Online Text Editor

Abraham Lenson
Rustenis Tolpežnikas

Contents

1 Introduction **2**

2 General Assumption **2**

3 Technology Stack **4**

4 System Architecture **5**

5 Behavioral Diagrams **6**

5.1 Sequence Diagram 6

6 Testing **7**

1 Introduction

This report documents the implementation of our online distributed text editor implementation. Starting from concept design, revisions and changes, and our general testing process. This report will heavily reference our previous concept design document as it is the basis of our project.

2 General Assumption

The goal of this project is to develop a Decentralized Online Text Editor. Here are the requirements given specifically to this project:

- Each user of the editor should be able to see the current state of the edited document, can edit and erase the text.
- There should be cursors visible to all users, symbolizing where other editors are currently positioned in the document (in other terms, which portion are they currently editing).

Furthermore, there were general requirements given to all the projects of this course:

- The developed system should be multi-node. Two or more computers should constitute the nodes; additional nodes can run in VMs or similar containers.
- The system should have a multi-part design, as we are a 2-person team, this would mean 2 parts.
- Each part should work on a different OS and be written in a different language.
- The system should have some fault tolerance, such as handling network disconnection and node failure, as well as not reset after a malfunction and restore the distributed state at least partially.

Given these initial requirements, in team we have decided that the 2 parts will be 2 similar but different implementations of the individual nodes. We prepared a backend node and a frontend node. The backend communicates with each other while the frontend handles the UI and could accept multiple clients.

For the system to work on different OS systems, we would be developing the 2 different node implementations in programming languages known to be OS-agnostic (such as Python, JavaScript, etc.).

Given these initial requirements and the defined general assumptions, we can categorize the requirements and refine them:

RE1	Real-Time Sync	All nodes must synchronize text edits in real-time.
RE2	Multi-Node Support	System must support at minimum 3 connected nodes, without a hard upper limit (but soft limit could be stated as 10).
RE3	Cursor Visibility	Remote cursors must be visible to all users with positional updates.
RE4	Editor UI	Provide a functional editor interface, which can be used to edit (insert, delete, select) the text.
RE5	Dynamic Election	If a distinguished node exists, elect a new leader upon failure.
NRE1	Cross-Platform	The implementation of the node should not be limited to a single programming language, and it should be possible to run the system on different Operating Systems.
NRE2	Fault Tolerance	System must recover from node/network failures without full reset (partial state restore).
NRE3	Debug Mode	The system should provide access to its inner state for the sake of debugging.

3 Technology Stack

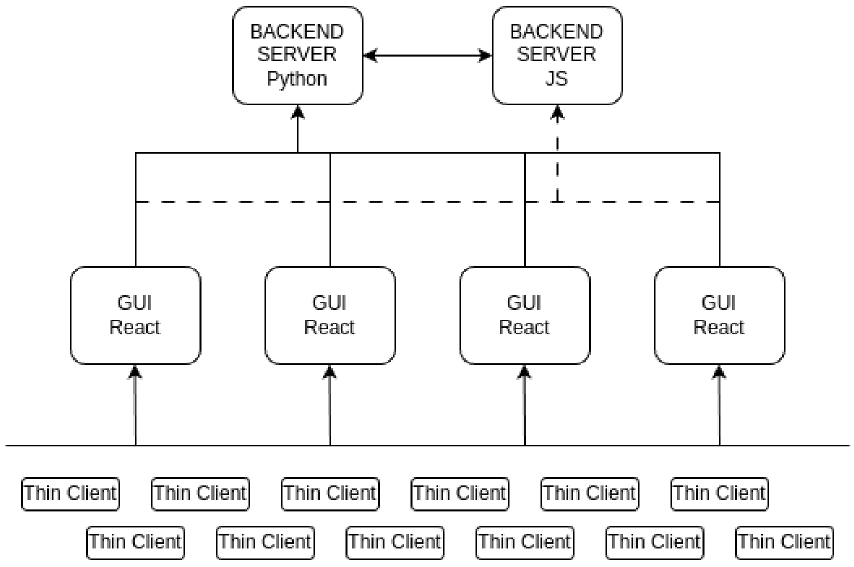
In order to implement the system, we need to make sure that the data are synchronised correctly across different clients. To do that, we need a framework to deal with state changes, desynchronization, merge conflicts, and other issues that may arise from a decentralised system. To deal with that we are going to use CRDTs (Conflict-Free Replicated Data Types), a data structure that allow distributed systems to sync seamlessly without coordination and ensuring eventual consistency. They handle concurrent edits by making all operations commutative, associative, and idempotent.

