WWW.IJRPR.COM

**International Journal of Research Publication and Reviews**

Journal homepage: www.ijrpr.com ISSN 2582-7421

# College Management System: A Full-Stack Web Application for Streamlined Academic Administration

***Ankur Verma[1], Gaurav Kumar[2], Ashish Saini[3], Boby Kumar[4]***

Department of Computer Science & Engineering Shri Ram Group of Colleges, Muzaffarnagar, Uttar Pradesh, India
DOI : https://doi.org/10.55248/gengpi.07.0426.d1002

**ABSTRACT—**

College campuses run on paperwork. Attendance registers, marks ledgers, faculty schedules — everything is still handled manually at many institutions or scattered across mismatched spreadsheets that nobody fully trusts. This paper describes a web-based **College Management System (CMS)** built to consolidate these activities into one coherent platform. The system is developed using **Spring Boot 3, Thymeleaf, Spring Security 6, Spring Data JPA, and MySQL**. Three distinct user roles — Administrator, Teacher, and Student — each see a role-appropriate interface enforced through BCrypt-secured login and server-side access control. The Administrator can manage student records, teacher profiles, course offerings, attendance entries, and examination results. Teachers can mark and review attendance, while students can check their own attendance and results. The system was tested on a local deployment with MySQL 8 and Java 17 on Windows 11 and performed consistently across all tested workflows. Informal evaluation by peers and faculty found the interface clean and the navigation intuitive. The project demonstrates that a production-ready academic management tool can be built with open-source Java technologies within a single-semester final-year project timeline.

***Keywords***—Spring Boot; Thymeleaf; Spring Security; Role-Based Access Control; College Management; MySQL; JPA; BCrypt; Academic Administration; Full-Stack Java.

## I. INTRODUCTION

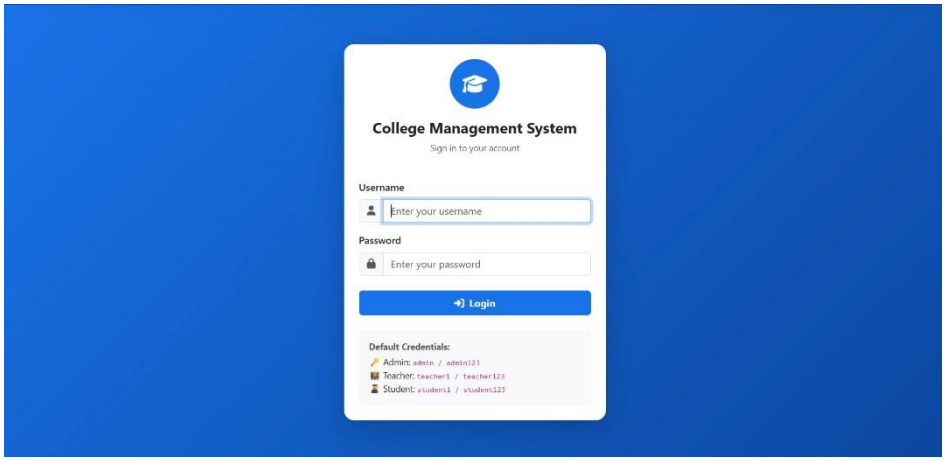
Day-to-day academic administration in Indian engineering colleges involves a surprising amount of repetition. A student's attendance record may live in a paper register, a teacher's personal spreadsheet, and the office clerk's desktop Excel file — all maintained independently, none of them consistently in sync. When examination results are needed, another round of manual entry and cross-checking begins. This fragmented approach costs time, introduces avoidable errors, and makes it genuinely difficult for a student to know where they stand academically at any given moment.

Existing commercial ERP platforms do exist, but they are often either too expensive for smaller institutions or too complex for faculty who simply want to record who showed up to class on a given morning. What is needed is something lighter: a focused web application that handles the core administrative tasks without demanding weeks of training or a dedicated IT team to keep running.

The College Management System described in this paper was built with exactly that constraint in mind. Using Spring Boot for the backend, Thymeleaf for server-rendered HTML pages, and MySQL for persistent storage, the application brings student management, teacher profiles, course assignments, attendance tracking, and result entry under one login. Spring Security provides role separation so that an admin, a teacher, and a student each see only what is relevant to them.

