IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 11, Issue 3, March 2024

DOI: 10.17148/IARJSET.2024.11347

# Collaborative Code Editor Using Web Application

**N.Jaya Santhi[1], D.Sireesha[2], E.Vindhya[3], D.Naga Jyothi[4]**

M.Tech, Asst.Professor, Computer Science & Engineering, Bapatla women's Engineering College, Bapatla, India[1]

B.Tech, Computer Science & Engineering, Bapatla women's Engineering College, Bapatla, India[2-4]

**Abstract**: Code Together is a collaborative code editor facilitating real-time communication among developers through voice calls and chat. Its whiteboard feature aids in algorithm framing and collaborative planning. Saved code updates are instantly reflected across collaborators' screens. Access to relevant articles aids in resolving coding queries. Developed by a team of remote developers, it caters to Agile teams and troubleshooting sessions alike. It's very good at pair programming, mob programming, and code review.Unlike traditional screen sharing, When another developer wants to edit the file, they download and edit the copy locally and send the edited copy to the interested group through a communication channel. It supports multiple IDEs including Eclipse, IntelliJ, and VS Code. Its versatility spans pair programming, code review, project design, and more. Code Together streamlines unit testing, education, interviews, and remote development. With its intuitive interface, it merges functionality and simplicity seamlessly. Whether for professional projects or educational purposes, Code Together enhances collaborative coding experience.

**Keywords**: Code Together,real-time communication,Integrated development environment,code editing tools

## I. INTRODUCTION

The idea of collaborative editing is the ability for multiple users who are geographically apart to work together on the same set of files. Users used to send a whole copy of a file via email or other internal communication channels when they wished to share it. When another user wishes to make changes to the file, they download it, make the necessary changes locally, and then broadcast the changed version over a communication channel to all interested parties.

A different user may have been working on the prior copy during this procedure; in that case, the user receiving the new copy must manually identify the changes made by the previous user, fix any collisions, and return the modified copy. This procedure moves slowly and time-consuming.

This issue can be resolved with collaborative code editing tools, which enable numerous users to view the same file and make changes that are immediately visible to other users. These days, they are highly well-known due to the apparent advantages of pair programming. Working on the same application together is possible with pair programming for two or more developers. This boosts output, enhances the quality of the code, and facilitates simpler problem-solving (many minds on one subject).

## II. BACKGROUND AND RELATED WORK

Past studies on collaborative editing and the tools created for it have used both client-server and peer-to-peer architectures. Client-server architecture can be understood in terms of numerous clients sharing a file on a server when collaborative editing is taking place. The server updates its copy whenever a client makes changes to its local copy, and the server then distributes the updates to the other clients that are still able to access the shared file.

Peer-to-peer networks are typically more dependable because they don't rely on a single party within the network. For collaborative editing, they are extensively studied, such as COE [8] and MUTE [9]. However, managing a larger network becomes more challenging, and the system may become unstable.

### A. Adaption of Lagoot's CRDT Algorithm

A modification of the Lagoot [4] framework has been employed. Every character has its own positional identifier, which makes it easier for us to distinguish between them when combining the modifications. An illustration of a single line of text encoded in CRDT is as follows:

"abcd" => [['a', [0]], ['b', [1]], ['c', [2]], ['d', [3]]
The position identifier for character an is 0, and for other characters it is as follows. The location ID would be "0.5" if we wanted to add a character between index 0 and 1.
"a1bcd" implies [['a', [0]], ['1', [0, 5]], ['b', [1]], ['c', [2]], ['d', [3]]]

Even in the absence of a central server for coordination, selecting the positional IDs with the above approach aids in the successful merging of the updates. In the example below, client 1 inserts a character (e) at index 1, and client 2 inserts a character (d) at index 2, both clients creating modifications to the CRDT data and sending them to the server. The deletion process could be classified along the same lines.
"a21bcd" indicates [['a', [0]], ['2', [0, 4]], ['1', [0, 5]], ['b', [1]], ['c', [2]], ['d', [3]]]

