Collego: An AI-Enabled ERP Platform for Academic Management

Prof. S. R. Karjol[1], Mr. Ramesh Karjol[2], Krishna Kulkarni[3], Shreyas Bewoor[4], Amith S. K[5]
[1]*Assistant Professor, Department of Computer Science and Engineering, B.V.V. Sangha's Basaveshwar Engineering College, Bagalkote*
[2]*Lecturer, Department of Mechanical, B.V.V. Sangha's Polytechnic (Autonomous) College, Bagalkote*
[3,4,5]*Students, Department of Computer Science and Engineering, B.V.V. Sangha's Basaveshwar Engineering College, Bagalkote*

***Abstract*—Colleges commonly run admissions, attendance, assessment, examinations, placements, and reporting through spreadsheets, paper registers, and messaging groups, which causes duplicate data entry, inconsistent records, and delayed reporting. This paper presents Collego, an AI-enabled ERP platform for academic management that unifies these workflows in role-based portals for students, faculty, and administrators.**

Collego combines a React single-page frontend, a layered Spring Boot REST backend, a PostgreSQL database, and a separate FastAPI microservice that exposes seven language-model features (chat, smart search, performance prediction, attendance-risk detection, resume analysis, placement assistance, and question-paper insights) through permission-scoped tools, output guarding, rate limiting, and deterministic fallback, so that core ERP functions never depend on the AI provider. The system is containerized and deployed on AWS (EC2, RDS, S3, ECR, CloudWatch) behind Nginx with HTTPS. A final verification run comprised 143 checks, all passing, including 88 automated backend tests. On a representative AI evaluation set, performance prediction achieved 84% direction accuracy, attendance risk achieved an F1 score of 0.84, and placement assistance achieved 92% eligibility-match accuracy. In the demonstration environment, routine tasks such as attendance entry, marks entry, and report generation required roughly 65–95% less effort than manual workflows. We also discuss limitations and a roadmap covering fees management, retrieval-augmented AI, and hardened CI/CD.

***Index Terms*—cloud deployment, college ERP, FastAPI, large language models, learning analytics, microservices, role-based access control, Spring Boot.**

I. INTRODUCTION

Higher education institutions manage a wide range of interconnected activities: admissions, attendance, internal assessment, examinations, placements, communication, and institutional reporting. In many colleges these activities are supported by spreadsheets, paper registers, shared drives, messaging groups, and isolated departmental applications rather than a unified system [1]. Records are therefore duplicated, inconsistent, and slow to report. Students receive late or incomplete information about attendance shortages, internal marks, examination schedules, and placement eligibility; faculty spend considerable time on repetitive data entry instead of teaching and mentoring; administrators lack a consolidated view of departments, courses, faculty workload, and outcomes; and the placement cell struggles to track companies, drives, applications, resumes, and interview stages in one place. Weak access control and informal password sharing add security and accountability risks.

Existing tools usually solve only one part of this problem. A standalone attendance tool does not manage marks, a placement portal does not maintain academic records, and a document-sharing drive offers neither role-based access nor auditability. Enterprise resource planning (ERP) systems integrate such functions in principle, but adoption in higher education is harder than in conventional enterprises because of complex academic structures, legacy systems, and diverse stakeholders. At the same time, learning analytics, machinelearning prediction, and generative AI promise timely, actionable insight from

institutional data, but they raise concerns about data privacy, ethics, equity, and over-reliance on automated judgments.

This paper presents Collego, a centralized, AI-enabled ERP platform for academic management that has been built, tested, and deployed on AWS. The main contributions are: an integrated, role-based ERP that covers academic, examination, placement, communication, and reporting workflows for students, faculty, and administrators in a single system of record; an architecture that isolates generative-AI features in a separate microservice with permission-scoped tools, output guarding, rate limiting, a circuit breaker, a daily call budget, and deterministic fallback, so that AI remains assistive rather than a dependency; a security design based on JWT authentication, role-based access control (RBAC), hashed passwords, hashed single-use password-reset tokens, and audit logging; and a containerized AWS deployment together with a multi-level evaluation of functionality, AI-feature quality, performance, and operational benefit.

The remainder of the paper is organized as follows. Section II reviews related work, Section III states the requirements, Section IV describes the methodology and system design, Section V covers implementation, Section VI reports evaluation results together with the implemented screens, Section VII discusses limitations and future work, and Section VIII concludes.

