Academic Notes Management System

Shaswat Sneh.*, Aashutosh Kumar Upadhyay., Anand Kumar Vishwakarma

Department of Computer Science & Engineering, Sarala Birla University, Jharkhand, India

***Corresponding Author**

DOI: https://doi.org/10.51244/IJRSI.2026.1304000181

Received: 18 April 2026; Accepted: 24 April 2026; Published: 13 May 2026

ABSTRACT

This paper presents the Academic Notes Management System (ANMS), a web application designed to address problems associated with the circulation and availability of notes in academic organizations. The system provides a comprehensive digital portal where authorized instructors can share notes in PDF format, while learners can search for notes based on course subjects and semesters. Role-based authorization is utilized in the application to ensure that only authorized instructors can upload and delete notes. JWT authentication is used in the Node.js and Express.js servers, whereas MongoDB is leveraged as the database management system. The frontend uses HTML5, CSS3, and vanilla JavaScript to develop an efficient, intuitive, and responsive user interface without the need for complex frontend frameworks.

Keywords: Note sharing, Academic management system, Role-based access control, Node.js, Express.js, MongoDB, JWT authentication, Full-stack web application.

INTRODUCTION

Advancements in digital technology have changed how knowledge is distributed and consumed significantly. In the vast majority of colleges and universities, students still use nonformal methods, such as messaging services, photocopied versions of books, or simply sharing files among themselves [1]. This method is chaotic, inefficient, and often leads to losing important notes, having different versions of those notes, and, generally speaking, lacks any quality control system whatsoever. Therefore, there is a need for a dedicated platform that can organize note sharing.

This platform is called the Academic Notes Management System (ANMS). It is a website designed for students and teachers of academic institutions, which allows users to manage study notes in one place. Access to uploading or deleting notes is limited to only registered instructors, meaning that all of the information stored on the platform is official and relevant to the course. After registration, students can access resources, search for certain notes by course name or semester, and download them in PDF format.

In terms of development, ANMS uses a modern web development stack, namely HTML5 and CSS3 for the frontend, together with JavaScript and Node.js & Express.js framework for the backend. MongoDB was used as the database because of its flexibility in working with document-oriented data [2]. The process of authorization relies on JWT authentication.

Overall, this project aims to optimize the time spent by students in acquiring notes from third-party websites and improve the quality and management of resources while giving teachers the opportunity to distribute notes without using messengers.

LITERATURE SURVEY

A significant body of research considers note sharing and managing digital learning resources within a classroom. The analysis of current technologies shows a set of achievements as well as shortcomings in existing solutions, which prompted the creation of a proposed solution.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue IV April 2026

Early versions of web-based note-sharing platforms featured simple uploading/downloading functionality with basic security features. There was no distinction made by these systems concerning the role of users, whether they were students or teachers. There was also no way to ensure the quality and authenticity of files being uploaded. Research on such platforms has demonstrated that lack of approval process leads to content duplication, incorrect information, and mistrust from users [3].

In the realm of Learning Management Systems like Google Classroom, Moodle, and Schoology, significant developments took place. The modern LMS offer a wide range of services such as assigning tasks, monitoring student performance, holding discussions, and managing resources. At the same time, high level of functionality, along with institutional licensing, makes LMS less practical for small colleges as well as individual project implementation. They are too complicated to be used as simple note-sharing resources that allow students to find and download PDFs on desired topics [4].

According to research on note management front-end implementations, a straightforward and clear design is positively correlated with higher engagement from users and more frequent visits. The key aspect that encourages users to visit a website often and even share their own notes is the convenience and usability of the interface [5]. Therefore, the ANMS was created using a simple HTML, CSS, and JavaScript code instead of a popular JavaScript framework.

Regarding security, studies of web application vulnerabilities emphasize that authentication, proper validation of input values, and limitation of file uploads are among some of the most important steps to take when building any web application. The least basic requirements to be met in any web application that accepts file uploads and stores personal data include bcrypt password hashing, JWT session management, and restrictions on file types [6].

As can be seen from the presented literature review, there is currently an absence of a lightweight full-stack notes management system that combines role-based access control, JWT authentication, MongoDB backend, and clean plain-JavaScript frontend. That is the gap that needs to be addressed.

METHODOLOGY

The ANMS design and development followed a process-oriented approach through iterations in well-defined stages such as requirement gathering, design, implementation, testing, and evaluation. These stages were done in sequence before moving to the next stage, in line with the incremental approach of the waterfall model.

Requirements Gathering

Functional and non-functional requirements were established by analyzing the normal interactions of note exchange among students and teachers within an academic institution. The main requirements include secure registration of users with roles (either student or teacher), authentication and authorization using JSON Web Tokens, uploading of notes by teachers with validation of PDFs, classification based on subjects and semesters, search and filter by students, downloads restricted to registered users only, and a teacher's panel to monitor and delete uploads.

