iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80864

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 14 Issue IV Apr 2026- Available at www.ijraset.com

Design and Development of an AI-Enhanced Online Integrated Development Environment

Potnuru Hari Shankar[1], Pallapati Syam[2], Parisiboina Venkata Revanth[3], Palivela Surya Teja[4]
Department of Computer Applications, Aditya University, Surampalem, India,

Abstract: ***The rapid evolution of web technologies and artificial intelligence has created new opportunities for building intelligent software development tools that are accessible, efficient, and scalable. This paper presents the design and development of an AI-enhanced Online Integrated Development Environment (IDE) that enables users to write, compile, and execute programs in multiple programming languages directly through a web browser, without any local system configuration. The platform integrates Google Gemini Large Language Models (LLMs) to provide intelligent capabilities including code generation, auto-completion, refactoring suggestions, and detailed error explanations, thereby supporting both experienced developers and learners. The system adopts a full-stack microservices architecture, employing React for the frontend, Node.js and Express for authentication services, and Python Flask for AI processing and file-sharing microservices. MongoDB is utilized for persistent data storage, while Redis handles temporary code storage and enables secure code sharing. User authentication is secured through JSON Web Tokens (JWT), bcrypt password hashing, and OTP-based email verification. The Monaco Editor serves as the core editing component, offering syntax highlighting, intelligent code completion, and support for over 16 programming languages. Experimental results demonstrate that the proposed system effectively enhances developer productivity, reduces the learning curve for new programmers, and provides a reliable, secure, and scalable environment for online software development.***
Index Terms: ***Online IDE, Web-Based Development Environment, Large Language Models, Google Gemini, AI-Assisted Programming, Code Generation, Monaco Editor, Microservices Architecture, React.js, Node.js, Python Flask, MongoDB, Redis, JSON Web Tokens, bcrypt, OTP Authentication, Syntax Highlighting, Multi-Language Support, Code Execution, Secure Authentication.***

I. INTRODUCTION

The increasing demand for flexible and accessible software development tools has driven a significant shift toward cloud-based and browser-based development environments. Traditional Integrated Development Environments (IDEs) require users to install and configure software locally, which can be a barrier for learners, teams working across multiple devices, or developers in resource-constrained environments. Online IDEs address these challenges by providing a web-based interface that allows users to write, compile, and execute code directly in the browser without any local setup.

Recent advancements in Artificial Intelligence, particularly in the area of Large Language Models (LLMs), have opened new possibilities for enhancing developer productivity. LLM-powered tools can assist developers with intelligent code generation, real-time error explanation, code refactoring, and context-aware auto-completion. Integrating such capabilities into an online IDE can significantly reduce the time required to write, debug, and optimize code, while also making programming more approachable for beginners.

Existing online IDEs such as Replit, CodeSandbox, and JDoodle provide basic code execution capabilities but offer limited AI integration. Most available platforms do not combine deep AI assistance with robust security mechanisms, multi-language support, and real-time code sharing in a unified architecture. There remains a clear gap for a comprehensive platform that addresses all these needs simultaneously.

This paper proposes an AI-enhanced Online IDE designed to address these limitations. The platform uses a microservices architecture with React on the frontend and separate backend services for authentication, AI processing, and file sharing. Google Gemini LLMs are integrated to deliver intelligent coding assistance. Secure authentication is implemented using JSON Web Tokens (JWT), bcrypt hashing, and OTP-based email verification. The Monaco Editor provides a powerful editing experience with support for over 16 programming languages.

The primary objective of this work is to build a scalable, secure, and user-friendly online development platform that merges the convenience of browser-based coding with the power of AI assistance. The system aims to benefit students, educators, developers, and technical teams who require a versatile coding environment accessible from any device.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

II. LITERATURE REVIEW

The development of cloud-based and browser-based IDEs has been an active area of research and industrial development. Early online editors such as Ace and CodeMirror [1] laid the foundation for in-browser code editing by providing syntax highlighting and basic language support. More sophisticated platforms such as Eclipse Theia and VS Code for the Web [2] have since extended these capabilities with file system integration, extensions, and remote execution environments.

Research on AI-assisted programming has grown substantially in recent years. Studies by Chen et al. [3] on GitHub Copilot demonstrated that LLM-based code completion tools can significantly reduce the time developers spend on repetitive coding tasks. Similarly, Fried et al. [4] introduced InCoder, a model capable of code infilling and generation, highlighting the value of generative AI in software development workflows. These works confirm the productivity benefits of integrating LLMs into development environments.

The Monaco Editor, developed by Microsoft [5], has emerged as one of the most widely adopted in-browser code editing components, powering VS Code and several cloud IDE platforms. Its support for IntelliSense, syntax highlighting, and multi-language operation makes it a strong foundation for building feature-rich online editors. Several recent systems have leveraged Monaco in combination with containerized execution engines to enable multi-language code execution [6].

