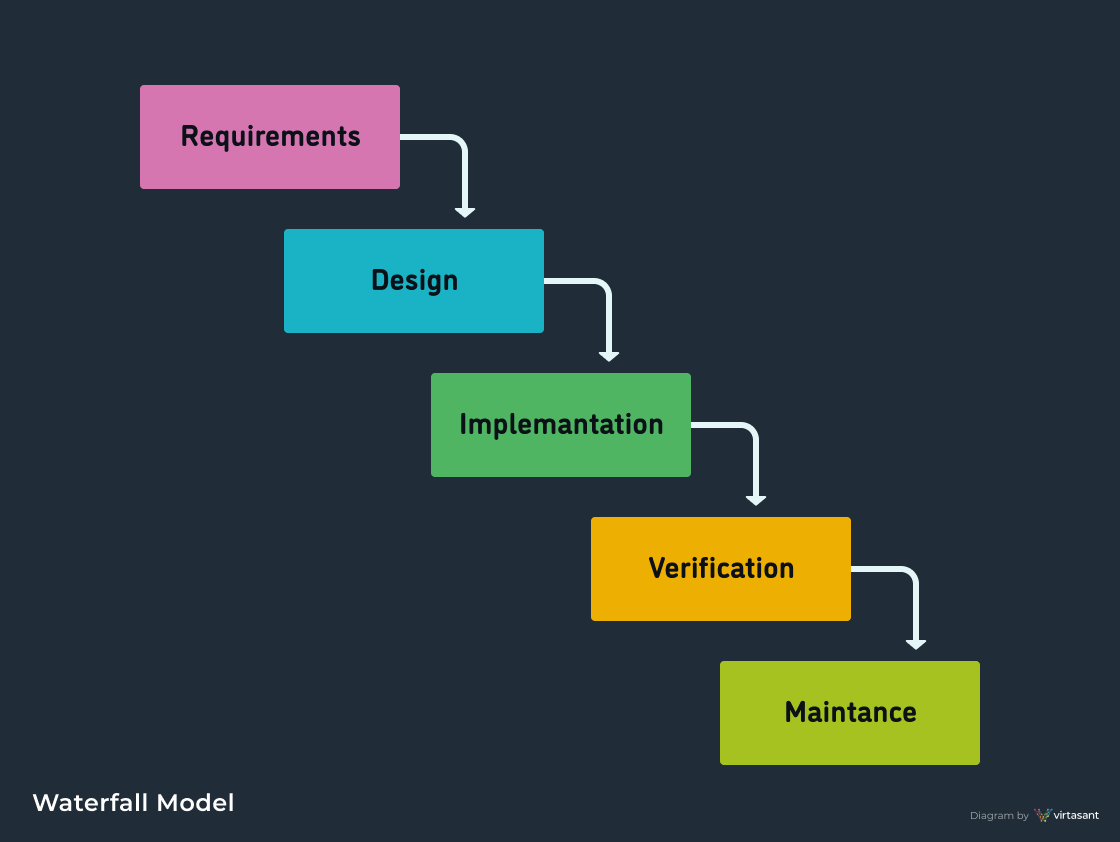
**System Architecture Document**

**1. Project Overview**

This document outlines the system architecture for the real-time collaborative documentation editor.



**Objective:** To build a secure, scalable, and performant web-based platform that allows multiple users to create, edit, and comment on documents simultaneously. The system prioritizes a seamless user experience with instant updates and robust data integrity.

**2. Tech Stack**

|  |  |  |
| --- | --- | --- |
| Layer | Technology | Details |
| Frontend | React.js (with Vite) | A modern, component-based library for building the user interface. |
| State Management | Context API | For managing global application state like user authentication. |
| Rich Text Editor | Tiptap | An extensible, headless editor framework that supports collaborative features. |
| Backend Services | Firebase | A comprehensive suite of tools for backend functionality. |
|  | Firebase Auth | Manages user sign-up, login, and secure authentication. |
|  | Firebase Firestore | A NoSQL database for storing all application data (documents, comments, etc.). |
| Real-Time Comm. | WebRTC + Socket.IO | A hybrid model using Socket.IO for reliable signaling and WebRTC for low-latency peer-to-peer data exchange. |
| Deployment | (Not yet deployed) | The architecture is designed for containerized deployment (e.g., using Docker). |

