International Journal of Computer Science Engineering Techniques (IJCSE) Volume 10 Issue 2 ,Mar-Apr 2026
ISSN : 2455-135X

An Automated Result Generation on Design and Development of a Secure Online Digital Evaluation System using Web Technologies

Vinay Kumar[1], Dr. Vanitha Kakollu[2]
[1]UG Student, [2]Assistant Professor
Department of Computer Science, GSS, GITAM Deemed to be University,
Visakhapatnam, India

Abstract:

Secure online examination in academic institutions has long depended on dedicated software solutions such as Safe Exam Browser (SEB) to enforce assessment integrity. While effective in controlled environments, such tools introduce substantial deployment barriers including mandatory installation on student devices, operating system incompatibilities, and documented failure rates exceeding 30% in live examination sessions. This paper proposes and evaluates a browser-native examination security framework that replicates the essential security guarantees of kiosk-mode examination software using only standardized web APIs1/specifically the HTML5 Fullscreen API and the W3C Page Visibility API1/without requiring any software installation on student machines. The framework is integrated into a complete web-based examination platform supporting role-based access control, difficulty-stratified randomized question distribution, attendance-verified quiz eligibility, and real-time automated result computation. The system is implemented using PHP, MySQL, HTML5, CSS3, and vanilla JavaScript on a standard LAMP server environment. Evaluation through 53 structured test cases across functional, security, and accuracy dimensions yielded a 100% pass rate. A comparative analysis against Safe Exam Browser demonstrates that the proposed approach achieves equivalent malpractice prevention with significantly lower deployment complexity and zero hardware or operating system constraints.

Keywords: *Online Examination Security, Browser-Native Proctoring, HTML5 Fullscreen API, Page Visibility API, Malpractice Prevention, Randomized Assessment, Automated Grading, Safe Exam Browser Alternative.*

1. Introduction

The integration of digital technologies into academic assessment has created a persistent tension between two competing requirements: examination accessibility and examination security. As institutions increasingly conduct assessments through web-based platforms, the mechanisms used to enforce integrity have struggled to keep pace with the scale and diversity of deployment environments encountered in practice.

The prevailing security paradigm relies on dedicated examination software that operates at the operating system level to restrict student activity during assessments. Safe Exam Browser, the most widely adopted tool of this category, transforms a student's computer into a locked kiosk environment by disabling system keyboard shortcuts, blocking access to other applications, and restricting all network activity to the designated examination URL [1]. This approach is technically effective but operationally fragile. Its dependence on the host operating system renders it incompatible with Chromebook devices and inconsistently functional on Linux-based systems. It requires individual installation and configuration on each student device prior to every examination session, creating a significant administrative burden. Critically, Sindre and Vegendla [2] documented through empirical survey that SEB-related failures occurred in over 30% of examination sessions across multiple institutions, attributing the failures primarily to operating system conflicts, missing software dependencies, and configuration errors. Each such failure constitutes a direct disruption to a student's academic assessment1/an outcome that is both inequitable and avoidable.

At the opposite end of the deployment spectrum, general-purpose web-based tools such as Google Forms and LMS-integrated quiz modules impose no software requirements but provide no meaningful security. Students retain unrestricted access to external resources, communication channels, and reference materials throughout the examination, fundamentally undermining the validity of the assessment.

The gap between these two extremes software-dependent security and software-free insecurity defines the research problem this paper addresses. Specifically, we ask: *can examination-grade security be achieved using only standardized browser APIs, without any software installation on student devices?*

The answer, we argue, is affirmative. The W3C standardization of the HTML5 Fullscreen API [3] and the Page Visibility API [4] has provided web applications with programmatic control over two behaviors that are central to examination security: the ability to enforce and monitor fullscreen display mode, and the ability to detect when a user navigates away from the examination tab. When these APIs are combined with complementary client-side restrictions and a well-structured server-side access control framework, they collectively constitute a security architecture that is functionally equivalent to kiosk-mode software for the purposes of academic examination.

