Project Documentation: Log File Analysis

VIVEKTEJA SAPAVATH

April 29, 2025

Contents

1 Introduction

This project involves a web application capable of processing Apache log files and generating insightful analysis. The system allows users to upload log files, filter entries based on `Level` and `EventId`, and generate custom plots using Python code. The application provides visualization tools that help users understand patterns and trends in their Apache log data through pie charts, line plots, and bar graphs. Additionally, the system offers time-based filtering to narrow analysis to specific timeframes.

2 Requirements

The project requires the following modules and software:

- **Python 3**
- **Flask** – For building the web application
- **Matplotlib** – For generating plots
- **Virtual Environment** – To isolate dependencies

Installation Commands

```
# Create and activate virtual environment
python -m venv venv_name
source venv_name/bin/activate  # On Linux/MacOS
# OR
venv_name\Scripts\activate     # On Windows

# Install required packages
pip install flask
pip install matplotlib
```

3 Running Instructions

1. Clone the repository or extract the project files.
2. Set up and activate the virtual environment using commands from the Requirements section.
3. Install the required packages.
4. Run the server using: `python3 app.py`
5. Open your browser and navigate to: `http://localhost:5000`

4 Website Layout

The website consists of the following pages:

- **Upload Page (upload_page.html):** Allows users to upload an Apache log file. If the uploaded file is invalid, an error message is displayed. The page also includes a link to access the Python plot generator.
- **Results Page (tabular_display.html):** Displays filtered log data and provides insights based on selected criteria. Users can filter the data by Level ([notice], [error]) and EventId (E1-E6), and download the filtered data as a CSV file.
- **Graphs Page (graphs_page.html):** Shows visualizations of the filtered data, including a pie chart of notice vs. error entries, a line plot of entries over time, and a bar graph of event distributions. Users can further filter the data by time range.
- **Plotter Page (plotter.html):** Provides an interface for users to write custom Python code to generate plots using Matplotlib.

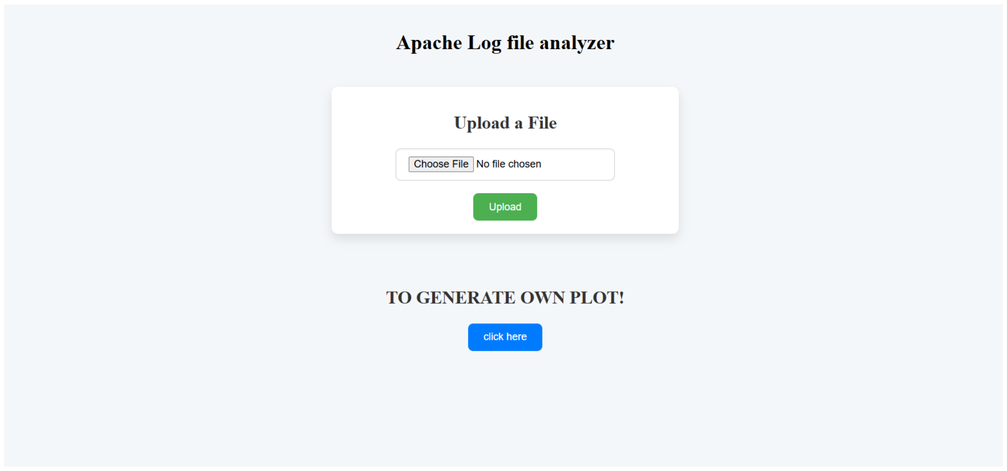


Figure 1: Upload Page (Conceptual)

5 Modules

The following Python libraries and components are used:

- **Flask:** Web framework used to handle routes and serve HTML templates. It processes form submissions, file uploads, and renders the appropriate pages.
- **Matplotlib:** Used for generating dynamic plots based on log data. The application creates pie charts, line plots, and bar graphs, and allows users to write custom plotting code.

- **os:** Used to execute bash scripts from within the Python application, enabling efficient log processing and filtering.
- **time:** Used for performance tracking and timestamps throughout the application.

