Log File Analysis

Vivekanand Siddharthi

28 April 2025

Contents

1 Introduction

Log files are essential monitoring aids for system performance, troubleshooting problems, and complying with security regulations. Raw log data cannot be analyzed manually as it is error-prone and time-consuming to do so. The aim of the project is to address this issue by creating a Flask-based web application through which users can upload log files, parse and convert them automatically into CSV formats using Bash and Awk scripts, and represent the processed data in the form of graphs and plots. The system improves on efficiency by providing users with filtering of logs on date ranges, downloading processed visualizations and data, and deriving more insights on system activities. Through the union of Bash in data extraction and Python in web development and visualization, the project offers a practical, automated mechanism for efficient analysis of log files.

2 Running Instructions

Follow the steps below to run the Flask-based web application locally:

1. **Install Python and its Libraries(Matplotlib, Flask):**
   Ensure Python 3.x is installed on your system. You can download it from https://www.python.org/downloads/. Open a terminal or command prompt and install Flask by running:

   ```
   pip install Flask
   ```

2. **Navigate to the Project Directory:**
   Move into the project folder where `Web.py` is located(Back End)
   (*It runs perfectly fine even if you run the program from any other Directory*):

   ```
   cd path/to/project/Back\ End
   ```

3. **Run the Flask Server:**
   Start the server using the following command:

   ```
   python3 Web.py
   ```

4. **Access the Web Application:**
   After running, the terminal will display a local URL, typically:

   ```
   http://127.0.0.1:5000/
   ```

   Open this URL in your web browser to access the application.

5. **Usage:**

   - Upload a log file through the `Upload.html` interface.
   - Processed data can be visualized through `Display.html` and `Plots.html`.

3 Website Navigation and Features

After opening the URL, the user will encounter these three pages in their respective order.

3.1 Upload Page

This page is where users can upload their log files.
The Upload page contains a form that lets the user choose a file from the their filesystem [See 3.1] and submit them in a user-friendly form using a submit button.
On clicking the button, the uploaded file is validated and if the file is erroneous , the user is redirected back to the same page and a suitable error message is displayed below the button [See 3.1].The system accepts only Apache_log type logs, refer [6].

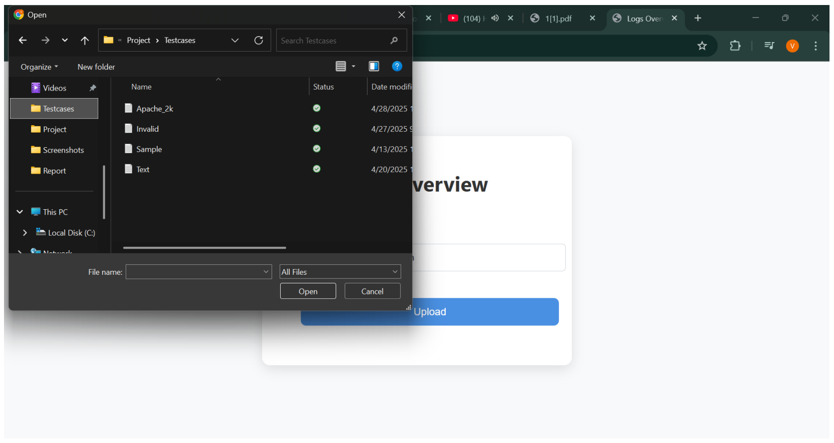
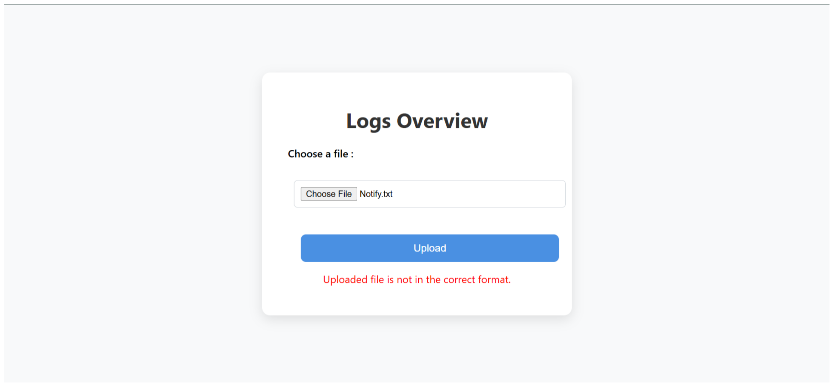