Microservices architecture has been widely adopted in modern web application design due to its scalability, fault isolation, and ease of deployment [7].

Research by Newman [8] and others has established best practices for decomposing monolithic applications into independently deployable services, which is particularly relevant for complex platforms such as online IDEs that require separate concerns for authentication, code execution, AI processing, and data storage.

Security in web-based applications remains a critical concern. JWT-based authentication combined with bcrypt hashing is widely recognized as a secure approach for stateless user authentication in RESTful APIs [9]. OTP-based email verification provides an additional layer of security against unauthorized account creation and access. Redis has been extensively studied as a high-performance in-memory data store suited for session management, temporary storage, and real-time data sharing [10]. The proposed system builds upon these established technologies and research findings to create a comprehensive, secure, and intelligent online development platform.

III. SYSTEM ARCHITECTURE

The proposed system is built on a full-stack microservices architecture that separates concerns across multiple independently deployable services. The architecture consists of four major layers: the frontend client layer, the authentication service layer, the AI and file-sharing service layer, and the data storage layer. Each layer is designed to operate independently, enabling scalability and ease of maintenance.

The frontend is built using React.js and incorporates the Monaco Editor as the core code editing component. The Monaco Editor provides syntax highlighting, intelligent auto-completion, and multi-language support for over 16 programming languages including Python, JavaScript, Java, C, C++, TypeScript, Go, Rust, PHP, Ruby, and more. The user interface is designed to be responsive and accessible across different devices and screen sizes.

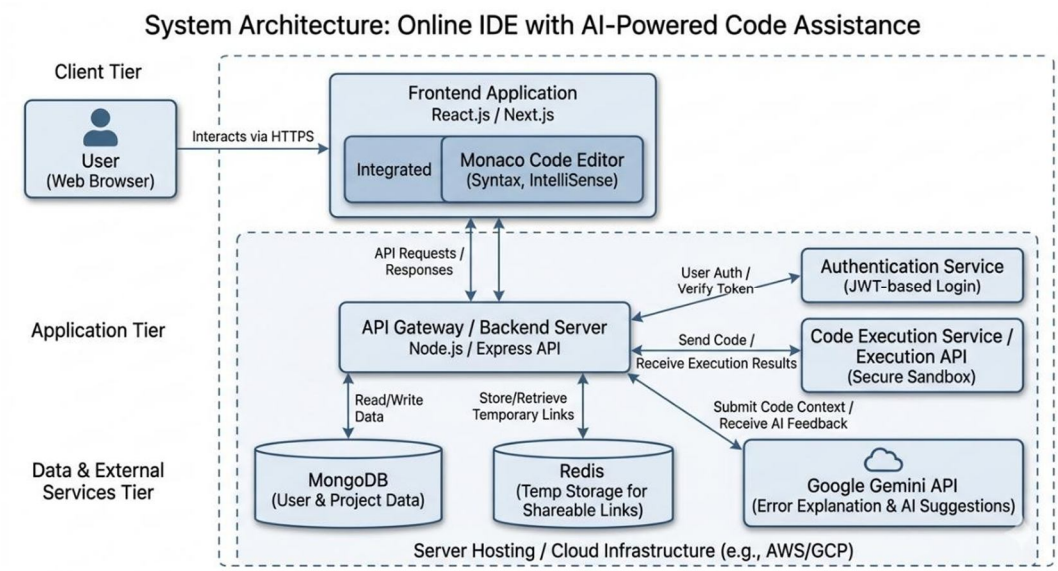
The authentication service is implemented using Node.js and Express. It handles user registration, login, and session management. Security is enforced through JWT-based stateless authentication, bcrypt password hashing, and OTP-based email verification. Upon successful authentication, a signed JWT is issued to the client, which is validated on every subsequent request to protected resources.

The AI microservice is developed using Python Flask and interfaces with the Google Gemini LLM API. This service processes user requests for code generation, error explanation, refactoring suggestions, and intelligent auto-completion. The service receives code context and natural language prompts from the frontend and returns structured AI-generated responses. The file-sharing microservice, also built in Flask, manages code snippet sharing using Redis for temporary storage, enabling users to share code via secure, time-limited links.

MongoDB serves as the primary persistent data store, managing user profiles, saved projects, code history, and settings. Redis is employed for two purposes: caching frequently accessed data to improve response times, and storing temporary code snippets for secure real-time sharing between users. The combination of MongoDB and Redis ensures both durability and high-performance access patterns for the platform.

