International Journal of Engineering and Science Invention (IJESI)
ISSN (Online): 2319-6734, ISSN (Print): 2319-6726
www.ijesi.org ||Volume 15 Issue 9 September 2026 || PP 18-26

Design and Implementation of a Secure MERN Stack-Based Online Examination System with JWT Authentication

Megavasan S[1] and Ramesh T R[2]
[1,2]*School of Science and Computer Studies, CMR University, Bengaluru, India*

Abstract*—Conventional paper-based examinations place a recurring administrative burden on educational institutions: question papers must be printed and distributed, scripts evaluated by hand, and marks compiled and re-checked before they can be released, a process that is slow and susceptible to clerical error. This paper documents the design, implementation, and empirical evaluation of a purpose-built Online Examination System developed on the MERN stack (MongoDB, Express.js, React.js, and Node.js) that automates this lifecycle end to end, from student registration through to instant, database-backed result generation. Two cooperating modules were engineered around this goal: a Student-facing module that governs registration, credential-based sign-in, timed objective-test delivery, and immediate score disclosure, and an Administrator module through which question banks, student records, and result archives are curated from a single dashboard. Rather than treating security as an add-on, the system enforces stateless authentication using signed JSON Web Tokens (JWT) and stores every credential as a salted bcrypt hash, with a RESTful Express.js API layer mediating all traffic between the React.js client and a MongoDB data store accessed through the Mongoose object-data-modeling library. The implementation was subjected to four testing passes, unit, integration, system, and user-acceptance testing, together with seven targeted security probes covering unauthorized route access, token expiry handling, and injection or cross-site-scripting attempts. Every one of the eight functional test cases, seven security test cases, and seven module-level checks that were executed returned a pass outcome, indicating that the completed system reliably automates objective-type assessment while preserving data integrity and restricting access to authorized users. Benchmarked against manual examination workflows and against general-purpose learning-management platforms such as Moodle and Canvas, the system that resulted from this work is deliberately narrower in scope but correspondingly lighter to deploy, configure, and extend for institutions whose principal requirement is dependable examination management rather than full course administration.*

Keywords*—Online Examination System, MERN Stack, JSON Web Token, bcrypt, Web Application Security, Automated Evaluation, MongoDB, Software Testing.*

Date of Submission: 25-08-2026 Date of acceptance: 05-09-2026

I. INTRODUCTION

Examinations remain the principal instrument by which educational institutions gauge learner progress, yet the machinery surrounding a traditional examination, drafting question papers, invigilating in person, marking scripts by hand, and reconciling marks into a gradebook, has changed comparatively little. Each of these steps consumes staff time in rough proportion to the number of candidates, and each introduces an opportunity for transcription error or delay between the sitting of an examination and the release of its outcome. As institutions digitize other areas of academic administration, from admissions to attendance, the case for an examination platform that is secure by default, centrally administrable, and capable of scoring itself becomes correspondingly stronger.

This paper documents such a platform: an Online Examination System engineered on the MERN stack, a pairing of MongoDB, Express.js, React.js, and Node.js that is widely used for building single-page web applications with a JSON-native data layer. The finished system automates the complete candidate journey, registration, authentication, timed attempt, and immediate scoring, while giving administrators a single point of control over student accounts, question banks, and historical results. Two design decisions were treated as non-negotiable from the outset rather than retrofitted later: authentication is stateless and token-based, using signed JSON Web Tokens rather than server-side sessions, and no password is ever written to the database in plaintext, since every credential is passed through bcrypt's adaptive hashing function before persistence.

Beyond describing the finished artefact, this paper reports how it was verified. Section V presents the outcomes of four testing passes and a battery of security-specific probes, and Section VII examines the boundaries of that verification honestly, since a project of this scope cannot claim the same evidentiary weight as a multi-institution field trial. The intent throughout is to give a reader enough architectural and empirical detail to judge, independently, whether the design choices made here are sound for a comparable deployment.