## B. System Design

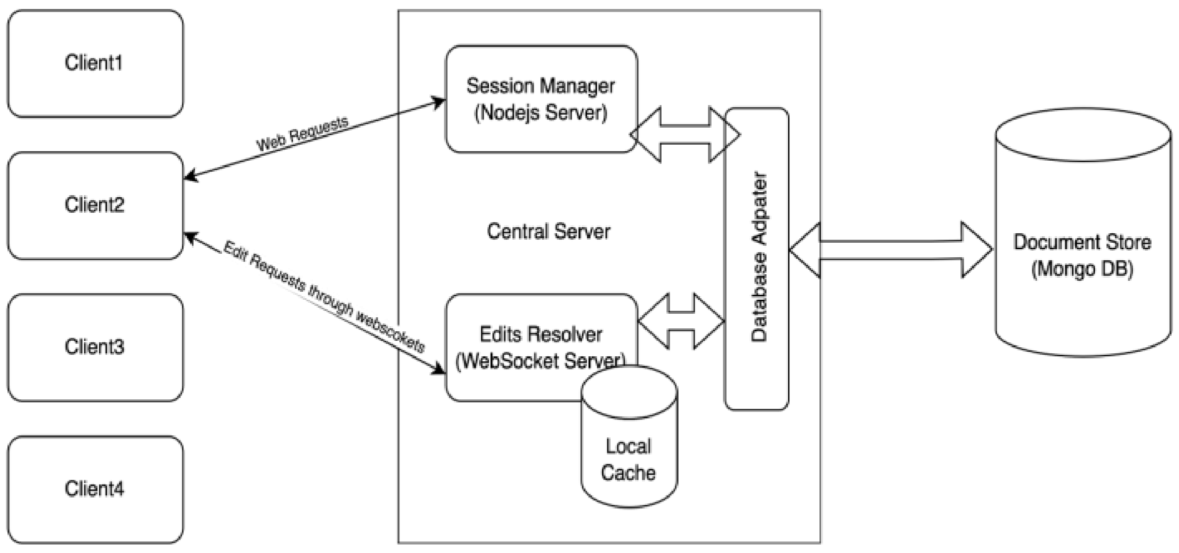

Client1
Client2
Client3
Client4
Web Requests
Edit Requests through webscokets
Session Manager (Nodejs Server)
Central Server
Edits Resolver (WebSocket Server)
Local Cache
Database Adpater
Document Store (Mongo DB)

Fig1:Revised High-level System

- **Process Flow**

1. Clients can choose a random roomID from the server URL to start a new editor document. As a result, a new document would be created and shown to the user.
2. Additional clients must join using the same roomID in order to access the same room or content.
3. Upon successful connection, a client establishes two connections: a WebSocket connection with the edit resolver server and an HTTPS connection with the session management server.
4. Via WebSockets, clients update the document and send the updates to the server. Using WebSocket, the socket server broadcasts the update to other clients in the same room by executing the CRDT logic for synchronization.
5.The change at the server side is cached and persisted to MongoDB cloud instance on Atlas using batch processing.
6.Once every client is disconnected, the session is closed, and the document is saved in DB.

## III. IMPLEMENTATION

### A. Modules Description

1. **Node.js**: The express framework runs 2 servers: HTTPS server and WebSocket server. HTTPS server is responsible for handling HTTP get requests and the WebSocket server is responsible for handling socket.

• **HTTPS server**: Clients are connected to the HTTPS server through HTTP requests and the server continuously listens to the incoming requests. The requests are handled using the request object and the response from the server is initiated using the response object. The following HTTP endpoint is exposed to the clients.

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁑ Peer-reviewed & Refereed journal ⁑ Vol. 11, Issue 3, March 2024

DOI: 10.17148/IARJSET.2024.11347

• **app.get('/:roomID') :**When the server receives the HTTP get request with the URL ending with /<roomId>, The server retrieves the document from the cache if it is available, else it fetches the document from the MongoDB. If the document doesn't exist in the database, we create a new entry for the document in the local cache with document content '// New document - <ID>' and insert this new entry into the database. It renders the HTML page and sends it as a response to the client.

