International Journal of Trend in Scientific Research and Development (IJTSRD)
Special Issue on Smart Innovations in Computer Science and Applications
Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Secure and Integrated Exam and Quiz Management System Using React and Node.js

Shruti Pimpalshende
Department of Science and Technology, G.H.Raisoni Skill Tech University, Nagpur, India

ABSTRACT

Running exams in colleges involves a lot more effort than most people realize. Setting question papers, evaluating hundreds of answer sheets, and publishing results every single step demands time, coordination, and manpower. And mistakes do happen. A wrong total here, a misplaced sheet there, and students start filing complaints while staff redo work they had already done. This project grew out of a real frustration with how much manual effort goes into something that repeats every semester.

I made a web based exam management system. This system does everything from making question papers to showing results to students. You do not need to use a lot of tools or spreadsheets.The frontend of the system uses React. The backend uses Node.js.There are login roles for admins and teachers and students. Each person can only see what they are supposed to see.When students answer questions they get scored right away when they hit submit.Descriptive answers go to a queue where teachers can review them.When we tested the system it worked well with a lot of users at the time.Making results for exams used to take days but now it only takes a few minutes after the exam is closed.The web based exam management system is really helpful, for managing exams.The system makes it easy for admins and teachers and students to do their work.

Keywords: Exam Management System, React, Node.js, Role-Based Authentication, Automated Scoring, Web Application, Relational Database.

1. Introduction

If you talk to any teacher who has been watching students take exams and grading their answer sheets for fifteen or twenty years they will tell you that the process is still much the same. The school prints out the question papers the students write their answers on paper the teachers grade the papers by hand the marks are put into spreadsheets or special books the results are added up checked again and finally made public often weeks after the students took the exam. This works okay when you are dealing with a number of students.When the school gets bigger this way of doing things starts to cause problems.It is not like people have not tried to make the process more modern. Plenty of colleges use computers for parts of the process. But the key word there is parts. A college might put the syllabus online while still printing question papers. They might use email to share exam schedules while still doing mark tabulation in Excel. That kind of partial digitization often creates more confusion than it solves data ends up scattered, nobody has a complete view of what's happening, and reconciliation errors are common enough that people just expect them.

I started thinking about this specifically around internal exams and unit tests the ones that happen multiple times a semester, involve shorter question sets, and currently pile up a disproportionate amount of work on faculty who are already stretched thin. If even these routine tests could be moved to a properly managed digital system, the time saved over a full academic year would add up fast. A faculty member spending four hours evaluating papers for a forty-student unit test, three or four times a semester, is using up nearly a full working week on evaluation alone and that's before factoring in paper preparation and result compilation.

The existing options are worth mentioning here. Google Forms can run a quiz. Microsoft Forms does too. Moodle has quiz modules. But none of them were really built for institutional exam management. They handle the student-facing side okay presenting questions and collecting answers. But the backend is thin. Reporting is limited, role separation is awkward or missing, scheduling is basic at best, and data security is often an afterthought. They weren't designed specifically for institutional use, and that shows when you try to use them that way.

This paper describes a system I built from scratch to fill exactly those gaps. I'll walk through the design decisions, the technical implementation, how I tested it, and what the results actually looked like. The goal wasn't to build something flashy it was to build something that genuinely works for institutional exam management and holds up when real users are on it.

2. Problem Statement

The problem here isn't just that exams are run manually. It's the chain of smaller issues that manual administration creates and each one carries its own real cost.First, there's the issue of question paper preparation and distribution. A faculty member writes a paper, usually in Word or by hand, and it gets printed. Sometimes it leaks before the exam, sometimes the wrong version goes to the printer, sometimes a set gets lost on the way to an exam hall. Each of those situations needs emergency handling from administrative staff. When multiple sections write on the same day, managing distribution logistics is genuinely stressful. And all of this happens before a single student has written a word.

Second, evaluation takes a long time. For a batch of sixty students, grading a thirty-mark paper manually can take four to six hours more if the questions involve descriptive answers or need judgment calls. Multiply that across multiple papers and faculty members, and you're looking at a significant block of working hours every exam cycle. During this time, faculty are often unavailable for teaching or student consultations, which cascades into other scheduling problems.

