International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

EduSubmit: A Web-Based Student Assignment and Submission Management Portal Using React, Spring Boot and MySQL

Dr. Nandhini A
Assistant Professor (SG), Department of Computer Application, Nehru College of Management, Coimbatore, Tamil Nadu, India
Email: ncmnandhini@nehrucolleges.com

Rajesh D
Student, II MCA, Nehru College of Management, Coimbatore, Tamil Nadu, India
Email: rajeshna2004@gmail.com

***Abstract*—Colleges still collect assignments through e-mail, messaging groups and paper, which causes missed deadlines, scattered marks and no single record of student progress. This paper presents EduSubmit, a role-based web portal that supports the complete assignment cycle (creation, submission, evaluation and feedback) together with a timed mock interview module for placement preparation. The system uses a three-tier architecture: a React 18 single-page frontend, a Spring Boot 3 REST backend secured with JSON Web Tokens, and a MySQL database of five tables. Faculty create assignments, evaluate each student's file with marks and feedback, and maintain an interview question bank. Students submit files before the deadline, track their status, and take five-question timed interviews whose answers are scored automatically by keyword coverage. A readiness score combines assignment marks and interview performance in one figure. A service layer lets the same frontend run against a browser-based mock API or the live backend without changing any page. Functional test cases covering deadline control, role-based access and mark validation are described.**

***Keywords*—Assignment management, mock interview, keyword-based scoring, React, Spring Boot, MySQL, JWT, role-based access.**

I. INTRODUCTION

Assignments are the main tool through which colleges measure continuous learning. In many institutions the whole cycle is still handled informally: faculty announce a task in a messaging group, students send files by e-mail or hand in paper copies, and marks are kept in separate spreadsheets. This creates three recurring problems. First, work arrives from many channels, so files are lost or are hard to match with the right student. Second, students have no single list showing what is due and when, which leads to missed deadlines. Third, marks and feedback are scattered, so a student cannot see his or her progress in one place.

At the same time, final-year students must prepare for campus placements, where technical and HR interviews decide selection. Practice for interviews normally happens outside the academic system, and faculty have no view of how well students are prepared. A portal that joins coursework and interview practice gives both groups a more complete picture.

This paper presents EduSubmit, a role-based web portal that manages the full assignment cycle and adds a timed mock interview module. The main contributions are: (i) a three-tier design using React, Spring Boot and MySQL with JWT-based security; (ii) a service-layer pattern that lets the same frontend work with a browser-based mock API or the real backend; (iii) a keyword-coverage method that scores interview answers automatically; and (iv) a readiness score that combines assignment marks and interview results.

The rest of the paper covers related work (Section II), objectives (III), architecture (IV), database (V), modules (VI), interview scoring (VII), implementation and security (VIII, IX) , testing (X), results and discussion (XI) and the conclusion (XII).

II. RELATED WORK

Learning management systems such as Moodle [1] and Google Classroom support assignment posting, file submission and grading for large institutions. They are general-purpose platforms with many features, and a college usually needs server set-up, course enrolment and administrator training before a single assignment can be collected. Their assignment tools are also separate from placement preparation.

Online interview-practice sites train students on technical questions, but their results stay with the platform, are not linked to the coursework record and are not visible to faculty. Many rely on multiple-choice questions, which do not show whether a student can explain a concept in his or her own words.

The REST architectural style [2] separates the user interface from the server logic, which makes each part easier to test, replace and scale. EduSubmit follows this style and keeps its scope deliberately small: one department, two roles and one complete workflow. As a result it can be installed on a single machine and used by students and faculty without special training.

Table I compares the usual ways of handling assignments with EduSubmit. The comparison is based on the standard features of each approach and not on a measured study.

