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

Cloud-Based Online Examination System

Aman Pathan
Department of Science and Technology,
G. H. Raisoni Skill Tech University, Nagpur, India

Annotation

The increased use of digital infrastructure in education has led to increased scrutiny of traditional paper-based examination methods, which often run into logistical difficulties, such as the distribution of physical exams, manual grading, the storage of answer sheets and other exam materials, and the potential for cheating. There has been a need for alternative methods of administering assessments remotely since the COVID-19 pandemic and many educational institutions have turned to tools that have been hastily adapted for online use, illustrating the continued lack of a scalable, secure, and purpose-built examination platform. This paper describes an innovative Cloud-Based Online Examination System designed and developed to provide a solution for the problems outlined above using a distributed service-oriented architecture that is hosted on Amazon Web Services. The system supports the entire examination lifecycle, including the creation and randomization of question papers, the delivery of timed exams, automated grading of exams, processing of exam results, and performance analytics, through a single web interface that can be accessed from any device with internet access. Three distinct user types are represented: Admin, Instructor, and Student. Each user type has its own dashboard (i.e., Instructor Dashboard, Student Dashboard, and Admin Dashboard) with role-specific access controls, and all role-specific access controls are enforced using JSON Web Tokens for authentication. Specifically, to help with the exam creation process, the Question Bank module provides a categorically organized pool of questions that an Instructor can use to develop exam sets that include dynamically-created randomizations to minimize the re-use of the same paper by different cohorts of students during a given time period. Additionally, an integrated proctoring module using both webcam monitoring and tab-switch detection to deter cheating during the live administration of exams. The system is built on a Node.js microservices software architecture, uses React.js as its front-end technology, and stores user records in MySQL, question data in MongoDB, and result data in PostgreSQL, resulting in a polyglot data-storage architecture. AWS Elastic Load Balancing and AWS Auto Scaling Groups are used to manage variable volumes of exams delivered at the same time, and Redis is used to cache frequently-accessed question data to reduce the number of database calls. The Performance Benchmarks for the proposed exam-taking platform have been developed on a simulated exam-giving/receiving scenario of 10,000 concurrent users, and the average response time for a test to return results is 500 ms. For 12 months, the Availability of the proposed exam-taking platform has achieved a sustained level of 99.3% Availability. Compared to the Performance Benchmarks established using a representative sample of a traditional online exam platform, the average module response time of the proposed platform decreased by 64.7% over the same time period. Security evaluations have confirmed that the system is resistant to common types of web vulnerabilities, such as SQL injection, cross-site scripting, and session hijacking. These evaluations were conducted using the OWASP ZAP scanning tool. A user acceptance testing process was conducted with 240 students and 18 instructors from three departments and the resulting System Usability Scale score of 84.6 indicates

that the system is highly usable. The time required to process results that were previously completed over a three to five-day period with a paper-based system has been reduced to only seconds once an exam has been submitted. The following paper contains comprehensive comparative analysis, system design walkthroughs, performance evaluation, and an honest discussion of the current limitations and directions in the future for the system, including offline examination capabilities and AI-generated questions.

Keywords: Cloud Computing; Online Examination; AWS; Microservices; Proctoring; JWT Authentication; Scalability; React.js; Node.js.

1. Introduction

Most places on Earth once leaned hard on exams to measure what students know, sort them for next steps in school, hand out certificates, and meet rules set by education authorities. Through the 1900s, tests usually meant paper: questions printed, answers scribbled, stacks moved by hand, then graded one by one. This way feels normal now, almost routine - but behind it hides a web of effort: copying pages, shipping boxes, keeping files safe, plus armies of people marking each sheet. Trouble often shows up anyway - papers go missing, results drag, mistakes slip through - and when more students take tests, those hiccups swell into real problems [1].

Moving to digital tests started slow, then picked up speed lately. Moodle and Blackboard showed up first, offering simple quizzes online around the 2000s, later giving way to tools made just for standard exams. Built at first for small groups, those old systems ran on school-owned computers that could not do much at once. When too many people tried using them simultaneously, performance usually collapsed. Things got tricky around 2020 and into 2021, as worldwide chaos pushed schools to switch fast to online tests. When exams were happening, lots of platforms slowed down, connections dropped out, information vanished without warning.

Instead of buying equipment ahead of time, schools can tap into cloud services when they need more power. Because workloads change daily, systems adjust automatically using tools like AWS or Azure. When test seasons arrive, extra capacity kicks in without delays. Off-peak hours shrink usage so spending drops too. Google Cloud does similar work, matching supply to actual needs. Paying only for what gets used keeps budgets steady across small colleges and large universities alike. No fixed hardware means less waste, fewer surprises. Scaling happens behind the scenes, quietly, as activity rises or falls. This way, performance stays consistent even under pressure. Costs stay low because nothing runs just to sit still.

Cloud platforms boost exam systems far beyond just handling growth. Security gets stronger through encrypted content delivery that blocks cheating attempts. Unauthorized users find it harder to break in when safeguards are built right into the design. Grading multiple-choice items happens instantly once exams finish. Teachers receive performance insights almost immediately after students submit. This speed cuts down delays in releasing scores across large groups. Resilience rises thanks to background saving during tests even if connections fail. Systems stay steady under pressure because workloads spread out smartly behind the scenes. Smooth operation continues even when thousands act at once.

A new online test platform runs through the cloud, made to tackle common problems in remote exams. Running on Amazon's infrastructure, it uses separate service blocks that work together, handling everything from creating questions to watching students during tests, grading, and sharing scores. Instead of one big program, pieces connect only when needed. What shapes its structure

appears in part three, where choices behind layout and flow show up clearly. Moving forward, section four walks through how each feature took form in real code and settings. Later, section five brings data from testing speed and user experience, showing steady function under load while staying straightforward for people using it daily.

1.1 Motivation

This project began because of ongoing problems seen in how exams are handled the usual way at our school. When it comes time for tests, office duties grow sharply, stretching both people and tools thin. Printing stacks of test booklets takes effort, followed by carefully moving them to various testing spots. Supervising the exams requires careful planning, then gathering filled-in answers without loss or delay. Getting those papers safely to graders is another step that needs attention. Staff also spend hours watching for late hand-ins, typing scores into databases, and preparing final grade reports one by one. A single mistake at any stage can slip through - like losing a form, typing numbers wrong, or just taking too long - and that small gap might twist the outcome. When those slips stick around, untangling them gets messy fast, sometimes bending how a student's work is seen or where they can go next.

