System Design Take-home

Objective

Your task is to design the backend architecture for the **video upload flow** in a video-sharing platform — think of the "upload to YouTube" process, excluding the viewing experience.

You are **not** required to build a UI or frontend. Instead, focus on architecting the systems and services needed to support **uploading**, **processing**, and **storing** video files so they are ready for future playback.

Scope

Rather than building a full YouTube clone, this assignment focuses solely on the **video upload workflow**. For example, your design should account for:

- Users uploading video files via a web or mobile UI.
- Storing uploaded videos efficiently and reliably.
- Transcoding or processing videos into multiple resolutions/formats.
- etc.

You can assume there is already a simple UI that lets users select and submit videos to upload.

Business Context

Please use the following business context to guide your design:

- You will have **5 developers** to build and maintain this system.
- The upload functionality is part of a **larger platform scheduled for public launch in 6 months**.
- Expected scale is **20,000–50,000 Daily Active Users**, mostly in **Canada and Europe**.
- Users are uploading videos from both **mobile and desktop** devices.

- Uploads may happen during spikes (e.g., events, viral trends).

Time Expectations

Please spend **less than 5 hours** on this assignment.

In your follow-up interview, you'll have **1 hour to present** your architecture and reasoning.

Required Deliverables

Submit the following via a GitHub repository:

1. **System Architecture Diagram**
    - Show how a video upload request flows through the system.
    - Include relevant components such as:
        - Upload APIs or presigned URL generation
        - Storage (e.g., object storage)
        - Processing/transcoding pipeline
    - The diagram should be self-contained and suitable for presentation.
2. **Technology Stack**
    - Describe the technologies, tools, and languages you'd use for each part of the system (e.g., S3 vs. GCS, Node.js vs. Python, ffmpeg for processing).
    - Justify your decisions in context of the team size, timeline, and user base.
3. **Design Trade-offs & Assumptions**
    - Document trade-offs you made (e.g., eventual vs. strong consistency, cost vs. speed, batch vs. real-time processing).
    - Highlight any key assumptions (e.g., max video size, average upload frequency).

Optional Bonus Deliverables

If you'd like to go beyond the base requirements, you may include:

- **API Specification**
  - Define APIs related to uploading videos, checking status, or submitting metadata.
  - Include HTTP methods, payloads, and sample requests/responses.
- **Code Snippets or Prototypes**
  - Sample code showing how an upload or transcoding job might be triggered.
  - Worker script examples for handling thumbnails or metadata.
- **Operational/Infrastructure Notes**
  - How you'd deploy and monitor this system (e.g., cloud provider, CI/CD, autoscaling).
  - Fault-tolerance and retry strategies.