TABLE I
COMPARISON OF ASSIGNMENT HANDLING APPROACHES

Feature	E-mail / paper	General LMS	EduSubmit
Central assignment list	No	Yes	Yes
Deadline control	No	Yes	Yes
Marks and feedback in one place	No	Yes	Yes
Timed mock interview	No	Not built in	Yes
Auto-scored answers	No	Not built in	Yes
Readiness score	No	Not built in	Yes
Set-up effort	None	High	Low

III. PROBLEM STATEMENT AND OBJECTIVES

The problem addressed is the absence of one portal where faculty can publish assignments and deadlines, students can submit work and follow its status, and both can see marks, feedback and interview readiness. The objectives of the system are:

1) to provide separate student and faculty areas with secure login and role-based access;
2) to let faculty create, edit and delete assignments with deadlines and maximum marks;

3) to let students upload a file before the deadline and track its status;
4) to let faculty view every student's submission, enter marks and feedback, and monitor performance;
5) to provide a timed mock interview with automatic scoring, a question bank managed by faculty, and a readiness score; and
6) to keep the frontend independent of the data source, so that it runs in demo mode or with the full backend.

IV. SYSTEM ARCHITECTURE

A. Overview

EduSubmit follows a three-tier architecture, shown in Fig. 1. The presentation tier is a single-page application built with React 18 [3] and bundled with Vite. The application tier is a Spring Boot 3 [4] REST service that contains the controllers, the security filter and the JPA repositories. The data tier is a MySQL 8 database [5] with five tables; uploaded files are kept in a protected folder on the server.

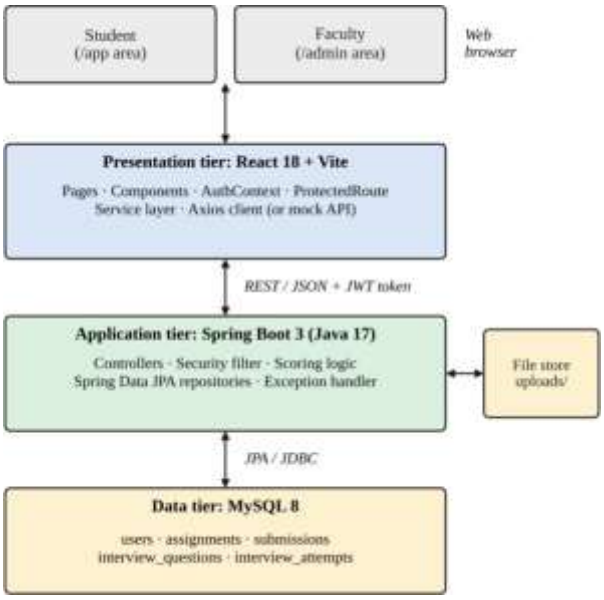

Fig. 1. Three-tier architecture of EduSubmit.

B. Communication and Authentication

The frontend and backend exchange JSON over HTTP. After a successful login the server returns a JSON Web Token (JWT) and the user's role. The Axios client attaches the token to every later request in the Authorization header. A request interceptor does this in one place, and a response interceptor returns the user to the login page if the token has expired.

C. Service Layer and Demo Mode

Pages never read data directly. They call functions in a services folder, such as assignmentService and submissionService. Every service sends its request through one http object. When the environment variable VITE_USE_MOCK is true, this object is a small in-browser mock server that answers the same URLs and returns the same JSON as the backend; when it is false, it is a real Axios client. The screens can therefore be demonstrated without a server and later connected to Spring Boot without changing any page, route or style.

D. Routing and Access Control

React Router divides the application into two areas: /app for students and /admin for faculty. A ProtectedRoute component checks that the user is logged in and that the role matches the area; a student who types a faculty address is redirected to the student dashboard. The same rule is enforced again on the server with @PreAuthorize annotations, so access control does not depend on the browser alone.

V. DATABASE DESIGN

The database was designed with the entity–relationship model [6] and has five tables, shown in Fig. 2. Students and faculty are stored in the same users table and are told apart by the role column (USER or ADMIN).