The remainder of the paper is organized as follows. Section II situates the system against existing examination platforms and identifies the gap it targets. Section III describes the proposed methodology, covering requirements, architecture, and data design. Section IV details the implementation: the technology stack, module boundaries, API surface, and security mechanisms. Section V reports the testing methodology and results. Section VI discusses those results and the system's limitations. Section VII examines threats to the validity of the evaluation, and Section VIII concludes with directions for future work.

II. LITERATURE REVIEW

The shift from paper-based to digital examination management has been underway for long enough that a reasonably clear pattern of trade-offs has emerged in the platforms institutions currently choose between. Early digitization efforts concentrated on removing paper from the process, typing question banks into a database and computing scores automatically, which cut down on physical handling but rarely paid equal attention to how those databases were secured or how credentials were stored. Later, browser-delivered platforms began treating remote accessibility as a first-class requirement, letting candidates sit an examination from any internet-connected device while administrators retained centralized oversight of scheduling and content, though the security posture of individual implementations still varies considerably across products.

At the more feature-rich end of the spectrum sit general-purpose Learning Management Systems such as Moodle and Canvas, which bundle quiz and examination tooling inside a much broader suite of course-management, content-hosting, and communication features. That breadth is a genuine asset for institutions running full online courses, but it is also a cost: configuring, patching, and administering a full LMS is disproportionate work for an institution or department that needs only to run periodic objective-type examinations. Purpose-built examination systems occupy the narrower niche this creates, trading course-management breadth for a smaller, more directly usable feature set focused on registration, timed delivery, automatic evaluation, and result reporting.

A. Comparative Analysis

Table I sets the proposed system alongside manual examination practice and the two categories of existing platform described above, comparing them on methodology, principal advantages, and the limitations each still carries. Manual examination remains the simplest option to reason about administratively but is the slowest to yield results and the most exposed to human error during marking. Moodle and Canvas support rich assessment and course management but carry configuration overhead that is difficult to justify for examination-only use. Generic web-based examination tools automate evaluation but, in the authors' review of publicly documented systems, frequently under-invest in administrative tooling and in the security of the authentication layer itself, a gap the proposed system was designed specifically to close.

TABLE I: COMPARATIVE ANALYSIS OF EXAMINATION SYSTEMS

System	Methodology	Advantages	Limitations
Manual Examination System	Paper-based examination and manual evaluation	Simple to conduct; no infrastructure dependency	Time-consuming; delayed results; susceptible to human error
Moodle	Learning Management System	Supports quizzes, assignments, course management	Configuration overhead disproportionate to examination-only use
Canvas LMS	Cloud-based LMS	User-friendly interface; integrated online assessments	Requires institutional licensing and continuous connectivity
Basic Online Examination Systems	Web-based examination platforms	Automatic evaluation; remote access	Limited administrative tooling; inconsistent security practice
Proposed System	MERN Stack with JWT authentication	Secure token-based login; automatic evaluation; centralized database; instant results	Scoped to objective-type educational examinations

B. Research Gap

Reading across the systems in Table I, a consistent pattern emerges: platforms tend to be either broad and heavyweight, in the manner of a full LMS, or narrow but under-secured, in the manner of many ad hoc examination portals that omit token-based authentication or store credentials without adequate hashing. Manual processes, meanwhile, remain slow regardless of how carefully they are administered, since their bottleneck is clerical rather than technical. The system presented in this paper targets the space between these extremes, combining JWT-based authentication, bcrypt-hashed credential storage, a centralized MongoDB data store, automated grading of objective-type questions, and instant result generation inside a single application that is small enough to deploy without the configuration overhead of a general-purpose LMS.

III. PROPOSED METHODOLOGY

Development followed six sequential stages: requirement analysis, feasibility assessment, system design, implementation, testing, and deployment preparation. The system itself follows a conventional three-tier web architecture, separating the presentation layer (React.js), the application layer (Node.js and Express.js exposing a REST API), and the data layer (MongoDB), a separation chosen specifically because it allows each tier to be developed, tested, and scaled independently of the other two.

