Technical specifications of collaborative code editor

Bailey Anthony Scanlan	23316363
Hoang Xuan Hai Mai	22115838

1. Introduction

1.1 Overview

This is a cross platform desktop collaborative text editor that uses tauri for the desktop component, Django as the central API for authorisation of user accounts, and likely persistence. Real-Time doc synching is done with YJS and is separated from the REST api via the render websocket service. We used React + Vite to provide the UI and used this in conjunction with Tauri to provide a native desktop shell for our code to run in. This code level communicates with the other levels of code via.
At runtime the application is composed of two parts, First the tauri desktop application provides the user interface and interaction layer, this includes the editor view (as a react component), session control, and the local desktop integration that Tauri enables. Secondly, the application depends on a collaboration backend service that communicates real-time updates between clients. This is provided by a WebSocket server which clients connect to on a configured host. The collaboration layer is responsible for distributing edits to all connected peers in the same session and maintaining a consistent shared document state across clients.

1.2 Glossary

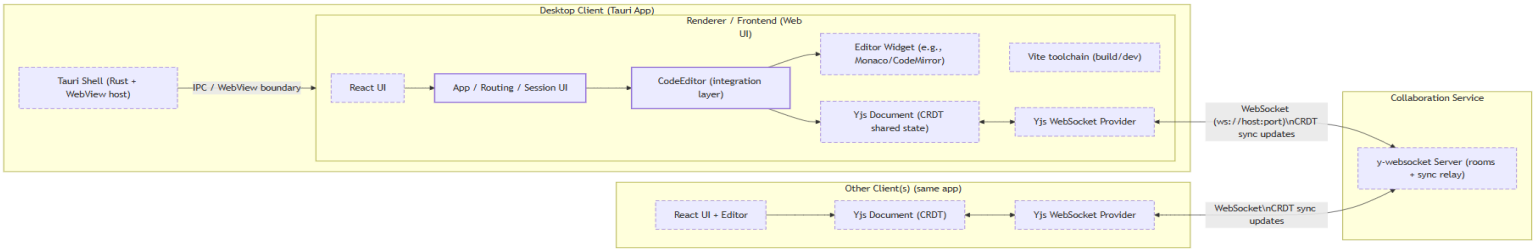
Term	Definition
YJS	Open Source javascript framework for making real-time collaborative applications
CRDT	specialized data structures used in distributed systems that allow multiple replicas to be updated independently and concurrently without coordination or locking

2. System architecture

Distribution of functions across system modules

The system is implemented as a Tauri desktop client connected to a separate real-time collaboration service. Real-time document consistency is provided by Yjs (CRDT) on the client with the y-websocket server acting as a relay for synchronization messages between clients in the same “room”.

2.1 Architecture diagram



2.2 Component descriptions

A) Desktop Client

- Tauri Shell - Provides the native desktop window, application lifecycle, and hosts the WebView that renders the frontend. It forms the security/process boundary between the operating system and the web UI.
- React - Implements the user interface, including session controls and the editor screen.
- Editor integration - The glue code that binds the editor widget to collaborative state.
- Yjs Document - Holds the shared document state using a CRDT, enabling conflict free merging of concurrent edits.
- Websocket Provider - Connects the yjs document to the collaboration server over websockets.

B) Collaboration Service

- Y-websocket server - A standalone server process that manages rooms and broadcasts synchronization updates between connected clients.

C) Persistence

- Document state is held in memory across connected clients.

2.3 Data flows

1) Application startup

1. The user launches the desktop app.
2. Tauri starts and loads the react renderer in a webview.
3. The frontend initializes the editor view and waits for a room selection.

2) Joining a collaborative session

1. The user selects or enters a room identifier.
2. The frontend creates a Yjs document instance.
3. The frontend creates a Websocket provider targeting the collaborative server and joins the specified room.
4. Provider performs initial synchronization with the server and any connected peers.

3) Editing and synchronization

1. User types in the editor widget.
2. The editor integration converts UI edits into updates on the Yjs document.
3. Yjs produces CRDT updates; the provider sends them over WebSocket to the server.
4. The server relays updates to all other clients in the same room.
5. Other clients apply updates to their Yjs documents, and their editor widgets re-render to reflect the new state.

4) Conflict handling

- If multiple users edit simultaneously, Yjs merges changes deterministically via CRDT rules, avoiding manual conflict resolution in typical cases.

5) Disconnect / Reconnect

- If a client disconnects, the provider stops receiving updates. On reconnection, it re-syncs the Yjs state with the room.