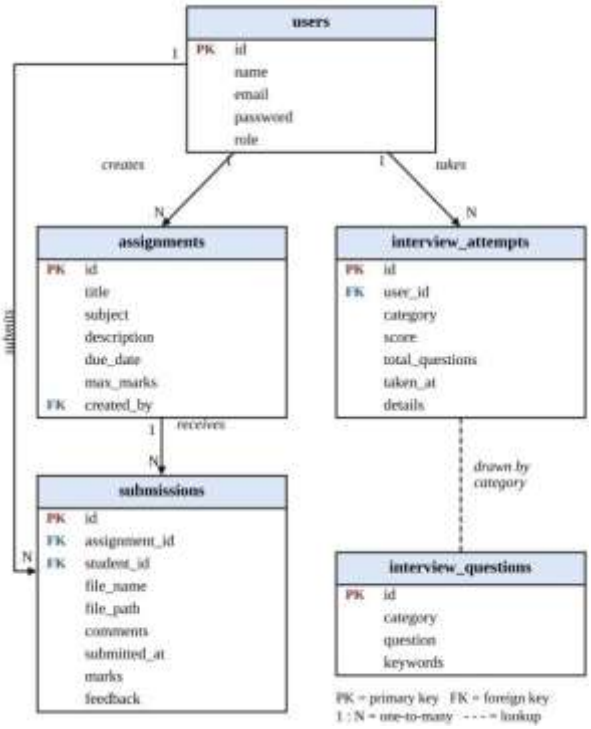

Fig. 2. Entity–relationship diagram of the EduSubmit database.

The assignments table stores the title, subject, description, due date, maximum marks and the faculty member who created it (created_by). The submissions table links one assignment to one student and stores the file name, the stored file path, the student's comments, the submission time, the marks and the feedback. A unique constraint on (assignment_id, student_id) allows one submission per student per assignment, and a null value in marks means the work has not yet been evaluated.

The interview_questions table holds the question bank, with a category and a comma-separated list of keywords for each question. The interview_attempts table stores the result of every mock interview: the user, topic, percentage score, number of questions, time taken, and a JSON copy of each answer with its score and missed keywords. This snapshot keeps past results unchanged when the question bank is edited later.

All relationships are one-to-many and are implemented as foreign keys. Deleting an assignment first removes its submissions, so no orphan rows remain. Passwords are never stored in plain text; only a BCrypt hash is saved.

VI. SYSTEM MODULES

The portal has two roles, each with its own set of pages. Fig. 3 shows the use cases of the student and the faculty member.

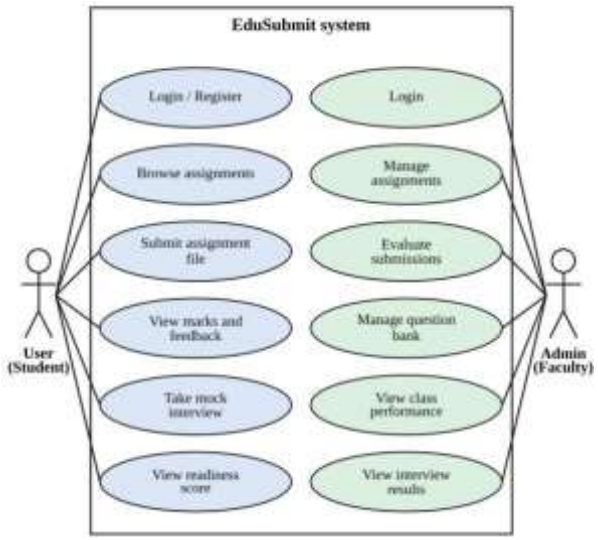

Fig. 3. Use case diagram of the student and faculty roles.

A. Student Module

After login the student opens a dashboard with four counters (total, pending, submitted and overdue assignments), a list of upcoming deadlines with the days remaining, the most recent submissions with marks, and the Mock Interview panel. The Assignments page shows cards and supports search by title and filtering by subject and status. Opening an assignment shows its description, its deadline and the student's own submission.

