International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 05 Issue: 04 | April – 2026 DOI:10.55041/ISJEM06375
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Codec: A Real-Time Collaborative Coding Platform with Multi-Language Execution

Heet Shah[1], Chirag Vanzara[2], Mrs. Sujaya Bhattcharjee[3], Mrs. Shubhangi Dhaygude[4]

[12] UG Students Department of Computer Science and Engineering, Parul Institute of Technology, Parul University, Vadodara, Gujarat , India

[3] Assistant Professor , Department of Computer Science and Engineering, Parul Institute of Technology, Parul University, Vadodara, Gujarat , India

[4] Lecturer , Department of Computer Science and Engineering, Parul Institute of Technology, Parul University, Vadodara, Gujarat , India

---

1. ABSTRACT

Real-time collaboration has become essential in modern software development and programming education. Traditional workflows rely on separate tools for editing, execution, and communication, which increases complexity and reduces productivity. This project presents Codec, a web-based real-time collaborative coding platform that allows multiple users to write, compile, and execute code together in various programming languages such as C, C++, Java, and Python. The system provides a centralized environment that integrates a browser-based editor, multi-language execution engine, user authentication, room-based collaboration via invite links, and an in-built group chat.

Codec is implemented using HTML, CSS, and JavaScript for the frontend, Node.js and Express.js for the backend, MongoDB as a NoSQL database, and Socket.IO over WebSockets for low-latency synchronization. Data is organized into collections for users, rooms, messages, and code sessions. Real-time synchronization is handled by Socket.IO events that broadcast code and chat updates to all participants in a room, while REST APIs manage authentication, room operations, and data retrieval. The platform is deployed on Render for cloud accessibility. Initial testing shows that Codec offers an effective and user-friendly environment for collaborative coding, reducing setup overhead and improving communication in both educational and professional contexts.

Keywords: Real-Time Collaboration, Online Code Editor, WebSockets, Socket.IO, Multi-Language Execution, Cloud Deployment, MongoDB, Node.js, REST API, Collaborative Programming.

2. INTRODUCTION

Software development and programming education are increasingly collaborative and geographically distributed. Students and developers often work remotely on shared projects, yet they still rely heavily on desktop IDEs, separate communication tools, and manual file sharing. This fragmented workflow leads to difficulties such as version confusion, lack of instant feedback, and high setup overhead for new participants.

Real-time collaborative coding platforms address these problems by allowing multiple users to edit and execute code within a shared, browser-based environment. Such systems

enable instructors to demonstrate concepts live, teams to debug code together, and interviewers to assess candidates in real time. However, many existing tools either focus on single-user online compilation or provide collaboration without integrated multi-language execution and structured session management.

The Codec platform is designed to bridge this gap. It offers a real-time collaborative editor with multi-language support, user authentication, room-based collaboration via shareable links, and an integrated chat interface. Built on widely used web technologies (HTML, CSS, JavaScript, Node.js, Express.js, MongoDB, and Socket.IO), Codec aims to be lightweight, accessible, and suitable for academic and project-based use. By centralizing coding, execution, and communication in a single interface, Codec simplifies the collaborative workflow and enhances learning and development experiences.

3. PROBLEM STATEMENT

Despite the availability of online IDEs and communication tools, several issues persist in collaborative programming scenarios:

- Lack of integrated tools: Teams often juggle between code editors, messaging applications, video conferencing, and terminals. This fragmentation causes frequent context switching and increases cognitive load.
- Inefficient sharing of code: Exchanging code via screenshots, file attachments, or version control commits for minor changes is slow and error-prone.
- Limited real-time visibility: Instructors and team members cannot always observe edits as they happen, making it difficult to guide, review, or debug collaboratively.
- Setup and environment challenges: Installing compilers, interpreters, and dependencies on each machine can be time-consuming and especially difficult for beginners.
- Inadequate session management: Many platforms lack structured room-based collaboration, persistent session data, and proper user roles.

Therefore, there is a need for a web-based real-time collaborative platform that:

1. Supports multi-user synchronous editing.
2. Offers multi-language code compilation and execution.
3. Integrates communication features such as group chat.
4. Provides secure authentication and session-based room management.
5. Is easily accessible via the browser without complex installation.