2.4 Third-party dependencies

Dependency	Where used	Why it's used
Tauri	Desktop client shell	Native packaging + lightweight WebView hosts; enables desktop delivery with a web UI.
React	Renderer/UI	Component-based UI, state management patterns, fast iteration.
Vite	Build/dev tooling	Fast dev server and bundling for the frontend during development.
Editor widget	Editor UI	Provides a production grade code editing experience.
Yjs	Collaboration core	CRDT based shared state for real time editing

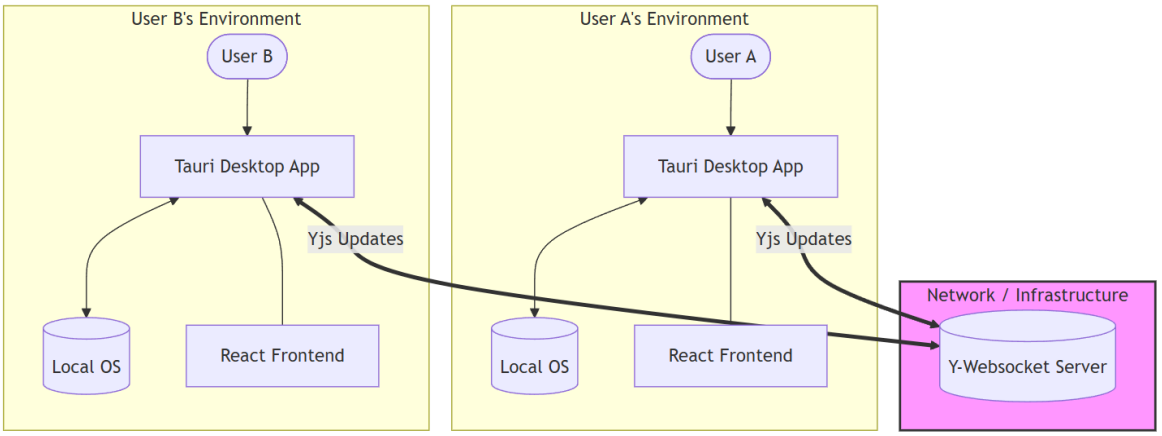
		experience.
y-websocket	Collaboration service	Off the shelf Websocket relay for Yjs rooms.

3. High-Level Design

This section describes the as built high-level design of the system as demonstrated. The system is a Tauri desktop application that enables real-time collaborative editing by synchronizing document state through a Websocket based collaborative server. Collaboration is implemented using Yjs (CRDT) on the client and the y-websocket as the server relay. The design separates concerns into: (i) desktop delivery and UI, (ii) collaborative document state, and (iii) network transport for synchronization.

3.1 System Context model

The system operates in a desktop environment and interacts with a collaborative server over the network. Users run one or more instances of the desktop app, which join a shared room to co-edit the same document.

