

PLACENEXA: Unified Campus Placement and Career Management Platform

Dr. B. Renuka Chowdary¹, Vangamudi Girija Rama Mallika¹, Peddagamalla Gayathri¹, Yamarthi Cholan¹, Mente Swetha¹, Potti Harshith¹

¹Department of Information Technology, Dr RVR NRI Institute of Technology Deemed to be University
Agiripalli, Pothavarappadu, Vijayawada Rural, Andhra Pradesh, India

ABSTRACT

Campus placement management connects students with employment opportunities and requires coordinated handling of student profiles, employer details, recruitment drives, eligibility information, applications and selection outcomes. When these activities rely on spreadsheets, paper forms and separate communication channels, information can become fragmented and repetitive work can increase. This paper presents PLACENEXA, a unified web-based campus placement and career management platform designed to centralize the major workflows for students and placement administrators. The platform supports student registration and authentication, profile and resume management, company records, placement-drive administration, application submission, application-status tracking and dashboard statistics. It is implemented using Python Flask, HTML, CSS, JavaScript, Bootstrap, SQLAlchemy, PyMySQL and MySQL. A layered design separates the presentation interface, application logic, object-relational mapping and persistent data storage. Session-based access checks support protected student and administrator workflows. The paper describes the problem context, related concepts, requirements, algorithms, architecture, data model, implementation approach and functional testing. The prototype demonstrates how a focused web application can organize placement information and reduce dependence on disconnected manual records. Quantitative performance benchmarks and advanced AI matching are not claimed as completed capabilities; these remain areas for future evaluation and development.

Keywords—Campus Placement Management, Career Management Platform, Recruitment Workflow, Flask, MySQL, SQLAlchemy, Session Authentication, Web Application.

Date of Submission: 09-10-2026

Date of acceptance: 20-10-2026

I. INTRODUCTION

Campus placement is a coordinated process through which students explore employment opportunities and organizations identify potential candidates. The workflow may include maintaining student profiles, publishing company requirements, announcing recruitment drives, communicating eligibility criteria, accepting applications, reviewing candidates and recording outcomes. Reliable information is essential because students and administrators need to know which opportunities are available, who has applied and what stage an application has reached.

In a manual or partially digitized environment, the information required for these activities may be distributed across spreadsheets, paper forms, email messages and messaging applications. Separate records can lead to repeated data entry, inconsistent updates and difficulty preparing summary reports. Students may also find it challenging to locate current drive details or follow the progress of submitted applications. These problems become more difficult to manage as the number of students, companies and recruitment activities grows.

PLACENEXA: Unified Campus Placement and Career Management Platform addresses this coordination problem through a centralized web application. Students use a dedicated interface to register, maintain profile information, view placement drives, submit applications and check their status. Administrators use protected pages to manage student records, company information, placement drives, applications and summary statistics. A common database supports both workflows.

A. Contributions

The paper documents a centralized record structure for students, companies, placement drives and applications; distinct student and administrator workflows; application submission and status tracking; dashboard statistics derived from stored records; and a modular architecture that can be extended in later versions. The work is a

system-design and prototype report, not a claim of validated employment outcomes or measured large-scale performance.

II. RELATED WORK AND RESEARCH GAP

A. Digital Placement Portals

Digital placement portals generally provide student registration, opportunity listings, application forms and access to recruitment information. Their central purpose is to make opportunities easier to discover and to reduce reliance on paper-based coordination. A useful placement listing should present relevant details such as company, role, eligibility conditions, location, drive date and application deadline. PLACENEXA follows this portal-oriented model by presenting recruitment drives through a structured interface.

B. Recruitment Management Systems

Recruitment management systems organize hiring activities into stages, including vacancy creation, candidate sourcing, screening, interview coordination and outcome recording. In a campus context, the stages must be interpreted around student eligibility, placement drives and institutional coordination. A centralized record system can reduce duplication by keeping company, drive and application data in linked entities rather than maintaining independent lists for each activity.

C. Relational Data Management

Relational databases are appropriate for structured records with clear relationships. Student profiles, company details, placement drives and applications have different attributes but need to be queried together. An application record can link a student to a particular drive and store its status. The use of SQLAlchemy models helps separate the Python application from the underlying relational tables, while MySQL provides persistent storage.