The submit form validates the file type and size (maximum 10 MB), accepts comments of up to 500 characters and shows an upload progress bar. After the deadline the page shows "Submission Closed" and the server refuses the upload. A student may replace a file until it has been evaluated, and the My Submissions page shows status, marks and feedback.

B. Faculty Module

The faculty dashboard shows total and active assignments, submissions received, pending evaluations and the interview average. Faculty create assignments with a title, subject, description, due date and maximum marks. Every field is validated before saving, and deleting an assignment asks for confirmation.

The Submissions page lists one row for every student, including those who have not submitted, with status, submission time and marks. The Evaluate page shows the student's file and comments with fields for marks and feedback; marks above the maximum are rejected. Once saved, the status changes to Evaluated and the student sees the result at once. The Performance page gives the class average, submission rate and interview average of every student.

C. Workflow

The core workflow is shown in Fig. 4. A user logs in, the server checks the credentials and issues a token, and the role decides which area opens. Faculty create an assignment, the student submits a file before the deadline, the faculty member evaluates it, and the marks become visible to the student.

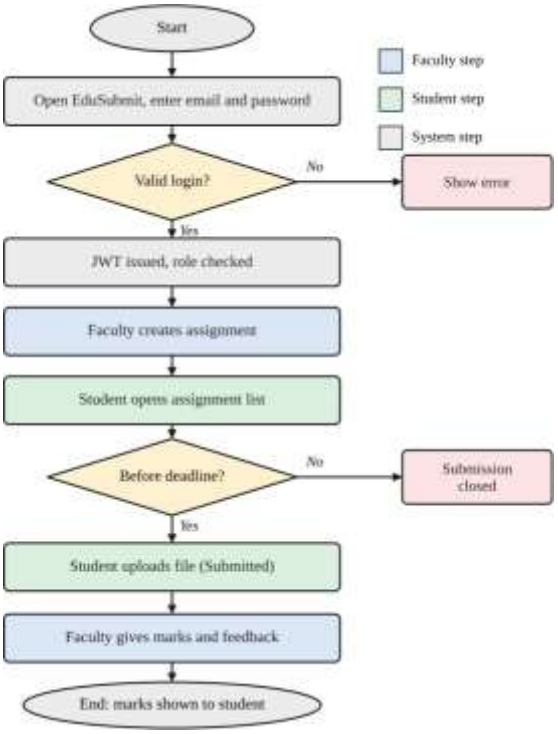

Fig. 4. Flow chart from login to evaluation of a submission.

VII. MOCK INTERVIEW AND READINESS SCORE

The mock interview module helps students practise explaining concepts under time pressure. The student chooses a topic (Java, React, SQL, Spring Boot or HR), and the server returns five random questions of that topic without their keywords, so the answers cannot be read from the browser. Each question has a two-minute countdown, and the next question opens automatically when the time ends. All answers are sent to the server in one request, which scores them and stores the attempt. Fig. 5 summarises the process.

Scoring uses keyword coverage. Faculty enter a list of key terms for every question in the question bank. For question i with k_i key terms, of which m_i appear in the answer (ignoring case), the score out of 10 is

$$S_i = \text{round}(10 \times m_i / k_i) \quad (1)$$

To discourage very short answers, an answer with fewer than eight words is capped:

$$S_i = \min(S_i, 4) \quad \text{if} \quad w_i < 8 \quad (2)$$

where w_i is the number of words in the answer. For n questions the interview percentage is

$$I = (\Sigma\, S_i / 10n) \times 100 \quad (3)$$



Fig. 5. Mock interview and scoring process.

