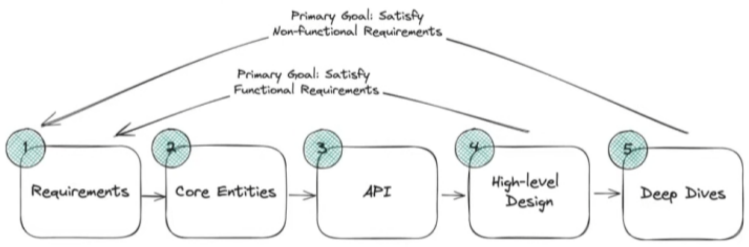
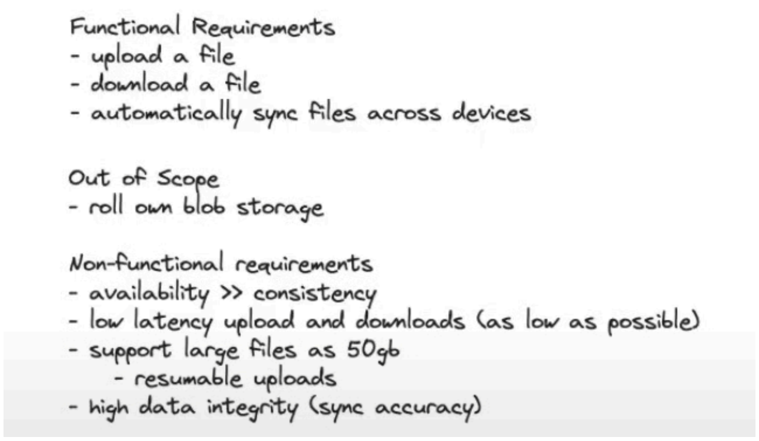
Youtube Video

# Issue Number	1
Progress	in progress
Type	reading
link	https://www.youtube.com/watch?v=_UZ1ngy-kOI

Detailed writeup of the video - https://www.hellointerview.com/learn/system-design/problem-breakdowns/dropbox


Primary Goal: Satisfy Non-Functional Requirements
Primary Goal: Satisfy Functional Requirements
1
Requirements
2
Core Entities
3
API
4
High-level Design
5
Deep Dives



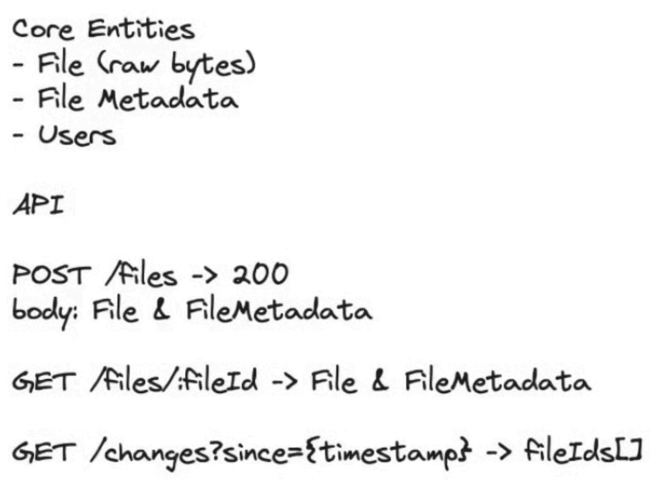
Functional Requirements
- upload a file
- download a file
- automatically sync files across devices

Out of Scope
- roll own blob storage

Non-functional requirements
- availability >> consistency
- low latency upload and downloads (as low as possible)
- support large files as 50gb
    - resumable uploads
- high data integrity (sync accuracy)

