CS104 Project Report

Bhavya Tiwari[*]

April 29, 2025

This project report is in partial fulfilment of CS104 course requirements, and details the structure of my project including packages used, design of the application and what all new things I learnt while attempting the problem statement. I have tried my best to make this enjoyable to read `:D`

Contents

[*] Roll number 24B0913

1 Introduction

In order to give us a feel of how larger code bases work, a project was assigned to us in CS104 which needed using multiple things that we had learnt simultaneosly. It was a different experience from exam problems, where you start afresh on a *niche* problem suitable to attack by a particular tool (which was mostly obvious after reading the problem statement); moreover, we had to *built up* on old code. It was quite an interesting experience.

1.1 Choice of project

For the project, we were given three options: making a log analysis website, or a cricket live score website, or a simple angry birds style game. I chose to make a log analysis website because its internal mechanics looked crystal clear to me. It inherently looked quite modular (see Internals) and therefore easy to not mess up.

1.2 Aim of the project

To reiterate [Gup25], this project aims to develop a Flask-based (see [Pro24]) web application that allows users to:

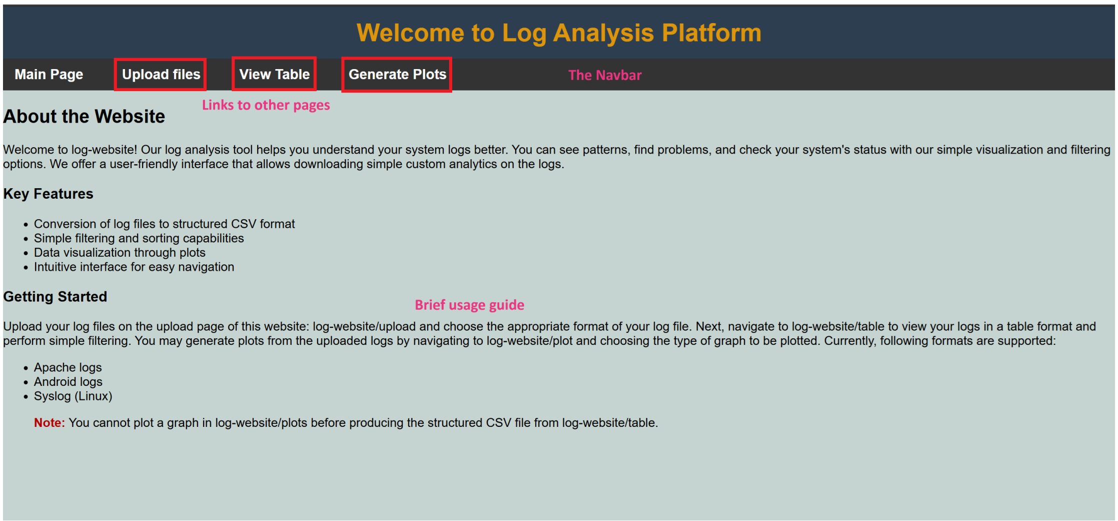
- Upload log files and convert them into structured CSV format.
- Filter and sort logs to extract meaningful insights.
- Generate visualizations to interpret log data effectively.
- Download processed logs and visualizations for further analysis.

2 Using the Application

To start the application execute the following command and you will be greeted by the home page as in the figure.

```
flask --app flaskwebsite_24b0913.py run
```

To get started with log analysis, proceed to the upload page and upload your log file using drag and drop interface (or a dialog box by clicking). All processing is done *lazily* except validation that is done instantly after the file is received (see Philosophy). Practically, this means that you can upload multiple files one by one but you have to *tell* the website which one do you want to process first.



Welcome to Log Analysis Platform

About the Website

Welcome to log-website! Our log analysis tool helps you understand your system logs better. You can see patterns, find problems, and check your system's status with our simple visualization and filtering options. We offer a user-friendly interface that allows downloading simple custom analytics on the logs.