A. Requirement Analysis and Feasibility

Requirements were gathered around two actors, students and administrators. Functional requirements span registration, authentication, question-bank management, examination delivery, automatic evaluation, and result reporting; the complete list is reproduced in Table form during implementation planning and is reflected directly in the module boundaries described in Section IV.B. Non-functional requirements centered on responsiveness under normal classroom-scale load, resistance to unauthorized access, reliable persistence of examination data, and compatibility across the major desktop browsers in current use. A feasibility review preceding implementation confirmed the project on three axes: technical feasibility, since React.js, Node.js, Express.js, and MongoDB are mature, well-documented, open-source technologies with no licensing barrier to adoption; economic feasibility, since no proprietary tooling was required at any stage of development; and operational feasibility, since both candidate-facing and administrator-facing interfaces were kept deliberately simple enough to require minimal onboarding.

B. System Architecture and Module Workflow

Fig. 1 depicts the three-tier arrangement described above. The React.js client communicates with the Express.js API exclusively over HTTPS using JSON payloads carried by Axios, and the API in turn communicates with MongoDB through Mongoose, which mediates every query and enforces schema-level validation before a document is written. JWT issuance and validation, together with bcrypt password hashing and general input validation, sit logically across the boundary between the presentation and data tiers as a cross-cutting concern applied uniformly to every protected route, rather than being scattered ad hoc through individual controllers.

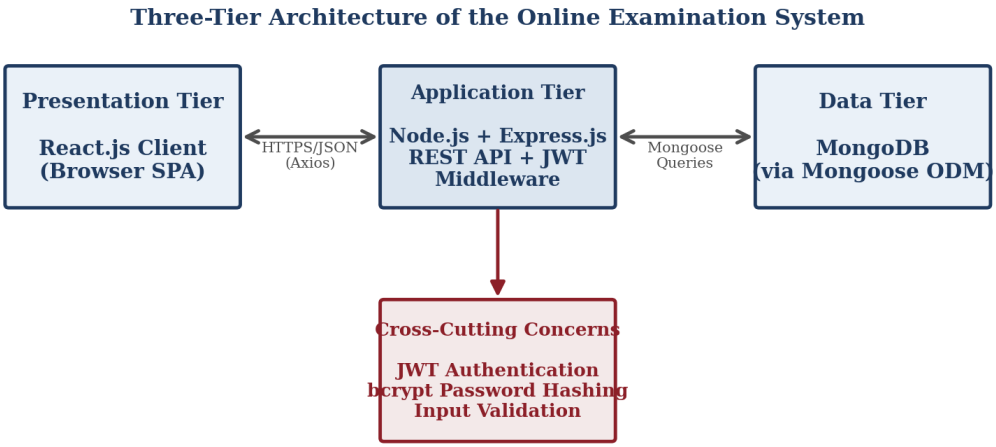

Fig. 1. Three-tier architecture of the proposed Online Examination System.

The administrator and student roles were kept as two decoupled workflows sharing only the authentication and data layers beneath them, which keeps the two areas of the application independently testable. Fig. 2 traces the administrator path: after authenticating, the administrator is routed to a dashboard from which existing student results and subject-wise question counts are retrieved on demand, and from which new questions are composed and posted to the database.

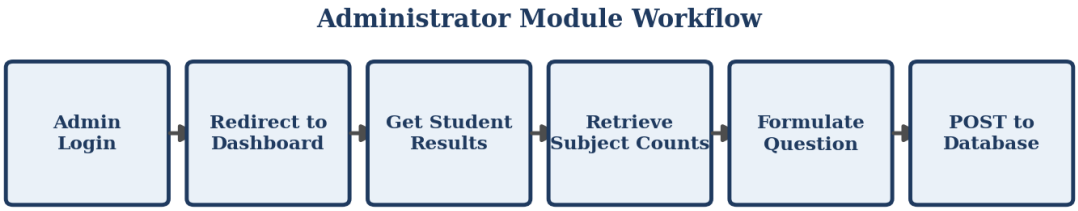

Fig. 2. Administrator module workflow.

Fig. 3 traces the corresponding student path: after authenticating, the student's assigned standard or special test is retrieved, a timer-governed question carousel is rendered, selected answers are captured client-side as the candidate progresses, and a single POST request submits the completed answer object for scoring, after which a score card is rendered immediately without further user action.