The result page lists each answer, its score and the key terms that were missed, so the student knows what to revise. The dashboard shows a readiness score R, the average of the student's assignment performance A (marks divided by maximum marks over evaluated work, as a percentage) and the interview percentage I (averaged over all attempts):

$$R = (A + I) / 2 \quad (4)$$

If only one of the two values exists, that value is used. For example, a student with A = 82% and an interview average of 70% has R = 76%.

Keyword coverage is transparent and fast, needs no external service, and can be tuned by faculty by editing the keyword lists. Its limitation is that it measures the presence of terms and not the correctness of the explanation, as discussed in Section XI.

VIII. IMPLEMENTATION

A. Frontend

The frontend uses React 18 with Vite, React Router 6 for routing, Axios for HTTP and plain CSS for styling, so no UI framework is needed. Login state is kept in a React Context, and a small useFetch hook handles the loading, error and retry states of every data request. Each screen handles validation errors, loading, empty lists, request errors, confirmation of dangerous actions and success messages, so the user never sees a blank page. The layout is responsive: on a phone the sidebar becomes a slide-in menu and wide tables scroll sideways.

B. Backend

The backend is a Spring Boot 3 application on Java 17. Controllers expose the REST endpoints under /api, and Spring Data JPA maps the entities to MySQL; Hibernate creates the tables on the first start. A seeding class inserts demo users, assignments and the 30 interview questions when the tables are empty. A global exception handler returns every error as a JSON message that the frontend displays. Uploaded files are saved with a generated name and can be downloaded only by their owner or by faculty. Table II lists the technologies and Table III the main endpoints.

C. Deployment

The application runs in two modes. In demo mode only npm install and npm run dev are needed, with no backend or database. In full-stack mode, MySQL is started, its password is set in application.properties, the Spring Boot service is launched with Maven, and the frontend is switched to the real API by setting VITE_USE_MOCK to false. The tables and demo accounts are created automatically on the first run. The JWT secret, upload folder, file size limit and allowed frontend address are kept in the properties file.

TABLE II
TECHNOLOGY STACK OF EDUSUBMIT

Layer	Technology	Purpose
Frontend	React 18, Vite	Component-based user interface
Routing	React Router 6	Nested layouts, protected routes
HTTP client	Axios	REST calls with JWT interceptor
State	Context API, hooks	Login state, data loading
Backend	Spring Boot 3, Java 17	REST controllers, business logic
Security	Spring Security, JWT	Stateless login, role checks
Persistence	Spring Data JPA	Entity mapping and queries
Database	MySQL 8	Five relational tables

TABLE III
KEY REST API ENDPOINTS

Endpoint	Method	Access
/api/auth/login, /register	POST	Public
/api/assignments	GET	Any user
/api/assignments[/{id}]	POST, PUT, DELETE	Faculty
/api/submissions	POST	Student
/api/submissions/my	GET	Student
/api/submissions	GET	Faculty
/api/submissions/{id}/evaluation	PUT	Faculty
/api/interview/start	GET	Any user
/api/interview/attempts	POST	Student
/api/interview/questions	GET, POST, PUT, DELETE	Faculty

IX. SECURITY

Security is applied at four levels. First, passwords are hashed with BCrypt [8] and are never returned by the API. Second, authentication is stateless: the server signs a JWT [7] that holds the user's e-mail and role, and a filter validates the token on every request. Third, authorisation is enforced on the server with role annotations, and each student can read only his or her own submissions. Fourth, uploads are restricted to an allow-list of file types and a size limit of 10 MB, the original file name is cleaned, and the download path is checked so that it stays inside the upload folder. Cross-origin requests are accepted only from the configured frontend address.

X. TESTING

Functional (black-box) testing [9] was used to check the main workflows. Table IV lists representative test cases and the expected outcome of each. The cases were run as automated checks in the demo mode of the application, which uses the same URLs and rules as the backend, and the same cases are used to verify the full-stack version with Spring Boot and MySQL. A formal performance and load study has not been carried out.