Codec is designed to satisfy these requirements through a modular, scalable system architecture.

4. OBJECTIVES

The main objectives of the Codec project are:

1. To simplify collaborative coding for students and developers by providing a browser-based platform that supports simultaneous editing and execution of code.
2. To create a unified environment that combines code editing, multi-language compilation and execution, and communication through integrated chat.
3. To enable real-time synchronization of code and messages among multiple users using WebSockets and Socket.IO.
4. To ensure secure access and session management through email-based user authentication, session tokens, and room-level access control using invite links.
5. To design a scalable backend using Node.js/Express.js and MongoDB, with collections for users, rooms, messages, and code sessions.
6. To deploy the platform to the cloud using Render, allowing users to access the system from any location with a web browser.

5. LITERATURE REVIEW

Advances in web technologies and real-time communication have enabled a new generation of collaborative coding tools. This section summarizes related work in four key areas.

5.1 Online Coding and Execution Platforms

Several web-based IDEs provide code editing and execution directly in the browser. These platforms often support multiple languages and server-side execution sandboxes. They reduce setup time and make it easier to experiment with code. However, in many cases, collaboration features are limited or restricted to asynchronous sharing of links and code snippets, rather than live multi-user editing.

5.2 Real-Time Collaborative Editors

Research on collaborative editing has produced algorithms such as Operational Transformation (OT) and Conflict-Free Replicated Data Types (CRDTs), used in tools like Google Docs and Etherpad. These systems demonstrate that real-time, low-latency multi-user editing is feasible at large scale. Yet, most of these editors are optimized for text documents, not for structured source code, and they typically do not integrate code execution and debugging.

5.3 Educational and Collaborative Programming Tools

Educational coding platforms and pair-programming environments have emerged to support remote teaching and learning. Some tools integrate code execution with chat or video conferencing, allowing instructors to observe student progress in real time. Studies indicate that such tools improve engagement, reduce communication overhead, and foster collaborative problem solving. Still, many are specialized for particular courses or languages and are not easily adaptable for general-purpose team development.

5.4 Real-Time Web Technologies

WebSockets and related frameworks like Socket.IO have become standard for real-time web applications, including chat systems, collaborative whiteboards, and multiplayer games. Socket.IO simplifies building event-driven applications with features such as automatic reconnection, room-based communication, and event broadcasting. These technologies are well-suited for synchronizing code and chat messages in a collaborative editor.

5.5 Identified Research Gap

Existing solutions either emphasize powerful, single-user IDE capabilities or strong real-time collaboration without integrated execution and structured session management. There is comparatively less focus on simple, educationally oriented platforms that unify collaborative editing, multi-language execution, authentication, and chat within a clean architecture. Codec is designed to fill this gap by providing a focused, extensible system based on standard web technologies.

6. PROPOSED SYSTEM

The proposed system, Codec, is a real-time collaborative coding platform accessible through a web browser. It enables multiple authenticated users to join a shared coding session (room) and:

- Edit source code together in real time.
- Compile and execute code in multiple languages (C, C++, Java, Python).
- Communicate via a group chat associated with each room.
- Manage access through shareable invite links.

The system organizes its data in MongoDB collections: users (stores user accounts, credentials, and profile metadata), rooms (stores room information, owner, and configuration details), messages (stores chat messages linked to rooms and users), and code_sessions (stores current and historical code states per room). The backend offers REST APIs for authentication and room management, while Socket.IO channels handle real-time events. The entire application is hosted on Render, ensuring high availability and easy access.

7. SYSTEM ARCHITECTURE

7.1 Architectural Overview

Codec follows a multi-tier architecture consisting of the following layers:

1. Presentation Layer (Frontend) - Implemented with HTML, CSS, and JavaScript. Provides user interfaces for login/sign-up, dashboard, room selection, editor, and chat.
2. Application Layer (Backend) - Implemented with Node.js and Express.js. Handles business logic, request routing, authentication, and integration with Socket.IO and the execution engine.
3. Real-Time Communication Layer - Powered by Socket.IO over WebSockets. Manages event-based interactions for code updates, chat messages, and user join/leave events.
4. Data Layer (MongoDB) - Manages persistence of user, room, message, and code session data.
5. Execution Engine Layer - Accepts code and language selection requests from the backend, compiles or interprets the code in a controlled environment, and returns the result.
6. Cloud Infrastructure - Render hosts the backend and serves the frontend. MongoDB is hosted as a managed service.

