

TalentSync: A Real-Time Multi-Interview Platform for Technical Hiring

Mr. Satyam Das

Student, Dept. of CSE,
GIFT Autonomous, Bhubaneswar

Mr. Ratikanta Mohanty

Student, Dept. of CSE,
GIFT Autonomous, Bhubaneswar

Dr. Sujit Kumar Panda

Professor, Dept. of CSE,
GIFT Autonomous, Bhubaneswar

Abstract— TalentSync is a web-based multi-interview platform designed to simplify and improve the process of conducting technical and behavioural interviews in a remote environment. The platform enables candidates and interviewers to communicate and collaborate in real time through integrated video calling, live chat, and a collaborative code editor. The main objective of the system is to provide an interactive and structured interview experience that closely replicates a real-time face-to-face technical interview process.

In many existing online interview systems, organizations often depend on multiple applications for communication, coding assessments, and interview management. This creates difficulties such as communication gaps, inefficient coordination, and lack of proper collaboration during interviews. TalentSync addresses these issues by providing all essential interview features within a single centralized platform. The system allows users to create interview sessions, join meetings securely, communicate effectively, and solve coding problems together in real time.

The project is developed using the MERN stack, which includes MongoDB, Express.js, React.js, and Node.js. MongoDB is used for storing user and session data, Express.js and Node.js manage backend operations and APIs, while React.js provides a responsive and user-friendly interface. The platform is scalable, efficient, and suitable for organizations conducting multiple interviews simultaneously.

Keywords— Remote Interview, Real-Time Collaboration, WebSocket, Collaborative Code Editor, MERN Stack, Technical Hiring, Video Communication, Docker Sandboxing, Socket.IO, Interview Management.

I. Introduction

TalentSync is a web-based real-time interview platform designed to improve and simplify the technical hiring process for organizations and candidates. It integrates multiple interview functionalities such as video communication, live chat, and a collaborative code editor into a single system. The platform aims to provide a seamless and structured interview experience that closely replicates real-world technical interviews. It helps reduce dependency on multiple tools and ensures efficient coordination between interviewers and candidates in a remote environment. [1]

In traditional recruitment processes, conducting technical

interviews often requires multiple applications for communication, coding tests, and evaluation. This fragmented approach leads to inefficiencies, poor synchronization, and lack of real-time collaboration. Candidates and interviewers face difficulties in switching between tools, which affects interview quality. TalentSync addresses these issues by providing a unified platform where all interview-related activities can be performed in one place, ensuring smooth interaction and better evaluation of technical skills. [2]

The application provides key features such as multi-user collaboration, unique room-based sessions, syntax highlighting, and real-time updates. Users can join a coding session using a room ID and collaborate with others, making it highly useful for team projects, coding interviews, and online learning. Furthermore, the integration of Docker-based code execution ensures that programs are executed securely in isolated environments, preventing system-level vulnerabilities and enhancing reliability. [3]

The main objective of TalentSync is to create an interactive and scalable system for conducting technical and behavioral interviews online. It allows interviewers to create interview sessions, share coding problems, evaluate solutions in real time, and communicate effectively with candidates. The platform is designed to enhance the quality of hiring decisions by enabling structured assessment methods and real-time observation of candidate problem-solving approaches and coding practices. [4]

The rapid increase in remote work adoption following the global shift toward distributed teams has accelerated the demand for robust online interview tools. Organizations of all sizes — from startups to large enterprises — require efficient mechanisms to evaluate technical talent without the logistical overhead of in-person interviews. TalentSync was developed in response to this growing demand, providing a purpose-built solution that combines professional-grade tooling with an accessible, modern interface suitable for both technical and non-technical interview formats. [3]

This paper presents the design, architecture, implementation, and evaluation of TalentSync. Section II reviews existing approaches and highlights their shortcomings. Section III describes the proposed system architecture. Section IV covers the technology stack and tools adopted. Section V explains the implementation details. Section VI reports testing outcomes and results. Section VII discusses limitations, and Section VIII concludes the paper with future directions. [4]