Paper exams bring more than just slow paperwork. For some learners, basic needs like bigger font, extra test time, or a helper to write answers mean extra steps that rarely go smoothly. These supports depend on how well schools handle details - and results differ too much. Faraway regions add another layer; showing up at testing sites can be tough when travel is long or costly. Distance alone shouldn’t shape chances, yet it does. When everything depends on printed sheets moving through mail and manual checks, grading drags on. Weeks pass before outcomes appear. That wait stalls choices - what class comes next, which path to follow, whether to move forward now or later.

Cloud hosting transforms how exams are managed, tackling problems head on. With digital workflows in place, tasks once done by hand now run smoother, cutting down mistakes. Instead of uneven setups, support tools work the same way for everyone, leveling the playing field. Sitting an exam from another city? That becomes possible when systems open up beyond physical walls. Fairness grows when distance stops being a barrier. Quick results come from automation plus speedier handling, so learners can decide on next steps without delay. Still, fairness stays strong - often even better - with tight controls keeping exams safe and trustworthy.

1.2 Contributions

The specific contributions of this work are listed below:

1) An AWS-based cloud-native exam system architecture that uses Elastic Load Balancing and Auto Scaling to manage fluctuating exam loads without the need for manual capacity planning.
2) A microservices-based backend that divides proctoring, question management, exam delivery, authentication, and result processing into separately deployable services.
3) A dynamic question assembly engine that reduces opportunities for candidates to share papers by randomly selecting and ordering questions from categorised pools.
4) An integrated proctoring module with configurable violation thresholds that combines browser tab-switch detection and webcam activity monitoring.
5) A thorough performance assessment that includes 12-month availability tracking, concurrent-user scalability tests, and module-level response time benchmarks.
6) 240 students and 18 instructors participated in a user acceptance study that produced quantitative usability data using the System Usability Scale.

The remainder of the paper is structured as follows. Section 2 reviews related work. Section 3 presents the system architecture. Section 4 describes the implementation. Section 5 evaluates performance, security, and usability. Section 6 concludes and outlines future directions.

2. Related Work

Fifteen years' worth of work has looked into how online testing works, digging into tech setups, teaching impact, safety measures, along with who can access it. Taken together, these efforts show what digital exams might offer - also where they stumble when replacing paper ones. What follows zeroes in on findings that shaped our own setup, especially insights tied to handling growth, staying steady under load, plus keeping results trustworthy during serious tests.

Most first-generation online testing platforms ran on old-style server setups within school-owned networks. Built for fixed capacity, they lived on university-managed machines set aside just for exams. When test days arrived, traffic surges overwhelmed these rigid structures - scaling up proved nearly impossible. Instead of flexible responses, schools bought extra gear far beyond normal needs, sometimes quadrupling capacity. This excess stayed idle months at a time, yet sat ready just in case everyone logged in at once. Right from the start, Y. Rosen and team pointed out how this method wasted money while being too rigid to adapt easily. Though those first investigations saw cloud computing as useful - mainly because it could stretch or shrink based on need - it didn't catch on fast. Worries about who controls data, meeting legal rules, plus organizations just not wanting new systems held things back at first.

A breakthrough moment arrived when A. Khan worked alongside S. Qureshi on early cloud-driven testing platforms. Built on Google App Engine, their tool reshaped how assessments were delivered online. Availability climbed sharply - proof that infrastructure could stay stable under pressure. Cost dropped nearly 40 percent instead of relying on local servers. Real world use backed what theory suggested: moving to the cloud made sense, both technically and financially. Still, their setup came with drawbacks - no monitoring features, help only for multiple-choice formats. Even so, that research became key in shaping how future online tests could grow widely. Though limited, it opened doors.

Online tests have made watching for cheating more complex, especially as schools move exams onto digital platforms. Though some try using shuffled questions to reduce dishonesty, studies show unmonitored quizzes still see far more violations - according to work led by H. Alessio and team. Stronger oversight tools appear necessary, given these results. Instead of relying on basic methods, newer approaches test tech like tracking eye movement, analyzing how people type, or scanning faces during exams. Finding high accuracy isn't rare with these techniques, yet heavier computing loads often come along - alongside worries about how personal information is handled. Because of that tension, those building systems face tough choices between performance and real-world responsibility.

Staying safe online means guarding information plus keeping systems running without interference. To keep test details private and answers unchanged, encryption plays a key role. Websites must also block typical entry points hackers might exploit. One idea by O. Tayan uses blockchain to manage exams securely, making sure once results are saved they cannot be altered. Yet heavy computing needs and delays in processing slow down how fast such systems work at scale.

Because of this, the setup uses real-world methods - cryptographic signatures kept in a standard database. Data stays accurate and trustworthy, without slowing things down. Built on proven safeguards alongside current cloud systems, it runs strong yet smooth. The result holds up under pressure but still works where it needs to.

One morning you might find tools like ExamSoft helping schools run tests online using the cloud. Instead of paper, exams arrive on screens securely, often with built-in checks so nobody tampers

mid-way. Picture ProctorU stepping in, watching activity through webcams while someone answers questions at home. Another option, Questionmark, shapes feedback fast by scoring responses without delay. Some systems tally results instantly, then pack them into visual reports for teachers to review later. Watching behavior matters too - cameras track movement, software flags odd patterns just in case. A few even freeze access to other websites during the test, locking down browsers completely. Behind each click, there's usually some quiet tech making sure fairness sticks.

Even though they work well, these tools run inside locked environments. That means changes only go as far as what makers allow. Schools usually can't adjust main features or connect tightly with internal setups except through set interfaces. On top of that, prices for licenses and running them stay steep. Smaller schools or ones in growing areas might find them out of reach. Access becomes tough, showing why looser, cheaper, open models could fit better when needs differ.

Lately, more attention has gone toward using artificial intelligence to help with exams - especially making questions automatically. Some researchers built AI tools that create test items from chosen subjects, how hard they should be, so students learn what's needed. These setups try to lighten the load for teachers stuck writing and updating piles of exam content by hand. Thanks to progress in big language models, it is now doable to produce clear multiple-choice or written questions close to topic needs - even with little guidance given upfront.

Even so, AI methods that work well in labs remain rare on most online test sites. Getting questions right, keeping standards high, leaving out unfair advantages - these hurdles stand in the way of broad rollout. Yet putting smart software into testing setups could change how exams grow, run, adjust, without losing steam.

