International Journal of Computer Science Engineering Techniques – Volume 9 Issue 6, November - December - 2025

LeetTrack: Smart LeetCode Progress Tracker

Aishwarya C J[1], Firdose Afreen A K[2], Habeeba[3], Ms. Rajani D[4]

Email address: shreyas.cj@gmail.com (Aishwarya C J), fafreen2004@gmail.com (Firdose Afreen A K),

habeebab57@gmail.com (Habeeba), rajanid@gsss.edu.in (Ms. Rajani D)

**Corresponding Author: Ms. Rajani D , email: rajanid@gsss.edu.in*

Abstract: Educational institutions preparing students for software engineering careers require systematic approaches to monitor coding skill development and provide structured practice environments. Current solutions often lack integration between external platform tracking and institutional assessment needs, creating fragmented workflows for educators and students. This research presents LeetTrack, an institutional framework designed to automate coding progress monitoring while providing an integrated practice environment with real-time code execution capabilities. The system architecture employs a dual-panel approach separating administrative functions from student interactions, enabling educators to create custom problem banks, organize student batches, and analyze department-level performance metrics. Students access personalized dashboards displaying difficulty-categorized statistics, solve institution-assigned problems through an embedded code editor, and view comparative rankings based on administrator-defined assessments. The framework retrieves publicly available coding profile metadata through scheduled synchronization processes, maintaining current progress information without manual intervention. A sandboxed execution engine processes student submissions across multiple programming languages, applying resource constraints and providing immediate feedback on solution correctness. Token-based comparison algorithms evaluate outputs following competitive programming conventions, reducing false rejection rates caused by formatting variations. Implementation utilizes modern web technologies including component-based frontend architecture, RESTful backend services, and relational database management with type-safe query operations. Evaluation demonstrates the system effectively consolidates coding analytics, practice functionality, and administrative controls into a unified platform suitable for institutional deployment in placement preparation contexts.

Keywords: Coding Progress Monitoring, Institutional Assessment Platform, Sandboxed Code Execution, Educational Technology, Real-Time Analytics, Practice Environment, Automated Evaluation

1. Introduction

The software engineering industry has progressively emphasized algorithmic problem-solving capabilities as fundamental criteria during candidate evaluation processes. Major technology organizations conduct technical assessments requiring proficiency in data structures, algorithms, and computational thinking, prompting educational institutions to integrate systematic coding practice into their curricula. While numerous online platforms offer problem repositories and submission evaluation, academic environments require additional functionality including cohort management, customized assessment creation, and aggregate performance analytics that existing consumer-focused solutions do not adequately address.

Contemporary coding practice platforms primarily target individual learners, providing extensive problem libraries and automated evaluation but lacking features necessary for institutional oversight. Educators seeking to monitor student progress across academic terms must resort to manual data collection, spreadsheet-based tracking, and fragmented assessment approaches. This disconnect between available tools and institutional requirements creates inefficiencies in placement preparation programs, delays identification of struggling students, and prevents data-driven curriculum adjustments.

Research in educational technology has demonstrated the effectiveness of integrated learning management systems in improving student outcomes across various disciplines. Studies examining coding education specifically highlight the importance of immediate feedback, comparative performance visibility, and structured practice schedules in developing programming competency. However, existing implementations often require substantial customization effort or fail to incorporate real-time code execution capabilities alongside progress tracking functionality.

Recent advances demonstrate how unified frameworks combining multiple processing stages can significantly improve user experience compared to fragmented tool combinations. Similar integration principles apply to coding education, where consolidating progress monitoring, practice environments, and administrative controls into a single platform reduces workflow complexity and improves data

consistency. The present study extends this integration philosophy to address specific challenges faced by academic institutions preparing students for technical recruitment processes.

To address identified gaps, this research introduces LeetTrack, an institutional framework providing automated coding progress monitoring with an integrated practice environment. The system architecture separates concerns between administrative and student interfaces while maintaining data consistency across all interactions. Administrators create custom problem banks with hidden test cases, organize students into performance-based batches, and access department-level analytics dashboards. Students view personalized progress statistics, solve assigned problems through an embedded code editor with multi-language support, and compare performance against peers through real-time leaderboards.