System Design

The system was implemented using a client-server approach. The front end is connected to the back end through a RESTful API using the HTTP protocol. The back end is responsible for performing all business logic, authentication, file handling, and database-related tasks, whereas the front end manages the rendering of the UI, data input, and displaying the output from the API.

The database schema consists of two main tables: Users and Notes. In the Users table, the following details are saved: First Name, Last Name, Email, Hashed Password, and Role, along with some other role-based attributes like Department for Teachers and Semester for Students. The Notes table saves the following information: Title, Description, Subject, Semester, File Path, File Name, File Size, Timestamp of Upload, User ID of Teacher, and Downloads Counter.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue IV April 2026

Technology Stack Selection

For choosing technology stack, three main factors were considered, namely, task fit, industry relevance, and ease of integration. Node.js technology was used in the back end in order to use the JavaScript programming language for implementing tasks in both the front and back ends and eliminate any need for context switching. The Express.js framework was used to create the REST API because it is lightweight and flexible. The MongoDB database was chosen because it is based on the principle of working with documents and therefore naturally works well with JSON outputs from API and requires no strict schema [7]. JWT technology was chosen for authentication because of its stateless nature, which makes it unnecessary to keep sessions on the server side [8]. For password encryption, bcrypt was used as the established technique. The front end was implemented using pure HTML5, CSS3, and JavaScript.

System Architecture

The system was designed with a multi-layer web application model consisting of the presentation layer, the application layer, and the data layer.

Presentation Layer

The presentation layer contains five HTML pages which include the homepage (index.html), notes viewing page (notes.html), login page (login.html), registration page (register.html), and dashboard (dashboard.html). The style.css file contains the design system that is applied to all pages through CSS custom properties for color, typography, and spacing. Additionally, each page has unique styling applied through separate CSS files. The JavaScript functionality for handling API calls, form validation, managing local states, and manipulating the DOM is applied inline on each page.

The navigation bar changes based on the authentication status of the user. A common JavaScript module called auth-nav.js checks if there is a valid JWT token stored on localStorage, and if the user is authenticated, then the login and registration links are replaced by the dashboard and sign-out links. On the notes page, actual data is fetched from the backend API and rendered as dynamic note cards, whereas previously there was static placeholder content.

Application Layer

The application layer comprises a Node.js server that uses Express.js. It provides a REST API consisting of two main categories of routes. The authentication routes allow for registering (POST /api/auth/register), logging in (POST /api/auth/login), and getting a profile (GET /api/auth/me). The notes routes include the possibility of getting all notes (GET /api/notes), uploading a note (POST /api/notes), downloading a note (GET /api/notes/:id/download), and deleting a note (DELETE /api/notes/:id).

There are two middleware functions responsible for restricting access to certain routes. The first middleware function, called protect, checks the validity of the provided JWT token in the Authorization header and associates the user document with it to the current request. The second middleware function, called teacherOnly, examines the role field of the user and gives a 403 Forbidden response if the role of the user is not teacher. Both middleware functions are required to protect the upload and delete routes from unauthorized access, hence neither an anonymous user can upload notes nor students can manage their notes.

Multer serves as the file storage library that handles the process of uploading files. The Multer library utilizes a disk storage engine that stores the files into the uploads/notes folder of the server [9]. A file filter function excludes any uploads that have a MIME type other than application/pdf. Also, the maximum file size allowed by Multer is 20MB. Files are saved together with their filename, the new filename, and file size in bytes along with the note fields to the Notes database collection in MongoDB.

Data Layer

MongoDB database is utilized in the application. Mongoose is used for managing connections with the database, which is an Object Data Model (ODM). It allows one to define schemas, validate data, and query

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue IV April 2026

MongoDB. The Users model defines the required fields and also contains a role field with an enumerated type, which can only contain the values “student” and “teacher.” The Notes model includes the user ID of the individual who uploaded the notes using ObjectId, and the Mongoose virtual property is used to calculate the average rating from an array of ratings.

Modules Of The System

User Authentication Module

The authentication module takes care of the entire life cycle of a user. When a user signs up, the system asks for their name, email address, password, role, and role-specific information. Before being stored, the password is hashed with bcrypt and a salt factor of 12. A JWT is created and sent back to the client after they successfully register. When log in is initiated, bcrypt's compare function checks the password entered against the hash that is stored. A successful comparison makes a new JWT that lasts for seven days. The token is saved in the browser's localStorage and sent as a Bearer token in the Authorization header of all requests that are made after that.

