Collaborative Text Editor

Swapnanil Mukherjee, Hrsh Venket

December 9, 2024

1 Overview

This project implements a real-time collaborative text editor. The editor supports multiple users editing simultaneously, rich text formatting, and export to PDF and DOCX formats.

2 Core Components

2.1 High-Level Protocols and Libraries

2.1.1 Yjs

Yjs is a comprehensive CRDT (Conflict-free Replicated Data Type) framework that enables real-time collaboration. It implements the YATA (Yet Another Transformation Approach) algorithm, specifically designed for collaborative editing.

Core Components In Yjs, each document modification is represented as an operation containing:

- A unique identifier (combining client ID and a local clock).
- The actual change (insert/delete).
- Positional information relative to other operations.
- The content being modified.

YATA Algorithm The YATA algorithm utilizes relative positioning where:

- Each character/element maintains a unique position in the document.
- Positions are defined relative to other elements rather than absolute indices.
- Each operation stores information about its left and right neighbors.
- This relative positioning ensures concurrent operations can merge without conflicts.

Conflict Resolution Process The conflict resolution follows a structured approach:

Local Change Processing:

- Assignment of unique identifier to the change.
- Calculation of position relative to surrounding elements.
- Local application and broadcasting to peers.

Remote Change Processing:

- Position determination using relative references.
- Ordering of concurrent operations using unique identifiers.
- Integration while maintaining intention preservation.

Concurrent Edit Handling:

- For simultaneous insertions at the same position, operation identifiers establish consistent order.
- Higher identifier operations are positioned after lower identifier operations..
- This ensures convergence to identical final states across peers.

State Synchronization When new peers join:

- They receive the complete document state
- The state includes all operations (including tombstones)
- Document reconstruction occurs using the YATA algorithm
- This ensures state consistency with existing peers

Algorithm Properties The YATA implementation maintains several crucial properties:

- All peers eventually achieve identical states.
- Operation effects remain preserved regardless of execution order.
- Convergence: Concurrent operations yield identical final states across peers.
- Operation application order doesn't impact the final result.

2.1.2 WebRTC

WebRTC (Web Real-Time Communication) is used for peer-to-peer communication:

- Enables direct communication between users' browsers.
- Handles connection establishment and message passing.
- Integrated with Yjs through `y-webrtc` provider.

2.1.3 Quill.js

Quill is a modern rich-text editor package. It was chosen for the following capabilities:

- Provides the core editing interface.
- Handles text formatting and styling.
- Manages editor state and content operations.
- Offers extensible API for custom functionality.

2.1.4 React

React serves as the frontend framework:

- Manages UI components and state.
- Handles user interactions.
- Integrates various libraries into a cohesive application.

3 Project Structure

3.1 File Organization

The project consists of three main files:

- `App.jsx`: Main application component.
- `Editor.css`: Editor-specific styles.
- `index.css`: Global styles.

4 Functional Documentation

4.1 Logic

The main React component containing the editor implementation.

4.1.1 State Management

```
const [editor, setEditor] = useState(null);
const [doc, setDoc] = useState(new Y.Doc());
```

- editor: Stores the Quill editor instance
- doc: Manages the Yjs document for collaboration

4.1.2 Core Functions

Editor Initialization (useEffect)

- Creates Quill editor instance
- Configures toolbar and formatting options
- Sets up WebRTC provider for collaboration
- Binds Quill editor to Yjs document

getFormattedContent()

- Processes editor content for DOCX export
- Converts Quill Delta format to DOCX structure
- Handles text formatting and styles
- Manages paragraph organization

downloadAsDocx()

- Creates DOCX document from editor content
- Preserves formatting and styles
- Handles document creation and download

downloadAsPDF()

- Converts editor content to PDF format
- Maintains text formatting and styles
- Handles page breaks and layout
- Manages font styles and sizes
- Processes lists and special formatting

5 UI

5.1 Editor.css

5.1.1 Layout Components

- `.editor-app`: Main container styling
- `.editor-header`: Fixed header styling
- `.editor-main`: Main content area layout
- `.editor-wrapper`: Editor container styling

5.1.2 Toolbar Styling

- `.ql-toolbar`: Toolbar container
- `.ql-formats`: Format button groups
- `.ql-picker`: Dropdown styling
- Button styling and hover states

5.1.3 Editor Content Styling

- `.ql-container`: Main editor container
- `.ql-editor`: Content area styling
- Text formatting and layout
- Page dimensions and margins

6 Project Setup

6.1 Requirements

- Node.js (v14 or higher)
- npm
- Modern web browser with WebRTC support

6.2 Dependencies

Core dependencies:

```
{
  "dependencies": {
    "react": "^18.0.0",
    "react-dom": "^18.0.0",
    "yjs": "^13.5.0",
    "y-webrtc": "^10.2.0",
    "quill": "^1.3.7",
    "y-quill": "^0.1.5",
    "docx": "^7.3.0",
    "jspdf": "^2.5.0",
    "file-saver": "^2.0.5",
    "lucide-react": "^0.263.1"
  }
}
```

6.3 Installation

```
# Clone the repository
git clone [repository-url]
cd [project-directory]

# Install dependencies
npm install
```

6.4 Building the Project

```
# Development build
npm run dev
```

6.5 Running the Project

```
# Start development server
npm run dev

# Access the application
# Open http://localhost:5173 in your browser
```

7 Additional Notes

7.1 Browser Support

The application requires:

- Modern browser with WebRTC support
- JavaScript enabled

- Stable internet connection for collaboration
- Port 5173 **NOT** used by any other process.

7.2 Collaboration Features

- Real-time synchronization across users
- Automatic conflict resolution
- Presence awareness
- No central server required for collaboration

7.3 Export Features

- PDF export with formatting preservation
- DOCX export with styles and structure
- Support for various text formats and styles

8 Limitations

8.1 Cross-Browser Communication

The collaborative editing feature only works between instances running on the same browser type. For example:

- Two Chrome windows/tabs can collaborate successfully
- A Chrome window cannot collaborate with a Firefox window, even when accessing the same port number
- This is a fundamental limitation of the WebRTC peer-to-peer protocol implementation in different browsers

8.2 Document Version Control

The peer-to-peer architecture, which operates without a central server, inherently lacks version control capabilities:

- No history of document changes is maintained
- Changes are synchronized in real-time between peers but not stored historically
- Previous versions of documents cannot be accessed or restored
- This limitation exists because there is no central database to store document versions

8.3 Document Storage and Management

The peer-to-peer implementation restricts document management capabilities:

- Documents exist only in browser session storage
- Users cannot open existing documents from their local system
- No ability to create new documents while maintaining current collaborative sessions
- Document state persists only as long as at least one peer remains connected
- Once all peers disconnect, the document state is lost