II. Existing Approaches

The rapid growth of remote hiring technologies has significantly increased the importance of online interview platforms. Traditional interview systems mainly rely on manual farming techniques — interviewers use video conferencing platforms such as Zoom or Google Meet for communication, coding platforms such as HackerRank or LeetCode for assessments, and separate evaluation forms for feedback. Although these tools are individually useful, they often generate coordination overhead and disrupt the natural flow of a technical interview session. [5]

Asynchronous interview platforms such as HireVue allow candidates to record video responses to predefined questions, which interviewers review at a later time. While this model improves scheduling flexibility, it fundamentally lacks the live, interactive nature of a real technical interview. Candidates cannot ask clarifying questions, and interviewers cannot probe deeper based on live responses. This disconnect reduces the effectiveness of the assessment for roles that require dynamic problem-solving under real-time conditions, such as software engineering and data analysis positions. [6]

Collaborative coding platforms have become important components of modern technical assessment. Tools such as CoderPad and CodePair allow shared coding environments where interviewers and candidates can view and edit code simultaneously. However, many such tools lack integrated video communication and structured session management, requiring interviewers to switch between multiple applications during a single session. [6]

Automated coding assessment platforms such as HackerRank and Codility are widely used by large organizations for initial screening of candidates. These platforms present timed algorithmic problems and evaluate solution correctness and efficiency automatically. However, they are designed for unproctored, asynchronous testing and do not support live interaction between the interviewer and candidate. They also lack the flexibility to ask open-ended design questions or conduct behavioral interview segments within the same platform, making them unsuitable for comprehensive technical interview workflows. [5]

Real-time communication technologies such as WebSocket have advanced significantly. These enable persistent, bidirectional communication between clients and servers, providing the foundation for collaborative editing and live chat. However, most existing interview platforms do not fully leverage these technologies to provide a seamlessly integrated interview environment combining video, chat, and collaborative coding in one interface. [7]

Security and privacy concerns also represent a significant gap in many existing platforms. Candidate code submissions on generic online judges are typically stored indefinitely and may be accessible to other users, raising confidentiality concerns during proprietary hiring assessments. Furthermore, most freely available collaborative editors do not enforce access control at the session level, making it possible for unauthorized individuals to access or modify active sessions if the room link is shared externally. Organizations with strict data governance requirements therefore find it difficult to rely on such platforms

for sensitive hiring activities. [8]

The analysis of existing approaches reveals several key limitations:

- a. Fragmented tools for communication, coding, and evaluation
- b. Lack of real-time collaborative code editing within interview sessions
- c. Limited integration between video conferencing and coding environments
- d. Absence of Docker-based secure and isolated code execution
- e. Poor scalability for multiple concurrent interview sessions
- f. Insufficient role-based access control and session management

These limitations highlight the need for a centralized, scalable, and integrated platform that combines video communication, real-time collaborative coding, secure code execution, live chat, and structured session management. TalentSync addresses all these challenges within a modern MERN Stack architecture. [8]

A comparative review of popular platforms further reinforces the motivation for TalentSync. HackerRank and LeetCode offer automated coding assessments but lack live communication and real-time collaborative editing in an interview session context. Zoom and Google Meet provide reliable video conferencing but have no coding infrastructure built in. HireVue supports asynchronous video interviews with AI analysis but does not support live coding collaboration. Pramp enables peer-to-peer practice interviews with coding but is not designed for organization-administered structured hiring workflows. Table I summarizes this comparison, showing that no existing platform fully covers all the functional requirements that TalentSync is designed to fulfill in an integrated, real-time manner. [7]

III. Proposed System Architecture

The overall system architecture of TalentSync is designed using a modular client-server model that clearly separates presentation, application logic, and data storage. This separation improves scalability, maintainability, and performance. The React frontend handles the user interface, while the Node.js and Express backend manages APIs, sessions, and real-time communication. MongoDB is used to store persistent data such as users, interview sessions, and coding room details. [9]

The data flow within TalentSync follows a clearly defined request-response cycle. When a candidate submits a request to join an interview session, the frontend sends a REST API call to the backend with the session ID and user credentials. The backend verifies the JWT token, validates the session ID against the MongoDB database, and returns a session token that enables the client to join the Socket.IO room. Once connected, all subsequent interactions — code changes, chat messages, question pushes — are handled via Socket.IO events rather than HTTP requests, ensuring sub-100ms latency for most real-time operations under normal network conditions. [9]

