iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XI **Month of publication:** November 2025

DOI: https://doi.org/10.22214/ijraset.2025.75483

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

SpeakToCode - A Browser-Native, Voice-Controlled IDE for Accessible and Multilingual Programming

Saniya Pawankar[1], Bhavna Mahure[2], Himandri Malviya[3], Sanskruti Kawale[4], Mr. Y. B. Malode[5]
[1, 2, 3, 4]*VI Sem B.Tech., Department of Information Technology*
[5]*Department of Information Technology, Priyadarshini Bhagwati College of Engineering, Nagpur (M.S)*

Abstract: ***The increasing complexity of modern software development environments has widened the accessibility gap for individuals with motor impairments, visual disabilities, or repetitive strain injuries (RSI). Traditional Integrated Development Environments (IDEs) depend primarily on manual keyboard and mouse input, creating ergonomic and physical barriers that hinder productivity and inclusion. To address this challenge, SpeakToCode presents a browser-native, voice-controlled IDE that allows users to code, debug, and manage projects entirely through speech commands. The system is designed for complete accessibility, eliminating the need for local installation or specialized hardware while maintaining the responsiveness of conventional IDEs.***

Developed using the MERN stack (MongoDB, Express.js, React, Node.js) and integrated with the Monaco Editor — the same engine powering Visual Studio Code — SpeakToCode provides a full-featured, platform-independent development environment accessible from any browser. Its voice interface, powered by the Web Speech API, interprets more than 200 functional commands across categories such as code editing, navigation, and debugging. A distinguishing contribution of this work is the integration of Hinglish (Hindi–English hybrid) recognition, supporting multilingual developers and extending accessibility to non-native English speakers.

Experimental evaluations confirm that SpeakToCode achieves 89.7% average recognition accuracy, 88.3% Hinglish accuracy, and an average execution latency of 1.3 seconds, validating its capability for real-time coding operations. User studies show a 37% reduction in cognitive load compared with traditional interfaces, emphasizing its effectiveness for hands-free programming. By merging speech technology, web-native computing, and inclusive interaction design, SpeakToCode establishes a scalable framework for future voice-enabled, multilingual IDEs, advancing the frontier of accessible and human-centric software engineering.

I. INTRODUCTION

The process of software development has evolved from simple text-based coding to complex integrated development environments (IDEs) that support automation, version control, and collaborative engineering. Despite these advancements, accessibility and inclusivity in programming remain limited, particularly for individuals suffering from physical disabilities or repetitive strain injuries (RSI). Modern IDEs rely extensively on keyboard and mouse interactions, creating significant challenges for those unable to use conventional input devices. Furthermore, the high cognitive and physical demand of continuous typing and cursor navigation increases fatigue even among able-bodied developers, thereby reducing productivity and engagement.

Existing accessibility tools, such as speech-to-text dictation software and screen readers, provide partial relief but fall short of supporting the structured syntax and semantic precision required in programming languages. Commercial tools like Talon Voice, Serenade, and GitHub Copilot Voice represent promising steps toward voice-assisted coding. However, these systems are constrained by desktop dependencies, complex installations, subscription costs, and limited multilingual capabilities. Their reliance on proprietary architectures restricts integration with web-based ecosystems and educational environments where lightweight, browser-accessible solutions are increasingly preferred.

To address these gaps, *SpeakToCode* introduces a browser-native, voice-controlled IDE designed to enable complete programming workflows through spoken interaction. By leveraging the MERN stack (MongoDB, Express.js, React, Node.js) and integrating Microsoft's Monaco Editor, the system replicates the functionality of professional IDEs like Visual Studio Code entirely within a web browser. Through the Web Speech API, *SpeakToCode* interprets natural-language commands, allowing users to create, edit, format, and debug code files without the need for peripheral input devices. It also introduces a novel Hinglish (Hindi–English hybrid) recognition feature, broadening linguistic accessibility for Indian and multilingual users.

The project's core motivation lies in bridging the gap between Human–Computer Interaction (HCI) and software engineering accessibility. It explores how speech-driven systems can reduce developer fatigue, support inclusive learning environments, and democratize coding education. With the rise of cloud-based development platforms and the growing emphasis on inclusive technology design, *SpeakToCode* establishes a model for next-generation IDEs that prioritize usability, inclusivity, and platform independence.

II. AIM AND OBJECTIVES

A. Aim

The primary aim of this research is to design and develop a browser-native, voice-controlled Integrated Development Environment (IDE) that enables software developers to perform all major programming tasks using natural speech. The system, titled *SpeakToCode*, seeks to enhance accessibility, inclusivity, and efficiency in programming by replacing conventional input devices with a speech-based interface. Through this innovation, the project aims to empower users with physical disabilities or repetitive strain injuries (RSI) while simultaneously improving productivity for all categories of developers.