Key Features

- Conversion of log files to structured CSV format
- Simple filtering and sorting capabilities
- Data visualization through plots
- Intuitive interface for easy navigation

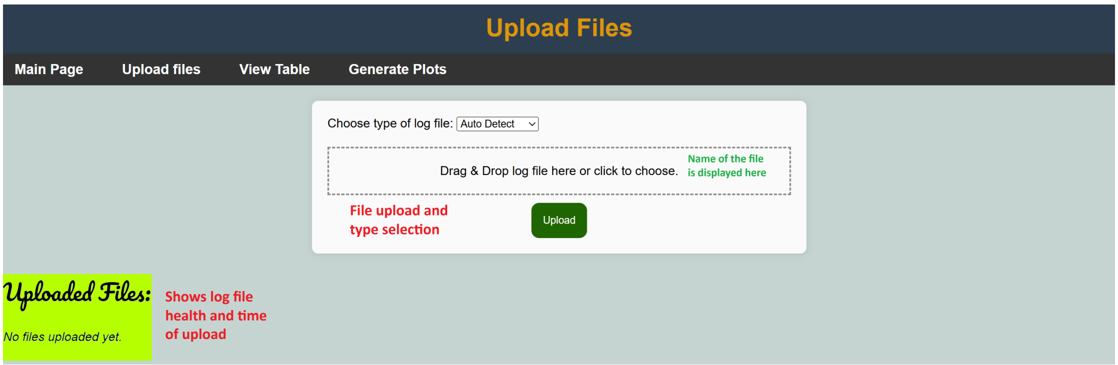
Getting Started

Upload your log files on the upload page of this website: log-website/upload and choose the appropriate format of your log file. Next, navigate to log-website/table to view your logs in a table format and perform simple filtering. You may generate plots from the uploaded logs by navigating to log-website/plot and choosing the type of graph to be plotted. Currently, following formats are supported:

- Apache logs
- Android logs
- Syslog (Linux)

Note: You cannot plot a graph in log-website/plots before producing the structured CSV file from log-website/table.

Figure 1: The home page of the website.



Upload Files

Main Page
Upload files
View Table
Generate Plots

Choose type of log file: Auto Detect

Drag & Drop log file here or click to choose.

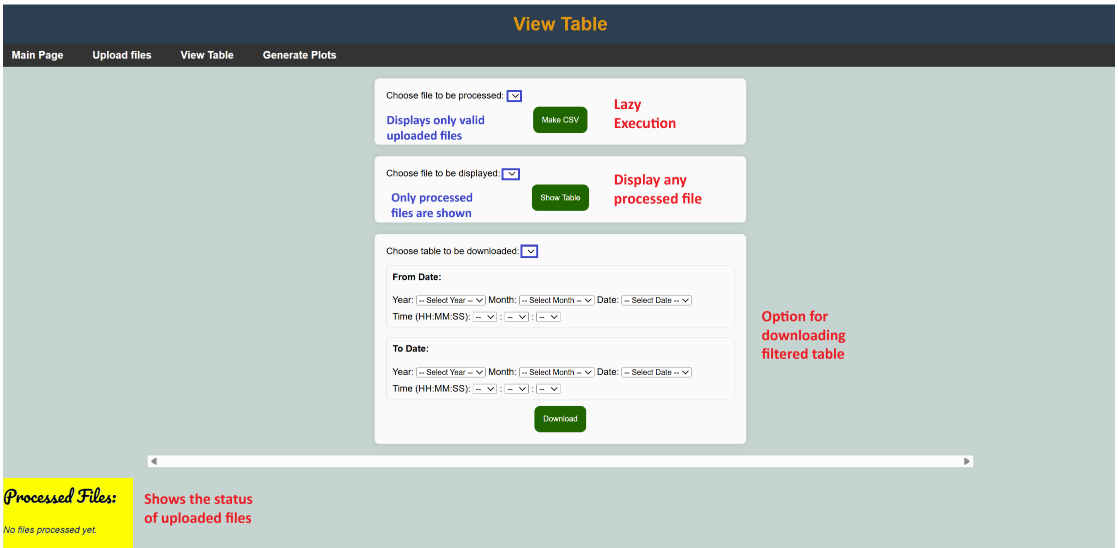
Name of the file is displayed here
File upload and type selection
Upload

Uploaded Files:

No files uploaded yet.

Shows log file health and time of upload

Figure 2: Form for uploading files on the upload page.