II. RELATED WORK

ERP in higher education. Abazi Chaushi et al. [1] review the information systems and ERP modules used by universities. They show that ERP platforms integrate academic, administrative, financial, and human-resource functions, but that complex academic structures, legacy systems, and diverse stakeholders make adoption difficult, so careful module selection, process mapping, integration, and change management are needed. This supports a modular design in which academic, examination, placement, and reporting functions are integrated yet separable, as in Collego.

Learning analytics and performance prediction.

Ifenthaler and Yau [3] systematically review learning analytics for study success and report that analytics can help identify students at risk of dropping out and support course completion, while rigorous large-scale evidence remains limited and institutions need clear variables, ethical guidelines, and intervention strategies. Albreiki et al. [4] review educational data mining literature and find that machine-learning techniques are widely used for at-risk and dropout prediction, but that data quality, model generalization, and limited deployment of remedial interventions remain challenges. These findings motivate treating Collego's prediction and attendance risk features as early-warning aids that are combined with faculty review and explanations rather than as definitive judgments.

Intelligent advising and AI in education. Assiri et al. [6] trace the move from traditional to intelligent electronic academic advising and identify personalization, privacy, user trust, and human oversight as key issues. Hwang et al. [5] describe AI as tutor, tutee, and tool, highlight ethics, privacy, equity, and teacher readiness, and argue for human-centered AI that augments educators. Collego follows these recommendations by using AI as a first line of support for routine queries, by scoping every AI tool to the authenticated user, and by leaving final academic and administrative decisions with faculty and administrators.

III. SYSTEM REQUIREMENTS

A. Functional Requirements

Authentication and access control. Users log in with email, university seat number (USN), or employee ID and are authenticated with JSON Web Tokens (JWT) [7]. RBAC is enforced on pages and APIs. A one-time password-reset link is sent by email, expires after a configured period, and is invalidated after one use. Administrators create, activate, deactivate, and manage accounts.

Student portal. Students view attendance, internal and semester marks, CGPA, timetables, examination schedules, hall tickets, revaluation details, assignments, study materials, question papers, and notices; they view eligible placement drives, apply to them, and upload resumes.

Faculty portal. Faculty view assigned sections, record attendance, enter internal and semester marks, upload assignments, materials, and question papers, publish notices, download section-wise student PDFs, view

upcoming placements, and access class-performance analytics.

Admin portal. Administrators manage departments, courses, semesters, sections, faculty-subject assignments, examinations, invigilation duties, holidays, timetables, companies, drives, eligibility criteria, applications, and interview stages. They approve question papers and faculty resumes, review audit logs, trigger database backups, and export student reports filtered by department and year.

AI and system services. The system provides AI chat, smart search, performance prediction, attendance-risk detection, resume analysis, placement assistance, and question-paper insights; restricts AI responses to the authenticated user's permitted data and degrades gracefully when the AI provider is unavailable; stores files in Amazon S3; sends notifications; and exposes health-check endpoints.

B. Non-Functional Requirements

The non-functional requirements are: security (HTTPS, JWT, BCrypt hashing, RBAC, hashed single-use reset tokens, input validation, restricted CORS, audit logging) in line with common web-application risks [8]; performance (normal API requests within two seconds under expected load, and AI requests that either return in reasonable time or fall back to a deterministic response); availability (restart policies, health checks, and graceful handling of AI-provider failure); scalability (a stateless, containerized backend with storage on RDS and S3 rather than local disk); usability (a responsive interface, consistent across the three portals, with clear validation messages); reliability (transactional operations and automated RDS backups); maintainability (a modular, layered architecture with automated tests); compatibility (current Chrome, Edge, Firefox, and Safari, and mobile browsers); and privacy and observability (least-privilege access, per-user AI data isolation, no plaintext passwords, and logs and health endpoints for monitoring).

IV. METHODOLOGY AND SYSTEM DESIGN

A. Development Approach and Feasibility

Collego followed an incremental, phase-wise approach [9], [10] over a planned 14-week schedule. Requirements were derived from common college workflows and the needs of students, faculty, administrators, and the placement cell, and the scope was divided into academic, examination, placement, communication, reporting, and AI-assisted modules. Each phase covered database changes, API implementation, frontend integration, validation, and documentation, so that every increment could be reviewed before the next began and contract mismatches between frontend, backend, database, and AI service surfaced early.