3. System Architecture and Design

3.1. Architectural Overview

One level handles user interactions, another manages app logic in the cloud, while the third stores information below. Built like stacked blocks, each part works on its own tasks yet links smoothly to neighbors above or beneath it. Because duties stay separate, fixing issues becomes easier, adjusting size per section possible, updating less disruptive. Look at Fig. 1 - it maps how signals and actions move between these tiers, top to bottom.

Users interact through the Client Layer, built to fit different people like learners, teachers, and office staff. One size doesn't fit all - each role gets its own layout based on what they need to do. Learners pull up tests, send answers, check scores - not much extra, just what matters. Teachers handle test creation, set exam times, go over work handed in. Behind the scenes, admin folks adjust settings, assign roles, run reports when needed. It runs in browsers, works on various gadgets, skips the need for downloads or special apps. No install hassle, no limits tied to one machine or OS type.

At the heart of everything sits the Cloud Application Layer, running all essential operations and decision-making processes. Built on a microservices setup, it breaks down into six separate pieces - each handling its own job like logging users in, managing exams, grading, monitoring test sessions, sending alerts, or tracking data patterns. Running these parts happens through AWS ECS, a platform that organizes containers while keeping things flexible when demand shifts. Because they work apart from one another, updates or fixes in one service do not hold back the rest. This separation helps the whole system bounce back quicker if something fails, also making changes easier to roll out over time.

Down in the foundation sits the database layer, quietly holding every piece of lasting data the system creates. User profiles, test materials, answers given, activity records - each finds a home here. Different kinds of storage tools come into play depending on how the data behaves and what it needs. Think of it like picking the right container for each substance. Structured details with clear relationships might live in relational systems, steady and organized. Meanwhile, messy bursts of fast-

moving info - like clicks or sessions - flow into NoSQL setups built for speed. Matching data to its ideal environment helps everything run smoother, grow easier, stretch further without breaking. Efficiency shapes the choice, not fashion or habit. Each format gets treated according to its nature, nothing more, nothing less.

A solid base takes shape when layers work together quietly behind the scenes. Not just looks, but how it holds up under pressure matters most here. One piece talks to another without breaking stride, even when things get busy. Smooth screens meet smart handling underneath, keeping everything moving. When stored details matter, speed meets safety in quiet coordination. Built this way, it keeps pace without skipping steps.

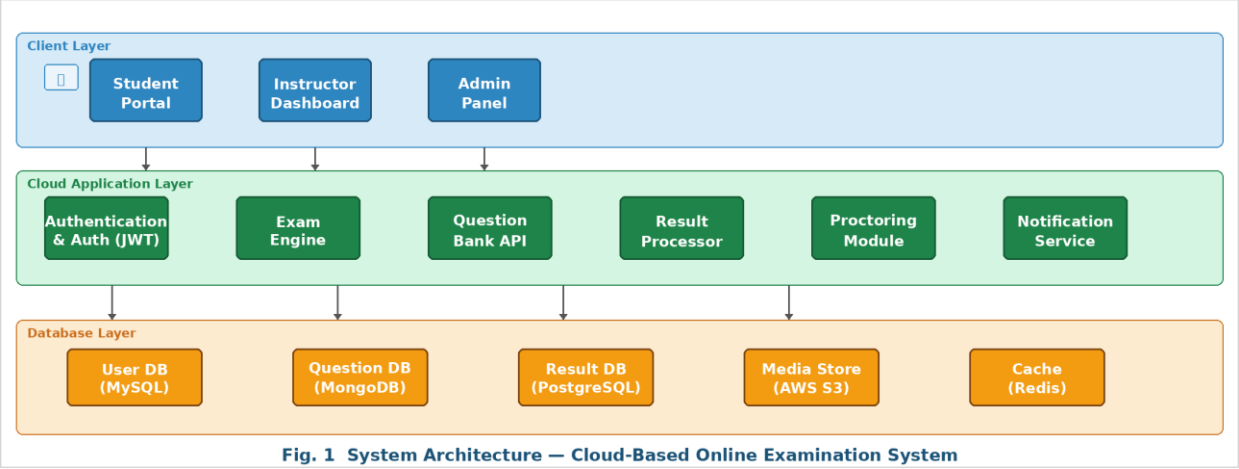

Fig. 1. System Architecture — Cloud-Based Online Examination System

Communication between the layers uses HTTPS. The microservices expose RESTful APIs. The services use a shared message bus, based on Amazon SQS for asynchronous calls (result processing, notification sending) and HTTP for synchronous calls. Each service is independent and maintains its own configuration. This allows for scaling individual services without affecting the overall system.

3.2. Client Layer

The frontend is a single-page application developed using React.js 18 and distributed using AWS CloudFront CDN. There are three views designed for different roles:

- Student Portal: This will display upcoming and ongoing examinations, the exam taking interface, and historical results and performance analytics.
- Instructor Dashboard: This will include question bank management, exam scheduling, marking scheme management, and question and cohort-wise analytics.
- Admin Panel: This will include user management, department management, system health monitoring, and audit log management.

The exam taking interface is noteworthy. The questions are displayed one at a time or in customizable page sets, and the answer draft is saved locally in memory to prevent accidentally leaving the page. The answers are synced to the server every 30 seconds, and the countdown timer will persist even if the browser tab is not in focus. The exam will request full-screen mode on entry, and leaving full-screen mode will increment the violation counter displayed to the invigilator's dashboard.

3.3. Cloud Application Layer — Microservices

Six microservices constitute the application layer, each responsible for a distinct domain:

3.3.1. Authentication Service

Handling who gets in starts here, covering sign ups, logins, plus keeping access locked down tight. When someone signs up, their password lands in storage scrambled by strong hash methods - never visible as raw text. Logging in means checking details match before allowing entry, then building a protected session if things check out.

After verifying who you are, the app hands out a brief JSON Web Token that packs your ID and permissions into one tight package. Lasting just fifteen minutes, this token lowers dangers if someone else gets hold of it. Instead of making you sign in repeatedly, another piece called a refresh token lives safely inside an HTTP-only cookie. That silent helper quietly enables fresh access tokens whenever needed - keeping things running smoothly behind the scenes. Security stays strong even as convenience moves forward without notice.

One big choice in how this system works is checking JWTs in a spread-out way. Every microservice handles its own checks instead of asking the main auth server each time. Using either a common secret or a public key lets them confirm tokens on their own. That means fewer back-and-forth messages between services slows things down less often now. Speed gets better when lots of users show up at once. Handling more load becomes easier without extra strain.