The frontend communicates with the backend using REST APIs for standard operations and WebSocket connections for real-time features like collaborative coding and live updates. This dual communication model ensures both reliability and instant synchronization. Users can create interview sessions, join rooms, and interact through video, chat, and code editor without delays or page refreshes. Authentication and authorization are handled through Clerk Auth, which secures routes and manages user sessions with role-based access control. [10]

The proposed architecture consists of the following major layers:

- g. User Interface Layer — React.js 19.2 + Vite frontend
- h. Backend API Layer — Node.js + Express.js
- i. Real-Time Layer — Socket.IO + WebSocket
- j. Database Layer — MongoDB + Mongoose ODM
- k. Code Execution Layer — Docker Sandbox (Isolated Containers)
- l. External Services — Stream Video/Chat SDK, Clerk Auth, Monaco Editor

When an interview session starts, authenticated users join a coding room where WebSocket maintains live synchronization of code changes, chat messages, and session status. Video and chat services operate in parallel through Stream SDK integrations, ensuring continuous communication throughout the interview. The Docker-based code execution engine processes submitted code in isolated containers, returning output to the frontend without affecting the host server. [11]

The session lifecycle within TalentSync follows a well-defined flow. An interviewer creates a new session by specifying the interview title, candidate list, question set, and duration preferences. Upon creation, the system generates a unique session ID and distributes secure join links to each invited candidate. When the interviewer activates the session, all participants receive a real-time notification and are redirected to the interview room. The session remains active until the interviewer explicitly ends it or the predefined duration expires. All session metadata, including creation timestamps, participant actions, and code snapshots, are persisted in MongoDB for post-interview review. [9]

Security is a core concern in the architectural design of TalentSync. All API routes are protected using middleware that validates JWT tokens issued by Clerk Auth. Role-based access control differentiates between interviewer and candidate privileges — interviewers can add questions, observe multiple candidates simultaneously, and terminate sessions, whereas candidates can only interact within their assigned room. Communication between client and server is secured using HTTPS for REST calls and WSS (WebSocket Secure) for real-time events. Docker containers used for code execution are isolated with restricted CPU and memory quotas to prevent resource abuse and protect the host environment from malicious code submissions. [10]

IV. Technology Stack and Tools

React.js 19.2 is used to build a fast, interactive, and component-driven user interface for TalentSync. Its declarative approach helps structure complex views such as dashboards, coding panels, interview rooms, and evaluation screens. Vite is adopted as the frontend build tool to provide extremely fast development startup and hot module replacement, with optimized production builds for minimal bundle size. [12]

Node.js is used as the backend runtime environment, enabling fast and scalable server-side processing with its non-blocking, event-driven architecture suited for handling multiple concurrent interview sessions. Express.js is implemented as the web framework to design RESTful APIs and manage server routes, request handling, validation, error handling, and security controls efficiently. [13]

MongoDB is used as the primary database to store user profiles, interview sessions, coding room information, and chat messages. Its flexible document structure suits the dynamic interview data. Mongoose is integrated as the Object Data Modeling library to manage structured schemas and validations. The Stream Video SDK and Stream Chat SDK are integrated to provide low-latency real-time video conferencing and instant messaging during interview sessions. [14]

The Monaco Editor — the same engine that powers VS Code — is integrated to provide a professional coding environment. It supports syntax highlighting, auto-completion, bracket matching, and error detection for multiple languages. Real-time synchronization is achieved by combining Monaco Editor with Socket.IO. Clerk Auth manages secure sign-up, sign-in, and role-based session handling, supporting email/password login, OTP verification, and social sign-in options. [15]

Socket.IO is one of the central libraries enabling real-time bidirectional communication in TalentSync. It abstracts over the native WebSocket protocol and provides automatic fallback mechanisms for environments where WebSocket connections are restricted. Socket.IO rooms are used to isolate events per interview session, ensuring that code updates, chat messages, and question broadcasts are scoped to the correct participants. The library's built-in reconnection logic also improves resilience in unstable network conditions, allowing clients to seamlessly resume active sessions after a brief disconnection without data loss. [12]