**3. Summary of Documentation**

This system provides a complete solution for real-time collaborative editing. Key aspects include:

* Secure User Management: Users can register and log in securely via Firebase Authentication.
* Data Persistence: All documents, comments, and user data are managed and stored in Firebase Firestore.
* Live Updates: A hybrid real-time layer using WebRTC and Socket.IO ensures that all collaborators see changes instantly.
* Modern Architecture: The system is built with a modern frontend stack and a scalable, decoupled backend design.

A diagram of a computer network

AI-generated content may be incorrect.

**Components:**

* Component diagram   
  A diagram of a software company

  AI-generated content may be incorrect.

**Client (Frontend)**

* React App: The user interface is a single-page application built with React and served via the Vite build tool for a fast development experience.
* Tiptap Editor Integration: The core of the editing experience, configured to handle real-time updates from collaborators.
* Auth UI: Provides login and registration forms that interface directly with Firebase Authentication.
* Real-time Sync Logic: Contains the client-side logic for connecting to the Socket.IO server and establishing WebRTC peer connections.

**Backend Services**

* Firebase Auth: Serves as the primary user authentication service.
* Firebase Firestore: The database for all documents, user profiles, and comments. It provides its own real-time update capabilities which serve as a backup and persistence layer.
* Socket.IO Server: A Node.js server that manages WebSocket connections. Its primary role is to act as a signaling server for WebRTC.

