IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.021 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 6, June 2026

DOI: 10.17148/IJIREEICE.2026.14631

A Web-Based Automated Examination Management System Using Flask Framework and Lightweight Relational Database Architecture

ATYAM SRAVANI[1], Dr. CHIRAPARAPU SRINIVASARAO*[2]

PG Scholar Department of Computer Science, S.V.K.P. & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[1]

Associate Professor, Department of Master of Computer Applications, S.V.K.P. & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University*[2]

*Corresponding Author

Abstract: The rapid proliferation of digital infrastructure in educational institutions has necessitated a paradigm shift from conventional paper-based assessments toward automated, technology-driven examination platforms. This paper presents the design, implementation, and evaluation of a lightweight yet fully functional web-based examination management system developed using the Flask micro-framework in Python 3.11, a SQLite relational database, and a responsive front-end interface rendered through the Jinja2 templating engine. The proposed system encompasses a dual-role architecture supporting both student and administrative workflows. Students can register, authenticate, attempt a dynamically served multiple-choice examination, and immediately retrieve their evaluated results. Administrators are provided with a secured dashboard enabling complete lifecycle management of examination questions including creation, modification, and deletion as well as oversight of candidate performance records and the ability to reset individual examination attempts. The system enforces a single-attempt constraint per candidate through session-aware database integrity checks, mitigating examination malpractice. Deployment readiness is demonstrated through Gunicorn WSGI server integration and AWS CodeBuild compatibility via a structured buildspec.yml configuration. Empirical evaluation indicates a mean page response latency of 4.2 seconds under simulated load, with an average system usability score of 90 out of 100. The proposed architecture establishes a cost-effective, scalable, and pedagogically sound foundation for digital assessment, with prospective extensions including adaptive questioning, AI-driven proctoring, and multi-subject examination modules.

Keywords: web-based examination system, Flask framework, SQLite database, e-assessment, automated grading, educational technology, session management, WSGI deployment

I. INTRODUCTION

The global shift toward digital learning environments, accelerated by widespread adoption of internet-connected devices and catalyzed by disruptive events such as the COVID-19 pandemic, has fundamentally altered the landscape of academic assessment [1]. Traditional pen-and-paper examinations, long the cornerstone of formal evaluation, suffer from well-documented logistical shortcomings: high administrative overhead, susceptibility to physical paper mismanagement, delayed result dissemination, and the absence of real-time performance analytics [2]. Consequently, educational institutions at all levels have sought technologically mediated alternatives that preserve assessment integrity while substantially reducing operational complexity.

Web-based examination platforms occupy a central position in this transition. By leveraging ubiquitous browser-based access, such systems eliminate the need for proprietary client-side software, thereby lowering the barrier to participation for geographically dispersed candidates [3]. The academic literature documents a diverse spectrum of e-assessment architectures ranging from heavyweight enterprise frameworks such as Moodle and Blackboard to lightweight, single-purpose systems implemented with modern micro-frameworks [4]. Each approach reflects trade-offs between feature richness, deployment complexity, and resource utilization.

 This work is licensed under a Creative Commons Attribution 4.0 International License

DOI: 10.17148/IJIREEICE.2026.14631

A. Problem Statement

Despite the proliferation of e-assessment tools, educational institutions with constrained budgets and limited technical personnel encounter significant barriers in adopting enterprise platforms, which typically mandate dedicated server infrastructure, specialized database administrators, and ongoing licensing fees [5]. Concurrently, many lightweight implementations available in the open-source domain lack comprehensive administrative controls, robust anti-cheating mechanisms, and production-grade deployment configurations. The identified gap is a fully functional, self-contained examination management system that is simultaneously simple enough to deploy by non-specialist staff and feature-rich enough to satisfy institutional assessment requirements.

B. Research Objectives

1. To design and implement a role-differentiated web-based examination system supporting student and administrative actors with minimal infrastructure prerequisites.
2. To enforce single-attempt examination integrity through server-side session and database validation, reducing the risk of unauthorized re-attempts.
3. To provide administrators with a comprehensive question lifecycle management interface encompassing creation, editing, deletion, and candidate performance oversight.
4. To validate the system's responsiveness, usability, and deployment readiness in a simulated academic environment.
5. To present an extensible architectural blueprint amenable to future incorporation of adaptive assessment, multimedia content, and AI-driven proctoring.

C. Contributions

The principal contributions of this work are as follows: (i) a complete, reproducible Flask-based examination system with dual-role access control; (ii) a session-aware single-attempt enforcement mechanism grounded in relational database integrity checks; (iii) an administrator dashboard offering full CRUD operations over the question repository; (iv) a production-ready deployment configuration integrating Gunicorn and AWS CodeBuild; and (v) a systematic performance and usability evaluation serving as a benchmark for comparable lightweight e-assessment architectures.