Third, mark compilation is prone to errors. This isn't a criticism of individual teachers human error under high-volume, repetitive conditions is completely normal. But in mark compilation, even a small mistake can seriously affect a student's grade or eligibility. Rechecking requests and appeals take up even more time on both sides. Automated scoring for objective questions eliminates this entire category of error, which is one of the clearest wins a digital exam system can offer.

Fourth, record keeping after exams is a logistical challenge. Physical answer scripts have to be stored for years sometimes for revaluation, sometimes for institutional audits. This needs storage space, proper organization, and index maintenance. When an accreditation body asks for historical exam data, finding it in physical archives can take days. A well-structured database can return the same information in seconds.

Fifth and this one doesn't get talked about enough there's the matter of transparency. In a manual system, students get a final mark. If they want to understand how each section contributed to that total, they have to go through a formal revaluation process. That opacity breeds suspicion, especially when results are unexpected. A digital system can automatically show students a breakdown of their marks, which is a meaningful improvement in how students experience the process.

3. Objectives of the Study

Before writing any code, I spent time thinking carefully about what the system actually needed to do not in terms of a feature checklist, but in terms of which specific problems it had to solve. These objectives came from that process.

The first objective was centralization. Everything related to an exam questions, scheduling, student responses, evaluated marks, published results should live in one place. The moment data splits across two different systems, you've got a synchronization problem waiting to happen. Faculty shouldn't need to export from one tool and import into another. Administrators shouldn't be reconciling two different result records from two different sources.

The second objective was genuine role separation enforced at the server level not just different screens for different users, but actual technical restrictions. A student account should be technically incapable of accessing another student's response data, even if they know the API endpoints and try to query them directly. Faculty members should only be able to modify exams they created. Administrators should have visibility across the system but shouldn't be able to alter submitted answer data without a clearly auditable action. These boundaries need to be enforced by backend middleware, not by frontend logic that anyone with developer tools can bypass.

The third objective was automated scoring for objective questions, paired with a clean workflow for descriptive ones. Full automation of descriptive evaluation wasn't the goal at this stage, but I wanted descriptive responses properly captured and easily accessible to faculty not buried in a folder structure that requires manual navigation.

The fourth objective was time-controlled access enforced server-side. An exam should be accessible only within its scheduled window and rejected outside it and this needs to happen on the backend, not just in the interface. A student who finds a way to submit answers ten minutes late should have that submission rejected by the server, not quietly accepted because the frontend timer already stopped.

The fifth objective was performance reporting that gives faculty class-level insight not just individual scores. Which questions did most students get wrong? What was the score distribution? These questions come up naturally in every post-exam conversation I've seen between faculty, and the system should be able to answer them automatically from data it already holds.

4. Review of Existing Literature

I went through a fair amount of existing research before finalizing the design partly to understand what had already been tried, and partly to spot recurring patterns in what tends to go wrong with these systems in practice. A few themes came up consistently enough to directly shape my decisions.

The earliest research on computer-based testing, going back to the 1990s, was mostly focused on proving equivalence showing that students taking the same exam on a computer got statistically similar results compared to paper. That was necessary groundwork at the time, but the equivalence question has been settled for most practical purposes. The more relevant research came later.

Studies from the mid-2000s onward looked at what digital exam systems specifically failed at in real institutional deployments. A recurring finding was that systems built primarily around objective questions struggled badly when institutions tried to use them for exams with descriptive components. The workarounds institutions developed sometimes handling objective sections digitally while continuing to manage descriptive answers manually created exactly the kind of fragmentation I was trying to avoid. Fragmented workflows are often worse than fully manual ones because they carry the overhead of both approaches while delivering the benefits of neither.

When it comes to security things get a bit boring because everyone is saying the thing. People who looked at exam platforms found the same problems over and over again. The problems are things like sending information without locking it up using special codes that do not expire only checking who

can do things on the user side where people can easily get around them and storing passwords in a way that is not safe. These are not ways to attack they are simple mistakes that keep happening because people do not think about security when they are building things. Security is something that is thought about later not when they are first designing the system. The security of exam platforms is a concern because people are not taking care of the basic failures. Online exam platforms have the security problems, such, as credentials sent without encryption and session tokens that never expire and access control checks done only in the frontend and passwords stored using reversible encoding instead of proper hashing. The security of exam platforms is important and people should think about it when they are designing the system, not later.

