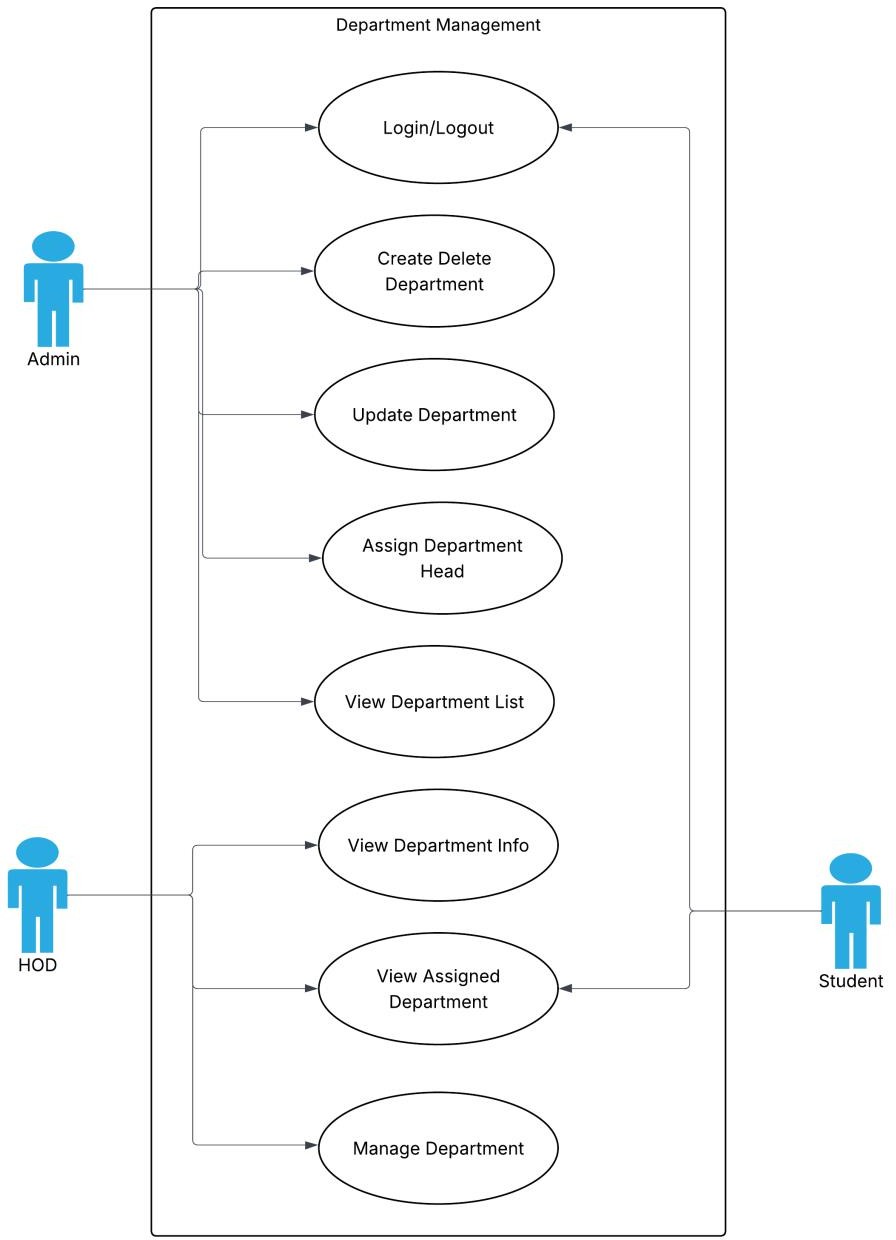
**Name:** Muhammad Usman Waqar Khan

**Registration No:** SP23-BSE-010

Chapter 1

Use Case Diagrams

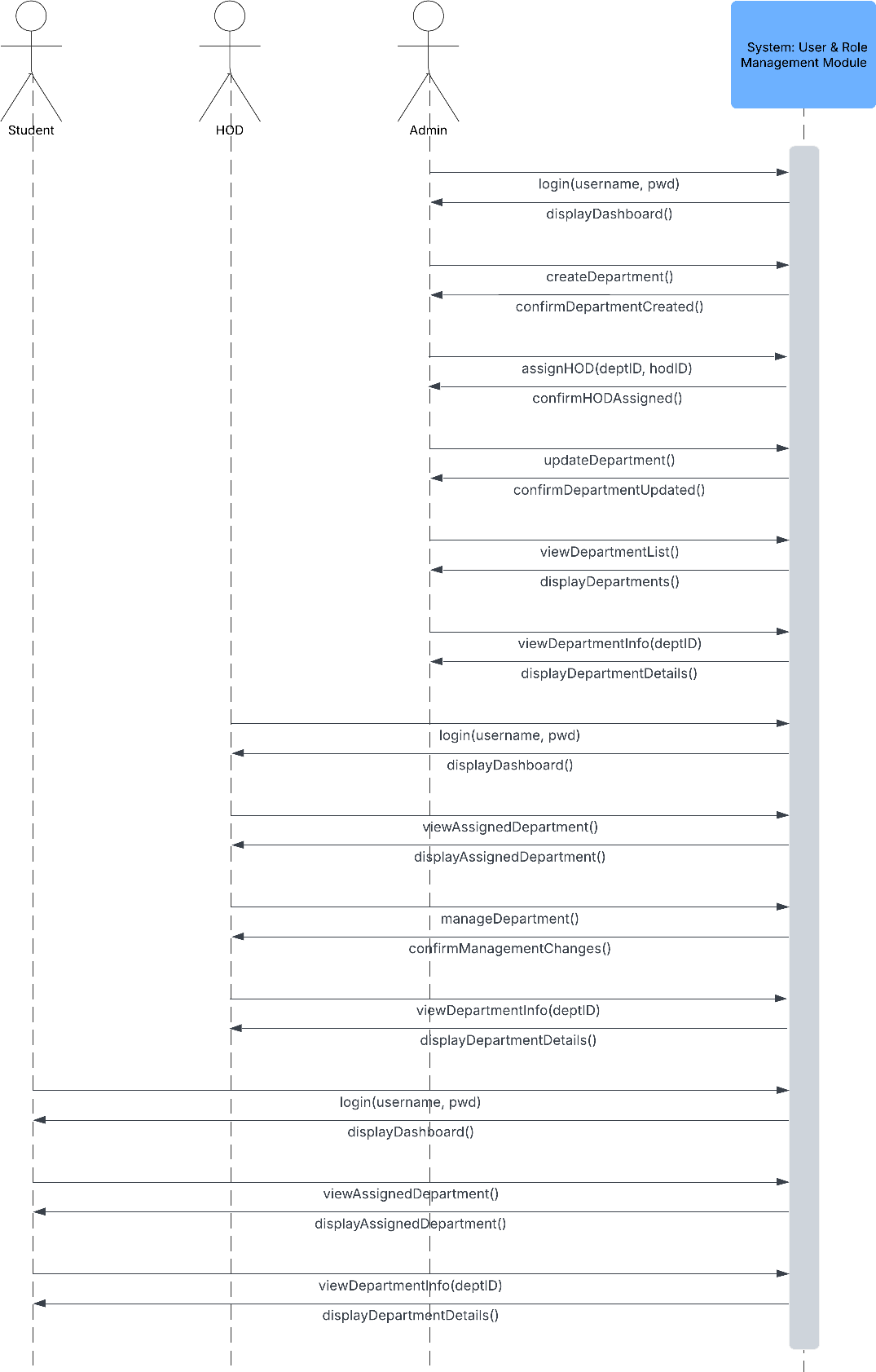


1. **Fully Dressed Use Cases**

**Name: M. Usman Waqar Khan**

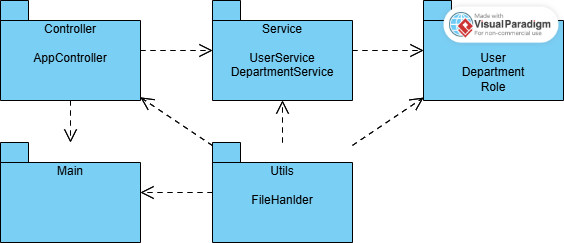
|  |  |
| --- | --- |
| **Use Case Element** | **Details** |
| **Use Case Name** | Assign Department Head |
| **Primary Actor** | Admin (System Administrator) |
| **Goal in Context** | To assign a specific user (e.g., a teacher) as the Head of a selected department |
| **Scope** | User and Role Management Module |
| **Level** | User goal |
| **Stakeholders and Interests** | - **Admin** wants each department to have a responsible head- **System** ensures role control and department integrity- **New Head** should be properly notified and assigned necessary permissions |
| **Preconditions** | - Admin is authenticated and authorized- Departments and eligible users exist in the system |
| **Postconditions (Success)** | - The selected user is marked as Head for the chosen department and can access Head privileges |
| **Postconditions (Failure)** | - The system shows an error; no changes are made to department roles or user status |
| **Main Success Scenario** | 1. Admin selects "Assign Department Head" 2. Admin chooses department 3. System shows current head (if any) and eligible users 4. Admin selects a user 5. Admin confirms 6. System saves assignment and shows success |
| **Extensions (Alternate Flows)** | - **2a.** No departments → System shows: "No departments available"-  **3a.** No eligible users → System shows: "No eligible users for this  department"-  **4a.** Admin cancels → System exits-  **5a.** Error saving → System shows error message |
| **Special Requirements** | - Only one Head per department allowed- Assigned user must be a teacher or staff- System logs the assignment in audit trail |
| **Frequency of Use** | Occasionally — only when assigning or changing department heads |
| **Open Issues** | - Should previous Head be notified or auto-demoted? - Should the new Head receive extra privileges instantly? |

1. **System Sequence Diagram Name: M. Usman Waqar Khan**



**4.Package Diagram**

**Name: M. Usman Waqar Khan**



**5. Class Diagram**



**6. Coding Standards**

✅ 1. Project Structure & Naming

|  |  |
| --- | --- |
| **Standard** | **Example** |
| Package names | model, service, controller, utils, db (all lowercase) |
| Class names | DepartmentService, UserController (PascalCase) |
| Variable names | departmentList, userId (camelCase) |
| Constants | public static final int MAX\_USERS = 100; (ALL\_CAPS) |
| File names | Match the class name exactly (e.g., User.java) |

✅ 2. Class & Method Standards

|  |  |
| --- | --- |
| **Guideline** | **Description** |
| One class per file | Makes debugging and collaboration easier |
| Class should be single responsibility | Department holds data, DepartmentService holds logic |
| Methods should do **one** task only | Split large methods into helper functions |
| Method names in verbs | createUser(), assignRole(), getDepartmentById() |

✅ 3. Comments & Documentation

|  |  |
| --- | --- |
| **Type** | **Guideline** |
| Class comments | Explain class purpose and responsibilities |
| Method comments | Use /\*\* Javadoc \*/ for public methods |
| Inline comments | Only when logic isn’t obvious |
| Block comments | Use to separate logical sections inside larger methods |

**Example:**

/\*\*

* Assigns a department head (HOD) to the given department.
* @param deptId The ID of the department.
* @param hod The user to assign as HOD.

\*/

public void assignHOD(int deptId, User hod) {

...

}

✅ 4. Code Formatting

|  |  |
| --- | --- |
| **Practice** | **Details** |
| Indentation | 4 spaces (no tabs) |

|  |  |
| --- | --- |
| **Practice** | **Details** |
| Brackets | Always use curly braces {} even for one-liners |
| Line Length | Wrap lines after ~100 characters |
| Blank Lines | Use between methods or logical blocks for readability |
| Group imports | Java standard, then third-party, then project imports |

✅ 5. Error Handling

|  |  |
| --- | --- |
| **Rule** | **Example** |
| Use meaningful messages | throw new IllegalArgumentException("User not found with ID: " + id); |
| Catch specific exceptions | Don’t use just Exception |
| Avoid silent failures | Always log or report errors |

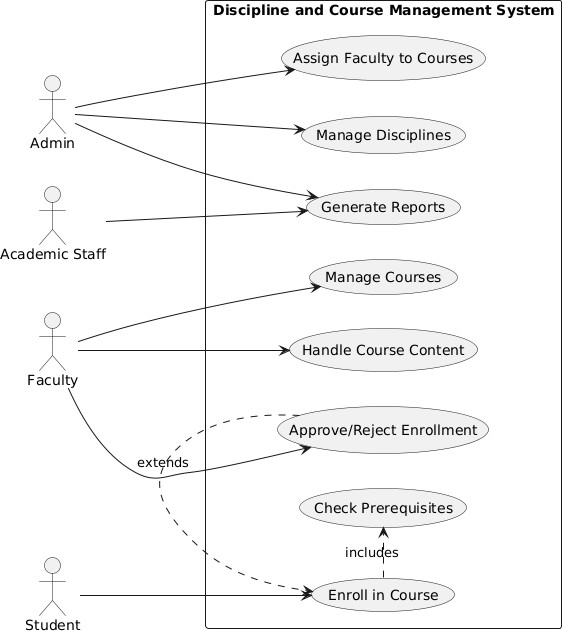
✅ 6. Modularity & Reusability

|  |  |
| --- | --- |
| **Best Practice** | **Description** |
| Don’t hardcode values | Use constants or config files |
| Avoid duplicate code | Reuse methods and utilities |
| Break down logic | Keep services small and focused |
| Interfaces for services | Helps in testing and future DB integration |