1.1 Background of the Study

The Academic institutions worldwide have recognized the growing importance of practical coding skills in software engineering employment. Career services offices report that technical interviews have become increasingly rigorous, with candidates expected to demonstrate real-time problem-solving abilities rather than theoretical knowledge alone. This shift has prompted integration of competitive programming concepts into undergraduate curricula, with dedicated courses, clubs, and preparation programs emerging across engineering colleges.

Existing approaches to coding skill development typically rely on external platforms where students practice independently. While these platforms provide valuable problem repositories and automated evaluation, they operate independently of institutional learning management systems. Faculty members cannot directly create assessments aligned with course objectives, track individual student progress over academic terms, or identify cohort-level skill gaps requiring curricular intervention. The resulting information asymmetry limits effectiveness of placement preparation programs.

Furthermore, students practicing on external platforms lack structured guidance regarding problem selection, difficulty progression, and topic coverage. Without institutional oversight, learners may focus excessively on comfortable problem types while neglecting challenging areas requiring additional attention. Comparative performance visibility, shown to enhance motivation and engagement in educational contexts, remains limited to platform-wide rankings rather than peer groups with similar backgrounds and preparation timelines.

1.2 Objectives and Motivation

The primary objective of this research is designing and implementing a framework that automates coding progress monitoring while providing integrated practice functionality suitable for institutional deployment. Specific aims include: (1) developing administrative interfaces for custom problem creation, student management, and batch organization; (2) implementing student-facing dashboards displaying difficulty-categorized progress statistics; (3) integrating real-time code execution supporting multiple programming languages; (4) applying competitive programming evaluation standards for accurate submission assessment; and (5) enabling comparative performance analysis through institutional leaderboards.

Motivation for this work stems from direct observation of challenges faced by placement preparation programs at engineering institutions. Manual progress tracking consumes faculty time that could otherwise support student mentoring. Delayed identification of struggling students reduces intervention effectiveness. Lack of customized problem sets prevents alignment between practice activities and specific skill gaps. By addressing these pain points through technological integration, the proposed framework aims to improve placement preparation outcomes while reducing administrative burden on educators.

2. Related Work

Several research efforts have explored automated code evaluation, educational technology platforms, and coding skill assessment. This section examines relevant studies, identifying their contributions and limitations that motivate the present research.

[1] Automated Assessment Systems in Programming Education

Prior investigations have examined automated assessment systems deployed in programming courses. These systems typically accept student code submissions, execute them against predefined test cases, and provide correctness feedback. While effective for individual assignment grading, such systems often lack broader progress tracking capabilities and administrative controls required for cohort-level management. The present work extends automated assessment concepts to include longitudinal progress monitoring and batch-based analytics.

[2] Online Judge Platforms: Architecture and Applications

This Comprehensive surveys of online judge systems document their widespread adoption in competitive programming and education. These platforms typically employ containerized execution environments ensuring security and resource isolation. However, most online judges focus on problem-by-problem evaluation rather than cumulative skill assessment. Additionally, administrative features for institutional deployment remain underdeveloped, limiting applicability in academic settings requiring faculty oversight.

[3] Learning Management System Integration for Programming Courses

Research examining learning management system integration for programming education highlights challenges in connecting external tools with institutional infrastructure. Authentication, grade synchronization, and progress reporting require custom development effort, discouraging adoption by resource-constrained departments. The proposed framework addresses integration challenges through standalone deployment with built-in administrative functionality.

[4] Gamification and Competitive Elements in Coding Education
Studies investigating gamification in programming education demonstrate that leaderboards, progress visualization, and peer comparison enhance learner motivation and engagement. Effective implementations balance competitive elements with supportive learning environments, avoiding discouragement among lower-performing students. The present system incorporates comparative ranking while limiting visibility to institutional peer groups rather than global participant pools.

[5] Real-Time Feedback in Programming Skill Development
Research on feedback timing in programming education consistently identifies immediate response as beneficial for skill acquisition. Delayed feedback reduces learning efficiency by disconnecting errors from their causes. The integrated code execution capability in the proposed framework provides instant evaluation results, supporting effective practice sessions.