Notes Management Module

This module gives the platform its main features. Teachers can use a dashboard with an upload form to enter the note's title, description, subject, semester, and attach a PDF file. Before being stored, the file is checked on both the client and server sides. The note shows up right away in the teacher's My Notes list after it is uploaded. There is a delete button that lets them get rid of it. When you delete something, it takes both the database record and the physical file off the server's storage directory.

Browse and Download Module

Anyone can look at notes on the notes page without having to log in. Notes are shown as cards that show the subject, semester, title, description, the name of the person who uploaded them, the date they were uploaded, the size of the file, and a star rating. When guests click on the locked download button, a modal pops up asking them to sign in. The download button is unlocked for authenticated users. It calls the download endpoint with their JWT. The server checks the token, adds one to the download counter in the database, and sends the file to the client as a binary attachment.

Search and Filter Module

There are three ways to filter on the notes page: a text search box, subject filter chips, and a semester dropdown. Filtering is done in real time on the client side using JavaScript on the rendered note cards. All three filters work at the same time, showing only cards that meet all of the active conditions. If there are no notes that match the current filter combination, an empty state message will show up

Implementation

Server.js starts the backend server. It uses the dotenv library to load environment variables, the connectDB function defined in config/db.js to set up the MongoDB connection, the auth and notes route handlers to register, and the HTTP server on the configured port. CORS is set up to let requests from the Live Server development origin go to other origins. This lets the HTML front end served by VS Code Live Server talk to the Node.js back end.

VS Code Live Server on port 5500 serves the frontend statically while the API runs on port 5000. If you deploy the app to production, the Express server would serve the frontend HTML files as static assets, so you wouldn't need CORS. At the top of each page's script block, there is a constant that defines the API base URL. This makes it easy to change when the site is deployed.

When the dashboard page loads, it makes an authenticated request to GET /api/auth/me right away. The user is sent to the login page if the request fails or there is no token in localStorage. A successful response fills the

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue IV April 2026

sidebar with the user's initials, full name, and role badge. Teachers can see more navigation options for uploading and managing notes. Students can only see the overview and browse options.

Table 1: Technology Stack Summary

Layer	Technology	Purpose
Frontend	HTML5, CSS3, JavaScript	User interface and interaction
Backend	Node.js, Express.js	REST API and business logic
Database	MongoDB, Mongoose	Persistent data storage
Authentication	JWT, bcrypt	Secure login and sessions
File Handling	Multer	PDF upload and storage
Environment	dotenv	Configuration management
Development	VS Code Live Server	Frontend development server

Security Considerations

Security was not an afterthought during the development of ANMS; it was a top priority. These steps were taken:

Password Security

bcrypt hashes all user passwords before they are saved in the database. There is never a time when plain text passwords are stored. The bcrypt algorithm has a configurable cost factor set to 12, which makes brute force attacks very expensive to run [10].

Token-Based Authentication

The server keeps a secret key that is used to sign JWT tokens. Tokens are only good for seven days, and every protected API request checks them. The server sends back a 401 Unauthorized response if a token is missing, has expired, or is not valid.

Role Enforcement

The teacherOnly middleware checks the role field of the user who is logged in every time they upload or delete something. A student can't call teacher-restricted endpoints even if they have a valid token.

File Validation

Multer is set up to reject any file that isn't an application/pdf MIME type. This stops potentially harmful scripts from being uploaded. To keep storage from running out, the maximum file size is 20 MB.

Protection of Environment Variables

The .env file that holds the MongoDB connection string, JWT secret, and server port is not included in version control. In source code, these values are never shown.

Ownership Check on Delete

The server checks that the authenticated teacher's user ID matches the uploadedBy field of the note document before deleting it. This stops one teacher from deleting notes that another teacher has uploaded.

RESULTS AND DISCUSSION

The implemented solution has successfully satisfied all functional requirements identified. User registrations and logins for the student and teacher accounts are functioning correctly. The dashboard navigation has

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue IV April 2026

provided teachers with appropriately displayed upload and management options and has limited student navigation to browsing and viewing only.

The upload function is working as designed, with the uploaded PDF document successfully saved to the server directory and the correct metadata stored in MongoDB. The uploaded document is available for immediate viewing within the Browse section on the dashboard without requiring an explicit refresh by the user. The delete function successfully deletes both the database record of the document and the physical file itself from storage, ensuring no orphaned documents exist in storage.

The download functionality requires users to be authenticated in order to view/download a document. Users who are unauthenticated will be redirected to the login page, while authenticated users will be able to download the document as a binary file. The number of times a document has been downloaded is accurately incremented within the database for each successful download.