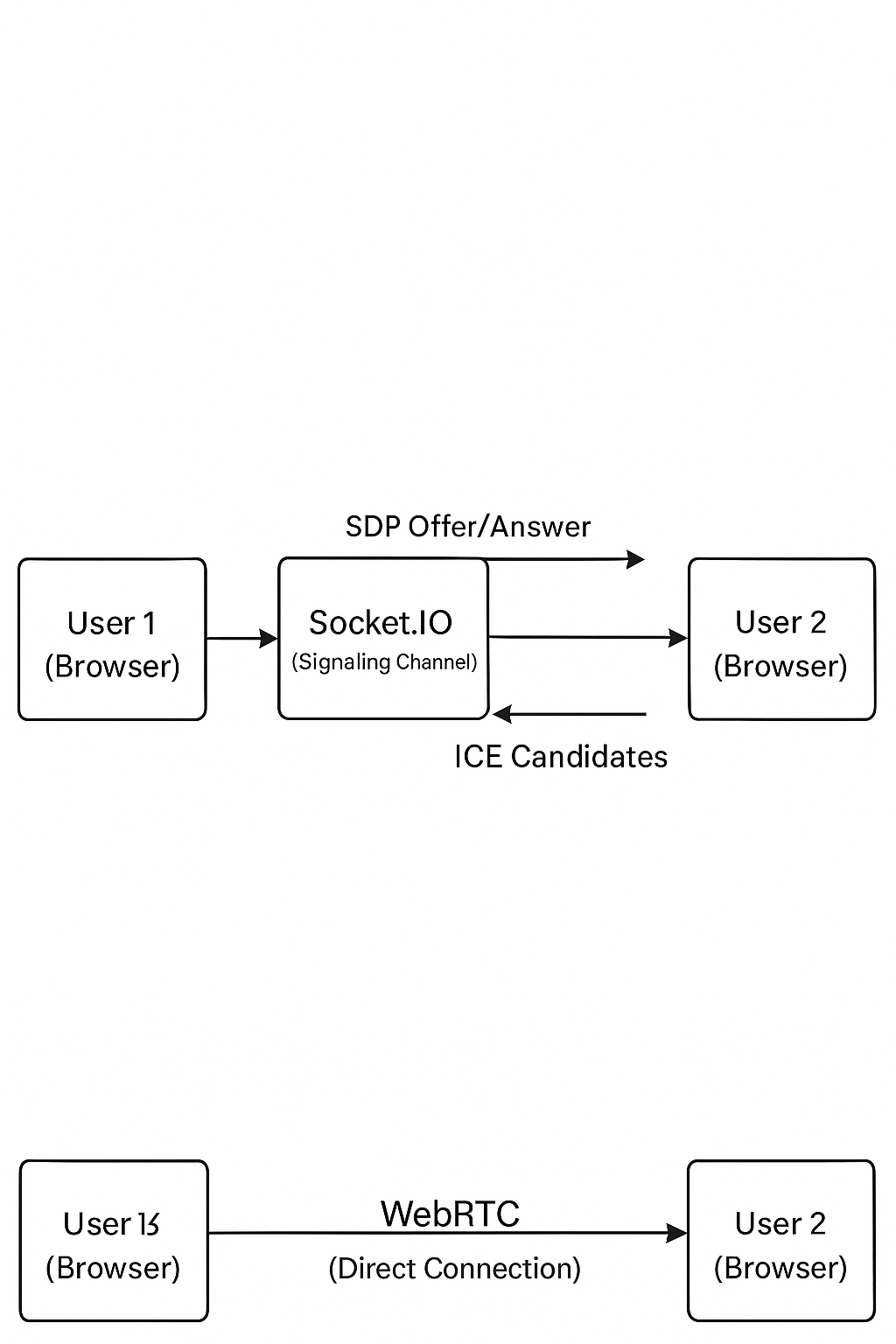
**Real-Time Layer**

* Signaling Server (Socket.IO): Manages the initial handshake for WebRTC setup. It helps clients exchange connection information (like SDP and ICE candidates).
* Direct P2P Channels (WebRTC): Once the connection is established, WebRTC handles the direct, low-latency synchronization of ephemeral data (like cursor positions) between peers.
* Persistence Layer (Firebase): Critical updates (like content changes) are sent to Firebase to ensure they are saved permanently and can be retrieved by users joining the session later.

**4. Real-Time Collaboration Overview**

The real-time layer is the core of the application and uses a combination of technologies to function effectively:

* Socket.IO: Initializes the connection between users and the server. It acts as the signaling server, allowing peers to find each other and negotiate a direct connection. It also serves as a reliable fallback channel for critical data.
* WebRTC: Once the initial handshake is complete, WebRTC establishes a direct peer-to-peer (P2P) connection between collaborators. This channel is used for high-frequency updates like cursor movements, minimizing server load and providing near-zero latency.
* Firebase Firestore: While WebRTC handles the immediate visual sync, Firestore acts as the single source of truth. All changes are ultimately persisted to the database to ensure data integrity and to act as a backup layer.
* Tiptap Editor: The editor itself is built to handle collaborative changes, applying updates from other users without disrupting the local user's experience.



A diagram with colorful text

AI-generated content may be incorrect.

**Justification for my approach**

**1. What is SDP Offer ?**

**SDP (Session Description Protocol)** is used in **WebRTC** to describe media capabilities (like codec, IP, ports, etc.) between two peers. It works like this:

**SDP Offer/Answer Model:**

* **Peer A** creates an **SDP Offer**:  
  It contains information about what kinds of media Peer A wants to send/receive.
* **Peer B** receives the offer and responds with an **SDP Answer**, saying what it agrees to.

This negotiation helps both peers agree on:

* Media formats (video/audio codec)
* IP address & port to connect
* Whether to send/receive audio/video/data

**Example**:

{

"type": "offer",

"sdp": "v=0\no=- 4611736600794295656 2 IN IP4 127.0.0.1..."

}

**2. What are ICE Candidates?**

**ICE (Interactive Connectivity Establishment)** candidates are:

* Different **network paths** (IP:port combinations) a peer can use to connect.
* These include:
  + Host (local IP)
  + STUN (public IP via NAT traversal)
  + TURN (relay server fallback)

Peers exchange ICE candidates after the SDP negotiation to find the **best network path** to establish a **direct P2P** connection.