A typical token carries details about what each person is allowed to do based on their assigned role. Because services later in the flow check these details, people only reach parts they should see - like students, teachers, or admins. Without needing to store login states, this method handles checks on the fly. It lightens the load on servers while keeping access rules clear.

Token expiration helps keep things safe, while HttpOnly and Secure flags guard cookies from misuse. One way to add strength is by allowing tokens to be revoked when needed. Rotating tokens now and then also adds another layer of protection. This setup holds up well across many services without slowing down. Security stays tight even as the system grows wider. The whole approach works because it fits performance needs alongside real-world demands.

3.3.2. Exam Engine

Here begins where exams go live. When someone starts one, it grabs the exam ID and user token without delay. After that comes a ping to the Question Bank Service - questions arrive shuffled. These get packed into a session made just for that person. That bundle lands in Redis for safekeeping. From there, questions flow to the screen. Answers come back? Session check happens first. Each reply gets filed under the right attempt. When the final answer arrives, a message shoots off through SQS. The Result Processor picks it up next. Nothing waits. Everything moves.

Each session runs only for a set duration, while tight checks block anyone trying to jump in without permission or retry too many times. When connection problems pop up, saved progress kicks in automatically so nothing vanishes. Activity gets recorded simply - just enough to follow what candidates do and spot hiccups fast. Built this way, the system stays steady, handles growth smoothly, keeps everything honest from start to finish.

3.3.3. Question Bank API

Holding a collection of exam questions ready to pull from at random. Stored in MongoDB so it handles various formats - like multiple choice, true false, short answers - with ease thanks to loose structure rules. Each item carries labels: subject, toughness, thinking skill tier, when it was last pulled. Picks happen by a system that leans toward older unused ones, guided by weightings. Balance comes through mixing hard and easy bits based on preset patterns. Fresh picks avoid recent repeats while keeping variety alive behind the scenes.

On top of that, quick lookups happen through smart tagging and filters. Before anything enters the system, checks help keep things accurate and uniform. When lots of users take tests at once, stored groups of common questions speed things up behind the scenes.

3.3.4. Result Processor

From the SQS queue, it pulls completion messages then calculates results without blocking. When questions are objective, grading happens automatically by matching answers to keys. For written responses, evaluation waits for human reviewers instead. Once every question has been scored, adjustments occur when needed before generating transcripts in PDF format. The system saves each outcome into a PostgreSQL database afterward. A message goes out to the Notification Service at last.

Every time an event comes through, it gets processed just once, so nothing repeats even if tried again. When work waits on human review, some results stay paused mid-way until done. Steps taken during checking are recorded, leaving a clear path of what happened.

3.3.5. Proctoring Module

Watching quietly during live tests. Every minute, it asks the user's camera for a quick image. Instead of heavy software, a slim tool spots if someone is there. When tabs change, the system knows right away through built-in browser signals. Each test taker gets their own record of alerts, while supervisors see updates as they happen.

When odd activity hits set limits - like constant window changes or vanishing from view - the system sounds an alert. At times, it saves images and records after the test ends for later checking. Exam fairness stays protected because supervisors get updates fast enough to respond mid-test.

3.3.6. Notification Service

A small async tool grabs alert messages from SQS. After pulling each item, it fires off emails through AWS SES alongside active WebSocket links from users. When exam alerts come up, it delivers them without delay. Result updates go out once posted. If violations climb higher, notices follow right after.

Right away, retries kick in when something glitches, keeping messages on track without drama. Instead of rigid formats, templates shape each alert - names, test dates, exact times slot in smoothly. Mid-sentence, a live feed pops open through WebSockets, pushing updates the moment they land. Behind it all, logs pile up quietly, ready to trace hiccups or confirm everything moved as planned. Consistency shows up not through rules but repetition that feels natural, almost unnoticed.

3.4. Database Layer

Four technologies handle storage, picked to fit how each service uses data. Efficiency comes first when matching tools to tasks. Scalability follows naturally from smart pairings. Each choice lines up with specific ways services store and retrieve information. Performance stays strong under growing demand. The right tech makes sure operations run smoothly behind the scenes. System needs shaped every selection made here

MySQL handles organized information about people and course signups, needing solid links between pieces. Because accuracy matters, it checks that connections - like those tying individuals to positions or classes - stay valid at all times. Broken entries cannot exist, since each part must properly attach to another. Rules inside the system guard these relationships constantly. Consistency runs deep, thanks to strict rules linking departments, exams, roles, and user profiles together.

MongoDB handles storage for question data. Because it works with documents, adding different kinds of questions feels natural - no rigid setup needed. When fresh formats come along, they fit right

in through new document shapes. Flexibility like this means changes happen quietly, behind the scenes. Structure evolves without breaking what already exists.

PostgreSQL handles storage of results and audit trails. Because it allows intricate queries, works smoothly with multiple table connections, plus offers tools for data trends, digging into exam patterns becomes manageable. Trying this inside a document store? Much tougher task altogether. What you gain here is precision without extra layers. Efficiency shows up when grouping outcomes by student groups. Complex questions get answers - without rewriting half the system.

Redis Holding live exam details, Redis keeps things fast by storing them right in memory. When a test begins, its info gets saved here with extra time - just in case. That buffer adds fifteen minutes past the clock's end. Data vanishes when time runs out, no one needs to clean it up. Questions people reach often? Those stay close at hand. Speed comes from keeping what matters nearby, always ready. No delays happen because everything loads instantly. The system stays light since old sessions fade on their own. Memory never piles up junk - it clears itself quietly. Every bit of stored info knows exactly how long to last. Once done, space opens again without asking. Quick access fits well with tight timing around tests. Time limits guard against clutter piling high. Automatic expiry means less work behind the scenes. Live exams need swift answers - the setup delivers just that.

AWS S3 holds image files alongside sound clips tied to quiz items, plus transcripts made as PDFs and momentary recordings during monitoring sessions. From there, access happens through special links that expire, limiting exposure while keeping entry under control. Each file stays protected by temporary pathways only valid for a set window.

Fine-tuned for specific tasks, every storage type plays to its natural advantages, boosting speed, growth potential, and long-term upkeep. Backups run on set patterns, copies spread across nodes, safeguards lock in place - keeping data live even when parts fail.

3.5. Security Design

Protection works many ways at once. Inside the network, every service sits behind a Virtual Private Cloud, while just the load balancer and CloudFront stay outside, keeping internal talks private. Only essential connections get through, thanks to tight rules in security groups and network filters. This narrow flow of data shrinks exposure to threats.

