CODEANDCOLLAB: A VIRTUAL CODING ENVIRONMENT

Nagesh B C[1], Priya N V[2], Namith[3], Rakesh S[4], Praveen N Patil[5]

[1]Student, [2]Assistant Professor, [3]Student, [4]Student, [5]Student
Department of Information Science and Engineering
[1]Bangalore Institute of Technology
[1]Bangalore, India

[1]nageshbc55@gmail.com, [2]priyanv@bit-bangalore.edu.in, [3] namithnamith37@gmail.com,
[4] rakesh.s552004@gmail.com, [5] praveenpatil8377@gmail.com

***Abstract*—This paper presents CodeAndCollab, a web based platform developed to support real time collaborative programming with secure code execution. The system addresses the increasing need for effective remote collaboration tools in software development and computer science education by enabling multiple users to edit code simultaneously, communicate instantly, and execute programs safely. The platform is built using modern web technologies such as React, Node.js, and Socket.io to provide efficient event driven synchronization with minimal delay. Secure execution is achieved through containerized deployment using Docker and sandboxed environments provided by the Piston API. Experimental evaluation shows that the system maintains low synchronization latency with multiple concurrent users, efficient resource utilization, and stable connections. User based evaluation indicates improvements in collaborative productivity and learning effectiveness. The proposed solution offers a scalable and open source approach to enhancing collaborative coding environment.**

***Index Terms*— collaborative Programming, Real-Time Syn chronization, WebSocket, React, Node.js, Docker Containeriza tion, Web IDE, Distributed Development.**

I. INTRODUCTION

The landscape of software development has evolved significantly toward distributed and remote collaboration, accelerated by global trends in remote work and geographically dispersed teams. While version control systems such as GitHub and GitLab have revolutionized asynchronous collaboration, they provide limited support for realtime synchronous programming activities. Traditional integrated development environments primarily focus on individual productivity, creating a substantial gap in tools that effectively support pair programming and team based coding sessions in distributed settings.

Existing collaborative text editors such as Google Docs demonstrate the feasibility of realtime synchronization for unstructured documents, but they lack the specialized capabilities required for programming contexts. These capabilities include syntax aware editing, structured code navigation, integrated debugging, and secure code execution environments. Commercial solutions such as Visual Studio Live Share offer some collaborative features, but they remain tightly coupled with specific ecosystems and lack the transparency and flexibility of open source alternatives.

CodeAndCollab addresses these limitations by providing a comprehensive web based platform that integrates realtime collaborative editing, communication tools, and secure execution capabilities. The system architecture is designed to support synchronous collaboration while maintaining the responsiveness and reliability expected of modern development tools.

The primary contributions of this work include a novel integration of realtime collaborative editing with secure programming environments, a scalable client server architecture that supports low latency synchronization through optimized WebSocket communication, comprehensive performance evaluation demonstrating the system's effectiveness under realistic usage scenarios, and an open source implementation that provides transparency and customization opportunities for the developer community.

II. LITERATURE SURVEY

A. Foundational Work in Collaborative Editing

The theoretical foundations of realtime collaborative editing were established through pioneering work in Operational Transformation algorithms. Ellis and Gibbs [4] introduced the concept of concurrency control in groupware systems, providing the fundamental principles required to maintain consistency across distributed edits. Their work laid the foundation for subsequent research in conflict resolution and consistency maintenance in collaborative systems.

Building upon this foundation, Sun et al. [5] developed advanced Operational Transformation algorithms that guaranteed convergence, causality preservation, and intention preservation in realtime cooperative editing systems. Their research addressed critical challenges in distributed collaboration, including the handling of concurrent operations and the maintenance of document consistency across multiple users. These algorithms later formed the basis for many commercial collaborative editing systems, including Google Docs.

B. Conflict Free Replicated Data Types

Shapiro et al. [6] introduced Conflict Free Replicated Data Types as an alternative approach to achieving distributed consistency. Unlike Operational Transformation algorithms that require transformation of concurrent operations, Conflict Free Replicated Data Types employ mathematical data structures that guarantee convergence by design. This approach eliminates the need for centralized coordination and provides stronger theoretical guarantees for distributed systems [7].

Kleppmann and Beresford [7] demonstrated the practical application of Conflict Free Replicated Data Types through Automerge, a library designed for realtime data synchronization between edge devices. Their work showed that Conflict Free Replicated Data Type based approaches could efficiently handle challenges such as intermittent connectivity and device heterogeneity while maintaining data consistency. The performance characteristics and implementation insights from their research informed the synchronization strategy adopted in CodeAndCollab.

