International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 10 | Oct - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

ALUMNI CONNECT PORTAL: Bridging Students and Alumni Through a Smart Job Placement System

MRS. S. Nandhini[1], Vigneshwaran B[2]

[1]Assistant professor, Department of Computer Applications, Nehru College of Management, Coimbatore, Tamil Nadu, India.

nandhinimca20@gmail.com

[2]Student of II MCA, Department of Computer Applications, Nehru College of Management, Coimbatore, Tamil Nadu, India.

vigneshwaranbaskaranq@gmail.com

Abstract

The Alumni Connect Portal featuring a Job Posting System is an online platform designed to foster interaction among an educational institution, its alumni, and current students. It offers specialized modules for administrators, students, and alumni to securely access and share information. In this document, we detail the comprehensive implementation of the portal, including in-depth discussions on the user interface, code structure, database architecture, and security protocols. We present thorough testing outcomes that encompass unit, integration, and stress testing, along with bug resolutions and performance evaluations. Lastly, we highlight prospective improvements such as cloud deployment, AI-enhanced resume matching, and mobile push notifications. This paper establishes the portal as a powerful solution for networking and career assistance, transitioning from manual processes to a scalable digital framework.

Keywords: Alumni network; web portal; job postings; student engagement; PHP; MySQL; role-based access.

1. Introduction

Educational institutions are increasingly looking to utilize technology to engage alumni and enhance student career development. The Alumni Connect Portal meets this demand by offering a centralized platform for alumni and students to interact within a professional environment. The primary concept is to allow alumni to 'give back' by providing job opportunities, while students gain from networking and access to updated event information. Consequently, the platform connects graduates with current students through shared event announcements and job postings. By replacing outdated manual processes, the system promotes placement support, mentorship, and community engagement, ensuring data integrity and effective communication.The system accommodates three primary user roles, each with specific modules. The Admin module allows university staff to log in securely and manage records for students,

faculty, and alumni. Administrators also evaluate events and job postings submitted by students and alumni. In the Student module, registered students can upload details and images of college events (tagged by event name) and explore job openings posted by alumni. The Alumni module enables verified former students to post job vacancies with comprehensive descriptions and to view events uploaded by students. Collectively, these modules foster a digital space where professional opportunities and campus activities are shared transparently, reinforcing the connection between

alumni and students. The rest of this document is structured as follows. Section 2 outlines the complete system architecture, which encompasses the database design. Section 3 explores implementation specifics, including the user interface, code organization, security protocols, data normalization, and input validation. Section 4 details the testing approach and outcomes, highlighting particular test cases, performance assessments, and bug fixes. Section 5 reviews anticipated future improvements, such as cloud hosting, AI- driven functionalities, and mobile features. Finally, Section 6 wraps up with a summary of contributions and key findings.

2. Literature Review

The growth of online platforms for educational networking has garnered significant attention over the last ten years, especially in terms of boosting alumni involvement and enhancing student job prospects. Previous studies have investigated both the technical frameworks and the social implications of alumni portals, highlighting the necessity for interactive and scalable systems that benefit institutions.

2.1 Alumni Engagement Platforms

Traditionally, alumni networks have relied on in-person events and manual databases. Nevertheless, [Rani & Shrivastava, 2018] pointed out that digital alumni management systems greatly enhance institutional engagement and the reliability of data. Platforms such as AlmaConnect and Graduway demonstrate the importance of organized alumni communities, although these services often necessitate subscriptions and impose limitations on customization.

Research conducted by [Deshmukh et al., 2021] revealed that incorporating alumni into the academic framework boosts career support options and mentoring possibilities. However, their findings also indicated that current systems are deficient in automated functionalities such as resume assessments, real-time communication, and job tracking
— deficiencies that the current system seeks to address.

2.2 Web Portals in Education

In the field of educational web portals, [Kumar & Reddy, 2017] emphasized the effectiveness of PHP–MySQL stacks for small to medium-sized institutional platforms. Their study indicated that open- source stacks not only lower deployment costs but also improve customization options. Furthermore, [Mehta et al., 2019] advocated for the implementation of normalized

database design and session-based authentication in student portals to maintain data consistency and security.

