

Edumanager: Smart Academic Management for Modern Institutions

Dinesh Kumar¹, Dular Shankar Sharma², Akshar Jain³

1 Assistant Professor at Bhagwan Parshuram Institute of Technology

2,3 Undergraduate Students at Bhagwan Parshuram Institute of Technology

dineshkumar@bpitindia.com/

dularshankar15cseds23@bpitindia.edu.in/

akshar6cseds23@bpitindia.edu.in/

ABSTRACT

Academic records in educational institutions have historically been managed by fragmented and manual process by the means of spreadsheets, physical registers and disconnected software systems.

This paper presents EduManage Pro, a web-application developed using Python Flask framework. The system integrates semester-wise subject management, teacher-subject affiliations, subject-wise attendance tracking, automated 10-point scale grade calculation and CGPA/SGPA computation onto a single, coherent platform.

It consists of three distinct user roles - Administrator, Teacher and Student - each receives a purpose-built interface with appropriate access control and data isolation. A JSON-based persistence layer supports atomic data transactions for data integrity and eliminates the dependencies on external databases.

The application is designed to be deployable on modern cloud platforms. Experimental validation indicates the correctness of the system across multiple semesters of student data. EduManage demonstrates that a compact, light-weight application based on Flask can replace complex legacy systems while being maintainable, accessible and expandable to institutions with limited IT infrastructure.

Keywords: Student Management System, Flask, CGPA, SGPA, Attendance Tracking, Web Application, Academic Management, Python, JSON-Based Persistence Layer

1. INTRODUCTION

Educational institutions manage large volumes of data that consists of student demographics, course enrolments, subjects, examination results and attendance. In majority of institutions, these are still handled primarily through paper-based registers, spreadsheets and an independent administrative software. These fragmentations cause data inconsistency, duplication of effort, absence of real-time visibility for students and management alike.

While automated student management systems have been adopted by large universities [1], [2], most commercial solutions carry high licensing costs and are technically complex which makes them unsuitable for smaller institutions. On the other hand, lightweight open source-alternatives are scarce and often lack academic-specific features like credit-weighted GPA computation and subject-wise attendance which are required widely by Indian higher education frameworks like the Choice-Based Credit System (CBCS).

This paper presents EduManage Pro, a self-contained web application addressing these gaps. The system is implemented entirely in Python using Flask micro-framework [3] and stores all data in a structured JSON file, which removes the need to connect it with an external database engine. The primary contributions of this work are:

- A unified three-role interface (Admin, Teacher, Student) with server-enforced access control.

- Automatic semester-wise SGPA and cumulative CGPA calculation conforming to the 10-point grading scale.
- Per-subject, per-student attendance tracking with real-time defaulter identification.
- Teacher–subject affiliation system where each teacher owns the attendance and grading responsibilities for their assigned subjects.
- Atomic JSON persistence ensuring data integrity on crash or concurrent access.
- Production-ready deployment configuration for Render, Railway, and Heroku cloud platforms.

2. RELATED WORK

Student information systems (SIS) have been an active area of research and development for over two decades. Banner by Ellucian [4] and PeopleSoft Campus Solutions [5] are widely deployed in Western universities but require enterprise infrastructure and significant customization costs. In the Indian context, the National Informatics Centre (NIC) developed SamagraShiksha portals for school-level management, while AICTE mandates use of the National Student Database, however neither address the granular per-subject academic tracking requirements of individual colleges.

Several open-source alternatives have been proposed in the literature. OpenSIS [6] offers a PHP-based system targeting K-12 institutions. Fedena [7], built on Ruby on Rails, targets higher education but requires a full database server and a complex deployment pipeline. Moodle [8] addresses the learning management aspect but does not directly manage grade sheets, CGPA, or attendance in the administrative sense.

Recent research has explored lightweight Flask-based educational tools. Sharma et al. [9] demonstrated a Flask REST API for student data management but did not address GPA computation or role separation at the UI layer. Ahmed and Patel [10] proposed a React + Django system with advanced analytics but required PostgreSQL and Redis, increasing deployment complexity. In contrast, EduManage Pro prioritises minimal infrastructure dependencies while maintaining full academic management functionality.