Docker plays a critical role in the secure execution of candidate-submitted code. Each code execution request triggers the creation of a short-lived Docker container configured with the appropriate language runtime. The container is given a strict timeout to prevent infinite loops or resource exhaustion, and file system access is restricted to prevent data exfiltration or host-system interference. Currently, TalentSync supports JavaScript execution via Node.js containers, with planned extensions for Python, Java, and C++. The output of the container — including standard output and error streams — is captured and returned to the frontend for immediate display, giving candidates quick feedback on their solutions. [13]

V. System Design and Implementation

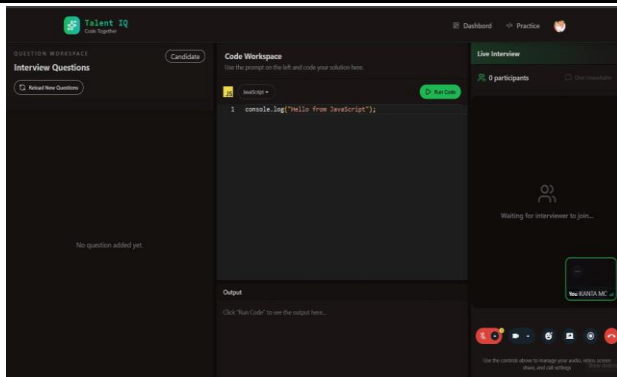


Fig. 1: Interviewer Dashboard showing interview sessions, candidates, and status management

TalentSync is implemented using a modular full-stack architecture with React (Vite) on the frontend and Node.js with Express.js on the backend. Real-time collaboration is implemented using Socket.IO, allowing candidates and interviewers to work together in a shared coding environment. When users join a session they are assigned to a unique room where code updates are instantly synchronized. The Monaco Editor captures changes and sends them to the backend, which broadcasts updates to all participants, ensuring a seamless live coding experience. [16]

Secure code execution is handled through Docker integration. When a user runs code, the backend creates a temporary file and forwards it to an isolated Docker container based on the selected language. The container compiles or interprets the code safely, preventing any impact on the host server. The execution result — including output, errors, and execution time — is returned to the frontend for display. The code execution flow proceeds as follows: (1) the candidate writes code in the Monaco Editor; (2) clicking Run sends the code and selected language to the backend via HTTP; (3) the backend detects the language and creates a temporary file; (4) the file is forwarded to a Docker container for isolated execution; (5) the result is sent back and displayed in the output panel. [17]

The frontend is organized into reusable components including login, dashboard, interview room, video panel, chat panel, and code editor. React Router manages navigation between pages, while the Context API handles global state for session data, user roles, and interview status. The backend follows a modular architecture separating controllers, routes, services, and middleware for clean code organization, easy debugging, and future extensibility. RESTful APIs handle room creation, session management, code execution requests, and user data operations. [18]

Authentication is integrated using Clerk Auth to securely manage user sign-up, login, sessions, and role-based access. Role-based access control ensures only interviewers can create sessions and only invited candidates can join interview rooms. Token validation and middleware checks protect sensitive data and prevent unauthorized access. The Stream SDK integration initializes video clients on the frontend, authenticates users, and establishes secure audio and video streams. Live chat is initialized within session rooms via Socket.IO, with messages displayed in a dedicated chat panel alongside the code editor

with timestamps for clarity. [19]

The collaborative code editor is one of the most technically complex components of TalentSync. Monaco Editor is embedded within a React component and configured to listen for local change events. On every keystroke or paste action, a debounced event handler emits the full editor content over Socket.IO to the server. The server then broadcasts this content to all other participants in the same room. To avoid echo loops, each client checks whether the incoming update originates from itself before applying it. This approach, while simpler than Operational Transformation, is effective for the low-latency, session-bound collaboration model used in technical interviews where only one active coder is typically expected at a time. [16]

The interview question management system allows interviewers to preload questions into a session before it begins. Questions are stored in MongoDB linked to the session document. During an active interview, the interviewer can select and push a question to the candidate's panel in real time. The candidate's Question Workspace panel updates instantly via a Socket.IO broadcast event, eliminating the need for manual communication of problem statements. This ensures clarity and reduces context-switching time for both parties. The system also supports reloading fresh questions dynamically without interrupting the ongoing session. [17]