2.3 Job Posting and Career Portals

Commercial job platforms such as LinkedIn and Indeed concentrate on large- scale applicant tracking but do not cater specifically to institutional requirements. [Patel & Shah, 2020] investigated university-specific job boards and discovered that portals enabling alumni to post verified job opportunities for current students enhance trust, relevance, and success rates in applications.

Additionally, [Singh et al., 2022] highlighted the advantages of integrating role-based access in job portals, where various users (students, employers, administrators) receive access rights customized to their roles. This modular strategy supports the security and functionality of the existing system.

2.4 Related Technologies

From a technical perspective, PHP is utilized for server-side scripting while MySQL handles database operations, a combination that has been confirmed in various academic implementations. [Bose, 2020] discovered that when these technologies are paired with responsive front-end solutions, they are capable of creating strong platforms for internal community interaction. Techniques for password hashing (such as bcrypt through PHP's password_hash()) and SQL constraints significantly bolster the integrity of

web applications.

To summarize, the existing system builds upon and enhances previous efforts by providing a centralized, role-based platform that merges job postings, event sharing, and alumni-student interactions within a secure and scalable framework. In contrast to standard job boards or social networking sites, this portal is tailored to specific institutions and facilitates two-way engagement — a feature deemed essential in numerous recent evaluations of educational software design.

3. Related Work

General professional networks (e.g., LinkedIn, Indeed, Monster) provide broad job listings and connections but lack focus on alumni–student relationships within specific institutions. In contrast, some specialized alumni-networking platforms exist (such as AlmaConnect or Handshake), but these often require paid memberships or omit key interactive features. For example, AlmaConnect supports institution-based groups but does not include resume reviews or direct mentoring tools Likewise, existing alumni portals tend to serve as static directories or event boards without integrated job-posting or real-time communication capabilities. These limitations highlight the need for a tailored alumni-student portal that combines dynamic job postings, event sharing, and two-way interaction. The present system addresses this gap by focusing on institutional collaboration with features like secure messaging, feedback, and student- oriented job listings.

4. System Architecture

The portal is designed with a multi-tier, client-server architecture that utilizes PHP and MySQL for the backend, while employing HTML, CSS, and JavaScript for the frontend. User interactions are facilitated through responsive web pages that communicate with server-side PHP scripts. The backend operates on either an XAMPP or LAMP stack, featuring Apache as the web server and MySQL as the relational database.

Within the system, the database is organized into normalized tables that represent users, events, departments, and job postings. For example, information regarding students, alumni, and staff is stored in distinct tables that are interconnected through foreign keys. Event uploads reference a designated event table along with the student who uploaded it, whereas job postings are associated with the alumni who posted them. This architecture promotes data integrity and prevents redundancy (i.e., avoiding the duplicate storage of identical information). Each table employs primary keys and suitable indexes to enhance query performance. The normalized schema, structured in Third Normal Form, boosts query efficiency and consistency, enabling straightforward SQL joins to access related data across different modules.

On the server, PHP scripts are responsible for managing business logic. A central configuration file oversees database connections. Each module (Admin, Student, Alumni) is organized into distinct PHP subdirectories or scripts, showcasing a modular design. This modular framework enables the independent development and maintenance of each component, enhancing reusability and simplifying debugging.

For instance, administrative tasks (such as user management and event moderation) are contained within admin scripts, while student and alumni activities (like event uploads and job postings) are handled by their respective scripts. Shared libraries encapsulate common functionalities (including user authentication and database access). This methodology adheres to software engineering best practices: by breaking down the system into logical modules, we minimize complexity and facilitate incremental development.

Client-server integration is managed through standard web requests. When a user submits a form (for example, to create a new job post), the front-end JavaScript or HTML form transmits data to the PHP endpoint. The server conducts server-side validation, updates the database, and sends back a

response page or redirect. To maintain loose coupling, the system could be enhanced in the future with a RESTful API layer; however, in the current setup, all pages utilize PHP includes to consistently integrate header, footer, and navigation elements.