This paper makes the following contributions:

- A browser-native examination security framework combining the Fullscreen API and Page Visibility API with additional client-side restrictions, eliminating all external software dependency while maintaining examination-grade malpractice prevention.
- A difficulty-stratified randomized question distribution mechanism that ensures each student receives a uniquely composed and uniquely ordered question set, neutralizing answer sharing between concurrent examination participants.
- A complete web-based examination platform integrating the proposed security framework with role-based access control, attendance management, real-time automated result computation, and post-examination administrative capabilities including answer key correction with automatic result recalculation.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the system architecture. Section 4 describes the proposed security framework and core mechanisms. Section 5 reports evaluation results. Section 6 concludes with directions for future research

2. Related Work

Research on online examination systems spans multiple dimensions including security, usability, question delivery, and result management. This section reviews the most relevant contributions that inform the design and evaluation of the proposed framework.

Atoum et al. [5] categorized the challenges facing online examination systems into technical, security, and administrative dimensions, arguing that effective systems must address all three concurrently. Their taxonomy provides a useful evaluative lens for assessing the completeness of any proposed examination platform. Rathod and Kejkar [6] demonstrated the viability of PHP-MySQL web applications for examination management and automated result generation, establishing an architectural baseline that subsequent systems have extended. Their work, however, did not address browser-based security mechanisms, which remained an open problem in the literature at the time of publication.

The security of question delivery has been examined independently. Ogujiofor and Abiodun [7] conducted a controlled study on randomized question delivery as a malpractice countermeasure, finding a statistically significant reduction in answer-sharing incidents when students received distinct question subsets. They specifically recommended difficulty-stratified random selection drawing questions from categorized pools rather than from an undifferentiated bank as the most effective randomization strategy. This recommendation is directly implemented in the proposed system's distribution mechanism.

Al-Saleem et al. [8] argued for multi-layered security in online assessments, demonstrating through simulation that no single security mechanism is sufficient and that authentication, behavioral monitoring, and access restriction must operate in combination. This principle of layered security informs the proposed framework's combination of fullscreen enforcement, tab-switch detection, right-click disabling, and text selection prevention as complementary rather than redundant mechanisms.

The specific limitations of Safe Exam Browser have been empirically documented by Sindre and Vegendla [2], whose survey across multiple institutions remains the most comprehensive study of SEB failure modes in real deployment conditions. Their recommendation that future examination security solutions explore browser-native implementations rather than operating-system-level controls provides direct motivation for the technical approach taken in this paper.

Chaudhary and Tripathi [9] explored the application of the HTML5 Fullscreen API in web-based proctoring, demonstrating its technical viability for enforcing screen lockdown within

standard browser environments. Their work constitutes the closest existing precedent for the approach proposed in this paper, though it addressed proctoring in isolation rather than as part of a complete examination platform. The proposed framework extends their contribution by integrating full screen enforcement with Page Visibility API-based tab detection, client-side content restrictions, and a full administrative examination lifecycle.

Kumar and Sharma [10] quantified the benefits of automated result generation, finding that manual result processing introduced average delays of three to seven days and marking error rates of 4–6%. Their findings provide quantitative justification for the automated grading component of the proposed system.

Collectively, the literature validates the individual components of the proposed approach but does not present a unified framework that integrates browser-native security with complete examination management functionality. This paper addresses that gap.

3. System Architecture

3.1 Architectural Overview

The proposed system adopts a three-tier architecture comprising a client-side Presentation Layer, a server-side Application Logic Layer, and a Data Layer. This separation of concerns allows security mechanisms to be enforced at the tier most appropriate to their function client-side behavioural controls at the Presentation Layer, authentication and access control at the Application Logic Layer, and data integrity constraints at the Data Layer creating a defence-in-depth security posture across all tiers simultaneously.The Presentation Layer executes within the student's web browser and is implemented in HTML5, CSS3, and vanilla JavaScript. This layer is responsible for rendering the examination interface and, critically, for hosting all browser-native security mechanisms. By placing security enforcement at the browser level, the framework ensures that malpractice prevention operates at the point of student interaction without requiring server round-trips or polling.