D. Authentication and Access Control

Authentication establishes a user's identity, while authorization determines which operations that user may perform. A placement platform must distinguish student actions from administrative actions. Session-based authentication can retain the identity of an authenticated user across requests, provided protected routes verify the relevant session value. Interface-level hiding alone is insufficient; access checks must also be enforced by the server.

E. Research Gap

The gap addressed by this project is the integration of common campus placement tasks in one compact application. Individual tools may address profiles, job listings or administrative records separately, while placement teams still need to reconcile information across them. PLACENEXA brings student management, company management, drive administration, application tracking and basic reporting into a unified workflow. Advanced company-side access, automated messaging and intelligent matching are identified as extensions rather than completed functions.

III. PROBLEM STATEMENT AND OBJECTIVES

A. Problem Statement

Manual and fragmented placement coordination creates challenges in keeping student information current, distributing drive details, preventing duplicate application records, tracking selection progress and generating reliable summaries. When each activity uses a separate record or communication channel, administrators may need to repeat data entry and students may not have a single location from which to review opportunities and application progress. The problem is therefore to design a centralized, role-aware platform that supports the core campus recruitment workflow using accessible web and database technologies.

B. Project Objectives

The principal objective is to design and implement a web-based platform that centralizes placement information for students and administrators. The specific objectives are: (i) to provide secure student registration, login and profile management; (ii) to let administrators maintain company and placement-drive records; (iii) to allow students to view available drives and apply online; (iv) to track each application through a controlled status workflow; (v) to store all records in a relational MySQL database; and (vi) to verify the main workflows through functional testing.

C. Scope

The current scope covers student and administrator access, student profile handling, resume information, company management, placement-drive management, application submission, status updates and dashboard statistics. The

design focuses on the core recruitment workflow rather than a complete enterprise recruitment suite. Company accounts, external job-platform integration, automated email and SMS, mobile applications, AI-based matching and advanced analytics are outside the implemented scope described in the source documentation.

D. Intended Users

Students are the end users who create accounts, maintain their information, review drives and monitor applications. Placement administrators maintain the operational data and review submitted applications. The platform's role separation is intended to make each user's available actions clearer and to reduce the risk of unauthorized access to administrative functions.

IV. REQUIREMENTS AND FEASIBILITY

A. Functional Requirements

Functional requirements describe the operations that the system must perform for each user role. They were derived from the project scope and cover account handling, profile maintenance, drive discovery, application submission and administrative management, as summarized in Table 1.

Table 1. Functional requirements derived from the project scope

ID	Requirement	Primary actor	Expected outcome
FR-01	Register with required student details	Student	Account record is created after validation
FR-02	Authenticate and end a session	Student / Administrator	Authorized dashboard is displayed
FR-03	View and update profile information	Student	Current profile values are saved
FR-04	View available placement drives	Student	Drive list and relevant details are displayed
FR-05	Submit an application to a drive	Student	Application record is linked to student and drive
FR-06	Maintain company information	Administrator	Company data can be created and updated
FR-07	Create and maintain placement drives	Administrator	Drive information is stored for students to view
FR-08	Review and update application status	Administrator	Status change is retained in the application record
FR-09	View summary information	Administrator	Counts are derived from stored records

B. Non-Functional Requirements

Non-functional requirements describe the qualities expected from the platform rather than individual features. Table 2 interprets each quality attribute for PLACENEXA and pairs it with a practical way of verifying it.

Table 2. Non-functional requirements and possible verification

Attribute	Requirement interpretation	Verification approach
Usability	Consistent forms, navigation and feedback	Walk through registration, drive browsing and application flows
Reliability	Database changes should be stored consistently	Repeat CRUD operations and inspect resulting records
Security	Restrict role-specific screens and protect credentials	Test invalid login and unauthorized route access
Maintainability	Keep routes, models, templates and configuration organized	Review module boundaries and configuration handling
Compatibility	Use standard browser-based interfaces	Exercise the core workflows in a modern browser
Performance	Common actions should respond within acceptable time	Measure response times under a defined workload

C. Technical Feasibility

The selected stack consists of widely used web-development and database components: Python, Flask, HTML, CSS, JavaScript, Bootstrap, SQLAlchemy, PyMySQL and MySQL. The application can be developed and tested in a local environment using a standard computer, a browser and a MySQL installation. The technologies support form processing, routing, relational storage and modular application design.

