Log File Analyzer

Sushant Padha

April 2025

Contents

1 Introduction

This is a project made as part of the CS104 course (Spring semester 2025) for analyzing log files.

Log files play a vital role in understanding system behavior, identifying faults, and protecting against threats. Despite their importance, manually analyzing them can be a drain on time and resources.

This project focuses on developing a Flask-based web application that enables users to:

- Upload and validate multiple Apache log files through a drag-and-drop interface.
- Parse validated log files into structured CSV format for easy exploration.
- View, filter, and sort structured data using a dynamic, tabular web interface.
- Generate visualizations using either predefined plot templates or custom Python code.
- Download processed CSV files, and generated plots as `.png` files, for further analysis.

The project emphasizes ease of use, responsiveness, and code modularity.

2 Running Instructions

- Install Python 3.10 and pip if not already installed.
- Clone the project repository or download the source code, and open the project root directory in a terminal.
- (Optional) Set up a virtual environment for installing dependencies.

  ```
  python3 -m venv "<name-for-virtualenv-folder>/"
  ```

- Install dependencies from `requirements.txt`:

  ```
  pip install -r requirements.txt
  ```

- Or, install the following libraries manually via:

  ```
  pip install Flask==3.1.0 matplotlib==3.10.1 numpy==2.2.4 pandas==2.2.3
  ```

- Run the Flask application using the entry-point script `run.py`:

  ```
  python3 run.py
  ```

- Visit `http://localhost:5000` in your web browser to view the website.

3 Website Layout

- The overall website layout consists of a navigation bar at the top with clickable links to the landing/home page on the left and individual links for the upload, display logs, and plot pages.
- The main content is placed below it (varies from page to page), in the center of the window, with UI/UX and informational display elements placed inside.
- The website follows a simple light themed style with UI/UX elements styled and placed appropriately for intuitive usage. Styling is consistent across all pages.

3.1 Log Upload Page

- This page allows users to upload log files and also serves as the home or landing page for the project, accessible at server root or at the URL `/upload`.
- It supports a drag and drop interface for dropping .log files.
- Log files are validated via a `bash-awk` script workflow.
- Status of the previous as well as newly processed log files is displayed below the upload area as shown in 1.