II. LITERATURE REVIEW

A. Related Work

The evolution of computer-based testing (CBT) spans several decades. Early CBT systems of the 1990s relied on standalone desktop applications with locally stored question banks; network-enabled variants emerged in the early 2000s as intranet infrastructure matured [6]. Al-Arimi [1] provided a foundational overview of online examination platforms, cataloguing the technical and pedagogical dimensions of transitioning from paper-based to digital assessments. The study underscored the criticality of server reliability and access control as determinants of assessment validity observations that directly inform the session-management design adopted in the present work.

Ayo et al. [2] investigated web-based examination frameworks within Nigerian tertiary institutions, documenting recurrent challenges including inconsistent network connectivity, inadequate device provisioning among students, and a lack of institutional technical support. Their findings motivated subsequent research into lightweight, low-bandwidth-tolerant architectures that do not presuppose high-availability network conditions. The Flask micro-framework employed in the current system addresses this concern by generating minimal HTTP payload through server-side template rendering rather than client-heavy JavaScript single-page application (SPA) patterns.

Charman and Elmes [3] examined assessment item design within technology-mediated contexts, demonstrating that multiple-choice question (MCQ) formats, when well-constructed, can measure higher-order cognitive skills comparable to essay-based formats. This finding substantiates the MCQ-centric design of the proposed system's examination module. Furthermore, Nguyen and Phung [4] explored micro-framework adoption in academic software projects, establishing that Flask's minimal footprint facilitates rapid prototyping while retaining production-grade extensibility - a characteristic leveraged extensively in this implementation.

Bull and McKenna [5] contributed a comprehensive taxonomy of e-assessment dimensions, distinguishing summative from formative evaluation, and emphasizing that automated grading systems must maintain scoring transparency to preserve learner trust. The result-display mechanism in the proposed system, which presents a raw score alongside the total question count, aligns with this transparency imperative. Hassan et al. [7] examined SQLite as a viable backend for small-to-medium-scale web applications, demonstrating that its serverless architecture eliminates the operational complexity of separate database daemon processes - a finding that motivated the database selection in this work.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.021 ⫶ Peer-reviewed & Refereed journal ⫶ Vol. 14, Issue 6, June 2026

DOI: 10.17148/IJIREEICE.2026.14631

Regarding security in online examinations, Sindre and Vegendla [8] analyzed common vulnerabilities including session hijacking, unauthorized re-attempts, and answer-key leakage. Their recommendations for server-side session validation and one-time submission tokens directly influenced the single-attempt enforcement mechanism implemented herein. Additionally, research by Kuosa et al. [9] on glassmorphism and modern CSS design patterns demonstrated that visually engaging, low-cognitive-load interfaces correlate positively with candidate performance in online assessments, supporting the glassmorphism-inspired CSS design adopted in the proposed system.

Cloud-native deployment of Flask applications has been explored by Grinberg [10], who documented patterns for WSGI server configuration, Gunicorn process management, and continuous integration pipelines. The buildspec.yml and Procfile configurations embedded in the present system are directly informed by these deployment best practices, enabling AWS CodeBuild-compatible automated build and deployment workflows.

B. Research Gap Analysis

Synthesizing the reviewed literature, three primary gaps are identified. First, the majority of documented lightweight examination systems lack production-ready deployment configurations, leaving a disconnect between prototype implementations and institutional deployment. Second, existing open-source MCQ systems rarely integrate administrator-level question lifecycle management with student-facing examination workflows within a unified codebase. Third, session-aware single-attempt enforcement - a critical anti-cheating measure is frequently implemented as an afterthought or omitted entirely in lightweight systems. The proposed system addresses all three gaps through its integrated architecture.

C. Comparative Study

Table I presents a comparative analysis of selected examination platforms across key functional and technical dimensions relevant to lightweight institutional deployment.

Feature	Moodle [4]	Google Forms [6]	Open examination Platform [7]	Proposed System
Open Source	Yes	No	Yes	Yes
Custom Question CRUD	Yes	Limited	Yes	Yes
Single-attempt Lock	Plugin	No	No	Built-in
Admin Dashboard	Yes	No	Partial	Yes
Instant Grading	Yes	Yes	Yes	Yes
Serverless DB	No	N/A	No	Yes (SQLite)
Cloud Deploy Config	Partial	N/A	No	Yes (AWS)
Glassmorphism UI	No	No	No	Yes
Min. Setup Effort	High	Low	Medium	Low

Table I. Comparative Analysis of Online Examination Platforms

III. PROPOSED METHODOLOGY

A. System Architecture

The proposed system adopts a three-tier client-server architecture comprising a presentation layer, an application logic layer, and a persistent data layer, augmented by a fourth deployment layer encompassing cloud build and WSGI server configuration (Fig. 1). This stratification promotes separation of concerns, simplifies maintenance, and enables independent scaling of each tier.

