1. System Requirements

1.1 Functional Requirements

The system should provide the following core functionalities:

1. **User Registration & Authentication**

   Students, club leaders, and admins can register and log in.
   Users must authenticate with email and password.

2. **Role-Based Access Control**

   Students: browse and register for events, join clubs.
   Club Leaders: create/manage clubs, post/manage events.
   Admins: manage all events, approve clubs, monitor system usage.

3. **Event Management**

   Club leaders can create, update, and cancel events.
   Students can browse events, view details, and register.
   Event participation lists are stored and viewable by organizers.

4. **Club Management**

   Club leaders can create clubs, approve membership requests.
   Students can apply to join clubs.
   Admins can monitor club activities.

5. **Notifications**

   Students receive notifications for upcoming events they registered for.
   Club leaders receive notifications for new members and registrations.

6. **Analytics & Reports**

   Admins can view event attendance statistics.
   Reports can be generated per club or per semester.

1.2 Non-Functional Requirements

- **Performance:** The system should handle at least 50 concurrent users.
- **Scalability:** Should allow adding more features such as payment for tickets in the future.
- **Security:** Passwords must be encrypted, role-based permissions enforced.
- **Usability:** Interface should be intuitive and mobile-friendly.
- **Maintainability:** Code should follow modular architecture (Spring Boot + REST API).

2. Use Case Design

2.1 Use Case List

1. Register as a new user
2. Log in to the system
3. Browse upcoming events
4. Register for an event (Student)
5. Create a club (Club Leader)
6. Apply to join a club (Student)
7. Create and publish an event (Club Leader)
8. Manage event registrations (Club Leader)
9. Approve or reject clubs (Admin)
10. Generate attendance reports (Admin)

2.2 Example Use Case: *Register for an Event*

- **Actor:** Student
- **Preconditions:** The student is logged in.
- **Main Flow:**
    1. Student browses the list of events.
    2. Student selects an event and clicks "Register."
    3. The system adds the student to the event's participant list.
    4. The student receives a confirmation notification.
- **Postconditions:** Student is registered for the event.
- **Alternative Flow:** If the event is full, the system displays a "Registration Closed" message.

3. Objects, Classes, and Relationships

3.1 Identified Objects

- **User** (general, with roles: Student, Club Leader, Admin)
- **Club**
- **Event**
- **Registration** (linking Student ↔ Event)
- **Membership** (linking Student ↔ Club)
- **Notification**
- **Report**

3.2 Classes and Relationships

- **User**
    - Attributes: id, name, email, password, role
    - Relationships:
        - can register for → Event
        - can join → Club
- **Club**
    - Attributes: id, name, description

    - Relationships:
        - created by → ClubLeader (User)
        - has → Members (Users)
        - organizes → Events
- **Event**
    - Attributes: id, title, date, location, capacity
    - Relationships:
        - belongs to → Club
        - has → Registrations (Users)
- **Registration**
    - Attributes: id, status (confirmed, canceled)
    - Links: Student ↔ Event
- **Membership**
    - Attributes: id, status (pending, approved)
    - Links: Student ↔ Club
- **Notification**
    - Attributes: id, message, timestamp
    - Sent to → User
- **Report**
    - Attributes: id, type, generatedDate
    - Generated by → Admin