Step 2

ENTITIES & API



Core Entities
- File (raw bytes)
- File Metadata
- Users

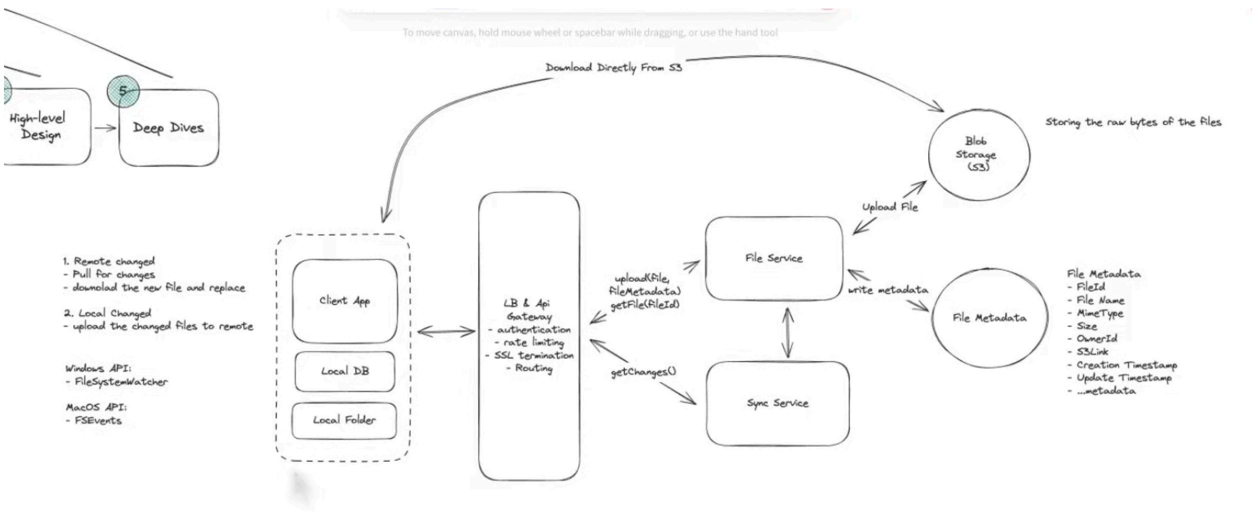
API

```
POST /files -> 200
body: File & FileMetadata

GET /files/:fileId -> File & FileMetadata

GET /changes?since={timestamp} -> fileIds[]
```

- userid will be present via a sessionid.
- Better to send the file metadata in GET request →File Metadata[]



1. Remote changed
- Pull for changes
- downolad the new file and replace

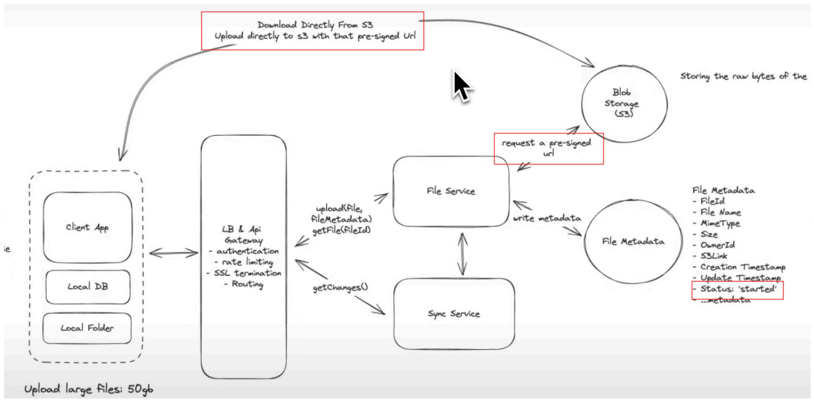
2. Local Changed
- upload the changed files to remote