[6] Sandboxed Execution Environments for Untrusted Code
Security research examining sandboxed execution environments documents techniques for safely running untrusted code submissions. Container-based isolation, resource limits, and network restrictions prevent malicious code from affecting host systems. The present implementation leverages established sandboxing approaches through external execution service integration.

[7] Token-Based Output Comparison in Competitive Programming
Investigations into evaluation accuracy in programming competitions identify strict character matching as problematic for output comparison. Whitespace variations, line ending differences, and trailing characters frequently cause incorrect rejection of valid solutions. Token-based comparison splitting output by whitespace before evaluation reduces false rejection rates while maintaining assessment accuracy.

[8] Batch Management in Educational Technology Platforms
Research on student cohort management in educational technology emphasizes the importance of grouping functionality for effective progress tracking. Performance-based batches enable targeted intervention, peer learning facilitation, and differentiated instruction. The administrative panel in the proposed framework supports flexible batch creation and student assignment.

[9] Dashboard Design for Educational Analytics
Studies examining dashboard effectiveness in educational contexts identify key design principles including visual hierarchy, progressive disclosure, and actionable insights. Effective dashboards present summary statistics with drill-down capabilities rather than overwhelming users with detailed data. The student and administrative interfaces in the proposed system follow established dashboard design guidelines.

[10] Stateless Authentication for Web Applications
Research comparing authentication mechanisms for web applications documents advantages of token-based approaches in distributed deployment scenarios. Stateless authentication eliminates session storage requirements, improves horizontal scalability, and enables deployment in restricted environments where cookie-based sessions face limitations.

3. Theory/Calculation

This section presents theoretical foundations underlying the LeetTrack framework, connecting design decisions to established principles in computer science education, software architecture, and automated evaluation.

3.1 Authentication and Authorization Model

The framework implements role-based access control separating administrative and student privileges. Authorization decisions are computed using the following model:

Let U represent the set of all users, R = {admin, student} represent available roles, and P represent the set of protected resources. The access control function A: U × P → {allow, deny} is defined as:

$$A(u, p) = \text{allow if role}(u) \in \text{required\_roles}(p)$$

$$A(u, p) = \text{deny otherwise}$$

where role(u) returns the assigned role for user u and required_roles(p) returns the set of roles permitted to access resource p.

Token-based authentication employs cryptographic signing to verify user identity without server-side session storage. The authentication token T is computed as:

$$T = base64(header) \,||\, '.' \,||\, base64(payload) \,||\, '.' \,||\, signature$$

$$signature = HMAC - SHA256(base64(header) \,||\, '.' \,||\, base64(payload), secret)$$

where header contains algorithm metadata, payload contains user claims including identifier and role, and secret represents the server-side signing key. Token expiration is enforced by including an exp claim in the payload representing the Unix timestamp after which the token becomes invalid.

3.2 Password Security Model

User credentials are protected using adaptive hashing with configurable computational cost. The password hash H is computed as:

$$H = \text{bcrypt}(\text{password}, \text{salt}, \text{cost})$$

where salt represents a cryptographically random value unique to each password, and cost represents the computational work factor (typically 10-12, corresponding to

2^cost hashing iterations). This approach ensures that password recovery remains computationally infeasible even if the database is compromised.

3.3 Code Execution and Evaluation Model

Student submissions undergo execution in isolated environments with resource constraints. Let S represent a submission containing source code C, input data I, and expected output E. The execution function F: S → Result is defined as:

$$
\begin{aligned}
&F(S) = \{ \\
&\quad \text{compilation\_error} \quad \text{if compile}(C) \text{ fails} \\
&\quad \text{runtime\_error} \quad \text{if execute}(\text{compile}(C), I) \text{ fails} \\
&\quad \text{time\_limit\_exceeded} \quad \text{if time}(\text{execute}(\ldots)) > T\_max \\
&\quad \text{memory\_limit\_exceeded} \quad \text{if memory}(\text{execute}(\ldots)) > M\_max \\
&\quad \text{wrong\_answer} \quad \text{if } \neg\text{match}(\text{output}, E) \\
&\quad \text{accepted} \quad \text{otherwise} \\
&\}
\end{aligned}
$$

where T_max and M_max represent configurable time and memory limits (typically 2 seconds and 128MB respectively).