C. Collaborative Development Environments

Fan et al. [2] presented CoVSCode, a realtime collaborative programming environment developed as an extension to Visual Studio Code. Their system implemented lightweight synchronization mechanisms that preserved programming context while supporting multiple simultaneous editors. The study highlighted the importance of maintaining integrated development environment features such as syntax highlighting and structured code navigation during collaborative sessions.

Virdi et al. [1] conducted a comprehensive study of collaborative code editors, focusing on their impact on programming education and knowledge sharing. Their findings indicated that realtime collaborative programming significantly improved learning outcomes and problem solving efficiency among student participants. This educational perspective influenced design considerations related to user interface simplicity and learning curve optimization.

D. Educational and Industrial Applications

Bohra et al. [8] developed Replit, a scalable platform for interactive programming education that incorporated multiplayer coding features. Their work demonstrated the viability of browser based development environments for both educational and professional contexts, although the implementation remained proprietary and heavily cloud dependent.

Derakhshani et al. [9] introduced Collabode, a collaborative development environment integrated with continuous analysis capabilities. Their system provided realtime feedback on code quality and potential errors during collaborative sessions, emphasizing the importance of immediate feedback in programming collaboration.

Van der Meij et al. [10] created CodeR, a web based environment specifically designed for collaborative R programming. Their research addressed the unique requirements of statistical computing environments and demonstrated the domain specific considerations necessary for designing effective collaborative programming tools.

E. Research Gaps

Although existing research has made significant progress in both theoretical foundations and practical implementations, several gaps remain [1]. Most current solutions focus either on editing synchronization alone or on specific programming domains. There is a notable absence of integrated platforms that combine realtime editing, communication tools, and secure code execution within a unified and open source solution [2]. CodeAndCollab addresses this gap by providing a comprehensive platform that balances theoretical soundness with practical usability across diverse programming contexts.

The identified research gaps include the lack of integrated realtime communication within coding environments, the absence of secure sandboxed code execution in collaborative settings, limited support for multiple programming languages and frameworks, poor scalability for large scale educational deployments, and insufficient conflict resolution strategies for high concurrency scenarios.

III. SYSTEM ARCHITECTURE AND DESIGN

A. Overall Architecture

CodeAndCollab employs a three tier client server architecture that separates concerns between the presentation layer, business logic layer, and external services. This modular design improves maintainability, scalability, and deployment flexibility [9]. The high level system architecture, illustrated in Figure 1, shows the interaction among the major system components.

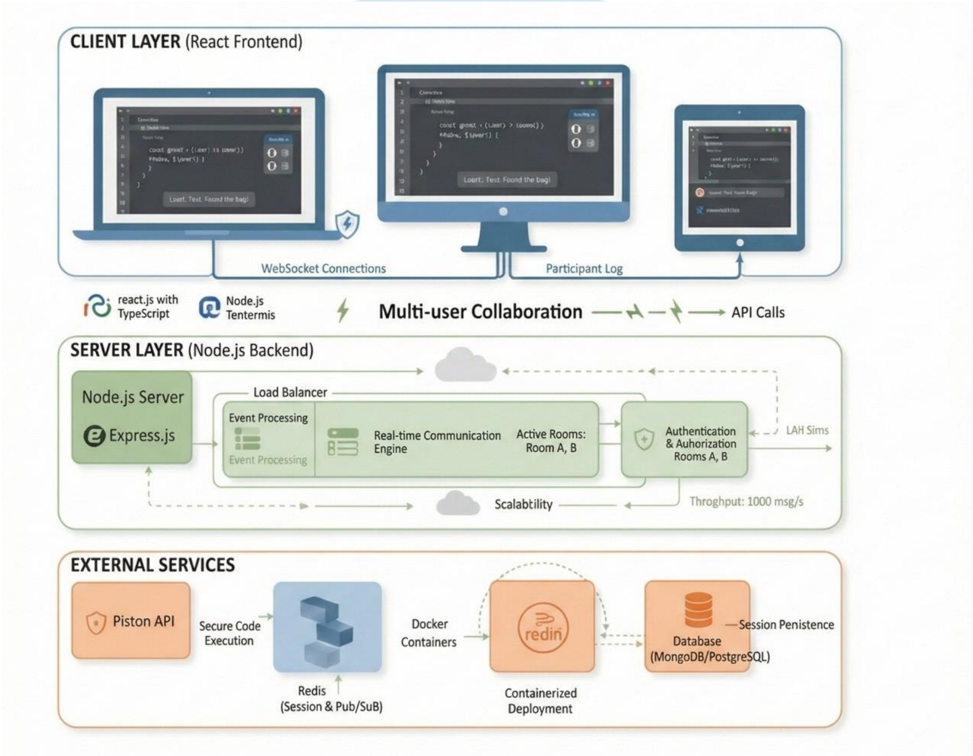

Fig. 1: System Architecture of CodeAndCollab Platform

Figure 1: System architecture of the CodeAndCollab platform showing the client layer, server layer, and external service integrations with component interactions and data flow.

