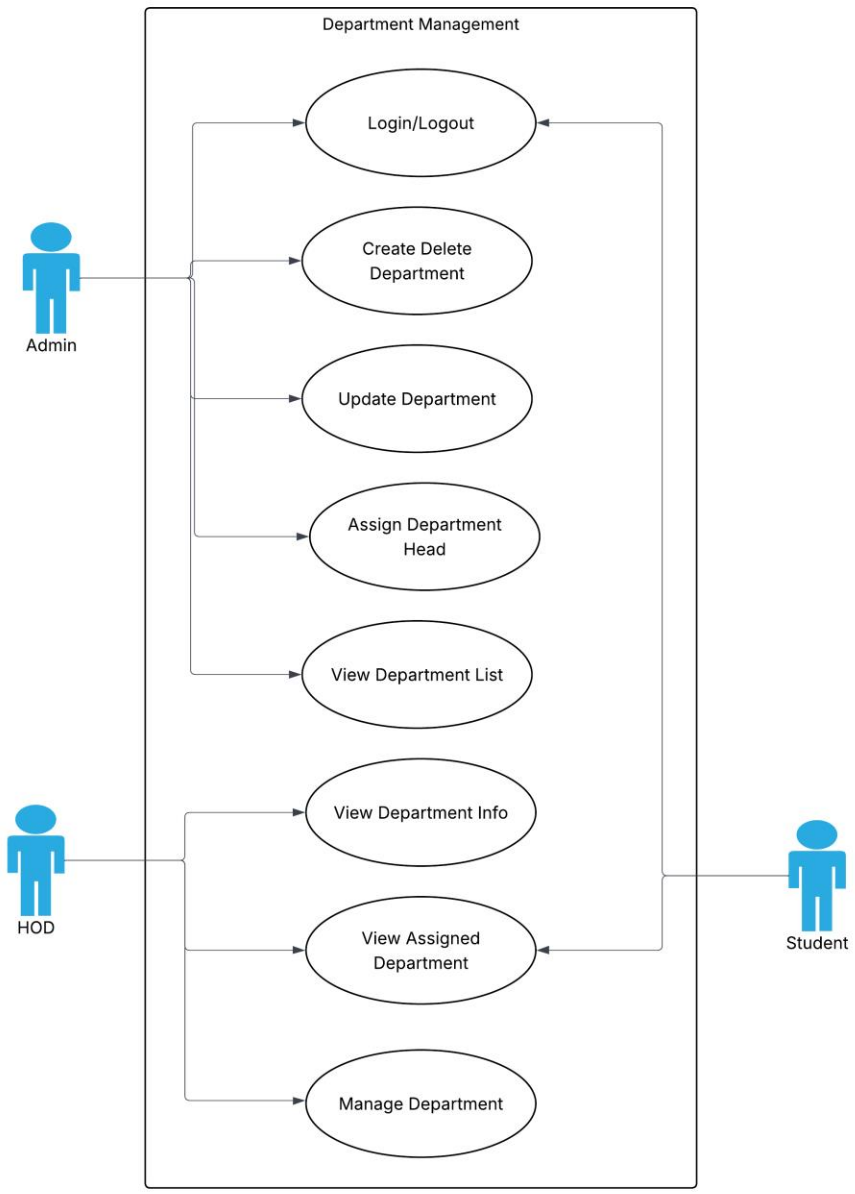
Name: Muhammad Usman Waqar Khan

Registration No: SP23-BSE-010

Chapter 1

Use Case Diagrams