6 Directory Structure

The project is organized as follows:

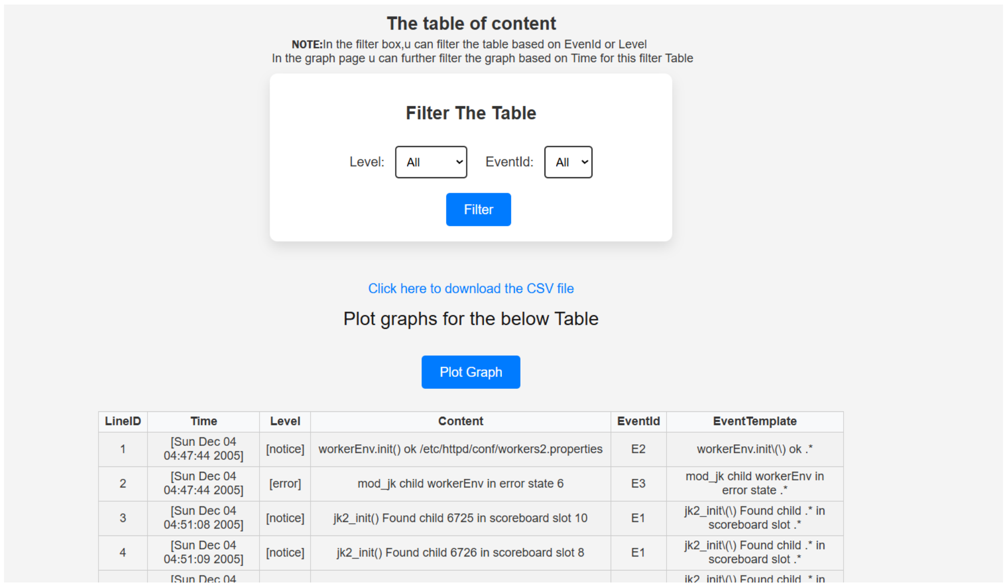
```
project/

 app.py                    # Main Flask application with all routes
 parsing_filtering.sh      # Bash script for parsing logs and filtering
 filter_time.sh            # Script for time-based filtering
 default_filter.sh         # Script for default visualization setup
 templates/                # HTML templates
    upload_page.html      # Home page with file upload form
    tabular_display.html  # Results page showing filtered data
    graphs_page.html      # Page displaying visualizations
    plotter.html          # Custom Python plotting interface
 static/                   # Static files and generated content
    css/                  # CSS stylesheets
       upload_page.css
       tabular_display.css
       graphs_page.css
       plotter.css
    Apache_2k.csv         # Processed log file (generated)
    pie_plot.png          # Generated visualization
    plot_plot.png         # Generated visualization
    hist_plot.png         # Generated visualization
    user_plot.png         # User-generated visualization
 Temporary_files/          # Temporary data files
     error_handling.txt
     filtered_notice_error
     filtered_time
     filtered_events
     line_numbers.txt
     submitted_times.txt
     total_number_of_lines.txt
```

7 Basic & Advanced Features

Basic Features

- **File Upload and Validation:** Users can upload Apache log files through a simple interface. The system validates that the uploaded file is a proper Apache log file.
- **Tabular Display:** Filtered log data is presented in a clear, tabular format for easy reading and analysis.
- **CSV Export:** Users can download the filtered data as a CSV file for further analysis in external tools.
- **Standard Visualizations:** The system automatically generates standard visualizations for the filtered data, including pie charts, line plots, and bar graphs.
- **Time-Based Filtering:** Users can filter visualizations based on specific time ranges, allowing for more focused analysis of log data patterns.
- **Bash Script Integration:** The application integrates with bash scripts for efficient log processing and filtering, leveraging the power of UNIX text processing tools.


The table of content

NOTE: In the filter box,u can filter the table based on EvenId or Level
In the graph page u can further filter the graph based on Time for this filter Table

