

**A REPORT
ON**

University Management System – Alumni Management Module

Submitted by,

**Ms. Karen Rena C - 20211CSD0169
Ms. Samprity Singha - 20211CSD0044
Ms. Nandini - 20211CSE0456
Mr. Piyush R - 20211CSE0651
Mr. Samrudd - 20211CSD0172**

Under the guidance of,

Dr. Jayanthi Kamalasekaran

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

At



PRESIDENCY UNIVERSITY

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Internship report “University Management System – Alumni Management Module” being submitted by “Karen Rena C, Samprity Singha, Nandini, Piyush R, Samrudd” bearing roll number(s) “20211CSD0169, 20211CSD0044, 20211CSE0456, 20211CSE0651, 20211CSD0172” in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a bonafide work carried out under my supervision.



**Dr. JAYANTHI
KAMALASEKARAN**
Associate Professor
PSCS
Presidency University



Dr. MYDHILI NAIR
Associate Dean
PSCS
Presidency University



Dr. ASIF MOHAMED H B
Associate Professor & HoD
PSCS
Presidency University



Dr. SAMEERUDDIN KHAN
Pro-Vice Chancellor - Engineering
Dean –PSCS / PSIS
Presidency University





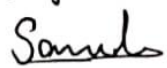
PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I hereby declare that the work, which is being presented in the report entitled “University Management System – Alumni Management Module” in partial fulfillment for the award of Degree of **Bachelor of Technology in Computer Science and Engineering**, is a record of my own investigations carried under the guidance of **Dr. Jayanthi Kamalasekaran, Associate Professor, Presidency School of Computer Science and Engineering, Presidency University, Bengaluru.**

I have not submitted the matter presented in this report anywhere for the award of any other Degree.

Name	Roll No	Signature
Karen Rena C	20211CSD0169	
Samprity Singha	20211CSD0044	
Nandini	20211CSE0456	
Piyush R	20211CSE0651	
Samrudd	20211CSD0172	

ACKNOWLEDGEMENTS

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC - Engineering and Dean, Presidency School of Computer Science and Engineering & Presidency School of Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Dean **Dr. Mydhili Nair**, Presidency School of Computer Science and Engineering, Presidency University, and Dr. “Asif Mohamed H B”, Head of the Department, Presidency School of Computer Science and Engineering, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Dr. Jayanthi Kamalasekaran**, Associate **Professor**, Presidency School of Computer Science and Engineering, Presidency University for her inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the internship work.

We would like to convey our gratitude and heartfelt thanks to the PIP4001 Internship/University Project Coordinator **Mr. Md Ziaur Rahman and Dr. Sampath A K**, department Project Coordinators “**Mr. Jerrin Joe Francis**” and Git hub coordinator **Mr. Muthuraj**.

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

Karen Rena C
Samprity Singha
Nandini
Piyush R
Samrudd

ABSTRACT

This project presents the design and development of a robust, secure, and user-friendly Alumni Management System (AMS), developed as part of Tech Mahindra's campus training initiative at Presidency University. The primary goal was to bridge the gap between academic institutions and their alumni by enabling efficient data management, communication, and event coordination within a centralized digital platform. Utilizing full stack technologies such as Spring Boot, MySQL, and Thymeleaf, the system provides role-based access control to administrator and alumni users. Key functionalities encompass creating and managing profiles, posting and registering events, maintaining a secure public alumni directory, secure credential submission for system access, and an internal messaging system. It employs layered MVC architecture, enforces data consistency through JPA, and incorporates responsive design for mobile usability. The project provided the team with an experience of developing an enterprise-grade application alongside real-world software engineering practices such as version control, modular design, secure credentialing, and agile teamwork. In addition to fostering sustained connection between the university and its alumni, the platform aims to sustain adaptably through future additions of mentorship, job boards, and donation modules.

LIST OF FIGURES

Sl. No.	Figure Name	Caption	Page No.
1	Figure 4.1	Entity Relationship Diagram	18
2.	Figure 6.1	Layered Architecture	24
3.	Figure 6.2	Alumni Management System Architecture	26
4.	Figure 6.3	Database Schema	30
5.	Figure 6.4	Frontend Implementation Flowchart	32
6.	Figure 6.5	Security Config Class	33
7.	Figure 7.1	Gantt Chart	35
8.	Figure B.1	Login Page	61
9.	Figure B.2	Admin Dashboard	61
10.	Figure B.3	View, Edit and Delete Alumni	62
11.	Figure B.4	Add New Alumni	62
12.	Figure B.5	Contacting Alumni	63
13.	Figure B.6	Posting new Event	63
14.	Figure B.7	Admin Activity Logs	64
15.	Figure B.8	Alumni Dashboard	64
16.	Figure B.9	View and Register for Events	65
17.	Figure B.10	View Registered Event	65
18.	Figure B.11	Public Alumni Directory	66

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	CERTIFICATE	ii
	DECLARATION	iii
	ACKNOWLEDGEMENT	iv
	ABSTRACT	v
	LIST OF FIGURES	vii
1.	INTRODUCTION	1
	1.1 Background Overview	1
	1.2 Training Program Overview	1
	1.3 University Management System	2
	1.4 Alumni Management System	2
	1.5 Technical Architecture and Implementation	3
	1.6 User Interface and Experience	4
	1.7 Development Process and Collaboration	4
2.	LITERATURE SURVEY	5
3.	RESEARCH GAPS OF EXISTING METHODS	8
4.	PROPOSED METHODOLOGY	11
	4.1 Requirement Analysis and Scope Definition	11
	4.2 Technology Stack	11
	4.3 Modular Role Based Architecture	14
	4.4 Database Design and Data Modeling	15
	4.5 Layered Application Design	16
	4.6 Backend Development and Business Logic	17
	4.7 Frontend Implementation and UI Design	18
5.	OBJECTIVES	20
6.	SYSTEM DESIGN AND IMPLEMENTATION	22

	6.1 System Architecture and Layers	23
	6.2 System Implementation	24
7.	TIMELINE FOR EXECUTION OF PROJECT	35
8.	OUTCOMES	36
	8.1 Introduction	36
	8.2 Achievements	36
	8.3 Educational Outcomes	37
	8.4 Future Scope	37
9.	RESULTS AND DISCUSSION	38
10.	CONCLUSION	40
11.	REFERENCES	42
12.	APPENDIX – A PSEUDOCODE	44
13.	APPENDIX – B SCREENSHOTS	61
14.	APPENDIX – C ENCLOSURES	67

Chapter 1

INTRODUCTION

1.1 Background Overview

In today's fast-growing digital environment where everything is changing due to fast-paced technological growth, simply having theoretical knowledge will not be enough to ensure students are ready to be successful in the IT industry. It's time for real-world experience, practical problem solving, and knowledge of tools and frameworks relevant in the industry. For this reason, Tech Mahindra, a global leader in digital transformation and IT consulting, has developed a multi-faceted on campus technical training plan with Presidency University for the students they selected on campus the prior year.

Tech Mahindra, with headquarters in Pune, India, as part of the well-known Mahindra Group formed in 1945, is recognized as one of the most respected IT service companies in the world. Tech Mahindra has over a thousand clients all over the globe, in more than 90 countries, including Fortune 500 clients at the highest levels. They continue to invest in the key industries of telecommunications, healthcare, banking, manufacturing, and retail, while innovating with current and evolving technology such as Artificial Intelligence, Machine Learning, Blockchain, 5G, and Cloud Computing. This training program represents Tech Mahindra's mission to build a "Connected World" for the future by developing and inspiring the future generation of IT professionals by working collaboratively with educational institutions.

The program at Presidency University lasted several months beginning in February 2025 and provided students with an active learning experience that bridged the gap between theory taught in the classroom and practice used in industry. Over 90 students joined in the event, forming a total of 17 teams, where they developed a functional module of a large-scale University Management System (UMS) — a complete enterprise-level web application to digitize university operations.

1.2 Training Program Overview

The training was designed to make proficient full stack developers leveraging a complete technology stack to industry practices as closely as possible. The training's content was developed by Tech Mahindra experts, and delivered by faculty from Presidency University. At the same time, the training emphasized hands-on development, collaborations, and best

practices in software engineering.

Each student learned Core Java, Spring Boot, JDBC, and JSP for back-end technologies; and HTML, CSS, JavaScript, and React for the front-end web development. The students also learned some experience of implementing relational databases with MySQL, and putting their applications on the Apache Tomcat server. The course emphasized application design security, and the students implemented JWT-based authentication to control roles and protect sensitive data.

In addition to the technology learning, the training courses covered other important skills for students to develop such as critical thinking, debugging, team coordination, version control with Git, and a lean agile software development process. Lastly, the training was sequential and reflected a project workflow normally seen in industry (with groups of students assigned roles and responsibilities and deadlines) which helped establish individual accountability and produce some team synergy with students.

1.3 University Management System: An Enterprise Application

The UMS is an all-in-one solution aimed at automating and integrating different functionality areas within a university from administrative to academic processes. The system is composed of several interrelated modules like Student Management, Attendance, Fees Management, Timetable Scheduling, and Alumni Management, to name a few, and was created to ease data transfer and promote effective functioning across various units.

Each group was in charge of implementing one module as a mini-project which formed part of the bigger application. This makes it possible to have a glimpse of a broader enterprise software development life cycle, whereby teams worked independently on their distributed modules, complete with testing and documentation, before merging into the main system.

1.4 Alumni Management System: Our Module

Our group undertook the service of designing the Alumni Management System which deals with the maintenance of the database of the alumni's detail and post graduation relations with program beneficiaries. The module was designed as a feature rich, interactive web application with secure role based access control and user friendly UI along with responsive Design and real-time CRUD (Create, Read, Update, Delete) functionalities.

The system outlined two broad categories of users. The first was Admin and Alumni. The Admin role covered the staff members of the university in charge of managing and maintaining

all records concerning alumni relations and communications. The alumni record admin could create, edit and delete alumni accounts and would also be able to manage relevant asset by posting account community event advertisement. Furthermore, the admin had the ability to contact the alumni directly which fostered proactive engagement and outreach.

On the other hand, the alumni users could also login into the system secure in the knowledge that their profiles were protected, update their professional and personal details and register for university activities and events. They could access the profiles of other alumni which helped in networking and collaborating with them. Communication with the admin to get assistance or queries was also possible. The reciprocal communication channel was indispensable in revitalizing consolidated networks and integrating them into the system feature rich interactive alumni network.

1.5 Technical Architecture and Implementation

The widespread Spring Boot framework was leveraged to create the Alumni Management System with its Java ecosystem due to its high speed of development and modular design. To implement the system, we designed the application according to the Model-View-Controller (MVC) architecture. This approach divides the application into sections: the Model, which is the data layer involving entity classes and databases, View, UI templates, and Controller receives requests from the client and returns appropriate responses.

The frontend templates were constructed utilizing Thymeleaf which is a Java server-side template engine that builds dynamic HTML pages by embedding data from the backend into the pages. This approach enabled us to create rich web applications that are responsive and data-centric without excessive client-side scripting and provided a clear and organized code structure. MySQL database was used and to facilitate it, Spring Data repositories were utilized. These were coupled with custom defined service classes that included all of the logic for the system. A normalized database structure was constructed containing alum's details, events, and users credential. In addition to this, Spring Security enforced role-based access control to pages where users interacted with the application.

The system provided complete CRUD functionality: the admin could manage alumni records and event posting, the alumni could edit their profile and register for events in real time. Forms were being validated client-side and server-side to verify data integrity and verify user input. Session management and security features ensured unauthorized access was not permitted for sensitive personal data.

1.6 User Interface and Experience

One of our key priorities was to deliver a user interface (UI) that is functional and intuitive, as well as responsive across various devices for a variety of campus constituents. The proposed UI included pages for the login, admin dashboard, alumni profile, events listing, and contact form. The design considered best practices regarding usability and accessibility, including clear navigation menus, responsive layouts for various screen sizes, and consistent styling to elicit a professional and cohesive vibe. Role-specific dashboards provided uniquely informative content: admin role can manage alumni and events with specialized tools, while an alumni user can view their profile and events as they pertain to that user. To create a more interactive experience, we included real-time feedback for the user through form validation messages and real-time updates while feasible to reduce page loads to only necessary interactions and improve responsiveness. The system can log audits of this nature, and provide forms of error reporting, to facilitate troubleshooting and ensure robust operational capability.

1.7 Development Process and Collaboration

Module development of the Alumni Management System was collaborative. We used GitHub as a version control system, which allowed issues to be tracked and for multiple developers to work together simultaneously. Utilizing GitHub allowed for code reviews, and features to be added via pull requests. In addition, weekly meetings of our team allowed us to monitor our progress on the project, resolve blockers, and prioritize future features. We emulated a proper workflow in a company, starting with identifying requirements, then moving on to design, implementation, testing, and deployment. Testing consisted of unit tests and integration tests to verify that individual components worked, that they operated independently on their own, and that they worked together as an application. Resources and ongoing, continuous feedback from our faculty mentors provided direction to make our implementation as best aligned with the intended scope and direction of the project, and the standards of professional coding practices. One of the valuable take-aways from contributing as one team against an enterprise level module on a global distributed project is the exposure to important lessons in communication, modularization, and agency in integration of code. We also gained valuable experience managing dependencies in our code, managing merge conflicts, and communicating with other teams--all necessary skills that you will encounter in the real world as a software developer.

Chapter 2

LITERATURE SURVEY

An effective Alumni Management System (AMS) is crucial for fostering engagement between an institution and its graduates. Over the years, technological advancements have enabled the evolution of AMS from rudimentary, manually operated systems to robust, web-based platforms with role-specific access, centralized databases, and real-time communication features. The reviewed literature offers a rich insight into various architectures, functionalities, and implementation strategies employed in different AMS projects. One of the consistent trends across recent systems is centralized and accessible data management. As noted in [1], "The database management system is a collection of particular data depending on the requirement. It is easier to alter and implement changes in the existing data." This approach is echoed in [2], which states, "The Alumni Database is a web database that can be viewed from anywhere in the world," emphasizing the importance of global access to information. Furthermore, [6] highlights that centralized systems improve operational speed and allow easy retrieval of records through automated forms.

Several studies also acknowledge the value of remote access and responsive design. For instance, [1] mentions, "As the new software technology is rising, we can store this data conventionally and effectively... that can be accessed remotely," while [3] reinforces this by stating, "We can store this data in a conventional and effective way... which can be accessed remotely." Such features empower users to access the platform across devices and geographic locations, enhancing the flexibility and efficiency of the AMS. Beyond accessibility, many systems incorporate two-way communication and feedback mechanisms. According to [1], "The system provides various communication channels, including email, newsletters, and discussion forums, allowing administrators and alumni to stay connected." This is expanded in [2], where alumni can engage in knowledge transfer by "sharing valuable insights and expertise with current students through note-sharing and discussion forums." Additionally, systems include feedback modules where both students and alumni can submit suggestions or concerns, which are reviewed by administrators for continuous improvement [4].

Another valuable innovation is the integration of social networking APIs. A standout example is the LinkedIn API integration discussed in [1], which notes, "There is no need to update the

Alumni Profile as it can be updated automatically when the alumni update their LinkedIn profile.” Such features reduce manual overhead, increase data accuracy, and enable alumni to maintain up-to-date profiles without re-engaging with the portal directly.

Security and privacy are essential components of any alumni system. As [1] highlights, “Users can determine what information they want to share and also whom they want to share it with.” This attention to privacy is supplemented by technical implementations such as encrypted authentication in [5], which describes a mechanism resistant to Man-in-the-Middle Attacks (MIMA). Furthermore, the systems described in [6] are hosted in private subnets with no public access, ensuring a secure data environment.

Most systems adopt a role-based architecture. For instance, [2] outlines separate modules for admin, alumni, and students, each with distinct responsibilities such as managing profiles, uploading notifications, or updating job postings. Similarly, [4] describes a 7-module structure covering functions from registration to feedback and gallery management, allowing fine-grained control over platform access and operations. These structured modules are designed to simplify administrative workflows and improve user autonomy.