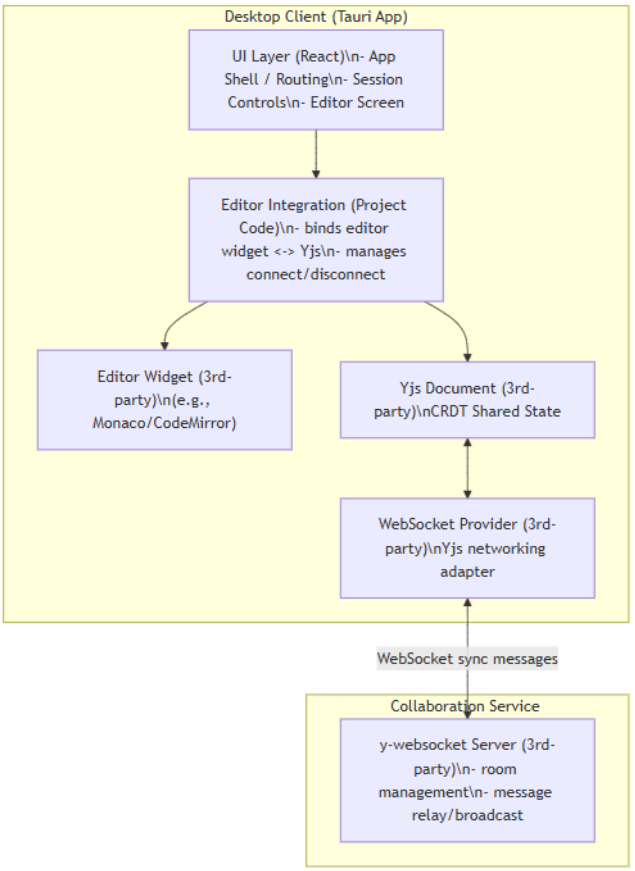


Environment assumptions

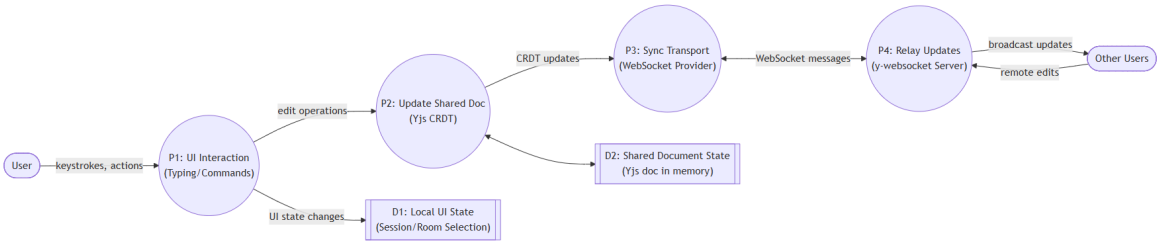
- The collaboration server is reachable at a configured host.
- Multiple clients connect to the same server and room to share state.
- If persistence is not implemented, document state exists across connected clients but is not guaranteed after server restart.

3.2 Component relationship model

At a logical level, the client contains a UI layer and a collaboration layer. The collaboration layer encapsulates CRDT state and it's websocket provider. The server is an external process responsible for routing/broadcasting updates by room.



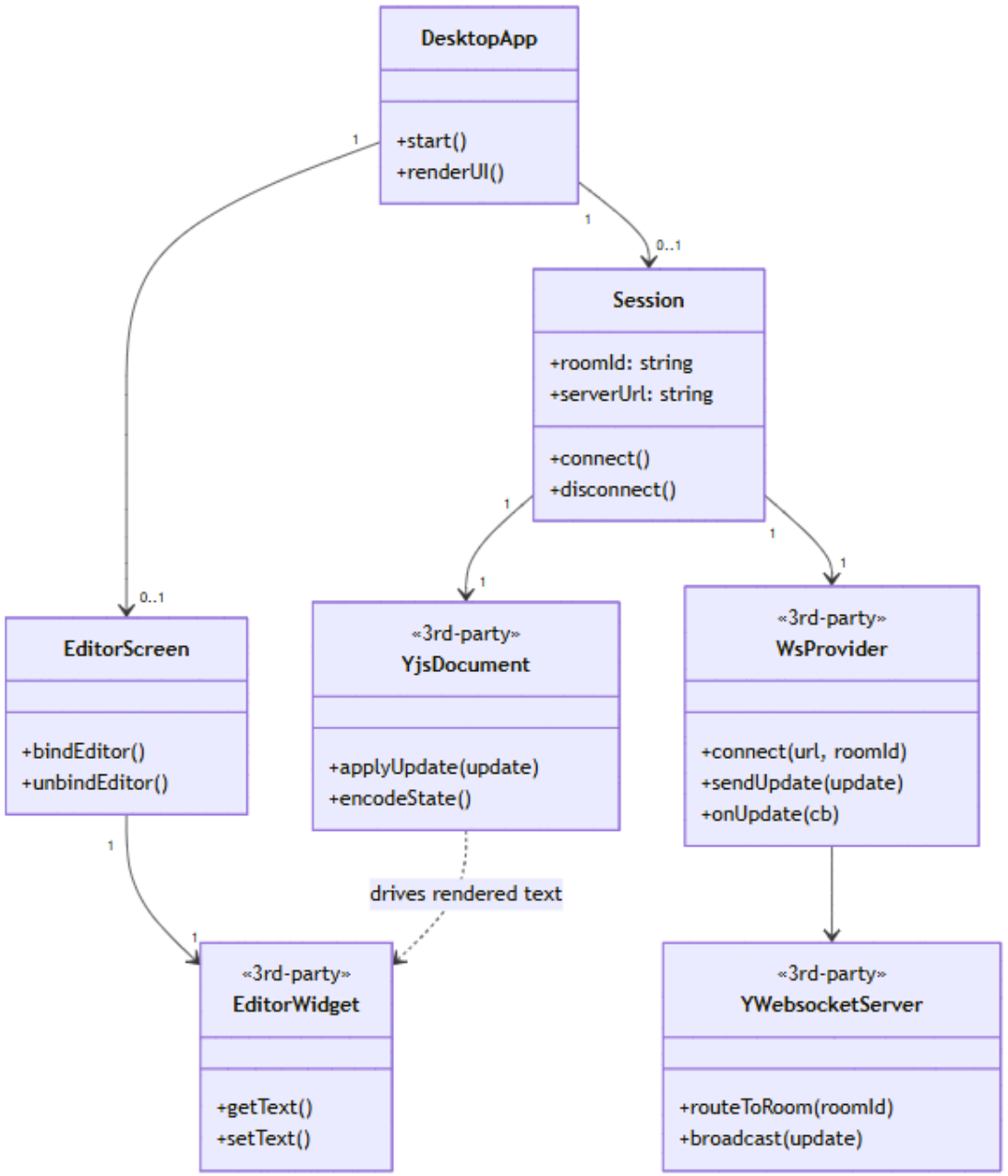
3.3 Data flow diagram



- The shared document is the in memory Yjs document.
- The server relays messages, it does not need to interpret documents beyond room routing.

