Indian Institute of Technology, Jodhpur

NAME: Aansh Chandrakant Dubey (B22AI058)

Mukund Gupta (B22CS086)

Banoth.Sri Kowshika Raj (B22CS018)

Software and Data Engineering
Major Project

Session: Fall 2025

Instructor: Professor Sumit Kalra

Introduction

The primary aim of this project is to build a full-stack, real-time collaborative document editor, named "Text Sync," that is highly reusable, maintainable, and capable of providing a rich, multi-user experience similar to Google Docs. It addresses the limitations of simplistic tutorial applications by implementing a robust, enterprise-ready solution for real-time collaboration. The project tackles the complexities of synchronizing state across multiple clients, including live cursors, simultaneous text editing, nested comments with mentions, and a complete permissions system.

By incorporating Liveblocks as the real-time backend, Clerk for authentication, and Sentry for monitoring, the application ensures seamless, low-latency interactions and enterprise-grade reliability. The project demonstrates how to build a feature-complete collaborative application using a modern stack, achieving functionalities like live cursors, floating comments, instant notifications, and role-based access control (editor/viewer) without building the complex backend infrastructure from scratch.

Tools and Technologies Used

- Next.js (Frontend Framework): Used for the client-side UI and server-side rendering.
  Reason: Provides a powerful React framework with features like server-side rendering for fast initial page loads, Server Actions for simplified backend communication, and a robust routing system, making it ideal for building a modern, performant web application.
- TypeScript (Language): Used throughout the entire application for static typing.
  Reason: Enhances code reliability and maintainability by catching errors during development. It provides type-safe code, which is crucial for managing the complex data structures involved in a collaborative editor (e.g., user permissions, document metadata).
- Liveblocks (Real-Time Collaboration Backend): Manages all real-time functionalities.
  Reason: Provides a complete backend-as-a-service for collaborative features, including presence (live cursors, active users), document state synchronization, commenting threads, and notifications. It abstracts the complexity of WebSockets and state management, reducing development time from months to hours.
- Clerk (Authentication Platform): Manages user sign-up, sign-in, and session management.
  Reason: Offers a hassle-free, fully customizable authentication solution that integrates in minutes. It securely handles user profiles, avatars, and session tokens, providing a robust foundation for the application's permission system.
- Lexical (Editor Framework): The underlying framework for the text editor.
  Reason: A powerful and extensible editor framework developed by Meta. It provides a solid foundation for building a rich text editor with features like markdown support, custom plugins, and high performance, which is essential for a collaborative environment.
- Shadcn & Tailwind CSS (UI): Used for building the user interface.
  Reason: Tailwind CSS provides utility-first classes for rapid and completely custom styling. Shadcn is used to provide beautifully designed, accessible, and unstyled components (like modals, buttons, and inputs) that can be easily customized to fit the project's design.
- Sentry (Application Monitoring): Integrated for error tracking and performance analysis.
  Reason: Ensures the application is "enterprise-ready" by providing tools to monitor bugs, analyze performance bottlenecks, and replay user sessions to debug issues quickly. This is critical for maintaining a high-quality user experience in a live, collaborative application.
- Vercel (Deployment Platform): Used to deploy and host the final application.
  Reason: Offers seamless, Git-based deployment for Next.js applications with automatic builds, global CDN, and easy environment variable management, making it simple to take the project from development to production.

Implementation Details

The project is structured as a Next.js application using the App Router, which organizes the codebase by routes and features. Key directories include:

- **app/**: Contains all the routes and UI pages.
  - app/(root)/page.tsx: The main homepage, which lists all the user's documents.
  - app/documents/[id]/page.tsx: The dynamic route for a specific document. It fetches the document data and renders the main <CollaborativeRoom /> component.
  - app/api/liveblocks-auth/route.ts: A server-side API route that acts as the authentication endpoint for Liveblocks, verifying a user's identity via Clerk before granting them a token to access the real-time services.
- **components/**: Contains all the reusable React components.
  - components/CollaborativeRoom.tsx: The core component that wraps the entire editor experience. It initializes the Liveblocks RoomProvider with the specific document ID, which enables all real-time features.
  - components/editor/Editor.tsx: The main editor component built on Lexical. It is wrapped with Liveblocks' configuration to enable real-time text synchronization.
  - components/ShareModal.tsx and DeleteModal.tsx: Self-contained components built with Shadcn UI that handle the logic for sharing documents and deleting them, respectively.
- **lib/actions/**: Contains all the Next.js Server Actions.
  - lib/actions/room.actions.ts: A file containing server-side functions marked with 'use server'. These functions (createDocument, updateDocument, deleteDocument, updateDocumentAccess) securely interact with the Liveblocks Node.js SDK to perform CRUD operations on documents and manage user permissions.