**3. Why use Socket.IO for WebRTC?**

WebRTC does **not** provide its own signaling mechanism. That means it **needs a signaling channel** to:

* Exchange SDP offers and answers
* Exchange ICE candidates

You use **Socket.IO** as that **signaling layer**. It sends the necessary WebRTC negotiation messages between clients via the server **before** the direct P2P connection is established.

**So, why not just use WebRTC alone?**

Because:

* WebRTC handles **media/data transmission**, but **not signaling**.
* You still need something (like **Socket.IO**) to handle signaling.

| **Purpose** | **WebRTC** | **Socket.IO** |
| --- | --- | --- |
| Send audio/video/data | yes | No |
| Exchange connection info (SDP/ICE) | No | Yes |
| Peer-to-peer communication | Yes | No |
| Initial negotiation setup | No | Yes |

**Summary**

* **WebRTC** = actual real-time connection.
* **Socket.IO** = helper tool for setting up the connection (signaling).
* **SDP Offer/Answer** = describe connection capabilities.
* **ICE Candidates** = potential network paths for peer connection.

**5. Use Cases**

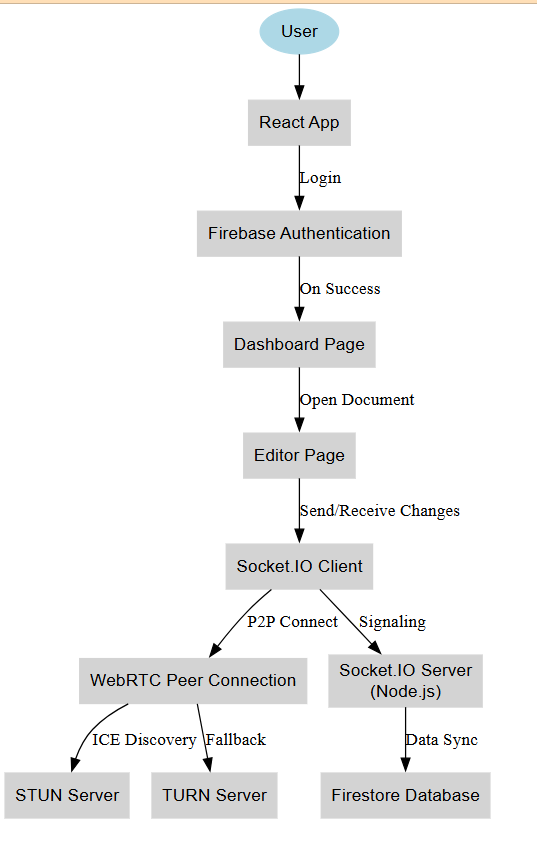
A diagram of a user flow

AI-generated content may be incorrect.

**6. Feature List**

* User Authentication (Register/Login via Firebase)
* Create, Edit, and Delete Documents
* Real-time Syncing of Edits between multiple users
* Commenting on specific parts of a document
* Document Listing Dashboard for users to manage their files
* Role-Based Access Control (Future feature)
* Document Versioning & History (Future feature)
* User Notifications (Future feature)

