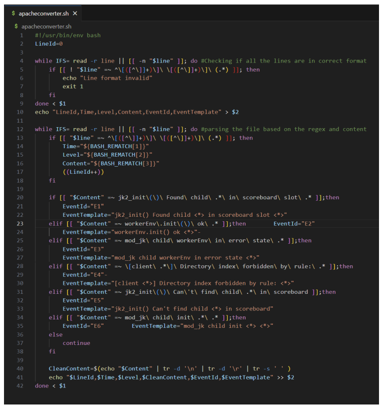
Drag & Drop your .log file here

Selected: Apache_2k.log

Figure 2: Loguploading page

7.3 Bash Parsing Script

After uploading the log file a bash script runs in the back converting the log file into a csv file. It uses regex matching to parse the file into a csv file based on event templates.



```bash
#!/usr/bin/env bash
LineId=0

while IFS= read -r line || [[ -n "$line" ]]; do #Checking if all the lines are in correct format
    if [[ ! "$line" =~ ^\[([^\]]+)\]\ \[([^\]]+)\]\ (.*) ]]; then
        echo "Line format invalid"
        exit 1
    fi
done < $1
echo "LineId,Time,Level,Content,EventId,EventTemplate" > $2

while IFS= read -r line || [[ -n "$line" ]]; do #parsing the file based on the regex and content
    if [[ "$line" =~ ^\[([^\]]+)\]\ \[([^\]]+)\]\ (.*) ]]; then
        Time="${BASH_REMATCH[1]}"
        Level="${BASH_REMATCH[2]}"
        Content="${BASH_REMATCH[3]}"
        ((LineId++))
    fi

    if [[ "$Content" =~ jk2_init\(\)\ Found\ child\ .*\ in\ scoreboard\ slot\ .* ]];then
        EventId="E1"
        EventTemplate="jk2_init() Found child <*> in scoreboard slot <*>"
    elif [[ "$Content" =~ workerEnv\.init\(\)\ ok\ .* ]];then        EventId="E2"
        EventTemplate="workerEnv.init() ok <*>"-
    elif [[ "$Content" =~ mod_jk\ child\ workerEnv\ in\ error\ state\ .* ]];then
        EventId="E3"
        EventTemplate="mod_jk child workerEnv in error state <*>"
    elif [[ "$Content" =~ \[client\ .*\]\ Directory\ index\ forbidden\ by\ rule:\ .* ]];then
        EventId="E4"-
        EventTemplate="[client <*>] Directory index forbidden by rule: <*>"
    elif [[ "$Content" =~ jk2_init\(\)\ Can\'t\ find\ child\ .*\ in\ scoreboard ]];then
        EventId="E5"
        EventTemplate="jk2_init() Can't find child <*> in scoreboard"
    elif [[ "$Content" =~ mod_jk\ child\ init\ .*\ .* ]];then
        EventId="E6"        EventTemplate="mod_jk child init <*> <*>"
    else
        continue
    fi

    CleanContent=$(echo "$Content" | tr -d '\n' | tr -d '\r' | tr -s ' ' )
    echo "$LineId,$Time,$Level,$CleanContent,$EventId,$EventTemplate" >> $2
done < $1
```

Figure 3: Bash Script

7.4 Success in parsing

The success page opens once the parsing is successful. Here, we get options to select between viewing the final csv file or the predefined graphs that can be formed to analyze the csv file's data.

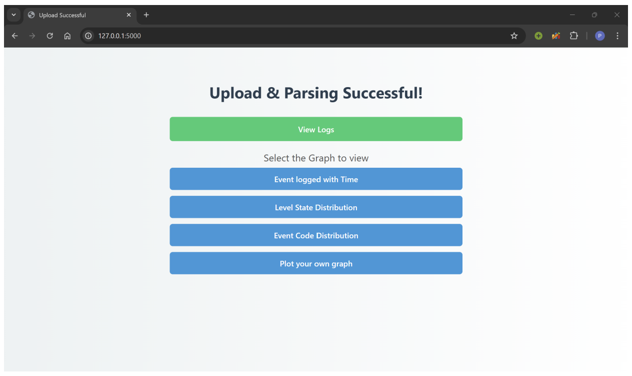
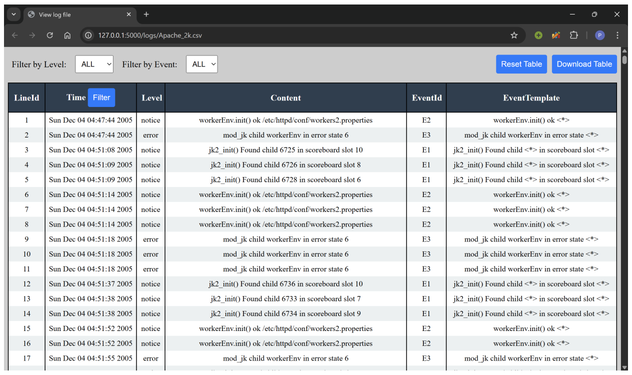