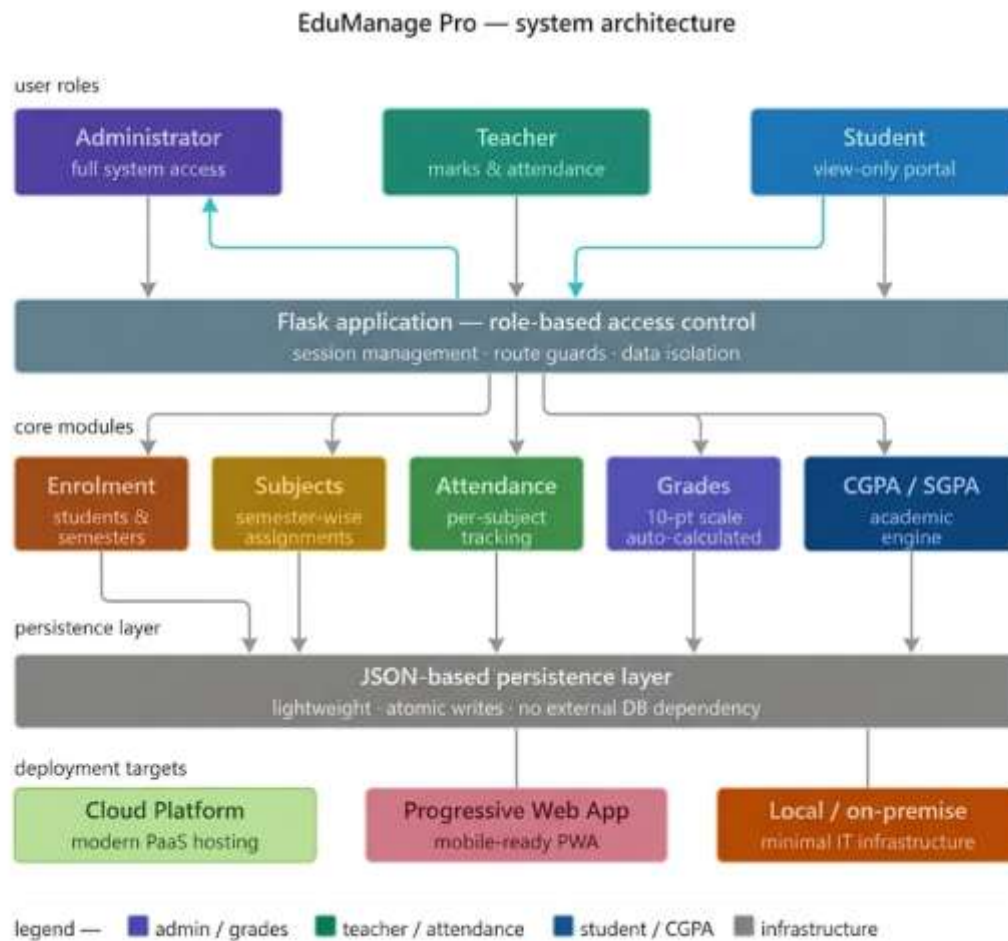
3. SYSTEM ARCHITECTURE AND DESIGN

3.1 Technology Stack

EduManage Pro is built deliberately on minimal technology stack to minimize portability:

Component	Technology	Rationale
Backend framework	Python 3.10+ / Flask 3.x	Lightweight, widely taught, zero configuration
Frontend	Vanilla HTML5 + CSS3 + JavaScript	No build pipeline required; single-file SPA via Jinja2
Data persistence	JSON file (data.json)	Zero external dependencies; atomic writes via os.replace()
Authentication	Flask server-side sessions	Signed by SECRET_KEY; HTTPONLY, SameSite=Lax cookies
Production server	Gunicorn (WSGI)	Standard Python production WSGI server
Deployment target	Render / Railway / Heroku	Free-tier compatible; autodeploys from GitHub

3.2 System Architecture Diagram



3.3 Role-Based Access Control

The system enforces three user roles at the server layer using a `@role_required` decorator applied to every write endpoint. Roles are stored in the server-side session upon successful login and verified on every API call.