Figure 1: Upload Page File Upload



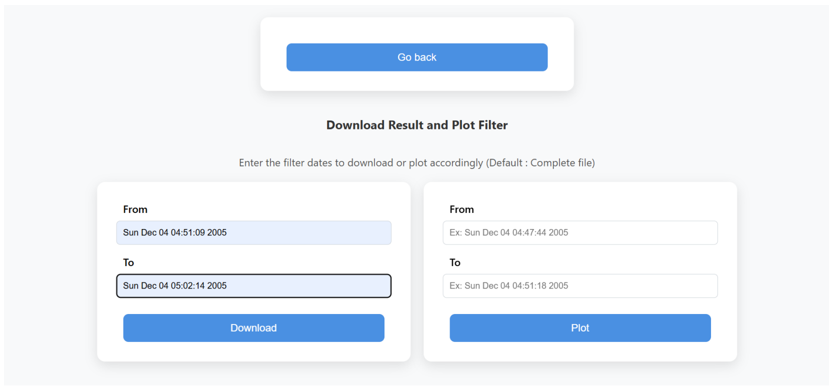
Uploaded file is not in the correct format.

Figure 2: Upload Page Error

3.2 Display Page

This page allows users to filter, download, or visualize the uploaded log data. The Display page is divided into three main sections to enhance usability:

- **Navigation Button:** A simple button at the top of the page allows the user to return to the Upload page.
- **Download Result and Plot Filter:** Users can specify a date range using 'From' and 'To' input fields(default being the entire file) [See 3]. They can then either download the filtered log file or generate a corresponding plot by clicking the respective buttons.
  If an invalid date range is entered, an appropriate error message is displayed below the button.



Download Result and Plot Filter

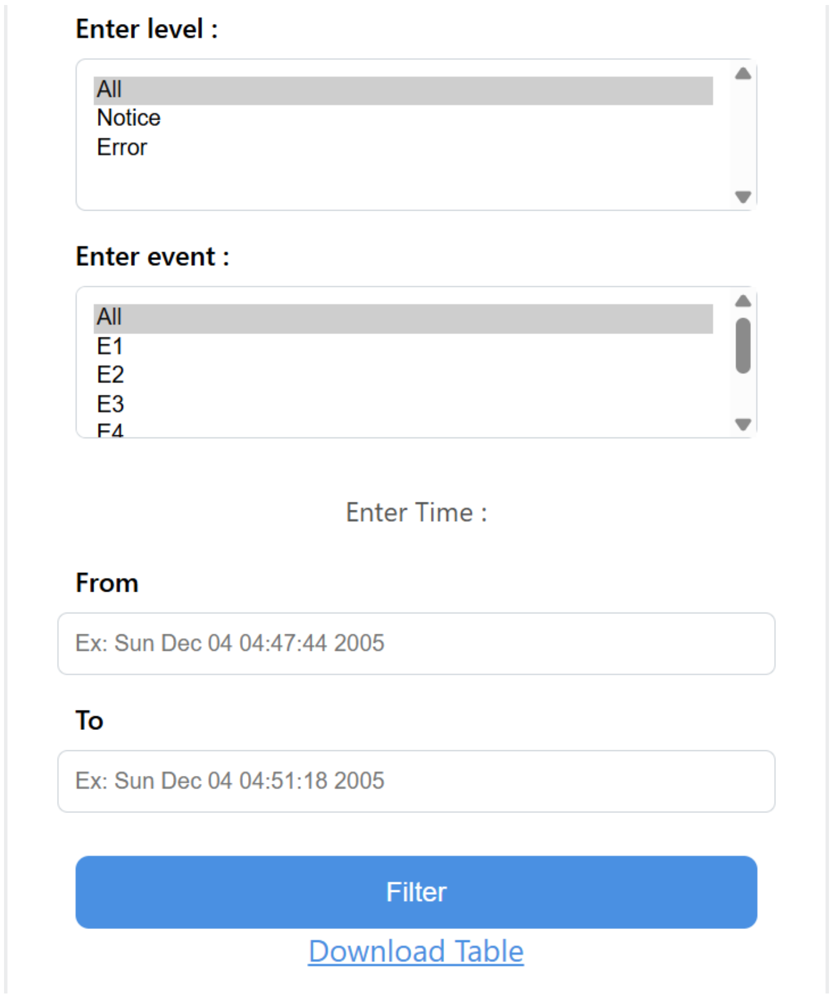
Enter the filter dates to download or plot accordingly (Default : Complete file)

From: Sun Dec 04 04:51:09 2005
To: Sun Dec 04 05:02:14 2005

From: Ex: Sun Dec 04 04:47:44 2005
To: Ex: Sun Dec 04 04:51:18 2005

Figure 3: Display Page Filters and Nav Button

- **Table Filter:** This section allows users to filter logs displayed in a table based on *level* (e.g., notice, error) and *event type* (e.g., E1, E2, etc.). Multiple selections are supported.
  Note: Be sure to hold `ctrl` to select multiple options. Additionally, users can apply a date range filter here as well [See 4].


