Log collector – user manual

List of contents

Setup

To use the app, you will need Docker and optionally a web browser.

Docker

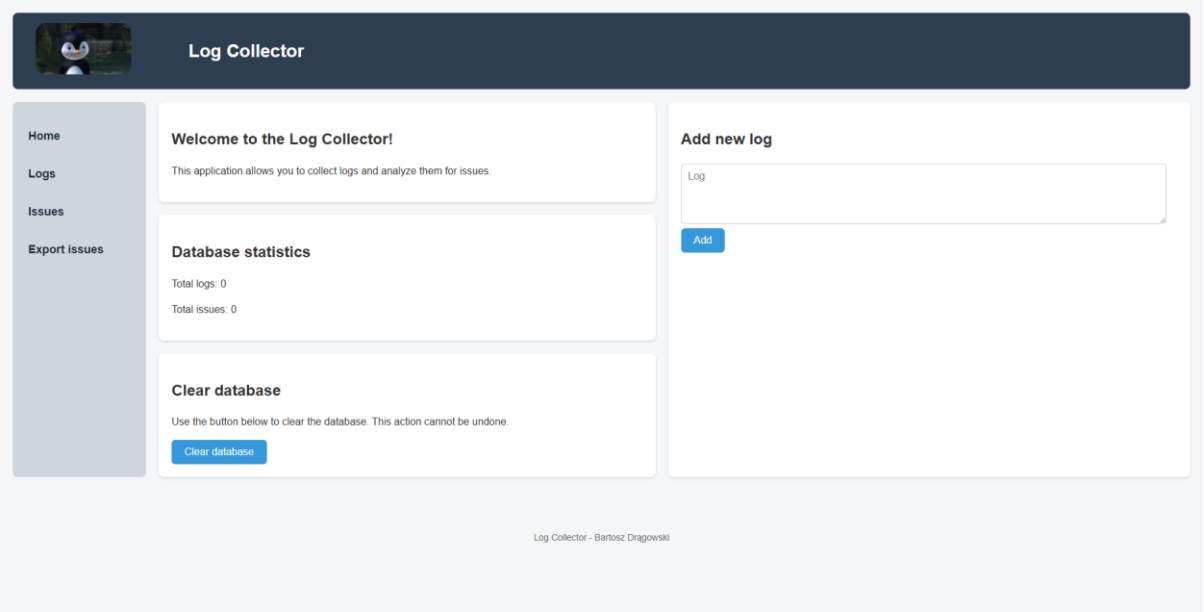
I used Docker Desktop, for ease of work.

1. Download Docker Desktop for your operating system [link].
2. Run installer and proceed according to instructions.
3. When prompted, reset the computer.
4. Start Docker Desktop and wait for it to complete the first startup (you can skip the sign up).
5. Extract the Log_collector.zip file provided with the instruction into a location of your choice.
6. Open terminal in that folder.
7. Run command `docker compose up` and wait for the app to start.

You can run `docker compose up -d` instead, if you don't care about logs or you want to view them in Docker Desktop.

GUI guide

You can access GUI via web browser http://localhost:5000. This link will direct you to main page of the app. It should look like this:



Right now, database is empty, so try to add your log in ”Add new log” section. You can see all the logs you added if you press ”Logs” button placed on the left-hand menu section. Issues are extracted automatically from your logs. You can access them from the same menu by pressing ”Issues” button, or ”Export issues” to download all issues in txt file.

The rest of the buttons and functions should be self-explanatory. Just be mindful of “Clear database” button, since it will remove all logs and issues from the database forever.

API guide

When you need more functionality than GUI provides or need to communicate with app from your own app, API is your best friend.

Before starting to work with it, you can check if the app works as intended on your OS by running tests. You will find the script in `tests/tests.py`. To start the tests, just run the script. If there are no red messages, you are good to go. The script also leaves 4 example logs for you to test your app.

Functions of API

To access the API functions, add `/api` at the end of the url. The rest of the HTTP endpoint is provided depending on the function.

API for logs

- Get all logs: GET /logs
- Get log by ID: GET /logs/{_id}
- Get log timestamp – can be empty: GET /logs/{_id}/datetime
- Get number of lines in the log: GET /logs/{_id}/line_number
- Add log: POST /logs
- Delete log: DELETE /logs/{_id}

Returned logs in json will have those indexes:

- _id – hashed mesage,
- log_message – entire log,
- log_datestamp – can be empty. If not, will provide date and time in format RRRR-MM-DD\THH:MM:SS

API for issues

- Get all issues with optional filter on status that can be “open” or “closed”: GET/issues?status={issue_status}&type={issue_type}
- Get issue by ID: GET /issues/{_id}
- Add issue: POST /issues
- Delete issue: DELETE /issues/{_id}
- Switch status of issue: /issues/{_id}

Returned issues in json will have those indexes:

- _id – hashed mesage,
- issue_message – entire log,
- issue_type – can be ”warning”, “error” or “callstack”
- parent_log – a list of IDs of logs that contain log with this message,
- issue_status – can be “open” or “closed”

API for both

- Clear the database: POST /clear_db

When adding a log, keep in mind the issues inside will be added automatically into database, but when you delete the log, issues will only remove this log from their parent list.