

# Arquitetura de Software

## YouTube System Design

### Architectural Requirements

- Functional Requirements:
  - User Features:
    - An authenticated user can upload a video with various resolutions and formats with a maximum size of 1GB, quickly and will be persistently stored.
    - A user can watch videos smoothly without interruption.
    - A user can access metadata from already uploaded videos.
- Non-Functional Requirements:
  - Usability:
    - A user anywhere in the world can watch videos, if authenticated, upload videos as well.
  - Modifiability:
    - The user, when the video playback is not optimal, can change the video quality making it run smoother.
    - 5 million users can make requests in a day without the system going into overload.
    - A user tries to watch a video that was uploaded in a format incompatible with his current device, but it streams without issues.
    - 150 Terabytes of data can be stored in the system daily.
    - When a user watches a video, he should have a smooth video streaming experience for a minimum of 30 minutes.
    - 100.000 Users can watch videos simultaneously without issues.
    - 500.000 videos can be uploaded per day without the system having issues.


- Availability:
  - Use of already built infrastructures:
    - The user tries to watch a video and fetches it in less than a second.
  - Video Transcoding:
    - The user tries to upload a video with a format not compatible with other devices, but the system accepts it.
    - The user tries to load and watch a video and fetches it with a latency of  $\leq 1$  second 90% of the time, 1 to 2 seconds 9% of the time, more than 2 seconds 1% of the time, and watches it continuously without interruption.
    - A user tries to upload a video and he gets an error, but the system responds promptly to the failure or bottleneck.
    - A Metadata DB server goes down the system will continue to function properly.
    - API server goes down, the system can adapt to maintain functionality.
    - If a service provided by a third party provider such as the CDN is not available, the system will recover in accord with the SLA defined between the service provider and the builder of the system where it should provide availability of 99%.
- Security:
  - Authorized user uploads a video, and its authenticity is upheld.
- Performance:
  - The user uploads a video, and it does so in the fastest way possible.
  - Users from different regions try to access a video and the time it takes must be short and equal for every single one.
  - A great number of users access the same video at the same time, the video should be watchable for each user simultaneously.
- Constraints:
  - Mobile App, Web App, Smart TV App
  - Recommendation to leverage existing cloud infrastructures provided by Amazon, Google, or Microsoft.
  - Use of CDN
- Architectural Concern:
  - Number of Users

- Localization of the Users, glob~~al~~ization of the app
- Lower infrastructure cost

Scenarios should be more rigours.

# Round Definition

- 1º Round
  - Purpose of Design:
    - Fully Functional System for Upload and Stream without Scalability and Availability
  - Iteration 1:
    - Primary Functionalities
  - Iteration 2:
    - Performance for uploading videos (Preprocessor of the DAG video transcoding architecture)
  - Iteration 3:
    - Performance of organizing tasks queues and manage resources for task workers (DAG scheduler, resource manager, tasks workers from DAG video transcoding architecture)
  - Iteration 4:
    - Performance for accessing storage and temporary storage (Temporary storage of the DAG video transcoding architecture)
  - Iteration 5:
    - Performance of encoding videos (Encode Video from DAG video transcoding architecture)
  - Iteration 6:
    - Security for uploads
  - Iteration 7:
    - Security for authenticity

- 2<sup>o</sup> Round
  - Purpose of Design:
    - Add Scalability
  - Iteration 1:
    - Increase Requests
  - Iteration 2:
    - Increase Users
  - Iteration 3:
    - Increase Storage
- 3<sup>o</sup> Round
  - Purpose of Design:
    - Add availability.
  - Iteration 1: 
    - Decrease inaccessible time.
  - Iteration 2:
    - Lower transcoding failure
  - Iteration 3:
    - Recover progress from DAG scheduler, resource manager and task worker.
  - Iteration 4:
    - Maintain user accessibility to the API.
  - Iteration 5:
    - Add cache rerouting for videos and metadata.
  - Iteration 6:
    - Recover database functionality using warm redundant spare and maintain master-slave structure.

Associate scenarios with iterations.