Authentication, master data, attendance, marks, and reports were built first; placements, question papers, and AI features were layered on top once the foundation was stable.

A feasibility study performed before implementation and revisited after the first deployment found the project technically feasible (a mature, widely supported stack and an alreadydeployed system), economically feasible (open-source software; recurring cost dominated by modest cloud compute, storage, and usage-based AI inference rather than licences or GPU hardware), operationally feasible (workflows mirror existing roles, and the portal is browser-based), schedulefeasible (working increments each phase), and legally and ethically acceptable (role-restricted access, hashed credentials, data minimization for AI, and audit trails).

B. System Architecture

Collego follows a layered, service-oriented architecture (Fig. 1). The presentation layer is a React [11] single-page application served by Nginx [12], which also terminates HTTPS and proxies API and AI requests.

The application layer is a Spring Boot [13] REST API organized into controllers, services, repositories, and DTOs, with JWT and RBAC enforced on every protected request. The intelligence layer is a separate FastAPI [14] service that calls a Groq-hosted language model and retrieves permitted data through scoped tools. The data layer consists of Amazon RDS for PostgreSQL [15] for transactional records and Amazon S3 for documents. External services are Groq, Gmail SMTP, and CloudWatch.

The layering gives failure isolation: an AI-provider outage does not block attendance, marks, or placements; a frontend redeploy does not restart the database; and a backend restart does not affect documents in S3.

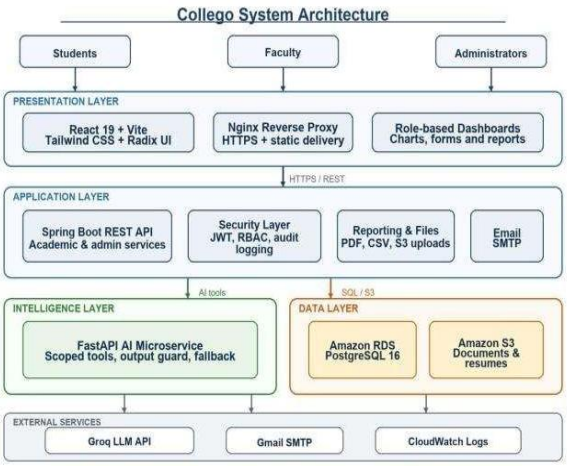

Fig. 1. Layered architecture of Collego.

It also allows the AI service to change models or providers without touching the academic modules. Security is applied at every layer, following the principle of least privilege: the frontend never stores plaintext passwords, the database is in a private subnet, S3 objects are reached only through the backend rather than public URLs, and sensitive actions are audit-logged.

C. Module Decomposition

The system is decomposed into twelve cohesive modules, each owning part of the domain, exposing REST endpoints, and enforcing RBAC: authentication and authorization, student portal, faculty portal, admin portal, academic management, examination management, placement management, question-paper repository, notification and document management, AI assistant and analytics, reporting and export, and audit, backup, and monitoring. Module boundaries follow college workflows, so a change such as new placement-eligibility rules does not ripple through the system. The faculty placement view, for example, deliberately shows only drive title, company, salary, drive date, department, and eligibility criteria, and only for active drives whose date has not passed.

D. Data Model and Governance

Collego does not rely on a public benchmark dataset. Its data are the operational records of a college, complemented by seeded demonstration data for testing and presentations. Structured data (users, profiles, departments, courses, sections, attendance, marks, examinations, placements, applications) lives in PostgreSQL; unstructured data (PDF and DOCX question papers and resumes) lives in S3 with ownership, access level, and academic metadata kept in the database; and semi-structured data (audit events, notification payloads, JSON exchanged between services) is handled at the API boundary. The volume is college scale (a few thousand students, a few hundred faculty, and tens of thousands of attendance and mark rows per semester) with seasonal peaks around attendance entry, result publication, and placement drives.

Data preparation begins at the API boundary: typed JSON payloads are mapped to DTOs and validated, the service layer applies rules such as section assignment, semester eligibility, duplicate detection, and placement eligibility, and the repository layer persists results transactionally. Uploaded files are validated for type and size before storage and are served only through backend endpoints. When several profiles share a USN, password-reset lookup deterministically prefers a deliverable non-college email, then an active account, then the newest profile.