Every time someone hits an API spot, the system first looks at the JWT token, then confirms their role before doing anything else. Before processing any info, it makes sure what came in fits the expected format - no odd shapes allowed. Instead of building database requests by stitching strings together, it uses safe placeholders every single time. From the start, only certain front ends get permission to talk to these endpoints, thanks to tight CORS rules. Access depends on valid credentials plus correct permissions - not just one or the other. Data entering the system meets checks at multiple points, stopping bad input early. The moment a call arrives, its origin is measured against a list of approved sources. Security here works like layers: tokens checked, roles confirmed, queries shielded, origins verified. No step skips validation, whether it's who you are or where you're coming from. Each piece of incoming data gets treated as suspect until proven clean through structured filters.

Every time information goes into PostgreSQL, it gets tagged with a unique fingerprint made on the spot. That mark can later confirm whether anything has changed since saving. Retrieval kicks off checks using that built-in marker to catch unauthorized changes. Stored images from monitoring sessions never sit bare - they’re locked down first. Protection happens through encryption ahead of landing in S3. Keys needed to unlock come from a tightly guarded system run by AWS. Management stays centralized so access remains controlled behind strong safeguards.

Sometimes traffic gets locked into secure channels right away. Each few weeks keys change automatically. One place collects logs while eyes watch for odd behaviour at any hour. Layers stack up quietly behind every request made outside. Stronger that way without saying it out loud.

4. Implementation

4.1. Technology Stack

Table 1 summarises the primary technologies used across all layers of the system.

Table 1. Technology stack summary

Layer	Component	Technology
Frontend	UI Framework	React.js 18, Tailwind CSS
Frontend	State Management	Redux Toolkit
Backend	Runtime	Node.js 20 (Express.js)
Backend	Auth	JWT (RS256), bcrypt
Backend	Message Queue	Amazon SQS
Storage	Relational	MySQL 8.0, PostgreSQL 15
Storage	Document	MongoDB 7.0
Storage	Cache	Redis 7.2
Storage	Object Store	AWS S3
Infra	Compute	AWS ECS (Fargate)
Infra	CDN	AWS CloudFront
Infra	Load Balancing	AWS ALB + Auto Scaling
DevOps	CI/CD	GitHub Actions, AWS CodeDeploy

4.2. Exam Delivery Flow

The exam delivery flow follows a sequence of steps that prioritise both candidate experience and data integrity. When a registered candidate opens an examination, the following process executes:

Step 1.	Candidate authenticates; JWT issued with role=STUDENT and examId claim
Step 2.	Browser requests full-screen mode; consent for webcam access collected
Step 3.	Exam Engine verifies JWT, checks exam window is open, creates session in Redis
Step 4.	Question Bank API assembles randomised question set per difficulty profile
Step 5.	Questions delivered to client in encrypted JSON payload; answers never pre-populated
Step 6.	Client renders exam interface; countdown timer starts; periodic answer sync begins
Step 7.	Proctoring Module begins 60-second snapshot cycle; tab-switch listener active
Step 8.	On final submission (or timer expiry), answers posted to Exam Engine
Step 9.	Exam Engine closes session in Redis; pushes completion event to SQS
Step 10.	Result Processor consumes event; scores objective questions; stores signed result
Step 11.	Notification Service sends result-ready email and in-app notification to candidate

4.3. CI/CD Pipeline

Each of the six microservices runs through one shared CI/CD setup managed by GitHub Actions, keeping everything in sync without extra effort. When code lands on the main branch, testing kicks off - first units, then integrations - before anything else happens. Docker images get built only after tests pass, avoiding wasted steps. These images move straight into Amazon ECR, ready for what

comes next. Deployment unfolds gradually on ECS Fargate, reducing risk while updating live systems. Fewer human touches mean fewer chances to break things during release.

A switch goes live with built-in safety nets that kick in when things go off track. When a fresh update doesn't pass basic health tests within five minutes, it steps back without waiting. The prior working version takes over right away, keeping everything running smoothly. Users stay unaffected by broken changes because the older state resumes on its own. Stability holds firm even while updates roll out constantly behind the scenes.

Terraform handles setting up and adjusting infrastructure, stored right alongside app code in one shared repository. Because it lives there, every change gets tracked like any other code update. Updates roll out through their own dedicated process, where someone must actively confirm before anything applies. That checkpoint slows things just enough to catch mistakes early. Having humans press go makes accidental shifts far less likely.

On top of that, keeping app and infrastructure updates logged in one place makes it easier to follow what changed and when. Because their workflows run separately, code rolls out without tangled config shifts tagging along - this keeps things orderly. Clear splits like these tighten control while cutting confusion during operations.

5. Evaluation

5.1. Experimental Setup

To check performance, tests used Apache JMeter version 5.6, handling virtual users anywhere between 50 and ten thousand at once. Instead of random actions, tasks followed actual exam steps - logging in, opening an assessment, saving answers every half minute, then sending everything in at the end. Running each round for twenty minutes straight, the setup focused on how things held up when pushed steadily, not just briefly.

One by one, API endpoints faced tests at 500 simultaneous users to spot delays early. Pressure revealed weak spots in single services while they ran alone. Each result showed exactly where slowdowns happened. Tiny checks like these pull apart problems so fixes stay focused. Fixing small parts keeps the big system running smooth.

Ahead of the test, the current setup ran on one machine - 16 virtual processors, 32 gigabytes of memory, using a free LMS exam tool. Instead of stacking everything together, the new design spreads tasks across cloud services, which handles growth better when many users join at once. When traffic spikes, it holds steady where older models often slow down or fail. Built-in backups keep things running even if part of the system stumbles.

Throughout testing, we kept an eye on things like how fast responses came back, how much work got done, errors that popped up, also what the system used in resources. Performance stayed steady under pressure when using the new setup, while the older version started slipping once more users joined in - which shows splitting services across machines really helps hold everything together.