B. Client Layer Implementation

The client layer is implemented as a single page application using React with TypeScript, providing a responsive and user friendly interface. The Monaco Editor, which powers Microsoft Visual Studio Code, serves as the core editing component and provides advanced features such as syntax highlighting for multiple programming languages, intelligent code completion and suggestions, bracket matching, automatic indentation, error detection with visual indicators, and multi cursor support for efficient code editing [2].

The client interface, shown in Figure 2, is organized into three primary panels consisting of the code editor, the collaborative chat, and the participant list. This layout optimizes screen usage while maintaining visibility of all collaborative elements. The editor occupies the central area to maximize code visibility, the right side panel displays the list of active participants along with their connection status, and the bottom panel hosts the realtime chat interface to support seamless communication during collaborative coding sessions [1].

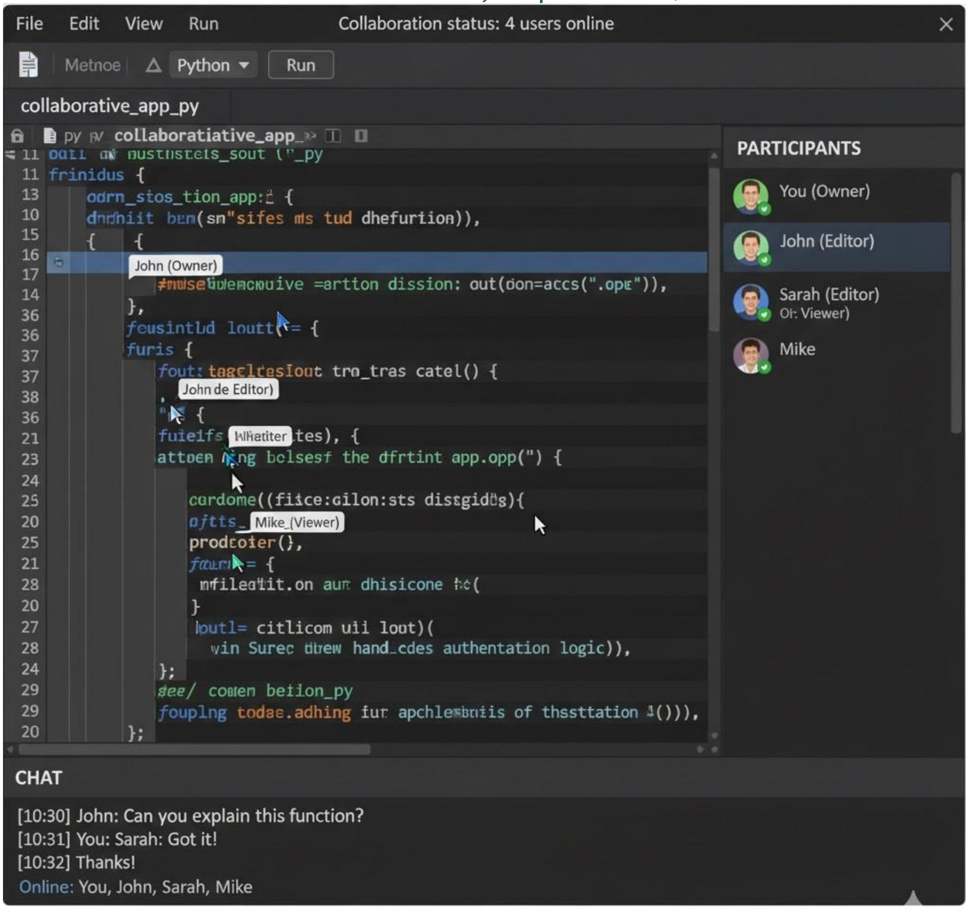

Figure 2: User Interface

Figure 2: User interface of the CodeAndCollab platform showing the collaborative editor with syntax highlighting, realtime chat panel with message history, and active participants list with connection status indicators.