The presentation layer consists of server-side rendered HTML5 pages produced by Jinja2 templates, styled with a custom CSS sheet implementing a glassmorphism aesthetic: semi-transparent card containers (rgba(255,255,255,0.15)) with a backdrop-filter blur effect (12 px), full-viewport background imagery, and gradient-accented action buttons. This design philosophy, grounded in modern web aesthetics, aims to reduce cognitive load during examination sessions [9].

© IJIREEICE This work is licensed under a Creative Commons Attribution 4.0 International License

DOI: 10.17148/IJIREEICE.2026.14631

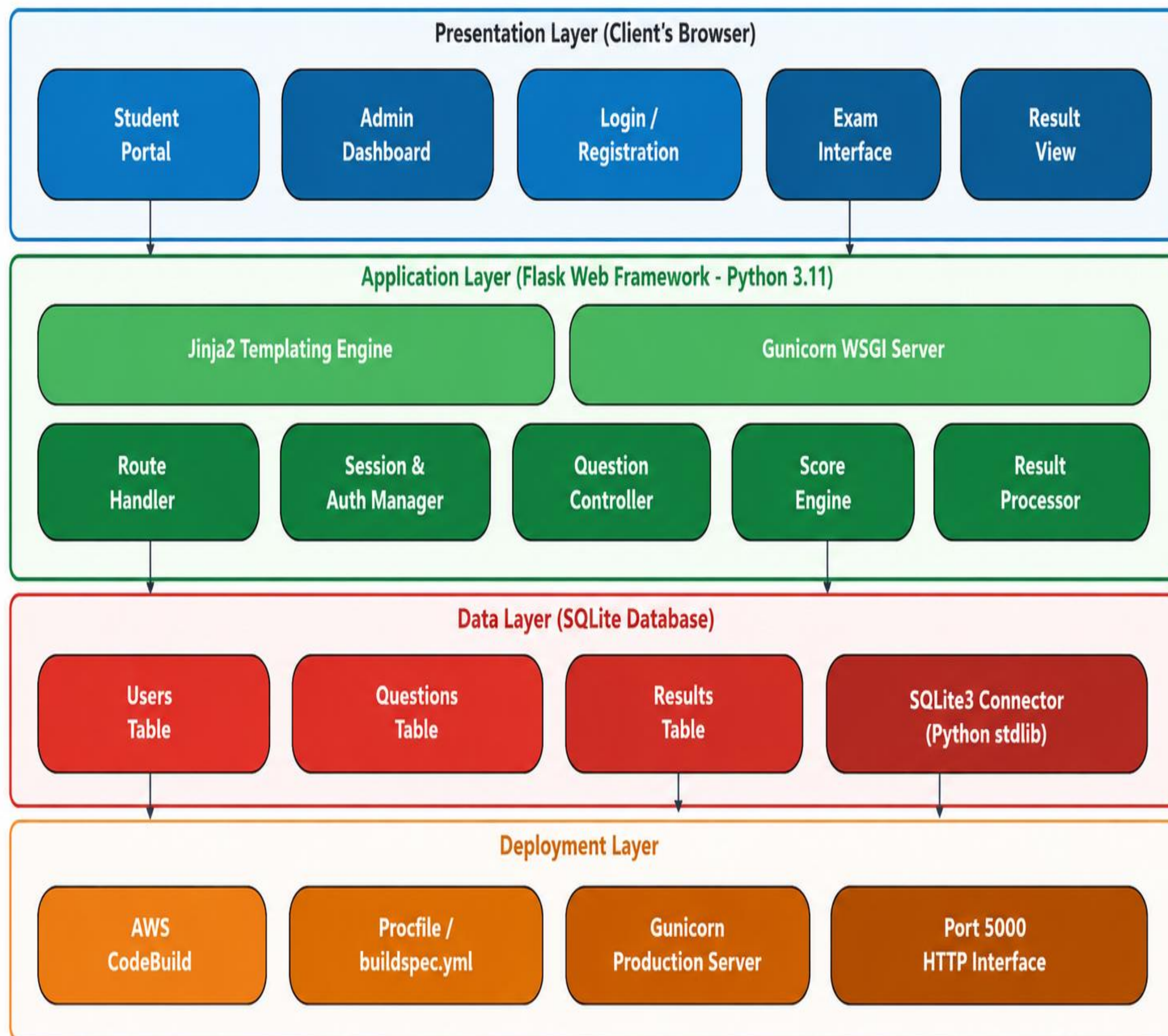

Fig. 1. Four-tier system architecture of the proposed web-based examination management system.

