International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 3, May-Jun-2026, ISSN (Online): 2395-566X

Real-Time Collaborative Code Editor Using WebSockets

Pruthviraj Pawar, Niranjan Rasal, Pruthviraj Deshmukh, S. B. Dighe
Department of Electronics and Computer Engineering Amrutvahini College of Engineering
Sangamner, Maharashtra, India

Abstract- Collaborative programming platforms are becoming increasingly important in modern software development, online education, and distributed teamwork environments. Traditional methods such as manual file sharing, screen sharing, or repeated version control synchronization are often inefficient during live coding sessions. This paper presents the design and implementation of a Real-Time Collaborative Code Editor developed using WebSockets and Socket.IO. The proposed system allows multiple users to edit source code simultaneously through a browser-based interface with minimal synchronization delay. The frontend is developed using HTML, CSS, JavaScript, and Monaco Editor, while the back- end is implemented using Node.js and Express.js. Socket.IO is used to establish persistent bidirectional communication between connected clients and the server. Experimental observations demonstrate synchronization la- tency below 100 milliseconds under normal conditions. Perfor- mance analysis confirms that WebSocket-based communication provides significantly lower delay and better bandwidth efficiency than traditional HTTP polling techniques.

Keywords – WebSockets, Socket.IO, Collaborative Program- ming, Monaco Editor, Browser-Based IDE, Real-Time Synchronization, Node.js.

I. INTRODUCTION

Collaborative programming platforms have become increasingly important in modern software development, online education, and distributed teamwork environments [9].

Traditional software development workflows usually depend on asynchronous collaboration using Git repositories, file sharing, or screen sharing applications. Although such methods are effective for long-term project management, they are less suitable during live collaborative sessions where developers need immediate synchronization of changes.

Real-time collaborative code editors allow multiple users to modify the same source code simultaneously. Any modification made by one participant becomes visible to all connected users almost instantly.

WebSockets provide a persistent communication channel between client and server, avoiding repeated connection establishment overhead associated with HTTP polling [1]. Socket.IO simplifies synchronization and event handling between connected users [2].

Monaco Editor is integrated to provide a professional coding experience directly in the browser [3].

II. PROBLEM STATEMENT

Traditional approaches for collaborative coding often introduce communication delays and synchronization overhead. File sharing and version control systems are not suitable for instant collaboration because users cannot observe updates immediately.

The major limitations include:

- Delays caused by repeated HTTP polling.
- Difficulty maintaining synchronization during simultaneous editing.
- Dependence on locally installed IDEs and plugins.
- Increased bandwidth usage during continuous synchronization.

The proposed system aims to overcome these issues by providing low-latency synchronization through WebSockets and efficient room-based collaborative editing.

III. OBJECTIVES

The main objectives of the proposed system are:

1. Develop a browser-based collaborative code editor.
2. Support simultaneous multi-user editing.
3. Achieve synchronization latency below 100 milliseconds.
4. Integrate Monaco Editor for browser-based editing.
5. Provide secure code execution support.

6. Improve communication efficiency compared to polling-based systems.

IV. LITERATURE SURVEY

Collaborative editing systems have been widely studied in distributed computing research.
WebSockets establish a persistent TCP connection between clients and the server, unlike HTTP polling which repeatedly creates new requests [1]. This reduces communication delay and bandwidth usage.

TABLE I
COMPARISON OF COMMUNICATION TECHNIQUES

Feature	WebSockets	HTTP Polling
Communication	Full Duplex	Request/Response
Connection Type	Persistent TCP	Repeated Requests
Latency	Very Low	Higher
Bandwidth Usage	Efficient	Higher Overhead

Operational Transformation (OT) and Conflict-Free Replicated Data Types (CRDT) are commonly used synchronization algorithms for collaborative editing systems [5], [6].

TABLE II OT VS CRDT

Aspect	OT	CRDT
Architecture	Centralized	Distributed
Offline Support	Limited	Supported
Conflict Handling	Transform Functions	Automatic Merge
Complexity	Medium	Higher

Recent projects such as CodeSync demonstrate the effectiveness of WebSocket-based synchronization with browser-based editors [9].

V. PROPOSED SYSTEM

The proposed system follows a client-server architecture where users collaborate through synchronization rooms man- aged by a Node.js server.

The frontend interface is implemented using HTML, CSS, JavaScript, and Monaco Editor [3]. Users connect to the backend server using Socket.IO [2].

Whenever a user edits code, synchronization events are immediately transmitted to the server. The server broadcasts these updates to all connected users in the same room.