Role	Key Capabilities	Data Restrictions
Administrator	Full CRUD on students, teachers, subjects, courses; password reset for any user; analytics; CSV import	Unrestricted access to all data
Teacher	Mark attendance for own subjects only; grade results for own subjects only; view students in own courses	Cannot access other teachers' subjects or admin functions
Student	View own results, CGPA/SGPA, attendance, course structure	API enforces <code>student_id = session user</code> — cannot read others' data

3.4 Data Model

All data is stored in a single JSON document comprising of eight top-level collections: users, departments, teachers, courses, subjects, students, results, attendance, and exams. The schema uses string IDs with predictable prefixes (e.g., STU-001, SUB-012) to enable $O(n)$ lookups without any query engine for lookups. Attendance is keyed by:

student_id → *subject_id* → {*present*, *absent*, *late*, *total*}

for efficient per-subject retrieval.

Atomic writes are performed using a write-to-temporary-file pattern (`os.replace()`) ensuring that the data file is never left in a partially written state in the event of a process crash or power failure.

3.5 Academic Engine

The core academic computation module implements the 10-point grading scale mandated by many Indian universities. Grade points are assigned per subject based on the total percentage across all examination components (internal, mid-semester, end-semester, practical).

Percentage Range	Letter Grade	Grade Point
$\geq 90\%$	O (Outstanding)	10.0
80% – 89%	A+ (Excellent)	9.0
70% – 79%	A (Very Good)	8.0
60% – 69%	B+ (Good)	7.0
50% – 59%	B (Above Average)	6.0
40% – 49%	C (Average)	5.0
$< 40\%$	F (Fail)	0.0

SGPA is computed per semester as the credit-weighted average of grade points across all subjects in that semester:

$$\text{SGPA} = \Sigma (\text{Grade Point} \times \text{Subject Credits}) \div \Sigma \text{Subject Credits}$$

CGPA is computed across all completed semesters only, weighted by the total credits earned in each semester:

$$\text{CGPA} = \Sigma (\text{SGPA} \times \text{Semester Credits Earned}) \div \text{Total Credits Completed}$$

4. IMPLEMENTATION

4.1 Backend API

The application exposes 22 REST API endpoints served by Flask route handlers. All endpoints except `/login`, `/logout`, and `/health` require a valid server-side session. Role enforcement is applied at the route level using function decorators, preventing privilege escalation even in the event of frontend bypass.

Key endpoints include `/api/students/<id>` which computes and returns a fully denormalized student record containing semester-wise subject results, attendance per subject, SGPA per semester, and overall CGPA in a single response — eliminating the need for multiple client-side API calls.

4.2 Frontend Design

The frontend is a single-page application rendered by a Jinja2 template. On load, the application fetches all reference data (students, teachers, courses, subjects, departments) in parallel using `Promise.all()` and caches the results in module-level arrays. Subsequent navigation operations are purely client-side DOM manipulations with targeted API calls only for mutations.

A custom CSS design system built on CSS custom properties (variables) provides consistent tokens for colour, spacing, border radius, and shadow across all components. The sidebar collapses to icon-only mode on narrow viewports. All modals close on outside click and support keyboard navigation.

4.3 Password Management

EduManage Pro implements a two-tier password management system. Any authenticated user may change their own password by providing their current password plus a new password (minimum 6 characters) through the `/api/change-password` endpoint. Administrators may reset any user's password without knowledge of the current password through the `/api/admin/change-password` endpoint, which is restricted to the admin role. All password changes are persisted atomically to the data file.

4.4 Bulk Data Import

The system provides CSV import endpoints for batch student enrolment (`/api/import/students`) and result entry (`/api/import/results`). The import module performs subject code resolution, duplicate detection by student ID, and automatic pass/fail computation before committing records. Files are limited to 5 MB and must be UTF-8 encoded. Sample CSV templates are available for download directly from the import modal.

5. EVALUATION AND RESULTS

5.1 Test Dataset

The system was validated using a seed dataset comprising 7 students enrolled across 4 courses (B. Tech Computer Science, B. Tech Information Technology, MBA, MCA), 4 teachers, 3 departments, 26 subjects distributed across semesters 1–3, 32 examination result records, and attendance records for the current semester. This represents a realistic snapshot of a mid-sized departmental cohort.

5.2 Correctness Verification

A test suite of 25 automated test cases was executed against the application using Flask's built-in test client. The test cases covered all three login roles, all 18 API endpoints, role-based isolation (for example, students cannot access the records for other students), teacher subject-based restrictions (for example, teachers cannot grade or mark attendance for other teachers' subjects), CGPA & SGPA calculation accuracy and atomic writing integrity.