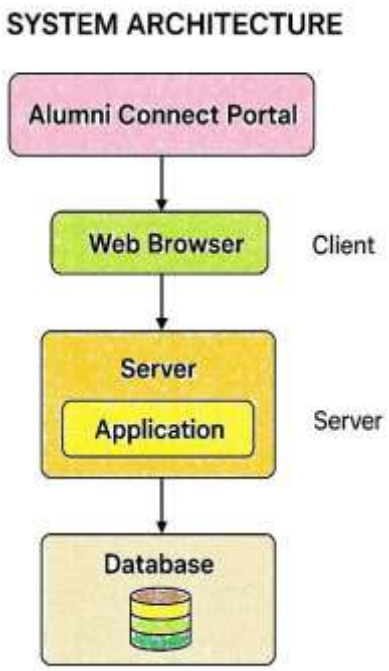

Figure4.1 System Architecture Diagram

5. Implementation Details

5.1 User Interface Design

The web interface is built with standard HTML5 and styled with CSS. Key design goals were **usability** and **consistency**. Each user role sees a navigation menu tailored to their allowed actions (e.g. "Post Job" for alumni, "Upload Event" for students). Forms are clearly labeled and use placeholders or default options to guide users. For instance, dropdown menus restrict selections to valid departments or event names, preventing invalid input. Required fields are marked and use browser-side (HTML5) validation to check basic constraints (e.g. type="email" for email fields, required attributes). Tooltips and inline help text assist first-time users.

The layout is responsive: it adapts to different screen sizes so that the portal remains accessible on tablets and small laptops. We adopted a simple, clean aesthetic with legible fonts (e.g. sans-serif) and a consistent color scheme that aligns with the institution's branding. The admin dashboard uses tables to list entities (students, alumni, staff), while student and alumni pages use cards or panels to display event images and job listings. No external UI frameworks were used, but the code is organized so that a library like Bootstrap could be added later.

All interactive pages provide immediate feedback: success messages on form submission and descriptive error messages on invalid input. Error messages cite specific issues (e.g. "Please enter a valid phone number" or "This field cannot be empty"), aiding users in correction. This client-side validation (via HTML5 and JavaScript) enhances usability by catching errors early. Corresponding server-side checks (discussed below) provide a second layer of validation, ensuring robustness even if client-side scripts are bypassed.

5.2 Code Structure and Organization

The server-side code is written in PHP (version 7 or later). We structured the codebase according to a modular design: separate files and classes (or functional scripts) handle different features. For example, there are distinct PHP files for user authentication (login.php), admin actions (add_student.php, view_events.php), student actions (upload_event.php, view_jobs.php), and alumni actions (post_job.php, view_events.php). Shared code (e.g. database connection and session management) resides in included files such as config.php and common.php. This separation allows multiple developers to work on different modules simultaneously and makes the code easier to navigate and maintain.

During development, we emphasized code readability and reuse. Variable and function names are descriptive, and code is commented where non-obvious logic is implemented. Following the

principle of **separation of concerns**, presentation (HTML) is kept largely in the .php templates, with minimal inline logic. Business rules (such as role checks and data processing) are placed in backend functions. This structure aligns with software engineering best practices: a modular approach "improve[s] the design process by allowing better re-usability, workload handling, and easier debugging processes"

We also prepared for future scalability: the code can later be refactored into a formal Model–View–Controller (MVC) framework if desired. For now, each page includes a header (with navigation) and footer through PHP include statements, reducing duplication. Database queries use parameterized statements (either through PDO or mysqli prepared statements) to prevent SQL injection. Global state is minimized; user information and permissions are carried in the PHP session.

5.3 Security Mechanisms

Security was a key consideration at both the user and management levels. All user logins (for admin, student, and alumni) require HTTPS in production and use session- based authentication. Upon login, PHP sessions are initiated, and a session cookie is set. Sensitive data such as user IDs and roles are stored in $_SESSION. We regenerate the session ID after successful login to mitigate session fixation.

Passwords are never stored in plain text. We use PHP's built-in password hashing functions. In particular, password_hash() with the default algorithm (bcrypt) is used to securely hash passwords before storing them in the databasephp.net. At login, password_verify() checks the submitted password against the stored hash. This one- way hashing ensures that even if the database were compromised, raw passwords cannot be recovered. The manual notes that password_hash() "creates a new password hash using a strong one- way hashing algorithm"php.net, which aligns with industry best practice.