View Table

Main Page Upload files View Table Generate Plots

Choose file to be processed:

Displays only valid uploaded files

Make CSV

Lazy Execution

Choose file to be displayed:

Only processed files are shown

Show Table

Display any processed file

Choose table to be downloaded:

From Date:

Year: -- Select Year -- Month: -- Select Month -- Date: -- Select Date --

Time (HH:MM:SS): -- : -- : --

To Date:

Year: -- Select Year -- Month: -- Select Month -- Date: -- Select Date --

Time (HH:MM:SS): -- : -- : --

Download

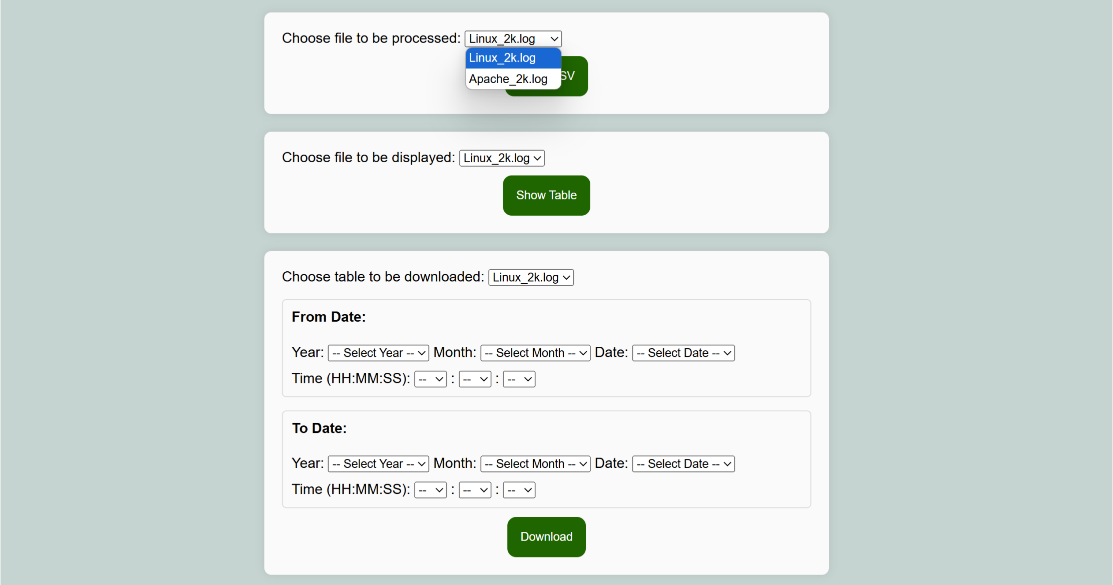
Option for downloading filtered table

Processed Files:

No files processed yet.

Shows the status of uploaded files

Figure 3: Formatting the log file into a table.



Choose file to be processed: Linux_2k.log

Linux_2k.log

Apache_2k.log

Choose file to be displayed: Linux_2k.log

Show Table

Choose table to be downloaded: Linux_2k.log

From Date:

Year: -- Select Year -- Month: -- Select Month -- Date: -- Select Date --

Time (HH:MM:SS): -- : -- : --

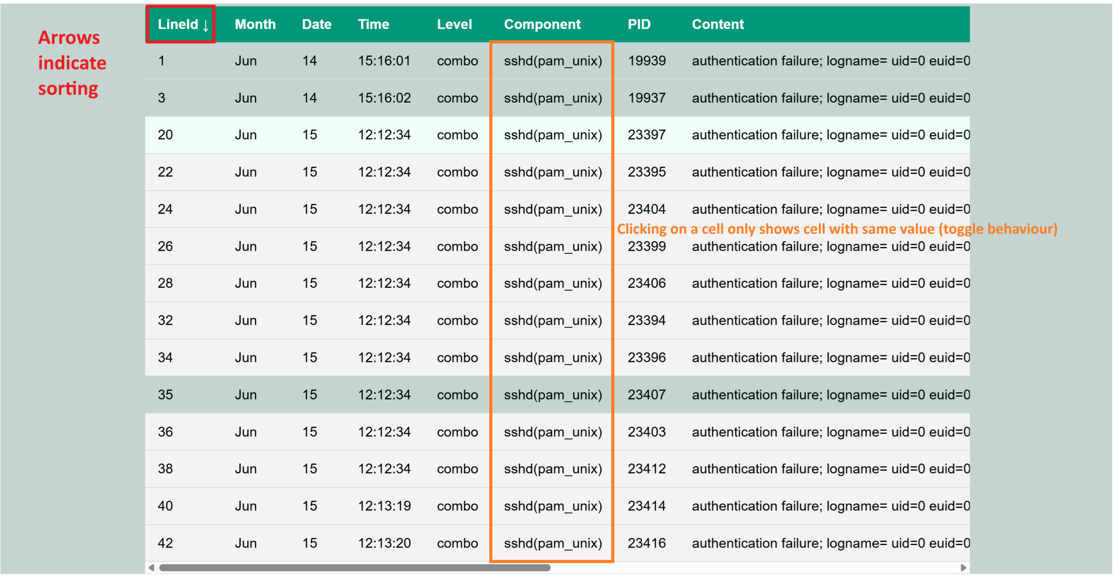
To Date:

Year: -- Select Year -- Month: -- Select Month -- Date: -- Select Date --

Time (HH:MM:SS): -- : -- : --

Download

Figure 4: Valid files are shown in a drop down list.



Arrows indicate sorting

Clicking on a cell only shows cell with same value (toggle behaviour)

LineId ↓	Month	Date	Time	Level	Component	PID	Content
1	Jun	14	15:16:01	combo	sshd(pam_unix)	19939	authentication failure; logname= uid=0 euid=0
3	Jun	14	15:16:02	combo	sshd(pam_unix)	19937	authentication failure; logname= uid=0 euid=0
20	Jun	15	12:12:34	combo	sshd(pam_unix)	23397	authentication failure; logname= uid=0 euid=0
22	Jun	15	12:12:34	combo	sshd(pam_unix)	23395	authentication failure; logname= uid=0 euid=0
24	Jun	15	12:12:34	combo	sshd(pam_unix)	23404	authentication failure; logname= uid=0 euid=0
26	Jun	15	12:12:34	combo	sshd(pam_unix)	23399	authentication failure; logname= uid=0 euid=0
28	Jun	15	12:12:34	combo	sshd(pam_unix)	23406	authentication failure; logname= uid=0 euid=0
32	Jun	15	12:12:34	combo	sshd(pam_unix)	23394	authentication failure; logname= uid=0 euid=0
34	Jun	15	12:12:34	combo	sshd(pam_unix)	23396	authentication failure; logname= uid=0 euid=0
35	Jun	15	12:12:34	combo	sshd(pam_unix)	23407	authentication failure; logname= uid=0 euid=0
36	Jun	15	12:12:34	combo	sshd(pam_unix)	23403	authentication failure; logname= uid=0 euid=0
38	Jun	15	12:12:34	combo	sshd(pam_unix)	23412	authentication failure; logname= uid=0 euid=0
40	Jun	15	12:13:19	combo	sshd(pam_unix)	23414	authentication failure; logname= uid=0 euid=0
42	Jun	15	12:13:20	combo	sshd(pam_unix)	23416	authentication failure; logname= uid=0 euid=0

Figure 5: Clicking on heading sorts the rows by value, while clicking on data cell only displays cells that have the same value in that column (as the clicked value).