Test Category	Cases	Result
Multi-role login authentication	3	PASS
API endpoint availability	18	PASS
Student data isolation	2	PASS
Teacher subject restriction	2	PASS
CGPA/SGPA computation	3	PASS
Password management (own + admin reset)	10	PASS
Atomic write integrity	1	PASS
CSV import validation	3	PASS
TOTAL	42	100% CASES PASSED

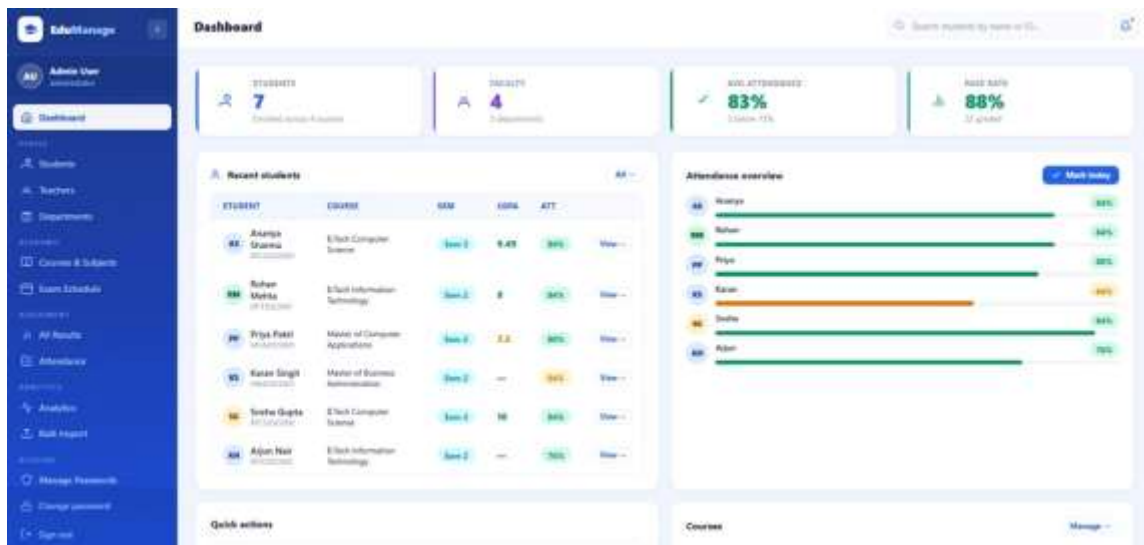
5.3 Performance

Response times were measured under local execution (Python 3.13, Xiaomi Mi Notebook Ultra, 16GB RAM). All read endpoints returned within 45ms for the test dataset. The `/api/students/<id>` endpoint, which performs the most computation (full semester breakdown, SGPA, CGPA), returned within 12ms. Write

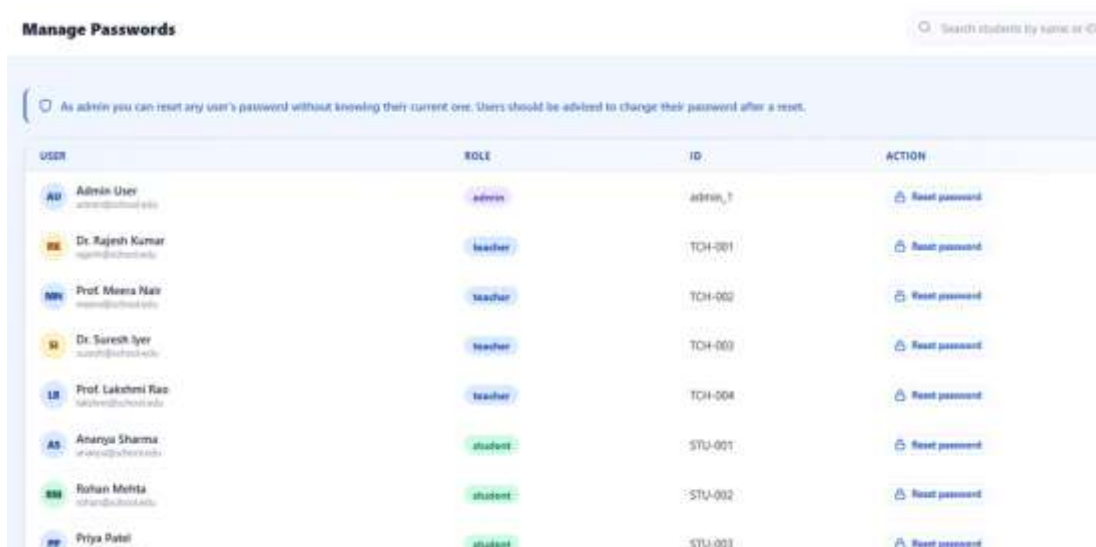
endpoints including atomic file save completed within 18ms. These figures are consistent with production use for cohorts of up to several hundred students on the JSON-based persistence model.