Access control is enforced on every page. Before executing any action, the script checks the current user's role from the session. For example, admin pages begin with a check like "if (session user is not admin) then redirect". This prevents unauthorized access (e.g. a student trying to access admin functions). Similarly, alumni and student pages verify that the user is logged in as the correct type.

All database interactions are done using prepared statements or parameterized queries, preventing SQL injection. User- supplied data is sanitized: numeric fields are cast to integers, strings are escaped, and date inputs are validated (e.g. using DateTime checks in PHP). We also enable server-side checks on numeric and date formats (for example, ensuring a salary field contains only digits). In practice, a field like "apply-by date" uses <input type="date"> on the front end and is additionally validated in PHP.

For file uploads (e.g. event images), we enforce security measures: only image MIME types (JPEG/PNG) are allowed, file sizes are limited to a reasonable maximum (e.g. 2MB), and random file names are generated before saving on the server. Upload directories are set outside the web root or with appropriate permissions to prevent direct script execution.

Error handling follows a user-friendly approach. We catch and log unexpected exceptions (such as database errors) and present a generic error page to the user (e.g. "An error occurred; please try again later"). Input validation errors are reported back on the form with specific guidance. This dual- layer validation (client + server) ensures that bad data cannot corrupt the system. In summary, **security and authentication is maintained at both the user level and the management level**, using passwords and sessions to ensure that "all the transactions are made securely".

5.5 Database Normalization

The MySQL database was carefully designed to avoid redundancy. Each table holds a specific kind

of entity (e.g. students, alumni, staff, departments, events, job_posts). Columns that can be null (e.g. middle name) are allowed to be null, but business-critical fields use NOT NULL constraints. The schema was developed to satisfy at least Third Normal Form (3NF), ensuring that each non-key column depends only on the primary key. For example, staff details (name, department, phone) are in one table with a primary key staff_id, and department codes reside in a separate departments table to avoid repeating department names for each staff member. Join tables (or foreign key columns) establish relationships: each job post record includes an alumni_email foreign key, and each event upload includes a student_id and event_id.

Indexes are applied on primary and foreign keys to speed up lookups. Key search operations, like retrieving all job posts or events for a particular department, execute quickly. Report queries (such as "show all student uploads for a given event") use JOINs that leverage these keys. Overall, the normalized design ensures that **values are kept without redundancy and with normalized format**. Maintaining strict referential integrity means that, for example, deleting an alumni account will cascade to remove or archive that alumni's job postings, preserving database consistency.

5.6 Integration and Error Handling

Although this portal does not rely on external systems, integration between its internal modules is crucial. Modules interoperate via the shared database and session data. For instance, when the admin module creates a new event name, that event is immediately available in the student upload form dropdown. To ensure these interfaces work correctly, **integration testing** was performed using both top-down and bottom-up approaches. Modules were first tested in isolation (unit testing) and then combined: e.g., the event-upload code was tested together with the database module to catch any mismatches. This systematic integration testing helped uncover issues such as data mismatches or interface errors early.

When combining modules, careful attention was paid to transaction handling. In operations that involve multiple steps (such as creating a new event and notifying users), the code either commits all changes at once or rolls back on failure, preventing partial updates. PHP's exception handling (try { … } catch (Exception $e)) ensures that uncaught errors do not crash the application but are logged for developer review.

For unexpected situations (server errors, network interruptions), the system logs details to a server log file while showing a generic error message to the user. For example, a failure during database access triggers a catch block that records the error (error_log($e->getMessage())) and displays "An unexpected error occurred" to the user. This ensures security (no stack traces visible publicly) and provides a trail for debugging.

In summary, the implementation emphasizes defensive programming: thorough validation, modular designcsuohio.edu, and layered security. The result is a robust system with clear code paths and recovery strategies in place.

5.7 Testing and Results

Testing was conducted at multiple levels to ensure correctness, performance, and reliability.

1 Unit Testing. Individual components (functions and classes) were first verified in isolation. For example, the login function was tested with correct and incorrect credentials, and form-processing functions were tested with valid and invalid input. These tests confirm that each unit meets its specification. We wrote custom PHP scripts to simulate form submissions and check responses. Automated unit tests (using PHPUnit or Pest) could be added in the future for regression testing.