TABLE IV
FUNCTIONAL TEST CASES

#	Test case	Expected result
1	Login with a wrong password	Error shown; no token issued
2	Student opens a faculty address	Redirected to student dashboard
3	Student calls a faculty API	Request refused (HTTP 403)
4	Upload after the deadline	"Submission closed"; file refused
5	Upload an .exe file	File type not allowed
6	Marks above the maximum	Marks rejected with message
7	Evaluate a submission	Status Evaluated; student sees marks
8	Interview answer under 8 words	Score capped at 4 out of 10
9	Delete an assignment	Confirmation; its submissions removed

XI. RESULTS AND DISCUSSION

A. Demonstration Data

The system was run with a demonstration dataset of five assignments, four students, seven submissions and four mock interview attempts. Table V shows the values that the dashboards compute from this dataset. For the demonstration student, two of the five assignments are submitted and one of them is evaluated at 42 out of 50 (84%). Two interview attempts (Java 70%, SQL 56%) give an interview average of 63%, so the readiness score is (84 + 63) / 2 = 73.5, shown as 74%. For faculty, four of the seven submissions wait for evaluation and the interview average of the class is 68%.

TABLE V
DASHBOARD VALUES FOR THE DEMONSTRATION DATA

View	Value	Result
Student	Total / pending / submitted / overdue	5 / 3 / 2 / 0
Student	Average assignment marks (A)	84%
Student	Interview average (I)	63%
Student	Readiness score (R)	74%
Faculty	Assignments: total / active	5 / 3
Faculty	Submissions / pending evaluation	7 / 4
Faculty	Class average marks	83%
Faculty	Class submission rate	35%
Faculty	Interview average of class	68%

B. Discussion

Compared with the manual process, the portal gives one list of tasks with deadlines, one place for files, and one record of marks and feedback. The deadline check is applied on the server, so late files are refused without manual checking, and the one-row-per-student view lets faculty see at a glance who has not submitted. The interview module gives students immediate and specific feedback in the form of missed key terms.

The keyword method is fast and transparent, but it can reward an answer that lists terms without explaining them. Faculty can reduce this by choosing keywords carefully and by reviewing the stored answers. Because the assignment percentage and the interview percentage are averaged with equal weight, the readiness score is a simple indicator and not a prediction of placement results. Studies with real classes are needed to measure usability and the effect of the portal on submission rates.

XII. CONCLUSION AND FUTURE WORK

This paper presented EduSubmit, a web portal that brings assignment management and mock interview practice into one system for students and faculty. The three-tier design, the service layer that separates the screens from the data source, and the role-based security make the system easy to demonstrate, maintain and extend. The keyword-based interview scoring and the readiness score give students quick feedback and give faculty a combined view of coursework and placement preparation.

The current version has limitations. Keyword scoring cannot judge whether an explanation is correct, files are stored on the disk of the application server, and no load testing has been done. Future work includes semantic answer scoring with natural language processing, e-mail or SMS reminders before deadlines, cloud file storage, plagiarism checks on submitted files, analytics for the head of the department, and a mobile application.

REFERENCES

[1] Moodle Pty Ltd, "Moodle documentation." [Online]. Available: https://docs.moodle.org
[2] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, USA, 2000.
[3] Meta Platforms, Inc., "React documentation." [Online]. Available: https://react.dev
[4] Spring Boot Reference Documentation. [Online]. Available: https://docs.spring.io/spring-boot/
[5] Oracle Corporation, *MySQL 8.0 Reference Manual*. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/
[6] P. P. Chen, "The entity-relationship model—Toward a unified view of data," *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, Mar. 1976.
[7] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF, RFC 7519, May 2015.
[8] N. Provos and D. Mazières, "A future-adaptable password scheme," in *Proc. USENIX Annu. Tech. Conf.*, Monterey, CA, USA, 1999, pp. 81–91.
[9] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. New York, NY, USA: McGraw-Hill, 2015.