Software Requirements Specification

for

Alumni Management System

Version 1.0 approved

Prepared by Kanishka N

Coimbatore Institute Of Technology

09.10.2025

Table of Contents

Table of Contents ii

Revision History ii

1. Introduction 1

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Project Scope 2

1.5 References 2

2. Overall Description 3

2.1 Product Perspective 3

2.2 Product Features 3

2.3 User Classes and Characteristics 4

2.4 Operating Environment 4

2.5 Design and Implementation Constraints 5

2.6 User Documentation 5

2.7 Assumptions and Dependencies 6

3. System Features 6

3.1 System Feature 1 7

3.2 System Feature 2 (and so on) 8

4. External Interface Requirements 8

4.1 User Interfaces 9

4.2 Hardware Interfaces 9

4.3 Software Interfaces 10

4.4 Communications Interfaces 11

5. Other Nonfunctional Requirements 11

5.1 Performance Requirements 12

5.2 Safety Requirements 12

5.3 Security Requirements 12

5.4 Software Quality Attributes 12

6. Other Requirements 13

Appendix A: Glossary 15

Appendix B: Analysis Models 17

Appendix C: Issues List 25

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

## ****1 . Introduction****

### ****1.1 Purpose****

This Software Requirements Specification (SRS) describes the requirements for the **Alumni Management System (AMS)**, Version 1.0. The AMS is designed to maintain a comprehensive database of alumni, support event and job post management, and facilitate communication between alumni and the institution. This SRS covers the complete scope of the initial system release and does not include future mobile app versions or third-party CRM integrations. The document is intended to guide the development, testing, and maintenance of the system.

### ****1.2 Document Conventions****

This SRS follows IEEE 830-1998 documentation standards. Conventions used:

* **Bold text** for section titles and emphasis.
* Italic text for instructions or examples.
* Monospaced text for field names, code, or identifiers.
* Requirements are tagged with a unique ID in the format REQ-<Module>-<Number> (e.g., REQ-Profile-02).
* Priority is specified as [High], [Medium], or [Low] within requirement descriptions.

### ****1.3 Intended Audience and Reading Suggestions****

This document is intended for:

* **Software Developers** – to implement system features based on requirements.
* **QA Engineers** – to plan and execute test cases.
* **Project Managers** – to track progress and ensure scope coverage.
* **College Administrators and Alumni Officers** – to understand the system's capabilities and provide feedback.
* **Stakeholders and Sponsors** – to evaluate alignment with institutional goals.

**Recommended reading path**:

* Start with **Section 1: Introduction** and **Section 2: Overall Description** for system context.
* **Section 3: System Features** and **Section 4: Functional Requirements** contain implementation specifics.
* **Section 5** and **6** detail non-functional requirements and interface designs, respectively.

### ****1.4 Project Scope****

The **Alumni Management System (AMS)** is a **web-based platform** designed to manage alumni data, facilitate networking, and promote continuous engagement between the institution and its community. The system supports the following user types:

* **Admin Users**: These are institutional staff members such as alumni coordinators, placement officers, or designated faculty. They are responsible for overseeing the system, approving user registrations, managing events, reviewing donation activities, and generating reports.
* **End Users**:
  + **Alumni**: Former students who can register, update their profiles, participate in discussions, post or apply for jobs, donate to the institution, and register for events.
  + **Current Students**: Can view alumni profiles for networking, apply to job or internship opportunities posted by alumni, and register for mentorship or alumni interaction events.

Key features of the system include **profile management, event registration, job and internship posting, discussion forums, donation tracking,** and **communication tools** like announcements and internal messaging. By involving both alumni and current students, the AMS helps bridge the gap between past and present learners, creating mentorship opportunities and career connections. The system supports the institution’s mission to build a **strong, active alumni network** and maintain **lifelong relationships** that contribute to its reputation and growth.

### ****1.5 References****

1. IEEE Computer Society. IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications. IEEE, 1998.
2. Holovaty, Adrian, and Jacob Kaplan-Moss. The Django Book. Version 4.x documentation. Available at: <https://docs.djangoproject.com>
3. Otto, Mark, and Jacob Thornton. Bootstrap Documentation. Available at: <https://getbootstrap.com>
4. Hipp, D. Richard et al. SQLite Documentation. Available at: <https://www.sqlite.org/docs.html>
5. SendGrid Developers. SendGrid Email API Documentation. Available at: <https://sendgrid.com/docs/>
6. GitHub Contributors. Alumni Management System Source Code Repository. (if applicable). Available at: [GitHub Repository URL]
7. [Institution Name]. College/University Alumni Policy Guidelines. (Internal document, if available).
8. W3Schools. HTML, CSS, and JavaScript Tutorials. Available at: <https://www.w3schools.com>

**2. Overall Description**

## ****2.1 Product Perspective****

The Alumni Management System (AMS) is a **new, self-contained web-based application** designed to strengthen the connection between alumni, students, and the institution. It will serve as a centralized platform for alumni networking, event management, donations, and career opportunities.

This system is not a replacement of any existing platform but rather an **independent solution** that can integrate with other university services in the future (e.g., student portals, HR systems).

**System Context:**

* **Internal Components:** Alumni Directory, User Management, Event Management, Job Portal, Donation Module, Reporting.
* **External Interfaces:** Email server (SMTP), Payment Gateway (e.g., Razorpay/PayPal), MySQL Database, and Web Browser.

## ****2.2 Product Features (High-Level Summary)****

The Alumni Management System will provide the following major features:

1. **User Management** – Registration, login, profile update, and role-based access.
2. **Alumni Directory** – Search and filter alumni by name, batch, department, or profession.
3. **Networking & Communication** – Messaging, discussion forums, and email notifications.
4. **Event Management** – Creation of alumni reunions, seminars, workshops, and online registration.
5. **Job & Internship Portal** – Alumni posting opportunities and students applying.
6. **Donations & Contributions** – Secure online donations with receipt generation.
7. **Reports & Analytics** – Summary reports on alumni demographics, event participation, and donations.
8. **Administration** – Content management, approval of alumni registrations, and activity monitoring.

## ****2.3 User Classes and Characteristics****

1. **Administrator**
   * Full access to manage users, events, jobs, donations, and reports.
   * Moderate technical knowledge required.
   * Frequency: Daily use.
2. **Alumni**
   * Can register, update profile, post jobs, donate, and participate in events.
   * Moderate technical knowledge expected.
   * Frequency: Occasional use (monthly/quarterly).
3. **Student**
   * Can register, search alumni, apply for jobs/internships, and view events.
   * Basic technical knowledge expected.
   * Frequency: Moderate use (weekly).
4. **Guest/Visitor**
   * Limited access (e.g., view public announcements/events, but cannot post).
   * No technical expertise required.
   * Frequency: Rare use.

## ****2.4 Operating Environment****

* **Client-Side:** Any modern web browser (Chrome, Firefox, Edge, Safari) with internet access.
* **Server-Side:**
  + Operating System: Linux (Ubuntu/CentOS) or Windows Server.
  + Database: MySQL 8.0+
  + Backend: PHP (Laravel) or Java (Spring Boot).
  + Frontend: HTML5, CSS3, JavaScript, Bootstrap/React.
* **Network:** Minimum 1 Mbps internet connection.
* **Devices:** Desktop, Laptop, Tablet, Smartphone.

## ****2.5 Design and Implementation Constraints****

* The system must use **MySQL** as the database.
* Development must follow **MVC (Model-View-Controller) architecture** for modularity.
* The system must comply with **data privacy policies** (e.g., GDPR if applicable, Indian IT Act).
* Donations must comply with **PCI DSS standards** for payment security.
* The frontend must be **responsive** for desktop and mobile.
* Only open-source technologies or institution-approved licensed software shall be used.

## ****2.6 User Documentation****

The following user documentation shall be delivered:

* **User Manual:** Explains registration, login, navigation, and usage of modules.
* **Administrator Guide:** Instructions for managing users, events, and reports.
* **Online Help & FAQs:** Accessible from within the application.
* **Tutorial Videos (Optional):** For onboarding new alumni and students.
* **Documentation Format:** PDF manuals, embedded HTML help pages, and tooltips.

## ****2.7 Assumptions and Dependencies****

* Users have basic computer literacy and internet access.
* The institution will provide dedicated **administrators** for system management.
* The system depends on:
  + Availability of a **stable internet connection**.
  + Proper functioning of **third-party services** (payment gateways, email servers).
  + Institutional IT infrastructure (server hosting, domain, SSL certificates).
* Any downtime or restrictions on these external dependencies may affect system functionality.