**7. High-Level Architecture (System Design)**

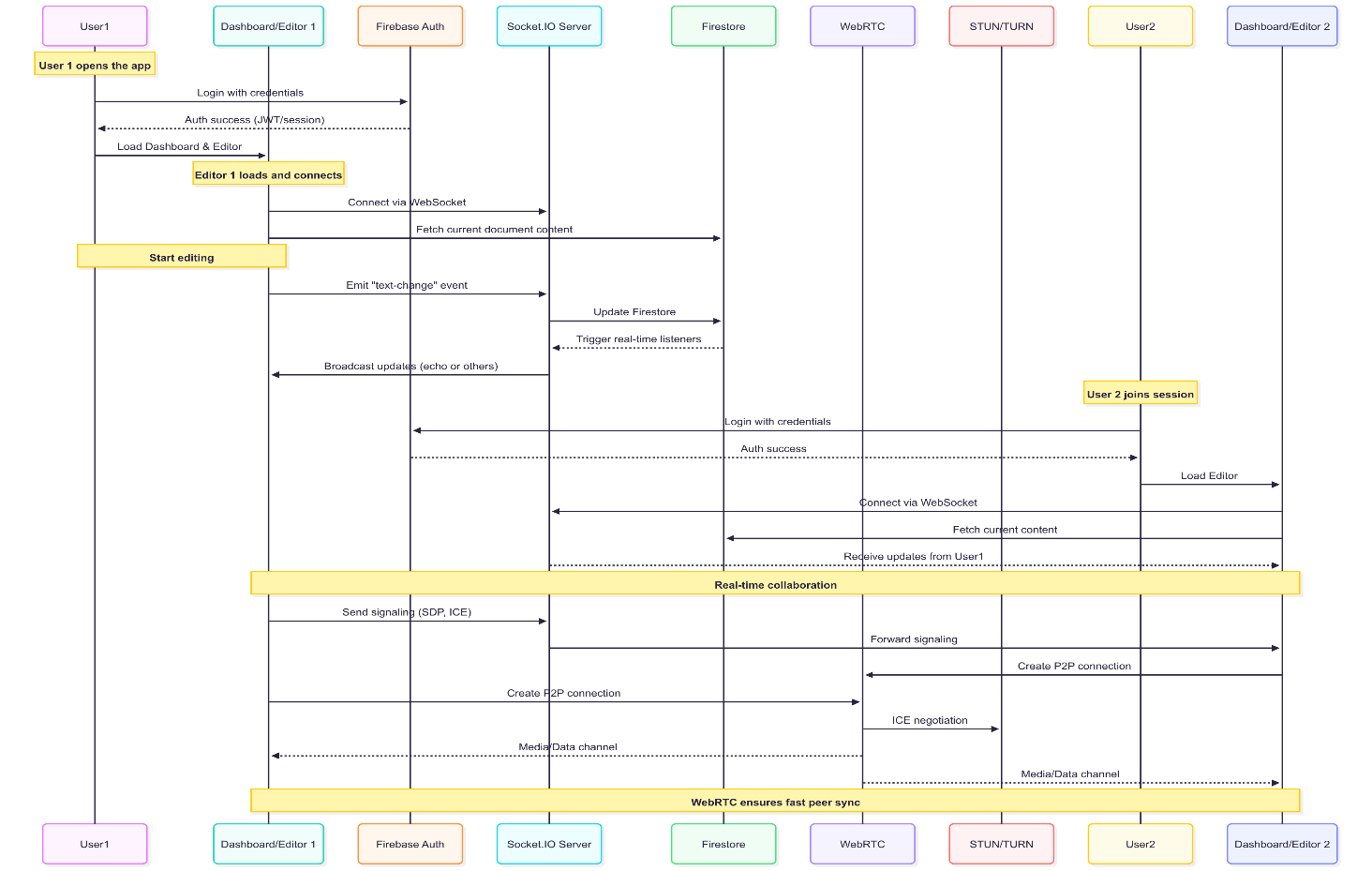


**This diagram is for multiple user**

A diagram of a software system

AI-generated content may be incorrect.

**Sequence diagram:**



**8. Development Lifecycle**

**SDLC (Software Development Lifecycle):**

The project follows an agile SDLC methodology:

Requirement Gathering → Design → Implementation → Testing → Deployment → Maintenance

**HDLC (High-Level Design Cycle):**

The design process for this system follows these key stages:

Architecture Planning → Component & Technology Selection → Integration Strategy → Real-time Setup & Logic → UI/UX Polishing