Research on scalability pointed toward asynchronous, event-driven backend architectures for handling exam traffic. Several studies modeled scenarios where large numbers of students submit answers within short windows which is exactly what happens at exam close time. Thread-per-request server models handled this poorly compared to event-loop models like Node.js. One study showed a traditional server starting to degrade at around eighty concurrent connections while the Node.js equivalent kept serving requests without queue buildup at several times that number. This finding directly influenced the technology choice for the backend.

The user interface research, while thinner than the security or scalability literature, consistently pointed toward simplicity as the most important quality in exam interfaces. Students are under stress during exams. An interface that requires explanation, behaves inconsistently across devices, or uses unfamiliar interaction patterns is a fairness issue, not just a usability inconvenience. This shaped the decision to use React for component-based interface construction rather than a more flexible but less consistent approach.

One genuine gap I noticed across the literature was the lack of direct comparison between systems and manual baselines. Most studies evaluate a system in isolation, reporting internal metrics without establishing what improvement over the previous manual process actually looks like in measurable terms. I tried to address this in my own testing by documenting specific time comparisons wherever data was available

5. System Architecture

The way things are set up is like a three step process: you have the part people see the part that does the work and the part that stores the information. I want to take a moment to talk about why we did it this way. It is not, about what we did but also why we did it.

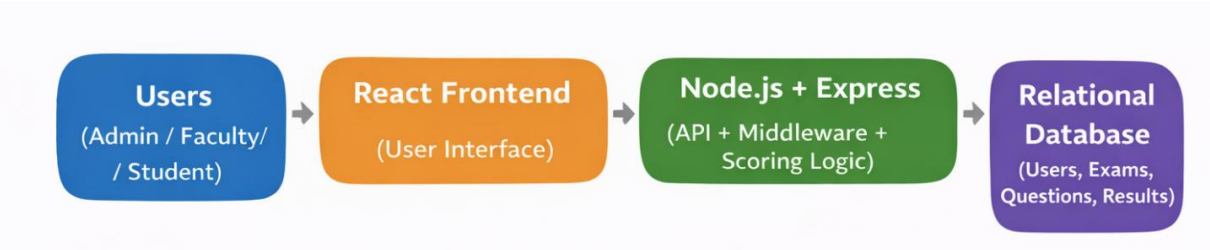

Fig 1: Three-Layer Architecture of the Exam and Quiz Management System

React is what we use to make the user interface. Every screen is a React component or a composition of smaller components. The critical decision here was to make the frontend purely a display and input layer. No business logic lives in the browser. No access control decisions happen on the frontend. The React application sends requests to the backend and renders whatever the backend returns it doesn't independently decide what data a user is allowed to see, and it doesn't calculate scores locally.This matters for a specific reasonFrontend code runs on a users browser. This means anyone, with some technical knowledge can read it. They can also change it. Send requests that are not what is expected.If access control is decided by JavaScript that runs on the browser then a student who knows about HTTP can easily get around it. They do not need to have difficulty. By keeping all meaningful logic

server-side, the frontend becomes a presentation concern only, and the attack surface shrinks significantly.

Node.js runs the server chosen primarily because of the event loop model. During active exam windows, many students will submit answers within a short timeframeA traditional server creates a thread for each connection. This uses a lot of resources when many connections happen at once.Node.js works differently. It uses one thread. Handles things in order. When it gets a request it starts the work like talking to a database or doing a calculation.Then it immediately waits for the request instead of waiting for the first one to finish.This is really good for times when lots of things happen at once like when many people submit exam answers at the time.Node.js handles this kind of traffic better, than a traditional server.Express provides routing and middleware on top of Node.js. Each protected route passes through a chain of middleware before reaching the handler that actually processes the request. First, token verification confirms the user is authenticated. Second, role checking confirms the user holds the required role for that specific route. Only if both pass does the request reach the route handler. This means access control cannot be bypassed by sending crafted requests directly to API endpoints the middleware catches unauthorized requests before any data is touched.