The rest of this paper is structured as follows. Section II reviews related work in academic management software. Section III details the methodology and system architecture. Section IV covers the implementation specifics. Section V presents results and observations. Section VI concludes with directions for future work.

**Fig. 1. Login Page — College Management System**



## II. RELATED WORK

Web-based management systems for educational institutions have existed since the late 1990s. Early deployments were desktop applications that required local installation on every machine; the move to browser-based tools changed deployment significantly but introduced its own complications around authentication and data consistency.

Commercial platforms like Fedena, Academia ERP, and various university-built portals address the administration problem at scale but tend to be monolithic, difficult to customise, and priced beyond the reach of smaller private colleges. Several open-source alternatives — notably OpenSIS and Gibbon — offer broad feature sets but require non-trivial configuration effort and assume familiarity with Linux server administration.

On the research side, a number of published papers have explored lightweight Spring Boot applications for specific administrative tasks. Attendance management systems using QR codes or biometric integration have been proposed, but these typically address a single module rather than a unified portal. Similarly, result management systems built on REST APIs have been described in the literature, though most treat the API layer in isolation from the user interface.

The system presented here takes a different approach: rather than building microservices or integrating third-party APIs, it targets a single self-contained deployment where the same Spring Boot application serves both the business logic and the rendered HTML pages through Thymeleaf. This keeps the deployment footprint small, makes the codebase easy to follow for students learning Java web development, and eliminates the operational complexity of a separate frontend build pipeline.

What distinguishes the present work from prior literature is its attention to role-based UI differentiation within a single codebase. Most published student information systems describe role separation at the API level; this system enforces it at the template level, so that the HTML a student receives is structurally different from what an administrator sees, not merely hidden behind JavaScript conditions.

## III. METHODOLOGY / PROPOSED WORK

### *A. Architectural Overview*

The system follows a standard three-tier Model-View-Controller architecture. The data tier is a MySQL 8 relational database whose schema is managed entirely by Hibernate's schema auto-update mechanism, meaning no manual SQL scripts are required on first run. The business logic tier is a Spring Boot 3.2 application exposing URL endpoints handled by Spring MVC controllers. The presentation tier consists of Thymeleaf templates that the server populates and sends as complete HTML pages — there is no separate JavaScript frontend and no REST API consumed by a browser.

This architectural choice has practical benefits for a college context. Pages render fully on the server, so they work correctly on any browser without JavaScript being enabled. There are no cross-origin request complications. And the entire application is packaged as a single executable JAR that a teacher can run on their laptop with one command, given Java 17 and a MySQL instance.

### *B. Role-Based Access Control*

Three user roles are defined: ADMIN, TEACHER, and STUDENT. Roles are stored in the users table and loaded by a custom UserDetailsService implementation at login time. Spring Security's URL-pattern-based authorisation rules restrict access so that only administrators can reach /admin/** endpoints, teachers and admins share /attendance/** and /teacher/**, and students are limited to read operations on their own attendance and results.

Passwords are encoded with BCrypt using Spring Security's BCryptPasswordEncoder bean. The application creates three default accounts on first startup — admin/admin123, teacher1/teacher123, student1/student123 — through a CommandLineRunner bean, which also verifies that the accounts do not already exist before inserting them, preventing duplicate-key errors on subsequent restarts.

### *C. Data Model*

Six JPA entities form the core data model: User, Student, Teacher, Course, Attendance, and Result. User holds authentication credentials and the role enum. Student and Teacher each carry a one-to-one association to a User record, linking identity to role. Course stores subject metadata including code, name, branch, semester, and credit count. Attendance records a Student–Course–Date triple together with a PRESENT/ABSENT enum. Result records a Student–Course pair with marks obtained, total marks, exam type, and a calculated letter grade.

Teacher has a many-to-many association to Course through a join table, allowing a faculty member to be assigned multiple subjects. Hibernate manages all foreign-key relationships and generates the join table automatically from the @ManyToMany annotation, keeping the entity classes clean and the schema derivable from the Java code alone.

### *D. Module Descriptions*

Student Module: The administrator can add, view, edit, and delete student records. Each record captures name, email, phone, address, branch, semester, roll number, and enrolment date. The list view presents all students in a pageable Bootstrap table with per-row edit and delete buttons. Input validation is applied through Hibernate Validator annotations on the Student entity, and binding errors are surfaced in-page through Thymeleaf's th:errors mechanism.