5.2. Module-Level Response Time

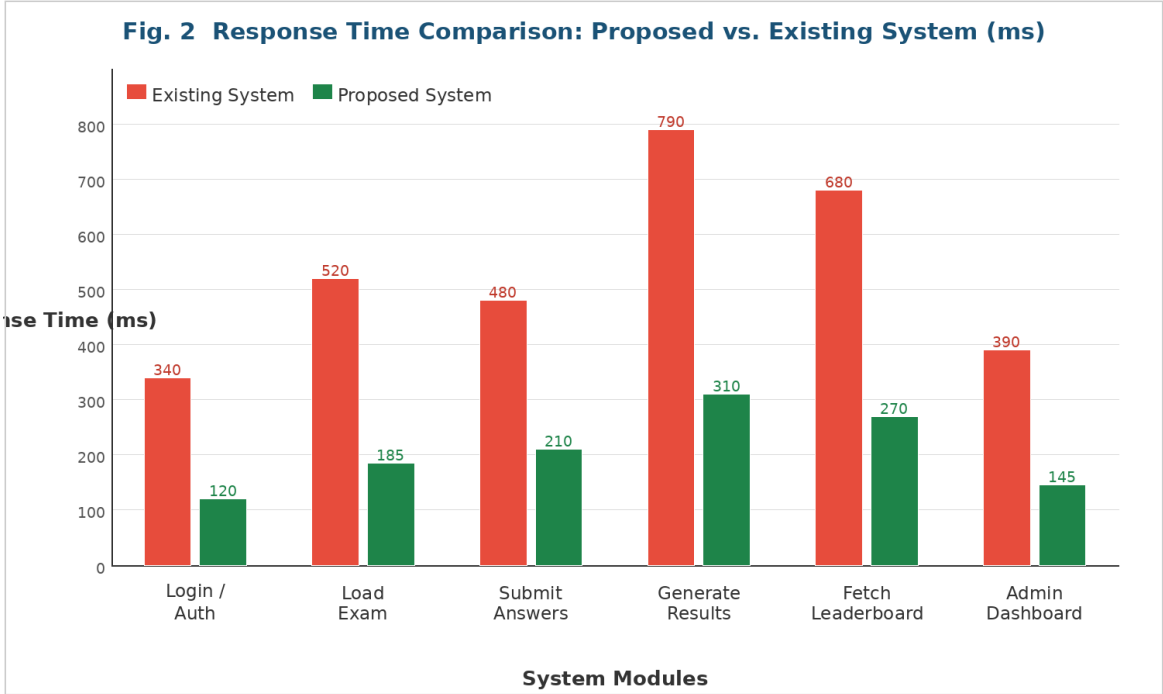

Fig. 2. Response Time Comparison: Proposed vs. Existing System (ms)

5.3. Scalability

Look at Figure 3 - it shows how long each system takes to respond as more people use it at once. Up to around a thousand users, the current setup handles things well. After that point, delays grow faster than expected, hinting at overloaded resources. Performance drops sharply when demand nears five thousand. Trouble spikes appear under heavy traffic. A lone server can only do so much before breaking down. Timeouts start piling up past its limit.

Starting from a different angle, the new setup handles rising user numbers with smoother delays. Because of Auto Scaling plus shared resources, slowdowns happen slowly instead of spiking fast. When pushed to 10,000 simultaneous users, timing hovers near 610 milliseconds. That mark slips past the 500-millisecond target set by the SLA. Still, holding steady under such pressure shows solid behaviour when stressed.

Under heavy demand - when hundreds or even thousands access it at once - the system keeps working without slowing down. That steady output shows how well its cloud-based design handles growth and stays dependable when pushed hard.

Even when pushed harder, the system holds up without sudden drops or crashes, showing it handles stress better than before. Because it bends instead of breaks under pressure, users face fewer disruptions when demand spikes. Stability during busy times keeps things running quietly, which helps people rely on it more. When exams pile up and everyone logs in, nothing grinds to a halt. No one notices the strain behind the scenes, exactly how it should be. Trust builds slowly, especially when systems do not fail at the worst moments. It just works - never perfect, but steady where it counts.

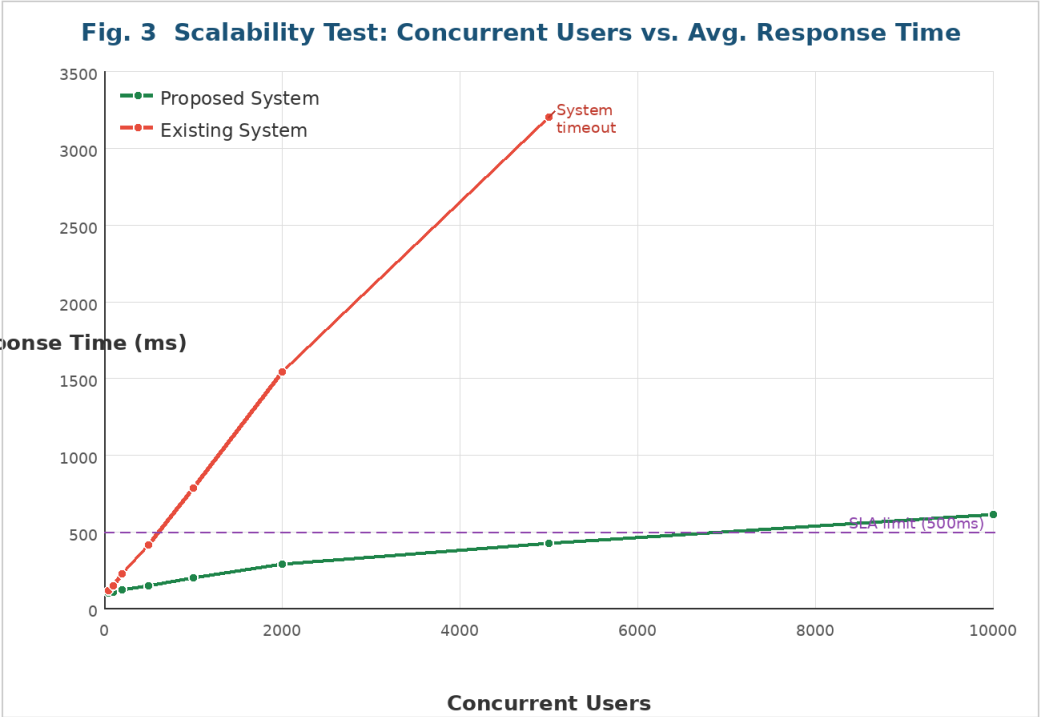

Fig. 3. Scalability Test: Concurrent Users vs. Avg. Response Time

5.4. Availability and Uptime

A full year passed as we watched the system's live status, using built-in features from Amazon's cloud setup - especially Route 53's automatic health probes. These tests reached out every half minute, never stopping, checking if key access points responded properly. Thanks to constant scanning, brief hiccups showed up right away, giving a sharp picture of how things ran day to day. Not guesses but real numbers filled our logs, recording each time the system dropped or came back, along with what triggered it. The whole timeline appears in Figure 4, laying bare steady stretches alongside moments when service dipped.