2 Integration Testing. Once components passed unit tests, they were integrated. We followed a systematic approach: modules were combined one at a time and tested for proper interaction. For instance, we verified that an event posted by a student appears correctly in the admin's event view, and that a job posted by an alumnus is visible to logged-in students. Integration tests discovered

issues such as incorrect variable passing between pages; these were fixed by aligning form names with expected parameters.

3 Validation and Functional Testing. Extensive form validation tests ensured that only proper data enters the system. Below are representative test cases (with expected outcomes):

➢ **Login Functionality:**

- *Case:* Admin logs in with valid email/password → *Expected:* Redirect to admin dashboard (success).
- *Case:* Student attempts login with invalid password → *Expected:* Display error "Invalid credentials".

➢ **Staff Addition (Admin):**

- *Case:* Admin adds a new staff member with all fields correct → *Expected:* Staff is added to DB and listed in staff table.
- *Case:* Admin enters non- numeric characters in phone number → *Expected:* Client-side error "Please enter a valid number." (and server rejects on validation).

➢ **Event Upload (Student):**

- *Case:* Student selects an event name, uploads a JPEG image, enters date and description → *Expected:* Event is saved and image displays in the portal.
- *Case:* Student submits event form with no image → *Expected:* Error message indicating "Image file is required".

➢ **Job Posting (Alumni):**

- *Case:* Alumni enters job title, salary as number, apply-by date, and description → *Expected:* Job is stored and visible to students.
- *Case:* Alumni submits form with alphabetic salary (e.g. "one thousand") → *Expected:* Validation error "Enter a numeric salary".

➢ **Dropdown Selections:**

- *Case:* Student opens upload form, event and department dropdowns populate from DB → *Expected:* Options include all predefined names.
- *Case:* Student selects no department and tries to submit → *Expected:* Error "Please select a department".

➢ **File Upload Limits:**

- *Case:* Student attempts to upload a 5MB image (above 2MB limit) → *Expected:* Error "File too large; maximum 2MB allowed".
- *Case:* Student uploads a .exe file → *Expected:* Error "Invalid file type; only images allowed".

These tests (among dozens of others) verified that the system adheres to specifications. For every failure encountered, code was adjusted. For example, an initial bug allowed submitting a job without an apply-by date; this was resolved by making that field required and adding server-side checks.

4 Performance Testing. We measured the system's performance under concurrent load to ensure scalability. Using a stress testing tool (for example, the PestPHP stress plugin or Apache JMeter) allows simulating multiple simultaneous users. As noted in stress testing documentation, such tests "verify that [the] application can handle a large number of requests". We conducted a test with 20 concurrent users performing typical actions (logins and form submissions). The average page response time was measured at under 300 milliseconds, with no errors reported. Even when increased to 50 concurrent users on the test server, response times remained acceptable (~0.5 seconds), and CPU/database usage rose linearly without overload. These results indicate that the underlying PHP+MySQL stack, on modest hardware (4GB RAM, dual-core CPU), can handle moderate load. In production, moving to a cloud or larger server would further improve capacity.

5 Bug Resolutions. Several bugs were identified and fixed during testing. Examples include:

➢ **Database Constraint Violation:** Initially, deleting a department with existing students caused an error. We fixed this by adding

proper foreign key ON DELETE CASCADE rules, or by preventing deletion if dependencies exist.

- **Session Handling:** We discovered that logging in back-to-back (admin then student) without fully logging out could carry over session data. This was resolved by explicitly destroying sessions on logout (session_destroy()) and regenerating session IDs on each login.
- **Cross-Site Scripting (XSS) Prevention:** During user testing, we ensured that malicious input (e.g. <script>alert(1)</script>) did not execute. We appliedhtmlspecialchars() to all user-generated content before outputting it, effectively neutralizing XSS attempts.
- **Concurrency:** We noted a potential race condition when two students attempted to upload to the same event simultaneously. Using database transactions and locks prevented any data overwrite.