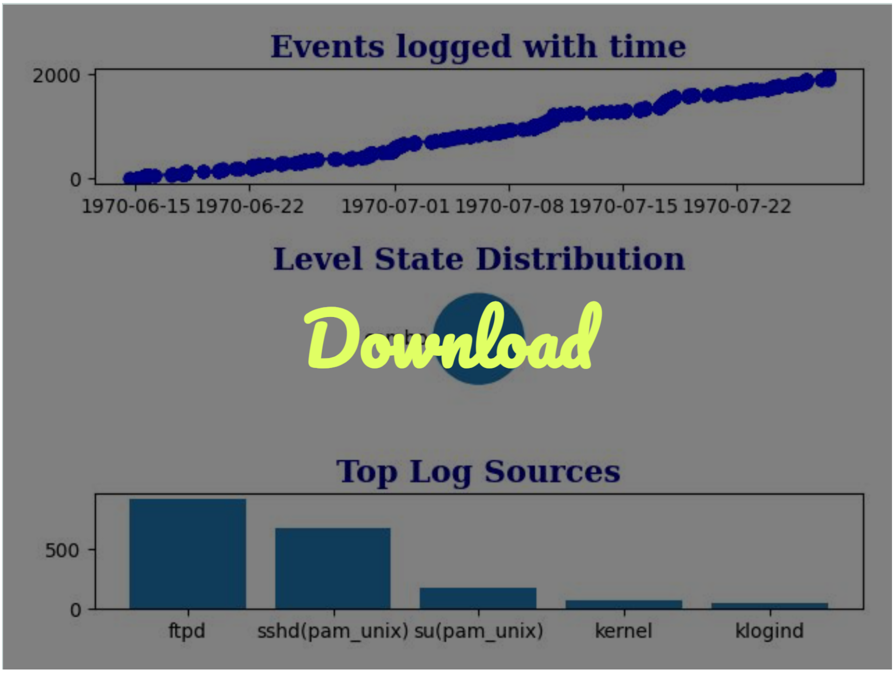


Figure 6: On hovering at the displayed plot in the plotting graph page, there is an option to download the plot.

3 Website Layout

3.1 Philosophy

1. **Lazy execution:** This bypasses the need for a scheduler if the user uploads multiple files that need to be processed and reduces the burden on the server. While it is easy (and perhaps more natural) to automate some of these things, this design allowed for easier debugging and decent performance on my PC (much less powerful than a server).

2. **Use simple webpage elements:** The only thing that I am importing from an external source is the caligraphic font for the status displayed at the bottom of the webpages. There are no imported libraries for HTML/CSS/JS because I wanted to only use things that I can understand as basic features of HTML/CSS/JS themselves rather than relying on "magic" done by somebody else.

3. **Consistent Interface:** All constituent webpages share as much HTML/CSS as possible; I have tried to make the functioning of the website clear and easy to remember and use (since bash scripts are the slowest part of the server-side, if the user explicitly clicks the buttons starting the bash scripts, then they will have some idea as to why it is taking time to load).

3.2 The Webpages

The website consists of four webpages, namely,

1. **Main Page:** Gives a short introduction to the website and a mini usage guide. I felt that the first page being an uploads page might feel out of context. It is possible to compress the pages together but as per project guidelines [Gup25], we needed at least three webpages.

2. **Upload Files:** Allows the user to upload files. Currently supported formats are shown in the drop down menu (Apache log files, Android log files and Linux Syslog). By default, it auto detects the log type and informs the user of any failure.

3. **Show Tables:** Makes structured csv and displays interactive tables; also allows the download of csvs filtered by `from` and `to` dates.

4. **Show Plots:** Plots the data in structured csv files over custom date ranges. The image can be downloaded in both `png` and `jpeg` formats by clicking on the displayed plot on the website (by default no download is started).

3.3 Implemented Features

Although the features were introduced alongside the discussion, here is a short summary.

1. **General:** There is a title bar, navbar and a status bar (depending on the function of the webpage) on each of the four webpages – `main, upload, table, plot`.

2. **Uploads:**
    - Auto-detection of log file type (manual selection also supported),
    - Drag-and-drop interface for uploading files (dialog box also supported),
    - A status bar indicating the health of uploaded log files
        - displays valid or invalid log file
        - type of log file detected and shown
        - declared invalid if detected and input types different
        - shows time of upload of file
        - multiple uploads allowed in same app instance (one by one)
3. **Table:**
    - Production of structured csv on demand independent of order of upload,
    - Displays table of the requested processed log file,
    - Only completely processed files appear in the "Show Table" drop-down list
        - invalid log files are automatically eliminated from further processing
        - tables from any of the uploaded files can be displayed
    - Interactive displayed tables,
        - clicking on a heading sorts by that in ascending order and every next click toggles the order of sorting
        - a unicode arrow shows the order of sorting of the last clicked column
        - this arrow is removed if a heading other than the last one is clicked
        - clicking on a data cell only displays rows whose entry in the column of the clicked element is same as the clicked element itself
        - clicking on a data cell twice restores the rest of the table
    - A status bar informing about all unprocessed valid log files,
    - Download of table from a start date to an end date
        - if a start or end date is not provided, then it is ignored in the sorting
        - default behaviour is to download the entire table
4. **Plots:**
    - Shows plot of the selected log from a start date to an end date
        - same behaviour as the sorting options implemented in table page
    - Hovering over the displayed plot hints the user of a download option
    - Clicking on the image takes the user to an image endpoint for downloads