The dashboard interface for interviewers provides an aggregated view of all interview sessions created under their account. Each session card displays the session title, status indicator (Scheduled, Active, or Completed), candidate count, question count, and planned duration. Interviewers can filter sessions by status using the tab controls, allowing efficient navigation between upcoming and historical interviews. The dashboard is implemented as a React component that fetches session data from the backend API on mount and refreshes in response to status change events broadcast over Socket.IO. This ensures the dashboard always reflects the current interview state without requiring manual page refresh. [18]

The candidate interface is designed to be simple and focused. Upon joining a session, the candidate sees a split-panel layout: the left panel displays the question workspace assigned by the interviewer, while the right panel contains the Monaco code editor and the output console below it. The video and chat panels are positioned on the right-hand side of the screen, providing continuous audio-visual communication without obscuring the coding area. This spatial organization was deliberately chosen to mirror the experience of in-person whiteboard-based coding interviews, where the problem statement is visible alongside the solution workspace at all times. [19]

VI. System Testing and Results

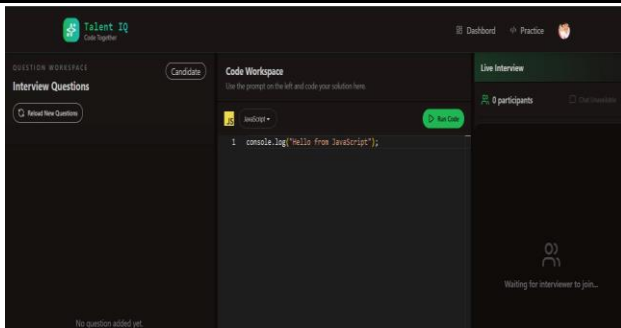


Fig. 2: Live Interview Code Workspace with collaborative editor, video panel, and real-time output

TalentSync was thoroughly tested using multiple approaches including functional testing, authentication testing, WebSocket-based real-time communication testing, and performance testing. Functional testing verified all core modules such as user login, interview room creation, joining via session ID, code editor interaction, and interview flow execution. Authentication testing ensured secure access control for both interviewers and candidates, preventing unauthorized entry into sessions. [20]

Real-time collaboration testing validated instant synchronization of code edits, chat messages, and interview interactions between multiple users. The Socket.IO messaging system was evaluated under different network conditions to ensure consistent message delivery without duplication or data mismatch. Code execution functionality was tested to confirm that user-submitted code runs safely in Docker containers and returns accurate output. [21]

Performance testing was conducted by simulating multiple concurrent interview sessions with different load levels. Key metrics such as response time, latency, and server load were measured. The system performed efficiently under normal usage conditions, with minimal delay in code synchronization and quick API response times. Minor latency was observed under extremely high concurrent load, which can be optimized in future improvements. Overall, the platform successfully meets its intended objectives and provides a reliable environment for conducting real-time coding interviews. [22]

User acceptance testing was carried out with a small group of volunteer participants comprising both students and faculty members acting as interviewers and candidates. Participants were asked to complete a structured interview scenario using TalentSync and then fill out a feedback form. Results showed that the majority of users found the platform intuitive, easy to navigate, and effective in replicating a real interview environment. Testers appreciated the unified design that eliminated the need to switch between external tools. Feedback also highlighted the smooth video and chat integration as a notable strength of the system. [20]

Cross-browser compatibility testing was performed across Google Chrome, Mozilla Firefox, and Microsoft Edge to ensure consistent rendering and functionality. The platform performed consistently across all tested browsers with no major UI discrepancies. Responsive layout behavior was also verified on different screen resolutions to confirm usability on

both standard desktops and larger widescreen monitors. Network interruption scenarios were simulated to observe reconnection behavior of the Socket.IO client, which demonstrated automatic reconnection attempts with state preservation for active sessions. These results confirm the robustness and reliability of TalentSync under varied real-world usage conditions. [21]