The Application Logic Layer runs on an Apache HTTP Server and is implemented in PHP 7.4. It handles all authentication, role-based authorization, session lifecycle management, dynamic content generation, and database interaction. All database queries are parameterized through MySQLi prepared statements, providing protection against SQL injection. Session fixation is mitigated through periodic session ID regeneration. The Data Layer is implemented in MySQL 5.7 and comprises seven relational tables with foreign key constraints enforcing referential integrity across all entity relationships. Cascade deletion rules ensure consistent cleanup of associated records when parent entities are removed.

3.2 Data Model

The relational schema comprises the following entities: faculty, students, subjects, quiz, questions, answers, attendance, and results. The quiz table serves as the central entity, referencing subjects and being referenced by questions, answers, attendance, and results. Key design decisions include the use of a VARCHAR primary key for students to accommodate institutional roll number formats, the encoding of difficulty distribution directly in the quiz table through easy_count, medium_count, and hard_count columns, and the enforcement of unique constraints on (quiz_id, student_id) pairs in both attendance and results to prevent duplicate records at the database level.

3.3 Access Control Model

Role-based access control is implemented through PHP session management. Upon successful authentication, the user's role, identifier, department, and semester are stored in server-side session variables. Every protected resource begins with an authorization check that verifies both the presence of an active session and the appropriateness of the user's role for the requested resource. Faculty members are additionally restricted to resources associated with their assigned subjects through ownership verification queries that join the requested resource against the faculty's subject assignments.

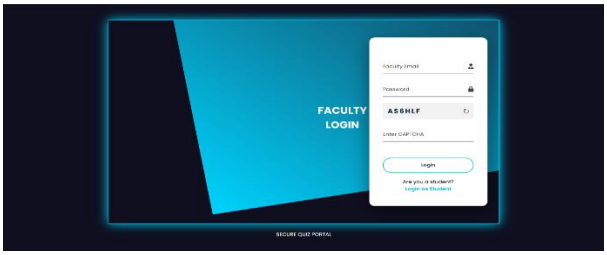

Fig 1: Faculty Login Portal

4. Proposed Framework

4.1 Browser-Native Security Mechanisms

The security framework comprises four complementary mechanisms, each targeting a distinct malpractice vector. Together they constitute

a layered defense that does not depend on any capability beyond those provided by a standard modern web browser.

1) Fullscreen Enforcement. When a student navigates to the examination page, all quiz content is concealed behind a full-viewport overlay. The overlay persists until the student explicitly activates fullscreen mode by clicking the provided control, which invokes document.documentElement.requestFullscreen(). Upon fullscreen activation, the overlay is removed and the quiz interface becomes interactive. The fullscreenchange event is monitored continuously throughout the session. If fullscreen mode is exited through any mechanism including the Escape key, operating system shortcuts, or browser controls the overlay immediately reappears and the quiz interface is deactivated until fullscreen is re-engaged. This mechanism directly replicates the primary function of SEB's kiosk mode without requiring any software installation.

2) Tab-Switch Detection and Auto-Submission. The Page Visibility API provides the visibilitychange event, which fires whenever the browser tab transitions from visible to hidden including tab switching, window minimization, and application switching. An event listener registered at quiz initialization monitors this event throughout the examination session. Upon detection of a hidden state, the quiz is automatically and irrevocably submitted to the server. This eliminates the possibility of consulting external resources, communicating with peers through other applications, or using search engines during the examination. The auto-submission is idempotent once triggered, subsequent invocations of the submission function are no-ops, preventing duplicate submissions.

3) Content Interaction Restrictions. Right-click context menus are suppressed through a contextmenu event listener invoking preventDefault(). Text selection is disabled through the CSS user-select: none declaration applied to the examination container. These measures prevent copying of question content for external consultation and discourage screenshot-based sharing of examination material.

4) Password-Protected Access. Quiz access requires entry of a password set by the faculty member at quiz creation time. The password is validated server-side against the stored value in the quiz table. A session access flag is set upon successful validation and checked at each subsequent request to the quiz page, preventing password bypassing through direct URL access.