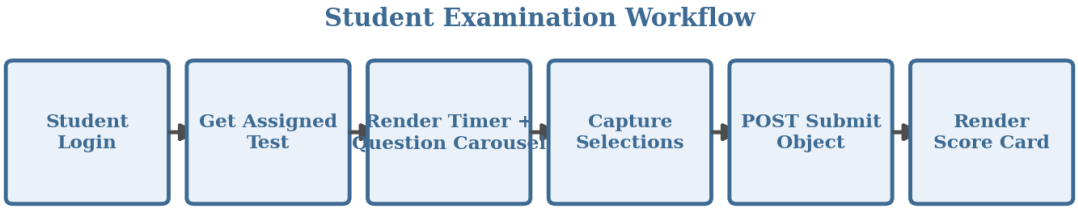

Fig. 3. Student examination workflow.

C. Non-Functional Design Considerations

Beyond the functional workflow, four non-functional properties shaped implementation decisions directly. Security was addressed through JWT-based route protection and bcrypt hashing, discussed further in Section IV.D. Responsiveness was addressed by keeping the API surface narrow and by letting MongoDB's document model avoid the join-heavy queries a relational schema would otherwise require for this data shape. Data integrity was addressed through Mongoose schema validation, which rejects malformed documents before they reach the database rather than after. Maintainability was addressed by keeping the five functional modules described in Section IV.B loosely coupled, so that, for instance, extending the Examination module to support descriptive answers would not require changes to the Authentication or Result modules.

D. Database Design

MongoDB was chosen over a relational alternative because its document-oriented model accommodates the naturally variable shape of question and result records, a multiple-choice question with four options and a fixed-duration timed test do not map cleanly onto a single rigid relational schema, without forcing an early, potentially brittle, normalization decision. Mongoose was layered on top of MongoDB as an Object Data Modeling library to define explicit schemas for five collections, Users, Questions, Results, SpecialTests, and SpecialResults, so that the flexibility of a NoSQL store is retained while still gaining schema-level field validation at write time. Fig. 4 summarizes the five collections and their principal fields, including the reference fields that link a Results or SpecialResults document back to the Users collection, and, for SpecialResults, to the corresponding SpecialTests document.

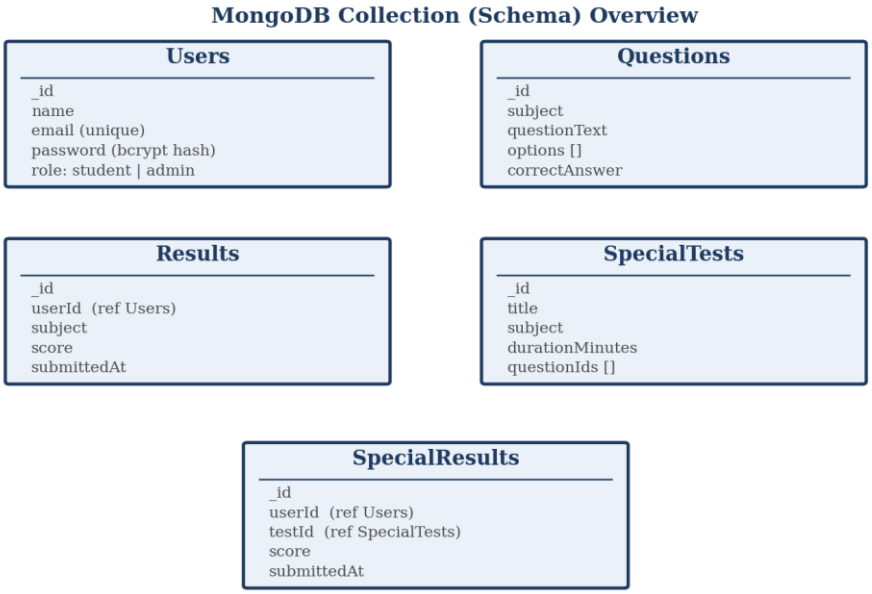

Fig. 4. MongoDB collection (schema) overview.