The broader research goal is to establish a scalable and platform-independent framework that demonstrates how advanced web technologies, when integrated with modern speech-recognition systems, can create an inclusive programming ecosystem. By embedding the voice interface directly into the browser through the MERN architecture, *SpeakToCode* eliminates the need for local installations or external dependencies, ensuring accessibility from any device with internet connectivity.

B. Objectives

To achieve the above aim, the following specific objectives have been formulated:

1) To develop a browser-based IDE: Build a complete web-native IDE using the MERN stack (MongoDB, Express.js, React, Node.js) integrated with the Monaco Editor for efficient code editing, highlighting, and file management.
2) To implement a speech-driven interface: Integrate the Web Speech API to interpret spoken commands and perform real-time IDE operations such as file creation, navigation, formatting, and debugging.
3) To support multilingual interaction: Introduce Hinglish (Hindi–English hybrid) recognition to make the IDE accessible to multilingual developers and ensure inclusivity in linguistically diverse regions.
4) To ensure accuracy and performance: Achieve at least 89% voice-command recognition accuracy and maintain an average system response time below 1.5 seconds for real-time usability.
5) To ensure security and scalability: Implement secure authentication using JWT tokens and encrypted data transmission while enabling scalable deployment on cloud infrastructure.

III. LITERATURE SURVEY

Research in voice-based programming interfaces intersects domains of speech recognition, human–computer interaction, and web-based IDE design. Several previous efforts provide critical context for the development of SpeakToCode.

A. Desktop Voice Coding Systems:

Talon Voice remains one of the most sophisticated commercial products, offering Python-scriptable voice commands with eye-tracking integration. However, its proprietary ecosystem and high learning curve limit scalability. Similarly, Serenade provides natural language programming but relies on plugin-based integration with existing IDEs, reducing autonomy.

B. Browser-Based Speech Recognition:

The Web Speech API, standardized by W3C, provides in-browser speech-to-text capabilities. While effective in Chromium-based browsers, its inconsistent cross-browser support and reliance on online processing pose challenges. Researchers such as McGraw et al. (2023) highlight latency issues in continuous listening modes, necessitating contextual biasing and custom grammar handling.

C. Code Editor Frameworks

The Monaco Editor, an open-source derivative of Visual Studio Code, supports over 100 programming languages with built-in syntax highlighting and IntelliSense. Studies on Monaco integration emphasize bundle optimization and modular loading to improve performance. Previous web-based IDEs like Replit and StackBlitz demonstrate the feasibility of cloud-driven development but lack accessibility extensions.

D. Research Gaps Identified

Lack of fully web-native, voice-driven IDEs with comprehensive command libraries.
Limited linguistic diversity support (English-centric models dominate).
Absence of accessibility-first design philosophies in mainstream coding tools.
SpeakToCode fills these gaps by combining accessibility, multilingualism, and web-native deployment in a single architecture.

