**Alumni Management System**

**A Comprehensive Platform for Alumni Engagement and Event Management**

**1st Author**

\_\_\_\_\_\_\_\_\_\_

Email:

**2nd Author**

\_\_\_\_\_\_\_\_\_\_

Email:

### ****Abstract****

The Alumni Management System is a platform designed to connect educational institutions with their alumni. It streamlines alumni registration through document verification, provides secure login access, and offers features like event management and profile updates. Alumni can browse and register for events, while admins review and approve new registrations to ensure authenticity.

With a user-friendly interface and robust backend, the system fosters seamless communication, secure data handling, and scalable architecture for future growth. It strengthens alumni relationships and enhances community engagement through an efficient, reliable, and connected portal.

Contents

[1. Introduction 4](#_Toc182691593)

[2. Methodology Selection (Q1) 5](#_Toc182691594)

[Reason 1: Flexibility and Incremental Development 5](#_Toc182691595)

[Reason 2: Frequent Testing and Feedback 6](#_Toc182691596)

[Conclusion 6](#_Toc182691597)

[3. Feasibility Study (Q2) 7](#_Toc182691598)

[Technical Feasibility 7](#_Toc182691599)

[Economic Feasibility 8](#_Toc182691600)

[Operational Feasibility 9](#_Toc182691601)

[Schedule Feasibility 9](#_Toc182691602)

[4. User Needs and Constraints (Q3) 11](#_Toc182691603)

[Requirements Gathering Techniques 11](#_Toc182691604)

[Justification for Techniques 13](#_Toc182691605)

[5. Activity Diagram (Q4) 15](#_Toc182691606)

[Alumni System Core Activities: 15](#_Toc182691607)

[Event Management: 15](#_Toc182691608)

[6. Perimeter of the To-Be System (Q5) 18](#_Toc182691609)

[Perimeter of the To-Be System 18](#_Toc182691610)

[7. Functional and Non-Functional Requirements (Q6) 19](#_Toc182691611)

[Functional Requirements 19](#_Toc182691612)

[Non-Functional Requirements 20](#_Toc182691613)

[8. Use Case Diagram (Q7) 20](#_Toc182691614)

[Actors 20](#_Toc182691615)

[Use Cases 21](#_Toc182691616)

[Relationships 21](#_Toc182691617)

[Sample Diagram Description 21](#_Toc182691618)

[9. System Sequence Diagram (Q8) 22](#_Toc182691619)

[Use Case: Register/Login (Sprint 1) 22](#_Toc182691620)

[Use Case: Event Management (Sprint 3) 23](#_Toc182691621)

[Use Case: Job Postings (Sprint 4) 25](#_Toc182691622)

[10. Domain Class Diagram Fragments for Each Sprint/Iteration (Q9) 28](#_Toc182691623)

[Sprint 1: User Registration and Login 28](#_Toc182691624)

[Sprint 2: Profile Management 28](#_Toc182691625)

[Sprint 3: Event Management 29](#_Toc182691626)

[Sprint 4: Job Postings 30](#_Toc182691627)

[11. Merged and Refined Domain Class Diagram (Q10) 31](#_Toc182691628)

[Classes 31](#_Toc182691629)

[Relationships 32](#_Toc182691630)

[12.Architectural Pattern for the To-Be System (Q11) 33](#_Toc182691631)

[Rationale for Choosing MVC 33](#_Toc182691632)

# 1. Introduction

The **Alumni Management System** is designed as a comprehensive platform that enables alumni to stay connected with their alma mater and fellow graduates. This system provides a centralized digital space where alumni can engage with one another, stay informed about institutional updates, participate in events, and access exclusive job opportunities posted by the institution or by alumni themselves.

The primary functionalities of the Alumni Management System include:

* **User Registration and Profile Management**: Alumni can create personal profiles, update their information, and showcase their career accomplishments. This serves as a professional networking tool within the alumni community.
* **Event Notifications and Participation**: The system provides a calendar of events, such as reunions, webinars, and workshops. Alumni can view event details, RSVP, and stay updated with event notifications.
* **Job Postings and Career Opportunities**: Alumni and institutional partners can post job openings, allowing graduates to explore career opportunities and network professionally within their community.
* **Alumni Directory and Search**: The system maintains an organized directory where alumni can search for and connect with others based on graduation year, industry, location, and other criteria.

The purpose of this report is to analyze, design, and present a structured approach to developing the Alumni Management System using the **Agile methodology**. Agile is well-suited to this project because it allows for continuous improvement based on user feedback and ensures that each component of the system meets the needs of alumni and administrators. The report will outline the steps taken in requirements analysis, design modeling, and system architecture to deliver a robust, user-centered solution that enhances alumni engagement and institutional relationships.

# 2. Methodology Selection (Q1)

For the development of the **Alumni Management System**, the **Agile methodology** has been selected as the most suitable approach. Agile is a highly flexible and iterative framework that focuses on delivering functional components incrementally while incorporating continuous feedback from stakeholders. This methodology is especially advantageous for projects where user requirements might evolve over time or need refinement based on real-world feedback. The following points justify this choice:

## Reason 1: Flexibility and Incremental Development

The Agile methodology emphasizes flexibility and incremental development, allowing the project to adapt to changes in requirements efficiently. In the context of the Alumni Management System, it is likely that alumni and administrators may identify additional features or modifications during the development process. For example:

* Alumni might request the addition of a networking feature after initial feedback.
* Administrators could propose enhancements to event management capabilities as they see how the system works in practice.

Agile supports this by delivering the system in smaller, manageable increments known as **sprints**, with each sprint focusing on a specific feature or functionality. This ensures that critical features, such as **user registration** and **event notifications**, are delivered early, allowing users to provide feedback that shapes subsequent development phases.

## Reason 2: Frequent Testing and Feedback

Agile facilitates frequent testing and feedback, ensuring that the system aligns closely with user expectations and needs. For an Alumni Management System, engaging alumni and institutional stakeholders throughout the project is essential to its success. Agile practices like regular sprint reviews and user feedback sessions help in:

* Validating that implemented features, such as **job postings**, meet user requirements.
* Identifying and resolving usability issues early in the development cycle, reducing the risk of delivering a system that users find difficult to adopt.