• **WebSocket server:** Continuous message sending between the client and server is made possible by web sockets. Clients first establish a connection with the server by sending an HTTP request. In this project, as soon as the server receives an HTTP request, the clients create socket connections with it. Through sockets, the client transmits the modifications to the server, which broadcasts them to the clients.

**Client-side socket events**

| **Event** | **Args** | **Events emitted** | **function** |
|---|---|---|---|
| connect | None | CONNECTED_TO_ROOM | Establishes socket connection with the server |
| INITIAL_DOCUMENT | crdtData | None | Receives initial CRDT data of the document from server. |
| CODE_CHANGED | changes | None | Receive the changes (made by other clients) from the server. |

**Server-side socket event**

| **Event** | **Args** | **Events emitted** | **function** |
|---|---|---|---|
| connect | None | None | Server listens for connections from the client's |
| disconnect | None | None | When a session gets disconnected, the corresponding user is deleted from the list of active users. |
| CONNECTED_TO_ROOM | roomID, userName | INITIAL_DOCUME NT | Stores the document ID and the username corresponding to the session. It sends the CRDT data of the corresponding documents by |
| CODE_CHANGED | changes | CODE_CHANGED | Update the changes received from the client in the local cache and broadcast the changes to all the clients working on that document. |

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 11, Issue 3, March 2024

DOI: 10.17148/IARJSET.2024.11347

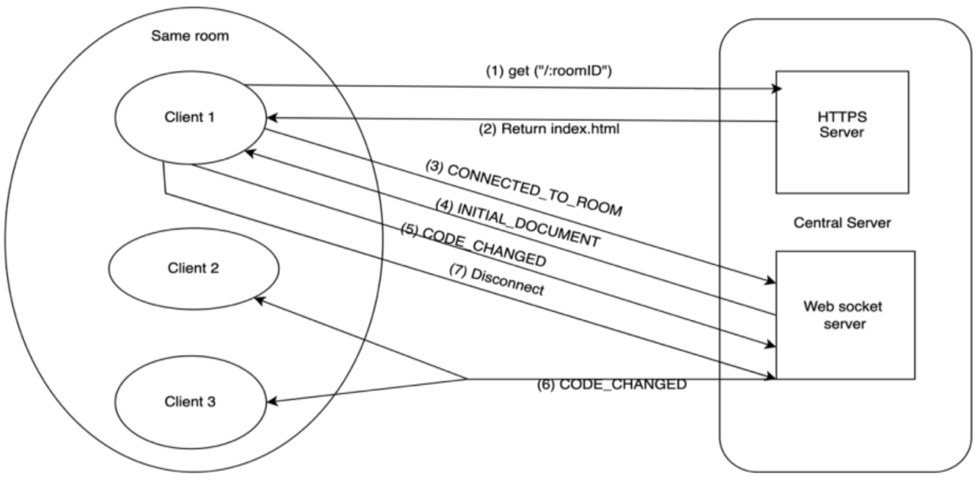


Fig1:HTTPS and Socket events from client

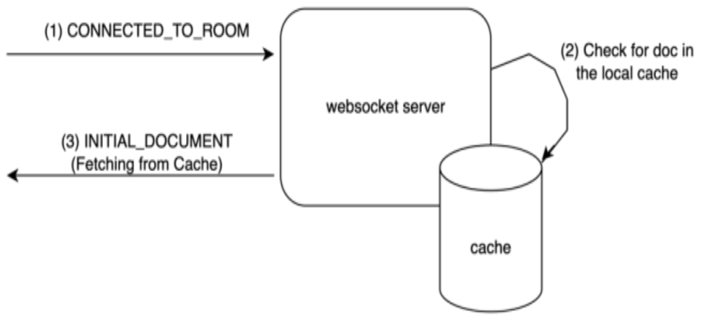
2. **MongoDB**: The shared files are stored in MongoDB. We used a free-tier MongoDB Atlas cloud cluster that was set up on AWS in the North Virginia area for the project. The 512MB storage limit of the free version in use allows for the creation of 500 collections over a maximum of 100 databases.