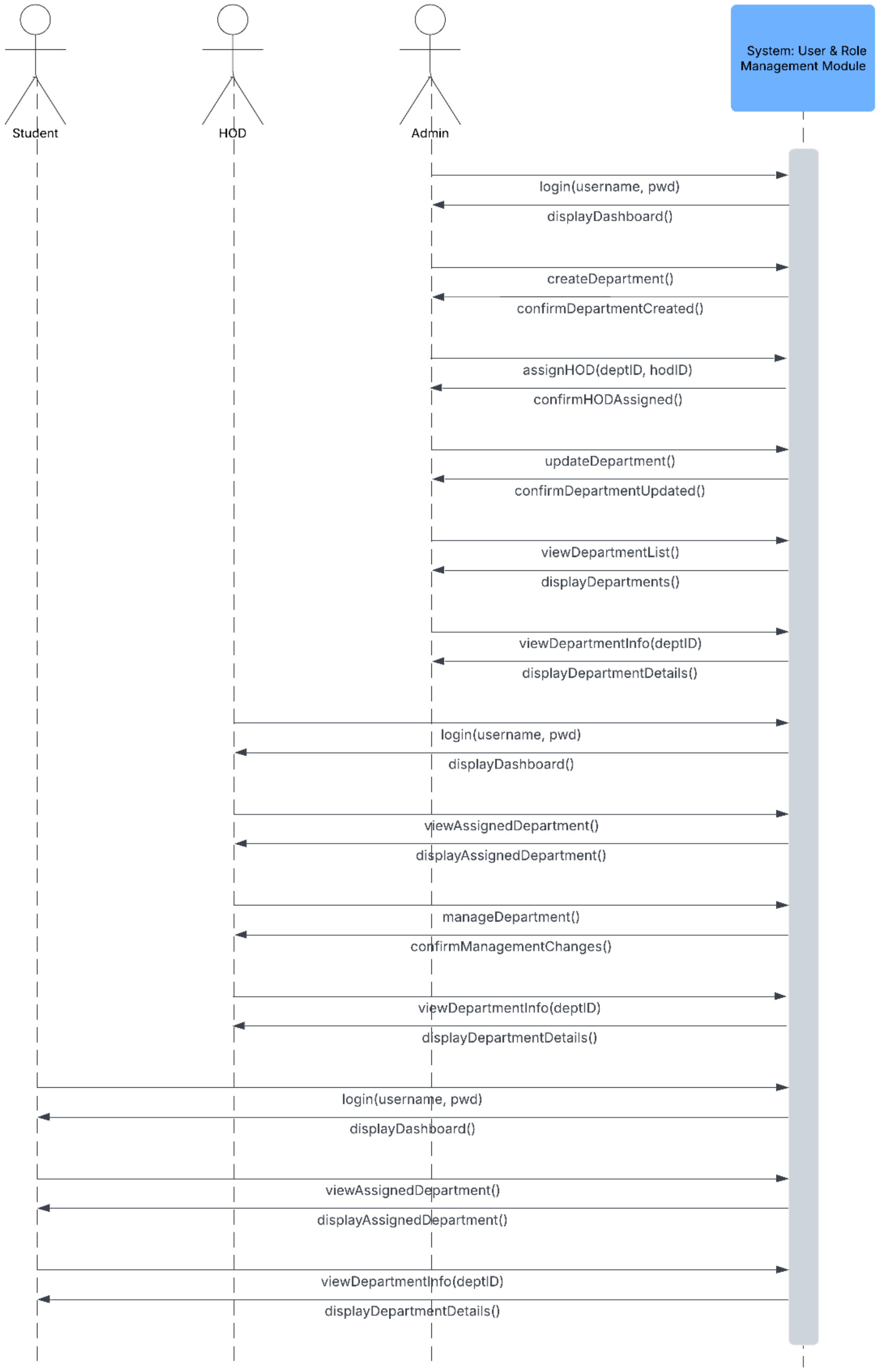
2. Fully Dressed Use Cases

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?