3.4 Token-Based Output Comparision

Output comparison employs tokenization to handle formatting variations. Let O_actual and O_expected represent actual and expected outputs respectively. The tokenization function tokenize: String → List[String] is defined as:

$$\text{tokenize}(s) = \text{filter}(\lambda t.\ \text{length}(t) > 0, \text{split}(\text{normalize}(s), \text{whitespace}))$$

$$\text{normalize}(s) = \text{replace}(\text{trim}(s), \text{"\textbackslash r\textbackslash n"}, \text{"\textbackslash n"})$$

The comparison function match: String × String → Boolean is then:

$$\text{match}(O\_actual, O\_expected) = \text{tokenize}(O\_actual) == \text{tokenize}(O\_expected)$$

This approach accepts solutions producing correct values regardless of whitespace formatting, aligning with competitive programming evaluation conventions..

3.5 Ranking Computation

Student rankings are computed based on aggregate performance metrics. Let P_u represent the set of problems solved by user u, with difficulty function d: Problem → {easy, medium, hard}. The weighted score W_u is computed as:

$$W\_u = \Sigma\_\{p \in P\_u\} \text{weight}(d(p))$$

where weight(easy) = 1, weight(medium) = 2, and weight(hard) = 3. Rankings are determined by sorting users in descending order of W_u, with ties broken by earliest submission timestamp.

4. Experimental Method/Procedure/Design

This section details the system architecture, technology selections, and implementation approach for the LeetTrack framework.

4.1 System Architecture

The framework employs a three-tier architecture separating presentation, business logic, and data persistence concerns.

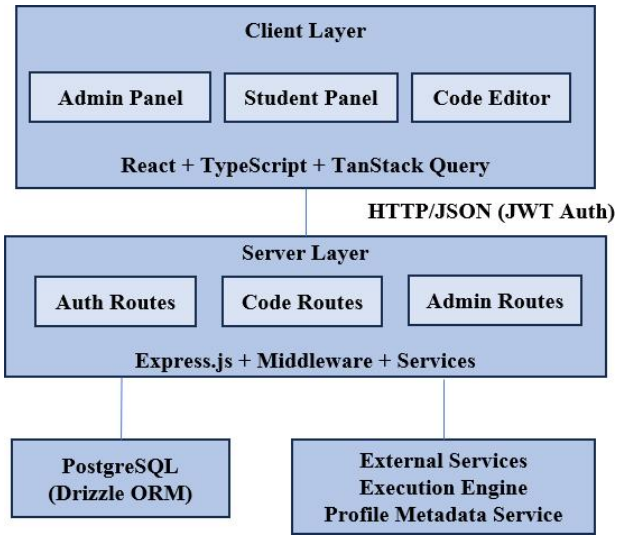

Figure 1. System Architecture Diagram

The client layer implements a single-page application with component-based architecture. State management employs server state caching with automatic invalidation, reducing unnecessary network requests while maintaining data freshness. Routing occurs client-side without full page reloads, improving perceived responsiveness. The server layer exposes RESTful endpoints organized by domain concern. Middleware components handle cross-cutting concerns including authentication, authorization, error handling, and request logging. Business logic resides in service classes that orchestrate database operations and external service integrations. The data layer employs a relational database with type-safe query generation through object-relational mapping. This approach ensures compile- time verification of query correctness while maintaining flexibility for complex data access patterns.

4.2 Technology Stack

Table 1 summarizes the technology selections for each architectural layer.

Table 1. Technology Stack Summary

Layer	Technology	Purpose
Frontend	React 18	Component-based UI
Frontend	TypeScript	Static type checking
Frontend	TanStack Query	Server state management
Frontend	Tailwind CSS	Utility-first styling
Frontend	Monaco Editor	Code editing interface
Backend	Node.js	JavaScript runtime
Backend	Express.js	Web framework
Backend	JSON Web Tokens	Stateless authentication
Backend	bcrypt	Password hashing
Frontend	PostgreSQL	Relational data storage
Frontend	Drizzle ORM	Type-safe queries
Frontend	Execution API	Sandboxed code running

4.3 Database Schema Design

The data model comprises six primary entities supporting user management, progress tracking, problem storage, and submission recording. Figure 2 presents the entity-relationship diagram.