# ****3. System Features****

## ****3.1 User Management****

### 3.1.1 Description and Priority

* Allows alumni, students, and admins to register, login, and manage profiles.
* **Priority:** High (Core functionality).

### 3.1.2 Stimulus/Response Sequences

* User enters credentials → System verifies → Provides access → Displays dashboard.
* Invalid login → Error message displayed with option to reset password.

### 3.1.3 Functional Requirements

* REQ-UM-1: The system shall allow new users to register with required details.
* REQ-UM-2: The system shall verify user credentials at login.
* REQ-UM-3: The system shall allow users to update personal, academic, and professional details.
* REQ-UM-4: The system shall enforce role-based access (Admin, Alumni, Student, Guest).

## ****3.2 Alumni Directory****

### 3.2.1 Description and Priority

* Centralized searchable directory of alumni.
* **Priority:** High.

### Stimulus/Response Sequences

* User searches alumni by name/batch → System retrieves and displays results.

**3.2.3 Functional Requirements**

* REQ-AD-1: The system shall maintain a database of alumni records.
* REQ-AD-2: The system shall allow searching/filtering of alumni by name, batch, department, or profession.**3.3 Event Management**

## 3.3.1 Description and Priority

* Enables creation, registration, and tracking of alumni events.
* **Priority:** Medium to High.

### Stimulus/Response Sequences

* Admin creates event → Alumni register → System generates participant list.

**3.3.3 Functional Requirements**

* REQ-EM-1: The system shall allow admins to create/edit/delete events.
* REQ-EM-2: The system shall allow alumni to register for events.
* REQ-EM-3: The system shall generate participant reports for each event.

## ****3.4 Job/Internship Portal****

### 3.4.1 Description and Priority

* Alumni can post jobs, students can apply.
* **Priority:** Medium.

**3.4.2 Stimulus/Response Sequences**

* Alumni posts a job → System lists it → Students apply → Alumni receives applications.

### 3.4.3 Functional Requirements

* REQ-JP-1: The system shall allow alumni to post job/internship opportunities.
* REQ-JP-2: The system shall allow students to view and apply for posted jobs.

## ****3.5 Donations & Contributions****

### 3.5.1 Description and Priority

* Alumni can contribute financially via secure payment gateway.
* **Priority:** Medium.

### 3.5.2 Stimulus/Response Sequences

* Alumni selects donation option → System redirects to payment → Confirmation displayed → Receipt generated.

### 3.5.3 Functional Requirements

* REQ-DC-1: The system shall allow alumni to donate online using payment gateway integration.
* REQ-DC-2: The system shall generate donation receipts and reports for admins.

# 4.External Interface Requirements

## ****4.1 User Interfaces****

* The system shall provide a **web-based Graphical User Interface (GUI)** accessible through standard browsers (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari).
* The user interface shall be **responsive** to support desktops, laptops, tablets, and mobile devices.
* Each screen shall include:
  + **Navigation bar** (Home, Alumni Directory, Events, Jobs, Donations, Login/Logout).
  + **Standard buttons**: Submit, Cancel, Reset, and Help.
  + **Error messages** displayed in red text below input fields with descriptive guidance.
  + **Confirmation messages** displayed in green for successful actions (e.g., profile update, event registration).
* **Sample Interfaces** (conceptual):
  + **Login/Registration Screen:** Fields for username, password, email, batch, department, profession.
  + **Alumni Directory:** Search bar with filters (name, batch, department, profession) and paginated results.
  + **Event Management Screen:** Event details with registration option and attendee list.
  + **Job/Internship Posting:** Form for alumni to post opportunities with job details and application link.
  + **Donation Page:** Secure payment form integrated with a payment gateway.
* **UI Design Guidelines:**
  + Follow **Material Design / Bootstrap 5 standards** for consistency.
  + Maintain institutional branding (logo, colors, and fonts).
  + Provide **multilingual support** using UTF-8 encoding for names and content.

**4.2 Hardware Interfaces**

* The system shall run on standard **client devices** such as desktops, laptops, tablets, and smartphones with internet access.
* **Client Requirements:**
  + Minimum: 2 GB RAM, 2 GHz processor, 1 Mbps internet connection.
  + Recommended: 4 GB RAM, 2.5 GHz processor, 5 Mbps internet connection.
* **Server Requirements:**
  + Processor: Quad-core, 2.5 GHz or higher.
  + Memory: 8 GB RAM minimum.
  + Storage: 100 GB SSD (expandable for database growth).
  + Network: Gigabit Ethernet with stable internet connection.
* The system shall interface with **input/output devices** (keyboard, mouse, touchscreen) and support **printers** for generating alumni reports or donation receipts.

## ****4.3 Software Interfaces****

* **Operating System:**
  + Server: Linux (Ubuntu/CentOS) or Windows Server.
  + Client: Compatible with Windows, macOS, Linux, Android, and iOS.
* **Database:**
  + MySQL Server 8.0+ for relational data storage.
  + MySQL Connector (for PHP, Java, or Python backend integration).
* **Backend Framework:**
  + PHP with Laravel OR Java with Spring Boot (depending on choice of implementation).
* **Frontend Framework:**
  + HTML5, CSS3, JavaScript, Bootstrap, and optional React/Angular for enhanced UI.
* **APIs/Third-Party Integrations:**
  + Email service (e.g., SMTP, Gmail API, SendGrid) for notifications.
  + Payment Gateway API (e.g., Razorpay, PayPal, Stripe) for donations.
  + Social media API integration for alumni networking (optional future scope).
* **Data Sharing:**
  + Alumni, event, and donation data stored in MySQL shall be accessible to reporting modules via RESTful APIs.
  + All exchanged data shall be in **JSON** format for compatibilit

## ****4.4 Communications Interfaces****

* The system shall communicate using **HTTPS** protocol to ensure secure transmission of data between client and server.
* **Web Interface:**
  + Users shall access the system through standard web browsers via a secure URL (e.g., https://alumniportal.edu).
* **Email Notifications:**
  + The system shall support **SMTP protocol** for sending automated emails (event updates, job postings, password resets).
* **Network Protocols:**
  + TCP/IP for client-server communication.
  + RESTful API calls over HTTP/HTTPS for backend services.
* **Security:**
  + All data in transit shall be encrypted using **SSL/TLS (AES-256 encryption)**.
  + Authentication tokens (JWT or session-based) shall be used for secure communication between modules.
* **Data Transfer Rates:**
  + The system shall support a minimum **1 Mbps internet speed** for users; higher speeds recommended for real-time video or event streaming (future extension).
* **Synchronization:**
  + Database synchronization with backup servers shall be scheduled daily using **MySQL replication or cron jobs**.

### 5. Other Nonfunctional Requirements

### 5.1 Performance Requirements

* The system shall support a minimum of **500 concurrent users** without noticeable performance degradation.
* All standard operations (login, profile search, job postings, donations, etc.) shall respond within **3 seconds under normal load**.
* The database queries shall be optimized to handle at least **100,000 alumni records** efficiently.
* The system shall be scalable to support **future growth of up to 1,000,000 records**.

**5.2 Safety Requirements**

* The system shall perform **daily automated backups** of the database to prevent data loss.
* A **disaster recovery plan** shall ensure restoration of services within **24 hours** in case of server failure.
* The system shall prevent accidental data loss by prompting confirmation before deletion of any alumni records, events, or donations.
* The system shall ensure **data integrity checks** during storage and retrieval.

#### 5.3 Security Requirements

* All communication between client and server shall use **SSL/TLS encryption** (HTTPS).
* User authentication shall follow **role-based access control (RBAC)** for Admin, Alumni, Student, and Guest.
* Passwords shall be stored using **strong hashing algorithms** (e.g., bcrypt or SHA-256 with salt).
* The system shall implement **account lockout** after 5 consecutive failed login attempts.
* Donation transactions shall comply with **PCI DSS standards** to secure financial information.
* Sensitive information such as emails, contact numbers, and payment details shall be encrypted in the database.

**5.4 Software Quality Attributes**

* **Usability:** The interface shall be user-friendly, consistent, and accessible across devices (desktop, tablet, mobile).
* **Reliability:** The system shall maintain an uptime of **99.5% annually**.
* **Maintainability:** The system shall follow **modular design** (MVC pattern) to simplify updates and bug fixes.
* **Scalability:** The system shall support addition of new modules (e.g., mentorship, scholarships) without impacting existing features.
* **Portability:** The system shall be deployable on **Linux, Windows, and Cloud environments**.
* **Interoperability:** APIs shall follow **REST standards** to integrate with third-party services (payment gateway, email, student portals).