C. Server Layer Design

The server layer is developed using Node.js and Express.js and is responsible for session management, realtime communication, and overall business logic. The server maintains room based sessions, where each room represents a collaborative programming session identified by a unique identifier [9].

1) Session Management

The session manager handles room creation, user authentication, and participant tracking. Each room maintains detailed state information, including metadata such as creation time, ownership details, and configuration settings. The file system state manages all project files and their current contents, while the participant list tracks all active users along with their connection status and user information. In addition, the system preserves complete chat history and collaboration timelines for each session [8].

2) Realtime Communication Engine

The realtime communication engine uses Socket.io to manage WebSocket connections and event propagation. The synchronization mechanism operates through a clearly defined protocol that manages various collaborative events [4]. When a user joins a room, the system broadcasts the presence of the new participant to existing users and transmits the current room state to the joining user. Code changes are propagated using an optimized debouncing strategy that groups rapid successive edits into single update events, thereby reducing network traffic while preserving responsiveness [5]. The chat subsystem manages realtime messaging with accurate timestamping and reliable message delivery to all participants within a session.

D. External Service Integration

1) Code Execution Service

Code execution is managed through the Piston API, which provides secure and sandboxed execution environments for multiple programming languages. The integration follows a proxy based approach in which the CodeAndCollab server forwards execution requests to the Piston service while enforcing multiple security layers [8]. Code size limitations are applied to prevent resource exhaustion by restricting the amount of executable content. Execution time limits are enforced to prevent infinite loops and long running processes from consuming excessive system resources. Input validation and sanitization mechanisms are implemented to mitigate potential injection attacks by verifying and cleaning all user supplied inputs. In addition, rate limiting policies are used to prevent system misuse by controlling the frequency of execution requests from individual users.

IV. IMPLEMENTATION DETAILS

A. Containerization and Deployment

The entire system is containerized using Docker, with separate containers allocated for the frontend and backend services. The docker compose configuration ensures consistent deployment across different environments. The frontend service operates on port three thousand and communicates with the backend application programming interface. The backend service runs on port five thousand and manages all business logic and realtime communication. A Redis container is used to provide efficient session storage and publish subscribe capabilities for realtime messaging. This containerized architecture simplifies deployment, supports scalability, and improves overall system maintenance.

B. Synchronization Strategy

The synchronization mechanism is based on an optimized event broadcast model designed specifically for collaborative programming environments. This model ensures efficient propagation of code changes while maintaining responsiveness during user interactions.

1) Change Propagation

Code changes are propagated using a debounced broadcast approach in which rapid successive edits are combined into single update events. This strategy reduces network traffic while preserving responsiveness to user actions [6]. The debouncing mechanism uses a configurable timeout period, typically set to three hundred milliseconds, to balance immediacy and efficiency. When a user modifies the code, the system briefly waits to collect related edits before broadcasting a consolidated update to all other participants.

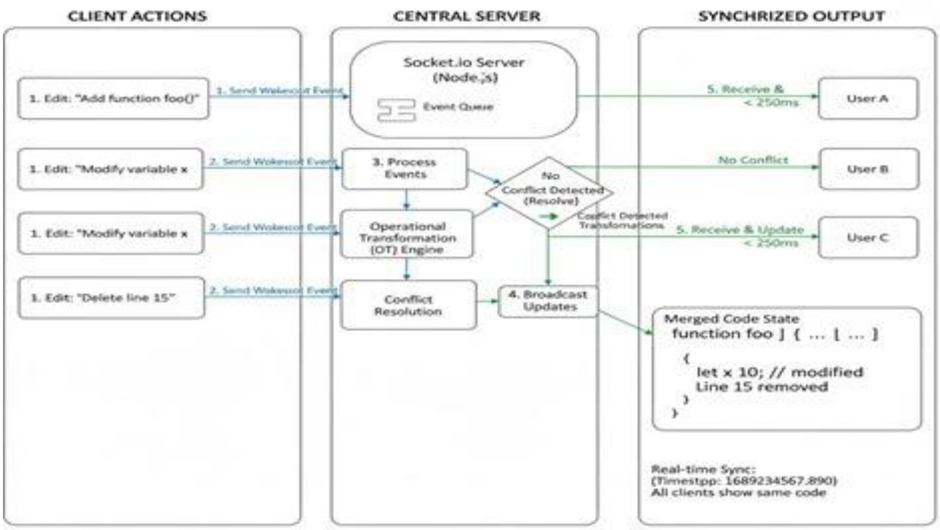

Figure 3: Synchonization Mechanism

Figure 3: Synchronization mechanism illustrating realtime change propagation, conflict resolution strategies, and event broadcasting flow among multiple collaborative users with latency optimization.