D. Economic and Operational Feasibility

The use of open-source development tools keeps the direct software licensing requirements low for a prototype. Operational feasibility is supported by workflows familiar to students and placement administrators: users complete forms, review tables and access dashboards. Adoption would still require user orientation, data-maintenance rules and clear responsibility for keeping company and drive information accurate.

E. Schedule Feasibility

The project can be completed in incremental phases, which supports schedule feasibility: requirement analysis, database design, implementation of the student modules, implementation of the administrator modules, integration, and finally testing and documentation. Because each module builds on the shared database models, work can proceed in parallel and be checked at the end of every phase. A single academic semester is therefore a reasonable timeline for the prototype, while later enhancements can be scheduled as separate releases.

V. TECHNOLOGY STACK AND DEVELOPMENT ENVIRONMENT

A. Frontend Layer

HTML structures pages, forms, navigation and data tables. CSS controls presentation, spacing and visual consistency. Bootstrap supplies responsive interface components that can be reused across registration pages, dashboards and administrative views. JavaScript supports client-side interaction and validation where required. The frontend communicates with the backend through browser requests and rendered templates.

B. Backend Layer

Python is the primary programming language and Flask is the web framework. Flask routes receive requests, validate submitted fields, invoke application logic and return the appropriate page or response. A modular route structure helps keep student operations separate from administrative operations. Server-side validation remains important even when browser-side validation is provided, because client input cannot be assumed to be valid.

C. ORM and Database Connectivity

SQLAlchemy is used as the object-relational mapping layer. It allows application code to work with model objects and relationships instead of manually constructing every SQL statement. PyMySQL provides connectivity between Python and MySQL. MySQL stores persistent entities including student records, administrator records, company details, placement drives and applications. The data layer must maintain consistent identifiers and relationships between these entities.

D. Development Tools

Visual Studio Code serves as the code editor, providing syntax highlighting, debugging and an integrated terminal. Google Chrome is used to test the interface and workflows, and its developer tools help inspect requests and page layout. MySQL client tools are used to create the schema and inspect stored records, while Python virtual environments and the pip package manager keep the Flask dependencies organized.

E. Hardware and Software Requirements

The development environment combines a standard personal computer with open-source software. Table 3 lists the software components and the role of each one in PLACENEXA, and the hardware needs are described after the table.

The documented prototype environment requires a computer with a dual-core or better processor, at least 4 GB RAM, approximately 10 GB of development storage and an internet connection for installing packages. A supported operating system, Python runtime, MySQL server and modern browser are needed. These are development-oriented requirements; production hosting needs should be estimated separately according to expected usage and availability requirements.

Table 3. Technology stack documented for the prototype

Component	Technology / environment	Role in PLACENEXA
Programming language	Python	Application logic and validation
Web framework	Flask	Routing and request handling
Presentation	HTML, CSS, JavaScript	Page structure, styling and interaction
UI toolkit	Bootstrap	Responsive layout and interface components
ORM	SQLAlchemy	Mapping between application objects and relational records
Database	MySQL	Persistent storage for accounts and placement records
Database connector	PyMySQL	Connectivity between Python application and MySQL
Development editor	Visual Studio Code	Code editing and local debugging
Browser	Google Chrome or equivalent	Interface testing and workflow verification

VI. SYSTEM ARCHITECTURE

PLACENEXA follows a layered web architecture. A user interacts with HTML templates styled with CSS and Bootstrap, with JavaScript supporting selected interface interactions. Requests reach Flask routes, where authentication, session checks, input validation and application logic are performed. SQLAlchemy maps application objects to database entities, and MySQL stores the persistent records. The response is rendered back to the user through the browser.

A. Presentation Layer

The presentation layer includes registration and login forms, student dashboards, placement-drive listings, application views and administrator management pages. Its purpose is to present data and collect user input in a consistent manner. Reusable templates and responsive components reduce the need to implement common page structures repeatedly.

B. Application Layer

The application layer is implemented with Flask. Its routes receive requests from the browser, check the session and user role, validate submitted fields and invoke the business logic that enforces placement rules such as duplicate checks and application-status handling. Results are returned to the presentation layer as rendered templates or feedback messages, while all database access is delegated to the ORM layer.