From a technical standpoint, frontend and backend integration is essential. Technologies such as HTML, CSS, and JavaScript are commonly used for the frontend [3], while backend processes often rely on PHP, Django, or JavaScript frameworks [3], [6]. The importance of database design is emphasized across studies. For example, [6] states, “The database is designed to maintain all the ACID properties” and uses relational databases for secure and normalized storage. In [5], the system's data layer is designed to roll back and maintain consistency in case of transaction errors, showcasing the robustness of the backend architecture. AMS platforms also incorporate automated transitions and lifecycle management. In [6], for instance, alumni are transitioned automatically from student modules, enabling a seamless flow of data post-graduation. This ensures that the system remains up to date and reduces manual administrative tasks, particularly during convocation and graduation cycles.

Functionality-wise, systems offer job posting, event management, mentorship, and fundraising capabilities. According to [4], “The alumni can also get the opportunity of mentoring the current students,” while [6] supports academic integration: “Access to the system can help them in building connections to their projects or for placements.” Additionally, event management modules allow administrators to share notifications for reunions, conferences, and webinars,

increasing alumni participation in campus activities [2][3].

The multi-functionality and emergency preparedness of these platforms are also highlighted. [2] states, “Thanks to this system, our department can easily obtain information about students in emergencies.” This demonstrates the wide-ranging utility of AMS beyond mere alumni tracking, extending into operational and institutional support during critical scenarios. Lastly, the strategic importance of AMS for institutional development and accreditation is recognized. [6] notes that “It will also be important to the NBA (National Board of Accreditation) activities,” underlining the broader administrative and regulatory relevance. Active alumni participation strengthens institutional networks, contributes to fundraising, and bolsters placement and mentorship programs—elements critical to institutional visibility and growth.

Chapter 3

RESEARCH GAPS OF EXISTING METHODS

Despite the growing emphasis on digital infrastructure in academic institutions, existing Alumni Management Systems (AMS) still exhibit critical shortcomings that hinder their full potential. While numerous solutions have been proposed in recent years, a review of current literature reveals several persistent gaps in functionality, accessibility, scalability, and user engagement. These gaps highlight the need for more integrated, secure, and interactive systems that not only serve as static data repositories but also facilitate dynamic, real-time collaboration among alumni, students, and administrators.

One of the most common limitations is the continued reliance on semi-automated or manual data handling processes. Several studies report that older systems used Excel sheets or cloud-based documents for record-keeping, which fail to support multiple users or provide adequate security. For instance, Kumar et al. state that “the existing system is a computerized system but which is maintained at individual databases i.e., in excel sheets... [and] doesn’t provide multiple user accessibility” [6]. This manual structure results in delays in record retrieval, risk of data inconsistency, and significant administrative burden, particularly when alumni data needs to be updated frequently or shared securely across departments.

Moreover, many platforms lack true two-way communication. Lavanya et al. (Juni Khyat) point out that “graduated class and understudy can impart just through the administrator consent” [5], indicating that alumni-student interaction is often constrained by outdated communication models. This limitation suppresses the spontaneous mentorship, networking, and career guidance opportunities that alumni could offer to current students. Similarly, Lavanya et al. (IJCRT) observe that current systems provide “only one-way communication and... have little room for future activities and interaction between the members” [1]. In an era where interactivity is central to user engagement, such design constraints significantly weaken the functional value of AMS platforms.

Another recurring issue is the lack of mobile accessibility and responsive design. Radhika et al. acknowledge that “using this system on mobile devices isn't smooth,” limiting access for a growing user base that primarily interacts with systems through smartphones or tablets [4].

Without mobile optimization, institutions risk alienating users and reducing overall engagement—especially among younger alumni who expect digital platforms to be mobile-first and fully functional on-the-go. Security and privacy also present significant concerns. Sawai et al. note that many current systems “suffer from usability issues, security concerns, and limited scalability” [2]. This exposes sensitive alumni data to potential threats and limits the trustworthiness of the system. While some systems have implemented role-based access, full encryption and compliance with modern security standards are inconsistently applied or entirely absent. The absence of standardized privacy controls is further emphasized by Lavanya et al. (IJCRT), who contrast this with the advantages of LinkedIn-based systems where “users can determine what information they want to share and also whom they want to share it with” [1]. This suggests that privacy, though recognized as critical, is still under-implemented in many AMS designs.

Additionally, the lack of integrated social media connectivity poses a major engagement barrier. The study by Radhika et al. clearly identifies that “the system doesn't link well with popular social media platforms,” which reduces alumni visibility and limits interactive reach [4]. Social media integration is increasingly recognized as essential for networking and information dissemination, and its absence prevents alumni systems from evolving into comprehensive digital communities. Scalability and extensibility are also underdeveloped in many of the systems reviewed. Sawai et al. mention that “institutions often struggle to maintain effective communication and engagement with their alumni, leading to fragmented alumni networks and missed opportunities” [2]. This fragmentation is partially a result of legacy systems that do not scale well with growing user databases or evolving institutional goals. The inability to modularly expand features or integrate with third-party services like job portals, academic tracking tools, or analytics dashboards limits the long-term utility of current systems.

Furthermore, some systems do not include advanced features like searchable alumni directories or smart filters. As noted by Radhika et al., “without an alumni directory, it's hard for students and administrators to find and connect with other alumni effectively” [4]. This restricts discovery and networking capabilities, which are among the core functions of any AMS. The lack of such features diminishes the user experience and reduces the value proposition of the platform for both administrators and users.

Lastly, although several papers mention event and job posting features, these functionalities are often manually managed by admins without automation or real-time syncing. Ved et al. report that even in modern systems, “event and job-related data must be updated manually,” which increases workload and the risk of outdated or missing information [3]. Automation of these processes could greatly enhance efficiency and ensure timely delivery of relevant updates to all stakeholders.

In summary, while significant progress has been made in the development of Alumni Management Systems, research reveals clear and actionable gaps: continued reliance on manual data entry, poor communication design, insufficient mobile access, underdeveloped privacy controls, lack of social media integration, weak scalability, and absence of advanced search and automation features. These limitations serve as a foundation for proposing an improved system that addresses these critical shortcomings through robust architecture, enhanced interactivity, and user-centered design.

Chapter 4

PROPOSED METHODOLOGY

4.1 Requirement Analysis and Scope Definition

The development of the Alumni Management System was triggered by the desire to maximize efficiency with a centralized, web-based entity for data management, event coordination, and communication to and from alumni. After a requirement analysis phase, we confirmed user needs and system objectives. We consulted key stakeholders, including administrators, previous students and technical staff using informal interviews and established paper workflows arrangements. Based on the analysis, the major functional requirements identified included secure authentication, role-based access, alumni registration and verification, event/job posting by admins, and alumni profile management. Non-functional requirements emphasized system security, scalability, and ease of use across devices. The scope was documented in a functional specification that served as the foundation for design and development.

4.2 Technology Stack

The Alumni Management System was developed using a comprehensive full-stack architecture with a strong emphasis on modularity, scalability, and secure access. A combination of backend and frontend technologies, database systems, and development tools were integrated to streamline the development workflow and improve system performance. The project followed the Model-View-Controller (MVC) design pattern to separate data handling, business logic, and user interface layers. Below is a detailed list of the technologies and tools used in the system and their respective roles:

4.2.1 Technologies

Technologies refer to the programming languages, frameworks, libraries, and frontend utilities that form the technical foundation of the application. These are responsible for implementing the application's logic, user interface, and interaction with data.

- **Java:** The primary programming language used in the backend to implement the application's business logic, service classes, controllers, and data models. Java's object-oriented principles and strong type system ensured code maintainability and scalability.
- **Spring Boot:** A powerful and widely adopted Java-based framework that simplifies the setup and development of web applications. It allowed for rapid prototyping by providing

embedded server support (Tomcat), auto-configuration, and starter templates for seamless integration of dependencies.

- **Spring MVC:** This module of the Spring Framework was used to build the web layer. It handles HTTP requests, maps URLs to controller methods, and bridges frontend forms with backend services using the Model-View-Controller (MVC) pattern. This architectural approach enhanced code modularity and separation of concerns.
- **Spring Security:** Implemented for authentication and access control, this module ensured that user roles (such as Admin and Alumni) had well-defined privileges. It enabled secure login, password encryption (typically using BCrypt), and session management, providing a strong defense against unauthorized access.
- **Java Persistence API (JPA) with Hibernate:** JPA allowed for object-relational mapping (ORM), simplifying the interaction between Java objects and relational database tables. Hibernate, the default JPA implementation in Spring Boot, generated database schemas automatically and translated method calls into SQL queries, improving developer productivity and reducing human error in SQL writing.
- **HTML5:** Used to create the structure and content of the web pages. It provided semantic elements that helped organize content logically and ensured accessibility across different devices and screen readers.
- **CSS3:** Responsible for styling the HTML content. It allowed customization of page layouts, color schemes, fonts, and responsiveness, contributing significantly to the visual design and user experience.
- **JavaScript:** Enabled client-side scripting to enhance user interaction. It was used for form validation, real-time field feedback, interactive elements like modals and dropdowns, and basic AJAX-like behavior to improve page responsiveness.
- **Thymeleaf:** A Java-based server-side templating engine integrated with Spring Boot. It was used to dynamically generate HTML pages by binding backend data directly to the view, making it easier to maintain logic within the HTML structure.
- **Bootstrap:** A frontend framework used in combination with HTML and CSS to build responsive, mobile-friendly interfaces. Bootstrap's component library—comprising buttons, modals, alerts, cards, forms, and grids—helped accelerate frontend development and maintain design consistency.

4.2.2 Platforms

Platforms refer to the software and systems upon which the application is hosted, run, and accessed. These form the operational environment for the execution and deployment of the system.

- **MySQL:** A widely-used relational database management system (RDBMS) chosen for its stability, performance, and integration with Java-based applications. MySQL was used to store all critical system data including alumni records, user credentials, event details, and messages. The database schema was normalized to reduce redundancy and enforce data integrity through foreign keys and indexing.
- **Apache Tomcat (Embedded):** Bundled with Spring Boot, the embedded Tomcat server acted as the servlet container responsible for processing HTTP requests and serving web content. Its embedded nature allowed for easier packaging and deployment of the application as a self-contained unit.
- **GitHub:** Functioned as the centralized version control and collaboration platform. It stored the codebase, tracked changes, and enabled coordination among team members through pull requests, issue tracking, and code reviews. It also served as a backup and deployment-ready repository.
- **Web Browser (Google Chrome):** The application was primarily accessed through modern web browsers like Google Chrome, which rendered the frontend content and served as the interface for user interaction.

4.2.3 Tools

Tools refer to the software that was used during the development, debugging, testing, and deployment phases. They allowed for efficient coding practices, interaction with the database, dependency management, and inspection of frontend code.

- **Spring Tool Suite (STS):** An Eclipse-based integrated development environment for the development of Spring-based applications. STS had advanced features that made it easier to code such as the Spring Initializer, live templates, auto-complete for the various Spring annotations, as well as integration with Maven and Git, which greatly improved the efficiency of developers.
- **Maven:** A build automation and dependency management tool, which was used to manage third-party libraries and plugins. Maven performed dependency management as it allowed for consistent builds as it would automatically resolve and download dependencies in the pom.xml file.

- **MySQL Command-Line Client:** It was used to directly interface with the database during development and debugging. Developers were able to use SQL commands to create schemas, insert test data, view tables, and manage users and privileges.
- **Chrome Developer Tools:** A set of debugging tools built into the Chrome, browser. These are used mainly to view HTML elements, style elements (using CSS), monitor network requests, and debug Javascript behavior. These tools were essential for frontend development and testing for responsiveness.
- **Git:** A distributed version control system used locally to track changes to a source code repository. git was paired with GitHub to track commits, branches, merges and maintain version history.
- **Postman (optional):** A tool for testing an API calling client's requests to backend endpoints in a way that helps a developer verify that the HTTP method, HTTP headers, parameters and payload are being processed correctly.

By leveraging this rich and coherent technology stack, the development team was able to implement a scalable, secure, and user-friendly alumni management system. The combination of proven backend frameworks, dynamic frontend technologies, and powerful development tools provided the necessary infrastructure to meet project goals and prepare for future enhancements.

4.3 Modular Role-Based Architecture

The Alumni Management System is architected using a modular, role-based access control (RBAC) model, which ensures that each user interacts with the system strictly within the boundaries of their assigned role. This model promotes data security, operational clarity, and maintainable code, while offering a flexible foundation to scale user roles and privileges in the future.

Two primary user roles are defined in the system:

1. Administrator (Admin)

The Admin role has full privileges and system-wide control. Admins are typically institutional staff or system operators responsible for managing core operations. Their key responsibilities include:

- Creating, editing, and deleting event entries (e.g., reunions, seminars, guest lectures).
- Sending announcements or updates via a broadcast system.

- Viewing and managing alumni records in a searchable, filterable format.
- Potentially analyzing engagement statistics or generating reports (planned feature).

This role is implemented with elevated access rights, and routes/actions associated with admin privileges are protected using Spring Security's `@PreAuthorize`, `hasRole("ADMIN")`, and configuration in the `SecurityConfig.java` file.

2. Alumni

The Alumni role is assigned to authenticated former students who have completed the registration process and received admin approval. Their privileges are more limited and focused on personal interaction and content consumption. Alumni can:

- Register an account through the signup form (subject to admin approval).
- Log in and access a personalized dashboard.
- Update their profile information, such as name, contact details, education, and professional status.
- View upcoming institutional events, job openings, and announcements posted by admins.
- Participate in potential future modules like feedback submission, mentorship offerings, and donation tracking.

The actions available to alumni are carefully scoped to protect institutional data while enabling meaningful participation. Access is restricted using `hasRole("ALUMNI")` configurations, and alumni cannot access any administrative endpoints or modify system-wide content.

4.4 Database Design and Data Modeling

The database for the Alumni Management System was designed using a relational data model with normalization principles to ensure data integrity, consistency, and efficient access. The design focuses on mapping real-world entities such as users, alumni, events, and interactions into structured database tables. The schema was developed in MySQL and supports role-based access, event participation tracking, and communication workflows within the system.

The primary tables in the database include:

- **admin:** This table stores administrative users' login credentials and profile information. Admins have the highest privileges in the system, including the ability to verify alumni, post jobs, manage events, and view system logs.
- **alumni:** This is the core table that contains the details of registered alumni. Fields include name, contact information, educational background, graduation year, and current

employment. Each alumni record is associated with a university (via a static field or assumed foreign relationship).

- **alumni_event:** This table represents the events organized by the institution or the alumni association. Fields typically include the event name, date, location, description, and organizer. Events are created and managed by admins and are visible to approved alumni through the frontend dashboard.
- **event_registration:** This table handles the many-to-many relationship between alumni and events. It records which alumni have registered for or attended which events. Each entry in this table links a specific alumni ID to an event ID, facilitating tracking of participation and engagement metrics.
- **contact_message:** This table stores messages submitted via the contact or inquiry forms on the website. It includes sender information, the subject of the inquiry, message content, and timestamps. It allows administrators to monitor incoming queries and respond appropriately.
- **activity_log:** This table maintains a log of system activities for auditing purposes. Logged actions might include user logins, profile updates, admin actions like alumni approval, or event creation. It provides a chronological record of system-level interactions, aiding in monitoring and debugging.

The database schema was built using Java Persistence API (JPA) annotations in Spring Boot. This allowed automatic generation of database tables from Java entity classes and significantly reduced the need for manual SQL scripting. Validation rules, such as field lengths, uniqueness, and not-null constraints, were enforced both at the database and application levels to maintain robust and clean data. The data model is designed to be scalable and modular, allowing for future enhancements such as job postings, alumni donations, mentoring programs, and multi-institution support without restructuring the core schema.

4.5 Layered Application Design

Layered Application Design

The Alumni Management System was created an architecture that uses a layered architecture based on Model-View-Controller (MVC) to segment the application into components, each with a distinct responsibility. This design helps to keep the system organized, maintainable, easy to test, and scalable in the future.