E. AI Service Design

The AI capability is a stateless Python microservice exposing seven live features: chat, smart search, performance prediction, attendance risk, resume analysis, placement assistance, and question-paper insights. For each request it builds a narrow, permission-scoped context (for example, one student's attendance summary or one selected question paper) rather than receiving the whole database, calls the GROQ-hosted OpenAI/GPT-OSS-120b model, validates the output, and returns a structured response. Student records are not used to train any model. Safeguards include scoping tools to the authenticated user, output checks, rate limiting, a circuit breaker, a daily call budget, and deterministic fallback responses when the model is unavailable, so that the portal continues to work and the AI feature returns a safe message instead of breaking the page. Predictions and analyses are presented as assistance; faculty and administrators remain responsible for decisions.

F. Security Design

Authentication uses Spring Security and signed JWTs [7] that carry user identity and role; the backend validates the token on every protected request and applies RBAC at the controller and service levels. Passwords are stored as BCrypt hashes. For recovery,

the backend accepts an email address or USN/employee ID, creates a random 32-byte URL-safe token, stores only its SHA-256 hash, and emails a one-time link that expires after a configurable period (currently 30 minutes) and is cleared after one successful use; expired, reused, and unknown tokens are rejected, and no plaintext password is ever sent. Further controls are role-based menus, backend ownership checks (for example, a faculty member can record attendance only for an assigned section), private database access, controlled S3 downloads, audit logging, and HTTPS at the Nginx layer.

G. Cloud Deployment Architecture

The production system runs in the AWS Mumbai region (Fig. 2). A Virtual Private Cloud holds a public subnet for the application host and a private subnet for the database; an Internet Gateway and security groups control ingress. An EC2 t3. medium instance runs Docker [16] Compose with Nginx, the React build, the Spring Boot backend, and the FastAPI service. Container images are stored in Amazon ECR. The database is RDS for PostgreSQL 16.12 (db.t3. micro, 20 GB gp3, automated backups, encryption) in the private subnet, not publicly accessible. S3 stores documents and backups, and CloudWatch collects logs and health signals [17]. A stable HTTPS endpoint is provided through an Elastic IP and an sslip.io hostname. Terraform describes the infrastructure, and GitHub Actions provides build automation; the current workflow does not yet run the full backend test suite. Keeping the application containers stateless and the data tier managed lets containers be replaced without risking data.

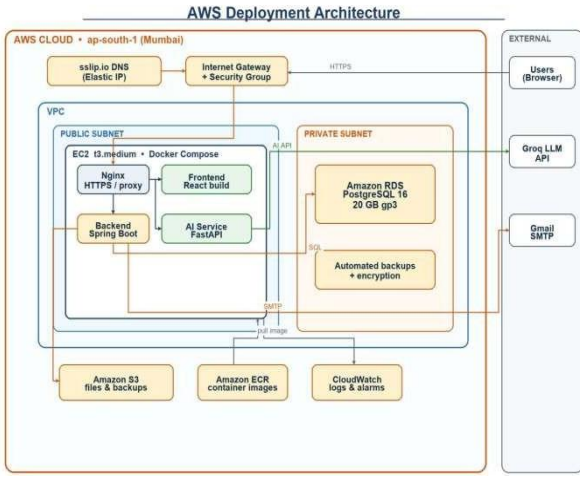

Fig. 2. AWS deployment architecture

V. IMPLEMENTATION

A. Technology Stack

Table I summarizes the stack, which was chosen for maturity, developer productivity, security, and cloud readiness. The system is deliberately polyglot but loosely coupled: Java handles transactional business logic, React the user interface, Python the AI orchestration, and PostgreSQL durable relational storage.

