Project Report

Project Name: Semester Project Management

Name	**Registration number**	**Semester**	**Section**	**Assigned Functionalities**	
Abdullah (Group Leader)	Sp23-Bse-116	5	A	Group	
Rana Asad Ur Rahman	Sp23-Bse-029	5	A	Project	
Syed Mustafa	Sp23-Bse-015	5	A	Student	
Zarak Khan	Sp23-Bse-031	5	A	Admin	
Abdul Haseeb	Sp23-Bse-002	5	A	Document	
Aizaz Afridi	Sp23-Bse-003	5	A	Login	
Ahmed Ayyar Khan	Sp23-Bse-021	5	A	Supervisor	

Abstract

The Fina Year Project (FYP) Management System" project aims to streamline the process of managing final year projects in educational institutions. The project's primary objective is to develop a comprehensive software solution that facilitates the efficient coordination, monitoring, and evaluation of final year projects.

The need for this project arises from the challenges faced in traditional manual management systems, including difficulties in project allocation, group management, and assessment. By leveraging technology, our goal is to address these challenges and enhance the overall management experience for students, faculty members, and administrators involved in the FYP process.

The project follows a systematic approach, beginning with requirement analysis and culminating in the development of a user-friendly web-based platform. Key features of the system include project proposal submission, supervisor/advisor allocation, group management, report generation, and evaluation management.

Through the implementation of this system, we anticipate several benefits, including improved transparency, enhanced communication, streamlined workflows, and better decision-making. Additionally, the system's reporting capabilities will provide valuable insights into project progress and performance.

In conclusion, the FYP Management System represents a significant advancement in project management practices within educational institutions. By embracing technology and innovation, we aim to optimize the FYP process, ultimately leading to better outcomes for students and faculty members involved in final year projects.

Introduction to the Problem

1.1 Introduction

Managing final year projects is a critical yet complex task for educational institutions. Current practices often involve manual processes or outdated systems, leading to inefficiencies, miscommunication, and difficulty in tracking progress. The need for a modern, centralized system is paramount to ensure smooth coordination and management.

1.2 Purpose

The purpose of this project is to develop an FYP Management System that provides a centralized platform for students, supervisors, and administrators. The system aims to:

- Automate project submission and evaluation processes.
- Enhance collaboration between stakeholders.
- Improve monitoring and reporting of project progress.

1.3 Objectives

The objectives of the project are:

- To design a user-friendly platform for FYP management.
- To enable features such as student and advisor registration, project submission, and evaluation tracking.
- To improve transparency, decision-making, and communication.
- To offer comprehensive reporting and analytics for better insights.

1.4 Existing Solution

Current systems rely on manual methods like paper-based records or fragmented digital tools. These methods result in:

- Difficulty in project tracking.
- Inefficient communication between students and supervisors.
- Time-consuming evaluation and reporting processes.

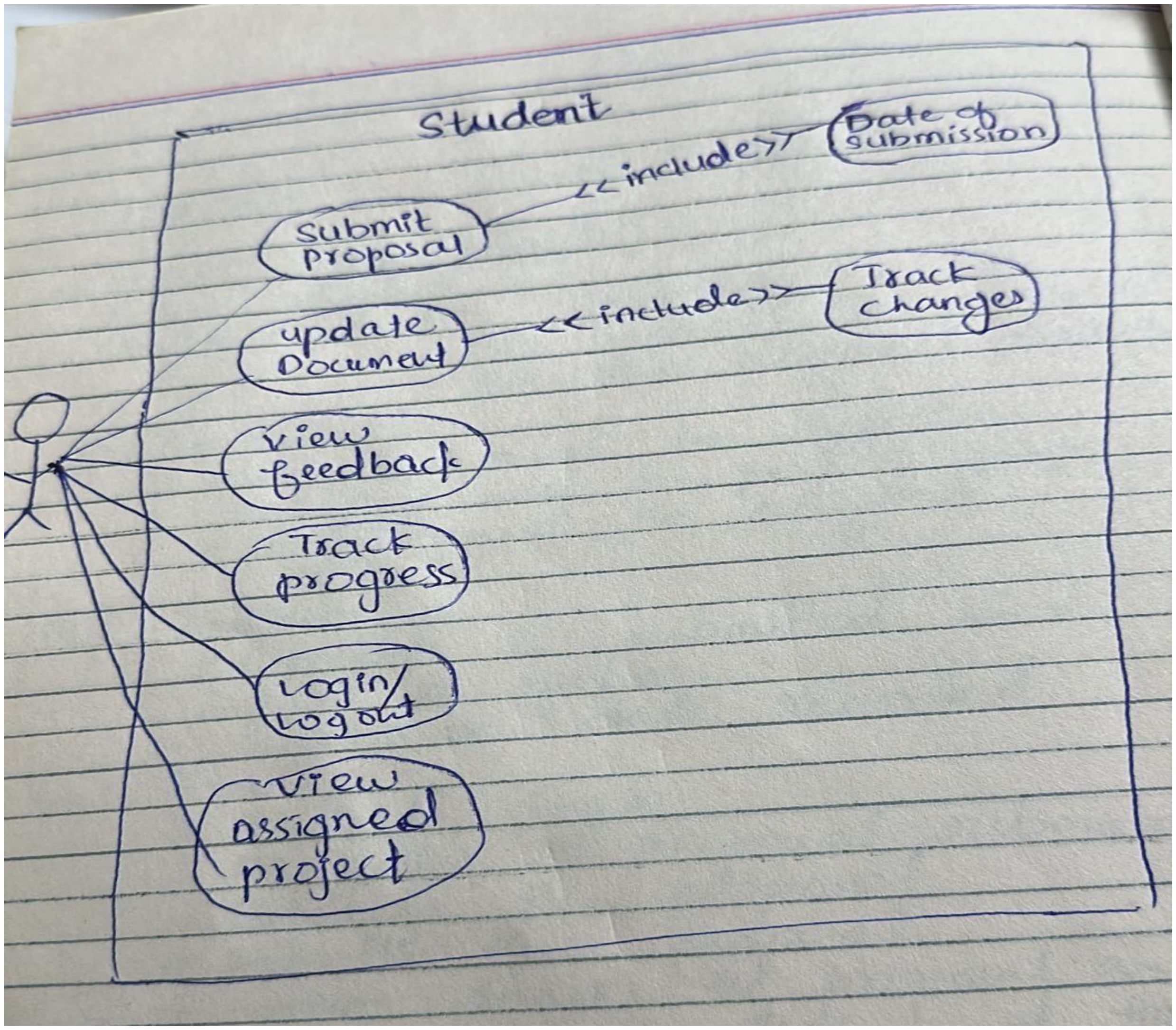
1.5 Proposed Solution

The proposed FYP Management System addresses these issues by providing a centralized, web- based solution. Features include:

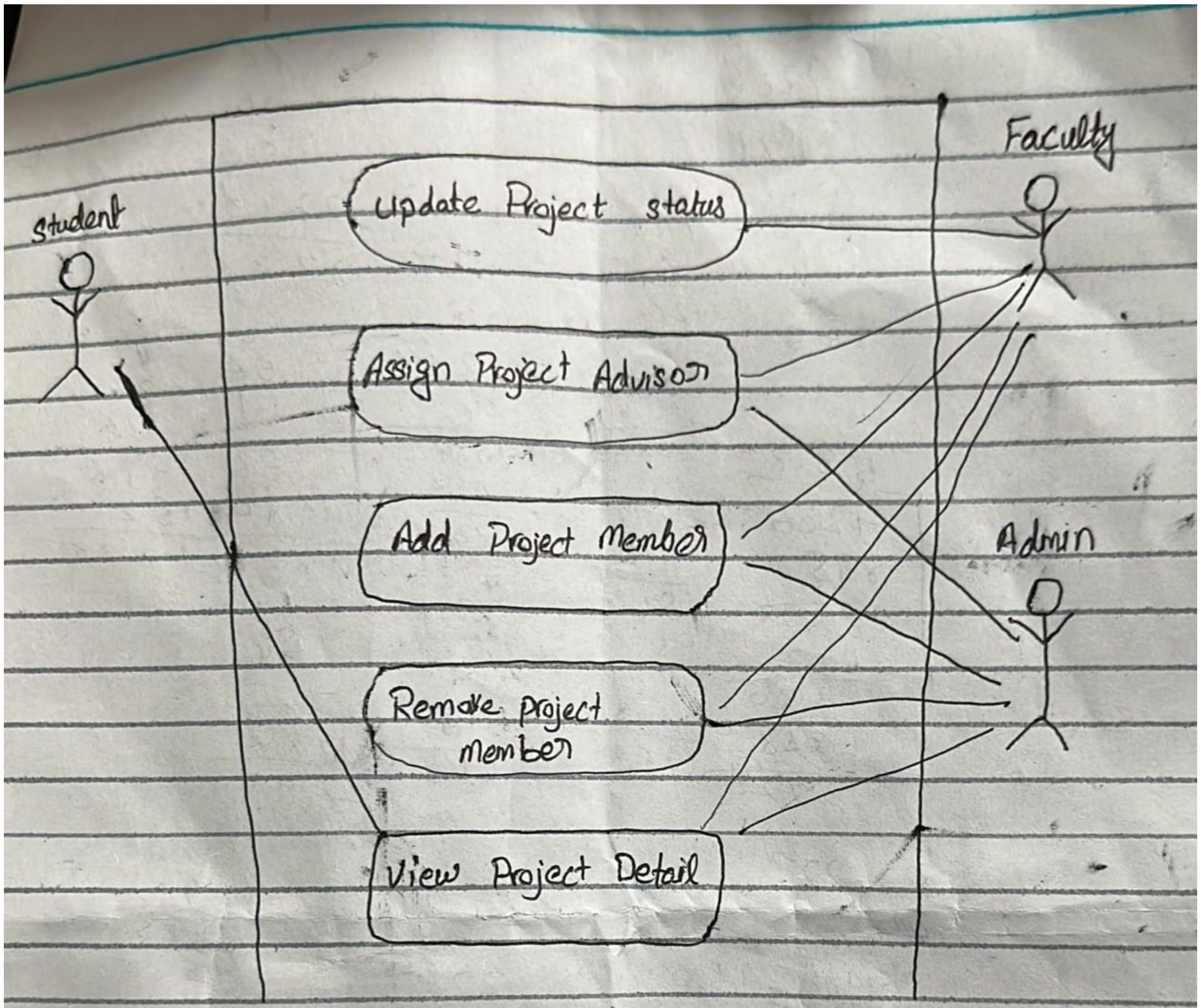
- Student and supervisor registration.
- Online project proposal submission.
- Group formation and supervisor assignment.
- Evaluation and marks management.
- Real-time progress tracking and reporting.

Chapter 1

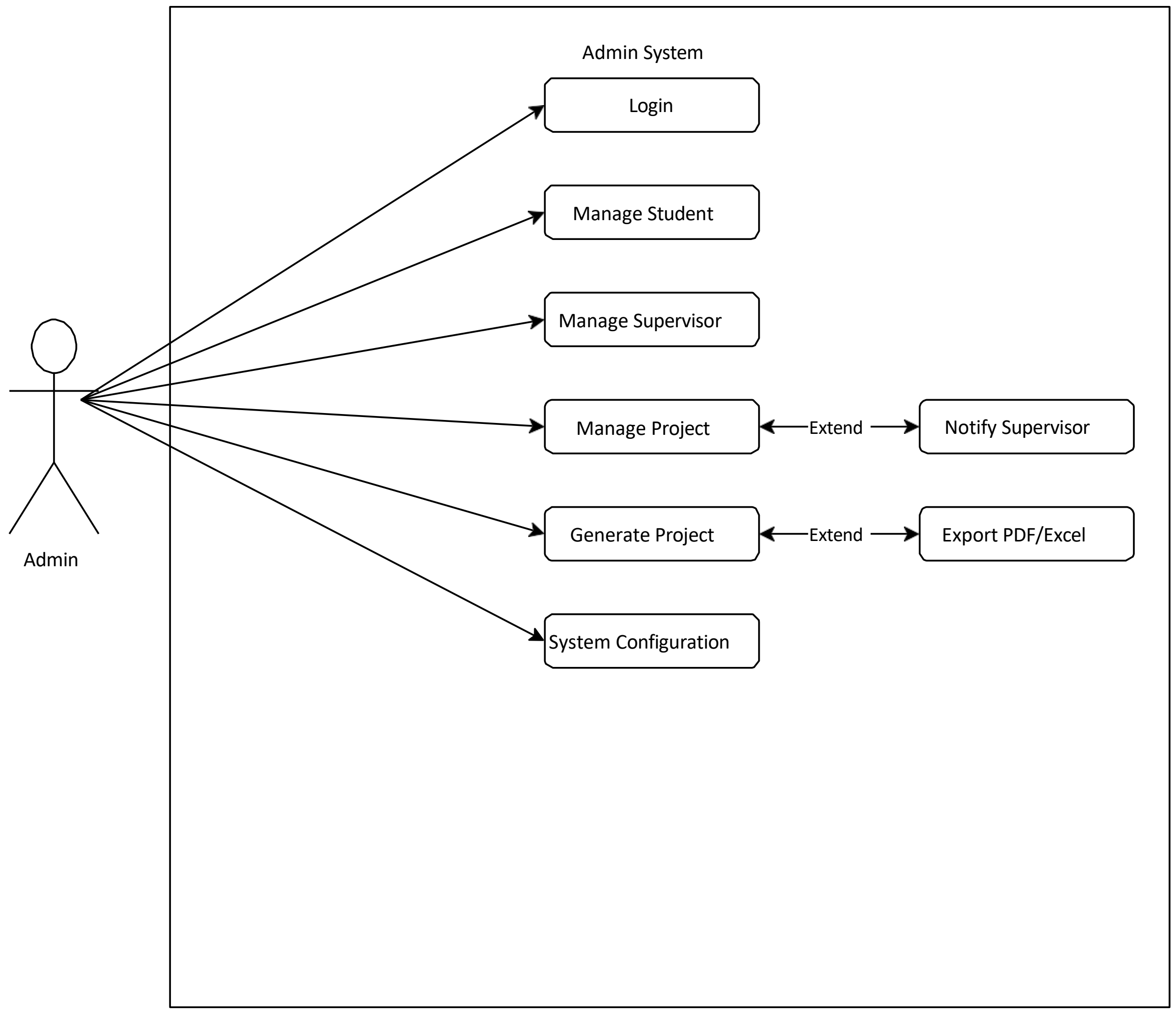
Use Case 1: Submit Proposal



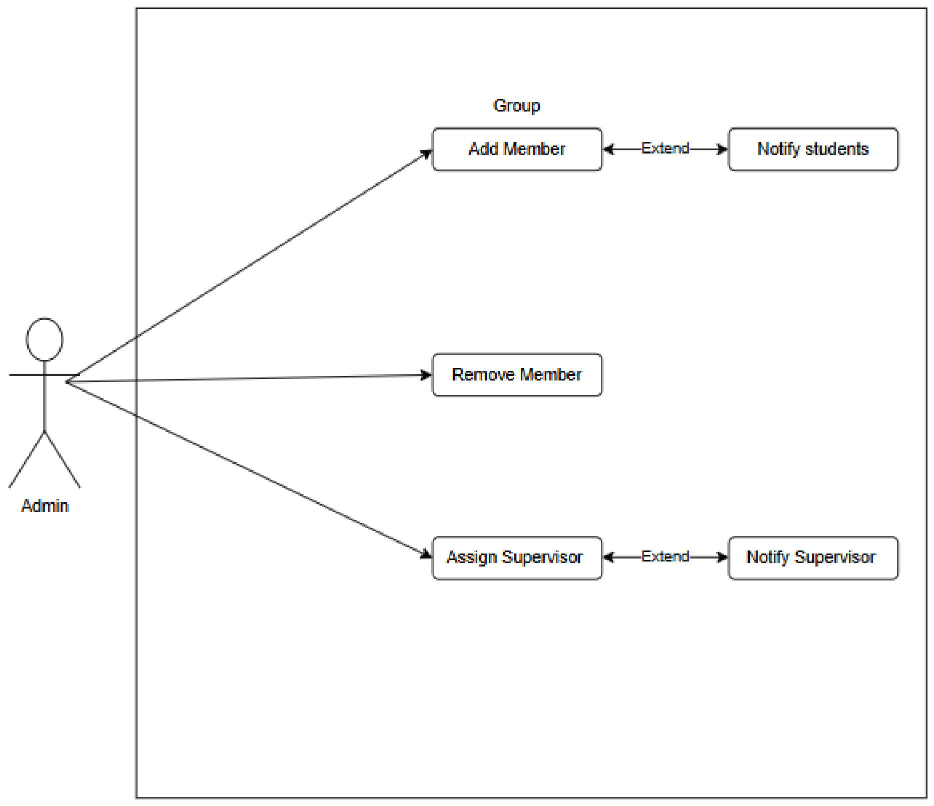
Rana Asad


Faculty
student
Update Project status
Assign Project Advisor
Add Project Member
Admin
Remove project member
View Project Detail

Zarak Khan


Admin System
Login
Manage Student
Manage Supervisor
Manage Project
Extend
Notify Supervisor
Generate Project
Extend
Export PDF/Excel
System Configuration
Admin

Abdullah


Group
Add Member
Extend
Notify students
Remove Member
Admin
Assign Supervisor
Extend
Notify Supervisor

Aizaz khan

The central component representing the system's purpose: managing student projects, supervision, and evaluations.

2. Registration

Function: Allows students and faculty to create accounts.

- Details:
  - Students register their groups.
  - Supervisors/Faculty register as advisors.

3. Login

- Function: Secure access for users (students, supervisors, admins).
- Details

Role-based permissions (e.g., students submit proposals, supervisors review).

4. Group Members

- Function: Manages student teams.
- Details:
  - Students form groups (typically 3–5 members).
  - Admin may assign/approve groups.

5. Supervisor

- Function: Guides and evaluates student projects.
- Details:
  - Assigns milestones.
  - Provides feedback/grades (linked to Feedback and Evolution).

6. Chat

- Function: Real-time communication between students and supervisors.
- Details:
  - Discuss project progress, clarify doubts.
  - May include file sharing.

7. Feedback

- Function: Supervisor’s evaluations on submissions.
- Details:

        - Covers proposals, reports, or presentations.
        - Tied to Evolution (grading).

8. Meeting Timings

- Function: Schedules discussions between students and supervisors.
- Details:
    - Calendar integra

Automated reminders.

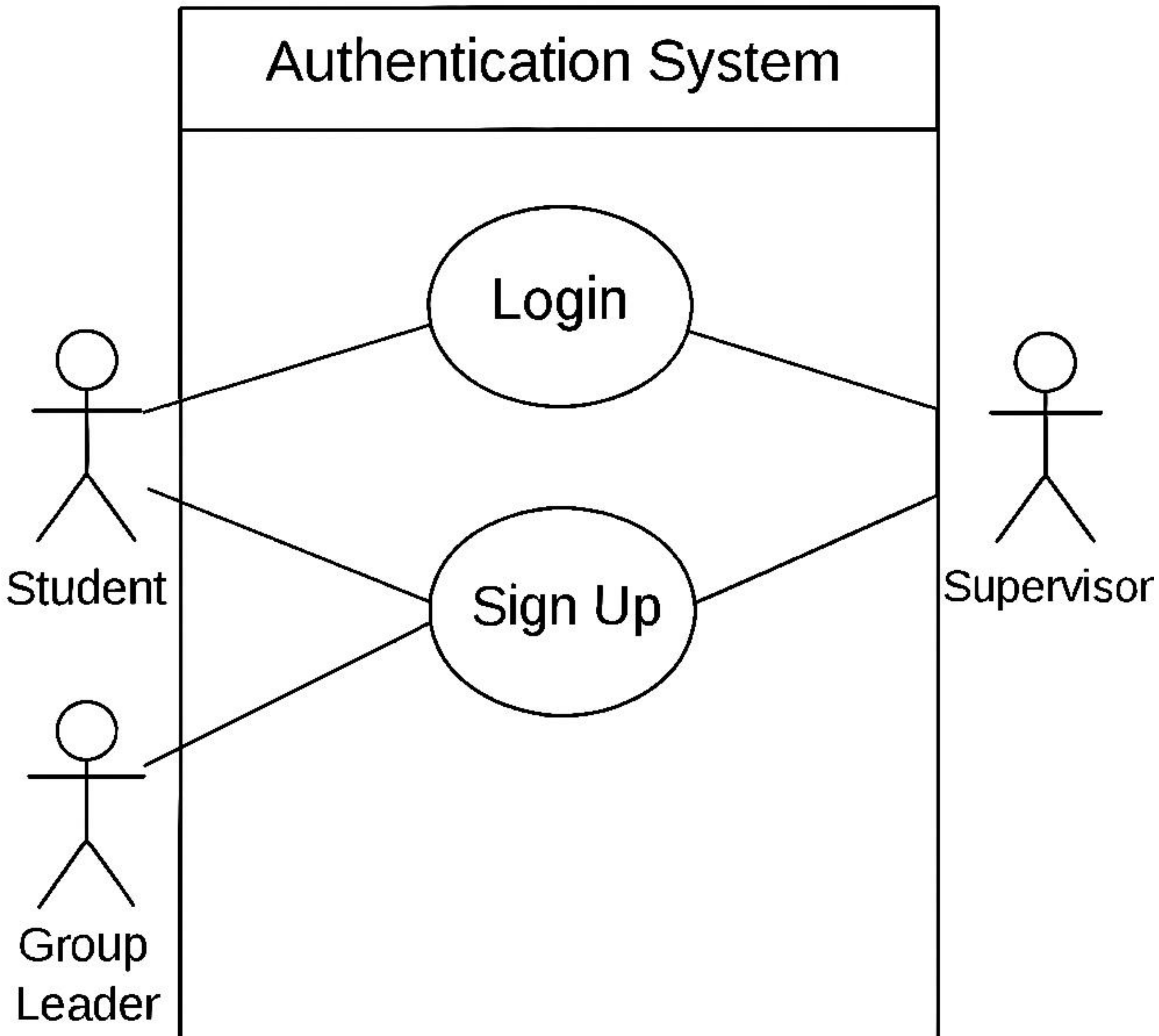
9. Evolution (Likely "Evaluation")

- Function: Grading and assessment of projects.
- Details:
    - Supervisors/faculty assign marks.
    - Generates final reports.

10. Students

- Primary actors who:
    - Form groups (Group Members).
    - Submit proposals.
    - Receive Feedback and grades (Evolution

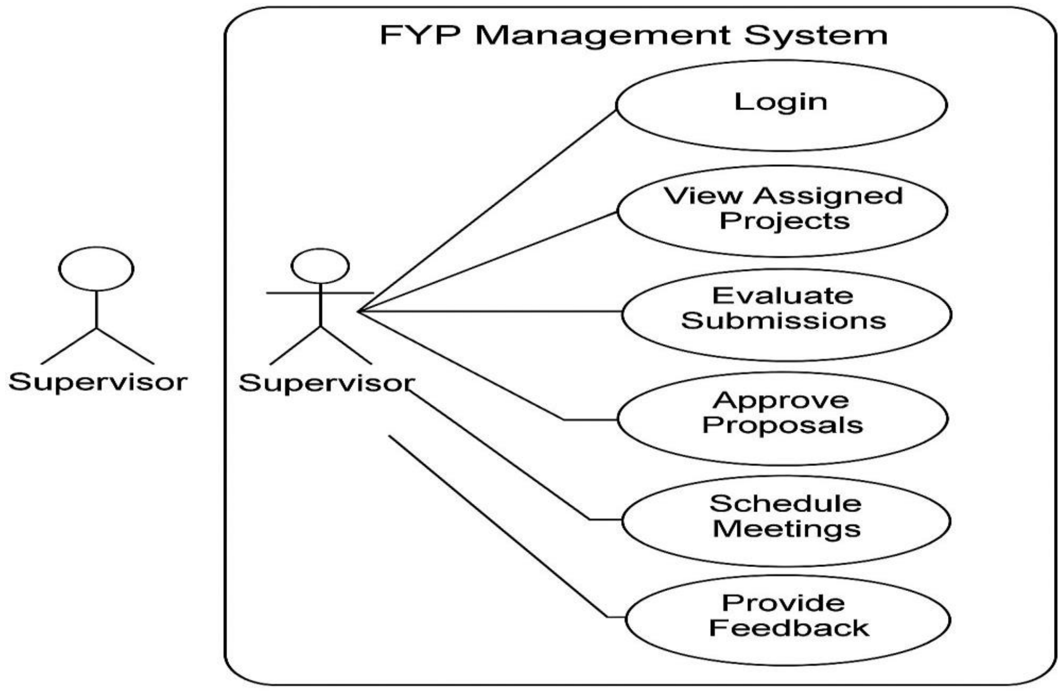
Aizaz ULLAH USE-CASE Diagram



Ahmad Ayyar Khan

Use case of supervisor:

- **Login** – Access the system securely.
- **View Assigned Projects** – See projects under their supervision.
- **Evaluate Submissions** – Grade or comment on reports, presentations, or milestones.
- **Approve Proposals** – Review and approve/reject project proposals.
- **Schedule Meetings** – Set meetings with students.
- **Communicate with Students** – Send messages or feedback.
- **Submit Evaluation Reports** – Submit grades or evaluation results to the system/admin.


FYP Management System
Supervisor
Supervisor
Login
View Assigned Projects
Evaluate Submissions
Approve Proposals
Schedule Meetings
Provide Feedback

Chapter 2

Use Case Name: Submit Proposal

Actor: Student

Goal in Context: The student wants to submit a proposal document for the assigned project.

Preconditions:

The student must be logged into the system.

The student must be part of a group.

The project must be assigned.

Postconditions:

- Proposal is submitted and saved in the system.
- Submission date is recorded.

Main Success Scenario (Basic Flow):

- Student logs into the system.
- Navigates to the proposal submission section.
- Uploads the proposal document.
- Enters any required metadata (e.g., title, description).
- Submits the proposal.
- System stores the proposal and confirms submission.
- System logs the date and time of submission.

Extensions (Alternative Flows):

1. If the file format is invalid, system shows an error message.
2. If required fields are missing, system prevents submission and displays a prompt.

Includes

Date of submission

Special Requirements

The uploaded document must be in PDF or DOCX format.

Maximum file size allowed: 10MB.

Name: Rana Asad Ur Rahman

Registration Number: SP23-BSE-029

Fully Dressed Use Case: Update Project Status

Use Case Name	Update Project Status
Scope	Project Management System
Level	User goal
Primary Actor	Faculty
Stakeholders and Interests	- **Faculty**: Wants to reflect the accurate status of the project based on its progress. - **Students**: Want to be informed of their project's progress. - **Admin**: Needs accurate records for tracking and reporting.
Preconditions	- Project exists in the system. - Faculty is assigned as the advisor to the project. - User is authenticated.
Post conditions	- Project status is updated successfully in the system.
Success Metrics	The status of the project is updated to the selected new status and stored correctly.
Minimal Guarantee	The system logs the update attempt even if it fails and shows an appropriate error message.

📋 Main Success Scenario (Basic Flow)

1. Faculty logs into the system.
2. Faculty navigates to the project list and selects a specific project.
3. Faculty clicks on the “Update Status” button.
4. The system displays current project status and a list of valid status options (e.g., "Not Started", "In Progress", "Completed").
5. Faculty selects a new status and clicks “Submit”.
6. The system validates the change.
7. The system updates the project status in the database.

8. The system confirms the update with a success message.
9. Project status is now reflected in the project details.

🔁 Extensions (Alternate Flows)

3a. Project not found

- 3a1. System shows an error: "Project not found."
- 3a2. Use case ends.

5a. Invalid status selected

- 5a1. System shows an error: "Invalid status."
- 5a2. Faculty selects a valid status.
- 5a3. Returns to step 6.

6a. Database update fails

- 6a1. System displays error: "Failed to update project status."
- 6a2. Faculty retries or contacts admin.
- 6a3. Use case ends.

🔐 Special Requirements

- The system must validate user permissions before allowing the update.
- Status values must be predefined and consistent (possibly from an enum).
- Audit trail should log the date/time of the update and the actor's ID.

Zarak Khan

Comprehensive Use Case Specification for Admin in FYP Management System

1. System Overview

The **Final Year Project (FYP) Management System** allows university administrators to manage student projects, supervisors, evaluations, and system configurations. The **Admin** is the primary actor responsible for maintaining data, generating reports, and ensuring smooth system operations.

---

2. Actors

Actor	Description
Admin	Manages students, supervisors, projects, and system settings.
Student	(Secondary) Submits project proposals and reports.
Supervisor	(Secondary) Guides students and evaluates projects.

---

3. Detailed Use Cases

Use Case 1: Login to System

- **Actor:** Admin
- **Description:** Admin logs into the FYP Management System.
- **Preconditions:** Admin has valid credentials.
- **Basic Flow:**

1. Admin enters email and password.
2. System verifies credentials.
3. On success, redirects to Admin Dashboard.

- **Alternative Flow:**
  - If credentials are invalid, system displays an error.

Post conditions: Admin gains access to the system.

Use Case 2: Manage Students

- **Actor:** Admin
- **Description:** Admin adds, edits, or removes student records.
- **Preconditions:** Admin is logged in.
- **Basic Flow:**

1. Admin clicks **"Manage Students"**.
2. System displays a list of students.
3. Admin selects:

- **Add Student**: Enters (Name, ID, Email, and Program).
- **Edit Student**: Modifies existing details.

- **Delete Student**: Removes record after confirmation.

4. System validates and updates the database.

- **Alternative Flows:**
  - If student ID already exists, system rejects duplication.

Post conditions: Student records are updated.

Use Case 3: Manage Supervisors

- **Actor:** Admin
- **Description:** Admin adds or assigns faculty supervisors.
- **Preconditions:** Admin is logged in.
- **Basic Flow:**

1. Admin clicks **"Manage Supervisors"**.
2. System shows the list of supervisors.
3. Admin:

- **Adds Supervisor** (Name, Department, Max Projects).
- **Assigns Supervisor** to a project.

4. System checks availability and updates records.

- **Alternative Flows:**
  - If supervisor’s project limit is exceeded, system shows an error.

Post conditions: Supervisor assignments are updated.

Use Case 4: Manage Projects

- **Actor:** Admin
- **Description:** Admin creates, assigns, and tracks FYP projects.
- **Preconditions:** Students and supervisors are registered.
- **Basic Flow:**

1. Admin clicks **"Manage Projects"**.
2. Chooses:

- **Create Project** (Title, Description, and Deadline).

▪ **Assign Student & Supervisor** (from dropdown lists).

3. System validates and links them in the database.

- **Alternative Flows:**

○ If a student is already assigned, system prevents duplication.

Post conditions: Project is added and visible in tracking.

Use Case 5: Generate Reports

- **Actor:** Admin
- **Description:** Admin exports project progress or evaluation reports.
- **Preconditions:** Projects exist with submission data.
- **Basic Flow:**

1. Admin clicks **"Generate Reports"**.
2. Selects report type:

▪ **Progress Report** (Student milestones).

▪ **Evaluation Report** (Supervisor feedback).

3. Filters by Department/Date Range.
4. System compiles data into **PDF/Excel**.
5. Admin downloads or prints.

- **Alternative Flows:**

○ If no data exists, system displays "No records found."

Post conditions: Report is generated for review.

Use Case 6: System Configuration

- **Actor:** Admin
- **Description:** Admin sets deadlines, grading rules, and permissions.
- **Basic Flow:**

1. Admin clicks **"System Configuration"**.
2. Updates:

▪ **Submission Deadlines** (e.g., Proposal: 30-May-2024).

- **Grading Criteria** (e.g., 70% for final report).

3. System saves changes and notifies affected users.

- **Alternative Flows:**
  - If a deadline is in the past, system rejects it.

  Postconditions: System settings are updated.

Name: Abdullah

Reg No: SP23-BSE-116

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.

fully Dressed Use Case: Submit Proposal

1. Basic Information*

Element   *Description* |

Use Case Name | Submit Proposal |

Actor | Student |

2. Preconditions

- Student is logged in.
- Student has a registered group.
- Submission period is open.

3. Main Success Scenario

1. Student selects "Submit Proposal".
2. System displays a submission form (title, description, file upload).
3. Student fills details and uploads a proposal file (PDF/DOCX).
4. System validates the file (format, size ≤10MB).
5. Supervisor is notified; proposal status changes to "Under Review".

4. Alternative Flows

- *A1: Invalid File
  - Ste 4a: System rejects non-PDF/DOCX files.

Aizaz Ullah ->Fully Dressed

Use Case Element	**Details**
Use Case Name	Login
Scope	Authentication System
Level	User Goal
Primary Actors	Student, Group Leader, Supervisor
Stakeholders and Interests	- Student/Group Leader/Supervisor: Want to access the system securely.
	- System: Ensures only valid users are authenticated.
Preconditions	User must already be registered in users.txt.
Postconditions (Success)	User is authenticated and redirected to their dashboard.
Postconditions (Failure)	User remains on login screen with an error message.
Main Success Scenario	1. User opens the login interface.
	2. User enters role, ID, and password.
	3. System reads users.txt.
	4. If a match is found, user is authenticated.
	5. System redirects user to respective dashboard.
Extensions (Alternate Flows)	3a. File not found – system logs error and displays error message.
	4a. Invalid credentials – user is shown an error and stays on login screen.
Special Requirements	- Password matching must be case-sensitive.
	- File I/O must be reliable (users.txt must exist and be accessible).

Fully Dressed use case

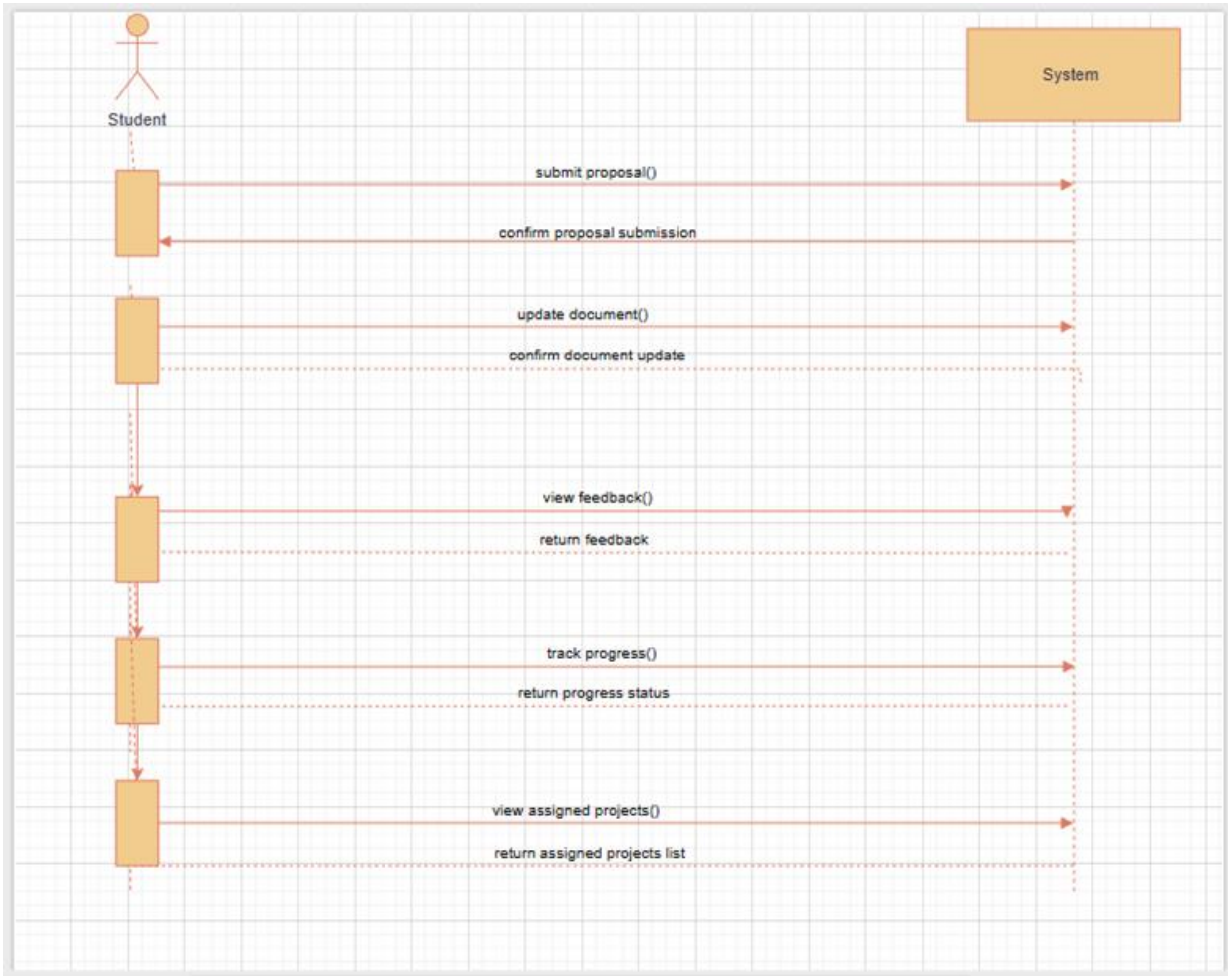
Ahamd Ayyar Khan

Use Case Element	Details
Use Case ID	UC-SUP-03
Use Case Name	Evaluate Submissions
Primary Actor	Supervisor
Stakeholders and Interests	Supervisor (wants to grade fairly and efficiently), Students (want timely feedback)
Preconditions	- Supervisor must be logged in. - The project must be assigned to the supervisor.
Postconditions	- Submission is marked as evaluated. - Grades and feedback are saved in the system.
Main Success Scenario	1. Supervisor logs into the system. 2. Navigates to assigned projects. 3. Selects a submission. 4. Views content of the submission. 5. Enters feedback and grade. 6. Clicks "Submit Evaluation". 7. System stores evaluation and notifies student.
Extensions (Alternative Flows)	3a. No submissions available: System shows message "No pending submissions". 5a. Supervisor cancels evaluation: System returns to project list without saving.
Special Requirements	- Evaluation should support file previews (PDF, DOCX). - Should allow file attachments in feedback.
Frequency of Use	Weekly or bi-weekly during submission periods.
Business Rules	- Grades must be within a valid range (e.g., 0–100). - Feedback is mandatory for final evaluation.

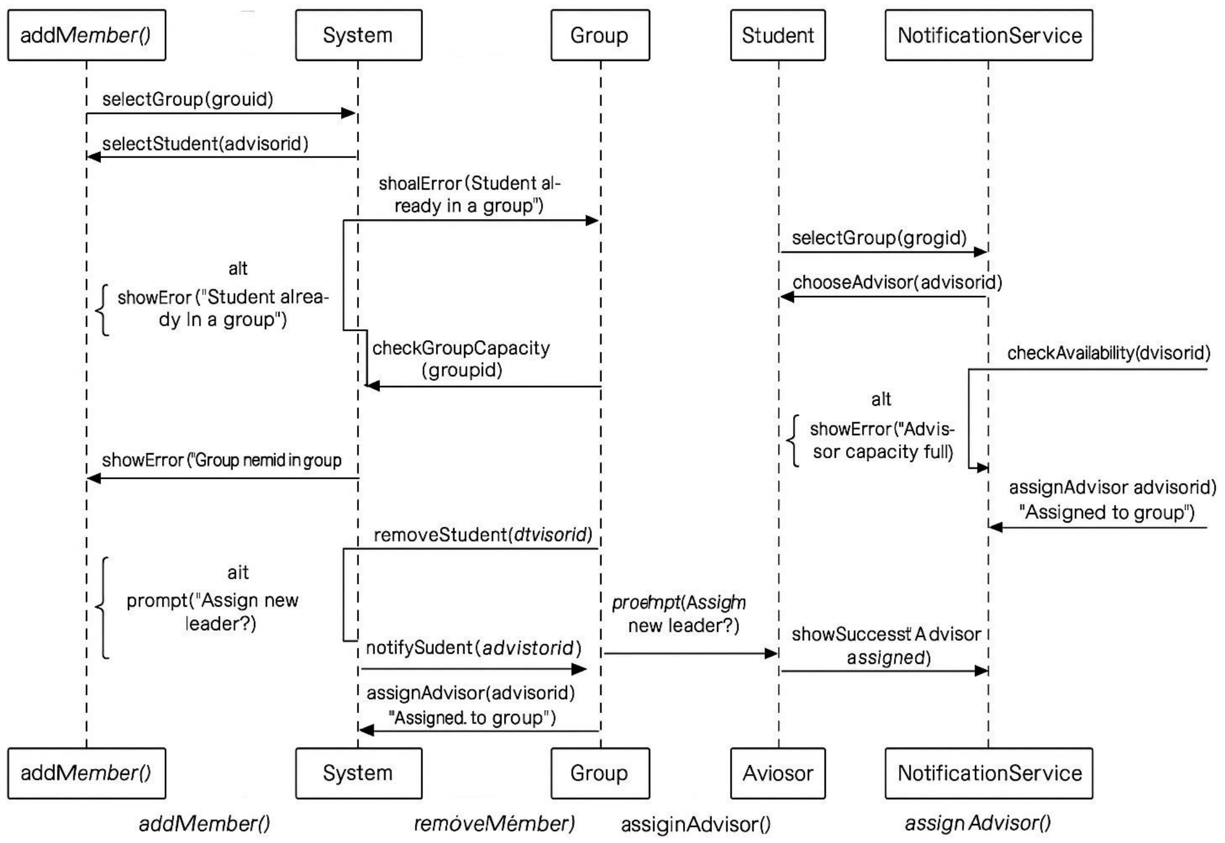
Chapter 3

Sequence Diagrams

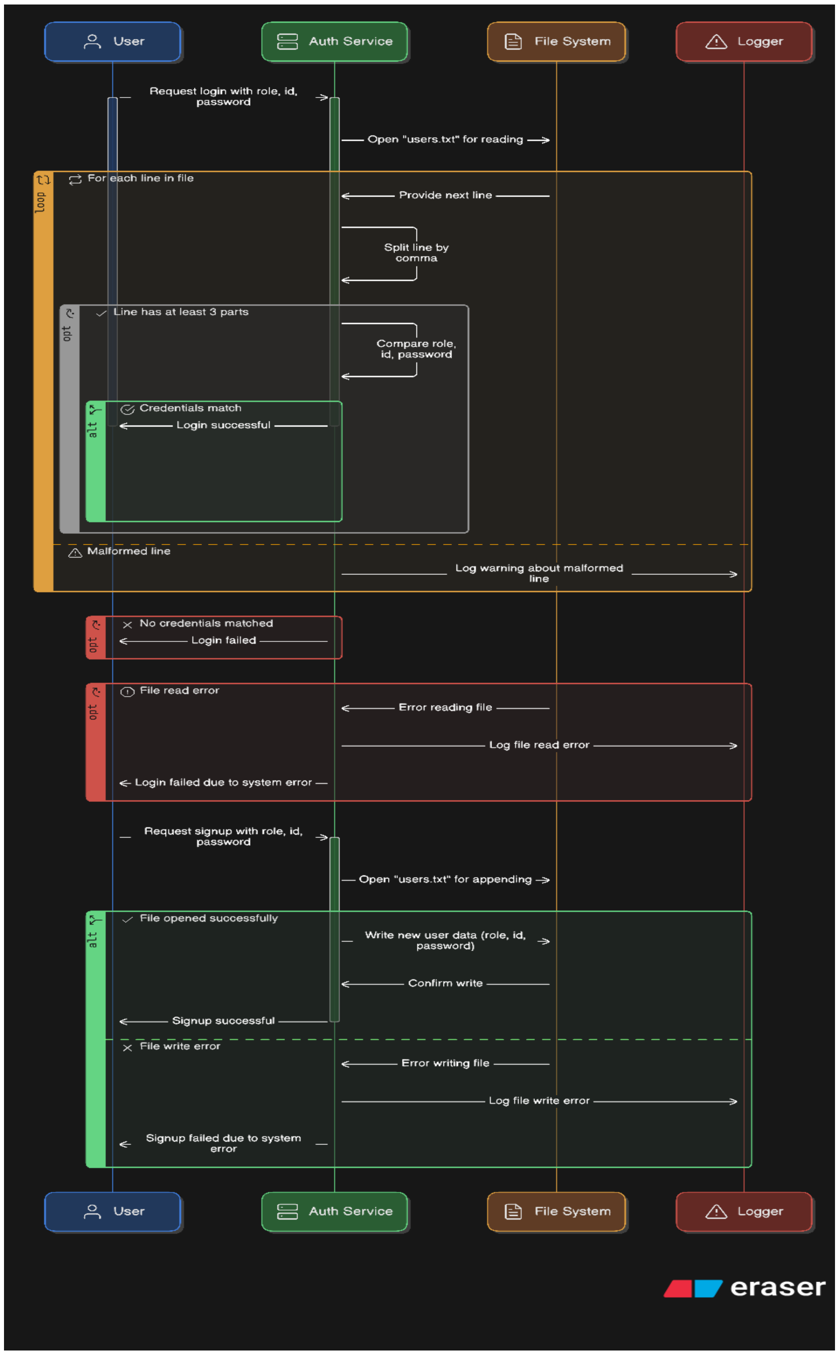
Mustafa



Abdullah

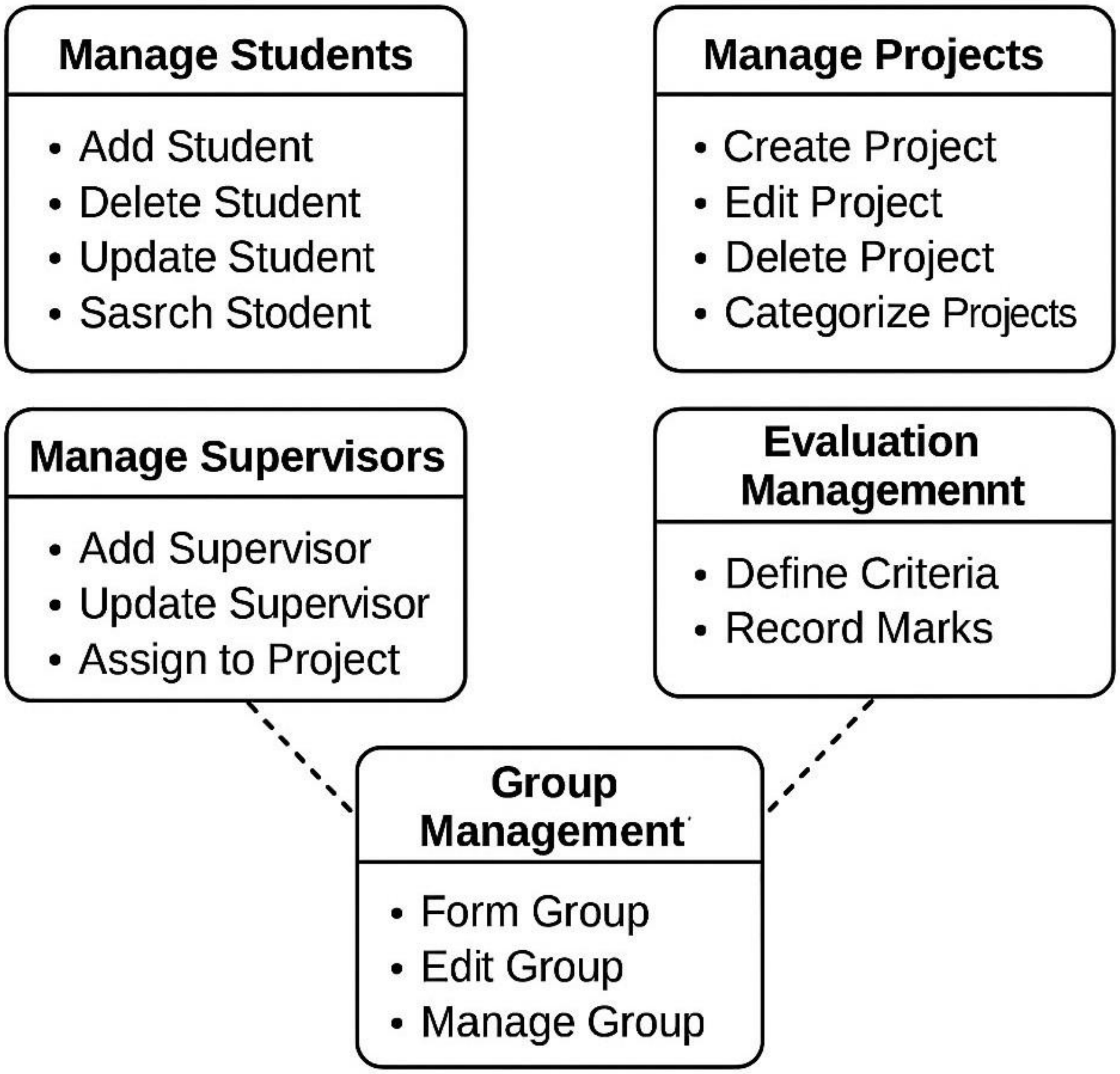

addMember()
System
Group
Student
NotificationService
selectGroup(grouid)
selectStudent(advisorid)
shoalError(Student al-ready in a group")
selectGroup(grogid)
alt
showEror ("Student alrea-dy In a group")
chooseAdvisor(advisorid)
checkGroupCapacity (groupid)
checkAvailability(dvisorid)
alt
showError("Advis-sor capacity full)
showError ("Group nemid in group
assignAdvisor advisorid) "Assigned to group")
removeStudent(dtvisorid)
ait
prompt("Assign new leader?)
proehmpt(Assigm new leader?)
showSuccess' Advisor assigned)
notifySudent(advistorid)
assignAdvisor(advisorid) "Assigned. to group")
addMember()
System
Group
Aviosor
NotificationService
addMember()
removeMember)
assiginAdvisor()
assign Advisor()

Aizaz Ullah



User
Auth Service
File System
Logger
Request login with role, id, password
Open "users.txt" for reading
loop
For each line in file
Provide next line
Split line by comma
opt
Line has at least 3 parts
Compare role, id, password
alt
Credentials match
Login successful
Malformed line
Log warning about malformed line
opt
No credentials matched
Login failed
opt
File read error
Error reading file
Log file read error
Login failed due to system error
Request signup with role, id, password
Open "users.txt" for appending
alt
File opened successfully
Write new user data (role, id, password)
Confirm write
Signup successful
File write error
Error writing file
Log file write error
Signup failed due to system error
User
Auth Service
File System
Logger
eraser

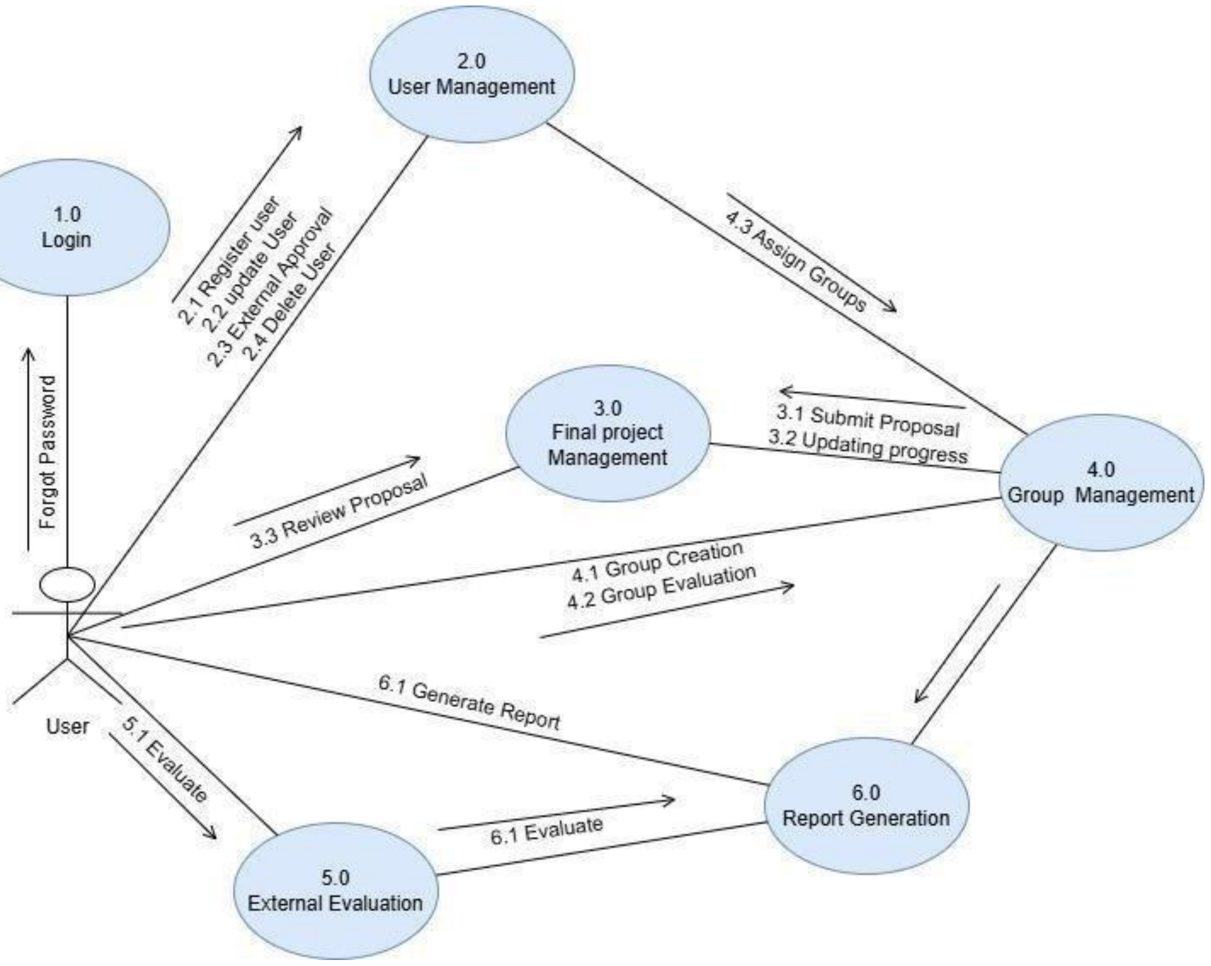
Chapter 4



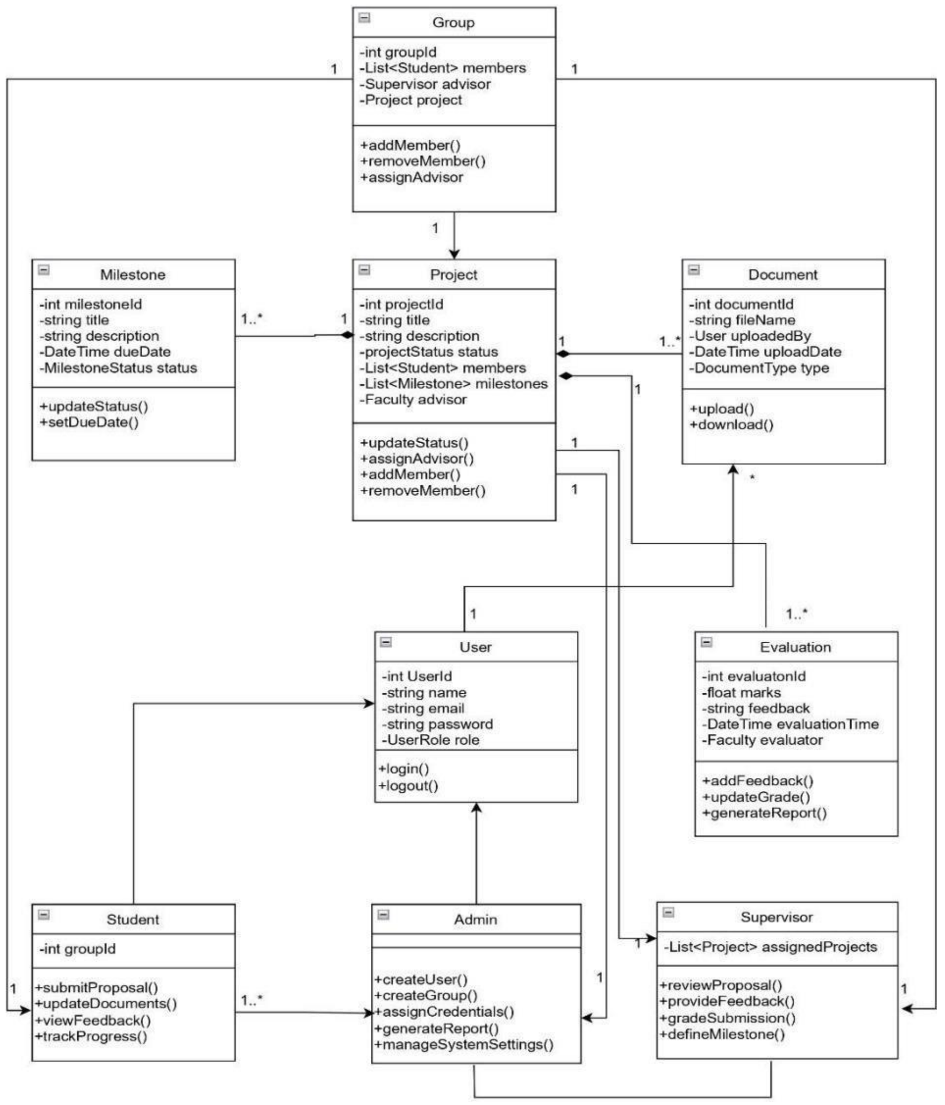
Package Diagram anrurm

Chapter 5

Collaboration Diagram

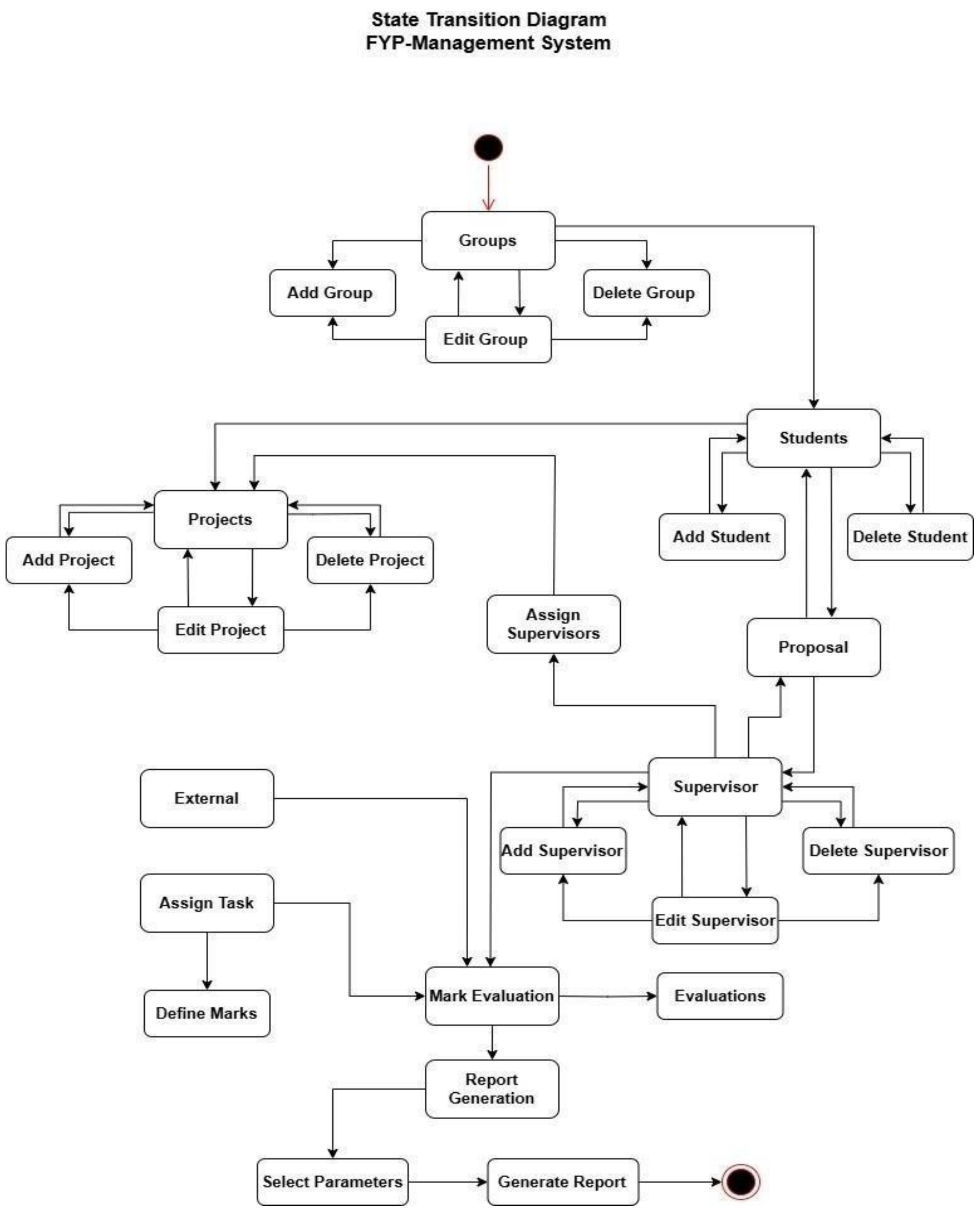


Chapter 6



Group
-int groupId
-List<Student> members
-Supervisor advisor
-Project project
+addMember()
+removeMember()
+assignAdvisor()
Milestone
-int milestoneId
-string title
-string description
-DateTime dueDate
-MilestoneStatus status
+updateStatus()
+setDueDate()
Project
-int projectId
-string title
-string description
-projectStatus status
-List<Student> members
-List<Milestone> milestones
-Faculty advisor
+updateStatus()
+assignAdvisor()
+addMember()
+removeMember()
Document
-int documentId
-string fileName
-User uploadedBy
-DateTime uploadDate
-DocumentType type
+upload()
+download()
User
-int UserId
-string name
-string email
-string password
-UserRole role
+login()
+logout()
Evaluation
-int evaluatonId
-float marks
-string feedback
-DateTime evaluationTime
-Faculty evaluator
+addFeedback()
+updateGrade()
+generateReport()
Student
-int groupId
+submitProposal()
+updateDocuments()
+viewFeedback()
+trackProgress()
Admin
+createUser()
+createGroup()
+assignCredentials()
+generateReport()
+manageSystemSettings()
Supervisor
-List<Project> assignedProjects
+reviewProposal()
+provideFeedback()
+gradeSubmission()
+defineMilestone()
1
1
1
1..*
1
1
1..*
1
1
1
*
1
1..*
1
1
1..*
1
1
1

State Transition Diagram



Chapter 7

Coding Standards for FYP Group Management System

1. Class Naming Convention

- All class names must use **PascalCase** format.
- Classes must be named based on their responsibilities:
    - Student, Advisor, Group, Administrator, System, NotificationService.

2. Method Naming Convention

- All method names must use **camelCase**.
- Method names must clearly reflect the operation they perform:
    - addMember, removeMember, addStudent, removeStudent, checkStudentEligibility, assignAdvisor, sendNotification.

3. Access Modifiers and Encapsulation

- All class fields must be declared as private.
- Access to fields must be provided through public getters and setters.
- Only relevant classes may access internal methods marked as private.

4. Constructor Standards

- Each class must define a constructor to initialize required fields.
- No business logic is allowed inside constructors.

5. Method Structure

- Each method must perform a single, clearly defined task.
- Each method must include:
    - Input validation.
    - Null checks before object operations.
    - Clear return values or appropriate exception handling.

6. Collection Handling

- Collections such as List<Student> must be initialized using concrete classes like ArrayList.
- Iteration through collections must use enhanced for loop or iterator.

7. Code Formatting Rules

- Indentation must be 4 spaces per level.
- Curly braces {} must be placed on the same line as the control structure or method name.
- A single blank line must be used between methods.

8. Notification Service Usage

- NotificationService.sendNotification(String message, Student recipient) must be called:
    - When a student is added to a group.
    - When a student is removed from a group.
    - When an advisor is assigned to a group.

9. Class Responsibilities

- Administrator class is only responsible for managing group membership (adding/removing students).
- Group class maintains a list of Student members and an optional Advisor.
- System class is responsible for:
    - Checking student eligibility.
    - Assigning advisors.
- NotificationService only handles notification logic.

10. Exception Handling

- All invalid operations must throw standard Java exceptions such as IllegalArgumentException.
- No silent failures are allowed.

11. Naming Standards

- Variables must be named clearly and meaningfully.
    - Examples: student, group, advisor, message, recipient.
- No abbreviation of names unless they are universally accepted (e.g., ID).

12. Comments and Documentation

- All public classes and methods must include JavaDoc comments.
- Inline comments must be used sparingly and only where logic is not self-explanatory.