7.2 Logical Workflow

1. User registers and logs in via REST APIs.
2. User creates or joins a room; room data is stored in rooms collection.
3. Client connects to Socket.IO and joins the corresponding room channel.
4. Code edits and chat messages are broadcast to all clients in that room.
5. Execution requests are forwarded to the execution engine, and results are pushed back.
6. Code snapshots and messages are periodically recorded in code_sessions and messages collections.

8. TECHNOLOGIES USED

8.1 HTML, CSS, JavaScript

HTML structures the user interface components such as forms, editor container, chat panel, and navigation. CSS ensures responsive layouts and clean styling for improved user experience. JavaScript handles dynamic behaviors including form validation, Socket.IO event handling, DOM updates, and REST API integration.

8.2 Node.js and Express.js

Node.js provides an event-driven runtime, ideal for handling many concurrent connections. Express.js simplifies the creation of RESTful APIs, middleware integration, and routing for endpoints such as /api/auth/signup, /api/auth/login, /api/rooms (create/get

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 05 Issue: 04 | April – 2026 DOI:10.55041/ISJEM06375
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

rooms), and /api/rooms/:id/history (fetch code or message history).

8.3 MongoDB

MongoDB stores structured JSON-like documents across four collections: users (fields: _id, email, passwordHash, createdAt), rooms (fields: _id, name, ownerId, inviteToken, createdAt), messages (fields: _id, roomId, senderId, content, timestamp), and code_sessions (fields: _id, roomId, language, sourceCode, updatedAt, version). Indexes are created on email in users and on roomId in messages and code_sessions for efficient queries.

8.4 Socket.IO / WebSockets

Socket.IO abstracts low-level WebSocket details, providing event-based communication (e.g., code_change, chat_message, run_code), room/group semantics for broadcasting events only to relevant participants, and automatic reconnection and fallback mechanisms.

8.5 Cloud Deployment (Render)

Render is used to deploy the backend and serve the frontend. Configuration involves defining build and start commands for the Node.js application, setting environment variables for MongoDB URI and authentication secrets, and enabling HTTPS and monitoring logs for performance and error analysis.

9. METHODOLOGY

9.1 Requirements Analysis

Key functional requirements include user registration and login via email, creation of collaboration rooms and management via invite links, real-time code editing and execution for multiple languages, and real-time textual communication through group chat. Non-functional requirements include low latency for real-time synchronization, secure handling of user credentials and code execution, and scalability to support multiple rooms and users.

9.2 System Architecture Design

Use case diagrams, activity diagrams, and sequence diagrams were prepared to describe how users interact with the system and how components communicate internally. The architecture was divided into modules such as authentication, room management, real-time synchronization, and execution services.