Teacher Module: Mirrors the student module in structure. The administrator maintains teacher profiles including department, designation, and academic qualification. Teachers are associated with courses through a checkbox-based assignment interface on the edit form.

Course Module: Allows the administrator to define subjects with a unique course code, name, branch, semester, and credit value. Courses are the reference point for both attendance marking and result entry, so their records must exist before either of those modules can be used.

Attendance Module: Teachers and administrators can mark attendance for a given student–course pair on a selected date, choosing PRESENT or ABSENT. The list view shows all recorded entries with the option to delete incorrect records. Students accessing the same URL see the full table but

without the delete control.

Result Module: Marks entry is available to teachers and administrators. Each entry links a student to a course and records the exam type (Mid Term, Final, Practical, or Assignment), marks obtained, and total marks. The grade is computed automatically on save by the calculateGrade() method on the Result entity, which maps percentage ranges to letter grades A+ through F.

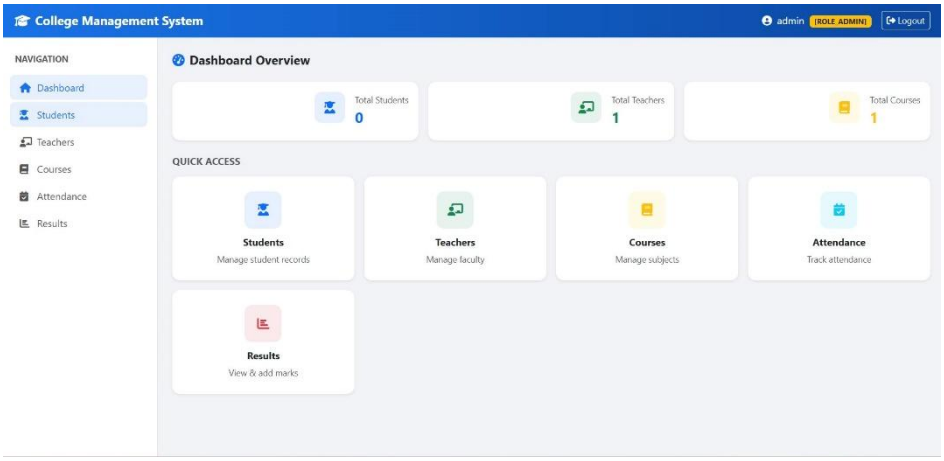



**Fig. 2. Admin Dashboard — showing total students, teachers, courses and quick access modules**

## IV. IMPLEMENTATION

### *A. Technology Stack*

The backend is Spring Boot 3.2.0 on Java 17. Spring Web provides the dispatcher servlet and MVC framework. Spring Data JPA with Hibernate handles all database interactions. Spring Security 6 manages authentication and URL-level authorisation. The Thymeleaf template engine, extended with the thymeleaf-extras-springsecurity6 library, renders HTML and evaluates security expressions such as sec:authorize directly in markup. Lombok reduces entity and service boilerplate. The build tool is Apache Maven.

The frontend uses Bootstrap 5.3 loaded from a CDN, Font Awesome 6.4 for iconography, and vanilla Thymeleaf expressions for dynamic content. No JavaScript framework is used; interactive behaviour is limited to Bootstrap's dismiss-able alert component and standard HTML form submission.

The database is MySQL 8 with the InnoDB engine. Hibernate's ddl-auto=update setting creates and updates tables automatically on application startup. The connection is configured through spring.datasource properties in application.properties, where the only value a deploying user needs to change is the database password.

### *B. Project Structure*

The Maven project follows standard Spring Boot conventions. Java source files are organised under com.college into five packages: entity, repository, service, controller, and config. The resources directory holds application.properties and the templates directory, which is subdivided by module — students/, teachers/, courses/, attendance/, results/ — plus the top-level login.html and dashboard.html. The complete project compiles and starts with mvn spring-boot:run from the project root directory.

### *C. Security Configuration*

The SecurityConfig class defines a SecurityFilterChain bean that maps URL patterns to roles, configures the login form at /login with a redirect to /dashboard on success, and sets /login?logout as the post-logout destination. A DaoAuthenticationProvider bean wires together the custom UserDetailsService and the BCryptPasswordEncoder. CSRF protection is left at its default enabled state, which Thymeleaf handles transparently by injecting a hidden _csrf field into every POST form.