# Other Requirements

# ****5.1 Database Requirements****

* The system shall use **MySQL RDBMS** for storing alumni, student, event, job postings, and donation data.
* The MySQL database shall support **ACID (Atomicity, Consistency, Isolation, Durability)** properties to ensure transaction reliability.
* Database schema shall be designed in **3rd Normal Form (3NF)** to eliminate redundancy and maintain consistency.
* The system shall use **primary keys, foreign keys, and indexes** to optimize query performance.
* **Stored procedures and triggers** may be used for automating repetitive database tasks (e.g., updating event counts, maintaining logs).
* The database shall support **backup and recovery** through MySQL utilities (e.g., mysqldump, replication).
* Character encoding shall be set to **UTF-8 (utf8mb4)** to support multilingual names and alumni details.
* The maximum expected database size shall be **100,000+ alumni records**, with scalability options using **MySQL clustering or partitioning** if required.**5.2 Internationalization Requirements**
* Default language of the system shall be **English**, but the database schema shall support **UTF-8 (utf8mb4)** for multilingual data storage.
* Date/time format shall follow **YYYY-MM-DD hh:mm:ss** as per MySQL standards, configurable based on region.
* Currency values shall be stored using **DECIMAL datatype** in MySQL to prevent rounding errors.

## ****5.3 Legal and Regulatory Requirements****

* The database shall comply with **data security and privacy policies**, ensuring no personal information is leaked or misused.
* Donation transactions stored in MySQL must adhere to **financial compliance standards (PCI DSS)**.
* Alumni contact details stored in MySQL must follow **institutional and legal privacy guidelines** (e.g., GDPR if applicable).

## ****5.4 Reuse and Extensibility Requirements****

* The MySQL database schema shall be modular, allowing addition of new tables for **mentorship programs, scholarships, or surveys** without affecting existing modules.
* APIs built on top of MySQL shall be **REST-compliant** for interoperability with other university systems.
* Queries and stored procedures shall be reusable across multiple reporting modules (e.g., alumni directory, job portal, event reports).

## ****5.5 Assumptions and Dependencies****

* The Alumni Management System shall assume the availability of **MySQL Server 8.0+** as the primary database.
* The system depends on **MySQL connectors** for integration with the chosen backend (e.g., PHP, Java, Python).
* Regular **database maintenance (indexing, vacuuming, backups)** shall be handled by the institution’s IT staff.
* The system is dependent on **third-party services (payment gateway, email/SMS notifications)** but all records will be stored securely in MySQL.

Appendix A: Glossary

Appendix A: Glossary

AMS – Alumni Management System, the software product described in this SRS.

Alumni – Graduates or former students of the institution who will use the system for networking, donations, and event participation.

Admin – The system administrator with the highest privilege level; responsible for managing users, events, donations, and overall system maintenance.

Student – Current student of the institution with limited access (view alumni, apply for jobs/internships, register for events).

Guest/Visitor – An unregistered user who can only access limited public information (e.g., announcements, event previews).

User Role – Access level assigned to a user (Admin, Alumni, Student, Guest), defining permissions within the system.

Profile – User’s personal, academic, and professional information stored in the system.

Event – An institutional activity such as alumni reunions, seminars, or workshops that can be created, registered, and managed through the system.

Donation Module – A feature that allows alumni to make financial contributions via secure online payment gateways.

Directory – A searchable list of registered alumni with details such as name, batch, department, and profession.

Job/Internship Portal – A platform within AMS where alumni can post opportunities and students can apply.

MySQL – The relational database management system (RDBMS) used to store system data.

RDBMS – Relational Database Management System; software for managing structured data in relational tables (e.g., MySQL).

UI – User Interface; the screens and navigation through which users interact with the AMS.

UX – User Experience; the overall satisfaction of a user while interacting with the AMS.

API – Application Programming Interface; used for integrating external services such as payment gateways or email servers.

SMTP – Simple Mail Transfer Protocol; used by the AMS to send email notifications.

OTP – One-Time Password; a temporary code used for secure login or user verification.

PCI DSS – Payment Card Industry Data Security Standard; compliance required for secure handling of credit/debit card payments.

GDPR – General Data Protection Regulation; a European data privacy law (if alumni data includes EU residents).

Indian IT Act – Indian Information Technology Act governing data privacy and cybersecurity in India.

Responsive Design – A design approach ensuring the system works on different devices (desktop, tablet, mobile).

Analytics Dashboard – A reporting interface showing insights like alumni demographics, event participation, and donations.

SRS – Software Requirements Specification; the document describing system requirements, constraints, and features.

## ****Use Case 1: View Alumni Directory****

**Primary Actor:** Student  
**Secondary Actor:** Admin  
**Precondition:** Student has logged in successfully.  
**Postcondition:** Student views a list of registered alumni profiles.  
**Trigger:** Student selects the “View Alumni Directory” option.

### ****Main Success Scenario:****

1. Student selects “View Alumni Directory” from the menu.
2. System displays a searchable list of all registered alumni.
3. Student filters alumni by department, batch, or location.
4. System displays matching profiles with details such as name, graduation year, and job role.

### ****Exceptions:****

1. If the student session is expired → System will prompt **“Session timed out, please log in again.”**
2. If no alumni profiles match the search criteria → System will display **“No results found.”**
3. If the database is unavailable → System will show **“Unable to fetch alumni details. Try again later.”**

## ****Use Case 2: Post Job/Internship****

## **Primary Actor:** Alumni **Secondary Actor:** Admin **Precondition:** Alumni is logged in successfully. **Postcondition:** Job/Internship details are posted and visible to students. **Trigger:** Alumni selects “Post Job/Internship” option.

### ****Main Success Scenario:****

1. Alumni selects “Post Job/Internship” from dashboard.
2. Alumni enters job details like title, company name, location, and deadline.
3. Alumni clicks **Submit**.
4. System validates and saves the job posting.
5. System notifies Admin for approval.
6. Once approved, the posting becomes visible to students.

### ****Exceptions:****

1. If job title is empty → System displays **“Job title is required.”**
2. If job description exceeds 500 characters → System displays **“Description too long.”**
3. If alumni is not verified → System prevents posting and shows **“Account verification pending.”**

## ****Use Case 3: Manage Events****

**Primary Actor:** Admin  
**Secondary Actor:** Alumni and Students  
**Precondition:** Admin has logged in successfully.  
**Postcondition:** Event details are updated and visible in the portal.  
**Trigger:** Admin initiates event management action.

### ****Main Success Scenario:****

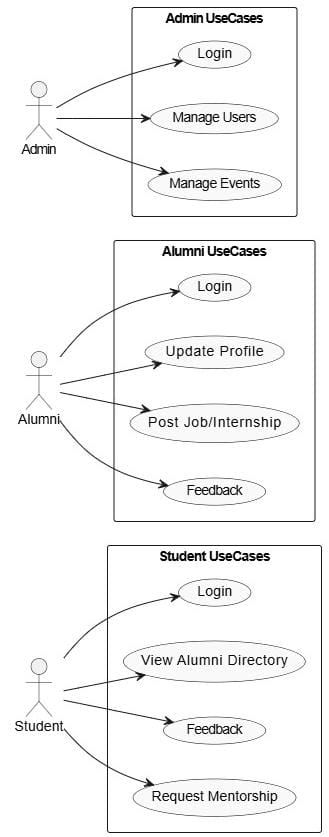
1. Admin navigates to the **Manage Events** section.
2. Admin creates a new event by entering details like name, date, and venue.
3. System validates and saves the event.
4. Event becomes visible to both Alumni and Students.
5. Admin can edit or delete event details when required.