IV. SYSTEM IMPLEMENTATION

Implementation converted the design described in Section III into a running web application using exclusively open-source tooling: React.js [1] for the client interface, Node.js [2] and Express.js [3] for the server and its REST API, MongoDB [4] as the document database, and Mongoose [5] as the schema and query layer sitting above it. Visual Studio Code served as the primary editor and Git provided version control throughout development.

A. Development Environment and System Requirements

Table II and Table III list the software and hardware environment used to build and run the system. None of the listed software carries a licensing cost, and the hardware baseline is modest enough to run comfortably on typical student or staff laptop hardware rather than requiring dedicated server infrastructure for development and demonstration purposes.

TABLE II: SOFTWARE REQUIREMENTS

Software	Purpose
Windows 10/11 or Ubuntu	Operating system
Visual Studio Code	Code editor
Node.js	Backend runtime
Express.js	Backend framework
React.js	Frontend framework
MongoDB	Database
MongoDB Compass	Database management
Postman	API testing
Google Chrome	Web browser
Git	Version control

TABLE III: HARDWARE REQUIREMENTS

Hardware	Specification
Processor	Intel Core i3 or above
RAM	Minimum 4 GB (8 GB recommended)
Storage	20 GB free disk space
Internet	Required for package installation
Display	1366 x 768 resolution or higher

B. System Modules

The application is organized into seven cooperating units, corresponding to the seven components independently verified during module-level testing in Section V.D. The Student module governs registration, sign-in, examination attempts, answer submission, and result viewing. The Administrator module manages student accounts, question banks, and examination records. The Authentication module verifies credentials and issues and validates JWTs on every protected route. The Question Management module handles creation, retrieval, and modification of question-bank entries. The Examination module governs timed delivery, answer capture, automatic evaluation, and score computation for objective-type questions. The Result module persists computed scores and exposes a student's examination history on request. The Database module, MongoDB together with its Mongoose schema layer, underlies and is exercised by all six of the modules above. Fig. 5 shows these seven modules as equally weighted components of the tested system.

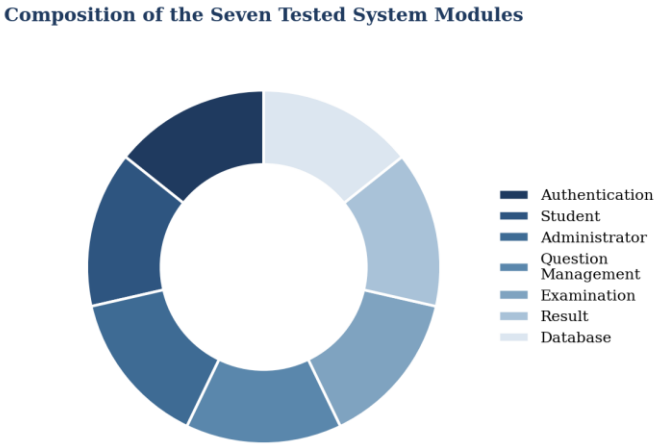

Fig. 5. Composition of the seven independently tested system modules.

C. API Implementation

The Express.js backend exposes a REST API consumed by the React.js frontend through Axios. Incoming requests pass through Express middleware that performs authentication checks on protected routes before any request reaches a controller, and controllers in turn delegate all persistence operations to Mongoose models rather than issuing raw MongoDB queries, which keeps validation logic in one place. Table IV summarizes the principal endpoints implemented, spanning authentication, examination delivery, and question-bank management.