Windows API:
- FileSystemWatcher

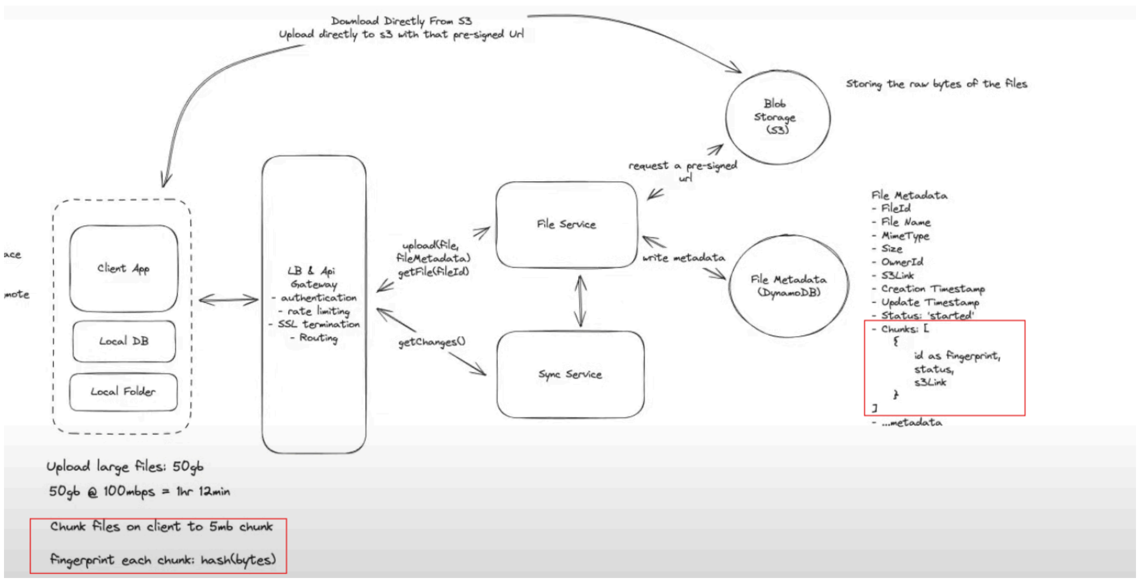
MacOS API:
- FSEvents

File Metadata
- FileId
- File Name
- MimeType
- Size
- OwnerId
- S3Link
- Creation Timestamp
- Update Timestamp
- ...metadata

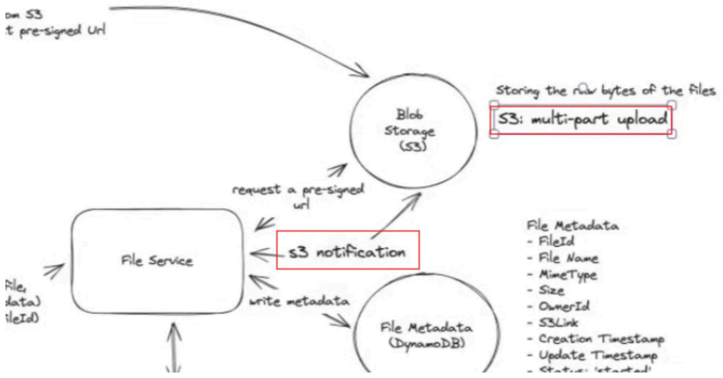
- This design will not work for large files.
    - upload path is redundant request body has a limit. it is set by browser, aws managed api gateway.
        - upload directly to S3. add a metadata filed to show the status of upload. get a presigned url from S3 so S3 will maintain a connection with the URL for some default time and user can directly upload the file using the URL



        - The problem with uploading a large file is the time required. 50GB file can take 1 hour in a 100mbps network. So we will send the data in chunks. and maintain the status of the chunk in metadata. We will use FingerPrint to get a unique id for the chunk.

