21 - App Limitations

08:10

Limitations and Future Work

> IMPORTANT: Don't forget to clean up your project after you are done with this course if you want to save your money.
> The resources we created include Cloud Run services, a Firestore database, Firebase

> Functions, a Cloud Pub/Sub topic and subscription, and Cloud Storage buckets.

In this section, we will discuss the limitations of our application, especially with how we architected our app.

We will also discuss some possible future work that can be done to improve and flesh out our app. Some of these tasks may be interesting to you and I encourage you to try to implement them.

I think this project is already an exceptional one to put on your resume, but if you want to take it to the next level, you can try to implement some of the future work items, or even re-architect the app. That's the best way to learn!

1. Limitations

1.1. Long Lived HTTP Requests

For Pub/Sub the max ack deadline is 600 seconds, but the max request processing time for Cloud Run is 3600 seconds.

We ensured that a video won't be processed multiple times by syncing the video status in Firestore.

But, if the video processing takes longer than 600 seconds, Pub/Sub will close the HTTP connection. Now while the video will still be processed by out service, we will never be able to send the `ack` to Pub/Sub. This means the message will be stuck in the Pub/Sub queue

forever, and will continuously be redelivered to our service.

So how can we fix this?

We can use the **Pull Subscription** method instead of the **Push Subscription** method. This way we can control when we want to pull messages from the Pub/Sub queue. We can pull a message, process it, and then send the `ack` to Pub/Sub.

After we pull a message it is temporarily removed from the queue. If we don't send the `ack` within the ack deadline (still a max of 10 minutes), the message will be redelivered to the queue.

But since we are still syncing the status with Firestore, we won't reproduce the video.

By pulling messages, we can ensure that no message will be stuck forever in the Pub/Sub queue.

Or can we?

1.2 Video Processing Failure

If we pull a message from the Pub/Sub queue, change the status of it to `processing` in Firestore, and then our video processing service fails, the message will be stuck in the Pub/Sub queue forever.

We could send an `ack` as soon as we pull the message from the queue, but then what would happen if we fail to process the video? We would have to

somehow requeue the message. The easiest solution would be to reset the status to `undefined`.

1.3 File Upload Time Limit

This is actually *not* a limitation. You might think that since the signed URL we generate is only valid for 15 minutes, that if we have a slow internet connection, we might not be able to upload the video in time.

But as long as we start the upload within 15 minutes, the upload will continue even after the signed URL expires. This is because the signed URL is only used to authenticate the upload request. After the upload request is authenticated, the upload will continue until it is completed.

> If anything, we should shorten the time limit for the signed URL to 1 minute.

1.4 Video Streaming

Google Cloud Storage provides a basic level of file streaming for videos. But it is not optimized for video streaming. This is good because we don't have to download the entire video before we can start watching it.

But it still isn't nearly as powerful as Youtube's custom video streaming / chunking solution.

Youtube uses **DASH** (Dynamic Adaptive Streaming over HTTP) to stream videos. This allows them to serve

different quality videos to users based on their internet connection. It also allows them to serve the video in chunks, so that the user doesn't have to download the entire video before they can start watching it.

Another popular alternative is **HLS** (HTTP Live Streaming). This is what Netflix uses.

> I believe Youtube also uses HLS in some capacity.

1.5 Content Delivery Network

Ideally, we would want to serve our videos from a CDN (Content Delivery Network). This would allow us to

Mark Lesson Complete

1.6 Illegal Videos

We have no way of checking if a video is illegal before serving it to users. This is a big problem. For this reason, it may be unsafe to deploy this app publically.

2. Future Work

Below is a non-exhaustive list of future work that can be done to improve our app.

- [ ] Add user's profile picture and email to Web Client
- [ ] (Bug fix) Allow users to upload multiple videos without refreshing the page
- [ ] Allow users to upload thumbnails for their videos

- [ ] Allow user's to add a title and description to their videos
- [ ] Show the uploader of a video
- [ ] Allow user's to subscribe to other user's channels
- [ ] Clean up raw videos in Cloud Storage after processing
- [ ] Use a CDN to serve videos
- [ ] Add unit and integration tests (lol... I know)

3. Conclusion

Thank you for making it this far! I hope you learned something new, but even more importantly I hope you improved your thought process when it comes to designing and architecting applications.

As you may now realize, there are a lot of nuances when it comes to designing and architecting applications. And there is no one right way to do it. It's all about tradeoffs.

> And also, no one can build an app like Twitter or Youtube in a weekend.

Please let me know if you have any feedback. I love to improve!

Full Stack Development
19 / 22
Storage