FYP MANAGEMENT SYSTEM

Name: Abdullah

Reg No: SP23-BSE-116

Use case: Admin

Use Case 1: addMember()

Use Case Name: Add Member

Scope: Group Management

Level: User goal

Primary Actor: Administrator

Stakeholders and Interests:

Administrator: Wants to add students to a group efficiently.

Students: Want to be part of the correct group for their semester project.

Preconditions:

- The group must already exist.
- The student must be registered in the system.
- The student is not already part of another group.

Postconditions:

The student is successfully added to the group's member list.

Main Success Scenario (Basic Flow):

- Administrator selects an existing group.
- Administrator selects a student to add.
- System checks if the student is already part of any group.
- System adds the student to the selected group.
- System confirms the addition and displays updated group information.

Extensions (Alternate Flows):

3a. If the student is already part of another group:

→ System shows an error message and aborts the addition.

4a. If the group has reached its member limit:

→ System prevents the addition and notifies the administrator.

Special Requirements:

- Real-time validation of student eligibility.
- Notification sent to the student upon successful addition.

✅ **Use Case 2:** removeMember()

Use Case Name: Remove Member

Scope: Group Management

Level: User goal

Primary Actor: Administrator

Stakeholders and Interests:

Administrator: Needs control over group membership.

Students: Should be properly removed if they withdraw or switch groups.

Preconditions:

- Group and student must exist.
- Student must already be a member of the group.

Postconditions:

The student is removed from the group member list.

Main Success Scenario (Basic Flow):

- Administrator opens group details.
- Administrator selects the student to remove.
- System verifies the student's membership in the group.
- Student is removed from the list.
- System confirms removal.

Extensions (Alternate Flows):

3a. If the student is not found in the group:

→ System notifies the admin and aborts the removal.

4a. If the removed student is a group leader:

→ System prompts to assign a new leader or continue.

Special Requirements:

Notification sent to the student upon removal.

✅ **Use Case 3:** assignAdvisor()

Use Case Name: Assign Advisor

Scope: Group Management

Level: User goal

Primary Actor: Administrator

Stakeholders and Interests:

Administrator: Needs to assign supervisors fairly and efficiently.

Supervisors: Want to be assigned groups within their capacity.

Students: Need an advisor for guidance.

Preconditions:

- Group must exist.
- Advisor must be registered.
- Advisor must be available (not exceeding their group limit).

Postconditions:

Advisor is successfully linked to the group.

Main Success Scenario (Basic Flow):

- Administrator selects a group.
- Administrator chooses an advisor from a list.
- System checks advisor's availability.
- Advisor is assigned to the group.
- Confirmation message is shown.

Extensions (Alternate Flows):

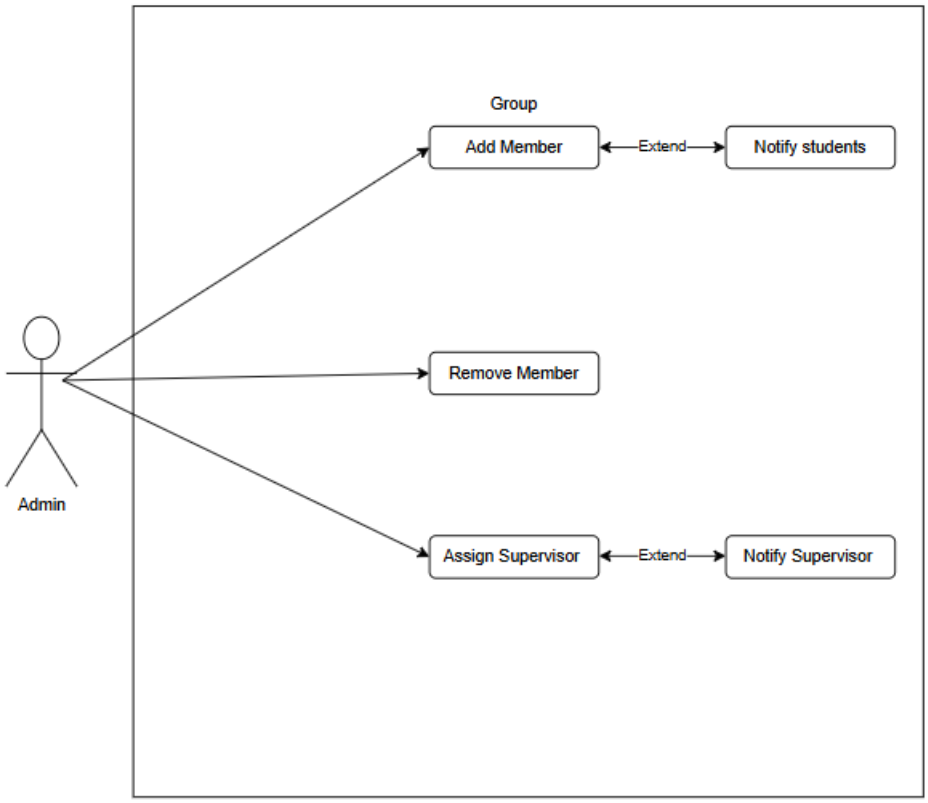
3a. If the advisor already has maximum assigned groups:

→ System blocks the assignment and shows a warning.