©IJRASET: All Rights are Reserved

Fig. 1. Proposed System Architecture



IV. PROPOSED METHODOLOGY

The proposed methodology follows a structured development workflow that begins with user authentication and proceeds through the complete code editing, AI assistance, and execution pipeline. The system is designed to ensure that every interaction is secure, responsive, and enhanced by AI capabilities wherever applicable.

The workflow begins with user registration, during which the user provides credentials that are validated and securely stored. The password is hashed using bcrypt before storage in MongoDB. An OTP is sent to the user's registered email address for verification, ensuring that only legitimate users gain access to the platform. Upon subsequent logins, JWT tokens are generated and returned to the client for stateless session management.

Once authenticated, the user accesses the Monaco Editor-based coding interface. The user selects a programming language from the supported list and begins writing code. The editor provides real-time syntax highlighting and basic auto-completion powered by the Monaco language services. When the user invokes an AI feature, such as code generation or error explanation, the current code context and user prompt are sent to the Flask-based AI microservice, which forwards the request to the Google Gemini API and returns the AI-generated response to the frontend.

For code execution, the user submits the code through the frontend interface. The execution request is routed to the appropriate runtime environment depending on the selected language. The output, including standard output and error messages, is returned to the user in real time. When an execution error occurs, the user may request an AI explanation, which sends the error message and code context to the Gemini service for a detailed, plain-language explanation.

For code sharing, the user triggers the share functionality, which sends the current code snippet to the Flask file-sharing service. The service stores the snippet in Redis with a unique key and a configurable expiry time, and returns a shareable link to the user. Any recipient with the link can access the code snippet within the validity period. This mechanism enables secure, temporary, and lightweight code collaboration without requiring persistent storage.

V. RESULTS AND DISCUSSION

The implemented system was evaluated across multiple dimensions including functionality, performance, security, and usability. The platform successfully supports code editing and execution in over 16 programming languages, with the Monaco Editor providing consistent syntax highlighting and auto-completion across all supported languages. The AI features powered by Google Gemini demonstrated accurate and contextually relevant responses for code generation, error explanation, and refactoring tasks in the majority of test cases.

Performance testing showed that the microservices architecture enables independent scaling of individual components under load. The authentication service maintained low latency for login and token validation operations, while the AI service response times were dependent on the Gemini API latency, which averaged within acceptable ranges for interactive use. Redis-based code sharing proved highly efficient, with snippet retrieval times measured in milliseconds even under concurrent access.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Security evaluation confirmed that the JWT and bcrypt-based authentication mechanism resists common attack vectors including brute-force and token replay attacks. OTP-based email verification effectively prevented unauthorized account creation during testing. The Redis-backed code sharing mechanism correctly enforced expiry policies, ensuring that shared code snippets were inaccessible beyond their configured time limits.

User feedback collected during usability testing indicated that the AI-assisted features, particularly code generation and error explanation, were perceived as highly useful by both novice and experienced programmers. Participants reported that the platform reduced the time required to identify and fix bugs, and that the AI code generation feature accelerated the process of writing boilerplate code.

VI. CONCLUSION

This paper presented the design and development of an AI-enhanced Online Integrated Development Environment that combines browser-based code editing, multi-language execution, and intelligent AI assistance in a unified, secure platform. The integration of Google Gemini LLMs provides meaningful support for code generation, refactoring, auto-completion, and error explanation, making the platform valuable for both learners and professional developers.

The microservices architecture ensures that the system is scalable and maintainable, with each service independently deployable and testable. Robust security mechanisms including JWT authentication, bcrypt hashing, and OTP verification protect user data and prevent unauthorized access. The use of MongoDB and Redis together provides a high-performance, reliable data management layer that supports both persistent and ephemeral storage needs.

Future work may focus on introducing real-time collaborative editing features, expanding AI capabilities with fine-tuned models specific to programming assistance, integrating containerized sandboxed execution environments for improved security, and supporting additional languages and frameworks. The system can also be extended with learning analytics to track user progress and provide personalized coding recommendations.