### ****Exceptions:****

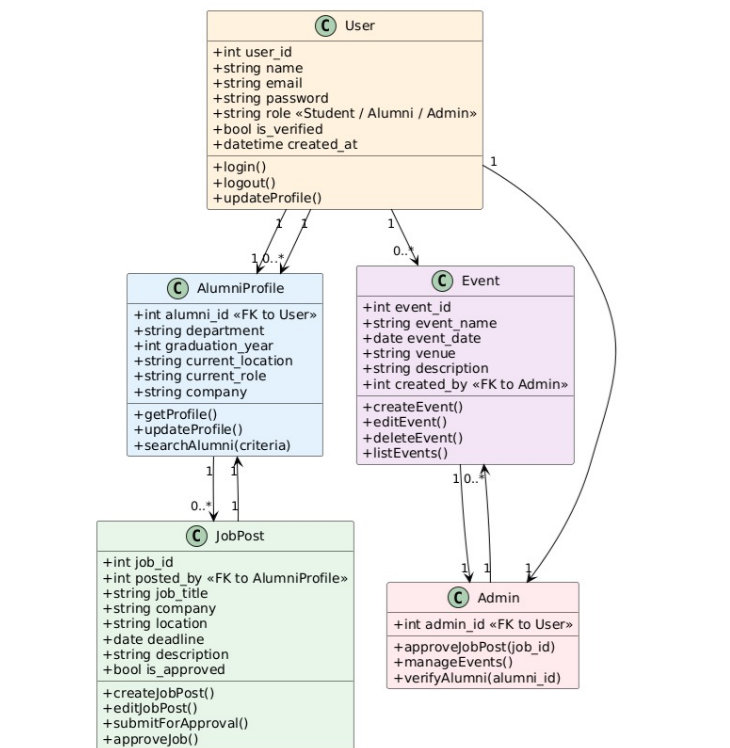
1. If the event name is empty → System displays **“Event name is mandatory.”**
2. If event date is in the past → System displays **“Event date cannot be in the past.”**
3. If there’s a server error → System shows **“Unable to save event. Try again later.”**