C. ORM and Database Layers

The SQLAlchemy layer represents database entities as models and coordinates database queries and changes. MySQL provides persistent storage. Keeping the database layer separate from templates improves maintainability and allows data relationships to be reviewed independently from interface layout.

D. Architecture Benefits

Separating the system into layers gives several benefits. Each layer can be modified with limited effect on the others, role checks are applied in one consistent place, database changes are isolated behind the ORM, and the structure is easier to test and explain. Table 4 summarizes the inputs, processing responsibilities and outputs of each layer.

Table 4. Responsibilities across the layered architecture

Layer	Inputs	Processing responsibility	Outputs
Presentation	Form submissions and navigation requests	Collect data and display feedback	Rendered pages and user-visible messages
Flask routes	HTTP request and session context	Route selection, authorization and validation	Application operation or error response
Business logic	Validated form data and model records	Apply placement rules and workflow transitions	Updated model state
SQLAlchemy ORM	Model operations	Translate object operations into database queries	Persisted or retrieved records
MySQL	Structured queries and transactions	Store and relate persistent entities	Consistent database results

Architecture flow: Student / Administrator → Browser Interface (HTML, CSS, Bootstrap, JavaScript) → Flask Routes and Business Logic → SQLAlchemy ORM → MySQL Database.

VII. DATABASE DESIGN AND DATA RELATIONSHIPS

A. Main Entities

The database is organized around five principal entities: students, administrators, companies, placement drives and applications. Each entity is stored as a table with its own primary key, and Table 5 summarizes the purpose of each table.

Table 5. Database tables and their purpose

Entity / Table	Purpose
Student (students)	Personal, academic, skill and resume information
Admin (admins)	Administrator authentication information
Company (companies)	Company name, role, package, eligibility and location
PlacementDrive (placement drives)	Drive title, dates, venue and description
Application (applications)	Student-drive association and application status

B. Relationships

A company may be associated with multiple placement drives. A student may submit applications to different drives, and each drive may receive applications from multiple students. The Application entity connects Student and PlacementDrive, allowing the platform to retrieve a student's applications or review applicants for a particular drive. These relationships should be enforced through consistent identifiers and suitable database constraints.

C. Data Integrity

Data integrity is maintained through primary keys, foreign keys that link drives to companies and applications to students and drives, uniqueness constraints on identifiers such as roll number and email, and required-field validation. A uniqueness rule on the student–drive pair prevents duplicate applications. These constraints should be combined with server-side validation so that invalid values never reach the database.

D. Reporting Queries

The application entity is central to reporting because it connects students to placement opportunities. The student-to-application relationship is typically one-to-many, and the drive-to-application relationship is also one-to-many. This means the application table represents a many-to-many business relationship between students and drives. The schema should enforce valid foreign-key references and, where the project rules require one application per student per drive, a uniqueness constraint across those two identifiers. The exact physical schema should be confirmed against the final SQLAlchemy models used by the running application.

Table 6. Principal data entities and their roles

Entity	Representative attributes	Relationship / purpose
Student	Student ID, name, roll number, email, branch, CGPA, skills	Owns profile details and may submit multiple applications
Administrator	Account ID, login identifier, role	Maintains company, drive and application records
Company	Company ID, name, job role, package, location, eligibility	Can be associated with multiple placement drives
Placement Drive	Drive ID, company ID, title, date, deadline, venue, eligibility	Represents a recruitment opportunity published to students
Application	Application ID, student ID, drive ID, submission date, status	Associates one student with one drive and tracks progress
Resume / Profile File	File reference and metadata, where configured	Supports student profile and resume handling

VIII. FUNCTIONAL MODULES

A. Student Registration and Profile Management

The registration module collects the information required to establish a student account. The project documentation identifies fields such as full name, roll number, email, phone, password, branch, CGPA, skills and resume information. Registration should validate required fields and check for duplicate identifiers before saving a record. The profile module allows relevant information to be maintained so that students do not need to re-enter the same details for every opportunity.

B. Authentication and Dashboard

The login workflow validates submitted credentials against a stored account. On successful authentication, the application stores an identifier in the session and opens the appropriate dashboard. The dashboard provides navigation to profile information, available placement drives and applications. Logout clears the session so that protected pages are no longer available through the active authenticated session.