Table I: Technology Stack of Collego

Layer	Technologies
Frontend	React 19, Vite 8, Tailwind CSS 4, Radix UI, React Router 7, Axios, Recharts, Framer Motion (Node.js 20)
Backend	Java 17, Spring Boot 3.2.5, Spring Security, Spring Data JPA, Maven 3.9; JJWT, OpenPDF, PDFBox, Apache POI, AWS SDK
AI service	Python 3.11, FastAPI 0.111.0, Uvicorn, Pydantic, HTTPX, Boto3, NumPy; Groq API (openai/gpt-oss-120b)
Database	PostgreSQL 16.12 on Amazon RDS (db.t3. micro, 20 GB gp3)
Hosting	EC2 t3. medium (2 vCPU, 4 GB), Docker Compose, Nginx, ECR, S3, CloudWatch
DevOps	Terraform, GitHub Actions, Git/GitHub
Email	Gmail SMTP with TLS

B. Backend, Frontend, and AI Service

The backend contains 23 REST controllers, 30 services, approximately 258 Java files, and around 16,500 lines of code. Controllers validate parameters and delegate to services, which enforce ownership and role rules and coordinate transactions; repositories use Spring Data JPA; DTOs keep internal entities away from the frontend. Section-wise student PDFs, CSV exports, and Excel workbooks are generated with OpenPDF, PDFBox, and Apache POI as reusable services. Error handling is centralized and returns consistent status codes and structured messages. Transactions protect atomic operations such as attendance batch saves and placement applications.

The frontend is a React 19 single-page application built with Vite 8 and organized around role-based routes and reusable components. After login it renders the dashboard for the user's role. Axios attaches the

token and handles common errors, Radix UI provides accessible dialogs and menus, and Recharts renders attendance and performance charts. Forms validate before submission, and tables handle loading, empty, and error states. The implemented screens are shown individually in Section VI-E (Figs. 3–12). The production build is a static bundle served by Nginx.

The AI service uses FastAPI with Pydantic request and response schemas, HTTPX for the Groq API, Boto3 for S3 documents, and NumPy for lightweight numerical analysis. It is reachable only through the application's routing layer and can be redeployed without restarting the core backend.

C. Deployment and Release

Frontend, backend, and AI images are built for Linux/AMD64, pushed to ECR, pulled on the EC2 host, and started with Docker Compose; health endpoints then confirm that the frontend, backend, and AI service are available. The current release was a controlled manual deployment rather than an unattended CI/CD run, because the CI workflow rewrites the production environment file, does not yet preserve all SMTP and LLM settings, and skips the backend test suite. In production, the frontend returned HTTP 200, the backend health endpoint reported UP, the AI health endpoint reported UP with live serving enabled, and password-reset emails were delivered to real inboxes.

D. Implementation Challenges

Five issues required correction:

- The forgot-password panel was nested inside the credentials step and did not render; it was moved out of that conditional.
- Duplicate USNs produced ambiguous lookups and an HTTP 500 response; a deterministic preference order resolved this (Section IV-D).
- The production host is Linux/AMD64, so images must be built for that platform rather than the developer's architecture.
- Configuration drift appeared as an unused local-AI service in the Compose file and a CI workflow that did not preserve all environment settings; cleanup and making CI the single source of truth for environment variables are planned.
- Hibernate schema updates, convenient during development, are not a complete migration strategy; versioned migrations (Flyway or Liquibase) are recommended.

VI. EVALUATION AND RESULTS

A. Test Strategy and Environment

Testing covered five levels: unit, service, API, integration, and acceptance. Automated tests verified business rules, authentication, password reset, exports, and placement projections; manual and scripted checks verified user-interface behavior, role navigation, uploads, downloads, and end-to-end workflows; and production smoke tests confirmed live health after deployment. The test environment mirrored production (Java 17 and Spring Boot 3.2.5 on PostgreSQL, the production Vite build, Python 3.11 with FastAPI and Groq, and an EC2 t3. medium host with RDS, S3, ECR, CloudWatch, Nginx, and Docker Compose). Automated tests used seeded departments, courses, sections, students, faculty, attendance, marks, drives, question papers, and resumes; real institutional data was not used. Boundary cases included empty sections, duplicate USNs, inactive accounts, expired reset tokens, ineligible applications, large section PDFs, and AI-provider unavailability.

B. Functional and Security Results

Table II summarizes the final verification run. All 143 checks passed. The 88 backend tests form the automated suite; the remaining API, security, AI, and production checks are representative manual or scripted checks.

Table II: Summary of Final Verification Run

Test category	Run	Passed	Pass rate
Backend unit and service tests	88	88	100%
API and integration checks	24	24	100%
Frontend production build and lint	2	2	100%
Security and role checks	10	10	100%
AI feature checks	7	7	100%
Production smoke tests	12	12	100%
Total	143	143	100%

Twenty-five functional test cases covering authentication, password reset, student, faculty, and admin workflows, reporting, AI features, and

production health all passed. The most important were the role-boundary tests (for example, a student calling an admin endpoint is rejected with HTTP 403) and the large-section PDF and department/year export tests, which exercise large data sets and file generation.