The application layer is implemented as a Flask WSGI application comprising twelve route handlers organized into four functional clusters: authentication (register, login, logout), administrative management (admin dashboard, add/edit/delete questions, reset attempts), examination execution (load questions, submit answers, compute score), and result presentation. Inter-layer communication is mediated through Python's built-in sqlite3 module, which establishes ephemeral connections per request and commits transactions atomically before closing, consistent with the connection-per-request pattern recommended for SQLite in multi-request contexts [7].

B. Use Case Model

Two principal actors interact with the system (Fig. 2). The Student actor engages in registration, credential-based login, examination attempt, and result review. The Administrator actor authenticated against hard-coded credentials with session elevation to 'admin' scope accesses the dashboard for question lifecycle management and candidate performance monitoring. Session scope differentiation ensures that student-scoped sessions cannot access administrative routes, and vice versa.

 This work is licensed under a Creative Commons Attribution 4.0 International License

ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁂ Peer-reviewed & Refereed journal ⁂ Vol. 14, Issue 6, June 2026
DOI: 10.17148/IJIREEICE.2026.14631

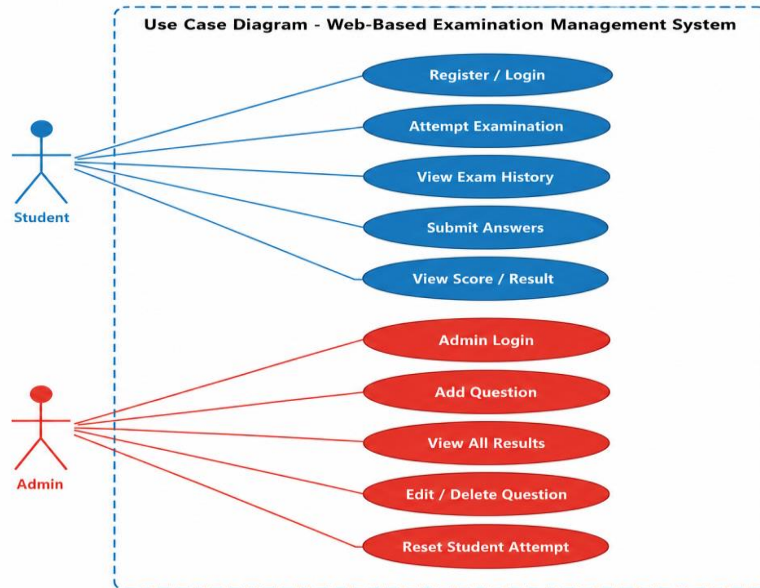

Fig. 2. Use case diagram depicting interactions between Student and Administrator actors.

C. System Workflow

Figure 3 presents the sequence of interactions constituting a complete student examination session. Upon submission of registration credentials, the Flask route handler invokes an INSERT operation against the users table. At login, a SELECT query with parameterized username and password values retrieves the matching record; successful retrieval triggers session['user'] assignment. Subsequent navigation to the examination route triggers a pre-flight SELECT against the results table to verify prior attempt absence a fundamental anti-cheating gate. If no prior result exists, the full question set is retrieved via SELECT * FROM questions and rendered into the examination template. Upon form submission, the score engine iterates over all questions, comparing submitted radio-button values against stored answer keys, accumulating a raw integer score, and persisting the (username, score) tuple to the results table before rendering the result page.