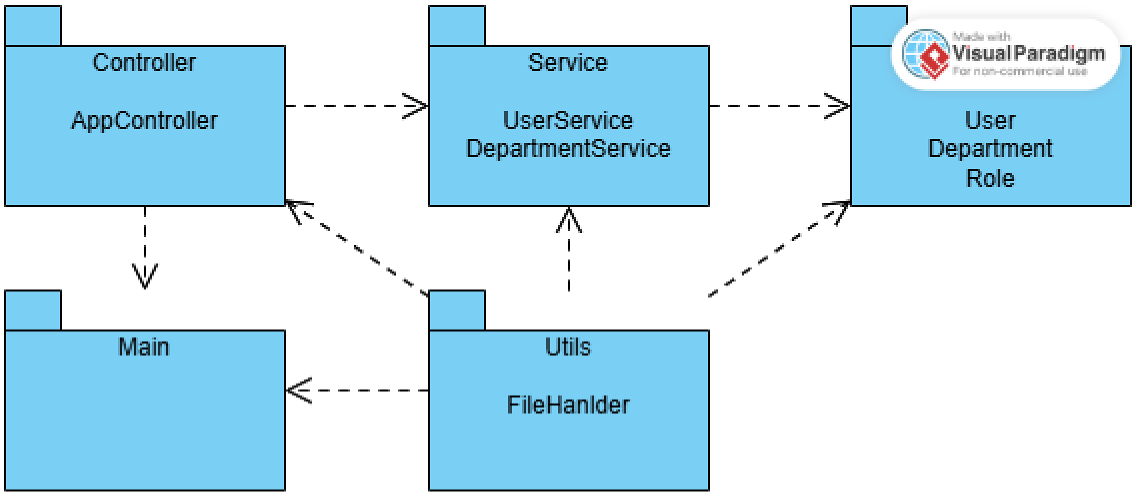
3. System Sequence Diagram

Name: M. Usman Waqar Khan

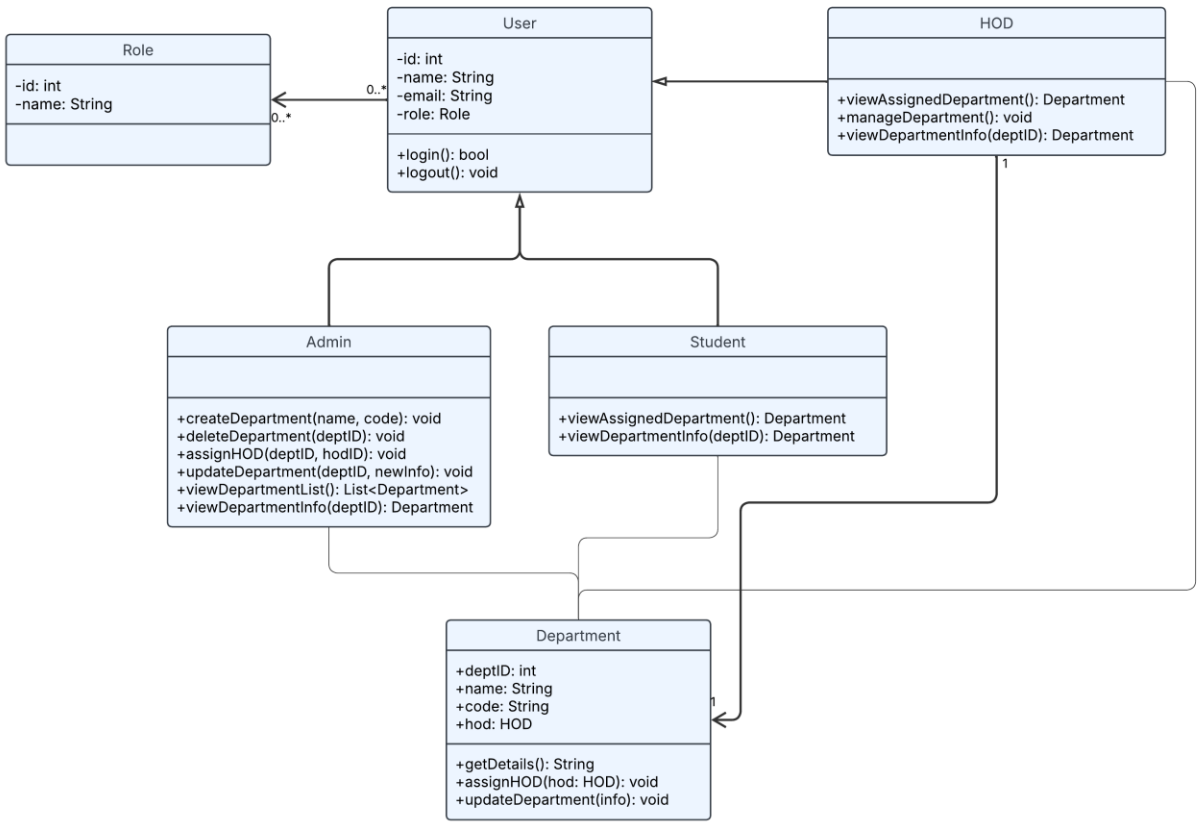

Student
HOD
Admin
System: User & Role Management Module
login(username, pwd)
displayDashboard()
createDepartment()
confirmDepartmentCreated()
assignHOD(deptID, hodID)
confirmHODAssigned()
updateDepartment()
confirmDepartmentUpdated()
viewDepartmentList()
displayDepartments()
viewDepartmentInfo(deptID)
displayDepartmentDetails()
login(username, pwd)
displayDashboard()
viewAssignedDepartment()
displayAssignedDepartment()
manageDepartment()
confirmManagementChanges()
viewDepartmentInfo(deptID)
displayDepartmentDetails()
login(username, pwd)
displayDashboard()
viewAssignedDepartment()
displayAssignedDepartment()
viewDepartmentInfo(deptID)
displayDepartmentDetails()

4. Package Diagram

Name: M. Usman Waqar Khan



5. Class Diagram


Role
-id: int
-name: String
User
-id: int
-name: String
-email: String
-role: Role
+login(): bool
+logout(): void
0..*
0..*
HOD
+viewAssignedDepartment(): Department
+manageDepartment(): void
+viewDepartmentInfo(deptID): Department
1
Admin
+createDepartment(name, code): void
+deleteDepartment(deptID): void
+assignHOD(deptID, hodID): void
+updateDepartment(deptID, newInfo): void
+viewDepartmentList(): List<Department>
+viewDepartmentInfo(deptID): Department
Student
+viewAssignedDepartment(): Department
+viewDepartmentInfo(deptID): Department
Department
+deptID: int
+name: String
+code: String
+hod: HOD
+getDetails(): String
+assignHOD(hod: HOD): void
+updateDepartment(info): void
1

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