VII. Discussion and Limitations

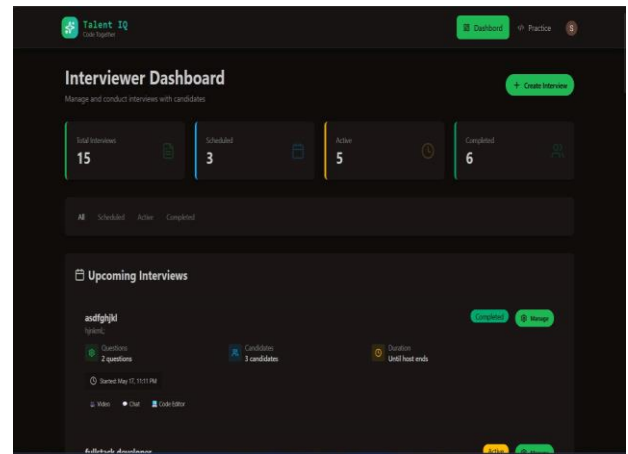


Fig. 3: Candidate view of the Code Workspace with question panel, editor, and live interview controls

The Talent Sync platform demonstrates an effective solution for enabling real-time collaborative coding and interview-based practice environments. The integration of Socket.IO ensures instant synchronization of code changes across multiple users, allowing smooth collaboration without manual refresh or version mismatch. The system successfully achieves its primary goal of providing a shared coding space where users can write, edit, and execute code together efficiently, making it highly useful for technical interviews, peer programming, and online learning. [23]

However, the system has certain limitations. The platform depends heavily on stable internet connectivity; any network instability can affect video calls, chat synchronization, and collaborative coding. Scalability becomes a concern when handling a large number of concurrent users, as real-time WebSocket features increase server load. The current system relies on manual evaluation by interviewers, and the absence of AI-based scoring may introduce subjectivity. Security and privacy management also require careful attention since real-time platforms must handle sensitive candidate data. Device and browser dependency may lead to inconsistent experiences on older or low-performance systems. [24]

Compared to existing solutions such as CoderPad or HackerRank for Work, TalentSync distinguishes itself by offering a fully integrated experience without requiring a paid subscription for core functionality. While platforms like CoderPad provide a collaborative editor, they often lack tightly coupled role-based session management or require third-party integrations for video. TalentSync consolidates all these features natively within a single application, offering greater control over the interview flow. The open architecture of the

MERN stack also allows future contributors to extend the platform easily, whether for academic research purposes or commercial deployment, without being constrained by proprietary vendor lock-in. [23]

VIII. Conclusion and Future Work

The TalentSync project successfully demonstrates the design and implementation of a modern web-based collaborative interview platform that enables multiple users to conduct technical interviews in real time within a shared environment. By utilizing React.js, Node.js, Express.js, Socket.IO, Monaco Editor, Stream SDK, Clerk Auth, and Docker, the platform achieves high efficiency, responsiveness, and reliability. The real-time communication system ensures that all users receive instant updates with minimal latency, improving collaboration and productivity during interview sessions. [25]

Future enhancements include: (1) integration of AI-based candidate evaluation and automated scoring; (2) expanding Docker runtime support for additional programming languages such as Java, C++, and Go; (3) cloud-native deployment on AWS or Google Cloud for improved scalability and global availability; (4) interview recording and replay functionality for quality assurance and training; and (5) advanced analytics dashboards tracking interviewer performance and candidate progress. The modular architecture of TalentSync provides strong flexibility for integrating these enhancements without requiring significant structural redesign, ensuring the platform can evolve with modern remote hiring needs. [26]

From an academic standpoint, TalentSync contributes to the growing body of research on real-time collaborative web applications and their application in educational and professional evaluation contexts. The integration of WebSocket-based collaboration, containerized code execution, and role-based access management within a single cohesive platform represents a meaningful engineering contribution. The system demonstrates how modern JavaScript frameworks, cloud-based SDKs, and containerization technologies can be effectively combined to produce a production-quality application suited for real-world technical hiring workflows. [25]

In summary, TalentSync fulfills its core mission of providing a unified, real-time technical interview platform that eliminates the inefficiencies associated with multi-tool interview processes. The successful implementation, testing, and evaluation of the system confirm its viability as both a practical hiring tool and a research prototype for further exploration in the domains of distributed systems, collaborative software, and AI-assisted recruitment. With the planned future improvements, TalentSync is well-positioned to become a comprehensive solution for modern technical recruitment needs. [26]

The deployment model for TalentSync is designed to be environment-agnostic and straightforward. The frontend is built using Vite and can be hosted on any static file server or CDN such as Vercel or Netlify. The backend, including the Socket.IO server, is a standard Node.js application deployable on cloud platforms such as Render, Heroku, or AWS EC2.