A relational database sits at the data layer. I chose relational over document-based storage because the data has clearly defined relationships with integrity requirements that relational databases enforce natively. A response must correspond to a real student and a real exam. A result must correspond to a real response set. These constraints prevent a whole category of bugs where records end up in inconsistent states. Document databases offer more flexibility, but that flexibility isn't needed here, and enforcing schema integrity at the database level is more reliable than enforcing it purely in application code.

6. Database Design

The schema has five main tables: Users, Exams, Questions, Responses, and Results. Getting this right was probably the most consequential design decision in the whole project. A poorly designed schema creates problems that are genuinely difficult to fix later without migrating data, which is expensive and error-prone.

The Users table stores basic identifying information along with a Role field that determines what parts of the system each account can access. Passwords are hashed before storage the database never holds the actual password, only a hash that can be verified against a new input without being reversed. This means a database breach doesn't automatically compromise user credentials, which is the minimum acceptable standard for any system handling real user authentication.

The Exams table holds each exam's metadata: title, subject, scheduled date and time, duration in minutes, and a foreign key to the faculty member who created it. That foreign key serves two purposes. It enforces data integrity you can't create an exam linked to a non-existent user. And it makes it easy to retrieve all exams created by a specific faculty member for the faculty dashboard, without scanning the entire exams table.

The Questions table is linked to Exams and stores question text, type, mark weight, and for objective questions, the correct answer option. Storing the correct answer in the database rather than hardcoding it in application logic matters for maintainability. If a faculty member needs to update a question or change the mark weight after creating an exam, they can do so through the interface without requiring a code change. The scoring logic reads the correct answer from this table at evaluation time.

The Responses table is where student submissions land. Every submitted answer creates a record linked to the student's UserID and the ExamID. For objective answers, comparison against the correct answer happens immediately at submission the mark is calculated and stored alongside the response.

For descriptive answers, the text is stored with a pending review status that faculty can update after manual evaluation.

The Results table summarizes final marks per student per exam. Foreign key constraints ensure that results can only reference students and exams that actually exist in the system. Once a result record is committed, modifying it should require an explicit administrative action with an audit trail. The normal evaluation workflow should not provide any path to altering finalized results.

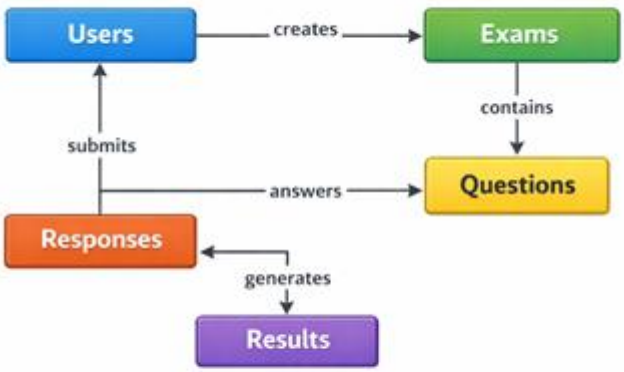

Fig 2 : Entity Relationship Diagram of Exam and Quiz Management System

7. Implementation Details

Frontend development followed a component-based approach throughout. The application is divided into logical modules: authentication, administration, faculty workflows, and student workflows. Each module is made up of smaller, reusable components. The question display component used in the student exam view and the question management component in the faculty interface share the same underlying structure with editing enabled or disabled based on the user's role. This reuse cuts down on code duplication and means fixes automatically apply to both places.

State management uses a combination of local component state for UI interactions and shared context for application-level state like the current user's identity and role. The user's role, once confirmed at login and validated by the backend, is stored in shared context and consulted by every component that makes rendering decisions based on permissions. Importantly, this state stays in sync with the server. If a session expires, the frontend detects the 401 unauthorized response on the next API call and redirects to login rather than continuing to display screens the user no longer has a valid session for.

On the backend, writing each route with an explicit middleware chain rather than embedding access checks inside route handlers was a deliberate choice. It keeps handler code focused on business logic and makes the security model easy to audit. You can look at any route definition and immediately see what authentication and authorization checks apply to it, without reading through the entire implementation.Password hashing uses bcrypt. Bcrypt is deliberately slow compared to general-purpose hash functions, which is its main security property. Slow hashing makes brute-force attacks against stolen hashes computationally expensive because each guess requires a full hash computation. The cost factor can be increased as hardware gets faster, keeping the protection relevant over time without changing the API or stored data structure.