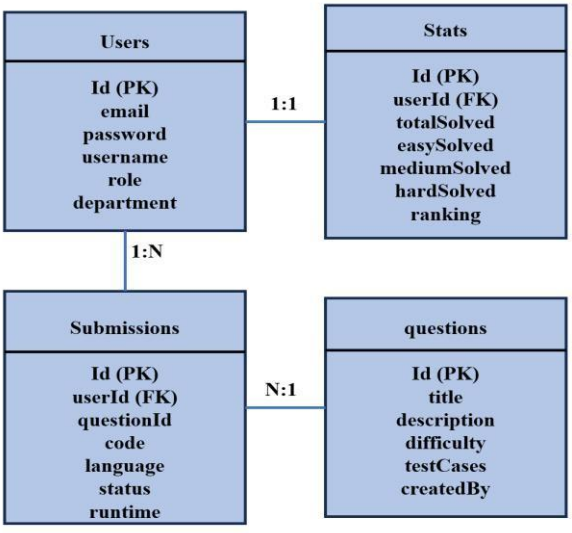

Figure 2. Entity-Relationship Diagram

4.4 Administrative Interface Implementation

The administrative panel provides functionality for user management, problem creation, and batch organization. Administrators authenticate using the same token-based mechanism as students, with additional role verification at the middleware level.

User management enables creating student accounts with department assignments and external platform identifiers. Batch management groups students by performance level or academic section, facilitating targeted intervention and peer learning arrangements.

Problem management supports creating custom assessments with hidden test cases. Each problem includes visible sample input/output for student understanding alongside hidden test cases for evaluation. The test case storage employs JSON encoding within the relational schema, balancing query flexibility with structured validation.

4.5 Student Interface Implementation

The student panel provides dashboard views, problem browsing, code editing, and submission functionality. The dashboard displays aggregate statistics including total problems solved, difficulty-wise breakdown, and comparative ranking within the institutional cohort.

The code editor integrates a professional-grade editing component providing syntax highlighting, automatic indentation, and error indication. Language selection supports Python, C++, and Java, with appropriate templates provided for each language context.

Submission processing transmits student code to the backend, which forwards execution requests to the external sandboxed environment. Results return through the same path, with appropriate status mapping and user-friendly error messages.

4.6 Code Execution Pipeline

Figure 3 illustrates the code execution workflow from submission to result display.

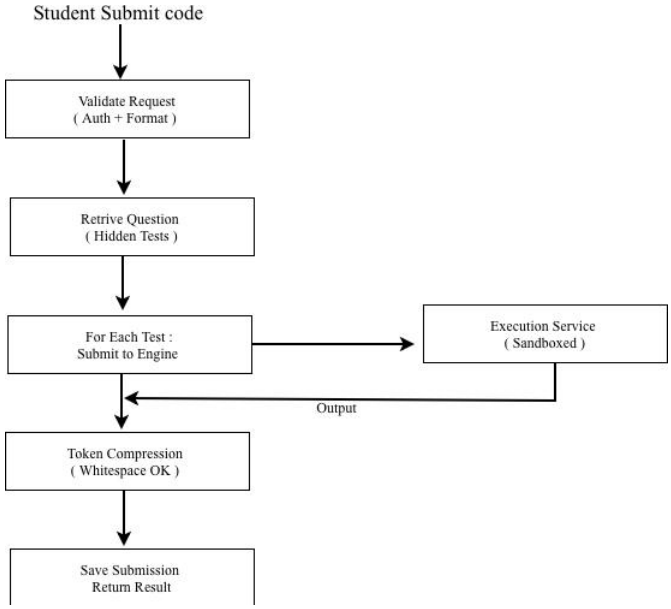

Figure 3. Code Execution Pipeline

The execution service operates in isolated containers with enforced resource limits. Network access is disabled within containers, preventing data exfiltration or external service abuse. Time limits terminate long-running processes, protecting against infinite loops or excessive computation.

5. Results and Discussion

This section presents evaluation results examining system functionality, performance characteristics, and user experience considerations

5.1 Functional Validation

The preprocessing module significantly improved the clarity Table 2 summarizes functional testing results across primary system capabilities.