The architecture will be described in terms of four layers:

- **Controller Layer:** This layer is responsible for accepting incoming HTTP requests and passing the requests to services, it acts as the connection between the front end and back end, since it accepts user inputs, validates them, and prepares responses to user actions.
- **Service Layer:** This layer uses the business logic of the application. It processes and applies rules to the data entered by the user (for example, user roles and business rules about event registration) and manages the connection between the controller layer and the repository layer.
- **Repository Layer:** This layer is responsible for all database operations, which is implemented using the Spring Data JPA abstraction. The repository uses a MySQL database with a general structure that organizes entity classes and repositories to allow CRUD operations and customized queries, but without requiring writing SQL to connect with the database directly.
- **View Layer:** This layer exists as a set of micro templates generated with Thymeleaf. The view layer allows for dynamic web pages to be rendered and displayed based on the data from the controllers. The views use HTML and CSS to provide a user interface that is enhanced with Bootstrap styles and control form submission by the user to be submitted to the controller.

By clearly separating concerns across these layers, the system remains well-organized and adaptable, supporting efficient development and the potential for future enhancements such as API integration or role expansion.

4.6 Backend Development and Business Logic

The backend of the Alumni Management System is built with Spring Boot, which has organized functionality for various components such as authentication, user management, data processing, and persistence/database transaction. The alumni system is designed around a multi-layered architecture. All incoming HTTP requests are directed to controller classes. The controllers delegate to the service layer, which handles business logic. Services utilize the repository layer, using Spring Data JPA to perform CRUD operations on a MySQL database.

Spring Security is responsible for security and access control in the backend. The system utilizes session-based login, user-roles, and ability to manage access authorization using both annotations and configuration settings. User accounts are assigned roles (Admin, Alumni),

which controls the access to protected resources. User passwords are hashed using BCryptPasswordEncoder prior to storing them in the database. Ultimately, this system is session-based for authentication and access management, as opposed to a stateless API driven aspect. The project does not currently utilize JWT authentication, but it could in the future, if chosen. The backend handles key workflows including user registration for alumni, admin approval of all new users, event creation, and profile editing. All workflows are captured in service classes (along with their associated business logic), allowing services to preserve purpose (concerns) and be maintained. Data validation, and error handling, are performed consistently across all endpoints throughout the backend, in order to provide reliable service to users and actionable feedback when errors are present.

Overall, the backend is designed for clarity, security, and extensibility, ensuring that each component interacts seamlessly while allowing room for future enhancements such as RESTful APIs, third-party integrations, or analytics modules.

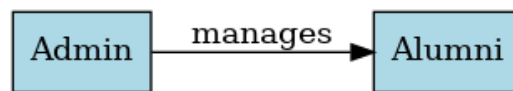


Figure 4.1 Entity Relationship Diagram

4.7 Frontend Implementation and UI Design

The Alumni Management System's front end was built on HTML5, CSS3, JavaScript, and Thymeleaf to provide a clear, responsive, and interactive interface for desktop and mobile devices. Thymeleaf provided a way to dynamically render server-side data in an HTML template. JavaScript was used to offer interactive features, such as form validation and confirmation windows with dropdowns. We styled the interface with most of our CSS written from scratch, and we used responsive design to make it easier to use on the desktop and mobile devices.

A detailed user experience was designed for each type of user, administrator, and alumni user. Admin users have full controls where they have the ability to manage alumni registrations to create and edit events, post jobs, view alumni data by department or graduation cohort, and send announcements from the system. The alumni-user experience aimed to keep the page simple and easy to access. Once the alumni user registered and had their registration approved by the admin user, the alumni user logs into their dashboard to update their profile, find events, post jobs, and receive updates on institutional activity. While the last version of the Alumni

Management System included the most basic version of profile updates and events, additional user experience extensions will be added to include mentorship, feedback, and interactive engagement tools.

Overall, the frontend was designed with a clear separation of user roles, offering targeted functionality and an intuitive interface that supports both administrative operations and alumni engagement.

Chapter 5

OBJECTIVES

The main objective of this project is to design and implement a complete Alumni Management System that connects educational institutions and alumni. This system will not only contain basic data storage but will include a way for alumni to engage meaningfully, handle data in a secure and reliable way, and help with administrative control and actions. What follows are the main objectives that guided the development of this alumni management system, each corresponding to a functional aspect, usability, and scalability.

1. To develop a centralized and secure platform for managing alumni data and institutional engagement:

A primary goal of this project is to modernize the alumni record-keeping process from legacy, decentralized systems (spreadsheets and paper) to a web-based centralized database. This system, built in Spring Boot with MySQL, allows alumni data to be recorded in an ordered, searchable, and secure way. The system also allows for role-based access so that only authenticated users can access and use the system, placing severe restrictions on unnecessary changes or data breaches. Admins can support the continual upkeep of records, monitor alumni interaction and manage communications as responsibilities can be allocated to a smaller group.

2. To implement a scalable, modular architecture that supports future functional enhancements:

This goal has to do with understanding structural design, a major part of the system's architecture. A flexible architecture allows for maintainability by using various extents of a separation of concerns (e.g. using an MVC architecture to keep backend logic easy to manage and reusable). The system is built with Spring Boot, Spring Security, and JPA. The application can be extended for alumni donations to add features for tracking, mentoring programs, third-party authentications (e.g. Google OAuth), and social network sites (e.g. LinkedIn). Using layered architecture supports the idea of isolated testing and allows the team to deploy updates clearly without impacting existing functionality.

3. To deliver an intuitive, role-specific user interface that encourages interaction and self-service:

The project seeks to develop a simple-to-use front end that will support HTML5, CSS3, JavaScript, and Thymeleaf templates for two different users: administrators and alumni users. The alumni user interface will allow alumni to register, manage their profile, and have access to job postings and information about events. The admin user interface will allow admin users to approve registrations, post announcements, manage events, and consult on their alumni. Since both roles will only be able to use the tools that are specifically designed for them, it will lead to a more straightforward experience and less confusion, thus leading to less user-error. Since the project has a responsive design, it is anticipated that alumni will use the system on all devices and encourage them to stay involved and connected over time.

Chapter 6

SYSTEM DESIGN & IMPLEMENTATION

The Alumni Management System is built using the Spring Boot framework and follows the Model-View-Controller (MVC) architecture to enable clear separation of concerns, scalability, maintainability, and ease of development. The application supports two user roles: Admin and Alumni, each having different access levels and functionalities. The system allows admins to manage alumni profiles and events, while alumni can update their profiles, view events, and communicate with the admin. All the application data is persistently stored in a MySQL database, and the user interface is built using HTML, Thymeleaf, and CSS, optionally enhanced with Bootstrap for responsiveness.

Key Components of the Alumni Management System

1. **User Management Module:** Manages user roles (Admin and Alumni), secure logins, registration, and role-based access control.
2. **Alumni Profile Management:** Allows alumni to view, edit, and manage their personal and professional information.
3. **Event Management Module:** Enables admins to create and manage events, while alumni can view and register for them.
4. **Messaging System:** Facilitates two-way communication between alumni and administrators within the platform.
5. **Activity Logging System:** Tracks admin actions such as profile updates and event creation for auditing purposes.
6. **Public Alumni Directory:** Displays verified alumni profiles publicly to promote engagement and network visibility.
7. **Responsive User Interface:** Offers a clean, intuitive, and mobile-friendly UI tailored for both alumni and admin roles.
8. **Database System:** Uses a normalized MySQL database to securely store user credentials, events, messages, and logs.
9. **Modular and Layered Architecture:** Follows the MVC design pattern using Spring Boot to ensure separation of concerns.

6.1 System Architecture and Layers

The project is built on a layered architecture, which comprises:

- a) **Controller Layer** – This layer lies at the core of the system which acts as the primary bridge between the user interface and backend logic. Controllers such as AdminController, AlumniController, and HomeController receive HTTP requests from users and route them to appropriate service methods. For example, when an admin accesses the dashboard via the /admin/dashboard endpoint, the corresponding controller method gathers relevant alumni and event data, prepares it, and forwards it to the view. This layer is annotated using @Controller and handles routing with @RequestMapping and @GetMapping annotations.
- b) **Service Layer** - This contains the core business logic and acts as the system's decision-maker. It abstracts the logic away from the controllers, promoting modularity. Services such as AdminServiceImpl and AlumniServiceImpl contain methods for operations like registering alumni, verifying logins, managing events, or updating profiles. For instance, when an alumni user edits their profile, the controller passes the form data to the updateAlumni() service method, which then performs validation and calls the repository to persist changes.
- c) **Repository Layer** – This layer uses Spring Data JPA to handle all database interactions. Interfaces like AlumniRepository, AdminRepository, and EventRepository extend JpaRepository, giving access to built-in CRUD operations without writing raw SQL. Where needed, custom query methods are defined using method naming conventions, such as findByEmailAndPassword() for login validation. This layer directly interacts with the MySQL database, performing operations like fetching alumni profiles, saving new events, or deleting records.
- d) **Model Layer** defines the data structure of the system using Java classes annotated with JPA annotations such as @Entity, @Table, @Id. These classes—Alumni, Admin, Event, Contact Message, and Activity Logs—map directly to database tables. For example, the Event entity has fields like title, description, event, date, and is linked to registrations and admins via foreign keys, enabling efficient tracking of event participation and ownership.
- e) **View Layer** - is managed via Thymeleaf templates, which are located in the templates/ folder. These templates are dynamic HTML pages that render data from controllers, and support user interaction such as form submission and appearance of real-time features on pages. Thymeleaf syntax (e.g., th:each, th:value, th:if) is used to bind backend data to page elements like dropdowns, tables, and input elements. Each page, like

alumni_dashboard.html, or admin_event_list.html, is role-based and provides access to content so that user roles can only view what they are allowed to see.

This layered structure is further complemented with Spring Security, which enforces role-based access control in which users have certain authorities associated with each user role (Admin or Alumni). The configuration file SecurityConfig.java restricts access to routes through `http.authorizeRequests()` and allows for a login feature using Spring's form-based authentication. Passwords are encrypted with `BCryptPasswordEncoder` adding another layer of security to credential management.

Overall, this architecture establishes a well-organized and modular web application, with proper security. The web application is set up to easily extend with new features like Job boards, mentorship programs, or API-based integrations, and it fits the real-world enterprise code structures allowing for contributors to gain experience working with scalable full stack development.

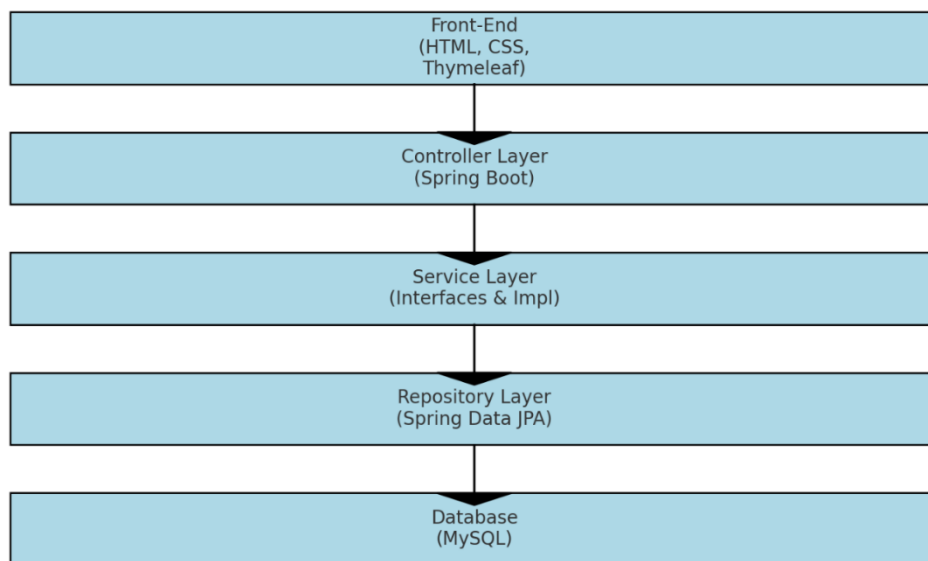


Figure 6.1 Layered Architecture

6.2 System Implementation

a) Admin Module

The Alumni Management System's Admin module allows total control over all data and user engagement activities. admin is the superuser for the system, managing alumni records, creating events, and sending meaningful messages to alumni.

The admin journey begins with a login. Credentials are checked against the admin table of the

admin database. The login form is typically at /admin/login. The form submits to a controller endpoint that checks the email and password using the AdminService. Spring Security manages admin sessions to ensure that only administrators can access admin routes like the dashboard. Passwords are encrypted using BCryptPasswordEncoder. After logging in, admins are routed to an Admin Dashboard. The Admin Dashboard is a central location for all administrative functions; from here admins can see statistics and lists of registered alumni, and perform actions like editing or deleting profiles. An alumni record is populated to each profile card using Thymeleaf, with respective links to "Edit", "Delete". For example, the /admin/edit/{id} route will trigger a method in the AdminController that gets the alumni record by its ID, loads, and submits against the record being edited.

Admins can also add new alumni directly via the "Add Alumni" interface. A form collects personal, academic, and professional information along with a profile photo, which is uploaded using Spring's MultipartFile and stored either in a static directory or linked as a path in the database. The AlumniService handles this logic, saving the data and invoking the AlumniRepository to persist it. On success, the UI provides confirmation, and the new entry appears immediately in the alumni listing.

Another key feature is the Event Management System, where admins can create, edit, and delete events such as reunions, webinars, or guest lectures. The form includes fields for event name, description, date, time, and venue. Once submitted, the event is saved in the event table and made visible to alumni on their dashboard. The admins on the site can view and manage events via routes such as /admin/events, where they can update or delete events from the UI.

In addition, the admin module contains a Messaging System to allow admins to easily commence direct contact with alumni. Admins can choose an alumni user to send a message to, compose the message, and send it through the platform. The message is saved to the messages table with the sender_id, receiver_id, subject, and created_at timestamp. This task is managed through MessageService and presented through MessageRepository when the system produces the message history or address book view for admins and alumni.

Also, to keep organization and help with debugging, the system logs all key actions performed by the admins in the Activity Log. Each time an admin adds, edits, or deletes some data, a corresponding entry is made in the activity_log table with the date and time and description of the event. The activity log entries can be accessed through the admin dashboard and function as an audit trail. Finally, admins are also able to update their own profile in the Admin Profile Management area, where they are able to change their name, their contact details, and even

their password. Any updates are processed through the AdminService and the actual admin in the admin table is updated. Similar to above, all routes are protected using role-based access in Spring Security and only users with the role "ADMIN" will have access to these Admin Profile Management features. Overall, the Admin module is robust, secure, and designed to give complete control to authorized university personnel, while ensuring data consistency, ease of use, and auditability.

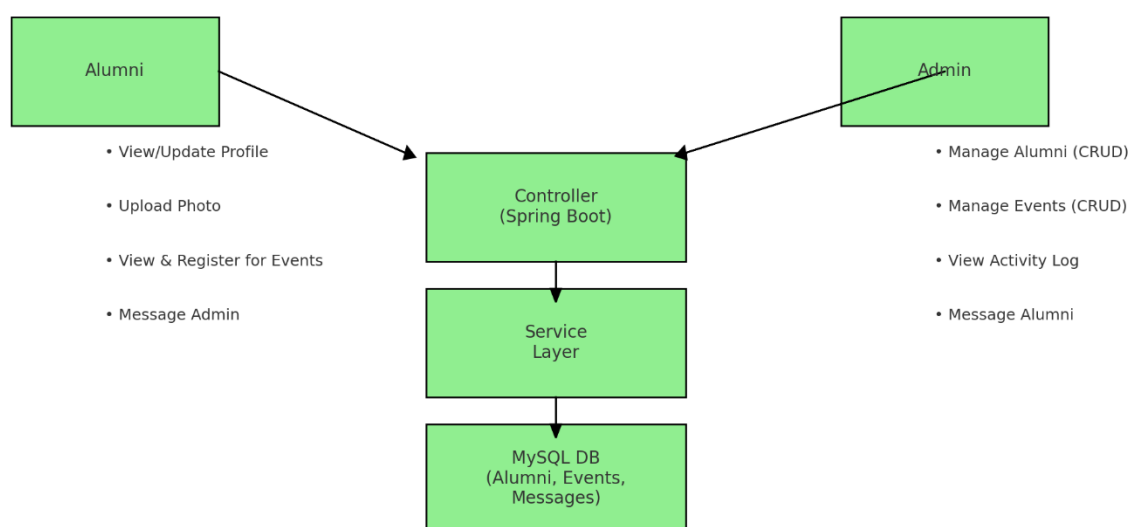


Figure 6.2 Alumni Management System Architecture

b) Alumni Module

The Alumni module of the Alumni Management System empowers former students to stay connected with their institution through profile management, event participation, and communication with administrators.

