Post Submission Logic:

My Proposal for executing post submission after a response from a user is using a **pipeline-based** approach.

The motivation and the idea for this approach are based on the pipelines and jobs that we see in GitLab CI/CD that run once we have pushed into a branch. The requirement is the same in our problem statement hence I am proposing a **pipeline-based** approach where we can easily configure, that is, add new features, remove new features, modify new features without changing the logic of response, that is, in a **plug n play fashion.**

Approach:

- Pipelines contain jobs. These are the post-submission features that we wish to add.
- When a form response submit API is called we create a pipeline that contains the jobs of the features we wish to provide.
- Each job executes a function/set of functions related to a feature.
- Run a scheduler in the background that executes pending pipelines from time to time.

Of course, there are various approaches, for example, we can simply call the necessary function for each feature once the form's response has been successfully saved in the database in the respective API call itself, but the **advantages** of the proposed approach over such approaches are as follows:

- Supports **Plug n play** implementation that is required, as adding a new feature implies adding a new job to the pipeline.
- The separation of logic and Scalability - Does not affect the execution of the form response submission endpoint. Suppose there is a large amount of post submission logic and the functions related to these features are directly called in the response submission API then the user needs to wait in the loading screen without any idea whether his response was submitted or not.
- Asynchronous execution - Parallel execution in the background and is independent of the API.
- Fail-proof from service outages as pipelines that are not executed will be executed when the server is back on.
- Extendable: This Logic is not specific to this problem statement and can be used for various other tasks as well. A pipeline is a list of jobs and using different job types we can create multiple types of pipelines.
- Ease of Monitoring: Can be easily monitored by using the pipeline and jobs data (status, etc) and their logs as well.
- Can incorporate users' interests in the pipeline. For example, a certain number of users would like to receive an email of their responses and some may not be interested, we can add a job to a pipeline based on the user's preference.

Disadvantages:

- Debugging failed pipelines/jobs: Although it is easy to identify failed pipelines from the monitoring statistics, it gets difficult to reproduce the same error and debug the cause and fix it. Also takes a lot of time as we don’t have the errors shown directly.