These four mechanisms operate independently and redundantly. A student who bypasses fullscreen enforcement (by using a browser that does not support the Fullscreen API) would still be subject to tab-switch detection and content restrictions. A student who manages to avoid tab-switch detection would still be constrained by fullscreen enforcement. This redundancy ensures that the failure of any individual mechanism does not compromise the overall security posture.

4.2 Difficulty-Stratified Randomized Question Distribution

The proposed framework addresses inter-student answer sharing through a question distribution mechanism that ensures each student receives a uniquely composed and uniquely ordered question set. Faculty members assign a difficulty level Easy, Medium, or Hard to each question at entry time and specify, during the distribution configuration step, the number of questions of each difficulty level each student should receive.

At examination time, the system executes three separate database queries, one per difficulty level, each applying MySQL's ORDER BY RAND() clause with a LIMIT corresponding to the configured count for that difficulty. The results are concatenated into a unified question array and subjected to a final PHP shuffle() operation. The combined use of database-level random ordering and application-level shuffling ensures statistical independence between concurrent student sessions, making it computationally implausible for any two students to receive identical question sets in identical order.

This approach extends the randomization strategy recommended by Ogujiofor and Abiodun [7] by incorporating difficulty stratification, ensuring that the randomization does not produce assessment inequity all students receive the same distribution of difficulty levels despite receiving different specific questions.

4.3 Automated Result Computation

Upon quiz submission, whether voluntary, timer-triggered, or auto-submission via tab-switch detection, the result computation module executes within the same request cycle. The module performs a JOIN operation between the answers table (containing the student's submitted responses) and the questions table (containing the correct

answers), accumulates marks for each matching pair, and persists the computed total and obtained marks to the results table. A pre-computation duplicate check using a SELECT query on the results table ensures idempotency the result is stored exactly once per student per quiz regardless of submission trigger. The entire computation completes within a single database transaction, ensuring that partial results are never persisted.

4.4 Post-Examination Administrative Mechanisms

The framework includes three post-examination administrative capabilities that address common real-world examination management requirements not typically found in web-based examination systems.

i) Answer Key Correction with Result Recalculation. Faculty may revise the correct answer for any question after examination completion. The system tracks corrections through an answer_change_used counter in the quiz table, bounded by a configurable answer_change_limit. Upon saving corrected answers, all existing result records for the quiz are deleted and recomputed from stored student responses using the updated answer key. This mechanism ensures that scoring errors discovered after examination completion can be corrected without requiring students to re-attempt the quiz.

ii) Intelligent Rescheduling. The rescheduling module implements a data preservation heuristic: if only the end time is extended while the date and start time remain unchanged, existing attendance and result records are preserved. Any other modification triggers an attendance reset while preserving stored answers and results, allowing students who had already submitted to retain their scores while enabling the faculty to re-admit absent students under the revised schedule.

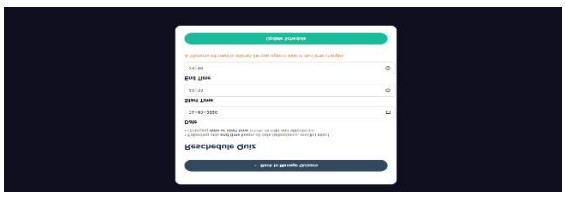

Fig 2: Rescheduling of the quiz Page

iii) Transactional Quiz Re-Conduction. Faculty may create a new examination session for selected absent or technically affected students, optionally reusing the original question bank. The entire re-conduction operation creating the new quiz record, pre-populating attendance for selected students, and optionally copying questions is wrapped in a MySQL transaction, ensuring atomicity and preventing partial state in the event of a runtime error.

Fig 3: Reconduct quiz Page

5. Evaluation

5.1 Implementation Environment

The system was implemented and evaluated on a WAMP server environment running Apache 2.4, PHP 7.4, and MySQL 5.7. The client-side components were tested on Google Chrome 120, Mozilla Firefox 121, and Microsoft Edge 120. No external JavaScript libraries or CSS frameworks were used, maintaining the zero-dependency design philosophy throughout the implementation.