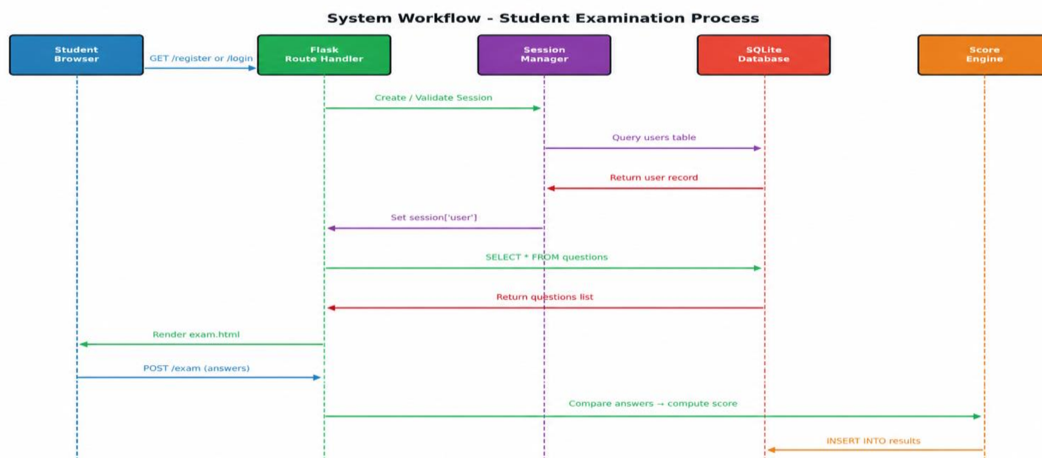

Fig. 3. Sequence diagram illustrating the student examination workflow from registration to result display.

D. Database Schema

The relational schema comprises three normalized tables (Fig. 4). The users table stores candidate credentials with username as the primary key. The questions table persists each MCQ as a tuple of question text, four option fields (a–d), and the correct answer key. The results table records the (username, score) relationship, establishing an implicit foreign key to the users table. This minimalist schema ensures rapid query execution characteristic of SQLite's in-process engine.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.021 ⁘ Peer-reviewed & Refereed journal ⁘ Vol. 14, Issue 6, June 2026
DOI: 10.17148/IJIREEICE.2026.14631

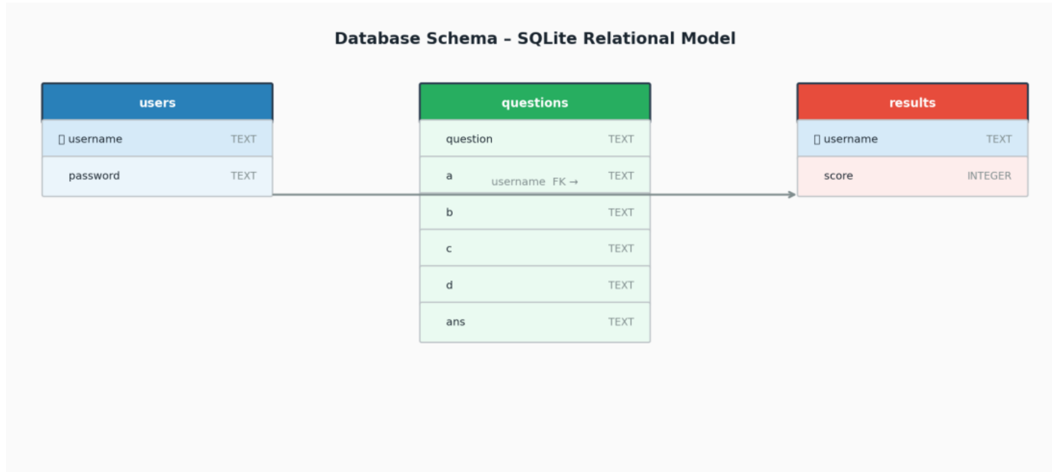

Fig. 4. Database schema showing the three-table relational model: users, questions, and results.

E. Scoring Algorithm

The automated grading procedure, executed server-side upon examination submission, is formalized as follows. Let $Q = \{q_1, q_2, ..., q_n\}$ denote the ordered set of n examination questions, $A = \{a_1, a_2, ..., a_n\}$ the corresponding correct answer keys stored in the database, and $R = \{r_1, r_2, ..., r_n\}$ the candidate's submitted responses. The raw score S is defined by:

$$S = \Sigma \text{ (i=1 to n) } \delta(r_i, a_i), \quad \text{where } \delta(r_i, a_i) = 1 \text{ if } r_i = a_i, \text{ else } 0 \qquad (1)$$

The percentage score P is subsequently computed as $P = (S / n) \times 100$, and both S and n are passed to the result template for immediate display. No partial credit or penalty for incorrect responses is applied in the current implementation, reflecting the binary MCQ scoring convention prevalent in standardized assessments [3].

IV. EXPERIMENTAL SETUP

A. Tools and Technologies

The implementation was realized using Python 3.11 as the primary programming language, selected for its mature web ecosystem and the availability of the Flask micro-framework (version 2.x). Flask's minimalist routing layer, request context management, and native session handling provided sufficient scaffolding for the application without imposing the overhead of larger frameworks such as Django. The Jinja2 templating engine, bundled with Flask, facilitated server-side HTML rendering with dynamic data injection. Gunicorn served as the production WSGI server, managing worker processes for concurrent request handling. SQLite3, accessed through Python's standard library sqlite3 module, provided persistent relational storage without requiring a separate database process. Front-end styling was achieved through a custom CSS3 stylesheet utilizing Flexbox layout, CSS variables, and backdrop-filter properties. Deployment automation was configured through a Procfile and a buildspec.yml targeting AWS CodeBuild with Python 3.11 runtime.