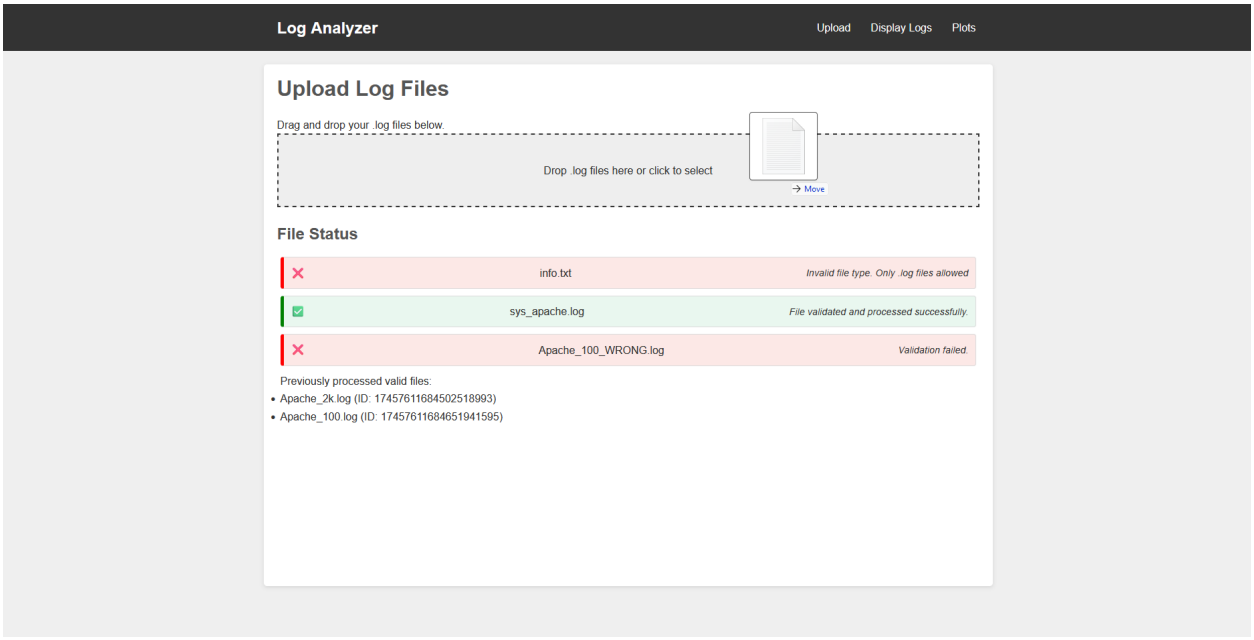

Figure 1: Upload page/Home page

3.2 Log Display Page

- This page allows users to display the processed log files as CSV data shown in the form of a table, accessible at the URL `/display`.
- It provides a drop-down selection list to choose the log file to display as shown in 3.
- Log files are displayed in a dynamically generated table (via AJAX requests) along with sorting and filtering controls and a download option as shown in 3.