Figure 4: Success page

7.5 Displaying parsed CSV files

1. **Basic features:**The page provides the basic features like:
   (a) This page is accessed by selecting the "View logs" option on the success page. It shows the entire CSV file in a tabular format. The page also provides an option for downloading the entire table as a CSV file.



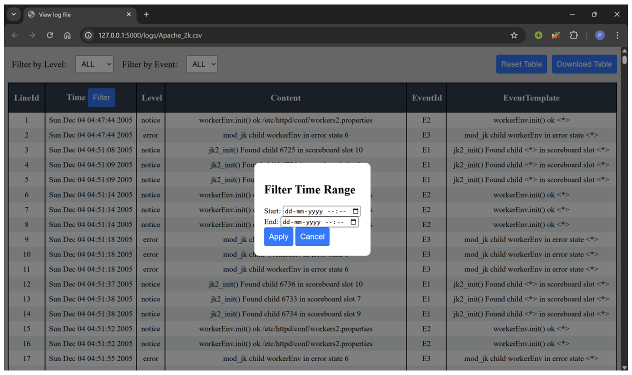
127.0.0.1:5000/logs/Apache_2k.csv

Filter by Level: ALL Filter by Event: ALL

LineId	Time Filter	Level	Content	EventId	EventTemplate
1	Sun Dec 04 04:47:44 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
2	Sun Dec 04 04:47:44 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
3	Sun Dec 04 04:51:08 2005	notice	jk2_init() Found child 6725 in scoreboard slot 10	E1	jk2_init() Found child <*> in scoreboard slot <*>
4	Sun Dec 04 04:51:09 2005	notice	jk2_init() Found child 6726 in scoreboard slot 8	E1	jk2_init() Found child <*> in scoreboard slot <*>
5	Sun Dec 04 04:51:09 2005	notice	jk2_init() Found child 6728 in scoreboard slot 6	E1	jk2_init() Found child <*> in scoreboard slot <*>
6	Sun Dec 04 04:51:14 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
7	Sun Dec 04 04:51:14 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
8	Sun Dec 04 04:51:14 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
9	Sun Dec 04 04:51:18 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
10	Sun Dec 04 04:51:18 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
11	Sun Dec 04 04:51:18 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
12	Sun Dec 04 04:51:37 2005	notice	jk2_init() Found child 6736 in scoreboard slot 10	E1	jk2_init() Found child <*> in scoreboard slot <*>
13	Sun Dec 04 04:51:38 2005	notice	jk2_init() Found child 6733 in scoreboard slot 7	E1	jk2_init() Found child <*> in scoreboard slot <*>
14	Sun Dec 04 04:51:38 2005	notice	jk2_init() Found child 6734 in scoreboard slot 9	E1	jk2_init() Found child <*> in scoreboard slot <*>
15	Sun Dec 04 04:51:52 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
16	Sun Dec 04 04:51:52 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
17	Sun Dec 04 04:51:55 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>

Figure 5: Log display page

   (b) The page also provides with a Time Filter in the Time column which allows the user to set a mintime and maxtime for the data to appear in the table.