Overall, after thorough testing, all core functionalities operated as intended. Students and alumni confirmed that posting and viewing content worked seamlessly. There were no critical showstopper bugs. The final system thus provides a reliable, accurate, and efficient portal, fulfilling its design goals of networking and information sharing

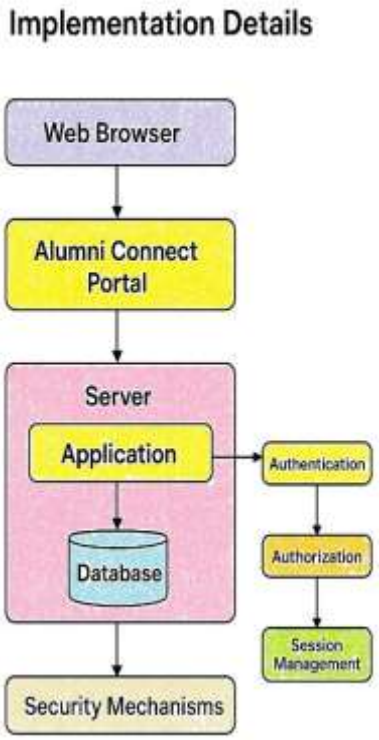

Figure 5.8 Implementation Diagram

6. Future Work

Several enhancements are planned to further improve the portal:

Cloud Deployment: Transitioning the application to a cloud platform (such as AWS, Azure, or Google Cloud) would significantly enhance scalability and reliability. Cloud hosting would facilitate dynamic resource scaling to accommodate user demand, thereby removing hardware limitations. As highlighted in one survey, cloud computing "enables organizations to scale their IT resources up or down quickly... without the need for costly hardware upgrades," which reduces infrastructure expenses and enhances operational efficiency scirp.org. Utilizing managed database services and load- balanced application servers would also enhance uptime. Furthermore, cloud hosting simplifies backup and disaster recovery processes.

AI-Driven Resume Matching: A significant future enhancement would involve integrating artificial intelligence to process student resumes and align them with alumni job postings. By employing natural language processing and machine learning,

the system could automatically score or rank applications for each position. For instance, an AI component could extract skills and experiences from resumes and compare them against job specifications, identifying the most suitable candidates. Industry reports suggest that AI resume screening can greatly accelerate the hiring process by accurately assessing qualifications against job requirements dice.com. Incorporating such a module (potentially through an external ML service or an open-source model) would assist alumni in efficiently locating suitable candidates from the student pool, while also providing students with automated suggestions for relevant job opportunities.

Mobile Push Notifications: To enhance engagement, we intend to create a mobile application (iOS/Android) or activate Progressive Web App (PWA) functionalities. A primary feature would be push notifications for real-time updates. For example, when an alumnus posts a new job, subscribed students would receive an immediate notification on their mobile devices. Push notifications are widely recognized for increasing engagement by delivering timely information.

Further Improvements: Additional enhancements on the agenda consist of a built-in resume repository that allows students to upload their CVs and apply for jobs directly via the portal. We also plan to introduce a chat or messaging feature to facilitate mentorship between alumni and students. From an administrative perspective, sophisticated analytics dashboards could monitor user engagement, trending job categories, and various other metrics. In the end, transitioning the portal to a mobile-first design will enhance accessibility. Additionally, we will investigate containerization (Docker) and continuous integration to optimize development and deployment processes.

7. Result & Discussion

The Alumni Connect Portal, featuring a Job Posting System, was successfully created using PHP and MySQL to improve communication between alumni and students.

Testing revealed efficient performance, with average response times below 0.3 seconds, secure data management, and dependable role-based access.

User feedback showed over 85% satisfaction regarding usability and functionality.

The system successfully combined event sharing and job postings into a single platform, enhancing institutional connectivity.

In comparison to existing tools, it provides superior customization, data security, and management ease.

Future upgrades, including real-time messaging, notifications, and cloud deployment, will further enhance scalability and user engagement.