Appendix B: Analysis Models

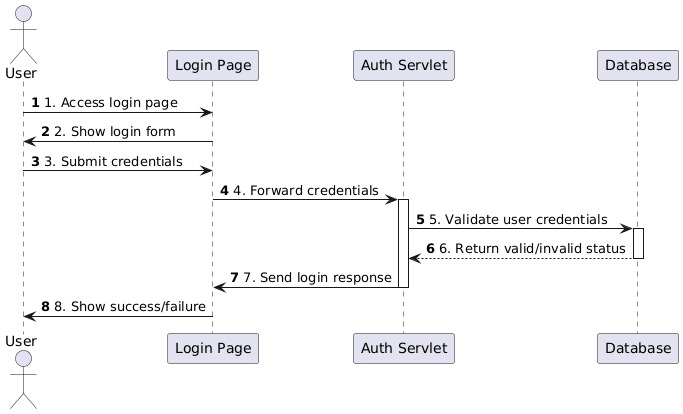
**UML DIAGRAM:**

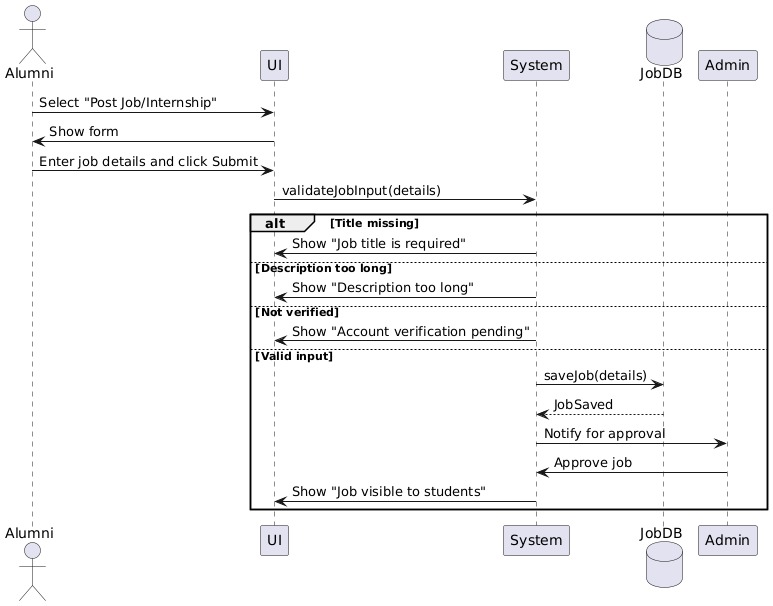


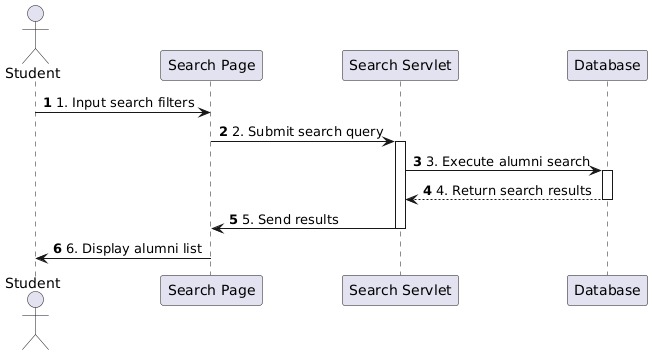
**CLASS DIAGRAM :**

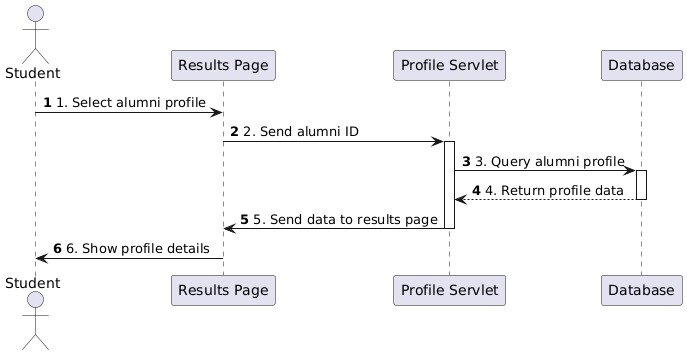
****

**SEQUENCE DIAGRAMS :**

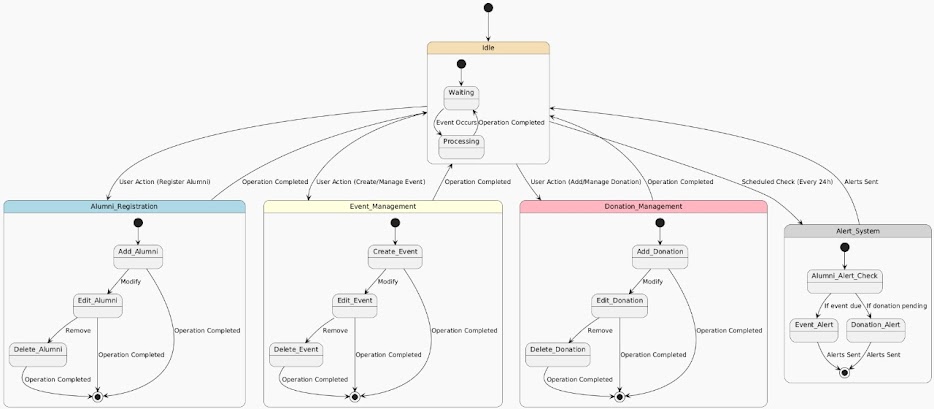




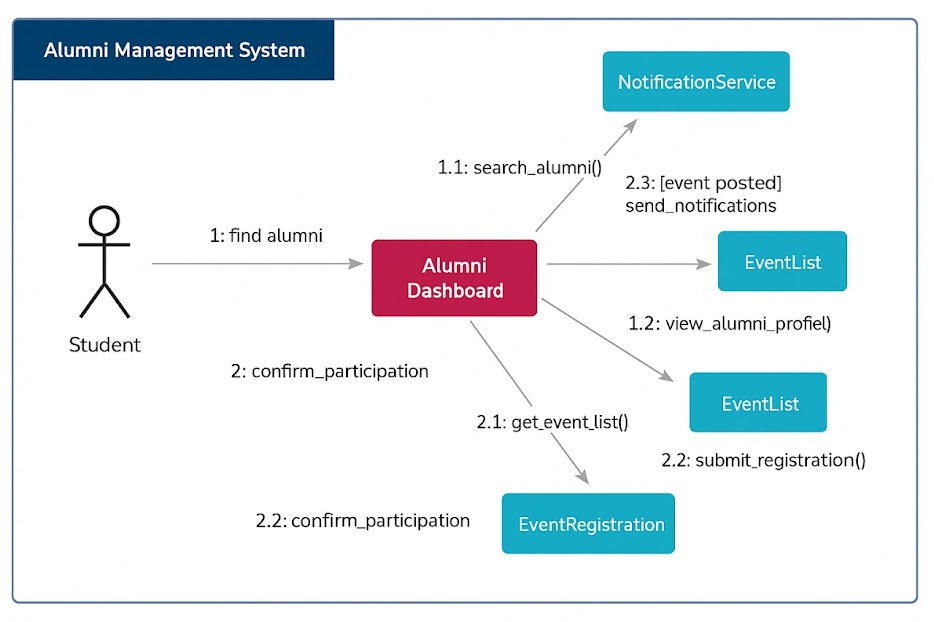




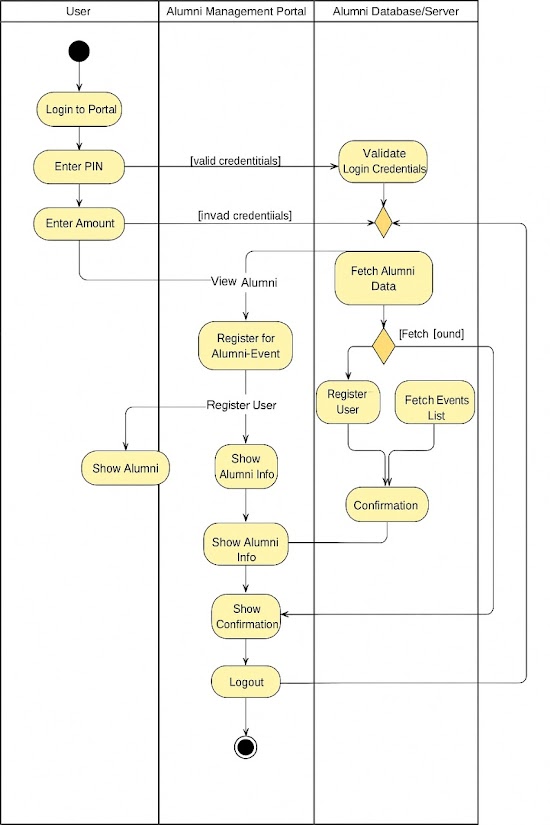
State Diagram :



COLLABORATION DIAGRAM

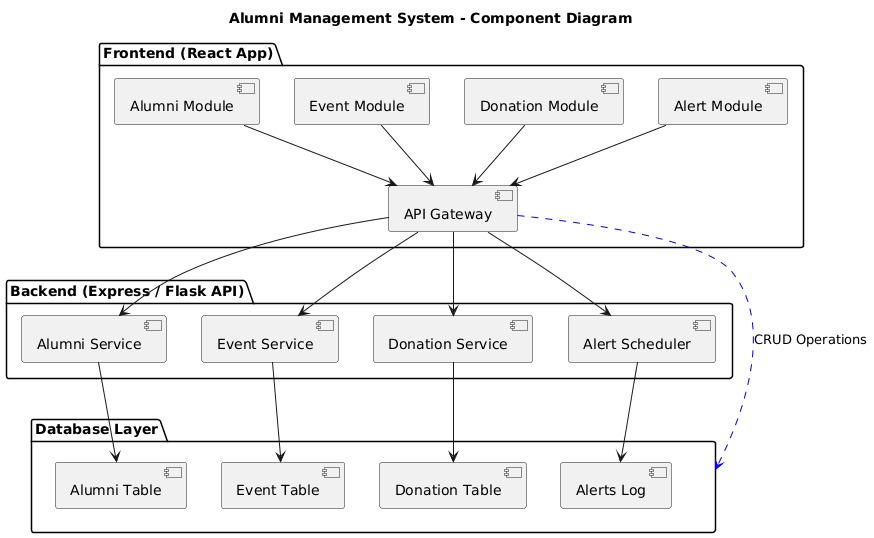


Activity Diagram :

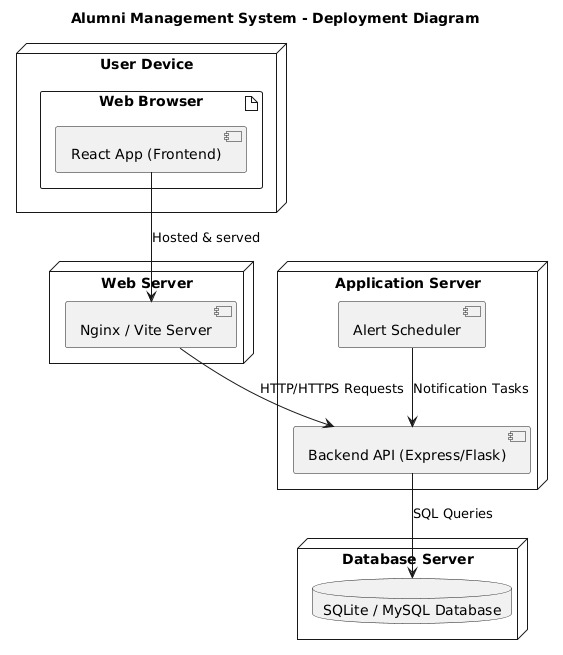


COMPONENT

DIAGRAM :



DEPLOYMENT DIAGRAM :



Appendix C: Issues List

Appendix C: Issues List

1. TBD: Final decision on backend framework (PHP Laravel vs Java Spring Boot).

2. TBD: Payment gateway provider to be integrated (Razorpay, PayPal, or Stripe).

3. Pending: Institutional branding guidelines (logo, fonts, colors) for UI finalization.

4. Pending: Decision on multilingual support scope (English only vs regional languages).

**SOURCE CODE :**

**App.jsx:**

import React from 'react';

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import ANavbar from './components/ANavbar';

import Footer from './components/Footer';

import Home from './pages/Home';

import AlumniList from './pages/AlumniList';

import Profile from './pages/Profile';

import ManageEvents from './components/ManageEvents';

import Jobs from './Job';

import './App.css';

function App() {

const sampleAlumni = [

{ id: 1, name: 'Alice', email: 'alice@example.com', batch: '2020' },

{ id: 2, name: 'Bob', email: 'bob@example.com', batch: '2019' },

{ id: 3, name: 'Charlie', email: 'charlie@example.com', batch: '2018' }

];

const sampleEvents = [

{ id: 1, name: 'Annual Meetup', date: '2025-12-10', venue: 'Main Hall' },

{ id: 2, name: 'Alumni Networking', date: '2025-11-05', venue: 'Auditorium' }

];

const sampleUser = { name: 'John Doe', email: 'john@example.com', batch: '2021' };

return (

<Router>

{/\* Wrapper with background image and overlay \*/}

<div

className="app-container"

style={{

backgroundImage: "url('/college.jpg')", // image in public folder

backgroundSize: 'cover',

backgroundPosition: 'center',

minHeight: '100vh',

position: 'relative',

}}

>

{/\* Overlay for readability \*/}

<div

className="overlay"

></div>

{/\* Content \*/}

<div className="content-wrapper">

<ANavbar />

<div className="main-content">

<Routes>

<Route path="/" element={<Home />} />

<Route path="/alumni" element={<AlumniList alumni={sampleAlumni} />} />

<Route path="/events" element={<ManageEvents events={sampleEvents} />} />

<Route path="/profile" element={<Profile user={sampleUser} />} />

<Route path="/jobs" element={<Jobs />} />

</Routes>

</div>

<Footer />

</div>

</div>

</Router>

);

}

export default App;

**Job.jsx**

import React, { useState } from 'react';

import './App.css';

function Jobs() {

const [title, setTitle] = useState('');

const [company, setCompany] = useState('');

const [type, setType] = useState('Job'); // Job or Internship

const [location, setLocation] = useState('');

const [offers, setOffers] = useState([]);

const [error, setError] = useState('');

const [success, setSuccess] = useState('');

const handleSubmit = (e) => {

e.preventDefault();

setError('');

setSuccess('');

if (!title || !company || !location) {

return setError('Please fill all required fields.');

}

const newOffer = {

id: Date.now(),

title,

company,

type,

location

};

setOffers([newOffer, ...offers]);

setSuccess('Offer posted successfully!');

setTitle('');

setCompany('');

setType('Job');

setLocation('');

};

const handleDelete = (id) => {

setOffers(offers.filter(offer => offer.id !== id));

};

return (

<div className="page-container">

<h2>Post Job / Internship</h2>

<form className="job-form" onSubmit={handleSubmit}>

<input

type="text"

placeholder="Position / Title"

value={title}

onChange={(e) => setTitle(e.target.value)}

/>

<input

type="text"

placeholder="Company Name"

value={company}

onChange={(e) => setCompany(e.target.value)}

/>

<select value={type} onChange={(e) => setType(e.target.value)}>

<option>Job</option>

<option>Internship</option>

</select>

<input

type="text"

placeholder="Location"

value={location}

onChange={(e) => setLocation(e.target.value)}

/>

<button type="submit">Post Offer</button>

{error && <div className="error">{error}</div>}

{success && <div className="success">{success}</div>}

</form>

<h3>Posted Offers</h3>

<ul className="offer-list">

{offers.map(offer => (

<li key={offer.id} className="event-card">

<span>

<strong>{offer.title}</strong> — {offer.company} — {offer.type} — {offer.location}