✅ 7. Version Control Standards (Git/GitHub)

|  |  |
| --- | --- |
| **Rule** | **Description** |
| Use feature branches | e.g., feature/user-management-module |
| Commit often | Small, meaningful commits |
| Use clear commit messages | "Add DepartmentService with create/update logic" |
| Pull and merge regularly | Avoid long-running branches |
| Document module usage | In README.md or JavaDocs |

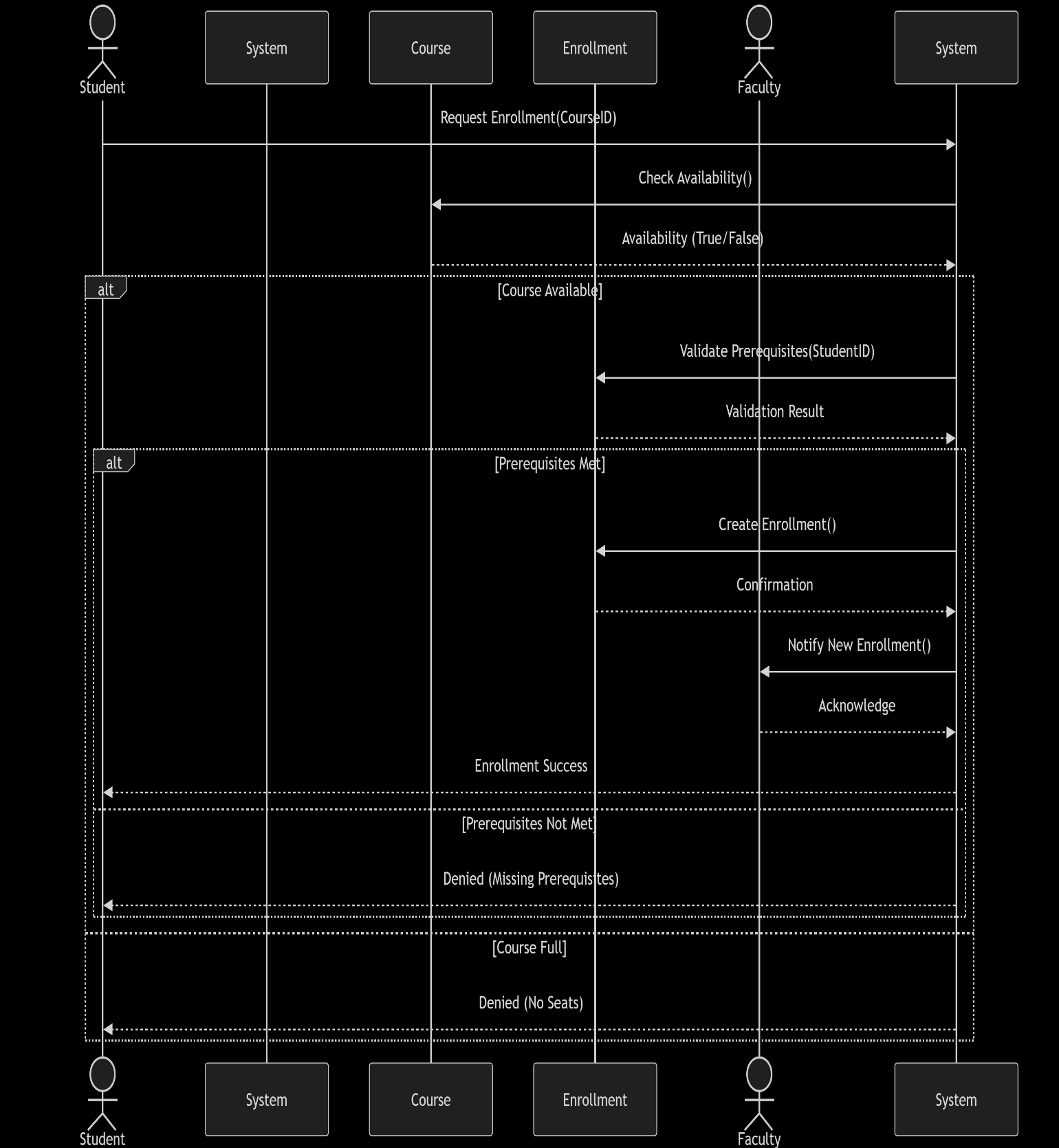
Name: Usman Jamil Abbasi 1: Use-Case Diagram:



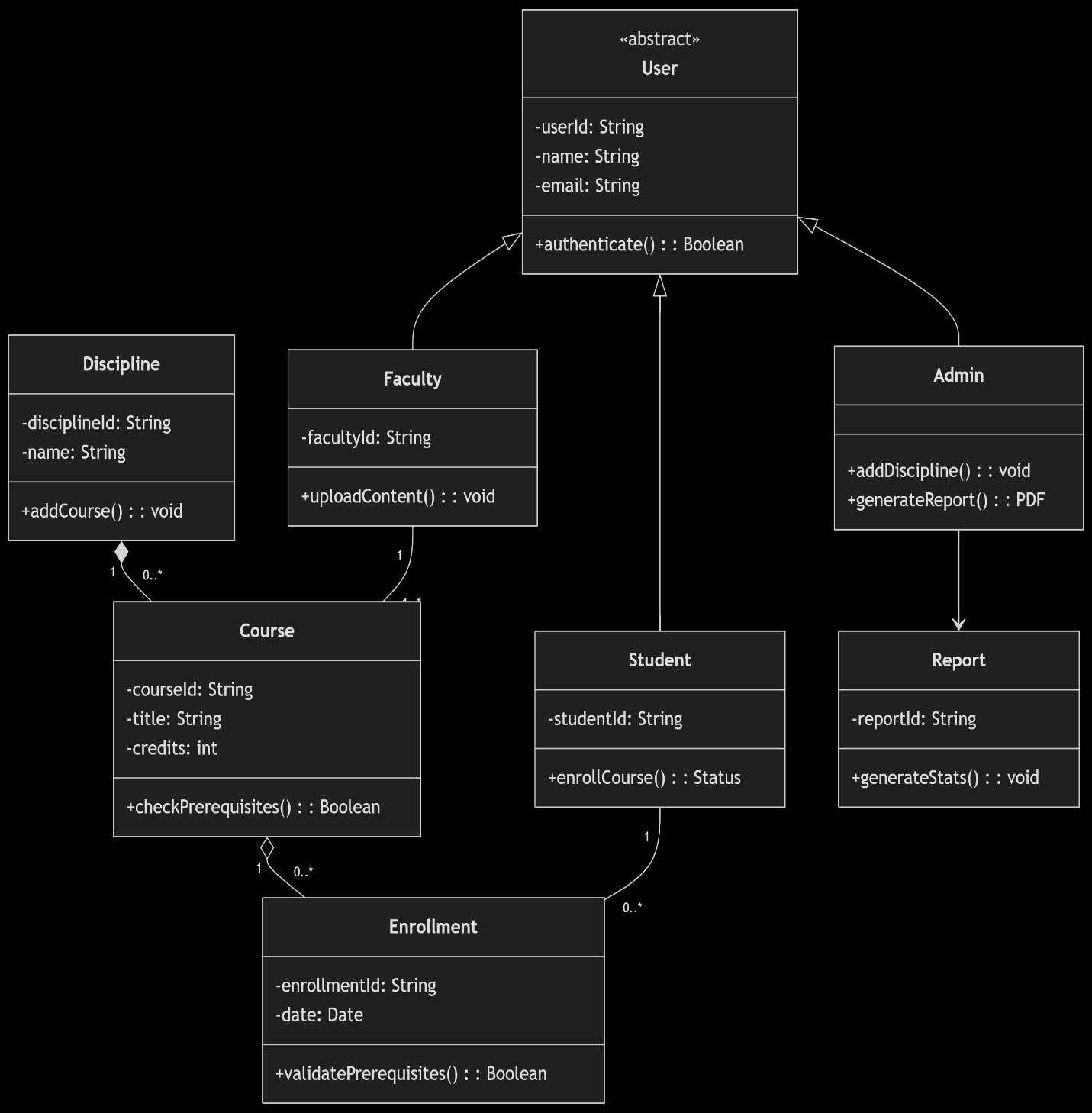
1. Fully Dressed Use Cases:

|  |  |
| --- | --- |
| **Field** | **Details** |
| **Use Case Name** | Enroll in Course |
| **Primary Actor** | Student |
| **Secondary Actors** | System, Faculty |
| **Preconditions** | - Student is logged in- Course exists and has available seats |
| **Main Success Scenario** | 1. Student selects desired course from catalog 2. System verifies course availability and prerequisites (UC8)3. System creates enrollment record 4. System notifies faculty of new enrollment 5. System confirms successful enrollment of student |
| **Extensions** | **3a. Prerequisites not met:** - System displays missing requirements - Use case ends unsuccessfully**3b. Course is**  **full:** - System adds student to waitlist - System notifies student of waitlist position |
| **Special**  **Requirements** | - Enrollment window must be open- Real-time seat availability check |
| **Postconditions** | - If successful: Enrollment record created- If waitlisted: Waitlist position assigned |
| **Related Use Cases** | - **Includes:** Check Prerequisites (UC8)- **Extended by:**  Approve/Reject Enrollment (UC7) |
| **Frequency of Occurrence** | High (hundreds per day during registration periods) |

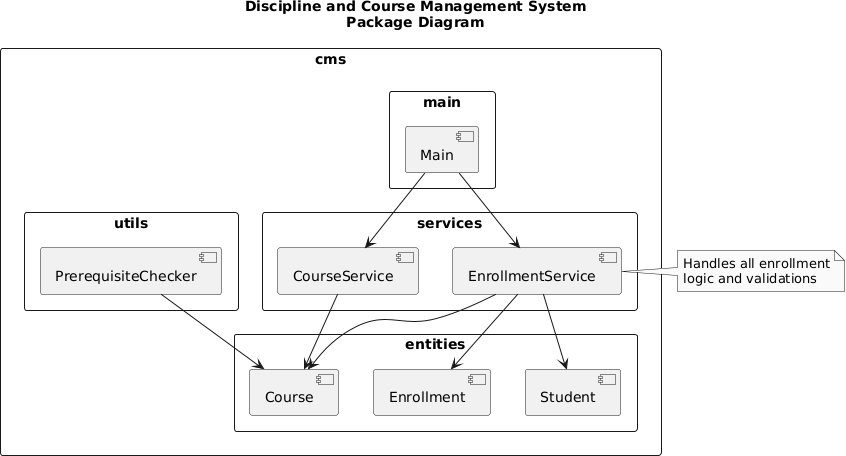
1. System Sequence Diagram:



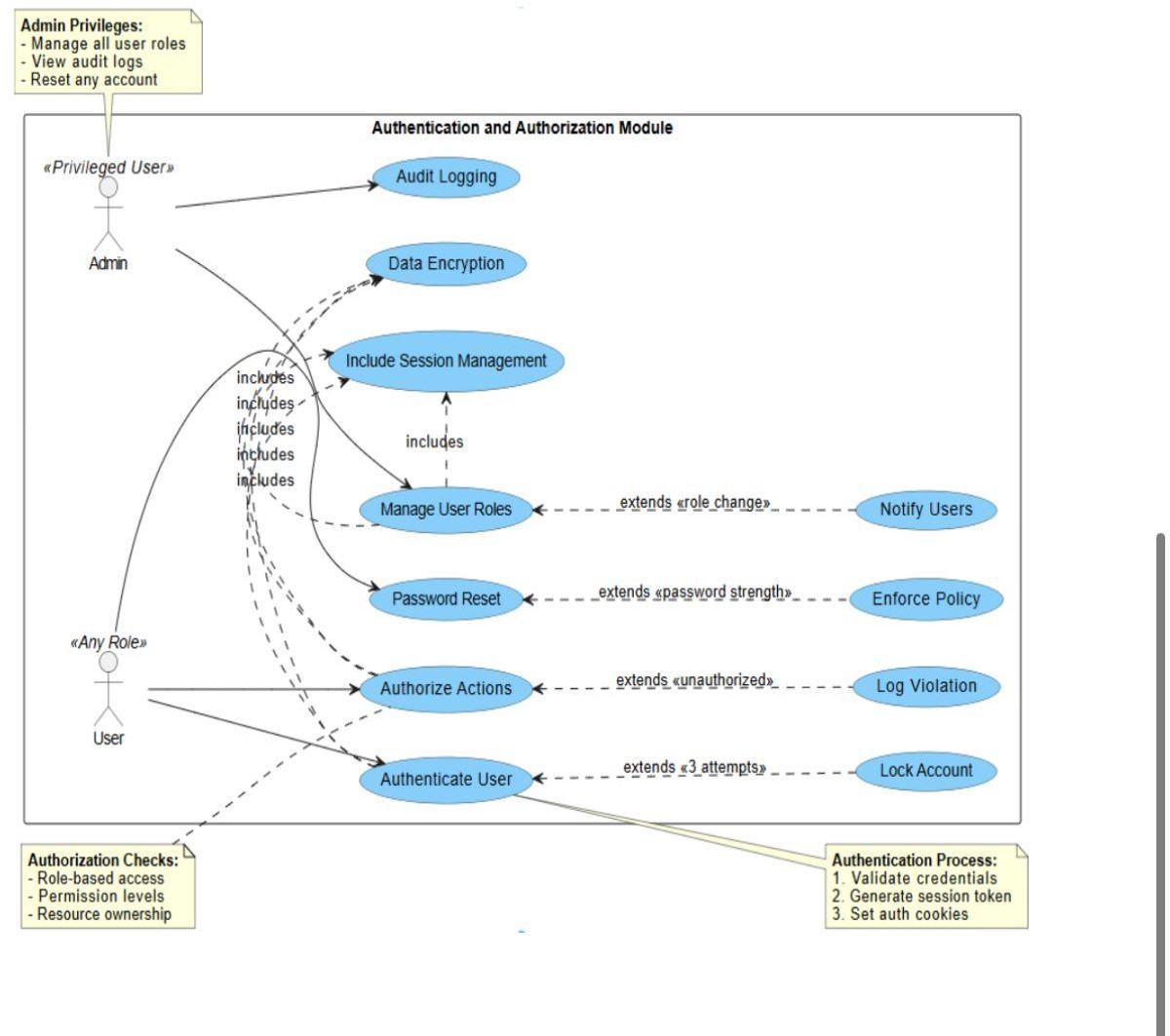
1. Class Diagram:



1. Package Diagram:



# NAME:WIHAJ TAHIR REG NO:FA22-BSE-158



**Use Case**: Manage Admin Privileges

**Actor**: System Admin

# Preconditions:

* Admin is logged in
* Admin has necessary privileges

# Basic Flow:

1. Admin selects action (manage roles/view logs/reset account)
2. System verifies permissions
3. Admin performs action
4. System:
   * Executes request
   * Logs activity
   * Notifies affected users (if needed)
5. Confirms completion

# Alternate Flows:

* *Unauthorized action* → Log violation, notify security
* *Invalid request* → Show error, cancel action

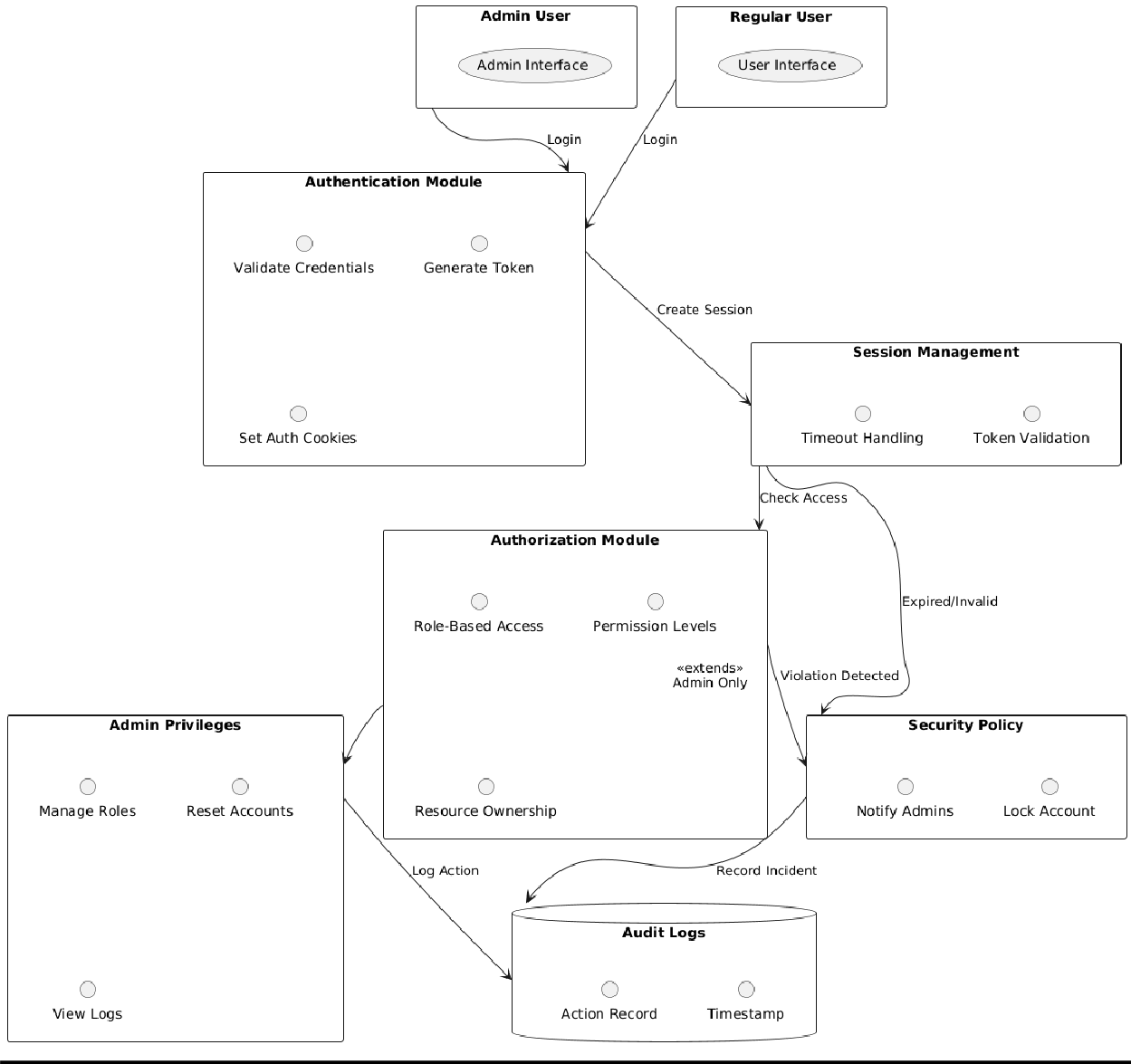
# Postconditions:

* Action completed
* Audit log updated

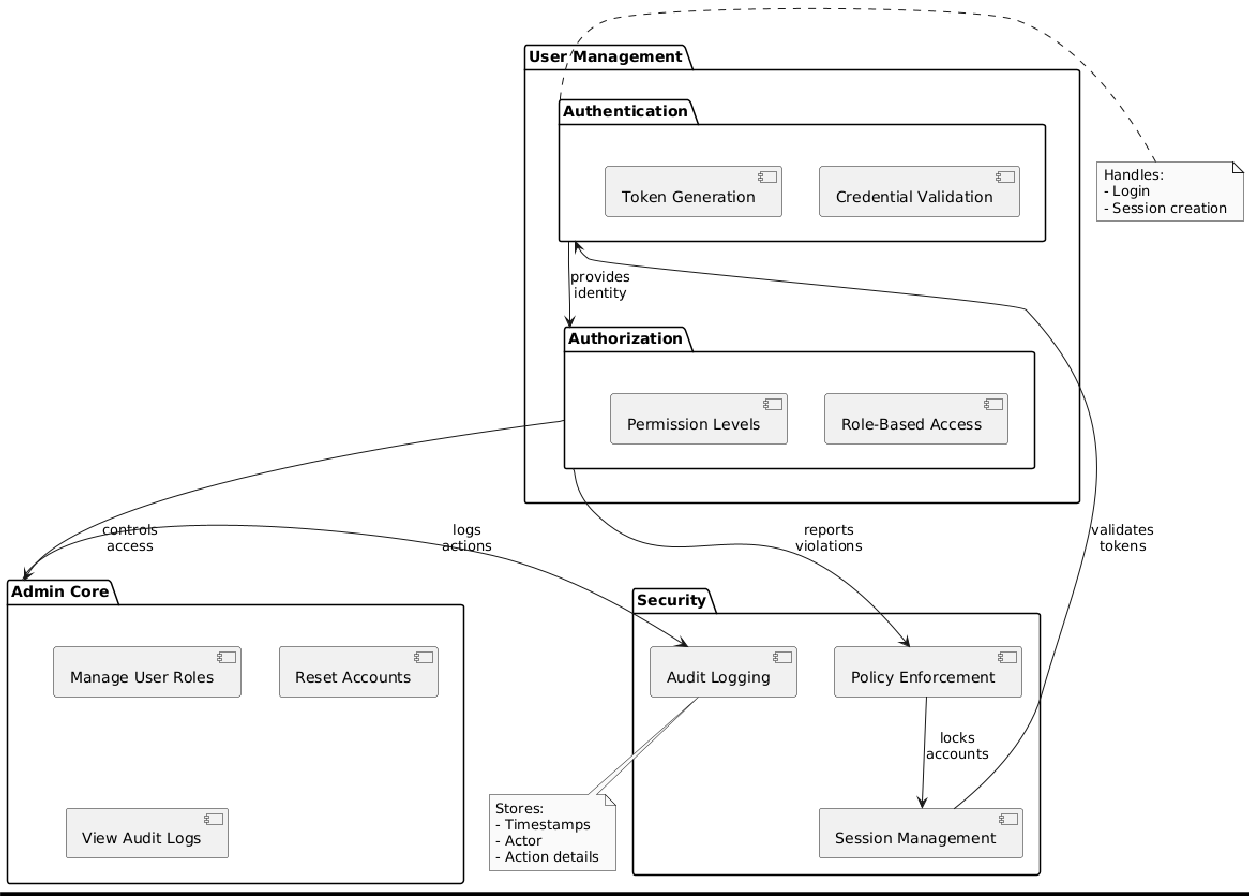
# Special Notes:

* All actions are logged
* Temporary passwords expire after 24hrs

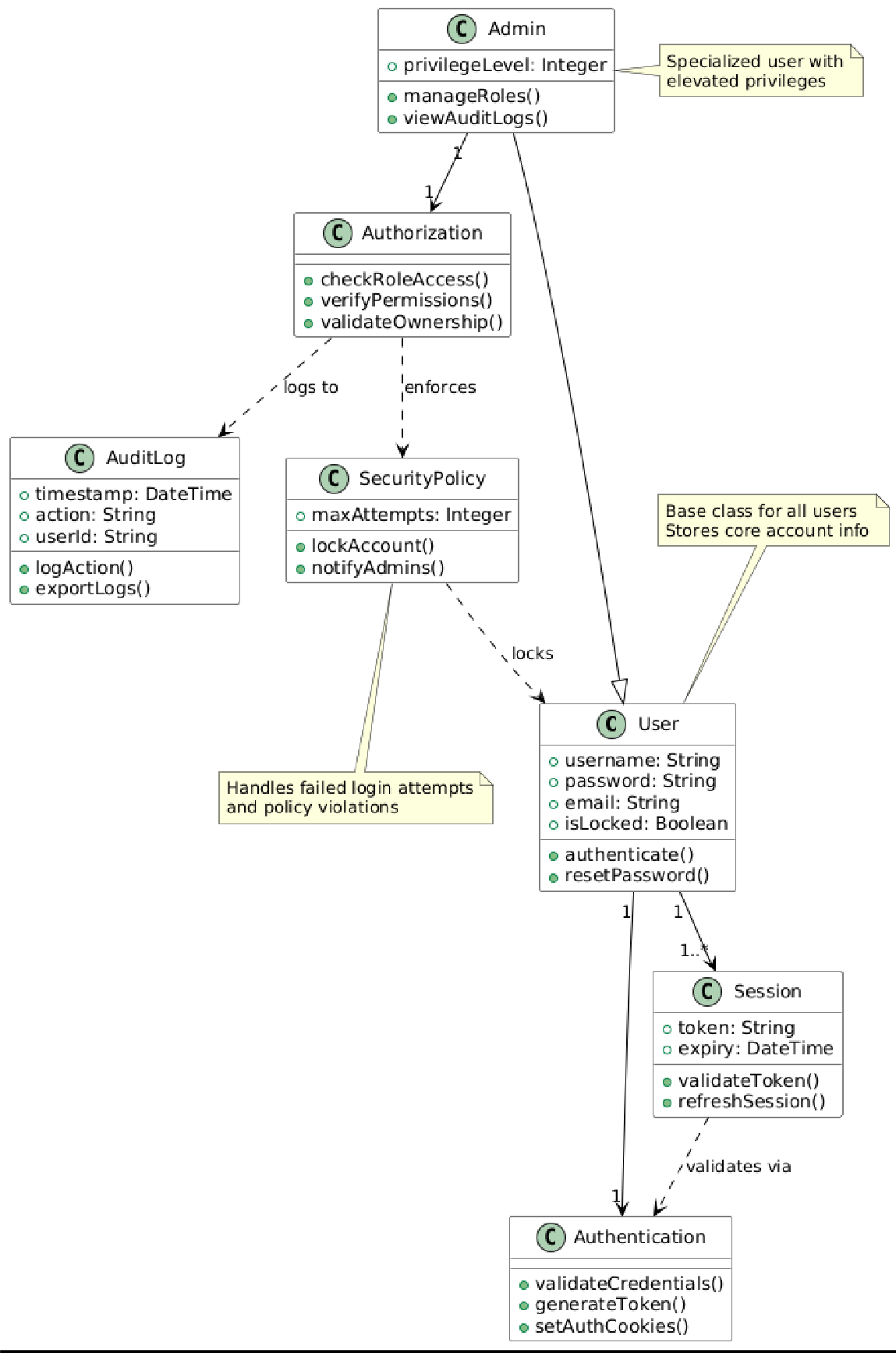
**System use case:**



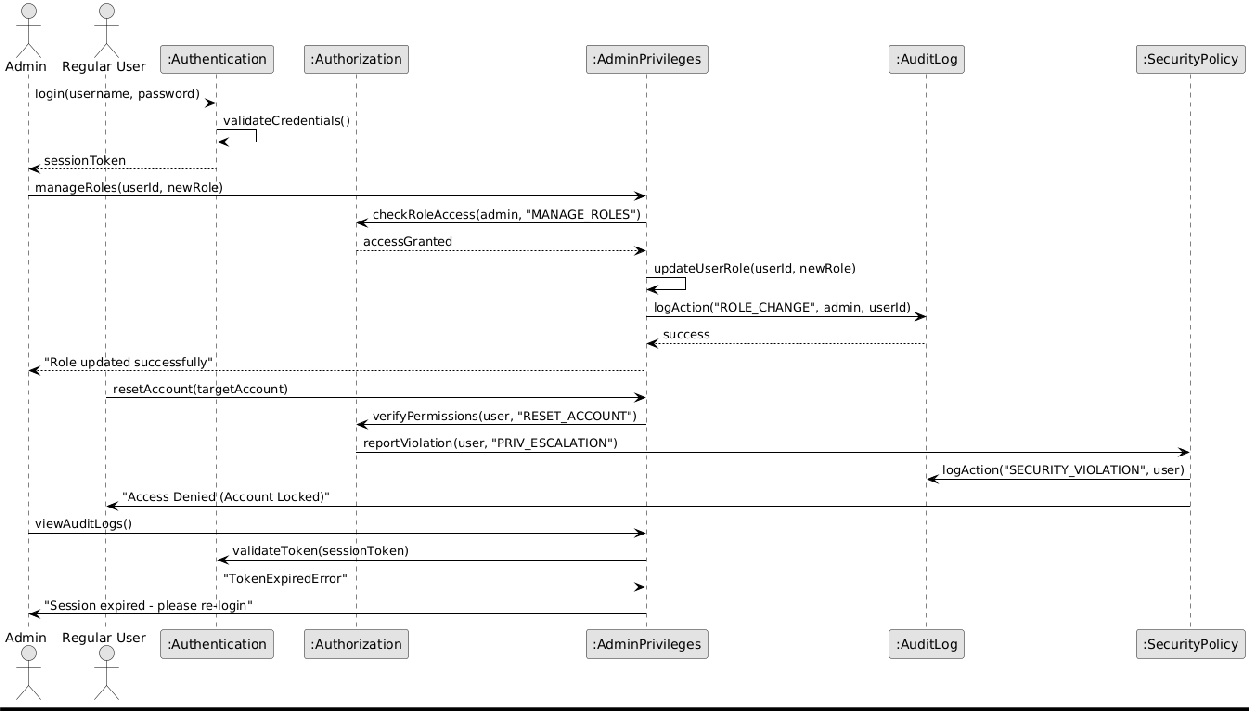
**PACKAGE DIAGRAM:**



**Class diagram:**



**SYSTEM SEQUENCE DIAGRAM:**



**Name: Hashir Habib**

**Reg No: Sp22-BSE-133**

****Course Allocation Module****

This module manages the assignment of courses to teachers for each academic session.

****1. Use Case (Fully Dressed)****

**Use Case Name**: Assign Course to Teacher  
**Primary Actor**: Admin (System Administrator)  
**Goal in Context**: Assign a course to a teacher for a specific semester/term  
**Scope**: Course Allocation Module  
**Level**: User Goal

Stakeholders and Interests:

* **Admin**: Wants to allocate courses fairly and efficiently
* **Teachers**: Want to know their assigned courses clearly
* **System**: Ensures a teacher does not get conflicting schedules or over-assignments

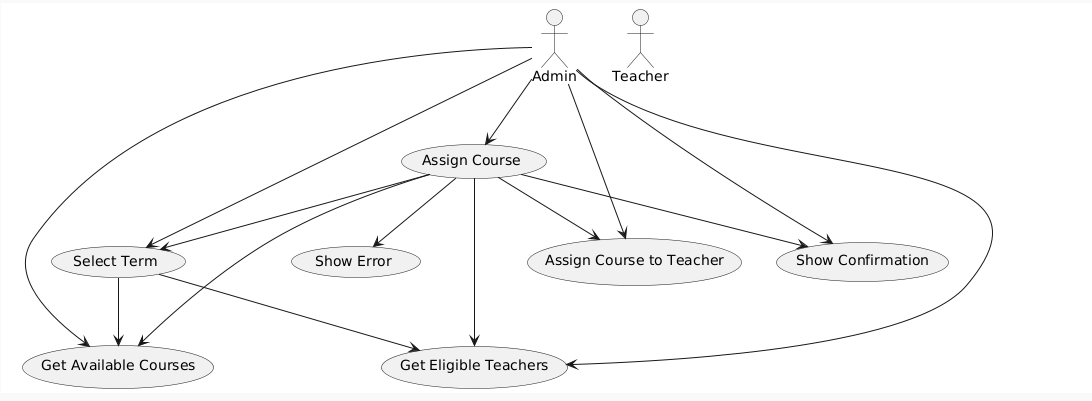
Preconditions:

* Admin is logged in and authorized
* Course and teacher records exist
* Academic term/session is defined

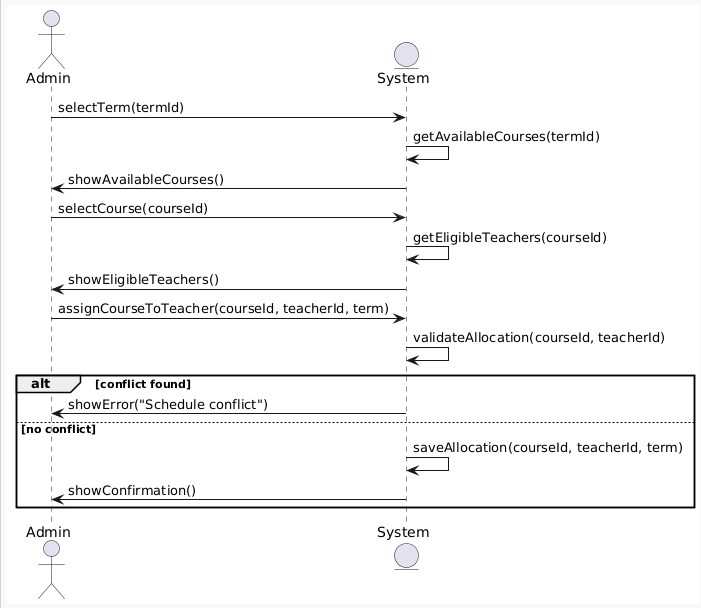
Postconditions:

* (Success) Course is assigned to teacher, and schedule is updated
* (Failure) No assignment made; appropriate error is shown

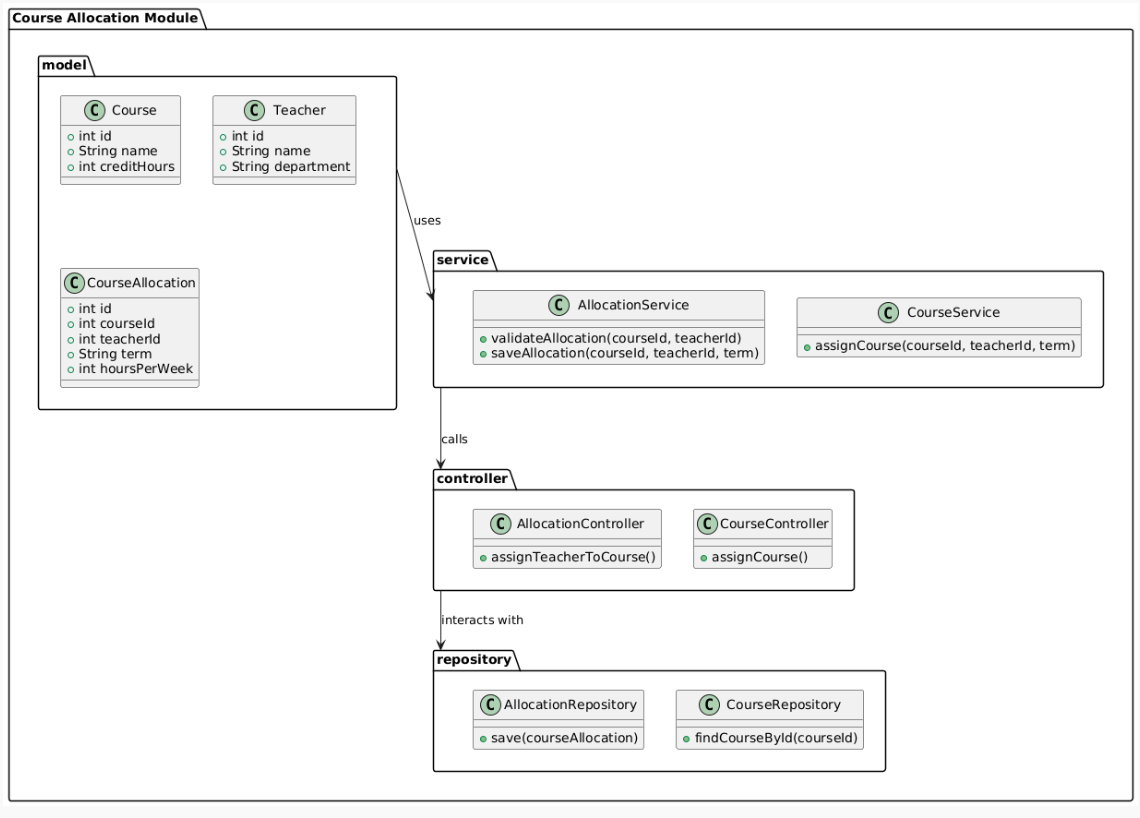
**Use case diagram**



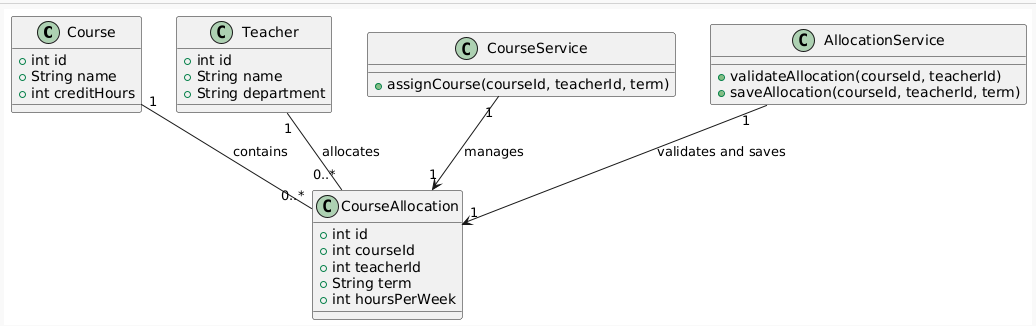
**System sequence diagram**



**Package diagram**



**Class diagram**



**6. Coding Standards**

✅ 1. Project Structure & Naming

|  |  |
| --- | --- |
| **Standard** | **Example** |
| Package names | model, service, controller, utils, db (all lowercase) |
| Class names | DepartmentService, UserController (PascalCase) |
| Variable names | departmentList, userId (camelCase) |
| Constants | public static final int MAX\_USERS = 100; (ALL\_CAPS) |
| File names | Match the class name exactly (e.g., User.java) |

✅ 2. Class & Method Standards

|  |  |
| --- | --- |
| **Guideline** | **Description** |
| One class per file | Makes debugging and collaboration easier |
| Class should be single responsibility | Department holds data, DepartmentService holds logic |
| Methods should do **one** task only | Split large methods into helper functions |
| Method names in verbs | createUser(), assignRole(), getDepartmentById() |

✅ 3. Comments & Documentation

|  |  |
| --- | --- |
| **Type** | **Guideline** |
| Class comments | Explain class purpose and responsibilities |
| Method comments | Use /\*\* Javadoc \*/ for public methods |
| Inline comments | Only when logic isn’t obvious |
| Block comments | Use to separate logical sections inside larger methods |

**Example:**

/\*\*

* Assigns a department head (HOD) to the given department.
* @param deptId The ID of the department.
* @param hod The user to assign as HOD.

\*/

public void assignHOD(int deptId, User hod) {

...

}

✅ 4. Code Formatting

|  |  |
| --- | --- |
| **Practice** | **Details** |
| Indentation | 4 spaces (no tabs) |

|  |  |
| --- | --- |
| **Practice** | **Details** |
| Brackets | Always use curly braces {} even for one-liners |
| Line Length | Wrap lines after ~100 characters |
| Blank Lines | Use between methods or logical blocks for readability |
| Group imports | Java standard, then third-party, then project imports |

✅ 5. Error Handling

|  |  |
| --- | --- |
| **Rule** | **Example** |
| Use meaningful messages | throw new IllegalArgumentException("User not found with ID: " + id); |
| Catch specific exceptions | Don’t use just Exception |
| Avoid silent failures | Always log or report errors |

✅ 6. Modularity & Reusability

|  |  |
| --- | --- |
| **Best Practice** | **Description** |
| Don’t hardcode values | Use constants or config files |
| Avoid duplicate code | Reuse methods and utilities |
| Break down logic | Keep services small and focused |
| Interfaces for services | Helps in testing and future DB integration |

✅ 7. Version Control Standards (Git/GitHub)

|  |  |
| --- | --- |
| **Rule** | **Description** |
| Use feature branches | e.g., feature/user-management-module |
| Commit often | Small, meaningful commits |
| Use clear commit messages | "Add DepartmentService with create/update logic" |
| Pull and merge regularly | Avoid long-running branches |
| Document module usage | In README.md or JavaDocs |

**SHEHRIYAR KHAN**

**SP23-BSE-162**

**Module: Department Management**

# Features:

* + Add/Edit/Delete/View Departments.
  + Assign Head of Department.
  + View department states.

# UML Deliverables:

* + Class Diagram for Department.
  + Activity Diagram for department creation flow.
  + Sequence Diagram for "Assign HoD" process.

# Java Code:

* + GUI form to manage departments.
  + Įogic + file/DB storage.

Department Management Module

### Introduction:

The Department Management Module is a web-based application that enables the management of university departments. It allows for adding, editing, and viewing departments through an intuitive user interface. The module uses HTML, CSS, and JavaScript to provide a professional and visually appealing frontend, while the backend is built to ensure seamless functionality for department-related tasks.

### Features:

1. Add Department: Allows the user to add new departments to the system.
2. Edit Department: Enables the modification of existing department details.
3. View Departments: Displays a list of all departments currently in the system.
4. Professional Layout: The frontend design includes a modern color scheme and user-friendly navigation.
5. Fully Functional Backend: The backend is implemented using JavaScript, ensuring smooth operations for all use cases.

**Detailed Description**

1. Add Department:

The 'Add Department' feature allows users to input details for a new department into the system. This includes the department's name, code, head of department, and other relevant information. The form is designed with clear labels and input validation to ensure the data entered is correct and complete. Upon submission, the department is added to the database, and the user is notified of the successful addition.

1. Edit Department:

The 'Edit Department' functionality enables users to modify existing department details. This feature is accessible through a list of all departments, where users can select a department to edit. Once selected, a form is populated with the department's current details, allowing the user to make changes. Upon updating the department, the changes are saved to the database, and the user is notified of the successful update.

1. View Departments:

The 'View Departments' feature provides users with an overview of all departments currently in the system. This feature displays key details about each department, such as the department name, code, head of department, and other relevant information. The list can be filtered or sorted based on specific criteria, such as department name, to make it easier for users to find specific departments.

1. Layout and Design:

The frontend of the Department Management Module is designed with a modern, clean, and professional layout. The color scheme has been carefully selected to reflect the university's branding while maintaining an easy-to-read interface. The layout is responsive, ensuring that the application is accessible across different devices and screen sizes. Navigation is intuitive, with clear call-to-action buttons and a well-organized layout to minimize user confusion and maximize efficiency.

1. Backend Functionality:

The backend of the Department Management Module is implemented using JavaScript. It handles all user input, including adding and editing department data, and ensures that the system operates smoothly. The backend interacts with the frontend to receive data from the user, validate it, and send it to the server for processing. Error handling is integrated to ensure that invalid or incomplete data is not submitted. Additionally, AJAX calls are used to update the list of

departments dynamically without requiring the page to reload, creating a seamless user experience.

* + USE Cases for Department Management Module:

## ​ UC-01: Add Department:

* + Actor: Admin
  + Description: Admin adds a new department by entering the department name, code, and description.
  + Preconditions: Admin is logged in.
  + Postconditions: New department is saved in the system.
  + Exceptions: Duplicate department name/code.

## ​ UC-02: Edit Department:

* + Actor: Admin
  + Description: Admin updates details of an existing department.
  + Preconditions: Department exists in the system.
  + Postconditions: Updated information is saved.
  + Exceptions: Editing restricted fields like ID.

## ​ UC-03: Delete Department:

* + Actor: Admin
  + Description: Admin deletes a department if it's not associated with any active programs or users.
  + Preconditions: Department has no dependencies.
  + Postconditions: Department is permanently removed.
  + Exceptions: Deletion blocked if dependencies exist.

## ​ UC-04: View Departments:

* + Actor: Admin
  + Description: Admin can view a list of all departments along with basic details.
  + Preconditions: None
  + Postconditions: Įist is displayed.
  + Exceptions: Database connection error.

## ​ UC-05: Assign Head of Department (HoD):

* + Actor: Admin
  + Description: Admin assigns or changes the HoD for a department from available faculty.
  + Preconditions: Faculty list and department exist.
  + Postconditions: HoD is assigned and updated in the system.
  + Exceptions: Selected person not part of department or already assigned elsewhere.

## ​ UC-06: Search Department:

* + Actor: Admin
  + Description: Admin searches for a department by name or code.
  + Preconditions: Department records exist.
  + Postconditions: Matching departments are shown.
  + Exceptions: No match found.

## ​ UC-07: Search Department:

* + Actor: Admin
  + Description: Allows searching departments by name, code, or faculty.
  + Benefit: Speeds up access to specific department data.

## ​ UC-08: View Department Details:

* + Actor: Admin
  + Description: View comprehensive profile of a department including faculty list, assigned HoD, related disciplines, and history.
  + Benefit: Provides insight into department structure.

## ​ UC-09: Filter Departments:

* + Actor: Admin
  + Description: Filters departments by faculty, HoD assigned status, or active/inactive status.
  + Benefit: Makes management easier in large institutions.

## ​ UC-10: Export Department Data:

* + Actor: Admin
  + Description: Exports department details to Excel, PDF, or CSV format.
  + Benefit: Useful for reports, accreditation, or audits.

## ​ UC-11: Archive Department:

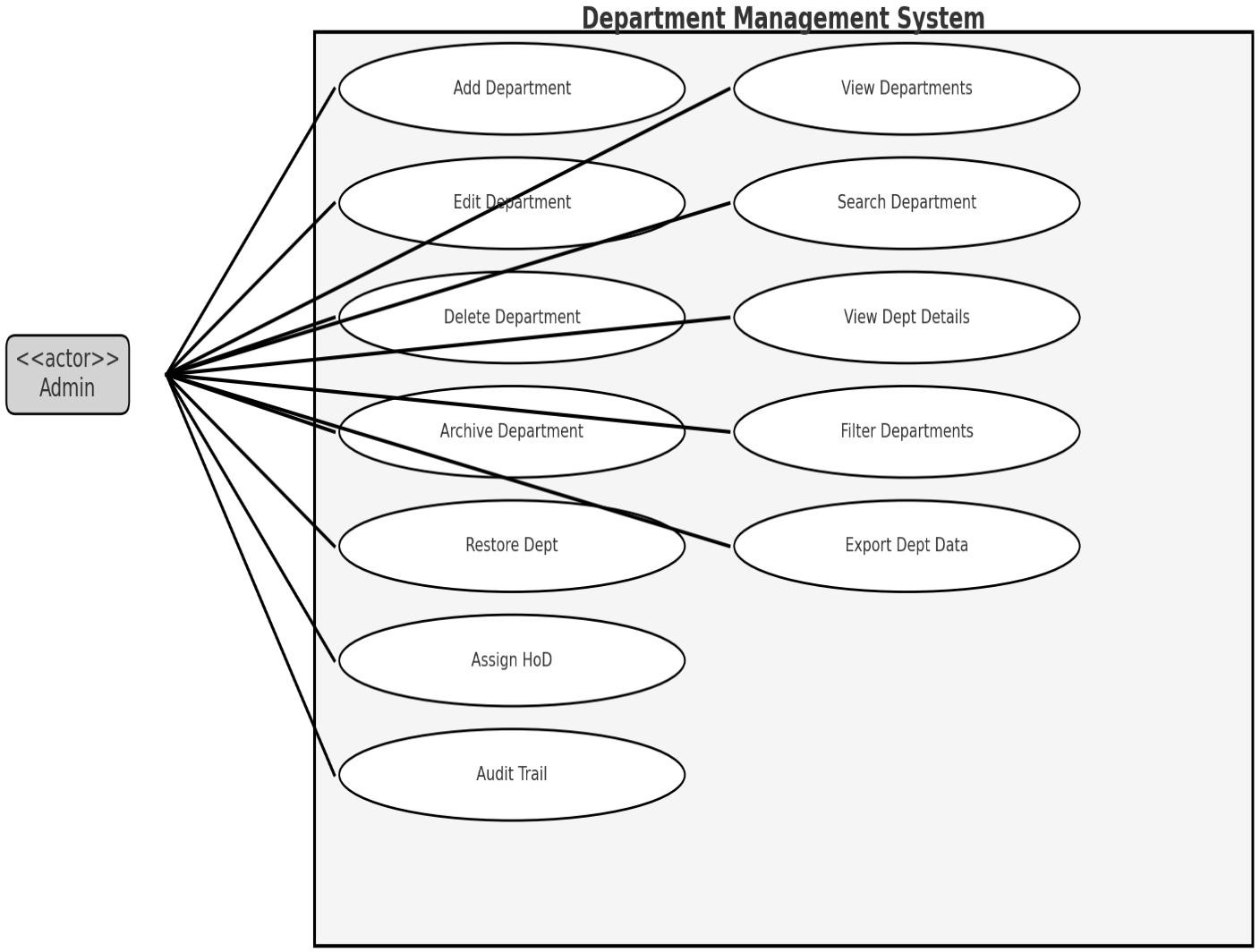
* + Actor: Admin
  + Description: Marks a department as inactive instead of deleting it permanently.
  + Benefit: Maintains historical data without cluttering active records.

## ​ UC-12: Restore Archived Department:

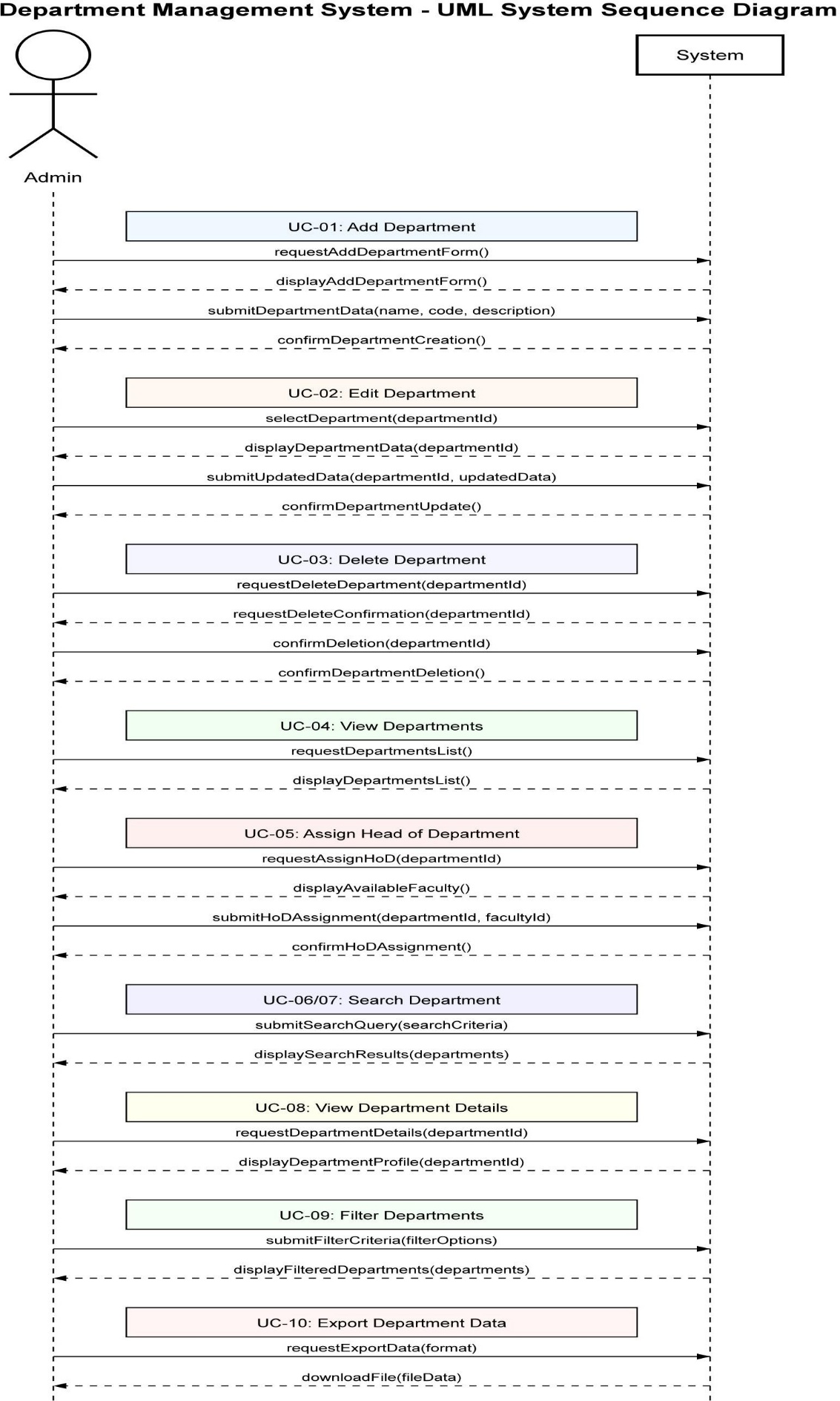
* + Actor: Admin
  + Description: Restores a previously archived department back to active status.
  + Benefit: Useful if a department is re-opened.

## ​ UC-13: Audit Trail for Department Changes:

* + Actor: Admin
  + Description: View a log of changes made to department details (who made the change and when).
  + Benefit: Ensures accountability and tracking

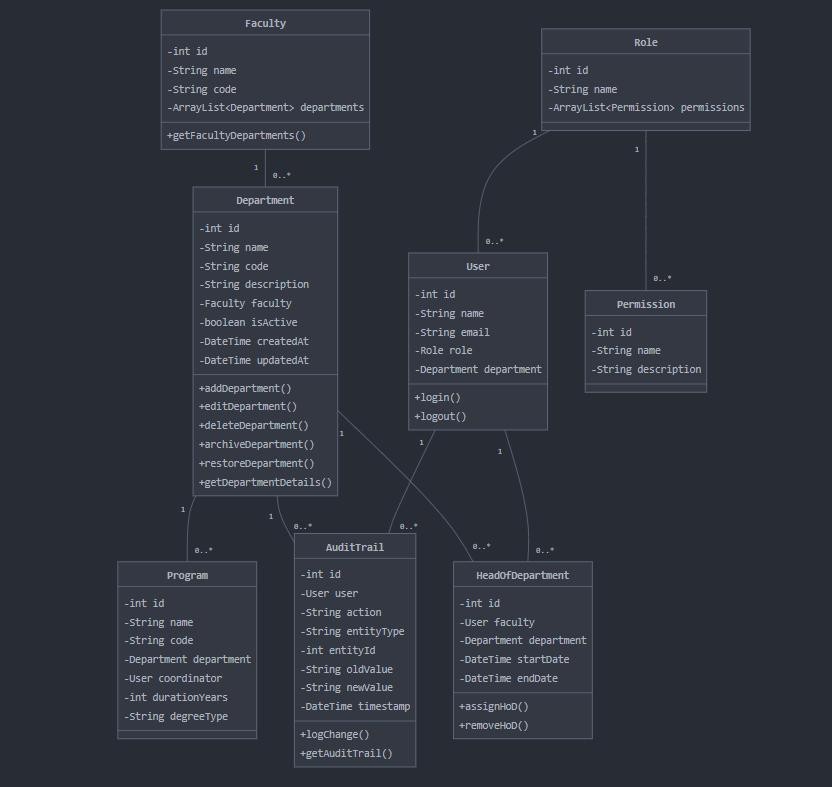


* **SSD Diagram for Department Management Module:**



A close-up of a document

AI-generated content may be incorrect.

**Class Diagram:**

**Fully Dressed Use Cases:**

Use Case 1: Edit Department:

|  |  |
| --- | --- |
| Field | Description |
| Use Case ID | UC-01 |
| Use Case Name | Add Department |
| Primary Actor | Admin |
| Description | Admin adds a new department by entering the department name,  code, and description. |
| Stakeholders and Interests | Admin – wants to successfully add  departments with proper validation. |
| Preconditions | Admin is logged in and has the necessary  permissions. |
| Postconditions | The department is successfully created and saved in the system database. |
| Exceptions: | If a department with the same name or code already exists, an error message  is shown indicating a duplicate department name/code. |
| Basic Flow: | * Admin logs into the system and navigates to the "Add Department" section. * Admin enters the department name in the provided field. * Admin enters the department code. * Admin provides a description for the department. * Admin clicks the "Save" button. * The system checks for duplicates |

|  |  |
| --- | --- |
|  | (department name or code).   * If no duplicate is found, the system saves the department to the database. * A confirmation message is displayed to the Admin, indicating that the department was successfully added. |
| Alternative Flow: | If a duplicate department name/code is detected, the system prevents saving and displays an error message with a prompt to re-  enter valid values. |
| Main Success Scenario | 1. Admin logs into the system. 2. Navigates to the “Add Department” section. 3. Enters department name, code, and description. 4. Clicks “Save”. 5. System validates and checks for duplicates. 6. If valid, department is saved. 7. Confirmation message is shown. |
| Extensions | 3a. Required field is missing – Prompt admin to fill all fields.  5a. Duplicate name or code – Show error and ask for correction. |
| Special Requirements | Unique department code validation. Real-  time field validation. |
| Frequency of Use | Occasionally when a new department is  established. |
| Assumptions | Admin has accurate information and requires  permissions. |
| Notes and Issues | Consider auto-generating codes to avoid duplication. |

Use Case 2: Edit Department:

|  |  |
| --- | --- |
| Field | Description |
| Use Case ID | UC-02 |
| Use Case Name | Edit Department |
| Primary Actor | Admin |
| Stakeholders and Interests | Admin – wants to maintain accurate  department data. |
| Description | The admin updates the details of an existing department (such as department name, code, and  description). |
| Preconditions | The department exists and Admin is logged in  with proper access. |
| Postconditions | Changes to the department are successfully  saved in the system. |
| Exceptions: | Admin is restricted from editing  certain fields like the department ID. |
| Basic Flow | * Admin logs into the system and navigates to the "Departments" section. * Admin selects the department they wish to edit from the list of existing departments. * Admin updates the department name, code, and/or description as needed. * Admin clicks the "Save" button to apply changes. * The system validates the changes, ensuring the new department code does not already exist. * The updated department details are saved in the system. |

|  |  |
| --- | --- |
|  | * A confirmation message is displayed to the Admin, indicating the successful update. |
| Alternative Flow | If the Admin attempts to change a restricted field (such as the department ID), the system prevents saving and displays an error message indicating that the  ID field cannot be changed. |
| Main Success Scenario | 1. Admin logs into the system. 2. Navigates to the “Departments” section. 3. Selects a department to edit. 4. Makes necessary changes to name, code, or description. 5. Clicks “Save”. 6. System validates changes. 7. Updates are saved. 8. Confirmation message is shown. |
| Extensions | 4a. Attempt to edit department ID – Field is disabled.  6a. Duplicate code entered – Error message shown. |
| Special Requirements | ID field is non-editable. Unique constraint on  department code. |
| Frequency of Use | Occasionally, based on administrative  decisions. |
| Assumptions | Admin ensures data accuracy and  consistency. |
| Notes and Issues | Include audit trail for edits in future  iterations. |

**Use Case: Delete Department:**

|  |  |
| --- | --- |
| Field | Description |
| Use Case ID | UC-03 |
| Use Case Name | Delete Department |
| Primary Actor | Admin |
| Description | Admin deletes a department if it's not associated with any active programs or users. |
| Goal in Context | To remove a department that is no longer in use and has no dependencies (e.g., linked users or programs). |
| Scope | Department Management System |
| Level | User goal |
| Stakeholders and Interests | Admin – needs to clean up inactive or deprecated departments.  Institution – avoids data clutter and maintains system hygiene. |
| Preconditions | Admin is logged in.  Department has no active users or programs associated. |
| Postconditions | Department is permanently deleted and no longer visible in lists. |
| Main Success Scenario | 1. Admin logs into the system. 2. Navigates to department list. 3. Selects a department. 4. Clicks “Delete”. 5. System checks for dependencies. 6. If none, department is deleted. 7. Confirmation is shown. |
| Alternative Flows | 5a. Dependencies found:   * System blocks deletion and displays dependency info. * Admin is prompted to remove dependencies first. |

|  |  |
| --- | --- |
| Exceptions | Database error, session timeout, or unexpected failure during dependency check. |
| Special Requirements | Confirmation prompt, deletion logs, secure access to delete functionality. |
| Frequency of Use | Occasional |
| Assumptions | Admin verifies department is truly no longer needed. |
| Open Issues | Should deletion be soft (archived) instead of permanent? |

**UC-04: View Departments**

* **Use Case ID:** UC-04
* **Use Case Name:** View Departments
* **Actor(s):** Admin
* **Description:** Admin can view a list of all departments along with basic details.
* **Assumptions:**
  + The Admin has the necessary permissions to view the department data.
* **Preconditions:**
  + Admin is logged in to the system.
  + Admin has access to department data stored in the system.
* **Postconditions:**
  + A list of departments is displayed, including department name, department code, and HoD.
* **Main Flow (Basic Path):**

1. Admin navigates to the "View Departments" section.
2. The system retrieves and displays a list of all departments with basic details such as department name, department code, and HoD.

* **Alternate Flow (if any):**
  + If there are no departments, the system displays a message: "No departments available."
* **Exceptions:**
  + If there is a database connection failure, the system displays an error message: "Unable to retrieve department data."
* **Frequency of Use:**
  + Typically used when the Admin wants to review the list of departments in the institution.

**UC-05: Assign Head of Department (HoD)**

* **Use Case ID:** UC-05
* **Use Case Name:** Assign Head of Department (HoD)
* **Actor(s):** Admin
* **Description:** Admin assigns or changes the HoD for a department.
* **Assumptions:**
  + The Admin has access to faculty records and department details.
* **Preconditions:**
  + Admin is logged in and authorized to make changes.
  + Faculty list and department records are available in the system.
* **Postconditions:**
  + The HoD is assigned to the selected department and updated in the system.
* **Main Flow (Basic Path):**

1. Admin selects the department to assign an HoD.
2. Admin chooses a faculty member from the available list.
3. The system validates if the faculty member is eligible to be assigned as HoD.
4. The system assigns the selected faculty member as the HoD for the department.
5. The system updates the department record with the new HoD information.
6. A success message is displayed confirming the HoD assignment.
   * **Alternate Flow (if any):**
     + If the faculty member is already assigned as HoD to another department, the system displays an error message: "Faculty member already assigned as HoD in another department."
   * **Exceptions:**
     + If no faculty members are available for assignment, the system displays a message: "No available faculty members."
   * **Frequency of Use:**
     + Typically used when a new HoD needs to be assigned or when an existing HoD is replaced.

**UC-06: Search Department**

* + **Use Case ID:** UC-06
  + **Use Case Name:** Search Department
  + **Actor(s):** Admin
  + **Description:** Admin searches for departments by name or code.
  + **Assumptions:**
    - Department data is available and up-to-date in the system.
  + **Preconditions:**
    - Admin is logged in and has access to the department database.
    - Department records exist in the system.
  + **Postconditions:**
    - Departments matching the search criteria (name or code) are displayed.
  + **Main Flow (Basic Path):**

1. Admin enters the department name or department code in the search field.
2. The system searches for matching departments.
3. The system displays a list of matching departments.
   * **Alternate Flow (if any):**
     + If no matching departments are found, the system displays: "No departments found."
   * **Exceptions:**
     + If a database error occurs, the system displays: "Unable to search for departments."
   * **Frequency of Use:**
     + This use case is used regularly to find a specific department based on its name or code.

**UC-07: Advanced Department Search**

* + **Use Case ID:** UC-07
  + **Use Case Name:** Advanced Department Search
  + **Actor(s):** Admin
  + **Description:** Admin can search departments using multiple criteria such as department name, department code, or faculty.
  + **Assumptions:**
    - Department records are structured to include faculty and other attributes.
  + **Preconditions:**
    - Admin is logged in and has access to the department and faculty data.
  + **Postconditions:**
    - The system displays departments that match the search criteria (e.g., name, code, faculty).
  + **Main Flow (Basic Path):**

1. Admin enters search criteria such as department name, code, or faculty.
2. The system processes the criteria and searches the department database.
3. The system returns matching departments and displays them to the Admin.
   * **Alternate Flow (if any):**
     + If no matching departments are found, the system displays: "No results found."
   * **Exceptions:**
     + If an error occurs while processing the search, the system displays: "Search failed. Please try again later."
   * **Frequency of Use:**
     + Typically used when Admin needs to search for specific departments based on detailed criteria.

**UC-08: View Department Details**

* + **Use Case ID:** UC-08
  + **Use Case Name:** View Department Details
  + **Actor(s):** Admin
  + **Description:** Admin can view the detailed profile of a department.
  + **Assumptions:**
    - The department's full profile, including the faculty list and HoD, is available in the system.
  + **Preconditions:**
    - Admin is logged in.
    - Department records are available in the system.
  + **Postconditions:**
    - The department’s detailed information is displayed, including HoD, faculty members, and related disciplines.
  + **Main Flow (Basic Path):**

1. Admin selects a department from the department list.
2. The system retrieves and displays the department's detailed profile.
   * **Alternate Flow (if any):**
     + If the department's details cannot be retrieved, the system displays an error message: "Unable to retrieve department details."
   * **Exceptions:**
     + If the system encounters a database failure, an error message is shown.
   * **Frequency of Use:**
     + Used when Admin needs a full overview of a specific department.

**UC-09: Filter Departments**

* + **Use Case ID:** UC-09
  + **Use Case Name:** Filter Departments
  + **Actor(s):** Admin
  + **Description:** Admin can filter departments based on various attributes such as faculty, HoD assigned status, and department status (active/inactive).
  + **Assumptions:**
    - Department and faculty records are maintained and categorized by status and other attributes.
  + **Preconditions:**
    - Admin is logged in and authorized to filter department records.
    - Department data is available in the system.
  + **Postconditions:**
    - Departments that meet the selected criteria are displayed.
  + **Main Flow (Basic Path):**

1. Admin selects the filter criteria (faculty, HoD assigned, active/inactive).
2. The system filters the departments based on the criteria.
3. The system displays the filtered list of departments.
   * **Alternate Flow (if any):**
     + If no departments meet the filter criteria, the system displays: "No results found."
   * **Exceptions:**
     + If filtering fails, the system displays: "Unable to filter departments."
   * **Frequency of Use:**
     + Used when Admin needs to quickly find departments based on specific criteria.

**UC-10: Export Department Data**

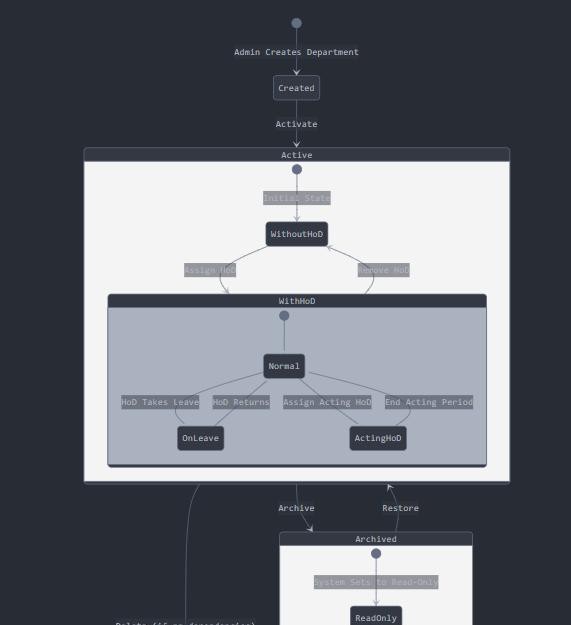
* + **Use Case ID:** UC-10
  + **Use Case Name:** Export Department Data
  + **Actor(s):** Admin
  + **Description:** Admin can export department details into formats such as PDF, Excel, or CSV.
  + **Assumptions:**
    - The system supports export functionality in multiple formats.
  + **Preconditions:**
    - Admin is logged in and has permission to export data.
    - Department data is available in the system.
  + **Postconditions:**
    - The department data is successfully exported to the selected file format.
  + **Main Flow (Basic Path):**

1. Admin selects the "Export" option.
2. Admin chooses the file format (PDF, Excel, or CSV).
3. The system generates the selected file and allows the Admin to download it.
   * **Alternate Flow (if any):**
     + If the export fails, the system displays: "Export failed. Please try again."
   * **Exceptions:**
     + If there is a system failure during the export process, an error message is displayed.
   * **Frequency of Use:**
     + This use case is typically used when reports or records need to be shared externally or archived.

**UC-11: Archive Department**

* + **Use Case ID:** UC-11
  + **Use Case Name:** Archive Department
  + **Actor(s):** Admin
  + **Description:** Admin can mark a department as inactive (archived) rather than deleting it permanently.
  + **Assumptions:**
    - Admin is authorized to archive departments.
    - Archived departments are stored for future reference.
  + **Preconditions:**
    - Admin is logged in.
    - The department exists in the system.
  + **Postconditions:**
    - The department is marked as archived, making it inactive in the system.
  + **Main Flow (Basic Path):**

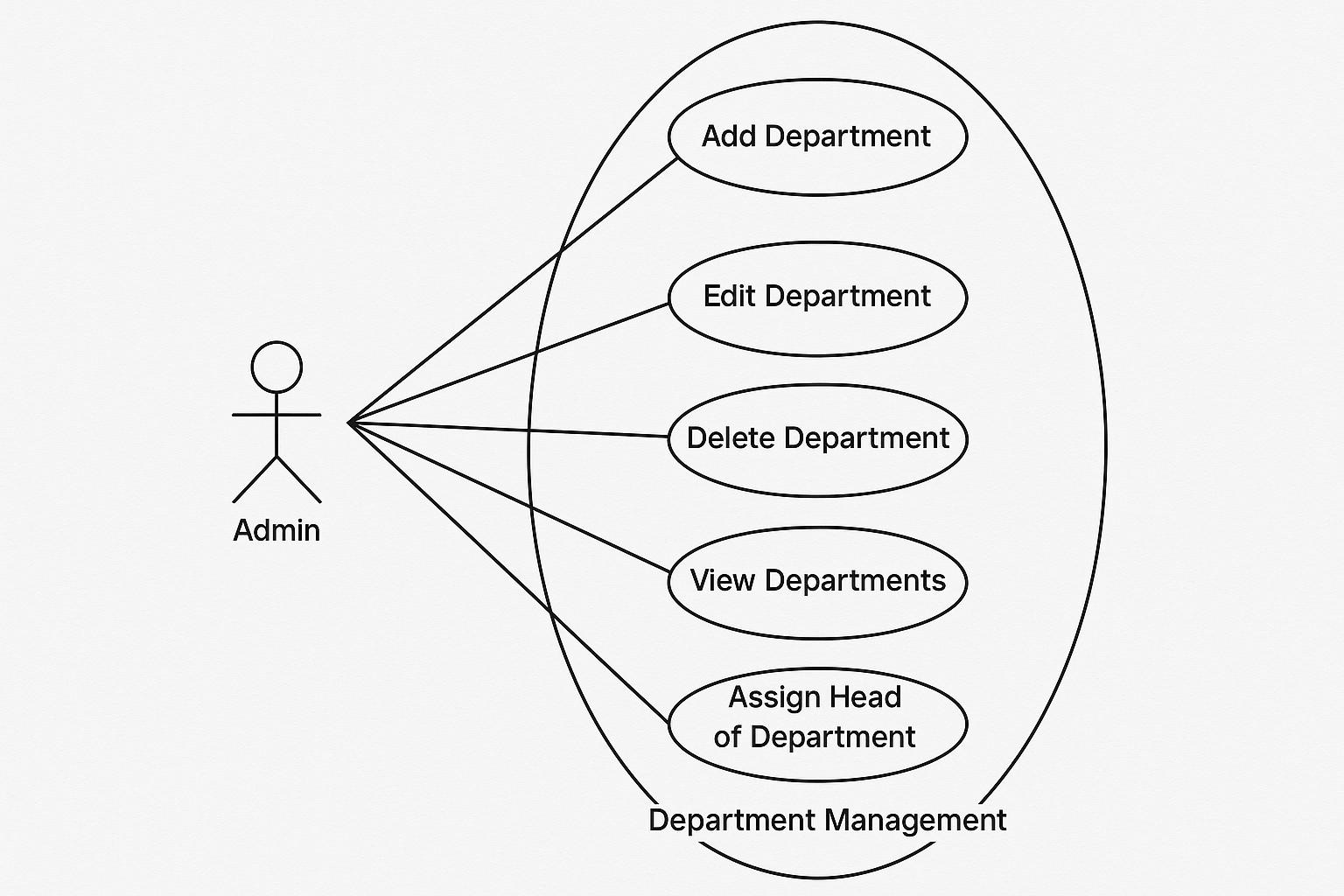
1. Admin selects the department to archive.
2. Admin clicks "Archive" to mark the department as inactive.
3. The system updates the department’s status to archived.
4. A success message is displayed confirming the department has been archived.
   * **Alternate Flow (if any):**
     + If the department cannot be archived, an error message is displayed: "Unable to archive department."
   * **Exceptions:**
     + If the department does not exist in the system, an error message is displayed.
   * **Frequency of Use:**
     + Typically used when a department is no longer operational but needs to be preserved for historical records.



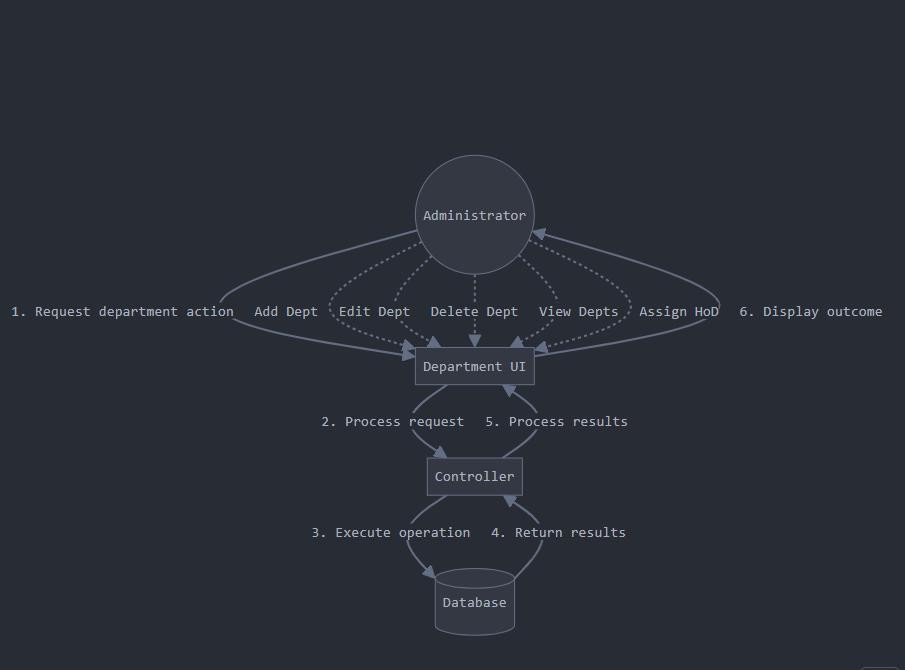
**State Diagram:**

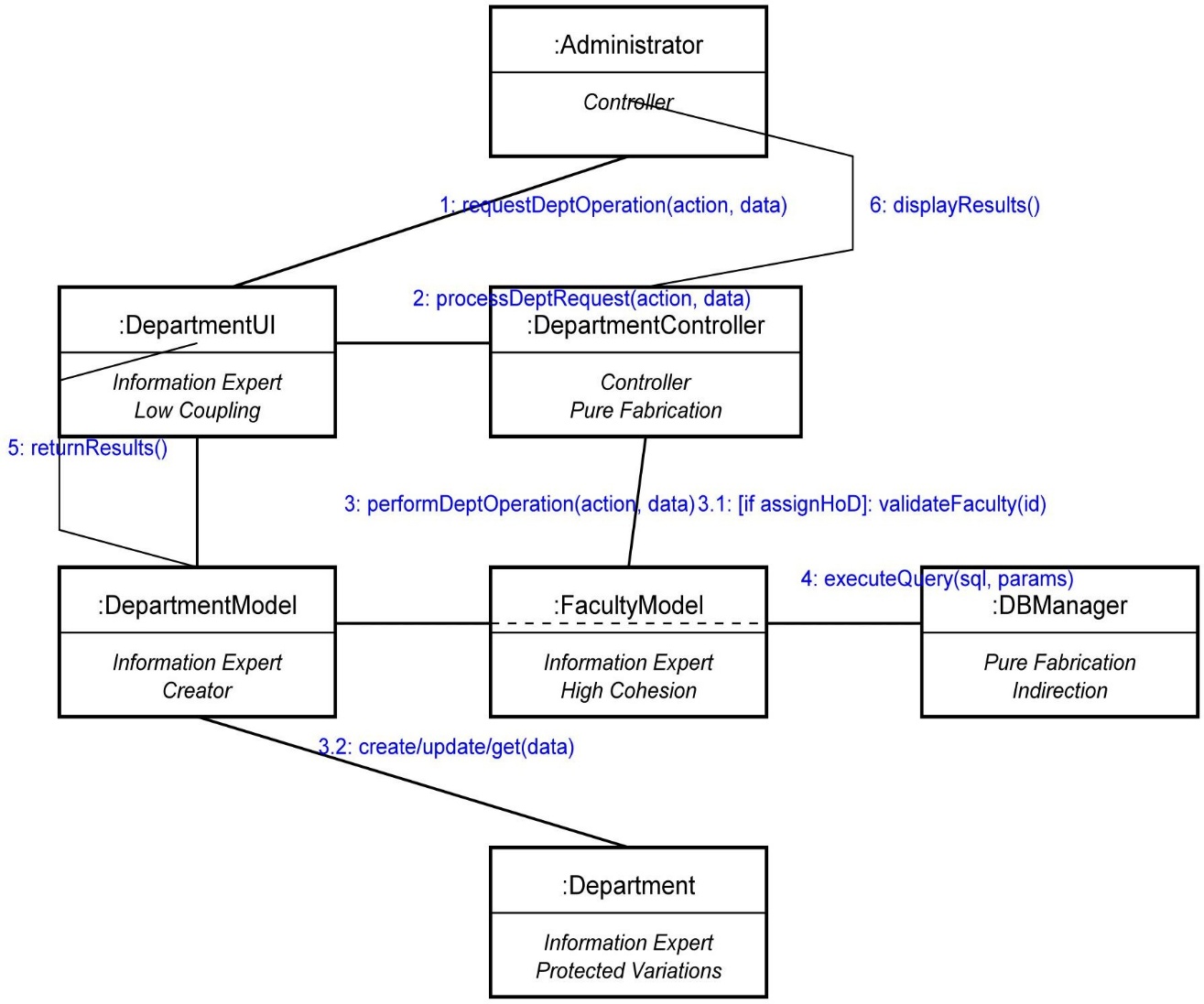
**Ahsaxaxasnx**

**USE Case diagram:**



**Communication Diagram:**





Department Management Communication Diagram with GRASP Principles

I've created a communication diagram that explicitly incorporates GRASP (General Responsibility Assignment Software Patterns) principles into the design. This

approach ensures proper assignment of responsibilities to objects in the Department Management Module.

**Objects and Their GRASP Roles:**

* + **GRASP Role**: Controller (initiates system operations)
  + Initiates all department management operations
  + **GRASP Roles**: Information Expert (for UI data), Low Coupling
  + Handles user interaction and displaying results
  + Maintains low coupling by delegating business logic to controllers
  + **GRASP Roles**: Controller, Pure Fabrication
  + Coordinates operations between UI and domain objects
  + Pure Fabrication as it's created specifically to handle control tasks
  + **GRASP Roles**: Information Expert, Creator
  + Maintains knowledge about department data
  + Creates Department objects (Creator pattern)
  + **GRASP Roles**: Information Expert, High Cohesion
  + Handles faculty-specific data (focused responsibility)
  + High cohesion as it handles only faculty-related operations
  + **GRASP Roles**: Pure Fabrication, Indirection
  + Abstracts database operations (indirection)
  + Shields other objects from database implementation details
  + **GRASP Roles**: Information Expert, Protected Variations
  + Contains core department data and behavior Protects system from variations in department requir

Name: Hammad Fareed

Use Case Diagram

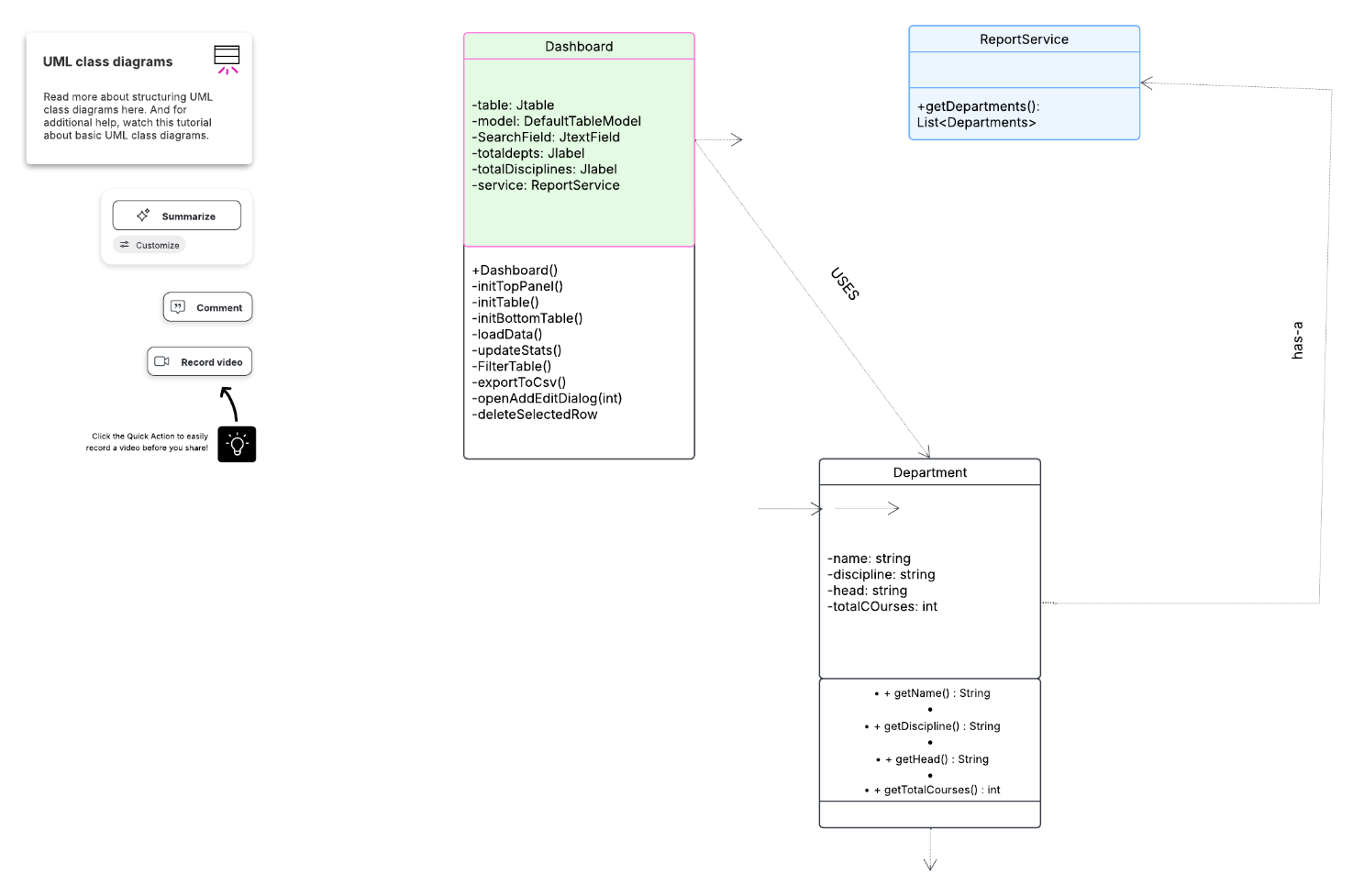
A diagram of a person with blue circles and text

AI-generated content may be incorrect.

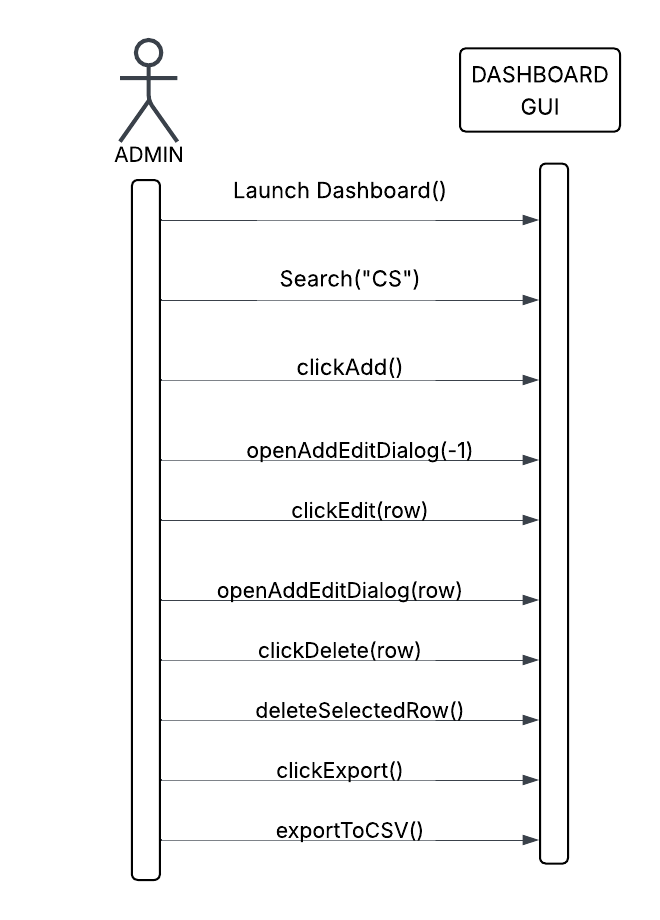
FULLY DRESSED USE-CASE:

|  |  |
| --- | --- |
| **Use Case Element** | **Details** |
| **Use Case Name** | Manage Department Reports |
| **Primary Actor** | Admin (System Administrator) |
| **Goal in Context** | Enable the Admin to view, search, add, edit, delete, and export department and discipline information using a GUI-based dashboard. |
| **Scope** | DeptTrack-Reporting Dashboard |
| **Level** | User Goal |
| **Stakeholder and Interests** | - **Admin** wants to efficiently manage department and discipline reports including CRUD and export features. –  **University** requires accurate data management and reporting functionality for departments and disciplines. |
| **Preconditions** | * Admin is authenticated and logged in. * The system (dashboard) is running and displays the main interface. |
| **Postconditions (Success)** | * The department data is updated (added/edited/deleted). * Report data may be exported to a CSV file. * Total stats (departments/disciplines) are refreshed. |
| **Postconditions (Failure)** | * Data is not updated correctly in the table. * CSV file is not saved; system doesn't notify clearly. * Operation is skipped or throws error; data remains unchanged. * Incorrect count of departments or disciplines shown. * No changes saved, postconditions like stats/export fail silently. |
| **Main Success Scenario** | 1. Admin opens the Dashboard. 2. The system displays all departments in a table. 3. Admin uses search fields to filter departments. 4. Admin selects one of the following:  * Clicks “+” to add a department. * Selects a row and clicks “Edit” to update department. * Selects a row and clicks “Delete” to remove department.  1. Admin clicks “Export to CSV” to download data. 2. System updates and displays:  * Total Departments * Total Disciplines |
| Extensions (Alternate Flows) | 3a. No departments exist:   * System shows empty table and stats = 0.   4a. Admin enters invalid course count:   * System shows error message "Invalid course count."   4b. Admin does not select a row for Edit/Delete:   * System shows warning: "Select a row to edit/delete."   5a. Export fails due to I/O error:   * System shows: "Export failed: [error message]" |
| **Special Requirements** | GUI must be responsive and user-friendly.   * File exported must be report.csv. * All inputs should be validated (e.g., course count must be integer). * Data updates should immediately reflect in table and stats. |
| **Frequency of Use** | Frequently – typically every time the system is used. |
| **Open Issues** | database persistence in current version is limited   No authentication integration yet. Done by another member |

Class Diagram:



SSD:



Package Diagram:

A diagram of a computer program

AI-generated content may be incorrect.

**✅ 6. Coding Standards – Module 4: Reporting & Admin Dashboard**

**✅ 1. Project Structure & Naming**

| **Standard** | **Example** |
| --- | --- |
| **Package names** | views, services, models, utils |
| **Class names** | Dashboard, ReportService, Department |
| **Variable names** | searchField, totalDisciplines, model |
| **Constants** | public static final String REPORT\_FILE = "report.csv"; |
| **File names** | Must match class name exactly e.g., Dashboard.java, ReportService.java |

**✅ 2. Class & Method Standards**

| **Guideline** | **Description** |
| --- | --- |
| **One class per file** | Each class like Dashboard or Department is in its own file. |
| **Single responsibility** | Dashboard handles UI, ReportService handles data fetching. |
| **One task per method** | filterTable() only filters, updateStats() only calculates totals. |
| **Verb-based method names** | loadData(), filterTable(), exportToCSV(), openAddEditDialog() |

**✅ 3. Comments & Documentation**

| **Type** | **Guideline** |
| --- | --- |
| **Class comments** | Describe UI purpose (Dashboard) or service logic (ReportService). |
| **Method comments** | Use /\*\* Javadoc \*/ for public methods. |
| **Inline comments** | Only when logic is not obvious. |
| **Block comments** | Use for GUI sections or loops like CSV writing or table updates. |

**Example:**

Filters the JTable based on the search keyword entered by the user.

private void filterTable() { ... }

**✅ 4. Code Formatting**

| **Practice** | **Details** |
| --- | --- |
| **Indentation** | Use 4 spaces (never tabs). |
| **Brackets** | Always use curly braces {} even for single-line blocks. |
| **Line Length** | Wrap lines longer than ~100 characters. |
| **Blank Lines** | Add between methods and inside logical blocks. |
| **Group imports** | Java → 3rd Party → Project packages |

**✅ 5. Error Handling**

| **Rule** | **Example** |
| --- | --- |
| **Meaningful messages** | throw new IOException("Failed to write report.csv"); |
| **Catch specific exceptions** | catch (NumberFormatException e) instead of generic Exception |
| **Avoid silent failures** | Always show dialog for errors in file export or input parsing. |

**✅ 6. Modularity & Reusability**

| **Best Practice** | **Description** |
| --- | --- |
| **Don’t hardcode** | Use constants like REPORT\_FILE instead of direct strings. |
| **Avoid duplicate code** | Reuse logic in loadData(), updateStats(). |
| **Break down logic** | GUI actions broken into helper methods like openAddEditDialog() |
| **Service abstraction** | ReportService acts as backend abstraction for future DB support. |

**✅ 7. Version Control Standards (Git/GitHub)**

| **Rule** | **Description** |
| --- | --- |
| **Use feature branches** | Example: feature/dashboard-reporting |
| **Commit often** | With each working feature like “Add Edit/Delete support” |
| **Clear commit messages** | "Add export to CSV functionality to Dashboard" |
| **Pull and merge regularly** | To avoid UI code conflicts |
| **Document module usage** | Briefly describe Dashboard in README or as class Javadoc |