The scoring logic for objective questions runs entirely on the backend. When a student submits an answer, the backend fetches the correct answer from the Questions table, compares it to the submitted

answer, calculates the mark, and writes the response record with the score included. The correct answer never travels to the frontend, and the frontend has no input into the score calculation.

Time enforcement works at two levels. The frontend timer gives students a visible countdown and submits automatically when it reaches zero. But the backend independently checks every submission's timestamp against the exam's scheduled end time. A submission arriving after the cutoff is rejected with an error response regardless of how it was triggered. This dual enforcement means bypassing the frontend timer which is trivial for anyone with network inspection tools doesn't result in a late submission being accepted.

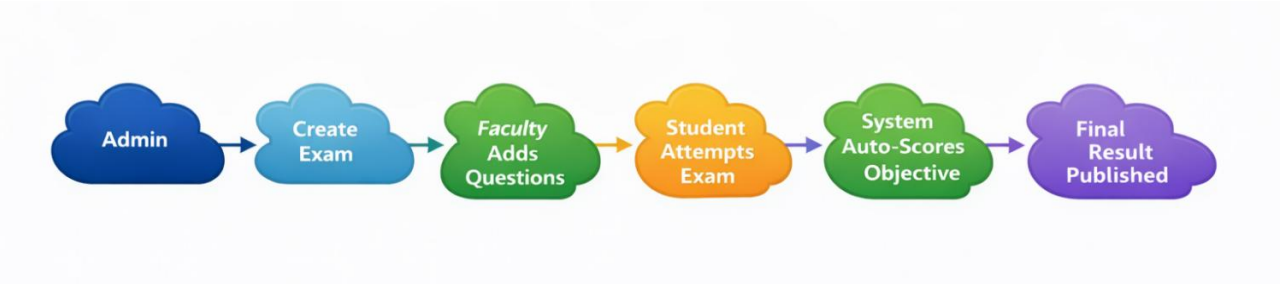

Fig 3: Examination Workflow Process

This diagram shows the basic flow of how an exam is managed in the system. First, the admin creates the exam and faculty add the questions. Students then attempt the exam, the system automatically checks objective answers, faculty review descriptive ones, and finally the results are published.

8. Scoring Model

The scoring formula is deliberately simple. For objective questions, the total score S equals the sum of the mark weight Mi multiplied by an indicator Ai for each question where Ai is 1 if the answer was correct and 0 if it was wrong:

$$S = \Sigma(Mi \times Ai)$$

No partial credit is awarded in the current implementation. The schema supports adding partial credit handling if institutional policy requires it, but keeping the base case simple reduces implementation complexity and edge cases.

Beyond raw scores, the system computes secondary metrics that are genuinely useful for faculty review. Per-student accuracy rate is the proportion of attempted questions answered correctly a normalized measure that lets you compare student performance across exams with different total mark values, which raw scores alone don't allow.

The question-level difficulty index is defined as the proportion of students who answered a given question correctly. A question that 90% of students got right isn't doing much to differentiate between students and may not be worth including in future exams. A question that only 15% answered correctly might indicate the topic was inadequately covered, the question was ambiguously worded, or it was genuinely hard and working as intended. Difficulty indices alongside student feedback help faculty figure out which.

These metrics require no additional data collection they're computed from what's already in the Responses and Questions tables. For faculty asking post-exam questions like which topic gave students the most trouble, the system can answer automatically from existing records rather than requiring manual analysis.

For descriptive questions, the manually entered mark from the faculty evaluator is incorporated into the same formula. The evaluator enters a mark through the application interface, the system stores it as the scored value for that question, and the final result calculation includes both automatically scored objective marks and manually entered descriptive marks in a unified total.

9. Testing and Validation

Testing was organized into three phases: functional testing, security testing, and load testing. Each phase had its own test cases and acceptance criteria defined before testing began, which helped avoid the tendency to declare something passing just because no obvious failures showed up in casual use.