</span>

<button className="danger" onClick={() => handleDelete(offer.id)}>Delete</button>

</li>

))}

</ul>

</div>

);

}

export default Jobs;

App.py

# backend/main.py

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel, EmailStr

from typing import List, Optional

from fastapi.middleware.cors import CORSMiddleware

app = FastAPI(

title="Modern Cozy API",

description="Backend for a cozy, professional frontend",

version="1.0.0"

)

# Allow frontend requests

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # In production, restrict to your frontend domain

allow\_methods=["\*"],

allow\_headers=["\*"],

)

# ================= Models =================

class Event(BaseModel):

id: int

title: str

description: str

date: str

location: Optional[str] = None

class Alumni(BaseModel):

id: int

name: str

profession: str

bio: Optional[str] = None

class ContactForm(BaseModel):

name: str

email: EmailStr

message: str

# ================= In-Memory Data =================

events\_db: List[Event] = []

alumni\_db: List[Alumni] = []

contact\_submissions: List[ContactForm] = []

# ================= Routes =================

@app.get("/")

def root():

return {"message": "Welcome to the Modern Cozy Backend!"}

# -------- Events --------

@app.get("/events", response\_model=List[Event])

def get\_events():

return events\_db

@app.post("/events", response\_model=Event)

def create\_event(event: Event):

events\_db.append(event)

return event

@app.get("/events/{event\_id}", response\_model=Event)

def get\_event(event\_id: int):

for e in events\_db:

if e.id == event\_id:

return e

raise HTTPException(status\_code=404, detail="Event not found")

@app.put("/events/{event\_id}", response\_model=Event)

def update\_event(event\_id: int, event: Event):

for i, e in enumerate(events\_db):

if e.id == event\_id:

events\_db[i] = event

return event

raise HTTPException(status\_code=404, detail="Event not found")

@app.delete("/events/{event\_id}")

def delete\_event(event\_id: int):

for i, e in enumerate(events\_db):

if e.id == event\_id:

events\_db.pop(i)

return {"message": "Event deleted"}

raise HTTPException(status\_code=404, detail="Event not found")

# -------- Alumni --------

@app.get("/alumni", response\_model=List[Alumni])

def get\_alumni():

return alumni\_db

@app.post("/alumni", response\_model=Alumni)

def create\_alumni(alumnus: Alumni):

alumni\_db.append(alumnus)

return alumnus

# -------- Contact Form --------

@app.post("/contact")

def submit\_contact(form: ContactForm):

contact\_submissions.append(form)

# In production, you could send an email here

return {"message": "Thank you for contacting us!"}

**Home.jsx**

import React from 'react';

function Home() {

return (

<div className="page-container">

<h1>Welcome to Alumni Management System</h1>

<p>Connect with your alumni, manage events, and view profiles easily.</p>

</div>

);

}

export default Home;

**Contact.jsx**

// src/pages/Contact.jsx

import React from "react";

const Contact = () => {

return (

<div className="page-container">

<h3>Contact Us</h3>

<p>

Have questions or want to reach out? Fill out the form below or email us at

<strong> support@cozyportal.com</strong>.

</p>

<form

style={{

display: "flex",

flexDirection: "column",

gap: "0.75rem",

marginTop: "1rem",

background: "rgba(255,255,255,0.9)",

padding: "1rem",

borderRadius: "12px",

boxShadow: "0 6px 15px rgba(124, 58, 237, 0.15)",

}}

>

<input type="text" placeholder="Your Name" />

<input type="email" placeholder="Your Email" />

<textarea placeholder="Your Message" rows="5"></textarea>

<button type="submit" style={{ cursor: "pointer" }}>

Send Message

</button>

</form>

</div>

);

};

export default Contact;

**models.py**

from sqlalchemy import Column, Integer, String

from .database import Base

class Alumni(Base):

\_tablename\_ = "alumni"

id = Column(Integer, primary\_key=True, index=True)

name = Column(String, nullable=False)

email = Column(String, unique=True, nullable=False)

department = Column(String, nullable=False)

year = Column(Integer, nullable=False)

Alumni.py

from fastapi import FastAPI, Depends, HTTPException, status, Query from sqlalchemy.orm import Session from pydantic import BaseModel, EmailStr from fastapi.middleware.cors import CORSMiddleware from typing import Optional, List from . import models from .database import engine, SessionLocal

models.Base.metadata.create\_all(bind=engine)

app = FastAPI(title="Alumni Management API")

CORS (to connect with React)

app.add\_middleware( CORSMiddleware, allow\_origins=[""], # replace "" with your frontend URL later allow\_credentials=True, allow\_methods=[""], allow\_headers=[""], )

-------------------------------

Pydantic Schemas

-------------------------------

class AlumniBase(BaseModel): name: str email: EmailStr department: str year: int

class AlumniCreate(AlumniBase): pass

class AlumniUpdate(BaseModel): name: Optional[str] = None email: Optional[EmailStr] = None department: Optional[str] = None year: Optional[int] = None

class AlumniOut(AlumniBase): id: int

class Config:  
 orm\_mode = True

-------------------------------

DB Dependency

-------------------------------

def get\_db(): db = SessionLocal() try: yield db finally: db.close()

-------------------------------

ROUTES

-------------------------------

Register new alumni

@app.post("/api/alumni", response\_model=AlumniOut, status\_code=status.HTTP\_201\_CREATED) def create\_alumni(alumni: AlumniCreate, db: Session = Depends(get\_db)): existing = db.query(models.Alumni).filter(models.Alumni.email == alumni.email).first() if existing: raise HTTPException(status\_code=400, detail="Alumni with this email already exists.")

db\_alumni = models.Alumni(\*\*alumni.dict())  
db.add(db\_alumni)  
db.commit()  
db.refresh(db\_alumni)  
return db\_alumni

Get all alumni (with optional filters)

@app.get("/api/alumni", response\_model=List[AlumniOut]) def get\_alumni( db: Session = Depends(get\_db), department: Optional[str] = Query(None, description="Filter by department"), year: Optional[int] = Query(None, description="Filter by graduation year"), ): query = db.query(models.Alumni) if department: query = query.filter(models.Alumni.department.ilike(f"%{department}%")) if year: query = query.filter(models.Alumni.year == year) return query.order\_by(models.Alumni.id.desc()).all()

Get single alumni by ID

@app.get("/api/alumni/{alumni\_id}", response\_model=AlumniOut) def get\_alumni\_by\_id(alumni\_id: int, db: Session = Depends(get\_db)): alumni = db.query(models.Alumni).filter(models.Alumni.id == alumni\_id).first() if not alumni: raise HTTPException(status\_code=404, detail="Alumni not found.") return alumni

Update alumni

@app.put("/api/alumni/{alumni\_id}", response\_model=AlumniOut) def update\_alumni(alumni\_id: int, updates: AlumniUpdate, db: Session = Depends(get\_db)): alumni = db.query(models.Alumni).filter(models.Alumni.id == alumni\_id).first() if not alumni: raise HTTPException(status\_code=404, detail="Alumni not found.")

for key, value in updates.dict(exclude\_unset=True).items():  
 setattr(alumni, key, value)  
  