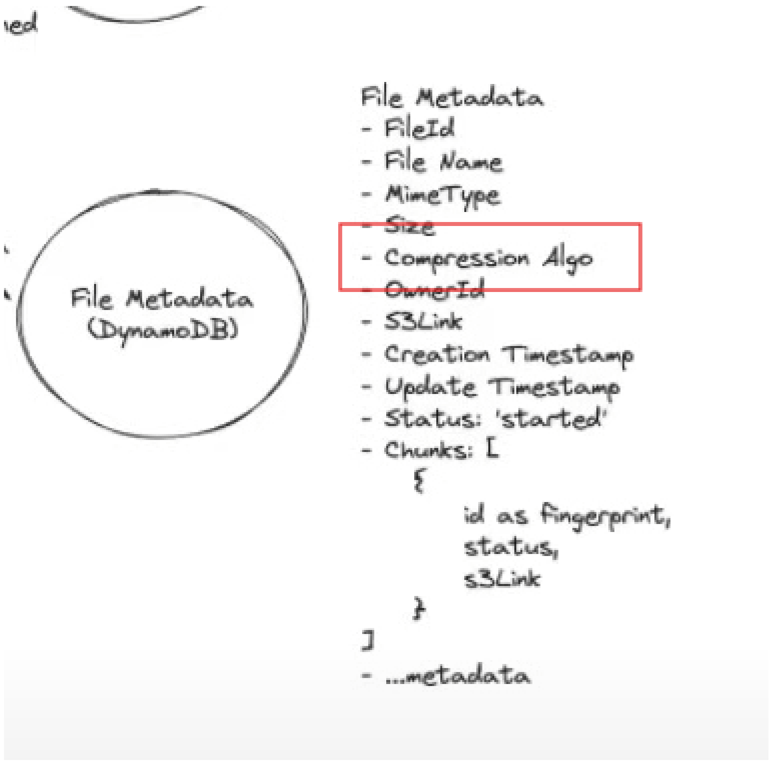


- Their is a issue → how to update the status of each chunk in metadata. We can relay on User to send a PUT request to update the status. But that will create inconsistancy between S3 storage and Metadata.
- We can do Trust but Verify system where we will verify with the S3 at the File Service step when a PUT request comes.
- S3 Notification→ when a new file is uploaded it will send a notification to the file service. S3 uses multi-part upload for this. But their are some cases where s3 notification doesn't comes using multi-part upload so he is recommending Trust but Verify approach.