C. Company Management

The administrator's company module maintains information about organizations participating in campus recruitment. Documented fields include company name, job role, package, eligibility CGPA, location and drive date. Create, read, update and delete operations support routine maintenance. Accurate company records form the foundation for publishing useful drive listings.

D. Placement-Drive Management

A placement drive represents a recruitment opportunity and may contain a title, company association, drive date, last date to apply, venue and description. Administrators can create and update these records. Students view available drives and review the information before deciding whether to apply. Clear dates and eligibility details help reduce uncertainty.

E. Application Management

The application status should be treated as a controlled workflow value rather than arbitrary text. At minimum, the implementation needs a defined initial state and documented administrator actions that move an application to later states. If additional states such as shortlisted, interview scheduled, selected or rejected are introduced, the interface and database validation should use the same vocabulary. Status history and timestamps are useful extensions because they allow placement staff to reconstruct how a decision progressed over time.

Table 7. End-to-end placement workflow

Workflow stage	Student action	Administrator action	Stored result
Account setup	Enter registration details	No action unless support is needed	Student account record
Profile preparation	Maintain academic and skill information	Review available profile information	Updated student profile
Opportunity discovery	Open available drive details	Create or update company and drive data	Drive record visible to eligible users
Application	Choose a drive and submit	View received applications	Application linked to student and drive
Review	Monitor application status	Change status after review	Status and related record updated
Reporting	View own application progress	Review dashboard summaries	Counts based on current database records

IX. ALGORITHMS AND WORKFLOW LOGIC

A. Student Authentication Algorithm

The authentication sequence begins when a user submits login credentials. The application validates that required fields are present, retrieves the relevant account record and checks the submitted credentials. If the credentials are valid, the user's identifier is stored in the session and the appropriate dashboard is displayed.

B. Duplicate Record Validation

During registration, the application checks whether a record with the relevant unique identifier already exists. Depending on the configured data rules, the check may include roll number or email address. If a duplicate is detected, the system rejects the new record and displays a useful message.

C. Placement-Drive Search

The search workflow retrieves available placement-drive records and optionally filters them by company name, drive title or job role. Matching records are displayed in a structured view. Search should handle empty queries sensibly and avoid exposing records that are not intended to be visible to the current user.

D. Eligibility Filtering

Eligibility filtering is best treated as an explicit comparison between a drive's requirements and the information maintained in a student profile. A rule may compare branch, CGPA or other conditions recorded for the drive. The result should be presented as guidance and should identify which recorded condition was not satisfied when a match is rejected.

E. Application Submission and Status Management

A high-level application submission algorithm can be expressed as follows: (1) verify that the user session identifies an authenticated student; (2) validate the selected drive identifier; (3) retrieve the drive and check that it is available for applications; (4) check whether a matching application already exists when duplicate prevention is required; (5) create the application with its initial status and timestamp; (6) commit the transaction; and (7) display a success message.

X. IMPLEMENTATION AND SECURITY

A. Module Integration

Implementation connects Flask routes, SQLAlchemy models, templates and MySQL tables. Registration and login are linked to account records; company and drive management pages perform database operations; application routes create and retrieve student-drive links; and dashboard routes query counts for display. Templates present the data returned by the backend, while static resources provide styling and client-side behavior.

B. Session Management

Session management should be consistent across all protected routes. A successful login should establish the minimum session information needed to identify the account and role. A logout operation should clear the session, and a protected route should check the session on every request rather than relying on the fact that the user previously opened a dashboard.

C. Credential and Input Protection

The paper's source documentation identifies session-based authentication and notes that security can be strengthened. Production readiness should include secure password hashing, input validation, CSRF protection where applicable, secure cookie settings, restricted database credentials and role-based authorization.

D. File and Resume Handling

Resume handling is included in the student-facing scope. If resume files are uploaded, the application should validate file type and size, use safe filenames, restrict file access and avoid executing uploaded content. The stored profile should remain consistent with the actual file state.

E. Maintainability

Maintainability is supported by keeping routes, models, templates and configuration in separate modules with consistent naming. Clear module boundaries make it easier to add fields, correct defects and introduce new features such as analytics. Documented test cases allow regression checks after every change, and Table 8 lists the security and operational controls that should be reviewed as the system evolves.

