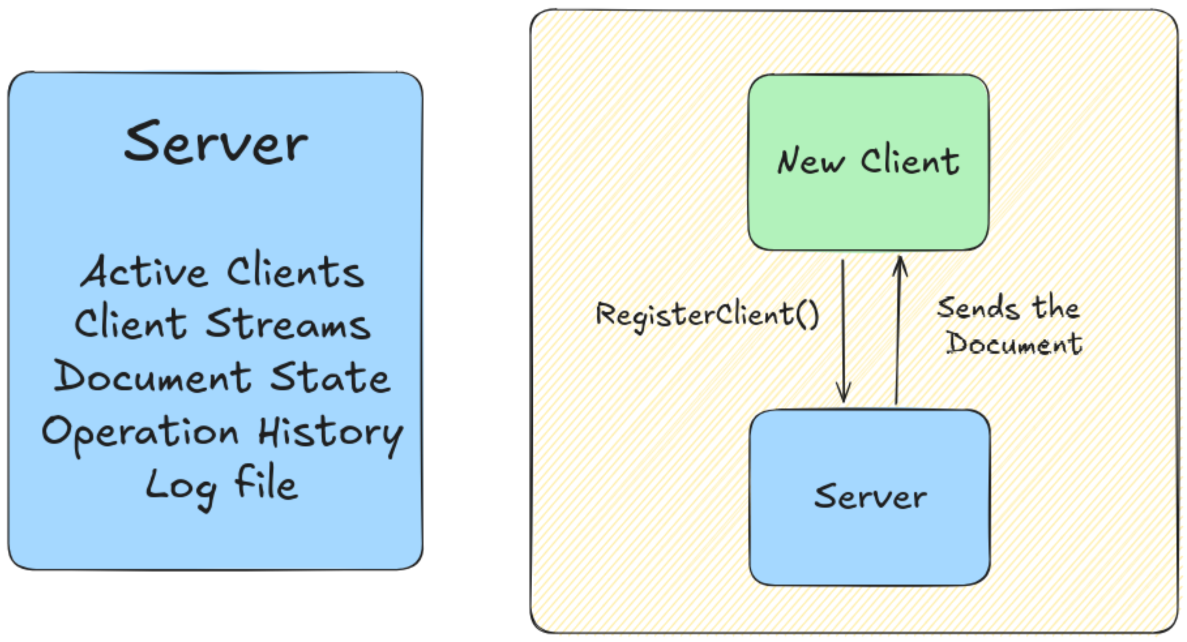
Real-time Document - gRPC

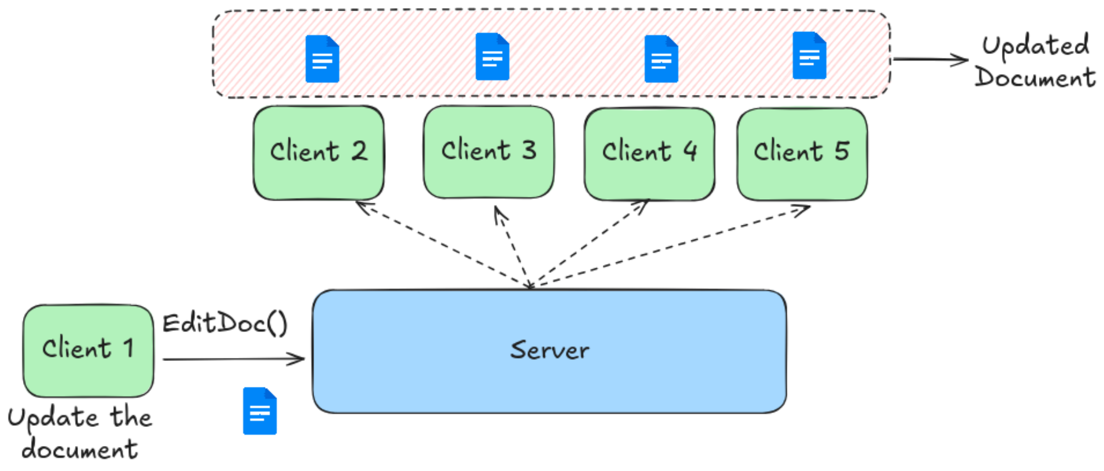
We take inspiration from the PairPad project, and follow the blog provided by authors, for this assignment we use their implementation of the CRDT data structure, as `Go` language has no implementation of it, unlike other relatively popular languages, and their interface setup (built on `termbox-go`). The refferences can be found in README submitted for this problem. While we do not use the CRDT to propogate operations across different nodes (done traditionally) instead, we use the data structure to update the document at the server via the operations done by the client. This document is then forwarded to other clients. We now begin descibing our system.

Architecture



The `Server` is responsible for maintaining consistency across all the `Clients`. It stores a list of streams to communicate to the clients, the active clients, and the history of operations performed on its `Document` . The operation performed by the client are also logged in the Log file.

When a new client joins the system, the server will send it the latest document (if it is not the first client) from then on clients can update the document, which will be registered and sent to the server in turn.



The client sends an EditDoc message via its stream. The main components (we are concerned with) are the Operation type (can be insert or delete). The operation is performed at a certain `index` and if the operation is of Insert type, then the document also comes with the `value` which is being inserted. We have a locking mechanism at the server to ensure no two operations are performed at the same time.

On the `Client` side we use a terminal interface, which listens to two types of events:

1. Keyboard events - These are events performed locally, when these are trigerred whenever a keyboard key has pressed, this updates the local state, as well as forwards this change to the Server.
2. Stream events - These events are triggered when the client recieves a message from the server. The message carries the last operation performed on the document with the updated document. The client then sees this

updated documents and sets it to his own. The cursor is adjusted accordingly on the client side.

This is achieved in `Go` via channels, where we listen to events based on whichever channel is triggered first, after each trigger we re-render our `termbox` interface via a signal (internally it is anoter channel).

The communication between the clients and the server occurs via gRPC streams. When a client registers (sending a JOIN message) to the server, the server creates a stream. This client can then communicate to the server via the stream based on above mentioned triggers.

Crash Handling

When a client crashes, the server naturally tries to send any updates to the client , not knowing it crashed. This leads to the server realizing an error and closes the stream. Our system is fault tolerant, regardless of the crashing of different clients, the server always holds the upto-date document and if new clients join, this document is again forwarded to them.