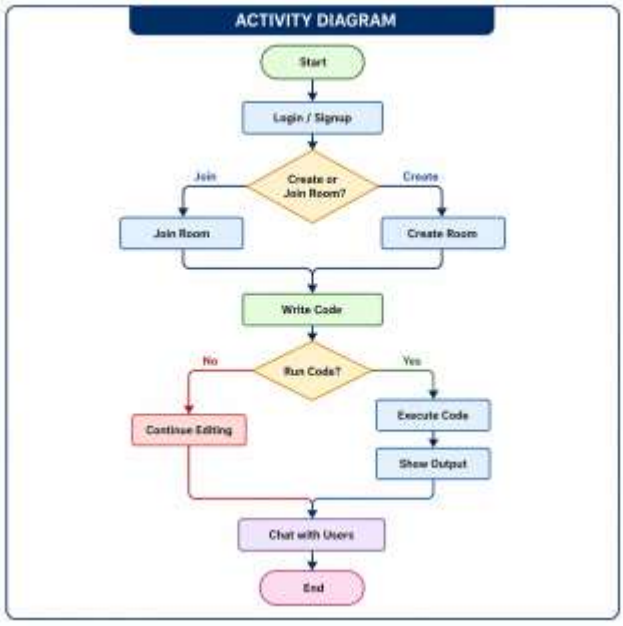

Fig -1 : Activity Diagram

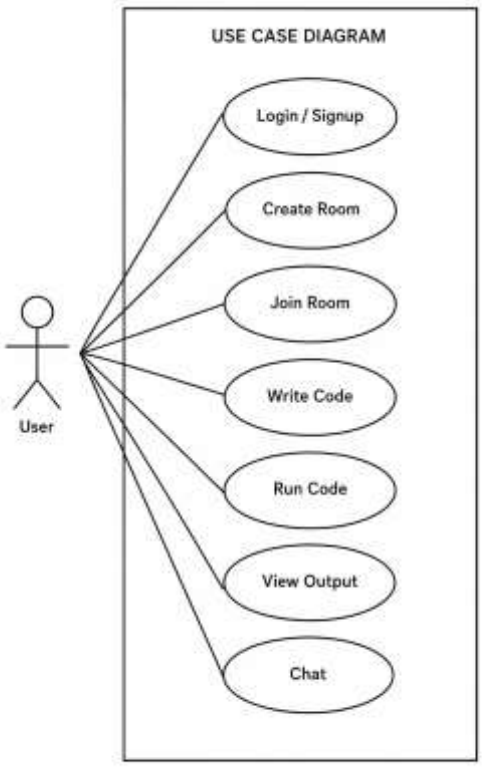

Fig - 2 : Use Case Diagram

9.3 Real-Time Data Flow and Socket.IO Events

The real-time operation relies on a well-defined set of Socket.IO events: join_room (client sends room ID and user token; server validates and adds client to Socket.IO room), code_change (emitted when a user edits the code; carries change details or full content; server broadcasts to other room members), cursor_update (shares cursor position), chat_message (transmits chat text; server stores it and broadcasts), run_code (sends the code, language, and optional input to the backend for execution), and run_result (returns execution output or error back to all room participants).

9.4 Handling Concurrent Edits and Consistency

To maintain consistency among multiple clients, the editor sends small, frequent updates so that each user's view stays close to the latest global state. The server maintains the latest canonical version of the code for each room in memory and periodically persists it to code_sessions. For initial versions, a last-write-wins policy is applied at the event level. For future upgrades, more advanced strategies like OT/CRDT can be introduced for fine-grained conflict resolution.

9.5 MongoDB Interactions

User lifecycle operations include inserting a new document into users with hashed password on sign-up and comparing provided password with stored hash on login. Room lifecycle operations include inserting a document into rooms with owner and generated invite token on room creation and validating invite token or room ID on join. Chat persistence involves saving message documents to messages when chat_message is received and fetching recent messages on entering a room. Code sessions are saved as snapshots periodically or on important events, and the latest code session is fetched when a user joins a room.

9.6 REST API Design

Representative endpoints include: POST /api/auth/signup (register user with validation and password hashing), POST /api/auth/login (authenticate and issue token), POST /api/rooms (create room, authorized), GET /api/rooms (list rooms owned or joined by user), and GET /api/rooms/:id/history (retrieve latest code session and recent messages). These endpoints return JSON responses and employ middleware for token validation.

9.7 Secure Authentication and Session Management

Security measures include passwords stored as salted hashes, JWT or session tokens issued upon successful login, token verification middleware for protected REST routes and Socket.IO connections, validation and sanitization of all external inputs to defend against injection attacks, and resource limits on the execution engine to prevent abuse (time, memory, and I/O restrictions).

9.8 Deployment Workflow

1. Source code is managed in a Git repository.
2. Render is configured to pull from the repository and build the Node.js application.
3. Environment variables are set for MongoDB URI and secret keys.
4. After deployment, smoke tests verify REST and WebSocket endpoints.
5. Logs are monitored to identify performance bottlenecks and runtime errors.