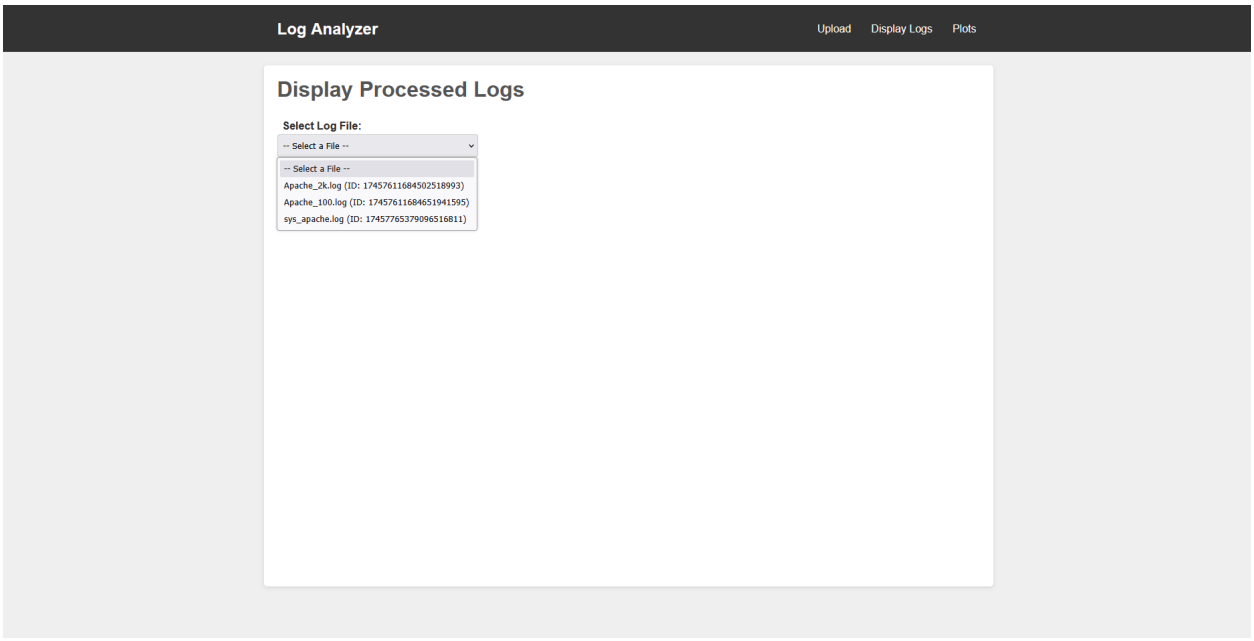

Figure 2: Log display page - Selection menu

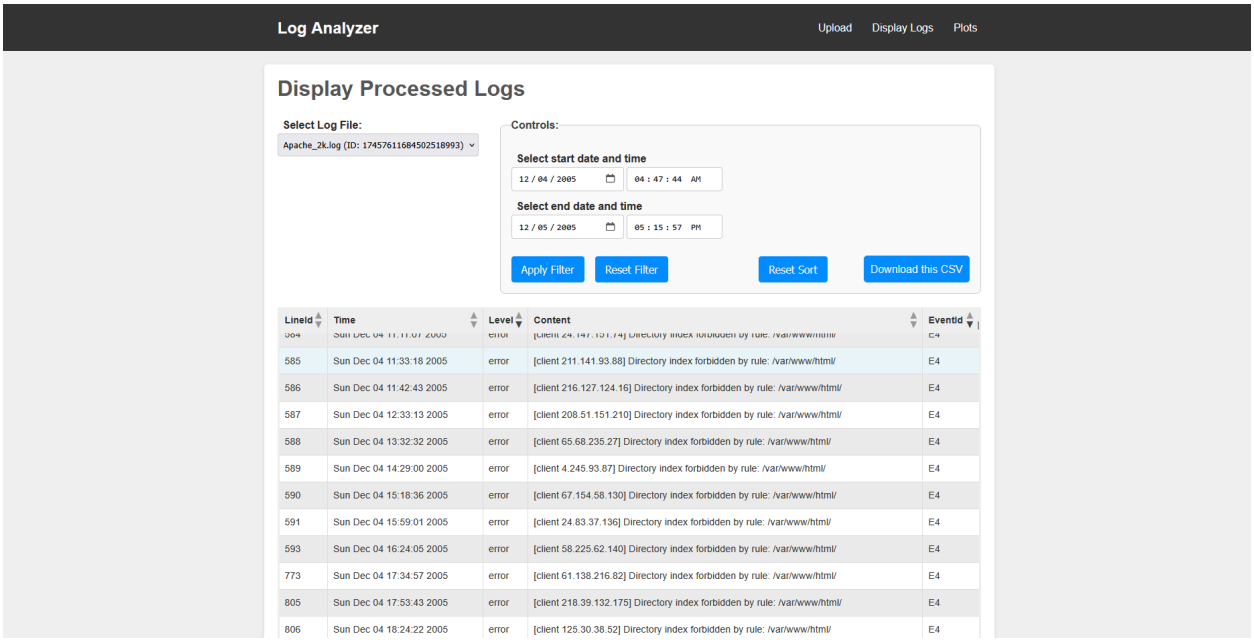

Figure 3: Log display page - Table and controls

3.3 Plot Display Page

- This page allows users to visualize the data from the log files as plots, accessible at the URL `/display`.
- It provides a drop-down selection list to choose the log file to display , similar to 2.
- Plot images are generated asynchronously and displayed on the page (via AJAX requests) along with filtering controls as shown in 4.

- If custom plot is enabled, a code editor element is displayed for entering python code as shown in 5

