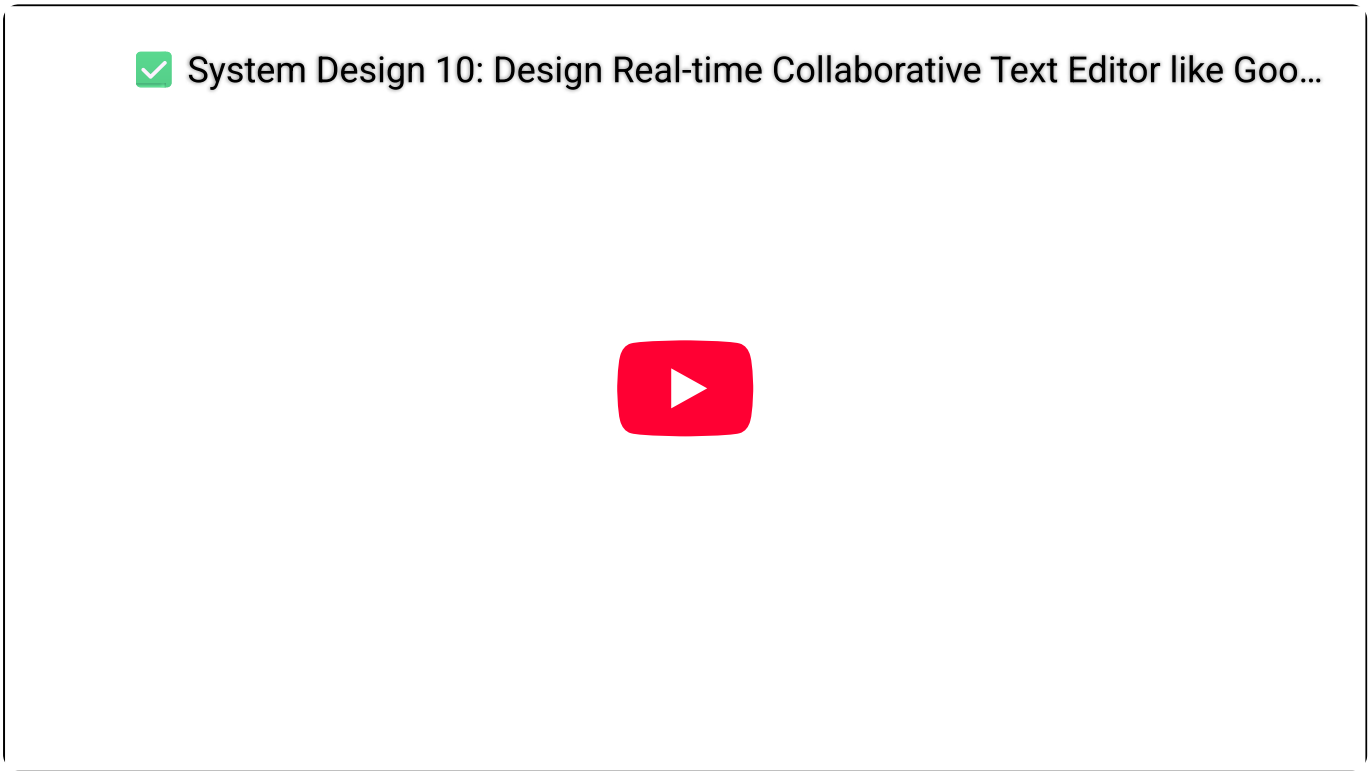
System Design 10: Design Real-time Collaborative Text Editor like Google Docs | Notion | HLD | LLD



In this system design tutorial, we walk through the architecture, components, and data flow needed to build a Google Docs clone that supports multiple users editing the same document simultaneously.

Design Diagram

Design a Real-time text editor (Google Docs)

- It is an online word processor, that lets you create and format documents and work with other people in near real time

1. Functional Requirements:

- Users should be able to create/update/delete documents.
- Multiple users can edit the same document simultaneously
- Users should be able to view each other's changes in real-time
- Users should be able to see the cursor position and presence of other users
- Versioning of the files

Non-Functional Requirements:

- Scale: Millions of users, with Billions of documents
- CAP Theorem: Availability > Consistency (Normal Document)
- Latency: UPdates should be low latency (~ 100ms)