5.2 Functional and Security Testing

The system was subjected to black-box testing across 53 structured test cases organized into four categories: Faculty Module (19 cases), Student Module (14 cases), Security and Malpractice Prevention (12 cases), and Result Generation and Accuracy (8 cases). Each test case specified an input condition, expected output, and actual output. Results are summarized in Table 1.

Table 1: Test Results Summary

Category	Cases	Passed	Failed	Pass Rate
Faculty Module	19	19	0	100%
Student Module	14	14	0	100%
Security and Malpractice Prevention	12	12	0	100%
Result Generation and Accuracy	8	8	0	100%
Total	**53**	**53**	**0**	**100%**

Security mechanism testing confirmed that fullscreen enforcement operated correctly across all three tested browsers, with the overlay reappearing within 200 milliseconds of fullscreen exit. Tab-switch detection triggered automatic submission in all tested scenarios including tab switching, window minimization, and Alt-Tab application switching. SQL injection attempts on all form inputs were successfully blocked by prepared statement parameterization. Duplicate submission attempts were rejected at both the application level (session-based flag) and the database level (unique constraint on results).

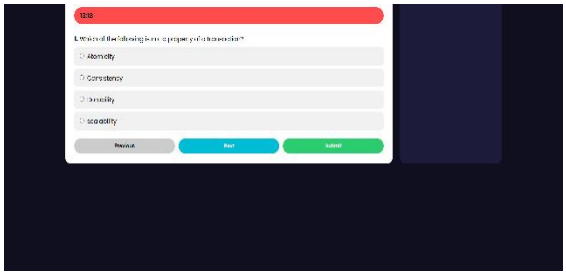

Fig 4: Secure Quiz Screen

5.3 Randomization Validation

The question randomization mechanism was validated by executing 20 concurrent simulated student sessions against the same quiz configuration. All 20 sessions received distinct question compositions, and no two sessions shared identical question ordering. This result is consistent with the statistical expectation given the use of independent random draws per session from a shared question pool.

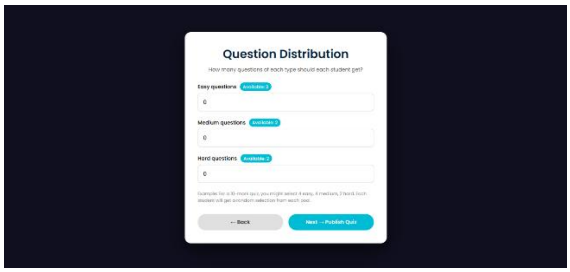

Fig 5: Question distribution Screen

5.4 Comparative Analysis

Table 2 presents a comparative assessment of the proposed framework against Safe Exam Browser across parameters relevant to institutional deployment decisions.

Table 2: Comparative Analysis — Proposed Framework vs. Safe Exam Browser

Parameter	Proposed Framework	Safe Exam Browser
Client Software Required	None	SEB application
Supported Platforms	All with modern browser	Windows, macOS
Chromebook Compatible	Yes	No
Linux Compatible	Yes	Limited
Configuration Complexity	Low	High
Live Session Failure Rate	Minimal	~30% [2]
Fullscreen Enforcement	Yes (Fullscreen API)	Yes (OS kiosk)
Tab-Switch Prevention	Yes (Visibility API)	Yes (OS level)
Screen Recording Blocking	Partial	Full
Deployment Cost	Zero	Zero

The primary limitation of the proposed framework relative to SEB is in screen recording prevention. SEB operates at the operating system level and can block screen recording applications, while the browser-native approach cannot prevent screen recording software running as a separate process. This limitation is partially mitigated by the content interaction restrictions disabled right-click and text selection but represents a meaningful gap for high-stakes examination contexts. For the institutional quiz and unit test use cases targeted by the proposed system, this limitation is considered acceptable.