C. AI Feature Evaluation

The AI features were evaluated on a representative curated set drawn from demonstration data and academic documents. The goal was not to benchmark a general-purpose model but to check that the service produces useful, grounded, and safe responses for its specific tasks. Metrics were relevance, groundedness (support by the supplied context), accuracy against expected outcomes recorded before each run, latency at the API boundary, and usefulness. Outputs were scored on a rubric of 1 (irrelevant or unsafe), 3 (partially correct, needs editing), or 5 (directly usable), and the same inputs were used for every run. A feature passed if it produced a usable answer, respected role boundaries, and returned a controlled fallback when the provider was unavailable.

Table III: Representative AI Feature Performance on The Evaluation Set

AI feature	Evaluation set	Result
AI chat	120 queries	Relevance 4.2/5; grounded 88%; 1.6 s
Smart search	60 queries	Top-3 relevance 90%; 0.9 s
Performance prediction	120 profiles	MAE 6.8 marks; direction accuracy 84%
Attendance risk	120 profiles	Precision 0.86; recall 0.82; F1 0.84
Resume analysis	45 resumes	Skill extraction 87%; usefulness 4.1/5
Placement assistant	45 students, 25 drives	Eligibility match 92%; usefulness 4.3/5
Question-paper insights	30 papers	Topic coverage 85%; usefulness 4.0/5

Table III shows representative results. The features are strongest when the task is tightly scoped and context is available. Smart search and placement assistance performed well because they operate on structured records, while resume analysis and question-paper insights were more sensitive to document quality (complex resume layouts need review, and scanned papers need OCR). Against baselines in the evaluation set, smart search raised top-3 relevance from 65% to 90%, and institution-scoped chat raised relevance from 3.4/5 for a generic LLM without institutional context to 4.2/5. Performance prediction and attendance risk are decision-support tools, not automatic graders, and are most valuable when a faculty member reviews the underlying records.

D. Operational Comparison and System Performance

Table IV compares Collego with the manual workflows it replaces; the values are representative of the demonstration environment. The largest gains occur in structured tasks with a clear correct answer, such as section PDFs, department/year exports, attendance and marks entry, and report generation, where automation removes repetitive formatting, filtering, and file-generation work. Password reset becomes self-service through a one-time email link, removing routine administrator intervention, and reports that took hours to compile can be generated on demand in minutes.

Table IV: Manual Workflows Versus Collego

Workflow	Manual	Collego	Effort saved
Attendance entry (section)	8–10 min	2–3 min	~70%
Internal marks (per subject)	15 min	5 min	~65%
Section student list	10–15 min	< 1 min	~90%
Dept./year student export	20–30 min	< 1 min	~95%
Placement notice preparation	20 min	Drive record and compact view	~75%
Question-paper search	10–15 min	Repository search	~90%
Resume screening	8–10 min	2–3 min	65–75%

Table V reports representative performance observed in the demonstration environment. Typical API reads

complete in 120–250 ms, well inside the two-second requirement, and the slowest operations are AI calls and document analysis, which are isolated from core workflows, so a slow AI response never blocks attendance, marks, or report generation. Export and PDF timings are acceptable at section and department scale and could be improved with asynchronous jobs if data volume grows.

Table V: Observed System Performance

Measure	Observed result
Frontend first load (broadband)	1.8–2.5 s
Simple API GET	120–250 ms
Attendance batch save (section)	300–600 ms
Section PDF (about 60 students)	1.0–1.5 s
CSV export (about 1,000 rows)	0.6–1.0 s
AI chat response	1.2–2.0 s
AI resume analysis	2.0–3.5 s
Production health	Frontend 200; backend UP; AI UP

E. Limitations of the Evaluation

Three limitations apply. First, the evaluation data set is smaller than a full institutional data set, so timings and accuracy values should be rechecked after a full academic year of use. Second, AI output quality depends on the completeness of records and documents; incomplete drive metadata or scanned question papers reduce quality. Third, the evaluation was done in a controlled deployment, so concurrent load, network variability, and provider rate limits must be monitored in production. The results therefore show fitness for the intended college use rather than absolute accuracy.