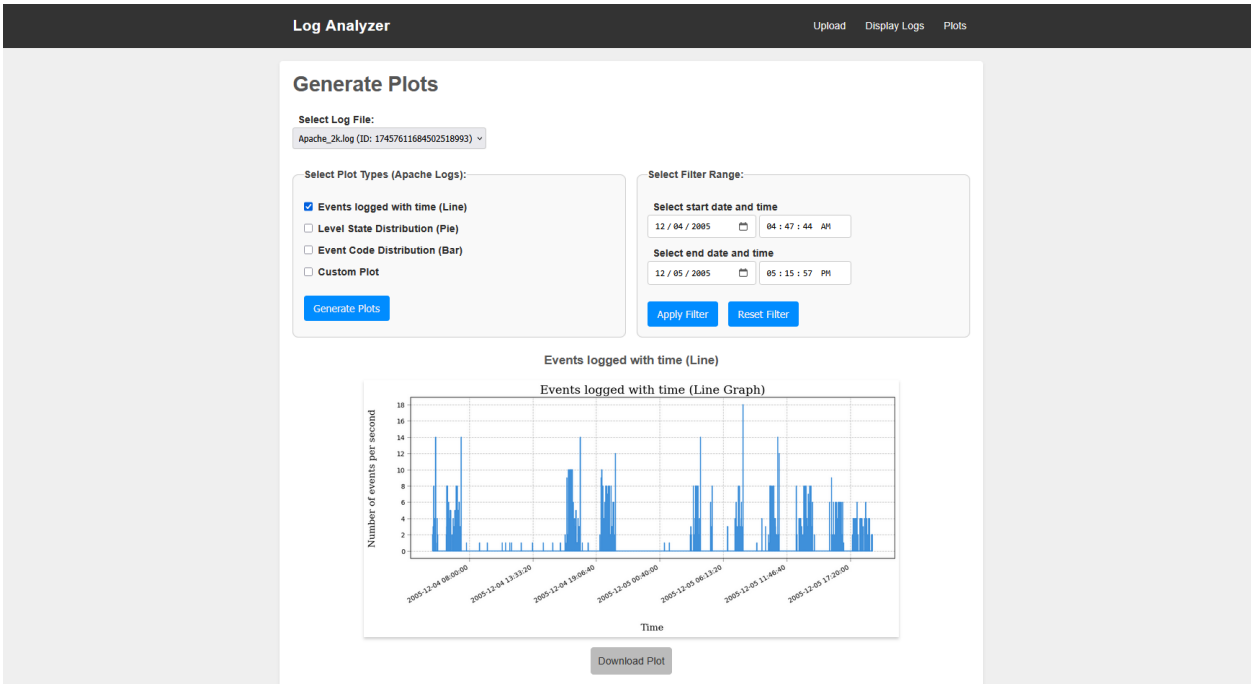

Figure 4: Plots page - Pre-defined plots and filter controls

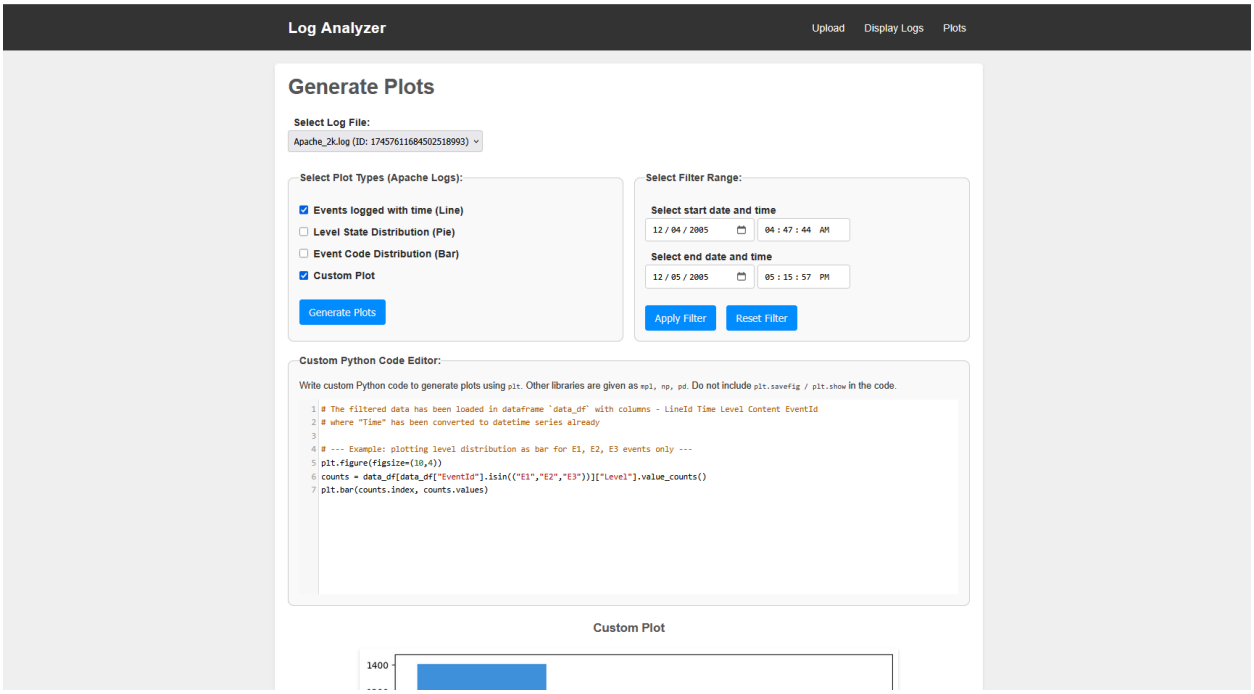

Figure 5: Plots page - Custom plot and code editor

4 Modules Used

4.1 Flask

Flask is a lightweight WSGI web application framework for Python. In this project, the main application logic loop, routing, request handling, template rendering and bash script interfacing are implemented using Flask..

4.2 NumPy

NumPy is a library for numerical computing in Python. In this project, NumPy is used for efficient numerical vector operations during data preprocessing for plotting of the *"Events logged with time (Line Graph)"* plot.

4.3 Matplotlib

Matplotlib is a library for creating static and/or interactive visualizations and plots in Python. It is used in this project to generate predefined plots as well as user-defined custom plots based on the processed log data and save them as downloadable images.

4.4 Pandas

Pandas is a powerful data analysis and manipulation library for Python. In this project, Pandas is used to parse structured CSV data into DataFrames which are exposed to the user for writing custom Python code in order to generate custom plots.

4.5 time, random

time and random (python standard library) are used for generating unique IDs for uploaded files based on upload time and an additional random numeric string.

4.6 os

os (python standard library) is used for directory creation, path manipulation and checking for existence.

4.7 subprocess

subprocess (python standard library) is used for running bash scripts and extracting output.

4.8 json

json (python standard library) is used for reading and writing JSON files to store server state (metadata and plot generation status).

5 Directory Structure

The directory structure as a tree is displayed in ASCII-encoded formatted text:

```
run.py
app/
  |---- __init__.py
  |---- config.py
  |---- routes/
  |    |---- __init__.py
  |    |---- display.py
  |    |---- plots.py
```

```
  |   `---- upload.py
  `---- utils/
      |---- __init__.py
      |---- csv.py