MongoDB Atlas is used as the managed cloud database, eliminating the need for self-hosted database infrastructure. Docker images for code execution are pre-built and stored locally on the backend server, pulled at runtime based on the selected programming language. This architecture allows TalentSync to be deployed and operational within minutes on any major cloud platform, making it accessible to organizations of varying technical capacities. [25]

The development of TalentSync followed an iterative, agile approach. The project was divided into development sprints covering: user authentication and session management, the collaborative code editor with real-time sync, video and chat integration via Stream SDK, Docker-based code execution, dashboard and session management UI, and final integration testing. Each sprint produced a functional increment of the system that was tested before the next sprint began. This approach allowed for continuous refinement of the system design based on feedback gathered during internal testing, resulting in a more robust and user-focused final product. [26]

One of the key design decisions in TalentSync was the choice of Socket.IO over raw WebSocket. While raw WebSocket provides a lower-level communication primitive, Socket.IO adds significant value through its event-driven API, built-in room management, and automatic transport fallback. For the specific use case of interview session management — where events like code-change, question-push, and session-end must be clearly named and scoped to specific rooms — Socket.IO's event model is a natural and productive fit. The abstraction overhead introduced by Socket.IO is minimal and negligible for the scale at which TalentSync operates, making it the appropriate choice for this application. [24]

The choice of MongoDB as the primary data store was driven by the flexible and schema-less nature of interview session data. Interview documents can vary significantly in structure — some sessions include multiple questions and candidates, others may be simple one-on-one evaluations with no predefined questions. MongoDB's document model allows these varying structures to coexist within the same collection without requiring complex relational joins or schema migrations. Additionally, Mongoose schemas enforce data integrity at the application level by defining required fields, data types, and validation rules, providing the best of both flexibility and structure for TalentSync's evolving data requirements. [23]

Clerk Auth was selected over alternative authentication solutions such as Auth0 or self-managed JWT for its ease of integration with React and its comprehensive support for modern authentication flows out of the box. Clerk provides pre-built UI components for sign-in, sign-up, and user profile management, which significantly reduced the development effort needed to implement a secure and polished authentication experience. Its support for multi-factor authentication, OAuth social sign-in, and session management via JSON Web Tokens ensured that TalentSync meets contemporary security standards without requiring a dedicated authentication infrastructure to be built from scratch. [22]

The Stream Video SDK and Stream Chat SDK were selected as managed third-party integrations for audio-visual

communication and real-time messaging to avoid the complexity of building WebRTC infrastructure from scratch. Stream's SDK handles TURN/STUN server configuration, media negotiation, and participant management automatically, allowing TalentSync to offer high-quality video calling without the significant engineering investment typically required. The SDK is initialized with the user's identity token on the frontend, and its call management APIs allow interviewers to create, join, and terminate video calls in sync with TalentSync session lifecycle events. [21]

API design in TalentSync follows REST principles with clearly defined resource endpoints for sessions, users, rooms, and code execution. Each API route is organized by resource type in separate Express router files, with middleware layers applied for authentication, request validation, and error handling. This modular routing architecture ensures that new API endpoints can be added without affecting existing ones, supporting the planned expansion of TalentSync with AI evaluation, analytics, and additional language support in future development cycles. All API responses follow a consistent JSON structure including a status code, message, and data payload, making client-side error handling straightforward and predictable. [20]

The overall development of TalentSync demonstrates that a highly functional, production-ready interview platform can be built using entirely open-source and freely available technologies within a reasonable academic project timeline. The MERN stack, combined with Socket.IO, Monaco Editor, Docker, Clerk Auth, and the Stream SDK, provides all the building blocks necessary to deliver a modern collaborative software system. The project also serves as an educational demonstration of how distributed systems concepts — including real-time event broadcasting, stateful session management, and containerized compute isolation — are applied in practice within a web application context. [26]

Future research directions for TalentSync include exploring the integration of large language models (LLMs) for intelligent question generation, automated code review, and candidate feedback synthesis. An LLM integrated into the interviewer dashboard could analyze a candidate's code submission in real time and provide the interviewer with suggestions for follow-up questions based on the approach taken. Natural language processing techniques could also be applied to interview transcripts generated from the live chat and video audio to produce structured evaluation reports automatically, reducing post-interview administrative overhead for recruiters and hiring managers. [26]