4 Internals

4.1 External Dependencies

- The *Pacifio cursive* font taken from Google APIs.
- Here is a simplified view of external python dependencies[1]:

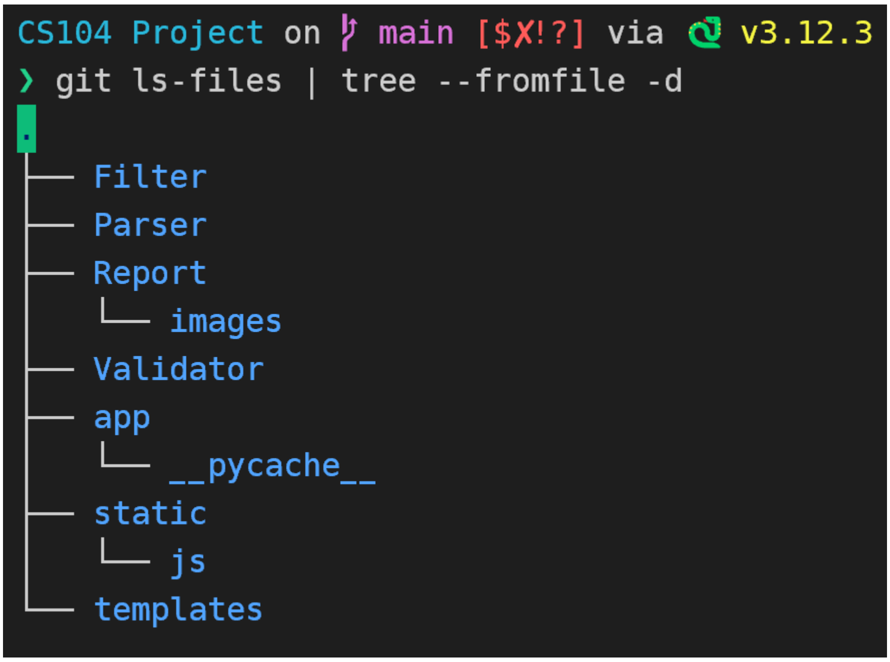
```
import numpy as np
import os, csv, subprocess
import matplotlib.pyplot as plt
from datetime import datetime
from flask import Flask, render_template
from flask import request, send_file, send_directory
from werkzeug.utils import secure_filename
```

`np.datetime64` was useful for plotting with `matplotlib`. The `os` module provides `os.path.join` and `os.makedirs` while the `subprocess` module calls relevant bash scripts. To display the table for files containing `,` and `""` in their fields[2], `csv` module was used (since support for these file formats is treated as a part of customization). The `datetime` library is used to keep a track of upload time of the file and nothing else. Finally, there is import of some flask utility functions and a standard security import `werkzeug.utils.secure_filename` (which tries to eliminate Javascript injection attacks in file names).

4.2 Directory Structure

The project uses three different tools: `bash`, `python`, and web development tools. We also have this LaTeX report. The folders that correspond to each of these things are as follows:

1. **Bash:** The `Validator` directory contains bash scripts for validation of Apache log files as well as Android and Syslog (Linux) files. Similarly, the scripts for producing structured csv is in `Parser` while those that filter the csv by date are in `Filter`.
2. **Python:** The kick-starting file named `flaskwebsite_24b0913.py` lies at the root of the project. All other python files lie inside the `app` folder.
3. **Web Tools:** All the HTML pages written using `Jinja2` templating engine lie in the `templates` directory. Javascript files are in `static/js` while all the CSS lies in the file `static/style.css`.
4. **LaTeX report:** Just for completeness, I added a `Report` directory that contains this report and an `images` folder inside that stores all figures.



```
CS104 Project on  main [$✘!?] via  v3.12.3
❯ git ls-files | tree --fromfile -d
.
├── Filter
├── Parser
├── Report
│   └── images
├── Validator
├── app
│   └── __pycache__
├── static
│   └── js
└── templates
```

Figure 7: An overview of the directory structure.

Note that there are three more directories that are only formed temporarily, namely `uploads`, `csv`, `img`. These contain the files uploaded by the user, the structured csvs after processing and the plot images respectively. Now we shall see what all files each of these folders contain.