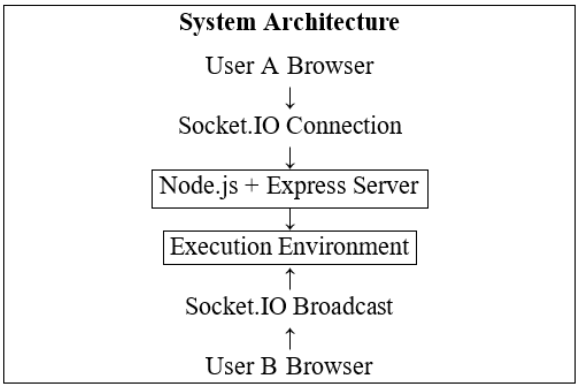

Fig. 1. Architecture of Collaborative Code Editor

VI. SYSTEM ARCHITECTURE

A. Frontend Module
The frontend provides the collaborative coding interface us- ing Monaco Editor [3]. The editor supports syntax highlighting and intelligent editing features.
B. Backend Module
The backend server is implemented using Node.js and Express.js [7], [8]. Socket.IO manages room creation and synchronization between connected users [2].
C. Execution Module
Source code execution is handled using Docker containers or Judge0 API [4].

VII. METHODOLOGY

The proposed system follows an event-driven synchroniza- tion methodology.

1. A user creates a collaboration room.
2. Other users join the room.
3. Editor changes generate synchronization events.
4. Socket.IO sends updates to the server.
5. The server broadcasts updates to connected users.
6. Users receive synchronized updates instantly.

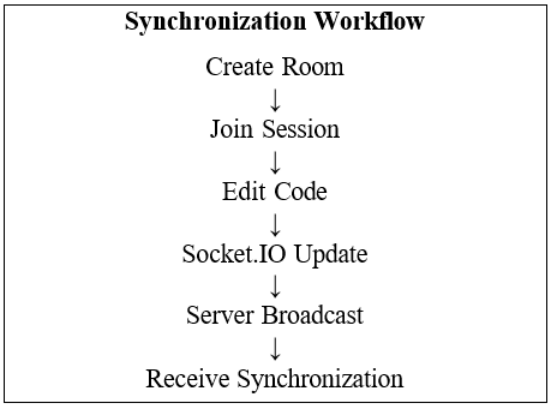

Fig. 2. Workflow of Real-Time Synchronization

VIII. ALGORITHMS AND TECHNIQUES

The system relies on several synchronization techniques.
A. WebSocket Communication
WebSockets establish persistent TCP communication be- tween client and server [1].
B. Event Broadcasting
Socket.IO broadcasts synchronization updates to all con- nected users [2].
C. Conflict Resolution
Operational Transformation and CRDT algorithms improve consistency during concurrent editing [5], [6].

IX. IMPLEMENTATION

The collaborative editor was implemented using modern web technologies.

TABLE III
TECHNOLOGIES USED

Component	Technology
Frontend	HTML, CSS, JavaScript
Editor	Monaco Editor
Backend	Node.js, Express.js
Communication	Socket.IO
Database	MongoDB
Execution Engine	Docker / Judge0 API

The frontend establishes Socket.IO communication using:
const socket = io();
The backend broadcasts synchronization events using:
socket.to(roomId)
.emit("editor-update", data);

X. RESULTS AND DISCUSSION

The proposed system was tested on local networks using multiple connected users.

TABLE IV
OBSERVED PERFORMANCE METRICS

Metric	Observed Value
Synchronization Delay	70–90 ms
CPU Usage	Below 12%
Memory Usage	Approximately 55 MB
Concurrent Users Tested	6 Users
Bandwidth per Update	Around 1 KB

Experimental observations confirmed that WebSocket com- munication significantly reduces synchronization delay com- pared to traditional polling mechanisms [1].

XI. ADVANTAGES

- Low-latency synchronization
- Browser-based accessibility
- Efficient bandwidth usage
- Multi-user collaborative editing
- Secure code execution
- Scalable architecture

XII. LIMITATIONS

- Monaco Editor increases frontend bundle size.
- Advanced conflict resolution is not fully implemented.
- High concurrency increases synchronization complexity.

Future Scope
Future improvements may include:
- AI-assisted code suggestions
- Cloud deployment support
- Mobile-friendly collaborative interface
- GitHub integration
- Advanced CRDT synchronization

XIII. CONCLUSION

This paper presented the design and implementation of a Real-Time Collaborative Code Editor using WebSockets and Socket.IO.

The proposed system successfully demonstrated browser-based collaborative programming with low synchronization delay and efficient communication. Experimental observa- tions confirmed that WebSocket communication significantly improves synchronization speed and reduces communication overhead compared to traditional polling mechanisms.
The proposed architecture provides a scalable and practical solution for collaborative programming environments.

REFERENCES

1. I. Fette and A. Melnikov, “The WebSocket Protocol,” RFC 6455, 2011.
2. Socket.IO Documentation, “Real-Time Bidirectional Event-Based Com- munication,” 2025.
3. Microsoft, “Monaco Editor Documentation,” 2025.
4. Docker Documentation, “Docker Container Platform,” 2025.
5. C. Sun and C. Ellis, “Operational Transformation in Real-Time Group Editors,” ACM CSCW, 1998.
6. M. Shapiro et al., “Conflict-Free Replicated Data Types,” IEEE Sympo- sium, 2011.
7. Node.js Documentation, “Node.js Runtime Environment,” 2025.

8. Express.js Documentation, “Minimal Web Framework for Node.js,” 2025.
9. M. Bagula and S. Kishore, “Real-Time Collaborative Code Editor,” IJCRT, 2026.