Table 8. Security and operational control considerations

Risk / control area	Potential issue	Recommended control
Authentication	Unauthorized access to protected pages	Check session and role on every protected route
Passwords	Credential exposure if stored as plain text	Store salted password hashes using a suitable password-hashing library
Input validation	Malformed or unexpected values	Validate server-side and constrain database fields
Database secrets	Credentials exposed in source code	Use environment-based configuration and restrict secret access
Resume uploads	Unsafe file or unauthorized download	Validate type and size; use safe names and controlled access
Session handling	Session misuse or fixation	Use secure cookie settings and rotate/clear sessions appropriately
Data recovery	Accidental loss or corruption	Create and test routine database backups

XI. TESTING AND FUNCTIONAL EVALUATION

Functional testing checks whether each workflow produces the expected interface and database result. The source paper documents tests for student registration, duplicate roll-number handling, student login, invalid login, administrator login, company CRUD operations, placement-drive CRUD operations, student application submission, application-status updates and logout. The results below summarize the functional test coverage reported in the project paper; they are not quantitative performance measurements.

Table 9. Reported functional test results

Test case	Expected result	Reported status
Student registration	Student record created	Pass
Duplicate roll number	Duplicate handled or rejected	Pass / message improvement noted
Student login	Student dashboard displayed	Pass
Invalid login	Invalid-credentials message	Pass
Administrator login	Administrator dashboard displayed	Pass
Company add/edit/delete	Record stored, updated or removed	Pass
Placement-drive add/edit/delete	Drive stored, updated or removed	Pass
Student application	Application record stored	Pass
Application status update	Stored status changed	Pass
Student/admin logout	Session cleared	Pass

A. Interpretation

The reported tests indicate that the primary user workflows were exercised in the prototype environment. The duplicate-record case includes a noted opportunity to improve the clarity of the user-facing message. Passing functional tests shows that the tested operations behaved as expected in the tested environment; it does not establish high-load performance, security certification, or effectiveness across every possible input.

B. Further Evaluation Needed

The reported tests are functional and were performed on a prototype, so further evaluation is required before institutional use. Table 10 proposes a repeatable matrix covering registration, access control, relational integrity, duplicate applications and input validation, which should later be extended with usability, performance and security testing.

Table 10. Functional test matrix for repeatable verification

Test ID	Test scenario	Expected outcome	Interpretation
TC-01	Submit valid registration data	Account is created and user receives confirmation	Positive registration path
TC-02	Submit a duplicate roll number or email	Duplicate is rejected with a clear message	Uniqueness validation
TC-03	Submit incorrect login credentials	Access is denied without exposing sensitive details	Authentication negative path
TC-04	Open a protected student page without login	User is redirected or denied access	Session enforcement
TC-05	Create a company record as administrator	Company data is stored and appears in the list	Administrator CRUD path
TC-06	Create a placement drive linked to a company	Drive is stored with valid relationship	Referential integrity
TC-07	Apply to a valid drive	Application record is created once	Application workflow
TC-08	Attempt duplicate application where prohibited	Second record is prevented	Duplicate application control
TC-09	Change application status	Updated status appears in relevant views	Status persistence
TC-10	Enter invalid or missing required values	Validation prevents invalid persistence	Input validation

XII. RESULTS, DISCUSSION AND LIMITATIONS

A. Functional Outcomes

The prototype organizes major placement activities within a single web application. Students can use the student workflow to register, access their dashboard, view drives, submit applications and review status. Administrators can maintain companies and drives, view applications and use summary statistics. Relational storage groups information by entity and supports retrieval through linked records.

B. Operational Implications

Centralization can reduce the number of separate files administrators need to reconcile and provides a common reference for students and placement staff. Application status tracking improves visibility into the stored state of a submission. Dashboard counts can provide a quick overview of the records currently held by the platform. These benefits depend on accurate data entry, consistent status updates and appropriate use of the system.

C. Limitations

The project is therefore best understood as a foundation for an institutionally governed placement workflow. Its success will depend not only on the code but also on accurate data ownership, defined administrative procedures, support for users and regular maintenance. The technical design can provide a common record of opportunities and applications, while the placement office remains responsible for confirming official drive details and interpreting recruitment outcomes. This distinction keeps the system useful without overstating what the prototype has demonstrated.