The process begins with the **registration system**, where alumni can sign up by providing their name, email, graduation year, department, current job role, and a password. The registration form is accessible without login, and once submitted, the data is captured and passed to the backend through the AlumniController. The AlumniService stores the details in the alumni table.

Once approved by an admin, the alumni can proceed to **log in** via a secure login page. Their credentials are validated using Spring Security, which checks the encrypted password against the stored value using BCryptPasswordEncoder. If authentication is successful and the alumni status is active, the user is redirected to a personalized dashboard view.

The Alumni Dashboard provides a simple, role-specific interface for alumni users. The page

displays their profile information and a list of upcoming events posted by the admin. The alumni can update their profile at any time by accessing the "Edit Profile" section. This form is pre-filled with existing data and supports uploading a new profile picture using Spring's file upload mechanism. Once updated, the new data is validated and saved to the alumni table through the `AlumniServiceImpl`.

Another major feature is event registration. Alumni can browse through events listed on their dashboard, each displayed with details like title, description, date, and venue. A "Register" button is shown for events they haven't joined yet. Clicking this button triggers a controller method that creates a new record in the registration table, establishing a many-to-many link between the alumni and the selected event. This allows the system to track participation, which can later be used for reporting or engagement analysis.

The alumni have access to a messaging system to communicate directly with the admins by way of an in-platform form. The alumni select an admin (which can be predefined or default), write a message, and submit the message. The messaging system itself is handled by the `MessageController`, which writes to the messages table with the important sender and receiver information. This allows alumni to make requests, provide feedback, or ask questions about events without needing outside communication tools.

This module has security limitations built in. Alumni are unable to access the admin routes and functionality due to Spring Security's role-based access security, as configured in the `SecurityConfig` class. Routes such as `/alumni/dashboard` can only be accessed by users with a role of 'ALUMNI'. If an illegal access is attempted, the system redirects the user to a login or generic error page. Further, the forms are validated at each level, both at the front and back ends to reduce the chances of an incorrect or malicious entry into the system.

Overall, the Alumni module is user friendly. It uses Thyemleaf, HTML5 and CSS to offer a responsive, intuitive interface. Several pages, such as `alumnipage.html` and `edit_profile.html`, are carefully created to be device agnostic and styled to facilitate navigation and readability.

c) Public Directory and Messaging System

The Public Directory is accessible without requiring login and allows any visitor to explore the profiles of verified alumni. This section acts as a digital showcase of the institution's alumni network and is designed using a responsive, card-based layout. Each card typically displays an alumni's profile picture, name, designation or job title, and graduation year. The data is dynamically fetched from the database using the `AlumniRepository` and rendered via

Thymeleaf templates like `PublicAlumniDirectory.html`. Visitors can click on individual cards to view a detailed profile page, which includes extended information such as academic background and professional history. This feature is particularly useful for prospective students, recruiters, or university officials who want to understand the scope and reach of the alumni base.

The Messaging System enables direct, two-way communication between alumni and administrators within the application. Alumni users, after logging in, can navigate to a “Contact Admin” section where they are presented with a form to compose and send messages. This form captures the subject and message body and is submitted through the `MessageController`, which handles form validation and delegates the persistence task to the `MessageService`. The messages are stored in the messages table along with metadata such as sender ID, recipient ID, timestamp, and read status.

On the admin side, a dedicated interface allows administrators to view incoming messages from alumni in a chronological table format. Admins can read, archive, or respond to messages, helping them address concerns, share information, or provide guidance in a timely manner. Each message is logged with user references to ensure traceability. The communication model promotes interaction while maintaining role boundaries—alumni can contact admins but not other alumni, ensuring focused and relevant exchanges.

Both the Public Directory and Messaging System are integrated with Spring Security to ensure appropriate data visibility and access control. While the directory is open to all users, the messaging features are strictly limited to authenticated users with the appropriate role. Additionally, the design ensures that message data cannot be tampered with or intercepted, preserving user privacy and data integrity.

d) Database Design

The Alumni Management System uses a relational database built with MySQL, designed to ensure data integrity, consistency, and scalability.

- The **admin** table stores credentials and profile information of administrative users who manage the system. It includes a primary key `admin_id`, along with a unique username, encrypted password, and optional `full_name`. This table serves as the source of authority for backend operations such as alumni verification, event creation, and message handling.
- The **alumni** table represents all former students who have registered in the system. Each entry is uniquely identified by `alumni_id` and includes personal, academic, and professional

fields like name, email, graduation_year, branch, company_name, and job_title. A username field is used for login and is set to be unique to prevent duplication. Profile images are referenced using the image_url, and a boolean field password_change_required is included for enforcing first-time login policies.

- The **alumni_event** table stores information about events organized by the university or alumni associations. Each event is uniquely identified by event_id and includes fields such as title, description, venue, event_date, and event_time. Events are created by admins and are visible to all approved alumni users.
- To track alumni participation in events, the **event_registration** table manages a many-to-many relationship between the alumni and alumni_event tables. It uses a composite structure referencing both alumni_id and event_id, with a unique registration_id as the primary key. This setup allows a single alumni to register for multiple events and each event to host multiple alumni. The registration_date timestamp captures when the alumni signed up.
- The **contact_message** table allows alumni to communicate with admins, while, each message is assigned a unique message_id, includes the alumni_id of the sender, a subject, and the content of the message. The sent_at field records the time of submission automatically. The contact_message table is truly two-way because admins can view and respond to messages in their dashboard.
- The **activity_log** table provides an important audit trail. Just about every action taken by an admin will be recorded with the activity_log. For example, if an admin adds a new alumni profile or changed the details of an event, it is recorded. The activity_log table provides a lot of detail, including log_id to uniquely identify each log, admin_id of the user that performed the action, a description of the action, an entity_type (e.g. "Alumni", "Event"), the affected entity_id, and the time of action. It will provide transparency and support debugging and compliance assurance. Together, these tables form a tightly integrated data model.

Key foreign key relationships include:

- activity_log.admin_id → admin.admin_id
- event_registration.alumni_id → alumni.alumni_id
- event_registration.event_id → alumni_event.event_id
- contact_message.alumni_id → alumni.alumni_id

Relationships

- One-to-many: Admin → Events
- Many-to-many: Alumni ↔ Events (via registration)
- One-to-many: Admin/Alumni → Messages
- One-to-many: Admin → Activity Logs

Utilizing Java Persistence API (JPA) and Hibernate, these tables will automatically be mapped to Java classes, and the operations for creating, updating, deleting, and querying entries in the database will all be completed behind the scenes through Spring Data JPA repositories. This database design successfully supports the system's core functionality while leaving it open for future extensibility to include job postings, alumni donations, or multi-university capabilities.

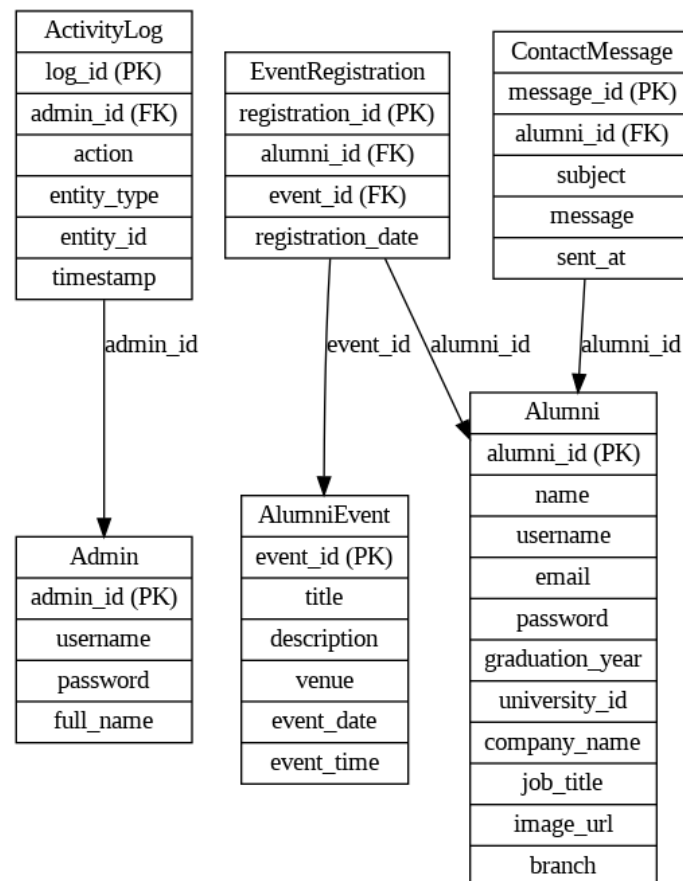


Figure 6.3 Database Schema

e) Frontend Design and User Experience

The front end of the Alumni Management System was designed to prioritize clarity, usability, responsiveness, and user-by-role improved user experience. The front end was coded with HTML5, CSS3, Thymeleaf, and some use of Bootstrap, with only optional JavaScript for

flexibility and quickness in interaction and rendering. Each page of the front end is organized into dynamic templates to provide user-by-role specific content, so users (both admin and alumni) only see what is relevant to their role. The need for templating and dynamic rendering of content uses Thymeleaf, an innovative server-side Java templating engine and integrated module with Spring Boot. Thymeleaf accepts data from the back end and can dynamically injectivity values directly into HTML pages using expressions like `th:text`, `th:each`, and `th:if`, giving you either a clean dynamic UI, or a clean static UI, depending on your development choice. Each page we structured corresponds to a specific action or a view (i.e., dashboards, profile editing form, listing of events, and a message inbox).

The login and registration pages are very accessible and simple. The input fields have well thought out styling for readability and client-side validation is applied for basic text inputs to ensure mandatory fields have been submitted by the user. For alumni, the registration form is full of all the academic and personal information needed, while still feeling less overwhelming so that it can be intuitive to work through and collect the pertinent academic and professional data. There is file upload support to allow alumni to upload a profile picture. Admin users do not register in this application. Admin user credentials are seeded through the database or by an existing admin user. When alumni or admin users log in they are both redirected to their respective dashboard pages. The admin user would see the admin dashboard which has quick access to alumni management, event creation, activity logs, etc., while the alumni user would see the alumni dashboard which has their profile information, as well as a list of events that they could register for, all while sitting in a dashboard context. The navigation menu would adjust dynamically based on the role of the person who is logged in to keep the menu clean by only showing pages and options authorized to the logged in user role.

The **event listings** are visually grouped using cards or tables, with consistent layouts that display key information such as title, venue, and date. Alumni users can register for events with a single click, triggering a controller-based registration process that updates the database without requiring page reloads. Event registration feedback is displayed immediately, improving interactivity and reducing confusion. A publicly accessible **alumni directory** page allows visitors to browse through verified alumni profiles in a visually appealing card-based layout. Each card displays a profile image, name, job title, and graduation year. Clicking on a card opens a detailed view of the alumni's profile, which can include extended academic and career history. This public-facing feature helps build institutional visibility and supports

networking. The event listings are presented visually in groups as cards or tables and with consistent layouts that display key pieces of information (title, venue, and date). Alumni users can register for events with a single click, which uses a controller to generate a robust registration process that updates the database without requiring a page reload. Event registration feedback is very contextual, further improving user interaction and reducing user confusion. The alumni directory page allows the public to browse verified alumni profiles with a visually appealing card-based design. Each card displays a profile image, name, job title, and graduation year. A click on a card opens an expanded view of the alumni's profile which can include more robust academic and career history. This public-facing feature creates institutional visibility and empowers networking.

Responsiveness is a key design principle. The front-end taps into Bootstrap's grid system and media queries to ensure that layouts adjust responsively to desktops, tablets, and smartphones. Navigation menus require a hamburger-style toggle on smaller screens, and input forms scale smoothly without needing to scroll horizontally. This ensures that all users can access the website comfortably on computers or mobile devices. Visual consistency is sustained through a global stylesheet that standardizes fonts, button styles, form inputs, and error messages. Colors are consistent across the application. In summary, the frontend is designed to offer a seamless and personalized experience to both alumni and admin users, while remaining visually clean, responsive, and functional across devices. The role-specific views, dynamic data binding, and smooth navigation all contribute to a user-centric platform that encourages engagement and long-term usability.

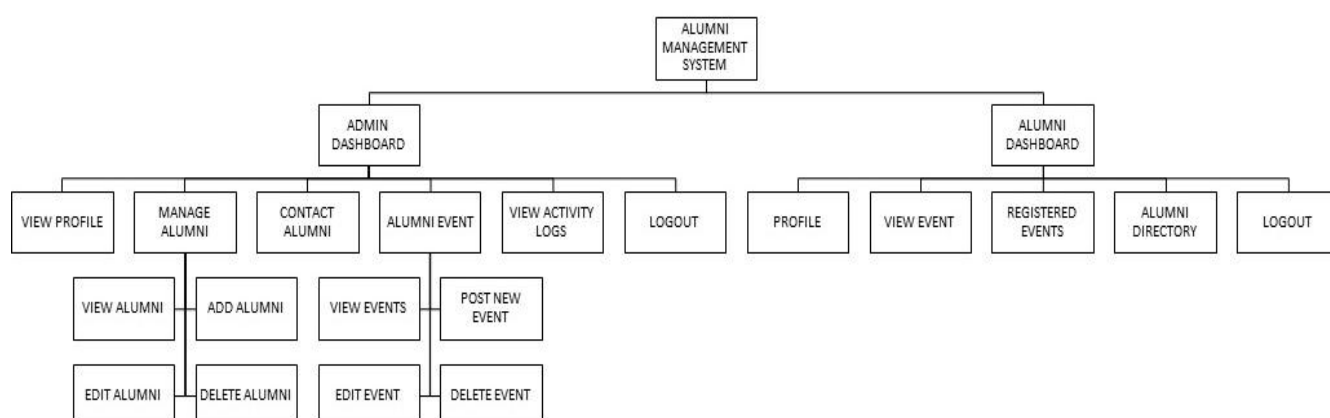


Figure 6.4 Frontend Implementation Flow Chart

f) Security and Access Control

Security is a major part of the Alumni Management System, ensuring end-user data is secured, access is role-based, and malicious behaviors are detoured. The System uses Spring Security, which is a strong authentication and authorization framework that tightly binds to the Spring Boot application stack.

The application has two roles in the system; Admin and Alumni, that have defined permissions associated with that role. The roles are constructed using the role-based access control (RBAC) in SecurityConfig. It uses the method `.antMatchers()` and `.hasRole()` to explicitly associate defined routes with user roles. For example, routes like `/admin/**` are restricted to users with the "ADMIN" role; routes like `/alumni/**` are restricted to users with the "ALUMNI" role.

