PROJECT DOCUMENTATION

NOTION – REAL-TIME COLLABORATION TOOL

ABSTRACT :

This project presents the development of a Notion-real-time collaborative editor, designed to enable multiple users to create, edit, and view shared documents simultaneously.built using React for the user interface, Tailwind CSS for responsive styling, Express.js as the backend server, WebSocket for real-time communication, and MongoDB for persistent data storage.

The application addresses the increasing demand for seamless, cloud-based collaborative environments by allowing live synchronization of document content across multiple clients. When one user makes changes, those updates are instantly broadcast to all other connected clients, ensuring consistency and minimizing latency.

The backend manages WebSocket connections, handles broadcasting updates, and stores the document in MongoDB. The frontend is responsible for rendering the content, capturing user input, and updating the UI based on incoming messages. Together, these components provide a smooth, scalable, and intuitive real-time editing experience.

This project demonstrates practical applications of real-time technologies and serves as a foundational prototype for building collaborative platforms similar to Google Docs, and other productivity tools.

TABLE OF CONTENTS :

INTRODUCTION

Notion is a real-time collaborative editor inspired by tools like Google Docs. This application allows multiple users to simultaneously edit and view changes to a shared document in real time. Built using React, Tailwind CSS, WebSocket, Express, and MongoDB, it showcases how to combine modern web technologies to create a smooth, collaborative editing experience.

The goal of this project is to demonstrate how real-time communication, live state synchronization, and document persistence can be achieved using open-source technologies in a full-stack application.

KEY FEATURES :

1. **Authorization and authentication**

- Signup and Login for authenticating users -Email, passwords by using JWT.
- Sharing and editing and viewing permissions-only authorized users can access and modify the Notion pages.

2.Editing Documents

- Create ,edit,and delete documents.
- Text support like bold,italic by using TipTap editor.
- Creating block based document like paragraphs ,headings etc.

3.Real Time Collaboration

- Real-time editing via WebSockets.
- Allowing multiple users to edit the document.
- Web Sockets - persistent connections between a client (like a browser) and a server.

4.Persistance data: All the changes made to the data can be stored and saved in the database.

Used mongoDB for storing databases.

5.Search

- Search across pages or entire document
- Easy categorization.

6.Theming

- Toggling the changes from light and dark modes.

7.Comments

- Text selections (e.g. highlight a sentence → add comment).
- User selects text in the editor.

- User types a comment → it's saved and the selection is highlighted.
- Other users see the comment in real time.

8.User presence indicators

- See which users are online in your app.
- Know who is currently viewing or editing a specific document.

OBJECTIVES:

- **Supporting real time collaboration:** Multiple users can edit the same document simultaneously using web sockets.

- **Authentication and Authorization:** Only allowing authorized users can edit and view the document.

- **Persistance data:** All the changes made to the data can be stored and saved in the database.

- Implement rich-text formatting using WYSIWYG -TipTap Editor.