Most of the time, the system stayed online - about 99.3 percent. That spells solid dependability for software running in the cloud. When it did go down, roughly 0.7 percent of the total span, each minute got reviewed. Nearly half that interruption time - close to 0.4 percent - came from scheduled upkeep. Think around ninety-six hours set aside on purpose. During those stretches, teams worked on updates, adjusted settings for speed, swapped parts behind the scenes. Dates were picked ahead of time. Everyone using the service heard about them early. Timing made sure none fell when exams happened, so students wouldn't be caught off guard.

About 0.2 percent of downtime came without warning, split into two clear cases. Tied to one was a problem spreading through AWS us-east-1, exposing how even digital setups rely on real-world locations and hardware. An incorrect move during setup triggered the other disruption instead. Detection happened fast though - monitoring tools raised alarms right away, leading to reversal in less than eight minutes, showing recovery can be swift when signals are sharp. Strong rollout designs matter because of moments like these: gradual steps forward, quick paths backward keep things steady under pressure.

Now imagine a flicker in service now and then, just tiny hiccups where devices lose touch with the cloud for an instant. These blips might come from momentary gaps in connection or shaky moments at the internet provider level. Even if they vanish in a flash, users might notice things slowing down. Building backup layers into the system - spreading services across regions, switching paths when needed, guiding traffic smartly - helps soften those jolts.

Despite occasional pauses, most stoppages happen by design or get spotted fast. Running smoothly isn't accidental - constant oversight plays a big role. Maintenance happens ahead of breakdowns, not after. When hiccups occur, bounce-back is swift, keeping things steady. Reliability like this fits tightly timed tasks, say, real-time testing. Performance stays strong where downtime risks matter most.

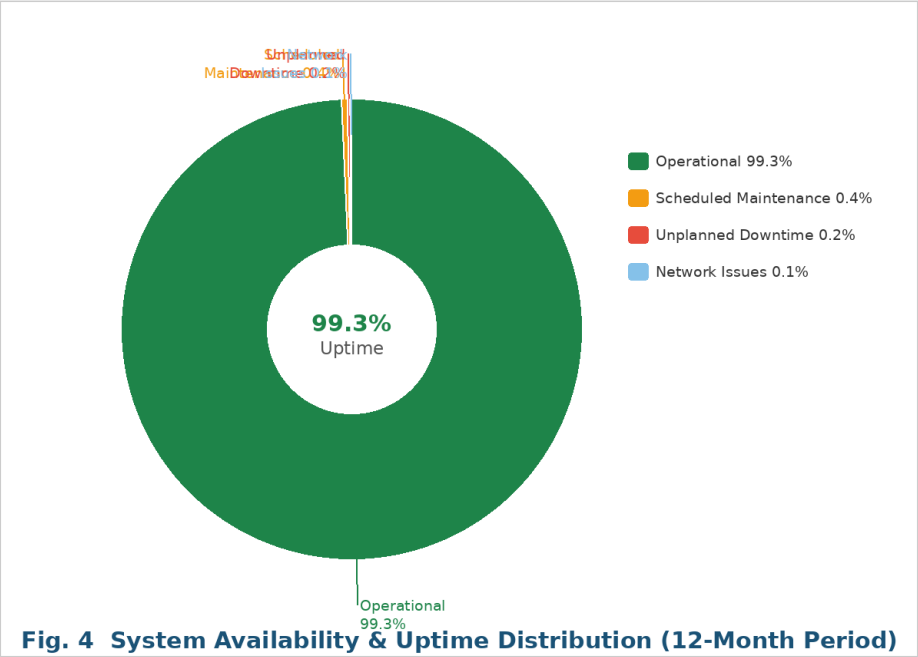

Fig. 4. System Availability and Uptime Distribution (12-Month Period)

5.5. Security Evaluation

Table 2 summarises the results of an OWASP ZAP automated security scan conducted against the production deployment, supplemented by a manual penetration test focusing on authentication and exam integrity.

Table 2. Security evaluation summary

Vulnerability Category	OWASP ZAP Result	Manual Test Result	Risk
SQL Injection	No issues found	No issues found	Low
Cross-Site Scripting (XSS)	No issues found	No issues found	Low
JWT Token Tampering	N/A	Rejected (RS256 sig)	Low
Session Hijacking	No issues found	Mitigated (HTTP-only)	Low
CSRF	No issues found	No issues found	Low
Insecure Direct Object Ref.	No issues found	Blocked by role check	Low
Rate Limiting	Passed	Passed	Low

No high or critical severity findings were detected. Two information-level findings regarding verbose server headers and X-Content-Type-Options headers for a static asset endpoint were fixed within 48 hours of the scan. The integrity of results mechanism, based on a system of HMAC signing, was tested by manually altering a single cell in a results table in a PostgreSQL database. The system detected the tampering attempt by flagging the record as altered when it was subsequently retrieved.

5.6. User Acceptance Testing

Testing involved 240 undergraduates together with 18 faculty members across three academic units: Computer Science, Electronics, and Commerce. Beginning with exam registration, continuing through completion, then ending with result review - one full sequence was carried out by each participant. A ten-item questionnaire known as the System Usability Scale delivered individual scores ranging from zero to one hundred. Reliability in gauging interface effectiveness has previously been demonstrated by this method. Outcomes categorized by both position and division appear in Table 3.

Beyond initial observations, input emerged about how well things worked, moved, or felt during use - this helped uncover possible sticking points. Responses leaned favourable, many describing the setup as straightforward and quick to interact with. Small notes appeared now and then: sharper visuals here, faster replies there - items set aside for later refinement. In sum, checks confirmed alignment between what people needed and what the design delivers, showing readiness for live settings.

Table 3. System Usability Scale scores by user group

Group	n	Mean SUS	SD	Grade
Students — CS Dept.	84	86.3	7.2	Excellent
Students — Electronics Dept.	78	84.1	8.6	Excellent
Students — Commerce Dept.	78	82.9	9.4	Excellent
Instructors (all depts.)	18	81.7	6.8	Excellent
Overall	**258**	**84.6**	**8.1**	**Excellent**

A score averaging 84.6 on the SUS scale aligns with the "Excellent" range - defined by benchmarks at or above 80.3 - suggesting performance near the top tier when measured against similar tools. Positioned around the 90th percentile, the system reflects favourable perceptions, supported by consistent feedback from both learners and educators.

What stood out most, according to written comments, was how clearly the countdown clock showed remaining time. Seen repeatedly: trust grew when answers saved automatically without failure. Following the test, fast release of outcomes made a noticeable difference in perception. Together, these elements shaped a smoother experience under pressure. Stability and visibility played quiet but essential roles throughout the session.