Enter level :

All
Notice
Error

Enter event :

All
E1
E2
E3
E4

Enter Time :

From

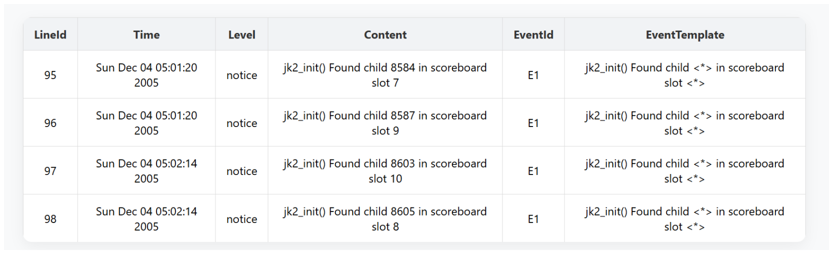
Ex: Sun Dec 04 04:47:44 2005

To

Ex: Sun Dec 04 04:51:18 2005

Figure 4: Display Page Table Filter

- After setting the desired filters and clicking the *Filter* button, the filtered table(default: entire file) is displayed below [See 5] and faulty inputs are handled.



LineId	Time	Level	Content	EventId	EventTemplate
95	Sun Dec 04 05:01:20 2005	notice	jk2_init() Found child 8584 in scoreboard slot 7	E1	jk2_init() Found child <*> in scoreboard slot <*>
96	Sun Dec 04 05:01:20 2005	notice	jk2_init() Found child 8587 in scoreboard slot 9	E1	jk2_init() Found child <*> in scoreboard slot <*>
97	Sun Dec 04 05:02:14 2005	notice	jk2_init() Found child 8603 in scoreboard slot 10	E1	jk2_init() Found child <*> in scoreboard slot <*>
98	Sun Dec 04 05:02:14 2005	notice	jk2_init() Found child 8605 in scoreboard slot 8	E1	jk2_init() Found child <*> in scoreboard slot <*>

Figure 5: Display Page Table

3.3 Plots Page

This page allows users to view and download the visual plots generated by filtering the uploaded log datain the Display page.
The Plots page offers a simple and clean interface with the following features:

- **Navigation Buttons:** At the top, two buttons allow users to either return to the Upload page for uploading a new file or navigate to the Table page for further filtering and viewing of log entries. (Refered [1])

- **Plot Display and Download:** The page displays three different plots:
  - **Line Plot** showing the trend over time.
  - **Pie Chart** representing proportion of notice and error type logs in the distribution.
  - **Bar Chart** comparing all the event categories.

  Below each plot [See 6], a download link is provided so that users can download the corresponding plot image for their own use.

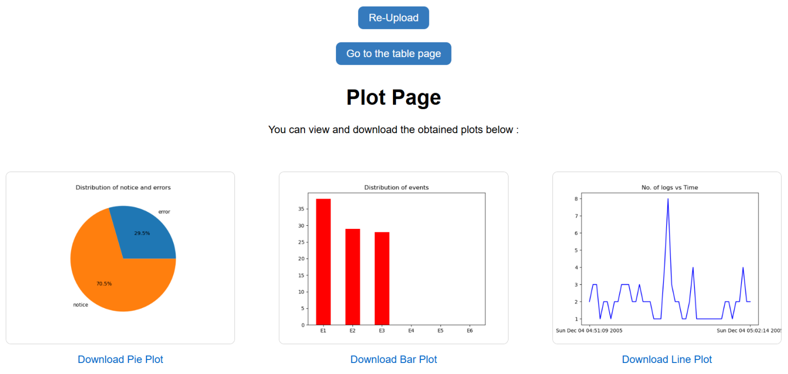


Figure 6: Plots Page View

4 Modules

- **Flask**: To build the web application, control routes, deal with file uploads, and display HTML templates. (refer [4])
- **render_template (Flask)**: To dynamically render HTML pages using templates.
- **request (Flask)**: To process incoming form data and file uploads from web pages.
- **session (Flask)**: (Imported but not used) Usually used to hold session data between routes.
- **send_from_directory (Flask)**: Used to send files (such as downloads) from a server directory to the client.
- **matplotlib**: Utilized for creating plots such as line graphs, pie charts, and bar charts from log data.