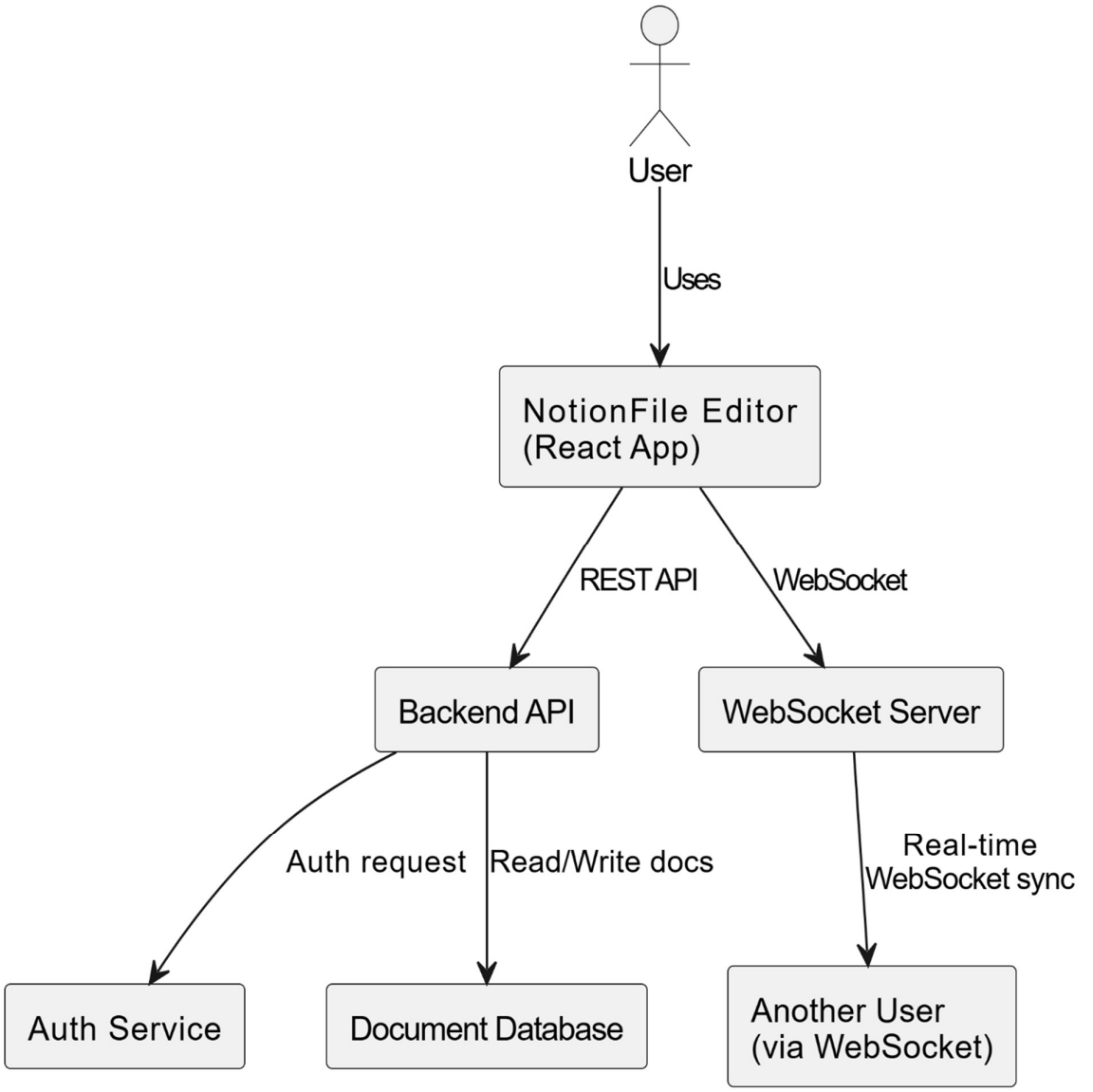
- Enable WebSocket-based live syncing.

TECH STACK :

Frontend	**React.js,TipTap Editor**
Backend	**Node.js,Express.js**
Real-Time Sync	**Socket.IO(WebSockets)**
Database	**MongoDB**
Authentication	**JWT**

SYSTEM DESIGN:

HIGH LEVEL ARCHITECTURE DIAGRAM



• This architecture represents a NotionFile-Real -time - collaborative app.

• User interacts with the NotionFile Editor built using React in the browser.

• The Editor connects to the Backend API via REST API for document operations and authentication.

• It also connects to the WebSocket Server to enable real-time collaboration.

• The Backend API communicates with the Auth Service to verify user identity.

• It stores and retrieves documents from the Document Database. • The WebSocket Server syncs changes between users in real-time for collaborative editing.

SYSTEM LEVEL ARCHITECTURE

• The diagram represents a real-time collaborative Notion-like editor using WebSockets and REST APIs.