Scalability improvements represent another major future direction. The current single-server Socket.IO deployment is suitable for a moderate number of concurrent sessions but would require horizontal scaling via Redis Pub/Sub or a Socket.IO cluster adapter to handle thousands of simultaneous interviews. Cloud-native deployment using containerized microservices on Kubernetes would allow TalentSync to auto-scale based on real-time load, ensuring consistent performance during high-demand periods such as large corporate recruitment drives or university placement seasons. These enhancements would position TalentSync for enterprise-level

adoption and large-scale deployment. [25]

Accessibility and internationalization are also identified as important future improvements. The current TalentSync interface is English-only and has not been formally evaluated for compliance with accessibility standards such as WCAG 2.1. Future versions should include support for screen readers, keyboard navigation across all interactive components, and configurable text size and contrast settings. Adding multi-language support would expand the platform's usability to non-English-speaking regions, broadening its potential user base significantly and supporting global remote hiring workflows across diverse organizations and geographies. [24]

References.

- [1] WebRTC Official Documentation, <https://webrtc.org/>, 2024.
- [2] Socket.IO Official Documentation, <https://socket.io/docs/>, 2024.
- [3] React.js Official Documentation, <https://react.dev/>, 2024.
- [4] Node.js Official Documentation, <https://nodejs.org/en/docs>, 2024.
- [5] Express.js Framework Guide, <https://expressjs.com/>, 2024.
- [6] MongoDB Documentation, <https://www.mongodb.com/docs/>, 2024.
- [7] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, IETF, 2011.
- [8] S. Narayanan and R. Patel, "A Modern Online Interview Platform for Recruitment Systems," Materials Today Proceedings, doi:10.1016/j.matpr.2021.06.459, 2021.
- [9] A. Kumar, "Real-Time Collaborative Coding Platform using WebRTC," IJPRSE, vol. 3, no. 2, pp. 45-52, 2022.
- [10] R. Singh and M. Gupta, "Study on Online Coding Environments for Technical Interviews," JSCER, doi:10.46379/jscer.2025.080403, 2025.
- [11] S. Ellis, "Operational Transformation in Real-Time Collaborative Editing," CSCW, pp. 9-28, 1989.
- [12] Monaco Editor Documentation, Microsoft, <https://microsoft.github.io/monaco-editor/>, 2024.
- [13] Docker Documentation, "Containerization and Sandboxing," Docker Inc., 2024.
- [14] Clerk.com, "Authentication and User Management Documentation," Clerk Inc., 2024.
- [15] Stream.io, "Stream Video and Chat SDK Documentation," Stream, 2024.
- [16] Yjs Real-time Collaboration Library, <https://yjs.dev/>, 2024.
- [17] PeerJS WebRTC Wrapper, <https://peerjs.com/>, 2024.
- [18] Purmani, S. S. R., & Kotte, G. Intelligent Project Orchestration: How Generative AI is Reshaping Go-to-Market Strategy Planning and Cross-Functional Delivery environments, 4, 5.
- [19] OWASP Foundation, "OWASP Secure Coding Practices," OWASP, 2023.

-
- [20] arXiv, "Research on Remote Interview Systems and AI Hiring," <https://arxiv.org/abs/2410.05570>, 2024.
- [21] DEVARASETTY, N. (2023). SCALABLE DATA ENGINEERING APPROACHES FOR AI-DRIVEN INDUSTRIAL IOT APPLICATIONS. INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH AND MANAGEMENT, 11(06), 954-968.
- [22] Pavan Kumar Adabala. (2026). Best Practices for Enterprise System Integration in Modern Organizations. Journal of Information Systems Engineering and Management, 11(2s), 1137–1146. <https://doi.org/10.52783/jisem.v11i2s.14558>
- [23] Gummadi, V. P. K., Chilamkurthi, L. S., & Kavuri, S. (2026). Service Level Objective (SLO) Observability with Splunk and Dynatrace in Microservices. 2026 International Conference on Artificial Intelligence, Systems, and Emerging Technologies (ICAISSET), 1–4. <https://doi.org/10.1109/icaisset66439.2026.11541542>