Filter The Table

Level: All EventId: All

Click here to download the CSV file

Plot graphs for the below Table

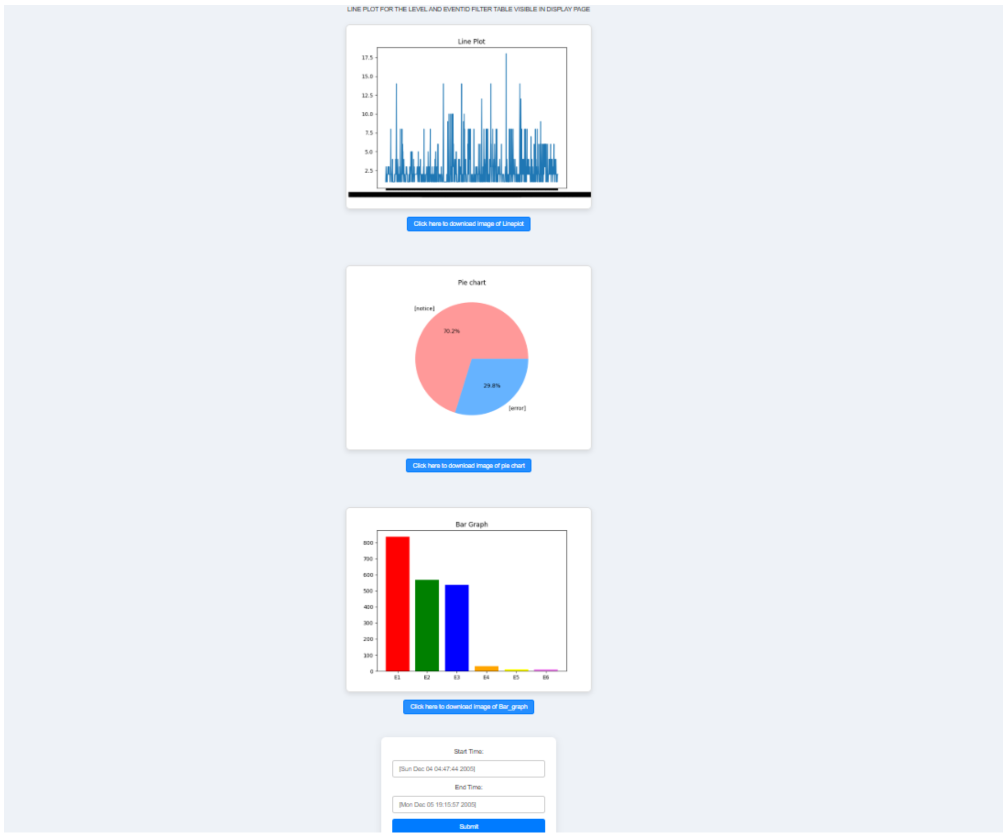
LineID	Time	Level	Content	EventId	EventTemplate
1	[Sun Dec 04 04:47:44 2005]	[notice]	workerEnv.init() ok /etc/httpd/conf/workers2.properties	E2	workerEnv.init\(\) ok .*
2	[Sun Dec 04 04:47:44 2005]	[error]	mod_jk child workerEnv in error state 6	E3	mod_jk child workerEnv in error state .*
3	[Sun Dec 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 .*
	[Sun Dec 04				jk2_init\(\) Found child .* in

Figure 2: Tabular Display (Conceptual)

Advanced Features

- **Data Filtering:** The application allows filtering log entries based on Level ([notice], [error]) and EventId (E1-E6), enabling users to focus on specific aspects of the log data.

- **Custom Plot Generation:** The system provides an interface for users to write their own Python code to generate custom plots using Matplotlib, offering flexibility for specialized analysis.
- **Dynamic Visualization:** Graphs automatically update based on filter criteria, providing an interactive experience for users exploring their log data.


LINE PLOT FOR THE LEVEL AND EVENTID FILTER TABLE VISIBLE IN DISPLAY PAGE
Line Plot
17.5
15.0
12.5
10.0
7.5
5.0
2.5
Click here to download image of Lineplot
Pie chart
[notice]
70.2%
29.8%
[error]
Click here to download image of pie chart
Bar Graph
800
700
600
500
400
300
200
100
0
E1
E2
E3
E4
E5
E6
Click here to download image of Bar_graph
Start Time:
End Time:
Submit

Start Time: [Sun Dec 04 04:47:44 2005]