For example, the feedback from alumni during a sprint review might highlight the need for better search filters in the **Alumni Directory**. Agile enables the development team to address such feedback in the next sprint, ensuring a user-centered design approach.

## Conclusion

The adaptability and user-focused nature of Agile make it the ideal methodology for developing the Alumni Management System. By delivering functional components incrementally and incorporating continuous feedback, Agile ensures that the final system is both effective and aligned with the evolving needs of alumni and institutional stakeholders.

### ****References****

1. Opentext.com. Why Agile? Retrieved from <https://www.opentext.com/what-is/agile-development>
2. Turk, Daniel, and Robert France. "Assumptions Underlying Agile Software Development Processes." Journal of Software and Systems Modeling 17 (2003): 19-25. Print.

# 3. Feasibility Study (Q2)

The feasibility study for the **Alumni Management System** evaluates the practicality of the project from technical, economic, operational, and schedule perspectives. This analysis ensures that the system is realistic, achievable, and beneficial for all stakeholders involved.



## ****Technical Feasibility****

The technical feasibility examines the tools, technologies, and infrastructure required to build and operate the Alumni Management System.

* **Development Tools**: The system can be built using modern web technologies such as:
  + **Django Framework**: For building a robust and scalable backend.
  + **HTML, CSS, JavaScript**: For a responsive and user-friendly frontend.
  + **Bootstrap**: For consistent and attractive UI components.
* **Database**: A relational database like **PostgreSQL** or **MySQL** or will manage alumni profiles, event details, and job postings efficiently.
* **Hosting and Deployment**: The application can be hosted on platforms like **AWS**, **Heroku**, or **Azure**, offering scalability and reliability.
* **Security Protocols**:

Use **HTTPS** for secure communication between users and the server.

Implement data encryption for sensitive information such as passwords and personal data.

Apply role-based access control to ensure only authorized users (e.g., administrators) can access certain features.

With these tools and technologies, the Alumni Management System can be developed and maintained efficiently while ensuring high performance and security.

## ****Economic Feasibility****

The economic feasibility assesses the costs involved in developing and operating the system. Since the project assumes access to educational resources, the budget is optimized:

* **Development Tools**: Most tools required (e.g., Django, Python, PostgreSQL) are open-source and free.
* **Hosting**: A small-scale application can start with free or low-cost hosting options:
  + Free tier options in **Heroku** or **AWS Free Tier**.
  + Paid plans for increased capacity if necessary (e.g., ~$10/month for a basic hosting plan).
* **Development Team**: As students, the development cost is absorbed as part of the academic project.

### Estimated Budget:

|  |  |
| --- | --- |
| Expense | Cost Estimate |
| Development Tools | Free |
| Hosting (initial) | $0 - $120/year |
| Miscellaneous | $50 |
| **Total** | ~$120-$170 |

This demonstrates that the system is economically feasible with minimal costs involved.

## ****Operational Feasibility****

The operational feasibility assesses how effectively the system meets its objectives and benefits stakeholders:

### **Benefits for Alumni**:

* + Enhances alumni networking by providing a platform to connect with peers.
  + Increases access to career opportunities through job postings and referrals.
  + Simplifies participation in events through notifications and online registrations.

### **Benefits for Administrators**:

* + Streamlines communication with alumni through centralized data management.
  + Offers tools to organize and promote events more efficiently.
  + Facilitates alumni engagement, boosting institutional reputation.

These benefits make the Alumni Management System an essential tool for fostering stronger relationships between alumni and the institution.

## ****Schedule Feasibility****

The schedule feasibility outlines a timeline for project completion aligned with Agile sprints. The project can be completed in **12-15 weeks**, divided into sprints focusing on core functionalities:

|  |  |  |
| --- | --- | --- |
| Sprint Number | Duration | Key Deliverables |
| Sprint 1 | Weeks 1-2 | User registration and profile management |
| Sprint 2 | Weeks 3-4 | Alumni directory and search functionality |
| Sprint 3 | Weeks 5-6 | Event notifications and registrations |
| Sprint 4 | Weeks 7-8 | Job postings and career opportunities module |
| Sprint 5 | Weeks 9-10 | User feedback integration and testing |
| Sprint 6 | Weeks 11-12 | Final testing, deployment, and documentation |

This iterative approach ensures that the system’s critical features are delivered early and refined based on feedback, making the schedule feasible and aligned with Agile practices.

### ****Conclusion****

The feasibility study confirms that the Alumni Management System is technically achievable, economically viable, operationally beneficial, and schedulable within the given timeframe. With careful planning and execution, the system can successfully meet its objectives and provide long-term value to its users.

# 4. User Needs and Constraints (Q3)

Understanding the needs and constraints of users is critical to developing a successful **Alumni Management System**. To achieve this, two **requirements-gathering techniques**—**interviews** and **surveys**—are employed to ensure comprehensive and accurate insights into user expectations and system requirements.

## ****Requirements Gathering Techniques****

Interviews  
Interviews are conducted with two primary stakeholder groups: **alumni** and **administrators**. These interviews are designed to gather detailed and qualitative information about the desired system features and functionalities.

#### ****Targeted Questions for Alumni****:

* "What features would you like to see in an Alumni Management System?"
* "How often would you use this platform, and what would motivate you to engage with it?"
* "Would you prefer real-time notifications for events or periodic updates?"

#### ****Targeted Questions for Administrators****:

* "What challenges do you face in managing alumni data and events?"
* "What features would make your work easier, such as automated reports or streamlined event management tools?"
* "How can the system assist in improving alumni engagement with the institution?"

Surveys  
Surveys are distributed to a larger group of alumni to collect quantitative data on preferences and feature priorities. These surveys consist of a mix of:

#### Closed-Ended Questions:

* "Which feature would you prioritize the most?" (Options: Event notifications, Job postings, Networking tools, Other).
* "How frequently do you expect updates from the platform?" (Options: Daily, Weekly, Monthly).

#### Likert Scale Questions:

* "Rate the importance of the following features on a scale of 1 to 5: (e.g., Profile Customization, Job Postings, Event Management)."

#### Open-Ended Questions:

* "What additional features would you like to suggest for the platform?"
* Surveys provide insights into the broader alumni community’s preferences, enabling the identification of features and designs that cater to a diverse user base.

### Observation

**Observation** involves monitoring the current processes and interactions of stakeholders (e.g., administrators managing alumni data and alumni engaging with institutional systems or events) to identify inefficiencies, challenges, and opportunities for improvement. This technique is particularly useful when stakeholders may not be fully aware of or articulate about their pain points or preferences.

#### ****How Observation is Conducted****

**For Administrators**:

* Observe how administrators currently manage alumni records, events, and communications (e.g., using spreadsheets, emails, or third-party tools).
* Take note of repetitive tasks, delays, or any challenges in their workflows.
* Example Observations:
  + Administrators often spend time manually updating alumni contact details.
  + Event notifications are sent inconsistently due to lack of a centralized system.

**For Alumni**:

* Observe alumni behavior during institutional events or while interacting with existing systems (if any).
* Identify points of friction, such as difficulties in event registration or lack of information about job opportunities.
* Example Observations:
  + Alumni often ask repetitive questions about event details, indicating inadequate communication.
  + Many alumni express frustration at the absence of a searchable directory for finding peers.

### ****Insights Gained from Observation****

**For Administrators**:

* + Need for automation in tasks like event reminders and data updates.
  + Requirement for a centralized dashboard to streamline alumni management.

**For Alumni**:

* + Preference for real-time event notifications via email or app.
  + Desire for features like a professional networking tool or peer search filters.

## ****Justification for Techniques****

Interviews  
Interviews are effective for gathering detailed, qualitative feedback directly from stakeholders. This technique:

* + Provides an in-depth understanding of specific user needs and constraints.
  + Enables follow-up questions to clarify responses, ensuring accuracy.
  + Captures unique perspectives that might not be revealed through standardized methods like surveys.

For example, during an interview, an administrator may express the need for automated email reminders for event participants—an insight that could shape the notification module.

Surveys  
Surveys complement interviews by collecting data from a larger audience efficiently. This technique:

* + Helps identify trends and commonalities across a diverse alumni base.
  + Quantifies user preferences, making it easier to prioritize features during development.
  + Allows data collection from alumni across different geographic locations, ensuring the system’s design addresses global needs.

For instance, survey results may show that most alumni prefer a **mobile-friendly platform** and value **job postings** more than other features, helping developers focus their efforts on these areas.

### ****Observation****

Observation is an invaluable technique for gathering practical insights that might not surface through interviews or surveys. Here’s why:

1. **Identifies Unspoken Challenges**:
   * Stakeholders may overlook or fail to articulate routine pain points in interviews or surveys. Observation fills this gap by directly capturing the real-time behaviors and interactions of users.
   * For example, observing administrators struggling to send bulk emails reveals the need for automated communication tools.
2. **Validates User Feedback**:
   * Cross-checks the data from interviews and surveys by seeing if stakeholder actions align with their stated needs.
   * For instance, while alumni might say event notifications are adequate, observing event attendance patterns may indicate otherwise.
3. **Reveals Workflow Inefficiencies**:
   * Provides actionable insights into repetitive tasks or bottlenecks in current processes.
   * For example, if administrators frequently switch between multiple tools to manage alumni data, it underscores the need for an integrated system.
4. **Encourages User-Centric Design**:

Observation helps designers empathize with stakeholders, leading to features that address their actual, day-to-day needs

### ****Conclusion****

By combining above techniques, the requirements-gathering process ensures a comprehensive understanding of user needs and system constraints. While interviews provide deep insights from a smaller group, surveys capture broader trends, enabling the design of a user-centered and effective **Alumni Management System**.

# 5. Activity Diagram (Q4)

## Alumni System Core Activities:

* New users can register by providing personal information and uploading verification documents
* Existing alumni can login using their credentials to access the dashboard
* Admin reviews and approves new registrations to ensure authentic alumni access

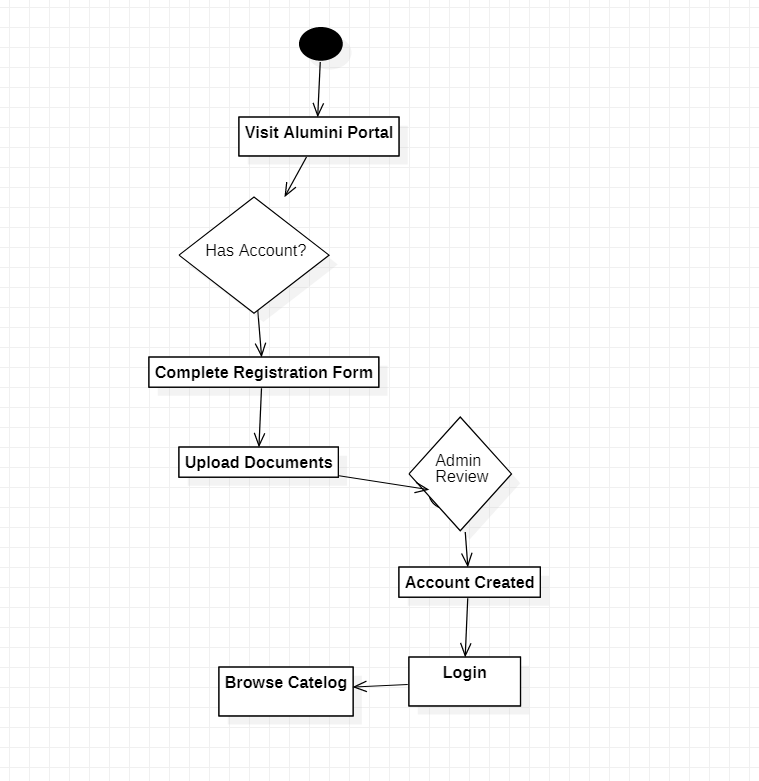
## Event Management:

* Alumni can browse and view upcoming events/reunions
* Register for events (both free and paid options available)
* View their event registration history and status

### Profile Management:

* Alumni can view and update their personal information
* Manage professional details (career, achievements, etc.)
* Update contact information and communication preferences

Alumni System Flow

Login System Activity Diagram

# 6. Perimeter of the To-Be System (Q5)

The **perimeter of the system** defines its scope, including the functionalities and limitations of the **Alumni Management System**. Here's a well-documented answer:

## ****Perimeter of the To-Be System****

The **Alumni Management System** is designed to support the following functionalities while operating within specific constraints:

### ****Functional Scope****

#### User Registration and Authentication:

* + New users can register by providing personal details and uploading verification documents.
  + Existing alumni can log in using their credentials to access personalized features.

#### Event Management:

* + Alumni can browse and register for upcoming events, including free and paid options.
  + View registration history and payment status for past events.
  + Admins can manage event details and participant lists.

#### Profile Management:

* + Alumni can update personal and professional information, including career achievements and contact preferences.

#### Admin Activities:

* + Approve or reject new user registrations to ensure authenticity.
  + Manage alumni data and event workflows.

### Constraints

#### Data Security:

* + Ensure secure user authentication using encrypted passwords and HTTPS for communication.

#### User Roles and Access:

* + Alumni can only access features specific to their role, while admins have permissions to approve users and manage events.

#### Platform Accessibility:

* + The system will be accessible via a web interface, optimized for mobile and desktop use.

### System Boundaries

#### Internal Boundaries:

* + The system encompasses registration, authentication, event management, and profile updates.

#### External Boundaries:

* + Integration with email systems for notifications.
  + Potential linkage with third-party payment gateways for paid event registrations.

### Exclusions

* The system does not include direct messaging or chat features between users.
* Advanced reporting or analytics for admin use is beyond the current scope.

# 7. Functional and Non-Functional Requirements (Q6)

## Functional Requirements

Functional requirements define the system's specific behaviors and funcionalities. Based on the **Alumni Management System** scenario, these are:

1. User Registration and Login:
   * New users can register by providing personal details and uploading verification documents.
   * Existing users can log in using their credentials.
2. Profile Management:
   * Users can update personal and professional information.
   * Alumni can set communication preferences (e.g., email notifications).
3. Event Management:
   * Alumni can browse, view, and register for events.
   * Admins can create, update, and manage event details.
   * Track event participation history for users.
4. Job Postings:
   * Alumni and administrators can post job opportunities.
   * Users can browse and search for jobs based on categories.
5. Admin Functions:
   * Approve or reject new user registrations.
   * Manage alumni data and events.

## Non-Functional Requirements

Non-functional requirements define the system's performance and operational qualities:

1. Performance:
   * The system should handle up to 1,000 concurrent users.
   * Event and job searches should return results within 2 seconds.
2. Security:
   * Use HTTPS for all communications.
   * Encrypt sensitive data such as passwords.
3. Scalability:
   * The system should be easily extendable to add more features in the future.
4. Usability:
   * The user interface should be mobile-responsive and intuitive.
   * Provide error messages and guidance for invalid user inputs.
5. Availability:
   * The system must have a 99.9% uptime guarantee.
6. Data Integrity:
   * Ensure data consistency during user profile updates and event registrations.
7. Compatibility:
   * The system must work across major browsers (Chrome, Firefox, Edge) and devices (desktop, mobile).

# Use Case Diagram (Q7)

A **Use Case Diagram** (UCD) is ideal for visually representing the functional requirements. Here’s how it maps to your scenario:

## ****Actors****

1. **Alumni (User)**: Represents regular users who interact with the system.
2. **Administrator (Admin)**: Responsible for managing users, events, and jobs.

## ****Use Cases****

1. Register/Login:
   * Alumni register by providing information.
   * Admin approves/rejects registrations.
2. Manage Profile:
   * Alumni update personal and professional details.
3. Browse and Register for Events:
   * Alumni browse available events and register (paid/free).
   * Admins create and manage events.
4. Post and Search Jobs:
   * Alumni and admins can post job opportunities.
   * Alumni search for jobs.

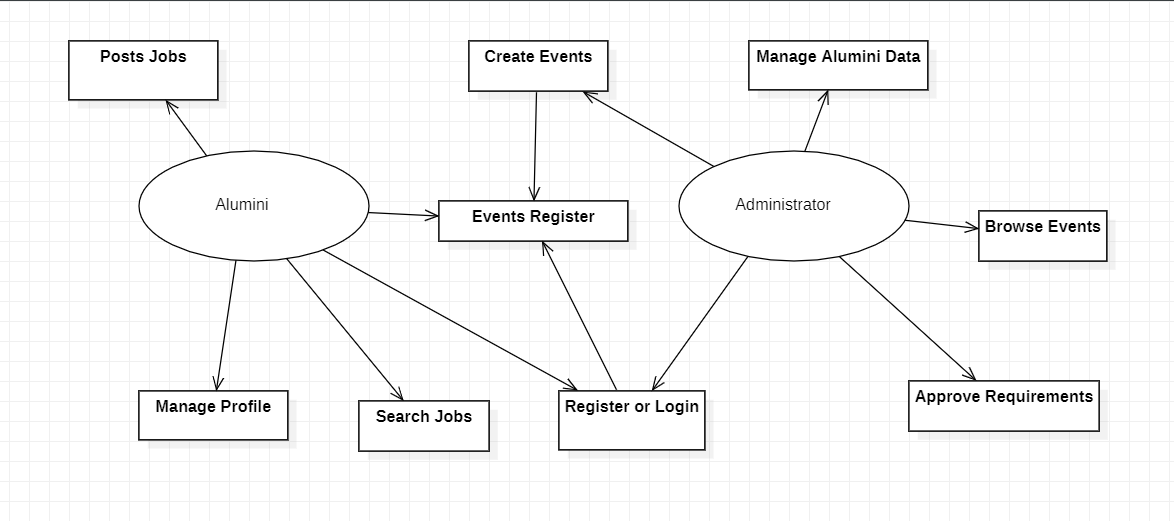
## ****Relationships****

1. **Include**:
   * **Manage Events** includes creating, updating, and deleting events by the admin.
2. **Extend**:
   * Event registration extends "Browse Events" when a user selects an event.

## ****Sample Diagram Description****

Below is how your diagram would look (use a tool like draw.io or StarUML):

1. **Alumni** interacts with:
   * **Register/Login** → **Manage Profile** → **Browse Events** → **Register for Events** → **Search Jobs**
2. **Administrator** interacts with:
   * **Approve Registrations** → **Create Events** → **Manage Alumni Data** → **Post Jobs**



Use Case Diagram

# System Sequence Diagram (Q8)

A **System Sequence Diagram (SSD)** illustrates the interaction between users (actors) and the system for specific use cases. Below is a detailed solution to meet the specified requirements:

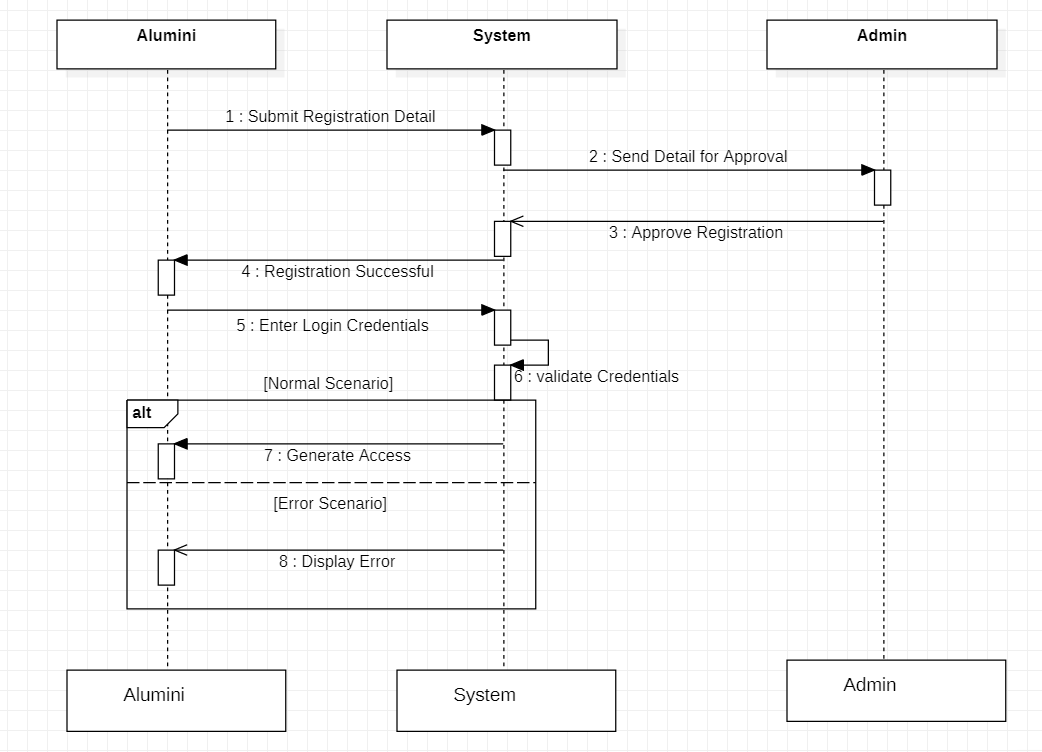
## ****Use Case: Register/Login (Sprint 1)****

### ****Scenario Description****

* **Normal Scenario**:
  1. User (Alumni) requests to register by providing personal information.
  2. System validates the data and sends it to the admin for approval.
  3. Admin approves the registration.
  4. User logs in with valid credentials.
* **Alternative Scenario**: Registration fails due to invalid input.
* **Error Scenario**: User enters incorrect login credentials.

### ****SSD Elements****

* **Participants**:
  1. **Alumni (User)**
  2. **Admin**
  3. **System**
* **Messages**:
  1. Alumni sends registration details to the system.
  2. System forwards the details to the admin.
  3. Admin approves or rejects the registration.
  4. Alumni sends login credentials.
  5. System validates credentials and grants or denies access.
* **Diagram**:



Register/Login Use Case

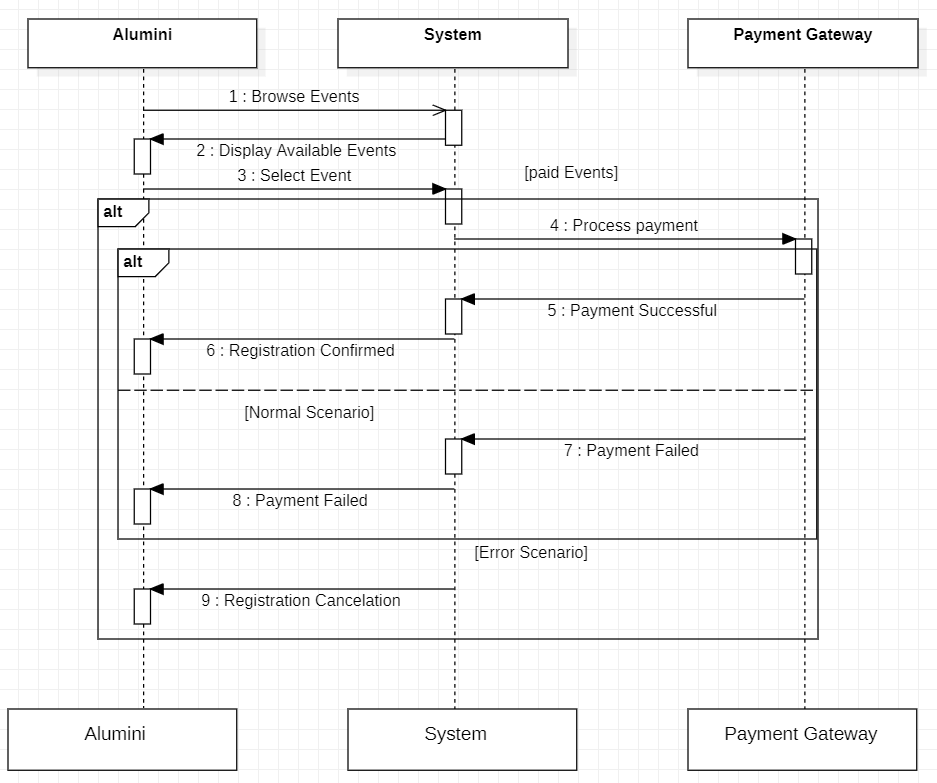
## ****Use Case: Event Management (Sprint 3)****

### ****Scenario Description****

* **Normal Scenario**:
  1. Alumni browses available events.
  2. Alumni selects an event and registers (free or paid).
  3. System confirms registration.
* **Alternative Scenario**: User cancels registration.
* **Error Scenario**: Payment fails for paid events.

### ****SSD Elements****

* **Participants**:
  1. **Alumni (User)**
  2. **Payment Gateway (For Paid Events)**
  3. **System**
* **Messages**:
  1. Alumni sends a request to browse events.
  2. System displays available events.
  3. Alumni selects an event and submits registration details.
  4. For paid events, the system communicates with the payment gateway.
  5. System confirms registration.
* **Diagram**:



Event Management Use Case

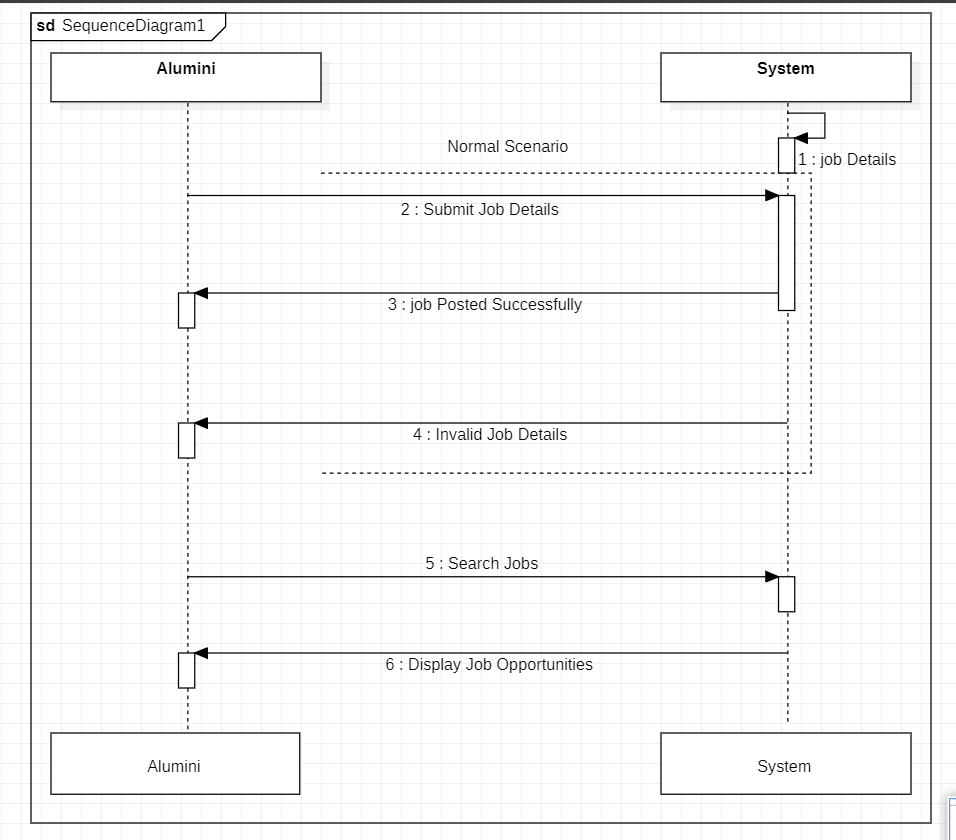
## ****Use Case: Job Postings (Sprint 4)****

### ****Scenario Description****

* **Normal Scenario**:
  1. Alumni posts a job opportunity.
  2. System verifies and stores the job details.
  3. Other users search for jobs and view available opportunities.
* **Alternative Scenario**: User cancels a job posting.
* **Error Scenario**: Invalid job details are rejected.

### ****SSD Elements****

* **Participants**:
  1. **Alumni (Poster)**
  2. **System**
* **Messages**:
  1. Alumni sends job details to the system.
  2. System verifies and stores the details.
  3. Alumni searches for jobs.
  4. System displays matching opportunities.
* **Diagram Structure**



Job Posting Use Case

# 10. Domain Class Diagram Fragments for Each Sprint/Iteration (Q9)

For each sprint/iteration, the domain class diagram focuses on specific functionalities to maintain clarity and incremental development. Below are the fragments for individual sprints/iterations:

## ****Sprint 1: User Registration and Login****

### Classes:

1. **User**
   * **Attributes**:
     + userID: int
     + email: string
     + password: string
     + firstName: string
     + lastName: string
     + isVerified: bool
   * **Methods**:
     + register(email: string, password: string): bool
     + login(email: string, password: string): bool
2. **Admin** (inherits from User)
   * **Attributes**:
     + adminID: int
   * **Methods**:
     + approveUser(userID: int): void
     + rejectUser(userID: int): void

### Relationships:

* **Generalization**: Admin is a subclass of User.
* **Association**: Admin approves/rejects User registrations.

### Diagram Structure:

* **User** → (Generalization) → **Admin**
* **Admin** → (Association) → **User**

## ****Sprint 2: Profile Management****

### **Classes**:

1. **Profile**
   * **Attributes**:
     + profileID: int
     + contactInfo: string
     + careerDetails: string
     + achievements: string
   * **Methods**:
     + updateProfile(profileID: int, details: dict): bool
2. **User** (existing class from Sprint 1)
   * **Relationships**:
     + Aggregation: A User has one Profile.

### **Diagram Structure**:

* **User** ⬤━━━◯ **Profile**

## ****Sprint 3: Event Management****

### Classes:

1. **Event**
   * **Attributes**:
     + eventID: int
     + eventName: string
     + eventDate: date
     + eventType: string
   * **Methods**:
     + createEvent(eventDetails: dict): bool
     + updateEvent(eventID: int, updatedDetails: dict): bool
2. **Registration**
   * **Attributes**:
     + registrationID: int
     + userID: int
     + eventID: int
   * **Methods**:
     + registerForEvent(userID: int, eventID: int): bool

Relationships:

* **Composition**: Registration exists only if an Event exists.
* **Association**: User can register for multiple Events.

### Diagram Structure:

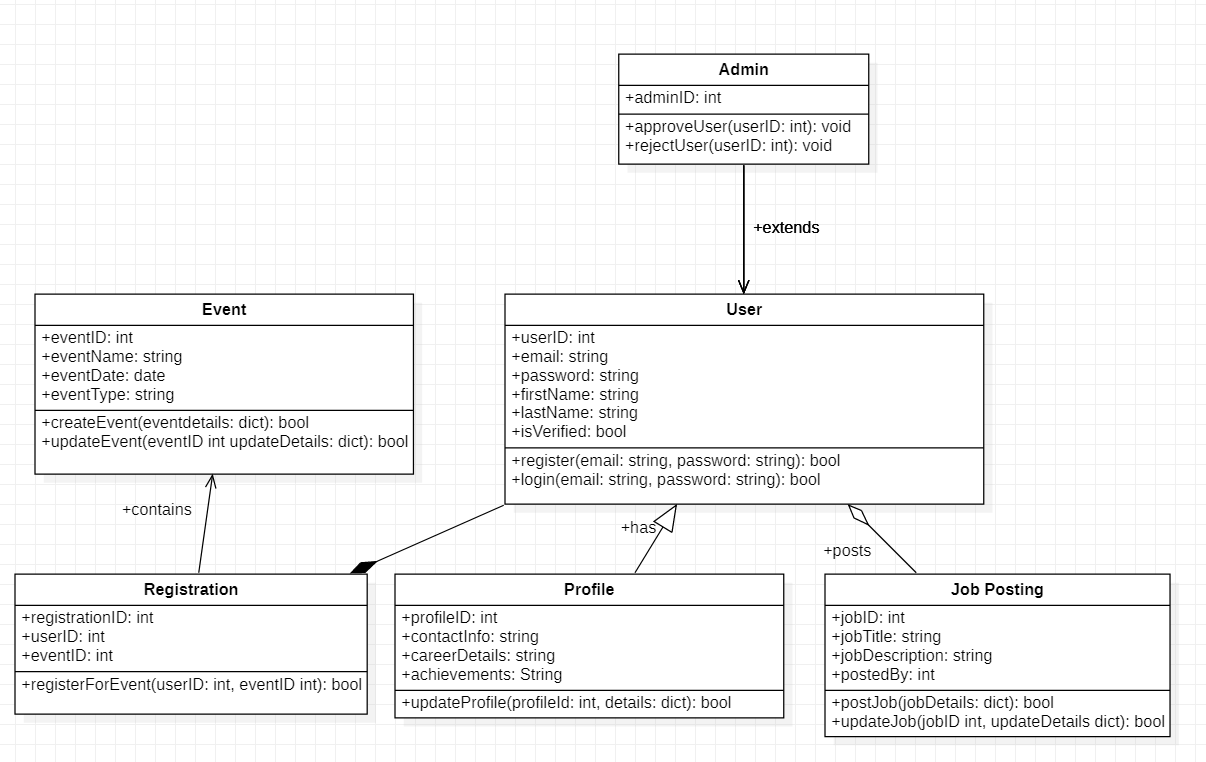
* **Event** ◯━━━⬤ **Registration**
* **User** ━━━ **Registration**

## ****Sprint 4: Job Postings****

Classes:

1. **JobPosting**
   * **Attributes**:
     + jobID: int
     + jobTitle: string
     + jobDescription: string
     + postedBy: int
   * **Methods**:
     + postJob(jobDetails: dict): bool
     + updateJob(jobID: int, updatedDetails: dict): bool
2. **User** (existing class from Sprint 1)
   * **Relationships**:
     + Aggregation: A User can post multiple JobPostings.

### Diagram Structure:

* 

Domain Class Diagram

# 11. Merged and Refined Domain Class Diagram (Q10)

The merged domain class diagram integrates all fragments into a single cohesive structure with low coupling and high cohesion.

## ****Classes****

1. **User**
   * **Attributes**:
     + userID: int
     + email: string
     + password: string
     + firstName: string
     + lastName: string
     + isVerified: bool
   * **Methods**:
     + register(email: string, password: string): bool
     + login(email: string, password: string): bool
2. **Admin** (inherits from User)
   * **Attributes**:
     + adminID: int
   * **Methods**:
     + approveUser(userID: int): void
     + rejectUser(userID: int): void
3. **Profile**
   * **Attributes**:
     + profileID: int
     + contactInfo: string
     + careerDetails: string
     + achievements: string
   * **Methods**:
     + updateProfile(profileID: int, details: dict): bool
4. **Event**
   * **Attributes**:
     + eventID: int
     + eventName: string
     + eventDate: date
     + eventType: string
   * **Methods**:
     + createEvent(eventDetails: dict): bool
     + updateEvent(eventID: int, updatedDetails: dict): bool
5. **Registration**
   * **Attributes**:
     + registrationID: int
     + userID: int
     + eventID: int
   * **Methods**:
     + registerForEvent(userID: int, eventID: int): bool
6. **JobPosting**
   * **Attributes**:
     + jobID: int
     + jobTitle: string
     + jobDescription: string
     + postedBy: int
   * **Methods**:
     + postJob(jobDetails: dict): bool
     + updateJob(jobID: int, updatedDetails: dict): bool

## ****Relationships****

1. **Generalization**:
   * Admin is a subclass of User.
2. **Aggregation**:
   * User ⬤━━━◯ Profile
   * User ⬤━━━◯ JobPosting
3. **Composition**:
   * Event ◯━━━⬤ Registration
4. **Associations**:
   * User ━━━ Registration

### 

Domain Class Diagram

# 12.Architectural Pattern for the To-Be System (Q11)

The **Model-View-Controller (MVC)** architectural pattern is selected to design the **Alumni Management System**. MVC is a widely used pattern for building scalable, maintainable, and user-centric web applications.

## ****Rationale for Choosing MVC****

1. **Separation of Concerns**:
   * MVC divides the system into three interconnected components:
     + **Model**: Handles data and business logic (e.g., alumni profiles, events, and job postings).
     + **View**: Manages the presentation layer and user interface.
     + **Controller**: Processes user input and updates the model or view accordingly.
   * This separation ensures that changes in one layer do not impact others, making the system easier to maintain and extend.
2. **Scalability and Flexibility**:
   * As new features (e.g., advanced reporting, notifications) are introduced, MVC’s modular nature allows these to be added without disrupting existing functionality.
   * For example, adding a new view for mobile users can be done without modifying the model or controller.
3. **Reusability**:
   * Reusable components (e.g., models for user data, views for event displays) streamline development and reduce redundancy.
4. **Ease of Testing**:
   * Each layer can be tested independently. For example:
     + The **model** can be tested for correct data processing.
     + The **controller** can be verified for handling user inputs properly.
5. **Alignment with Tools**:
   * Frameworks like **Django**, which will be used for this system, natively support MVC, ensuring smooth implementation and better utilization of existing libraries.

### ****Conclusion****

The MVC pattern aligns with the project requirements of scalability, maintainability, and user-centric design. By adopting MVC, the Alumni Management System will remain robust, modular, and adaptable to future needs.

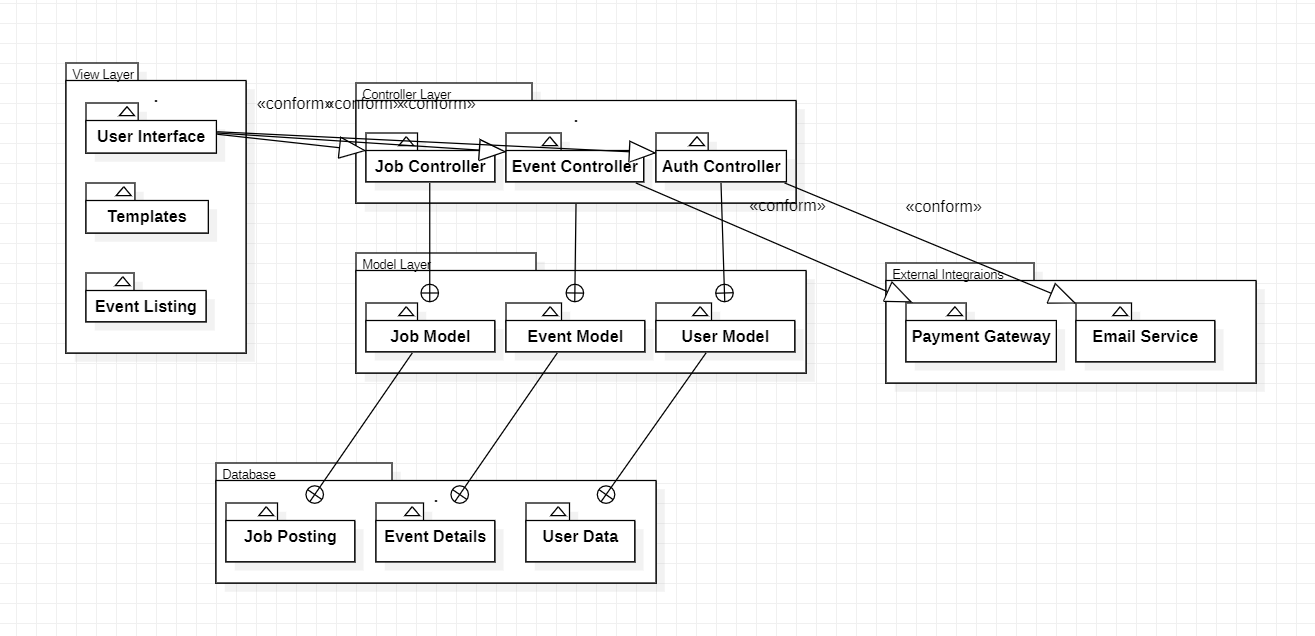
# 13.High-Level Architectural Model of the To-Be System (Q12)

The high-level architectural model represents how different components interact to deliver the system's functionalities based on the MVC pattern.

## ****Components of the System****

1. **Model Layer**:
   * **Purpose**:
     + Manages business logic and data.
     + Interacts with the database.
   * **Key Components**:
     + UserModel: Stores alumni and admin data.
     + EventModel: Handles event details and registration.
     + JobModel: Manages job postings.
2. **View Layer**:
   * **Purpose**:
     + Displays data to the user in an interactive format.
   * **Key Components**:
     + Templates for:
       - User Dashboard.
       - Event Listings and Registration.
       - Job Search and Postings.
3. **Controller Layer**:
   * **Purpose**:
     + Processes user inputs and coordinates between the model and view layers.
   * **Key Components**:
     + AuthController: Handles user registration and login.
     + EventController: Processes event-related actions.
     + JobController: Manages job posting and searching functionality.
4. **Database**:
   * A relational database (e.g., PostgreSQL) stores data for all entities like users, events, and jobs.
5. **Third-Party Integrations**:
   * **Payment Gateway**: For paid event registrations.
   * **Email Service**: For notifications about events and job updates.

## Diagram



High Level Architectural Model