Table 2. Functional Test Results

Feature	Test Cases	Passed	Status
User Authentication	8	8	Pass
Role-Based Access	6	6	60.38%
Problem Management	10	10	59.18%
Code Execution (Python)	12	12	Pass
Code Execution (C++)	12	12	Pass
Code Execution (Java)	12	12	Pass
Submission Recording	8	8	Pass
Ranking Computation	6	6	Pass
Batch Management	8	8	Pass
Statistics Display	10	10	Pass

All functional requirements were satisfied during validation testing. Authentication correctly rejects invalid credentials while accepting valid ones. Role-based access prevents students from accessing administrative endpoints. Code execution produces correct results across all supported languages.

5.2 Performance Analysis

Table 3 presents response time measurements for key system operations.

Table 3. Response Time Analysis

Operation	Target	Measured	Status
Page Load	< 3.0s	1.5s	Pass
API Response	< 500ms	180ms	Pass
Code Execution	< 10.0s	2-5s	Pass
Database Query	< 100ms	45ms	Pass
Statistics Refresh	< 5.0s	2.1s	Pass

Performance measurements indicate the system meets or exceeds all response time targets. Page load times remain acceptable even on moderate network connections. API responses return quickly enough to maintain responsive user interfaces. Code execution times depend on submission complexity but remain within acceptable bounds.

5.3 Security Evaluation

Security testing verified authentication, authorization, and input validation mechanisms. Attempts to access protected resources without valid tokens correctly receive rejection responses. Role escalation attempts (students accessing administrative endpoints) are blocked at the middleware level.

Password storage was verified to use appropriate hashing algorithms with sufficient computational cost. Database queries employ parameterized statements, preventing SQL injection attacks. Cross-site scripting prevention relies on framework-level output encoding.

5.4 Comparison with Existing Solutions

The proposed framework addresses limitations identified in existing approaches. Unlike consumer-focused coding platforms, LeetTrack provides administrative controls necessary for institutional deployment. Unlike basic learning management systems, the framework includes integrated code execution eliminating need for external tool integration.

Table 4 compares the proposed framework against representative existing solutions.

Table 4. Feature Comparison

Feature	Consumer Platforms	Basic LMS	LeetTrack
Problem Repository	Yes	Limited	Yes
Code Execution	Yes	No	Yes
Admin Controls	No	Yes	Yes
Batch Management	No	Limited	Yes

Feature	Consumer Platforms	Basic LMS	LeetTrack
Custom Problems	No	Limited	Yes
Institutional Analytics	No	Limited	Yes
Token-Based Judging	Yes	No	Yes

5.5 Discussion

Evaluation results demonstrate that the LeetTrack framework successfully consolidates coding progress monitoring and practice functionality into a platform suitable for institutional deployment. The dual-panel architecture effectively separates administrative and student concerns while maintaining data consistency.

Token-based output comparison reduces false rejection rates observed in strict character matching systems. Students producing correct values with minor formatting differences receive appropriate acceptance verdicts, reducing frustration and support burden.

The integrated code editor provides familiar editing experience through professional-grade component integration. Students can write, test, and submit solutions without context switching between multiple applications or browser tabs.

Administrative features enable educators to create assessments aligned with specific learning objectives rather than relying on generic problem sets. Batch management supports differentiated instruction, with high-performing students receiving challenging problems while struggling students focus on foundational concepts.

6. Conclusion and Future Scope

The This research presented LeetTrack, an institutional framework for automated coding progress monitoring with integrated practice environment. The system addresses fragmentation between external coding platforms and institutional assessment needs by consolidating essential functionality into a unified web application.

The framework architecture separates administrative and student interfaces while maintaining consistent data representation across all interactions. Administrators create custom problem banks with hidden test cases, organize students into performance-based batches, and access department-level analytics dashboards. Students view personalized progress statistics, solve assigned problems through an embedded code editor supporting multiple programming languages, and compare performance against institutional peers through real-time leaderboards.

