CloudySky Automoderator

1 Overview

As an extension to the CloudySky project, I implemented a standalone automoderator that performs basic automated content review using the platform's existing API. The base system supports hiding posts and comments through dedicated endpoints, but all moderation activity is manual. The automoderator introduces an automated pipeline that reviews newly submitted content, checks for violations based on a configurable list of banned terms, and issues hide requests when needed. The tool runs independently of the Django server and interacts with it only through authenticated API calls.

2 Motivation

The specification provides endpoints for content suppression, but relies entirely on administrators to find and hide problematic posts and comments. This approach works for low activity levels, but becomes slow and inconsistent as the amount of content grows. Manual moderation also leaves a window of time where violating content remains visible.

The automoderator demonstrates how CloudySky's API can support automated tooling, and shows what a basic rule-driven moderation system could look like if the platform needed to scale.

3 System Description

The automoderator script performs the following steps:

1. Logs in as an administrator using Django's session-based authentication.
2. Fetches the complete feed through the `/api/dumpFeed/` endpoint.
3. Scans each post title, post body, and comment for banned terms.
4. Sends requests to `/api/hidePost/` or `/api/hideComment/` when a violation is detected.
5. Records the actions taken and prints a short report at the end of the run.

The banlist is stored as a Python list near the top of the script and can be edited without modifying any logic.

4 Design

The script is organized into three main components:

CloudySkyClient

Handles all communication with the Django server. This includes session cookie management, CSRF handling, logging in, fetching the feed, and issuing hide requests.

ModerationEngine

Coordinates the moderation pass by iterating through posts and comments, determining which items should be hidden, and recording actions and statistics.

Checking Functions

Encapsulate the rule-checking logic. At the moment they perform simple case-insensitive substring matching, but they can be replaced or extended with more advanced scoring or machine-learning models.

5 Extensibility

Because the checking logic is separated into small functions, it is straightforward to extend the system. Possible additions include:

- Incorporating toxicity or spam detectors.
- Adjusting rules based on user roles or posting history.
- Running the tool on a schedule for continuous monitoring.
- Adding support for external moderation services.

6 Testing

The script was tested locally on a CloudySky instance using a mixture of clean posts and intentionally violating content. In these tests, the automoderator:

- Correctly detected every post and comment containing banned terms.
- Left clean content untouched.
- Hid violating content through the appropriate endpoints.
- Produced a clear summary including counts and reasons for all suppression actions.
- Respected CloudySky's access control rules (admins can still see hidden content; regular users cannot).

The script handled unexpected or incomplete data without crashing.

7 Usage

The script is run from the command line:

```
python automoderator.py --url http://localhost:8000 \
    --username admin --password admin
```

Optional flags allow customization of the server URL, timeout, and login credentials. During execution, the script prints live status updates, then displays a summary of all actions taken.

8 Conclusion

The automoderator is an optional extension that illustrates how CloudySky can support automated moderation workflows through its API. The tool reduces the amount of manual review required, provides consistent rule enforcement, and offers a base that can be expanded with more sophisticated analysis in the future.