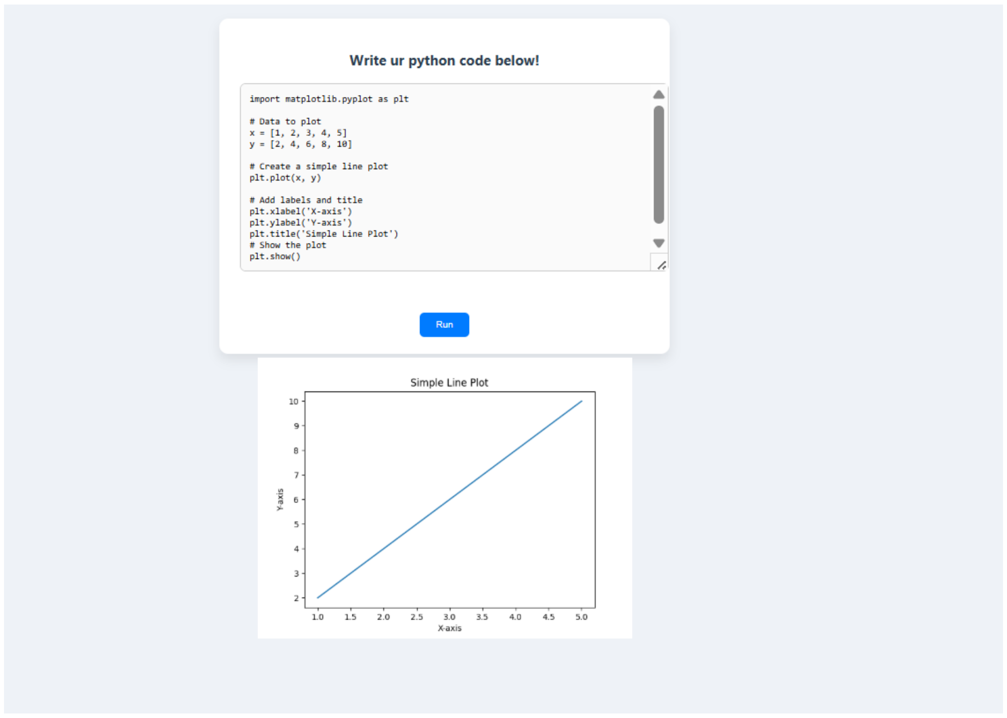
End Time: [Mon Dec 05 19:15:57 2005]

Figure 3: Generated Graph Example (Conceptual)

8 Project Journey

- **Web Application Architecture:** Developed a structured Flask application with proper routing and template rendering. Learned how to organize code for maintainability and separation of concerns.
- **Log Parsing Techniques:** Implemented efficient log parsing using bash scripts with tools like `sed` and `awk`. Gained experience in pattern matching and text processing for structured data extraction.
- **Python-Bash Integration:** Created a seamless integration between Python and bash scripts using the `os` module and file-based communication, enabling efficient processing of large log files.

- **Dynamic Visualization:** Implemented dynamic visualization generation using Matplotlib, including customizable plotting based on user-provided Python code.
- **Error Handling:** Developed robust error handling for file validation, ensuring that the application only processes valid Apache log files and provides clear feedback to users when issues arise.
- **Frontend-Backend Integration:** Created a cohesive user experience by connecting frontend form submissions with backend processing, ensuring smooth data flow throughout the application.
- **Performance Optimization:** Implemented performance tracking and optimization techniques to ensure responsive processing even with large log files.



```python
import matplotlib.pyplot as plt

# Data to plot
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a simple line plot
plt.plot(x, y)

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')
# Show the plot
plt.show()
```

Figure 4: Generated Graph Example (Conceptual)

9 Bibliography

- Flask Documentation: https://flask.palletsprojects.com/
- Matplotlib Documentation: https://matplotlib.org/stable/index.html
- Bash Scripting Guide: https://tldp.org/LDP/abs/html/
- Python Documentation: https://docs.python.org/3/

- Flask Tutorial (Real Python):https://realpython.com/tutorials/flask/
- awk lecture slides:https://mayur.bodhi.cse.iitb.ac.in/courseware/course/4/content/multimedia/concept/42?element=88
- bash lecture slides:https://mayur.bodhi.cse.iitb.ac.in/courseware/course/4/content/multimedia/concept/40?element=82
- python lecture slides:https://mayur.bodhi.cse.iitb.ac.in/courseware/course/4/content/multimedia/concept/43?element=96
- matplotlib lecture slides:https://mayur.bodhi.cse.iitb.ac.in/courseware/course/4/content/multimedia/concept/43?element=112