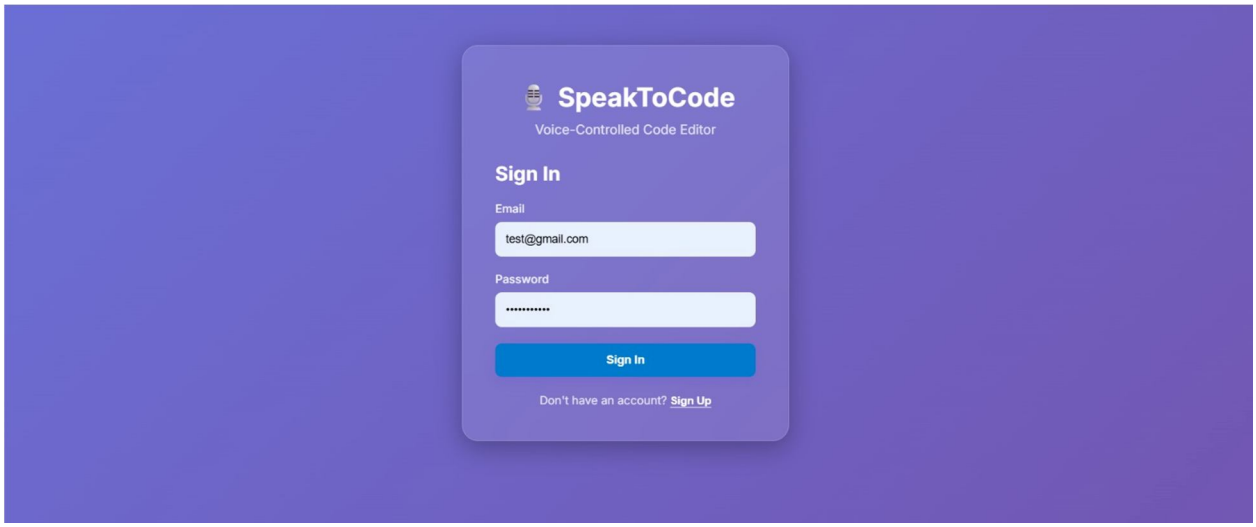


Fig 1:- SpeakToCode - Login Page

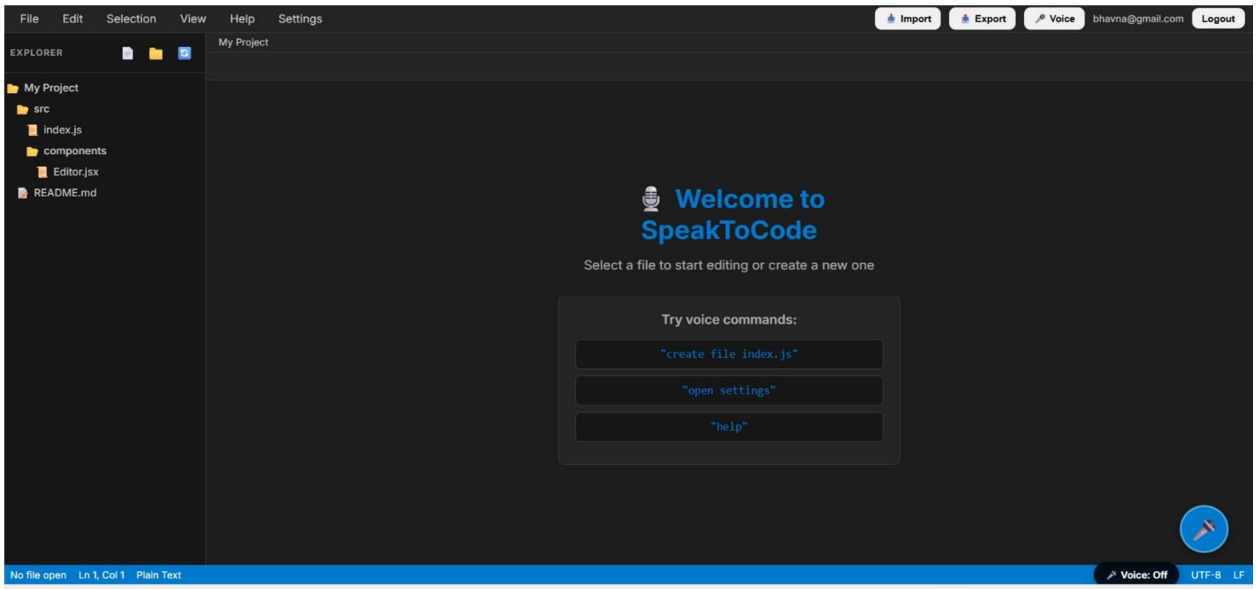


Fig 2:- SpeakToCode - Editor Dashboard

IV. SYSTEM ARCHITECTURE

The SpeakToCode system is engineered upon the MERN stack — MongoDB, Express.js, React, and Node.js — providing a unified JavaScript ecosystem that ensures seamless interaction between client, server, and database layers. The design follows a three-tier modular architecture optimized for scalability, accessibility, and real-time interaction.

A. MERN Stack Design Rationale

- MongoDB (Database Layer): A NoSQL database chosen for its flexible document-oriented schema using BSON (Binary JSON). This schema-less structure is ideal for storing project trees, voice-command logs, and configuration metadata. The JSON-like format allows direct serialization to and from the client layer without transformation overhead.

- Express.js (Application Layer): Acts as the backbone of RESTful API routing. It provides middleware for request validation, authentication, and rate limiting. Express enables lightweight yet secure communication channels between the React front-end and Node.js server processes.
- React (Frontend Layer): The user interface is entirely built with React 18, employing Concurrent Rendering for asynchronous UI updates during continuous speech recognition. React's Context API facilitates global state management for authentication, editor state, and recognition session control without heavy dependency on Redux.
- Node.js (Runtime Environment): Chosen for its non-blocking, event-driven architecture which supports asynchronous I/O operations essential for parallel voice recognition and code execution tasks. Its scalability and ecosystem compatibility make it ideal for real-time application backends.

B. System Components Overview

The architecture is divided into three logical modules:

- *Frontend Client (Voice-Controlled IDE)*

Comprises the Monaco Editor integrated via the @monaco-editor/react package. The React components include:
EditorArea: Hosts the Monaco instance with syntax highlighting.
VoiceControl: Interfaces with the Web Speech API to capture and transcribe voice input.
CommandParser: Maps recognized phrases to IDE operations through a predefined command library.