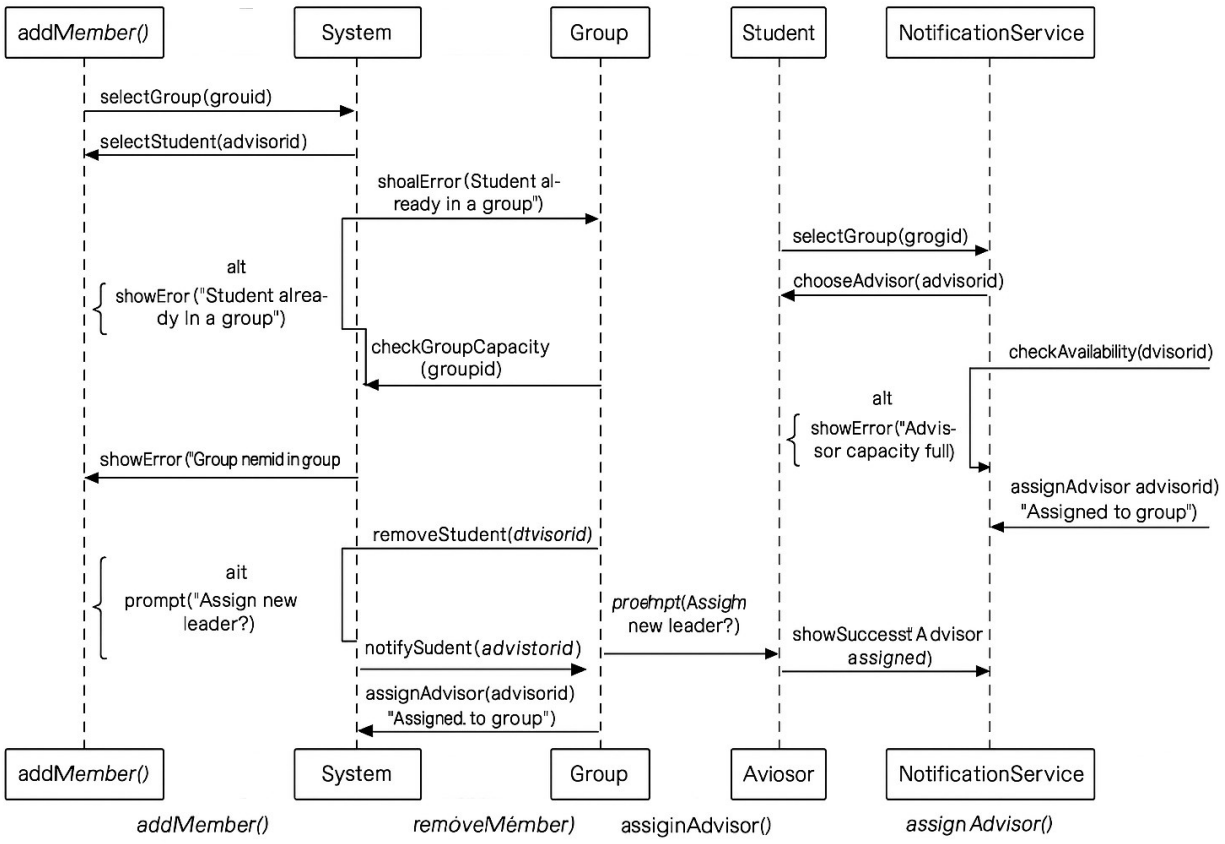
Special Requirements:

- System should prevent duplicate advisor assignments to the same group.
- Email notification to advisor and group members upon assignment.

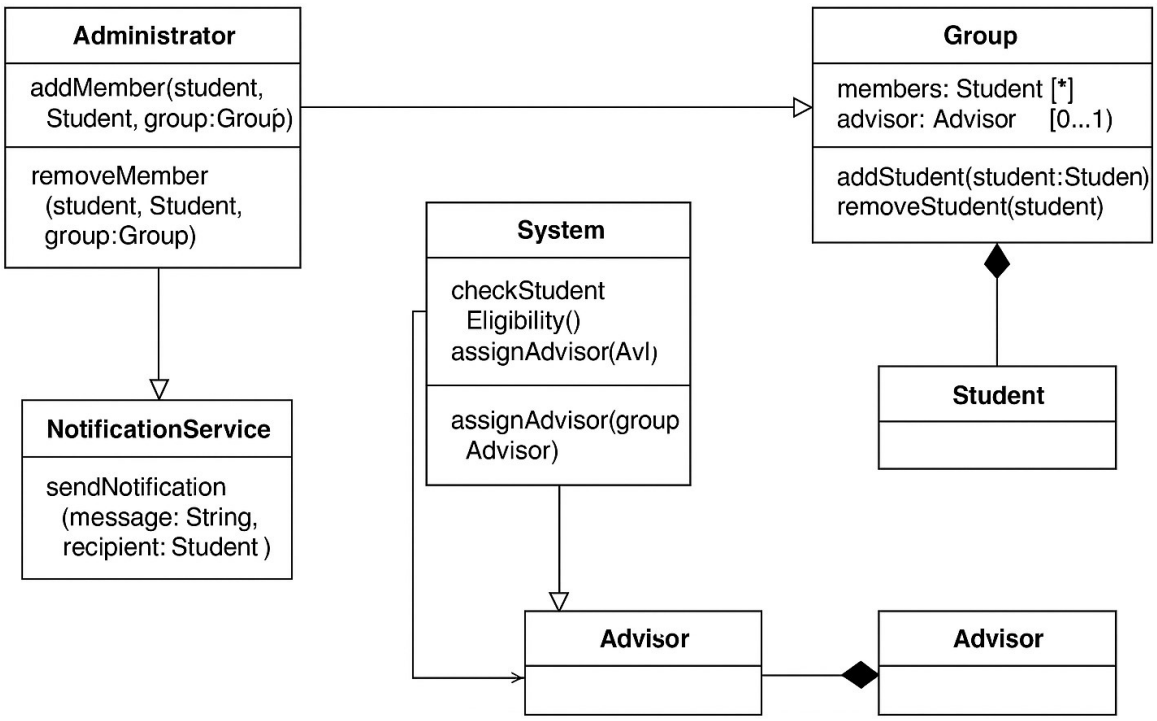
Use Case Diagram:



Sequence Diagram:



Class Diagram:



Coding Standards for FYP Group Management System

1. Class Design

- Each class must follow **Single Responsibility Principle**:
    - Student: Represents individual student info.
    - Advisor: Represents group advisor data.
    - Group: Manages members and advisor.
    - Administrator: Handles member management.
    - System: Controls core logic (eligibility, advisor assignment).
    - NotificationService: Sends notifications to students.

2. Naming Conventions

- Use **PascalCase** for class names: Group, Advisor, NotificationService.
- Use **camelCase** for variables and methods: groupList, addMember, checkStudentEligibility.
- Method names should start with a **verb**: assignAdvisor, sendNotification.

3. Access Modifiers

- All instance variables must be private.
- Provide **public getters and setters** where necessary to enforce encapsulation.
- Utility or helper methods that should not be accessed outside their class must be marked private.

4. Method Structure & Clarity

- Methods should be:
    - Short (ideally less than 20 lines).
    - Named clearly based on their task.
    - Responsible for a **single logical operation**.

5. Class Responsibility

Class	Responsibility
Student	Holds student information
Advisor	Represents an advisor entity
Group	Manages list of students and assigned advisor
Administrator	Adds/removes students from groups
System	Business logic (eligibility check, advisor assignment)
NotificationService	Sends messages to students when changes occur (like advisor assigned)

6. Error Handling

- Validate student input (e.g., check nulls, duplicates).
- When assigning an advisor, ensure no group has more than one advisor.
- Show user-friendly error messages.

7. Collections & Loops

- Use List<Student> and List<Group> initialized with ArrayList.

- Always loop using enhanced for or stream methods for readability:

java

CopyEdit

for (Student student : group.getMembers()) { ... }

8. Notifications

- Any major action (e.g., adding/removing member, assigning advisor) must trigger a call to NotificationService.sendNotification().

9. Code Comments

- Use **JavaDoc** for public methods and classes.
- Write inline comments only where logic is non-trivial.

10. Main Class / Entry Point

- Main class should:
    - Use a menu-driven interface.
    - Avoid logic — only call service methods.
    - Clearly separate UI and backend logic.