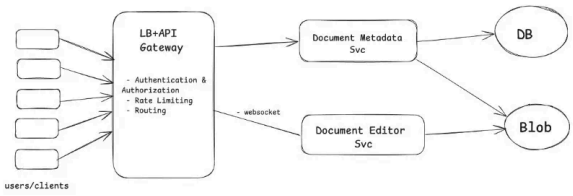
2. Core Entities

- User/Editor
- Documents
- Edit
- Cursor

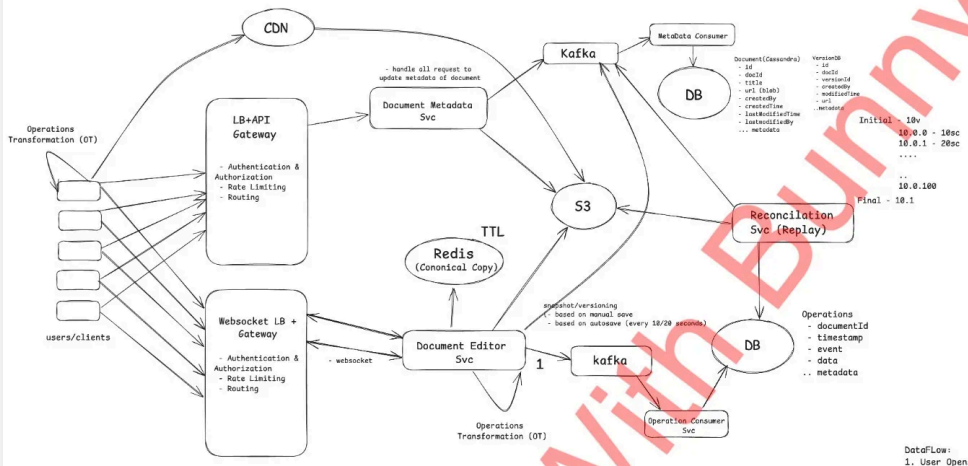
3. API Designing:

1. POST: /v1/api/docs/create - create a document (Returns a document Id)
2. GET: /v1/api/docs/{docId} - view Document (ReadOnly)
3. WS: /v1/api/docs/{docsId} - Edit document
4. GET: /v1/api/docs/{docId}/version - give all the version
5. GET: /v1/api/docs/{docId}/{versionId} - open a specific version

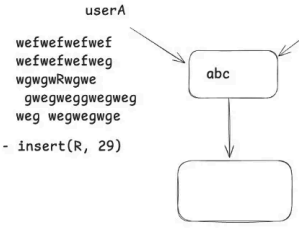
4. High Level Designing



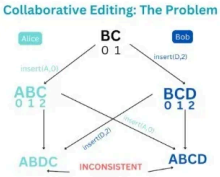
5. Low Level Designing



DataFLow:
1. User Opens a Document (Read-Only Mode)
- Frontend calls GET /doc/{docId} via Document Metadata Service
- Frontend fetches the document snapshot from S3 (Blob Storage)
- Document is shown in read-only mode
2. User Starts Editing (Triggers WebSocket)
- Frontend initiates WebSocket to Document Editor Service
3. Editor Service Handles WebSocket Join
- Document Editor Service checks Redis:
- If Redis has doc → continue (perform OT and send latest canonical copy to user
- If Redis is empty: Load snapshot from S3. Replay unprocessed ops from Kafka (v52 → vN)
- If Kafka is empty → assume snapshot is up-to-date
4. Real-Time Editing Starts
- Client sends operations (insert/delete) via WebSocket
- Editor Service:
  - Applies Operational Transformation (OT) to reconcile
  - Updates Redis (canonical doc) in real-time
  - Broadcasts transformed ops to all connected users
  - Streams each op to Kafka for durability
5. Background Processing (Versioning & Durability)
- Kafka consumers (Operation Consumer Service) do:
- Rebuild updated document state in memory
  Every X ops or Y seconds:
  Save snapshot to S3
  Update version metadata in DB
  Commit Kafka offsets



1. File Replacement
2. Locking Protocol ->
   Optimistic (Git Merge)
   Pessimistic (Banking)
3. Events passing + websocket Connector



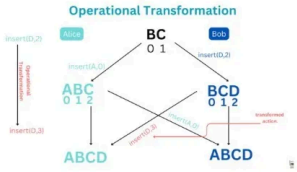
4. OT - Operational Transformation
5. CRDT - Conflict Free Replicated Data Types

Approach 4: Operations Transformation (OT)

- Operational Transformation (OT) transforms concurrent operations during merging. The transformation function ensures that the effect of an operation is adjusted to account for the changes made by other operations

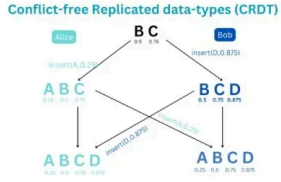
** Limitation:

- All operations must go to a single server which does the transformation
- Implementating the OT is highly complex



Approach 5: Conflict Free Replicated Data-types (CRDT)

Conflict-free Replicated Data Types (CRDTs) are data structures designed for distributed systems that automatically resolve conflicts caused by concurrent updates. They ensure eventual consistency without requiring coordination by using mathematical properties that allow operations to be merged in any order.



This is part of the complete "System Design Interview" course by Interview With Bunny.