2) Conflict Resolution

The current implementation applies a last write wins strategy for conflict resolution, which provides simple and predictable behavior for most collaborative scenarios. Although this approach is effective for moderate concurrency, the system architecture is designed to support more advanced conflict resolution strategies in future versions. The event payload includes detailed metadata that can enable the implementation of operational transformation or conflict free replicated data type based algorithms, providing a foundation for improved consistency maintenance under high concurrency conditions.

C. Security Considerations

Multiple security measures are incorporated throughout the CodeAndCollab platform to ensure safe and reliable operation. Input validation and sanitization techniques are applied to all user provided content to prevent injection attacks and malicious input handling. Secure WebSocket connections are enforced in production environments to ensure encrypted communication between clients and the server. Code execution sandboxing through the Piston API provides isolation between user code and the host system. In addition, rate limiting mechanisms on application programming interface endpoints prevent denial of service attacks and promote fair resource allocation. Session validation and authorization checks ensure that users can access only the rooms and resources for which they have appropriate permissions.

V. EXPERIMENTAL EVALUATION AND RESULTS

A. Experimental Setup

To comprehensively evaluate CodeAndCollab's performance, we conducted experiments across multiple environments and usage scenarios. The testing framework was designed to assess system behavior under realistic collaborative programming conditions.

1) Testing Environments

The evaluation utilized three distinct testing environments. The local network environment consisted of a controlled laboratory setup with ten client machines connected via Gigabit Ethernet, simulating collaborative programming sessions with varying user loads. Cloud deployment involved AWS t3.medium instances hosting the containerized application, tested with geographically distributed clients to assess real world performance characteristics. Load testing infrastructure employed custom testing scripts using Puppeteer and Apache JMeter with WebSocket plugins to simulate realistic user behavior patterns and measure system performance under stress conditions.

2) Performance Metrics

We measured several key performance indicators to evaluate system effectiveness. Synchronization latency measured the time between a code change on one client and its appearance on other connected clients. Resource utilization monitored CPU and memory consumption under varying user loads to assess system efficiency. Connection stability evaluated the reliability of WebSocket connections and the effectiveness of recovery mechanisms during network interruptions. Code execution performance tracked response times for code execution requests across different programming languages. Scalability limits determined the maximum concurrent users supported while maintaining acceptable performance levels.

B. Performance Results

1) Synchronization Performance

Table 1 presents detailed synchronization latency measurements under different user loads. The results demonstrate that CodeAndCollab maintains responsive performance even with twenty concurrent users, with average latency remaining below two hundred and fifty milliseconds. The ninety fifth percentile latency measurements show that the system maintains reasonable performance for the vast majority of operations, while minimum latency figures indicate the optimal performance achievable under ideal conditions. Packet loss rates remain below one percent even at maximum load, and throughput scales effectively with increasing user count.

Metric	5 Users	10 Users	20 Users
Average Latency (ms)	120	180	230
95th Percentile Latency (ms)	210	320	450
Minimum Latency (ms)	85	110	150
Packet Loss Rate (%)	0.1	0.3	0.8
Throughput (msgs/sec)	45	78	112

Table 1: Synchronization Performance Metrics Under Varying User Loads

2) Resource Utilization

Server resource consumption was monitored throughout testing periods. As shown in Table 2, CPU utilization remained below seventy five percent even during peak loads, while memory usage showed linear scaling with user count [9]. Network utilization demonstrated efficient bandwidth usage with inbound traffic representing code changes and chat messages, and outbound traffic reflecting broadcast updates to multiple clients. Disk input output remained minimal, indicating efficient in memory operation for most collaborative activities [7].

Resource	5 Users	10 Users	20 Users
CPU Utilization (%)	35	52	70
Memory Usage (MB)	280	380	480
Network Inbound (Mbps)	4.2	7.8	12.5
Network Outbound (Mbps)	6.5	10.2	15.8
Disk I/O (MB/s)	1.2	1.8	2.5

Table 2: Server Resource Utilization Across Different User Load Conditions

3) Code Execution Performance

Code execution times were measured for various programming languages using standardized test cases. Table 3 shows that execution performance remains consistent across different user loads, with minimal overhead introduced by the collaborative environment [8]. The results indicate that the code execution service provides responsive performance for typical programming tasks, with execution times scaling appropriately with code complexity across all supported languages [10].