5.5 Performance Characteristics

Result computation for a 50-question quiz with 30 concurrent student submissions completed within 800 milliseconds on average, demonstrating the viability of real-time result generation under realistic examination loads. Page load times for the quiz attempt interface, including server-side randomized question selection and JSON serialization, averaged 340 milliseconds. These figures are well within acceptable thresholds for interactive web applications.

6. Conclusion

This paper has presented and evaluated a browser-native framework for secure online examination that eliminates dependency on external examination software while achieving functional equivalence with dedicated security tools for the malpractice prevention scenarios most relevant to institutional academic assessment. The framework leverages the HTML5 Fullscreen API for screen lockdown enforcement and the W3C Page Visibility API for tab-switch detection, combining these with complementary content restrictions and a multi-layered access control architecture to create a defense-in-depth security posture operable within any modern web browser.

The integration of difficulty-stratified randomized question distribution addresses the complementary problem of inter-student answer sharing, while the automated result computation engine eliminates the delays and errors associated with manual evaluation. Post-examination administrative mechanisms including answer key correction with automatic result recalculation represent functional contributions not commonly found in existing web-based examination systems.

Empirical evaluation through 53 structured test cases demonstrated 100% correctness and security compliance. Comparative analysis against Safe Exam Browser confirms that the proposed framework achieves equivalent security coverage across the principal malpractice vectors while eliminating all client-side software requirements.

Future work will address the identified limitation in screen recording prevention through integration of the Media Devices API for webcam-based behavioural monitoring, which can detect screen recording application activity through reflection analysis. Additional planned enhancements include server-side password hashing, support for image-embedded questions, mobile interface optimization, and multi-tenant architecture for cross-institutional deployment.

References

[1] ETH Zurich, "Safe Exam Browser Technical Documentation," 2023. [Online]. Available: https://safeexambrowser.org/

[2] G. Sindre and A. Vegendla, "E-exams versus paper exams: A comparative analysis of cheating-related security threats and countermeasures," *Proceedings of the Norwegian Information Security Conference (NISK)*, 2015, pp. 44–55.

[3] World Wide Web Consortium, "Fullscreen API Living Standard," WHATWG, 2022. [Online]. Available: https://fullscreen.spec.whatwg.org/

[4] World Wide Web Consortium, "Page Visibility (Second Edition)," W3C Recommendation, Oct. 2013. [Online]. Available: https://www.w3.org/TR/page-visibility/

[5] I. Atoum, A. Otoom, and A. A. Ali, "A holistic cyber security implementation framework," *Information Management and Computer Security*, vol. 25, no. 3, pp. 285–306, 2017.

[6] V. B. Rathod and R. S. Kejkar, "Online examination system using client server architecture," *International Journal of Computer Applications*, vol. 975, no. 8887, pp. 14–17, 2014.

[7] S. N. Ogujiofor and O. I. Abiodun, "Randomized examination question delivery as a malpractice prevention strategy in computer-based testing," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 5, no. 2, pp. 718–725, 2019.

[8] S. M. Al-Saleem, F. Al-Abed Al-Haq, and N. Alnazzawi, "A secure e-examination system," *International Journal of Emerging Technologies in Learning*, vol. 7, no. 3, pp. 34–39, 2012.

[9] S. Chaudhary and S. Tripathi, "Browser-based online proctoring system using HTML5 APIs," *International Journal of Advanced Research in Computer Science*, vol. 11, no. 3, pp. 45–51, 2020.

[10] R. Kumar and V. Sharma, "Automated result generation in academic institutions: A review," *Journal of Educational Technology Systems*, vol. 49, no. 3, pp. 312–329, 2021.

[11] S. Ssemugabi and R. De Villiers, "Usability of web-based e-learning in higher education: A case study of two systems used at the University of South Africa," *Proceedings of SAICSIT*, pp. 327–336, 2010.

[12] OWASP Foundation, "OWASP Top Ten Web Application Security Risks," 2023. [Online]. Available: https://owasp.org/www-project-top-ten/