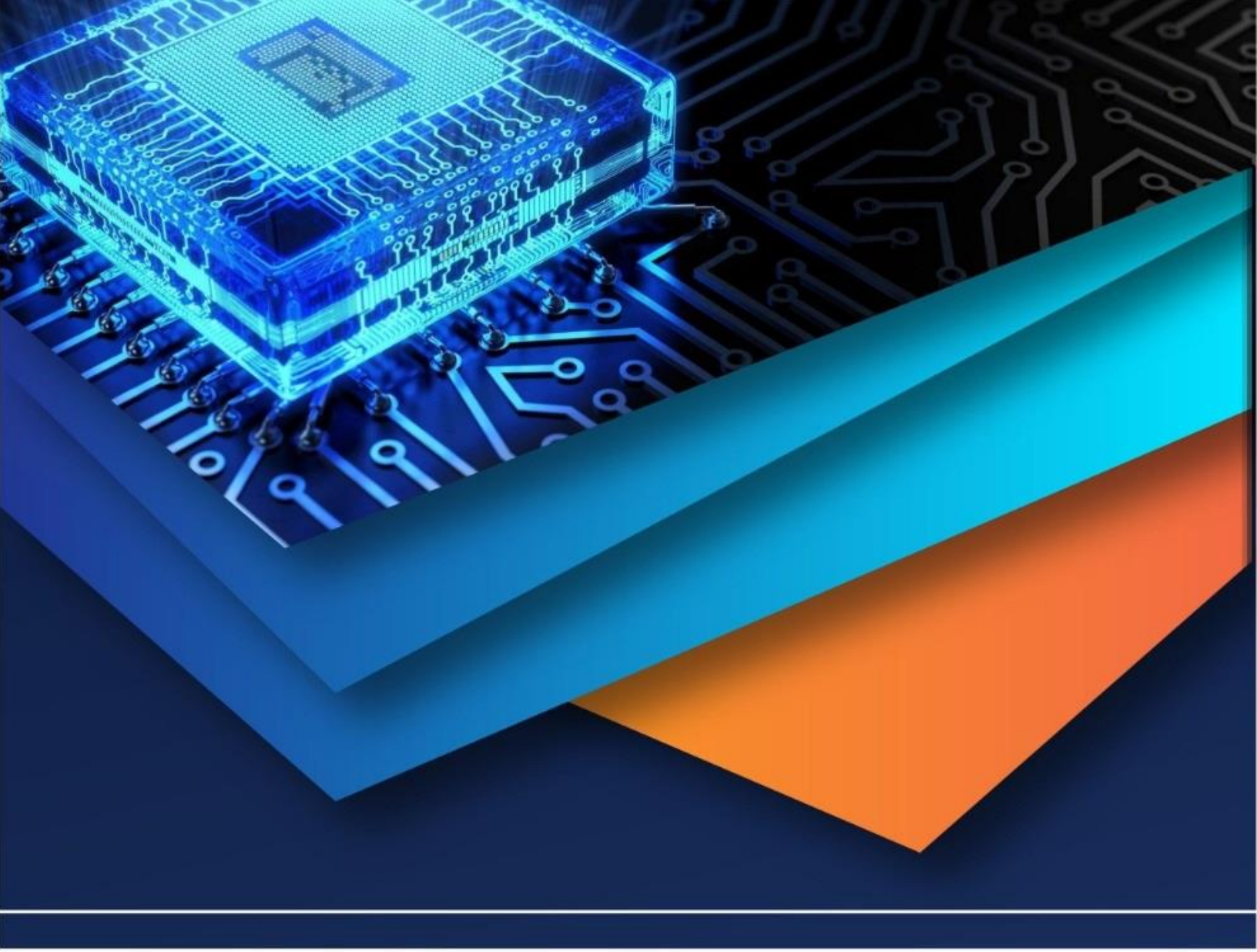
VII. ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the Department of Computer Applications, Aditya University, for providing the necessary support and resources to carry out this research work. The authors also thank the faculty members and guides for their valuable guidance and encouragement throughout the development of this paper.

REFERENCES

[1] Heine, “CodeMirror: A Versatile Text Editor for the Browser,” [Online]. Available: https://codemirror.net, 2020.
[2] Microsoft, “Visual Studio Code for the Web,” [Online]. Available: https://vscode.dev, 2021.
[3] M. Chen et al., “Evaluating Large Language Models Trained on Code,” arXiv preprint arXiv:2107.03374, 2021.
[4] D. Fried et al., “InCoder: A Generative Model for Code Infilling and Synthesis,” arXiv preprint arXiv:2204.05999, 2022.
[5] Microsoft, “Monaco Editor,” [Online]. Available: https://microsoft.github.io/monaco-editor/, 2023.
[6] R. Kang et al., “Cloud-Based Multi-Language Code Execution Environments: Architecture and Challenges,” in Proc. IEEE International Conference on Cloud Computing, 2022, pp. 215–223.
[7] C. Richardson, Microservices Patterns: With Examples in Java, Shelter Island, NY, USA: Manning Publications, 2018.
[8] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed., Sebastopol, CA, USA: O’Reilly Media, 2021.
[9] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” RFC 7519, Internet Engineering Task Force, May 2015.
[10] S. Sanfilippo, “Redis: An In-Memory Data Structure Store,” [Online]. Available: https://redis.io, 2023.
[11] Google, “Gemini API Documentation,” [Online]. Available: https://ai.google.dev, 2024.
[12] MongoDB,Inc.,“MongoDB Documentation,” [Online]. Available: https://www.mongodb.com/docs, 2023.

©IJRASET: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)