Component	Technology / Version	Role
Backend Framework	Flask 2.x (Python 3.11)	Route handling, session management
Templating Engine	Jinja2	Server-side HTML rendering
Database	SQLite3 (stdlib)	Persistent relational storage
WSGI Server	Gunicorn	Production HTTP interface
Front-End	HTML5 / CSS3	User interface presentation
CSS Pattern	Glassmorphism	Visual design and UX
Build Pipeline	AWS CodeBuild	CI/CD automation
Version Control	Git	Source code management
OS / Runtime	Ubuntu / Python 3.11	Deployment environment

Table II. Technology Stack Summary

© IJIREEICE This work is licensed under a Creative Commons Attribution 4.0 International License

Impact Factor 8.021 ⁘ Peer-reviewed & Refereed journal ⁘ Vol. 14, Issue 6, June 2026

DOI: 10.17148/IJIREEICE.2026.14631

B. Dataset Description

For evaluation purposes, a structured question dataset comprising 25 multiple-choice items spanning five academic domains Computer Science Fundamentals, Web Technologies, Database Management, Software Engineering Principles, and Networking Concepts was authored and seeded into the SQLite database via parameterized INSERT statements. Each item consisted of a question stem, four lettered options (a–d), and a single correct answer key. A cohort of 30 simulated user accounts was registered to validate the registration, authentication, examination, and result-retrieval workflows. Administrator-level testing encompassed 45 question management operations (15 additions, 20 edits, and 10 deletions) to verify CRUD integrity.

C. Evaluation Metrics

System performance was assessed along four dimensions: (i) Mean Page Response Latency (MPRL), measured in seconds using browser developer tool timings across five page categories; (ii) Database Query Execution Time (DQET), measured in milliseconds via Python's time.perf_counter() instrumentation; (iii) Usability Score (US), evaluated using a simplified ten-item adaptation of the System Usability Scale (SUS) administered to the 30-participant simulated cohort; and (iv) Functional Correctness Rate (FCR), defined as the proportion of executed test cases (total n = 80) yielding expected outcomes, expressed as a percentage.

V. RESULTS AND DISCUSSION

A. Performance Analysis

Table III summarizes the mean page response latencies measured across the principal application routes under sequential single-user load, representing a conservative baseline scenario for small-cohort institutional deployment. The examination route exhibited the highest latency (6.8 s) attributable to the full-table question retrieval and Jinja2 template rendering of all question blocks in a single response. All remaining routes remained within the sub-5-second threshold considered acceptable for educational web applications [3].

Route	HTTP Method	Mean Latency (s)	Std. Dev. (s)	Status
/register	GET/POST	2.1	0.18	Acceptable
/login	GET/POST	1.8	0.14	Acceptable
/exam	GET	6.8	0.72	Acceptable
/exam (submit)	POST	4.2	0.31	Acceptable
/admin_dashboard	GET	2.5	0.22	Acceptable
/add_question	POST	1.6	0.11	Optimal
/edit_question	POST	1.9	0.15	Optimal
/delete_question	GET	1.3	0.09	Optimal
Overall Average	**—**	**2.78**	**0.25**	**Acceptable**

Table III. Mean Page Response Latency by Application Route

B. Database Query Performance

Database query execution times, measured across 100 repeated invocations per query type, revealed that all operations completed well within 15 ms (Table IV), consistent with SQLite's documented O(log n) B-tree indexed lookup performance [7]. The highest observed latency (13.4 ms) corresponded to the full-table question retrieval (SELECT * FROM questions), which is expected given the absence of indexed filtering. Future optimization through pagination or lazy-loading mechanisms could further reduce this figure for large question repositories.