      |---- files.py
      |---- parse.py
      |---- plotting.py
      `---- timestamps.py
bash/
  |---- filter_by_date.awk
  |---- filter_by_date.sh
  |---- validate_parse.awk
  `---- validate_parse.sh
instance/
  |---- status.json
  `---- metadata.json
templates/
  |---- base.html
  |---- display.html
  |---- filter_controls.html
  |---- plots.html
  `---- upload.html
static/
  |---- css/
  |   `---- style.css
  `---- js/
      |---- display.js
      |---- plots.js
      |---- upload.js
      `---- utils.js
uploads/
  `---- ...
plots/
  `---- ...
processed/
  `---- ...
requirements.txt
run.py
cleanup.sh
LICENSE
README.md
```

File Descriptions

App Files (Python)

- **run.py**: Entry point for the application. Starts the Flask server.
- **app/__init__.py**: Initializes the Flask application, sets up configuration, and registers routes.
- **app/config.py**: Contains configuration settings for the application, such as folder paths and allowed file types.
- **app/routes/__init__.py**: Empty file used to mark the directory as a Python package.
- **app/routes/upload.py**: Handles routes related to uploading log files and serving the upload page.

- **app/routes/display.py**: Handles routes for displaying processed logs, metadata, and CSV data.
- **app/routes/plots.py**: Handles routes for generating and serving plots, as well as managing plot-related requests.
- **app/utils/__init__.py**: Used to mark the `utils` directory as Python package and exposes key functions from its submodules for easier importing.
- **app/utils/csv.py**: Contains utility functions for processing and generating CSV files and producing related data and metadata.
- **app/utils/files.py**: Contains utility functions for validating files and fetching information on processed files.
- **app/utils/parse.py**: Contains utility functions for parsing requests and sorting data.
- **app/utils/plotting.py**: Contains utility functions for generating plots and managing plot generation status.
- **app/utils/timestamps.py**: Contains utility functions for handling, formatting and validating timestamp strings.