Due to the cross-language and cross-platform approach required, we scrapped our original idea of using YATA based approaches. Instead, we keep things mostly simple with a Last-Write-Wins based approach through websocket. This generally works pretty well over a local network, and generally doesn't encounter issue such as tombstoning or bloating from previously deleted elements.

4 System Architecture

Abstract top-down view of the system:

Figure 1: Abstract



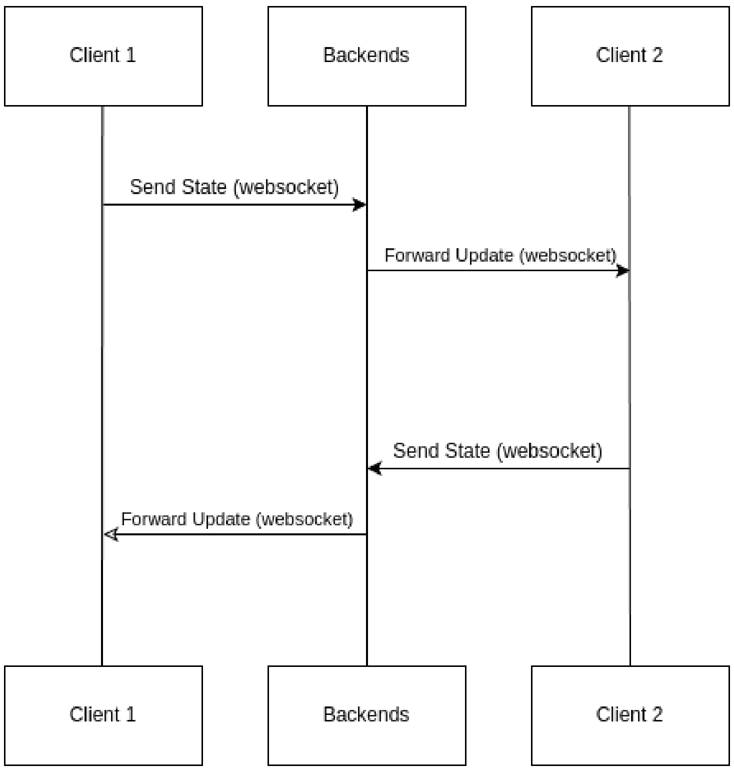
Key concepts explained:

Concept	Description
GUI Server	• Allows the client to see the up-to-date version of the document. • Enables the textual editing of the said document. • Shows the concurrent user sessions as cursors. • Handles Document Processing
Backend Server	• Bridges to the GUI Server through WebSocket layer • Handles synchronisation between clients • Switches Role between backup and main server depending on server status
Websocket	• Standardized message format • Routes messages between language backends • Handles connection state management
Thin Clients	• Shows UI through a browser • Handles User Input

5 Behavioral Diagrams

5.1 Sequence Diagram

Figure 2: Sequence Diagram



6 Testing

To evaluate the system's reliability, real-time synchronization, and multi-node support, we set up a controlled networked environment. The testbed consisted of multiple laptops connected within a single local area network (LAN). The setup included two backend server nodes and several frontend client nodes, each running on different operating systems to comply with the cross-platform requirement.

Each server node handled backend logic, synchronization, and message routing, while the client nodes provided the editor interface and user interaction. We connected additional laptops to the client nodes to simulate concurrent usage by multiple users.

During the test, users simultaneously edited shared documents. Edits included insertions, deletions, and cursor movements. The system successfully synchronized changes in real-time across all connected nodes. Cursor positions were updated accurately and promptly, reflecting each user's activity.

We also introduced basic fault scenarios, such as temporarily disconnecting a node. The system managed to recover without a full reset and restored a partially consistent state, validating our fault-tolerance assumptions.

Overall, the system operated as expected under realistic network conditions, meeting the key functional and non-functional requirements outlined in our design.