TABLE IV: SUMMARY OF PRINCIPAL REST API ENDPOINTS

Method	Endpoint	Description
POST	/api/auth/register	Register a new student
POST	/api/auth/login	Authenticate user and issue JWT
GET	/api/auth/me	Retrieve logged-in user profile
GET	/api/exam/start/:userId/:subject	Start an examination
POST	/api/exam/submit	Submit examination answers
GET / POST	/api/questions	View / add examination questions
PUT / DELETE	/api/questions/:id	Update / delete a question
GET	/api/results	View examination results

D. Security Implementation

Security was designed in rather than layered on afterward. Authentication is stateless: a signed JWT is issued at login and re-validated on every subsequent protected request, which removes the need for server-side session storage and lets the API scale horizontally without a shared session store. No password is ever persisted in plaintext; each is passed through bcrypt's adaptive hashing algorithm, which incorporates a per-password salt, before being written to the Users collection. Every protected route rejects requests that lack a valid, unexpired token, and centralized input validation and error handling reduce the surface exposed to malformed or adversarial input, including basic injection and cross-site-scripting attempts, both of which were exercised directly in the security test cases reported in Section V.C.

V. TESTING AND RESULTS

A. Testing Methodology

Verification proceeded through four complementary passes. Unit testing exercised individual modules in isolation, principally authentication, question management, and result generation. Integration testing verified communication across the React.js frontend, the Express.js backend, and the MongoDB database, checking that data submitted at the client surfaced correctly at the persistence layer and back again. System testing exercised the assembled application under ordinary operating conditions end to end. User-acceptance testing checked that the resulting interface met the practical needs of both a student sitting an examination and an administrator managing one. Sections V.B through V.D report the outcomes of the functional, security, and module-level checks performed during this process.

B. Functional Test Results

Table V reports the eight functional test cases exercised against the system, covering the complete candidate and administrator journey from registration through to result viewing. All eight cases returned the expected outcome.

TABLE V: FUNCTIONAL TEST RESULTS

ID	Test Case	Expected Result	Status
TC-01	Student Registration	Student account created successfully	Pass
TC-02	Student Login	Valid user authenticated	Pass
TC-03	Administrator Login	Admin dashboard displayed	Pass
TC-04	Add Question	Question added to database	Pass
TC-05	Start Examination	Examination page rendered with timer	Pass
TC-06	Submit Examination	Score calculated correctly	Pass
TC-07	View Result	Result displayed to student	Pass
TC-08	Logout	User redirected to login page	Pass

C. Security Test Results

Table VI reports the seven security-focused test cases, targeting the authentication and access-control mechanisms described in Section IV.D directly. All seven cases returned the expected outcome, with unauthorized and malicious requests consistently rejected.

TABLE VI: SECURITY TEST RESULTS

Test Case	Expected Result	Status
Invalid Login Credentials	Access denied	Pass
Unauthorized API Access	Access blocked without valid token	Pass
Password Encryption	Password stored using bcrypt hash	Pass
JWT Authentication	Token validated on protected routes	Pass
Expired JWT Token	Session terminated	Pass
SQL / NoSQL Injection	Malicious request blocked	Pass
Cross-Site Scripting (XSS)	Input sanitized	Pass

D. Module-Level Test Results

Table VII reports module-level testing, in which each of the seven components identified in Section IV.B was exercised independently before assembled system testing began. All seven modules passed.

TABLE VII: MODULE-LEVEL TEST RESULTS

Module	Result
Authentication Module	Pass
Student Module	Pass
Administrator Module	Pass
Question Management Module	Pass
Examination Module	Pass
Result Module	Pass
Database Module	Pass

E. Result Analysis

Fig. 6 consolidates the three testing passes reported above into a single view. Across all three categories, functional, security, and module-level, every executed case returned a pass outcome, which is consistent with the system's authentication, examination delivery, automatic evaluation, and result-generation logic behaving as specified, and with unauthorized access being consistently prevented through JWT validation and bcrypt-based password encryption. Section VII discusses the scope within which this 100 percent pass rate should be interpreted.

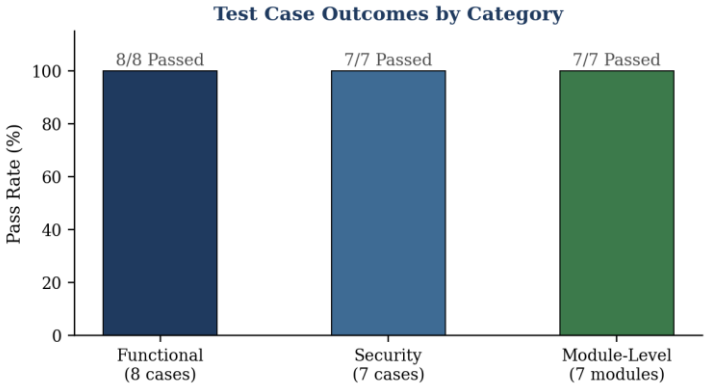

Fig. 6. Test case outcomes by category (functional, security, module-level).