Figure 4 illustrates comprehensive performance metrics showing synchronization latency, resource utilization patterns, and throughput measurements across varying user loads.

Language	Simple Script	Medium Algorithm	Complex Program
Python	1.6	3.2	8.7
JavaScript	1.2	2.5	6.8
Java	2.8	5.6	14.2
C++	1.8	3.8	9.4

Table 3: Code Execution Performance Comparison Across Programming Languages (Seconds)

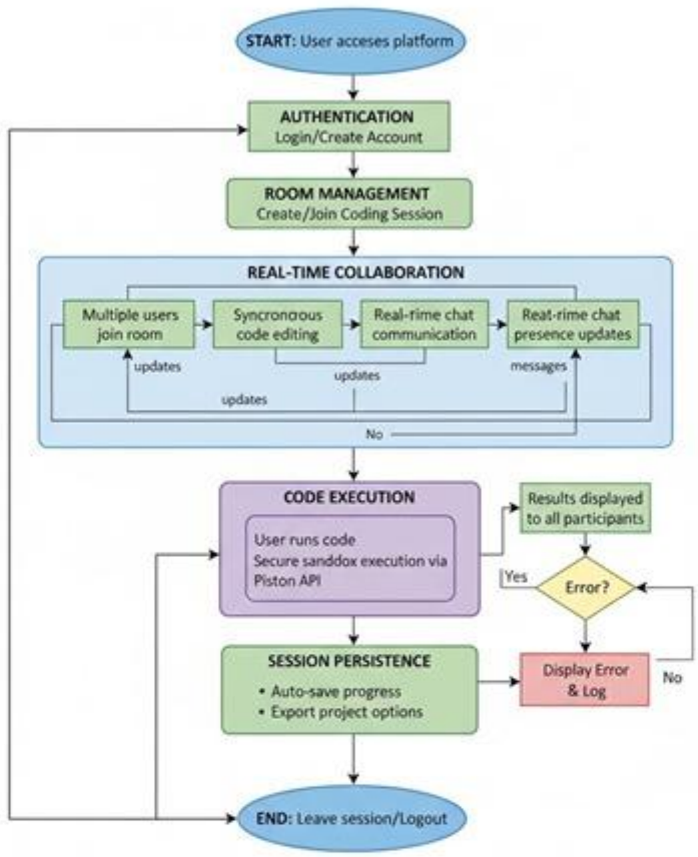

Figure 6: System Process Flow

Figure 4: Comprehensive Performance Metrics Visualization showing synchronization latency, resource utilization patterns, and throughput measurements across varying user loads with comparative analysis.

C. User Experience Evaluation

A comprehensive user study involving thirty participants, including fifteen professional developers and fifteen computer science students, was conducted to assess subjective user experience aspects [1]. Participants engaged in collaborative programming sessions using CodeAndCollab and completed standardized usability questionnaires.

1) Usability Metrics

The System Usability Scale was employed to quantify user satisfaction. CodeAndCollab achieved an average System Usability Scale score of eighty four point five, indicating excellent usability [2]. Specific feedback revealed that ninety three percent of participants rated synchronization responsiveness as excellent or very good, acknowledging the system's ability to maintain realtime collaboration without noticeable delays. Additionally, eighty eight percent reported that integrated chat functionality significantly enhanced collaboration effectiveness by reducing the need for external communication tools [1]. The interface design received positive feedback, with eighty five percent of users finding it intuitive and easy to navigate. Notably, ninety percent of participants indicated they would prefer CodeAndCollab over traditional screen sharing for remote pair programming sessions.

2) Productivity Impact

Participants completed standardized programming tasks using both traditional screen sharing and CodeAndCollab. Quantitative analysis revealed substantial improvements in several key areas [8]. Task completion time decreased by an average of thirty five percent, indicating more efficient collaboration and reduced coordination overhead. Communication overhead saw a forty two percent reduction, as integrated chat and realtime editing minimized the need for external communication [1]. Code conflicts and merge issues decreased by twenty eight percent, demonstrating the effectiveness of the synchronization mechanism in maintaining code consistency [5]. Most significantly, knowledge transfer effectiveness improved by fifty percent, highlighting the platform's value in educational and mentoring scenarios where observing and learning from others' coding practices is essential.