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/admin/**").hasRole("ADMIN")
            .requestMatchers("/alumni/**").hasRole("ALUMNI")
            .requestMatchers("/css/**", "/js/**", "/images/**", "/login", "/", "/view-alumni").permitAll()
            .anyRequest().authenticated()
        )
}
```

Figure 6.5 SecurityConfig Class

User authenticates through a Spring Security mechanism for a form-based login. In the case of login, the user will be taken to the login form that collects credentials for administrative or alumni users. The credentials are then submitted, and the credentials must be verified in the application's back-end. It should be recognized that the passwords are hashed using BCrypt and then stored safely in the database. BCrypt is a strong one-way cryptographic hashing algorithm. During user authentication, the password entered by the user is hashed and then again hashed when Spring Security's BCryptPasswordEncoder compares the hash in the database. This way, the raw password is not returned to the application as well as available to data leaks. After successful user authentication, Spring Security creates the authentication context for the user session and keeps track of their identity and role information for as long as the session is alive. Every entity within the application is related to its owner through unique identifiers. For example, if an alumni user updates their profile or registers for an event, they can only update their profile or register for an event, not someone else's. Likewise, an admin user can explore and activate on all other records of data. This data ownership logic adds an additional level of

protection for unwanted data manipulation even though the route level protection might have been overcome.

The application also allows for enforced first-time login through the system. The `encourage_change_password` field in the `alumni` table, allows for admins to enforce password resets for imported users, or for the new users to the system, now the user on initial login replaces their default credentials.

Chapter-7

TIMELINE FOR EXECUTION OF PROJECT (GANTT CHART)

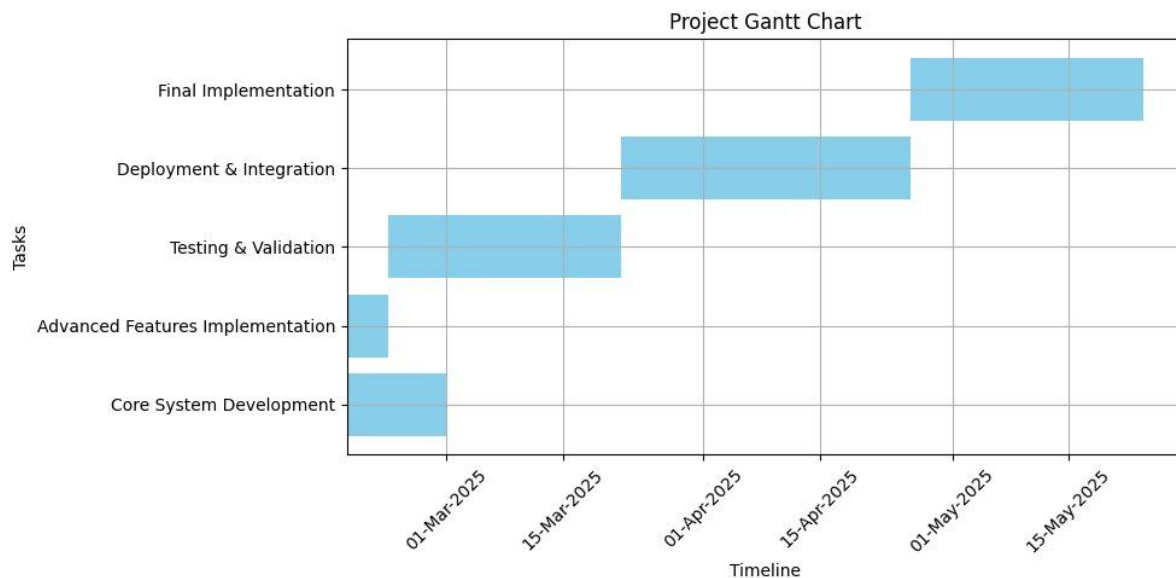


Figure 7.1 Gantt Chart

- The project spans from late February to mid-May 2025, covering all key development phases.
- Core System Development and Advanced Features Implementation were completed by early March, laying the foundation for the system.
- Testing & Validation was conducted from early March to early April, overlapping slightly with development for early issue detection.
- Deployment & Integration followed testing, running throughout April to ensure smooth system rollout.
- Final Implementation began in early May, focusing on refinements, bug fixes, and final user adjustments.
- Phases are sequenced logically with some overlap, reflecting an incremental development approach.
- The chart demonstrates clear planning with a balanced allocation of time across development, testing, and deployment stages.

Chapter 8

OUTCOMES

8.1 Introduction

This chapter presents the major outcomes of the Alumni Management System project, including technical output, functional deliverables, and the value added to possible users. The outcomes of the project reflect a successful implementation of the methodology proposed and indicate that the project's overall objectives have been met.

8.2 Achievements

The project successfully delivered a fully-functional, secure, and scalable online Alumni Management System (AMS) with the following features:

- **User Role Management:** Created customized roles for Admin and Alumni, with their own dashboards and their own access to features.
- **Alumni Registration and Alumni Profile Management:** Alumni were allowed to register with a secure way to register as alumni, edit their profile, along with uploading their images, and would be verified by admin approval.
- **Event Management:** Built full CRUD support for the ability to create, edit, delete, register for events, and retain all items in the database, with full functionality.
- **Admin Control Panel:** Provided Admin the tools to manage alumni, broadcast their events, and review activity logs for auditing.
- **Public Alumni Directory:** Built a browsable listed directory visible to the public which supports networking and transparency of the institution.
- **Messaging System:** Admin and alumni can communicate securely, in both directions, using the built-in messaging module.
- **Activity Logging:** Activity logs of admin activities, such as data edits, provide more understanding and accountability of processes taken by admin users.
- **Responsive Design:** Provided a user-focused, mobile-friendly interface using HTML, CSS, Bootstrap, and Thymeleaf.
- **Data Security:** Provided a secured delivery of data with Spring Security (including hashed passwords), role-based access control, CSRF and session management.

8.3 Educational Outcomes

In addition to the system deliverables, the project also provided valuable hands-on experience in:

- Full-stack web application development using Spring Boot, MySQL, and Thymeleaf.
- Applying software engineering principles including modular architecture, version control, and MVC design.
- Collaborative development using GitHub, issue tracking, and code reviews.
- Real-world problem-solving involving security, user experience, and data integration.

8.4 Future Scope

Though the Alumni Management System has reached its initial goals well, there are many possibilities for enhancing the functionality and reach of the system for future versions. The modular and clean design of the code makes the system very extensible and allows easy additions of functions while maintaining existing functionality.

An additional possibility would be to add a Job and Internship Board where alumni could post, or get access to job openings and internship opportunities to help strengthen the alumni-student ecosystem. This would also support career placements and increase engagement with the platform.

Another opportunity would be to add a Mentorship Module that permits alumni to volunteer as mentors and to provide guidance to current students based upon personal experiences with the industry. Some features could include mentor search, one-on-one chat, progress tracker tools, and mentor rating/review of students.

Connecting to social media (including, for example, LinkedIn and Google) could allow for better login functionality and permit alumni to automatically fill in professional information into their profiles. By using social authentication through OAuth2, the ease of the user's experience can be increased, as well as the validity of the profile data.

If we are going to do more fundraising and institutional development, we should include a Donations Module. This module would allow alumni to make financial contributions to the university using a secure payment gateway, and university administrators would be able to view donation tracking and receipt generation.

Lastly, we were supposed to explore some further enhancements in analytics and reporting, where the administrator will be able to view interactively,

Chapter 9

RESULTS AND DISCUSSIONS

9.1 Introduction

This chapter provides an analysis of the results obtained during the development, testing and validation of the Alumni Management System. It will analyze how the system fulfilled the original goals, provide information about the technical successes and limitations, and discuss the performance and usability of the final application.

9.2 Role-Based Functionality

A major design goal was to enforce strict role-based access control between Admin users and Alumni users. At the end of the project, the system met this goal by implementing Spring Security which allowed user-specific logins, online session management, and route access policy restrictions. Admin users were able to administer Alumni records, events, and sending Messages while Alumni users were able to view their profiles, public directory, and register for events. Finally, I performed a number of test cases for unauthorized access. All cases were intercepted and cascaded to data isolation and privacy.

9.3 Core Functional Modules

All of the core modules were fully built and the associated functions implemented in each core module (Alumni Management, Event Management, Messaging) were implemented with complete CRUD. For example, an administrator could add a new alumni profile, edit an existing alumni profile, or delete an old profile; an event could be created, registrants could register for an event, and all processes would update and validate in real-time; and messaging could occur two ways between the alumni and administrator with the data being save securely in the database backend. All of the modules were tested were simulated user calls, and all of the core workflows passed without major defects.

9.4 User Interface and Usability

The front end, built using Thymeleaf, HTML5, and CSS3, is supposed to be clear, responsive, and user role specific. Feedback we received from users described the front end as clear and easy to navigate. The design choices made for the prototype were based on changing the layout of the pages dynamically based on the user type. Actions like registration, signing up for events,

or modifying profiles could be completed in a few clicks. The front end was also able to perform across numerous devices and screen resolutions, confirming our responsive design process.

9.5 Challenges Faced

Despite the overall success of the project, there were certainly a few obstacles along the way to consider. The ways to handle file uploads for profile pictures had specific implementations tied to the environment of the application, which required different configurations for path access and storage. Our other challenge near the end of the project was making sure the messaging system was secure, while at the same time not complicating the backend logic of the application - we needed to prevent abuse, but also allow open messaging. Similar to the issues outlined in prior paragraphs, these challenges were culminated through testing and improvement, but they pointed to areas for growth in the future, especially around scalability and configuration context.

9.6 Collaboration and Version Control

The team used Git and GitHub for their code management, version control, branch management, and issues management. Having this level of organization ensures that the codebase is well-maintained. Pull requests and committing on a regular basis kept everyone in sync, and many Git history references were useful during debugging and being traceable. This type of professionalism in a workflow will help you deal with complex, interdependent code changes and space the team from potential dangers such as overwriting code and/or conflicts in configuration.

9.7 Discussion and Analysis

The end system met all critical functional requirements and used reasonable engineering techniques including layered architecture, data validation, and modularity. The good examples of role-based features, efficient interfaces, and interacting with a real-time database reflected good experience interpreting the principles of full-stack development. For the discussion, the outcomes from the system showed that it could be scaled further with other future modules to include job postings, mentorships, or donations. This outcome related to extensibility and planning because each application has the ability to build upon each previous application.

Chapter 10

CONCLUSION

The process of developing the Alumni Management System has been an extensive and fulfilling experience, blending academic knowledge with its practical application in order to produce a thorough, reliable, secure, and user-friendly web application. The system simply was, and is, envisioned as a product to close the gap between educational institutions and their alumni, while tackling desirable functionalities related to alumni information management, event management and communication to institutions. As a practical application, the project successfully applied a full-stack architecture in the form of Spring Boot, MySQL and Thymeleaf, which follows the accepted Model-View-Controller (MVC) paradigm. With Spring Security behind it to uphold user authentication and role-based access, user accounts, sensitive data and functions are assumed to be protected at all levels of the application. With a focus on modular development, the integration of key modules incorporating profile management, event registration and messaging has been achieved with an eye to long term functionality, scale and maintainability.

Aside from producing a working application, the project also afforded me an opportunity to learn about important aspects of software development such as right-hand version control with Git; how to separate backend logic, the design of a database schema, the necessity of responsiveness in front-end applications under the constraints of accessibility and the importance of safe form handling. Furthermore, developing collaboratively through GitHub with my teammates was relevant to establishing professional and production-like workflows, including best practices like branching, pull requests and issue management. Not everything went smoothly, the issues we wrestled with, especially with uploading files (uploading documents), and implementing routing based on interrogating roles and validation against sponsored/entity-specific data were notable.

The Alumni Management System was not only operational, but it was successful in how well users accepted it. It had a responsive interface, was easy to navigate through, and provided users with distinct roles to enable a smooth experience across devices. The systematic database design and layered architecture, provide a solid foundation for adding in new feature improvements, including mentorship systems, job boards, donation tools, and so on.

The project also goes beyond the technical aspects, as the Alumni Management System, fulfills the grander vision of continual engagement and relationship building. The system will serve educational institutions by allowing them to keep alumni engaged and connected as well as tracking participation in programs to provide future touchpoints to alumni with events, updates and reminders.

In summary, the project is an example of how applied learning, teamwork, and structured projects can translate academic work into real-world digital solutions, not only for future developments of features, but also as a launching pad for the team to create more significant enterprise applications in professional settings. The lessons learned from this project, go beyond writing code, as they reinforce the importance of planning, collaboration and user-centered design in the field of software engineering.

REFERENCES

- [1] Zatke, Siddhi & Tandel, Chinmaya. (2024). **Alumni Management System Using LinkedIn API**. International Journal of Creative Research Thoughts (IJCRT), Volume 12, Issue 5, May 2024. ISSN: 2320-2882. <https://www.ijcrt.org/papers/IJCRT2405887.pdf>
- [2] Sawai, Parth P., Chambhare, Prajyot V., Jaysingpure, Aditya N., Karhe, Atharav G., Rathod, Disha, & Gulhane, V. S. (2024). **Alumni Connect Hub: A Comprehensive Alumni Management System**. International Journal of Ingenious Research, Invention and Development (IJIRID), ISSN (Online): 2583-648X. <https://doi.org/10.5281/zenodo.10822623>
- [3] Mitali Ved, Hitakshi Tanna, Pratik Yeole, Pradnya Kamble, **Alumni Management System -Web Application, Journal, I. R. J. E. T.** (2022) IRJET.
- [4] Radhika, A., Mayraaj, Shaik, Devisree, B., Surendra Sai, B., & Uma Ganesh, B. (2024). **Alumni Management System**. International Journal of Engineering Science and Advanced Technology (IJESAT), Vol. 24, Issue 05, May 2024. ISSN: 2250-3676. https://www.ijesat.com/ijesat/files/V24I0504_1714750369.pdf
- [5] Lavanya, K., Supriya, Y., Bhargavi, K., Aswini, N., & SwarajyaLakshmi, G. (2021). **Alumni Management System Using Web Technologies**. Juni Khyat, Vol-11 Issue-01, 487. ISSN: 2278-4632. (UGC Care Group I Listed Journal). http://www.junikhyatjournal.in/no_1_Online_21/65.pdf
- [6] Tarun Kumar, Yeeshu Prateek, Prajwal Atharga, Dr. Rajashekarappa, Prof. V. K. Parvati, 2019, **Alumni Database Management System**, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NCRACES – 2019 (Volume 7, Issue 10)
- [7] Mukherjee, Aritra & Roy, Adrita & Lath, Manish & Ghosal, Arnab & Sengupta, Diganta. (2019). **Centralized Alumni Management System (CAMS)** - A Prototype Proposal. 967-971. 10.1109/AICAI.2019.8701383.

- [8] A. A. Shinu, V. Keerthi, M. Manoj, M. Shaheer, N. S. Nair and A. John, "**Customized Alumni Portal implementing Natural Language Processing Techniques**," 2019 International Conference on Intelligent Computing and Control Systems (ICCS), 2019, pp. 927-931, doi: 10.1109/ICCS45141.2019.9065320.
- [9] Dodake, Prajkta, Shinde, Rugved, Kakad, Makarand, Ghodke, Shital, & Shinde, T. R. (2022). **Alumni Management System Solution to Alumni Database**. **International Journal of Engineering Research & Technology (IJERT)**, <https://www.ijert.org/research/alumni-management-system-solution-to-alumni-database-IJERTV11IS050158.pdf>
- [10] Sinduja, S., Thamaraiselvi, A., Prarthana, T., Poorviga, B., Shalini, C., & Ezhilsri, K. (2024). **Alumni Connect: Building Bridges Beyond Graduation Through Mobile App**. **International Journal of Engineering Research & Technology (IJERT)**, Volume 13, Issue 10, October 2024. <https://www.ijert.org/research/alumni-connect-building-bridges-beyond-graduation-through-mobile-app-IJERTV13IS100098.pdf>
- [11] Patel, M., Rami, D., & Soni, M. (2020). **Next Generation Web for Alumni Web Portal**. In S. Balaji, Á. Rocha, & Y. N. Chung (Eds.), *Intelligent Communication Technologies and Virtual Mobile Networks (ICICV 2019) (Lecture Notes on Data Engineering and Communications Technologies, Vol. 33)*. Springer, Cham. https://doi.org/10.1007/978-3-030-28364-3_16

APPENDIX-A

PSUEDOCODE

Backend

1. Model Layer

- **Admin**

DEFINE ENTITY Admin:

INTEGER adminId [Primary Key, Auto-Increment]
STRING username [Not Null, Unique]
STRING password [Not Null]
STRING fullName

CONSTRUCTOR Admin():

// Default constructor

CONSTRUCTOR Admin(adminId, username, password, fullName):

SET this.adminId = adminId
SET this.username = username
SET this.password = password
SET this.fullName = fullName

METHOD getAdminId():

RETURN adminId

METHOD setAdminId(adminId):

SET this.adminId = adminId

METHOD getUsername():

RETURN username

METHOD setUsername(username):

SET this.username = username

METHOD getPassword():

 RETURN password

METHOD setPassword(password):

 SET this.password = password

METHOD getFullName():

 RETURN fullName

METHOD setFullName(fullName):

 SET this.fullName = fullName

- **Alumni**

DEFINE ENTITY Alumni:

 INTEGER alumniId [Primary Key, Auto-Increment]

 STRING name

 STRING username

 STRING email

 STRING password

 INTEGER graduationYear

 STRING universityId

 BOOLEAN passwordChangeRequired = TRUE [Not Null]

 STRING companyName

 STRING jobTitle

 STRING imageUrl

 STRING branch

CONSTRUCTOR Alumni():

 // Default constructor

CONSTRUCTOR Alumni(...):

 INITIALIZE all fields with parameters

METHOD getAlumniId()

RETURN alumniId

METHOD setAlumniId(alumniId)

SET this.alumniId = alumniId

METHOD getName()

RETURN name

METHOD setName(name)

SET this.name = name

METHOD getUsername()

RETURN username

METHOD setUsername(username)

SET this.username = username

METHOD getEmail()

RETURN email

METHOD setEmail(email)

SET this.email = email

METHOD getPassword()

RETURN password

METHOD setPassword(password)

SET this.password = password

METHOD getGraduationYear()

RETURN graduationYear

METHOD setGraduationYear(graduationYear)

SET this.graduationYear = graduationYear

METHOD getUniversityId()

RETURN universityId

METHOD setUniversityId(universityId)

SET this.universityId = universityId

METHOD isPasswordChangeRequired()

RETURN passwordChangeRequired

METHOD setPasswordChangeRequired(value)

SET this.passwordChangeRequired = value

METHOD getCompanyName()

RETURN companyName

METHOD setCompanyName(companyName)

SET this.companyName = companyName

METHOD getJobTitle()

RETURN jobTitle

METHOD setJobTitle(jobTitle)

SET this.jobTitle = jobTitle

METHOD getImageUrl()

RETURN imageUrl

METHOD setImageUrl(imageUrl)

SET this.imageUrl = imageUrl

METHOD getBranch()

RETURN branch

METHOD setBranch(branch)

SET this.branch = branch

2. Service Layer

- **AdminService**

DEFINE INTERFACE AdminService:

METHOD insertAdmin(admin):

// Add a new admin to the system

RETURN created Admin object

METHOD updateAdmin(admin, id):

// Update an existing admin with given ID

RETURN updated Admin object

METHOD deleteAdmin(id):

// Remove an admin from the system by ID

RETURN nothing

METHOD fetchAllAdmins():

// Retrieve a list of all admin users

RETURN List of Admin objects

METHOD fetchAdminById(id):

// Fetch a specific admin using their unique ID

RETURN Admin object

METHOD fetchAdminByUsername(username):

// Fetch an admin using their username (used in login)

RETURN Admin object

- **AdminServiceImpl**

DEFINE CLASS AdminServiceImpl IMPLEMENTS AdminService:

INJECT AdminRepository adminRepo

METHOD insertAdmin(admin):

 CALL adminRepo.save(admin)

 RETURN saved Admin

METHOD updateAdmin(admin, id):

 FETCH optionalAdmin FROM adminRepo.findById(id)

 IF optionalAdmin is present:

 SET existingAdmin = optionalAdmin.get()

 UPDATE existingAdmin.username WITH admin.username

 UPDATE existingAdmin.password WITH admin.password

 UPDATE existingAdmin.fullName WITH admin.fullName

 CALL adminRepo.save(existingAdmin)

 RETURN updated Admin

 ELSE:

 THROW Exception("Admin not found with ID: " + id)

METHOD deleteAdmin(id):

 CALL adminRepo.deleteById(id)

METHOD fetchAllAdmins():

 RETURN adminRepo.findAll()

METHOD fetchAdminById(id):

 CALL adminRepo.findById(id)

 IF admin found:

 RETURN admin

 ELSE:

```
    THROW Exception("Admin not found with ID: " + id)
```

```
METHOD fetchAdminByUsername(username):
```

```
    CALL adminRepo.findByUsername(username)
```

```
    IF admin found:
```

```
        RETURN admin
```

```
    ELSE:
```

```
        THROW Exception("Admin not found with username: " + username)
```

- **AlumniService**

```
DEFINE INTERFACE AlumniService:
```

```
    METHOD insertAlumni(a):
```

```
        // Add a new alumni record to the system
```

```
        RETURN newly created Alumni object
```

```
    METHOD updateAlumni(a, id):
```

```
        // Update an existing alumni's profile using their ID
```

```
        RETURN updated Alumni object
```

```
    METHOD deleteAlumni(id):
```

```
        // Delete an alumni record by ID
```

```
        RETURN nothing
```

```
    METHOD fetchAllAlumni():
```

```
        // Get a list of all alumni users
```

```
        RETURN List of Alumni objects
```

```
    METHOD fetchAlumniById(id):
```

```
        // Retrieve a specific alumni record by ID
```

```
        RETURN Alumni object
```

```
    METHOD fetchAlumniByEmail(email):
```

// Retrieve an alumni record using their email

RETURN Alumni object

METHOD fetchAlumniByUsername(username):

// Retrieve an alumni record using their username

RETURN Alumni object

- **AlumniServiceImpl**

DEFINE CLASS AlumniServiceImpl IMPLEMENTS AlumniService:

INJECT AlumniRepository alumniRepo

INJECT ActivityLogService activityLogService

INJECT HttpSession session

METHOD insertAlumni(a):

SAVE alumni 'a' using alumniRepo

IF a logged-in admin exists in session:

CREATE ActivityLog with:

adminId = logged-in admin ID

action = "Added new Alumni"

entityType = "Alumni"

entityId = saved alumni's ID

timestamp = current time

CALL activityLogService.insertLog(log)

RETURN saved Alumni

METHOD updateAlumni(a, id):

FIND alumni by id using alumniRepo

IF alumni exists:

UPDATE existing alumni fields with data from 'a'

IF new imageUrl is not blank:

UPDATE existingAlumni.imageUrl

SAVE updated alumni to database

IF a logged-in admin exists in session:

CREATE ActivityLog with:

adminId = logged-in admin ID

action = "Updated Alumni"

entityType = "Alumni"

entityId = updated alumni's ID

timestamp = current time

CALL activityLogService.insertLog(log)

RETURN updated Alumni

ELSE:

THROW Exception("Alumni not found")

METHOD deleteAlumni(id):

DELETE alumni from alumniRepo using ID

IF a logged-in admin exists in session:

CREATE ActivityLog with:

adminId = logged-in admin ID

action = "Deleted Alumni"

entityType = "Alumni"

entityId = deleted alumni ID

timestamp = current time

CALL activityLogService.insertLog(log)

METHOD fetchAllAlumni():

RETURN list of all alumni from alumniRepo

METHOD fetchAlumniById(id):

FIND alumni by ID

IF found:

RETURN alumni

ELSE:

THROW Exception("Alumni not found")

METHOD fetchAlumniByEmail(email):

 FIND alumni by email

 IF found:

 RETURN alumni

 ELSE:

 THROW Exception("Alumni not found")

METHOD fetchAlumniByUsername(username):

 FIND alumni by username

 IF found:

 RETURN alumni

 ELSE:

 THROW Exception("Alumni not found")

3. Repository Layer

- **Admin Repository**

DEFINE INTERFACE AdminRepository EXTENDS JpaRepository<Admin, Integer>:

 METHOD findByUsername(username):

 // Find an admin based on their unique username

 RETURN Optional<Admin>

- **Alumni Repository**

DEFINE INTERFACE AlumniRepository EXTENDS JpaRepository<Alumni, Integer>:

 METHOD findByEmail(email):

 // Retrieve an alumni by their email address

 RETURN Optional<Alumni>

 METHOD findByUsername(username):

 // Retrieve an alumni by their username

 RETURN Optional<Alumni>

```
METHOD findByEmailOrUsername(email, username):  
    // Retrieve an alumni by either email or username  
    RETURN Optional<Alumni>
```

4. Controller Layer

- **Admin Controller**

DEFINE CONTROLLER AdminController:

```
INJECT AdminService  
INJECT AlumniService  
INJECT EventService  
INJECT ActivityLogService  
INJECT HttpSession
```

ROUTE GET /admin/dashboard:

```
    IF admin not in session:  
        REDIRECT to /AdminLogin  
    ELSE:  
        LOAD admin profile  
        RETURN AdminPage view
```

ROUTE GET /admin/alumni/fetch:

```
    RETURN list of all alumni from AlumniService
```

ROUTE POST /admin/alumni/add:

```
    IF image URL is empty:  
        SET default profile image  
    SET passwordChangeRequired to false  
    CALL insertAlumni()  
    REDIRECT to /admin/dashboard
```

ROUTE GET /admin/alumni/view:

```
    IF admin not in session:
```

REDIRECT to /AdminLogin

ELSE:

LOAD all alumni

RETURN AdminAlumniDirectory view

ROUTE POST /admin/alumni/update:

FETCH existing alumni by ID

UPDATE alumni fields from form

PRESERVE existing image and sensitive fields if missing

CALL updateAlumni()

REDIRECT to /admin/alumni/view

ROUTE POST /admin/alumni/delete:

DELETE alumni by ID

REDIRECT to /admin/dashboard

ROUTE POST /admin/update-profile:

IF admin not in session:

RETURN 401 Unauthorized

UPDATE admin fields from request

CALL updateAdmin()

UPDATE session

RETURN success message

ROUTE POST /admin/events/save:

IF eventId is missing:

THROW error

CALL insertEvent()

SET session flag to show event section

REDIRECT to /admin/dashboard

ROUTE GET /admin/events/view:

IF admin not in session:

REDIRECT to /AdminLogin

ELSE:

LOAD all events

RETURN AdminViewEvents view

ROUTE GET /admin/edit-event/{eventId}:

IF admin not in session:

REDIRECT to /AdminLogin

FETCH event by ID

RETURN event edit view

ROUTE POST /admin/delete-event:

DELETE event by ID

IF admin exists:

CREATE activity log for deleted event

REDIRECT to /admin/events/view

ROUTE GET /admin/logs:

RETURN activity logs for current admin (hardcoded ID = 1 for now)

- **Alumni Controller**

DEFINE CONTROLLER AlumniController:

INJECT AlumniService

INJECT EventService

INJECT EventRegistrationService

INJECT HttpSession

ROUTE GET /alumni/changepassword:

IF logged-in alumni not in session:

REDIRECT to /AlumniLogin

ELSE:

RETURN ChangePassword view

ROUTE POST /alumni/changepassword:

IF logged-in alumni not in session:

REDIRECT to /AlumniLogin

IF newPassword != confirmPassword:

SHOW error message

RETURN ChangePassword view

UPDATE alumni's password and mark passwordChangeRequired as false

SAVE alumni

REDIRECT to /Alumni/dashboard

ROUTE GET /Alumni/dashboard:

IF alumni not in session:

REDIRECT to /AlumniLogin

LOAD:

- profile

- all events

- registered events

- list of all alumni

RETURN AlumniPage view

ROUTE POST /alumni/update:

IF alumni not in session:

REDIRECT to /AlumniLogin

UPDATE alumni fields (name, email, company, etc.)

IF image is uploaded:

SAVE image to local directory and set image URL

IF no image provided:

SET default image if image URL is blank

SAVE updated alumni

UPDATE session

REDIRECT to /Alumni/dashboard

ROUTE GET /alumni/view-events:

IF alumni not in session:

REDIRECT to /AlumniLogin

LOAD all events

RETURN AlumniViewEvents view

ROUTE GET /alumni/registered-events:

IF alumni not in session:

REDIRECT to /AlumniLogin

LOAD events registered by alumni

RETURN AlumniRegisteredEvents view

ROUTE POST /alumni/events/register:

IF alumni not in session:

REDIRECT to /AlumniLogin

REGISTER alumni for event (check for duplicates)

ADD success or info message to redirect attributes

REDIRECT to /alumni/view-events

ROUTE GET /alumni/directory:

IF alumni not in session:

REDIRECT to /AlumniLogin

LOAD all alumni for display

RETURN AlumniDirectory view

ROUTE POST /alumni/contact:

IF alumni in session:

DISPLAY log of message sent (placeholder for future service call)

REDIRECT to /Alumni/dashboard

ROUTE GET /alumni:

REDIRECT to /Alumni/dashboard

Frontend

- **Admin Dashboard**

DEFINE HTML TEMPLATE AdminPage:

SECTION Sidebar:

DISPLAY links for:

- View Admin Profile
- Manage Alumni (View/Add)
- Contact Alumni
- Alumni Events (View/Post)
- Activity Logs
- Logout

SECTION Main Content:

DISPLAY heading: "Admin Dashboard"

IF 'adminProfileSection' selected:

SHOW admin profile form

FIELDS: Full Name, Username, Password (with Edit/Save buttons)

FORM submits via JavaScript fetch to /admin/update-profile

IF 'alumniTableSection' selected:

IF alumniList is not empty:

DISPLAY table with:

- Alumni ID, Name, Username, Email, Graduation Year, University ID
- Actions: Edit, Delete

ELSE:

DISPLAY message: "No alumni records found."

IF 'addFormSection' selected:

DISPLAY form to add new alumni

FIELDS: name, username, email, password, graduation year, university ID, branch, company name, job title, image URL

FORM submits to /admin/alumni/add

IF 'contactAlumniSection' selected:

DISPLAY contact form to send message to an alumni

FIELDS: Message ID, Alumni ID, Subject, Message

FORM submits to /api/messages/send using AJAX

IF 'postEventSection' selected:

DISPLAY event posting form

FIELDS: Event ID, Title, Description, Venue, Date, Time, Image

FORM submits to /admin/events/save

IF 'activityLogSection' selected:

LOAD activity logs via AJAX call to /api/logs/admin/{adminId}

DISPLAY table with Log ID, Action, Entity Type, Entity ID, Timestamp

IF 'viewEventsSection' is shown (after posting event):

DISPLAY table of all alumni events

FIELDS: Event ID, Title, Description, Venue, Date, Time, Created At

FOOTER:

DISPLAY university contact details (email, phone, location)

SCRIPT LOGIC:

- Toggle between sections using `toggleSection(sectionId)`
- Handle inline edit and save in alumni table
- Submit admin profile via fetch to update data
- Submit contact form and show confirmation message
- Load and display activity logs dynamically

APPENDIX-B

SCREENSHOTS

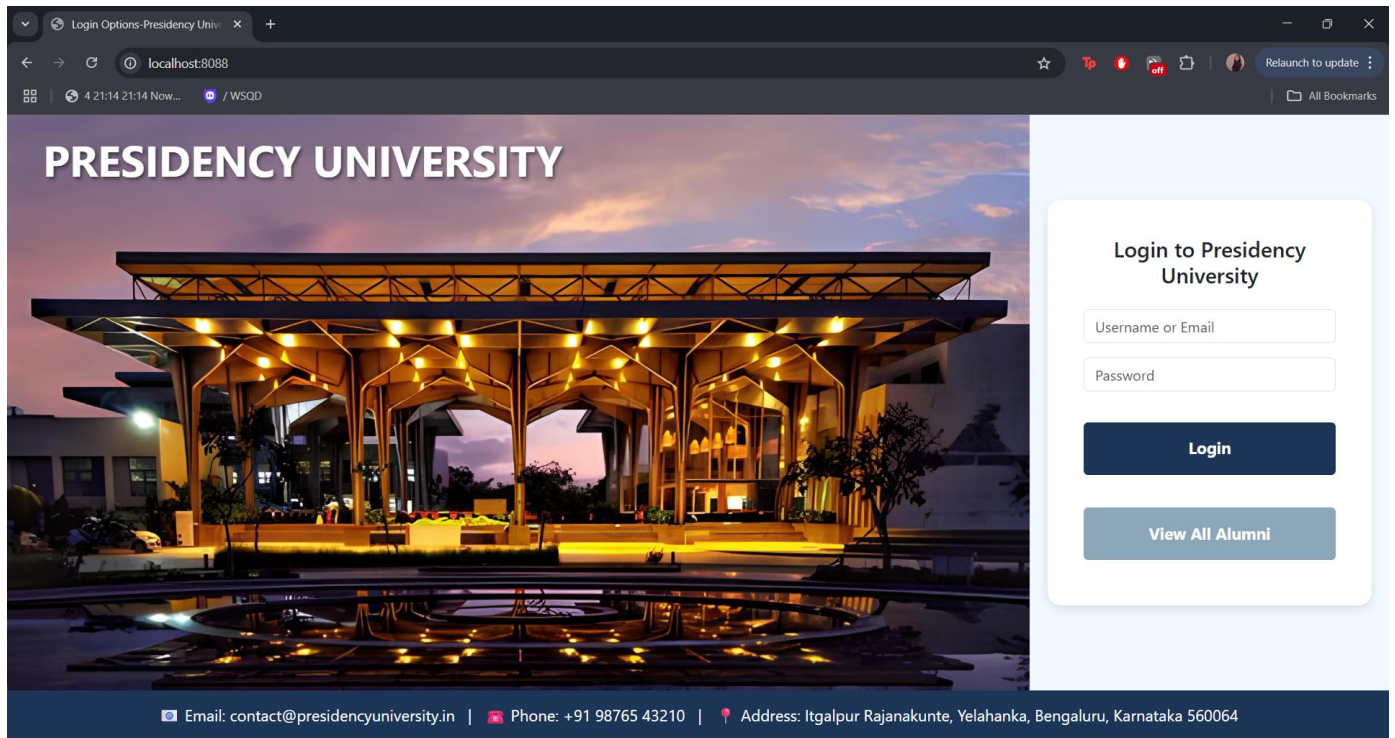


Figure B.1 – Login page

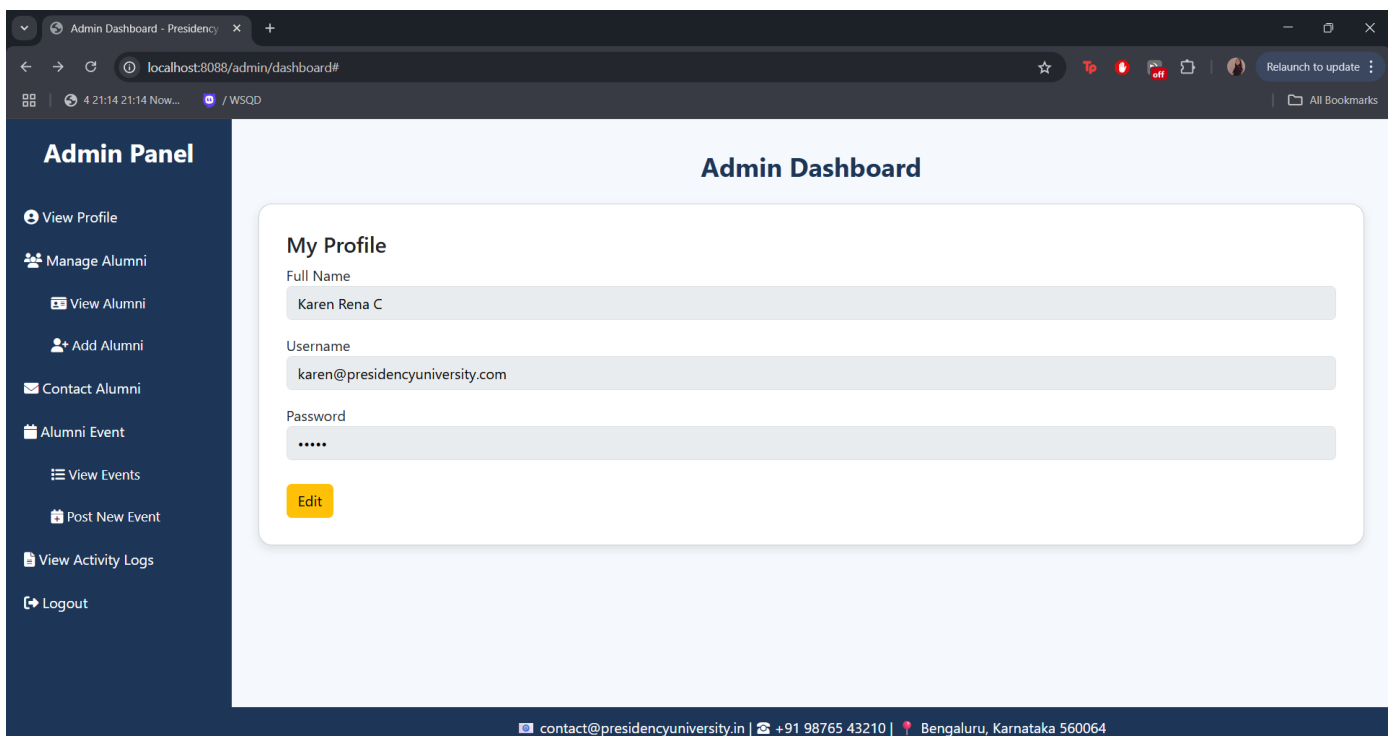


Figure B.2 – Admin Dashboard

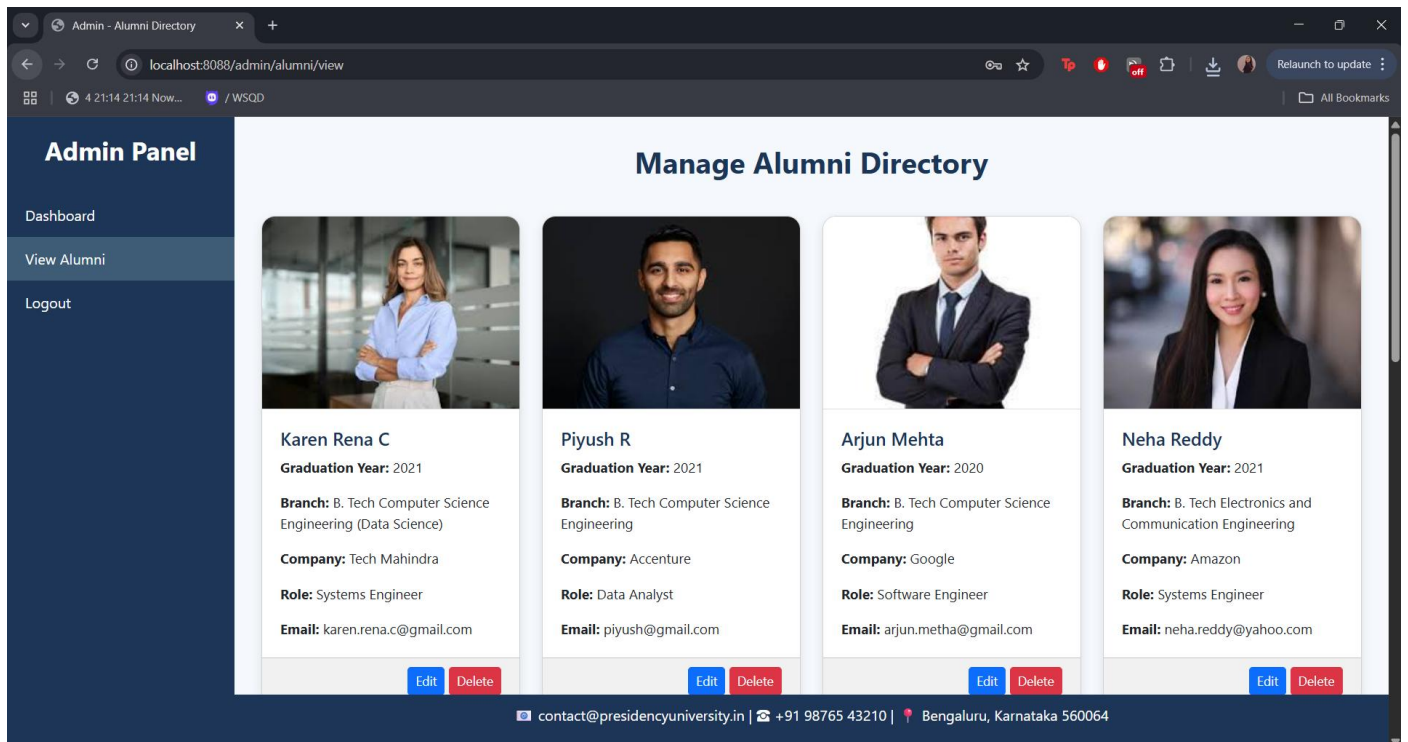


Figure B.3 – View, Edit and Delete Alumni

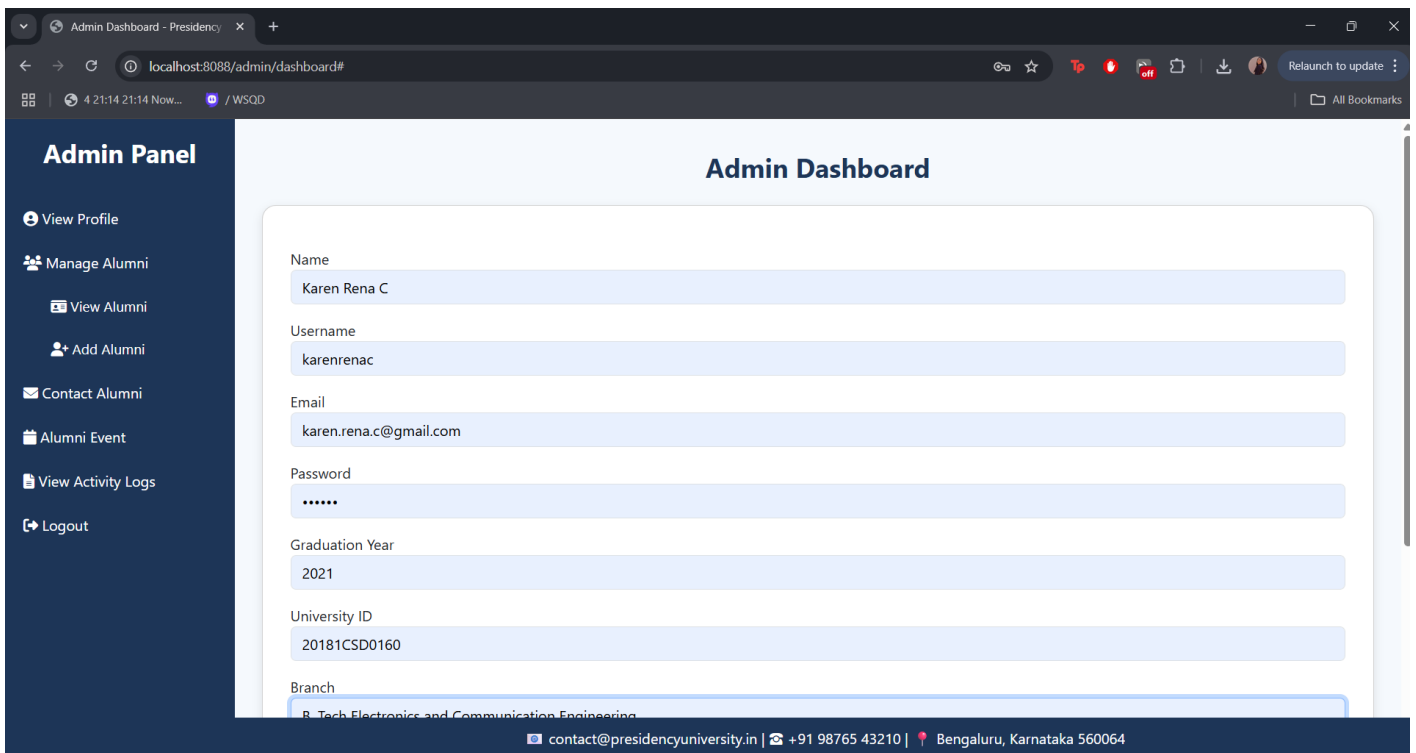


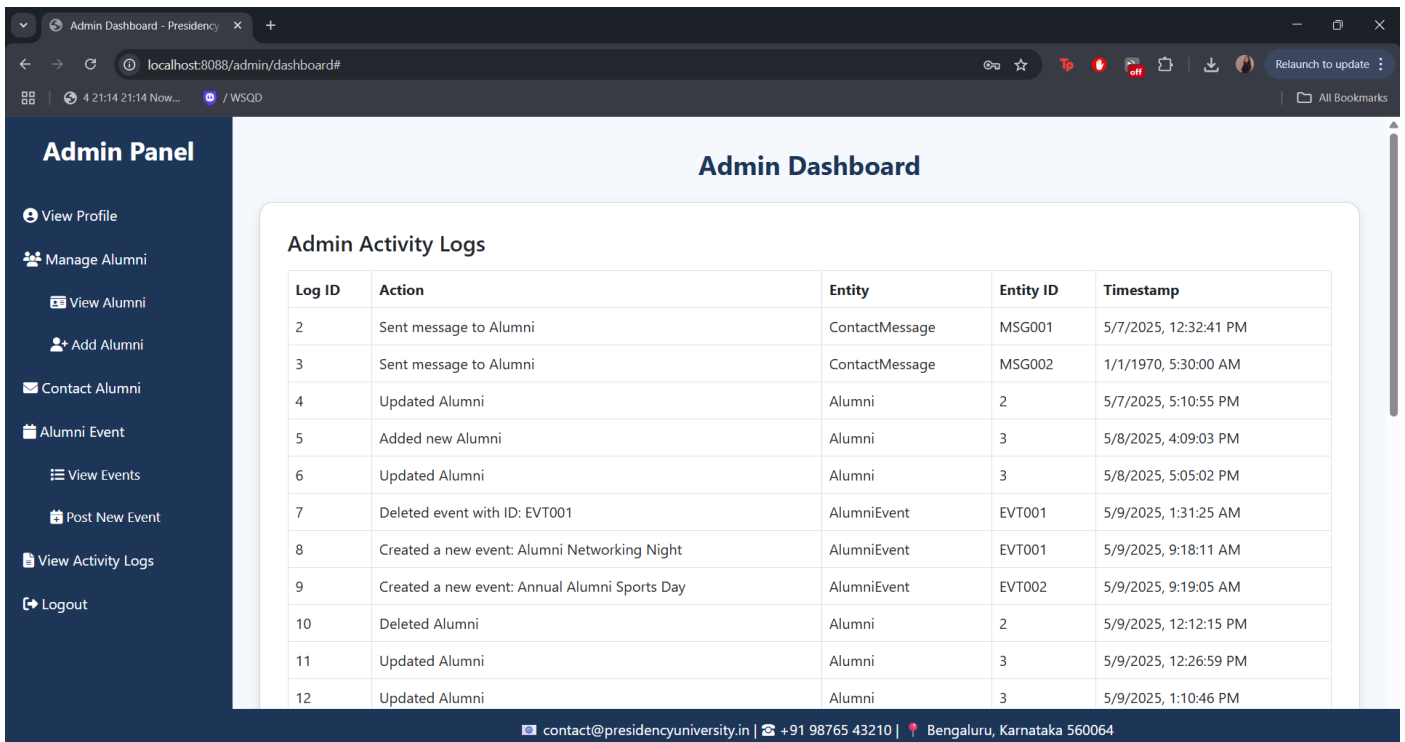
Figure B.4 – Add new Alumni