127.0.0.1:5000/logs/Apache_2k.csv

Filter by Level: ALL Filter by Event: ALL

Filter Time Range

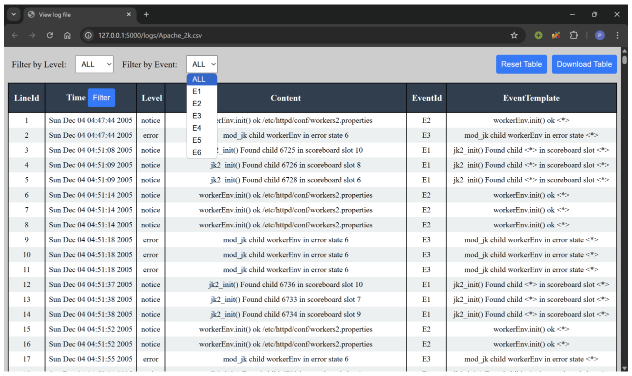
Start: dd-mm-yyyy --:--

End: dd-mm-yyyy --:--

LineId	Time Filter	Level	Content	EventId	EventTemplate
1	Sun Dec 04 04:47:44 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
2	Sun Dec 04 04:47:44 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
3	Sun Dec 04 04:51:08 2005	notice	jk2_init() Found child 6725 in scoreboard slot 10	E1	jk2_init() Found child <*> in scoreboard slot <*>
4	Sun Dec 04 04:51:09 2005	notice	jk2_init() Fou	E1	jk2_init() Found child <*> in scoreboard slot <*>
5	Sun Dec 04 04:51:09 2005	notice	jk2_init() Fou	E1	jk2_init() Found child <*> in scoreboard slot <*>
6	Sun Dec 04 04:51:14 2005	notice	workerEnv.init()	E2	workerEnv.init() ok <*>
7	Sun Dec 04 04:51:14 2005	notice	workerEnv.init()	E2	workerEnv.init() ok <*>
8	Sun Dec 04 04:51:14 2005	notice	workerEnv.init()	E2	workerEnv.init() ok <*>
9	Sun Dec 04 04:51:18 2005	error	mod_jk cl	E3	mod_jk child workerEnv in error state <*>
10	Sun Dec 04 04:51:18 2005	error	mod_jk ch	E3	mod_jk child workerEnv in error state <*>
11	Sun Dec 04 04:51:18 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
12	Sun Dec 04 04:51:37 2005	notice	jk2_init() Found child 6736 in scoreboard slot 10	E1	jk2_init() Found child <*> in scoreboard slot <*>
13	Sun Dec 04 04:51:38 2005	notice	jk2_init() Found child 6733 in scoreboard slot 7	E1	jk2_init() Found child <*> in scoreboard slot <*>
14	Sun Dec 04 04:51:38 2005	notice	jk2_init() Found child 6734 in scoreboard slot 9	E1	jk2_init() Found child <*> in scoreboard slot <*>
15	Sun Dec 04 04:51:52 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
16	Sun Dec 04 04:51:52 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
17	Sun Dec 04 04:51:55 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>

Figure 6: Log display page with time filter

2. **Advanced features:** The page also provides advanced features like:
   (a) Sorting the columns in ascending and descending order on clicking the column headers
   (b) Toolbar at the top consisting of filter options, to filter the table based on different types of Events or different types of Levels.
   (c) The page finally also provides with a reset table option which can be selected at any moment to revert back to the original table.



Filter by Level: ALL   Filter by Event: ALL

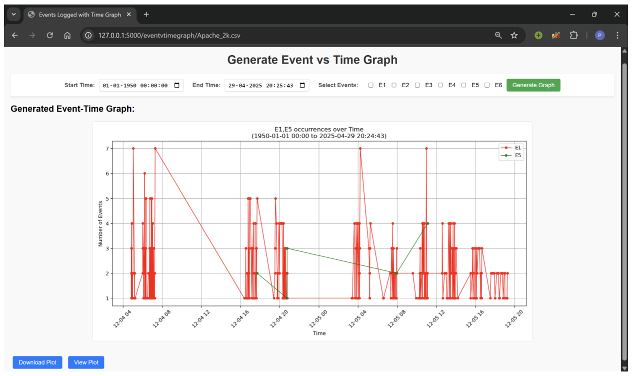
LineId	Time	Level	Content	EventId	EventTemplate
1	Sun Dec 04 04:47:44 2005	notice	rEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
2	Sun Dec 04 04:47:44 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
3	Sun Dec 04 04:51:08 2005	notice	2_init() Found child 6725 in scoreboard slot 10	E1	jk2_init() Found child <*> in scoreboard slot <*>
4	Sun Dec 04 04:51:09 2005	notice	jk2_init() Found child 6726 in scoreboard slot 8	E1	jk2_init() Found child <*> in scoreboard slot <*>
5	Sun Dec 04 04:51:09 2005	notice	jk2_init() Found child 6728 in scoreboard slot 6	E1	jk2_init() Found child <*> in scoreboard slot <*>
6	Sun Dec 04 04:51:14 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
7	Sun Dec 04 04:51:14 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
8	Sun Dec 04 04:51:14 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
9	Sun Dec 04 04:51:18 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
10	Sun Dec 04 04:51:18 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
11	Sun Dec 04 04:51:18 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>
12	Sun Dec 04 04:51:37 2005	notice	jk2_init() Found child 6736 in scoreboard slot 10	E1	jk2_init() Found child <*> in scoreboard slot <*>
13	Sun Dec 04 04:51:38 2005	notice	jk2_init() Found child 6733 in scoreboard slot 7	E1	jk2_init() Found child <*> in scoreboard slot <*>
14	Sun Dec 04 04:51:38 2005	notice	jk2_init() Found child 6734 in scoreboard slot 9	E1	jk2_init() Found child <*> in scoreboard slot <*>
15	Sun Dec 04 04:51:52 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
16	Sun Dec 04 04:51:52 2005	notice	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init() ok <*>
17	Sun Dec 04 04:51:55 2005	error	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state <*>