Functional testing covered every workflow from end to end. User registration and login were tested across all three roles. Exam creation was tested with valid inputs, invalid inputs like missing required fields, and boundary conditions like scheduling an exam in the past or creating one with no questions. Student exam participation was tested for normal cases and edge cases: starting before the exam opens, submitting after it closes, losing connectivity mid-exam and reconnecting. The automated scoring workflow was tested by submitting known answer sets and verifying that calculated scores matched manually computed expected values across different mark-weight configurations.

Security testing was explicitly adversarial I specifically tried to break the access controls rather than just confirming that normal usage worked. Test cases included sending requests to faculty-only endpoints using valid student tokens, trying to retrieve another student's response records using a valid student session, attempting to modify the correct answer for a question using a faculty account that didn't create that exam, and sending requests with no authentication token at all to protected endpoints. Every one of these attempts was correctly rejected by middleware before any data was accessed or modified.

Load testing simulated concurrent examination sessions by incrementally increasing the number of simultaneous users submitting answers and measuring response times and error rates at each level. The system maintained acceptable response times up to loads representative of medium-sized institutional batches roughly 80 to 100 concurrent submissions. At higher simulated loads, response times increased but the system remained stable. No data corruption occurred, no valid requests were dropped, and no crashes happened during load testing.

Integration testing confirmed that data passed correctly between frontend and backend across all tested scenarios, and that database records were consistent with the operations performed through the application interface. No cases were found where the database state diverged from what the interface indicated had occurred.

10. Results and Discussion

The results across all three testing dimensions were encouraging. Functionality worked as specified, security held up against adversarial test cases, and performance was adequate for the institutional scale the system was designed for. But I want to discuss both the wins and the honest limitations rather than just reporting positive outcomes.The most concrete improvement over manual processes was result turnaround time for objective exams. In the manual workflows I referenced during background research, result publication for a mid-semester unit test typically takes three to five working days from the exam date accounting for evaluation, mark compilation, verification, and administrative processing. With automated scoring, results for purely objective exams were available within minutes of the scheduled exam end time. For mixed exams with descriptive sections, the objective portion was scored immediately, leaving faculty to evaluate only the descriptive answers rather than the entire paper. Even partial automation meaningfully reduces the overall evaluation burden.

The security testing results gave me confidence in the access control architecture. Passing a planned set of adversarial tests isn't a guarantee of security against all possible attacks I want to be clear about that. Security is an ongoing process that needs monitoring, updating, and periodic review. But the tests did confirm that the basic architecture is sound and that the most common attack patterns against web applications of this type are properly defended against in the current implementation.One result worth mentioning without softening is mobile usability. The web application works on mobile browsers, but it wasn't specifically designed for small screens, and the experience is noticeably worse than on a

desktop or tablet. For student populations that rely on smartphones as their primary computing device which is common in many institutional contexts this is a real limitation that I've noted as a priority for future development.

Feedback gathered during testing was mixed but useful. The student exam-taking interface was generally well-received most users found it intuitive without needing instruction, which was the goal. The faculty exam creation interface was more complex and needed some initial orientation. The reporting interface received the most critical feedback; several faculty testers found it harder to navigate than expected, and the data presentation could be clearer. These are design problems that can be addressed without changing the underlying architecture, but they're real problems that affect usability.