- `Validator`: This contains the files named `*_validator.sh`.
- `Parser`: The main scripts have names of the form `log_parser_*.sh`. Event templates have been manually converted to regexes and stored in files named `*.txt` and `*.csv`. Finally, there are some associated `awk` scripts with questionable file names (for example, `main.awk` is only used for Android logs[3]). They were emotionally named based on the thinking process I had followed.
- `Filter`: This contains organized files with names `*_filter.sh` and `*_filter.awk` because the ugly work has been done during parsing.
- `static/js`: The files are named based on their purpose, e.g., `table_click.js` handles the click events in the displayed table that allow for dynamic sorting and filtering using Javascript.

[1] However, observe that different modules are imported in different files.

[2] Making use of some complex but widely acknowledged csv standards.

[3] To be fair, it is possible to use it in other log parsers but they all have their own regex files in different formats and revamping is quite doable but the work has been already done.

- `templates:` The `layout.html` file is the backbone of the website design and contains all the dreaded HTML boilerplate. Each webpage also has its own file.
- `app:` This folder contains the following files:
  - `config.py:` It contains the global dictionary tracking uploaded files and some metadata. Some auxillary variables like `SUPPORTED_LOG_FORMATS` and `*_TIME_LADDER` act as conveniences. Folders used for validations, parsing, filtering, uploading files, etc have their names stored in this file.
  - `routes.py:` This contains the logic for handling requests made by the client and all the `@app.routes`. It imports functions providing a high-level interface from `*utils.py`.
  - `utils.py:` This contains my implementation of `Counter.most_common` as we were not allowed to import that module. `find_*_date` functions are useful for casting the input received from forms into a compact format that can be given to a bash script. These functions could not be grouped into other files.
  - `pathutils.py:` The site is designed to keep files processed to a certain extent in the same location. Keeping this in mind, functions in this file do the ugly formatting work and path joining.
  - `*utils.py:` I have tried to keep function names fairly intuitive, so there should not be much issue with the semantics. One thing that deserves special mention is the `plt.close()` line in `plotutils.py`. It turns out that without this line, there is an issue with the flask application which is unable to return to the main loop (because of intereference from `matplotlib`). I learnt this the hard way `;)`

5 Project Journey

Throughout this project, I learnt a variety of new things. Let us go topic-wise.

5.1 Bash – Awk and Sed

The heart of the log processing are the bash scripts. More precisely, `awk` scripts (`sed` was mostly used for sanitization against ugly `\r` and in general is less powerful to process arrays of regex). The builtin `awk` functions had to be used extensively and this activity gave me an exposure of how to use them.

One of the hardest thing was to debugging the regexes that I had written using event templates. The way the `awk` regex engine parses is different for variables and raw strings so I was there in escaping and quotation hell.

Earlier I learnt to debug by using the excel skills that we had learnt in class and sorting the columns in various ways depending on lines that were missing. When it would come to absolute fine tuning though, I had to resort to the `diff` command to compare my output with the sample structured csv.

Debugging meant running some `diff` commands and grepping the appropriate lines and the regexes and looking at them until it clicked. Many times it was just a weird case that I had not seen before. For instance, in some fields some name was ending with a colon while others were not (and the sample's fields did not contain colons – they were junk to be eliminated[4]).

This project yet again showed me the inportance of reading documentation from official sources like [The24] for `awk` functions because they are most reliable and mostly to the point (although it is true that some of them like the bash manual are not the most friendly ones to read even though they contain all what you need).

5.2 Python – Flask and Jinja2

I had never used a framework like `Flask` to build a website. Thus, I referred to tutorials like [Sch18] and [Fre19] along with the official website [Pro24]. Initially all of my code was in a single file named `flaskwebsite_24b0913.py`, but as the project began to grow I decided to split it into multiple files. It was mostly clear how to split because my code was structured roughly like this:

```
@app.route('/path/to/page')
def name_of_route():

if condition:
  do something
  call_function()
  ...
```

[4] One of the cases where choosing the appropriate `awk` function was especially helpful.

```
return render_template(...)

def call_function(args):
  ...
```

Therefore, the main body went into app/routes.py while the associated functions were put in *utils.py. Before the refactoring I thought that I knew how the applciation worked. I did not. The errors during refactoring taught me how the application worked (first I searched online sources and asked ChatGPT that why the Jinja2 templating engine was unable to detect my HTML templates when they were present in the default directory and that it was fine before I had tried refactoring).

