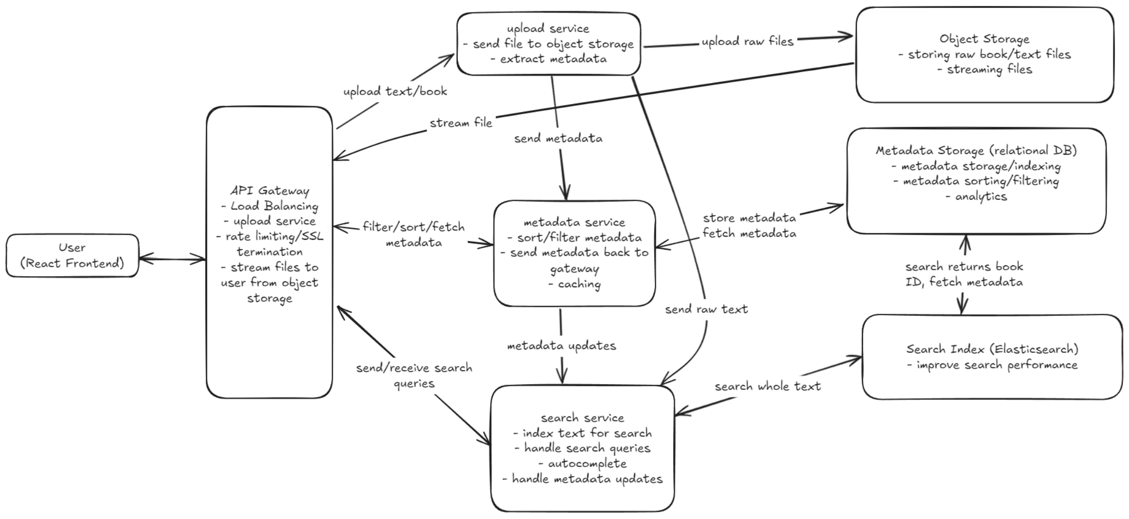
Distributed Dictionary System Design

Summary

- **User Interface (Frontend):** The frontend can be built with React.js to build a user-friendly UI with reusable components and cross-platform support on web.
- **Storing Metadata:** Metadata such as title, author, description, and date uploaded can be stored in a relational database such as PostgreSQL to provide efficient indexing and improve search performance, improving the user experience
- **Storing Book/Text files:** Raw files can be stored in a distributed object storage solution such as Amazon S3
- **Search:** Searching through full texts can be done by Elasticsearch to provide quick, efficient lookups

Rough Flowchart:


upload service
- send file to object storage
- extract metadata
upload raw files
Object Storage
- storing raw book/text files
- streaming files
upload text/book
stream file
send metadata
Metadata Storage (relational DB)
- metadata storage/indexing
- metadata sorting/filtering
- analytics
API Gateway
- Load Balancing
- upload service
- rate limiting/SSL termination
- stream files to user from object storage
User
(React Frontend)
filter/sort/fetch metadata
metadata service
- sort/filter metadata
- send metadata back to gateway
- caching
store metadata
fetch metadata
search returns book ID, fetch metadata
send raw text
metadata updates
Search Index (Elasticsearch)
- improve search performance
send/receive search queries
search whole text
search service
- index text for search
- handle search queries
- autocomplete
- handle metadata updates

Uploading Files

- The file is first sent to an API gateway to route it to an upload service, which will extract and upload metadata to the relational database, store the raw file in an object storage bucket, and send the extracted text to be indexed by Elasticsearch.
- The API Gateway will implement techniques including rate limiting and SSL Termination for security/performance purposes
- The text processing will be handled by an asynchronous process with a service like Apache Kafka to avoid bottlenecks that may block or slow down requests

Viewing Files

- The frontend sends a request to the API
- The relational database fetches the metadata for that file
- The file is retrieved from object storage and streamed to the frontend UI

Search Flow

- User submits a search query, and Elasticsearch queries its index for metadata/full-text matches
- Ranked results are returned to the frontend UI

Scalability and Fault Tolerance

- Storing the files in a distributed storage solution enables the system to handle large book/text files.
- The database can be scaled with increased traffic by implementing read replicas and sharding the database into regions, along with implementing a caching system for frequent queries
- Fault tolerance can be added by replicated backup storage and using multiple Elasticsearch nodes for redundancy