5 Directory Structure

The project is organized into three main sections: **Back End**, **Front End** and **Styles**.

- **Back End:** This folder contains all the server-side logic, scripts, and sample data used for processing log files.
  - `.log` files (`Apache_2k.log`, `Sample.log`, `Invalid.log`) are sample log files used for testing and validation.
  - `.awk` scripts (`Convert.awk`, `Filter.awk`, `Validate.awk`) are responsible for parsing, filtering, and validating log file contents.
    - `Validate.awk` is an AWK script used to validate the structure and format of incoming log files.
    - `Convert.awk` is an AWK script to convert raw log files into a structured CSV format.
    - `Filter.awk` is an AWK script to filter log entries based on specific conditions (e.g., status code, IP).
  - `.sh` scripts (`Convert.sh`, `Filter.sh`) are shell scripts that automate the execution of corresponding AWK scripts.
    - `Convert.sh` first invokes `Validate.awk` and after passing the requirements invokes `Convert.awk`.
    - `Filter.sh` invokes `Filter.awk`.
  - `.csv` files (`Filtered.csv`, `Summary.csv`, `Table.csv`) store structured data generated after processing the log files.

  - `Web.py` is the Python Flask application that handles file uploads, invokes scripts for processing, and routes the processed output to the front-end.

- **Front End:** This folder contains HTML templates used to display the processed data to the user.
  - `Upload.html` is the page that provides an interface for file uploads.
  - `Display.html` is used to render processed log data in a structured format.
  - `Plots.html` is used to display visual plots generated from the log data.
  - `Template.html` serves as the base template for all other pages, ensuring consistency in design across the application.

- **Styles:** This folder contains CSS files corresponding to the HTML templates to enhance the display with smooth and classy textures.
  - `Upload.css` styles the file `Upload.html`.
  - `Display.css` styles the file `Display.html`.
  - `Plots.css` styles the file `Plots.html`.

6 Features

6.1 Basic Features

- Clean and user-friendly interface.
- User's flexibility to both view and download graphical interpretations of their data based on their own filter.
- Placeholders and error messages that lets the user easily figure out their required input.
- Allowing users to download a csv file with filters for their further use.
- Clean, ordered table to display the processed data.
- A Navigation tool at the top of each page to go back and forth between pages.

6.2 Advanced Features

- Optimisation of process-time using `Awk` and other time-shortening measures to display the output relatively quickly(0.3 sec) even for large files.
- Table filter that allows multiple selects for filter keys like *levels* and *events* along with a time filter, all of which can be set simultaneously.

- The Web works smoothly when run on any OS since **os** module is used instead of hardcoding paths. (Refered [5])
- The user can run the application irrespective of the current directory that they are in. This is achieved by using **os** and **url_for** instead of hardcoding paths.

7 Project Journey

7.1 Learnings

1. Learnt how to use Flask and an insight on web applications.
2. Learnt about unfamiliar modules like **os**, **sys**, **subprocess**.
3. Learnt using HTML, CSS, JS with Jinja-Flask template engine.
4. Understanding how URLs work and the relationship between the filesystem and URLs. (Refered [3])

7.2 Challenges Adressed

1. Handling a **Matplotlib** bug that overlaps when the GUI is opened on multiple attempts one after the other. (used [2])
2. Starting from scratch after execution of parse and validation in **Python** to redo everything in **Bash**.
3. Making paths work effectively irrespective of the operating system being used to run the code.
4. Handling incorrectly entered input by users.
5. Realising that `gawk` was set to default by a friend and that was the reason for error occurence because `gawk` handles regex differently.

References

[1] MDN Web Docs. Difference between get and post methods in http, 2024. Accessed: 2025-04-29.

[2] John D. Hunter. Matplotlib documentation. https://matplotlib.org/stable/index.html.

[3] Pallets Projects. *Jinja Documentation*, 2024. Accessed: 2025-04-29.

[4] Pallets Projects. Flask web framework documentation.

[5] Python Software Foundation. *Python os — Miscellaneous operating system interfaces*, 2024. Accessed: 2025-04-29.

[6] Jieming Zhu, Pinjia He, Zibin Li, et al. Loghub: A large collection of system log datasets. https://github.com/logpai/loghub, 2019. Accessed: April 29, 2025.