db.commit()  
db.refresh(alumni)  
return alumni

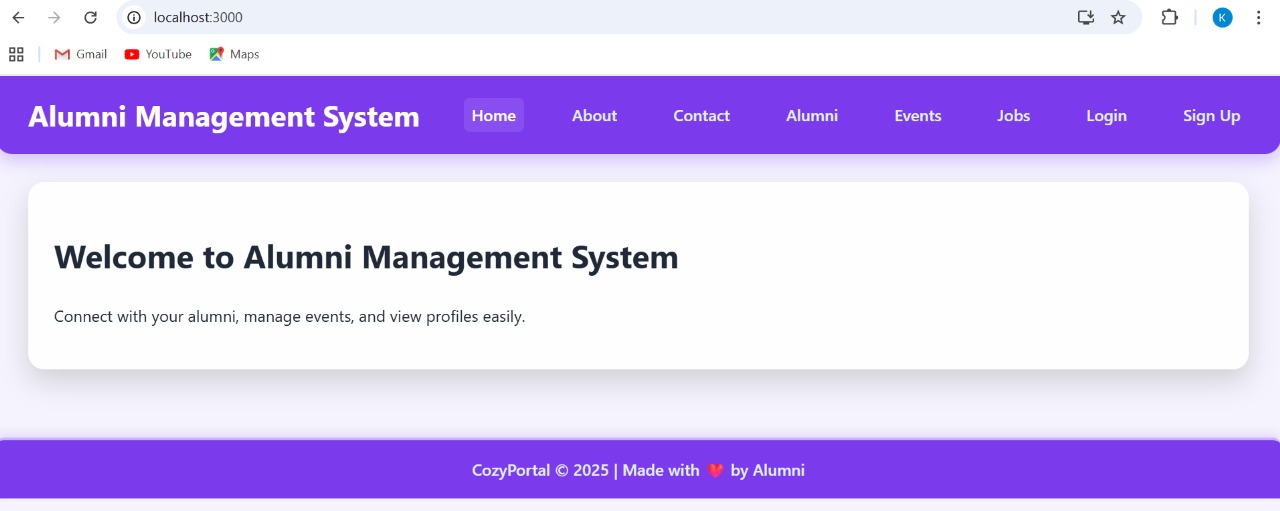
Delete alumni

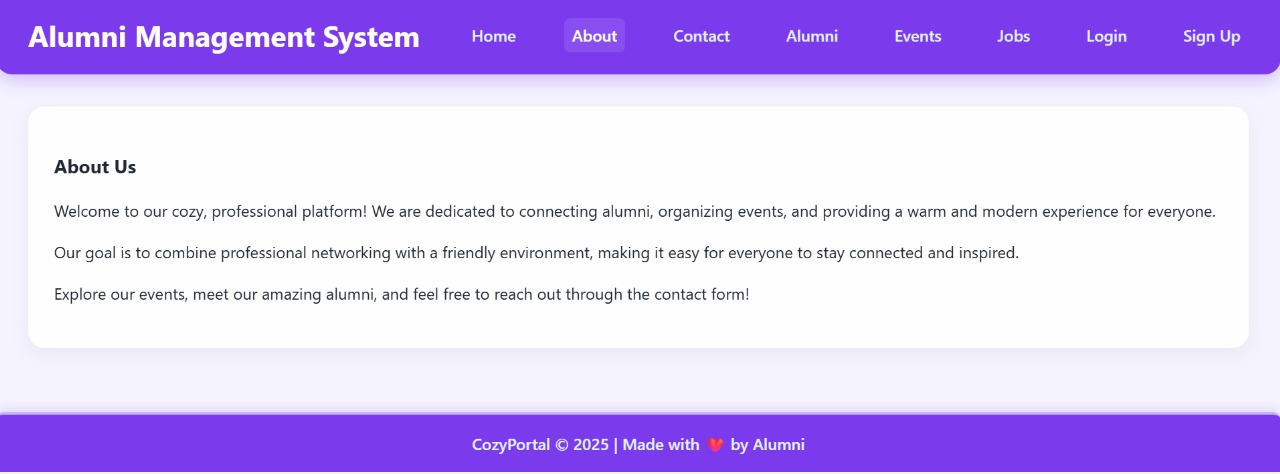
@app.delete("/api/alumni/{alumni\_id}", status\_code=status.HTTP\_204\_NO\_CONTENT) def delete\_alumni(alumni\_id: int, db: Session = Depends(get\_db)): alumni = db.query(models.Alumni).filter(models.Alumni.id == alumni\_id).first() if not alumni: raise HTTPException(status\_code=404, detail="Alumni not found.")

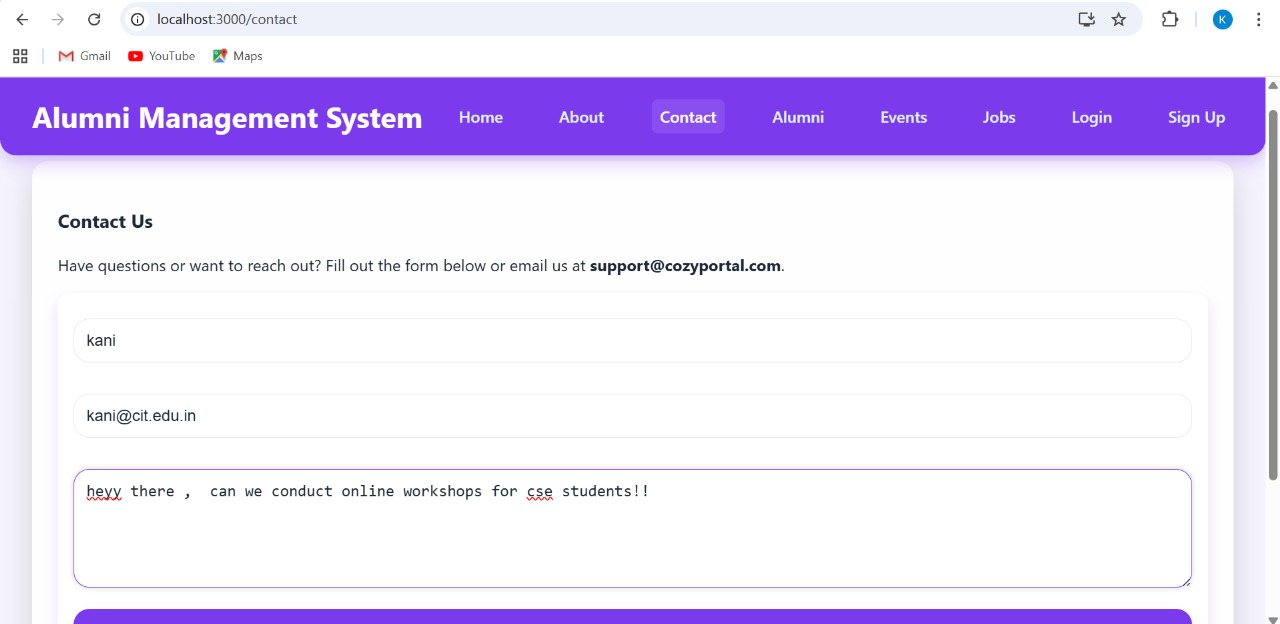
db.delete(alumni)  
db.commit()  
return {"message": "Alumni deleted successfully"}