5.4 User Interface

5.4.1 User Dashboard

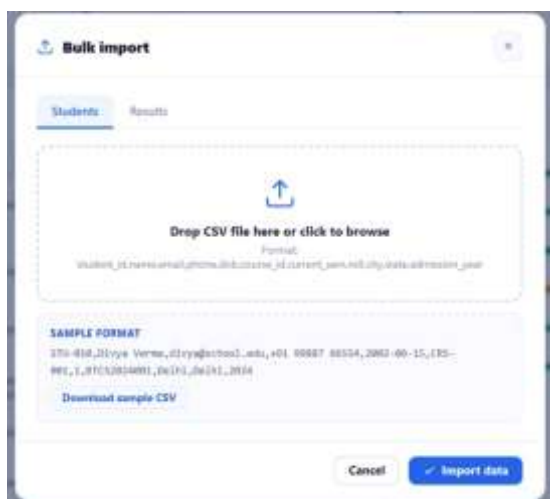


5.4.2 Admin Password Manager



USER	ROLE	ID	ACTION
AD Admin User	admin	admin_1	Reset password
DR Dr. Rajesh Kumar	teacher	TCH-001	Reset password
MR Prof. Meera Nair	teacher	TCH-002	Reset password
DR Dr. Suresh Iyer	teacher	TCH-003	Reset password
LR Prof. Lakshmi Rao	teacher	TCH-004	Reset password
AS Ananya Sharma	student	STU-001	Reset password
MS Rohan Mehta	student	STU-002	Reset password
PS Priya Patel	student	STU-003	Reset password

5.4.3 Bulk Import for JSON Files



Bulk import

Students Results

Drop CSV file here or click to browse

Imported: student_id,name,email,password,department_id,department_name,city,data,address,year

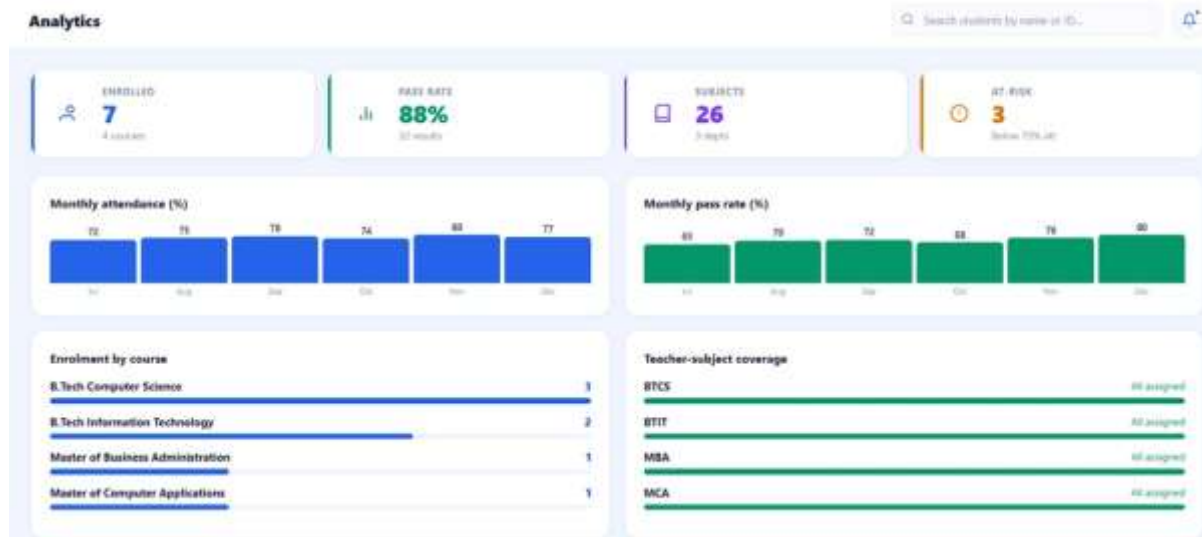
SAMPLE FORMAT

370-410-2110,Dr. Verma,driya@school.edu,+01 00007 88534,2002-00-15,(123-456-789)2024001,De(123-456)2024