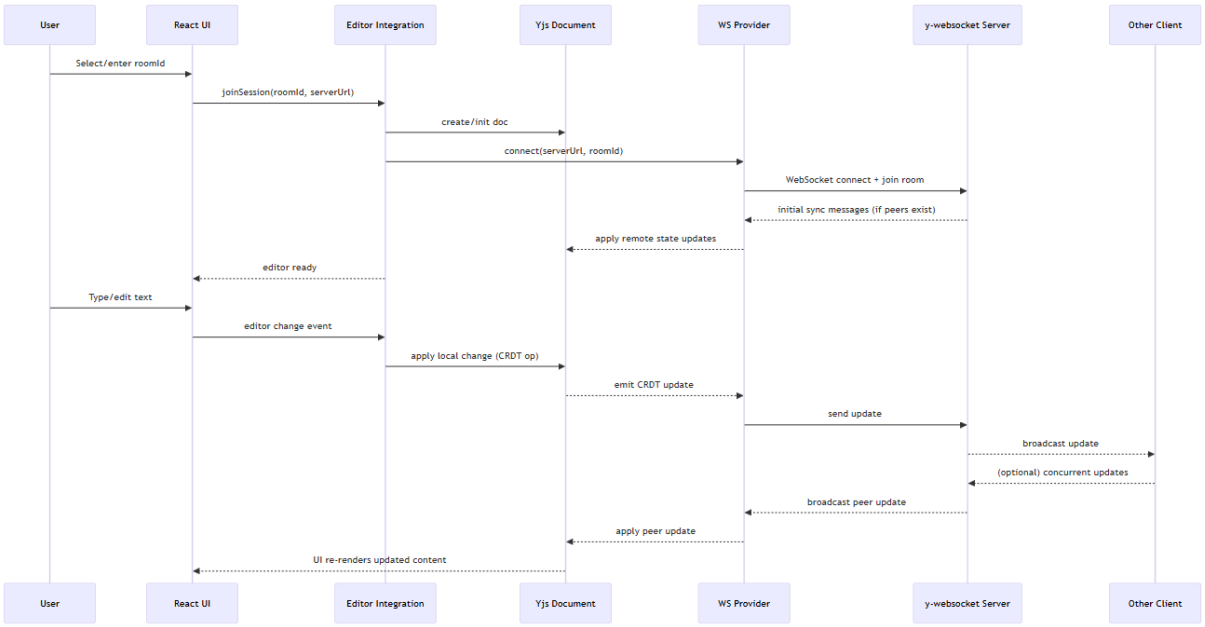
3.4 Object / Domain model

This is a logical model intended to document relationships in the system.



3.5 Key interaction sequence

This captures runtime behaviour, a client joins a room, syncs an initial state, then propagates edits to peers through the server.



3.6 Design constraints and boundaries

- Process boundary - Tauri hosts react in a WebView, collaboration networking is managed in the frontend runtime and external server process.
- Network boundary - Collaboration requires websocket connectivity to the y-websocket server.

4. Problems and resolution

This section is to document the main issues encountered during implementation of the system, and how they were resolved.

4.1 Frontend module resolution and project structure

Problem - The frontend failed to build due to missing or incorrect components. This prevented the UI from rendering and made the application appear blank even though the Tauri window opened.
Cause - The react renderer uses a specific folder structure, import paths were mismatched due to a non standardised file structure.
Resolution - The renderer was standardised, components were relocated to a common folder for easier pathing.

5. Installation Guide

This is to guide the user in the installation of the collaborative code editor.

5.1 System requirements

Operating system

- Windows 10 (version 1803+) or windows 11

Recommended hardware

- CPU - 2+cores
- RAM - 4 GB minimum (8GB recommended)
- Disk - 300 MB + free for the app
- Network Connection

5.2 Installing the desktop application

1. Download the installer.
    a. Go to the github releases page and download the latest release (v0.1.01).
2. Run the installer.
    a. When windows smartscreen comes up choose more info and run anyway.
3. Complete setup
    a. Follow installer instructions
4. First launch
    a. The app should start automatically and create a desktop shortcut.
    b. Create an account via clicking the button at the bottom of the login box.

5.3 Uninstallation

1. Settings -> Apps -> installed apps
2. Find the application name (desktop)
3. Click uninstall