Figure 7: Log display page with selectable filters

7.6 Predefined Graphs

The success page provides with three predefined graphs which we can view:

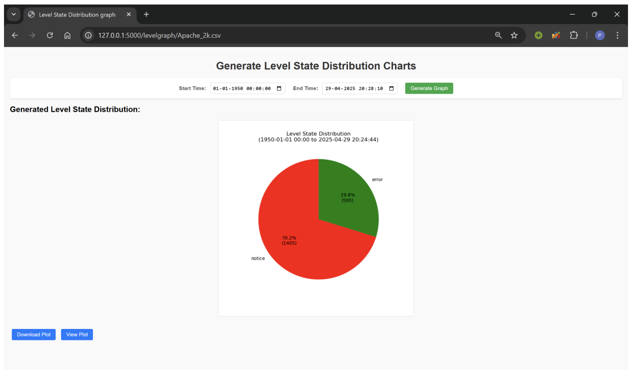
1. **Events Logged With Time Page:** This page provides the user with the option to see the number of occurrences of each event at different timestamps. It allows the user to set the mintime and maxtime as the limits between which the data should be used for plotting of the graph. The page allows the user to download the graph after it plots or view it in a new window. As an Advanced feature, the page also allows the user to merge the data for multiple events into a single graph for better comparisons.


Events Logged with Time Graph
127.0.0.1:5000/eventvtimegraph/Apache_2k.csv
Generate Event vs Time Graph
E1
E2
E3
E4
E5
E6
Generate Graph
Generated Event-Time Graph:
E1,E5 occurrences over Time
(1950-01-01 00:00 to 2025-04-29 20:24:43)
E1
E5
Number of Events
Time
12-04 04
12-04 08
12-04 12
12-04 16
12-04 20
12-05 00
12-05 04
12-05 08
12-05 12
12-05 16
12-05 20
Download Plot
View Plot

Start Time: 01-01-1950 00:00:00   End Time: 29-04-2025 20:25:43   Select Events:

Figure 8: Events Logged with Time graph

2. **Level State Distribution:** This page provides the user with the option to see a pie chart distribution of how frequently different events that have taken place. It allows the user to set the mintime and maxtime as the limits between which the data should be used for plotting of the graph. The page allows the user to download the graph after it plots or view it in a new window.


Level State Distribution graph
127.0.0.1:5000/levelgraph/Apache_2k.csv

Generate Level State Distribution Charts

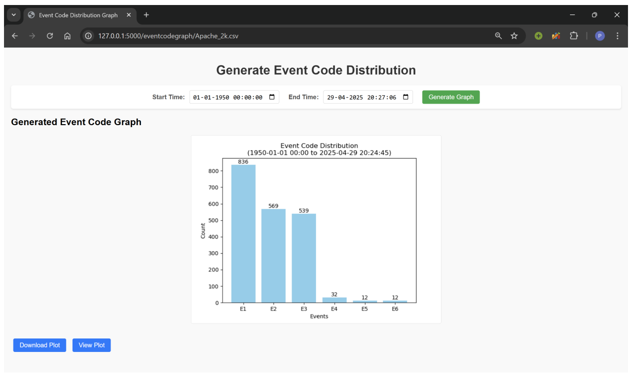
Start Time: 01-01-1950 00:00:00 End Time: 29-04-2025 20:28:10

Generated Level State Distribution:

Level State Distribution
(1950-01-01 00:00 to 2025-04-29 20:24:44)
error
29.8%
(595)
70.2%
(1405)
notice
Download Plot
View Plot

Figure 9: Level Distribution Graph

3. **Event Code Distribution:** This page provides the user with the option to see a bar chart distribution of total number of each of the events that took place. It allows the user to set the mintime and maxtime as the limits between which the data should be used for plotting of the graph. The page allows the user to download the graph after it plots or view it in a new window.


