fix cronjob and readme
Karim Hamscho authored 47 minutes ago

Verified

README.md 5.89 KiB

Index

1. Architecture

Clipper is an online clipboard service allowing users to upload and save files and text for a specified duration. The App consists of five Docker services managed via Docker Compose:

1. **Postgres Database**:
    - Acts as the primary database for storing user data, including uploaded files and text entries.
2. **Clipper Services (3 Instances)**:
    - These are identical instances of the Clipper Django application.
    - They handle user requests, process uploads, and manage data interactions with the Postgres database.
    - Running multiple instances ensures load distribution and high availability.
3. **Nginx Server**:
    - Serves as a reverse proxy and load balancer.
    - Distributes incoming traffic evenly across the three Clipper service instances.
    - Also handles serving of static files efficiently.

This setup ensures that the Clipper app is robust, scalable, and can handle multiple user requests simultaneously.

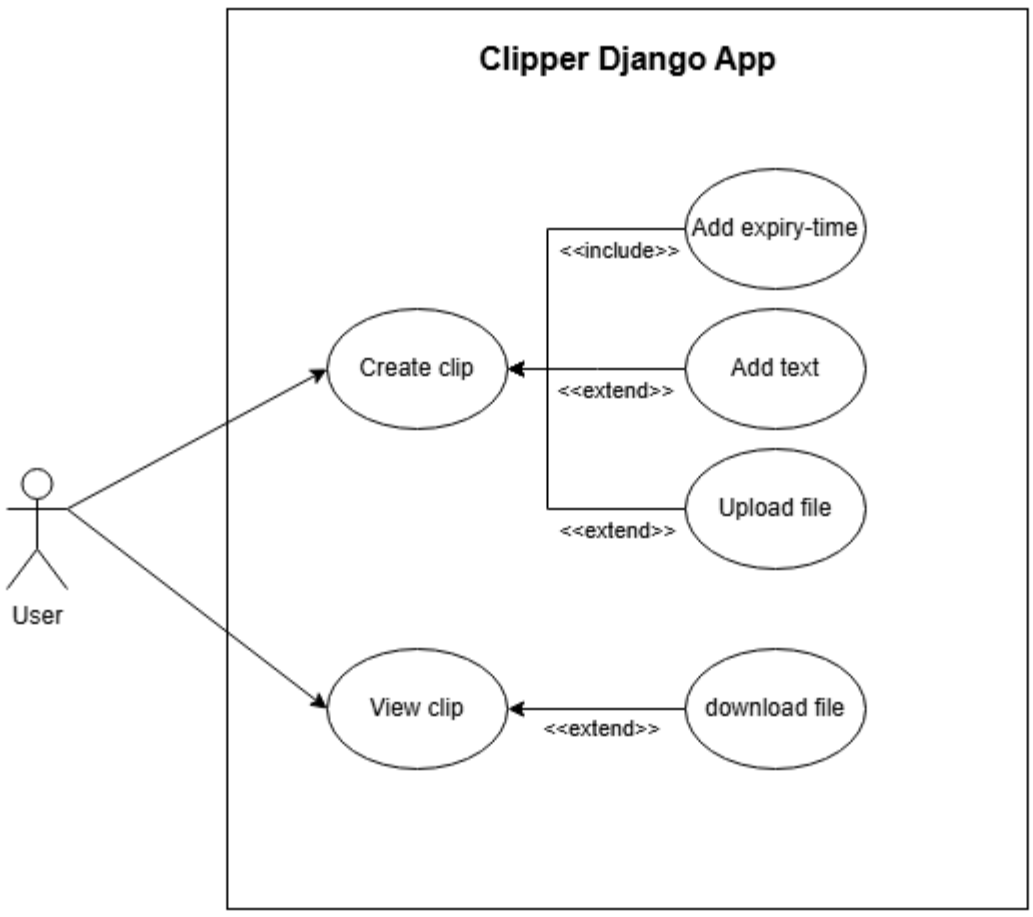
2. Django (Clipper app)

This chapter discusses the functionality, design pattern and components of our Clipper app written in the Django framework.

1. **Functionality**

The user sets an expiry time for this clip at it's creation.
The clip is then saved on this suburl and another user can visit the clip and download the attached file or copy the text.\

The following use-case diagram displays these functionalities:



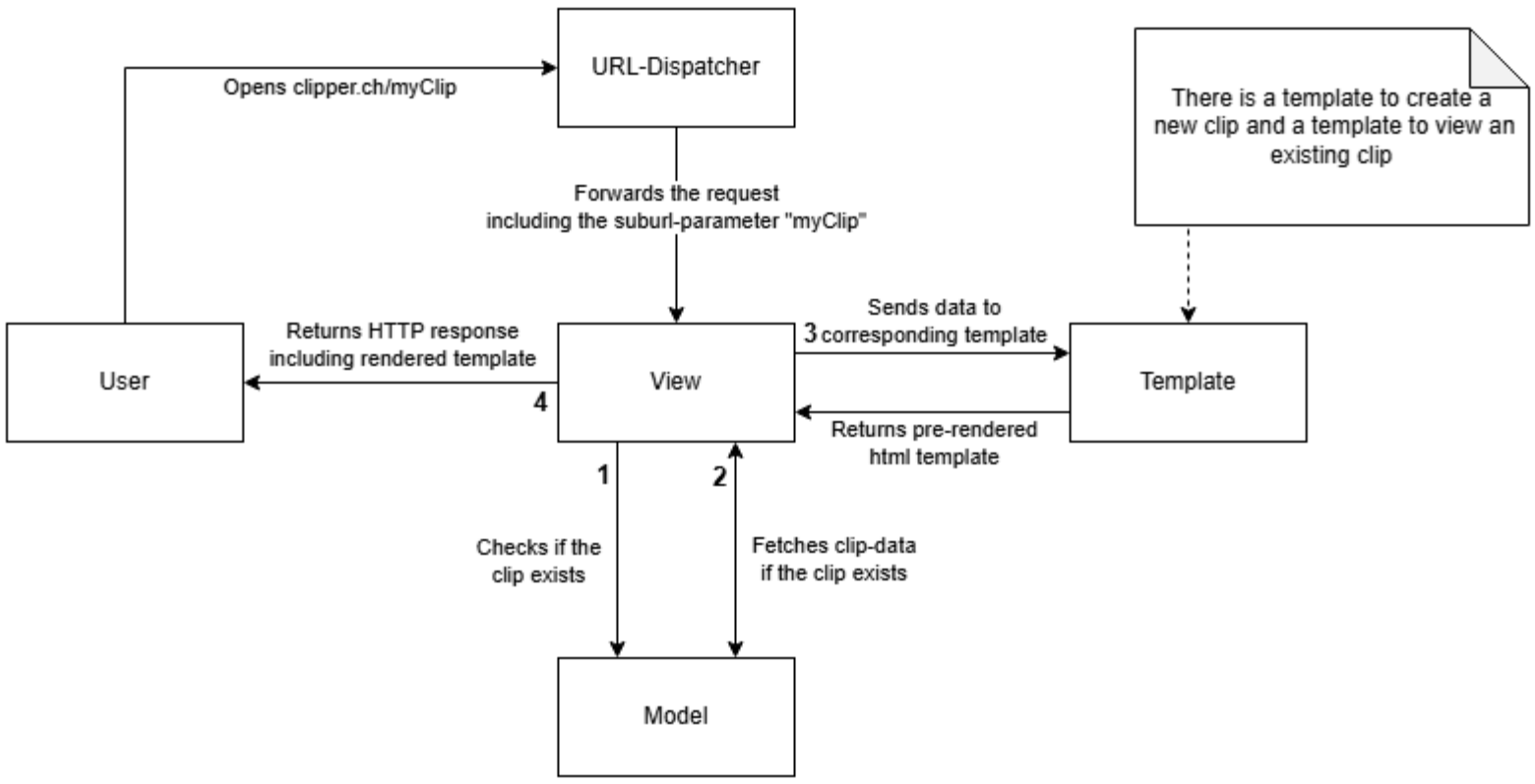
2. **Design pattern**

   Our application uses the Model-View-Template design pattern provided by Django.
   How the components of this design pattern interact with each other and the user will be explained in the following section.

3. **Components**

   Our clipper app works using the following components:
   URL-Dispatcher: Catches the suburl provided by the user and forwards the request to the view.
   Model: Defines the application's data, representing database tables and handles data querying and manipulation through Django's ORM.
   Template: Defines the HTML structure and presentation of the response, using a templating language to dynamically generate web pages based on the data provided by the view.
   View: Contains the logic to process a request, interact with the model, and return an HTTP response including the rendered template. \

   The following diagram shows how a request by a user is processed: The user accesses a suburl and the system has to return either a page to view an existing clip or create a new one. (The creation of the clip is not included)



3. Postgres

First, we wanted to use SQLite as the database since it seemed the easiest way to get the application running.However, when we later focused on load balancing, it became clear that SQLite was not suitable for access by multiple services. Therefore, we decided to use Postgres, which is relatively easy to operate with Docker and integrates well with Django. With Postgres, we no longer had issues with inconsistent data.

4. Nginx

We use Nginx for serving the static files of our application. We have persistent storage in the form of Docker volumes, which are accessible to both Nginx and the Python Django services. The "static_volume" stores static files such as CSS files for styling. The "media_volume" stores all files uploaded by users. The logic in our Django app places the files in the corresponding volume, and Nginx serves them from the same volume.

4.1 Load Balancing

For load balancing, we use Nginx to distribute incoming requests across multiple instances of our Clipper app. This is configured in the Nginx configuration file using the `upstream` directive. The `upstream clipper` block defines three backend servers: `clipper1`, `clipper2`, and `clipper3`, each running on port 8000.

The server block listens on port 80 and uses the `proxy_pass` directive to forward requests to the defined upstream servers. Additional proxy settings ensure proper handling of client requests and responses:

- `proxy_connect_timeout` and `proxy_read_timeout` set timeouts for establishing connections and reading responses from the backend servers.

5. Docker Compose

To manage all services, we use Docker Compose. This setup includes three instances of our Clipper app which work together to enable load balancing via the Nginx service. Each Clipper instance runs with Gunicorn, ensuring the robustness and scalability of our application.

The Nginx service is configured to distribute incoming requests across these three instances, enhancing reliability. Additionally, we have a PostgreSQL database service (`db`) that serves as our backend database.

To ensure persistent storage and data consistency, we have defined three Docker volumes:

- `postgres_data` for storing the PostgreSQL data, to make sure that our database does not lose data across container restarts.
- `static_volume` for holding static files such as CSS.
- `media_volume` for storing user-uploaded files, ensuring that these files are preserved and accessible.

For our cronjob we use a service which is specified in the composefile. This service registers a cronjob which executes a simple shell script. The script connects to our postgres database and deletes all entries that have expired.

7. Usage

1. `docker compose up -d --build`
2. Visit http://localhost:1337/ and type anything you want after the /
3. `docker compose down` to stop all services