Overview

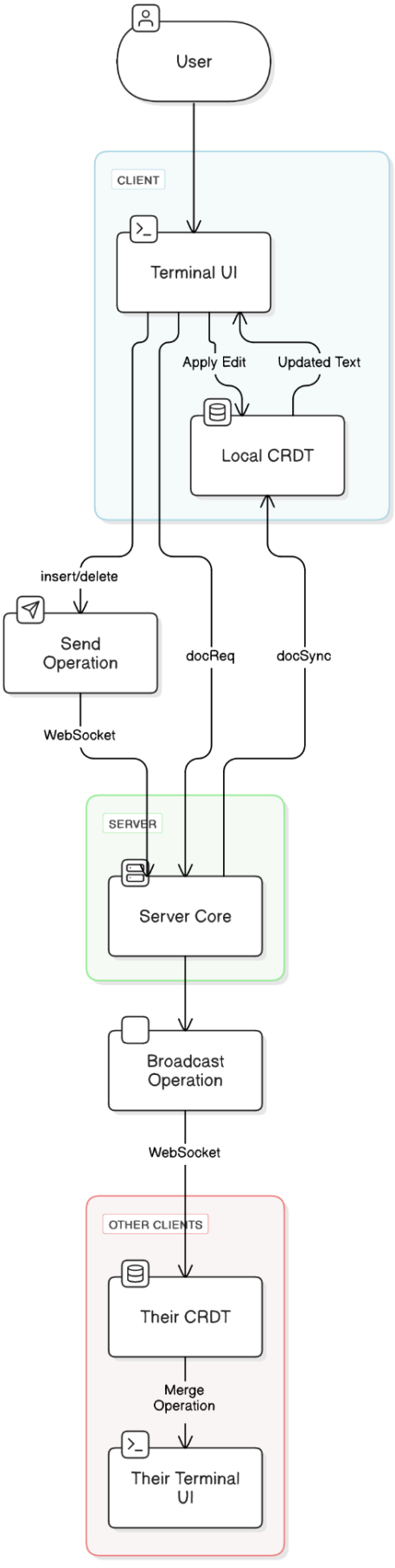
This project is a **terminal-based, real-time collaborative text editor** that allows multiple users to simultaneously edit a shared document. It comprises:

- A **Server** responsible for:
  - Managing **WebSocket** connections.
  - Broadcasting edits/updates among all connected clients.
  - Assigning unique site IDs for conflict-free operation.
- **Clients** that:
  - Provide a **text-based UI** (TUI) for user interaction.
  - Maintain a local copy of the document using a **CRDT** (Conflict-free Replicated Data Type), ensuring consistent merges of edits from multiple users.
  - Continuously synchronize changes with the server and, by extension, other clients.

High-Level Flow

Below is an illustrative diagram of how interactions typically flow among the **User**, **Terminal UI**, **CRDT**, **Client**, and **Server** components:

Collaborative Editing Flowchart



1. **User Interaction**
    - The user types text or deletes characters in the terminal interface.
    - The TUI captures these keystrokes and relays them to the local data structure.
2. **Local CRDT Updates**
    - The CRDT processes each user action (insert, delete, etc.) and immediately updates the local document state so that the user sees changes in real time.
3. **Server Communication**
    - After updating locally, the client sends an **operation message** through a WebSocket connection to the server.
4. **Broadcast to Other Clients**
    - The server receives that operation and **broadcasts** it to all other connected clients, ensuring each peer applies the same edit in their own CRDT.
5. **Real-Time Convergence**
    - Each client merges incoming updates into its local copy of the text.
    - Because the CRDT is designed to handle concurrent inserts/deletes, conflicts are avoided, and the document eventually converges to a single consistent state across every client.

Key Architectural Points

1. **CRDT Core**
    - At the heart of the editor lies a CRDT implementation that guarantees consistency even when edits arrive out of order or concurrently from different users.
2. **WebSocket Networking**
    - The server upgrades incoming HTTP requests to WebSocket connections, enabling full-duplex communication for real-time collaboration.
3. **Terminal UI**
    - Each client runs a text-based interface, capturing user keystrokes and displaying the updated document.
    - The UI remains responsive because local edits apply instantly (optimistic local updates) and network traffic is handled asynchronously.
4. **Logging and Error Handling**
    - Both client and server maintain logs of connection events, message handling, and errors.

    - On connection failures or unexpected issues, the server gracefully removes the affected client, and the client updates its UI to reflect the disconnection.

Benefits of the Architecture

- **Instant Feedback**: Edits appear immediately in the user's terminal, ensuring a responsive experience.
- **Conflict-Free Collaboration**: The CRDT approach seamlessly merges concurrent changes without data loss or manual conflict resolution.
- **Modular Design**: Server and client code are cleanly separated—networking, CRDT logic, and UI each maintain clear responsibilities.
- **Scalability**: With a broadcast-based approach, additional clients can connect without significantly complicating the synchronization logic.