VII. LIMITATIONS AND FUTURE WORK

The highest-priority enhancement is a fees management module with programme-, semester-, category-, and quota-wise fee structures, a student ledger, receipts, concessions, and refunds, and online payment through a gateway using tokens and server-side confirmation so that raw card data is never stored, with every change audit-logged. Further work includes a parent portal and mobile or progressive-web access; retrievalaugmented generation over approved academic documents, multilingual support, explainable risk scores, and confidence indicators; multi-factor authentication, single sign-on, secret management, and formal data-retention and consent controls; a hardened CI/CD pipeline that preserves environment configuration and runs the full test suite, with versioned database migrations, automated RDS snapshots and restore drills, and a disaster-recovery plan; and integrations with payment, messaging, learning-management, library, and accounting systems, together with library, hostel, transport, and alumni modules.

VIII. CONCLUSION

This paper presented Collego, a role-based, AI-enabled ERP platform that brings student, faculty, and administrator workflows into a single application covering attendance, internal marks, examinations, placements, question papers, notifications, reports, and seven AI-assisted services. Its threetier architecture, with a React frontend, Spring Boot backend, PostgreSQL database, and a separate FastAPI AI service, is deployed on AWS and isolates AI failures from core academic operations. All 88 automated backend tests passed, and all 143 checks in the final verification run past.

The AI features performed best when tightly scoped and grounded in institutional data, and routine workflows required substantially less time and effort than their manual equivalents. Limitations include a smaller-than-institutional evaluation set, dependence on record and document quality, and the need to monitor production load and provider limits. With the planned fees, parent-portal, retrieval-augmented AI, security, and DevOps enhancements, Collego can evolve into a complete institution wide platform.

REFERENCES

[1] B. Abazi Chaushi, A. Chaushi, and F. Ismaili, “ERP systems in higher education institutions: Review of the information systems and ERP modules,” in *Proc. 41st Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, 2018, pp. 1487–1494, doi: 10.23919/MIPRO.2018.8400268.

[2] K. C. Laudon and J. P. Laudon, *Management Information Systems: Managing the Digital Firm*, 16th ed. Harlow, U.K.: Pearson, 2020.

[3] D. Ifenthaler and J. Y.-K. Yau, "Utilising learning analytics to support study success in higher education: A systematic review," *Educational Technology Research and Development*, vol. 68, no. 4, pp. 1961–1990, 2020, doi: 10.1007/s11423-020-09788-z.

[4] B. Albreiki, N. Zaki, and H. Alashwal, "A systematic literature review of student performance prediction using machine learning techniques," *Education Sciences*, vol. 11, no. 9, Art. no. 552, 2021, doi: 10.3390/educsci11090552.

[5] G.-J. Hwang, H. Xie, B. W. Wah, and D. Gašević, "Vision, challenges, roles and research issues of artificial intelligence in education," *Computers and Education: Artificial Intelligence*, vol. 1, Art. no. 100001, 2020, doi: 10.1016/j.caeai.2020.100001.

[6] A. Assiri, A. A. Al-Ghamdi, and H. Brdesee, "From traditional to intelligent academic advising: A systematic literature review of e-academic advising," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 4, pp. 507–516, 2020, doi: 10.14569/IJACSA.2020.0110467.

[7] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force, RFC 7519, May 2015, doi: 10.17487/RFC7519.

[8] OWASP Foundation, "OWASP Top 10: 2021," 2021. [Online]. Available: https://owasp.org/Top10/

[9] I. Sommerville, *Software Engineering*, 10th ed. Boston, MA, USA: Pearson, 2016.

[10] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. New York, NY, USA: McGraw-Hill Education, 2020.

[11] Meta Open Source, "React documentation." [Online]. Available: https://react.dev/

[12] NGINX, "NGINX documentation." [Online]. Available: https://nginx.org/en/docs/

[13] Spring, "Spring Boot reference documentation, version 3.2.5." [Online]. Available: https://docs.spring.io/spring-boot/docs/3.2.5/reference/html/

[14] S. Ramírez, "FastAPI documentation." [Online]. Available: https://fastapi.tiangolo.com/

[15] PostgreSQL Global Development Group, "PostgreSQL 16 documentation." [Online]. Available: https://www.postgresql.org/docs/16/

[16] Docker Inc., "Docker documentation." [Online]. Available: https://docs.docker.com/