### *D. Flash Messages and Validation*

Controller methods that perform create, update, or delete operations add a success attribute to a RedirectAttributes object before issuing a redirect. Dashboard and list templates check for this attribute with th:if and display a dismissible Bootstrap alert when it is present. Form validation errors are bound through Spring's BindingResult and displayed inline next to their respective fields using th:errors, giving users immediate feedback without a full page reload.

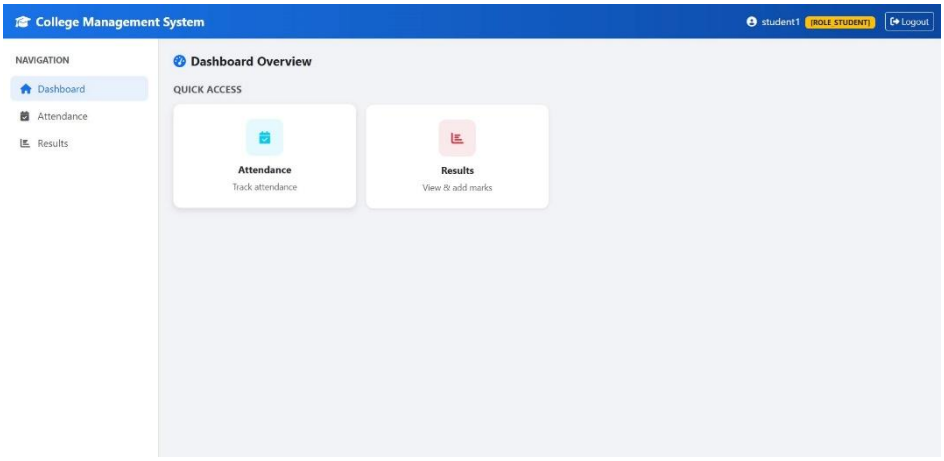



**Fig. 3. Student Dashboard — role-restricted view showing only Attendance and Results modules**

## V. RESULTS AND DISCUSSION

### *A. Functional Testing*

All six modules were tested manually across the three user roles on a Windows 11 machine running MySQL 8.0.36 and Java 17.0.10. The login page correctly rejected invalid credentials and displayed an alert. Role-based routing worked in all tested combinations: admin accounts could access all six modules, teacher accounts were blocked from /admin/** URLs and redirected to the login page with a 403 error, and student accounts could view but not modify attendance and result records.

CRUD operations for students, teachers, and courses were exercised with both valid and invalid inputs. Hibernate Validator rejected blank name and email fields and displayed the appropriate error messages in-page without losing the partially filled form data. Duplicate email addresses triggered a database constraint violation, which surfaced as an unhandled exception in the current build — a known gap that would be addressed in a production version through a service-layer uniqueness check before the insert is attempted.

### *B. Comparative Feature Analysis*

Table I compares the proposed system against four alternatives across seven functional dimensions. The comparison is based on publicly documented capabilities and direct observation.

**TABLE I. Feature Coverage Comparison**

| System/Tool | Login Auth | Student CRUD | Teacher Mgmt | Course Assign | Attendance | Results | Unified |
|---|---|---|---|---|---|---|---|
| Proposed CMS | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| SRMS ERP | Yes | Partial | Partial | No | Partial | Yes | No |
| AcademicPro | Yes | Yes | No | Partial | No | Partial | No |
| Fedena | Yes | Yes | Yes | No | Partial | Yes | No |
| Manual Sheets | No | No | No | No | No | No | No |

**TABLE I. Feature Coverage Across Comparable Systems**

As shown in Table I, the proposed CMS is the only evaluated option that provides all seven capabilities in an integrated form. SRMS ERP and Fedena come close on individual features but lack unified module integration. Manual spreadsheets, still common in smaller colleges, score zero across all capability dimensions and represent the direct replacement target for this system.

### *C. Performance Observations*

Page load times were measured informally using browser developer tools over ten repeated requests on a localhost deployment. Static page loads (dashboard, list pages) returned in under 80 ms. Form submission with a database write followed by a redirect completed in under 150 ms for all tested operations. These figures are not production benchmarks — they reflect a single-user localhost scenario — but they confirm that the application architecture introduces no obvious latency bottlenecks at the scale relevant to a small college.