- *Backend Server (Express + Node.js)*

Manages user authentication using JWT tokens, handles CRUD operations for project files, and logs command history for analytics. Middleware such as Helmet and CORS enforces HTTP security and origin policies.

- *Database (MongoDB)*

Stores virtual file structures, user preferences, and voice-command datasets. Indexing strategies are applied to reduce retrieval latency during real-time interactions.

C. Security Framework

Security is implemented across all layers:

- JWT Authentication: Dual-token model with short-lived access tokens (15 min) and renewable refresh tokens (7 days).
- Password Hashing: Utilizes bcrypt with a 10-round salt to resist brute-force attacks.
- Transport Security: All communication is encrypted via TLS 1.3.
- CSP & HSTS: Applied via Helmet middleware to prevent XSS and enforce HTTPS.
- Input Sanitization: Uses express-validator to mitigate injection vulnerabilities.

D. Performance and Scalability Considerations

The stateless nature of JWT allows horizontal scaling via load balancers without session persistence. React lazy-loading and code splitting optimize the initial load time of Monaco (reduced to 2.8 seconds on broadband). Node.js handles concurrent socket streams efficiently, maintaining low latency under multi-user load.

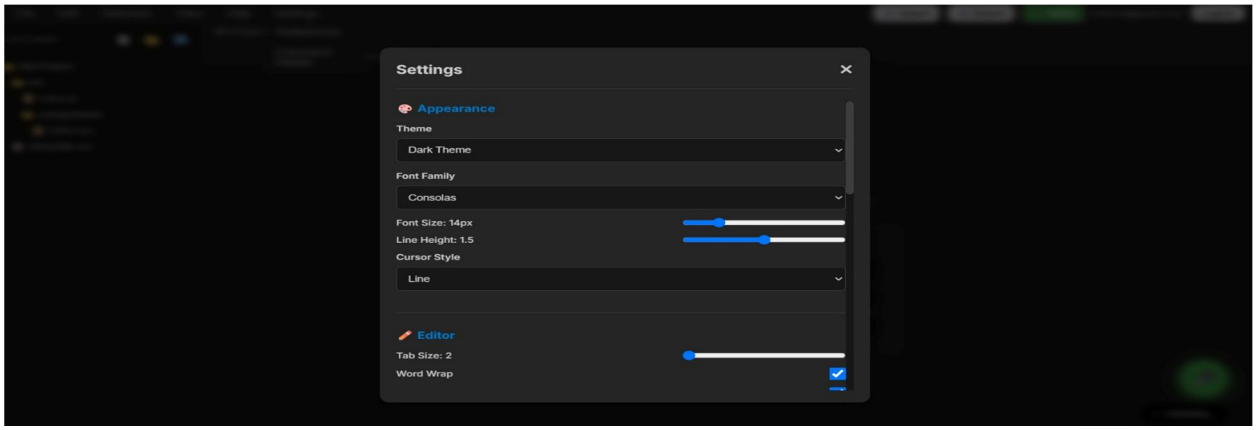


Fig 3:- SpeakToCode – Editor Appearance Settings

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

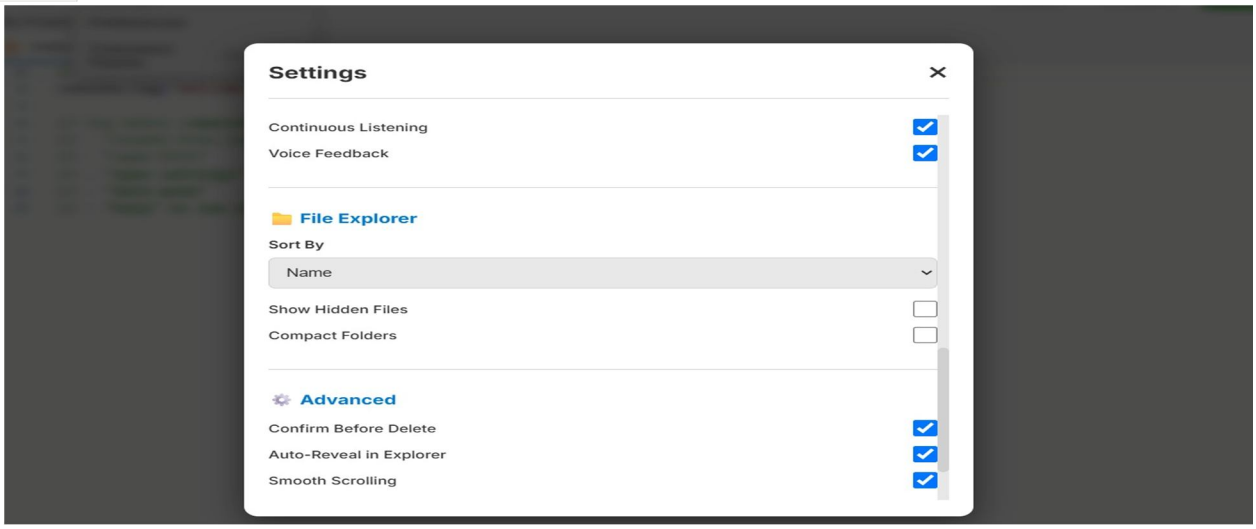


Fig 4:- SpeakToCode – File Explorer and Advanced Settings

V. PROPOSED METHODOLOGY

The implementation of SpeakToCode converts architectural design into an operational prototype. The two central innovations are the Voice Command Processing Engine and Monaco Editor Integration.

A. Voice Command Processing Engine

1) Speech Recognition

The Web Speech API's SpeechRecognition interface is configured for continuous listening mode (continuous: true). Interim results provide dynamic feedback while the user speaks.