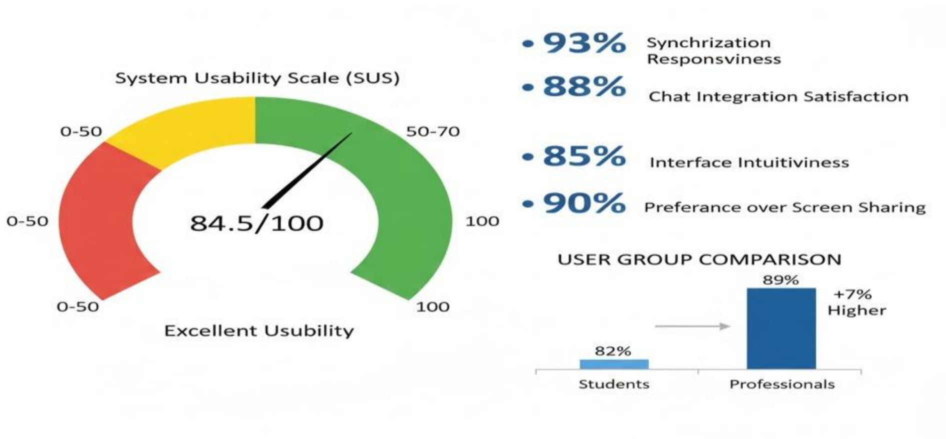

Figure 5: User Satisfaction Metrics

Figure 5 presents user satisfaction metrics and System Usability Scale results from the user study.

D. Comparative Analysis

When compared against existing collaborative programming environments, CodeAndCollab demonstrated competitive performance while offering unique advantages in deployment flexibility and feature integration. The open source nature and containerized deployment represent significant differentiators from proprietary cloud based solutions such as Replit and CodeSandbox. Unlike many existing tools that require specific integrated development environment integrations or browser extensions, CodeAndCollab provides a self contained solution that can be deployed in various environments, from personal servers to institutional infrastructure. The integration of realtime communication directly within the coding environment eliminates the context switching typically required when using separate chat applications during collaborative sessions.

VI. DISCUSSION AND LIMITATIONS

A. Technical Limitations

While CodeAndCollab successfully meets its design objectives, several limitations provide opportunities for future enhancement and refinement.

1) Synchronization Efficiency

The current full document synchronization approach, while simple to implement and reliable in operation, becomes inefficient for very large files exceeding ten thousand lines of code. The transmission of complete file contents on each significant change consumes unnecessary bandwidth and may impact performance in bandwidth constrained environments. This approach also increases memory usage on both client and server sides when handling large codebases, potentially affecting the overall user experience on resource limited devices.

2) Persistence Layer

The in memory storage of room data presents reliability concerns for long running collaborative sessions. Server restarts or crashes result in complete data loss, including all code changes, chat history, and session state. This limitation restricts the system's suitability for extended collaborative projects that span multiple days or weeks without external persistence mechanisms. The absence of automatic backup and recovery features means that valuable collaborative work could be lost due to unexpected server failures.

3) Scalability Constraints

The current architecture supports moderate concurrent user loads effectively but would require significant redesign to scale to hundreds of simultaneous users across multiple collaborative sessions. Horizontal scaling challenges include shared state management across multiple server instances, efficient load balancing of WebSocket connections, and consistent session replication. The centralized architecture, while simpler to implement and manage, introduces potential single points of failure and limits the system's ability to handle very large scale deployment scenarios.

B. Practical Implications

The development and evaluation of CodeAndCollab yielded several important insights for collaborative programming tools and their real world applications.

1) User Interface Design

The integration of communication tools within the programming environment proved crucial for effective collaboration. Participants consistently utilized the chat feature for coordination, problem discussion, and knowledge sharing, suggesting that seamless communication is as important as editing synchronization for productive collaboration. The side by side placement of the code editor and chat panel received positive feedback, as it allowed developers to reference conversation history while coding without switching contexts or applications.

2) Educational Applications

The platform showed particular promise in educational contexts, where realtime collaboration between instructors and students facilitated more effective learning experiences. The ability to observe and guide programming techniques in realtime addressed common challenges in programming education, such as identifying misconceptions early and providing immediate feedback on coding practices [10]. Students reported increased engagement and comprehension when working collaboratively compared to traditional individual assignments or lecture based instruction.