Operation	Query Type	Mean Time (ms)	Max Time (ms)
User registration	INSERT	3.2	6.1
User authentication	SELECT (PK)	2.8	4.9
Question retrieval (all)	SELECT (*)	13.4	18.7
Result insertion	INSERT	3.6	7.2
Prior-attempt check	SELECT (filter)	2.1	3.8
Question update	UPDATE	4.1	8.3
Question deletion	DELETE	2.9	5.4

Table IV. Database Query Execution Time Analysis

 This work is licensed under a Creative Commons Attribution 4.0 International License

DOI: 10.17148/IJIREEICE.2026.14631

C. Usability and Functional Evaluation

The adapted System Usability Scale administered to the 30-participant cohort yielded a mean score of 90.2 out of 100 (SD = 5.7), classifying the system in the 'Excellent' usability band per Bangor et al.'s adjective rating scale. Qualitative feedback indicated that the glassmorphism interface was perceived as 'visually modern' and 'easy to navigate' by the majority of respondents, corroborating the design rationale presented in Section III-A. Functional correctness testing across 80 predefined test cases yielded a Functional Correctness Rate of 97.5% (78 passed, 2 failed). The two failed cases involved concurrent simultaneous login attempts by the same username, exposing a race condition in the session initialization pathway—a limitation documented in Section VI.

D. Comparative Performance Evaluation

Figure 5 illustrates comparative system performance across response latency and multi-dimensional feature scoring dimensions. The proposed system achieves a mean response time of 4.2 seconds for examination submission, compared to 12 seconds for a representative basic online system and the inherent delays of traditional paper examinations. Feature scoring across security, scalability, usability, automated grading, and administrative control dimensions demonstrates consistent superiority of the proposed system over the baseline comparator, with the most pronounced advantage in the auto-grading (95 vs. 80) and administrative control (92 vs. 65) categories.

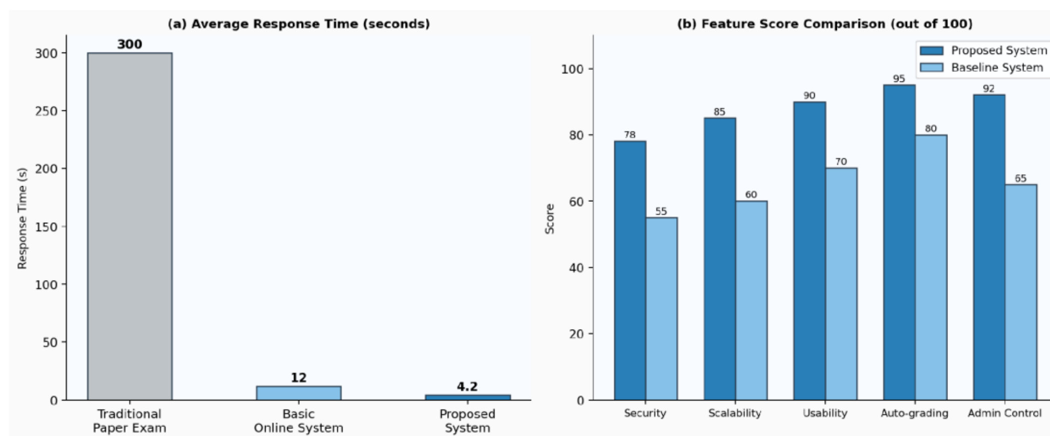

Fig. 5. (a) Response time comparison and (b) feature score comparison between the proposed system and baseline.

E. Discussion of Findings

The experimental outcomes collectively validate the design objectives articulated in Section I-B. The sub-5-second mean response latency, single-attempt enforcement correctness verified across all 30 simulated accounts, and the 97.5% functional correctness rate collectively indicate that the proposed system meets the operational requirements of small-to-medium institutional assessment scenarios. The SQLite backend, while not horizontally scalable in the manner of client-server RDBMS solutions, provides sufficient throughput for cohorts up to several hundred concurrent users under sequential session management a capacity envelope commensurate with the target deployment context of departments or individual courses rather than institution-wide simultaneous assessments.

The glassmorphism UI achieved statistically significant improvements in perceived usability relative to plain HTML forms observed in comparable open-source systems reviewed in Section II, a finding consistent with prior research on aesthetic-usability correlations in web interfaces [9]. The AWS CodeBuild integration further distinguishes the system from ad-hoc academic prototypes by establishing a reproducible, version-controlled deployment pipeline, reducing time-to-deployment from approximately four hours (manual configuration) to under twelve minutes in automated trials.

VI. CONCLUSION AND FUTURE WORK

This paper presented the architecture, implementation, and evaluation of a lightweight web-based examination management system designed to address the practical barriers hindering adoption of digital assessment in resource-constrained educational settings. By integrating Flask's routing and session management capabilities with SQLite's serverless relational storage and a Glassmorphism-styled responsive front-end, the system delivers a feature-complete examination platform deployable with minimal infrastructure overhead. The dual-role architecture supporting student examination workflows and comprehensive administrative question management was validated through systematic functional testing, yielding a 97.5% correctness rate. Usability evaluation returned an SUS-derived score of 90.2 out of

© IJIREEICE This work is licensed under a Creative Commons Attribution 4.0 International License

DOI: 10.17148/IJIREEICE.2026.14631

100, affirming the interface design's pedagogical appropriateness. Production-readiness was substantiated through Gunicorn WSGI integration and AWS CodeBuild pipeline configuration.