Web site Files (HTML, JS and CSS)

- **templates/base.html**: Base template for all webpages. Implements navbar and overall website layout.
- **templates/upload.html**: Template for log upload page. Provides the user interface for uploading log files, validating them, and displaying upload statuses.
- **templates/display.html**: Template for log display page. Implements sorting and filtering controls and displaying of tabular data.
- **templates/plots.html**: Template for plot display page. Implements layout for plot controls and plot display and loading/error message display.
- **templates/filter_controls.html**: Short template for implementing controls for filtering by timestamp. Used as in `plots.html` and `display.html`.
- **static/css/style.css**: External stylesheet for implementing basic, common styling for all web pages. Styles common elements and overall layout.
- **static/js/upload.js**: Manages log file uploads, sends validation requests to the server, and dynamically updates the interface based on validation results.
- **static/js/display.js**: Handles querying for structured CSV data and rendering it in a dynamic, scrollable, and sortable table format on the web page.
- **static/js/plots.js**: Handles plot generation logic and user interactions for the plotting interface, importing utility functions as a JavaScript module.
- **static/js/utils.js**: Provides reusable functions for sending and processing AJAX requests, parsing and handling sorting/filtering options and other convenience functions, used for log display and plot display pages.

Validation & parsing and filtering scripts (Bash, AWK)

- **bash/validate_parse.sh**: Validates Apache log files and parses them into structured CSV format by via an AWK script. Called by app on new log file is uploaded.
- **bash/validate_parse.awk**: AWK script for processing raw log data to perform format validation and structured extraction into CSV fields.
- **bash/filter_by_date.sh**: Filters parsed CSV data based on user-specified timestamp ranges and outputs a new processed CSV. Called by app when data filtering request is received from log display or plots display page.
- **bash/filter_by_date.awk**: AWK script for filtering row by row based on specified range.

Functions in Each Python File

run.py

- (No functions; called as entry point script)

app/__init__.py

- `create_app()`: Initializes the Flask app, sets up folders, and registers route modules.

app/config.py

- (No functions; only a `Config` class)

app/routes/__init__.py

- (Empty file)

app/routes/upload.py

- `register_upload_routes(app)`: Defines decorated functions:
  - `upload_page()` and `handle_upload()`

app/routes/display.py

- `register_display_routes(app)`: Defines decorated functions:
  - `display_page()`, `get_csv(log_id)`, `serve_metadata(log_id)` and `download_csv(log_id)`

app/routes/plots.py

- `register_plots_routes(app)`: Defines decorated functions:
  - `plots_page()`, `handle_generate()`, `get_status()`, `get_plot(plot)` and `download_plot(plot)`.

app/utils/__init__.py

- (Imports functions from all submodules for easier exposing)

app/utils/csv.py

- `filter_csv(csv_fpath, opts)`
- `parse_csv(fpath)`
- `write_csv(fpath, header, data)`:
- `get_csv_data(csv_fpath, sort_opts, filter_opts, for_download=False)`
- `get_csv_metadata(log_id)`

- ...

app/utils/files.py

- `validate_filename(filename)`
- `get_processed_files()`

app/utils/parse.py

- `sort_data(data, opts)`
- `parse_opts(opts)`
- `parse_csv_request(log_id, request)`

app/utils/plotting.py

- `generate_plots(_app, data, plot_opts, plot_files, custom_code=None)`
- `set_plot_generation_status(status_str, plot_files=None, error_str=None)`

app/utils/timestamps.py

- `seconds_from_timestamp(timestamp)`
- `timestamp_from_seconds(total_seconds, pos=None)`
- ...