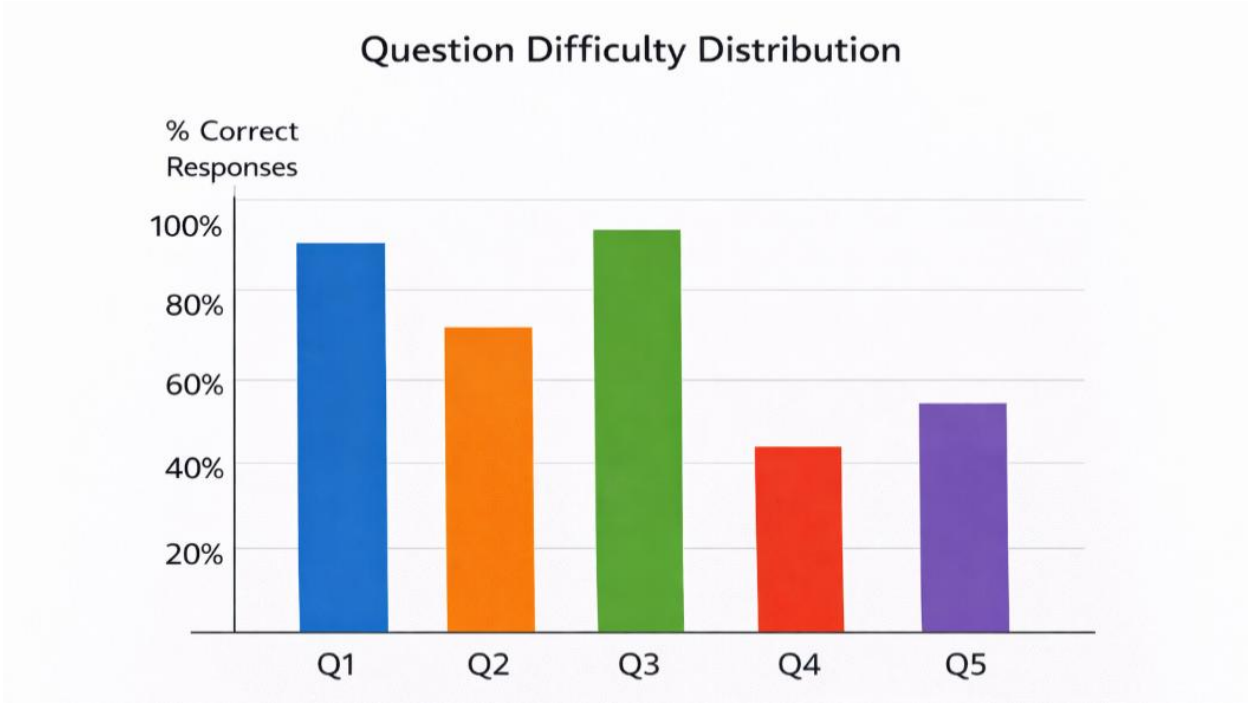

Fig 4: Comparison of Result Generation Time Between Manual and Automated Methods

The graph simply shows how much time is spent on evaluation in both manual and automated methods. When the system handles objective grading automatically, teachers only need to review the descriptive answers. This naturally reduces the total time and effort required for evaluation

11. Comparison With Existing Approaches

Against fully manual processes, the advantages are real and significant. Faster results, eliminated mark compilation errors for objective assessments, centralized record storage, structured performance reporting these are concrete improvements with measurable impact on faculty workload and administrative overhead. The cost is the investment in deploying and maintaining the system and training users on it. For institutions that run frequent internal assessments throughout the semester, this investment pays off fairly quickly. For institutions that run only a few major exams per year, the case is less clear-cut.

Against established platforms like Moodle or Google Classroom, the comparison is more nuanced and I want to be honest about it. These platforms have vastly larger feature sets, more mature codebases, much larger user communities, and more integration options than what I built. If an institution already uses Moodle, extending its quiz capabilities would make more sense than deploying a separate system. The case for this system isn't that it's better than Moodle in absolute terms it's that it's simpler, more targeted, and easier to deploy and customize for institutions that don't have complex LMS infrastructure already in place.

Against lightweight tools like Google Forms, the advantages are substantial. Google Forms doesn't have meaningful role-based access management, doesn't support institutional scheduling, doesn't offer structured result reporting at the exam level, and handles security poorly for academic contexts. It works for informal surveys and low-stakes quizzes. It's genuinely not appropriate for formal institutional assessment where data integrity and access control matter.

11.1 Comparative Analysis of Existing Systems

Table 1 below compares the proposed system against traditional manual methods and basic online quiz tools:

Table 1: Comparative Analysis of Examination Systems

Feature	Traditional System	Basic Online Quiz System	Proposed Integrated System
Paper Usage	High	None	None
Manual Evaluation	Yes	Partial	Minimal
Objective Auto-Grading	No	Yes	Yes
Descriptive Answer Support	Yes	Limited	Yes (Structured)
Role-Based Access	Manual Control	Limited	Fully Implemented
Performance Analytics	Basic	Limited	Detailed Reports
Centralized Database	No	Partial	Yes
Security Mechanisms	Physical Control	Basic Login	Role-Based & Session Managed

12. Future Development Directions

There are several directions I'd want to take this system given more time, and I think it's worth being specific about them rather than listing vague possibilities. Three of them feel genuinely important enough to be near-term priorities rather than long-term aspirations.