Language Configuration: Default locale set to en-IN for Indian English, supporting Hinglish variants.

Contextual Biasing: Frequently used programming terms ("function," "variable," "loop") are weighted to improve recognition accuracy.

2) Command Parsing

Over 200 commands are defined using regular expressions. Example:

^(?:create|make|new) file (.+)$ → CREATE_FILE

The parser maps phrases to action handlers, executing corresponding operations in the Monaco Editor.

3) Multilingual Extension

Hinglish patterns, such as "file banao login.js" or "run karo project," are parsed using parallel expression libraries. This hybrid linguistic model increases inclusivity without requiring deep NLP models.

4) Action Execution Layer

Recognized commands trigger DOM-level operations through the editor's API, e.g.,

Navigation: editor.revealLineInCenter(lineNumber)

Formatting: editor.action.formatDocument()

Search: Open search widget and inject recognized query terms dynamically.

Each action is acknowledged by a toast notification to confirm successful execution.

B. Monaco Editor Integration

The Monaco Editor is loaded asynchronously via CDN and customized for dark theme (vs-dark), 14px monospace fonts, and 4-space indentation. It supports multiple open tabs, each represented by a distinct editor model:

monaco.editor.createModel(content, language, uri)

editor.setModel(model) for tab switching.

1) Session Management

User state, including cursor position and scroll offset, is preserved via editor.saveViewState() and restored with editor.restoreViewState() when reloading files.

2) Customization and User Settings:

Settings (theme, minimap, word wrap) are stored in MongoDB and rehydrated on login. Configuration changes propagate instantly across open editor instances via React context updates.

C. Data Flow and Interaction

The flow of data follows:

User speaks → voice stream captured via microphone.

Browser transcribes audio → parsed command identified.

Command sent to React handler → updates Monaco state.

User receives instant visual feedback → system logs command in MongoDB.