6 Basic and Advanced Features

Basic Features

File Upload (Drag & Drop)

- **Description**: Users can upload multiple log files easily through a drag and drop interface, or by clicking and choosing files via file explorer pop-up.
- **Explanation**: The front end implements a `drop-area` element that supports clicking and drag and drop via DataTransfer object API [7]. The uploaded file is packaged as POST request payload using the FormData API [8] which is sent using the Fetch API [9] and handled by the `handle_upload()` method.

Log Validation and CSV Generation

- **Description**: Uploaded files are validated against the Apache event log format and parsed and converted into CSV format in a single pass.
- **Explanation**: The uploaded file is passed an argument to the validation and parsing script written using bash and AWK. It matches each non-empty line against the standard Apache event log format to validate the file. It uses regex matching (`match` function in AWK) to extract the relevant fields as matched groups for CSV generation. The content is then mapped to one of the templates [5] to extract event ID field.
- **Explanation (Detailed)**: The request is sent at the endpoint `/upload` via POST method, containing the file. The `handle_upload()` method handles this and returns a JSON response indicating success/-failure and other relevant information. First, a unique ID is generated using the current time and a random numeric string. Then the file is first checked for `.log` extension and then passed to the script.

CSV Viewing, Filtering, Sorting and Downloading

- **Description**: CSV data is presented in a tabular format, which can be dynamically sorted and filtered. Supports responsive, scrollable display and an option to download.
- **Explanation (Detailed)**: The display page queries the server at endpoint `/get_csv/<log_id>` with the filter and sort options passed as query parameters in the URL. The `get_csv(log_id)` method parses the request, and calls `get_csv_data(...)` to return the data as a python iterable. The filtering is done on-the-fly by `filter_csv(...)` and filtered data is saved (using the filtering script). CSV data is read using the `parse_csv(...)` method which does not use the `csv` module. Sorted data is returned by the `sort_data(...)` method.
- **Additional Notes**: If the download button is clicked, the web page queries the `/download_csv/<log_id>` endpoint executing the `download_csv(log_id)` method. Same steps are executed as the above explanation, and then the sorted data is written to a file via the `write_csv(...)` method which is sent as a downloadable attachment to the client.

Generating and Displaying Plots

- **Description**: Data from the log files can be used to generate detailed plots which are displayed on the page and may be downloaded for further analysis.
- **Explanation**: Desired plot type and filtering (by timestamp) can be set from the plot display page. This sends a request to the `/generate_plots/` via POST method containing the chosen options and custom plotting code (if any). The `generate_plots (...)` method is called to generate the relevant plots in the `plots/` directory. The web interface queries for plot generation status and once done, queries for the actual plot images to be displayed on the website. An option to download the plots is also shown.
- **Additional Notes**: The plot generation request is handled by `handle_generate()` which returns a JSON response containing the names of the plot files (if successful) which is used to query the `/get_plot/<plot>` endpoint once plots are generated. The plot generation is handled asynchronously, with more details given at 6.

Advanced Features

Drop-down Selection List

- **Description**: Multiple log files can be uploaded for analysis and can be conveniently selected on the log display and plot display via a simple drop-down selection list element.
- **Explanation:** The `get_processed_files()` method returns the list of all files along with their original names (extracted from metadata). This is then passed as an argument to the rendered template, where the {`% for ... in ... %`} jinja block is used.

Storing File Metadata and Enhanced Filtering Options

- **Description**: Upon processing of the log files into CSV data, relevant metadata is also extracted and saved in `instance/metadata.json` file. It consists of objects mapping unique log ID's to objects containing the following data fields - original file name, start (earliest) timestamp, end (latest) timestamp - for convenient access later.
- **Explanation**: After validation and processing of data in the `handle_upload()` method, it extracts and saves the relevant metadata.

- **Use in Filtering Options**: Filtering options are presented as two pairs of `<input type="date">` and `<input type="time">` fields with reset and apply buttons. Using methods implemented in `utils.js`, when the user selects a log ID, the endpoint `/get_metadata/<log_id>` is queried and the minimum, maximum and default values for the input fields are set corresponding to start and end timestamp, which makes it more convenient for the user.