Implementation Flow (A Collaborative Edit):

1. A user logs in via Clerk and creates or opens a document, navigating to /documents/[id].
2. The <CollaborativeRoom /> component establishes a WebSocket connection to Liveblocks for that specific document's "room."
3. When User A types in their editor, the Liveblocks-Lexical integration captures the text changes (deltas).
4. These deltas are sent to the Liveblocks backend in real-time.
5. Liveblocks broadcasts these deltas to all other users (e.g., User B) currently present in the same room.
6. User B's client receives the deltas and applies them to their local instance of the Lexical editor. User A's live cursor position is also broadcasted and rendered on User B's screen.
7. If a user wants to change the document's title or share it, they interact with the UI, which calls a Next.js Server Action. This action securely runs on the server, validates the user's permission, and calls the Liveblocks API to update the document's metadata or permissions.

Software Quality Attributes

Scalability

The application is architected to handle a large number of concurrent users and a high volume of real-time interactions without performance degradation. This is achieved by offloading the most intensive workloads to specialized, distributed services.

Offloaded Real-Time Backend with Liveblocks: Instead of building a stateful WebSocket backend that would become a bottleneck, the project delegates all real-time communication to Liveblocks. Liveblocks is a managed service with a globally distributed infrastructure designed to handle millions of concurrent connections and synchronize data with low latency. As the transcript states, it's designed to manage "millions of collaborators in real time." This allows the application's own backend to remain stateless and lightweight, focusing only on business logic.

Serverless Deployment with Vercel: The Next.js application is deployed on Vercel, which runs the application logic (Server Actions, API routes) as serverless functions. This means Vercel automatically scales the application's resources horizontally. If traffic spikes, Vercel spins up more instances of the serverless functions to handle the load and scales them down when traffic subsides. This provides elastic scalability for the application layer without any manual server management.

Security

The project implements robust security practices by delegating critical functions to specialized services and adhering to modern web development principles that minimize common vulnerabilities.

Managed Authentication with Clerk: Rather than implementing a custom authentication system, which is prone to security flaws, the project uses Clerk. As stated in the transcript, this provides a "secure authentication Flow" and "hassle-free auth." Clerk securely handles user registration, password hashing and storage, session management, and protection against common attacks like credential stuffing. This delegation to a dedicated security platform ensures that authentication is handled according to industry best practices.

Server-Side Permissions and Logic: All critical operations, such as changing a document's title, deleting a document, or modifying user permissions, are handled by Next.js Server Actions. These actions run securely on the server, not the client. This prevents malicious users from bypassing security checks by manipulating client-side code. For example, a user cannot grant themselves "editor" access because the permission change must be validated by a server action that checks if the current user is the document's creator.

Usability

The application is designed to be highly usable, ensuring that different classes of users can easily and intuitively invoke its functions with a minimal learning curve.

Familiar User Interface and Experience: The project is a "version of Google Docs," and its features are designed to be immediately familiar to users. The transcript notes that the commenting feature "works exactly like the commenting feature in Google Docs." By modeling

its core interactions on a widely-used application, the project ensures that users do not need to learn a new mental model, making it highly intuitive for both novice and expert users.

Fully Responsive and Accessible Design: The UI is built to be "native-like" and "fully responsive," working "flawlessly on any device." This means the layout, editor, and modals adapt seamlessly to different screen sizes, from a large desktop monitor to a small mobile phone. This ensures a consistent and accessible user experience for everyone, regardless of the device they are using, which is a key aspect of modern usability.

Portability

The software is highly portable, designed to run in a variety of environments without requiring modifications to the core codebase.