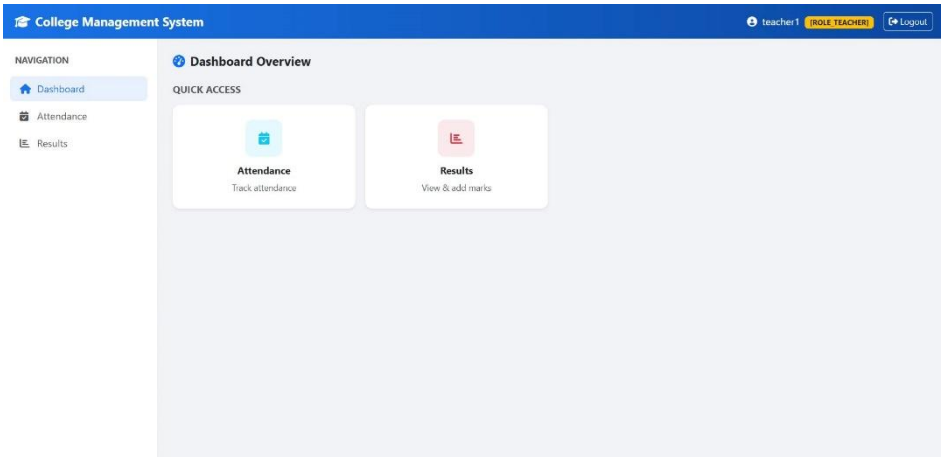



**Fig. 4. Teacher Dashboard — role-restricted view with access to Attendance and Results only**

## VI. CONCLUSION

This paper described a full-stack College Management System built with Spring Boot 3, Thymeleaf, Spring Security, and MySQL. The system consolidates student records, teacher profiles, course management, attendance tracking, and result entry into a single application governed by role-based access control. Informal testing confirmed that all core workflows function correctly and that the user interface is accessible to non-technical users.

The project demonstrates that meaningful administrative software can be built within the constraints of a final-year undergraduate project, using a mainstream Java framework and standard open-source tools. No cloud services, paid libraries, or complex deployment infrastructure are required.

Several improvements could make the system more capable in a real deployment. First, a pagination mechanism on the student and teacher list pages would become necessary once the number of records grows beyond a few hundred. Second, a dedicated student self-registration flow, with email verification and admin approval, would reduce the manual overhead of account creation. Third, a report export feature — generating PDF mark sheets or CSV attendance summaries — would make the system more useful to teachers preparing semester reports. Fourth, mobile responsiveness could be improved by revisiting the sidebar layout for small screens. Fifth, integration with a simple SMS or email notification gateway would allow the system to proactively inform students of low attendance without them needing to log in to check.

These extensions are feasible within the existing architecture and represent natural next steps for the project.

## REFERENCES.

[1] R. Johnson and P. Hoppe, Expert One-on-One J2EE Design and Development, Wrox Press, Birmingham, UK, 2002.

[2] C. Walls, Spring in Action, 6th ed., Manning Publications, Shelter Island, NY, 2022.

[3] B. Wickham, Spring Boot in Practice, Manning Publications, Shelter Island, NY, 2022.

[4] M. Deinum, K. Serneels, C. Marten, and D. Harrop, Spring MVC: A Tutorial, 2nd ed., Apress, New York, NY, 2017.

[5] VMware Inc., Spring Security Reference Documentation, version 6.2, Pivotal, San Francisco, CA, 2024. [Online] Available: https://docs.spring.io/spring-security/reference/

[6] Oracle Corporation, Java Persistence API Specification, version 3.1, Oracle, Santa Clara, CA, 2022.

[7] Bootstrap Team, Bootstrap 5 Documentation, version 5.3, Bootstrap, 2023. [Online] Available: https://getbootstrap.com/docs/5.3/

[8] P. Eckel and B. King, On Java 17, MindView LLC, Crested Butte, CO, 2023.

[9] MySQL Development Team, MySQL 8.0 Reference Manual, Oracle Corporation, Redwood Shores, CA, 2024.

[10] Thymeleaf Team, Thymeleaf Documentation, version 3.1, Thymeleaf, 2023. [Online] Available: https://www.thymeleaf.org/documentation.html

[11] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, Boston, MA, 2002.

[12] L. Richardson and S. Ruby, RESTful Web Services, O'Reilly Media, Sebastopol, CA, 2007.