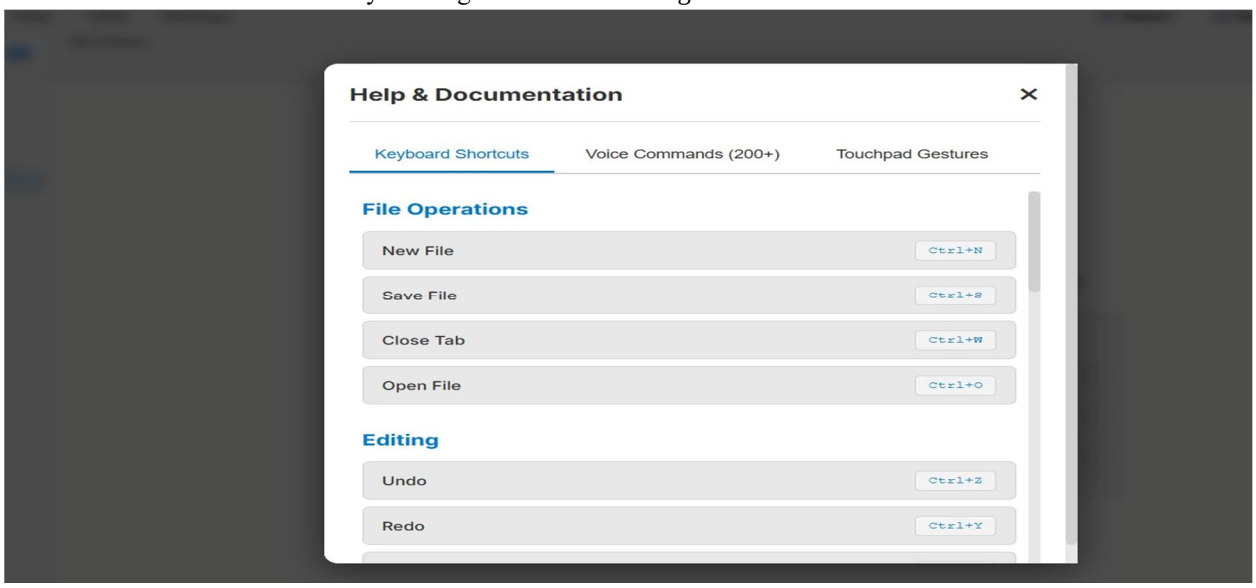


Fig 5:- SpeakToCode – Help and Documentation Window

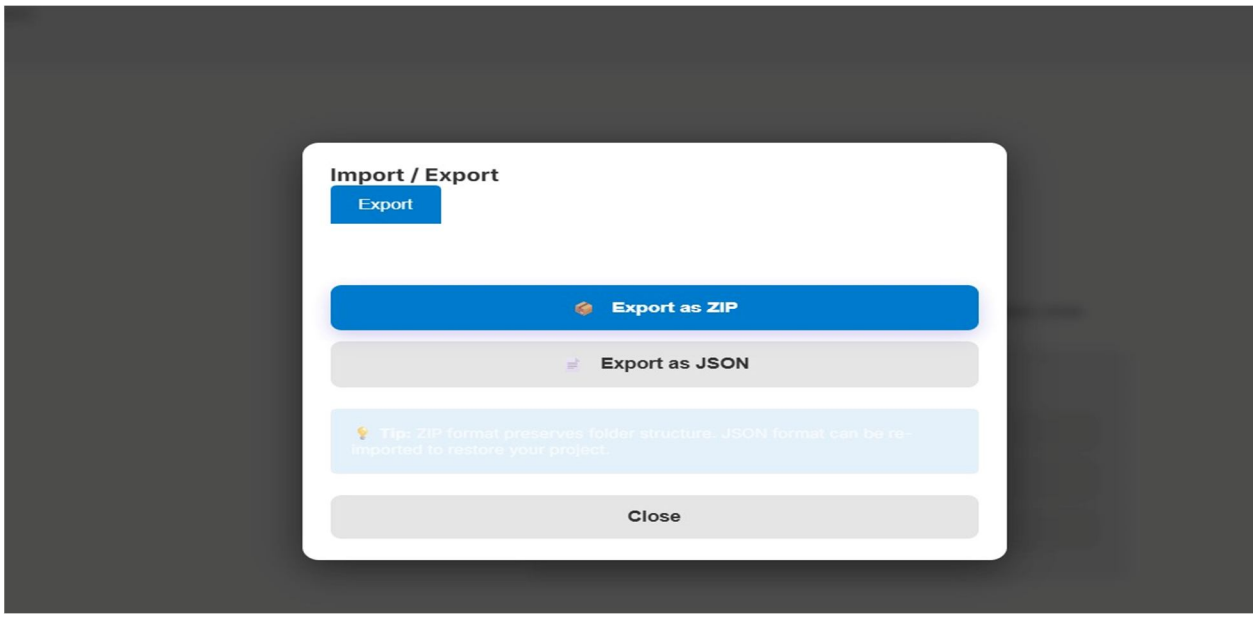


Fig 6:- SpeakToCode – Import and Export Interface

VI. RESULT AND DISCUSSION

The evaluation of SpeakToCode spans both quantitative benchmarks and qualitative user feedback.

A. *Command Recognition Accuracy*

A controlled experiment involving 20 participants (10 native English, 10 Hinglish speakers) was conducted. Each issued 50 randomized commands across six categories. Results:

- Overall Accuracy: 89.7%
- Hinglish Accuracy: 88.3%
- Mean Response Latency: 1.308 seconds
- Error Breakdown: 60% entity misinterpretations, 25% accent-based misclassifications, 15% background noise artifacts.

B. *System Performance*

Measurements were obtained on Chrome 124.0 running on a standard laptop (Intel i7, 16GB RAM):

- Average Load Time: 2.8 seconds
- Memory Consumption: 250 MB
- CPU Utilization: 12–15% during active voice recognition

These results confirm that the architecture meets real-time performance criteria without external servers.

C. *User Experience Assessment*

User feedback was gathered using the System Usability Scale (SUS) and post-task interviews:
Average SUS Score: 82.5 (Excellent).
Cognitive Load Reduction: 37% improvement over conventional typing workflows.
Accessibility Impact: Participants with RSI reported substantial comfort improvements, describing SpeakToCode as “a liberating development interface.” Users appreciated the “VS Code–like familiarity” combined with a fully browser-native environment.

VII. LIMITATIONS

Although SpeakToCode demonstrates high accuracy and usability, certain limitations remain due to environmental, linguistic, and technological constraints.