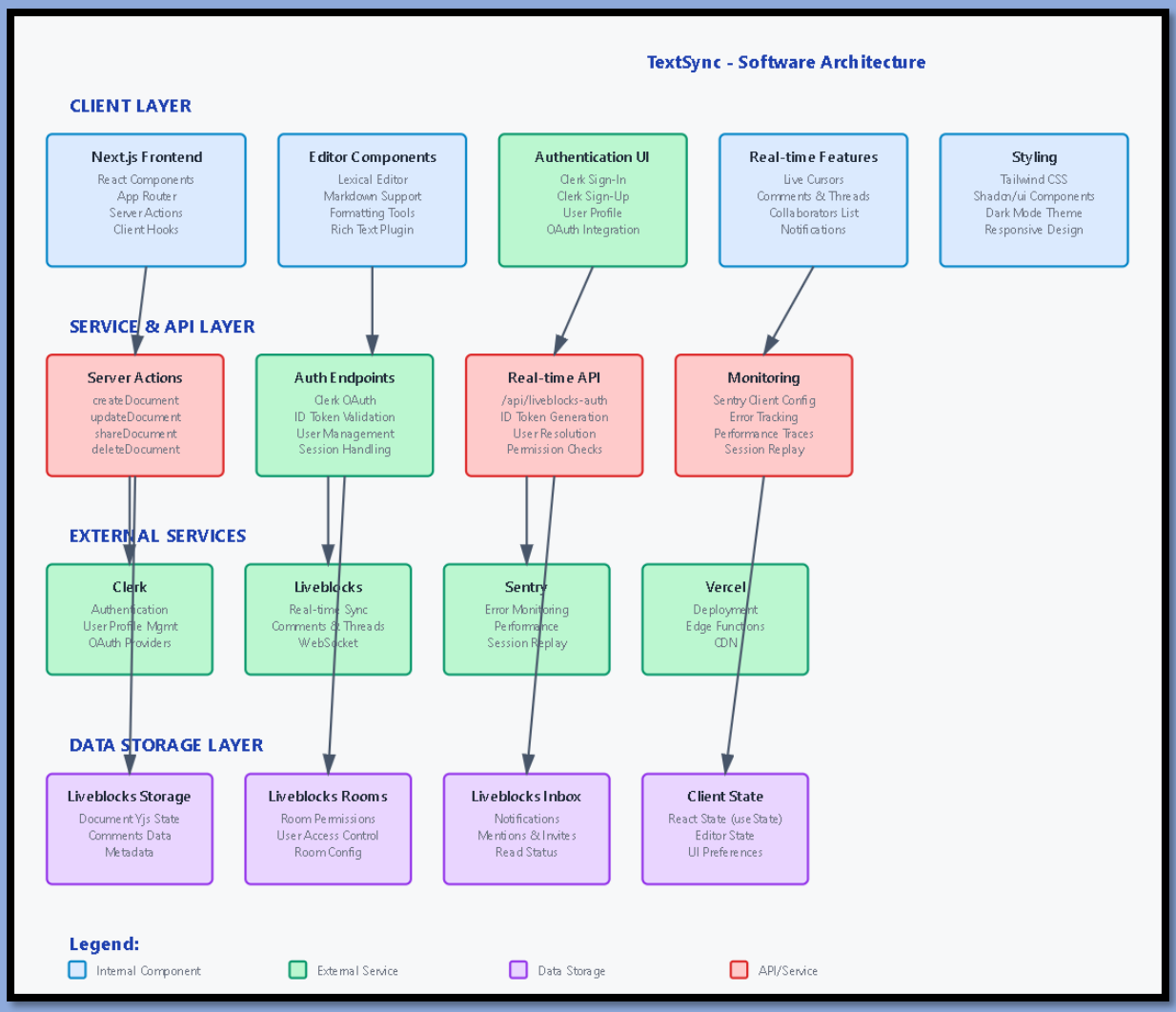
Browser-Based Web Application: As a web application built with Next.js, the project is inherently portable across different user machines and operating systems. It does not require any local installation. Any user on any device (Windows, macOS, Linux, iOS, Android) with a modern web browser can access and use the full functionality of the application, satisfying the requirement of working in different software environments.

Containerized and Managed Deployment: The application is deployed on Vercel, which abstracts away the underlying server infrastructure. Vercel builds the application into a standardized, containerized format that can run on its global edge network. This means the developer does not need to worry about the specific operating system or dependencies of the server environment. The application code can be ported from a local development machine to Vercel's production environment without any changes.

Architecture	Used for	Description
1. Client–Server Architecture	Core communication model	The project follows a classic client–server model: the client (browser) runs the editor UI (React + Next.js), and communicates with the server (Next.js API routes) for fetching/saving documents. The server handles requests, applies authentication, and talks to the database.
2. Real-Time / Broker-Based Architecture	Collaborative editing	TextSync uses a realtime broker (Liveblocks) acting as a middle layer between multiple clients. It manages live connections (WebSockets), synchronizes edits, and ensures all clients see the same content instantly. This is a publish–subscribe pattern, where clients publish changes and the broker broadcasts updates to all others.
3. Layered (N-Tier)	Separation of concerns	The system is logically

Architecture		divided into layers: Presentation (frontend UI), Logic (Next.js APIs + Liveblocks integration), and Data (DB + object storage). Each layer focuses on its own responsibility, improving maintainability and scalability.
4. Shared Data Repository Architecture	Collaboration state	All users share the same document data through a shared data model — stored centrally (DB / Liveblocks room state). This ensures consistency and collaboration on a single version of truth.
5. Microservices / Serverless Architecture (Lightweight)	API routes & scaling	Each API route in Next.js acts like a microservice that can scale independently. Examples: /api/document, /api/auth, /api/snapshot. This improves fault isolation and scalability.

Architecture Diagram



Contribution

Aansh Chandrakant Dubey (B22CS058)

Built the Lexical editor integration and Liveblocks hooks. Implemented editor plugins, presence/cursor sync and editor state utilities using React, TypeScript, Lexical, and Liveblocks.

Banoth Sri Kowshika Raj (B22CS018)

Implemented global styles, Tailwind/PostCSS pipeline, type definitions for the UI, ESLint rules and Sentry client config. Focused on responsive UI, design tokens and consistent component styling.

Mukund Gupta (B22CS086)

Implemented API routes, middleware, server instrumentation and project configuration. Managed TypeScript/Next.js configs, package/deployment settings, and coordinated final merges and deployment readiness.