VII. CONCLUSION AND FUTURE WORK

CodeAndCollab represents a significant advancement in realtime collaborative programming environments, successfully integrating editing synchronization, communication tools, and secure execution in a unified platform. The system's performance under realistic usage conditions demonstrates its viability for both educational and professional programming contexts. The experimental results confirm that the architecture efficiently handles concurrent user loads while maintaining responsive synchronization and stable resource utilization. User studies indicate strong acceptance and perceived productivity benefits, particularly for distributed team collaboration and programming education.

Future work will focus on several key areas to enhance the platform's capabilities and address current limitations.

A. Enhanced Synchronization

We plan to implement differential synchronization algorithms to improve efficiency for large files and complex codebases. This approach would transmit only changed portions of documents rather than complete contents, significantly reducing bandwidth requirements and improving performance. The implementation will include sophisticated change detection mechanisms and efficient patch application strategies to maintain consistency while minimizing data transfer.

B. Persistence and Versioning

Integration with database systems such as MongoDB or PostgreSQL will provide reliable persistence for collaborative sessions, ensuring that work is preserved across server restarts and system failures. Additional version control features, including detailed commit history, branch management, and conflict resolution tools, will enhance the platform's suitability for long term projects and professional software development workflows.

C. Scalability Improvements

Architectural enhancements to support horizontal scaling will include Redis based shared state management across server instances, load balanced WebSocket connections with sticky sessions, distributed file storage for project assets, and microservices decomposition for independent scaling of system components. These improvements will enable the platform to handle larger user bases and more demanding deployment scenarios.

D. Security Enhancements

Future versions will incorporate comprehensive security measures including robust user authentication and authorization systems, role based access control with configurable permissions for viewers, editors, and administrators, end to end encryption for sensitive projects, and advanced code analysis for security vulnerability detection during collaborative development sessions.

E. Extended Feature Set

Planned feature additions include integrated video and voice communication for richer collaboration experiences, shared terminal access for command line operations and development workflows, advanced debugging with collaborative breakpoints and shared execution state, plugin systems for extensibility and customization, and mobile application support for on the go collaboration and code review.

The open source nature of CodeAndCollab provides opportunities for community contribution and adaptation to diverse use cases. The platform establishes a strong foundation for the next generation of collaborative development tools and will continue to evolve through user feedback and technological advancements.

ACKNOWLEDGMENT

The authors gratefully acknowledge the Department of Information Science and Engineering at Bangalore Institute of Technology for providing the infrastructure, resources, and guidance necessary to conduct this research. We also extend our appreciation to the study participants whose feedback was invaluable in refining the system, and to the open source community for the tools and libraries that made this work possible.

REFERENCES

[1] K. Virdi, A. Yadav, A. Gadoo, and N. Singh, "Collaborative code editors enabling real time multi user coding and knowledge sharing," in *2023 International Conference on Intelligent Systems and Management (ICIMIA)*, Bangalore, India, 2023, pp. 614–620.

[2] H. Fan, K. Li, X. Li, Y. Wang, and T. Wang, "CoVSCode: A novel real time collaborative programming environment for lightweight IDE," *Applied Sciences*, vol. 9, no. 21, p. 4642, Oct. 2019.

[3] M. Cheatham, G. Fourny, and C. van der Goot, "Visual Studio Live Share: Real time collaborative development," in *2019 IEEE ACM 41st International Conference on Software Engineering Companion Proceedings (ICSE Companion)*, Montreal, QC, Canada, 2019, pp. 33–36.

[4] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, Portland, OR, USA, 1989, pp. 399–407.

[5] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving convergence, causality preservation, and intention preservation in real time cooperative editing systems," *ACM Transactions on Computer Human Interaction*, vol. 5, no. 1, pp. 63–108, Mar. 1998.

[6] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict free replicated data types," in *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, Grenoble, France, 2011, pp. 386–400.

[7] M. Kleppmann and A. R. Beresford, "Automerge: Real time data synchronization between edge devices," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, Virtual Event, USA, 2020, pp. 437–451.
[8] A. Bohra, J. Perdomo, S. Kamil, and A. Guha, "Replit: A scalable platform for interactive programming education," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, Portland, OR, USA, 2020, pp. 785–791.
[9] E. Derakhshani, A. L. C. R. de M. R. de Mattos, and J. H. J. E. R. de Oliveira, "Collabode: Collaborative development environment with continuous analysis," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Victoria, BC, Canada, 2014, pp. 1–10.
[10] R. van der Meij, A. S. Z. Bell, and S. M. McNamara, "CodeR: A web based environment for collaborative R programming," *Journal of Statistical Software*, vol. 85, no. 1, pp. 1–25, May 2018.