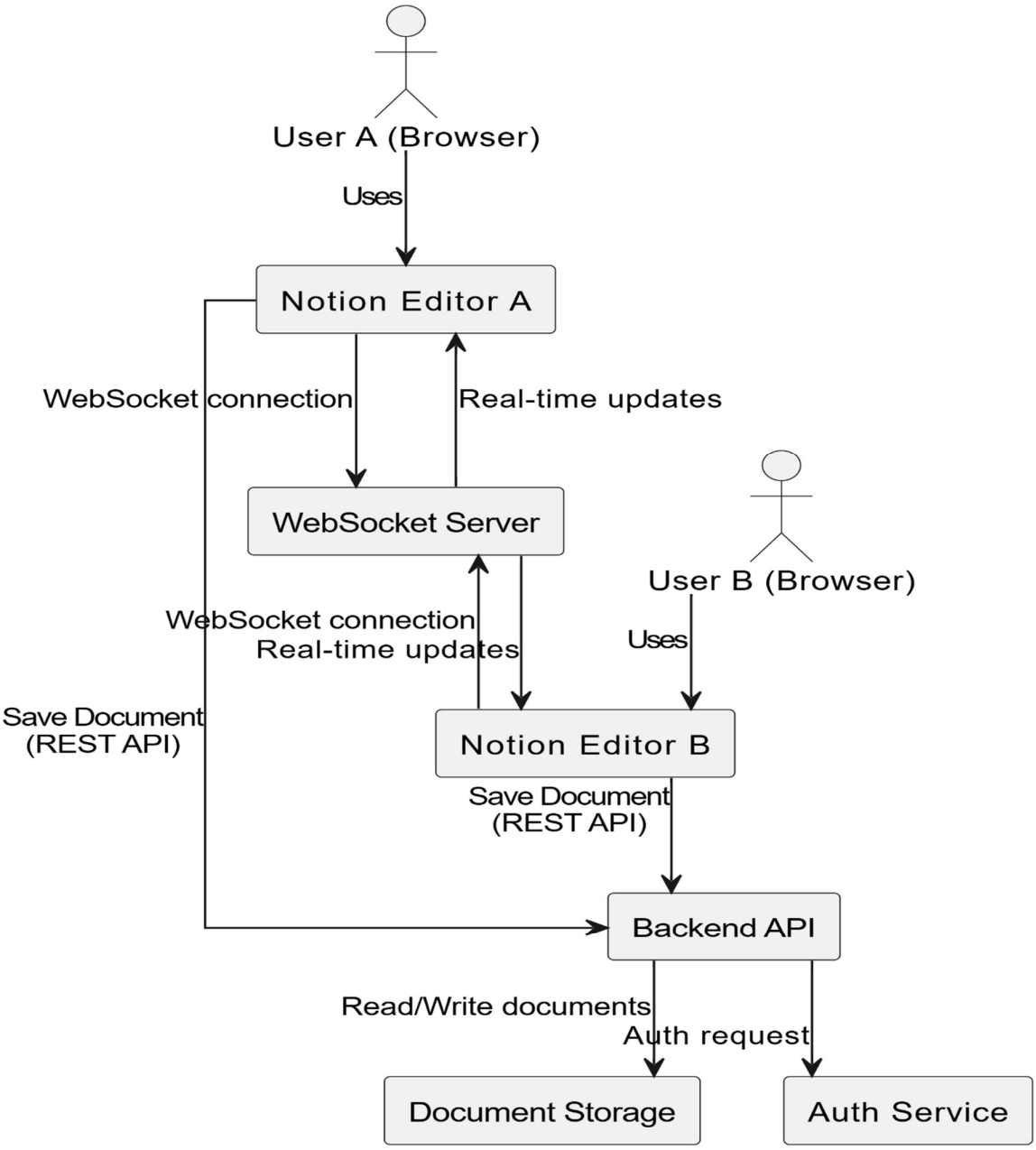
• User A and User B access the application from their browsers.

• User interacts with their own instance of the Notion Editor.

• The editors connect to a WebSocket server to exchange real-time updates.

• Document saving is handled via REST API calls to a backend API.

• The backend API communicates with document storage and an authentication service.


User A (Browser)
Uses
Notion Editor A
WebSocket connection
Real-time updates
WebSocket Server
User B (Browser)
WebSocket connection
Real-time updates
Uses
Save Document
(REST API)
Notion Editor B
Save Document
(REST API)
Backend API
Read/Write documents
Auth request
Document Storage
Auth Service

IMPLEMENTATION :

Editor:

- Implemented using TipTap Editor for rich text and collaborative.
- Different text styles like bold,italic and underline are used.

Collaboration:

- Real-time collaboration is achieved using Socket.IO, which allows bi-directional communication between the server and multiple clients.
- Changes made by any user are immediately broadcasted to all other users editing the same document via WebSocket channels.

Backend:

- Built using Express.js, the backend provides a RESTful API for managing users, documents, and metadata (CRUD operations).
- Each document has an associated WebSocket editing room, managed dynamically based on active user sessions and document IDs.
- Middleware is used to verify authentication tokens before allowing access to protected API routes or WebSocket events.

Authentication:

- Authentication in this app is implemented using JWT (JSON Web Token) to secure both REST API requests and WebSocket connections.
- Auth token stored in localStorage for protected routes.

Persistence:

- MongoDB is used as the primary database to store:
- User profiles and authentication metadata.
- Document metadata (title, owner, collaborators, timestamps).
- Access permissions and collaboration history.

ALTERNATIVE TECH STACK

1.WebSockets- Centralized, easier to manage, improving collaboration speed and accuracy. but not P2P.

Alternative:

WebRTCs - Real-Time Collaboration

- Enables peer-to-peer communication directly between browsers,reducing latency and server load.

2. React (Notion Editor UI)

- React is component-based model fits well with block-based editors like Notion.
- **Alternative:**
- Angular: Full-featured framework, but heavier.

3.Document Storage (MongoDB, Firestore)

- NoSQL databases are schema-flexible — good for storing documents.
- **Alternative:**
- PostgreSQL : Structured + semi-structured hybrid

CHALLENGES:

1.WebSocket Connections:

- Intermittent disconnects: Network instability causes clients to disconnect and reconnect frequently.
- Preventing unauthorized access over WebSockets.