3. **MongoDB Adapter**: The class provides methods to interact with MongoDB. For the project, we created a database *'ice's* and collection *'docs'*. It exposes the following methods:

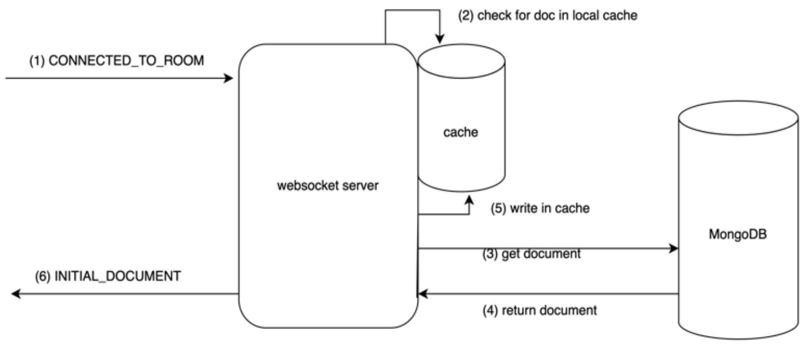
- document Exists(room ID): Check if there is an entry in the *docs* collection with given room ID.
- addDocumentToDB(room ID, data): Inserts a new document entry to the *docs* collection.
- document Exists(room ID): Check if there is an entry in the local cache with the given room ID.
- addDocumentToCache(room ID, data): Inserts a new document entry in the local cache.
- getCRDTData(room ID): Fetches the CRDT data of the document from the local cache.
- insertToCRDT(room ID, changes): Insert operation on CRDT data of document in the cache.
- deleteFromCRDT(room ID, changes): Delete operation on CRDT data of document in the cache.

## A. Server-Side Events:

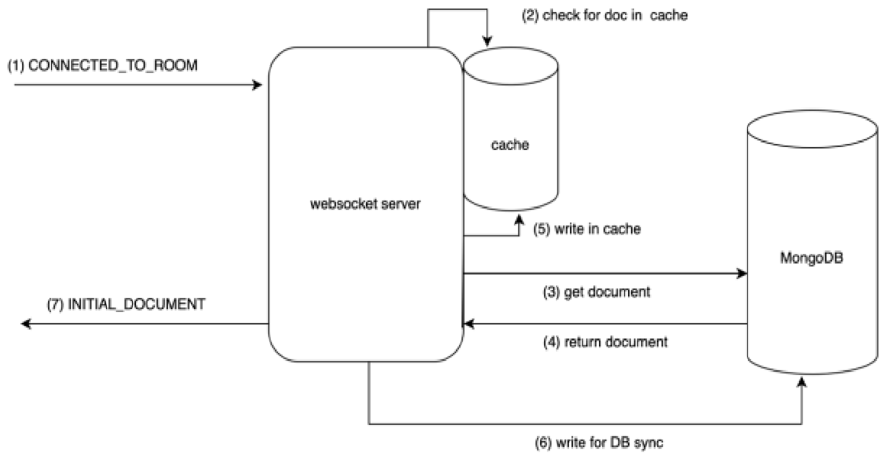
Case1: When the document exists in the cache,


(1) CONNECTED_TO_ROOM
websocket server
(2) Check for doc in the local cache
(3) INITIAL_DOCUMENT (Fetching from Cache)
cache

Case 2: When the document does not exist in the cache and is fetched from MongoDB,