Event Code Distribution Graph
127.0.0.1:5000/eventcodegraph/Apache_2k.csv

Generate Event Code Distribution

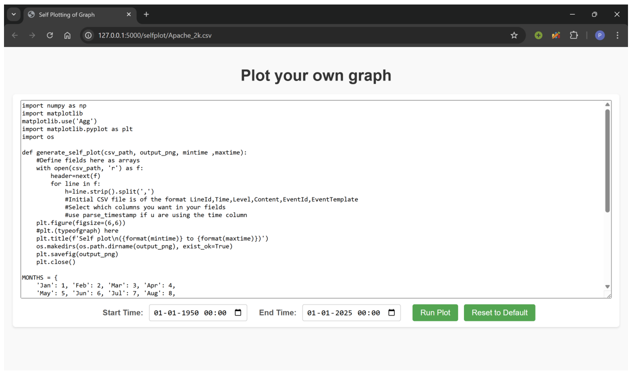
Start Time: 01-01-1950 00:00:00 End Time: 29-04-2025 20:27:06

Generated Event Code Graph

Figure 10: Event Code Distribution

7.7 Self Plotting of Graph

As one of the main advanced features, the success page has a option to navigate to selfplot.html where the user can plot graphs based on the user's input script. On this page the user can enter his own python script based on a template which is already in place to generate his choice of graph. The user gets options to select a mintime and maxtime to set as limits for the data that can be included on his graph. The user also gets option to reset the page back to default incase there has been a mistake in the syntax. Once the plot is generated the user gets options to download the plot to his device. He also gets the option to view it in a newtab.



Plot your own graph

```
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import os

def generate_self_plot(csv_path, output_png, mintime ,maxtime):
    #Define fields here as arrays
    with open(csv_path, 'r') as f:
        header=next(f)
        for line in f:
            h=line.strip().split(',')
            #Initial CSV file is of the format LineId,Time,Level,Content,EventId,EventTemplate
            #Select which columns you want in your fields
            #use parse_timestamp if u are using the time column
    plt.figure(figsize=(6,6))
    #plt.(typeofgraph) here
    plt.title(f'Self plot\n({format(mintime)} to {format(maxtime)})')
    os.makedirs(os.path.dirname(output_png), exist_ok=True)
    plt.savefig(output_png)
    plt.close()

MONTHS = {
    'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4,
    'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
```

Start Time: 01-01-1950 00:00 End Time: 01-01-2025 00:00

Figure 11: Plotting self graphs

8 Project Journey

Key Learnings:

- Integration of back-end and front-end with the use of flask.
- Parsing of files using regex in bash scripts.
- Integrating a Python Editor into HTML.
- Time parsing, Sorting and Filtering of tables using Javascript.
- Managing user code execution securely while keeping an eye out for possible errors.
- Importance of effective code modularisation in large projects.

Challenges:

- Figuring out non complicated ways to handle javascript filtering, sorting and time parsing.
- Learning about the functioning of Flask and its syntax.
- Managing the effective flow of files between front-end and back-end.
- Shortening of code into a concise manner.

Solutions:

– Browsed the web for different types of javascript inbult functions, their syntax and how they function. Took help of lecture notes and different Stack Overflow pages to get around bugs and issues in javascript.

– Watched provided videos and read the documentation to get a further understanding of how Flask functions.

– Tried many smaller examples to test the flow of files between different documents and from the front-end to back-end to understand how different functions operate and how they can be used effectively.

– Used web resources like Stack overflow and general Google searches to find shorter alternatives for long form codes.

9 Bibliography

– Flask Documentation: `https://flask.palletsprojects.com/en/stable/`

– Matplotlib Documentation: `https://matplotlib.org/stable/api/pyplot_summary.html`

– Provided Log Files: `https://github.com/logpai/loghub`

– Provided Flask Tutorials: `https://realpython.com/tutorials/flask/`

– Numpy Documentation: `https://numpy.org/doc/2.2/numpy-user.pdf`

– MZN Web Docs for Javascript: `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

– W3 Schools - Javascript: `https://www.w3schools.com/js/default.asp`

– CSS tricks Guides: `https://css-tricks.com/guides/`

– Many General Stack Overflow Queries and Codes for understanding of Python, Html, and Javascript: `https://stackoverflow.com/`