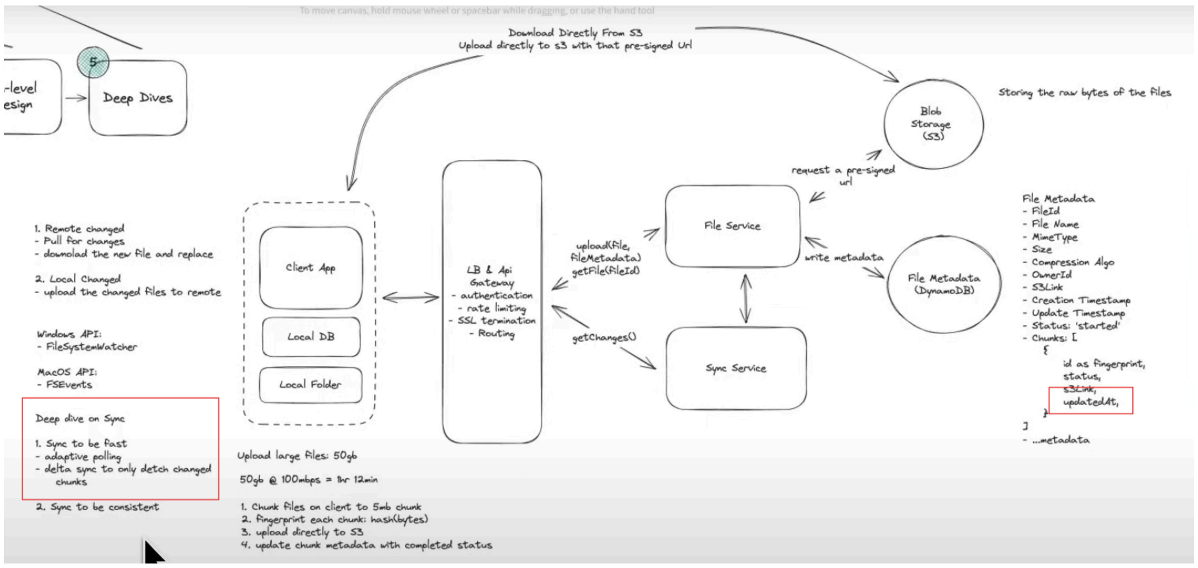


- Low latency upload and download

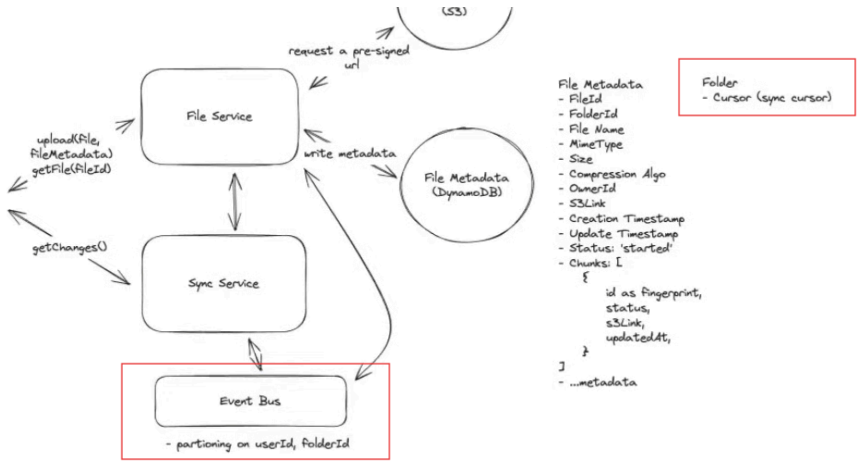
- For download we used a CDN to cache the file closer to the user. but CDN are expensive. So it will add value for users who are traveling or we are uploading a file with lot of popularity.
- Transfer fewer bytes. using compression algo according to the file size and network bandwidth.



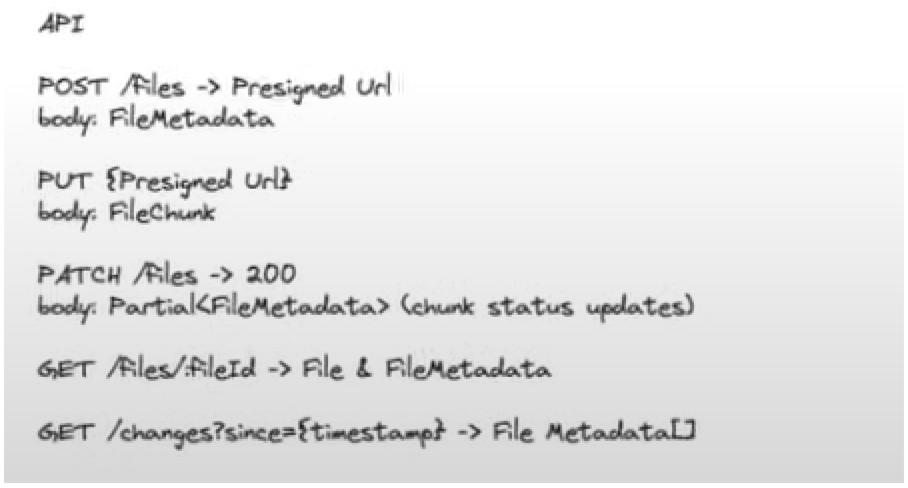
- high data integrity for sync accuracy
  1. Sync to be fast
  2. Sync to be consistent
  - Adaptive polling is better way to go. We don't need to see the changes made within milli-seconds. Long polling or opening a persistent connection using webSocket is overkilling not required.
  - If changes are made, We don't want to download the whole file. We will use chunking. We will download the only the chucks with later updatedts.



- At the time of polling giving the folderid and asking what are the file changes after a particular timestamp.
- Using a event Bus like kafka. any changes occured to any file we will put that change event to the event bus. we are maintaining a sync cursor and update the cursor where the update process is. Next time we will go to the event bus and check the last event we read and apply all the next changes.
- Event bus with cursor is great for **data recovery**, version control and audit trail.



- reconcilliation → periodically (weekly, daily) we will sync all the folder and make sure everything is consistent.



```
API

POST /files -> Presigned Url
body: FileMetadata

PUT {Presigned Url}
body: FileChunk

PATCH /files -> 200
body: Partial<FileMetadata> (chunk status updates)

GET /files/:fileId -> File & FileMetadata

GET /changes?since={timestamp} -> File Metadata[]
```