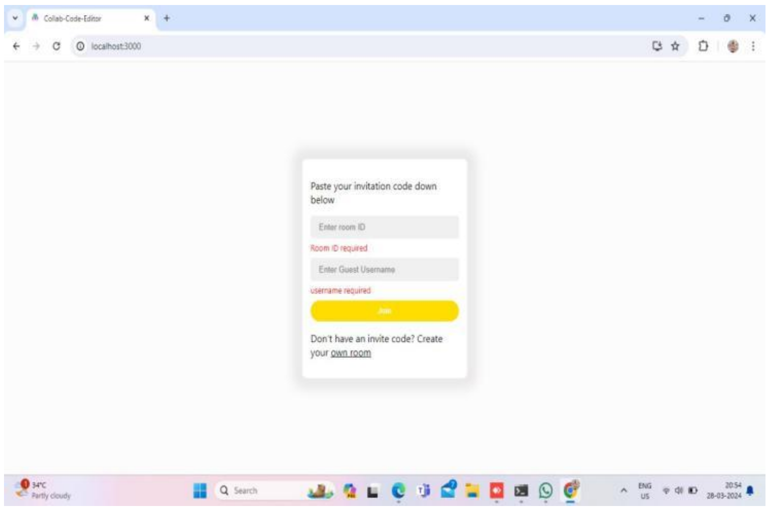


Case 3: When the document does not exist in the cache as well as MongoDB,



## IV. RESULTS AND ANALYSIS

**User Interface**: HTML, CSS, and JavaScript are the building blocks of the web user interface. The username field appears when the client tries to view the document at https://localhost:8080/1, as seen in the following figure.