Essentially, the idea is that the Flask(__name__) that produces the current app instance should be present at the root of the project (made in flaskwebsite_24b0913.py for me). Then you may import your routes from whichever files you have in app/. However, these have to import the app instance from *the file that produced the app instance* and *not* flask.

As a side not, I also got to know that Jinja2 is used in contexts outside of Flask and developing websites – for templating network device configurations and database replication.

5.3 Javascript – File Event Handling

Haha, the crazy dialect. I thought that I could get away without using Javascript in this project at all (so strongly believing in my mistaken thoughts that they became partial motivation to do this project; in retrospect, the initial impulse of avoiding Javascript was stereotype-driven but now it is experience-driven), but as I tried to add more customization to the website, I understood why it was actually required.

For example, I wanted to do the table filtering-on-click customization but I was unable to figure out how do I send python this table that is being made using a Jinja2 template. Eventually I understood that instead of reloading the entire webpage anytime you clicked, it makes more sense to do this on the client-side using Javascript.

As another example, in the plotting webpage I want the user to download the plot and also see the plot. If I try to code both at server-side using the same button then what will the server respond with? Should it send a file or a newly rendered webpage? Such self questioning helped me to gain a better understanding of how the web works. Ultimately I felt that making two buttons was too ugly so I added a feature where clicking the image sents you to the download route for the image file (thanks to Javascript).

I realized that I did not understand and know many of the object attributes that were needed to complete my tasks. Just for completeness, I will detail the basic principles behind all three of my Javascript files. It was truly fascinating to know that what I know as "click to download", for example, was something completely else.

- start_plot_download.js: You first find the plot image on the webpage using querySelector and create a link element. The target of this link was set using `link.href = plot.src`. Finally to detect clicks, you add an event listener which does a weird thing: it creates the hypothetical object as a part of the document, *clicks* it, and *then deletes* it (the flashy effects are done using CSS and have nothing to do with Javascript at all . . . not quite click to download[5])!

- `drag_and_drop.js`: You make a `div` that constitutes the drag-and-drop zone. Then you add an event handler that checks whether the user has dragged a file into that zone (and highlight it using CSS). Apparently there is a `dataTransfer.files` attribute that event objects have that allows them to "catch the file to be sent to the server". More precisely, it is a `FileList`, which is like an array of files that were dropped. Input fields of type `file` have a `files` attribute that allows them to keep track of which files where uploaded (these are updated as necesssary).
- `table_click.js`: This was a bit hard to do. I tried to write an implementation which failed without giving errors (even in the browser console) so I had to take substantial amount of help from the internet. There are two event listeners: one listening for clicks on headings of the table and the other listening to clicks on the data cells. They make use of some special attributes of objects that I did not know. Conceptually, it checks the value of the cell nearest to user click and acts appropriately depending on the internal state.

Finally comes our honorable mention: **Git**. This was the first time I used git for a relatively large project involving thousands of lines of codes. It gave me a sense of progress with a touch of smugness since I had worked hard to remember and understand the usage of various git commands. I felt greater confidence while adding new features because of this additional security. For documentation purposes, I explicitly note that WSL's git was used so that line endings remain `\n` and so not interefere if the app has to run on Unix based systems.

6 Conclusion

It was a long and meaningful journey to making my first Flask application and a challenging experience to write the necessary shell scripts for the job. I learnt how to use `flask`, `Jinja2` templates, structure web applications (which was mostly done based intuition) and organize my `python` code. Javascript is a powerful but spooky language.

[5] it *is* click to download but the semantics is different

References

[Fre19] FreeCodeCamp. Learn flask for python, 2019. Accessed: 2025-04-28.

[Gup25] Kritin Gupta. Log file analysis, 2025. Official problem statement.

[Pro24] Pallets Projects. Flask documentation (stable), 2024. Accessed: 2025-04-27.

[Sch18] Corey Schafer. Flask tutorials, 2018. Accessed: 2025-04-28.

[The24] The GNU Project. *GNU Awk User's Guide: Built-in Functions.* Free Software Foundation, 2024. Accessed: 2025-04-28.