9.**Mind map**

Backend

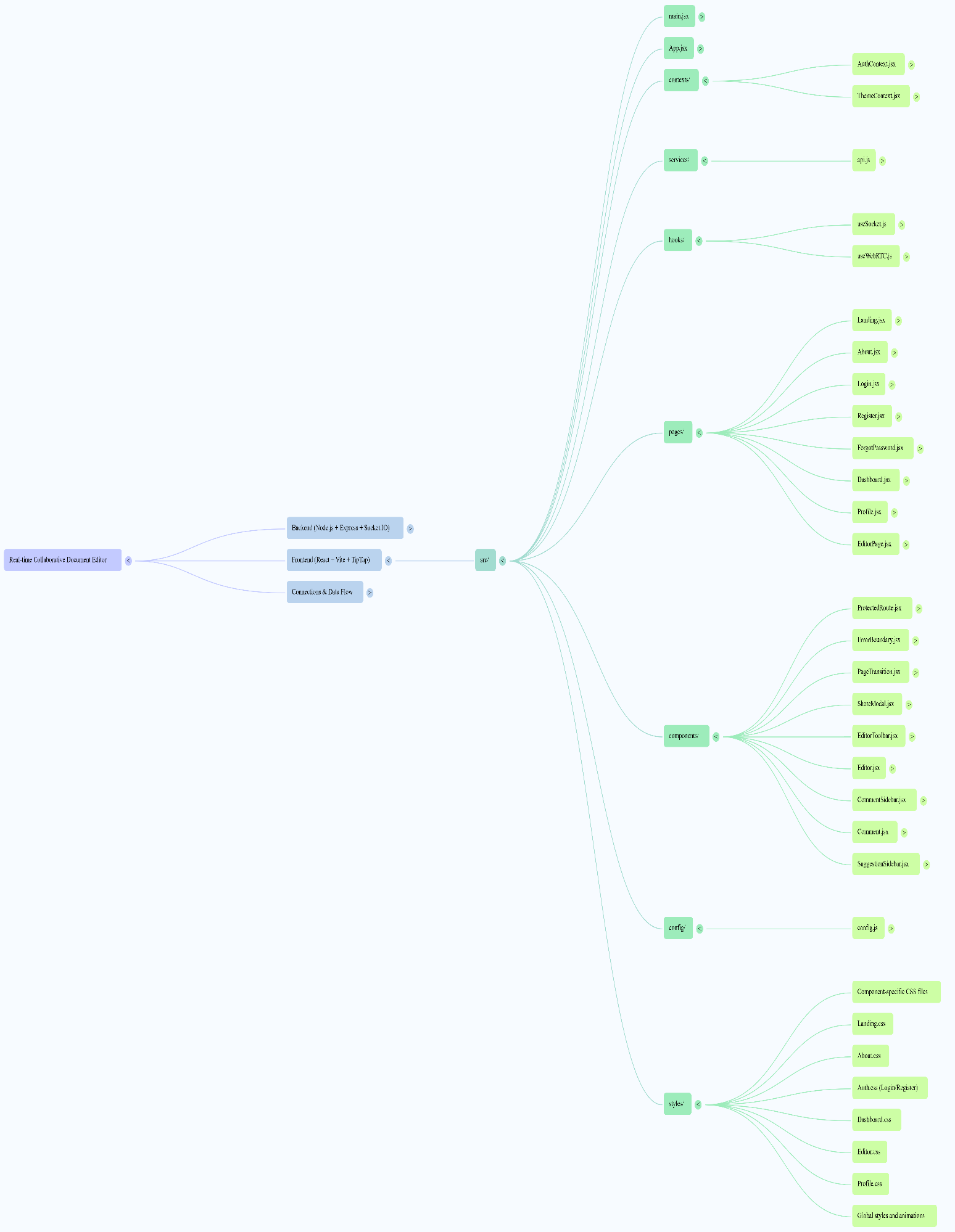
A screenshot of a diagram

AI-generated content may be incorrect.

**Connection and Dataflow**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Frontend:-  
**

References:

[Power of Notion. Overview and 🎁 bonus template for a design project | by Markiyan Vaskiv | OffGrid Design Community | Medium](https://medium.com/offgrid-design-community/power-of-notion-overview-and-bonus-template-for-a-design-project-bcf200a0878c)

<https://medium.com/@shreyasmanolkar123/building-notion-clone-part-1-planning-the-architecture-f50342e58019>

<https://tiptap.dev/docs/ui-components/components/overview>

<https://dev.to/abdelraman_ahmed_e83db59f/building-a-richtext-editor-with-tiptap-in-react-with-mentions-2bdp>

https://vikramthyagarajan.medium.com/how-to-build-a-notion-like-text-editor-in-react-and-tiptap-7f394c36ed9d