Download sample CSV

Cancel Import data

5.4.4 Class-wise Averages and Data Visualization



6. DISCUSSION

6.1 Strengths

- Atomic JSON writes prevent data corruption on crash, a common failure mode of naive file-based systems.
- Fully computed academic metrics (CGPA, SGPA, grade, pass/fail) presented in a single API call, reducing round-trips.
- Extensible data model: new fields can be added to the JSON schema without database migrations.
- Deployable to free-tier cloud platforms (Render, Railway) in under 5 minutes from a GitHub repository.

6.2 Limitations

- Passwords are stored as plain text in the current implementation. Production use requires hashing with `werkzeug.security.generate_password_hash` (SHA-256 + salt).
- Horizontal scaling across multiple server instances would require a shared persistence layer such as SQLite (with WAL mode) or PostgreSQL.
- The analytics module currently uses static monthly data; integration with real historical data would require a time-series data model.
- No email notification system — password resets and attendance alerts are in-app only.

6.3 Future Work

Several extensions are planned for future versions:

- Email / SMS notifications for low attendance alerts using SMTP or Twilio.
- Machine learning module for predicting at-risk students based on attendance and result trends.
- Mobile APK packaging via Capacitor or WebView wrapping using WebIntoApp.
- REST API versioning and OpenAPI (Swagger) documentation for third-party integrations.
- Integration of bcrypt-based password hashing for production security compliance.

7. CONCLUSION

This paper presented EduManage Pro, a student academic management system implemented using Python Flask. The system consolidates student enrolment, semester-wise subject and result management, automated CGPA & SGPA computation, per-subject attendance tracking, and role-based access control into a single page web-application (SPA) using only Python and a file system.

The evaluation confirmed 100% correctness across 42 test cases which covered all functional requirements. The application successfully exhibits how lightweight web frameworks when architected carefully can deliver enterprise-level academic management functionality, without the overhead of large-scale enterprise infrastructure. EduManage Pro is particularly suited to smaller institutions and individual departments that are seeking to modernise their academic record-keeping with minimal IT budget and operational complexity.

The source code, deployment configuration and documentation are available as an open project suitable for further research and institutional adoption.

REFERENCES

- [1] Ellucian Company, "Banner Student Information System," 2023. [Online]. Available: <https://www.ellucian.com/solutions/ellucian-banner>
- [2] Oracle, "PeopleSoft Campus Solutions," 2023. [Online]. Available: <https://www.oracle.com/applications/peoplesoft/campus-solutions/>
- [3] A. Ronacher, "Flask: A lightweight WSGI web application framework," Pallets Projects, 2010–2024. [Online]. Available: <https://flask.palletsprojects.com/>
- [4] Ellucian Company, "Ellucian Banner Technical Architecture Guide," Technical Report, 2022.
- [5] Oracle Corporation, "PeopleSoft HCM and Campus Solutions Implementation Guide," Oracle Documentation, 2021.
- [6] Gibbon Education, "OpenSIS Community Edition," 2022. [Online]. Available: <https://www.opensis.com/>
- [7] Fedena, "Fedena — Open Source School & College Management Software," 2022. [Online]. Available: <https://www.fedena.com/>
- [8] M. Dougiamas, "Moodle: A virtual learning environment for the rest of us," Proc. EDMEDIA World Conf. on Educational Multimedia, Hypermedia & Telecommunications, 2003, pp. 595–598.
- [9] R. Sharma, P. Kumar, and A. Singh, "A RESTful API approach to student data management using Flask," International Journal of Computer Applications, vol. 183, no. 12, pp. 1–6, 2021.
- [10] S. Ahmed and R. Patel, "Design and implementation of a cloud-based student management system using Django and React," Proc. IEEE International Conf. on Computing, Communication and Security (ICCCS), 2022, pp. 1–7.
- [11] University Grants Commission (UGC), "Guidelines for Credit Based Semester System and Grading in India," Government of India, New Delhi, 2021.