
Mastering System Design







Design a Collaborative Document Editor (aka Google Docs)

Understanding the Problem








- A web-based collaborative document editor that allows:
 - Multiple users to edit the same document simultaneously
 - Changes to be reflected in real-time
 - Conflict-free and consistent document state
 - Document versioning and access control
- Examples: Google Docs, Microsoft Word Online, Notion





Functional Requirements (MVP)

-  Create, edit, and delete text documents
-  Real-time multi-user collaboration
-  Real-time syncing of edits across clients
-  Document version history & change tracking
-  Permission control (read/write access)
-  Invite collaborators via shareable link or email






Non-Functional Requirements

-  Performance: Real-time sync with <100ms latency
-  Scalability: Support millions of documents & active users
-  Availability: 99.9% uptime, seamless reconnects
-  Security: TLS encryption, access control, basic abuse protection
-  Reliability: Autosave, crash recovery, eventual consistency
-  Maintainability: Modular, clean APIs for easy feature evolution
-  Testability: Real-time flows must be testable under load




Assumptions and Constraints

-  Assumptions
 - Users have stable internet connections
 - Clients support modern browsers with WebSocket capabilities
 - Most documents are text-heavy, not rich media
 - Collaboration groups are small to medium (2–50 users)
 - System usage is read-write heavy (frequent edits)
-  Constraints
 - Real-time sync must work under high concurrency
 - Conflict resolution must maintain document consistency
 - System must support eventual consistency across clients
 - Latency targets must be met globally (not just regionally)
 - Only basic text formatting supported in MVP





Collaboration Challenges

-  Concurrency control: How to handle multiple edits at once?
-  Conflict resolution: Who “wins” in conflicting edits?
-  Real-time sync: How to efficiently broadcast updates?
-  Consistency vs. latency: Trade-offs between speed & accuracy
-  Failure recovery: Network loss, reconnects, partial updates






Key Scale Estimates & System Volumes

-  Users & Activity
 - 10M+ Daily Active Users (DAU)
 - Avg. 100 documents/user
 - 200K+ concurrent editors during peak hours
-  Real-Time Sync Load
 - ~10B events/day
 - ~1-2 events/sec per active user
 - 5-20 ops/sec/document during editing
-  Data & Storage
 - Avg. 100KB/document
 - 2-5x storage overhead for versioning/history
 - Hot storage: 1-5 TB/day
 - Cold storage: 100s of TBs (long-term archive)

Traffic Pattern Observations

-  Spikes during working hours, especially in shared orgs
-  Hotspot documents (e.g., meeting notes, shared templates)
-  Frequent small writes vs. bulk reads
-  Auth + permission checks add latency under load

Potential Bottlenecks

-  Operational Transformation / CRDT overhead
 - CPU-intensive; needs fast in-memory ops and ordering
-  WebSocket scaling & fan-out
 - Requires persistent, low-latency connections with horizontal scaling
-  Storage write throughput
 - Edits, version history, and autosaves add heavy write pressure
-  Conflict resolution latency
 - Must not block the real-time editing experience
-  Sync propagation
 - Ensuring consistent state across all clients in milliseconds

High-Level Architecture Overview

- **Client:** Rich text editor (browser/mobile), WebSocket client
- **API Gateway:** Auth, routing, rate limiting
- **Collab Service (Real-Time Sync):** Handles operational transformation (OT) or CRDT logic
- **Document Service:** Manages save/load, metadata, permissions
- **Versioning Service:** Handles document history, snapshots
- **Storage Layer:** Hot (fast-access), cold (archival), backup
- **Messaging Layer:** Pub/Sub for real-time event fan-out (e.g., Kafka, Redis Streams)








 Use microservices with stateless APIs, backed by real-time sync infrastructure.

Document Model Design




- Use CRDT/OT-compatible data structure (e.g., Yjs, Automerge)
- Content updates are operation-based, not full text blobs
- Maintain version metadata for sync, diff & rollback

```
{
  "document_id": "abc123",
  "title": "Design Notes",
  "owner_id": "user_001",
  "collaborators": ["user_001", "user_002"],
  "content": {
    "type": "CRDT", // or OT
    "data": "<internal structured text representation>"
  },
  "version": 148,
  "last_modified": "2025-04-30T12:03:22Z"
}
```

Real-Time Sync Flow

1.  User edits document (e.g., inserts text)
 2.  Client sends operation over WebSocket
 3.  Sync service applies OT/CRDT transform
 4.  Broadcasts changes to collaborators
 5.  Periodic autosave to persistent store
 6.  Snapshots saved every X versions for recovery
-  Optimized for low-latency & consistency via versioned ops.

Communication Patterns

-  External Communication
 - **REST (HTTP/JSON):** Used for public-facing APIs like document creation, retrieval, and version history.
 - Clients (e.g., Web, Mobile) make synchronous RESTful calls to interact with the document service.
 - Rate limiting, authentication, and authorization are handled here.
-  Internal Communication
 - **gRPC (Protocol Buffers):** Used for communication between internal services (e.g., document service, sync service, versioning).
 - Fast, lightweight communication ideal for microservices interactions.
 - Bidirectional streaming can be used to keep services synchronized in real-time.
 - **Message Queues** (e.g., Kafka, RabbitMQ): For event-driven communication and decoupling services (e.g., triggering background tasks, syncing document updates across services).
 - Used for asynchronous workflows such as document processing, backup syncs, and event logging.
-  Real-Time Communication
 - **WebSockets:** Used for real-time collaboration and document sync.
 - Establish persistent WebSocket connections for each active document session.
 - Low latency bidirectional communication to propagate changes instantly between clients.

Key APIs





- GET /documents/:id
 - Fetch document metadata and content
- POST /documents
 - Create a new document
- PUT /documents/:id/content
 - Save content snapshot (full/partial)
- POST /documents/:id/operations
 - Send an edit operation (insert, delete, etc.)
- GET /documents/:id/history
 - Retrieve document version history
- POST /documents/:id/collaborators
 - Add or remove collaborators
- WebSocket /ws/collab?doc_id=abc123&token=xyz
 - Real-time collaboration sync

Consistency & Conflict Handling

- Use CRDT (conflict-free replicated data types) or Operational Transformation (OT)
- Maintain operation logs per document
- All clients apply the same transformed ops for convergence
- Snapshots + deltas used to resync lagging clients
- Handle network partition recovery with reconnection protocol

 Client state must converge even with out-of-order operations.

Strategic Tech & Infra Decisions

-  Tech Stack
 - Frontend: React (Web) / React Native (Mobile) for rich, interactive editor
 - Backend: Node.js (WebSocket for real-time sync), gRPC (internal microservices)
 - APIs: REST for document management, WebSockets for real-time collaboration
-  Data Storage
 - Storage: AWS S3 / GCS for document storage
 - Database: PostgreSQL for metadata, MongoDB for flexible document schemas
-  Infrastructure
 - Orchestration: Kubernetes for scaling and service management
 - Message Queue: Kafka for event-driven architecture and decoupling services
-  Security
 - Authentication: OAuth2 / JWT for secure user authentication
 - Encryption: TLS for in-transit, AES for at-rest encryption

The Final Design - Collaborative Document Editor