VI. RESULTS AND DISCUSSION

The completed system demonstrates that a comparatively small MERN stack application can automate objective-type examinations end to end while maintaining a security posture appropriate for institutional use. Set against manual examination procedures, the system replaces paper handling and manual marking with instantaneous, database-backed scoring. Set against general-purpose LMS platforms, it trades feature breadth for a narrower surface that is faster to deploy and needs no course-management configuration an institution does not want, precisely the gap identified in Section II.B.

A. Limitations

Several limitations were identified honestly during development and testing. The system currently evaluates only objective-type, multiple-choice questions and cannot grade descriptive answers. An examination attempt requires continuous connectivity throughout, since no offline mode or client-side auto-save has yet been implemented, so a mid-examination connectivity loss can interrupt an attempt. No proctoring capability is present, and no email or SMS notification channel currently exists. Finally, the system was designed and tested for single-campus deployment and has not been evaluated under multi-campus or high-concurrency conditions.

VII. THREATS TO VALIDITY

The testing reported in Section V shows that the implemented functionality behaves as specified under the conditions in which it was exercised, but several factors bound how far that finding should be generalized. All test cases were executed by the development team on development-scale data rather than by an independent evaluator against production-scale enrolment figures, so the 100 percent pass rate reflects correctness against the authors' own test design rather than an external audit. Load and concurrency were not systematically varied; the functional and security test cases in Tables V and VI were exercised under normal, low-concurrency conditions, so claims about performance at examination-hall scale remain untested. The security test cases also target a specific, named set of threats, invalid credentials, missing or expired tokens, and basic injection and cross-site-scripting payloads, and a pass on these checks does not amount to a comprehensive penetration test. Readers deploying a similar system in a high-stakes examination setting should treat the results in Section V as evidence of a sound baseline rather than as a substitute for independent security review.

VIII. CONCLUSION AND FUTURE WORK

This paper presented the design, implementation, and evaluation of a secure Online Examination System built on the MERN stack. By combining JWT-based authentication, bcrypt password encryption, a centralized MongoDB data store, and automated evaluation of objective-type examinations, the system reduces manual administrative effort while giving students and administrators a responsive, centralized platform for online assessment. Functional, security, and module-level testing, reported in Section V and discussed critically in

Section VII, indicated that the implemented modules behave as specified and that unauthorized access is consistently prevented.

Future work will extend the system along several lines identified during development: AI-assisted proctoring, automatic saving of in-progress responses, facial-recognition-based candidate verification, native Android and iOS clients, email and SMS notification channels, cloud-based deployment beyond single-campus scale, descriptive-answer evaluation, multi-language support, and integration with existing Learning Management Systems. Independent, third-party security review and load testing at examination-hall scale are planned as prerequisites before any production deployment beyond a single institution.

REFERENCES

[1] React Team. React Documentation. Available: https://react.dev
[2] Node.js Foundation. Node.js Documentation. Available: https://nodejs.org/docs
[3] Express.js Team. Express.js Documentation. Available: https://expressjs.com
[4] MongoDB Inc. MongoDB Documentation. Available: https://www.mongodb.com/docs
[5] Mongoose Team. Mongoose Documentation. Available: https://mongoosejs.com/docs
[6] OpenJS Foundation. Node Package Manager (npm) Documentation. Available: https://docs.npmjs.com
[7] Flanagan, D. (2020). JavaScript: The Definitive Guide (7th ed.). O'Reilly Media.
[8] Chodorow, K. (2018). MongoDB: The Definitive Guide (3rd ed.). O'Reilly Media.
[9] Freeman, E. (2022). Head First JavaScript Programming (2nd ed.). O'Reilly Media.