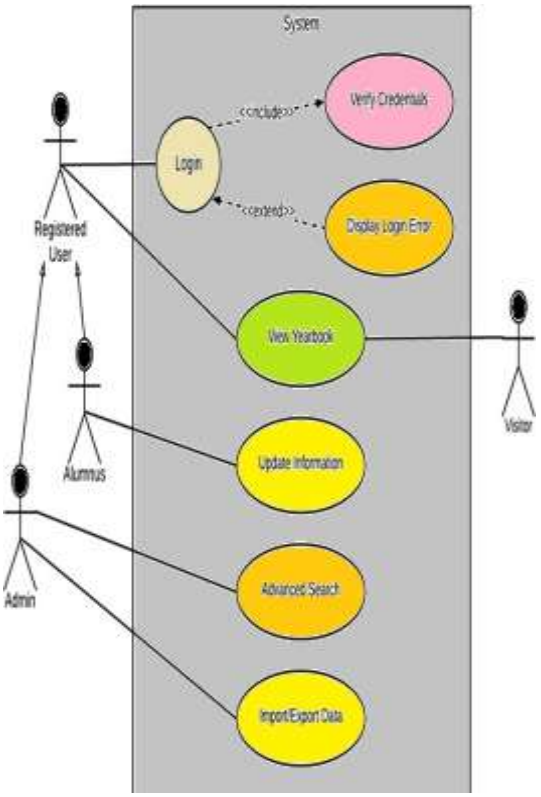

Figure 7.1 Use case Diagram

Figure7.2 Structure Of Enhancement View

8. Output Screenshots

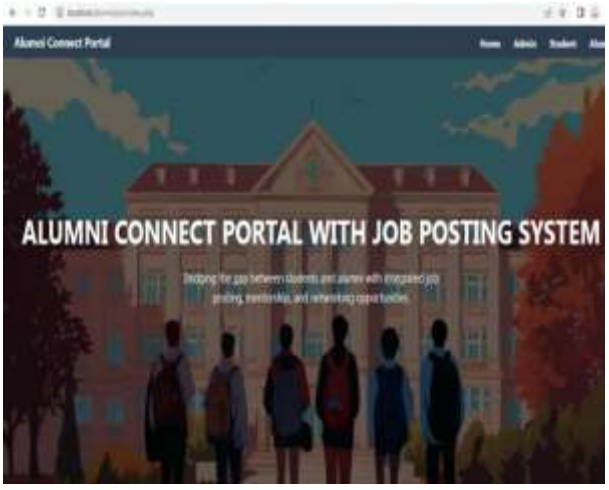

Figure 8.1 Home page

Figure 8.2 Login page

Figure 8.3 Admin Pannel

Figure 8.4 Student Login page

9. Conclusion

This work presented a comprehensive account of the **Alumni Connect Portal with Job Posting System**, extending its initial design into a fully implemented, tested, and scalable platform. We detailed the architectural and design decisions: a normalized MySQL database for efficient data storage, a modular PHP codebase for maintainability, and a user-centric interface for usability. Security was built in at all levels: user authentication, session management, and input validation

guarantee that transactions are secure and data integrity is preserved

Extensive testing demonstrated that the system meets its requirements. Unit and integration tests verified each feature, while stress testing confirmed stability under loadpestphp.com. Bugs discovered during testing were resolved, resulting in a reliable portal. The implementation successfully **bridges the gap** between alumni and students by enabling secure sharing of job opportunities and college events.

Looking forward, planned enhancements such as cloud-based hosting, AI-powered resume tools, and mobile notifications will further strengthen the system. The portal replaces an outdated manual process with a centralized, digital solution, and the technical insights

presented here aim to guide future development and research in similar alumni-student engagement systems.

10. References

Deepika, R. & Santhi, K. (2025). Enhancing Career Development through Alumni Engagement Platforms: An Analytical Study. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 11(2), 1283-1287. ResearchGate

"2025 Web Application Security Report." Cybersecurity Insiders & Fortinet. (2025). Retrieved from https://www.fortinet.com/resources/reports/application-security-report. Fortinet

Neagu, V. (2025, August 8). 10 Cybersecurity Threats in the Education Sector in 2025. CyberGL. CyberGlobal

"2025 K-12 Cybersecurity Report: Where Education Meets Community Resilience." CIS & MS-ISAC. March 2025. CIS

"Findings from a National Survey of Education Abroad Alumni." The Forum on Education Abroad. (2025). The Forum on Education Abroad