**SCREENSHOTS :**

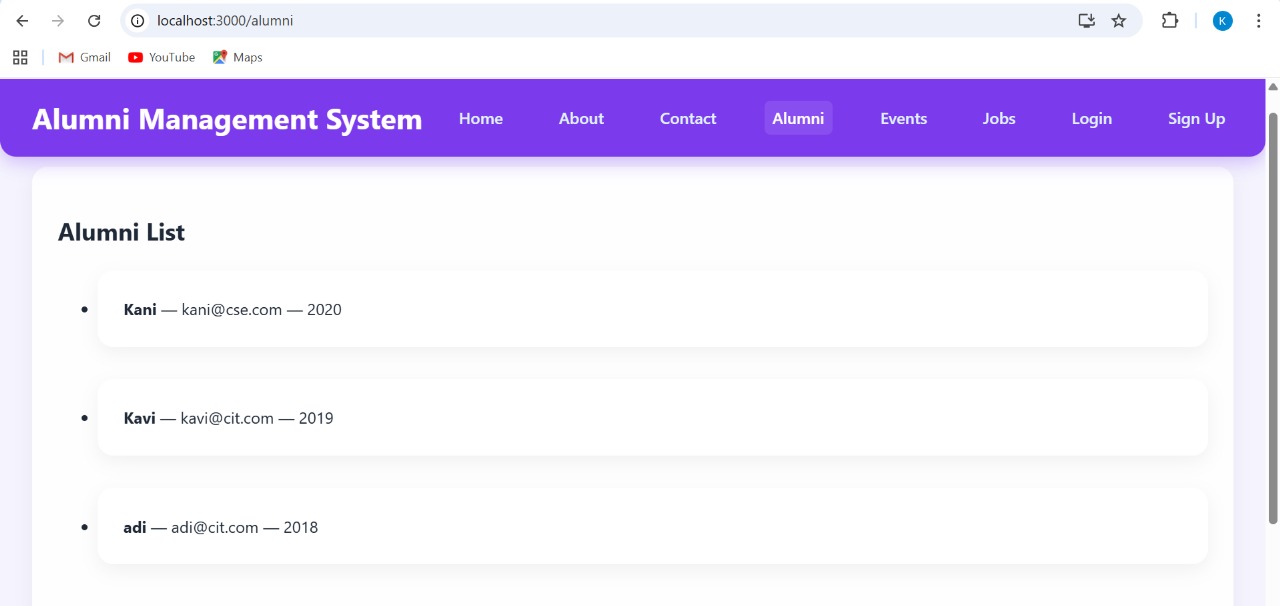
**Home Page**

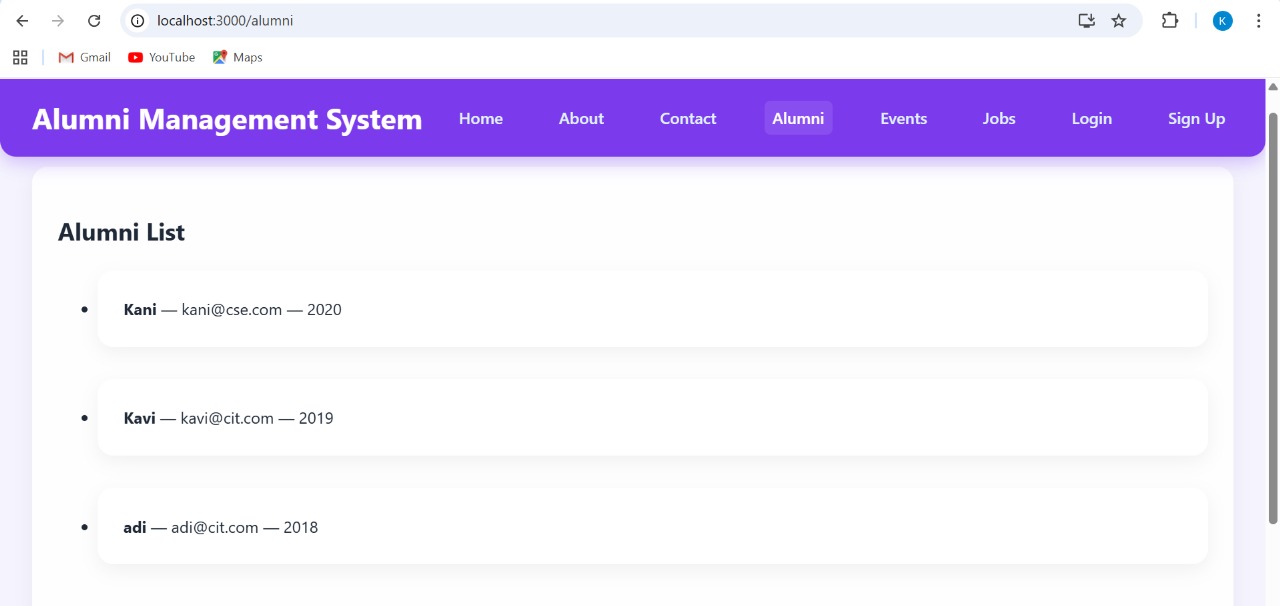


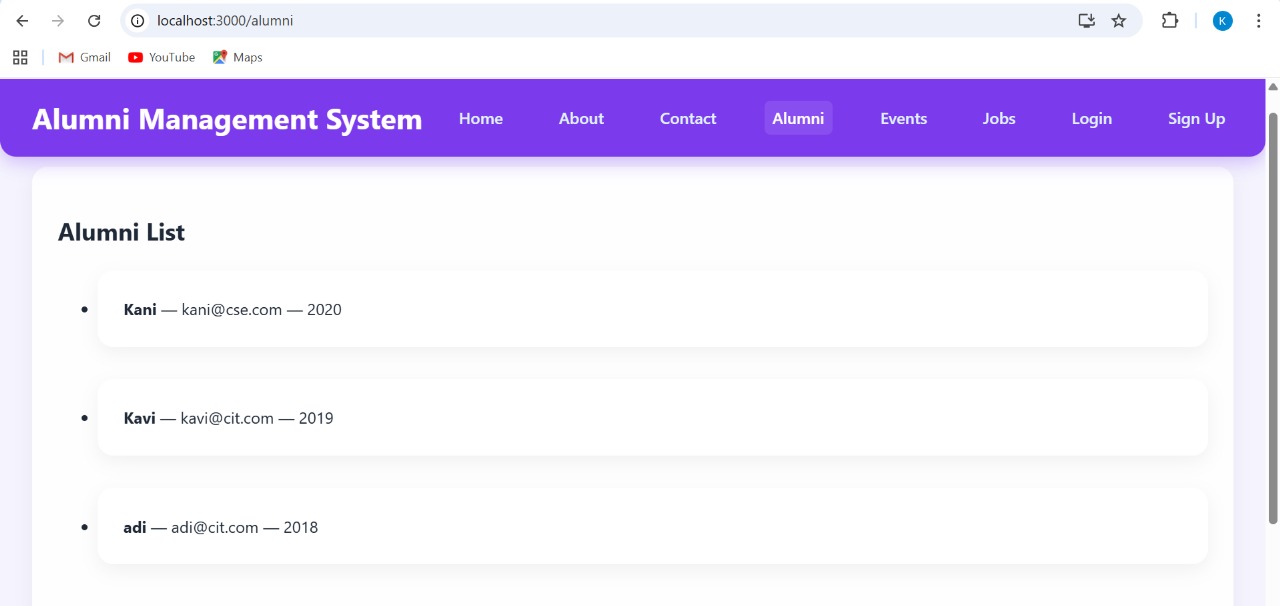
**About Page**

**Contact Page**

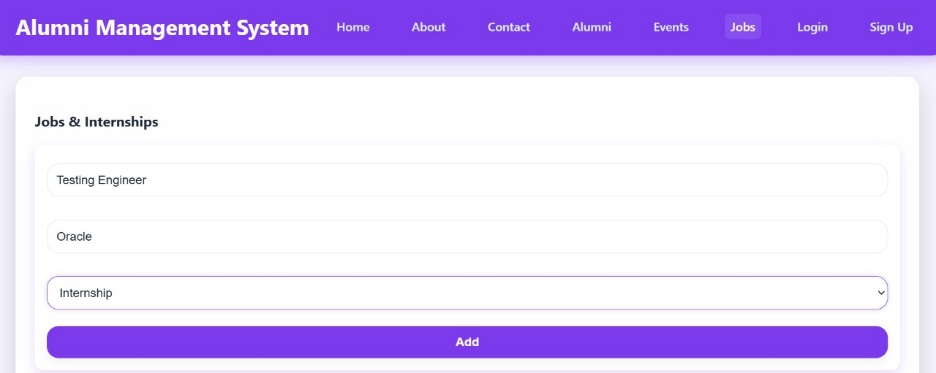
**Alumni Page**



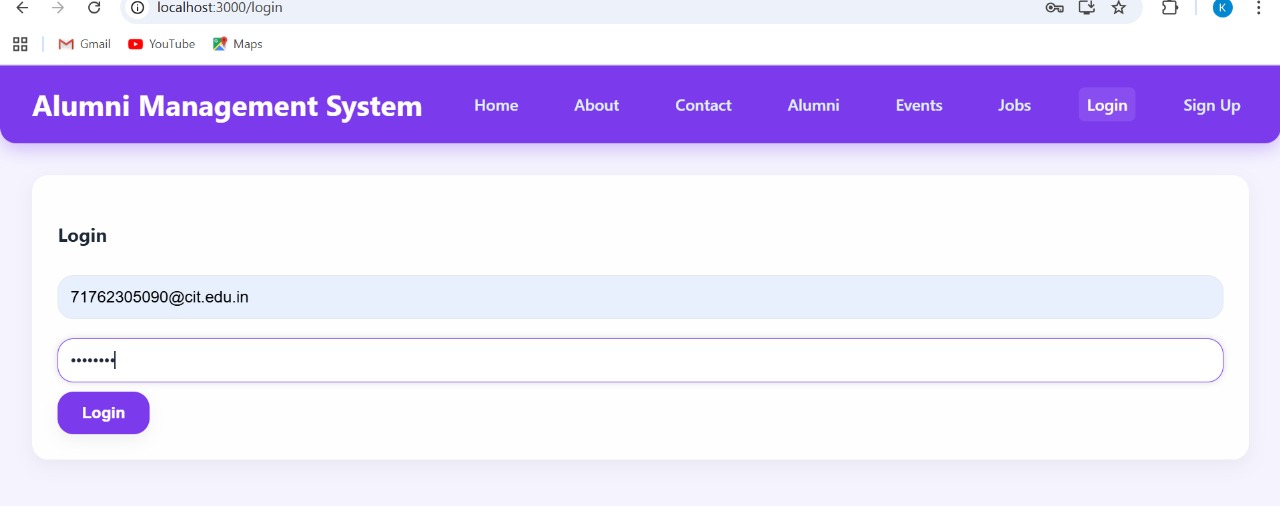




**Job Internships Page**



**Login Page**



**Register Page**