The screenshot shows a web browser window with the URL `localhost:8088/admin/dashboard#`. The page has a dark blue sidebar on the left titled "Admin Panel" with the following menu items: View Profile, Manage Alumni, View Alumni, Add Alumni, Contact Alumni, Alumni Event, View Activity Logs, and Logout. The main content area is titled "Admin Dashboard" and contains a form titled "Contact Alumni". The form fields are: Message ID (MSG001), Alumni ID (5), Subject (College Fest Reminder), and Message (Invincia on 30th of May). A "Send Message" button is at the bottom of the form. At the bottom of the page, there is a footer with contact information: `contact@presidencyuniversity.in`, `+91 98765 43210`, and "Bengaluru, Karnataka 560064".

Figure B.5 – Contacting Alumni

The screenshot shows the same web browser window as Figure B.5, but the main content area displays a form titled "Post New Event". The form fields are: Event ID (EVT001), Event Title (Alumni Networking Night), Description (A casual evening for alumni to reconnect, expand professional networks, and meet current students. | Presidency University, Auditorium), Location / Online Link (Presidency University, Auditorium), Date (23-05-2025), and Time (10:00). At the bottom of the page, the same footer with contact information is visible.

Figure B.6 – Posting new event

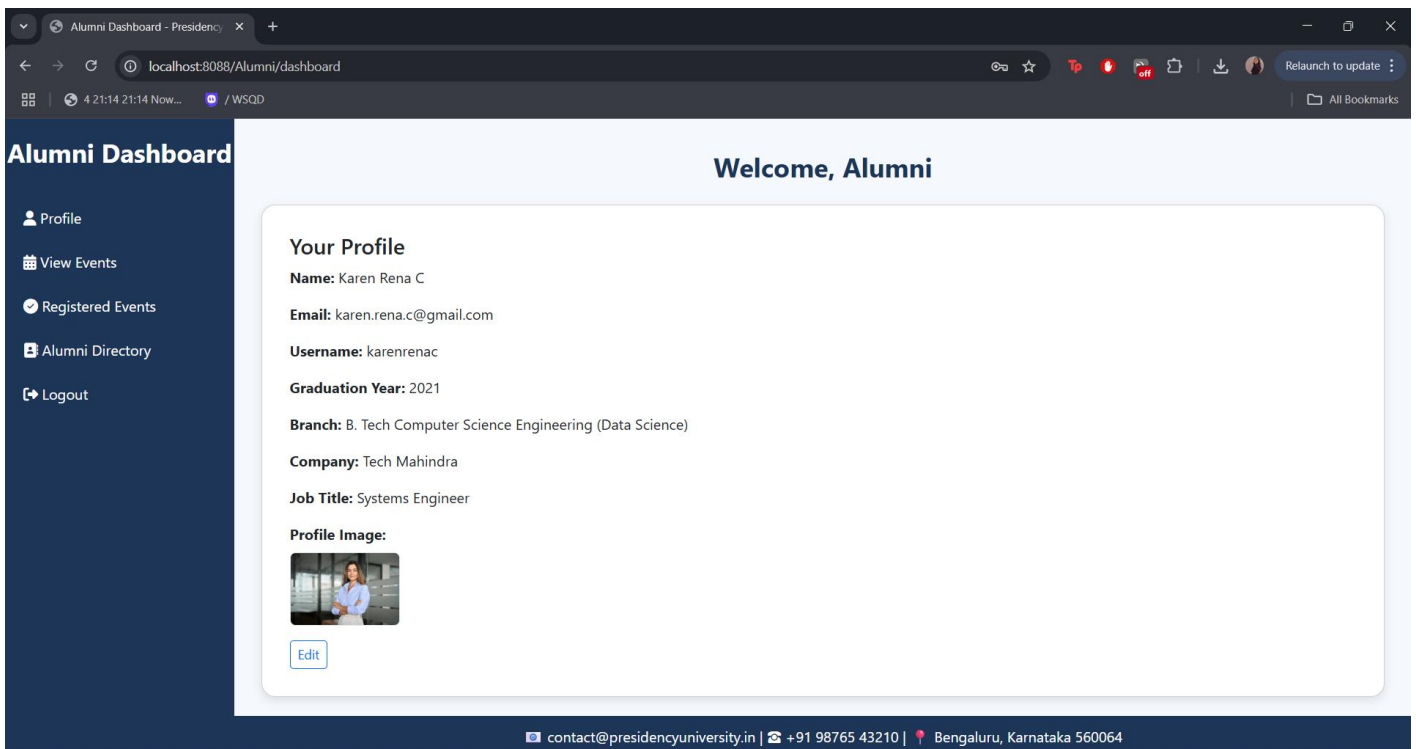


The screenshot shows the Admin Dashboard with a sidebar menu on the left and a main content area. The sidebar menu includes options like View Profile, Manage Alumni, View Alumni, Add Alumni, Contact Alumni, Alumni Event, View Events, Post New Event, View Activity Logs, and Logout. The main content area displays the Admin Activity Logs table.


Log ID	Action	Entity	Entity ID	Timestamp
2	Sent message to Alumni	ContactMessage	MSG001	5/7/2025, 12:32:41 PM
3	Sent message to Alumni	ContactMessage	MSG002	1/1/1970, 5:30:00 AM
4	Updated Alumni	Alumni	2	5/7/2025, 5:10:55 PM
5	Added new Alumni	Alumni	3	5/8/2025, 4:09:03 PM
6	Updated Alumni	Alumni	3	5/8/2025, 5:05:02 PM
7	Deleted event with ID: EVT001	AlumniEvent	EVT001	5/9/2025, 1:31:25 AM
8	Created a new event: Alumni Networking Night	AlumniEvent	EVT001	5/9/2025, 9:18:11 AM
9	Created a new event: Annual Alumni Sports Day	AlumniEvent	EVT002	5/9/2025, 9:19:05 AM
10	Deleted Alumni	Alumni	2	5/9/2025, 12:12:15 PM
11	Updated Alumni	Alumni	3	5/9/2025, 12:26:59 PM
12	Updated Alumni	Alumni	3	5/9/2025, 1:10:46 PM

contact@presidencyuniversity.in | +91 98765 43210 | Bengaluru, Karnataka 560064

Figure B.7 – Admin Activity Logs



The screenshot shows the Alumni Dashboard with a sidebar menu on the left and a main content area. The sidebar menu includes options like Profile, View Events, Registered Events, Alumni Directory, and Logout. The main content area displays the Welcome, Alumni message and the user profile details.

Your Profile	
Name:	Karen Rena C
Email:	karen.rena.c@gmail.com
Username:	karenrenac
Graduation Year:	2021
Branch:	B. Tech Computer Science Engineering (Data Science)
Company:	Tech Mahindra
Job Title:	Systems Engineer
Profile Image:	
	Edit

contact@presidencyuniversity.in | +91 98765 43210 | Bengaluru, Karnataka 560064

Figure B.8 – Alumni Dashboard

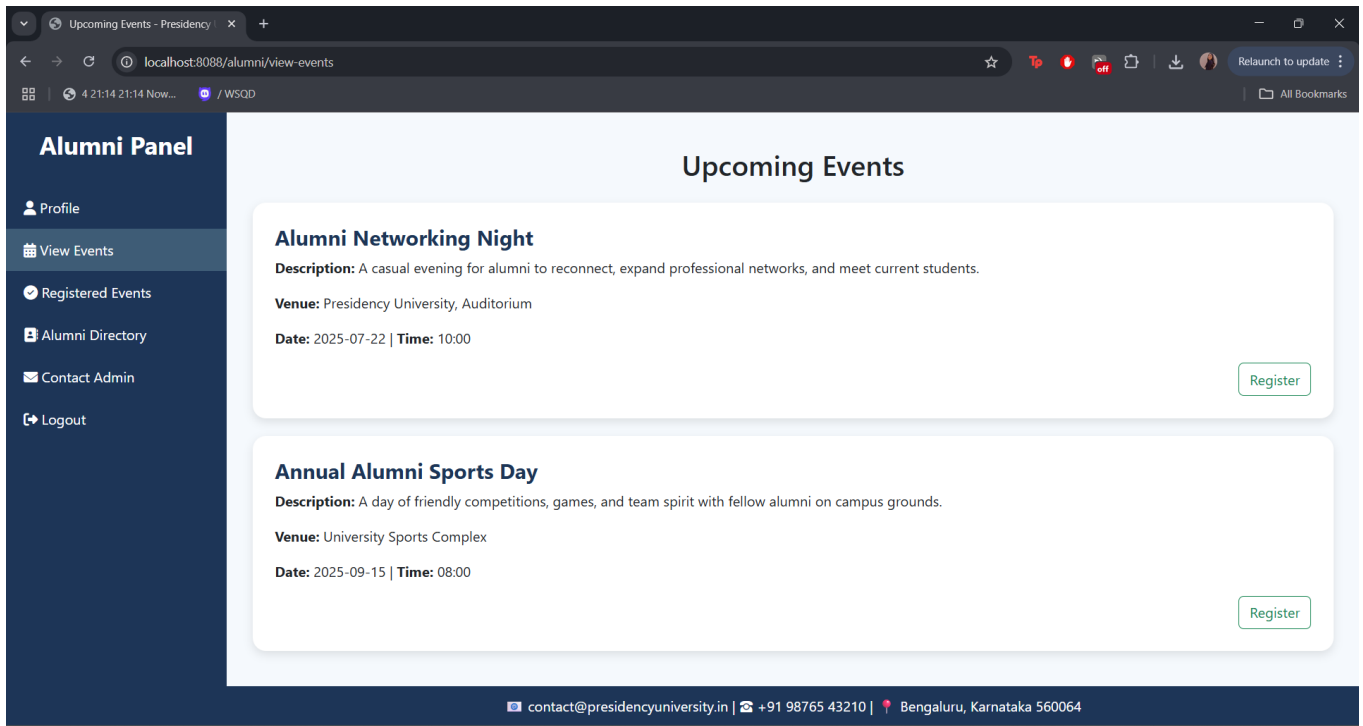


Figure B.9 – View and Register for Events

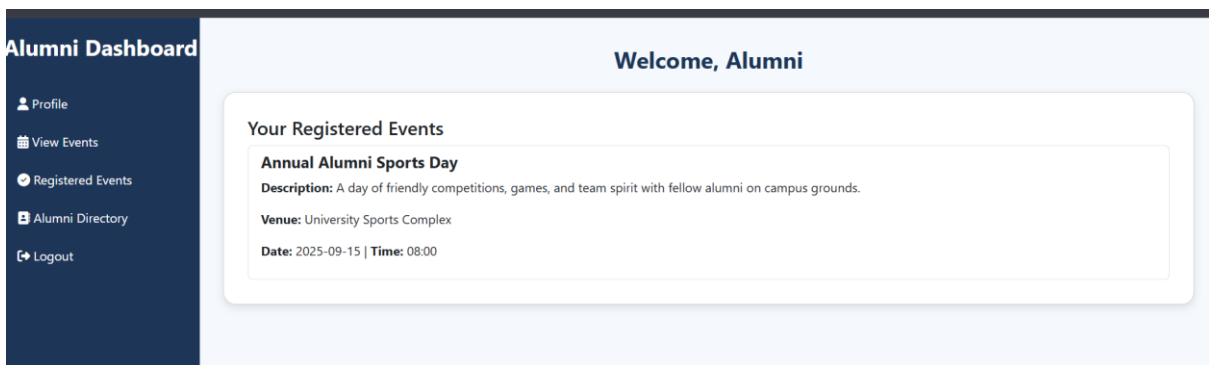


Figure B.10 – View Registered Events

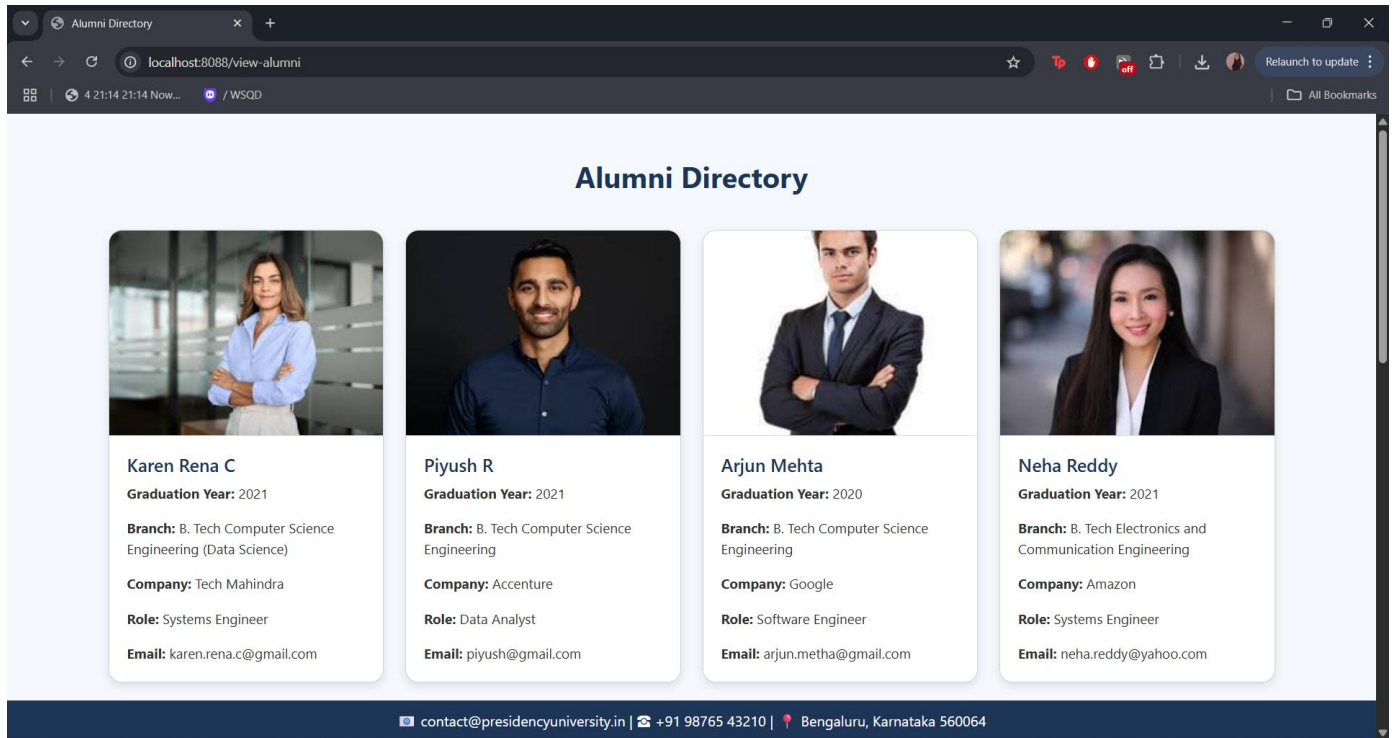


Figure B.11 – Public Alumni Directory

APPENDIX-C

ENCLOSURES

SIMILARITY INDEX / PLAGIRISM REPORT

GROUP 14 - REPORT

ORIGINALITY REPORT

10 %	7 %	4 %	8 %
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Presidency University Student Paper	4 %
2	Submitted to Symbiosis International University Student Paper	3 %
3	www.ijirid.in Internet Source	<1 %
4	ijsret.com Internet Source	<1 %
5	www.ijert.org Internet Source	<1 %
6	morioh.com Internet Source	<1 %
7	Submitted to Buckinghamshire Chilterns University College Student Paper	<1 %
8	ijsrset.com Internet Source	<1 %
	lutpub.lut.fi	

SUSTAINABLE DEVELOPMENT GOALS

SUSTAINABLE DEVELOPMENT GOALS



SDG 4 – Quality Education

Your system creates a platform for alumni to stay connected, share experiences, and provide mentorship, indirectly enhancing access to education resources for current students. The alumni directory, events, and networking opportunities contribute to lifelong learning and institutional development.

SDG 8 – Decent Work and Economic Growth

By maintaining strong alumni networks, institutions can facilitate job referrals, internships, and career guidance — promoting employability and economic opportunities among students and graduates.

SDG 9 – Industry, Innovation, and Infrastructure

The system itself is a technological innovation that strengthens digital infrastructure for educational institutions. It encourages the adoption of modern tools in academic administration and alumni engagement.

SDG 10 – Reduced Inequalities

The system ensures equal opportunity for all alumni to connect and be recognized, regardless of their background or location. Public directories and open access (for viewing) reduce barriers to visibility and participation.

SDG 11 – Sustainable Cities and Communities

Alumni events and community outreach features can support civic engagement, volunteerism, and social development. By strengthening alumni ties, institutions can promote active citizenship and knowledge-sharing within communities.

SDG 17 – Partnerships for the Goals

Alumni systems foster partnerships — not just among alumni, but between institutions, industries, and communities. These collaborations can lead to funding, mentoring, and capacity-building initiatives.