The most impactful addition would be some form of automated assistance for descriptive answer evaluation. Natural language processing has improved enough that short-answer scoring where responses are typically one to three sentences is now tractable for many question types. I'm not suggesting fully automated evaluation of descriptive answers as a replacement for faculty judgment, especially for high-stakes assessments. But even a system that pre-scores responses and gives faculty a suggested mark to review and override would significantly cut evaluation time. An institution running this system for one semester would accumulate enough evaluation history to begin training a reasonable scoring model on their own question types.

Mobile support is the most straightforward gap to address technically. A React Native version of the student exam interface would dramatically improve the experience for smartphone users without requiring any backend changes the backend API stays the same, only the frontend delivery changes. Given how many students in typical institutional contexts rely on smartphones as their primary device, this feels like a near-term necessity rather than an optional enhancement.

Cloud deployment would make the system accessible to smaller institutions without dedicated server infrastructure. A containerized deployment on a managed cloud platform would handle scaling automatically during peak examination periods and reduce operational burden on institutional IT staff. The current system runs on local infrastructure, which introduces dependencies on hardware reliability and network stability that cloud deployment would eliminate.

Integration with Moodle or other LMS platforms would allow user data and course records to be shared rather than maintained separately. This would remove one of the main friction points for adoption at institutions that already use an LMS, where maintaining two separate user databases is a real administrative overhead.

13. Conclusion

This project started from a fairly direct observation: exam management in many institutions is harder than it needs to be, and the digital tools available don't address the problem at the right level. Quiz tools designed for informal use are too lightweight. Enterprise LMS platforms are too heavy for institutions without the infrastructure or IT staffing to support them. There's a gap in the middle where a purpose-built, practical, open exam management system could genuinely help.

What I built is an attempt to fill part of that gap. It covers the full examination lifecycle within one application, enforces role-based access control at the server level, automates objective scoring, provides structured performance reporting, and holds up under realistic concurrent load. Testing confirmed it works as designed across functional, security, and performance dimensions. The improvements over manual processes in result turnaround time and scoring accuracy are concrete and measurable.

I want to be straightforward about what this project is and isn't. It's a working prototype that demonstrates technical feasibility and establishes a foundation for a complete system. It's not a finished product ready for widespread institutional deployment without further development particularly around mobile usability and the reporting interface. The gaps I identified are real and need to be addressed before I'd confidently recommend this for production use at a real institution.

What the project does establish is that a carefully designed, open-source web application can handle institutional exam management in a way that's meaningfully better than manual processes and more appropriate for academic contexts than the general-purpose tools currently in wide use. The architecture is sound, the security model works, and the improvement in operational efficiency is real. Building this out into a fully deployable system is achievable and, based on what the testing showed, worth pursuing.

References

1. Ozden, M. Y., Erturk, I., and Sanli, R. (2004). Students' opinions about and satisfaction with an online examination system. Innovations in Education and Teaching International, 41(2), 145-157.

2. Al-Saleh, M., and Al-Shathri, A. (2010). Building a web-based examination system. Journal of Theoretical and Applied Information Technology, 11(2), 78-85.

3. Kumar, A., and Tiwari, M. (2014). Security in online examination systems: A review. International Journal of Computer Applications, 99(10), 12-17.

4. Singh, G., and Chhabra, I. (2014). Framework for designing web-based online examination system with emphasis on security. International Journal of Computer Applications, 108(3), 1-7.

5. Gambhir, V., Rajput, N. S., and Saini, J. R. (2016). Online examination systems: Perspectives and practices. International Journal of Advanced Research in Computer Science, 7(2), 185-191.

6. Biswas, S. (2018). Analysis of role-based access in educational management platforms. International Journal of Engineering and Technology, 7(3), 312-317.

7. Usha Kosarkar, Gopal Sakarkar, Shilpa Gedam, "Deepfakes, a threat to society", *International Journal of Scientific Research in Science and Technology (IJSRST)*, 13th October 2021, 2395-602X, Volume 9, Issue 6, PP. 1132-1140, https://ijsrst.com/IJSRST219682