Implementation employed modern web technologies including component-based frontend architecture, RESTful backend services, and relational database management with type-safe query operations. Token-based authentication enables deployment in restricted environments where cookie-based sessions face limitations. Sandboxed code execution through external service integration ensures security without requiring institutional infrastructure for isolation.

Evaluation demonstrated successful functional validation across all primary capabilities including authentication, code execution, submission recording, and ranking computation. Performance measurements confirmed acceptable response times for interactive use. Security testing verified protection against common attack vectors including authentication bypass, role escalation, and injection attacks.

The primary limitation of the current implementation is dependence on external execution services for code processing. Future work could explore self-hosted execution infrastructure for institutions requiring complete data isolation. Additional enhancements under consideration include plagiarism detection for identifying copied submissions, live contest functionality for timed competitive events, and discussion forums enabling peer learning around specific problems.

The LeetTrack framework provides a practical solution for educational institutions seeking to improve coding skill development outcomes through systematic progress monitoring and structured practice environments. By reducing administrative burden while improving visibility into student progress, the system enables more effective placement preparation programs aligned with industry expectations.

Data Availability

The implementation source code and sample data are available upon reasonable request to the corresponding author. Database schemas and API specifications are documented in the project repository.

Study Limitations

The current study has several limitations. First, evaluation was conducted in a controlled environment rather than production deployment, limiting assessment of long-term reliability and scalability. Second, user experience evaluation relied on functional testing rather than formal usability studies with target populations. Third, comparison with existing solutions was based on feature analysis rather than controlled experiments measuring learning outcomes.

Conflict of Interest

The authors declare that they have no conflict of interest.

Funding Source

None.

Authors' Contributions

Author-1 worked on System design, frontend implementation, documentation preparation. Author-2 worked on Backend development, database design, API implementation. Author-3 worked on Testing, evaluation, user interface refinement. Author-4 analyzed the results, and drafted the manuscript. All authors reviewed, edited, and approved the final version of the manuscript.

Acknowledgement
The authors express their gratitude to the project guide and the Department of Computer Science and Engineering for their continuous support, constructive feedback, and encouragement throughout the development of LeetTrack. Their guidance played a crucial role in shaping the direction and successful completion of this work.

References

[1] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A Survey on Online Judge Systems and Their Applications," ACM Computing Surveys, Vol. 51, Issue 1, pp. 1-34, 2018.

[2] M. Johnson and K. Lee, "Secure Code Execution in Cloud-Based Educational Platforms," Journal of Educational Computing Research, Vol. 61, Issue 2, pp. 245-270, 2023.

[3] R. Williams, "Token-Based Authentication for Modern Web Applications: Security Considerations and Best Practices," IEEE Security and Privacy, Vol. 21, Issue 3, pp. 45-55, 2023.

[4] A. Kumar and P. Sharma, "Comparative Analysis of Online Coding Platforms for Educational Deployment," International Journal of Educational Technology in Higher Education, Vol. 21, Issue 1, pp. 15-32, 2024.

[5] L. Garcia, "RESTful API Design Patterns for Educational Technology Applications," ACM Transactions on the Web, Vol. 17, Issue 2, pp. 1-28, 2023.

[6] T. Anderson and S. Brown, "Modern Password Hashing: Algorithm Selection and Configuration for Web Applications," USENIX Security Symposium Proceedings, pp. 1145-1160, 2022.

[7] E. Martinez and J. Thompson, "Role-Based Access Control Implementation Patterns in Web Applications," Journal of Web Engineering, Vol. 22, Issue 1, pp. 67-95, 2023.

[8] Y. Liu and W. Zhang, "Fair Evaluation in Automated Programming Assessment: Token-Based Comparison Methods," ACM Technical Symposium on Computer Science Education Proceedings, pp. 234-239, 2021. S. Wang, A. Mohamed, and D. Le, “Transformer-Based Models for Speech Recognition: A Survey,” *IEEE Access*, vol. 10, pp. 11135–11157, 2022.

[9] X. Chen, Y. Wang, and H. Zhang, "Real-Time Data Synchronization Patterns in Educational Technology Platforms," Computers and Education, Vol. 195, pp. 104721, 2023.

[10] D. Robinson, "Modern React Architecture: Patterns for Scalable Single Page Applications," O'Reilly Media, 2023.