Table 11. Deployment and maintenance checklist

Release area	Before deployment	During operation
Configuration	Disable debug mode; protect secrets; verify environment settings	Rotate secrets when required and review configuration changes
Database	Check schema, constraints, indexes and backup procedure	Monitor availability and test restore procedures
Access control	Verify student and administrator permissions	Remove obsolete accounts and review privileged access
Application files	Validate resume type, size, storage and download access	Review storage growth and remove files according to retention rules
Testing	Run registration, drive, application and authorization tests	Repeat regression tests after material changes
Monitoring	Define logs and operational contacts	Review errors and respond to incidents
User support	Publish instructions for common workflows	Collect feedback and clarify confusing messages

D. Threats to Validity

There are several limits to the conclusions that can be drawn from a local functional prototype. First, the test environment may not represent the hardware, network latency or concurrency of a campus-wide deployment. Second, the reliability of results depends on the quality of the entered test records. Third, functional success does not establish that users find the application easy to use, that every security threat has been addressed, or that performance remains acceptable at peak recruitment periods. A future evaluation should state the environment and test data clearly so results can be reproduced and compared.

E. Discussion

A final consideration is the balance between automation and human responsibility. The platform can organize information and apply explicit eligibility checks, but recruitment decisions may include additional criteria and procedures set by an employer. The system should not imply that a student is guaranteed selection merely because a profile appears to match a drive. Likewise, summary counts should be presented as descriptions of stored records, not as evidence of placement quality unless an evaluation method supports that conclusion. Clear labels and transparent rules help users understand the role of the software in the wider campus recruitment process.

Table 12. Scope of the reported evaluation and remaining validation

Evaluation dimension	Evidence available in the project	What remains to be measured
Workflow coverage	Registration, authentication, profile, drive and application paths are documented	Completion rate across a larger set of end-to-end tests
Data consistency	Relational entities and application links are described	Constraint coverage and recovery from failed transactions
Usability	Browser-based student and administrator views	Task completion time and user feedback
Performance	Prototype can be exercised in a local environment	Response time, throughput and concurrent-user behavior
Security	Role separation and session-based access are in scope	Authorization tests, secure configuration and vulnerability review
Operational readiness	Modular web application and database stack are identified	Deployment, monitoring, backup and restore evidence

XIII. FUTURE ENHANCEMENTS AND CONCLUSION

A. Future Enhancements

Potential analytics include the number of drives published during a period, application volume by role or branch, the distribution of application statuses and the time between application and status update. Such summaries can help administrators identify process bottlenecks, but they should be interpreted with context. For example, a low application count may reflect limited eligibility, a short application window or a mismatch between the role and student interests. Any future recommendation feature should explain its inputs and should not make unsupported claims about a student's employability.

B. Conclusion

From an implementation perspective, the selected technology stack supports a conventional web application that can be developed and tested incrementally. Flask provides a small routing layer, SQLAlchemy structures database interaction, and MySQL stores the related records. The separation of student and administrator workflows gives the project a foundation for later extensions without requiring the core placement records to be redesigned. The

next development stage should prioritize documented access controls, automated regression tests, database constraints and a repeatable deployment process.

REFERENCES

- [1] M. Grinberg, Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
- [2] Python Software Foundation, "Python Documentation and Programming Reference." Available: <https://docs.python.org/3/>.
- [3] SQLAlchemy, "SQLAlchemy ORM and Core Documentation." Available: <https://docs.sqlalchemy.org/>.
- [4] MySQL, "MySQL Reference Manual." Available: <https://dev.mysql.com/doc/refman/8.0/en/>.
- [5] Bootstrap, "Bootstrap Documentation and Components." Available: <https://getbootstrap.com/docs/5.3/>.
- [6] Mozilla Developer Network, "HTML, CSS and JavaScript Web Documentation." Available: <https://developer.mozilla.org/>.
- [7] I. Sommerville, Software Engineering. Pearson Education.
- [8] R. S. Pressman and B. R. Maxim, Software Engineering: A Practitioner's Approach. McGraw-Hill.
- [9] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems. Pearson.
- [10] A. Silberschatz, H. F. Korth and S. Sudarshan, Database System Concepts. McGraw-Hill.