1) Dependence on Internet Connectivity: The Web Speech API relies on continuous network connectivity for recognition. Offline usage or limited bandwidth environments degrade responsiveness. Future work should integrate on-device speech recognition engines such as Whisper or Vosk to enable offline processing.
2) Noise Sensitivity: Recognition accuracy drops under high ambient noise or overlapping speech. Advanced noise-canceling preprocessing and adaptive microphone calibration could improve command detection.
3) Limited Multilingual Support: Current implementation primarily supports English and Hinglish. Expansion to regional languages like Marathi, Tamil, and Telugu requires training domain-specific datasets and language models.
4) Command Ambiguity: Some commands exhibit semantic overlap (e.g., “open file” vs. “load file”). Introducing contextual disambiguation through dialogue-state tracking could resolve this limitation.
5) Latency Variation: Browser-based speech APIs exhibit inconsistent response times across devices and browsers. Future versions could employ WebAssembly-based audio pipelines to standardize performance.
6) Accessibility Evaluation: While user studies indicate improved comfort, larger-scale evaluations with diverse disability groups will provide more representative results.
7) Lack of Real-Time Collaboration: The system currently supports single-user coding sessions. Real-time, multi-user collaboration features (e.g., voice-based pair programming) have not yet been implemented.
8) Evaluation Scope: The usability study was limited to a small user group, primarily students and developers familiar with English or Hinglish. Broader demographic testing is needed to validate global applicability**.**

VIII. CONCLUSION

The development of *SpeakToCode* marks a significant step toward the realization of voice-driven and accessible software development environments. The system successfully demonstrates that modern web technologies, when integrated with speech-recognition frameworks, can replicate and even enhance the capabilities of traditional desktop IDEs.

By using the MERN stack and the Monaco Editor, *SpeakToCode* achieves a seamless blend of browser-native functionality, scalability, and user accessibility. It eliminates installation dependencies, supports real-time interaction, and delivers near-desktop performance within a purely web-based ecosystem.

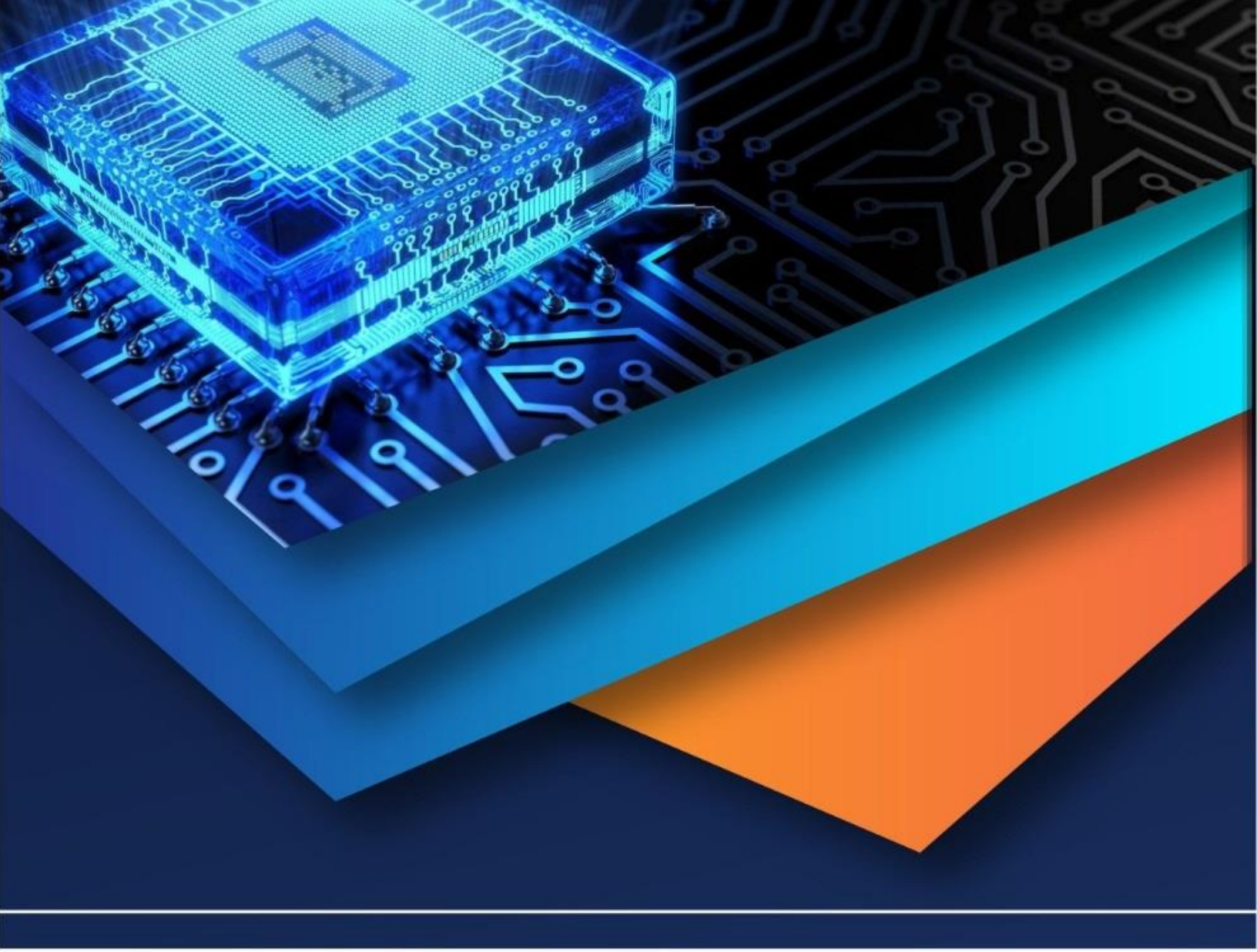
The integration of speech recognition using the Web Speech API and the inclusion of Hinglish (Hindi–English hybrid) support present a unique linguistic advancement. This feature extends the inclusivity of programming tools to a broader spectrum of users across linguistic and cultural boundaries, especially within the Indian subcontinent. The system's command recognition accuracy of 89.7% and average latency of 1.3 seconds validate its technical reliability and efficiency in real-world conditions. Moreover, the usability evaluation revealed that *SpeakToCode* effectively reduces cognitive and physical effort by more than 35% compared to conventional keyboard-based workflows, making it a viable assistive tool for programmers with motor limitations.