Paste your invitation code down below

Enter room ID

Room ID required

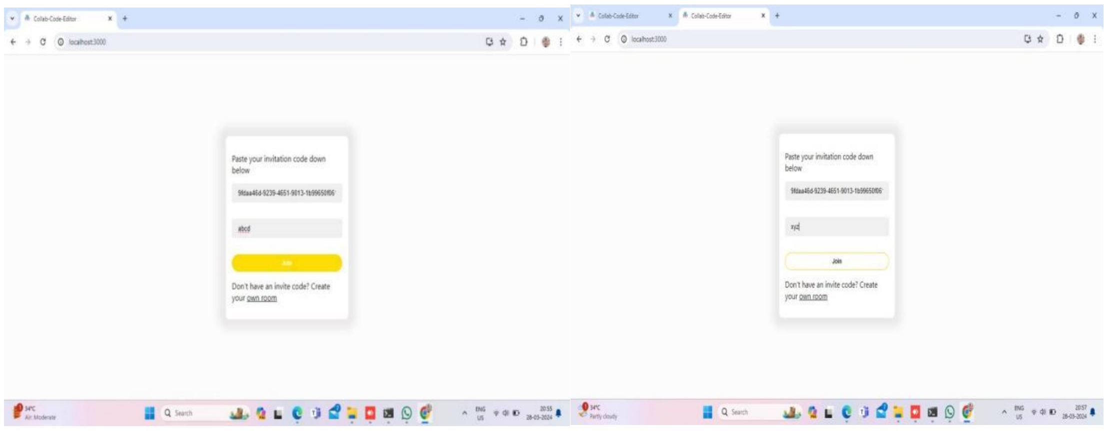
Enter Guest Username

username required

Join

Don't have an invite code? Create your own room

Fig2:create a own room



Paste your invitation code down below

9fdaa46d-9239-4651-9013-1b99650f06

abcd

Join

Don't have an invite code? Create your own room

Paste your invitation code down below

9fdaa46d-9239-4651-9013-1b99650f06

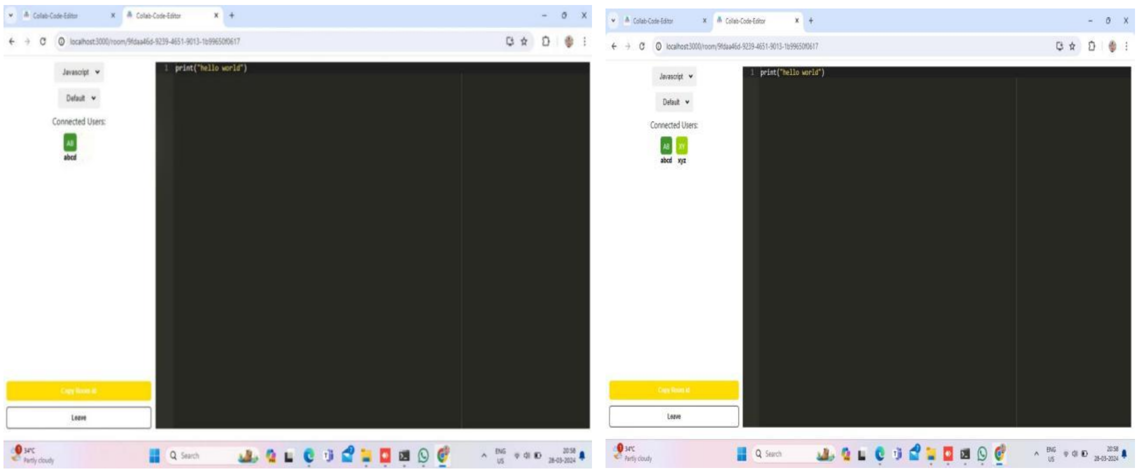
xyz

Join

Don't have an invite code? Create your own room

Fig3:users join the room

DOI: 10.17148/IARJSET.2024.11347



```
print("hello world")
```

```
print("hello world")
```

Fig4:multi users join the room and code edit

## V. CONCLUSION

Our objective for this project was to create a collaborative editor solution that provides users with a smooth code editing experience. While developing the tool, we looked into a variety of architectures and algorithms that provided a wide view of current systems, their benefits and drawbacks, design limitations, and trade-offs.

We were able to determine which architecture would work best for our particular application by analyzing a number of ideas that were offered in research articles. The hardest aspect was creating algorithms that would work for different users. We made it happen by putting CDRT into practice. If CDRT is implemented successfully, we should be able to update users more quickly. Despite having simple text editing features, CCE implements them well.

## VI. FUTURE WORK

The project will require work in the future to switch from stateful to stateless, construct a global cache, and use a Redis pub sub-system to alert other servers' sockets to code-change events.

Version control can be incorporated into the program. The ability to generate a branch at any moment would result in the development of a new child session using the snapshot of the current document. For increased security and improved user experience, additional features like Chat/Call and user authentication could be included.

## REFERENCES

[1]. Nichols, David & Curtis, Pavel & Dixon, Michael & Lamping, John. (1995). High-Latency, Low Bandwidth Windowing in the Jupiter Collaboration System. 111-120. 10.1145/215585.215706.
[2]. Christopher R. Palmer and Gordon V. Cormack. 1998. Operation transforms for distributed shared readsheet. In Proceedings of 1998 ACM conference on Computer supported cooperative work (CSCW '98). Association for Computing Machinery, New York, USA, 69–78.
[3]. C.A. Ellis and S.J. Gibbs. Concurrency control in groupware systems. In J. Cli ord, B. Lindsay, and D. Maier, editors, Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 399{407, Portland, OR June 1989. ACM SIGMOD Record, 18:2.
[4]. Stephan Weiss, Pascal Urso, Pascal Molli. Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks. 29th IEEE International Conference on Distributed Computing Systems - ICDCS 2009, Jun 2009, Montreal, Canada. pp.404-412 .
**[5].**"Specification and Complexity of Collaborative Text Editing",[online] Available:**https://software.imdea.org/~gotsman/papers/editing-podc16.pdf**
[6]. Collaborative opportunistic network coding for persistent data stream in disruptivesensor networks Publisher : IEEE - 2021 Author : Mingsen Xu
[7]. " MUTE: A Peer-to-Peer Web-based Real-time Collaborative Editor" [online] Available: **https://hal.inria.fr/hal-01655438/document**

[8]. "COE: A collaborative ontology editor based on a peer-to-peer framework", [online] Available: https://www.sciencedirect.com/science/article/pii/S1474034605000364
[9]. "Shared editing on the web: A classification of developer support libraries", [online] Available: https://ieeexplore.ieee.org/abstract/document/6680014

## BIOGRAPHY

**N.Jaya Santhi,** M.tech, Asst.Professor , Dept of Computer science & Engineering, BWEC, Andhra Pradesh, India

**D.Sireesha** [B.Tech],Student, Dept of Computer science & Engineering, BWEC, Andhra Pradesh, India

**E.Vindhya**[B.Tech], Student, Dept of Computer science& Engineering, BWEC, Andhra Pradesh, India

**D.Naga Jyothi**[B.Tech],Student ,Dept of Computer science& Engineering, BWEC,Andhra Pradesh, India