# System Design GDrive/DropBox

Functional Req: 1)User Login/Authentication - User should be able to login / create account. Each user given around 30 GB of free space

2)CRUD operations on files

3)Multi device support

4) Sharing of files – Either via expirable link share or permission to read/edit.

5) Version history of files - Multiple versions can be stored. The users with whom files are shared can view only the latest version.

Non Functional : 1)Scalable

2) ACID Compliant

## Services

- User Authentication
- File Sharing
- CRUD Operations
- Notification System
- Synchronization

## APIs

1) User Authentication - SessionToken login(username, encrptedpass)
2) String generateSharableLink( sessionToken, FileId)
3) shareWithUser( sessionToeken, userId )
4) upadateChunks(sessionToken, Chunk[] chunksToAdd, Chunk[] chunksToRemove, String checkSum )
5) Chunk[] openFile( sessionToken, FileId, deviceId) //Get all chunks for given file and mark that file as currently in R/W
6)Chunk[] refreshFile ( sessionToken, FileId , String[] chunksHashes) // Retrieve all dirty chunks

## Table Model

### File sharing NoSQL

1) File ID -> {userId1:permissionLevel,

userId2:permissionLevel...}

2) FileID -> sharableLink

* Using NoSQL since this schema matches K-V pairs, Is mostly write once

| User | |
|---|---|
| **PK** | **UserID** |
| | UserName<br>Encrpyted Pass |

User table needs not be partitioned since assuming 1B users, all their records can be stored in one server and indexed on userID.

| File Metadata | |
|---|---|
| **PK**<br>**FK** | **FileID + VersionID**<br>**UserID** |
| | FileName<br>File location Dir URL<br>File Size |

Files table can be huge and needs to be partitioned. 2 options - by UID, by FileID. If UID then possibility of uneven distribution into partitions. But files for same user will be in same partition. Searching of files will be fast.

If partition by FileID, then even distribution. but getting all files for current user would require search multiple partitions. To get over this an index can be created with key as userID and values as all partitions for that user.

| File Chunks | |
|---|---|
| **PK** | **FileID + ChunkID** |
| **FK** | **UserID** |
| **FK** | **FileID** |
| | IsDirty<br>ChunkName |

Chunks data stored in Cloud storage. The table contains URL of folder containing chunks. Table can be partitioned by FileID so that all chunks details are in same partition.

## CRUD service

1) When uploading file, the client has a splitter process which splits the file into chunks.

2) To maintain ACID, After uploading all chunks, a checksum is sent at the end. This checksum is compared with checksum generated at server. If match then transaction marked completed.

3) On Updation, Hashes of all chunks are generated, The

list of hashes are sent. The server compares with existing chunks, The hashes whose chunks are not present on server, are again requested from client. The file version is incremented and chunks are stored with that FID+VID+CID key

## Sync Service Between Devices

1) When user opens file, an entry is generated in Cache of all devices currently reading the file with ID and Version.

2) When one device edits file and uploads the modified version. The cache is looked up to get all devices currently reading older file.

3) Notification service sends notification to all devices to refresh content. The devices on accepting that requests the new version.

4) Hashes of older file version and newer file version chunks are compared. The modified chunks are sent to the devices.

## File Sharing Service

1) When user shares file with other users, entry is added in file sharing NoSQL db.

2) Notification service sends mail to shared user's email about the file URL

3) The user can invoke openFile API. The session token is used to get user id, From the file sharing db, list of valid user ids is retrieved, If the current user id is present, the API returns all chunks from the location and also the checksum. The file hashes and current user are added to cache.

4) If the owner modifies the file, Notification service sends info to all users currently reading file to refresh.

5) On refresh, the sync service is used to update file for all users.