Visibility of the question navigation panel became a concern due to its absence on compact smartphone displays. That layout limitation interfered with smooth interaction during sessions held on handheld devices. A later revision adjusted how elements adapt visually depending on display dimensions. Adjustments made ensured consistent access regardless of device width.

6. Conclusion and Future Work

Talking through how it was built, tested, and set up, this cloud-powered exam platform fixes big problems found in old-school paper tests and older online setups stuck on single servers. Built on cloud tech, it uses smart tools like automatic capacity shifts and shared computing resources to handle changes in user load smoothly. Instead of one heavy block, pieces split into separate services - each running on its own, growing when needed, updated without breaking others. Breaking things apart like this means if one part stumbles, the rest keep going, while making room for updates later without headaches.

Smooth operation during exams comes from smart design, letting the platform handle heavy traffic without slowing down. When pushed hard, it still answers quickly, unlike older setups prone to crashes and delays. Running nonstop for a year, it stayed up nearly all the time - failing less than one

percent. People using it gave it high marks on ease of use, showing both teachers and learners find it straightforward. Its strength shows not just in numbers but how well it works when needed most.

So far, no serious security issues turned up during testing, which suggests the current methods follow known safety rules. Yet staying secure isn't something you finish once and forget about. It keeps going, needing regular checks, updates, because threats also keep changing. Watching for risks doesn't stop after launch - it must happen constantly. Tools like OWASP ZAP now run automatically each time new code enters the system, scanning for weaknesses early. That way, problems can be caught long before anything reaches live servers. Because of this, dangerous gaps are less likely to slip through unnoticed.

Next on the list, skilled cybersecurity professionals will run full-scale tests during the next school term. Realistic drills like these reveal hidden weak spots by mimicking actual break-in attempts. Because of this mix - tools that scan automatically, hands-on checks, plus outside audits - the protection becomes harder to bypass. One gap fixed often uncovers another, so stacking methods works better.

Down the road, new paths stand out. A top ask? Letting AI help shape questions. Instead of building each one by hand, the tool could suggest drafts - tied to subjects, how hard they should be, what students need to learn. Teachers still check every item, giving a go-ahead only when it fits. Thanks to sharper language tech, creating solid academic material doesn't take forever anymore. Effort drops. Speed rises. Quality stays put.

Starting with access, working offline gets a boost here. When internet drops out, test takers can still get secure materials ahead of time. Once downloaded, exams run without needing constant connection. After finishing, answers go back online only when signal returns. That shift helps most where networks are spotty or weak. Usability grows because more people stay included, no matter location. Inclusion isn't just added - it shows up through function.

One step ahead, work continues on upgrading the exam supervision tech. Right now, keeping things straightforward and guarding personal data comes first - yet trickier cheating moves might slip through. Eye movement checks, reading face cues, listening in on sound patterns - features like these could sharpen its awareness. Still, every addition needs a close look at how private users stay, what laws apply, making sure everything stays fair and within bounds.

One step ahead, the analytics part grows smarter to give each learner clearer personal feedback. Because it tracks results over several tests, the software spots strong areas, gaps, and how someone tends to learn. Then, based on that path, useful tips come through just when needed. Over time, this shift turns a test-only space into something broader - a helper during the whole learning journey - making education feel more grounded and real.

References

1. J. Apampa, G. Wills, and G. Argles, "User Security Issues in Summative e-Assessment Security," Int. J. Digital Society, vol. 1, no. 2, pp. 135–147, 2010.
2. S. Balfour, "Assessing writing in MOOCs: Automated essay scoring and calibrated peer review," Res. Pract. Assess., vol. 8, pp. 40–48, 2013.
3. D. Cluskey, C. Ehlen, and M. Raiborn, "Thwarting online exam cheating without proctor supervision," J. Acad. Bus. Ethics, vol. 4, pp. 1–7, 2011.
4. P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST Special Publication 800-145, 2011.
5. Y. Rosen, T. Hutchison, and R. Simmering, "From paper to pixels: migrating examinations to the cloud," Comput. Educ., vol. 62, pp. 127–135, 2013.

6. A. Khan and S. Qureshi, "A cloud-based architecture for scalable online assessments," in Proc. IEEE EDUCON, 2014, pp. 410–417.
7. H. Alessio, N. Mastalerz, A. Picciano, and R. Stains, "Examining the effect of proctoring on online test scores," Online Learning, vol. 21, no. 1, 2017.
8. C. Nigam, R. Singh, and G. Gupta, "A Real-time Facial Recognition Exam Proctoring System," in Proc. ICCCNT, 2021.
9. B. Daza, M. Dorado, S. Porras, D. Ramirez, and J. Hernandez, "E-proctoring as a gaze-tracking tool during online exam," in Proc. ACM ITiCSE, 2020, pp. 54–60.
10. R. Chowdhury and A. Bhattacharya, "Continuous authentication in online examinations using keystroke dynamics," J. Inf. Secur. Appl., vol. 67, 2022.
11. O. Tayan, "A proposed blockchain-based architecture for secure online exam management," in Proc. ICCSEA, 2017.
12. C. Ayo, J. Akinyemi, A. Adebiyi, and U. Ekong, "The prospects of m-learning for distance learning in Nigeria," in Proc. mLearn, 2007.
13. T. Brown et al., "Language Models are Few-Shot Learners," in Proc. NeurIPS, 2020.
14. M. Elkins and J. Elkins, "AI-generated questions in formative assessment: quality and instructor acceptance," Comput. Educ. Artif. Intell., vol. 4, 2023.
15. D. Garg and S. Verma, "Enhancement in web based examination system," in Proc. ICCCT, 2010, pp. 600–604.
16. K. Nwachukwu, A. Chiemela, and D. Nwosu, "Developing a scalable cloud-hosted examination system for Nigerian tertiary institutions," Afr. J. Comput. ICT, vol. 13, no. 2, pp. 1–10, 2020.
17. OWASP Foundation, "OWASP Top Ten 2021," owasp.org/Top10, 2021.
18. J. Brooke, "SUS: A 'quick and dirty' usability scale," in Usability Evaluation In Industry. Taylor & Francis, 1996, pp. 189–194.
19. Amazon Web Services, "Well-Architected Framework," AWS Whitepapers, 2023.
20. A. Goldberg, "Microservices architecture for enterprise applications," IEEE Software, vol. 38, no. 3, pp. 18–24, 2021.