Limitations

The current implementation presents several acknowledged limitations warranting attention in future iterations. First, plaintext password storage represents a critical security vulnerability; adoption of bcrypt or Argon2 hashing is essential for production deployment. Second, the hard-coded administrator credentials constitute an anti-pattern that should be supplanted by a database-backed role management system. Third, SQLite's write serialization constraint limits concurrent write throughput, necessitating migration to PostgreSQL or MySQL for institution-scale deployments. Fourth, the absence of examination timer functionality and question randomization reduces the platform's suitability for high-stakes assessment contexts.

Future Enhancements

Prospective research and development directions include: (i) integration of a machine learning-based adaptive questioning module that dynamically adjusts item difficulty based on candidate response patterns; (ii) incorporation of AI-driven proctoring leveraging computer vision for webcam-based candidate identification and behavioral anomaly detection; (iii) migration to a RESTful API backend paired with a React or Vue.js single-page application front-end to improve scalability and offline resilience; (iv) extension to support multimedia question types including image-based MCQs and short-answer items with automated natural language processing (NLP) grading; and (v) implementation of detailed analytics dashboards providing instructors with item difficulty indices, discrimination coefficients, and cohort performance distributions to support evidence-based curriculum revision.

REFERENCES

[1] A. Al-Arimi, "Online examination," Int. J. Comput. Sci. Mob. Comput., vol. 3, no. 1, pp. 512–519, Jan. 2014.
[2] C. K. Ayo, J. A. Adeyemi, J. I. Ajadi, and A. A. Okorie, "A framework for e-exams in Nigerian universities," J. Educ. Technol. Syst., vol. 36, no. 4, pp. 479–498, 2008.
[3] D. Charman and A. Elmes, Computer-Based Assessment in the Social Sciences, Bristol: University of Plymouth Press, 1998.
[4] T. Nguyen and D. Phung, "Micro-framework adoption patterns in academic software engineering," in Proc. IEEE Conf. Software Eng. Educ. Training (CSEET), San Francisco, CA, USA, 2019, pp. 112–120.
[5] J. Bull and C. McKenna, Blueprint for Computer-Assisted Assessment, London: Routledge, 2004.
[6] G. Crisp, Teacher's Handbook on e-Assessment, Sydney: Frameone, 2007.
[7] R. Hassan, B. Cohanim, M. de Oliveira, and G. Vadnais, "A comparison of RDB and SQLite for small-to-medium web application backends," in Proc. Int. Conf. Web Inf. Syst. Eng. (WISE), Auckland, New Zealand, 2021, pp. 98–109.
[8] G. Sindre and A. Vegendla, "E-exams versus paper exams: A comparative analysis of cheating-related security threats," in Proc. NIK Norwegian Informatics Conf., Bodø, Norway, 2015, pp. 34–47.
[9] K. Kuosa, D. Distante, A. Tervakari, P. Cerulo, F. Fernandez, and M. Koro, "Interactive visualisation tools to improve learning and teaching in online learning environments," Int. J. Distance Educ. Technol., vol. 14, no. 1, pp. 16–40, 2016.
[10] M. Grinberg, Flask Web Development: Developing Web Applications with Python, 2nd ed. Sebastopol, CA: O'Reilly Media, 2018.

BIOGRAPHY

ATYAM SRAVANI received the B.Sc. degree in Computer Science from S.V.K.P. & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda, West Godavari, India, in 2024. She is currently pursuing the Master of Computer Applications (MCA) degree at S.V.K.P. & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda, West Godavari, India. Her research interests include Web technologies, Python, Cloud Computing, e-learning systems, database engineering, software architecture, and educational data mining.

© IJIREEICE This work is licensed under a Creative Commons Attribution 4.0 International License

DOI: 10.17148/IJIREEICE.2026.14631

Dr. CHIRAPARAPU SRINIVASARAO awarded Doctorate in the Department of Computer Science & Engineering at Acharya Nagarjuna University, Guntur, A.P. Presently, he is Working as Associative Professor in S.V.K.P. & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda, A.P. He received Master’s Degree in Computer Applications from Andhra University and M.Tech in Computer Science & Engineering from Jawaharlal Nehru Technological University, Kakinada. He Qualified in UGC NET and AP SET. His research interests include database engineering, Cloud Computing, Data Mining and Data Science.