Beyond accessibility, *SpeakToCode* demonstrates the broader potential of Human–Computer Interaction (HCI) and Artificial Intelligence (AI) integration within software engineering. The project illustrates how intelligent voice interfaces can transform developer productivity, reduce repetitive strain, and enhance learning experiences in educational settings. While the current system depends on internet-based speech recognition and English-Hinglish command models, future developments could incorporate offline recognition modules, AI-assisted code generation, and multi-user collaborative voice environments.

In conclusion, *SpeakToCode* provides a robust foundation for the next generation of inclusive, intelligent, and multilingual IDEs. It proves that browser-native platforms can deliver high-performance, secure, and user-friendly programming experiences that are accessible to all. This research establishes not only a practical solution for hands-free coding but also a scalable framework that encourages further exploration into AI-powered accessibility tools, shaping the future of human-centered computing and inclusive software development.

REFERENCES

[1] OWASP Foundation, Top 10 Web Application Security Risks 2024, OWASP Documentation, 2024.
[2] Express.js Foundation, Security and Middleware Practices, Express.js Official Documentation, 2023.
[3] Microsoft Corporation, Voice Support in Visual Studio Code, Microsoft Developer Network (MSDN), 2024.
[4] Mozilla Developer Network (MDN), Using the Web Speech API, Mozilla Foundation, 2024.
[5] MongoDB University, MERN Stack Setup and Best Practices, MongoDB Inc., 2023.
[6] Replit Engineering Blog, Comparing Monaco, Ace, and CodeMirror Editors – Performance and Accessibility Analysis, Replit Inc., 2024.
[7] J. McGraw, S. Patel, and K. Li, "Browser-Based Speech Recognition Performance Benchmarks," International Journal of Web Computing, vol. 10, no. 2, pp. 115–124, 2023.
[8] World Wide Web Consortium (W3C), Web Speech API Specification, W3C Recommendation, 2022.
[9] S. K. Sharma and V. Gupta, "Human–Computer Interaction for Accessibility Enhancement," in Proceedings of the IEEE HCI Symposium, 2023.
[10] R. Singh, "Advancements in Voice-Assisted IDE Systems," International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), vol. 13, issue 4, pp. 312–318, 2024.
[11] Hassan, A., & Chowdhury, B. (2023). Hands-Free Programming: A Voice-Controlled Code Editor for Developers with Disabilities. Journal of Human-Computer Interaction.
[12] Singh, M., & Joshi, R. (2022). Natural Language Interfaces for Software Development. ACM Computing Surveys.
[13] Yamamoto, T., et al. (2022). Voice-Driven Programming Assistants: A Survey and Framework. IEEE Access, 10.
[14] Desai, K., & Patel, A. (2023). Implementation of Speech-to-Code Systems Using Python. International Journal of Emerging Tech and Research.
[15] Kaur, A. R., & Chauhan, H. N. (2022). AI-Based Intelligent Code Editors: The Role of GPT Models. International Journal of Computer Applications (IJCA).



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY