**Experiment 4: State the precondition, postcondition, and function of each use case.**

**1. Manage Users**

**- Precondition:** Admin is logged in.

**- Postcondition:** Users are managed (added, updated, or deleted).

**- Function:**

**- Add User:** Admin can create new user accounts with roles (e.g., student, teacher, admin).

**- Update User:** Admin can modify user details such as name, email, role, and status.

**- Delete User:** Admin can remove users from the system.

**- Search Users:** Admin can search for users by username, email, or role.

**- Security:** Only admin users have access to this functionality.

**2. Profile Settings**

**- Precondition:** User is logged in.

**- Postcondition:** User profile is updated.

**- Function:**

**- Edit Profile Information:** Users can update their name, bio, contact information, profile picture, etc.

**- Change Password:** Users can change their password after entering the current password for validation.

**- Privacy Settings:** Users can adjust privacy settings, such as who can view their profile or contact them.

**- Validation:** Ensure valid input for fields like email, phone number, and password strength.

**3. Log In**

**- Precondition:** User has a registered account.

**- Postcondition:** User is authenticated and logged into the system.

**- Function:**

**- Authentication:** Validate username/email and password.

**- Forgot Password:** Users can reset their password via email link or OTP.

**4. Register Account**

**- Precondition:** User is not logged in.

**- Postcondition:** New user account is created.

**- Function:**

**- User Registration:** Collect necessary details like name, email, password, and optional profile information.

**- Terms and Conditions:** Users must accept terms and conditions before registration.

**- Validation:** Ensure password strength, unique email, and username.

**- Security:** Encrypt passwords and sensitive data before storing in the database.

**5. Create Post**

**- Precondition:** User is logged in.

**- Postcondition:** New post is created and visible to other users.

**- Function:**

**- Text, Image, or Video Posts:** Users can create posts with various content types.

**- Tags and Categories:** Users can add tags to make posts searchable.

**- Privacy Settings:** Users can set posts as public, friends-only, or private.

**6. View Post**

**- Precondition:** User is logged in.

**- Postcondition:** User views the post.

**- Function:**

**- View Post Details:** Display the full content of the post, including images, videos, and comments.

**- Like, Comment, and Share:** Users can interact with posts by liking, commenting, or sharing.

**- Report Post:** Users can report inappropriate content to admins.

**7. Suspend User**

**- Precondition:** Admin is logged in and managing users.

**- Postcondition:** User account is suspended.

**- Function:**

**- Suspend or Reactivate Accounts:** Admin can temporarily disable or re-enable user accounts.

**- Security:** Suspended users are immediately logged out and prevented from logging in until reactivated.

**8. Chatting and Messages**

**- Precondition:** User is logged in.

**- Postcondition:** Messages are sent and received.

**- Function:**

**- Real-Time Messaging:** Users can send instant messages with real-time updates.

**- Group Chats:** Users can create group chats with multiple participants.

**- Media Sharing:** Support for sending images, videos, documents, etc.

**- Emojis and Reactions:** Users can react to messages with emojis.

**9. Jobs**

**- Precondition:** User is logged in.

**- Postcondition:** User views or applies for jobs.

**- Function:**

**- Job Listings:** Users can browse job openings based on category, location, and other filters.

**- Saved Jobs:** Users can bookmark jobs for later.

**- Job Alerts:** Users can set up job alerts based on specific criteria.

**10. Chat Room**

**- Precondition:** User is logged in.

**- Postcondition:** User participates in a chat room.

**- Function:**

**- Join or Create Rooms:** Users can create or join existing chat rooms.

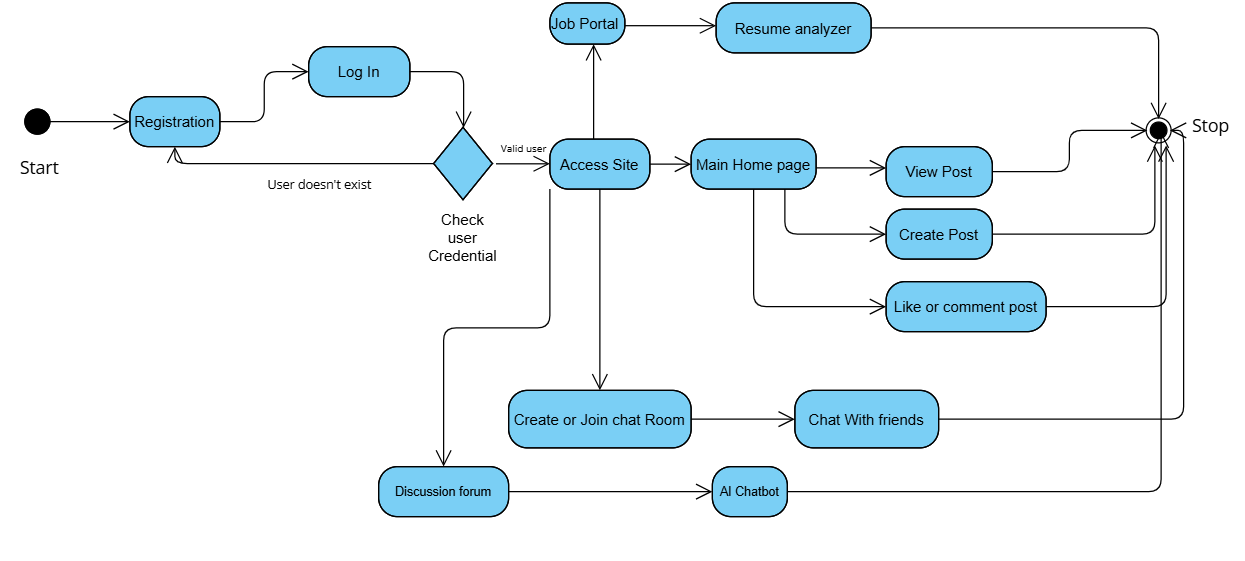
**- Audio Calls:** Users can initiate group calls within the chat room.

**- Whiteboard Feature:** Collaborative whiteboard for drawing or note-taking during discussions.

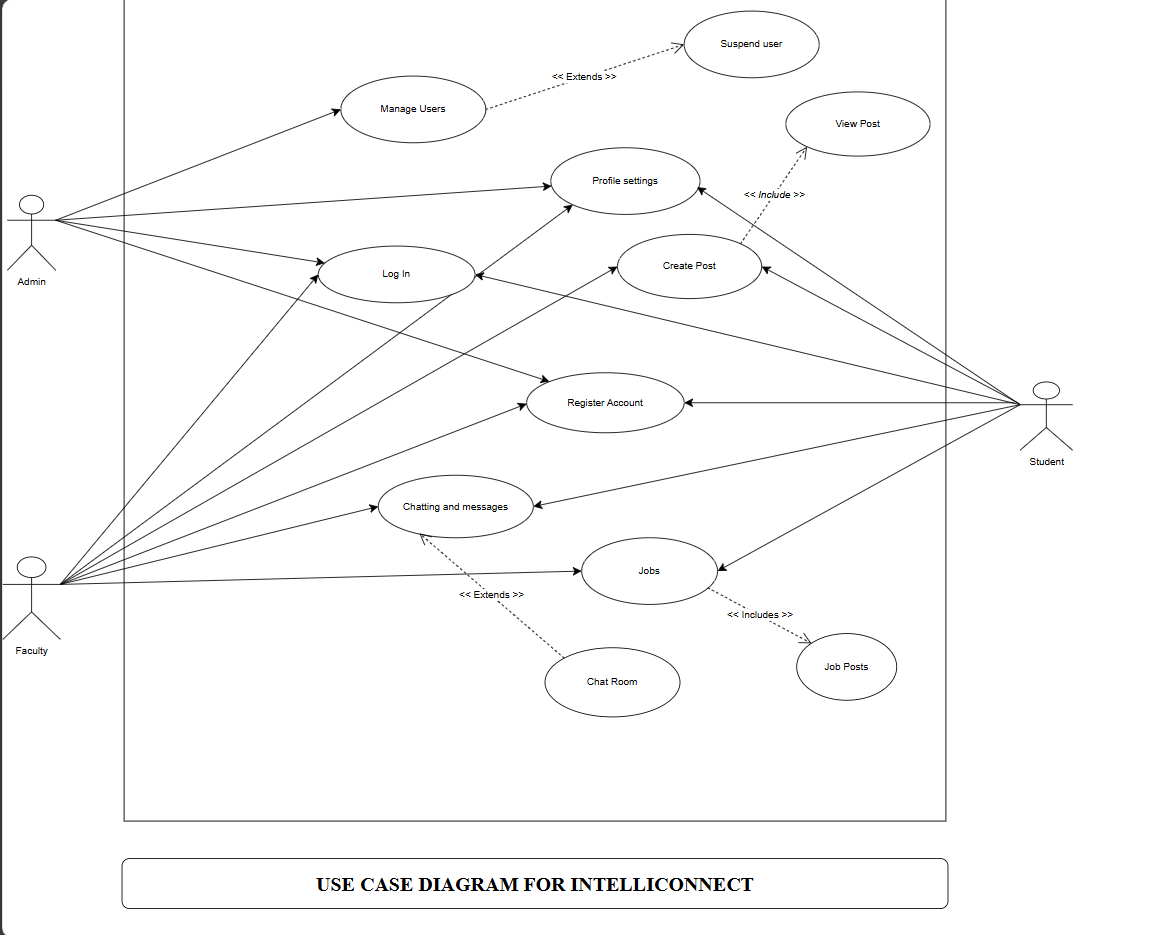
**- Screen Sharing:** Users can share their screens during meetings.

**- Security:** Implement end-to-end encryption for secure communication.

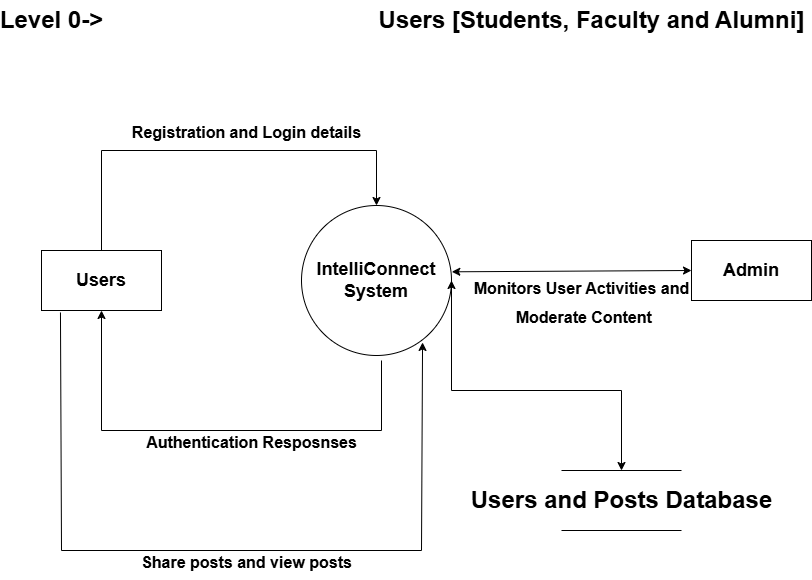
**Experiment 5: Draw Activity Diagram.**



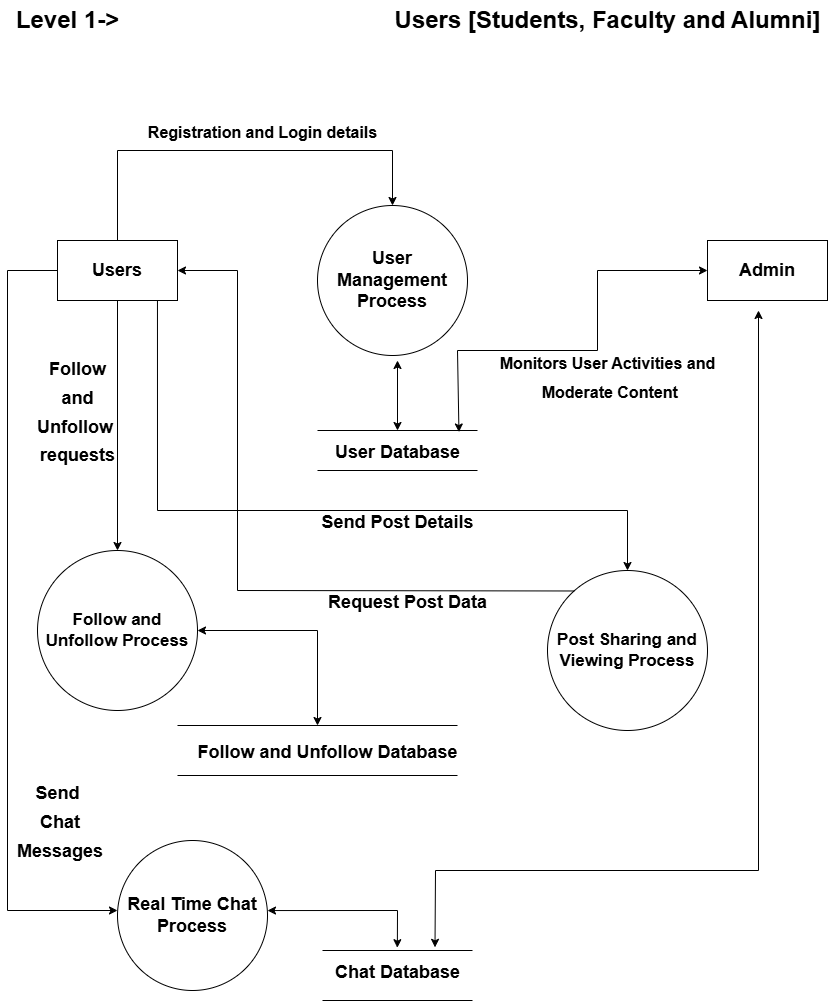
**Experiment 3: Draw Use Case Diagram.**



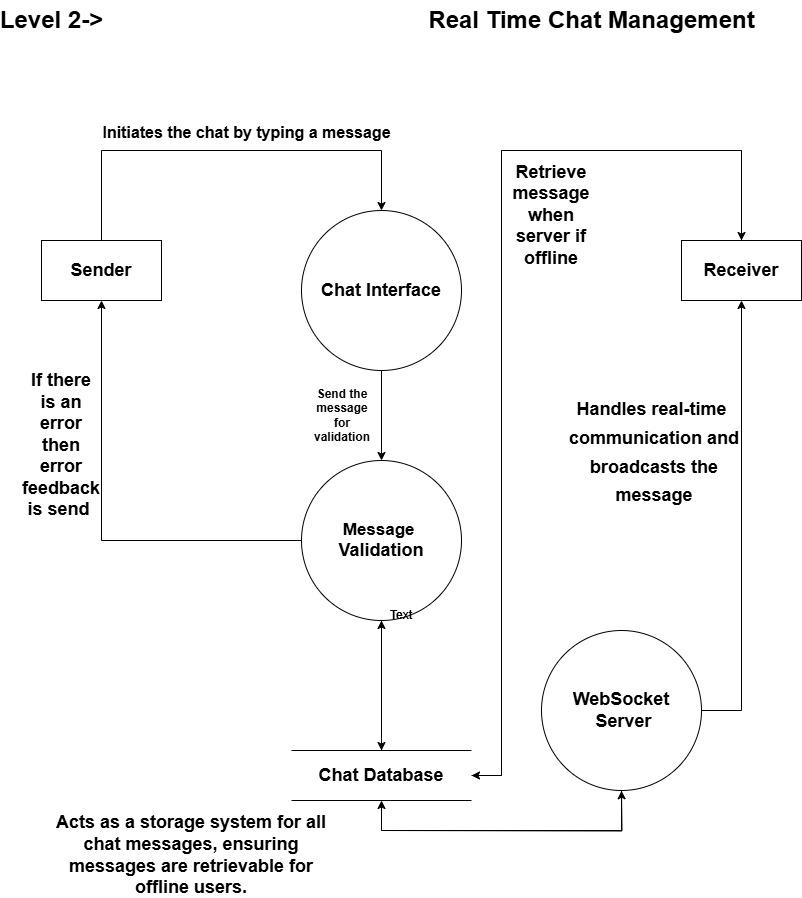
**Experiment 6: Draw Data Flow Diagram.**



**Experiment 6: Draw Data Flow Diagram.**



**Experiment 6: Draw Data Flow Diagram.**

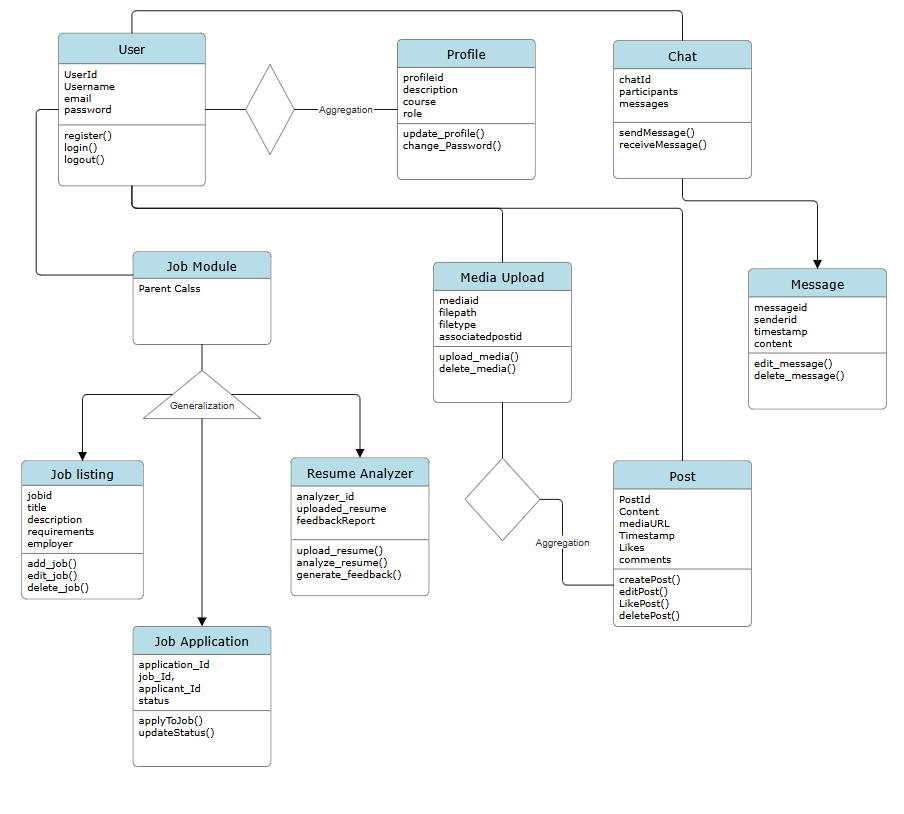


**Experiment 3: Draw Use Case Diagram.**

**The roles of the actors can be described as follows:**

1. **Admin**:
   * Responsible for managing the system and users.
   * Can manage user accounts and has the ability to suspend users.
   * Can log in, adjust profile settings, and create posts.
   * Has overarching access to ensure the platform operates smoothly.
2. **Faculty**:
   * Can register an account, log in, and manage their profile settings.
   * Can participate in chatting and messaging.
   * Can access and post jobs.
   * May use chat room functionality for collaboration.
3. **Student**:
   * Can register an account, log in, and update profile settings.
   * Can view posts, create posts, and access job-related features.
   * Uses chatting and messaging features, including chat rooms, to communicate.
   * Can view job posts and participate in other student-centric activities.

**Experiment 7: Draw Class Diagram.**



**Experiment 7: Identify Strong and Weak Classes in Class Diagram.**

**Strong Classes:**

1. **User**:
   * **Attributes**: userId, username, email, password.
   * **Methods**: register(), login(), logout().
   * **Relationships**:
     + One-to-One with Profile.
     + One-to-Many with Post.
     + Many-to-Many with FollowSystem.
     + One-to-Many with JobApplication.
2. **Post**:
   * **Attributes**: postId, content, mediaURL, timestamp, likes.
   * **Methods**: createPost(), editPost(), deletePost(), likePost().
   * **Relationships**: Many-to-One with User.
3. **Chat**:
   * **Attributes**: chatId, participants, messages.
   * **Methods**: sendMessage(), receiveMessage().
   * **Relationships**:
     + One-to-Many with Message.
     + Many-to-Many with User (participants).
4. **ResumeAnalyzer**:
   * **Attributes**: analyzerId, uploadedResume, feedbackReport.
   * **Methods**: uploadResume(), analyzeResume(), generateFeedback().
   * **Relationships**: One-to-One with User.
5. **JobListing**:
   * **Attributes**: jobId, title, description, requirements, employer.
   * **Methods**: addJob(), editJob(), deleteJob().
   * **Relationships**:
     + One-to-Many with JobApplication.
6. **JobApplication**:
   * **Attributes**: applicationId, jobId, applicantId, status.
   * **Methods**: applyToJob(), updateStatus().
   * **Relationships**:
     + Many-to-One with User.
     + Many-to-One with JobListing.

**Weak Classes:**

1. **Profile**:
   * **Attributes**: profileId, description, college, role, course.
   * **Methods**: updateProfile().
   * **Relationships**: One-to-One with User.
2. **MediaUpload**:
   * **Attributes**: mediaId, filePath, fileType, associatedPostId.
   * **Methods**: uploadMedia(), deleteMedia().
   * **Relationships**: Many-to-One with Post.
3. **Message**:

* **Attributes**: messageId, senderId, timestamp, content.
* **Methods**: editMessage(), deleteMessage().
* **Relationships**: Many-to-One with Chat.

**Aggregation**

* **Definition**: Aggregation represents a "whole-part" relationship where the part can exist independently of the whole.
* **Examples in the Prompt**:
  1. **User ↔ Profile**:
     + *Aggregation*: A User is associated with a Profile, but the Profile can exist independently (e.g., the system could store inactive profiles or profiles without an active user).
  2. **Post ↔ MediaUpload**:
     + *Aggregation*: A Post can have multiple MediaUpload instances, but the media files can exist independently in the cloud or database.

**Generalization**

* **Definition**: Generalization is the process of extracting shared characteristics (attributes or methods) and placing them in a parent class to create a hierarchy.
* **Examples in the Prompt**:
  1. **JobApplication and JobListing**:
     + *Generalization*: Both JobApplication and JobListing could be specialized classes inheriting common attributes or methods (e.g., applicationId, jobId, status) from a generalized JobModule.

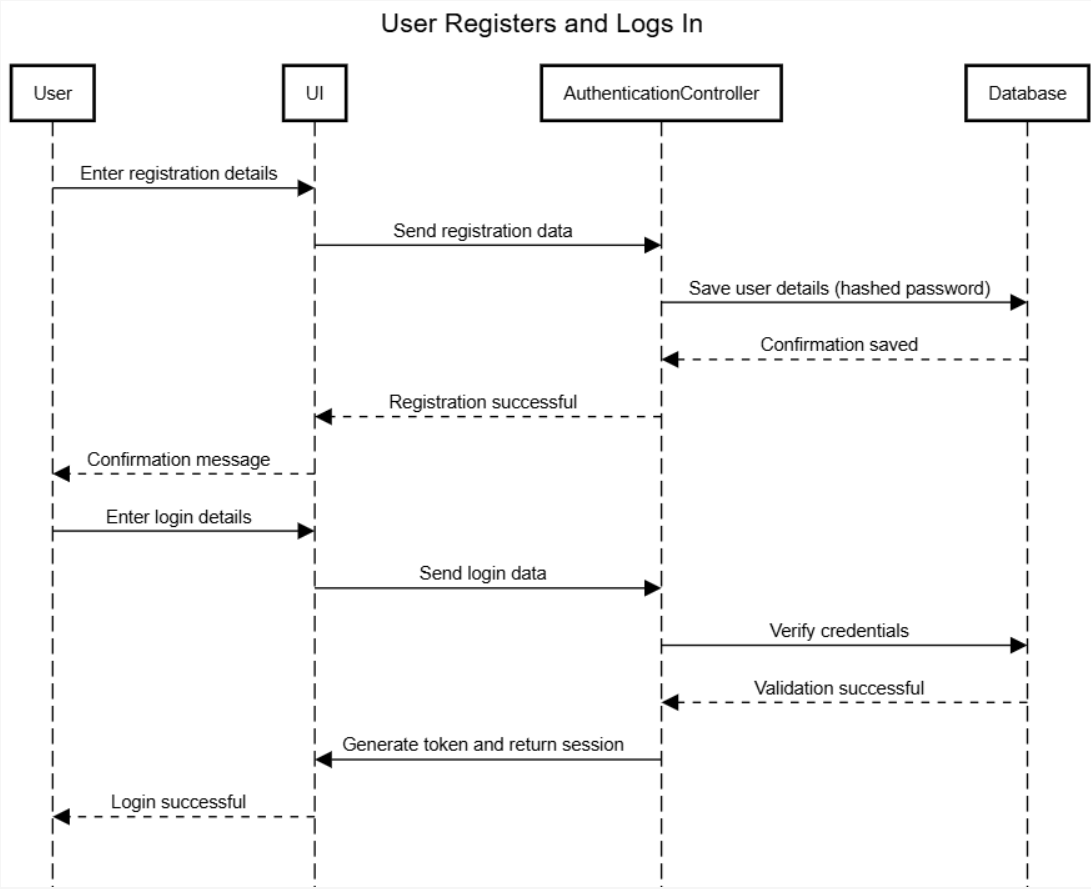
**Experiment 8: Sequence Diagram For any two scenarios.**

**Scenario 1: User Registers and Logs In**

**Description**: A new user creates an account and logs into the system.

**Steps to Create the Sequence Diagram:**

1. **Actors and Objects**:
   * Actor: User
   * Objects: User Interface (UI), Authentication Controller, Database
2. **Steps in the Sequence:**
   * User enters registration details in the UI.
   * UI sends the registration details to the Authentication Controller.
   * Authentication Controller validates the input and hashes the password.
   * The hashed password and user details are saved in the Database.
   * A confirmation message is sent back to the UI.
   * User logs in by entering credentials in the UI.
   * UI sends login details to the Authentication Controller.
   * The controller verifies credentials with the Database.
   * If valid, a session token is generated and returned to the UI.



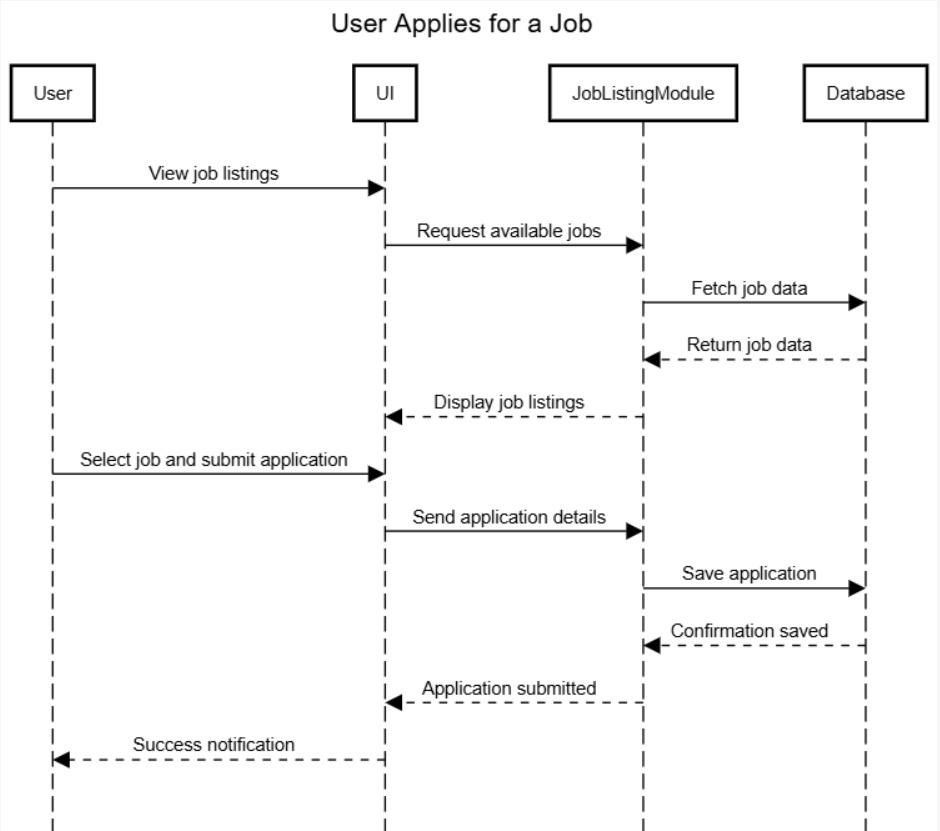
**Experiment 8: Sequence Diagram For any two scenarios.**

**Scenario 2: User Applies for a Job**

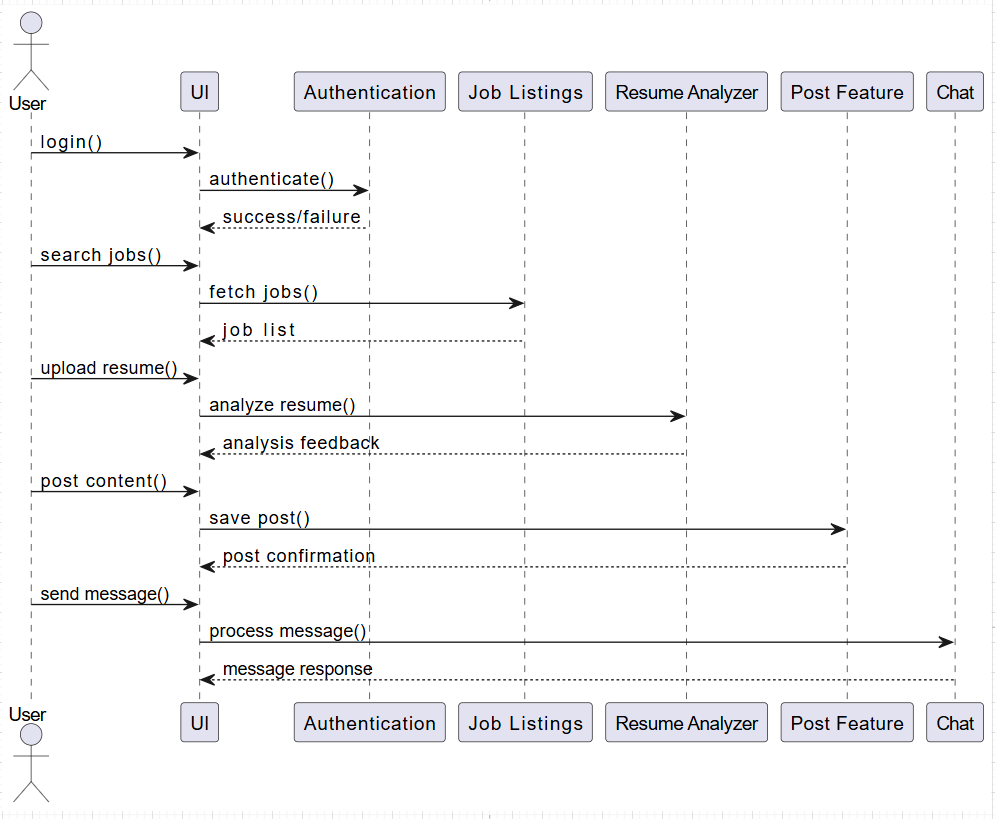
**Description:** A user views a job listing and submits an application.

**Steps to Create the Sequence Diagram:**

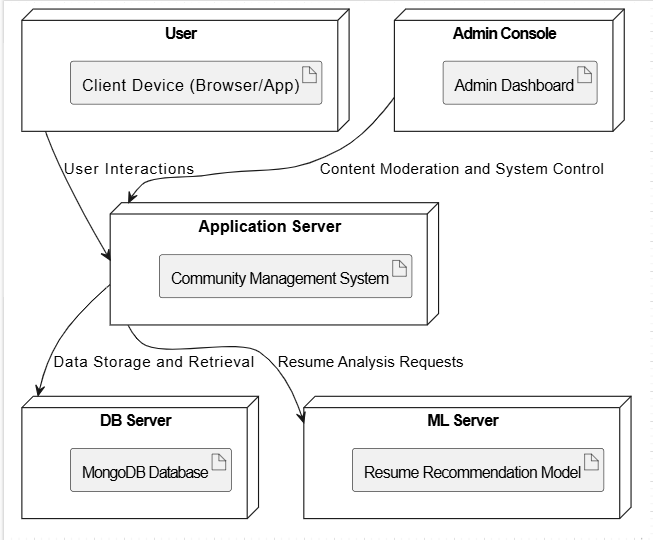
1. **Actors and Objects:**
   * Actor: User
   * Objects: User Interface (UI), Job Listing Module, Database
2. **Steps in the Sequence:**
   * User views job listings on the UI.
   * UI requests available job data from the Job Listing Module.
   * The Job Listing Module retrieves job data from the Database and displays it in the UI.
   * User selects a job and submits an application.
   * The application details are sent to the job Listing Module.
   * The job Listing Module validates the data and saves the application in the Database.
   * A success notification is sent back to the UI.



**Experiment 9: Draw the collaboration Diagram.**



**Experiment 10: Designing The Deployment Diagram.**



**Experiment 11: Designing a Test Suite.**

| **Test Case ID** | **Test Case Description** | **Test Steps** | **Expected Result** | **Pass/Fail** |
| --- | --- | --- | --- | --- |
| TC-001 | User login functionality | 1. Open the IntelliConnect app.2. Enter valid credentials.3. Click on "Login." | User is successfully logged in and redirected to the dashboard. | Pass |
| TC-002 | Invalid user login | 1. Open the IntelliConnect app.2. Enter invalid credentials.3. Click on "Login." | Error message “Invalid credentials” is displayed. | Pass |
| TC-003 | User registration | 1. Open the registration page.2. Enter valid user data.3. Click "Register." | User is successfully registered and redirected to the login page. | Pass |
| TC-004 | Profile update | 1. Log in as a user.2. Go to the profile section.3. Update name and email.4. Save changes. | Profile is successfully updated, and changes are reflected in the profile view. | Pass |
| TC-005 | Job application submission | 1. Log in as a user.2. Browse job listings.3. Apply for a job.4. Submit application. | Job application is submitted successfully with confirmation. | Pass |
| TC-006 | Real-time messaging | 1. Log in as a user.2. Send a message to a contact.3. Check if the message is received. | The message is sent and received in real-time without delay. | Pass |
| TC-007 | Admin user access | 1. Log in as an admin.2. Access the admin dashboard.3. Perform admin actions. | Admin has access to all necessary controls and can perform administrative actions. | Pass |
| TC-008 | Admin content moderation | 1. Log in as an admin.2. Review and moderate community posts.3. Delete inappropriate posts. | Admin can successfully moderate posts, deleting those that violate rules. | Pass |
| TC-009 | Resume Analyzer functionality | 1. Log in as a user.2. Upload a resume.3. Analyze the resume. | Resume is analyzed, and feedback is provided on improvements. | Pass |
| TC-010 | Real-time job listing updates | 1. Log in as a user.2. Browse the job listing.3. Wait for job postings to refresh. | New job listings appear in real-time when refreshed. | Pass |
| TC-011 | Performance under heavy load | 1. Simulate 100+ users logging in concurrently.2. Monitor response times and system behaviour. | The system should handle concurrent logins without significant performance degradation. | Pass |
| TC-012 | Database query performance | 1. Perform a search query for job listings.2. Measure response time of the query. | Query results are returned in a reasonable time (e.g., <2 seconds). | Pass |
| TC-013 | Session expiration | 1. Log in as a user.2. Remain idle for the session timeout duration.3. Attempt to perform an action. | User session expires after the specified idle time, and the user is prompted to log in again. | Pass |
| TC-014 | Security - SQL Injection | 1. Open the job search page.2. Attempt to inject SQL code in the search box. | The system rejects the input and does not allow SQL injection. | Pass |
| TC-015 | Security - Password encryption | 1. Register as a new user.2. Check the user password in the database (through admin tools). | Password is stored in an encrypted format, not in plain text. | Pass |
| TC-016 | Usability - Navigation | 1. Log in as a user.2. Navigate through the main sections (dashboard, profile, job listings, etc.). | The interface is intuitive, and the user can easily navigate to different sections without confusion. | Pass |
| TC-017 | Cross-browser compatibility | 1. Open IntelliConnect on different browsers (Chrome, Firefox, Safari).2. Test the functionality. | The application should work seamlessly across all tested browsers. | Pass |
| TC-018 | Integration with external services | 1. Log in as a user.2. Upload a resume for analysis.3. Check if the resume analyzer API works. | The resume analyzer API integrates and processes data without errors. | Pass |
| TC-019 | Data synchronization between client and server | 1. Log in as a user.2. Make updates to the profile.3. Check if changes are reflected across devices. | The updates are synchronized across devices when the user logs in. | Pass |
| TC-020 | Resume analyzer AI model accuracy | 1. Upload a resume with specific keywords (e.g., Python, Java).2. Review the feedback provided. | The resume analyzer correctly identifies skills and provides relevant feedback. | Pass |