2.Storage in MongoDB:

- Schema Design for Real-Time Data - Modeling real-time collaboration data (e.g., comments, presence, document edits) can be complex.
- Handling simultaneous writes to documents.

3.Authentication:

- Ensure JWT is sent over secure WebSocket to prevent interception.
- Verify token server-side before accepting connection.

RESULTS:

1.Real-Time Collaborative Editor

- Multiple users can edit the same document at the same time.
- Changes sync in real time using WebSockets (e.g., Socket.IO).

2. Rich Text Editing

- Using TipTap editor for editing the text.
- Rich text features: bold, italic, links, code, etc.

3. User Presence Indicators

- Show **who is currently viewing or editing** the document.

4.JWT-Based Authentication for Secure Access

- Uses JSON Web Tokens to securely authenticate users without maintaining session state.
- Tokens are validated on every WebSocket connection and API call for secure access control.

5.MongoDB Storage for Documents, Users, Comments, and Presence

- All application data — including documents, user profiles, comments, and online presence — is stored in MongoDB collections.

6.Access Control and document sharing

- Implements role-based permissions (e.g., owner, editor, viewer) to control who can view or edit each document.
- Ensures only authorized users can access or modify content in real time.

CONCLUSION:

This Notion project highlights how real-time collaboration can transform documentation workflows. By using Notion's collaborative features—such as shared pages, inline comments —we've created a dynamic, centralized space where contributors can work together seamlessly.
Web Sockets enable instant, two-way updates across all users, improving collaboration speed and accuracy.
JWT ensures secure, stateless authentication for all user sessions and socket connections.
MongoDB provides a flexible, scalable NoSQL backend suited for nested, document-style data.
Real-time updates keep everyone synced without page refreshes or delays. Authentication and access control are handled efficiently with token-based logic. The architecture supports fast data retrieval, live editing, and multi-user concurrency.
The system is scalable and more secure.

Endpoints Description:

Runs locally at : http://localhost:3000/-Backend

Runs locally at: http://localhost:5173/-Frontend

http://localhost:3000/{endpoint}

Authentication

1. **POST /api/auth/signup**
   Registers a new user with username, email, and password. Returns a JWT token on success.
2. **POST /api/auth/login**
   Authenticates a user with email and password. Returns a JWT token and user info.
3. **GET /api/protected**
   Accesses a protected route available only to logged-in users. Requires a valid JWT token.

Documents

4. **GET/POST /api/documents**
   Fetches all user documents (GET) or creates a new one (POST). Requires authentication.
5. **GET /api/documents/:documentId/history**
   Returns the edit history of a specific document. Shows past changes or versions.
6. **GET /api/:documentId**
   Retrieves a document by ID. Works for both owned and shared documents.

SharedDocuments

7. **POST/api/shares**
   Shares a document with another user (POST)
8. **GET/api/shares-users**
   Get the lists of shared docs (GET).

Comments Routes

9. **GET/POST /api/comments**
   Fetches comments for a document (GET) or adds a new comment (POST).
10. **POST /-create comment-post**
    (Typo: should be /api/comments) Adds a new comment to a document.
11. **DELETE /api/:id**
    Deletes a comment by its ID. Only accessible to the comment owner or document owner.

Document Routes

1. **POST https://localhost:3000/-create-document**
   Creates a new document with provided title/content. Requires user authentication.
2. **GET /:id + auth, checkPermission("view")**
   Retrieves a specific document by ID if the user has view access. Checks both ownership and share permission.
3. **GET /getDocuments**
   Returns all documents created by the authenticated user. Requires login.
4. **PUT /:id + auth, checkPermission("edit")**
   Updates a document if the user has edit permissions. Requires valid JWT and permission check.
5. **DELETE /:id**
   Deletes a document by its ID. Only the owner can delete it.
6. **GET /user/me**
   Returns the profile info of the currently logged-in user. Requires JWT token.
7. **GET /shared/me**
   Lists all documents shared with the current user. Requires authentication.

Protected Routes

1. **GET /admin/data + verifyToken, authRole(['admin'])**
Fetches admin-specific data. Accessible only to authenticated users with the admin role.

2.**POST/PUT /edit/doc + verifyToken, authRole(['admin', 'editor'])**
Allows editing of documents. Only authenticated users with admin or editor roles can access this route.

Document History Routes

1. POST / restore/:historyId
Adds a new entry to the document's edit history. Typically used to log changes when a document is updated.

2.GET /
Fetches the version history or edit timeline of a specific document. Helps track changes made over time by collaborators.