Asynchronous Plot Generation

- **Description**: Plots are generated asynchronously so that the main server logic loop is not blocked.
- **Explanation**: The `generate_plots()` method runs in a background thread, and communicates its status by writing to a `status.json` file, whose contents are queried by the web page via the `/status` endpoint. Polling occurs every 500ms for around 30s. When finished status is recieved, the web page queries `/get_plot/<plot>` to get the actual plot images.

Asynchronous UI Updates

- **Description**: AJAX requests [6] used to load new content without reloading pages.
- **Explanation**: The Flask app defines separate routes for serving the web page and serving the required data. The data is queried using `fetch` calls in the javascript code which is then processed and displayed in the desired layout. This ensures that the overall website content (including stylesheet etc.) don't have to be reloaded whenever data is refreshed. This also prevents longer loading times since the data can be thousands of lines long.
- **Examples**: The log file selection list and sorting and filtering options on the log display page, use (mostly `async`) `fetch` calls to query the data. Similar implementation is used for the plot display page.

Error Handling

- **Description**: Comprehensive error handling has been implemented on both the server side and client side. Whenever appropriate, a relevant error message is sent to the client for the user's discretion.
- **Additional Notes**: Every web page implements elements for displaying loading and error messages when needed. Server-side error handling ensure the server does not crash due to non-fatal errors and client-side error handling ensures only well-formed data is displayed and in case of an exception the user is informed.

Custom Plot Generation and Embedded Code Editor

- **Description**: The user can enter custom code which can be used to generate additional plots.
- **Explanation**: The plot display page implements a code editor using CodeMirror [10]. The server parses the CSV data into a pandas dataframe which is exposed along with common modules, for the user's custom code. The code is then run in limited context (to prevent malicious access) and the corresponding plot is saved and displayed.

Intuitive Sorting Controls

- **Description**: Log display page implements intuitive sorting controls for tabular data.
- **Explanation**: Structured data can be sorted with respect to all fields. Sorting is implemented using the `sort_data(...)` method in python.

7 Project Journey

Challenges and Solutions

- **Handling General Log file Formats**: Solved by analyzing log file entries to build regex patterns that are expressive and robust.
- **Avoiding Main Thread Blocking**: Used Python `threading` module to run plot generation separately.
- **Dynamic Content Management**: Managed with client-side AJAX requests and server-side JSON responses.
- **Efficiently Plotting Events over Time**: Used `numpy` to convert the range of timestamps into an array, which is used the build the plotting data more efficiently. Moreover, used `matplotlib`'s formatter [12] and locator [11] classes to generate ticks.

Key Learnings

- Integration of asynchronous operations (AJAX) with Flask routes.
- Handling file uploads, validation, and parsing efficiently.
- Implementing multi-threaded execution for long-running tasks.
- Implementing comprehensive error handling in applications.

8 Bibliography

References

[1] Python Standard Library Docs: https://docs.python.org/3/library

[2] Flask Documentation: https://flask.palletsprojects.com/

[3] Pandas Documentation: https://pandas.pydata.org/

[4] Matplotlib Documentation: https://matplotlib.org/

[5] Apache Log Templates: https://github.com/logpai/loghub/blob/master/Apache/Apache_2k.log_templates.csv

[6] MDN Web Docs - AJAX Concepts: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

[7] MDN Web Docs - DataTransfer: https://developer.mozilla.org/en-US/docs/Web/API/DataTransfer

[8] MDN Web Docs - FormData API: https://developer.mozilla.org/en-US/docs/Web/API/FormData

[9] MDN Web Docs - Fetch API: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

[10] CodeMirror 5 Manual: https://codemirror.net/5/doc/manual.html

[11] Matplotlib Tick Locators: https://matplotlib.org/stable/gallery/ticks/tick-locators.html

[12] Matplotlib Tick Formatters: https://matplotlib.org/stable/gallery/ticks/tick-formatters.html