The search/filter system is responsive from the client-side perspective with no perceivable latency associated with the levels of data typically associated with a college course repository. Searching/filtering by subject, semester and keywords simultaneously produces accurate results.

The user interface uses one stylesheet — which is created in accordance with design tokens — to maintain consistency across all screens. Dashboard sidebars are displayed on authenticated sessions only and when users log in with a teacher's role, appropriate navigation for their role is displayed on the dashboard's sidebar.

Table 2: Functional Requirements Test Summary

Feature	**Expected Behavior**	**Result**
Teacher Registration	Account set up as teacher	Pass
Student Registration	Account set up as student	Pass
Login	A valid token generated	Pass
Teacher Upload	PDF saved, metadata stored in DB	Pass
Student Upload Attempt	403 Forbidden response	Pass
Note Browse (Guest)	Cards visible, download locked	Pass
Note Download (Auth)	File streamed, counter incremented	Pass
Note Delete (Owner)	File and DB record removed	Pass
Note Delete (Non-Owner)	403 Forbidden returned	Pass
Search and Filter	Results filtered in real time	Pass

CONCLUSION

Through the Academic Notes Management System, real-world problems of students and teachers can be successfully solved through a well-structured and focused full stack application. The technology stack chosen for the solution was a modern stack made up of Lightweight full stack Development Frameworks (HTML, CSS, JS, Node.js, Express.js, MongoDB) allowing for reduced application complexity while still providing all the functionality necessary for the project (e.g., role-based access control, JWT authentication, PDF file management, responsive user interface).

The project provided hands-on experience of designing a REST API, Mongoose schema design modelling, JWT session token management, bcrypt secure password management, Multer multipart file handling, and Fetch API client/server communications. These technologies are standards used throughout the industry and are directly applicable to the work performed in professional web development.

Possible future improvements to the system would be an admin module that would allow the admin to approve or disapprove notes that have been uploaded by users before being visible to students, an integration between the system and cloud object storage (AWS S3 or Cloudinary), to facilitate system scalability by replacing local file storage, a rating and review system built into the system that would allow users to rate and provide feedback about the quality of each note uploaded into the system, an email verification process during

registration to ensure that users are actually who they say they are, and a progressive web application/mobile responsive version of the system. The current implementation of the system is a strong base for expanding upon these future enhancements.

ACKNOWLEDGEMENT

We would like to extend our deepest thanks to our project guide & all faculty of the Computer Science and Engineering Department, as well as other people who offered assistance, guidance and advice during the course of development and documentation of this project. Lastly, we want to acknowledge the institution for its support of the project by providing infrastructure and materials to enable its successful completion.

Ethical Approval

This study relies on technical analysis and software development. None of the authors conducted research involving human participants or animals.

Conflict Of Interest

The authors assert that they possess no conflicting financial or personal interests that may affect this work. There were no outside funds or grants used to write this manuscript.

REFERENCES

1. Pawar, R., Raut, Y., Ethape, V., & Raut, K. (2025). Online note sharing system. International Research Journal of Modernization in Engineering Technology and Science, 7(10), 3586–3588.
2. MongoDB, Inc. (2026). MongoDB Manual. https://www.mongodb.com/docs/
3. Bhagwat, R., Karale, A., Kolhal, C., Pachare, S., Kardile, R. (2026). Online notes sharing system. International Journal of Innovative Research in Technology, 12(8), 8521–8524.
4. Jha, S., & Kumar, R. (2023). A user-friendly front-end notes management system. International Advanced Research in Science, Communication and Technology, 3(13), 205–209.
5. Deng, P., Chen, B., & Wang, L. (2023). Predicting students' continued intention to use e-learning platform for college English study: The mediating effect of e-satisfaction and habit. Frontiers in Psychology, 14. doi.org
6. Skanda, C., Srivatsa, B., & Premananda, B. S. (2022). Secure hashing using Bcrypt for cryptographic applications. 2022 IEEE North Karnataka Subsection Flagship International Conference (NKCon), 1–5. doi.org
7. Vitale, V. -N., et al. (2022). A comparative study of MongoDB and document-based MySQL for big data application data management. 2022 IEEE International Conference on Big Data (Big Data), 340–345. doi.org
8. InfoTech Hub. (2022, June 1). Stateless authentication with JWT and Spring Security: A developer's handbook. https://www.infotechhub.org/Articles/Java/Spring/Stateless-Authentication-with-JWT-and-Spring-Security--A-Developer-s-Handbook.html
9. Express.js. (2026). Multer: Node.js middleware for handling multipart/form-data [Software]. GitHub. https://github.com/expressjs/multer
10. OWASP Foundation. (2025). OWASP Top 10 web application security risks. https://owasp.org/www-project-top-ten/