10. RESULTS AND DISCUSSION

Prototype deployment and testing with small user groups produced the following observations:

- Real-Time Collaboration: Users could see each other's code edits almost instantly, allowing efficient pair programming and group debugging.
- Ease of Onboarding: New participants joined sessions via invite links and standard browsers, with no installation of compilers or IDEs required.
- Effective Communication: Integrated chat kept discussions and decisions closely tied to the code under development.
- Multi-Language Execution: Sample programs in C, C++, Java, and Python executed successfully, demonstrating the correctness of the execution pipeline.

Some challenges were noted: when several users edited the exact same line rapidly, minor flickering or overwriting could occur, indicating the limitations of simple concurrency handling in highly contested areas of the code. Execution time increased for larger code snippets or compute-heavy operations, stressing the need for improved resource management and possibly dedicated execution servers. Overall, Codec met its primary objectives of providing an accessible, real-time collaborative environment for coding and learning.

11. ADVANTAGES

- Centralized Workflow: Users can edit, execute, and discuss code without leaving the browser.
- Real-Time Synchronization: Socket.IO ensures low-latency updates for both code and chat messages.
- Multi-Language Support: The platform supports popular programming languages, making it suitable for varied coursework and projects.
- Scalability and Flexibility: Node.js and MongoDB enable easy scaling to support more rooms and users.
- Accessibility: Cloud deployment on Render allows users to access Codec from any device with internet connectivity.
- Educational Suitability: Instructors can observe and guide students in real time, making it valuable for labs and live coding sessions.

12. LIMITATIONS

- Basic Conflict Resolution: Current concurrency control is simple and may not handle heavy simultaneous editing gracefully.
- Execution Resource Constraints: The shared execution environment may struggle with very resource-intensive tasks.
- Limited Advanced IDE Features: Features like step-by-step debugging, refactoring, and advanced auto-completion are minimal compared to full desktop IDEs.
- Dependency on Network Quality: Poor network conditions can impact real-time synchronization quality.

13. FUTURE SCOPE

- Advanced Concurrency Algorithms: Implement OT or CRDT-based algorithms to provide robust conflict resolution.
- Richer Tooling: Add debugging tools, linting, code completion, and integration with external repositories (e.g., GitHub).
- Role-Based Access Control: Introduce roles such as instructor, presenter, or viewer to better support classrooms and workshops.
- Analytics and Monitoring: Provide metrics on participation, code changes, and session time for instructors and team leads.
- Mobile and Tablet Optimization: Refine the UI for touch interfaces and smaller screens.
- Dedicated Execution Infrastructure: Use containers or serverless functions to isolate executions and scale language runtimes independently.

14. CONCLUSION

Codec demonstrates that modern web technologies can be combined to create an effective real-time collaborative coding platform with multi-language support. By integrating a browser-based editor, multi-language execution, authentication, room-based collaboration, and group chat, Codec reduces dependence on separate tools and complex local setups. The use of Node.js, Express.js, MongoDB, Socket.IO, and cloud deployment on Render provides a solid, scalable foundation.

Initial use of Codec in small groups indicates improvements in collaboration efficiency, communication, and learning experience. While limitations remain in areas such as advanced concurrency handling and execution performance, the system serves as a robust base for future enhancements. Codec can be further extended for academic, professional, and training environments where real-time collaborative coding is essential.

15. REFERENCES

[1] A. Gupta and R. Verma, "Design of a Web-Based Real-Time Collaborative Code Editor," International Journal of Computer Applications, vol. 180, no. 5, pp. 10-18, 2023.

[2] L. Zhang and M. Chen, "Operational Transformation Techniques for Collaborative Editing Systems: A Survey," Journal of Systems and Software, vol. 190, pp. 1-15, 2022.

[3] S. Kumar and P. Rao, "Cloud-Based Integrated Development Environments for Teaching Programming," International Journal of Engineering Education, vol. 37, no. 4, pp. 950-960, 2021.

[4] J. Lee and H. Park, "Real-Time Web Applications Using WebSockets and Node.js," Journal of Web Engineering, vol. 19, no. 2, pp. 123-140, 2020.

[5] D. Singh and N. Patel, "A Study on Collaborative Programming Tools for Remote Software Development," International Journal of Software Engineering and Knowledge Engineering, vol. 31, no. 3, pp. 415-432, 2021.